ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Partialist

Σύστημα Επεξεργασίας Ερωτήσεων Για Δενδρικά Δεδομένα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΜΑΥΡΟΓΕΩΡΓΗ ΖΑΦΕΙΡΑΣ

Επιβλέπων : Ιωάννης Βασιλείου
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2008

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Partialist

Σύστημα Επεξεργασίας Ερωτήσεων

Για Δενδρικά Δεδομένα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΜΑΥΡΟΓΕΩΡΓΗ ΖΑΦΕΙΡΑΣ

Επιβλέπων : Ιωάννης Βασιλείου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 9η Ιουλίου 2008.

.....	.....	.....
Ιωάννης Βασιλείου	Τίμος Σελλής	Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Αναπλ. Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2008

.....

ΜΑΥΡΟΓΕΩΡΓΗ ΖΑΦΕΙΡΑ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τους καθηγητές κ.Ιωάννη Βασιλείου και κ.Τιμολέων Σελλή για τη δυνατότητα που μου έδωσαν να εκπονήσω τη διπλωματική μου εργασία στο εργαστήριο συστημάτων βάσεων γνώσεων και δεδομένων και συνολικά για την υποστήριξή τους στο διάστημα της εκπόνησης, καθώς και για την υποδειγματική τους παρουσία και διδασκαλία όλα αυτά τα χρόνια της φοίτησής μας στη Σχολή.

Επίσης ένα μεγάλο ευχαριστώ στον υποψήφιο διδάκτωρ Στέφανο Σουλδάτο και στον ερευνήτη Θεόδωρο Δαλαμάγκα, στη δουλειά των οποίων στηρίχθηκε η εργασία αυτή, για τη διαρκή συνεργασία και την αμέριστη συμπαράσταση που μου προσέφεραν.

Τέλος θα ήθελα να ευχαριστήσω θερμά την οικογένειά μου για την υπομονή και την πολύπλευρη συμπαράστασή της.

Περίληψη

Η παρούσα διπλωματική εργασία ασχολείται με το σχεδιασμό και την υλοποίηση ενός συστήματος σύνταξης, διαχείρισης και επεξεργασίας ερωτήσεων για δενδρικά δεδομένα XML, του **Partialist**. Βασικό χαρακτηριστικό των ερωτήσεων είναι ότι επιτρέπουν μερικό προσδιορισμό δομής. Τα XML δέντρα και τα ερωτήματα θα δημιουργούνται ή θα ανοίγονται από αρχεία και θα αποτυπώνονται γραφικά. Ο χρήστης θα έχει τη δυνατότητα να τα επεξεργάζεται και να τα αποθηκεύει σε αρχεία. Κύρια όμως λειτουργία της εφαρμογής είναι να μπορεί ο χρήστης να κάνει ένα ερώτημα σε ένα XML δέντρο και το αποτέλεσμα της αποτίμησης να εμφανίζεται πάλι με γραφικό τρόπο ως ένα νέο XML δέντρο. Στηρίζεται κυρίως στο paper του Στέφανου Σουλδάτου και άλλων με τίτλο : "**Evaluation of Partial Path Queries on XML Data**" [**CIKM07**].

Λέξεις Κλειδιά: XML, γλώσσα ερωτήσεων, ερωτήσεις μονοπατιού μερικής δομής, αλγόριθμοι αποτίμησης ερωτήσεων δενδρικού προτύπου και προτύπου μονοπατιού, PathStack, PartialMJ, PartialMJPlus, PartialPathStack

Abstract

This diploma thesis deals with the design and the implementation of a system for the administration of partial path queries for XML data, called **Partialist**. A basic characteristic of these queries is that they allow paths with partial structure. XML trees and queries will be created graphically and stored in files. Moreover, the user will be able to pose queries over XML data and see the evaluation results graphically. This work mainly relies on the paper of Stefanos Souldatos et al.: "**Evaluation of Partial Path Queries on XML Data**" [**CIKM07**].

Keywords: XML, query language, partial path pattern queries, evaluation algorithms for path and twig pattern queries, PathStack, PartialMJ, PartialMJPlus, PartialPathStack

Πίνακας περιεχομένων

1

Εισαγωγή

1.1 Αντικείμενο διπλωματικής

Η παρούσα διπλωματική εργασία περιγράφει την ανάπτυξη μιας πλατφόρμας μοντελοποίησης και αναπαράστασης XML δέντρων και ερωτημάτων καθώς και την αποτίμηση μερικώς ορισμένων ερωτημάτων μονοπατιού σε XML δέντρα. Βασικό χαρακτηριστικό των ερωτήσεων είναι ότι επιτρέπουν μερικό προσδιορισμό δομής. Στηρίζεται κυρίως στο paper του Στέφανου Σουλδάτου και άλλων με τίτλο : "**Evaluation of Partial Path Queries on XML Data" [CIKM07].**

Πρόκειται για την υλοποίηση ενός σχεδιαστικού εργαλείου, του ***Partialist***, το οποίο δίνει τη δυνατότητα σχεδίασης και διαχείρισης XML δέντρων μορφής ετικετών, XML δέντρων δενδρικής μορφής και ερωτημάτων. Ακόμα δίνει τη δυνατότητα αποτίμησης ενός ερωτήματος πάνω σε ένα XML δέντρο και αναπαράσταση του δέντρου-αποτελέσματος. Επίσης παρέχει βασικές λειτουργίες όπως προσθήκη κόμβου/κόμβων δέντρου ή ερωτήματος, ενημέρωση ενός κόμβου, προσθήκη κόμβου-παιδιού σε δέντρο, προσθήκη ακμής ερωτήματος που δηλώνει σχέση πατέρα-παιδιού, προσθήκη ακμής ερωτήματος που δηλώνει σχέση προγόνου-απογόνου, διαγραφή κόμβων και ακμών, αντικατάσταση κόμβου-πατέρα ενός κόμβου σε δέντρο. Συνάμα επιτρέπει, μέσω ενός συνόλου από άλλες λειτουργίες, μια πλειάδα επεμβάσεων πάνω στα ερωτήματα όπως εύρεση των άκυρων ακμών, έλεγχο πλήρως

ορισμένου ερωτήματος μονοπατιού, έλεγχο ύπαρξης ‘ξεκρέμαστων’ κόμβων ερωτήματος. Επίσης κατά την επεξεργασία ενός ερωτήματος, μετατρέπει σε πραγματικό χρόνο το ερώτημα σε κατάλληλη μορφή για αποτίμηση.

Όταν θέλουμε να θέσουμε ένα ερώτημα σε διάφορα XML έγγραφα με το ίδιο περιεχόμενο μπορεί να συναντήσουμε κάποιες δυσκολίες. Οι πολλαπλές πηγές μπορεί να δίνουν διαφορετικές δομές. Μπορεί δύο XML έγγραφα να έχουν διαφορετική δομή, δηλαδή μπορεί το ένα έγγραφο να περιέχει ένα στοιχείο ενώ το άλλο να μην το περιέχει. Επίσης μπορεί να έχουν δομική ασυνέπεια, δηλαδή να υπάρχουν και στα δύο έγγραφα τα στοιχεία που ψάχνω αλλά να μην είναι με την ίδια ιεραρχία. Μπορεί ακόμα να μην γνωρίζουμε τη δομή ενός εγγράφου. Οπότε η αποτίμηση ενός ερωτήματος να μην είναι εφικτή για όλα αυτά τα έγγραφα. Η λύση είναι να ορίσουμε ερωτήματα με μερικό προσδιορισμό δομής. Μια ερώτηση μερικού μονοπατιού μπορεί να εκφραστεί ισοδύναμα από ένα σύνολο ερωτήσεων μονοπατιού που καλύπτουν όλες τις αποδεκτές μεταθέσεις των κόμβων της ερώτησης. Αυτός ο μετασχηματισμός όμως μπορεί να οδηγήσει σε εκθετικό αριθμό ερωτήσεων. Έχει γίνει έρευνα για την αποτελεσματική αποτίμηση ερωτήσεων μερικού μονοπατιού. Ο αλγόριθμος PartialPathStack είναι αποδοτικός για την αποτίμηση ερωτήσεων μονοπατιού μερικής δομής και είναι αυτός που θα χρησιμοποιήσουμε κυρίως στη διπλωματική μας. Στηρίζεται στον αλγόριθμο PathStack που είναι αποδοτικός για ερωτήματα μονοπατιού πλήρους δομής, τον οποίο θα χρησιμοποιήσουμε επίσης.

1.1.1 Συνεισφορά

Η συνεισφορά της διπλωματικής συνοψίζεται ως εξής:

1. Μελετήσαμε ερωτήματα μονοπατιού με μερικώς ορισμένη δομή και πώς αυτά αποτιμώνται πάνω σε XML δεδομένα.
2. Σχεδιάσαμε και υλοποιήσαμε ένα σύστημα επεξεργασίας ερωτήσεων για δενδρικά δεδομένα. Στο σύστημα υπάρχει η δυνατότητα επεξεργασίας και διαχείρισης XML δέντρων και ερωτημάτων μονοπατιού μερικούς ή πλήρους δομής.
3. Στο σύστημά μας επεξεργαστήκαμε τα ερωτήματα σε πραγματικό χρόνο ώστε να τα μετατρέπουμε σε κατάλληλη μορφή για την αποτίμησή τους.
4. Ενσωματώσαμε τους αλγορίθμους PathStack και PartialPathStack στο σύστημά μας που εκτελούν την αποτίμηση των ερωτημάτων.

1.2 Οργάνωση του τόμου

Το υπόλοιπο μέρος αυτής της εργασίας έχει την εξής δομή :

Στο δεύτερο κεφάλαιο παρουσιάζονται κάποια βασικά χαρακτηριστικά της XML. Ακόμα παρουσιάζονται συνοπτικά κάποιες εργασίες που σχετίζονται με το θέμα της διπλωματικής μας.

Στο τρίτο κεφάλαιο παρουσιάζεται η δομή ενός XML δέντρου και η κωδικοποίηση θέσης των κόμβων του δέντρου. Ορίζονται τα ερωτήματα μονοπατιού πλήρους δομής, αλλά και μερικής δομής που κυρίως μας ενδιαφέρουν. Τέλος δίνεται η επεξεργασία ερωτήσεων μερικού μονοπατιού για επιτάχυνση της αποτίμησης του ερωτήματος.

Στο τέταρτο κεφάλαιο παρουσιάζονται οι αλγόριθμοι αποτίμησης ερωτήσεων μονοπατιού πάνω σε δενδρικά δεδομένα. Για ερωτήματα μονοπατιού πλήρους δομής χρησιμοποιούμε τον αλγόριθμο PathStack. Για ερωτήματα μονοπατιού μερικής δομής παρουσιάζουμε τους αλγόριθμους PartialMJ, PartialMJPlus και PartialPathStack. Καταλήγουμε στο τέλος ότι ο PartialPathStack είναι ο πιο αποδοτικός για την αποτίμηση ερωτημάτων μερικού μονοπατιού.

Στο πέμπτο κεφάλαιο γίνεται η ανάλυση απαιτήσεων της εφαρμογής μας. Χωρίζουμε το σύστημα μας σε επιμέρους υποσυστήματα, τη Διαπροσωπεία Χρήστη, το Διαχειριστή XML Δέντρων μορφής ετικετών, το Διαχειριστή XML Δέντρων δενδρικής μορφής , το Διαχειριστή Ερωτημάτων και τον Αποτιμητή.

Στο έκτο κεφάλαιο παρουσιάζεται η σχεδίαση της εφαρμογής μας. Δίνονται οι κλάσεις του κάθε υποσυστήματος. Στο τέλος δίνονται και οι κωδικοποιήσεις των αρχείων που χρησιμοποιεί η εφαρμογή μας. Υπάρχουν τρία είδη αρχείων που αφορούν XML δέντρα (XML αρχεία, αρχεία κωδικοποίηση θέσης, αρχεία μορφής treeData) καθώς και ένα είδος αρχείων που αφορούν τα ερωτήματα.

Στο έβδομο κεφάλαιο μπαίνουμε στο ζήτημα της υλοποίησης που στηρίζεται στην ανάλυση και το σχεδιασμό που έχουν προηγηθεί. Δίνονται οι γλώσσες προγραμματισμού και τα προγραμματιστικά εργαλεία που χρησιμοποιούμε στην εφαρμογή μας. Επίσης, δίνονται συνοπτικά τα πεδία και οι μέθοδοι των κλάσεων του προγράμματός μας. Ακόμα αναλύουμε τα σημαντικότερα επιμέρους θέματα υλοποίησης που αντιμετωπίσαμε δίνοντας έτσι μια πληρέστερη εικόνα του προγράμματος.

Στο όγδοο κεφάλαιο, που αποτελεί τον επίλογο, κάνουμε μια σύνοψη της εργασίας μας και παρουσιάζουμε τα συμπεράσματά μας. Επίσης προτείνουμε πιθανές μελλοντικές επεκτάσεις της διπλωματικής.

Στο ένατο κεφάλαιο παρουσιάζεται η σχετική βιβλιογραφία.

Το Παράρτημα Α αποτελεί το Εγχειρίδιο Χρήσης Εξηγούνται αναλυτικά οι λειτουργίες της εφαρμογής και περιγράφεται η διεπαφή του προγράμματος, ώστε ο μελλοντικός χρήστης του να αποκτήσει μια εξοικείωση με το περιβάλλον εργασίας.

Το Παράρτημα Β αποτελεί την αναλυτική περιγραφή των κλάσεων του προγράμματος. Είναι το documentation της εφαρμογής, όπου μπορεί να ανατρέξει κανείς για να δει τη δομή του interface των κλάσεων.

2

Τεχνολογίες XML

2.1 Εισαγωγή στην XML

Η XML (Extended Mark-up Language) είναι μια δηλωτική γλώσσα πού ήρθε μετά την ASN1. Κύρια διαφορά είναι ότι στην XML χρησιμοποιούνται αποκλειστικά χαρακτήρες (character based, έναντι της ASN1 πού είναι bit coded). Επιπλέον η δομή της είναι ταυτοχρόνως πιο απλή αλλά και πιο εύκαμπτη. Από άλλη οπτική γωνία, στην XML ένα κείμενο χρησιμοποιείται για την αναπαράσταση δομημένης πληροφορίας στο διαδίκτυο.

Όπως και στην HTML (Hyper Text Mark-up Language) και η XML είναι γλώσσα που επισημαίνει επιπλέον πληροφορία αναφερόμενη στο βασικό κείμενο. Στην XML προσδιορίζουμε την πληροφορία αυτή με την βοήθεια ετικετών (tags), δηλ. με ένα αναγνωριστικό περικλειόμενο από <>. Με τις ετικέτες (tags) επιτυγχάνουμε το “markup”. Tags υπάρχουν και στην HTML, π.χ. το ζεύγος <B> </B> ή το μοναδικό
. Αυτά όμως είναι προκαθορισμένα και σχετίζονται μόνο με την εμφάνιση του κειμένου. Τα tags της XML είναι προσδιορίσιμα ελεύθερα από τον χρήστη, ο οποίος μπορεί να τους δίδει και όποια σημασία αυτός ορίσει. Για αυτό και πρέπει πάντα να ζευγαρώνονται, δηλ. <text1>my text is here</text1>. Το αναγνωριστικό <text1> </text1> δεν προσδιορίζει αναγκαστικά το πώς θα εμφανισθεί το περικλειόμενο κείμενο (όπως θα έκανε το <B> </B> της HTML), αλλά μπορεί π.χ. να συνδυάζεται με μία ενέργεια, όπως θα καθορίζει δικός μας κώδικας που θα

επεξεργασθεί το αρχείο XML. Το ζεύγος <text1> και </text1> ορίζει ένα **στοιχείο** (element), του οποίου το όνομα είναι το text1 και το περιεχόμενο οτιδήποτε περικλείεται μεταξύ του <text1> (opening tag) και του <text1> (closing tag).

Το επόμενο βήμα είναι η ενθυλάκωση (nesting) που δίδει την δύναμη να κατασκευασθούν στοιχεία με πολύπλοκη εσωτερική δενδρική δομή, π.χ.

```
<message>
   <to>you@yourAddress.com</to>
   <from>me@myAddress.com</from>
   <subject>my subject</subject>
   <text>
        This is my text…
   </text>
  <another_text/>
</message>
```

Τα πυκνοτυπωμένα είναι tags (αναγνωριστικά), πάντα ζευγαρωμένα. Τόσο τα tags, όσο και η περικλειόμενη πληροφορία είναι πάντα κείμενο (text) (μία εξαίρεση παρακάτω). Η XML είναι εύκαμπτη, ευκόλως εννοούμενη, άλλα βεβαίως και σπάταλη σε κωδικοποίηση (text αντί bits). Στο παραπάνω παράδειγμα έχουμε στοιχεία με το όνομα message, to, from, subject, text, another_text. Το στοιχείο another_text είναι κενό (empty element) και αντί ζεύγους <another_text> (ανοίγοντος) και </another_text> (κλείνοντος tag), έχουμε την συντομογραφία <another_text/>, πού ανοίγει και κλείνει συγχρόνως. Συνήθως, όπως παρατηρούμε και στο παραπάνω παράδειγμα, το περιεχόμενο του στοιχείου είναι άλλο (άλλα) ενθυλακωμένα στοιχεία, ή / και κείμενο (text). Ένα στοιχείο με περιεχόμενο μόνο κείμενο λέγεται text element. Επίσης για το περικλειόμενο κείμενο χρησιμοποιείται και ο ορισμός PCDATA (Parsed Character Data), αφού το κείμενο αυτό λαμβάνεται υπ΄όψιν σαν ενιαία και ξεχωριστή οντότητα από τους parsers.

Η δενδρική μορφή είναι προφανής και μπορούμε στον browser να ανοιγοκλείνουμε τα στοιχεία (φύλλα κάποιου βάθους). Υπάρχουν κάποια δεσμευμένα σύμβολα που για να τα χρησιμοποιήσουμε πρέπει να τα γράψουμε αλλιώς. Για παράδειγμα το " γίνεται " και τα <,> < , > αντίστοιχα. Ο browser απλώς αντιλαμβάνεται και αποκωδικοποιεί το συντακτικό της δομής και την παρουσιάζει σαν δένδρο. Παραβιάζοντας την σύνταξη βλέπουμε σχετικές παρατηρήσεις που εκδίδει ο browser.

Μεταβλητές Ιδιοτήτων – Attributes

Κάθε στοιχείο (element) δυνατόν να έχει και ιδιότητες (attributes) οι μεταβλητές των οποίων μαζί με τις τιμές τους συντάσσονται σύμφωνα με το παράδειγμα

<movie type=”comedy” rating=”for adults” year=”1998”> …. </movie>

Το tag (movie) συνοδεύεται από ότι ο χρήστης θέλει να περιλάβει σαν ιδιότητες (comedy, rating, year). Κάθε μεταβλητή ιδιότητας φέρει σαν τιμή ένα text string. Η σημασία και χρήση των attributes είναι θέμα του χρήστη ή της εφαρμογής που επεξεργάζεται το XML. Κάθε μεταβλητή ιδιοτήτων θα μπορούσε να αποτελέσει υποστοιχείο (sub element), δηλ. το παραπάνω παράδειγμα θα μπορούσε να γραφεί σαν:

```
<movie>
        <type>comedy</type>
        <rating>for adults</rating>
        <year>1998</year>
</movie>
```

Το τι θα προτιμηθεί έγκειται συνήθως στην εμπειρία του προγραμματιστή (π.χ. ένα μακρύ κείμενο δεν θα ήταν κομψό ή πρακτικό να περιλαμβάνεται στις ιδιότητες).

Η πρώτη γραμμή <?xml version="1.0"?> αποτελεί εντολή προς τον parser (όλες οι εντολές αυτές περικλείονται από <? ?>), δηλ. το πρόγραμμα που αποκωδικοποιεί την δομή του XML. Μέσα σε αυτήν την εντολή το ‘version’ είναι μεταβλητή ιδιοτήτων. Το string xml είναι δεσμευμένη λέξη και δεν μπορεί να χρησιμοποιηθεί για attribute, ούτε για tag.

Άλλα Περιεχόμενα Στοιχείου

Πέραν του κειμένου, μπορούμε να περιλάβουμε οποιαδήποτε κωδικοποιημένη ή μη ψηφιακή πληροφορία (εικόνα, video), η οποία όμως περικλείεται από <![CDATA[..]]>. Ο όρος CDATA (Character Data) είναι σε αντίθεση με τον PCDATA (Parsed CDATA), δηλαδή η CDATA δεν υπόκειται σε parsing. Ο parser αγνοεί το περικλειόμενο, το οποίο διοχετεύεται όπου προβλέπει η εφαρμογή.

Ένα πλήρες κείμενο XML (XML document), είναι ένα αρχείο text (δημιουργημένο με Notepad, Word, κάποια εφαρμογή κλπ) το οποίο έχει την κατάληξη .xml. Περιέχει μία και

μόνη δενδρική δομή, κάτω από μία και μοναδική ρίζα. Το στοιχείο ρίζας (root element) συμβολίζεται γενικά με / και δεν φαίνεται στο αρχείο κειμένου που αντιπροσωπεύει το .xml. Ακριβώς κάτω από την ρίζα υπάρχει το μοναδικό document element. Το document element δεν χρειάζεται αναγκαστικά να συμπίπτει με το όνομα του αρχείου. Έτσι κάθε κείμενο XML παρομοιάζεται με ένα σύστημα αρχειοθέτησης με root directory το .xml αρχείο, folders, subfolders τα elements και files το περιεχόμενο των elements. Αυτή η αναλογία βοηθά μνημοτεχνικά την κατανόηση της δομής του XML, χωρίς όμως περαιτέρω προεκτάσεις.

Namespaces

Με τα namespaces εξασφαλίζεται ελευθερία και ανεξαρτησία στην επιλογή του κειμένου των tags, δηλαδή στην ονοματοδότηση των επί μέρους στοιχείων (elements) ανάμεσα σε διαφορετικά κείμενα. Ας υποθέσομε ότι ένα στοιχείο υπάρχει σε διαφορετικά κείμενα με το ίδιο όνομα. Δηλαδή έχομε <any_tag>text1</any_tag> στο doc1.xml και <any_tag>text2</any_tag> στο doc2.xml. Αν τώρα φέρουμε μαζί τα δύο αυτά στοιχεία, για παράδειγμα σε ένα τρίτο doc7.xml, δεν θα μπορούν να διαφοροποιούνται με βάση το όνομά τους. Η λύση έρχεται με ένα πρόθεμα και μία σύμβαση που δηλώνει το πού έχει ορισθεί το κάθε στοιχείο. Έτσι γράφουμε στο doc7.xml το tag του πρώτου σαν nsp1:any_tag και το tag του δεύτερου σαν nsp2:any_tag. Όλα τα στοιχεία που ορίζονται τοπικά μέσα στο doc7.xml δεν φέρουν κανένα πρόθεμα, δηλαδή έχουν το default namespace. Επομένως είναι δυνατόν να συνυπάρχουν και τα τρία tags: nsp1:any_tag, nsp2: any_tag και any_tag. Πώς όμως εξασφαλίζεται η μοναδικότητα των προθεμάτων που επιλέγονται ανεξάρτητα σε διάφορα σημεία του κόσμου; Τα ίδια τα προθέματα nsp1, nsp2 ορίζονται αποκλειστικά στο doc7.xml και διασυνδέονται με κάποια URIs, των οποίων η μοναδικότητα είναι έτσι και ή αλλιώς εξασφαλισμένη. Έτσι μπορούμε να ορίσουμε στην αρχή του δικού μας vehicles.xml (μέσα στο opening tag αυτού) τα nsp1, nsp2

<vehicles xmlns:it = "http://www.italian_trolleys.com"

xmlns:ru = "http://www.russian_trolleys.com">

και μετά να αναφερόμαστε στο κείμενο μας σε it:trolleys, ru:trolleys, τα οποία είναι διαφορετικά του δικού μας trolleys. Άρα τα namespaces δίδουν την δυνατότητα συσχέτισης στοιχείων ενός κειμένου XML με συγκεκριμένα URIs. Η χρήση κάποιου tag σαν it:trolleys, ru:trolleys μέσα στο κείμενο δεν απαιτεί σύνδεση με το αντίστοιχο URI, ούτε καν να είναι αυτό το URI υπαρκτό.

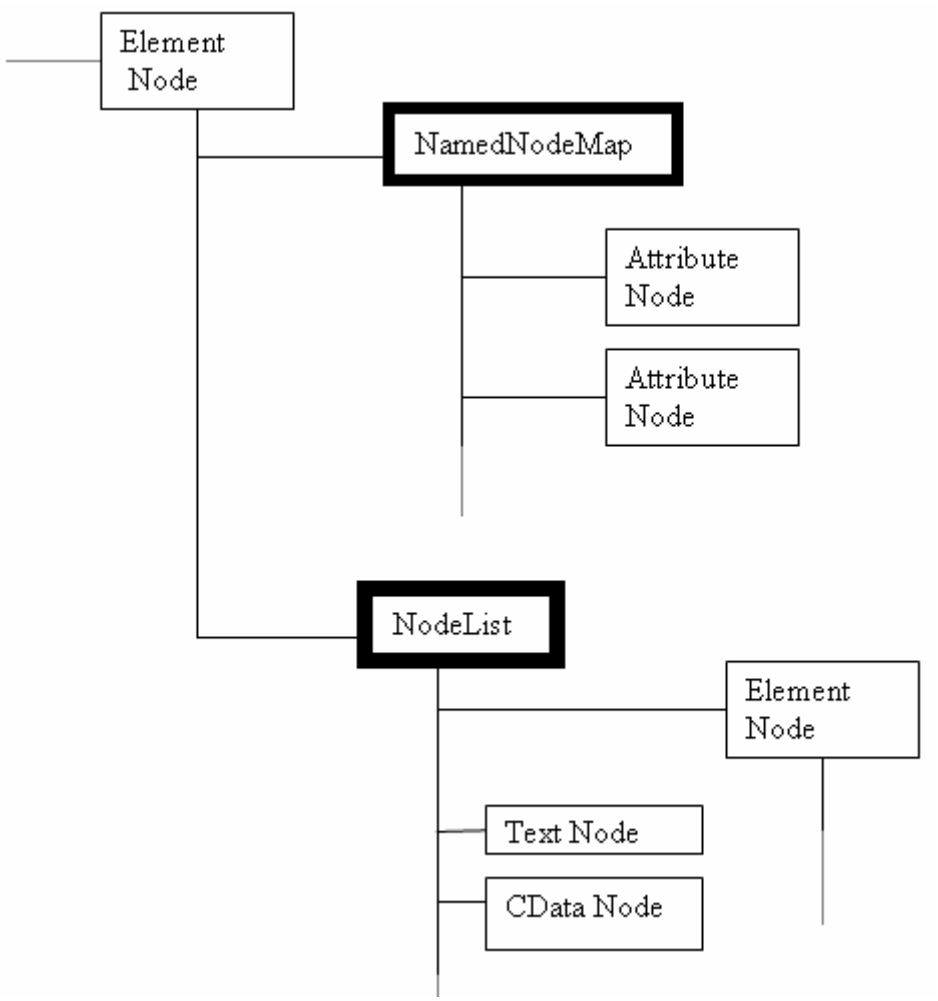
2.2 Σημασία και Χρήση της XML

Η σημασία της XML έγκειται στο γεγονός ότι διατίθενται έτοιμοι τρόποι σύνδεσης της δομής των κειμένων της με τα πλέον σύγχρονα προγραμματιστικά περιβάλλοντα. Τέτοια υπάρχουν για διάφορες γλώσσες, αλλά εμείς θα επικεντρωθούμε κυρίως στην Java. Στην διαδικασία αυτή εμπλέκεται πάντα ένας parser. Ο parser κατέχει κεντρικό ρόλο στην αντιστοίχηση και αλληλοσύνδεση της δενδρικής δομή του κειμένου XML με την αντίστοιχη (συνήθως αντικειμενοστραφή) δομή στην χρησιμοποιούμενη γλώσσα, με δεδομένο ότι το κείμενο XML είναι ένας συρμός χαρακτήρων ανταλλασόμενος μέσω μίας επικοινωνιακής σύνδεσης, ενός αρχείου στον δίσκο, κλπ. Ο parser αποκτά τέτοια σημασία, που θεωρείται πλέον core technology. Οι παρακάτω παρουσιαζόμενοι parsers είναι οι πλέον διαδεδομένοι και open source, αλλά υπάρχει εκτεταμένο περαιτέρω αντικείμενο για ειδικά περιβάλλοντα, ιδιαίτερες απαιτήσεις απόδοσης, ελαχιστοποιημένο κώδικα (π.χ. για ενσωματωμένα συστήματα), κλπ.

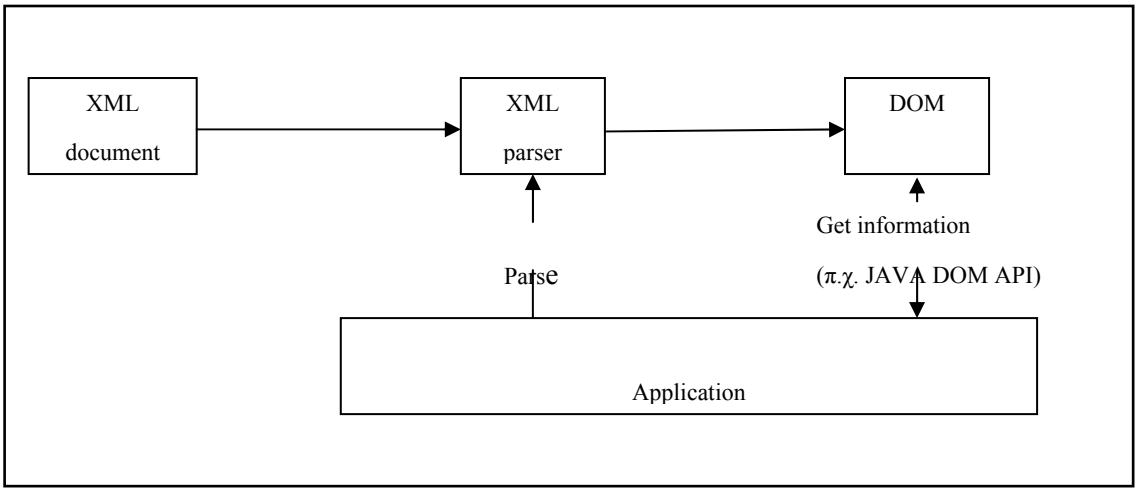
XML και DOM (Document Object Model)

Το **Document Object Model** (**DOM**) μας βοηθά να θεωρήσουμε την δενδρική μορφή του XML κειμένου με αντικειμενοστρεφή τρόπο και να επέμβουμε σε αυτό με συγκεκριμένα interfaces. Με το XML DOM μπορούμε να δημιουργήσουμε ένα XML κείμενο, να πλοηγηθούμε μέσα σε αυτό και να προσθέσουμε, μεταβάλουμε και αφαιρέσουμε στοιχεία του. Ένα XML κείμενο περνά από έναν parser, ο οποίος δημιουργεί και τα σχετικά αντικείμενα όπως προδιαγράφει το DOM. Με τα διατιθέμενα interfaces (μέσω JavaScript, VBScript, Java, κλπ) κάνουμε τις επιθυμητές επεξεργασίες και κατόπιν, αν θέλουμε, στην αντίθετη κατεύθυνση δημιουργούμε ένα νέο παραλλαγμένο κατά τις επιθυμίες μας XML κείμενο. Κεντρική ιδέα του **DOM** είναι το **Node Object** με το αντίστοιχό του **Node Interface**, το οποίο θα δούμε παρακάτω. Όμως εδώ σαν Node (κόμβος) νοείται όχι μόνον το κάθε στοιχείο (element) του XML, αλλά πέραν του Element Node, έχουμε το Attribute Node, το Text Node, το CDATA Node και άλλα περιφερειακής σημασίας. Η όλη δενδρική δομή, στην οποία εντάσσεται κατά DOM κάθε κείμενο XML, δίνεται στο Σχήμα 2.1. Όλα τα μη μαυρισμένα τετράγωνα είναι **Node Objects.**

Σημαντικό χαρακτηριστικό του DOM είναι ότι η διαδικασία του parsing γίνεται δια μιας και χωρίς δική μας επέμβαση κατά την εξέλιξή της. Σύμφωνα με το Σχήμα 2.2 , ο parser μετατρέπει το κείμενο XML στο αντίστοιχο δένδρο (κατά το Σχήμα 2.1), το οποίο διατίθεται στο προγραμματιστικό μας περιβάλλον.



Σχήμα 2.1 : Η ιεραρχία κατά το DOM



Σχήμα 2.2 : Γενική αρχή DOM

Ερχόμαστε τώρα στα μαυρισμένα τετράγωνα του Σχήματος 2.1. Ο **NamedNodeMap** περιέχει τα attributes. Είναι μία μη διατεταγμένη συλλογή (collection), καθόσον κάθε attribute ανευρίσκεται με το όνομά του, χωρίς να ενδιαφέρει η σειρά. Αντίθετα η NodeList αποτελούμενη από Text Node, άλλα Element Nodes, κλπ. είναι μία διατεταγμένη λίστα. Συνεπώς, σύμφωνα με όσα γνωρίζουμε, ένα τυπικό XML element αποτελείται αφενός από έναν NamedNodeMap (εντός του οποίου περικλείονται όλα τα attributes, σαν Attribute Nodes) και αφεταίρου από μία NodeList (η οποία περιλαμβάνει xyz Nodes, όπου xyz τα ενδεχόμενα Text, CDATA, κλπ, και εν τέλει κάποια (child) Element). Αυτό ακριβώς θέλει να δείξει το Σχήμα 2.1.

Για τη διάσχιση μέσα στο XML αρχείο χρησιμοποιούμε τις ιδιότητες firstChild, lastChild, previousSibling, nextSibling. Όλες αυτές οι ιδιότητες επιστρέφουν άλλους κόμβους. Όταν πέσουμε σε attributes, χρησιμοποιούμε το NamedNodeMap interface. Αυτό αντιπροσωπεύει μία μη διατεταγμένη συλλογή (collection) κόμβων, τα πάντα ονοματισμένα attributes. Μπορούμε να αναφερθούμε σε κάθε attribute ξεχωριστά με το item (*index*), όπου το *index* αρχίζει από το 0 (0 το πρώτο attribute, 1 το δεύτερο, κοκ) . Τα ίδια ισχύουν και για το NodeList interface, αλλά εδώ έχουμε διατεταγμένη συλλογή. Τέλος τα nodeName, nodeValue είναι οι πιο χρήσιμες ιδιότητες του εκάστοτε κόμβου. Ανάλογα με τον τύπο του κόμβου, επιστρέφεται και η αντιστοιχούσα τιμή κατά τον πίνακα :

NodeType	nodeName	nodeValue
Element Node	Tag name	NULL
Attribute Node	Name of attribute	Value of attribute
Text Node	#text	Content of the text node
CDATA Node	#cdata-section	Content of the CDATA section

Πίνακας 2.1 : Όνομα και τιμή για κάθε τύπο κόμβου

Για να μετατρέψουμε το κείμενο XML μέσω του DOM, υπάρχουν οι μέθοδοι removeChild(oldChild), appendChild(newChild), replace(newChild,oldChild), insertBefore(newChild,refChild), με αναφορές στον τρέχοντα (refChild), παλιό και νέο κόμβο. Με την cloneNode(deep) κλωνοποιούμε τον κόμβο, είτε μόνον του (με deep=false),

είτε αυτόν και όλο το υποδένδρο που κρέμεται από κάτω του (με deep=true). Για να δημιουργήσομε ένα κόμβο, άσχετο και ασύνδετο κατ΄ αρχήν με το κείμενο XML, χρησιμοποιούμε την μέθοδο createElement(tagName) ενός "document". Ανάλογες είναι και οι createTextNode(data), createAttribute(name), createCDATASection(data). Ό,τι δημιουργήσουμε με αυτόν τον τρόπο μπορεί μετά να κρεμασθεί στο δένδρο.

DOM Java API

Τα παραπάνω ήταν μία γρήγορη εισαγωγή. Θα δούμε τώρα πιο συστηματικά το DOM (Level2) API που δίνει η Java. Το API (Application Programming Interface) είναι ένα σύνολο classes τα οποία διατίθενται στην εφαρμογή που γράφουμε, προκειμένου αυτή να χρησιμοποιήσει τις δυνατότητες που δίνει (στην συγκεκριμένη περίπτωση) το DOM. Όταν λοιπόν κάποιος ισχυρίζεται ότι μας διαθέτει προς χρήση ένα DOM **Java API** θα πρέπει, μεταξύ άλλων, να έχει πραγματοποιήσει σε Java, την class Node, η οποία θα πρέπει να μας επιτρέψει να κάνουμε ορισμένα πράγματα με κάθε αντικείμενο του τύπου (της class) Node. Διατυπώνουμε τούτο πιο αυστηρά: Ένα αντικείμενο μίας class στον ορισμό της οποίας αναφέρεται ...implements Node, διαθέτει οπωσδήποτε μία σειρά από μεθόδους (methods), όλες σχετιζόμενες με όσα ο χρήστης (η εφαρμογή μας) μπορεί να κάνει με κάθε DOM node.

Στην κάθε (πάντα public) μέθοδο του Node interface, φαίνεται καθαρά τι παράμετρο ζητά κάθε μέθοδος και τι επιστρέφει και τα περισσότερα είναι προφανή. Έστω nodeobj ένα αντικείμενο μίας class που πραγματοποιεί (implements) το Node interface. Τότε η nodeobj.getNodeName() επιστρέφει το όνομα του nodeobj σαν String. Το τι σημαίνει ΄όνομα΄ καθορίζεται κατά περίπτωση σύμφωνα με τον Πίνακα 2.1 ανάλογα με τον τύπο του nodeobj. Ο τύπος λαμβάνεται με nodeobj.getNodeType(), οπότε το short που επιστρέφεται θα είναι μία από τις παραπάνω οριζόμενες σταθερές. Σύμφωνα με τον Πίνακα 2.1 εκτελείται στην αντίθετη κατεύθυνση και η nodeobj.setNodeValue(nodeValue). Την περιδιάβασή μας την επιτυγχάνουμε με nodeobj.getFirstChild() και τα παρεμφερή που είδαμε παραπάνω, που όλα επιστρέφουν άλλο αντικείμενο της class Node. Ανάλογα ισχύουν για παρεμβολή (insert), αντικατάσταση (replace), εξαφάνιση (remove), προσθήκη (append), κλωνοποίηση (clone) κόμβων. Εδώ χρειάζονται δεδομένα εισόδου της class Node κατά περίπτωση, μας επιστρέφεται δε πάλι ο κατά περίπτωση κόμβος. Η πεμπτουσία του DOM φαίνεται με τις nodeobj.getChildNodes() και nodeobj.getAttributes(), που μας επιστρέφουν αντικείμενο της class NodeList και NamedNodeMap αντίστοιχα, πάντα κατά το Σχήμα 2.1. Οι class NodeList

και class NamedNodeMap είναι βεβαίως και αυτές interface του DOM. Η διαφορά τους είναι ότι η 1η έχει να κάνει με διατεταγμένη λίστα, ενώ η 2η με μη διατεταγμένης συλλογής.

Όπως γνωρίζουμε κόμβος (node) είναι και κάθε attribute. Οπότε έχουμε την Attr extends Node, δηλαδή είναι ότι και το interface Node, αλλά με περαιτέρω ιδιότητες και μεθόδους..

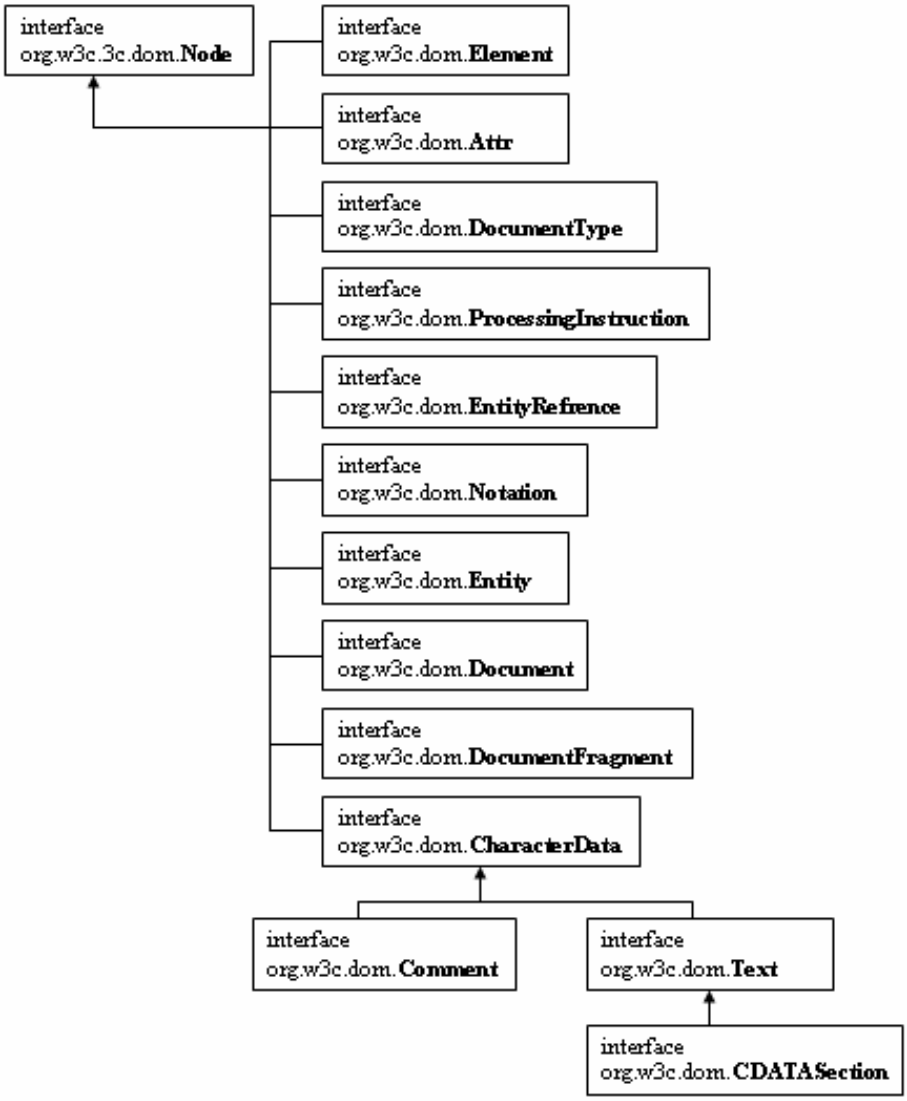
Το ίδιο το Element Node, σαν επέκταση του Node έχει ιδιαίτερο ενδιαφέρον, καθόσον περιλαμβάνει, μεταξύ άλλων, και την διαχείριση των attributes (εισαγωγή νέων, διαγραφή παλαιών, εξαγωγή ενός για επεξεργασία, κλπ). Βεβαίως το Element είναι το μόνον Node που έχει attributes (ενσωματωμένα σε NamedNodeMap).

Η getDocumentElement() επιστρέφει το root element. Αυτό είναι της class Element, η οποία είναι εξειδίκευση της Node (Element extends Node, πράγματι έχουμε κατά το Σχήμα 2.1 και 'Element Node'). Κάθε νέο Element (οποιασδήποτε class) το οποίο θέλουμε να κατασκευάσομε και να εντάξουμε στο υπάρχον μοντέλο XML κατά DOM, ανήκει αναγκαστικά σε κάποιο document. Γι' αυτό υπάρχουν τα διάφορα createXYZ (createNS λαμβάνοντας υπ' όψιν το namespace) που επιστρέφουν αντικείμενο τύπου XYZ. Το κάθε τέτοιο xyz είναι (εξειδίκευση του) node και μπορεί να αποτελέσει κατόπιν όρισμα εισόδου στις σχετικές μεθόδους του Node interface.

Σημειώνουμε επίσης ότι υπάρχουν και μία σειρά ορισμών public interface Xyz extends Node {…} όπου το Xyz είναι Attr, CDATASection, CharacterData, Document, DocumentFragment, DocumentType, Element, Entity, EntityReference, Notation, ProcessingInstruction. Τα περισσότερα είναι γνωστά μας σαν περιπτώσεις (εξειδικεύσεις) του Node κατά DOM. Τέλος υπάρχουν και τα Comment και Text σαν εξειδικεύσεις της class CharacterData, ενώ το CDATASection είναι εξειδίκευση της Text.

Η πλήρης εικόνα των classes (interfaces) του DOM API και της ιεραρχίας των, δίδεται στο παρακάτω Σχήμα 2.3. Παρατηρούμε ότι η ιεραρχία αυτή αντικατοπτρίζει την κληρονομικότητα της Java (ή του αντικειμενοστραφούς προγραμματισμού) και όχι την ιεραρχία ενός κειμένου XML. Το κάθε interface x προσθέτει μεθόδους επιπλέον αυτών που βρήκε στον πατέρα του y, δηλ. x extends y. Με την class x (κάθε interface είναι και class !) μπορούμε να κάνουμε περισσότερα πράγματα διότι η x είναι πιο εξειδικευμένη από την y. Χαρακτηριστική περίπτωση είναι η Document, η οποία στην έννοια του XML περιέχει ένα Node, αλλά εδώ σαν class (ή interface) είναι ιεραρχικά παιδί του Node. Για αυτό κληρονομεί

όλες τις μεθόδους του και έχει και άλλες επιπλέον. Σε όλες τις περιπτώσεις δίδονται οι προς χρήση μέθοδοι υπό την μορφή των interfaces. Οι classes ευρίσκονται στα packages org.w3c.dom.xyz.



Σχήμα 2.3 : Ιεραρχία του DOM API

XML και JDOM (Java Document Object Model)

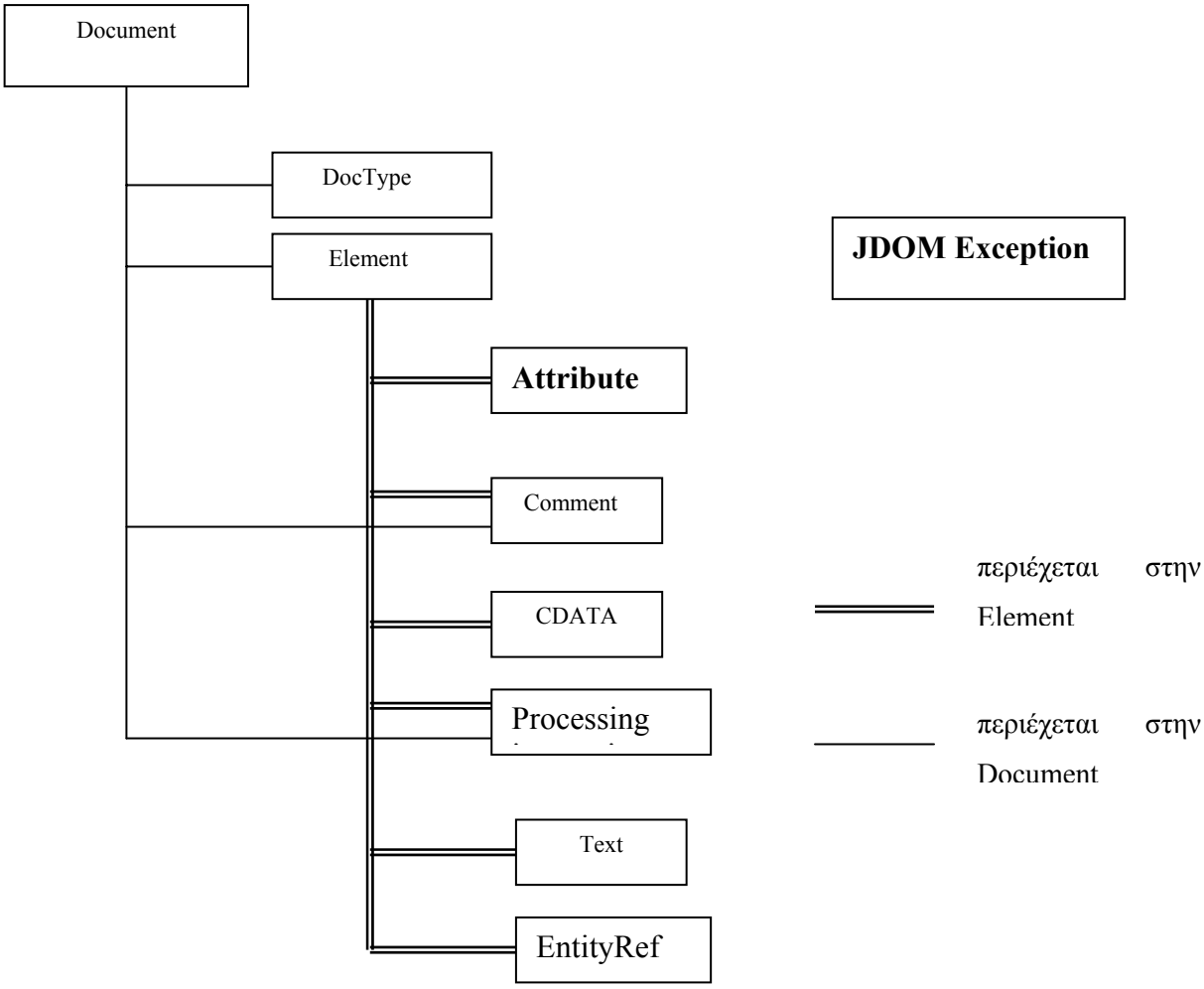
Το **Java Document Object Model (JDOM)** μας προσφέρει πρόσβαση σε ένα κείμενο XML μέσω μίας δενδρικής δομής (όπως και το DOM), αλλά εδώ αποκλειστικά μέσω ανοικτού Java API. **Το JDOM είναι διαφορετικό του Java API του DOM, που παρουσιάσαμε στην προηγούμενη ενότητα**. Με τις μεθόδους του JDOM επεξεργαζόμαστε την δομή που δημιουργείται σαν αναπαράσταση του XML. Ένα αντικείμενο Document είναι η αναπαράσταση του όλου κειμένου XML και συγχρόνως περιέχει τα άλλα JDOM αντικείμενα που αντιστοιχούνται σύμφωνα με τις ονομασίες τους απ΄ ευθείας στις αντίστοιχες του XML. Το (XML) Element του JDOM δεν περιέχει τα μέρη του ομαδοποιημένα σε NodeList και NamedNodeMap όπως στο DOM. Επιπλέον μπορούμε να επηρεάσουμε (θέσουμε, αλλάξουμε, σβήσουμε) το περιεχόμενο ενός (XML) Element, χωρίς να μεταφερόμαστε και να δουλεύουμε σε κάθε επιμέρους 'κόμβους' (Attribute, Text, CDATA Node, κλπ) ξεχωριστά. Για παράδειγμα η getAttributes() μέθοδος της Element class επιστρέφει μια (Java) List. Επιπλέον δεν υπάρχει εδώ TextNode. Το κείμενο (text) ενός Element απλά επιστρέφεται από την getText() μέθοδο του Element class. Κατά συνέπειαν, αντικείμενα όπως Element, Attribute, ProcessingInstruction, Comment, κλπ μπορούν να δημιουργηθούν με το New. Τούτο χρησιμοποιείται για την κατασκευή ενός XML κειμένου εκ του μηδενός. Η ουσιώδης διαφορά έγκειται ότι ενώ το JDOM δεν νοείται έξω από την Java, οι δομές του DOM είναι ανεξάρτητες και πέραν από συγκεκριμένη γλώσσα προγραμματισμού.

Το JDOM παρέχει τον τρόπο **πρόσβασης** στα στοιχεία του κειμένου XML, αλλά για την **εισαγωγή / εξαγωγή** του XML αναγκαστικά βασίζεται σε κάποιον parser (συγκεκριμένα τον SAX parser που θα δούμε παρακάτω) ή στην δενδρική δομή της ιεραρχίας του DOM (αν διατίθεται τέτοια).

Εισαγωγή ενός XML μέσω του SAXparser

Θα δούμε τώρα το μέρος της εισαγωγής ενός XML μέσω του SAX parser. Δημιουργούμε πρώτα έναν SAXBuilder. Αυτός χρησιμοποιώντας το SAX, μετατρέπει με την μέθοδό του build το κείμενο του αρχείου xml σε ένα document, δηλαδή στο αντικείμενο doc της γενικότερης class Document του Σχήματος 2.4. Έπειτα με τη συνάρτηση processElement διερχόμαστε αναδρομικά όλα τα elements (στοιχεία) του doc μέσω JDOM.

```
SAXBuilder builder = new SAXBuilder();
Document doc = builder.build(new FileInputStream("test.xml"));
processElement(doc.getRootElement());
```



Σχήμα 2.4 : Τα αντικείμενα (object model) του JDOM με βάση την σχέση 'περιέχεται'

Η μέθοδος processElement εξάγει το περιεχόμενο του κάθε στοιχείου σε μία list (mixedContent). Εδώ τοποθετούνται όλα τα αντικείμενα που αποτελούν το περιεχόμενο του τρέχοντος element, εκτός από τα attributes. Τα attributes δεν θεωρούνται περιεχόμενο, αλλά ιδιότητες του Element. Τα attributes εξάγονται μέσω μίας λίστας κατ΄ ευθείαν από το element (getAtributes() για όλα, getAttribute(String name) για ένα προς ένα – αλλά τότε πρέπει να γνωρίζουμε εκ των προτέρων το όνομα). Επίσης το κείμενο μπορεί να εξαχθεί και αυτό απ΄ ευθείας από το Element με την getText(). Για αναφορά στα πραγματικά περιεχόμενα του Element, χρησιμοποιείται ένα γενικό αντικείμενο (object), το οποίο κατά περίπτωση γίνεται casted σαν Text, Comment, κλπ, Element (οπότε τώρα παιδί του τρέχοντος), βλ. Σχήμα 2.4. Τα στοιχεία της mixedContent είναι αντικείμενα με διάφορες ιδιότητες και χρησιμοποιούμε την getText για τα Text και Comment προκειμένου να τυπώσουμε το κείμενό τους σαν

String. Το αντικείμενο CDATA είναι αντικείμενο Text (η CDATA είναι κατασκευασμένη σαν subclass της Text). Το αντικείμενο ProcessingInstruction έχει κατά την θεωρία PITarget (η εφαρμογή στην οποίαν απευθύνεται η ακολουθούσα εντολή), δεδομένα (DATA, δηλ. η εκφώνηση της εντολής) και βεβαίως πατέρα στον οποίον ανήκει (Parent).

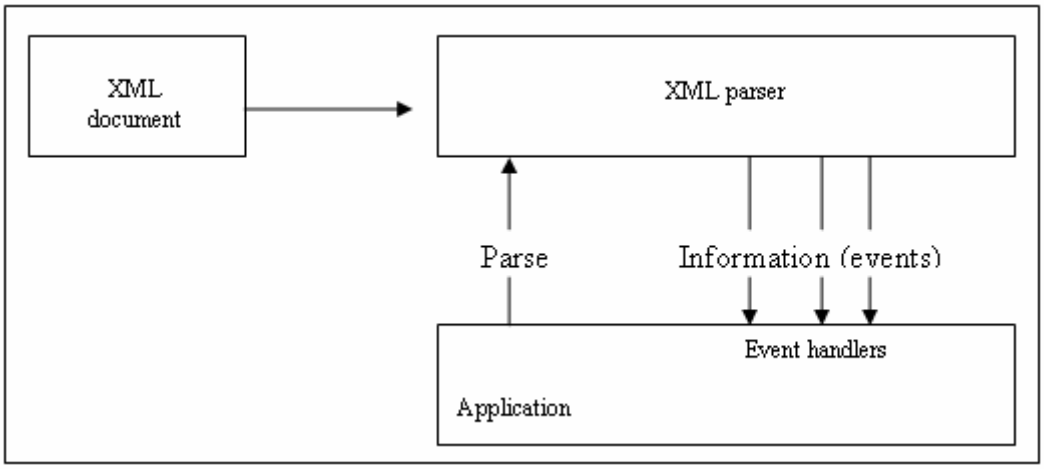
Σειριακή Εξαγωγή Δομής κατά JDOM σε αρχείο XML

Στην αντίθετη κατεύθυνση, για την σειριακή εξαγωγή σε αρχείο .xml, χρειάζεται ένας XMLOutputter. Κατασκευάζεται στην αρχή και του παραδίδουμε στο τέλος το υπό δημιουργία doc και το επιθυμητό FileOutputStream (με το όνομα του προς δημιουργία αρχείου.xml). Κατασκευάζουμε το doc εκ του μηδενός. Χρησιμοποιείται το new καθώς και η μέθοδος addContent του κάθε δημιουργηθέντος element για να πάμε πιο κάτω (αρχίζοντας από το root element0). Επίσης το root element ορίζεται σαν doc. Με την setxxxx (xxxx για Attribute, Text) εμπλουτίζουμε ένα element με τις ιδιότητες και το κείμενο του. Τα αντικείμενα των JDOM classes συμπεριφέρονται όπως κάθε object της Java.

```
XMLOutputter outputter = new XMLOutputter("  ", true);
FileOutputStream output = new FileOutputStream("test.xml");
Element rootElement = new Element("root");
Document doc = new Document(rootElement);
```

XML και SAX (Simple API for XML)

Στη βάση όλων των παραπάνω ευρίσκεται το SAX (Simple API for XML), προσφερόμενο από τον SAX parser, τον οποίο μπορούμε να φανταστούμε σαν έναν σειριακό αναγνώστη (SAX reader) του κειμένου XML, χαρακτήρα προς χαρακτήρα. Ο SAX parser αναγνωρίζοντας τα διάφορα μέρη του XML αρχείου, δημιουργεί αντίστοιχα συμβάντα (events), στα οποία το δικό μας πρόγραμμα οφείλει να ανταποκριθεί σύμφωνα με τις επιθυμίες μας. Ο SAX parser διαβάζει την ταινία (συρμό χαρακτήρων) του XML κειμένου και ανιχνεύει και σταματά σε συγκεκριμένα σημεία. Σε κάθε σταμάτημα δημιουργείται ένα συμβάν (event) προς τον δικό μας event handler. Μόλις ο event handler αντιμετωπίσει το event, συνεχίζει ο parser μέχρι το επόμενο σημείο. Στο Σχήμα 2.5 φαίνεται η σειριακή είσοδος του κειμένου XML στον SAX parser. Αυτός ελέγχεται από την εφαρμογή στην οποία δηλώνει πίσω (callbacks) συμβάντα, δηλαδή την ανίχνευση των επί μέρους μερών του υπό ανάγνωση XML



Σχήμα 2.5 : Γενική αρχή SAX parser

Ο ContentHandler (ενσωματωμένος σε μορφή interface μέσα στον SAX parser) είναι ένας event handler κατά τα γνωστά της Java, ο οποίος δημιουργεί κλήσεις πίσω στην εφαρμογή κάθε φορά που ένα αξιοπρόσεκτο γεγονός (parsing event) διαπιστώνεται κατά την σειριακή ανάγνωση του κειμένου XML. Ένα interface είναι ο ορισμός μεθόδων (π.χ. startDocument, startElement, κλπ). Στην περίπτωσή μας η πλειοψηφία αυτών των μεθόδων θα καλείται όταν διαπιστώνονται οι ομώνυμες περιπτώσεις (αρχή του κειμένου, στοιχείου XML αντίστοιχα). Η κλάση που θα δημιουργεί ο χρήστης και θα υλοποιεί το ContentHandler interface, οφείλει να προγραμματίσει όλες τις μεθόδους αυτές, έστω και με τετριμμένο τρόπο (άμεση επιστροφή χωρίς καμία ενέργεια). Σαν τον ContentHandler, υπάρχουν στον SAX και άλλα τρία παρόμοια interfaces ErrorHandler, DTDHAndler, EntityResolver, με αντίστοιχη φιλοσοφία χρήσης. Δημιουργούμε πρώτα ένα αντικείμενο SAXParser και μετά μέσω αυτού ένα αντικείμενο XMLReader, το οποίο και θα διαβάσει το XML κείμενό μας. Στον XMLReader αντικείμενο προσκολλάται και το ContentHandler αντικείμενο που δημιουργήσαμε. Τέλος καλούμε την μέθοδο parse, η οποία αυτόματα δημιουργεί όλα τα callbacks, δηλαδή τις κλήσεις που γίνονται κάθε φορά που στην διαδικασία της σειριακής ανάγνωσης ο parser ανακαλύπτει και διαπιστώνει το αντίστοιχο συμβάν. Ανάλογα με το τι βρίσκει, μας το διαθέτει μέσω αυτής της κλήσης και διάφορα σχετικά αντικείμενα που ενδιαφέρουν, π.χ. στην περίπτωση του element όλη την πληροφορία που σχετίζεται με το όνομά του. Στο συμβάν εμφάνισης νέου element, μας δίδεται η ευκαιρία να διαβάσουμε και τα attributes. Μέσω του interface Attributes (σαν Vector της Java) δίνουμε στο παράδειγμα ένα εκ των προτέρων γνωστό μας όνομα attribute, μας επιστρέφεται η θέση πού βρίσκεται (μηδενική, πρώτη, δεύτερη κλπ μέσα στο element) και κατόπιν χρησιμοποιώντας αυτήν διαβάζουμε την τιμή.

Σημειωτέον, ότι χωρίς επιπλέον δική μας προσπάθεια δεν υπάρχει ουσιώδης πληροφορία για το που βρίσκεται μέσα στο XML η διαδικασία του parsing , π.χ. σε ποιο βάθος του δένδρου. Κρατώντας στην εφαρμογή μας μία προϊστορία είναι δυνατόν να καταγράφουμε δομημένη την πληροφορία για το τι συνέβη / διαβάστηκε σε κάθε στιγμή. Δεν υπάρχει εν τούτοις κανένας τρόπος να γνωρίζουμε τι έπεται, δεδομένου ότι το δικό μας μέρος (της εφαρμογής) προχωρά παράλληλα με τον SAXParser και XMLReader, ο οποίος δεν έχει ακόμη διαβάσει πιο πέρα. Αν πάλι επιθυμούμε γνώση του σημείου, όπου βρίσκεται ανά πάσα στιγμή ο parser χρησιμοποιούμε την class Locator. Δημιουργούμε το αντικείμενό της lc και το δηλώνουμε στον ContentHandler χρησιμοποιώντας την μέθοδο setDocumentLocator με την μοναδική εντολή this.lc = locator; Μία από τις μεθόδους της είναι η getLineNumber, που μας δίδει την θέση (αριθμό γραμμής) μέσα στο αρχείο, όπου διαπιστώθηκε από τον SAXParser το συγκεκριμένο συμβάν.

```
SAXParserFactory spf = SAXParserFactory.newInstance();
SAXParser sp = spf.newSAXParser();
XMLReader sr = sp.getXMLReader();
sr.setContentHandler(this);
sr.parse("test.xml");
```

Οι μέθοδοι του ContentHandler που πρέπει να υλοποιηθούν είναι :

setDocumentLocator(Locator locator), **startDocument**(), **endDocument**(), **startPrefixMapping**(String prefix, String uri), **endPrefixMapping**(String prefix) , **startElement**(String namespaceURI, String localName, String qName, Attributes atts), **endElement**(String namespaceURI, String localName, String qName), **characters**(char ch[], int start, int lenght) , **ignorableWhitespace**(char ch[], int start, int lenght) , **processingInstruction**(String target, String data) και **skippedEntity**(String name)

Με τον SAX parser βρισκόμαστε στο κατώτερο προγραμματιστικά περιβάλλον που μας δίνει πλήρη πρόσβαση στο XML. Η ένταξη των δικών μας επιθυμιών (εφαρμογή μας) είναι πιο επίπονη από ότι με το DOM ή JDOM, αλλά οι διαθέσιμες δυνατότητες και η ταχύτητα εδώ υπερτερούν. Δεν περιμένουμε να δημιουργηθεί και να μας προσφερθεί η πλήρης δομή του XML σύμφωνα με το DOM ή JDOM μοντέλο, αλλά ειδοποιούμεθα και μπορούμε να ανταποκριθούμε στα σχετικά συμβάντα on the fly, καθώς διαβάζεται το XML κείμενο και χωρίς να περιμένουμε το τέλος αυτής της ανάγνωσης.

2.3 Ερωτήσεις XML

Το θέμα που απασχολεί την παρούσα διπλωματική είναι οι XML ερωτήσεις. Μπορούμε να παρομοιάσουμε μια XML ερώτηση με μια SQL ερώτηση σε μία σχεσιακή βάση δεδομένων, αν και οι μέθοδοι αποτίμησης των δύο ειδών ερωτήσεων διαφέρουν πολύ. Οι δύο γλώσσες που συμβάλλουν στην τυποποίηση XML ερωτήσεων είναι η **XPath** και η **XQuery**. Η XPath είναι μια γλώσσα για παραστάσεις διαδρομών. Η XQuery είναι η κατεξοχήν γλώσσα ερωτήσεων πάνω σε XML δεδομένα και μπορούμε να την παρομοιάσουμε με τη γλώσσα SQL για ερωτήσεις πάνω σε δεδομένα βάσεων δεδομένων. Η XQuery χτίζεται πάνω στις εκφράσεις της XPath. Χρησιμοποιεί παραστάσεις διαδρομής για να διασχίσει τα στοιχεία ενός XML εγγράφου και χρησιμοποιεί κατηγορήματα επιλογής για να περιορίσει τα δεδομένα από τα XML έγγραφα και να πάρει μόνο αυτά που πληρούν τις συνθήκες που ορίστηκαν στα κατηγορήματα. Όμως με την XQuery περιγράφουμε τη σημασιολογία της ερώτησης, δεν υπεισερχόμαστε στο πώς θα γίνει η αποτίμηση αυτή, που είναι αυτό που απασχολεί την διπλωματική μας.

Στην παρούσα διπλωματική ενδιαφερόμαστε κυρίως για ερωτήσεις με μερικό προσδιορισμό δομής. Τέτοια ερωτήματα είναι χρήσιμα γιατί μπορεί να θέλουμε να κάνουμε το ίδιο ερώτημα σε πολλαπλά έγγραφα που μπορεί να έχουν διαφορετική δομή ή που μπορεί να μην γνωρίζουμε την ακριβής δομή τους. Η XPath έχει τη δυνατότητα να εκφράζει τέτοια ερωτήματα με τους ανάστροφους άξονες. Για παράδειγμα το ερώτημα **//title[descendant-or-self::*[ancentor-or-self::year] [ancentor-or-self::author]]** ψάχνει για κόμβους title σε μονοπάτια που περιέχουν επίσης τους κόμβους year και author, αλλά καμία καθορισμένη σχέση δεν απαιτείται μεταξύ των κόμβων. Ο Olteanu και άλλοι [17,18] έχουν δείξει ότι οι XPath ερωτήσεις με ανάστροφους άξονες μπορούν να γραφτούν ισοδύναμα ως ένα σύνολο από δενδρικές ερωτήσεις ολικής διάταξης. Ωστόσο δείχνουν επίσης ότι ο μετασχηματισμός αυτός μπορεί να οδηγήσει σε εκθετική έκρηξη ως προς τον αριθμό των δενδρικών ερωτήσεων. Επίσης ο Gottlob και άλλοι [19] έχουν αποδείξει ότι η πολυπλοκότητα της XPath είναι P-hard.

Οπότε για να αντιμετωπίσουμε αυτά τα προβλήματα πρέπει να υιοθετήσουμε νέες γλώσσες XML ερωτήσεων μονοπατιού που αίρουν την ανάγκη για τον προσδιορισμό της ολικής διάταξης μεταξύ των κόμβων και επιτρέπουν μερικώς καθορισμένες δομές. Ακόμα χρειάζονται να βρεθούν αλγόριθμοι που θα αντιμετωπίσουν αποδοτικά τέτοιου είδους ερωτήσεις.

2.4 Σχετικές Εργασίες

Οι ερωτήσεις για δεδομένα XML περιλαμβάνουν δομικά πρότυπα αποτελούμενα από στοιχεία-κόμβους και δομικές σχέσεις μεταξύ τους. Το ταίριασμα αυτών των προτύπων σε ένα XML δέντρο είναι βασική λειτουργία των αλγορίθμων αποτίμησης ερωτήσεων XML. Πολλοί αλγόριθμοι έχουν ήδη προταθεί για την αποτίμηση ειδικών μορφών ερωτήσεων, όπως οι δυαδικές δομικές σχέσεις (binary structural relationships), οι ερωτήσεις μονοπατιού και οι ερωτήσεις δέντρου.

Η πρώιμη έρευνα στο χώρο εστιάζει στην αποτίμηση δυαδικών σχέσεων. Στο [16] παρουσιάζεται ο αλγόριθμος **Multi-Predicate Merge Join (MPMGJN)** για την αποτίμηση αυτών των ερωτήσεων. Οι **Al-Khalifa et al.** [6] εισάγουν μια οικογένεια αλγορίθμων στοίβας που είναι πιο αποτελεσματικοί από τον MPMGJN, καθώς δεν απαιτούν πολλαπλή διάσχιση του XML δέντρου.

Πολλές μελέτες καταπιάνονται με την εφαρμογή διαφόρων μορφών ευρετηρίων για την επιτάχυνση της αποτίμησης των ερωτήσεων αυτών [9,12,15]. B+-δέντρα μπορούν να χρησιμοποιηθούν ώστε να αποφευχθεί η επεξεργασία κόμβων που δε συμμετέχουν στην απάντηση της ερώτησης [9] Ένα νέο δυναμικό ευρετήριο ειδικά μελετημένο για δεδομένα XML, το **XR-tree**, εισήχθηκε από τους Jiang et al [12]. Οι Yang et al. [15] συνδύασαν ευρετήρια μονοπατιού και πληροφορία προγόνου για να παράγουν τους **virtual cursors**, ένα νέο ευρετήριο που διαχειρίζεται αποτελεσματικά τα δεδομένα.

Τεχνικές διάσπασης της ερώτησης σε δυαδικές δομικές σχέσεις και συνένωσης των επιμέρους αποτελεσμάτων μπορούν να εφαρμοστούν σε πολλές μορφές ερωτήσεων, όπως οι ερωτήσεις μονοπατιού ή δέντρου. Αυτό όμως είναι ανεπαρκές λόγω του μεγάλου αριθμού ενδιάμεσων αποτελεσμάτων. Για να αποφύγουν αυτό το πρόβλημα, οι Bruno et al. [1] παρουσίασαν δυο αλγορίθμους στοίβας, τον **PathStack** και τον **TwigStack**, για την αποτίμηση ερωτήσεων μονοπατιού και δέντρου αντίστοιχα. Ο PathSatck είναι βέλτιστος για ερωτήσεις μονοπατιού, ενώ TwigStack είναι βέλτιστος για ερωτήσεις δέντρου χωρίς σχέσεις γονέα-παιδιού.

Πολλοί ερευνητές έχουν καταπιαστεί με διάφορες επεκτάσεις του **TwigStack**. Για παράδειγμα, στο [14], ο αλγόριθμος **TwigStackList** αποτιμά αποτελεσματικά δεντρικές

ερωτήσεις με σχέσεις γονέα-παιδιού. Επιπλέον, στο [8], ο αλγόριθμος **Twig2Stack** αποτιμά γενικευμένες ερωτήσεις δέντρου με προαιρετικές δομικές σχέσεις. Οι Chen et al. [7] προτείναν αλγορίθμους που χειρίζονται ερωτήσεις σε δεδομένα με δομή **dag**. Μέθοδοι αποτίμησης ερωτήσεων δέντρου με κατηγορούμενα **OR** αναπτύσσονται στο [11]. Στο [13], το ευρετήριο **XR-tree** [12] χρησιμοποιείται για την αποφυγή του χειρισμού δεδομένων που δε συμμετέχουν στην απάντηση της ερώτησης. Το [10] παρουσιάζει τον αλγόριθμο **TwigOptimal**, που χρησιμοποιεί τους virtual cursors [15] για να επιταχύνει τη διάσχιση του XML δέντρου.

3

Δεδομένα XML και Ερωτήσεις Μονοπατιού

3.1 Δεδομένα XML

Μια βάση δεδομένων XML μοντελοποιείται συνήθως από μία δενδρική δομή. Οι κόμβοι του δέντρου αναπαριστούν στοιχεία (elements), χαρακτηριστικά (attributes) ή τιμές (values). Οι ακμές του δέντρου αναπαριστούν σχέσεις στοιχείου-υποστοιχείου, στοιχείου-χαρακτηριστικού ή στοιχείου-τιμής.

Για την αναπαράσταση της θέσης των κόμβων στο δέντρο έχουν προταθεί διάφορες μορφές κωδικοποίησης. Η πιο κοινή αναφέρεται ως κωδικοποίηση θέσης.

3.2 Κωδικοποίηση Θέσης

Η κωδικοποίηση θέσης είναι μια αποδοτική τεχνική για την αναπαράσταση της θέσης των κόμβων σε ένα XML δέντρο.

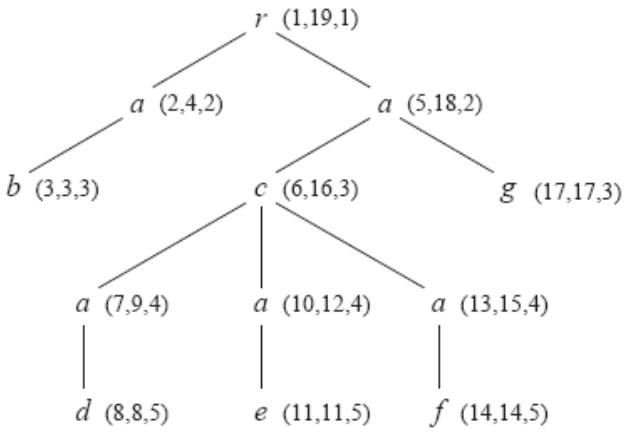
Οι κόμβοι αναπαριστώνται με τριπλέτες της μορφής (start, end , level). Αν έχουμε πολλά δέντρα, τότε υπάρχει και ένα τέταρτο πεδίο treeID. Οπότε έχουμε τα πεδία :

- start : πρώτη επίσκεψη του κόμβου
- end : τελευταία επίσκεψη του κόμβου
- level : επίπεδο κόμβου στο δέντρο
- treeID : αριθμός εγγράφου

Η πρώτη και τελευταία επίσκεψη κάθε κόμβου καθορίζονται από την κατά βάθος διάσχιση του δέντρου.

Με την κωδικοποίηση θέσης απλοποιείται ο έλεγχος δομικών σχέσεων :

- ο κόμβος n_1 είναι πρόγονος του κόμβου n_2 αν και μόνο εάν $n_1.start < n_2.start$ και $n_2.end < n_1.end$
- ο κόμβος n_1 είναι πατέρας του κόμβου n_2 αν και μόνο εάν $n_1.start < n_2.start$ και $n_2.end < n_1.end$ και $n_1.level = n_2.level - 1$.



Σχήμα 3.1 : Κωδικοποίηση θέσης ενός XML δέντρου

3.3 Ερωτήσεις Μονοπατιού Πλήρους Δομής

Σε αυτήν την ενότητα παρουσιάζουμε τις Ερωτήσεις Μονοπατιού (Path Queries, PQs), εξηγώντας τη σύνταξή τους και τη σημασιολογία τους.

Σύνταξη

Μία **Ερώτηση Μονοπατιού** (**Path Query**, **PQ**) είναι μία αλυσίδα από κόμβους, όπου διαδοχικοί κόμβοι συνδέονται με σχέσεις προγόνου-απογόνου (εκφράσεις της μορφής a//b) καθώς και με σχέσεις πατέρα-παιδιού (εκφράσεις της μορφής a/b).

Συγκεκριμένα:

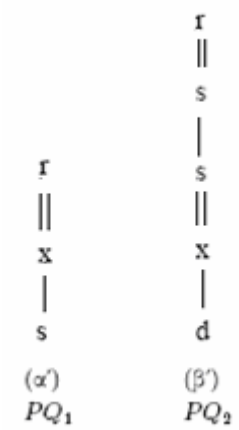
Το Σχήμα 3.2 δείχνει δύο PQs. Το PQ_1, για παράδειγμα, περιλαμβάνει μία σχέση προγόνου-απογόνου από τον κόμβο ρίζα στον κόμβο που σημαίνεται από x και μία σχέση πατέρα-παιδιού από τον κόμβο που σημαίνεται από x στον κόμβο που σημαίνεται από s. Ιδιαίτερο, μάλιστα, ενδιαφέρον παρουσιάζει η δεύτερη ερώτηση, όπου δύο κόμβοι σημαίνονται από το ίδιο στοιχείο s. Τέτοιες ερωτήσεις, όπου δύο τουλάχιστον κόμβοι της ερώτησης σημαίνονται από το ίδιο στοιχείο, θα ονομάζονται στο εξής αναδρομικές.

(a) PQ1 = r//x/s

(b) PQ2 = r//s/s//x/d

Σχήμα 3.2 : Παραδείγματα Ερωτήσεων Μονοπατιού

Απεικονίζουμε γραφικά τα PQs χρησιμοποιώντας ορολογία γράφων. Κάθε κόμβος της ερώτησης αντιστοιχεί σε έναν κόμβο γράφου σημασμένου από το αντίστοιχο στοιχείο. Οι σχέσεις πατέρα-παιδιού και προγόνου-απογόνου απεικονίζονται χρησιμοποιώντας μονές (-) και διπλές (=) γραμμές μεταξύ των αντίστοιχων κόμβων. Η γραφική αναπαράσταση των δύο PQs του Σχήματος 3.2 απεικονίζεται στο Σχήμα 3.3.



Σχήμα 3.3 : Γραφική αναπαράσταση ερωτήσεων μονοπατιού

3.4 Ερωτήσεις Μονοπατιού Μερικής Δομής

Σε αυτήν την ενότητα παρουσιάζουμε τα Ερωτήματα Μονοπατιού Μερικής Δομής, δίνοντας τη σύνταξη και τη σημασιολογία.

Σύνταξη

Μία ερώτηση μονοπατιού μερικής δομής (Partial Path Query, PPQ) είναι μία ερώτηση μονοπατιού, όπου όμως η δομή δεν είναι πάντα αυστηρά καθορισμένη. Τυπικά:

Ορισμός

Μία **Ερώτηση Μονοπατιού Μερικής Δομής** (**Partial Path Query** , **PPQ**) σε ένα XML δέντρο T είναι ένα σύνολο από σχέσεις διαδοχής ορισμένες πάνω σε ένα μη κενό σύνολο από κόμβους, δηλαδή ένα σύνολο από εκφράσεις της μορφής r/b_j ή a_i/b_j (σχέσεις πατέρα-παιδιού) και/ή της μορφής $r//b_j$ ή $a_i//b_j$ (σχέσεις προγόνου-απογόνου), όπου r είναι ο κόμβος ρίζα της ερώτησης, οι a_i και b_j είναι κόμβοι σημασμένοι από τα στοιχεία a και b, αντίστοιχα, και τόσο ο a όσο και ο b είναι στοιχεία στο T.

Το Σχήμα 3.4 απεικονίζει τέσσερα τέτοια ερωτήματα.

$q_1 = \{a_1/c_1, c_1//e_1\}$

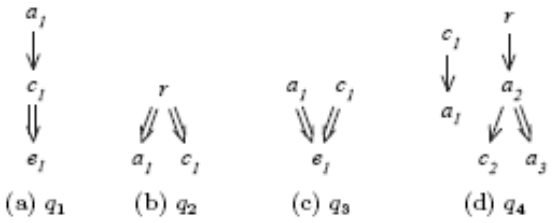
$q_2 = \{r//a_1, r//c_1\}$

$q_3 = \{a_1//e_1, c_1//e_1\}$

$q_4 = \{c_1/a_1, r/a_2, a_2/c_2, a_2//a_3\}$

Σχήμα 3.4 : Ερωτήματα Μονοπατιού με Πρότυπα Μερικής Δομής

Παριστάνουμε γραφικά τις ερωτήσεις αυτές χρησιμοποιώντας ορολογία γράφων. Κάθε κόμβος της ερώτησης αντιπροσωπεύεται από έναν κόμβο στο γράφο που σημαίνεται από το αντίστοιχο στοιχείο. Οι σχέσεις πατέρα-παιδιού και προγόνου-απογόνου απεικονίζονται χρησιμοποιώντας μονά και διπλά βέλη αντίστοιχα μεταξύ των αντίστοιχων κόμβων. Η γραφική αναπαράσταση των τεσσάρων ερωτημάτων του Σχήματος 3.4 παρουσιάζεται στο Σχήμα 3.5.



Σχήμα 3.5 : Γραφική αναπαράσταση ερωτήσεων

Το ερώτημα q_2 του Σχήματος 3.5 έχει δομή παρόμοια με αυτή ενός κλαδιού σε μια ερώτηση με πρότυπο δενδρικής δομής, ωστόσο η σημασιολογία είναι διαφορετική. Όταν πρόκειται για ερωτήματα μονοπατιού μερικής δομής, οι κόμβοι a_1 και c_1 πρέπει να βρίσκονται στο ίδιο μονοπάτι σε όλες τις απαντήσεις, ενώ στις δενδρικές ερωτήσεις οι κόμβοι αυτοί θα μπορούσαν να ταιριάζουν ακόμη κι αν ανήκαν σε διαφορετικά μονοπάτια της XML βάσης δεδομένων.

Ο χρήστης έχει τη δυνατότητα να καθορίσει τη δομή μίας τέτοιας ερώτησης, πλήρως, μερικώς ή ακόμη και καθόλου. Από τη μια πλευρά, μία γλώσσα ερωτήσεων μερικής δομής επιτρέπει το σχηματισμό και τη διατύπωση ερωτήσεων που είναι πλήρως δομημένα μονοπάτια με σαφώς ορισμένες σχέσεις διαδοχής. Από την άλλη πλευρά, ένα PPQ μπορεί να μην περιέχει καμία απολύτως τέτοια σχέση διαδοχής, αλλά να περιέχει μόνο ένα σύνολο από κόμβους. Μεταξύ των δύο ακραίων περιπτώσεων, υπάρχουν ερωτήσεις που παρέχουν μερική περιγραφή της δομής χωρίς ωστόσο να καθορίζουν ένα μόνο συγκεκριμένο μονοπάτι.

Σημασιολογία.

Η απάντηση σε μία ερώτηση μονοπατιού μερικής δομής PPQ πάνω σε ένα XML δέντρο είναι ένα σύνολο από ακολουθίες κόμβων. Οι κόμβοι κάθε ακολουθίας πρέπει να βρίσκονται στο ίδιο μονοπάτι του δένδρου και να διατηρούν τις σχέσεις πατέρα-παιδιού και προγόνου-απογόνου του PPQ. Ορίζουμε με μεγαλύτερη αυστηρότητα την έννοια της απάντησης σε μία ερώτηση μονοπατιού μερικής δομής με την εισαγωγή κάποιων νέων ορισμών που ακολουθούν.

Ορισμός.

Εμβάπτισμα μίας ερώτησης μονοπατιού μερικής δομής Q σε ένα XML δέντρο T είναι μία αντιστοίχιση M από τους κόμβους του Q στους κόμβους του T έτσι ώστε :

(α) κάθε κόμβος του Q σημασμένος από το e αντιστοιχίζεται μέσω του M σε έναν κόμβο του T σημασμένου επίσης από το e

(β) οι κόμβοι του Q αντιστοιχίζονται μέσω του M σε κόμβους που βρίσκονται πάνω στο ίδιο μονοπάτι πάνω στο T

(γ) $\forall$ a_i/b_j (αντίστοιχα $a_i//b_j$) στο Q, $M(b_j)$ είναι παιδί (αντίστοιχα απόγονος) του $M(a_i)$ στο T.

Καλούμε **εικόνα** του Q κάτω από την εμπέδωση M και γράφουμε M(Q), μία ακολουθία κόμβων που περιλαμβάνει όλες τις εικόνες των κόμβων του Q κάτω από το M.

Έχοντας υπόψη μας τους προηγούμενους ορισμούς, είμαστε πλέον σε θέση να εισάγουμε την έννοια της απάντησης σε ένα ερώτημα μερικής δομής:

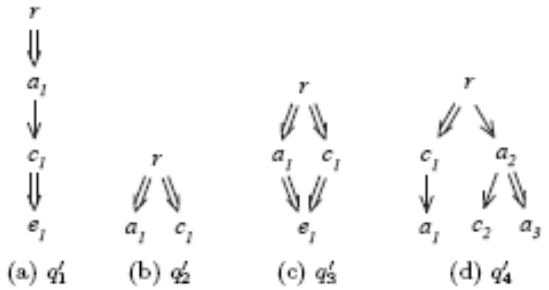
Ορισμός

Η **απάντηση** της ερώτησης Q πάνω στο T είναι το σύνολο όλων των εικόνων του Q κάτω από όλες τις δυνατές εμπεδώσεις του Q στο T.

Ως ένα παράδειγμα, ας θεωρήσουμε το ερώτημα q_3 του Σχήματος 3.5γ. Η απάντηση του q_3 στο XML δέντρο του Σχήματος 3.1 είναι : { $[\alpha_1:(5,18,2), c_1:(6,16,3), e_1:(11,11,5)]$, $[c_1:(6,16,3), a_1:(10,12,4), e_1:(11,11,5)]$ }.

Σημειώνουμε ότι οι ερωτήσεις μονοπατιού μερικής δομής μπορεί να περιέχουν διακριτούς κόμβους που σημαίνονται όμως από το ίδιο στοιχείο, για παράδειγμα οι κόμβοι c_1 και c_2 στο ερώτημα q_4 του Σχήματος 3.5. Οι εικόνες δύο τέτοιων κόμβων μπορεί να συμπέσουν, εκτός και αν σχετίζονται μέσα από μία ακολουθία σχέσεων διαδοχής μεταξύ τους, οπότε και αντιστοιχίζονται σε διαφορετικούς κόμβους του δέντρου T.

Μία ερώτηση μονοπατιού μερικής δομής μπορεί να περιλαμβάνει περισσότερους από έναν κόμβους χωρίς εισερχόμενες ακμές (δηλαδή πηγές). Όμως κάθε XML δέντρο έχει ως ρίζα έναν κόμβο που σημαίνεται από r. Έτσι μπορούμε να προσθέσουμε, αν δεν υπάρχει ήδη, έναν κόμβο r και διπλά βέλη προς όλους τους κόμβους πηγές της ερώτησης χωρίς να αλλάζουμε τη σημασιολογία της ερώτησης. Με αυτόν τον τρόπο, κάθε ερώτηση μπορεί να παρασταθεί με έναν κατευθυνόμενο γράφο με ρίζα, οπότε αρκεί να περιορίσουμε το ενδιαφέρον μας μόνο σε ερωτήσεις αυτής της μορφής. Το Σχήμα 3.6 παρουσιάζει τα ερωτήματα του Σχήματος 3.5 μετά από ένα τέτοιο μετασχηματισμό.



Σχήμα 3.6: Ερωτήματα με κόμβο-ρίζα.

3.5 Επεξεργασία Ερωτήσεων Μονοπατιού Μερικής Δομής

Δεδομένου ότι η δομή στις ερωτήσεις που μελετούμε σε αυτό το κεφάλαιο είναι μερικώς καθορισμένη, νέες σχέσεις διαδοχής μπορούν να εξαχθούν από αυτές που άμεσα καθορίζονται στην ερώτηση. Η αποτίμηση μίας τέτοιας ερώτησης μπορεί να επιταχυνθεί, αν τροποποιηθεί κατάλληλα με βάση τις παραγόμενες σχέσεις διαδοχής.

Εξαγωγή Δομικών Σχέσεων

Ας θεωρήσουμε το ερώτημα q'_4 του Σχήματος 3.6. Αφού ο a_2 είναι πατέρας του c_2 και πρόγονος του a_3 μπορούμε να συμπεράνουμε ότι ο c_2 είναι επίσης πρόγονος του a_3. Με άλλα λόγια, αφού ο c_2 είναι παιδί του a_1, ο a_3 δεν μπορεί να τοποθετηθεί ανάμεσά τους. Έχουμε επομένως:

Ορισμός

Μία **σχέση διαδοχής** p είναι **παραγόμενη** από το σύνολο των σχέσεων διαδοχής της ερώτησης μονοπατιού Q, αν και μόνο αν για κάθε εμβάπτισμα M του Q σε οποιοδήποτε XML δέντρο, το M ικανοποιεί την p. Το **κλείσιμο** ενός συνόλου από σχέσεις διαδοχής P είναι το σύνολο που περιλαμβάνει τις σχέσεις διαδοχής στο P, καθώς και όλες τις παραγόμενες σχέσεις διαδοχής από το σύνολο P.

Για να υπολογίσουμε το σύνολο ενός τέτοιου συνόλου σχέσεων P, εισάγουμε ένα σύνολο από συμπερασματικούς κανόνες, οι οποίοι απεικονίζονται στο Σχήμα 3.7. Έστω a_i, b_j και c_k

διακριτοί κόμβοι, x, y, z, και w μεταβλητές πάνω στο σύνολο των κόμβων, και r ο κόμβος ρίζα. Χρησιμοποιούμε το σύμβολο ├ για να δηλώσουμε ότι οι σχέσεις που προηγούνται του συμβόλου παράγουν τη σχέση που έπεται αυτού. Η απουσία σχέσεων που έπονται του συμβόλου ├ δηλώνει αξίωμα.

$$
\begin{aligned}
&\text{(IP1)} \quad \vdash\ r//a_i \\
&\text{(IP2)} \quad a_i/b_j \ \vdash\ a_i//b_j \\
&\text{(IP3)} \quad a_i//b_j,\ b_j//c_k \ \vdash\ a_i//c_k \\
&\text{(IP4)} \quad a_i/b_j,\ a_i//c_k,\ b \neq c \ \vdash\ b_j//c_k \\
&\text{(IP5)} \quad a_i/b_j,\ c_k//b_j,\ a \neq c \ \vdash\ c_k//a_i \\
&\text{(IP6)} \quad x/y,\ y/w,\ x//z,\ z//w \ \vdash\ x/z \\
&\text{(IP7)} \quad x/y,\ x//z,\ w/z,\ w//y \ \vdash\ x/z \\
&\text{(IP8)} \quad x/y,\ y/w,\ x/z \ \vdash\ z/w \\
&\text{(IP9)} \quad x//y,\ y//w,\ x/z \ \vdash\ z//w \\
&\text{(IP10)} \quad x/y,\ x/z,\ w/z \ \vdash\ w/y \\
&\text{(IP11)} \quad x//y,\ x/z,\ w//z \ \vdash\ w//y \\
&\text{(IP12)} \quad x/y,\ y/w,\ z/w \ \vdash\ x/z \\
&\text{(IP13)} \quad x//y,\ y//w,\ z/w \ \vdash\ x//z
\end{aligned}
$$

Σχήμα 3.7 : Κανόνες Εξαγωγής

Το επόμενο θεώρημα που παραθέτουμε χωρίς απόδειξη δείχνει ότι αυτοί οι κανόνες ορθά και πλήρως χαρακτηρίζουν την παραγωγή δομικών σχέσεων.

Θεώρημα

Έστω R ένα σύνολο από σχέσεις διαδοχής μίας ερώτησης μονοπατιού μερικής δομής και p μία δομική έκφραση που δεν ανήκει στο σύνολο R. Το σύνολο των κανόνων εξαγωγής του Σχήματος 3.7 είναι **συνεπές** (αν η p μπορεί να παραχθεί από το R χρησιμοποιώντας τους συμπερασματικούς κανόνες, τότε η p είναι πάντα μία παραγόμενη σχέση από το R), και **πλήρες** (αν η p μπορεί να παραχθεί από το R, τότε η p εμφανίζεται στο R, ή μπορεί να παραχθεί από R χρησιμοποιώντας τους συμπερασματικούς κανόνες).

Αν το σύνολο R των σχέσεων διαδοχής του Q ισούται με το κλείσιμο του R, λέμε ότι το Q είναι σε **πλήρη μορφή**.

Προφανώς, μπορεί να υπάρξει πολυωνυμικός μόνος αριθμός σχέσεων διαδοχής στο κλείσιμο ενός συνόλου σχέσεων διαδοχής. Στην πράξη, μόνο ένα μικρό ποσοστό του μέγιστου δυνατού αριθμού των σχέσεων εμφανίζονται στο κλείσιμο των σχέσεων διαδοχής μίας ερώτησης, και συνεπώς, το κόστος υπολογισμού του κλεισίματος είναι σχετικά ασήμαντο.

Ικανοποιησιμότητα Ερωτήσεων

Ο εντοπισμός της μη ικανοποιησιμότητας μιας ερώτησης μπορεί να οδηγήσει σε σημαντικά μικρότερους χρόνους εκτέλεσης των αλγορίθμων αποτίμησης, αφού μπορούμε να αποφύγουμε την πρόσβαση σε δεδομένα που ούτως ή άλλως δε θα μπορούσαν να μας οδηγήσουν σε μη κενά σύνολα απαντήσεων.

Ορισμός

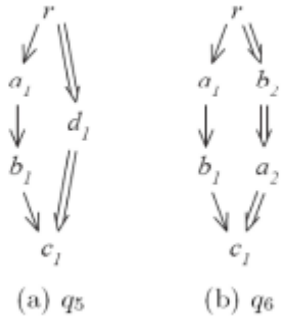
Μία ερώτηση μονοπατιού μερικής δομής καλείται **ικανοποιήσιμη**, αν και μόνο αν έχει μη κενή απάντηση σε κάποιο XML δέντρο. Διαφορετικά, καλείται μη ικανοποιήσιμη.

Σε αντίθεση με τις αυστηρώς δομημένες ερωτήσεις μονοπατιού, οι ερωτήσεις μονοπατιού μερικής μορφής μπορεί να είναι μη ικανοποιήσιμες. Ας θεωρήσουμε, για παράδειγμα, την ερώτηση q_5 του Σχήματος 3.8. Προφανώς, αυτή η ερώτηση είναι μη ικανοποιήσιμη, αφού κανένα μονοπάτι σε XML δένδρο δεν μπορεί να επαληθεύει και τις τέσσερις δομικές σχέσεις που περιγράφει. Το επόμενο θεώρημα παρέχει αναγκαίες και επαρκείς συνθήκες για ικανοποιησιμότητα.

Θεώρημα

Μία ερώτηση μονοπατιού μερικής δομής είναι **μη ικανοποιήσιμη**, αν και μόνο αν η πλήρης μορφή της δεν περιέχει έναν τετριμμένο κύκλο, ήτοι δύο δομικές σχέσεις της μορφής a//b και b//a.

Για παράδειγμα, ας θεωρήσουμε, τις ερωτήσεις q_5 και q_6 του Σχήματος 3.8. Μπορεί εύκολα να δει κανείς πως η πλήρης μορφή και των δύο ερωτήσεων περιέχει τον τετριμμένο κύκλο $r//a_1$ και $a_1//r$.



Σχήμα 3.8 : Δύο μη ικανοποιήσιμα ερωτήματα

Ο έλεγχος για ικανοποιησιμότητα ισοδυναμεί με τον έλεγχο της πλήρους μορφής της ερώτησης για τετριμμένους κύκλους. Η πολυπλοκότητα αυτού του ελέγχου είναι στη χειρότερη περίπτωση τετραγωνική ως προς τον αριθμό των κόμβων της ερώτησης. Δεδομένου ότι το πλήθος των κόμβων μίας ερώτησης δεν αναμένεται να είναι καν συγκρίσιμο προς το μέγεθος της XML βάσης δεδομένων, το κόστος για έλεγχο ικανοποιησιμότητας είναι στην πράξη ασήμαντο.

Πλεονάζοντες Κόμβοι

Μερικοί κόμβοι σε μία ερώτηση μπορούν να απομακρυνθούν χωρίς να επηρεάζεται η σημασιολογία της ερώτησης. Καλούμε αυτούς τους κόμβους πλεονάζοντες:

Ορισμός

Ένας κόμβος σε μία ερώτηση μονοπατιού μερικής δομής είναι **πλεονάζων**, αν και μόνο αν σε οποιαδήποτε εγγραφή της απάντησης αυτής της ερώτησης έχει την ίδια τιμή όπως ένας άλλος (όχι αναγκαστικά ο ίδιος) κόμβος της ερώτησης.

Οι πλεονάζοντες κόμβοι μπορούν να ανιχνευθούν με βάση το παρακάτω θεώρημα:

Θεώρημα

Ένας κόμβος y_j σε μία ερώτηση μονοπατιού μερικής δομής είναι **πλεονάζων**, αν και μόνο αν η πλήρης μορφή της ερώτησης περιλαμβάνει ένα από τα παρακάτω σύνολα δομικών σχέσεων:

(α) x/y_i και x/y_j (ισχυρή)

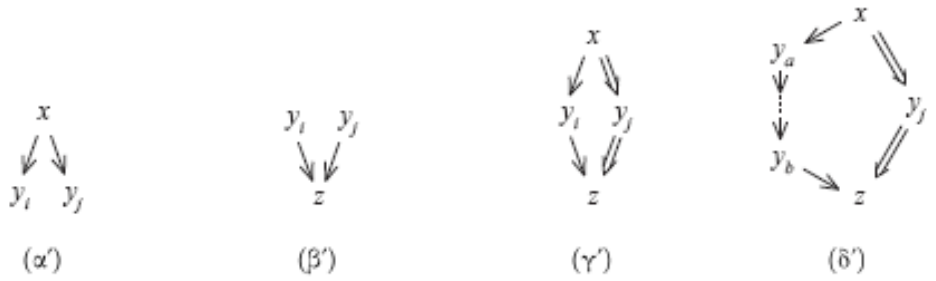
(β) y_i/z και y_j/z, (ισχυρή)

(γ) x/y_i, y_i/z, $x//y_j$ και $y_j//z$ (ισχυρή)

(δ) x/y_a, … , y_b/z, $x//y_j$ και $y_j//z$ (ασθενής)

όπου x,y,z είναι κόμβοι της ερώτησης, ενώ y_i , y_j , y_a, y_b είναι κόμβοι της ερώτησης που έχουν την ίδια σήμανση.

Στο Σχήμα 3.9 απεικονίζονται τα 4 πιθανά σχέδια στα οποία ένας κόμβος είναι πλεονάζων.



Σχήμα 3.9 : Ερωτήσεις Μονοπατιού Μερικής Δομής με περιττό κόμβο y_j

Είναι προφανές ότι ο εντοπισμός πλεοναζόντων κόμβων σε μία ερώτηση μπορεί να γίνει με αποδοτικό τρόπο.

Κανονική Μορφή Ερωτήσεων

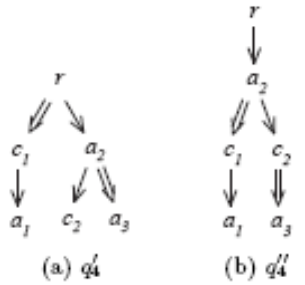
Τώρα, ας θεωρήσουμε μία ερώτηση Q και το σύνολο R των σχέσεων διαδοχής στο Q. Κατά την αποτίμηση του Q, η εξέταση των σχέσεων διαδοχής που μπορούν να εξαχθούν από τους κανόνες IR2 και IR3 είναι επίσης περιττή. Για παράδειγμα, αν η σχέση διαδοχής a_i/b_j είναι στο σύνολο των σχέσεων διαδοχής μίας ερώτησης Q, τότε το σύνολο των εικόνων της a_i/b_j κάτω από όλα τα πιθανά εμβαπτίσματα του Q σε ένα XML δέντρο είναι ένα υποσύνολο του συνόλου των εικόνων της $a_i//b_j$ (παραχθείσας από τον IR2). Η αγνόηση της $a_i//b_j$ μπορεί να επιταχύνει επομένως την αποτίμηση της Q χωρίς να επηρεάσει την απάντηση. Συνεπώς, εισάγουμε μία μορφή για τις ερωτήσεις μονοπατιού μερικής δομής που ονομάζεται κανονική μορφή.

Ορισμός

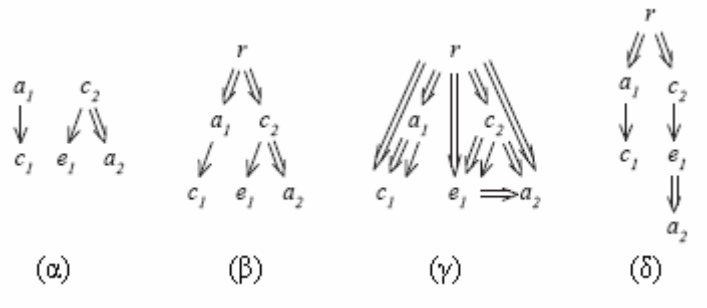
Μία ερώτηση μονοπατιού με πρότυπο μερικής δομής Q είναι σε **κανονική μορφή**, αν το σύνολο P των σχέσεων διαδοχής του Q ισούται με το κλείσιμο του P αφαιρουμένου του συνόλου των σχέσεων διαδοχής που μπορούν να εξαχθούν από το P με χρήση των συμπερασματικών κανόνων IR2 και IR3.

Αφού μία ικανοποιήσιμη ερώτηση δεν περιέχει κύκλους στην πλήρη της μορφή, θα πρέπει να έχει μοναδική κανονική μορφή. Αυτή η κανονική μορφή μπορεί να αναπαρασταθεί ως ένας κατευθυνόμενος ακυκλικός γράφος με ρίζα.

Τα ερωτήματα q'_1, q'_2, q'_3 του Σχήματος 3.6 είναι ήδη σε κανονική μορφή. Το Σχήμα 3.10 παρουσιάζει πάλι το ερώτημα του q'_4 Σχήματος 3.6 και μετά την κανονική του μορφή. Το Σχήμα 3.11 συγκεντρώνει τα βήματα που ακολουθήσαμε για την μετατροπή του ερωτήματος q_4 στην κανονική του μορφή.



Σχήμα 3.10 : Το ερώτημα q''_4 είναι η κανονική μορφή του q'_4.



Σχήμα 3.11 : Μετατροπή ερωτήματος μερικού μονοπατιού στην κανονική του μορφή (α) αρχική μορφή (β) μορφή ρίζας (γ) πλήρης μορφή (δ) κανονική μορφή

Ο υπολογισμός της κανονικής μορφής μίας ερώτησης μπορεί να γίνει αποδοτικά απομακρύνοντας σε οποιαδήποτε σειρά από την πλήρη του μορφή ακμές που μπορούν να εξαχθούν από άλλες ακμές χρησιμοποιώντας τους συμπερασματικούς κανόνες IR2 και IR3, μέχρις ότου δεν είναι δυνατό να απομακρυνθούν άλλες ακμές.

Έχοντας αναφέρει όλα αυτά μπορούμε στη συνέχεια να υποθέσουμε ότι οι ερωτήσεις είναι ικανοποιήσιμες, σε κανονική μορφή, και χωρίς πλεονάζοντες κόμβους. Ένα σπουδαίο χαρακτηριστικό αυτής της θεώρησης είναι ότι υπάρχει μία τοπολογική διάταξη των κόμβων της ερώτησης που ικανοποιεί τις δομικές της σχέσεις (πατέρα-παιδιού και προγόνου-απογόνου). Αυτό ακριβώς το χαρακτηριστικό είναι που θα εκμεταλλευτούμε στο επόμενο κεφάλαιο για τη σχεδίαση αλγορίθμων αποτίμησης ερωτήσεων μονοπατιού μερικής δομής.

4

Αποτίμηση Ερωτήσεων Μονοπατιού πάνω σε XML Δέντρα

Στο προηγούμενο κεφάλαιο ορίσαμε την έννοια των ερωτήσεων μονοπατιού πλήρους και μερικής δομής. Στο κεφάλαιο αυτό θα παρουσιάσουμε τους βασισμένους σε στοίβα αλγόριθμους για την αποτίμηση των ερωτήσεων μονοπατιού. Αρχικά θα δούμε τον αλγόριθμο **PathStack** που αποτιμά ερωτήματα μονοπατιού πλήρους δομής. Ο αλγόριθμος αυτός προτάθηκε από το Bruno και άλλους στην εργασία [1]. Στη συνέχεια θα δούμε τους αλγόριθμους **PartialMJ**, **PartialMJPlus** και **PartialPathStack** που αποτιμούν ερωτήματα μονοπατιού μερικής δομής. Οι αλγόριθμοι αυτοί προτάθηκαν από το Σουλδάτο και άλλους στην εργασία [2,3] και αποτελούν την πρώτη ολοκληρωμένη προσπάθεια να αντιμετωπιστεί αποδοτικά το πρόβλημα της αποτίμησης ερωτήσεων μονοπατιού μερικής δομής.

4.1 Αποτίμηση Ερωτήσεων Μονοπατιού Πλήρους Δομής

Εισαγωγικά

Έστω q η υπό εξέταση ερώτηση σε μορφή μονοπατιού πάνω στο XML δένδρο, καθώς και ο κόμβος ρίζα της εν λόγω ερώτησης.

Οι βασικές λειτουργίες που αφορούν τους κόμβους της ερώτησης είναι :

- `isRoot(Node)` : επιστρέφει true αν ο κόμβος Node δεν έχει εισερχόμενες ακμές
- `isLeaf(Node)` : επιστρέφει true αν ο κόμβος Node δεν έχει εξερχόμενες ακμές
- `parent(Node)` : επιστρέφει τον πατέρα του κόμβου Node
- `children(Node)` : επιστρέφει το παιδί του κόμβου Node.

Συσχετισμένο με κάθε κόμβο n μιας ερώτησης είναι μία ροή T_n. Η ροή περιέχει τις αναπαραστάσεις θέσης των κόμβων της βάσης δεδομένων (επί της οποίας γίνεται η ερώτηση) οι οποίοι ικανοποιούν το κατηγόρημα του κόμβου n (εναλλακτικά στη δική μας απλή περίπτωση που σημαίνονται από το ίδιο στοιχείο με τον κόμβο n). Η ροή αυτή στην περίπτωσή μας αποκτάται κάνοντας preorder διάσχιση του δέντρου και τοποθετώντας στην ίδια ροή κόμβους που σημαίνονται από το ίδιο στοιχείο. Σε πιο γενική περίπτωση θα μπορούσε να αποκτηθεί από αποδοτικούς μηχανισμούς προσπέλασης, όπως δομές ευρετηρίων. Οι κόμβοι στη ροή ταξινομούνται κατά αύξουσα σειρά της τιμής τους start (αν μεριμνούμε για την περίπτωση πολλών εγγράφων, τότε τοποθετούμε τους κόμβους κατά αύξουσα σειρά των τιμών του ζεύγους (treeID, start)).

Οι βασικές λειτουργίες μίας ροής είναι :

- `eof`
- `advance`
- `next`
- `nextS` : επιστρέφει την τιμή start της κωδικοποίησης θέσης του επόμενου στοιχείου στη ροή.
- `nextE` : επιστρέφει την τιμή end της κωδικοποίησης θέσης του επόμενου στοιχείου στη ροή.

Στο βασισμένο σε στοίβα αλγόριθμο που θα αναπτύξουμε στη συνέχεια, πρέπει επίσης να συσχετίσουμε με κάθε κόμβο του ερωτήματος μία στοίβα S_n. Κάθε κόμβος στη στοίβα αποτελείται από το ζεύγος: (αναπαράσταση θέσης από έναν κόμβο από την Tn, δείκτης σε έναν κόμβο στη στοίβα $S_{parent(n)}$).

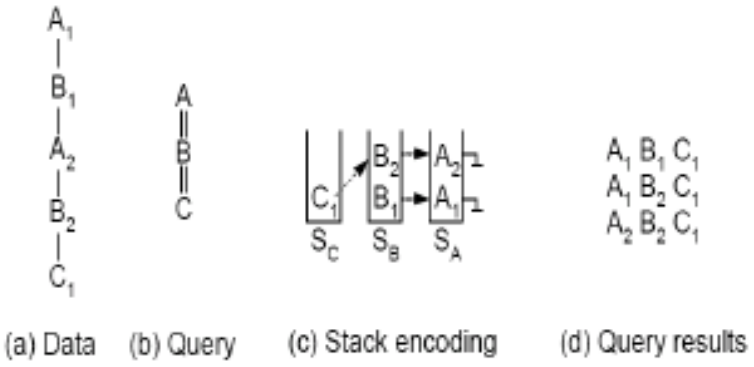
Οι βασικές λειτουργίες μίας στοίβας είναι :

- `empty(S)` : επιστρέφει true αν η στοίβα S είναι άδεια
- `push(S,value)` : βάζει την τιμή value στην κορυφή της στοίβας S
- `pop(S)` : βγάζει από τη στοίβα S το στοιχείο που βρίσκεται στην κορυφή της
- `topS(S)` : επιστρέφει την τιμή start της κωδικοποίησης θέσης του πρώτου στοιχείου στη στοίβα S.
- `topE(S)` : επιστρέφει την τιμή end της κωδικοποίησης θέσης του πρώτου στοιχείου στη στοίβα S.

Σε κάθε σημείο κατά τη διάρκεια του αλγορίθμου :

(α) όλοι οι κόμβοι στη στοίβα S_n (από την αρχή ως την κορυφή) βρίσκονται εγγυημένα πάνω σε ένα μονοπάτι από τη ρίζα σε ένα φύλλο στην XML βάση δεδομένων, και

(β) το σύνολο όλων των στοιβών περιέχει μία συμπαγή (άρα και αποδοτική) κωδικοποίηση μερικών και πλήρων απαντήσεων στην αρχική ερώτηση, η οποία μπορεί σε γραμμικό μέγεθος χώρου να αντιπροσωπεύει έναν δυνάμει εκθετικό αριθμό λύσεων (ως προς τον αριθμό των κόμβων της ερώτησης) στην ερώτηση, όπως φαίνεται στο παρακάτω σχήμα.



Σχήμα 4.1 : Συμπαγής Κωδικοποίηση των λύσεων χρησιμοποιώντας στοίβες

Το Σχήμα 4.1 απεικονίζει τη συμπαγή κωδικοποίηση απαντήσεων σε μία απλή ερώτηση μονοπατιού πάνω σε ένα πολύ απλό XML δέντρο. Η απάντηση $[A_2,B_2,C_1]$ προφανώς είναι κωδικοποιημένη αφού το C_1 δείχνει στο B_2, και το B_2 δείχνει στο A_2. Αφού το A_1 είναι κάτω από το A_2 στη στοίβα S_A, η $[A_1,B_2,C_1]$ είναι επίσης μια απάντηση. Τέλος, αφού το B_1 είναι κάτω από το B_2 στη στοίβα S_B, η $[A_2,B_1,C_1]$ είναι επίσης μία απάντηση. Σημειώνεται ότι η $[A_2,B_1,C_1]$ δεν είναι απάντηση, αφού το A_2 είναι πάνω από τον κόμβο A_1 στη στοίβα S_A, όπου δείχνει ο B_1.

4.1.1 Αλγόριθμος PathStack

Ο αλγόριθμος που χρησιμοποιούμε για την αποτίμηση ερωτήσεων μονοπατιού είναι ο PathStack. Η υλοποίηση σε ψευδοκώδικα του αλγόριθμου PathStack φαίνεται στο Σχήμα 4.2 ενώ στο Σχήμα 4.3 φαίνεται η υλοποίηση της συνάρτησης showSolutions που καλεί ο PathStack. Έχουμε κάνει όμως δύο υποθέσεις :

1. οι ροές περιέχουν κόμβους από ένα μόνο XML έγγραφο και
2. υπάρχουν μεταξύ κόμβων μόνο σχέσεις προγόνου-απογόνου.

Επέκταση αλγόριθμου :

1. Όταν έχουμε κόμβους από πολλά XML έγγραφα, αρκεί να ελέγξουμε και την ισότητα του αριθμού εγγράφου πριν προχωρήσουμε στην επεξεργασία των κόμβων στις ροές και τις στοίβες.
2. Όταν έχουμε σχέσεις πατέρα-παιδιού, πρέπει να συμπεριλάβουμε και το στοιχείο επίπεδο της κωδικοποίησης θέσης των κόμβων. Σε αυτήν την περίπτωση δεν χρειάζεται να αλλάξει ο PathStack παρά μόνο η συνάρτηση showSolutions στο σημείο της αναδρομικής κλήσης της. Εκεί όταν θα υπάρχει ακμή πατέρα-παιδιού, θα ελέγχει αν το επίπεδο των δύο κόμβων διαφέρει μόνο κατά ένα και αν ισχύει θα γίνεται μόνο μία φορά αναδρομική κλήση, αποφεύγοντας κιόλας έτσι περιττές αναδρομές.

Η κύρια ιδέα του αλγορίθμου είναι να κωδικοποιήσει επαναληπτικά μέσα στις στοίβες αρχικά μερικές και τελικά πλήρεις απαντήσεις στην ερώτηση μονοπατιού, περιπλανώμενος στους κόμβους των ροών κατά σειρά της αρχής του δείκτη κάθε κόμβου. Επομένως, οι κόμβοι της ερώτησης θα ταιριάξουν με τα δεδομένα από τη ρίζα της ως και το φύλλο της (εφόσον βέβαια τέτοια δεδομένα υπάρχουν). Αρχικά, ο αλγόριθμος βρίσκει τη ροή που περιέχει τον επόμενο υπό επεξεργασία κόμβο (Σειρά 4). Δεδομένου του υπό επεξεργασία κόμβου, μπορούμε να απομακρύνουμε από τις στοίβες τις μερικές απαντήσεις που δεν μπορούν να επεκταθούν σε πλήρεις απαντήσεις, επειδή ο τρέχων κόμβος δεν ανήκει στο ίδιο μονοπάτι με τους κόμβους των μερικών λύσεων (Σειρά 5-8). Στη συνέχεια, επαυξάνουμε τις μερικές λύσεις με το νέο κόμβο, αρκεί αυτός ο κόμβος να αντιστοιχεί στη ρίζα της ερώτησης ή η στοίβα του προηγούμενου στο ερώτημα κόμβο να μην είναι κενή (γιατί στην τελευταία περίπτωση δεν γίνεται να ταιριάξουν με δεδομένα από τις ροές οι κόμβοι της ερώτησης μονοπατιού που είναι πάνω από τον τρέχοντα κόμβο στην ερώτηση μονοπατιού) (Σειρά 9-10). Αν ο τρέχων κόμβος προστέθηκε στη στοίβα Sq_{min}, όπου q_{min} είναι το μοναδικό φύλλο της ερώτησης, τότε οι στοίβες περιέχουν κωδικοποίηση πλήρων απαντήσεων στην αρχική ερώτηση, οπότε καλείται η showSolutions για να προβάλει αυτές τις απαντήσεις (Σειρά 11-13).

```
Algorithm PathStack(q)
 1: rootNode = root(q)
 2: leafNode = leaf(q)
 3: while ¬empty(T_leafNode) do
 4:    q_min = getMinSource(q)
 5:    examinedNodes = {q_min}∪{parent(q_min)}
 6:    for q_i in examinedNodes(q) do
 7:       while ¬empty(S_qi) do
 8:          pop(S_qi)
 9:    if (q_min == rootNode || pointer to top(S_parent(q_min)) ≠ null) then
10:       moveStreamToStack(T_q_min, S_q_min, pointer to top(S_parent(q_min)))
11:       if isLeaf(q_min) then
12:          showSolutions(q_min, 1)
13:          pop(S_q_min)
14:       pop(T_q)

Function getMinSource(q)
  return  q_i∈subTreeNodes(q) such that nextS(Tq_i) is minimal

Procedure moveStreamToStack(T_q, S_q, p)
 1: push(S_q, (next(T_q), p))
```

Σχήμα 4.2 : Αλγόριθμος PathStack

```
Procedure showSolutions(S_q , index)
 1: // Ας υποθέσουμε χάριν απλότητας ότι οι στοίβες των κόμβων της ερώτησης από τη ρίζα έως
 2: // και το τρέχον φύλλο μπορούν να προσπελαστούν ως S_1,..,S_n.
 3: // Επίσης, έστω ένας global πίνακας outputIndex[1,...,n] δεικτών στα στοιχεία των στοιβών.
 4: // Ο outputIndex [i] παριστάνει τη θέση της στοίβας i που μας ενδιαφέρει για την τρέχουσα θέση,
 5: // όπου η θέση του πυθμένα κάθε στοίβας είναι 1.
 6:
 7: // Σημειώνουμε τη θέση index που μας ενδιαφέρει για τη στοίβα S_Q.
 8: outputIndex[S_q] = index
 9: if q == 1 then
10:    // Εμφανίζουμε την τρέχουσα απάντηση από τις στοίβες.
11:    output(S_1[outputIndex[1]].value,...,S_n[outputIndex[n]].value)
12: else
13:    new_index = S_q[index].pointer_to_parent
14:    for i = 1 to new_index do
15:       showSolutions(parent(q), i)
```

Σχήμα 4.3 : Διαδικασία showSolutions

Υπάρχουν δύο βασικοί τρόποι να προβάλουμε τις απαντήσεις, δεδομένου του κόμβου φύλλου. Ο πιο φυσικός είναι ως εγγραφές n στοιχείων, όπου n ο αριθμός των κόμβων της αρχικής ερώτησης, ταξινομημένες κατά το σχήμα φύλλο-προς-ρίζα. Αυτός ο τρόπος διασφαλίζει ότι τελικά οι (πλήρεις) απαντήσεις που επιστρέφει ο αλγόριθμος θα είναι επίσης ταξινομημένες από φύλλο προς ρίζα. Εμείς, ωστόσο, ακολουθήσαμε το δεύτερο τρόπο, που είναι η προβολή των εγγραφών ταξινομημένων σύμφωνα με το σχήμα ρίζα-προς-φύλλο. Ο

λόγος για τη χρησιμοποίηση αυτού του σχήματος είναι η συμβατότητά του με την έξοδο της διαδικασίας μπλοκαρίσματος των απαντήσεων, η οποία αναπτύσσεται και αναλύεται διεξοδικά στην επόμενη ενότητα. Ο αλγόριθμος showSolutions για την περίπτωση που η ερώτηση περιέχει μόνο ακμές προγόνου-απογόνου φαίνεται στο Σχήμα 4.3.

Αν επιθυμούμε οι τελικές απαντήσεις στην ερώτηση να παρουσιάζονται ταξινομημένες κατά ρίζα-προς-φύλλο σειρά, τότε είναι προφανές ότι δεν αρκεί σε κάθε επανάληψη της showSolutions να γράφονται στην έξοδο οι απαντήσεις που είναι κωδικοποιημένες στη στοίβα σε ρίζα-προς-φύλλο διάταξη. Απεναντίας, πρέπει με κάποιον τρόπο να μπορέσουμε να μπλοκάρουμε τις απαντήσεις, και να καθυστερήσουμε το γράψιμό τους μέχρις ότου σιγουρευτούμε ότι δεν μπορεί πια να υπολογιστεί καμία απάντηση που να είναι στη διάταξη των απαντήσεων πριν από αυτήν. Οι λεπτομέρειες του πώς πετυχαίνουμε αυτήν την ταξινόμηση παρουσιάζονται στην επόμενη ενότητα.

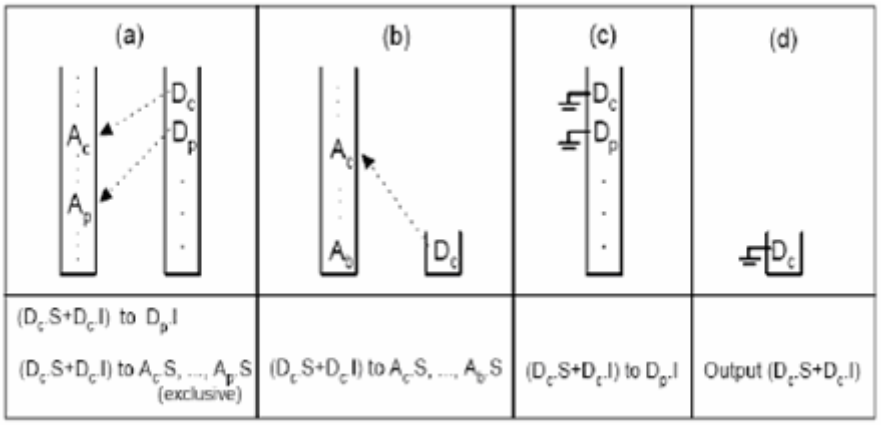
Μπλοκάρισμα Αποτελεσμάτων

Ας θεωρήσουμε το απλό ερώτημα A//D. Σε οποιοδήποτε σημείο κατά τη διάρκεια της επεξεργασίας της ερώτησης, αν ο κόμβος a από τη στοίβα S_A βρεθεί να είναι πρόγονος του κόμβου d από τη ροή T_D, τότε για κάθε κόμβο α' από τη στοίβα S_A που είναι πρόγονος του a πρέπει να είναι και πρόγονος του d. Αφού α'.start < α.start, πρέπει να αναβάλουμε την έξοδο της λύσης (a,d) μέχρις ότου η λύση (a',d) γραφεί. Παραμένει όμως η πιθανότητα ένα νέο στοιχείο d' στη ροή T_D που είναι απόγονος του d να μπορεί να δείξει στο a' όσο το a' ανήκει στη στοίβα S_A. Επομένως, δεν μπορούμε να γράψουμε στην έξοδο το ζεύγος (a,d), μέχρι ο κόμβος a' να βγει από τη στοίβα S_A. Στο μεταξύ, μπορούμε να κατασκευάσουμε ενδιάμεσα αποτελέσματα συνένωσης, τα οποία ωστόσο δεν μπορούν ακόμη να γραφούν στην έξοδο. Έτσι, η κατάσταση μπορεί να γίνει πολύ άσχημη για ερωτήσεις μεγάλου μήκους, με πολλά ενδιάμεσα αποτελέσματα. Η διαδικασία που θα περιγράψουμε επιστρέφει τις λύσεις στην ερώτησή μας σε διάταξη ρίζα-προς-φύλλο και βασίζεται στην ιδέα του μπλοκαρίσματος.

Για αυτόν το σκοπό, διατηρούμε δύο συνδεδεμένες λίστες σχετιζόμενες με κάθε στοιχείο n στις στοίβες: η πρώτη, **(S)elf-list**, παριστάνει όλες τις απαντήσεις στην ερώτηση που προκύπτει από το αρχικό θεωρώντας όμως ότι τώρα ρίζα είναι ο n, ενώ όλοι οι προγονοί του διαγράφονται από την ερώτηση, ώστε ο n να γίνει η ρίζα της ερώτησης - η δεύτερη, **(I)nherit-list**, παριστάνει όλες τις απαντήσεις στις προεκτάσεις της αρχικής ερώτησης με ρίζα

τους απογόνους του n, όπου οι υπόλοιποι κόμβοι (ο n και οι πρόγονοί του) έχουν διαγραφεί από το ερώτημα (μόνο η κορυφή κάθε στοίβας έχει inherit-list). Σε οποιοδήποτε σημείο κατά την εκτέλεση του αλγόριθμου, η self-list και η inherit-list του κόμβου n πρέπει να επεκταθούν με τα στοιχεία στις στοίβες των κόμβων που είναι πρόγονοί του στο ερώτημα, κατά παρόμοιο τρόπο με όσα κάναμε στη showSolutions, ώστε τελικά να πάρουμε τι πλήρεις απαντήσεις στο αρχικό ερώτημα. Η κύρια ιδέα του αλγορίθμου παραμένει παρόμοια με πριν, αλλά τώρα δε γράφουμε μία πλήρη απάντηση στη λίστα εξόδου αμέσως μόλις την εντοπίσουμε, αλλά συσσωρεύουμε στις δύο προαναφερθείσες λίστες τις μερικές λύσεις που βρίσκουμε κάθε φορά σε διάταξη ρίζα-προς-φύλλο, οπότε όταν στο τέλος γράψουμε τις πλήρεις λύσεις στη λίστα εξόδου, αυτές θα είναι στη σωστή σειρά.

Πιο συγκεκριμένα, όταν ένα νέο στοιχείο προστίθεται σε μία στοίβα, αρχικοποιούμε τις δύο λίστες του με τις κενές λίστες. Έστω πως σε κάποιο σημείο του αλγορίθμου βγάζουμε το στοιχείο D_C από τη στοίβα S_D. Ανάλογα με την τρέχουσα διάταξη των στοιβών, πράττουμε ως εξής:



Σχήμα 4.4 : Δυνατές διατάξεις των στοιβών κατά το μπλοκάρισμα αποτελεσμάτων

(α) Ο κόμβος D δεν είναι η ρίζα της ερώτησης, αλλά έχει πατέρα τον κόμβο A. Το στοιχείο D_C δεν είναι στον πυθμένα της στοίβας, αλλά έχει πρόγονο το στοιχείο D_P (Σχήμα 4.4α). Σε αυτήν την περίπτωση, πρώτα καθορίζουμε τους κόμβους A_C και A_P (τους κόμβους στη στοίβα S_A στους οποίους δείχνουν τα D_C και D_P αντίστοιχα). Τότε συνενώνουμε τις self-list και inherit-list (με αυτήν τη σειρά) του D_C με τις self-list των στοιχείων της στοίβας S_A αρχίζοντας από το A_C (συμπεριλαμβανομένου) και τελειώνοντας στο A_P (μη συμπεριλαμβανομένου). Επίσης, συνενώνουμε τις self-list και inherit-list του D_C στην inherit-list του D_P.

(β) Ο κόμβος D δεν είναι η ρίζα της ερώτησης, αλλά έχει πατέρα τον κόμβο A. Το στοιχείο D_C είναι στον πυθμένα της στοίβας (Σχήμα 4.4β). Σε αυτήν την περίπτωση, πρώτα καθορίζουμε τον κόμβο A_C (τον κόμβο στη στοίβα S_A στον οποίο δείχνει το D_C). Τότε συνενώνουμε τις self-list και inherit-list (με αυτήν τη σειρά) του D_C με τις self-list στα στοιχεία της στοίβας S_A από το A_C (συμπεριλαμβανομένου) μέχρι και τον πυθμένα της στοίβας.

(γ) Ο κόμβος D είναι η ρίζα της ερώτησης. Το στοιχείο D_C δεν είναι στον πυθμένα της στοίβας, αλλά έχει πρόγονο το στοιχείο D_P (Σχήμα 4.4γ). Σε αυτήν την περίπτωση, συνενώνουμε τις self-list και inherit-list (με αυτήν τη σειρά) του D_C με την inherit-list του στοιχείου D_P.

(δ) Ο κόμβος D είναι η ρίζα της ερώτησης. Το στοιχείο D_C είναι στον πυθμένα της στοίβας (Σχήμα 4.4δ). Πρώτα γράφουμε στην έξοδο τα στοιχεία της self-list και στη συνέχεια τα αποτελέσματα της inherit-list του στοιχείου D_C.

Είναι αρκετά εύκολο να δούμε ότι στις περιπτώσεις (α), (β), και (γ) διατηρούμε όλες τις λύσεις, όταν αναδιατάσσουμε τις συνδεδεμένες λίστες. Επίσης, κάθε φορά που συνενώνουμε μία λίστα με μία άλλη, είναι εγγυημένο ότι κανένα μελλοντικό στοιχείο δε θα συνενωθεί εκτός σειράς. Άρα, στην περίπτωση (δ) γράφουμε το αποτέλεσμα στην έξοδο στην επιθυμητή σειρά (ρίζα-προς-φύλλο). Ας δώσουμε μία πρόχειρη απόδειξη για την περίπτωση (α). Οι λίστες self-list και inherit-list του D_C πρέπει να συνενωθούν με όλα τα στοιχεία της στοίβας S_A από το στοιχείο A_C ως τον πυθμένα της. Για αυτό το λόγο, συνενώνουμε τις self-list και inherit-list του D_C σε όλους τους κόμβους της S_A από τον A_C μέχρι και πριν τον A_P. Αφού ταυτόχρονα συνενώνουμε τις λίστες self-list και inherit-list του D_C στην inherit-list του D_P, είναι βέβαιο ότι σε κάποια μελλοντική επανάληψη του αλγορίθμου αυτές οι μερικές λύσεις θα φτάσουν σε όλους τους εναπομείναντες κόμβους της στοίβας S_A. Επομένως δεν χάσαμε κάποια μερική λύση όσο χειριζόμασταν τις λίστες. Επίσης, όταν εξάγουμε το στοιχείο D_C, γνωρίζουμε ότι κανένα νέο στοιχείο από τη ροή T_D δε θα αρχίσει πριν το D_C (και όλους τους απογόνους του), επομένως κάθε νέα λύση θα ξεκινά μετά από κάθε μερική λύση στις δύο λίστες του D_C. Από την άλλη, δεν μπορούμε να συνενώσουμε αυτές τις λίστες στη self-list του A_P και των προγόνων του, επειδή κάποιες λύσεις που αρχίζουν πριν από αυτές μπορεί να μπλοκαριστούν στον D_P και τους προγόνους του. Αντιμετωπίζουμε αυτήν την περίπτωση συνενώνοντας τη self-list του D_C στην inherit-list του D_P. Οι υπόλοιπες περιπτώσεις αποδεικνύονται παρόμοια.

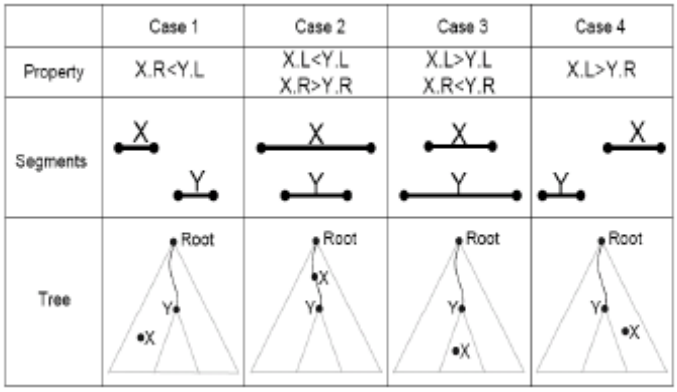
Η μόνη λειτουργία που πραγματοποιούμε στις λίστες είναι η συνένωση (εκτός από το τέλος, που τις διαβάζουμε για να γράψουμε τα αποτελέσματα στην έξοδο). Εφόσον η υλοποίηση των συνδεδεμένων λιστών διατηρεί λίστες στο κεφάλι και την ουρά των λιστών, αυτές οι λειτουργίες συνένωσης μπορούν να πραγματοποιηθούν σε σταθερό χρόνο και δε χρειάζονται επιπλέον αντιγραφές. Επομένως, χρειαζόμαστε μόνο να έχουμε πρόσβαση στην ουρά κάθε λίστας κατά τη διάρκεια του υπολογισμού. Το υπόλοιπο μέρος της λίστας μπορεί να κρατιέται στο swap part στο σκληρό δίσκο. Όταν η λίστα x συνενώνεται στο τέλος της λίστας y, δεν είναι απαραίτητο να είναι το κεφάλι της x στη μνήμη, αφού η λειτουργία συνένωσης απλά παράγει ένα δείκτη από την ουρά της y στο κεφάλι της x. Άρα το μόνο που χρειάζεται να ξέρουμε είναι το δείκτη για το κεφάλι κάθε λίστας, ακόμη κι αν μεταφερθεί στο swap part του σκληρού δίσκου. Κάθε σελίδα της μνήμης μεταφέρεται επομένως στο swap part του σκληρού δίσκου το πολύ μια φορά, και επανέρχεται στην κύρια μνήμη, μόνο όταν η λίστα είναι έτοιμη για γράψιμο στην έξοδο. Επίσης, ο συνολικός αριθμός των εισόδων στις λίστες είναι ακριβώς ίσος με τον αριθμό των εισόδων της εξόδου του αλγορίθμου. Έχουμε συνεπώς ότι οι απαιτούμενες I/O λειτουργίες για να διατηρήσουμε τις λίστες είναι ανάλογος του μεγέθους της εξόδου του αλγορίθμου, δεδομένου ότι υπάρχει αρκετή μνήμη για να κρατάμε στους καταχωρητές την ουρά κάθε λίστας.

Ανάλυση του PathStack

Η πρόταση που ακολουθεί είναι πολύ βασική για να αποδείξουμε την ορθότητα του αλγορίθμου PathStack.

Πρόταση

Έστω ότι ο κόμβος Y ενός XML δέντρου είναι σταθερός. Τότε η ακολουθία των περιπτώσεων σχετικής θέσης μεταξύ του κόμβου Y και όλων των κόμβων X του XML δέντρου κατά αύξουσα θέση της τιμής start είναι: (1\2)*3*4*. Οι περιπτώσεις 1 και 2 δεν είναι σαφώς οριοθετημένες, αλλά μπλέκεται η μία με την άλλη, ακολουθούν όλοι οι κόμβοι της περίπτωσης 3 πριν από οποιοδήποτε κόμβο της περίπτωσης 4 και, τέλος, έρχονται οι κόμβοι της περίπτωσης 4 (Σχήμα 4.5).



Σχήμα 4.5 : Περιπτώσεις για τον PathStack

Λήμμα

Έστω ότι για έναν τυχαίο κόμβο q στην υπό εξέταση ερώτηση έχουμε πως getMinSource(q)=q_N. Επίσης, έστω ότι tq_N είναι το επόμενο στοιχείο στη ροή Tq_N. Τότε, αφού το στοιχείο tq_N τοποθετηθεί στη στοίβα Sq_N, η αλυσίδα των στοιβών από την Sq_N ως την Sq_s ικανοποιεί ότι οι σημάνσεις των κόμβων τους ανήκουν στην αλυσίδα των κόμβων στο XML δέντρο από τον κόμβο q_N ως τη ρίζα.

Για κάθε κόμβο tq_{min} που τοποθετείται στη στοίβα Sq_{min}, είναι εύκολο να διαπιστώσουμε από το παραπάνω λήμμα και από την επαναληπτική φύση του αλγορίθμου ShowSolutions ότι όλες οι απαντήσεις, στις οποίες το tq_{min} είναι ταίριασμα για τον κόμβο q_{min} της ερώτησης, θα εμφανιστούν. Αυτό μας οδηγεί στο παρακάτω αποτέλεσμα ορθότητας:

Θεώρημα

Δοθέντων μιας ερώτησης μονοπατιού q και ενός XML δέντρου D, ο αλγόριθμος PathStack επιστρέφει σωστά όλες τις απαντήσεις για το q πάνω στο D.

Δείχνουμε, τέλος, τη **βελτιστότητα** του αλγορίθμου. Δοθείσης μιας XML ερώτησης μονοπατιού μήκους n, ο PathStack παίρνει n ροές εισόδου διατεταγμένες κατά τιμή του start της κωδικοποίησης θέσης, και υπολογίζει μία λίστα από εγγραφές μεγέθους n, οι οποίες ικανοποιούν την εξεταζόμενη ερώτηση. Προκύπτει άμεσα ότι ,εξαιρουμένων των κλήσεων στη ShowSolutions, το I/O και CPU κόστος του PathStack είναι γραμμικό ως προς το άθροισμα των μεγεθών των n ροών εισόδου. Αφού, τέλος, το κόστος της ShowSolutions είναι ανάλογο (γραμμικό) ως προς τη λίστα εξόδου, έχουμε το επόμενο θεώρημα βελτιστότητας:

Θεώρημα

Δοθέντων μιας ερώτησης μονοπατιού q με n κόμβους και ενός XML δέντρου D, ο αλγόριθμος PathStack έχει I/O και CPU χρονική πολυπλοκότητα γραμμική ως προς το άθροισμα των μεγεθών των n ροών εισόδου και της λίστας εξόδου. Επιπρόσθετα, η χωρική πολυπλοκότητα του PathStack είναι το ελάχιστο των: (α)το άθροισμα των μεγεθών των n ροών εισόδου, και (β)το μέγιστο μήκος ενός μονοπατιού από τη ρίζα σε ένα φύλλο στο D

Είναι ιδιαίτερης σημασίας να σημειώσουμε ότι η χρονική πολυπλοκότητα του αλγορίθμου PathStack είναι **ανεξάρτητη των μεγεθών όλων των ενδιάμεσων αποτελεσμάτων**.

4.2 Αποτίμηση Ερωτήσεων Μονοπατιού Μερικής Δομής

Εισαγωγικά

Έστω q μία ερώτηση μονοπατιού μερικής δομής σε κανονική μορφή και n ο αριθμός των κόμβων σε αυτή.

Οι λειτουργίες που αφορούν τους κόμβους της ερώτησης είναι :

- `nodes(q)` : επιστρέφει όλους τους κόμβους του ερωτήματος q
- `isRoot(n)` : επιστρέφει true αν ο κόμβος n δεν έχει εισερχόμενες ακμές στο ερώτημα q
- `isSink(n)` : επιστρέφει true αν ο κόμβος n δεν έχει εξερχόμενες ακμές στο ερώτημα q
- `parents(n)` : επιστρέφει όλους τους κόμβους στο ερώτημα q που έχουν εξερχόμενες ακμές στον κόμβο n

Υπάρχουν και οι ακόλουθες λειτουργίες που αφορούν μόνο τους αλγορίθμους PartialMJ και PartialMJPlus :

- `isLeaf(n)` : επιστρέφει true αν ο κόμβος n είναι φύλλο στο επικαλύπτον δέντρο q_s
- `parent(n)` : επιστρέφει τον γονιό του n στο επικαλύπτον δέντρο q_s.
- `depth(n)` : επιστρέφει το βάθος του n στο q_s . Είναι 1 αν n είναι η ρίζα του q.

Κάθε κόμβος n της ερώτησης q που σημαίνεται από το στοιχείο l σχετίζεται με μία **ροή εισόδου T_n** όλων των κόμβων (με αναπαράσταση κωδικοποίησης θέσης) που σημαίνονται από το στοιχείο l στο XML δέντρο. Για να έχουμε πρόσβαση στα στοιχεία της ροής T_n, διατηρούμε ένα φυσικό **κέρσορα C_n**, ο οποίος αρχικά δείχνει στο πρώτο στοιχείο της T_n. Χάριν απλότητας, ο C_n μπορεί εναλλακτικά να αναφέρεται στο ίδιο το στοιχείο που δεικτοδοτείται από τον C_n στη ροή T_n.

Οι βασικές λειτουργίες μίας ροής είναι :

- `eos(`C_n`)` : επιστρέφει true αν ο C_n έχει φτάσει στο τέλος της T_n, αλλιώς false
- `advance(`C_n`)` : μετακινεί τον κέρσορα στο επόμενο στοιχείο της T_n.
- C_n`.begin` : δηλώνει το πεδίο start της κωδικοποίησης θέσης του Cn.

Μία **στοίβα S_n** σχετίζεται με κάθε κόμβο n της ερώτησης q.

Στην περίπτωση των αλγορίθμων PartialMJ και PartialMJPlus, κάθε είσοδος στη S_n είναι ένα ζευγάρι από έναν κόμβο του ρεύματος T_n και από ένα δείκτη στην καταχώρηση του χαμηλότερου προγόνου του κόμβου στο XML δέντρο που υπάρχει στη στοίβα S_m, όπου m είναι ο γονιός του n στο επικαλύπτον δέντρο της ερώτησης.

Στην περίπτωση του αλγορίθμου PartialPathStack, κάθε είσοδος στη στοίβα S_n είναι ένα ζεύγος από έναν κόμβο από τη ροή T_n και από ένα σύνολο από δείκτες σε κατάλληλες θέσεις των στοιβών όλων των γονιών του n στην ερώτηση q.

Οι βασικές λειτουργίες μίας στοίβας :

- `empty(S)` : επιστρέφει true αν η στοίβα S είναι άδεια
- `push(S,value)` : βάζει την τιμή value στην κορυφή της στοίβας S
- `pop(S)` : βγάζει από τη στοίβα S το στοιχείο που βρίσκεται στην κορυφή της
- `bottom(S)` : επιστρέφει τη θέση του πυθμένα της στοίβας S.
- `top(S)` : επιστρέφει τη θέση της κορυφής της στοίβας S

Σε κάθε σημείο κατά τη διάρκεια της εκτέλεσης των αλγορίθμων έχουμε :

(α) κάθε στοιχείο σε μία θέση της στοίβας είναι απόγονος στο XML δένδρο όλων των στοιχείων που βρίσκονται στις θέσεις της στοίβας κάτω από αυτόν, και

(β) όλοι οι κόμβοι σε μία στοίβα βρίσκονται στο ίδιο μονοπάτι (από ρίζα προς φύλλο) στο XML δένδρο.

4.2.1 Αλγόριθμος PartialMJ

Δοθείσης μίας ερώτησης μονοπατιού μερικής δομής σε κανονική μορφή, ο αλγόριθμος PartialMJ αρχικά υπολογίζει ένα **επικαλύπτον δένδρο** του γράφου της ερωτήσεως. Στη συνέχεια, βρίσκει ταιριάσματα για όλα τα μονοπάτια από τη ρίζα προς τα φύλλα του επικαλύπτοντος δένδρου πάνω στο XML δένδρο χρησιμοποιώντας μία **επέκταση του αλγόριθμου PathStack**. Τα αποτελέσματα για κάθε μονοπάτι του επικαλύπτοντος δένδρου είναι σύνολα από εγγραφές από το XML δένδρο που είναι ταξινομημένες λεξικογραφικά σε διάταξη από τη ρίζα προς τα φύλλα. Αυτή η απαίτηση μπορεί να διασφαλιστεί με τη διαδικασία μπλοκαρίσματος που περιγράφηκε στον PathStack. Οι εγγραφές αυτές συνδυάζονται και **συγχωνεύονται στο τέλος της διάσχισης του δέντρου** έτσι ώστε :

(α) τα ταιριάσματα των κοινών κόμβων στα μονοπάτια να είναι ταυτόσημα (**ταυτοτική συνθήκη - ic**)

(β) όλοι οι κόμβοι στις υπό συγχώνευση μερικές λύσεις ικανοποιούν τις δομικές σχέσεις προγόνου-απογόνου και πατέρα-παιδιού που εμφανίζονται στο γράφο της ερώτησης αλλά όχι στο επικαλύπτον δένδρο που χρησιμοποιήσαμε (**συνθήκη σχέσης - sc**)

(γ) όλοι οι κόμβοι στις διάφορες μερικές λύσεις βρίσκονται στο ίδιο μονοπάτι (**συνθήκη μονοπατιού - pc**).

Προφανώς, αν τα φύλλα δύο αποτελεσμάτων βρίσκονται στο ίδιο μονοπάτι του XML δέντρου, τότε όλοι οι κόμβοι πριν από αυτά βρίσκονται επίσης στο ίδιο μονοπάτι. Έτσι, η συνθήκη μονοπατιού αρκεί να ελεγχθεί στα φύλλα του επικαλύπτοντος δέντρου.

```
q: partial path query
q_s: a spanning tree of q
E: the set of edges in q which do not appear in q_s

Algorithm PartialMJ()
01 while (¬end())
02     n = getNextQueryNode()
03     cleanStacks(C_n)
04     if (isRoot(n) or ∀ m ∈ parents(n): ¬empty(S_m))
05       moveToStack(n)
06       if (isLeaf(n))
07         showSolutionsWithBlocking(S_n, top(S_n))
08     advance(C_n)
09 joinPathSolutions()

Function end()
   return ∀ n ∈ nodes(q): isSink(n) ⇒ eos(C_n)

Function getNextQueryNode()
   return n ∈ nodes(q) such that C_n.begin is minimal

Procedure cleanStacks(C_n)
01 for m in nodes(q)
02     pop all entries in S_m whose nodes are not
       ancestors of C_n in the XML tree

Procedure moveToStack(n)
01 ptr = pointer to top of S_parent(n)
02 push(S_n, (C_n, ptr))

Procedure joinPathSolutions()
01 merge-join the results of the paths of q_s that satisfy
     the identity conditions on their common nodes (ic),
     the structural conditions that appear in E (sc),
     and the path condition on their lowest nodes (pc)
```

Σχήμα 4.6 : Αλγόριθμος PartialMJ

Ο αλγόριθμος PartialMJ παρουσιάζεται στο Σχήμα 4.6.

Ο αλγόριθμος PartialMJ διασχίζοντας κατά βάθος το XML δέντρο, κρατάει σε στοίβες εκείνους τους κόμβους που μπορούν να συνεισφέρουν στα αποτελέσματα του ερωτήματος. Αυτοί οι κόμβοι είναι αυτοί που αντιστοιχούν στους κόμβους του ερωτήματος των οποίων οι γονείς έχουν ήδη βρει ταίριασμα στο ίδιο μονοπάτι του XML δέντρου. Όποτε ένας κόμβος δέντρου **τοποθετείται στη στοίβα ενός φύλλου** του επικαλύπτοντος δέντρου, παράγονται αποτελέσματα μονοπατιού **από το αντίστοιχο μονοπάτι** του επικαλύπτοντος δέντρου. Μόλις η διάσχιση του δέντρου τελειώσει, τα αποτελέσματα όλων των μονοπατιών του επικαλύπτοντος δέντρου συγχωνεύονται σύμφωνα με τις συνθήκες ταυτότητας (ic), σχέσης (sc) και μονοπατιού (pc).

Στις γραμμές 01-08, ο αλγόριθμος σκανάρει τις ροές εισόδου και βρίσκει ταιριάσματα για όλα τα μονοπάτια από τη ρίζα προς τα φύλλα που ανήκουν στο επικαλύπτον δένδρο της ερώτησης. Η γραμμή 02 καθορίζει ποιος είναι ο επόμενος κόμβος της ερώτησης προς επεξεργασία σύμφωνα με την κατά βάθος διάσχιση του δέντρου. Η γραμμή 03 εξάγει από την κορυφή των στοιβών όλους τους κόμβους που δεν βρίσκονται στο ίδιο μονοπάτι με τον υπό επεξεργασία κόμβο C_n πάνω στο XML δένδρο. Ο κόμβος C_n της ροής εισόδου T_n εισάγεται στην κορυφή της στοίβας S_n μόνο αν οι στοίβες των γονιών του κόμβου n δεν είναι άδειες (γραμμές 04-05). Με αυτόν τον τρόπο, αποφεύγουμε να αποθηκεύουμε στη στοίβα και να επεξεργαζόμαστε κόμβους που εγγυημένα δεν συνεισφέρουν στο τελικό αποτέλεσμα. Όταν τοποθετούμε τον κόμβο C_n στην κορυφή της στοίβας S_n, προσθέτουμε επίσης και ένα δείκτη προς το κορυφαίο στοιχείο της στοίβας S_m, όπου m είναι ο πατέρας του κόμβου n στο επικαλύπτον δένδρο της ερώτησης. Η γραμμή 06 ελέγχει αν ο κόμβος n είναι φύλλο στο επικαλύπτον δένδρο (*προφανώς ένας κόμβος που είναι φύλλο στην αρχική ερώτηση θα είναι φύλλο και στο επικαλύπτον δένδρο, το αντίστροφο όμως δεν ισχύει*). Αν ο έλεγχος είναι θετικός, τότε η γραμμή 07 καλεί τη διαδικασία showSolutionsWithBlocking, η οποία παράγει τα ενδιάμεσα (μερικά) αποτελέσματα για το αντίστοιχο μονοπάτι του επικαλύπτοντος δένδρου που καταλήγει στον κόμβο-φύλλο n. Αυτά τα αποτελέσματα ταξινομούνται σε διάταξη από τη ρίζα προς τα φύλλα, έτσι ώστε να μπορούν μέσα σε γραμμικό χρόνο ως προς το μέγεθος τους να συνδυαστούν και να συγχωνευτούν στα τελικά αποτελέσματα. Χρησιμοποιείται επομένως κάποια τεχνική μπλοκαρίσματος των αποτελεσμάτων σαν αυτή που περιγράψαμε στον αλγόριθμο PathStack. Ο συνδυασμός και η συγχώνευση των τελικών αποτελεσμάτων γίνονται στη γραμμή 09 με τη διαδικασία joinPathSolutions όταν έχει τελειώσει η διάσχιση του δέντρου. Η συνένωση ελέγχει :

(α) αν τα μερικά αποτελέσματα βρίσκονται στο ίδιο μονοπάτι του XML δένδρου (συνθήκη μονοπατιού - pc),

(β) αν τα ταιριάσματα των κοινών κόμβων στα μονοπάτια είναι ταυτόσημα (ταυτοτική συνθήκη - ic), και

(γ) αν οι δομικές σχέσεις της ερώτησης που δεν εμφανίζονται στο επικαλύπτον δένδρο ικανοποιούνται (συνθήκη σχέσης - sc).

Όλες αυτές οι συνθήκες μπορούν να ελεγχθούν άμεσα χρησιμοποιώντας την αναπαράσταση θέσης των κόμβων στο XML δένδρο.

Επέκταση αλγόριθμου

Ο αλγόριθμος που μόλις παρουσιάστηκε είναι σχεδιασμένος για την αποτίμηση ερωτήσεων που περιέχουν μόνο σχέσεις προγόνου-απογόνου. Όταν υπάρχουν και σχέσεις πατέρα-παιδιού πρέπει να γίνουν δύο αλλαγές :

1. όποτε ένας κόμβος C_b από μία ροή T_b είναι υπό επεξεργασία, και η a/b είναι μία σχέση πατέρα-παιδιού στην ερώτηση, ο κόμβος C_b τοποθετείται στη στοίβα S_b μόνο αν ο γονικός του κόμβος στο XML δένδρο εμφανίζεται στην κορυφή της στοίβας S_a.
2. κατά τον υπολογισμό των αποτελεσμάτων ενός μονοπατιού από τη ρίζα σε κάποιο φύλλο του επικαλύπτοντος δένδρου της ερώτησης, ένας κόμβος στη στοίβα S_b εμφανίζεται μόνο σε εκείνα τα αποτελέσματα του μονοπατιού που περιέχουν επίσης από τη στοίβα S_a το γονέα του στο XML δένδρο.

Επικαλύπτον Δέντρο

Περισσότερα του ενός επικαλύπτοντα δέντρα μπορούν να εξαχθούν από το γράφο ενός ερωτήματος. Αυτά τα επικαλύπτοντα δέντρα συνήθως περιλαμβάνουν διαφορετικό αριθμό μονοπατιών. Ο PartialMJ θα έχει καλύτερη απόδοση αν χρησιμοποιήσει επικαλύπτον δέντρο με μικρό αριθμό μονοπατιών. Ένα τέτοιο επικαλύπτον δέντρο θα χρειαστεί λιγότερες συγχωνεύσεις. Επίσης έχει μεγαλύτερα μονοπάτια, οπότε θα παράγονται λιγότερα ενδιάμεσα αποτελέσματα.

Συγχωνεύσεις

Για να αποφύγουμε περιττές ταξινομήσεις, μονοπάτια που μοιράζονται μεγαλύτερο αριθμό κοινών κόμβων προηγούνται στις συνενώσεις.

Ανάλυση του PartialMJ

Ο αλγόριθμος PartialMJ γεμίζει τις στοίβες σε ένα μοναδικό πέρασμα των ροών εισόδου. Επιπλέον, χρησιμοποιεί τη διαδικασία showSolutionsWithBlocking για να παράγει αποτελέσματα για κάθε μονοπάτι από τη ρίζα προς ένα φύλλο του επικαλύπτοντος δένδρου της ερώτησης. Αυτή η διαδικασία έχει αποδειχτεί ότι είναι βέλτιστη για την αποτίμηση ερωτήσεων μονοπατιού. Ωστόσο, μπορεί να υπάρχουν συνδυασμοί από αποτελέσματα στα διάφορα μονοπάτια του επικαλύπτοντος δένδρου που δεν μπορούν να συνδυαστούν για να δώσουν κάποιο αποτέλεσμα στη αρχική ερώτηση.

Εξαιτίας των ενδιάμεσων αποτελεσμάτων, ο αλγόριθμος αυτός **δεν είναι ασυμπτωτικά βέλτιστος**. Προφανώς, στην περίπτωση που η ερώτηση εκφυλίζεται σε μία ερώτηση με πρότυπο μορφής μονοπατιού, ο αλγόριθμος PartialMJ είναι ασυμπτωτικά βέλτιστος.

4.2.2 Αλγόριθμος PartialMJPlus

Ο αλγόριθμος PartialMJPlus αποτελεί **επέκταση του PartialMJ**. Εδώ, τα αποτελέσματα για τα μονοπάτια του επικαλύπτοντος δέντρου παράγονται και συνενώνονται χωριστά σε κάθε μονοπάτι του XML δέντρου. Έτσι, δεν απαιτείται έλεγχος αν ανήκουν στο ίδιο μονοπάτι του δέντρου κατά τη συνένωση. Ένας (πιθανότατα μεγάλος) αριθμός ενδιάμεσων αποτελεσμάτων αποφεύγεται.

Δοθείσης μίας ερώτησης μονοπατιού μερικής δομής σε κανονική μορφή, ο αλγόριθμος PartialMJPlus αρχικά υπολογίζει ένα **επικαλύπτον δένδρο** του γράφου της ερωτήσεως. Στη συνέχεια, βρίσκει ταιριάσματα για όλα τα μονοπάτια από τη ρίζα προς τα φύλλα του επικαλύπτοντος δένδρου πάνω στο XML δένδρο χρησιμοποιώντας μία **επέκταση του αλγόριθμου PathStack**. Όμως τώρα τα αποτελέσματα μονοπατιού του επικαλύπτοντος δέντρου δεν συγχωνεύονται στο τέλος της διάσχισης του δέντρου. **Συγχωνεύονται ξεχωριστά για το κάθε μονοπάτι** του XML δέντρου. Για το λόγο αυτό δεν χρειάζεται να ελέγξουμε αν τα αποτελέσματα βρίσκονται στο ίδιο μονοπάτι του XML δέντρου (συνθήκη μονοπατιού - pc). Μόνο η **συνθήκη ταυτότητας** (**ic**) και η **συνθήκη σχέσης** (**sc**) χρειάζονται να ελεγχθούν κατά τη συγχώνευση. Ένας (πιθανώς μεγάλος) αριθμός ενδιάμεσων αποτελεσμάτων αποφεύγονται με αυτόν τον τρόπο.

```
q: partial path query
q_s: a spanning tree of q
E: the set of edges in q which do not appear in q_s

Algorithm PartialMJPlus()
01 while (¬end())
02     n = getNextQueryNode()
03     cleanStacks(C_n)
04     if (isRoot(n) or ∀ m ∈ parents(n): ¬empty(S_m))
05       moveToStack(n)
06       if (isSink(n) and ∀ m ∈ nodes(q): ¬empty(S_m))
07         showPathSolutionsPlus(n)
08         joinPathSolutionsPlus()
09     advance(C_n)

Procedure showPathSolutionsPlus(n)
01 for m ∈ nodes(q)
02     if(m == n)
03       showSolutions(m, top(S_m))
04     else if (isLeaf(m))
05       for i = bottom(S_m) to top(S_m)
06         showSolutions(m, i))
07     sort the path results in root-to-leaf order

Procedure joinPathSolutionsPlus()
01 merge-join the results of the paths of q_s that satisfy
     the identity conditions on their common nodes (ic),
     and the structural conditions that appear in E (sc).
```

Σχήμα 4.7 : Αλγόριθμος PartialMJPlus

Ο αλγόριθμος PartialMJPlus παρουσιάζεται στο Σχήμα 4.7.

Ο αλγόριθμος PartialMJPlus διασχίζοντας κατά βάθος το XML δέντρο, κρατάει σε στοίβες εκείνους τους κόμβους που μπορούν να συνεισφέρουν στα αποτελέσματα του ερωτήματος. Αυτοί οι κόμβοι είναι αυτοί που αντιστοιχούν στους κόμβους του ερωτήματος των οποίων οι γονείς έχουν ήδη βρει ταίριασμα στο ίδιο μονοπάτι του XML δέντρου. Όποτε ένας κόμβος δέντρου **τοποθετείται στη στοίβα ενός φύλλου** του επικαλύπτοντος δέντρου **και όλα τα φύλλα δεν έχουν άδειες στοίβες**, παράγονται αποτελέσματα μονοπατιού **από όλα τα μονοπάτια** του επικαλύπτοντος δέντρου. Αυτά τα αποτελέσματα αμέσως συγχωνεύονται σύμφωνα με τις συνθήκες ταυτότητας (ic) και σχέσης (sc).

Στις γραμμές 01-07, ο αλγόριθμος διασχίζει τα ρεύματα κόμβων και βρίσκει αποτελέσματα για τα μονοπάτια του επικαλύπτοντος δέντρου της ερώτησης. Τα αποτελέσματα συνενώνονται κατευθείαν, πριν προχωρήσει στο επόμενο μονοπάτι του XML δέντρου, (γραμμή 08) και όχι στο τέλος όπως γινόταν στον PartialMJ. Πιο λεπτομερώς, η γραμμή 02 καθορίζει τον επόμενο κόμβο n προς επεξεργασία σύμφωνα με την κατά βάθος διάσχιση του

δέντρου. Η γραμμή 03 βγάζει από τις στοίβες όλους τους κόμβους που δεν βρίσκονται στο ίδιο μονοπάτι του δέντρου με τον κόμβο C_n. Ο κόμβος C_n από το ρεύμα T_n εισάγεται στη στοίβα S_n μόνο αν οι στοίβες των γονιών του κόμβου n στην ερώτηση δεν είναι άδειες (γραμμές 04-05). Με αυτόν τον τρόπο, αποφεύγουμε να αποθηκεύουμε στη στοίβα και να επεξεργαζόμαστε κόμβους που εγγυημένα δεν συνεισφέρουν σε αποτελέσματα της ερώτησης. Μόλις μπει ένας κόμβος C_n στη στοίβα S_n, αποθηκεύεται μαζί του ένας δείκτης προς το στοιχείο στην κορυφή της στοίβας S_m, όπου m είναι ο γονιός του n στο επικαλύπτον δέντρο της ερώτησης. Η γραμμή 06 ελέγχει αν μπορούν να εξαχθούν αποτελέσματα. Εξαγωγή αποτελεσμάτων γίνεται όταν ο τρέχων κόμβος είναι φύλλο του επικαλύπτοντος δέντρου και κανένας κόμβος του ερωτήματος δεν έχει άδεια στοίβα. Αν ο έλεγχος είναι θετικός, τότε η γραμμή 07 παράγει αποτελέσματα για τα μονοπάτια του επικαλύπτοντος δέντρου. Η γραμμή 08 συνενώνει τα αποτελέσματα αυτά. Τέλος, στη γραμμή 09 ο αλγόριθμος προχωρά.

Η συνάρτηση showPathSolutionsPlus παράγει αποτελέσματα για τα μονοπάτια του επικαλύπτοντος δέντρου με κλήση της showSolutions. Αυτά τα αποτελέσματα είναι ταξινομημένα σε διάταξη φύλλο-προς-ρίζα. Τεχνικές μπλοκαρίσματος, που είδαμε σε προηγούμενη ενότητα, δεν μπορούν να χρησιμοποιηθούν για ταξινόμηση ρίζα-προς-φύλλο, καθώς αυτό θα δέσμευε κάποια αποτελέσματα ως το τέλος του αλγορίθμου και δεν θα ήταν δυνατή η ταχεία συνένωση των επιμέρους αποτελεσμάτων. Τα αποτελέσματα που παράγονται για τα μονοπάτια του επικαλύπτοντος δέντρου διατηρούνται, ώστε να μην παραχθούν ξανά σε επόμενη κλήση της συνάρτησης πάνω στο ίδιο μονοπάτι του δέντρου. Αυτά τα αποτελέσματα μπορούν να συνενωθούν με νέα αποτελέσματα των υπολοίπων μονοπατιών του επικαλύπτοντος δέντρου.

Η συνάρτηση joinPathSolutionsPlus συνενώνει τα αποτελέσματα μονοπατιών του επικαλύπτοντος δέντρου. Στο μονοπάτι του εισερχόμενου κόμβου C_n, μόνο τα αποτελέσματα που περιέχουν τον C_n συνενώνονται, ώστε να παραχθούν νέα αποτελέσματα. Η συνένωση ελέγχει :

(α) αν το ταίριασμα των κοινών τους κόμβων είναι ταυτόσημο (συνθήκη ταυτότητας - ic)

(β) αν οι δομικές σχέσεις της ερώτησης που δεν εμφανίζονται στο επικαλύπτον δέντρο ικανοποιούνται (συνθήκη σχέσης - sc).

Όλες αυτές οι συνθήκες μπορούν να ελεγχθούν εύκολα με χρήση της αναπαράστασης θέσης των κόμβων του XML δέντρου. Η συνθήκη μονοπατιού (pc) που ελεγχόταν στον PartialMJ, εδώ είναι περιττή.

Επέκταση αλγορίθμου

Ο αλγόριθμος που παρουσιάστηκε είναι σχεδιασμένος για την αποτίμηση ερωτήσεων που περιέχουν μόνο σχέσεις προγόνου-απογόνου. Όταν υπάρχουν και σχέσεις γονέα-παιδιού μπορούν να αντιμετωπιστούν όπως στον PartialMJ.

Ανάλυση του PartialMJPlus

Η συνέπεια και η πληρότητα μπορούν να δειχθούν όπως στον PartialMJ.

Θέματα όπως εξαγωγή του επικαλύπτοντος δέντρου και προτεραιότητα συγχωνεύσεων μπορούν αντιμετωπιστούν όπως στην περίπτωση του PartialMJ που αναφέραμε στην προηγούμενη ενότητα.

Ο PartialMJPlus παράγει επίσης ενδιάμεσα αποτελέσματα όπως και ο PartialMJ, μόνο που παράγει μικρότερο αριθμό αφού δεν χρειάζεται να ελέγχει αν βρίσκονται στο ίδιο μονοπάτι (συνθήκη μονοπατιού - pc).

Είδαμε πιο πάνω ότι δεν είναι δυνατή η τεχνική μπλοκαρίσματος για την ταξινόμηση των αποτελεσμάτων κατά ρίζα-προς-φύλλο. Παρόλα αυτά απαιτείται λιγότερος χρόνος για την ταξινόμηση ρίζα-προς-φύλλο των αποτελεσμάτων παρά για την συγχώνευση των αποτελεσμάτων όλων των μονοπατιών στο τέλος της διάσχισης του δέντρου.

Για τους παραπάνω λόγους ο PartialMJPlus είναι πιο γρήγορος από τον PartialMJ , αλλά στη χειρότερη περίπτωση έχει την ίδια πολυπλοκότητα.

4.2.3 Αλγόριθμος PartialPathStack

Για να αποφύγουμε το πρόβλημα των ενδιάμεσων αποτελεσμάτων των PartialMJ και PartialMJPlus, ο Σουλδάτος και άλλοι στην εργασία [2,3] προτείνουν έναν καινούριο **ολιστικό** αλγόριθμο βασισμένο σε στοίβες για την αποτίμηση ερωτήσεων μονοπατιού μερικής δομής. Σε αντίθεση με τον PartialMJ και τον PartialMJPlus, ο PartialPathStack δεν αποσυνθέτει την αρχική ερώτηση σε απλά μονοπάτια, αλλά προσπαθεί να **ταιριάξει το γράφο της ερώτησης στα XML δεδομένα συνολικά**. Το κύριο χαρακτηριστικό του αλγορίθμου αυτού είναι ότι χρησιμοποιεί μία **τοπολογική διάταξη** των κόμβων της ερώτησης. Πρόκειται για μία γραμμική διάταξη των κόμβων που σέβεται τη μερική διάταξη, όπως αυτή προκύπτει από τις δομικές σχέσεις στο γράφο της ερώτησης.

```
q: partial path query with N nodes

Algorithm PartialPathStack()
01 extract a topological order 1..N of the query nodes
02 while (¬end())
03     n = getNextQueryNode()
04     cleanStacks(C_n)
05     if (isRoot(n) or ∀ m ∈ parents(n): ¬empty(S_m))
06       moveToStack(n)
07       if (isSink(n) and ∀ m ∈ nodes(q): ¬empty(S_m))
08         showSolutionsPPS(n, N, top(S_N))
09     advance(C_n)

Procedure moveToStack(n)
01 ptrs = pointers to top of all stacks of parents(n) in q
02 push(S_n, (C_n, ptrs))

Procedure showSolutionsPPS(n, m, stackPos)
01 solution[m] = stackPos
02 if (m = 1)    //root
03     output (S1[solution[1]],...,S_N[solution[N]])
04 else
05     for c in children(m − 1)
06       ptrFrom[c] = S_c[solution[c]].ptrs[m − 1]
07     maxPos = minarg_{c∈children(m−1)} {ptrFrom[c]}
08       showSolutionsPPS(n, m − 1, maxPos)
09 if (m≠n and stackPos≠bottom(S_m))
10     showSolutionsPPS(n, m, stackPos − 1)
```

Σχήμα 4.8 : Αλγόριθμος PartialPathStack

Ο αλγόριθμος PartialPathStack απεικονίζεται στο Σχήμα 4.8 .

Ο αλγόριθμος PartialPathStack διασχίζοντας κατά βάθος το XML δέντρο, κρατάει σε στοίβες εκείνους τους κόμβους που μπορούν να συνεισφέρουν στα αποτελέσματα του ερωτήματος. Αυτοί οι κόμβοι είναι αυτοί που αντιστοιχούν στους κόμβους του ερωτήματος των οποίων οι γονείς έχουν ήδη βρει ταίριασμα στο ίδιο μονοπάτι του XML δέντρου. Όποτε ένας κόμβος δέντρου **τοποθετείται στη στοίβα ενός φύλλου** του δέντρου **και όλα τα φύλλα δεν έχουν άδειες στοίβες**, παράγονται αποτελέσματα του ερωτήματος που βασίζονται σε μία **τοπολογική διάταξη** των κόμβων του ερωτήματος.

Η γραμμή 01 εξάγει μία τοπολογική διάταξη των κόμβων της ερώτησης, διασχίζει το XML δέντρο και βρίσκει ταιριάσματα για τους κόμβους του ερωτήματος βασιζόμενος στην τοπολογική διάταξη (γραμμές 02-09). Πιο λεπτομερώς, η γραμμή 03 καθορίζει τον επόμενο κόμβο C_n προς επεξεργασία σύμφωνα με την κατά βάθος διάσχιση του δέντρου. Η γραμμή 04 βγάζει από τις στοίβες όλους τους κόμβους που δεν βρίσκονται στο ίδιο μονοπάτι του δέντρου με τον κόμβο C_n. Ο κόμβος C_n από το ρεύμα T_n εισάγεται στη στοίβα S_n μόνο αν οι

στοίβες των γονιών του κόμβου n στην ερώτηση δεν είναι άδειες (γραμμές 05-06). Με αυτόν τον τρόπο, αποφεύγουμε να αποθηκεύουμε στη στοίβα και να επεξεργαζόμαστε κόμβους που εγγυημένα δεν συνεισφέρουν σε αποτελέσματα της ερώτησης. Μόλις μπει ένας κόμβος C_n στη στοίβα S_n, αποθηκεύεται μαζί του ένα σύνολο από δείκτες προς τα στοιχεία στις κορυφές των στοιβών των γονιών του. Η γραμμή 07 ελέγχει αν μπορούν να εξαχθούν αποτελέσματα. Εξαγωγή αποτελεσμάτων γίνεται όταν ο τρέχων κόμβος είναι φύλλο και κανένας κόμβος του ερωτήματος δεν έχει άδεια στοίβα. Αν ο έλεγχος είναι θετικός, τότε η γραμμή 08 παράγει αποτελέσματα με τη διαδικασία showSolutionsPPS.

Η συνάρτηση showSolutionsPPS παράγει τα αποτελέσματα σε μία ανεστραμμένη τοπολογική διάταξη, έτσι ώστε η στοίβα ενός κόμβου τίθεται υπό επεξεργασία αφού οι κόμβοι παιδιά του στη ερώτηση έχουν τεθεί υπό επεξεργασία. Συνδυάζει κόμβους από όλες τις στοίβες για να σχηματίσει τα αποτελέσματα. Για να αποφύγουμε την περιττή παραγωγή αποτελεσμάτων, ο αλγόριθμος εξάγει σε αυτό το σημείο μόνο αποτελέσματα που περιέχουν τον κόμβο C_n. Αντιθέτως, όλοι οι κόμβοι από τις άλλες στοίβες των φύλλων μπορούν να χρησιμοποιηθούν για να παράγουν αποτελέσματα (γραμμές 09-10). Όλοι οι κόμβοι από τις στοίβες που δεν αντιστοιχούν σε φύλλα της ερώτησης μπορούν να χρησιμοποιηθούν αν είναι πρόγονοι στο XML δέντρο των επιλεγμένων κόμβων στις στοίβες όλων των παιδιών τους στο ερώτημα (γραμμές 05-07).

Επέκταση αλγόριθμου

Ο αλγόριθμος που παρουσιάστηκε προηγουμένως είναι σχεδιασμένος για την αποτίμηση ερωτήσεων που περιέχουν μόνο σχέσεις προγόνου-απογόνου. Όταν υπάρχουν και σχέσεις πατέρα-παιδιού πρέπει να τροποποιήσουμε τη διαδικασία αποθήκευσης στις στοίβες και τη συγχώνευση των αποτελεσμάτων, όπως εξηγήσαμε παραπάνω στον αλγόριθμο PartialMJ.

Ανάλυση του PartialPathStack

Μπορούμε να παρατηρήσουμε ότι :

(α) κάθε κόμβος C_n μίας ροής εισόδου T_n μπορεί να τοποθετηθεί στη στοίβα S_n μόνο αφού πρώτα οι πρόγονοί του στο XML δένδρο έχουν θεωρηθεί (διαδικασία getNextQueryNode), και

(β) ένας κόμβος δεν εξάγεται από τη στοίβα εκτός κι αν ο τρέχων κόμβος C_n εμφανίζεται σε διαφορετικό μονοπάτι στο XML δένδρο (διαδικασία cleanStacks).

Βασισμένοι σε αυτές τις παρατηρήσεις, μπορούμε να δείξουμε το επόμενο λήμμα:

Λήμμα

Οι κόμβοι σε ένα αποτέλεσμα μίας ερώτησης εμφανίζονται στις στοίβες του αλγόριθμου PartialPathStack ταυτόχρονα σε κάποιο σημείο κατά την εκτέλεσή του.

Χρησιμοποιούμε το παραπάνω λήμμα για να δείξουμε την ορθότητα και πληρότητα του αλγόριθμου:

Θεώρημα

Δοθέντων μιας ερώτησης με πρότυπο μονοπάτι μερικής δομής q και ενός XML δέντρου T, ο αλγόριθμος PartialPathStack ορθά επιστρέφει όλες τις απαντήσεις για το q πάνω στο T.

Δοθέντων μιας ερώτησης με πρότυπο μονοπάτι μερικής δομής q και ενός XML δέντρου T, έστω είσοδος το άθροισμα των μεγεθών των ροών εισόδου, έξοδος το άθροισμα των αποτελεσμάτων της ερώτησης q πάνω στο T, βαθμός εισόδου ο μέγιστος αριθμός εισερχόμενων ακμών σε κάποιον κόμβο της ερώτησης q, βαθμός εξόδου ο μέγιστος αριθμός εξερχόμενων ακμών από κάποιον κόμβο της ερώτησης, και μέγιστο μονοπάτι το μέγιστο μήκος ενός μονοπατιού στο T από τη ρίζα προς κάποιο φύλλο του δέντρου.

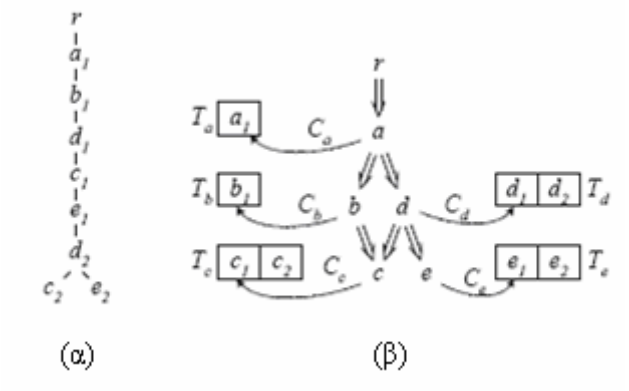
Θεώρημα

Ο αλγόριθμος PartialPathStack έχει χειρότερη I/O και CPU χρονική πολυπλοκότητα ίση με O(βαθμός εισόδου * είσοδος, βαθμός εξόδου * έξοδος). Η χειρότερη χωρική του πολυπλοκότητα είναι ίση με O(βαθμός εισόδου * min(είσοδος, μέγιστο μονοπάτι)).

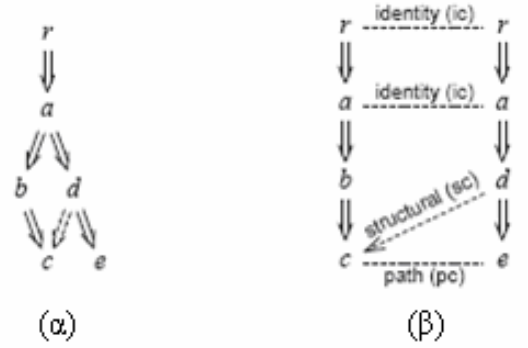
Το προηγούμενο θεώρημα υπονοεί ότι ο PartialPathStack είναι **ασυμπτωτικά βέλτιστος αν ο βαθμός εισόδου και ο βαθμός εξόδου είναι φραγμένοι από κάποια σταθερά**. Προφανώς, στην περίπτωση ερωτήσεων των οποίων ο γράφος είναι δέντρο και όχι απλά γράφος, μόνο ο βαθμός εξόδου χρειάζεται να φραχθεί από κάποια σταθερά για να επιτύχουμε ασυμπτωτική βελτιστότητα. Σε κάθε περίπτωση, ο PartialPathStack **δεν δημιουργεί άχρηστα ενδιάμεσα αποτελέσματα**.

4.2.4 Σύγκριση Αλγορίθμων

Αρχικά παρουσιάζεται ένα παράδειγμα αποτίμησης ενός ερωτήματος μονοπατιού μερικής δομής που φαίνεται στο Σχήμα 4.9β πάνω σε ένα XML δέντρο που φαίνεται στο Σχήμα 4.9α. Μάλιστα στο Σχήμα 4.9β απεικονίζονται και οι ροές εισόδου του κάθε κόμβου ερωτήματος. Η αποτίμηση θα γίνει και με τους 3 αλγορίθμους που παρουσιάστηκαν στην προηγούμενη ενότητα. Στα Σχήματα 4.11, 4.12 και 4.13 απεικονίζονται οι στοίβες και τα αποτελέσματα των αλγορίθμων PartialMJ, PartialMJPlus και PartialPathStack αντιστοίχως. Στο Σχήμα 4.10α απεικονίζεται το επικαλύπτον δέντρο του γράφου του ερωτήματος του Σχήματος 4.9β, που χρειαζόμαστε για την αποτίμηση των PartialMJ και PartialMJPlus, ενώ στο Σχήμα 4.10β παρουσιάζονται οι συνθήκες συνένωσης που πρέπει να πληρούνται.



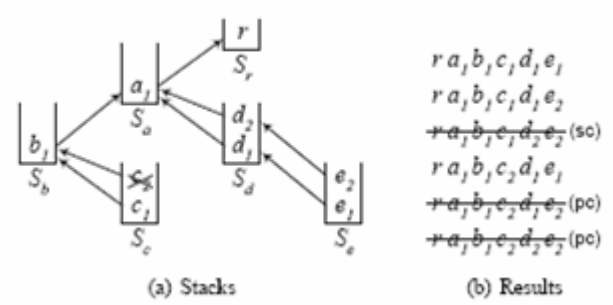
Σχήμα 4.9 : (α) XML δέντρο (β) Ερώτημα και οι ροές εισόδου του



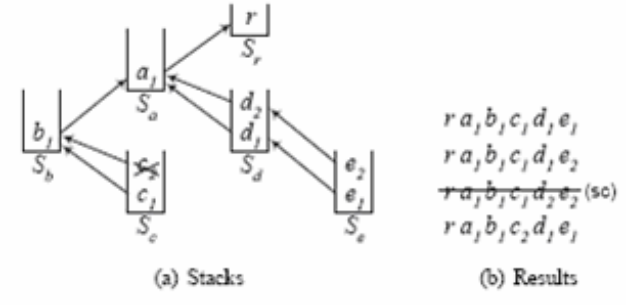
Σχήμα 4.10 : (α) Επικαλύπτον δέντρο του ερωτήματος του Σχήματος 4.9β (β) Συνθήκες συγχώνευσης των αποτελεσμάτων.

Το Σχήμα 4.10α δείχνει ένα επικαλύπτον δέντρο q_s της ερώτησης μερικού μονοπατιού q του Σχήματος 4.9β. Η ακμή d//c της q απουσιάζει από το q_s. Το Σχήμα 4.10β δείχνει τις συνθήκες που πρέπει να ισχύουν για τη συνένωση των αποτελεσμάτων των δύο μονοπατιών q_s. Δύο αποτελέσματα των μονοπατιών αυτών μπορούν να συνενωθούν αν ικανοποιούν τη συνθήκη ταυτότητας στα r και a, τη συνθήκη σχέσης στο d//c και τη συνθήκη μονοπατιού στα c και e.

Τα αποτελέσματα του αριστερού μονοπατιού του q_s είναι { $ra_1b_1c_1$, $ra_1b_1c_2$ } και του δεξιού μονοπατιού είναι { $ra_1d_1e_1$, $ra_1d_1e_2$, $ra_1d_2e_2$ }. Από τα έξι πιθανά αποτελέσματα της συνένωσης, τρία διαγράφονται επειδή δεν σέβονται τη συνθήκη μονοπατιού (pc) ή τη συνθήκη σχέσης (sc), όπως φαίνεται στο Σχήμα 4.11β. Αυτά τα αποτελέσματα ονομάζονται ενδιάμεσα αποτελέσματα. Το Σχήμα 4.11α δείχνει την κατάσταση των στοιβών μετά την αποτίμηση της ερώτησης q του Σχήματος 4.9β πάνω στα μονοδιάστατα XML δεδομένα του Σχήματος 4.9α..

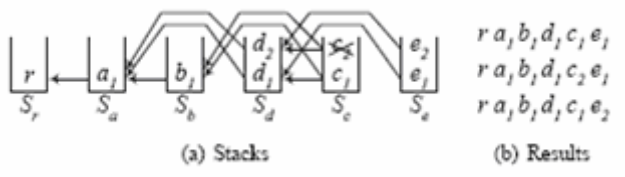


Σχήμα 4.11 : Στοίβες και αποτελέσματα του αλγορίθμου PartialMJ



Σχήμα 4.12 : Στοίβες και αποτελέσματα του αλγορίθμου PartialMJPlus

Το Σχήμα 4.12β δείχνει τα αποτελέσματα της ερώτησης q που παράγονται από τον PartialMJPlus. Από τα τέσσερα αποτελέσματα της συνένωσης, ένα διαγράφεται καθώς δεν σέβεται τη δομική σχέση d//c.



Σχήμα 4.13 : Στοίβες και αποτελέσματα του αλγορίθμου PartialPathStack

Το Σχήμα 4.13β δείχνει τα αποτελέσματα της ερώτησης q που παράγονται από τον PartialPathStack. Η τοπολογική διάταξη των κόμβων του ερωτήματος που χρησιμοποιείται είναι η : rabdce.

Μετά από την εκτέλεση ενός συνόλου πειραμάτων διαπιστώθηκε ότι ο ο PartialPathStack είναι πιο αποδοτικός από τον PartialMJ και τον PartialMJPlus.

Μάλιστα σε πείραμα που μετρήθηκε ο χρόνος εκτέλεσης των αλγορίθμων PartialPathStack και PartialMJPlus για την αποτίμηση των ερωτήσεων σε συνθετικά δεδομένα διαφόρων μεγεθών διαπιστώθηκε ότι ο PartialPathStack υπερτερεί του PartialMJPlus.

Στην αποτίμηση όλων των ερωτήσεων, μια αύξηση στο μέγεθος των δεδομένων (input size) συνεπάγεται αύξηση στον αριθμό των αποτελεσμάτων (output size). Όταν τα δεδομένα και τα αποτελέσματα αυξάνονται, ο χρόνος εκτέλεσης του PartialMJPlus και του PartialPathStack αυξάνεται. Αυτό επιβεβαιώνει την πολυπλοκότητα που δείχνει εξάρτηση του χρόνου εκτέλεσης από τα δεδομένα και τα αποτελέσματα. Η αύξηση στο χρόνο εκτέλεσης του PartialMJPlus όμως είναι έντονη από ότι του PartialPathStack. Αυτό οφείλεται στο γεγονός ότι ο PartialMJPlus περιλαμβάνει επιπλέον κόστος για (α) το χειρισμό του επικαλύπτοντος δέντρου, (β) τη συνένωση των αποτελεσμάτων των μονοπατιών του επικαλύπτοντος δέντρου, και (γ) τον εντοπισμό των αποτελεσμάτων που δεν σέβονται τις δομικές σχέσεις που απουσιάζουν από το επικαλύπτον δέντρο.

Συνολικά, ο PartialPathStack είναι πιο κλιμακούμενος από τον PartialMJPlus σε σχέση με το μέγεθος του δέντρου.

5

Ανάλυση Απαιτήσεων Συστήματος

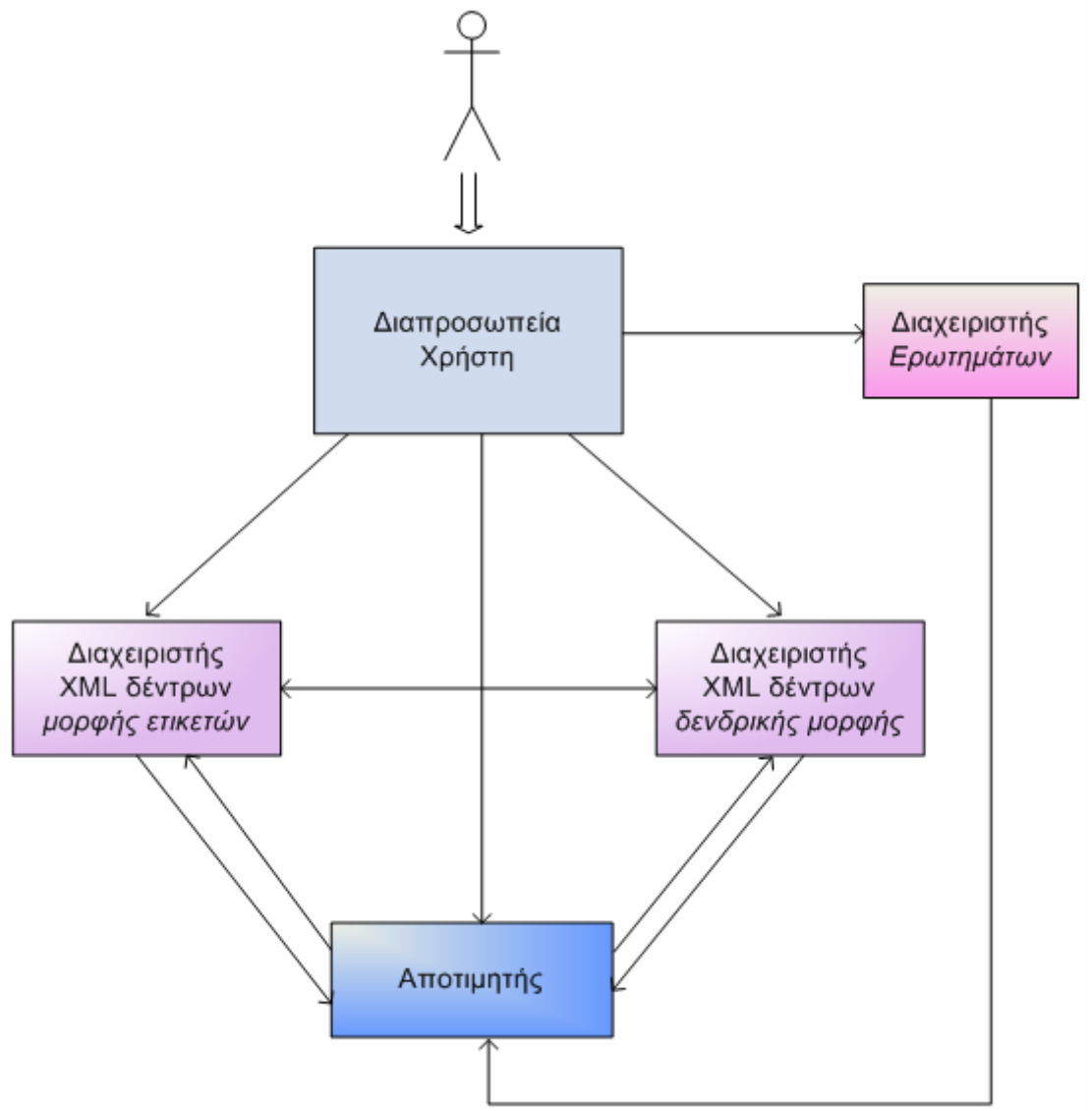
Στο κεφάλαιο αυτό παρουσιάζεται ο διαχωρισμός του συστήματος σε υποσυστήματα, απαριθμούνται οι απαιτήσεις κάθε υποσυστήματος και περιγράφεται η λειτουργία που το υποσύστημα θα πρέπει να επιτελεί.

5.1 Περιγραφή Αρχιτεκτονικής

Το σύστημα επεξεργασίας ερωτήσεων για δενδρικά δεδομένα χωρίζεται σε υποσυστήματα καθένα από τα οποία επιτελεί μια συγκεκριμένη λειτουργία. Τα υποσυστήματα αυτά συνεργάζονται μεταξύ τους ανταλλάσσοντας δεδομένα, χωρίς όμως το ένα να επεμβαίνει στις λειτουργίες του άλλου. Τα βασικά μέρη που απαρτίζουν το σύστημα είναι τα εξής:

- Διαπροσωπεία χρήστη
- Υποσύστημα Διαχείρισης XML Δέντρων μορφής ετικετών
- Υποσύστημα Διαχείρισης XML Δέντρων δενδρικής μορφής
- Υποσύστημα Διαχείρισης Ερωτημάτων
- Υποσύστημα Αποτίμησης ερωτήματος σε XML δέντρο

Ακολουθεί το σχήμα με την αρχιτεκτονική του συστήματος



Σχήμα 5.1 : Αρχιτεκτονική Συστήματος

5.2 Περιγραφή Υποσυστημάτων

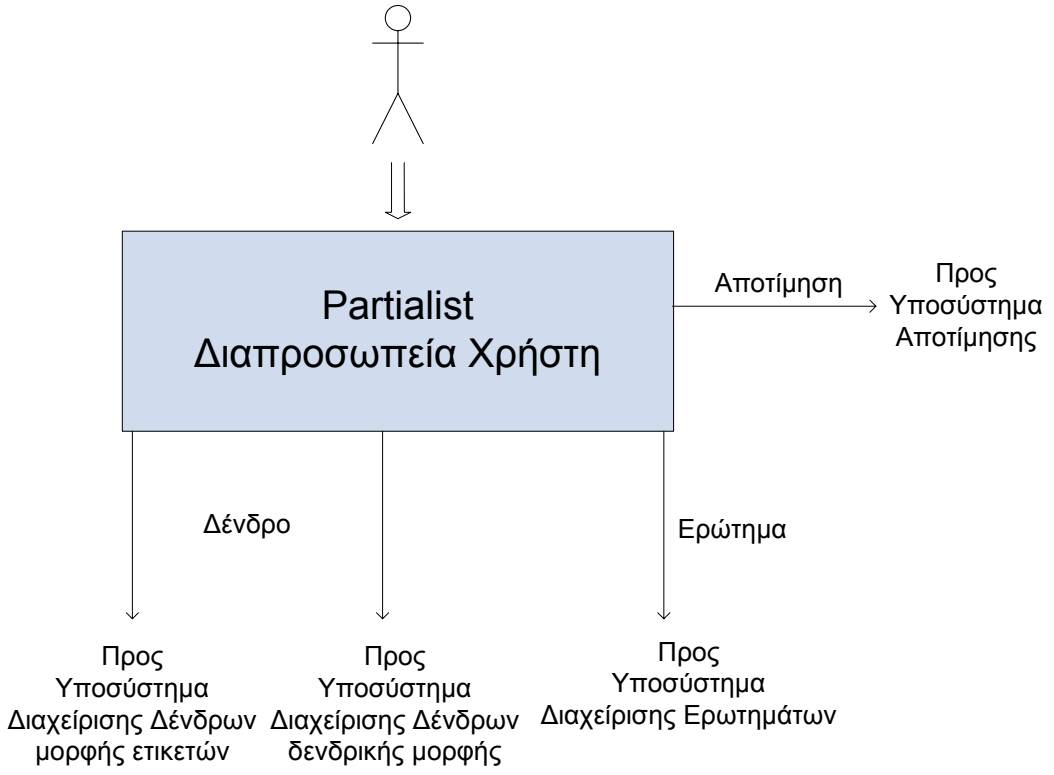
Εδώ, παρουσιάζονται οι απαιτήσεις κάθε υποσυστήματος και οι λειτουργίες που θα πρέπει αυτό να επιτελεί.

5.2.1 Διαπροσωπεία Χρήστη

Οι απαιτήσεις από το υποσύστημα της διαπροσωπείας είναι οι ακόλουθες:

- Το σύστημα θα πρέπει να είναι φιλικό προς το χρήστη.
- Η εκμάθηση χρήσης του συστήματος θα πρέπει να μην απαιτεί ούτε χρόνο ούτε το διάβασμα κάποιου εγχειριδίου χρήσης, αλλά να γίνεται μόνο με την ανάγνωση απλών οδηγιών που θα παρέχονται από το σύστημα είτε με μόνιμα μηνύματα είτε με την εμφάνιση εκτάκτων μηνυμάτων κάθε φορά που ο χρήστης αλληλεπιδρά με το σύστημα.
- Το περιβάλλον διαπροσωπείας θα πρέπει να βοηθά το χρήστη να αποφεύγει τα σφάλματα. Στην περίπτωση σφαλμάτων θα πρέπει να τον κατευθύνει και να του προτείνει ενέργειες προς αποφυγή τους.

Ακολουθεί το διάγραμμα που παρουσιάζει τις λειτουργίες του υποσυστήματος.



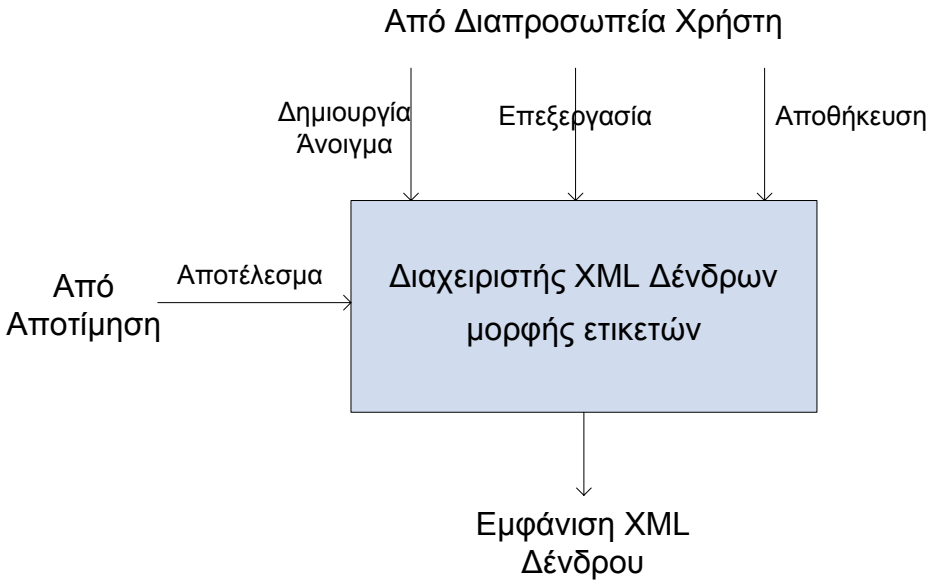
Σχήμα 5.2 : Διάγραμμα Ροής Δεδομένων στη Διαπροσωπεία Συστήματος.

Όπως φαίνεται και στο διάγραμμα ο χρήστης αρχικά εισάγεται στο *βασικό παράθυρο του συστήματος* και μπορεί να κάνει τέσσερεις ενέργειες :

- Διαχείριση δέντρου μορφής ετικετών όπου καταλήγει στο υποσύστημα διαχείρισης XML δέντρων μορφής ετικετών.
- Διαχείριση δέντρου δενδρικής μορφής όπου καταλήγει στο υποσύστημα διαχείρισης XML δέντρων δενδρικής μορφής.
- Διαχείριση ερωτήματος όπου καταλήγει στο υποσύστημα διαχείρισης ερωτημάτων.
- Αποτίμηση ερωτήματος σε δέντρο όπου καταλήγει στο υποσύστημα αποτίμησης. Τα αποτελέσματα της αποτίμησης είναι νέα XML δέντρα και καταλήγουν στο κατάλληλο υποσύστημα διαχείρισης δέντρων.

5.2.2 Διαχειριστής XML Δέντρων μορφής ετικετών

Ακολουθεί το διάγραμμα που παρουσιάζει τις λειτουργίες του υποσυστήματος.

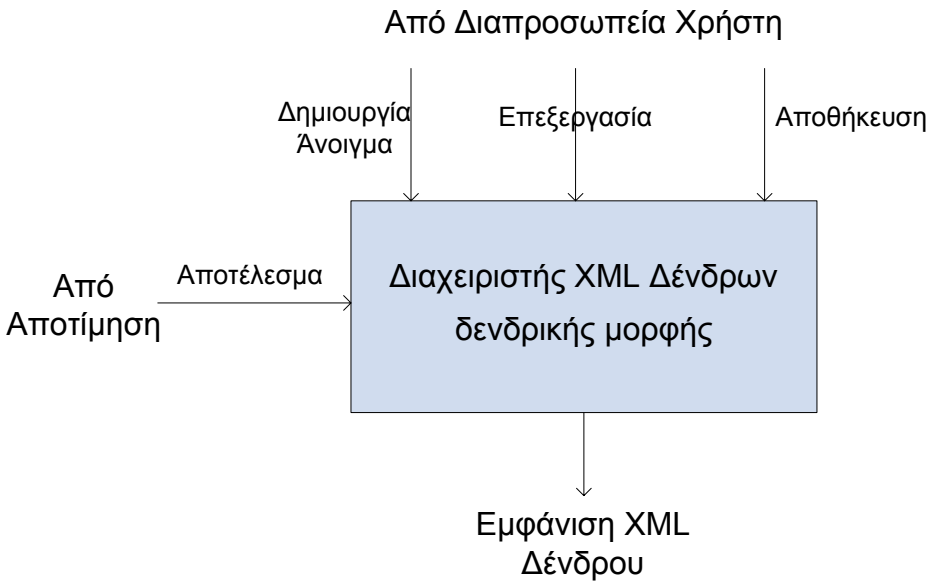


Σχήμα 5.3 : Διάγραμμα Ροής Δεδομένων στο Διαχειριστή XML Δέντρων μορφής ετικετών.

Από τη διαπροσωπεία χρήστη μπορούν να έρθουν ενέργειες δημιουργίας, ανοίγματος, επεξεργασίας και αποθήκευσης XML δέντρων μορφής ετικετών στο διαχειριστή XML δέντρων αυτής της μορφής. Επίσης ενέργειες δημιουργίας και επεξεργασίας XML δέντρων μπορούν να προέλθουν και από τα αποτελέσματα του υποσυστήματος αποτίμησης. Σε κάθε περίπτωση ο διαχειριστής XML δέντρων εμφανίζει το XML δέντρο σε μορφή ετικετών.

5.2.3 Διαχειριστής XML Δέντρων δενδρικής μορφής

Ακολουθεί το διάγραμμα που παρουσιάζει τις λειτουργίες του υποσυστήματος.



Σχήμα 5.4 : Διάγραμμα Ροής Δεδομένων στο Διαχειριστή XML Δέντρων δενδρικής μορφής.

Από τη διαπροσωπεία χρήστη μπορούν να έρθουν ενέργειες δημιουργίας, ανοίγματος, επεξεργασίας και αποθήκευσης XML δέντρων δενδρικής μορφής στο διαχειριστή XML δέντρων αυτής της μορφής. Επίσης ενέργειες δημιουργίας και επεξεργασίας XML δέντρων μπορούν να προέλθουν και από τα αποτελέσματα του υποσυστήματος αποτίμησης. Σε κάθε περίπτωση ο διαχειριστής XML δέντρων εμφανίζει το XML δέντρο σε δενδρική μορφή.

5.2.4 Διαχειριστής Ερωτημάτων

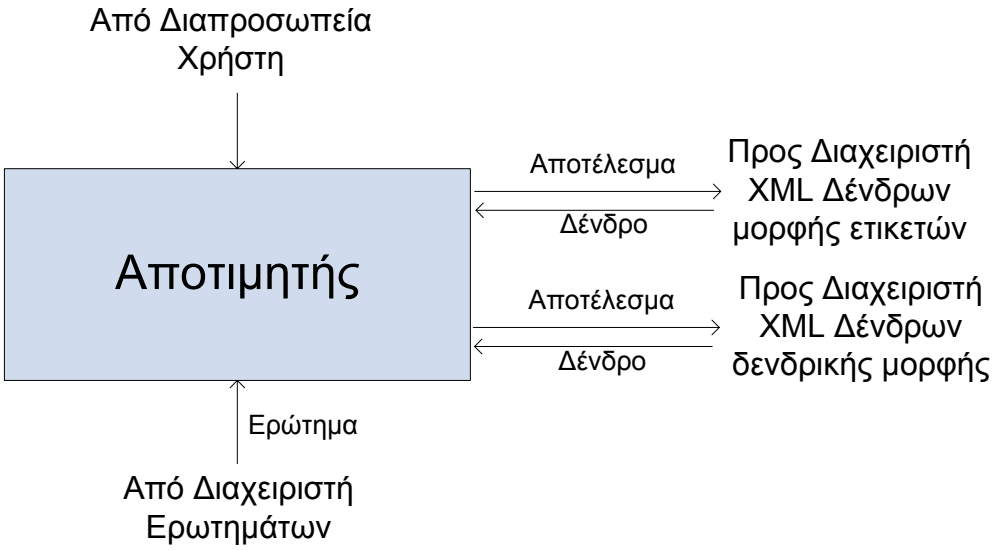
Ακολουθεί το διάγραμμα που παρουσιάζει τις λειτουργίες του υποσυστήματος.

Σχήμα 5.5 : Διάγραμμα Ροής Δεδομένων στο Διαχειριστή Ερωτημάτων.

Από τη διαπροσωπεία χρήστη μπορούν να έρθουν ενέργειες δημιουργίας, ανοίγματος, επεξεργασίας και αποθήκευσης ερωτήματος στο διαχειριστή ερωτημάτων ο οποίος εμφανίζει το ερώτημα.

5.2.5 Αποτιμητής

Ακολουθεί το διάγραμμα που παρουσιάζει τις λειτουργίες του υποσυστήματος.



Σχήμα 5.6 : Διάγραμμα Ροής Δεδομένων στον Αποτιμητή.

Από τη διαπροσωπεία χρήστη ζητείται να γίνει αποτίμηση ενός ερωτήματος που προέρχεται από το διαχειριστή ερωτημάτων πάνω σε ένα XML δέντρο που προέρχεται είτε από το διαχειριστή XML δέντρων μορφής ετικετών είτε από το διαχειριστή XML δέντρων δενδρικής μορφής. Ο αποτιμητής στέλνει τα αποτελέσματα στο κατάλληλο διαχειριστή XML δέντρων, δηλαδή σε αυτόν από τον οποίο πήρε το δέντρο προς αποτίμηση και τότε αυτός τα παρουσιάζει.

5.2.6 Σύνδεση Επιμέρους Υποσυστημάτων με τη Διαπροσωπεία Χρήστη

Τα τμήματα “Διαχείριση XML Δέντρων μορφής ετικετών”, “Διαχείριση XML Δέντρων δενδρικής μορφής”, “Διαχείριση Ερωτημάτων” και “Αποτίμηση” όπως είδαμε υλοποιούν το βασικό μέρος της εφαρμογής μας , εσωκλείοντας σχεδόν ολόκληρη τη λογική δομή της. Η διεπαφή όμως που παρέχουν στον εκάστοτε χρήστη είναι προγραμματιστική, δηλαδή η κλήση των λειτουργιών που έχουν περιγραφεί μπορεί να γίνει μόνο στα πλαίσια ενός προγράμματος.

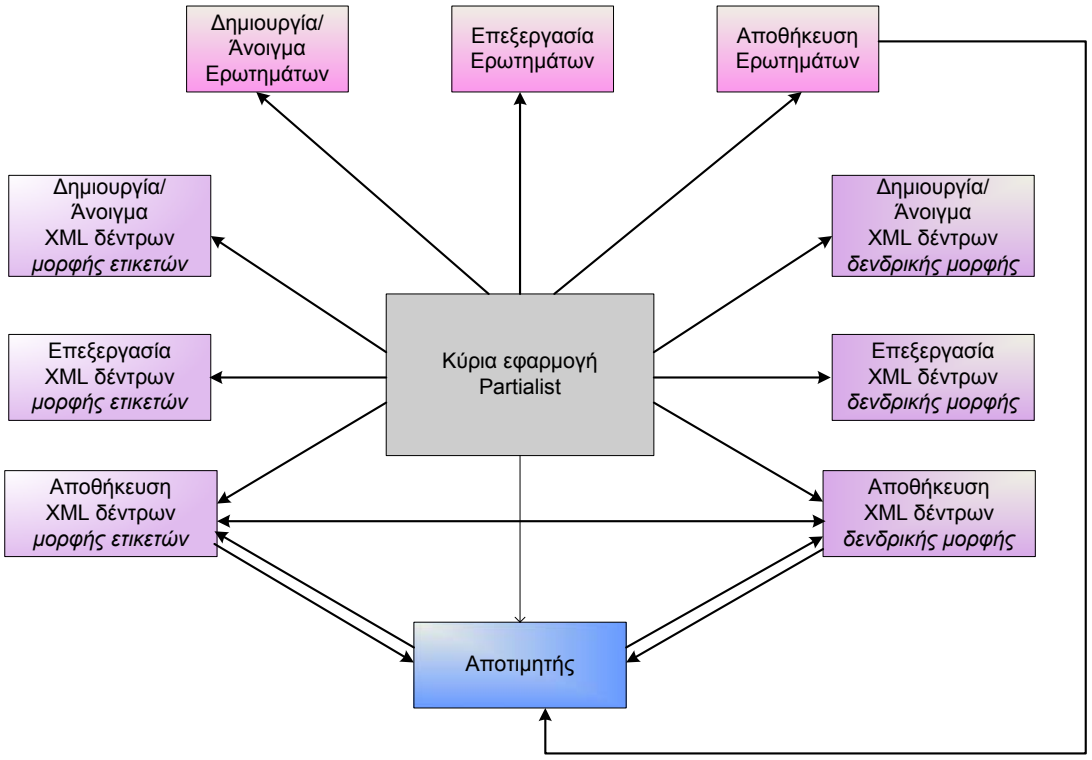
Το τμήμα "Διαπροσωπεία Χρήστη" επικάθεται πάνω από τα άλλα τέσσερα τμήματα, προσφέροντας μια πιο φιλική διεπαφή ανάμεσα στο χρήστη και τις λειτουργίες του προγράμματος. Μέσα από ένα γραφικό περιβάλλον, συνηθισμένο από άλλες "παραθυρικές" εφαρμογές των Windows, δίνεται η δυνατότητα στο χρήστη να καλέσει τις λειτουργίες του προγράμματος με τη βοήθεια μενού και κουμπιών. Έτσι λοιπόν με τα διάφορα μενού και κουμπιά που υπάρχουν, εκτελούνται βασικές λειτουργίες αλλαγής της κατάστασης ενός γράφου (δημιουργία κόμβων και ακμών κάθε τύπου, αφαίρεση κόμβων και ακμών), λειτουργίες μετατροπής της μορφής ενός XML δέντρου, λειτουργίες φόρτωσης και αποθήκευσης ενός γράφου καθώς και λειτουργίες που αφορούν την αποτίμηση ενός ερωτήματος πάνω σε ένα δέντρο.

Ο γράφος απεικονίζεται σε μια επιφάνεια σχεδιασμού, όπου με τη βοήθεια του ποντικιού μπορεί να γίνει η επιλογή και μετακίνηση ενός ή περισσότερων κόμβων έτσι ώστε κάθε φορά να έχουμε το επιθυμητό οπτικό αποτέλεσμα. Επιλέγοντας έναν κόμβο ή μια ακμή μπορούμε να δούμε πληροφορίες για το επιλεγμένο αντικείμενο ή μπορούμε να το αφαιρέσουμε από το γράφο.

Στην επιφάνεια σχεδιασμού γίνονται ορατά και τα αποτελέσματα της αποτίμησης ενός ερωτήματος πάνω σε ένα ερώτημα.

Επίσης σε περίπτωση που ο χρήστης προσπαθήσει να εκτελέσει μια λειτουργία που δεν ικανοποιεί τις προδιαγραφές του συστήματος τότε εμφανίζονται στο χρήστη κατάλληλα μηνύματα. Αν πρόκειται για ερώτημα όπου υπάρχουν αρκετές απαγορεύσεις όσον αφορά την προσθήκη κόμβων και ακμών, όπως θα δούμε παρακάτω, τότε επιλέγονται πάνω στο γράφο οι κόμβοι και οι ακμές που ήδη υπάρχουν και είναι η αιτία που δεν μπορεί να προστεθεί ο νέος κόμβος ή η νέα ακμή.

Η τρέχουσα μορφή του γράφου ενός δέντρου μπορεί να αποθηκευτεί σε XML αρχείο ενώ ταυτόχρονα παράγεται και η κωδικοποίηση θέσης του που αποθηκεύεται σε ένα txt αρχείο. Η τρέχουσα μορφή ενός ερωτήματος αποθηκεύεται σε ένα txt αρχείο σε μια προσυμφωνημένη μορφή που θα δούμε στην Ενότητα 6.2. Από τα αρχεία αυτά μπορεί να γίνει ανάκτηση του κάθε γράφου.



Σχήμα 5.7 : Βασικές Λειτουργίες του συστήματος.

5.3 Ανάλυση Απαιτήσεων

Στην ενότητα αυτή δίνονται οι απαιτήσεις του συστήματος. Αρχικά δίνονται οι βασικές λειτουργίες της εφαρμογής. Μετά δίνονται οι προδιαγραφές της εμφάνισης ενός XML δέντρου σε δύο ισοδύναμες μορφές. Έπειτα δίνονται οι προδιαγραφές των κόμβων ενός δέντρου και ενός ερωτήματος. Τέλος δίνονται οι απαιτήσεις που αφορούν τις ακμές ενός ερωτήματος.

5.3.1 Βασικές Λειτουργίες

Οι βασικές λειτουργίες της εφαρμογής μας παρουσιάζονται συνοπτικά αμέσως μετά. Στις ενότητες που ακολουθούν θα παρουσιαστούν οι λεπτομέρειες τους.

Διαχείριση Αρχείων (μενού File)

- Δημιουργία νέου XML δέντρου σε δενδρική μορφή
- Δημιουργία νέου XML δέντρου σε μορφή ετικετών
- Δημιουργία νέου ερωτήματος

- Άνοιγμα υπάρχοντος XML δέντρου σε δενδρική μορφή
- Άνοιγμα υπάρχοντος XML δέντρου σε μορφή ετικετών
- Άνοιγμα υπάρχοντος ερωτήματος

- Αποθήκευση XML δέντρου
- Αποθήκευση ερωτήματος

- Κλείσιμο XML δέντρου
- Κλείσιμο ερωτήματος

- Εξαγωγή Εικόνας XML δέντρου σε δενδρική μορφή
- Εξαγωγή Ερωτήματος

- Έξοδος

Επεξεργασία Δεδομένων (μενού Edit)

Το μενού Edit παίρνει τρεις μορφές. Αλλάζει ανάλογα με το αν έχω XML δέντρο σε δενδρική μορφή ή σε μορφή ετικετών ή αν έχω ερώτημα.

1. XML δέντρο σε μορφή ετικετών
    - Μετονομασία κόμβου
    - Προσθήκη κόμβου/κόμβων
    - Διαγραφή κόμβου/κόμβων
    - Αλλαγή πατέρα κόμβου

2. XML δέντρο σε δενδρική μορφή
    - Προσθήκη κόμβου/κόμβων
    - Διαγραφή κόμβου/κόμβων
    - Αλλαγή πατέρα κόμβου

3. Ερώτημα
    - Προσθήκη κόμβου/κόμβων
    - Προσθήκη ακμής παιδιού-πατέρα
    - Προσθήκη ακμής προγόνου-απόγονου
    - Προσθήκη κόμβου και ακμής
    - Διαγραφή κόμβου/κόμβων
    - Διαγραφή ακμής/ακμών
    - Ορισμός κόμβου εξόδου

Προβολή Δεδομένων (μενού View)

Το μενού View παίρνει τρεις μορφές. Αλλάζει ανάλογα με το αν έχω XML δέντρο σε δενδρική μορφή ή σε μορφή ετικετών ή αν έχω ερώτημα.

- Εμφανίζει διάφορα στοιχεία που μας ενδιαφέρουν για το συγκεκριμένο δέντρο ή ερώτημα.

Μετατροπή Μορφής XML Δέντρου (μενού Conversion)

- Μετατροπή ενός XML δέντρου από τη δενδρική μορφή στη μορφή με τις XML ετικέτες.
- Μετατροπή ενός XML δέντρου από τη μορφή με τις XML ετικέτες στη δενδρική μορφή.
  Απαγορεύουμε τη μετατροπή από τη μορφή ετικετών στη δενδρική μορφή αν έχουμε μεγάλο αρχείο.

Αποτίμηση (μενού Evaluation)

- Εξαγωγή κωδικοποίησης θέσης ενός XML δέντρου.
- Εξαγωγή treeData ενός XML δέντρου.

- Έλεγχος αν έχουμε πλήρως ορισμένο ερώτημα μονοπατιού ώστε να υπάρχει δυνατότητα αποτίμησής του με τον αλγόριθμο PathStack.
- Σύνδεση κόμβων ερωτήματος με τη ρίζα ή παιδί της όταν δεν έχουν εισερχόμενες ακμές ώστε να έρθει το ερώτημά μας σε κανονική μορφή, κατάλληλη για την αποτίμηση.

- Αποτίμηση μιας ερώτησης σε ένα XML δέντρο

- Καθαρισμός αποτελεσμάτων αποτίμησης

5.3.1.1 Δημιουργία – Άνοιγμα – Αποθήκευση

Το σύνολο των λειτουργιών new – open – save εκφράζουν τη λειτουργικότητα εισόδου – εξόδου της εφαρμογής.

Δημιουργία

Όταν δημιουργώ ένα δέντρο ή ένα ερώτημα, αρχικά δημιουργώ τα νέα δεδομένα του δέντρου ή του ερωτήματος και μια νέα καρτέλα για να τα παρουσιάσω, με τίτλο όπως αναφέρεται στην Ενότητα 5.3.2. Έπειτα ορίζεται η ρίζα του δέντρου ή ερωτήματος. Δεν τελειώνει η διαδικασία της δημιουργίας μέχρι να δοθεί από το χρήστη ένα έγκυρο όνομα στη ρίζα (πχ root).

Άνοιγμα

Κάθε νέο αρχείο που ανοίγω το παρουσιάζω σε νέα καρτέλα. Αν το αρχείο όμως που επιλέγω είναι ήδη ανοιχτό, δεν μπορώ να το ανοίξω σε δεύτερη καρτέλα.

Ακόμα, το αρχείο που ανοίγω πρέπει να είναι αποθηκευμένο σε μια προσυμφωνημένη μορφή (το δέντρο σε έγκυρο XML αρχείο, ενώ το ερώτημα όπως παρουσιάζεται αναλυτικά στην Ενότητα 6.2.4) .

- Άνοιγμα XML Δέντρου
  Το αρχείο που επιλέγω να ανοίξω πρέπει να έχει κατάληξη xml και να είναι ένα σωστά ορισμένο XML αρχείο, αλλιώς δεν είναι δυνατή η παρουσίασή του σε δέντρο. Επίσης δεν πρέπει να είναι ήδη ανοιχτό.

  Όταν ανοίγω ένα έγκυρο δέντρο, αρχικά δημιουργώ τα νέα δεδομένα του δέντρου και μια νέα καρτέλα για να τα παρουσιάσω, με τίτλο το όνομα του αρχείου. Έπειτα ορίζεται η ρίζα του δέντρου από το πρώτο στοιχείο του XML αρχείου. Μετά διασχίζω όλο το XML αρχείο και εμφανίζω όλα τα στοιχεία του.

  Αν πρόκειται για δέντρο σε δενδρική μορφή, πρέπει να κάνω έναν επιπλέον έλεγχο του αρχείου που ανοίγω. Πρέπει το μέγεθος του αρχείου να μην είναι μεγάλο. Αν είναι, δεν προχωράω στην απεικόνισή του σε δενδρική μορφή γιατί δεν θα μπορούν να διακριθούν οι κόμβοι αφού θα πέφτει ο ένας πάνω στον άλλον (βλέπε Ενότητα 5.3.3).

- Άνοιγμα Ερωτήματος
  Το αρχείο που επιλέγω να ανοίξω πρέπει να έχει κατάληξη txt και να είναι ένα σωστά ορισμένο ερώτημα, όπως παρουσιάζεται αναλυτικά στην Ενότητα 6.2.4, αλλιώς δεν είναι δυνατή η παρουσίασή του. Επίσης δεν πρέπει να είναι ήδη ανοιχτό.

  Όταν ανοίγω ένα έγκυρο ερώτημα, αρχικά δημιουργώ τα νέα δεδομένα του ερωτήματος και μια νέα καρτέλα για να τα παρουσιάσω, με τίτλο το όνομα του αρχείου. Έπειτα κάνω parsing στο αρχείο με την προσυμφωνημένη μορφή, ορίζω τη ρίζα του ερωτήματος και εμφανίζω όλα τα στοιχεία του (κόμβους και ακμές).

  Αν έχει οριστεί κόμβος εξόδου, τον εμφανίζω στο γράφο.

Αποθήκευση

Επιλέγω σε ποιο αρχείο θέλω να αποθηκεύσω τα δεδομένα μου. Σαν προεπιλεγμένη επιλογή είναι το αρχείο που αποθήκευα μέχρι στιγμής τις αλλαγές, και αν δεν υπάρχει τέτοιο αρχείο τότε έχει οριστεί ένας φάκελος στον οποίο θα αποθηκευτεί το νέο αρχείο, το οποίο θα πάρει το όνομα του τίτλου της καρτέλας με κατάληξη xml αν πρόκειται για δέντρο ή με κατάληξη txt αν πρόκειται για ερώτημα.

Αφού επιλέξω το αρχείο, αποθηκεύω τα δεδομένα που παρουσιάζονται στην καρτέλα σε μια προσυμφωνημένη μορφή :

- Αν πρόκειται για δέντρο, τότε τα δεδομένα του πρέπει να αποθηκευτούν σε έγκυρο XML αρχείο. Ταυτόχρονα παράγεται η κωδικοποίηση θέσης του δέντρου, η οποία αποθηκεύεται σε ένα αρχείο που τοποθετείται στον ίδιο φάκελο που βρίσκεται το XML αρχείο που αποθήκευσα και που παίρνει το ίδιο όνομα με αυτό αλλά με κατάληξη txt.

  Δηλαδή στην αποθήκευση ενός δέντρου αποθηκεύονται δύο αρχεία με το ίδιο μονοπάτι και όνομα αλλά με διαφορετική κατάληξη :

  1. Το XML αρχείο με κατάληξη xml
  2. Το αρχείο κωδικοποίησης θέσης με κατάληξη txt
- Αν πρόκειται για ερώτημα, τότε τα δεδομένα του πρέπει να αποθηκευτούν στη μορφή που παρουσιάζεται αναλυτικά στην Ενότητα 6.2.4.

Από τη στιγμή που θα γίνει η αποθήκευση ο τίτλος της καρτέλας θα πάρει το όνομα του αρχείου, χωρίς '*' (βλέπε Ενότητα 5.3.2).

Κλείσιμο – Έξοδος

Πριν κλείσει ο χρήστης μία καρτέλα ή την εφαρμογή, του ζητείται αν θέλει να αποθηκεύσει πρώτα τα δεδομένα.

Εξαγωγή Εικόνας

Αφορά μόνο XML δέντρα σε δενδρική μορφή και ερωτήματα.

Αποθηκεύουν την εικόνα του γράφου σε ένα αρχείο εικόνας.

5.3.1.2 Επεξεργασία

Η επεξεργασία είναι διαφορετική για το δέντρο ή το ερώτημα που επεξεργαζόμαστε. Αλλάζει ανάλογα με το αν έχω XML δέντρο σε δενδρική μορφή ή σε μορφή ετικετών ή αν έχω ερώτημα. Στη συνέχεια βλέπουμε αυτές τις λειτουργίες της επεξεργασίας.

Επεξεργασία Δέντρου σε μορφή ετικετών

Οι λειτουργίες επεξεργασίας σε ένα XML δέντρο σε μορφή ετικετών είναι :

- Αλλαγή ονόματος επιλεγμένου κόμβου
- Προσθήκη κόμβου ως παιδί του επιλεγμένου κόμβου
- Προσθήκη κόμβου ως προηγούμενος αδερφός του επιλεγμένου κόμβου
- Προσθήκη κόμβου ως επόμενος αδερφός του επιλεγμένου κόμβου
- Προσθήκη πολλαπλών κόμβων με κοινό πατέρα τον επιλεγμένο κόμβο
- Διαγραφή κόμβου/κόμβων που έχουν επιλεχθεί καθώς και των υποδέντρων τους
- Διαγραφή παιδιών και απογόνων του επιλεγμένου κόμβου (δηλαδή διαγραφή του υποδέντρου του). Ο επιλεγμένος κόμβος παραμένει.
- Αντικατάσταση πατέρα του επιλεγμένου κόμβου

Επεξεργασία Δέντρου σε δενδρική μορφή

Οι λειτουργίες επεξεργασίας σε ένα XML δέντρο σε δενδρική μορφή είναι :

- Προσθήκη κόμβου ως παιδί του επιλεγμένου κόμβου
- Προσθήκη κόμβου ως ενδιάμεσο παιδί των κόμβων πηγής και προορισμού της επιλεγμένης ακμής
- Προσθήκη πολλαπλών κόμβων ως παιδιά του επιλεγμένου κόμβου
- Διαγραφή κόμβου/κόμβων που έχουν επιλεχθεί μαζί με τα υποδέντρα τους
- Διαγραφή κόμβου/κόμβων που έχουν επιλεχθεί χωρίς τα υποδέντρα τους. Τα παιδιά των κόμβων που διαγράφονται παίρνουν τώρα σαν πατέρα τον παππού τους.
- Αντικατάσταση πατέρα του επιλεγμένου κόμβου

Επεξεργασία Ερωτήματος

Οι λειτουργίες επεξεργασίας σε ένα ερώτημα είναι :

- Προσθήκη κόμβου
- Προσθήκη πολλαπλών κόμβων
- Προσθήκη ακμής παιδιού-πατέρα
- Προσθήκη ακμής προγόνου-απόγονου
- Προσθήκη κόμβου και ακμής
- Διαγραφή κόμβου/κόμβων γραφικά ή από λίστα. Όταν διαγράφεται κάποιος κόμβος, τότε διαγράφονται και όλες οι συνδεδεμένες του ακμές.
- Διαγραφή ακμής/ακμών γραφικά ή από λίστα
- Διαγραφή κόμβων και ακμών που έχουν επιλεχθεί γραφικά.
- Διαγραφή ακμών του επιλεγμένου κόμβου
- Ορισμός κόμβου εξόδου

5.3.1.3 Προβολή

Προβολή Δεδομένων Δέντρου σε μορφή ετικετών

Οι λειτουργίες προβολής δεδομένων σε ένα XML δέντρο σε μορφή ετικετών είναι :

- Εμφάνιση όλου του υποδέντρου του επιλεγμένου κόμβου, σε περίπτωση που έχει κρυμμένους απογόνους

Προβολή Δεδομένων Δέντρου σε δενδρική μορφή

Οι λειτουργίες προβολής δεδομένων σε ένα XML δέντρο σε δενδρική μορφή είναι :

- Επιλογή όλου του υποδέντρου του επιλεγμένου κόμβου
- Προβολή δεδομένων του επιλεγμένου κόμβου
- Προβολή θέσης του επιλεγμένου κόμβου
- Εμφάνιση όλων των κόμβων
- Προβολή πηγής και προορισμού της επιλεγμένης ακμής

Προβολή Δεδομένων Ερωτήματος

Οι λειτουργίες προβολής δεδομένων σε ένα ερώτημα είναι :

- Προβολή όλων των κόμβων και των ακμών του ερωτήματος
- Προβολή των ακμών του ερωτήματος
- Προβολή δεδομένων του επιλεγμένου κόμβου

5.3.1.4 Μετατροπή Μορφής XML Δέντρου

Η εφαρμογή μας δίνει τη δυνατότητα της μετατροπής ενός XML δέντρου στις εναλλακτικές μορφές του :

- Αν το XML δέντρο εμφανίζεται με τη δενδρική μορφή, τότε μπορούμε να το μετατρέψουμε στη μορφή ετικετών
- Αν το XML δέντρο εμφανίζεται με τη μορφή ετικετών, τότε μπορούμε να το μετατρέψουμε στη δενδρική μορφή. Εδώ όμως πρέπει να ελέγξουμε πρώτα αν το αρχείο είναι μεγάλο. Αν πρόκειται για μεγάλο αρχείο, τότε δεν γίνεται η μετατροπή σε δενδρική μορφή γιατί θα είναι πολύ φορτωμένη
  Υπάρχει λεπτομερέστερη αναφορά για το θέμα αυτό στην Ενότητα 5.3.3.

5.3.1.5 Αποτίμηση

Αποθήκευση XML Δέντρου στη μορφή κωδικοποίηση θέσης

Αποθηκεύει σε txt αρχείο τα δεδομένα του δέντρου στη μορφή κωδικοποίησης θέσης

Βλέπε Ενότητα 6.2.2 για τη μορφή αυτή.

Αποθήκευση XML Δέντρου στη μορφή treeData

Αποθηκεύει σε txt αρχείο τα δεδομένα του δέντρου στη μορφή treeData.

Βλέπε Ενότητα 6.2.3 για τη μορφή αυτή.

Έλεγχος ερωτήματος μονοπατιού

Ελέγχει αν έχουμε πλήρως ορισμένο ερώτημα μονοπατιού ώστε να υπάρχει δυνατότητα αποτίμησής του με τον αλγόριθμο PathStack.

Εύρεση κόμβων χωρίς εισερχόμενες ακμές

Βρίσκει και συνδέει τους κόμβους του ερωτήματος με τη ρίζα ή παιδί της που δεν έχουν εισερχόμενες ακμές

Καθαρισμός αποτελεσμάτων αποτίμησης

Αφορά δέντρο σε δενδρική μορφή στο οποίο φαίνονται οι κόμβοι αποτελέσματος.

Η λειτουργία αυτή επαναφέρει το δέντρο στη μορφή πριν την αποτίμηση, οπότε δεν είναι πλέον μαρκαρισμένοι στο δέντρο οι κόμβοι εξόδου.

Αποτίμηση μιας ερώτησης σε ένα XML δέντρο

Αποτιμά ένα ερώτημα πάνω σε ένα XML δέντρο. Υπάρχει λεπτομερέστερη αναφορά για το θέμα αυτό στην Ενότητα 5.3.4.

5.3.2 Καρτέλες

Η εφαρμογή μας μπορεί να περιέχει ένα σύνολο από καρτέλες.

Περιεχόμενο καρτέλας

Κάθε καρτέλα μπορεί να περιέχει ένα δέντρο ή ένα ερώτημα. Κάθε καρτέλα που περιέχει δέντρο μπορεί να το εμφανίζει είτε σε μορφή ετικετών είτε σε δενδρική μορφή κάθε φορά , που μπορεί όμως να την εναλλάξει στην άλλη μορφή.

1. Δέντρα σε μορφή ετικετών
   Εμφάνιση
   - Οι κόμβοι παρουσιάζονται σε εμφωλιασμένη δομή.
   - Ο πατέρας έχει εμφωλιασμένα τα παιδιά του και μπορεί
     - πατώντας το κουτάκι + που έχει στα αριστερά του να τα εμφανίζει , αν είναι κρυμμένα
     - πατώντας το κουτάκι - που έχει στα αριστερά του να τα κρύβει , αν αυτά εμφανίζονται.

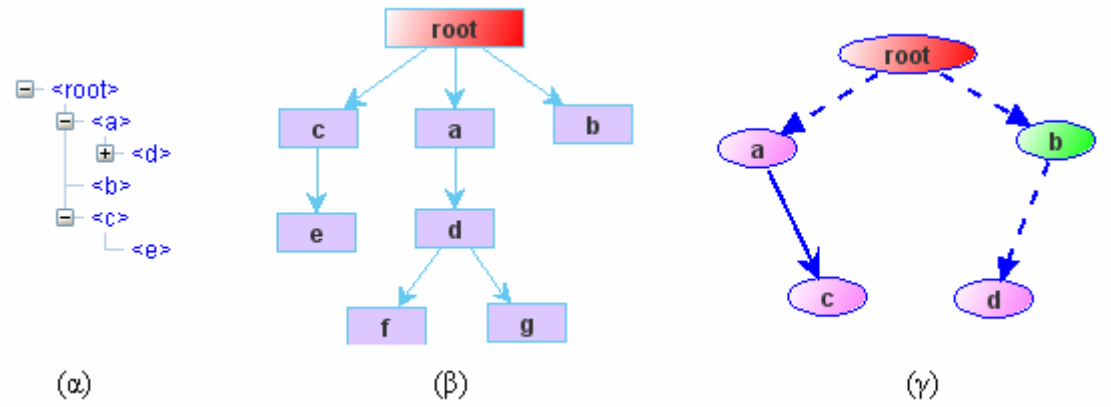
2. Δέντρα σε δενδρική μορφή
   Εμφάνιση - Χρώματα

   - Όλοι οι κόμβοι έχουν σχήμα ορθογωνίου
   - Ο κόμβος-ρίζα έχει κόκκινο χρώμα.
   - Οι κοινοί κόμβοι έχουν απαλό μωβ χρώμα (συνεχές).
   - Οι κόμβοι αποτελέσματος ενός ερωτήματος πάνω στο δέντρο αυτό έχουν πράσινο χρώμα.
   - Οι ακμές είναι λεπτές, συνεχείς με γαλάζιο χρώμα.

3. Ερωτήματα.
   Εμφάνιση - Χρώματα

   - Όλοι οι κόμβοι έχουν σχήμα κύκλου
   - Ο κόμβος-ρίζα έχει κόκκινο χρώμα.
   - Οι κοινοί κόμβοι έχουν ροζ χρώμα (αρχίζουν από αριστερά με απαλό ροζ και όσο προχωράει δεξιά σκουραίνει σταδιακά).
   - Ο κόμβος εξόδου έχει πράσινο χρώμα.
   - Οι ακμές που δηλώνουν σχέσεις πατέρα-παιδιού είναι συνεχείς με χρώμα μπλε.
   - Οι ακμές που δηλώνουν σχέσεις προγόνου-απόγονου είναι διακεκομμένες με χρώμα μπλε.

Στο σχήμα που ακολουθεί απεικονίζονται τα τρία αυτά είδη.



Σχήμα 5.8 : Περιεχόμενο καρτέλας. (α) Δέντρο μορφής ετικετών (β) Δέντρο δενδρικής μορφής (γ) Ερώτημα.

Χρώμα Τίτλου Καρτέλας

Για να διαχωρίζουμε τα δέντρα από τα ερωτήματα με την πρώτη ματιά χωρίς να χρειαστεί να δούμε κάθε καρτέλα ξεχωριστά ποια από τις παραπάνω τρεις μορφές έχουμε, χρωματίζουμε με τον εξής τρόπο τον τίτλο κάθε καρτέλας :

- με απαλό μωβ τα δέντρα
- με έντονο μωβ το επιλεγμένο δέντρο
- με απαλό ροζ τα ερωτήματα
- με έντονο ροζ το επιλεγμένο ερώτημα
- με πράσινο το δέντρο αποτελέσματος που παράγεται στο τέλος ενός ικανοποιήσιμου ερωτήματος πάνω σε ένα δέντρο. Το πράσινο χάνεται από τη στιγμή που ο χρήστης επιλέξει να δει το δέντρο-αποτέλεσμα και παίρνει τα κλασικά χρώματα που ορίσαμε παραπάνω (μωβ έντονο όταν είναι επιλεγμένο και απαλό όταν δεν είναι).

Σχήμα 5.9 : Καρτέλες εφαρμογής.

Στο Σχήμα 5.9 υπάρχουν 5 δέντρα (3 μη-επιλεγμένα δέντρα με απαλό μωβ, 1 επιλεγμένο δέντρο με έντονο μωβ και 1 δέντρο-αποτέλεσμα με πράσινο) και 3 ερωτήματα (3 μη-επιλεγμένα ερωτήματα με επιλεγμένο ροζ). Θα ακολουθήσουν σε επόμενα κεφάλαια αρκετά σχήματα όπου θα δούμε και πως εμφανίζονται επιλεγμένα ερωτήματα.

Όνομα Τίτλου Καρτέλας

Τα δεδομένα κάθε καρτέλας αποθηκεύονται σε ένα αρχείο. Ο τίτλος κάθε καρτέλας είναι το όνομα του αρχείου στο οποίο είναι αποθηκευμένο τα δεδομένα της καρτέλας.

- Δημιουργία Δέντρου : ο τίτλος της νέας καρτέλας θα είναι '* Tree 0', όπου το 0 θα αυξάνεται σε κάθε νέα δημιουργία δέντρου.
- Δημιουργία Ερωτήματος : ο τίτλος της νέας καρτέλας θα είναι '* Query 0', όπου το 0 θα αυξάνεται σε κάθε νέα δημιουργία ερωτήματος.
- Άνοιγμα Δέντρου : αν το όνομα του αρχείου που ανοίγω είναι το myTree.xml τότε ο τίτλος της νέας καρτέλας θα είναι 'myTree.xml'
- Άνοιγμα Ερωτήματος : αν το όνομα του αρχείου που ανοίγω είναι το myQuery.txt τότε ο τίτλος της νέας καρτέλας θα είναι 'myQuery.txt '

Το σύμβολο '*' στην αρχή του τίτλου, δηλώνει ότι τα δεδομένα της καρτέλας έχουν αλλάξει και δεν έχουν αποθηκευτεί οι νέες αλλαγές στο αρχείο.

Οπότε :

- Αλλαγή δεδομένων (προσθήκη, διαγραφή, αλλαγή κόμβου εξόδου κτλ) :
  - αν δεν υπάρχει στον τίτλο '*', προσθέτουμε στην αρχή του τίτλου ένα '*', μετά ένα κενό χαρακτήρα και έπειτα ακολουθεί το όνομα του αρχείου.
  - αν υπάρχει στον τίτλο '*', σημαίνει ότι ήδη έχουν γίνει αλλαγές που δεν έχουν ακόμα αποθηκευτεί, οπότε δεν πειράζουμε τον τίτλο.
- Αποθήκευση δεδομένων :
  - αν δεν υπάρχει στον τίτλο '*', σημαίνει ότι δεν έχουν γίνει αλλαγές και είναι ήδη αποθηκευμένο το αρχείο, οπότε δεν πειράζουμε τον τίτλο.
  - αν υπάρχει στον τίτλο '*', το αφαιρούμε μαζί με τον κενό χαρακτήρα που έπεται και μένει μόνο το όνομα του αρχείου.

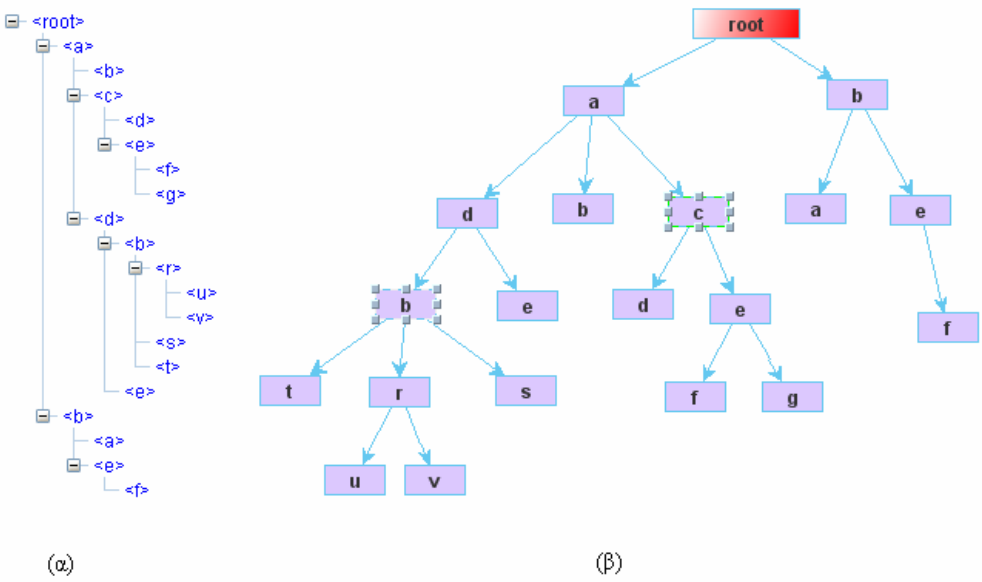
Στο Σχήμα 5.9 βλέπουμε ότι σε 5 καρτέλες έχουν σωθεί σε αρχείο τα δεδομένα τους (απουσία του συμβόλου '*' στην αρχή του τίτλου) ενώ στις υπόλοιπες 3 δεν έχουν σωθεί (ύπαρξη του συμβόλου '*' στην αρχή του τίτλου).

5.3.3 Εμφάνιση XML Δέντρου

Υπάρχουν δύο μορφές απεικόνισης ενός XML δέντρου στην εφαρμογή μας

1. δενδρική μορφή
2. μορφή με XML ετικέτες

Οι δύο μορφές είναι ισοδύναμες. Μπορώ να μετατρέψω ένα XML δέντρο από τη μια μορφή στην άλλη.



Σχήμα 5.10 : XML δέντρο. (α) το δέντρο στη μορφή ετικετών και (β) το δέντρο στη δενδρική μορφή. Οι δύο μορφές είναι ισοδύναμες.

.

Μεγάλο XML αρχείο

Αν όμως το XML αρχείο είναι πολύ μεγάλο, τότε η δενδρική μορφή δεν είναι κατάλληλη για την αναπαράστασή του και την απαγορεύουμε, αφού το πλήθος των κόμβων θα είναι μεγάλο και έτσι οι κόμβοι θα πέφτουν ο ένας πάνω στον άλλο.

Οι μόνες λειτουργίες που επηρεάζονται από τον παράγοντα αυτό είναι το άνοιγμα ενός δέντρου σε δενδρική μορφή και η μετατροπή ενός δέντρου από τη μορφή ετικετών στη δενδρική μορφή.

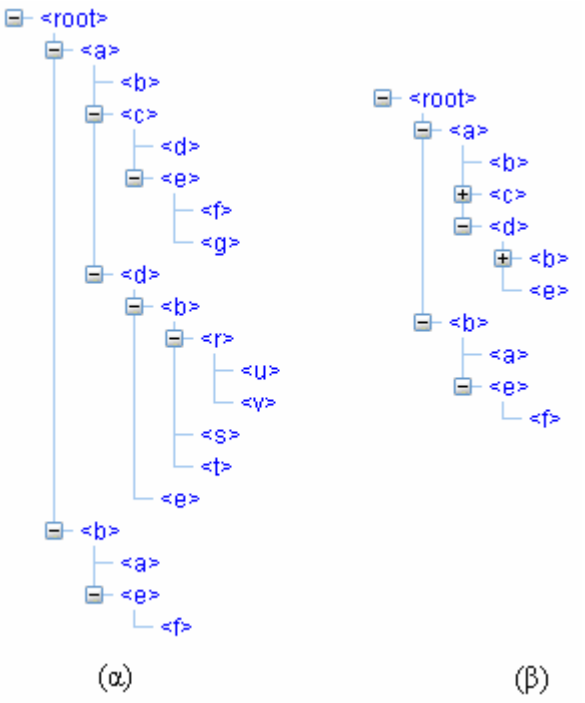
- Αν πρόκειται για άνοιγμα του αρχείου, τότε υπάρχει η επιλογή να το ανοίξουμε στη μορφή ετικετών.
- Αν πρόκειται για μετατροπή από τη μορφή ετικετών στη δενδρική μορφή, τότε απαγορεύουμε να γίνει η μετατροπή.

Οι υπόλοιπες λειτουργίες δεν επηρεάζονται.

Δέντρα σε μορφή ετικετών

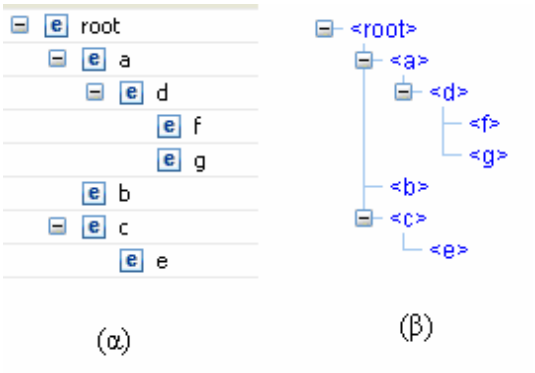
Για τα δέντρα σε μορφή ετικετών έχουμε να παρατηρήσουμε δύο πράγματα :

1. Οι κόμβοι παρουσιάζονται σε εμφωλιασμένη δομή. Ο πατέρας έχει εμφωλιασμένα τα παιδιά του και μπορεί :
   - πατώντας το κουτάκι + που έχει στα αριστερά του να τα εμφανίζει , αν είναι κρυμμένα
   - πατώντας το κουτάκι - που έχει στα αριστερά του να τα κρύβει , αν αυτά εμφανίζονται.



Σχήμα 5.11 : Δέντρο σε μορφή ετικετών (α) Φαίνονται όλα τα στοιχεία (β) Το ίδιο δέντρο με κρυμμένα κάποια στοιχεία, εκεί που υπάρχει το κουτάκι +.

2. Ο λόγος που εμφανίζουμε ένα XML δέντρο στη μορφή ετικετών οφείλεται στο γεγονός ότι προσπαθήσαμε να μοιάζει με τις μορφές που εμφανίζουν ένα δέντρο τα κλασικά προγράμματα, όπως ο Internet Explorer, ο Mozilla Firefox, ο JCreator, η Eclipse κτλ. Τελικά η εμφάνιση σε μορφή ετικετών μοιάζει περισσότερο με την εμφάνιση ενός XML αρχείου στην Eclipse



Σχήμα 5.12 : Δέντρο μορφής ετικετών (α) το XML δέντρο όπως εμφανίζεται με το πρόγραμμα Eclipse, (β) και η αναπαράσταση του δέντρου στην εφαρμογή μας στη μορφή ετικετών

5.3.4 Κόμβοι

5.3.4.1 Κόμβοι Δέντρου

Οι κόμβοι ενός δέντρου πρέπει να έχουν τα εξής χαρακτηριστικά :

- Όλοι οι κόμβοι δέντρου πρέπει να έχουν μοναδικό πατέρα, οπότε μόνο μία εισερχόμενη ακμή, εκτός από τη ρίζα που δεν πρέπει να έχει.
- Όλοι οι κόμβοι μπορούν να έχουν οποιοδήποτε αριθμό παιδιών, οπότε οσεσδήποτε εξερχόμενες ακμές.
- Κάθε ακμή προστίθεται στο γράφο μόνο με τη προσθήκη του νέου κόμβου με τον πατέρα του.

Κόμβος Ρίζα

Σε κάθε δέντρο ορίζουμε υποχρεωτικά έναν κόμβο-ρίζα.

- Ο κόμβος αυτός είναι μοναδικός σε κάθε δέντρο.
- Η ρίζα ενός δέντρου δεν μπορεί να έχει ούτε πατέρα ούτε αδέρφια.
- Η ρίζα ενός δέντρου μπορεί να έχει οποιοδήποτε αριθμό παιδιών.
- Απαγορεύεται η διαγραφή της ρίζας → ρωτάμε το χρήστη αν θέλει να διαγράψει όλο το υποδέντρο της ρίζας. Η ρίζα σε κάθε περίπτωση παραμένει.

Κόμβοι Αποτελέσματος

Στο τέλος κάθε αποτίμησης παίρνουμε τα αποτελέσματα, δηλαδή όλα τα μονοπάτια του δέντρου που ικανοποιούν το ερώτημα. Αν το πλήθος των αποτελεσμάτων είναι μη-μηδενικό τότε ορίζονται οι κόμβοι αποτελέσματος του δέντρου που αντιστοιχούν στον κόμβο εξόδου του ερωτήματος. Σε κάθε μονοπάτι του δέντρου που ικανοποιεί το ερώτημα υπάρχει ένας κόμβος του δέντρου που αντιστοιχεί στον κόμβο εξόδου του ερωτήματος.

- Με το τέλος της αποτίμησης οι κόμβοι αποτελέσματος εμφανίζονται στο δέντρο. Ο χρήστης έχει την δυνατότητα να ΄καθαρίσει' το δέντρο από τα αποτελέσματα, οπότε οι κόμβοι αποτελέσματος ξαναπαίρνουν τη συμπεριφορά των κοινών κόμβων.
- Το πλήθος των κόμβων αποτελέσματος είναι ίσο ή μικρότερο από το πλήθος των αποτελεσμάτων της αποτίμησης.
  1. ίσο → όταν όλα τα αποτελέσματα αντιστοιχούν σε διαφορετικούς κόμβους του δέντρου
  2. μικρότερο → όταν κάποια από τα αποτελέσματα αντιστοιχούν σε ίδιους κόμβους του δέντρου, δηλαδή στην περίπτωση που δύο μονοπάτια αποτελέσματος έχουν ως κόμβο αποτελέσματος τον ίδιο κόμβο του δέντρου

  (βλέπε ανάλυση στην Ενότητα 7.2.6).

5.3.4.2 Κόμβοι Ερωτήματος

Οι κόμβοι ενός ερωτήματος πρέπει να έχουν τα εξής χαρακτηριστικά :

- Ένας κόμβος ερωτήματος μπορεί να έχει ένα μοναδικό πατέρα ή οποιοδήποτε αριθμό προγόνων, εκτός από τη ρίζα που δεν πρέπει να έχει εισερχόμενες ακμές (βλέπε Ενότητα 5.3.5 για περιορισμούς ακμών ερωτήματος).
- Ένας κόμβος ερωτήματος μπορεί να έχει ένα μοναδικό παιδί ή οποιοδήποτε αριθμό απογόνων (βλέπε Ενότητα5.3.5 για περιορισμούς ακμών ερωτήματος).
- Ένας κόμβος έχει τη δυνατότητα να μην έχει ούτε εισερχόμενες ούτε εξερχόμενες ακμές.

Κόμβος Ρίζα

Σε κάθε ερώτημα ορίζουμε υποχρεωτικά έναν κόμβο-ρίζα.

- Ο κόμβος αυτός είναι μοναδικός σε κάθε ερώτημα.
- Η ρίζα ενός ερωτήματος δεν μπορεί να έχει ούτε πατέρα ούτε προγόνους, οπότε καμία εισερχόμενη ακμή. Σε αντίθετη περίπτωση θα είχαμε δημιουργία κύκλου που δεν είναι επιθυμητή.
- Η ρίζα ενός ερωτήματος μπορεί να έχει ένα μοναδικό παιδί ή οποιοδήποτε αριθμό απογόνων (βλέπε Ενότητα5.3.5 για περιορισμούς ακμών ερωτήματος).
- Απαγορεύεται η διαγραφή της ρίζας.

Κόμβος Εξόδου

Σε κάθε ερώτημα ορίζουμε προαιρετικά έναν κόμβο εξόδου.

- Ο κόμβος αυτός είναι μοναδικός σε κάθε ερώτημα
- Ο κόμβος εξόδου συμπεριφέρεται όπως ένας κοινός κόμβος του ερωτήματος, αλλά έχει μία επιπλέον ιδιότητα. Είναι ο κόμβος του ερωτήματος που αντιστοιχεί στους κόμβους του δέντρου που θα είναι τα αποτελέσματα της αποτίμησης.

- Στην περίπτωση που για κόμβο εξόδου ορίσουμε τη ρίζα του ερωτήματος, τότε αν ικανοποιείται το ερώτημα η απάντηση θα είναι όλο το αρχικό δέντρο.

- Αποθήκευση ερωτήματος

Μπορούμε να σώσουμε ένα ερώτημα χωρίς να έχουμε ορίσει κόμβο εξόδου. Αν και υπάρχει αυτή η δυνατότητα, ενημερώνουμε πάντως το χρήστη πριν την αποθήκευση ότι δεν έχει ορίσει κόμβου εξόδου μήπως θέλει να το κάνει εκείνη τη στιγμή.

- Άνοιγμα ερωτήματος

Όπως θα δούμε και στην Ενότητα 6.2.4 μπορούμε να ανοίξουμε ένα αρχείο στο οποίο δεν έχει οριστεί κόμβος εξόδου. Σε αυτήν την περίπτωση, ανοίγουμε κανονικά το αρχείο και απλά δηλώνουμε στο χρήστη ότι δεν έχει οριστεί κόμβος εξόδου για αυτό το ερώτημα.

- Αποτίμηση ερωτήματος σε ένα δέντρο

  Στην περίπτωση όμως της αποτίμησης ενός ερωτήματος, απαγορεύουμε να συνεχιστεί η αποτίμηση αν δεν έχει οριστεί κόμβος εξόδου.

5.3.4.3 Ονόματα Κόμβων

Τα ονόματα των κόμβων προέρχονται είτε από τον χρήστη κατά τη δημιουργία νέων κόμβων είτε διαβάζοντας τα από αρχείο.

- Τα ονόματα των κόμβων δεν μπορεί να είναι *κενά*.

- Τα ονόματα των κόμβων πρέπει να είναι *έγκυρα ονόματα ετικετών XML*. Για παράδειγμα δεν πρέπει να περιέχουν τους χαρακτήρες #, $, % κτλ. Αν δεν είναι έγκυρα ονόματα, δεν γίνεται η προσθήκη του κόμβου.

- Στην περίπτωση που ο πρώτος χαρακτήρας του ονόματος είναι *αριθμός* τότε στην αποθήκευση του αρχείου προσθέτουμε στην αρχή το χαρακτήρα '_' για να είναι έγκυρο όνομα XML ετικέτας, ενώ στο άνοιγμα του αρχείου αφαιρούμε τον χαρακτήρα αυτό. Έτσι στην εμφάνιση και στην περαιτέρω επεξεργασία αυτού του κόμβου το όνομα εμφανίζεται όπως το ορίσαμε αρχικά.

- Στην περίπτωση που έχουμε δέντρο σε μορφή ετικετών, σε κάθε παρουσίαση ενός κόμβου προσθέτουμε στην αρχή και στο τέλος του ονόματος τα σύμβολα *'<' και '>'* αντιστοίχως. Στην αποθήκευση όμως του αρχείου αφαιρούμε τα σύμβολα αυτά ώστε το στοιχείο να πάρει το όνομα που αρχικά δώσαμε.

- Αν ο κόμβος είναι *ρίζα ερωτήματος*, τότε του δίνω ρητά το όνομα 'root'.

- Αν ο κόμβος είναι *ρίζα δέντρου*, και το όνομα που του δίνεται διαφέρει από το 'root', τότε στην εμφάνιση του κόμβου-ρίζα προσθέτω στο τέλος του ονόματος που δόθηκε ' (root)'. Στην αποθήκευση και περαιτέρω επεξεργασία το όνομα θα είναι αυτό που δόθηκε από το χρήστη. Για παράδειγμα αν δοθεί στη ρίζα του δέντρου το όνομα 'myroot' τότε στην εμφάνιση του κόμβου θα παίρνει το όνομα 'myroot (root)'.

  Αυτό γίνεται για να δηλώνουμε στο χρήστη ότι ο κόμβος αυτός είναι η ρίζα του δέντρου και ότι αντιστοιχεί πάντα στη ρίζα του ερωτήματος κατά την αποτίμηση.

- Κατά την αποτίμηση τα ονόματα όλων των κόμβων του ερωτήματος πρέπει να αντιστοιχούν σε ονόματα κόμβων του δέντρου προς αποτίμηση, αλλιώς δεν είναι δυνατή η αποτίμηση του ερωτήματος (βλέπε Ενότητα 7.2.6).

5.3.5 Απαιτήσεις Ερωτήματος

5.3.5.1 Περιορισμοί Κόμβων και Ακμών Ερωτήματος

- Κόμβοι και ακμές μπορούν να προστεθούν ανεξάρτητα.
- Μπορεί να υπάρχουν ξεκρέμαστοι κόμβοι.
- Δεν μπορεί να υπάρχουν ξεκρέμαστες ακμές.
- Η ρίζα δεν μπορεί να έχει πατέρα ή πρόγονο.
- Η ρίζα είναι μοναδική για κάθε ερώτημα
- Ο κόμβος εξόδου, αν έχει οριστεί, είναι μοναδικός για κάθε ερώτημα.

- Επιτρέπεται να υπάρχουν κόμβοι χωρίς εισερχόμενες ακμές, δηλαδή μπορώ να μην προσδιορίσω κανένα πατέρα ή πρόγονο σε ένα κόμβο.
- Επιτρέπεται να υπάρχουν κόμβοι χωρίς εξερχόμενες ακμές, δηλαδή μπορώ να μην προσδιορίσω κανένα παιδί ή απόγονο σε ένα κόμβο.
- Όλοι οι κόμβοι, εκτός της ρίζας, κατά τη διάρκεια της επεξεργασίας ενός ερωτήματος έχουν τη δυνατότητα να μην έχουν εισερχόμενες ακμές. Όμως πριν πάει να αποθηκευτεί ή αποτιμηθεί ένα ερώτημα τότε οι κόμβοι αυτοί πρέπει να συνδεθούν με τη ρίζα ή παιδί της.

- Όλοι οι κόμβοι εκτός της ρίζας επιτρέπεται να έχουν οποιοδήποτε αριθμό προγόνων, αν δεν έχει οριστεί πατέρας.
- Όλοι οι κόμβοι επιτρέπεται να έχουν οποιοδήποτε αριθμό απογόνων, αν δεν έχει οριστεί παιδί.
- Αν σε έναν κόμβο έχω ορίσει τον πατέρα του, τότε ο κόμβος αυτός δεν μπορεί να έχει προγόνους *(βλέπε παρακάτω σημείωση)*.
- Αν σε έναν κόμβο έχω ορίσει το παιδί του, τότε ο κόμβος αυτός δεν μπορεί να έχει απογόνους *(βλέπε παρακάτω σημείωση)*.
- Αν έχει οριστεί πατέρας σε έναν κόμβο, αυτός πρέπει να είναι μοναδικός.
- Αν έχει οριστεί παιδί σε έναν κόμβο, αυτό πρέπει να είναι μοναδικό

- Σε κάθε προσθήκη ακμής πρέπει να έχει προσδιοριστεί ο κόμβος-πηγή και ο κόμβος προορισμού.
- Δεν επιτρέπεται ο κόμβος προορισμού μιας ακμής να είναι η ρίζα του ερωτήματος. Αλλιώς θα είχαμε μη ικανοποιήσιμο ερώτημα, αφού θα είχαμε δημιουργία κύκλου που δεν είναι επιθυμητός.
- Δεν επιτρέπεται ο κόμβος-πηγή και ο κόμβος προορισμού μιας ακμής να ταυτίζονται (self-loop).
- Δεν επιτρέπεται μεταξύ δύο κόμβων να υπάρχουν δύο ακμές με αντίθετη κατεύθυνση.
- Δεν επιτρέπονται κύκλοι.

ΣΗΜΕΙΩΣΗ

Λέγοντας πρόγονο εννοούμε ακμή που δηλώνει σχέση προγόνου-απογόνου με απόγονο τον τρέχοντα κόμβο. Έτσι, λέγοντας ότι δεν μπορεί να έχει προγόνους ένας κόμβος που έχει πατέρα, εννοούμε ότι δεν μπορεί να έχει άλλες εισερχόμενες ακμές παρά μόνο αυτή του πατέρα. Οπότε δεν έχει συνδεδεμένους προγόνους. Πρόγονοί του όμως είναι όλοι οι πρόγονοι του πατέρα του.

Λέγοντας απόγονο εννοούμε ακμή που δηλώνει σχέση προγόνου-απογόνου με πρόγονο τον τρέχοντα κόμβο. Έτσι, λέγοντας ότι δεν μπορεί να έχει απογόνους ένας κόμβος που έχει παιδί, εννοούμε ότι δεν μπορεί να έχει άλλες εξερχόμενες ακμές παρά μόνο αυτή του παιδιού. Οπότε δεν έχει συνδεδεμένους απογόνους. Απόγονοί του όμως είναι όλοι οι απόγονοι του παιδιού του.

Είναι προφανές ότι μια σχέση πατέρα-παιδιού είναι πιο περιοριστική από μια σχέση προγόνου-απογόνου.

Στην ενότητα που ακολουθεί υπάρχει μια εκτενέστερη αναφορά στους περιορισμούς των ακμών ενός ερωτήματος.

5.3.5.2 Απαγορεύσεις και Μετατροπές Ακμών Ερωτήματος

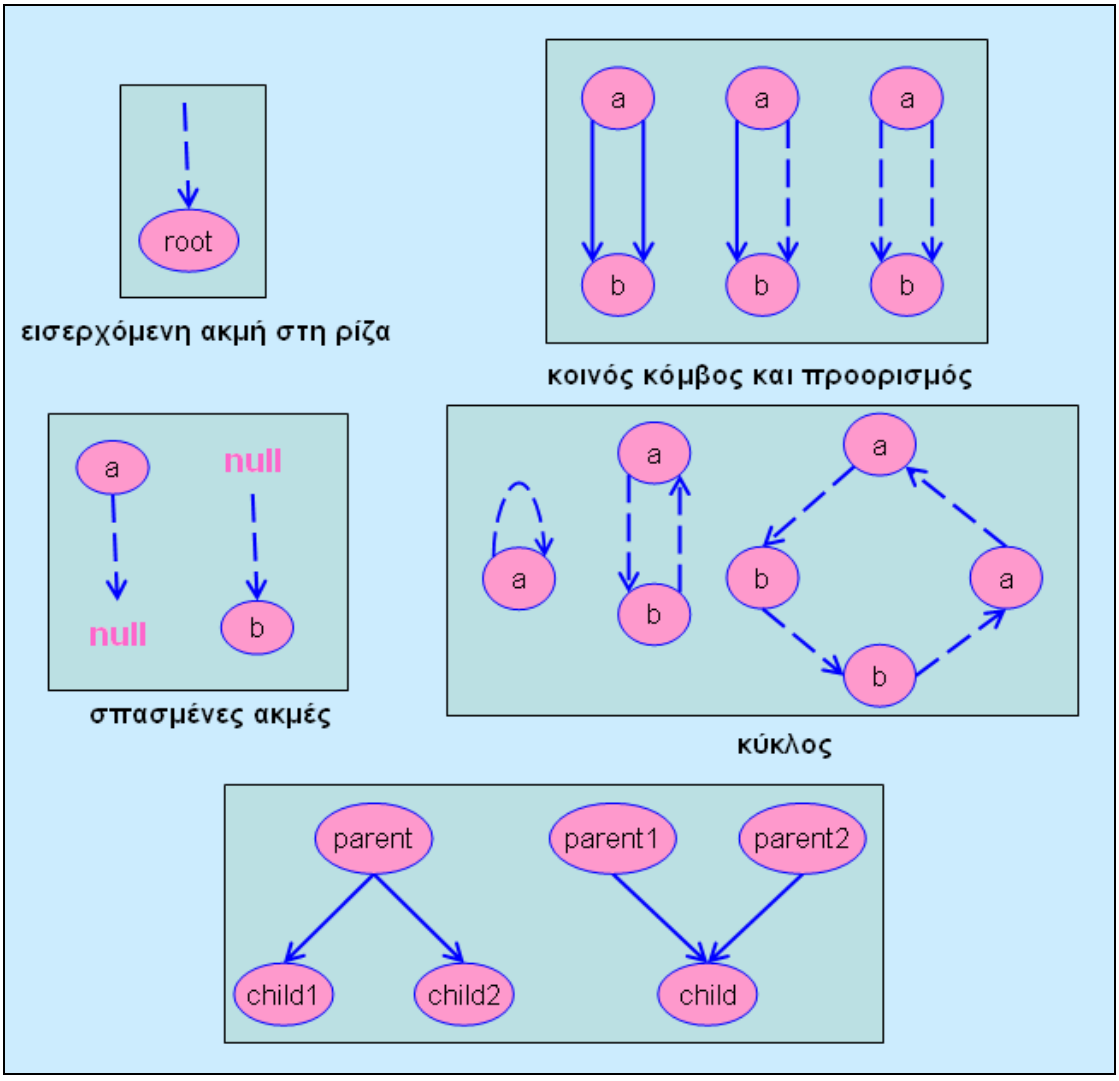
Σε κάθε ενέργεια αλλαγών που κάνει ο χρήστης πάνω στο ερώτημα ελέγχουμε σε πραγματικό χρόνο (real time) αν προδιαγράφονται οι παραπάνω απαιτήσεις. Αν όχι, τότε αναλόγως την περίπτωση, απαγορεύουμε την ενέργεια ή τη μετατρέπουμε σε έγκυρη ενέργεια, χωρίς να αλλάζει η σημασιολογία του ερωτήματος.

Απαγορεύσεις

Ο χρήστης μπορεί να ζητήσει να γίνουν ενέργειες πάνω στο ερώτημα που δεν επιτρέπονται και έτσι εμείς τις απαγορεύουμε. Τέτοιες ενέργειες είναι :

1. Δεν επιτρέπεται ο κόμβος προορισμού μιας ακμής να είναι η ρίζα του ερωτήματος.
2. Δεν επιτρέπεται να μην έχει οριστεί ο κόμβος-πηγή ή/και ο κόμβος προορισμού μιας ακμής → ξεκρέμαστες ακμές
3. Δεν επιτρέπεται ο κόμβος-πηγή και ο κόμβος προορισμού μιας ακμής να ταυτίζονται (self-loop).
4. Δεν επιτρέπεται δύο ακμές να έχουν κοινό κόμβο-πηγή και κοινό κόμβο προορισμού.
5. Δεν επιτρέπεται μεταξύ δύο κόμβων να υπάρχουν δύο ακμές με αντίθετη κατεύθυνση
6. Δεν επιτρέπονται κύκλοι.
7. Δεν επιτρέπεται ένας κόμβος να έχει δύο πατεράδες ή δύο παιδιά, αφού πρόκειται για ερωτήματα μονοπατιού. Σε αυτήν την περίπτωση ρωτάμε το χρήστη ποια από τις δύο σχέσεις θέλει να κρατήσει, την ήδη υπάρχουσα ή τη νέα.

Στο επόμενο σχήμα συγκεντρώνονται όλες αυτές οι μη-επιτρεπτές ακμές.



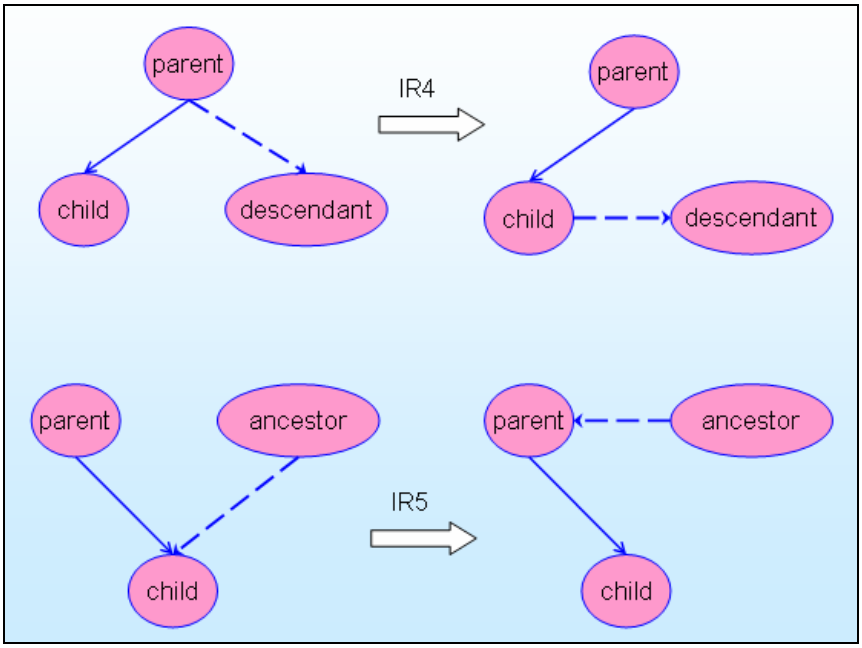
Σχήμα 5.13 : Μη-επιτρεπτές ακμές ερωτήματος.

Μετατροπές

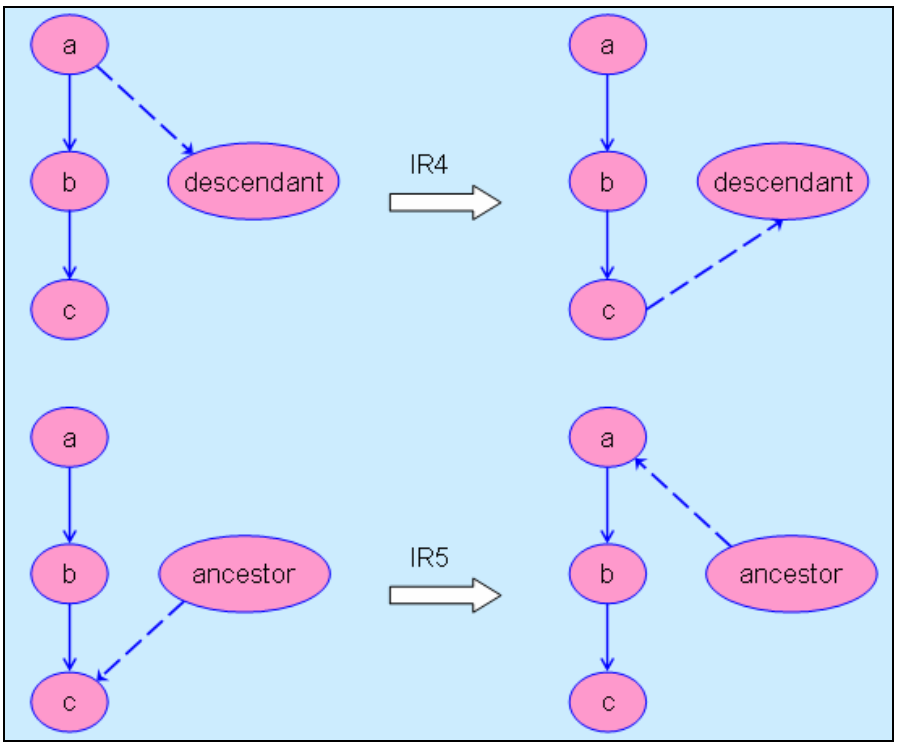
Ο χρήστης μπορεί να ζητήσει να γίνουν ενέργειες πάνω στο ερώτημα που δεν πληρούν τις προδιαγραφές, αλλά εμείς να μπορούμε να τις μετατρέψουμε σε πραγματικό χρόνο σε έγκυρες χωρίς να αλλοιώνεται το αίτημα του χρήστη. Τέτοιες ενέργειες είναι :

1. Αν σε έναν κόμβο έχω ορίσει το παιδί του, τότε οι απόγονοι του κόμβου *(βλέπε παραπάνω σημείωση)* που ήδη υπάρχουν ή που ο χρήστης δημιουργήσει στο μέλλον παίρνουν ως πρόγονο το παιδί του κόμβου, σύμφωνα με τον ***κανόνα IR4*** (βλέπε Ενότητα 3.5). Η διαδικασία μπορεί να είναι επαναληπτική.
2. Αν σε έναν κόμβο έχω ορίσει τον πατέρα του, τότε οι πρόγονοι του κόμβου *(βλέπε παραπάνω σημείωση)* που ήδη υπάρχουν ή που ο χρήστης δημιουργήσει στο μέλλον παίρνουν ως απόγονο τον πατέρα του κόμβου, σύμφωνα με τον ***κανόνα IR5*** (βλέπε Ενότητα 3.5). Η διαδικασία μπορεί να είναι επαναληπτική.
3. Αν και υπάρχει η δυνατότητα οι κόμβοι να μην έχουν εισερχόμενες ακμές, πριν την αποθήκευση ή την αποτίμηση του ερωτήματος ή μετά από απαίτηση του χρήστη ορίζουμε οι κόμβοι αυτοί να έχουν ως πρόγονο τη ρίζα ή παιδί της, σύμφωνα με τον ***κανόνα IR1*** (βλέπε Ενότητα 3.5).

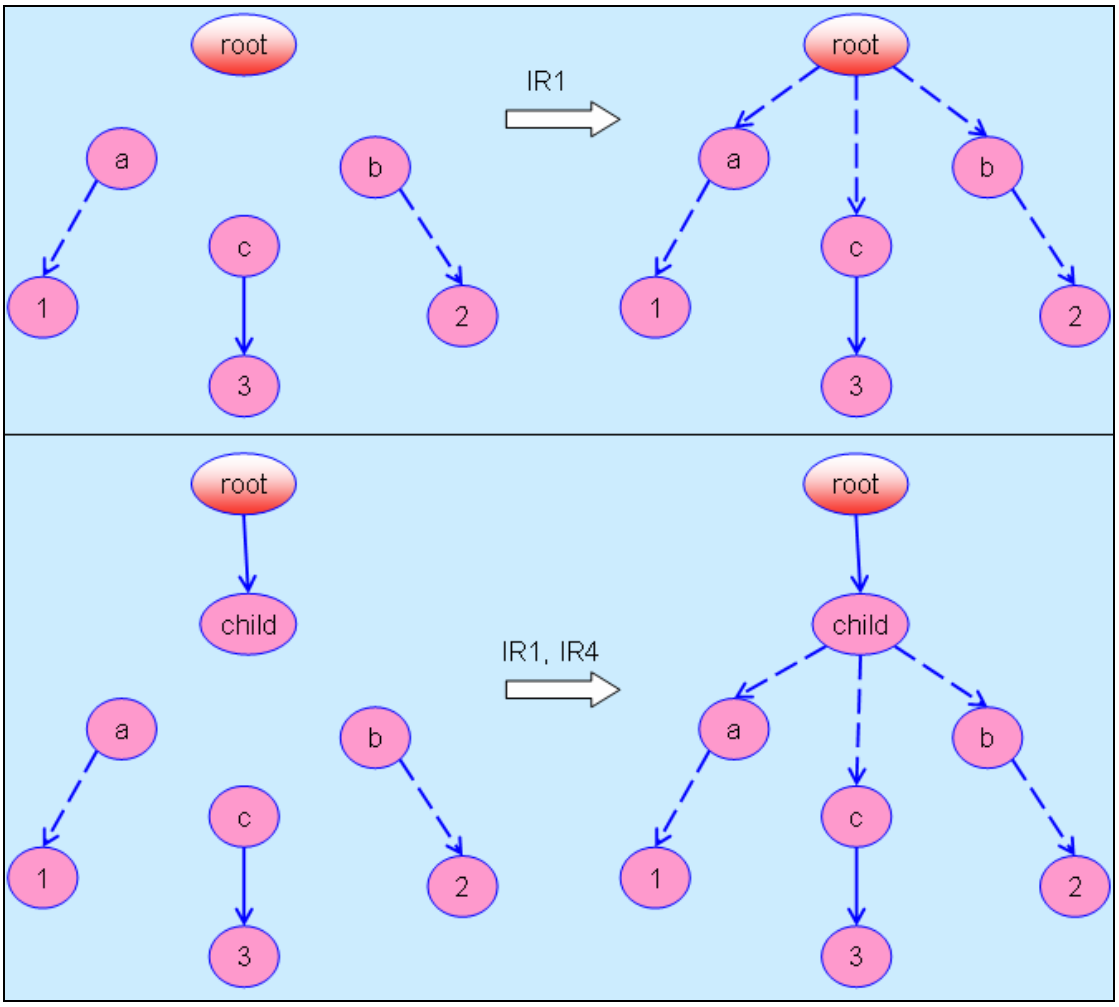
Στα σχήματα που ακολουθούν απεικονίζονται όλες αυτές οι μετατροπές.



Σχήμα 5.14 : Μετατροπές ακμών ερωτήματος : εφαρμογή κανόνων IR4 και IR5.



Σχήμα 5.15 : Μετατροπές ακμών ερωτήματος : εφαρμογή κανόνων IR4 και IR5 επαναληπτικά



Σχήμα 5.16 : Μετατροπές ακμών ερωτήματος : εφαρμογή κανόνα IR1. Οι κόμβοι που δεν έχουν εισερχόμενες ακμές συνδέονται με τη ρίζα ή παιδί της.

5.3.6 Αποτίμηση Ερωτήματος

Ο χρήστης θα επιλέγει ένα δέντρο, ένα ερώτημα και τον αλγόριθμο με τον οποίο θέλει να γίνει η αποτίμηση. Όλοι οι κόμβοι του ερωτήματος πρέπει να αντιστοιχούν σε κόμβους του δέντρου και ένας από αυτούς να έχει οριστεί ως κόμβος εξόδου. Αν επιλεχθεί ο αλγόριθμος PathStack πρέπει το ερώτημα μονοπατιού να είναι πλήρως ορισμένο.

- Αν δεν υπάρχουν αποτελέσματα, ενημερώνουμε με κατάλληλο μήνυμα τον χρήστη και δεν κάνουμε τίποτα άλλο. Οι λόγοι μηδενικών αποτελεσμάτων μπορεί να είναι ένας από τους εξής :
  1. Να μην έχει οριστεί στο ερώτημα ο κόμβος εξόδου
  2. Να μην είναι σωστά ορισμένο ένα ερώτημα, για παράδειγμα να περιέχει κύκλους.
  3. Να έχει επιλεγεί ο αλγόριθμος PathStack αλλά να μην πρόκειται για πλήρως ορισμένο ερώτημα μονοπατιού.
  4. Να υπάρχει έστω και ένας κόμβος ερωτήματος που να μην υπάρχει ως κόμβος δέντρου.
  5. Να γίνεται η αποτίμηση, αλλά να δίνει μηδενικά αποτελέσματα.

- Αν υπάρχουν αποτελέσματα, τότε δημιουργείται ένα <u>νέο δέντρο-αποτέλεσμα</u> ίδιας μορφής με το δέντρο που ερωτήθηκε, που κατά τα άλλα θα συμπεριφέρεται ως ένα κοινό δέντρο. Το δέντρο αποτελέσματος θα έχει μια ρίζα η οποία θα έχει ως παιδιά τους κόμβους αποτελέσματος, στους οποίους θα συνδέεται όλο το υποδέντρο που έχουν στο αρχικό δέντρο. Αν ένας κόμβος αποτελέσματος εμφανιστεί περισσότερες από μία φορές στην αποτίμηση, τότε στα αποτελέσματα τον βάζουμε μόνο μία φορά.

  Αν πρόκειται για δέντρο δενδρικής μορφής, οι κόμβοι αποτελέσματος θα σημειώνονται και στο αρχικό δέντρο. Μετά θα υπάρχει δυνατότητα καθαρισμού των αποτελεσμάτων και το αρχικό δέντρο θα επανέρχεται στην αρχική του κατάσταση.

<u>Αρχεία</u>

Για την αποτίμηση χρειαζόμαστε δύο αρχεία : την κωδικοποίηση θέσης του XML δέντρου και το ερώτημα στη μορφή που περιγράφεται στην Ενότητα 6.2.4. Αυτά τα δύο αρχεία αποθηκεύονται προσωρινά σε συγκεκριμένο σημείο του δίσκου και διαγράφονται με το τέλος της εφαρμογής.

6

Σχεδίαση Συστήματος

Στο κεφάλαιο αυτό, περιγράφεται η σχεδίαση του συστήματος, οι κλάσεις και η κωδικοποίηση των αρχείων που χρησιμοποιούνται .

6.1 Σχεδίαση Συστήματος

Στην ενότητα αυτή παρουσιάζεται η σχεδίαση κάθε υποσυστήματος σε αντιστοιχία με την αρχιτεκτονική που παρουσιάστηκε στο προηγούμενο κεφάλαιο.

6.1.1 Σχεδίαση τμήματος "Διαπροσωπεία Χρήστη"

Τα τμήματα "Διαχείριση XML Δέντρων μορφής ετικετών", "Διαχείριση XML Δέντρων δενδρικής μορφής", "Διαχείριση Ερωτημάτων" και "Αποτίμηση" είναι πολύ σημαντικά λόγω του ότι υλοποιούν το λογικό τμήμα της εφαρμογής. Εξίσου σημαντικό όμως είναι και το τμήμα "Διαπροσωπεία Χρήστη" που αποτελεί το μεσάζοντα μεταξύ του χρήστη και του λογικού τμήματος του προγράμματος. Η "Διαπροσωπεία Χρήστη" μεταφέρει τις εντολές του χρήστη στο κατάλληλο υποσύστημα του οποίου η απόκριση παρουσιάζεται με εποπτικό τρόπο πάλι στη "Διαπροσωπεία Χρήστη"

Η "Διαπροσωπεία Χρήστη" αφορά τα ορατά συστατικά του κυρίως παραθύρου της εφαρμογής: την επιφάνεια σχεδιασμού, τα διάφορα μενού που διαθέτει και το πλαίσιο με τα κουμπιά.

Οι λειτουργίες πάνω στο γράφο γίνονται μόνο από το λογικό τμήμα, ενώ η "Διαπροσωπεία Χρήστη" περιορίζεται μόνο στην απεικόνισή τους πάνω στην επιφάνεια σχεδιασμού. Ακόμα δίνεται η δυνατότητα στο χρήστη να αλλάξει τις θέσεις των κόμβων, ώστε να επιτυγχάνεται καλύτερο οπτικό αποτέλεσμα. Επίσης , η τρέχουσα εικόνα του γράφου στην επιφάνεια σχεδίασης μπορεί να αποθηκευτεί σε κατάλληλα αρχεία, και να ανακτηθεί όποτε είναι αυτό επιθυμητό.

6.1.1.1 Κλάσεις του τμήματος "Διαπροσωπεία Χρήστη"

Η ***Διαπροσωπεία Χρήστη*** υλοποιείται με τη βοήθεια των κλάσεων : Trees_and_Queries, NewTab, NewTab.DataTab, ButtonTabComponent, ImageExport.

- Trees_and_Queries → σχετίζεται με τη διαπροσωπεία χρήστη. Με βάση τις επιλογές του χρήστη, επικοινωνεί με τα υπόλοιπα υποσυστήματα.
- NewTab → δημιουργεί νέες καρτέλες για την απεικόνιση των XML δέντρων σε μορφή ετικετών και σε δενδρική μορφή και για την απεικόνιση των ερωτημάτων.
- NewTab.DataTab → αποθηκεύει τα δεδομένα κάθε καρτέλας
- ButtonTabComponent → επεξεργάζεται τη μορφή των καρτελών και τους δίνει τη δυνατότητα κλεισίματος.
- ImageExport → εξάγει σε εικόνα το γράφο δέντρου δενδρικής μορφής και το γράφο ερωτήματος

6.1.1.2 Σχέσεις μεταξύ των κλάσεων

Η βασική κλάση Trees_and_Queries διαχειρίζεται όλες τις λειτουργίες και επικοινωνεί κάθε φορά με το κατάλληλο υποσύστημα. Όταν ζητηθεί η δημιουργία ή άνοιγμα δέντρου ή ερωτήματος αναλαμβάνει η κλάση NewTab να δημιουργήσει μια νέα καρτέλα με μια επιφάνεια σχεδίασης όπου θα απεικονίζονται όλες οι αλλαγές που κάνουμε στο κάθε δέντρο ή ερώτημα, ενώ η εσωτερική της κλάση NewTab.DataTab αποθηκεύει τα δεδομένα της κάθε καρτέλας για να είναι δυνατός ο διαχωρισμός της από τις υπόλοιπες καρτέλες. Η κλάση ButtonTabComponent επεξεργάζεται τη μορφή των καρτελών και τους δίνει τη δυνατότητα κλεισίματος. Τέλος η ImageExport αναλαμβάνει την εξαγωγή εικόνας του γράφου ενός δέντρου δενδρικής μορφής ή ενός ερωτήματος

6.1.2 Σχεδίαση τμήματος "Διαχείριση XML Δέντρων σε μορφή ετικετών"

6.1.2.1 Κλάσεις του τμήματος "Διαχείριση XML Δέντρων σε μορφή ετικετών"

Η ***Διαχείριση XML Δέντρων σε μορφή ετικετών*** γίνεται μέσω των ακόλουθων κλάσεων : NewTree_JTree, NewTree_JTree.DataTree, VerticesTree_JTree, OpenTree_JTree, SaveTree_JTree, SaveRegionEncoding_JTree.

- NewTree_JTree → δημιουργία XML δέντρου σε μορφή ετικετών
- NewTree_JTree.DataTree → αποθήκευση δεδομένων δέντρου σε μορφή ετικετών
- VerticesTree_JTree → επεξεργασία XML δέντρου μορφής ετικετών
- OpenTree_JTree → άνοιγμα XML αρχείου και αναπαράστασή του σε XML δέντρο μορφής ετικετών
- SaveTree_JTree → αποθήκευση XML δέντρου μορφής ετικετών σε XML αρχείο
- SaveRegionEncoding_JTree → αποθήκευση κωδικοποίησης θέσης του XML δέντρου μορφής ετικετών σε txt αρχείο

6.1.2.2 Σχέσεις μεταξύ των κλάσεων

Η βασική κλάση είναι η VerticesTree_JTree που διαχειρίζεται όλα τα δεδομένα του δέντρου μορφής ετικετών.
Η κλάση NewTree_JTree είναι αυτή που αρχικοποιεί το δέντρο μορφής ετικετών ενώ η εσωτερική της κλάση NewTree_JTree.DataTree αποθηκεύει τα δεδομένα του δέντρου. Η κλάση OpenTree_JTree αναλαμβάνει τη φόρτωση του δέντρου μορφής ετικετών από XML αρχείο. Η κλάση SaveTree_JTree αναλαμβάνει την αποθήκευση του δέντρου σε ένα XML αρχείο. Η κλάση SaveRegionEncoding_JTree αναλαμβάνει την αποθήκευση της κωδικοποίησης θέσης του δέντρου σε ένα txt αρχείο.

6.1.3 Σχεδίαση τμήματος "Διαχείριση XML Δέντρων σε δενδρική μορφή"

6.1.3.1 Κλάσεις του τμήματος "Διαχείριση XML Δέντρων σε δενδρική μορφή"

Η ***Διαχείριση XML Δέντρων σε δενδρική μορφή*** γίνεται μέσω των ακόλουθων κλάσεων : NewTree_JGraph, NewTree_ JGraph.DataTree, VerticesTree_JGraph, VerticesTree_JGraph.DataVertex, Edges, OpenTree_JGraph, SaveTree_JGraph, SaveRegionEncoding_JGraph, SaveTreeData.

- NewTree_ JGraph → δημιουργία XML δέντρου σε δενδρική μορφή
- NewTree_ JGraph.DataTree → αποθήκευση δεδομένων δέντρου σε δενδρική μορφή
- VerticesTree_ JGraph → επεξεργασία XML δέντρου δενδρικής μορφής
- VerticesTree_JGraph.DataVertex → αποθήκευση δεδομένων ενός κόμβου του δέντρου
- Edges → επεξεργασία ακμών δέντρου δενδρικής μορφής

- OpenTree_ JGraph → άνοιγμα XML αρχείου και αναπαράστασή του σε XML δέντρο δενδρικής μορφής
- SaveTree_ JGraph → αποθήκευση XML δέντρου δενδρικής μορφής σε XML αρχείο
- SaveRegionEncoding_ JGraph → αποθήκευση κωδικοποίησης θέσης του XML δέντρου δενδρικής μορφής σε txt αρχείο
- SaveTreeData → αποθήκευση XML δέντρου δενδρικής μορφής στη μορφή treeData σε txt αρχείο

6.1.3.2 Σχέσεις μεταξύ των κλάσεων

Η βασική κλάση είναι η VerticesTree_JGraph που διαχειρίζεται όλα τα δεδομένα του δέντρου δενδρικής μορφής. Η εσωτερική της κλάση VerticesTree_JGraph.DataVertex αποθηκεύει τα δεδομένα κάθε κόμβου του δέντρου. Υπάρχει και η κλάση Edges που διαχειρίζεται τις ακμές του δέντρου.
Η κλάση NewTree_JGraph είναι αυτή που αρχικοποιεί το δέντρο δενδρικής μορφής ενώ η εσωτερική της κλάση NewTree_JGraph.DataTree αποθηκεύει τα δεδομένα του δέντρου. Η κλάση OpenTree_JGraph αναλαμβάνει τη φόρτωση του δέντρου δενδρικής μορφής από XML αρχείο. Η κλάση SaveTree_JGraph αναλαμβάνει την αποθήκευση του δέντρου σε ένα XML αρχείο. Η κλάση SaveRegionEncoding_JGraph αναλαμβάνει την αποθήκευση της κωδικοποίησης θέσης του δέντρου σε ένα txt αρχείο. Η κλάση SaveTreeData αναλαμβάνει την αποθήκευση της μορφής treeData του δέντρου σε ένα txt αρχείο.

6.1.4 Σχεδίαση τμήματος "Διαχείριση Ερωτημάτων"

6.1.4.1 Κλάσεις του τμήματος "Διαχείριση Ερωτημάτων"

Η ***Διαχείριση Ερωτημάτων*** γίνεται μέσω των ακόλουθων κλάσεων : NewQuery, NewQuery.DataQuery, VerticesQuery, VerticesQuery.DataVertex, Edges_child_descendant, Edges_child_descendant.DataEdge, Cells, GPCellViewFactory, JGraphEllipseView, OpenQuery, SaveQuery.

- NewQuery → δημιουργία ερωτήματος
- NewQuery.DataQuery → αποθήκευση δεδομένων ερωτήματος
- VerticesQuery → επεξεργασία κόμβων ερωτήματος
- VerticesQuery.DataVertex → αποθήκευση δεδομένων ενός κόμβου του ερωτήματος
- Edges_child_descendant → επεξεργασία ακμών ερωτήματος
- Edges_child_descendant.DataEdge → αποθήκευση δεδομένων μιας ακμής του ερωτήματος
- Cells → επεξεργασία κόμβων και ακμών ερωτήματος
- GPCellViewFactory και JGraphEllipseView → για το κυκλικό σχήμα των κόμβων ερωτήματος
- OpenQuery → άνοιγμα txt αρχείου με προσυμφωνημένη μορφή και αναπαράστασή του σε ερωτήμα
- SaveQuery → αποθήκευση ερωτήματος σε προσυμφωνημένη μορφή.

6.1.4.2 Σχέσεις μεταξύ των κλάσεων

Οι βασικές κλάσεις είναι η VerticesQuery και Edges_child_descendant που διαχειρίζονται όλους τους κόμβους και τις ακμές του ερωτήματος αντιστοίχως. Οι εσωτερικές τους κλάσεις VerticesQuery.DataVertex και Edges_child_descendant.DataEdge αποθηκεύουν τα δεδομένα κάθε κόμβου και κάθε σχέσης του ερωτήματος αντιστοίχως. Υπάρχει και η κλάση Cells που διαχειρίζεται τους κόμβους και τις ακμές του ερωτήματος ως σύνολο. Οι κλάσεις GPCellViewFactory και JGraphEllipseView προσδίδουν το κυκλικό σχήμα των κόμβων του ερωτήματος.
Η κλάση NewQuery είναι αυτή που αρχικοποιεί το ερώτημα ενώ η εσωτερική της κλάση NewQuery.DataQuery αποθηκεύει τα δεδομένα του ερωτήματος. Η κλάση OpenQuery αναλαμβάνει τη φόρτωση του ερωτήματος από ένα txt αρχείο. Η κλάση SaveQuery αναλαμβάνει την αποθήκευση του ερωτήματος σε ένα txt αρχείο.

6.1.5 Σχεδίαση τμήματος "Αποτιμητής"

6.1.5.1 Κλάση του τμήματος "Αποτιμητής"

Η ***Αποτίμηση Ερωτημάτων σε Δενδρικά Δεδομένα*** γίνεται με την κλάση Evaluation.

- Evaluation → αποτίμηση ερωτήματος σε XML δέντρο

6.1.6 Σχεδίαση Βοηθητικών Κλάσεων

6.1.6.1 Βοηθητικές Κλάσεις - Singleton pattern

Υπάρχουν ακόμα οι κλάσεις με singleton pattern που καλούνται από τα διάφορα υποσυστήματα: NameTranslation, Selection, Save. Πρόκειται για βοηθητικές κλάσεις που δεν ανήκουν σε κάποιο μεμονωμένο υποσύστημα, αλλά καλούνται από περισσότερα του ενός υποσυστήματα.

- NameTranslation → μετατρέπει τα ονόματα των κόμβων σε κατάλληλη μορφή αναλόγως την περίπτωση : δημιουργία, άνοιγμα, αποθήκευση, επεξεργασία, αποτίμηση. Ζητάει από το χρήστη να δώσει ένα νέο όνομα κόμβου και ελέγχει αν πρόκειται για έγκυρο XML όνομα.
- Selection → αφορά τις επιλογές που γίνονται στους γράφους (δέντρο δενδρικής μορφής ή ερώτημα). Μετράει πόσα αντικείμενα είναι επιλεγμένα στο γράφο και διαχωρίζει αν είναι κόμβοι ή/και ακμές.
- Save → αφορά τις αλλαγές που κάνουμε σε ένα δέντρο ή ερώτημα και αν τις αποθηκεύουμε, και την παραγωγή κωδικοποίησης θέσης ενός δέντρου. Ελέγχει αν οι αλλαγές που έχουμε κάνει σε ένα δέντρο ή ερώτημα είναι αποθηκευμένες ή όχι. Ακόμα ελέγχει αν είναι περιττό να παράγουμε την κωδικοποίηση θέσης ενός δέντρου στην περίπτωση που υπάρχει ήδη και δεν έχουν γίνει αλλαγές στο δέντρο.

6.1.6.2 Σχέσεις μεταξύ των κλάσεων

Η κλάση NameTranslation καλείται σε κάθε προσθήκη κόμβου (Edit) για να ελέγξει αν είναι έγκυρο XML όνομα. Ακόμα καλείται σε κάθε δημιουργία δέντρου ή ερωτήματος (New) για την προσθήκη της ρίζας. Επίσης καλείται σε κάθε άνοιγμα δέντρου (Open XML tree), σε κάθε αποθήκευση δέντρου (Save XML tree) και σε κάθε αποθήκευση κωδικοποίηση θέσης δέντρου (Save Region Encoding) για να ελέγξει αν ένα όνομα κόμβου αρχίζει με ακέραιο. Τέλος καλείται όταν θέλουμε να πάρουμε μόνο το όνομα του κόμβου ενός δέντρου μορφής ετικετών χωρίς τους χαρακτήρες ‘<’ και ’>’ που συμβαίνει στην αποθήκευση του δέντρου μορφής ετικετών (Save tree – xml tags) και στην αποτίμηση όταν το δέντρο είναι σε μορφή ετικετών (Evaluation).

ή προβολή στοιχείων (View) ενός κόμβου ή μιας ακμής σε ένα γράφο. Αφορά μόνο δέντρα δενδρικής μορφής και ερωτήματα.

Η κλάση Selection καλείται σε κάθε επεξεργασία (Edit) ή προβολή στοιχείων (View) ενός κόμβου ή μιας ακμής σε ένα γράφο. Αφορά μόνο δέντρα δενδρικής μορφής και ερωτήματα.

Η κλάση Save καλείται σε κάθε επεξεργασία (Edit) ενός γράφου και σε κάθε αποθήκευσή του (Save). Στην $1^{η}$ περίπτωση δηλώνει ότι τα δεδομένα δεν είναι αποθηκευμένα ενώ στη $2^{η}$ ότι είναι. Ακόμα καλείται σε κάθε αίτηση παραγωγής κωδικοποίησης θέσης ενός δέντρου ώστε να ελέγξει αν υπάρχει η έγκυρη κωδικοποίηση θέσης του δέντρου και αν ναι να μην την παράξει από την αρχή, αλλά να την αντιγράψει.

6.2 Κωδικοποίηση Αρχείων

Στην ενότητα αυτή παρουσιάζονται τα αρχεία που χρησιμοποιεί το σύστημά μας.

Τα XML δέντρα κωδικοποιούνται σε τρεις μορφές, οπότε έχουμε τρία είδη αρχείων.

1. XML έγγραφο
2. κωδικοποίηση θέσης
3. treeData

Τα ερωτήματα.κωδικοποιούνται σε μία μορφή.

6.2.1 XML Αρχείο Δέντρου

Στο αρχείο αυτό αποθηκεύεται το δέντρο μας στη γνωστή XML μορφή. Για τη διάσχιση στο δέντρο χρησιμοποιούμε το JDOM ενώ για την εισαγωγή και την εξαγωγή XML αρχείων χρησιμοποιούμε τον SAXParser. Υπάρχει αναλυτικότερη αναφορά για το θέμα αυτό στην Ενότητα 2.2.

6.2.2 Αρχείο Κωδικοποίησης Θέσης ενός XML Δέντρου

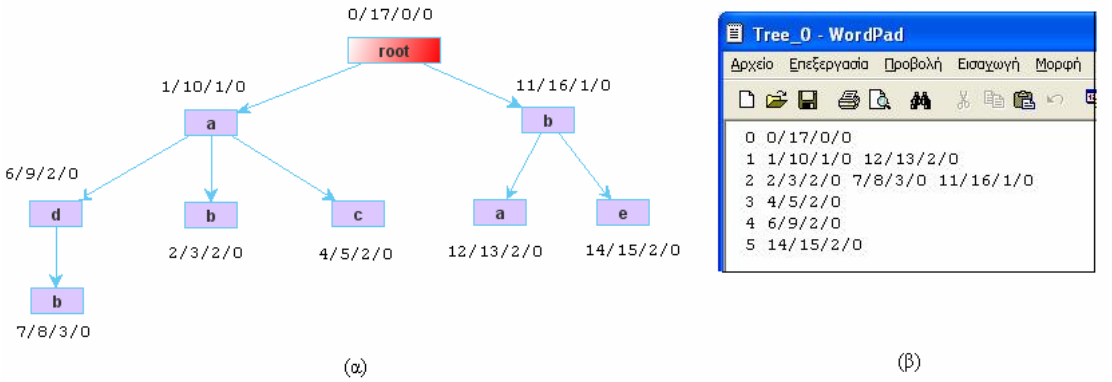
Το αρχείο αυτό αποτελείται από τόσες γραμμές όσους διαφορετικούς κόμβους έχει το δέντρο. Λέγοντας διαφορετικούς, εννοούμε με διαφορετικό όνομα (βλέπε Ενότητα 7.2.5).

Κάθε γραμμή έχει μπροστά έναν κωδικό του στοιχείου, ακολουθεί ένα κενός χαρακτήρας και μετά ένα σύνολο από κωδικοποιήσεις θέσης.

Η κάθε κωδικοποίηση θέσης εμφανίζεται στη μορφή start/end/level/0. Βάζουμε πάντα 0 στο τέλος γιατί έχουμε μόνο ένα XML έγγραφο (βλέπε Ενότητα 3.2).

Αν ένα στοιχείο εμφανίζεται στο δέντρο περισσότερες από μία φορές, τότε έχει πολλές κωδικοποιήσεις θέσης που διαχωρίζονται με ένα κενό χαρακτήρα.

Στο Σχήμα 6.1 φαίνεται ένα παράδειγμα αποθήκευσης σε αρχείο της κωδικοποίησης θέσης ενός XML δέντρου.



Σχήμα 6.1 : Αποθήκευση κωδικοποίησης θέσης δέντρου σε txt αρχείο. (α) Το δέντρο σε δενδρική μορφή. Δίπλα από κάθε κόμβο υπάρχει η κωδικοποίηση θέσης του. (β) Κωδικοποίηση θέσης του σε αρχείο. Παρατηρούμε ότι ο κόμβος a εμφανίζεται στο δέντρο 2 φορές και ο b 3, γι'αυτό στο αρχείο υπάρχουν στον κόμβο a με κωδικό 1 και στον κόμβο b με κωδικό 2 αντίστοιχα 2 και 3 κωδικοποιήσεις θέσης. Οι υπόλοιποι κόμβοι (root, c, d, e με κωδικούς 0, 3, 4, 5 αντίστοιχα) εμφανίζονται από μία φορά στο δέντρο και έχουν μία κωδικοποίηση θέσης.

6.2.3 Αρχείο Δέντρου σε μορφή treeData

Το αρχείο αυτό αποτελείται από τόσες γραμμές όσους κόμβους έχει το δέντρο (γραμμές ακμών) και μία ακόμα (γραμμή κόμβων).

Στην $1^{η}$ γραμμή καταχωρούμε όλους τους κόμβους στη μορφή : κωδικός~όνομα και τους χωρίζουμε με ένα κενό χαρακτήρα. Γι'αυτό η $1^{η}$ γραμμή λέγεται και γραμμή κόμβων.

Έτσι έχουμε μια ακολουθία :

id_1~$name_1$ id_2~$name_2$ id_3~$name_3$ id_4~$name_4$ κοκ.

Μετά ακολουθούν τόσες γραμμές όσο είναι το πλήθος των κόμβων, που αποτελούν τις γραμμές ακμών. Κάθε γραμμή ακμών αντιστοιχεί στις ακμές ενός κόμβου.

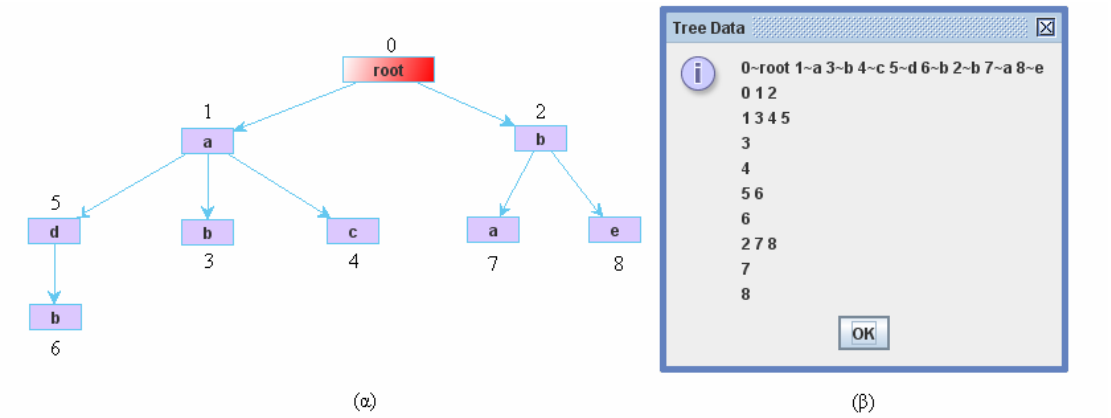
Η καθεμία τέτοια γραμμή αποθηκεύει τις ακμές του κάθε κόμβου, βάζοντας στην αρχή τον κωδικό του κόμβου και μετά όλους τους κωδικούς των παιδιών του, τους οποίους χωρίζουμε με ένα κενό χαρακτήρα.

Έτσι έχουμε μια ακολουθία :

$id_{πατέρα}$ $id_{παιδιού\ i}$ $id_{παιδιού\ j}$ κοκ

Αν δεν υπάρχουν παιδιά του κόμβου τότε η γραμμή θα περιέχει μόνο τον κωδικό του κόμβου.

Στο Σχήμα 6.2 φαίνεται ένα παράδειγμα αποθήκευσης σε αρχείο της μορφής treeData ενός δέντρου δενδρικής μορφής.



Σχήμα 6.2 : Αποθήκευση μορφής treeData του δέντρου σε txt αρχείο. (α) Το δέντρο σε δενδρική μορφή. Δίπλα από κάθε κόμβο υπάρχει ο κωδικός του. (β) Μορφή treeData του δέντρου σε αρχείο. Παρατηρούμε ότι ο κόμβος α με κωδικό 1 έχει 3 παιδιά, οπότε στη μορφή treeData δίπλα από τον κωδικό 1 υπάρχουν 3 κωδικοί που αντιστοιχούν στους κόμβους-παιδιά του α, ενώ ο κόμβος α με κωδικό 7 δεν έχει κανένα παιδί, οπότε στη μορφή treeData δεν ακολουθεί τον κωδικό 7 κάποιος άλλος κωδικός.

6.2.2 Αρχείο Ερωτήματος

Ένα ερώτημα σώζεται στην εξής μορφή :

Αποτελείται από τρεις βασικές γραμμές

Η $1^{η}$ γραμμή αποτελείται από όλους τους κόμβους του ερωτήματος, ενώ οι άλλες δύο περιλαμβάνουν όλες τις σχέσεις μεταξύ των κόμβων και πιο συγκεκριμένα η $2^{η}$ γραμμή περιέχει όλες τις σχέσεις πατέρα-παιδιού και η $3^{η}$ όλες τις σχέσεις προγόνου-απόγονου.

$1^{η}$ γραμμή → κόμβοι

Κάθε κόμβος του ερωτήματος καταχωρείται σε αυτήν τη γραμμή στη μορφή id~name. Οι κόμβοι διαχωρίζονται μεταξύ τους με ένα κενό.

Έτσι έχουμε μια ακολουθία :

id_1~$name_1$ id_2~$name_2$ id_3~$name_3$ id_4~$name_4$ κοκ.

$2^η$ γραμμή → σχέσεις πατέρα-παιδιού

Όπου συναντήσουμε σχέση πατέρα-παιδιού στο ερώτημα την καταχωρούμε σε αυτήν τη γραμμή στη μορφή $id_{πατέρα}$-$id_{παιδιού}$. Οι σχέσεις διαχωρίζονται μεταξύ τους με ένα κενό.

Έτσι έχουμε μια ακολουθία :

$id_{πατέρα\ n}$ -$id_{παιδιού\ m}$ $id_{πατέρα\ i}$ -$id_{παιδιού\ j}$ κοκ.

Αν δεν έχουμε σχέσεις πατέρα-παιδιού αφήνουμε κενή τη γραμμή.

$3^η$ γραμμή → σχέσεις προγόνου-απόγονου

Όπου συναντήσουμε σχέση προγόνου-απόγονου στο ερώτημα την καταχωρούμε σε αυτήν τη γραμμή στη μορφή $id_{προγόνου}$=$id_{απόγονου}$. Οι σχέσεις διαχωρίζονται μεταξύ τους με ένα κενό.

Έτσι έχουμε μια ακολουθία :

$id_{προγόνου\ n}$ =$id_{απογόνου\ m}$ $id_{προγόνου\ i}$ =$id_{απογόνου\ j}$ κοκ.

Αν δεν έχουμε σχέσεις προγόνου-απόγονου αφήνουμε κενή τη γραμμή

Σημείωση

Στην επεξεργασία του ερωτήματος χρησιμοποιούμε μόνο τους κωδικούς id των κόμβων, που είναι μοναδικοί και όχι τα ονόματά τους που μπορεί να είναι ίδια σε πολλούς κόμβους.

Η αρχική γραμμή είναι μία από τις :

```
ptpq.0.0.0.0.0.0.txt 1
ptpq.0.0.0.0.0.0.txt 1     //The output node has id : 2
```

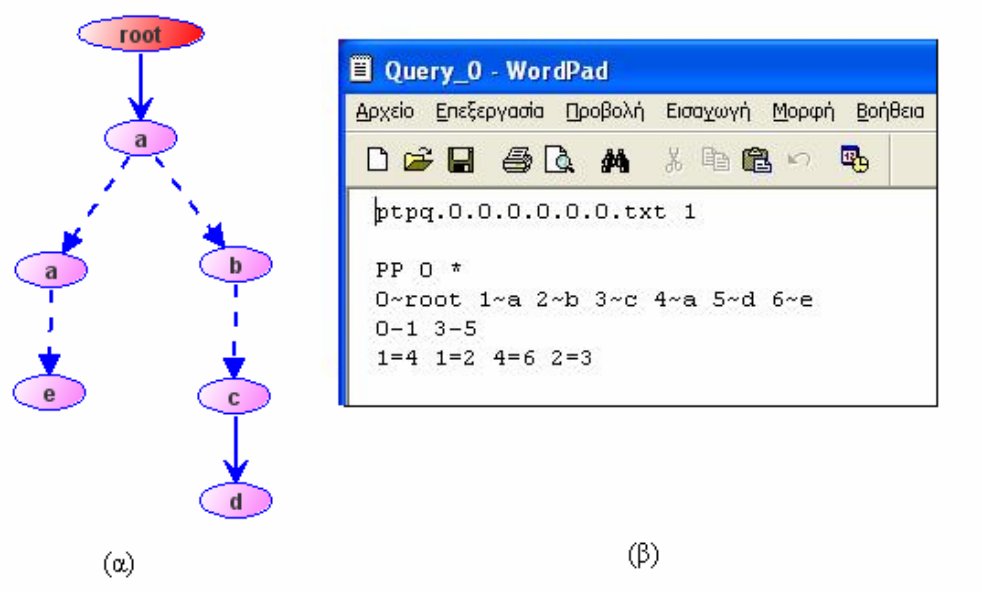
Στη δεύτερη περίπτωση ορίζουμε τον κόμβο εξόδου, όπου ο αριθμός στο τέλος δηλώνει το id του (εδώ id του κόμβου εξόδου είναι το 2).

Ακολουθεί μια κενή γραμμή και μετά η γραμμή :

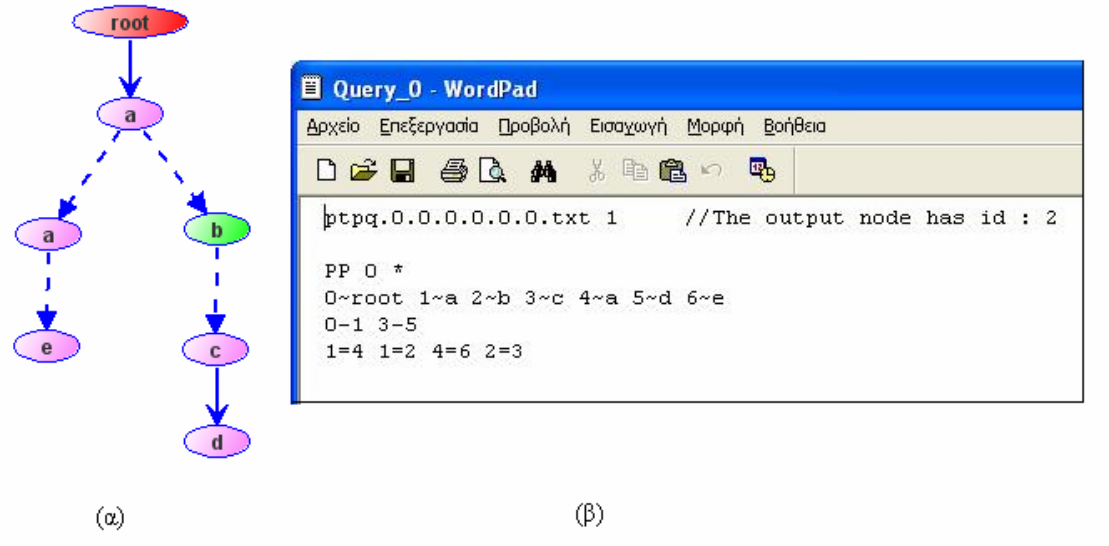
```
PP 0 *
```

που είναι κοινή για όλα τα ερωτήματά μας αφού περιέχουν μόνο ένα ανεξάρτητο ερώτημα κάθε φορά.

Στα Σχήματα που ακολουθούν παρουσιάζεται ένα παράδειγμα.



Σχήμα 6.3 : Αποθήκευση ερωτήματος χωρίς ορισμένο κόμβο εξόδου σε txt αρχείο (α) το ερώτημα χωρίς ορισμένο κόμβο εξόδου και (β) αποθήκευσή του σε αρχείο. Παρατηρούμε ότι ο κόμβος α εμφανίζεται 2 φορές στο ερώτημα και στο αρχείο αποθηκεύεται με διαφορετικό κωδικό για να ξεχωρίζει.



Σχήμα 6.4 : Αποθήκευση ερωτήματος με ορισμένο κόμβο εξόδου σε txt αρχείο (α) το ερώτημα με ορισμένο κόμβο εξόδου τον κόμβο b και (β) αποθήκευσή του σε αρχείο. Παρατηρούμε ότι το αρχείο είναι ίδιο με το αρχείο του Σχήματος 6.3 που δεν έχει οριστεί ο κόμβος εξόδου εκτός από την πρώτη γραμμή που ορίζεται ο κόμβος εξόδου με κωδικό 2 που αντιστοιχεί στον κόμβο b.

7

Υλοποίηση

Στο κεφάλαιο αυτό περιγράφεται η διαδικασία υλοποίησης του συστήματος επεξεργασίας ερωτημάτων σε δενδρικά δεδομένα ***Partialist***.

Αρχικά αναφέρουμε τους λόγους που μας οδήγησαν στην επιλογή της συγκεκριμένης γλώσσας προγραμματισμού και του αντίστοιχου εργαλείου. Έπειτα παρουσιάζουμε τον τρόπο υλοποίησης της εφαρμογής,. Τέλος παραθέτουμε μερικά θέματα υλοποίησης που θεωρήσαμε ότι έχουν ξεχωριστό ενδιαφέρον ή μια αυξημένη πολυπλοκότητα. Πρόκειται για ζητήματα που έχουν ιδιαίτερη βαρύτητα στην εκπόνηση της διπλωματικής και που άλλοτε μας προβλημάτισαν και άλλοτε μας δυσκόλεψαν.

7.1 Πλατφόρμες και Προγραμματιστικά Εργαλεία

7.1.1 Γλώσσα Προγραμματισμού

Java

Ως γλώσσα προγραμματισμού για την ανάπτυξη της εφαρμογής μας επιλέχθηκε η Java. Πρόκειται για μια σύγχρονη, αντικειμενοστραφή γλώσσα. Τα πλεονεκτήματα που προσφέρει

η Java σε σύγκριση με άλλες γλώσσες προγραμματισμού, μάς οδήγησαν στην επιλογή της για την υλοποίηση του ***Partialist***.

Αρχικά, η Java σου δίνει τη δυνατότητα να τρέξεις την εφαρμογή σου οπουδήποτε, αρκεί να υποστηρίζεται εκεί η πλατφόρμα της Java, κάτι που συμβαίνει σε όλα τα κύρια λειτουργικά συστήματα σήμερα. Αυτό εξηγεί και το σύνθημα της Sun για τη Java : "*Write once, run anywhere*". Επίσης η Java είναι δυναμική και επεκτάσιμη. Σαν μια καθαρά αντικειμενοστραφής γλώσσα, οργανώνει τον κώδικα σε αυτόνομες μονάδες που λέγονται *classes*. Αυτές μπορούν να χρησιμοποιηθούν από την εφαρμογή καθώς τρέχει και να φορτωθούν όποτε αυτή τις χρειαστεί. Σημαντικό είναι ακόμη το πόσο απλή και κομψή είναι σαν γλώσσα, με ένα καλοσχεδιασμένο API, και έτσι δίνει τη δυνατότητα παραγωγής κώδικα με λιγότερα bugs μειώνοντας το χρόνο ανάπτυξης μιας εφαρμογής. Επιπλέον η Java είναι εύχρηστη και κατάλληλη για γραφικά, γεγονός που έπαιξε σημαντικό ρόλο στην επιλογή της, αφού έχουμε υλοποίηση interface. Τέλος σε συνδυασμό με τη μεταφερσιμότητα, προσφέρει και ικανοποιητικές επιδόσεις, κυρίως με τις τελευταίες εκδόσεις της, κάτι που λίγες γλώσσες μπορούν να προσφέρουν.

C++

Ένα μικρό κομμάτι της εφαρμογής υλοποιήθηκε στη γλώσσα προγραμματισμού C++.

Η C++ παράγει κώδικα που τρέχει γρήγορα. Εξάλλου η ταχύτητα είναι ένα χαρακτηριστικό που κληρονομεί από τη C. Ακόμα προσφέρει στοιχεία αντικειμενοστρεφούς προγραμματισμού, όπως οι κλάσεις ή η κληρονομικότητα, οι οποίες μπορούν να διευκολύνουν σημαντικά την οργάνωση και ανάπτυξη του συστήματος.

Βασικός όμως λόγος της επιλογής αυτής είναι η επαναχρησιμοποίηση κώδικα. Η αποτίμηση ερωτημάτων σε δενδρικά δεδομένα υπήρχε υλοποιημένη από το εργαστήριο Συστημάτων Βάσεων Γνώσεων και Δεδομένων στη γλώσσα C++ που είναι πιο κατάλληλη σε αυτήν την περίπτωση από τη Java, αφού είναι πιο αποδοτική στην επίλυση αλγορίθμων. Τα αποτελέσματα της αποτίμησης τα επεξεργαστήκαμε επίσης σε C++ ώστε να στείλουμε στον κώδικα της Java μόνο τα απαραίτητα δεδομένα για την αναπαράσταση των αποτελεσμάτων (βλέπε Ενότητα 7.2.6).

Ένωση Java με C++

Ενώσαμε τον κώδικα της βασικής εφαρμογής υλοποιημένο σε Java με τον κώδικα της αποτίμησης υλοποιημένο σε C++ καλώντας ένα εξωτερικό εκτελέσιμο (βλέπε λεπτομέρειες στην Ενότητα 7.2.6).

7.1.2 Προγραμματιστικά Εργαλεία

Java

Στην υλοποίηση της εφαρμογής μας χρησιμοποιήσαμε Java έκδοση jdk 1.6.0_02 , jre 1.6.0_02 που είναι διαθέσιμο στο site της sun στη διεύθυνση *http://java.sun.com/*.

Το περιβάλλον ανάπτυξης (IDE) που χρησιμοποιήσαμε είναι το Eclipse (έκδοση 3.3).

Για την εκτέλεση της εφαρμογής χρειάστηκαν να προστεθούν οι εξής βιβλιοθήκες :

- jgraph.jar → απαραίτητο για την απεικόνιση των γράφων δέντρων δενδρικής μορφής και των γράφων ερωτημάτων.
- jdom.jar → απαραίτητο για το άνοιγμα, τη διαχείριση και την αποθήκευση XML αρχείων.

Για την αναπαράσταση των δέντρων και των ερωτημάτων χρησιμοποιήσαμε δύο βασικές βιβλιοθήκες :

- JGraph → για τα δέντρα δενδρικής μορφής και για τα ερωτήματα
- JTree → για τα δέντρα μορφής ετικετών.

C++

Ο μεταγλωττιστής που χρησιμοποιήθηκε για τη δημιουργία του εκτελέσιμου κώδικα από τα C++ πηγαία αρχεία ήταν ο g++ ή αλλιώς GNU C++, ο οποίος είναι μέρος της σουίτας μεταγλωττιστών GNU. Δεδομένου ότι η σουίτα μεταγλωττιστών GNU αφορά συστήματα τύπου Unix, ενώ το διαθέσιμο λειτουργικό σύστημα ήταν το Microsoft Windows XP, τρέξαμε τον εν λόγω μεταγλωττιστή μέσα στο λογισμικό Cygwin, το οποίο επιτρέπει σε λειτουργικά τύπου Microsoft Windows να συμπεριφέρονται με χαρακτηριστικά συστημάτων τύπου Unix.

Χαρακτηριστικά υπολογιστή

Η ανάπτυξη της εφαρμογής έγινε σε προσωπικό υπολογιστή με επεξεργαστή Centrino στα 2 GHz και μνήμη RAM 1 GB. Η ταχύτητα εκτέλεσης του προγράμματος κρίθηκε ικανοποιητική.

7.2 Λεπτομέρειες Υλοποίησης

Στην επόμενη παράγραφο θα αναφερθούμε συνοπτικά στο πως είναι δομημένη καθεμιά από τις κύριες κλάσεις της εφαρμογής, ποιες δομές δεδομένων χρησιμοποιεί, τι λειτουργίες επιτελεί η κάθε ρουτίνα, δίνοντας πάντα έμφαση στα κύρια και σημαντικά σημεία και προσπερνώντας τα τετριμμένα . Μετά ακολουθούν κάποια βασικά στοιχεία της υλοποίησης που βοηθούν την διαχείριση των διάφορων δεδομένων. Στη συνέχεια ακολουθούν μερικά θέματα υλοποίησης που θεωρήσαμε ότι έχουν ξεχωριστό ενδιαφέρον ή μια αυξημένη πολυπλοκότητα.

7.2.1 Περιγραφή των Κλάσεων

Πριν ξεκινήσουμε την περιγραφή των κλάσεων που αφορούν και υλοποιούν τις βασικές λειτουργίες και έννοιες της εφαρμογής, πρέπει να αναφέρουμε ότι αναλυτική παράθεση της διαπροσωπείας κάθε κλάσης δίνεται στο παράρτημα στο τέλος του τόμου. Εκεί βρίσκεται και σύντομη επεξήγηση της λειτουργίας που επιτελεί κάθε ρουτίνα σε προγραμματιστικό επίπεδο και μπορεί να χρησιμεύσει σε οποιονδήποτε επιθυμήσει να επεκτείνει ή να βελτιώσει την εφαρμογή μας. Πρόκειται για μια μορφή documentation, που όμως μπορεί να είναι κουραστικό για τον αναγνώστη της παρούσας διπλωματικής που θέλει να πάρει μια ιδέα μόνο της διαδικασίας της υλοποίησης. Αυτή την ανάγκη για μια συντομότερη περιγραφή των κυριότερων κλάσεων έρχεται να καλύψει η παράγραφος αυτή.

Σε πλήρη αντιστοιχία με το περίγραμμα της εφαρμογής που δόθηκε στο κεφάλαιο *Σχεδίαση* θα παρουσιάσουμε εδώ τη γενική μορφή της υλοποίησης..

7.2.1.1 Διαπροσωπεία Χρήστη

Trees_and_Queries

Είναι η βασική κλάση. Αρχικοποιεί όλα τα πεδία της εφαρμογής που διαχειρίζονται γενικά δεδομένα. Δημιουργεί το παράθυρο/διαπροσωπεία της εφαρμογής με όλες τις λειτουργίες που προδιαγράφονται.

Βασικά πεδία κλάσης

Τα πεδία JFrame general_frame και JTabbedPane tabbed_pane που είναι το βασικό παράθυρο της εφαρμογής και το σύνολο των καρτελών της εφαρμογής αντιστοίχως.

Τα πεδία int num_tabs, int num_trees, int num_queries που είναι το πλήθος των καρτελών της εφαρμογής, το πλήθος των δέντρων της εφαρμογής, το πλήθος των ερωτημάτων της εφαρμογής αντιστοίχως και το πεδίο Vector<File> streamFiles που είναι τα αρχεία που είναι ανοιχτά στην εφαρμογή.

Τα πεδία Hashtable<String, NewTab.DataTab> componentName2dataTab, Hashtable<NewTab.DataTab, NewTree_JTree.DataTree> dataTab2dataTree_JTree, Hashtable<NewTab.DataTab,NewTree_JGraph.DataTree> dataTab2dataTree_JGraph και Hashtable<NewTab.DataTab, NewQuery.DataQuery> dataTab2dataQuery που είναι οι δομές που σχετίζονται με τις καρτέλες και με το περιεχόμενό τους.

Επίσης, τα πεδία που αφορούν το interface, όπως τα μενού, τα κουμπιά, οι καρτέλες κτλ.

```
enum Application{Tree_jtree, Tree_jgraph, Query}
enum Create{New, Open}
```

Βασικές μέθοδοι κλάσης

- public static void main(String[] args)
  εκκίνηση προγράμματος

Δημιουργία toolbar και κουμπιών

- createToolBar() , void
- addButtons(JToolBar toolbar) , void
- addButtons_convertTrees(JToolar toolbar) , void

Δημιουργία των μενού και ενεργοποίησή τους

- JMenuBar_panel() , JMenuBar

- JMenuItems_File() , void
- JMenuItems_Conversion() , void
- JMenuItems_Evaluation() , void
- JMenuItems_JTree() , void
- JMenuItems_JGraph() , void
- JMenuItems_Query() , void
- enableMenus(Application application) , void
- menuEdit(Application application) , void
- menuView(Application application) , void
- enableMenuTree_JTree(boolean is_enabled) , void
- enableMenuTree_JGraph(boolean is_enabled) , void
- enableMenuQuery(boolean is_enabled) , void
- selectApplication(Create create) , void

- actionPerformed(ActionEvent evt) , void (implements ActionListener)
  εκτέλεση κατάλληλης λειτουργίας όταν ο χρήστης το απαιτήσει

- stateChanged(ChangeEvent evt) , void (implements ChangeListener)
  αλλαγή του μενού επιλογών όταν αλλάζει η καρτέλα στο προσκήνιο.

Δημιουργία - Άνοιγμα

- newTree_JTree() , void
  δημιουργία δέντρου μορφής ετικετών

- openTree_JTree() , NewTab
  άνοιγμα δέντρου μορφής ετικετών

- newTree_JGraph() , void
  δημιουργία δέντρου δενδρικής μορφής

- openTree_JGraph() , NewTab
  άνοιγμα δέντρου δενδρικής μορφής

- newQuery () , void
  δημιουργία ερωτήματος

- openQuery () , NewTab
  άνοιγμα ερωτήματος

Αποθήκευση

- save(NewTab.DataTab dataSaveTab) , void
- saveAs(NewTab.DataTab dataSaveTab) , void
- saveRegionEncoding(NewTab.DataTab dataSaveTab) , void
  αποθήκευση κωδικοποίησης θέσης δέντρου

Μετατροπή μορφής δέντρου

- JGraph2JTree() , boolean
  μετατροπή δέντρου από δενδρική μορφή σε μορφή ετικετών

- JTree2JGraph() , boolean
  μετατροπή δέντρου από μορφή ετικετών σε δενδρική μορφή

- exit() , void
  κλείσιμο εφαρμογής
- closeTab(int i) , void
  κλείσιμο καρτέλας
- removeHashtableData(Component component, NewTab.DataTab dataCloseTab) , void
  διαγραφή δεδομένων καρτέλας.
- printHashtable(Hashtable<?,?> hashtable, String hashtable_name) , static String
  εμφάνιση δεδομένων δομής.

NewTab

Δημιουργεί μία νέα καρτέλα στην εφαρμογή μας που περιέχει είτε ένα δέντρο οποιαδήποτε μορφής είτε ένα ερώτημα ανάλογα με το πεδίο application.

inner class **DataTab**

Περιέχει τα δεδομένα μιας καρτέλας.

Πεδία

- int id : κωδικός καρτέλας
- Application application : είδος δέντρου ή ερωτήματος που περιέχει η καρτέλα
- File file : αρχείο που είναι αποθηκευμένα τα περιεχόμενα της καρτέλας
- boolean isSaved : ελέγχει αν είναι σωσμένα τα δεδομένα του δέντρου ή του ερωτήματος που περιέχει η καρτέλα.

Βασικά πεδία κλάσης

Το πεδίο Application application δηλώνει το είδος που περιέχει η καρτέλα (δέντρο μορφής ετικετών, δέντρο δενδρικής μορφής ή ερώτημα) και το πεδίο DataTab dataTab που περιέχει τα δεδομένα της καρτέλας.

Βασικές μέθοδοι κλάσης

- initDataTab() , DataTab
  αρχικοποίηση των δεδομένων της νέας καρτέλας.
- CreateTab(String tabName, JTree tree) , DataTab
  δημιουργία νέας καρτέλας για δέντρο μορφής ετικετών
- CreateTab(String tabName, JGraph graph, boolean isTree) , DataTab
  δημιουργία νέας καρτέλας για δέντρο δενδρικής μορφής ή για ερώτημα

- storeTab(String tabName, Component component_tab) , void
  αποθήκευση δεδομένων της καρτέλας.
- PanelJGraph(JGraph graph) , JPanel - static
  δημιουργία του panel του γράφου ενός δέντρου δενδρικής μορφής ή ενός ερωτήματος
- PanelJTree(JTree tree) , JPanel - static
  δημιουργία του panel του δέντρου μορφής ετικετών
- getDataTab() , DataTab
  παίρνει τα δεδομένα της καρτέλας
- printTab() , void
  τύπωση στοιχείων της καρτέλας

ButtonTabComponent

Επεξεργάζεται τη μορφή των καρτελών και τους δίνει τη δυνατότητα κλεισίματος

ImageExport

Αποθηκεύει σε αρχείο την εικόνα ενός γράφου. Αφορά μόνο δέντρα δενδρικής μορφής και ερωτήματα.

Βασικό πεδίο κλάσης είναι το JGraph graph που είναι ο γράφος που θα εξαχθεί σε εικόνα

Βασικές μέθοδοι κλάσης

Εξαγωγή εικόνας σε αρχείο

- saveFile() , void
- imageExport(File file) , void

7.2.1.2 Διαχειριστής XML Δέντρων μορφής ετικετών

NewTree_JTree

Δημιουργεί ένα νέο δέντρο σε μορφή ετικετών.

Σε αυτήν την κλάση έχουμε 2 κατασκευαστές. Ο ένας καλείται με τη δημιουργία ενός δέντρου και ο άλλος με το άνοιγμα ενός δέντρου.

inner class **DataTree**

Κρατάει τα δεδομένα ενός δέντρου μορφής ετικετών

Πεδία

- int id : κωδικός δέντρου μορφής ετικετών
- JTree tree : δέντρο μορφής ετικετών
- DefaultTreeModel treeModel : μοντέλο δέντρου μορφής ετικετών
- VerticesTree_JTree ourVertices : κόμβοι δέντρου μορφής ετικετών
- DefaultMutableTreeNode rootNode : κόμβος-ρίζα δέντρου μορφής ετικετών

Βασικά πεδία κλάσης

Το πεδίο JTree tree που είναι το δέντρο μορφής ετικετών, το πεδίο DefaultTreeModel treeModel που είναι το μοντέλο του δέντρου, το πεδίο VerticesTree_JTree ourVertices που είναι οι κόμβοι του δέντρου, το πεδίο DefaultMutableTreeNode rootNode που είναι η ρίζα του δέντρου και το πεδίο DataTree dataTree που περιέχει τα δεδομένα του δέντρου.

Βασικές μέθοδοι κλάσης

- putRoot() , void
  προσθήκη της ρίζας του δέντρου
- initTree_JTree(String rootName) , void
  αρχικοποίηση δέντρου
- initDataTree() , DataTree
  αρχικοποίηση δεδομένων δέντρου
- getDataTree() , DataTree
  παίρνει τα δεδομένα του δέντρου

OpenTree_JTree

Ανοίγει ένα υπάρχον XML αρχείο σε δέντρο σε μορφή ετικέτων

Βασικά πεδία κλάσης

Το πεδίο File openFile που είναι το αρχείο που ανοίγω και το πεδίο NewTree_JTree.DataTree dataTree που περιέχει τα δεδομένα του δέντρου

Βασικές μέθοδοι κλάσης

- openGraph() , String
  επιλογή XML αρχείου
- generateXMLTree(File file) , void
  μετατροπή XML αρχείου σε δέντρο σε μορφή ετικετών.

- getChildren(Element el, DefaultMutableTreeNode parentNode) , void
  διάσχιση σε όλα τα στοιχεία του XML αρχείου
- getDataTree() , NewTree_JTree.DataTree
  παίρνει τα δεδομένα του δέντρου
- getOpenFile() , File
  παίρνει το αρχείο που άνοιξε

SaveXML_JTree

Αποθηκεύει ένα δέντρο μορφής ετικετών σε ένα XML αρχείο.

Σε αυτήν την κλάση έχουμε 2 κατασκευαστές. Ο ένας καλείται μόνο στην περίπτωση save as όταν υπάρχει ήδη το αρχείο και δεν έχουν γίνει αλλαγές, οπότε δεν διασχίζω το δέντρο για να εξάγω το XML αρχείο, αλλά το αντιγράφω από το υπάρχον.

Βασικά πεδία κλάσης

Το πεδίο int num_elements που είναι το πλήθος των στοιχείων του XML αρχείου, το πεδίο Hashtable<Integer, DefaultMutableTreeNode> idElement2TreeNode που είναι η δομή στην οποία αποθηκεύουμε για κάθε στοιχείο το ζευγάρι {κωδικός στοιχείου, κόμβος}, το πεδίο DefaultMutableTreeNode rootNode που είναι η ρίζα του δέντρου, το πεδίο boolean doStoreTree που δηλώνει αν θα κάνω αποθήκευση ή αντιγραφή αρχείου και τα πεδία File alreadySavedFile, saveFile_XMLTree που είναι το αποθηκευμένο αρχείο του δέντρου πριν την τωρινή αποθήκευση και το τωρινά αποθηκευμένο αρχείο αντίστοιχα.

Βασικές μέθοδοι κλάσης

- saveFile() , boolean
  επιλογή XML αρχείου όπου θα αποθηκευτεί το δέντρο μορφής ετικετών
- storeTree(File file) , boolean
  αποθήκευση του δέντρου μορφής ετικετών σε ένα XML αρχείο.
- showChildren_JDOM(Document doc, DefaultMutableTreeNode parentNode, Element parentElement) , void
  διάσχιση σε όλους τους κόμβους του δέντρου και δημιουργία αντίστοιχων XML στοιχείων
- getSaveFile() , File
  παίρνει το XML αρχείο όπου αποθηκεύτηκε το δέντρο
- printFileData(File file) , void
  τύπωμα δεδομένων αρχείου

SaveRegionEncoding_JTree

Δημιουργεί την κωδικοποίηση θέσης του δέντρου σε μορφή ετικετών και την αποθηκεύει σε ένα txt αρχείο.

Έχει 2 κατασκευαστές.

inner class *NodeEncoding*

Η κωδικοποίηση θέσης ενός στοιχείου

Πεδία

- int start : η πρώτη επίσκεψη του κόμβου
- int end : η τελευταία επίσκεψη του κόμβου
- int level : το επίπεδο του κόμβου

inner class *Stream*

Πεδία

- Vector<NodeEncoding> stream : το διάνυσμα που αποθηκεύω την κωδικοποίηση θέσης όλων των κόμβων του δέντρου
- public Stack<Integer> st : η στοίβα που κρατάει τη θέση στη λίστα των κόμβων που δεν έχουν τελειώσει ακόμα

Βασικά πεδία κλάσης

Τα πεδία int num_tags, int tagCounter , int regCounter και int currentDepth που είναι οι 4 μετρητές που περιγράφονται στην Ενότητα 7.2.5.

Τα πεδία Hashtable<Integer, Stream> streams, Hashtable<String, Integer> tag2id, Hashtable<Integer, String> id2tag, Hashtable<Integer, Integer> start2id, Hashtable<Integer, Integer> startNodeEncoding2idElement και Hashtable<Integer, DefaultMutableTreeNode> idElement2TreeNode που χρησιμεύουν για την κωδικοποίηση θέσης του δέντρου.

Τα πεδία File saveFile_XMLTree, saveFile_XMLTree_RE που είναι το XML αρχείο που αποθηκεύω το δέντρο και το txt αρχείο που αποθηκεύω την κωδικοποίηση θέσης του δέντρου αντίστοιχα

Βασικές μέθοδοι κλάσης

Παραγωγή των κωδικοποιήσεων θέσης των κόμβων

- generateRegionEncoding() , void

διάβασμα του XML αρχείου για την παραγωγή της κωδικοποίησης θέσης του δέντρου.

- processElement(Element el) , void
  διάσχιση όλων των στοιχείων του XML αρχείου.
- startElement(Element el) , void
  καταχώρηση των τιμών start και level στην κωδικοποίηση θέσης του στοιχείου.
- endElement(Element el) , void
  καταχώρηση της τιμής end στην κωδικοποίηση θέσης του στοιχείου.
- printstreams(Hashtable<Integer, Stream> streams) , void
  συγκέντρωση όλων των κωδικοποιήσεων θέσης των κόμβων του δέντρου.
- printStream(int tagid) , String
  τύπωμα της κωδικοποίηση θέσης ενός κόμβου στη μορφή start/end/level/0.

Αρχείο

- saveFileRE(String dataStreams) , void
  επιλογή αρχείου όπου θα αποθηκευτεί η κωδικοποίηση θέσης του δέντρου
- saveFileRE_withoutJFileChooser(String dataStreams) , void
  ορισμός αρχείου όπου θα αποθηκευτεί η κωδικοποίηση θέσης του δέντρου
- storeFileRE(File f, String dataStreams) , void
  αποθήκευση των κωδικοποιήσεων θέσης των κόμβων σε αρχείο
- getSaveFile_RE() , File
  παίρνει το txt αρχείο όπου αποθηκεύτηκε η κωδικοποίηση θέσης του δέντρου

Επιστρέφουν βασικές δομές της κλάσης

- getHash_tag2id() , Hashtable<String, Integer>
- getHash_startNodeEncoding2idElement() , Hashtable<Integer, Integer>
- getHash_idElement2TreeNode() , Hashtable<Integer, DefaultMutableTreeNode>

VerticesTree_JTree

Χειρίζεται όλα τα δεδομένα ενός δέντρου σε μορφή ετικετών

Βασικά πεδία κλάσης

Το πεδίο JTree tree που είναι το δέντρο μορφής ετικετών, το πεδίο DefaultTreeModel treeModel που είναι το μοντέλο του δέντρου, το πεδίο DefaultMutableTreeNode rootNode που είναι η ρίζα του δέντρου

Ακόμα τα πεδία Hashtable<String, Integer> tag2id, Hashtable<Integer, Integer> startNodeEncoding2idElement, Hashtable<Integer, DefaultMutableTreeNode> idElement2TreeNode, Hashtable<Integer, Integer> idElement2idJGraph που σχετίζονται με την παραγωγή της κωδικοποίησης θέσης του δέντρου.

Επίσης, τα πεδία που αφορούν το αναδυόμενο μενού.

Βασικές μέθοδοι κλάσης

- renameNode() , boolean
  μετονομασία κόμβου.
- addNode() , boolean
  προσθήκη κόμβου ως παιδί του επιλεγμένου κόμβου
- addNode_before() , boolean
  προσθήκη κόμβου ως προηγούμενος αδερφός του επιλεγμένου κόμβου.
- addNode_after() , boolean
  προσθήκη κόμβου ως επόμενος αδερφός του επιλεγμένου κόμβου
- addManyNodes() , void
  αίτηση προσθήκης πολλών κόμβων.
- addNodes_table(int num_vertices) , void
  ορισμός ονομάτων των νέων κόμβων.
- addManyNodes_ok() , boolean
  προσθήκη πολλών κόμβων ως παιδιά του επιλεγμένου κόμβου.
- removeNode() , boolean
  διαγραφή επιλεγμένων κόμβων και υποδέντρων τους.
- removeChildren() , boolean
  διαγραφή υποδέντρου (παιδιών και απογόνων) του επιλεγμένου κόμβου.
- replaceParent() , boolean
  αντικατάσταση του πατέρα του επιλεγμένου κόμβου.
- expandDescendants(DefaultMutableTreeNode node) , void
  εμφάνιση όλων των στοιχείων του υποδέντρου του επιλεγμένου κόμβου αν είναι κρυμμένα.
- actionPerformed(ActionEvent evt) , void (implements ActionListener)
  εκτέλεση κατάλληλης λειτουργίας όταν ο χρήστης το απαιτήσει

Χειρισμός αναδυόμενου μενού που εμφανίζεται με το δεξί κλικ του ποντικιού

- createPopupMenu() , void
- PopupMenuItems() , void

class PopupListener extends MouseAdapter

Fields

- JPopupMenu popup

Methods

- mousePressed(MouseEvent e) , void
- mouseReleased(MouseEvent e) , void

7.2.1.3 Διαχειριστής XML Δέντρων δενδρικής μορφής

NewTree_JGraph

Δημιουργεί ένα νέο δέντρο σε δενδρική μορφή.

inner class **DataTree**

Κρατάει τα δεδομένα ενός δέντρου δενδρικής μορφής.

Πεδία

- int id : κωδικός δέντρου δενδρικής μορφής
- JGraph graph : γράφος δέντρου δενδρικής μορφής
- GraphModel model : μοντέλο δέντρου δενδρικής μορφής
- VerticesTree_JGraph ourVertices : κόμβοι δέντρου δενδρικής μορφής

Βασικά πεδία κλάσης

Το πεδίο JGraph graph που είναι ο γράφος του δέντρου δενδρικής μορφής, το πεδίο GraphModel model που είναι το μοντέλο του δέντρου, το πεδίο VerticesTree_ JGraph ourVertices που είναι οι κόμβοι του δέντρου και το πεδίο DataTree dataTree που περιέχει τα δεδομένα του δέντρου.

Βασικές μέθοδοι κλάσης

- putRoot() , void
  προσθήκη της ρίζας του δέντρου
- initTree_ JGraph () , void
  αρχικοποίηση δέντρου
- initDataTree() , DataTree
  αρχικοποίηση δεδομένων δέντρου
- getDataTree() , DataTree
  παίρνει τα δεδομένα του δέντρου

OpenTree_JGraph

Ανοίγει ένα υπάρχον XML αρχείο σε δέντρο σε δενδρική μορφή. Αν είναι μεγάλο αρχείο, δεν το αναπαριστώ σε δενδρική μορφή (βλέπε Ενότητα 5.3.3).

Βασικά πεδία κλάσης

Το πεδίο File openFile που είναι το αρχείο που ανοίγω και το πεδίο NewTree_ JGraph.DataTree dataTree που περιέχει τα δεδομένα του δέντρου

Βασικές μέθοδοι κλάσης

- openGraph() , String
  επιλογή XML αρχείου
- generateXMLTree(File file) , void
  μετατροπή XML αρχείου σε δέντρο σε δενδρική μορφή.
- getChildren(Element el, DefaultGraphCell vertexParent) , void
  διάσχιση σε όλα τα στοιχεία του XML αρχείου
- getDataTree() , NewTree_JGraph.DataTree
  παίρνει τα δεδομένα του δέντρου
- getOpenFile() , File
  παίρνει το αρχείο που άνοιξε

SaveXML_JGraph

Αποθηκεύει ένα δέντρο δενδρικής μορφής σε ένα XML αρχείο.

Σε αυτήν την κλάση έχουμε 2 κατασκευαστές. Ο ένας καλείται μόνο στην περίπτωση save as όταν υπάρχει ήδη το αρχείο και δεν έχουν γίνει αλλαγές, οπότε δεν διασχίζω το δέντρο για να εξάγω το XML αρχείο, αλλά το αντιγράφω από το υπάρχον.

Βασικά πεδία κλάσης

Το πεδίο int num_elements που είναι το πλήθος των στοιχείων του XML αρχείου, το πεδίο Hashtable<Integer, Integer> idElement2idJGraph που είναι η δομή στην οποία αποθηκεύουμε για κάθε στοιχείο το ζευγάρι {κωδικός στοιχείου, κωδικός κόμβου}, το πεδίο VerticesTree_JGraph ourVertices που είναι οι κόμβοι του δέντρου, το πεδίο boolean doStoreTree που δηλώνει αν θα κάνω αποθήκευση ή αντιγραφή αρχείου και τα πεδία File alreadySavedFile, saveFile_XMLTree που είναι το αποθηκευμένο αρχείο του δέντρου πριν την τωρινή αποθήκευση και το τωρινά αποθηκευμένο αρχείο αντίστοιχα.

Βασικές μέθοδοι κλάσης

- saveFile() , boolean
  επιλογή XML αρχείου όπου θα αποθηκευτεί το δέντρο δενδρικής μορφής
- storeTree(File file) , boolean
  αποθήκευση του δέντρου δενδρικής μορφής σε ένα XML αρχείο.
- showChildren_JDOM(Document doc, DefaultGraphCell parentNode, Element parentElement) , void
  διάσχιση σε όλους τους κόμβους του δέντρου και δημιουργία αντίστοιχων XML στοιχείων
- getSaveFile() , File
  παίρνει το XML αρχείο όπου αποθηκεύτηκε το δέντρο
- printFileData(File file) , void
  τύπωμα δεδομένων αρχείου

SaveRegionEncoding_JGraph

Δημιουργεί την κωδικοποίηση θέσης του δέντρου σε δενδρική μορφή και την αποθηκεύει σε ένα txt αρχείο. Έχει 3 κατασκευαστές.

inner class *NodeEncoding*

Η κωδικοποίηση θέσης ενός στοιχείου

Πεδία

- int start : η πρώτη επίσκεψη του κόμβου
- int end : η τελευταία επίσκεψη του κόμβου
- int level : το επίπεδο του κόμβου

inner class *Stream*

Πεδία

- Vector<NodeEncoding> stream : το διάνυσμα που αποθηκεύω την κωδικοποίηση θέσης όλων των κόμβων του δέντρου
- public Stack<Integer> st : η στοίβα που κρατάει τη θέση στη λίστα των κόμβων που δεν έχουν τελειώσει ακόμα

Βασικά πεδία κλάσης

Τα πεδία int num_tags, int tagCounter , int regCounter και int currentDepth που είναι οι 4 μετρητές που περιγράφονται στην Ενότητα 7.2.5.

Τα πεδία Hashtable<Integer, Stream> streams, Hashtable<String, Integer> tag2id, Hashtable<Integer, String> id2tag, Hashtable<Integer, Integer> start2id, Hashtable<Integer, Integer> startNodeEncoding2idElement και Hashtable<Integer, Integer> idElement2idJGraph που χρησιμεύουν για την κωδικοποίηση θέσης του δέντρου.

Τα πεδία File saveFile_XMLTree, saveFile_XMLTree_RE που είναι το XML αρχείο που αποθηκεύω το δέντρο και το txt αρχείο που αποθηκεύω την κωδικοποίηση θέσης του δέντρου αντίστοιχα

Βασικές μέθοδοι κλάσης

Παραγωγή των κωδικοποιήσεων θέσης των κόμβων

- generateRegionEncoding() , void
  διάβασμα του XML αρχείου για την παραγωγή της κωδικοποίησης θέσης του δέντρου.
- processElement(Element el) , void
  διάσχιση όλων των στοιχείων του XML αρχείου.
- startElement(Element el) , void
  καταχώρηση των τιμών start και level στην κωδικοποίηση θέσης του στοιχείου.
- endElement(Element el) , void
  καταχώρηση της τιμής end στην κωδικοποίηση θέσης του στοιχείου.
- printstreams(Hashtable<Integer, Stream> streams) , void
  συγκέντρωση όλων των κωδικοποιήσεων θέσης των κόμβων του δέντρου.
- printStream(int tagid) , String
  τύπωμα της κωδικοποίηση θέσης ενός κόμβου στη μορφή start/end/level/0.

Αρχείο

- saveFileRE(String dataStreams) , void
  επιλογή αρχείου όπου θα αποθηκευτεί η κωδικοποίηση θέσης του δέντρου
- saveFileRE_withoutJFileChooser(String dataStreams) , void
  ορισμός αρχείου όπου θα αποθηκευτεί η κωδικοποίηση θέσης του δέντρου
- storeFileRE(File f, String dataStreams) , void
  αποθήκευση των κωδικοποιήσεων θέσης των κόμβων σε αρχείο
- getSaveFile_RE() , File
  παίρνει το txt αρχείο όπου αποθηκεύτηκε η κωδικοποίηση θέσης του δέντρου

Επιστρέφουν βασικές δομές της κλάσης

- getHash_tag2id() , Hashtable<String, Integer>
- getHash_startNodeEncoding2idElement() , Hashtable<Integer, Integer>
- getHash_idElement2idJGraph() , Hashtable<Integer, Integer>

SaveTreeData

Δημιουργεί τη μορφή treeData του δέντρου δενδρικής μορφής (βλέπε Ενότητα 6.2.3) και την αποθηκεύει σε ένα txt αρχείο.

Βασικά πεδία κλάσης

Το πεδίο GraphModel model που είναι το μοντέλο του δέντρου δενδρικής μορφής, το πεδίο VerticesTree_JGraph ourVertices που είναι οι κόμβοι του δέντρου και το πεδίο File alreadySavedFile που είναι το αρχείο στο οποίο είχαμε αποθηκεύσει την τελευταία φορά το δέντρο.

Βασικές μέθοδοι κλάσης

- saveFileTreeData() , void
  επιλογή αρχείου όπου θα αποθηκευτεί η μορφή treeData του δέντρου
- storeFileTreeData() , void
  αποθήκευση της μορφής treeData του δέντρου στο txt αρχείο.
- getTreeData() , String
  παίρνουμε τη μορφή treeData του δέντρου
- getNodes(DefaultGraphCell vertex, String treeData) , String
  παραγωγή γραμμή κόμβων της μορφής treeData.
- getEdges(DefaultGraphCell vertex, String treeData) , String
  παραγωγή γραμμών ακμών της μορφής treeData.

VerticesTree_JGraph

Διαχειρίζεται όλα τα δεδομένα ενός δέντρου σε δενδρική μορφή.

inner class **DataVertex**

Κρατάει τα δεδομένα ενός κόμβου του δέντρου δενδρικής μορφής.

Πεδία

- int id : κωδικός κόμβου vertex
- String name : όνομα κόμβου vertex
- DefaultGraphCell vertex : ο τρέχων κόμβος
- int level : το επίπεδο του δέντρου που βρίσκεται ο κόμβος vertex
- DefaultGraphCell vertexParent : ο κόμβος-πατέρας του κόμβου vertex.
- DefaultGraphCell vertexChildren : οι κόμβοι-παιδιά του κόμβου vertex.

Βασικά πεδία κλάσης

Τα πεδία JGraph graph, GraphModel model, DefaultGraphCell rootNode και DefaultGraphCell vectorOutputNodes που είναι ο γράφος,: το μοντέλο, η ρίζα και οι κόμβοι αποτελέσματος του δέντρου δενδρικής μορφής.

Τα πεδία Vector<DefaultGraphCell> streamVertices, Hashtable<Integer, DataVertex> id2dataVertex, Hashtable<DefaultGraphCell, DataVertex> vertex2dataVertex που διαχειρίζονται τα στοιχεία των κόμβων του δέντρου.

Το πεδίο int num_nodes που παρέχει σε κάθε κόμβο μοναδικό κωδικό.

Ακόμα τα πεδία Hashtable<String, Integer> tag2id, Hashtable<Integer, Integer> startNodeEncoding2idElement, Hashtable<Integer, DefaultMutableTreeNode> idElement2TreeNode, Hashtable<Integer, Integer> idElement2idJGraph που σχετίζονται με την παραγωγή της κωδικοποίησης θέσης του δέντρου.

Επίσης, τα πεδία που αφορούν το αναδυόμενο μενού.

Construstor

- VerticesTree_JGraph(Trees_and_Queries gui, JGraph graph, GraphModel model) : Αρχικοποιώ τα πεδία της κλάσης και δημιουργώ το μενού που εμφανίζεται με το δεξί κλικ του ποντικιού

Βασικές μέθοδοι κλάσης

Προσθήκη

- createRootVertex(String rootName) , DefaultGraphCell
  δημιουργία της ρίζας του δέντρου δενδρικής μορφής.
- createVertex(String name, double x, double y, double w, double h, Color bg) , DefaultGraphCell
  δημιουργία ενός νέου κόμβου.
- storeDataVertex(DefaultGraphCell vertex, DefaultGraphCell vertexParent, int level) , DefaultGraphCell
  αποθήκευση δεδομένων νέου κόμβου σε κατάλληλες δομές.
- addVertexChild() , boolean
  προσθήκη νέου κόμβου ως παιδί του επιλεγμένου κόμβου.

- addVertexMiddleChild() , boolean
  προσθήκη νέου κόμβου ως παιδί της πηγής της επιλεγμένης ακμής και ως πατέρας του προορισμού της.
- createChildVertex(DefaultGraphCell vertexParent, String childName) , DefaultGraphCell
  προσδιορισμός θέσης νέου κόμβου και δημιουργία του.
- createChildVertex(DefaultGraphCell vertexSource, DefaultGraphCell vertexTarget, String childName, boolean isMiddleChild) , DefaultGraphCell
  προσδιορισμός θέσης νέου κόμβου σύμφωνα με τη θέση του πατέρα και τη θέση του παιδιού του, αν υπάρχει, και δημιουργία του.

- addMultipleVertices () , void
  αίτηση προσθήκης πολλών κόμβων.
- addNodes_table(int num_vertices) , void
  ορισμός ονομάτων των νέων κόμβων.
- addMultipleVertices _ok() , boolean
  προσθήκη πολλών κόμβων ως παιδιάτου επιλεγμένου κόμβου.

Διαγραφή

- deleteDataVertex(DataVertex dataRemovedVertex) , void
  διαγραφή των στοιχείων του κόμβου από τις κατάλληλες δομές.
- deleteRoot() , void
  διαγραφή όλου του υποδέντρου της ρίζας αν επιλεχθεί προς διαγραφή η ρίζα.
- dialogDeleteVertices_list() , void
  επιλογή από λίστα των κόμβων προς διαγραφή.
- deleteVertices(Object[] deleteCells) , boolean
  διαγραφή κόμβων.
- deleteVertex_withDescendants(DataVertex dataRemovedVertex) , void
  διαγραφή κόμβου μαζί με όλο το υποδέντρο του καθώς και τις εισερχόμενες ακμές του κόμβου για να μην μείνουν αιωρούμενες.
- deleteVertex_withoutDescendants(DataVertex dataRemovedVertex) , void
  διαγραφή κόμβου χωρίς το υποδέντρο του καθώς και των εισερχόμενων ακμών του κόμβου. Σύνδεση παιδιών του κόμβου που διαγράφεται με τον παππού τους.
- removeIncomingEdges(DefaultGraphCell vertex) , void
  διαγραφή των εισερχόμενων ακμών του κόμβου.
- fixLevel(DataVertex dataRemovedVertex, boolean increase) , void
  διόρθωση του πεδίου επίπεδο του κόμβου στο δέντρο.

- replaceParent() , boolean
  αντικατάσταση πατέρα του επιλεγμένου κόμβου.

- getSelectedVertex(DefaultGraphCell defaultSelectedVertex, String message, String title) , DefaultGraphCell
  επιλογή από λίστα ενός κόμβου του δέντρου.

- clearAnswers() , void
  καθαρισμός δέντρου από τυχόν κόμβους αποτελέσματος.

Απόγονοι κόμβου

- showDescendants(DefaultGraphCell parent) , Vector<DefaultGraphCell>
  εντοπισμός και τύπωμα των απογόνων του κόμβου.
- selectDescendants(DefaultGraphCell parent) , Vector<DefaultGraphCell>
  εντοπισμός και τύπωμα των απογόνων του κόμβου και επιλογή τους στο γράφο.
- getDescendants(DefaultGraphCell parent) , Vector<DefaultGraphCell>
  αποθήκευση απογόνων του κόμβου σε ένα διάνυσμα
- isDescendant(DefaultGraphCell vertex, DefaultGraphCell descendant, boolean showResult) , boolean
  έλεγχος αν ένας κόμβος είναι απόγονος ενός άλλου.
- checkRelationshipBetween2Nodes(DefaultGraphCell vertex1, DefaultGraphCell vertex2) , void
  έλεγχος σχέσης μεταξύ 2 κόμβων

Εμφάνιση δεδομένων

- showDataVertex(DefaultGraphCell vertex) , void
  εμφάνιση στοιχείων ενός κόμβου.
- showLocationVertex(DefaultGraphCell vertex) , void
  εμφάνιση θέσης ενός κόμβου.
- showEdgeTree(DefaultEdge edge) , void
  εμφάνιση στοιχείων μιας ακμής

- showChildren(DefaultGraphCell parent) , String
  τύπωμα όλου το υποδέντρο (παιδιά και απογόνους) ενός κόμβου.
- showChildren_name(DefaultGraphCell parent) , String
  τύπωμα των ονομάτων των κόμβων του υποδέντρου (παιδιά και απογόνους) ενός κόμβου

- actionPerformed(ActionEvent evt) , void (implements ActionListener)
  εκτέλεση κατάλληλης λειτουργίας όταν ο χρήστης το απαιτήσει

Χειρισμός αναδυόμενου μενού που εμφανίζεται με το δεξί κλικ του ποντικιού

- createPopupMenu() , void
- createPopupMenu_vertex() , void
- createPopupMenu_edge() , void
- PopupMenuItems() , void

class PopupListener extends MouseAdapter

Fields

- JPopupMenu popup

Methods

- mousePressed(MouseEvent e) , void
- mouseReleased(MouseEvent e) , void
- maybeShowPopup(MouseEvent e) , void
  εμφάνιση κατάλληλου μενού κάνοντας δεξί κλικ στο ποντίκι, ανάλογα με τί έχει επιλεχθεί. Αν έχει επιλεχθεί κόμβος, εμφανίζει το μενού για κόμβους. Αν έχει επιλεχθεί ακμή, εμφανίζει το μενού για ακμές.

Edges

Δημιουργεί τις ακμές πατέρα-παιδιού ενός δέντρου δενδρικής μορφής.

Βασικό πεδίο κλάσης

Το πεδίο GraphModel model που είναι το μοντέλο του δέντρου δενδρικής μορφής

Βασική μέθοδος κλάσης

- createEdge(String name, Color bg, DefaultGraphCell source, DefaultGraphCell target) , DefaultEdge
  δημιουργία ακμής δέντρου δενδρικής μορφής που δηλώνει σχέση πατέρα-παιδιού.

7.2.1.4 Διαχειριστής Ερωτημάτων

NewQuery

Δημιουργεί ένα νέο ερώτημα

inner class **DataQuery**

Κρατάει τα δεδομένα ενός ερωτήματος.

Πεδία

- int id : κωδικός ερωτήματος
- JGraph graph : γράφος ερωτήματος
- GraphModel model : μοντέλο ερωτήματος
- VerticesQuery ourVertices : κόμβοι ερωτήματος
- Edges_child_descendant ourEdges : ακμές ερωτήματος

Βασικά πεδία κλάσης

Το πεδίο JGraph graph που είναι ο γράφος του ερωτήματος, το πεδίο GraphModel model που είναι το μοντέλο του ερωτήματος, το πεδίο VerticesQuery ourVertices που είναι οι κόμβοι του ερωτήματος, το πεδίο Edges_child_descendant ourEdges που είναι οι ακμές του ερωτήματος και το πεδίο DataTree dataTree που περιέχει τα δεδομένα του ερωτήματος.

Βασικές μέθοδοι κλάσης

- putRoot() , void
  προσθήκη της ρίζας του ερωτήματος
- initQuery _ JGraph () , void
  αρχικοποίηση ερωτήματος
- initDataQuery () , DataQuery
  αρχικοποίηση δεδομένων ερωτήματος
- getDataQuery () , DataQuery
  παίρνει τα δεδομένα του ερωτήματος

OpenQuery

Ανοίγει ένα υπάρχον ερώτημα

Βασικά πεδία κλάσης

Το πεδίο File openFile που είναι το αρχείο που ανοίγω και το πεδίο NewQuery.DataQuery dataQuery που περιέχει τα δεδομένα του ερωτήματος.

Τα πεδία int idOutputNode και int get_idOutputNode που αφορούν τον κόμβο εξόδου του ερωτήματος.

Υπάρχουν και τα βοηθητικά πεδία StringBuffer input, String graphData, String labels[], int nodelist[], String[] nodesNameList, int nodesPerSet[], private String cross, private float[] nodes , private int[] edgeSet που αφορούν την επεξεργασία των δεδομένων του αρχείου για την εξαγωγή των κόμβων και των ακμών του ερωτήματος.

Βασικές μέθοδοι κλάσης

Διάβασμα αρχείου ερωτήματος

- openGraph() , String
  επιλογή κατάλληλου txt αρχείου που περιέχει ερώτημα
- pspHandler (File file) , void
  μετατροπή αρχείου σε ερώτημα.
- inputFile(File file) , File
  έλεγχος αν τα περιεχόμενα του αρχείου ερωτήματος ικανοποιούν τις προδιαγραφές που έχουμε καθορίσει στην Ενότητα 6.2.4.
- readFile(File file) , String
  διαβάζει κόμβους και σχέσεις μεταξύ τους και καταχωρεί τα δεδομένα

Επεξεργασία δεδομένων αρχείου

- makeLink(String inp) , String
  δημιουργία συνδέσεων μεταξύ κόμβων αναλόγως το είδος της σχέσης
- uniqueNodes() , void
  προσδίδει μοναδικούς κωδικούς στους κόμβους
- removeRepeatedNodes(String inp) , String
  αφαίρεση των περιττών ακμών
- edgesPerGraph(String graphData), void
  επεξεργασία ακμών

Εμφάνιση των κόμβων και των ακμών

- showGraph(String graphData) , void
  εμφάνιση κόμβων και ακμών
- createVertices(int count) , DefaultGraphCell[]
  προσθέτει τους κόμβους του αρχείου στο γράφο του ερωτήματος.

- createEdges(DefaultGraphCell[] v, String open_edges[]) , void
  προσθέτει τις ακμές του αρχείου στο γράφο του ερωτήματος, διαχωρίζοντας τις σχέσεις πατέρα-παιδιού από τις σχέσεις προγόνου-απογόνου.

- getDataQuery() , NewQuery.DataQuery
  παίρνει τα δεδομένα του ερωτήματος
- getOpenFile() , File
  παίρνει το αρχείο που άνοιξε

SaveQuery

Αποθηκεύει ένα ερώτημα σε ένα txt αρχείο στη μορφή που αναπτύχθηκε στην Ενότητα 6.2.4.

Σε αυτήν την κλάση έχουμε 2 κατασκευαστές. Ο ένας αποθηκεύει το ερώτημα σε ένα txt αρχείο. Αν δεν έχει οριστεί κόμβος εξόδου ρωτάει το χρήστη εξακολουθεί να θέλει να γίνει η αποθήκευση και συνδέει τους κόμβους χωρίς εισερχόμενες ακμές με τη ρίζα ή παιδί της. Ο άλλος καλείται μόνο στην περίπτωση save as όταν υπάρχει ήδη το αρχείο και δεν έχουν γίνει αλλαγές, οπότε δεν διασχίζω το ερώτημα για να εξάγω το txt αρχείο, αλλά το αντιγράφω από το υπάρχον.

Βασικά πεδία κλάσης

Τα πεδία VerticesQuery ourVertices που είναι οι κόμβοι του ερωτήματος και Edges_child_descendant ourEdges που είναι οι ακμές του ερωτήματος, το πεδίο boolean doStoreQuery που δηλώνει αν θα κάνω αποθήκευση ή αντιγραφή αρχείου και τα πεδία File alreadySavedFile, saveFile_Query που είναι το αποθηκευμένο αρχείο του ερωτήματος πριν την τωρινή αποθήκευση και το τωρινά αποθηκευμένο αρχείο αντίστοιχα.

Βασικές μέθοδοι κλάσης

- saveFile() , boolean
  επιλογή txt αρχείου όπου θα αποθηκευτεί το ερώτημα.
- storeQuery(File file) , boolean
  αποθήκευση του ερωτήματος σε ένα txt αρχείο.
- storeQuery(File file, Hashtable<String, Integer> name2id) , boolean
  αποθήκευση του ερωτήματος σε ένα txt αρχείο κατά την αποτίμηση, λαμβάνοντας υπόψη και το δέντρο που ερωτάται.
- getSaveFile() , File
  παίρνει το XML αρχείο όπου αποθηκεύτηκε το ερώτημα

VerticesQuery

Διαχειρίζεται όλα τα δεδομένα-κόμβους ενός ερωτήματος

inner class **DataVertex**

Κρατάει τα δεδομένα ενός κόμβου

Πεδία

- int id : κωδικός κόμβου vertex
- String name : όνομα κόμβου vertex
- DefaultGraphCell vertex : ο τρέχων κόμβος
- DefaultGraphCell vertexChild : ο κόμβος-παιδί του κόμβου vertex. Αν ο κόμβος vertex δεν έχει παιδί, τότε το πεδίο αυτό είναι null.
- DefaultGraphCell vertexParent : ο κόμβος-πατέρας του κόμβου vertex. Αν ο κόμβος vertex δεν έχει πατέρα, τότε το πεδίο αυτό είναι null.

Βασικά πεδία κλάσης

Τα πεδία JGraph graph και GraphModel model που είναι ο γράφος και το μοντέλο του ερωτήματος αντίστοιχα. Το πεδίο Edges_child_descendant ourEdges που είναι οι ακμές του ερωτήματος. Τα πεδία DefaultGraphCell rootNode και DefaultGraphCell outputNode που είναι η ρίζα και ο κόμβος εξόδου του ερωτήματος αντίστοιχα.

Τα πεδία Vector<DataVertex> streamVertices, Hashtable<Integer, DataVertex> id2dataVertex και Hashtable<DefaultGraphCell, DataVertex> vertex2dataVertex που σχετίζονται με την διαχείριση των στοιχείων των κόμβων

Το πεδίο int num_nodes που παρέχει μοναδικό κωδικό σε κάθε κόμβο.

Επίσης, τα πεδία που αφορούν το αναδυόμενο μενού.

Ακόμα τα βοηθητικά πεδία για τον προσδιορισμό της θέσης των κόμβων

Construstor

- VerticesQuery(Trees_and_Queries gui, JGraph graph, GraphModel model, Edges_child_descendant ourEdges) : Αρχικοποιώ τα πεδία της κλάσης και δημιουργώ το μενού που εμφανίζεται με το δεξί κλικ του ποντικιού

Βασικές μέθοδοι κλάσης

Προσθήκη

- addVertex() , DefaultGraphCell
  προσθήκη νέου κόμβου στο ερώτημα.
- locationVertex() , DefaultGraphCell
  ορισμός θέσης του νέου κόμβου.

- createRootVertex(String rootName) , DefaultGraphCell
  δημιουργία της ρίζας του ερωτήματος.

- createVertex(String name, double x, double y, double w, double h, Color bg) , DefaultGraphCell
  δημιουργία νέου κόμβου και τοποθέτησή του στο γράφο.

- storeDataVertex(DefaultGraphCell vertex) , DefaultGraphCell
  αποθήκευση των δεδομένων του νέου κόμβου.

- addMultipleVertices() , void
  αίτηση προσθήκης πολλών κόμβων

- addNodes_table(int num_vertices) , void
  ορισμός ονομάτων των νέων κόμβων.

- addMultipleVertices_ok() , boolean
  προσθήκη πολλών κόμβων.

- addRelationship_newNode(boolean is_child) , boolean
  προσθήκη σχέσης με πηγή τον επιλεγμένο κόμβο και προορισμό έναν νέο κόμβο του ερωτήματος που θα δημιουργήσει τώρα ο χρήστης.

- addRelationship_existingNode(boolean is_child) , boolean
  προσθήκη σχέσης με πηγή τον επιλεγμένο κόμβο και προορισμό έναν υπάρχοντα κόμβο του ερωτήματος που θα επιλέξει ο χρήστης από μια λίστα.

Διαγραφή

- deleteVertices(Object[] deleteCells, Edges_child_descendant ourEdges) , boolean
  διαγραφή επιλεγμένων κόμβων καθώς και των ακμών που συνδέονται σε αυτούς.

- remove_connectedEdges(DefaultGraphCell vertex, Edges_child_descendant ourEdges) , boolean
  διαγραφή ακμών που συνδέονται στον κόμβο.

- connectedEdges(DefaultGraphCell vertex, Edges_child_descendant ourEdges, boolean remove) , List<DefaultEdge>
  εντοπισμός ακμών που συνδέονται στον κόμβο.

- deleteDataVertex(DataVertex dataRemovedVertex) , void
  διαγραφή στοιχείων του κόμβου από τις κατάλληλες δομές.

- dialogDeleteVertices(Edges_child_descendant ourEdges) , void
  επιλογή από λίστα των κόμβων προς διαγραφή.

Εμφάνιση δεδομένων

- showDataVertex(DefaultGraphCell vertex) , void
  εμφάνιση των στοιχείων του κόμβου.
- showLocationVertex(DefaultGraphCell vertex) , void
  εμφάνιση της θέσης του κόμβου.

Κόμβος εξόδου

- outputVertex(DefaultGraphCell vertex) , void
  ορισμός του κόμβου εξόδου.

- getNumEdges(GraphModel, DefaultGraphCell vertex) , static int
  παίρνει το πλήθος των συνολικών ακμών ενός κόμβου.
- checkCanonicalForm(JGraph graph, GraphModel model, VerticesQuery ourVertices, Edges_child_descendant ourEdges) , static void
  σύνδεση κόμβων χωρίς εισερχόμενες ακμές με τη ρίζα ή παιδί της (εφαρμογή κανόνα IR1).
- isPathQuery(GraphModel model, VerticesQuery ourVertices) , static boolean
  έλεγχος αν το ερώτημα μονοπατιού είναι πλήρως ορισμένο.

- actionPerformed(ActionEvent evt) , void (implements ActionListener)
  εκτέλεση κατάλληλης λειτουργίας όταν ο χρήστης το απαιτήσει

Χειρισμός αναδυόμενου μενού που εμφανίζεται με το δεξί κλικ του ποντικιού

- createPopupMenu() , void
- createPopupMenu_vertex() , void
- createPopupMenu_edge() , void
- PopupMenuItems() , void

class PopupListener extends MouseAdapter

Fields

- JPopupMenu popup

Methods

- mousePressed(MouseEvent e) , void
- mouseReleased(MouseEvent e) , void
- maybeShowPopup(MouseEvent e) , void
  εμφάνιση κατάλληλου μενού κάνοντας δεξί κλικ στο ποντίκι, ανάλογα με τί έχει επιλεχθεί. Αν έχει επιλεχθεί κόμβος, εμφανίζει το μενού για κόμβους. Αν έχει επιλεχθεί ακμή, εμφανίζει το μενού για ακμές.

Edges_child_descendant

Διαχειρίζεται όλες τις ακμές ενός ερωτήματος

inner class *DataEdge*

Κρατάει τα δεδομένα μιας ακμής

Πεδία

- int id : κωδικός edge
- String name : όνομα ακμής edge (συνήθως είναι null)
- DefaultEdge edge : ο τρέχων κόμβος
- boolean isChild : true αν είναι σχέση πατέρα-παιδιού, false αν είναι σχέση προγόνου-απογόνου.
- DefaultGraphCell source : ο κόμβος-πηγή της ακμής edge. Το πεδίο αυτό δεν μπορεί να είναι null.
- DefaultGraphCell target : ο κόμβος προορισμού της ακμής edge. Το πεδίο αυτό δεν μπορεί να είναι null.

Βασικά πεδία κλάσης

Τα πεδία JGraph graph, GraphModel model και VerticesQuery ourVertices που είναι ο γράφος, το μοντέλο και οι κόμβοι του ερωτήματος.

Τα πεδία Vector<DataEdge> streamEdges, Hashtable<Integer, DefaultEdge> id2edge, Hashtable<DefaultEdge, DataEdge> edge2dataEdge που σχετίζονται με τη διαχείριση των στοιχείων των ακμών.

Το πεδίο int num_edges που παρέχει μοναδικό κωδικό σε κάθε ακμή. Το πεδίο boolean relationshipChild που προσδιορίζει το είδος της σχέσης που παριστάνει η ακμή.

Βασικές μέθοδοι κλάσης

- dialogEdgeData() , void
  ορισμός χαρακτηριστικών της νέας ακμής (πηγή, προορισμός, όνομα, είδος σχέσης)
- createEdge (String name, Color bg, boolean relationshipChild, DefaultGraphCell source, DefaultGraphCell target) , DefaultEdge
  δημιουργία ακμής εφόσον ικανοποιεί τις προδιαγραφές.
- addEdge(String name, Color bg, boolean relationshipChild, DefaultGraphCell source, DefaultGraphCell target) , DefaultEdge
  προσθήκη της νέας ακμής στο γράφο
- storeDataEdge(DefaultEdge edge, boolean isChild, DefaultGraphCell source, DefaultGraphCell target) , DefaultEdge
  αποθήκευση των στοιχείων τις ακμής στις κατάλληλες δομές.

- fixOutgoingDescendants(DefaultGraphCell source, DefaultGraphCell target) , void
  επεξεργασία των απογόνων της πηγής-πατέρα της νέας σχέσης πατέρα-παιδιού.
- fixIncomingDescendants(DefaultGraphCell source, DefaultGraphCell target) , void
  επεξεργασία των προγόνων του προορισμού-παιδιού της νέας σχέσης πατέρα-παιδιού.

Διαγραφή ακμών

- deleteDataEdge(DataEdge dataRemovedEdge) , boolean
  διαγραφή δεδομένων ακμής
- deleteDataEdge(DefaultEdge removedEdge) , boolean
  διαγραφή δεδομένων ακμής
- deleteEdges(Object[] deleteEdges, boolean select) , boolean
  διαγραφή ακμών
- deleteEdges_list(Object[] deleteEdges) , boolean
  διαγραφή ακμών επιλεγμένων από λίστα

- dialogDeleteEdges_list() , void
  επιλογή των ακμών προς διαγραφή από λίστα.

- dialogShowEdges_list() , void
  εμφάνιση των ακμών με τα χαρακτηριστικά τους σε μορφή λίστας.
- dialogShowEdges_table() , void
  εμφάνιση των ακμών με τα χαρακτηριστικά τους σε μορφή πίνακα.

- actionPerformed(ActionEvent evt) , void (implements ActionListener)
  εκτέλεση κατάλληλης λειτουργίας όταν ο χρήστης το απαιτήσει

Cells

Διαχειρίζεται τους κόμβους και τις ακμές ενός ερωτήματος. Παρουσιάζει και διαγράφει κόμβους και ακμές σαν σύνολο.

Βασικό πεδίο κλάσης

Το πεδίο NewQuery.DataQuery dataQuery που είναι οι κόμβοι του ερωτήματος.

Βασικές μέθοδοι κλάσης

- ShowCells() , void
  παρουσίαση κόμβων και ακμών του ερωτήματος.
- PanelShowCells() , JPanel
  δημιουργία panel που περιέχει τις 2 καρτέλες με τους κόμβους και τις ακμές του ερωτήματος.
- PanelNodes() , JPanel
  δημιουργία panel κόμβων ερωτήματος.
- PanelEdges() , JPanel
  δημιουργία panel ακμών ερωτήματος.
- showCells() , void
  τύπωμα στοιχείων κάθε κόμβου και κάθε ακμής.

- removeAll() , boolean
  διαγραφή επιλεγμένων κόμβων και ακμών καθώς και των ακμών που είναι συνδεδεμένες στους κόμβους προς διαγραφή.

- actionPerformed(ActionEvent evt) , void (implements ActionListener)
  εκτέλεση κατάλληλης λειτουργίας όταν ο χρήστης το απαιτήσει

GPCellViewFactory & JGraphEllipseView

Δίνουν στους κόμβους των ερωτημάτων κυκλικό σχήμα.

7.2.1.5 Αποτιμητής

Evaluation

Βασικά πεδία της κλάσης

Τα πεδία NewTree_JGraph.DataTree dataEvaluateTree_JGraph, NewTree_JTree.DataTree dataEvaluateTree_JTree , NewQuery.DataQuery dataEvaluateQuery περιέχουν τα δεδομένα του δέντρου και του ερωτήματος που θα αποτιμηθούν ενώ το πεδίο Algorithm algorithm δηλώνει τον αλγόριθμο με τον οποίο θα γίνει η αποτίμηση.

Τα πεδία Hashtable<Integer, DataTab> idTreeCombo2dataTab, int idTreeCombo, Hashtable<Integer, Component> idTreeCombo2component, Component componentEvaluateTree και Hashtable<String, DataTab> nameQuery2dataTab ,

Hashtable<String, Component> nameQueryCombo2component, Component componentEvaluateQuery βοηθούν στη διάκριση του δέντρου και του ερωτήματος αντίστοιχα που έχουμε επιλέξει για την αποτίμηση

Βασικές μέθοδοι της κλάσης

- chooseFiles_and_algorithm_combobox() , void
  επιλογή δέντρου, ερωτήματος και αλγόριθμου αποτίμησης

Άνοιγμα νέων αρχείων

- selectTree() , void
  επιλογή σε ποια μορφή να ανοιχτεί το νέο δέντρο
- openComboFile(Application application) , void
  άνοιγμα αρχείου
- openComboTree_jtree() , void
  άνοιγμα νέου δέντρου μορφής ετικετών
- openComboTree_jgraph() , void
  άνοιγμα νέου δέντρου δενδρικής μορφής
- openComboQuery () , void
  άνοιγμα νέου ερωτήματος

Αποτίμηση

- evaluation() , void
  Κάνει τους πρώτους ελέγχους για την αποτίμηση και εντοπίζει το δέντρο προς αποτίμηση.
- evaluate_JGraph(File directory, DataTab dataEvaluateTab_tree) , void
  παίρνει την κωδικοποίηση θέσης του επιλεγμένου για την αποτίμηση δέντρου δενδρικής μορφής
- evaluate_JTree(File directory, DataTab dataEvaluateTab_tree) , void
  παίρνει την κωδικοποίηση θέσης του επιλεγμένου για την αποτίμηση δέντρου δενδρικής μορφής
- getQueryResults(File directory, Hashtable<String, Integer> tag2id) , String
  παίρνει το επιλεγμένο ερώτημα και εκτελεί την αποτίμηση
- checkVerticesNames(Hashtable<String, Integer> hash_tag2id_tree, Vector<VerticesQuery.DataVertex> streamVertices_query) , boolean
  έλεγχος για έγκυρα ονόματα κόμβων ερωτήματος ως προς το δέντρο.

Κλήση εξωτερικών εκτελέσιμων σε γλώσσα C++ για την αποτίμηση του αντίστοιχου αλγορίθμου

- executeCpp_PartialPathStack_inMap(int idNode) , String
  εκτέλεση του PartialPathStack με αποτελέσματα τους κωδικούς των κόμβων αποτελέσματος του δέντρου.
- executeCpp_PathStack_inMap(int idNode) , String
  εκτέλεση του PathStack με αποτελέσματα τους κωδικούς των κόμβων αποτελέσματος του δέντρου
- executeCpp_PartialPathStack () , String
  εκτέλεση του PartialPathStack με αποτελέσματα όλα τα μονοπάτια αποτελέσματος καθώς και το πλήθος των αποτελεσμάτων και ο χρόνος που χρειάστηκε για την αποτίμηση.
- executeCpp_PathStack () , String
  εκτέλεση του PathStack με αποτελέσματα όλα τα μονοπάτια αποτελέσματος καθώς και το πλήθος των αποτελεσμάτων και ο χρόνος που χρειάστηκε για την αποτίμηση.

Επεξεργασία αποτελεσμάτων στην περίπτωση δέντρου δενδρικής μορφής

- processResults_JGraph(String results, Hashtable<Integer, Integer> startNodeEncoding2idElement, Hashtable<Integer, Integer> idElement2idJGraph) , void
  επεξεργασία των αποτελεσμάτων και εντοπισμός των κόμβων αποτελέσματος
- showResults_JGraph(Vector<DefaultGraphCell> vector_outputVertices) , void
  εμφάνιση των αποτελεσμάτων στο αρχικό δέντρο και δημιουργία του νέου δέντρου αποτελέσματος σε δενδρική μορφή
- storeSubtreeResultVertex_JGraph (NewTree_JGraph.DataTree dataResultTree, DefaultGraphCell vertexParent, DefaultGraphCell resultVertex) , void
  προσθήκη στο δέντρο-αποτέλεσμα των κόμβων αποτελέσματος μαζί με όλο το υποδέντρο τους που έχουν στο αρχικό δέντρο

Επεξεργασία αποτελεσμάτων στην περίπτωση δέντρου μορφής ετικετών

- processResults_JTree(String results, Hashtable<Integer, Integer> startNodeEncoding2idElement, Hashtable<Integer, DefaultMutableTreeNode> idElement2TreeNode) , void
  επεξεργασία των αποτελεσμάτων και εντοπισμός των κόμβων αποτελέσματος
- showResults_JTree(Vector<DefaultMutableTreeNode> vector_outputVertices) , void
  δημιουργία του νέου δέντρου αποτελέσματος σε μορφή ετικετών

- storeSubtreeResultVertex_JTree(NewTree_JTree.DataTree dataResultTree, DefaultMutableTreeNode parentNode, DefaultMutableTreeNode answerNode) , DefaultMutableTreeNode
  προσθήκη στο δέντρο-αποτέλεσμα των κόμβων αποτελέσματος μαζί με όλο το υποδέντρο τους που έχουν στο αρχικό δέντρο

- copy(File source, File destination) , static void
  αντιγραφή ενός αρχείου σε ένα άλλο

- actionPerformed(ActionEvent evt) , void (implements ActionListener)
  εκτέλεση κατάλληλης λειτουργίας όταν ο χρήστης το απαιτήσει

Κώδικας σε C++

Ακόμα στο τμήμα αυτό χρησιμοποιούνται οι κλάσεις υλοποιημένες σε C++ από το εργαστήριο Συστημάτων Βάσεων Γνώσεων και Δεδομένων.

- Tree → XML δέντρο πάνω στο οποίο κάνουμε την ερώτηση
- Tag → κωδικοποίηση θέσης κόμβου XML δέντρου
- PP → ερώτηση μονοπατιού μερικής ή ολικής δομής
- FileParser → διαβάζει το δέντρο και το ερώτημα προς αποτίμηση
- PathStack → αλγόριθμος PathStack
- PartialPathStack → αλγόριθμος PartialPathStack

Τέλος υπάρχουν και η κλάση MyVector και το αρχείο test.cpp υλοποιημένα επίσης σε C++.

- MyVector → κληρονομεί τη vector.
  Περιέχει επιπλέον τη μέθοδο contains(T value) που ελέγχει αν το διάνυσμά μας περιέχει την τιμή value.

- test.cpp
  Το αρχείο test.cpp δεν είναι συστατικό κάποιας κλάσης. Απλά περιέχει τη μέθοδο main(), από την οποία εκκινεί κάθε φορά η εκτέλεση του συστήματος. Χρησιμοποιεί τις κλάσεις FileParser, PathStack και PartialPathStack που είναι υλοποιημένες από τον Συστημάτων Βάσεων Γνώσεων και Δεδομένων και την κλάση MyVector.

  Διαβάζει το δέντρο και το ερώτημα με τη βοήθεια της κλάσης FileParser και εκτελεί τον αλγόριθμο PathStack ή PartialPathStack. Έπειτα επεξεργάζεται τα αποτελέσματα που δίνει ο αλγόριθμος (βλέπε Ενότητα7.2.6) και με τη βοήθεια της MyVector επιστρέφει τις μοναδικές τιμές start της κωδικοποίησης θέσης.

7.2.1.6 Βοηθητικές Κλάσεις – singleton pattern

NameTranslation

Ελέγχει τα ονόματα των κόμβων να είναι έγκυρα XML ονόματα ετικετών και τα μετατρέπει σε κατάλληλη μορφή ανάλογα με την περίσταση.

Πεδίο κλάσης : static NameTranslation nameTranslation

Βασικές μέθοδοι κλάσης

- getInstance() , static NameTranslation
- getRootName(boolean isTree) , String
  παίρνει ένα όνομα για τον κόμβο της ρίζας και ελέγχει αν είναι έγκυρο
- getValidName() , String
  παίρνει ένα όνομα για τον νέο κόμβο και ελέγχει αν είναι έγκυρο
- setIntegerName(String childName) , String
  αν το όνομα του κόμβου ξεκινάει με ακέραιο, τότε προσθέτουμε τον χαρακτήρα '_' για την αποθήκευση του κόμβου
- getIntegerName(String childName) , String
  αν το όνομα του κόμβου που ανοίγουμε από αρχείο ξεκινάει με τον χαρακτήρα '_' και μετά ακολουθεί ακέραιος, τότε αφαιρούμε τον χαρακτήρα '_' για την αναπαράστασή του
- checkValidName(String childName) , Boolean
  ελέγχει αν είναι έγκυρο XML όνομα
- getNodeName(String name) , String
  αφαιρεί τους χαρακτήρες '<' και ΄>΄ από τον κόμβο του δέντρου μορφής ετικετών για την επεξεργασία ή την αποθήκευσή του

Save

Διαχειρίζεται 2 θέματα. Αν υπάρχουν αλλαγές στο δέντρο ή στο ερώτημα που δεν έχουν αποθηκευτεί και αν υπάρχει έγκυρο αρχείο κωδικοποίησης θέσης ενός δέντρου.

Πεδίο κλάσης : static Save save

Μέθοδοι κλάσης

- getInstance() , static Save

- isSaved(Trees_and_Queries gui, NewTab.DataTab dataTab) , void
  Δηλώνει ότι τα δεδομένα της καρτέλας έχουν αποθηκευτεί

- isUnSaved(Trees_and_Queries gui, NewTab.DataTab dataTab) , void
  Δηλώνει ότι τα δεδομένα της καρτέλας έχουν αλλάξει και ότι δεν έχουν αποθηκευτεί οι αλλαγές.

Αποθήκευση των δομών που αφορούν την κωδικοποίηση θέσης μετά από αποθήκευση των αλλαγών του δέντρου.

- storeHash_JGraph(VerticesTree_JGraph ourVertices, SaveRegionEncoding_JGraph re) , void
- storeHash_JTree(VerticesTree_JTree ourVertices, SaveRegionEncoding_JTree re) , void

Άδειασμα των δομών που αφορούν την κωδικοποίηση θέσης μετά από αλλαγή στο δέντρο.

- clearHash_JGraph(VerticesTree_JGraph ourVertices) , void
- clearHash_JTree(VerticesTree_JGraph ourVertices) , void

Αρχείο κωδικοποίησης θέσης

- getValidRegionEncoding_JGraph(File fileTree, VerticesTree_JGraph ourVertices) , File
  παίρνει το έγκυρο αρχείο κωδικοποίησης θέσης του δέντρου δενδρικής

- getValidRegionEncoding_JTree(File fileTree, VerticesTree_JTree ourVertices) , File
  παίρνει το έγκυρο αρχείο κωδικοποίησης θέσης του δέντρου μορφής ετικετών.

Selection

Ελέγχει τα αντικείμενα που είναι επιλεγμένα στο γράφο ενός δέντρου δενδρικής μορφής ή στο γράφο ερωτήματος.

Πεδίο κλάσης : static Selection selection

Μέθοδοι κλάσης

- getInstance() , static Selection
- getOneSelectedVertex(JGraph graph, GraphModel model String title) , DefaultGraphCell
  παίρνει τον μοναδικό επιλεγμένο κόμβο του γράφου.

- getOneSelectedEdge(JGraph graph, GraphModel model String title) , DefaultEdge
  παίρνει τη μοναδική επιλεγμένη ακμή του γράφου

7.2.2 Βασικά Στοιχεία Υλοποίησης

Σε αυτό το σημείο είναι καλό να συγκεντρώσουμε κάποια βασικά στοιχεία της υλοποίησης.

Υπάρχουν έξι βασικές δομές-κλάσεις στην εφαρμογή μας που αποθηκεύουν τα δεδομένα μας. Τα πεδία τους υπάρχουν αναλυτικά στην προηγούμενη ενότητα αλλά και αναλυτικότερα στο Παράρτημα Β.

- **`NewTab.DataTab`** → αποθήκευση δεδομένων καρτέλας
- **`NewTree_JTree.DataTree`** → αποθήκευση δεδομένων δέντρου μορφής ετικετών
- **`NewTree_JGraph.DataTree`** → αποθήκευση δεδομένων δέντρου δενδρικής μορφής
- **`NewQuery.DataQuery`** → αποθήκευση δεδομένων ερωτήματος
- **`VerticesTree_JGraph.DataVertex`** → αποθήκευση δεδομένων κόμβου δέντρου δενδρικής μορφής
- **`VerticesQuery.DataVertex`** → αποθήκευση δεδομένων κόμβου ερωτήματος
- **`Edges_child_descendant.DataEdge`** → αποθήκευση δεδομένων ακμής ερωτήματος.

Ακολουθούν κάποιες βασικές δομές στην εφαρμογή μας που χρησιμοποιούνται για να διακρίνουμε για ποια δεδομένα αναφερόμαστε κάθε φορά και πού ανήκουν αυτά.
Είναι εύκολο να καταλάβουμε από την ονομασία τί κάνουν. Υπάρχει πάντως και αναλυτική περιγραφή τους στο Παράρτημα Β.

Trees_and_Queries
Για τη διαχείριση των αρχείων που έχει ανοιχτά η εφαρμογή.

```
Vector<File> streamFiles;
```

Για τη διαχείριση των δέντρων και των ερωτημάτων. Από την πρώτη δομή εντοπίζω τι περιέχει η κάθε καρτέλα. Κάθε καρτέλα μπορεί να περιέχει ένα δέντρο μορφής ετικετών, ένα δέντρο δενδρικής μορφής ή ένα ερώτημα. Ανάλογα με τι περιέχει παίρνω τα δεδομένα του από τις υπόλοιπες τρεις δομές.

```
Hashtable<String, NewTab.DataTab> componentName2dataTab;


Hashtable<NewTab.DataTab, NewTree_JTree.DataTree>
dataTab2dataTree_JTree;
Hashtable<NewTab.DataTab, NewTree_JGraph.DataTree>
dataTab2dataTree_JGraph;
Hashtable<NewTab.DataTab, NewQuery.DataQuery> dataTab2dataQuery;
```

VerticesTree_JTree
Για την αποθήκευση και παραγωγή της κωδικοποίησης θέσης του δέντρου μορφής ετικετών.

Με τις δομές αυτές ελέγχω αν είναι περιττή η κωδικοποίηση θέσης (βλέπε Ενότητα 7.2.5).

```
Hashtable<String, Integer> tag2id;

Hashtable<Integer, Integer> startNodeEncoding2idElement;

Hashtable<Integer, Integer> idElement2idJGraph;

Hashtable<Integer, DefaultMutableTreeNode> idElement2TreeNode;
```

VerticesTree_JGraph
Για την αποθήκευση και παραγωγή της κωδικοποίησης θέσης του δέντρου δενδρικής μορφής.

Με τις δομές αυτές ελέγχω αν είναι περιττή η κωδικοποίηση θέσης (βλέπε Ενότητα 7.2.5).

```
Hashtable<String, Integer> tag2id;

Hashtable<Integer, Integer> startNodeEncoding2idElement;

Hashtable<Integer, Integer> idElement2idJGraph;

Hashtable<Integer, DefaultMutableTreeNode> idElement2TreeNode;
```

Για τη διαχείριση των κόμβων του δέντρου δενδρικής μορφής
```
Vector<DefaultGraphCell> streamVertices;

Hashtable<Integer, DataVertex> id2dataVertex;

Hashtable<DefaultGraphCell, DataVertex> vertex2dataVertex;
```

VerticesQuery
Για τη διαχείριση των κόμβων του ερωτήματος

```
Vector<DataVertex> streamVertices;

Hashtable<Integer, DataVertex> id2dataVertex;

Hashtable<DefaultGraphCell, DataVertex> vertex2dataVertex;
```

Edges_child_descendant
Για τη διαχείριση των ακμών του ερωτήματος

```
Vector<DataEdge> streamEdges;

Hashtable<Integer, DefaultEdge> id2edge;

Hashtable<DefaultEdge, DataEdge> edge2dataEdge;
```

Τέλος για διακρίσεις λειτουργιών χρησιμοποιούμε κάποια enumeration.

`enum Application{Tree_jtree, Tree_jgraph, Query}` → για να διαχωρίζω τι από τα τρία περιέχει μια καρτέλα. Ανήκει στην κλάση Trees_and_Queries.

enum Create{*New*, *Open*} → για να διαχωρίζω αν θέλω να δημιουργήσω ή να φορτώσω ένα αρχείο. Ανήκει στην κλάση Trees_and_Queries.

enum Algorithm{*PartialPathStack*, *PathStack*} → για να επιλέξω τον αλγόριθμο αποτίμησης. Ανήκει στην κλάση Evaluation.

7.2.3 Προσθήκη Ακμής Ερωτήματος

Για να προσθέσω μία ακμή πρέπει να έχω προσδιορίσει τον κόμβο-πηγή και τον κόμβο προορισμού.

Αρχικά κάνω τους εξής ελέγχους :

- ελέγχω αν οι κόμβοι πηγής και προορισμού είναι κενοί.
- ελέγχω αν ο κόμβος προορισμού είναι η ρίζα του ερωτήματος
- ελέγχω αν οι κόμβοι πηγής και προορισμού ταυτίζονται.

Αν ισχύει ένα από αυτά τότε δεν μπορώ να προσθέσω τη νέα ακμή.

Αλλιώς συνεχίζω τη διαδικασία διαχωρίζοντας τις εξής περιπτώσεις ανάλογα με το είδος της σχέσης :

1. Σχέση προγόνου-απογόνου
   Ελέγχω αν υπάρχει στον κόμβο-πηγή και στον κόμβο προορισμού σχέση πατέρα-παιδιού.

   - IR4
     Ελέγχω αν ο κόμβος-πηγή έχει παιδί. Αν ναι, προχωράω στο παιδί και ελέγχω αν αυτό έχει παιδί. Αν ναι προχωράω στο παιδί του. Κάνω αυτή τη διαδικασία προχωρώντας προς τα κάτω μέχρι να συναντήσω κόμβο που να μην έχει παιδί. Τότε ορίζεται αυτός ο κόμβος ως κόμβος-πηγή.

     Εν τω μεταξύ αν συναντήσω καθώς προχωράω προς τα κάτω τον κόμβο προορισμού, τότε δημιουργείται κύκλος και η διαδικασία σταματάει χωρίς καμιά προσθήκη ακμής.

   - IR5
     Ελέγχω αν ο κόμβος προορισμού έχει πατέρα. Αν ναι, προχωράω στον πατέρα και ελέγχω αν αυτός έχει πατέρα. Αν ναι προχωράω στον πατέρα του. Κάνω αυτή

τη διαδικασία προχωρώντας προς τα πάνω μέχρι να συναντήσω κόμβο που να μην έχει πατέρα. Τότε ορίζεται αυτός ο κόμβος ως κόμβος προορισμού.

Εν τω μεταξύ αν συναντήσω καθώς προχωράω προς τα πάνω τον κόμβο-πηγή που είχα αρχικά ή αυτόν που προέκυψε από την εκτέλεση του κανόνα IR4, τότε δημιουργείται κύκλος και η διαδικασία σταματάει χωρίς καμιά προσθήκη ακμής.

Ακόμα, αν προκύψει ότι ο νέος κόμβος προορισμού είναι η ρίζα του ερωτήματος, τότε πάλι η διαδικασία σταματάει χωρίς καμιά προσθήκη ακμής.

Με αυτόν τον τρόπο προσδιορίζουμε τους τελικούς κόμβους πηγής και προορισμού που συνδέονται με σχέση προγόνου-απογόνου και η διαδικασία συνεχίζεται

2. Σχέση πατέρα-παιδιού
Ελέγχω αν υπάρχει στον κόμβο-πηγή και στον κόμβο προορισμού και άλλη σχέση πατέρα-παιδιού. Αν υπάρχει δεν επιτρέπεται, γιατί κάθε πηγή πρέπει να έχει μοναδικό παιδί και κάθε προορισμός μοναδικό πατέρα, αφού έχουμε ερωτήσεις μονοπατιού. Δηλαδή κάθε κόμβος δεν μπορεί να έχει 2 πατεράδες ούτε μπορεί να έχει 2 παιδιά.

- <u>Ο κόμβος-πηγή πρέπει να έχει μοναδικό παιδί</u>, αφού πρόκειται για ερωτήσεις μονοπατιού.
  Αν υπάρχει ήδη ένα παιδί, τότε έχουμε :
  - Αν το υπάρχον παιδί ταυτίζεται με το παιδί που θέλω τώρα, τότε αφού υπάρχει ήδη η σχέση που θέλω να προσθέσω, ενημερώνω απλά το χρήστη ότι υπάρχει ήδη αυτή σχέση πατέρα-παιδιού και η διαδικασία σταματάει.
  - Αν το υπάρχον παιδί διαφέρει από το παιδί που θέλω τώρα, τότε ενημερώνω τον χρήστη ότι υπάρχει ήδη ένα παιδί στην πηγή και τον ρωτάω αν θέλει να διαγράψω την σχέση αυτή για να προσθέσω την καινούρια.
    - Αν διαφωνήσει η διαδικασία σταματάει.
    - Αν συμφωνήσει παίρνω τις εξερχόμενες ακμές της πηγής (που πρέπει να είναι μία, αφού υπάρχει παιδί), εντοπίζω το υπάρχον παιδί, διαγράφω τη σχέση αυτή και η διαδικασία συνεχίζεται.

- <u>Ο κόμβος προορισμού πρέπει να έχει μοναδικό πατέρα</u>, αφού πρόκειται για ερωτήσεις μονοπατιού.
  Αν υπάρχει ήδη ένας πατέρας, τότε έχουμε :
  - Αν ο υπάρχον πατέρας ταυτίζεται με τον πατέρα που θέλω τώρα, τότε αφού υπάρχει ήδη η σχέση που θέλω να προσθέσω δεν κάνω τίποτα και η διαδικασία σταματάει.

- Αν ο υπάρχον πατέρας διαφέρει από τον πατέρα που θέλω τώρα, τότε ενημερώνω τον χρήστη ότι υπάρχει ήδη ένας πατέρας στον προορισμό και τον ρωτάω αν θέλει να διαγράψω την σχέση αυτή για να προσθέσω την καινούρια.
  - Αν διαφωνήσει η διαδικασία σταματάει.
  - Αν συμφωνήσει παίρνω τις εισερχόμενες ακμές του προορισμού (που πρέπει να είναι μία, αφού υπάρχει πατέρας), εντοπίζω τον υπάρχοντα πατέρα, διαγράφω τη σχέση αυτή και η διαδικασία συνεχίζεται.

Αν έχουμε φτάσει σε αυτό το σημείο τότε ελέγχουμε αν υπάρχει ήδη μια σχέση μεταξύ της πηγής και του προορισμού Οπότε κάνουμε τα εξής :

1. Παίρνω τις εξερχόμενες ακμές της πηγής και βρίσκω αυτές που έχουν ως προορισμό τον προορισμό της σχέσης που θέλω να προσθέσω. Δηλαδή εντοπίζω τις ακμές που έχουν κοινή πηγή και κοινό προορισμό με τη νέα μας σχέση.
   - Αν η σχέση της υπάρχουσας ακμής είναι πατέρα-παιδιού, τότε ενημερώνω το χρήστη ότι υπάρχει σχέση πατέρα-παιδιού μεταξύ των κόμβων πηγής και προορισμού της σχέσης που θέλω και γι'αυτό δεν είναι δυνατή η νέα προσθήκη και η διαδικασία σταματάει.
   - Αν η σχέση της υπάρχουσας ακμής είναι προγόνου-απογόνου ενώ η νέα σχέση είναι πατέρα-παιδιού, τότε ενημερώνω το χρήστη ότι υπάρχει σχέση προγόνου-απογόνου μεταξύ των κόμβων πηγής και προορισμού της σχέσης που θέλω και τον ρωτάω αν θέλει να την αντικαταστήσει με σχέση πατέρα-παιδιού (πιο περιοριστική).
     - Αν διαφωνήσει η διαδικασία σταματάει.
     - Αν συμφωνήσει διαγράφω την παλιά σχέση προγόνου-απογόνου, προσθέτω τη νέα σχέση πατέρα-παιδιού και η διαδικασία σταματάει.
   - Αν η σχέση της υπάρχουσας ακμής είναι προγόνου-απογόνου καθώς και η νέα σχέση, τότε απλά ενημερώνω το χρήστη ότι υπάρχει αυτή σχέση προγόνου-απογόνου και η διαδικασία σταματάει.
2. Παίρνω τις εισερχόμενες ακμές της πηγής και βρίσκω αυτές που έχουν ως πηγή τον προορισμό της σχέσης που θέλω να προσθέσω.
   - Αν υπάρχει τέτοια ακμή τότε ενημερώνουμε τον χρήστη ότι υπάρχει ήδη μια ακμή μεταξύ των κόμβων πηγής και προορισμού με αντίθετη κατεύθυνση από αυτή που θέλουμε και ότι δεν είναι δυνατή η προσθήκη της νέας σχέσης γιατί θα σχηματιστεί ένας trivial κύκλος που απαγορεύεται και έτσι η διαδικασία σταματάει.
   - Αν δεν υπάρχει τέτοια ακμή τότε η διαδικασία συνεχίζεται.

Αν έχουμε φτάσει σε αυτό το σημείο και δεν έχει τελειώσει η διαδικασία, τότε γίνεται η προσθήκη της νέας σχέσης και αποθηκεύονται τα στοιχεία της στις δομές μας.

Τέλος, στην περίπτωση που πρόκειται για σχέση πατέρα-παιδιού πρέπει να ελέγξω αν υπάρχουν συνδεδεμένοι πρόγονοι ή απόγονοι στους κόμβους πηγής και προορισμού ώστε να τους μετακινήσω κατάλληλα.

Έχουμε λοιπόν :

1. Πρόγονοι πηγής ➔ δεν επηρεάζονται από τη νέα σχέση

2. Απόγονοι προορισμού ➔ δεν επηρεάζονται από τη νέα σχέση

3. Απόγονοι πηγής
   - Παίρνω τον προορισμό της νέας σχέσης πατέρα-παιδιού (δηλαδή το παιδί) και ελέγχω αν έχει παιδί. Αν ναι, προχωράω στο παιδί και ελέγχω αν αυτό έχει παιδί. Αν ναι, προχωράω στο παιδί του. Κάνω αυτή τη διαδικασία προχωρώντας προς τα κάτω μέχρι να συναντήσω κόμβο που να μην έχει παιδί. Τότε ορίζεται αυτός ο κόμβος ως κόμβος πηγής όλων των κόμβων-απογόνων της πηγής-πατέρα της νέας σχέσης. (κανόνας IR4).
   - Παίρνω τις εξερχόμενες ακμές της πηγής. Για κάθε απόγονο της πηγής κάνω τα εξής:
     - Παίρνω τις εισερχόμενες ακμές του προορισμού του εκάστοτε απογόνου της πηγής και ελέγχω αν υπάρχει ακμή με πηγή τη νέα πηγή των απογόνων της πηγής.
       - Αν υπάρχει τέτοια ακμή, τότε είναι πλέον περιττή η αλλαγή της πηγής αυτού του απογόνου και έτσι η σχέση αυτή διαγράφεται αφού υπάρχει ήδη.
       - Αν δεν υπάρχει τέτοια ακμή, τότε αντικαθιστώ την πηγή του απογόνου στην κοινή πηγή όλων των απογόνων που εξάγαμε στην αρχή.

4. Πρόγονοι προορισμού
   - Παίρνω την πηγή της νέας σχέσης πατέρα-παιδιού (δηλαδή τον πατέρα) και ελέγχω αν έχει πατέρα. Αν ναι, προχωράω στον πατέρα και ελέγχω αν αυτός έχει πατέρα. Αν ναι, προχωράω στον πατέρα του. Κάνω αυτή τη διαδικασία προχωρώντας προς τα πάνω μέχρι να συναντήσω κόμβο που να μην έχει πατέρα. Τότε ορίζεται αυτός ο κόμβος ως κόμβος προορισμού όλων των κόμβων-προγόνων του προορισμού-παιδιού της νέας σχέσης. (κανόνας IR5).
     - Αν ο κόμβος αυτός είναι η ρίζα του ερωτήματος και υπάρχουν πρόγονοι του προορισμού, τότε ενημερώνω το χρήστη ότι δεν μπορεί η ρίζα να έχει προγόνους και γι'αυτό τους διαγράφω όλους και η διαδικασία σταματάει.
   - Παίρνω τις εισερχόμενες ακμές του προορισμού. Για κάθε πρόγονο του προορισμού κάνω τα εξής :
     - Παίρνω τις εξερχόμενες ακμές της πηγής του εκάστοτε προγόνου του προορισμού και ελέγχω αν υπάρχει ακμή με προορισμό το νέο προορισμό των προγόνων του προορισμού.
       - Αν υπάρχει τέτοια ακμή, τότε είναι πλέον περιττή η αλλαγή του προορισμού αυτού του προγόνου και έτσι η σχέση αυτή διαγράφεται αφού υπάρχει ήδη.
       - Αν δεν υπάρχει τέτοια ακμή, τότε αντικαθιστώ το προορισμό του προγόνου στον κοινό προορισμό όλων των προγόνων που εξάγαμε στην αρχή.

7.2.4 Άνοιγμα – Αποθήκευση XML Αρχείων

Για την πρόσβαση στα στοιχεία του κειμένου XML χρησιμοποιούμε το JDOM, ενώ για την εισαγωγή και την εξαγωγή του XML αρχείου πρέπει να βασιστούμε σε κάποιον parser, εμείς χρησιμοποιούμε τον SAX parser. Υπάρχουν λεπτομέρειες στην Ενότητα 2.2.

7.2.5 Κωδικοποίηση Θέσης

Αλγόριθμος παραγωγής κωδικοποίησης θέσης

Διαβάζουμε το XML αρχείο του δέντρου για το οποίο θέλουμε να εξάγουμε τις κωδικοποιήσεις θέσης των στοιχείων του.

Μετρητές

Χρησιμοποιούμε 4 βασικούς μετρητές :

1. Μετρητής όλων των στοιχείων : Προσδιορίζει μοναδικά το κάθε στοιχείο. Αυξάνεται σε κάθε συνάντηση νέου στοιχείου.
2. Μετρητής των διαφορετικών στοιχείων : Προσδιορίζει μοναδικά το κάθε διαφορετικό στοιχείο. Αυξάνεται σε κάθε συνάντηση νέου στοιχείου που δεν έχει το ίδιο όνομα με τα στοιχεία που έχουμε ήδη επισκεφτεί..
3. Μετρητής επίσκεψης : Προσδιορίζει σε ποια επίσκεψη βρισκόμαστε. Αυξάνεται σε κάθε είσοδο και έξοδο από κάθε στοιχείο.
4. Μετρητής επιπέδου : Προσδιορίζει σε ποιο επίπεδο του δέντρου βρισκόμαστε. Αυξάνεται σε κάθε είσοδο στοιχείου και μειώνεται σε κάθε έξοδο από στοιχείο.

Διαφορά των δύο πρώτων μετρητών

Κάθε στοιχείο προσδιορίζεται από δύο κωδικούς :

- έναν που είναι μοναδικός και δεν μπορεί να τον έχει κανένα άλλο στοιχείο του δέντρου (προσδιορίζεται από τον $1^{ο}$ μετρητή) και
- έναν που θα είναι ίδιος με όλα τα στοιχεία του δέντρου που έχουν το ίδιο όνομα (προσδιορίζεται από τον $2^{ο}$ μετρητή).

Όταν έχουμε λοιπόν 2 στοιχεία με το ίδιο όνομα, ο $1^{ος}$ μετρητής δίνει διαφορετικό κωδικό στα 2 στοιχεία, ενώ ο $2^{ος}$ μετρητής δίνει τον ίδιο κωδικό.

Οι δύο αυτοί μετρητές ταυτίζονται αν όλα τα στοιχεία του δέντρου έχουν μοναδικό όνομα. (βλέπε παράδειγμα στους πίνακες Πίνακα 7.1 και Πίνακα 7.2).

Διαδικασία

Κάνουμε ***κατά βάθος διάσχιση*** του δέντρου.

Αν συναντήσουμε ένα στοιχείο για πρώτη φορά τότε αποθηκεύουμε την τιμή του μετρητή επίσκεψης στο πεδίο start της κωδικοποίησης θέσης του ($1^{η}$ επίσκεψη στοιχείου) καθώς και

το μετρητή επιπέδου στο πεδίο level της κωδικοποίησης θέσης του. Όταν διασχίσουμε όλο το υποδέντρο του, ξαναεπισκεπτόμαστε το στοιχείο αυτό για τελευταία φορά και καταχωρούμε στο πεδίο end τον μετρητή επίσκεψης (τελευταία επίσκεψη στοιχείου). Έτσι με την τελευταία επίσκεψη του στοιχείου έχουμε παράγει την κωδικοποίηση θέσης του.

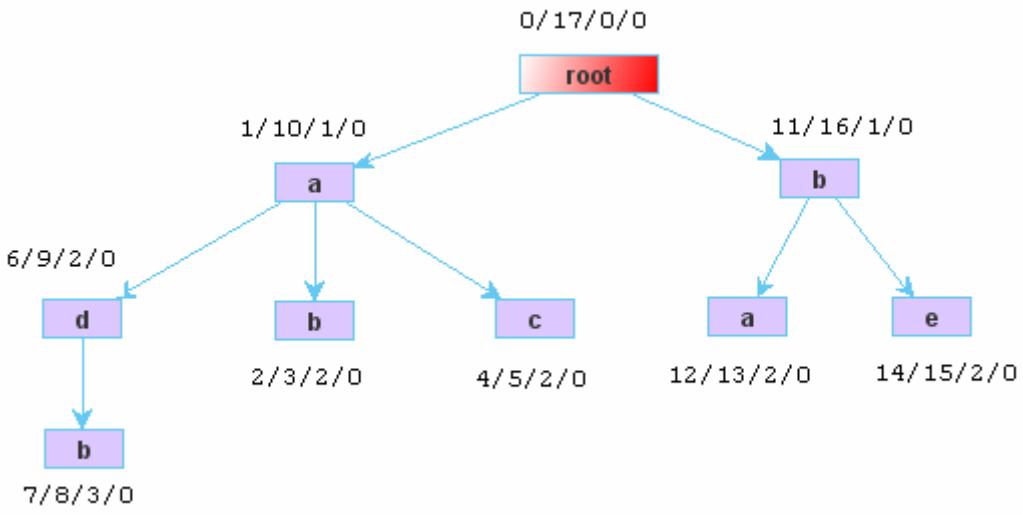
Αξίζει να αναφέρουμε ότι στα φύλλα του δέντρου, που δεν έχουν υποδέντρο, τα πεδία start και end διαφέρουν μόνο κατά ένα.

Κάνοντας αυτή τη διαδικασία για όλα τα στοιχεία του δέντρου τότε έχουμε πάρει όλες τις κωδικοποιήσεις θέσεις των στοιχείων του.

Μπορούμε να πάρουμε κάθε στοιχείο ξεχωριστά και να πάρουμε την κωδικοποίηση θέσης του ή μπορούμε να 'μαζέψουμε' τα στοιχεία με το ίδιο όνομα και να πάρουμε όλες τις κωδικοποιήσεις θέσης των στοιχείων αυτών.

- Στην αποτίμηση ερωτημάτων χρησιμοποιούμε τον πρώτο τρόπο για να εντοπίσουμε τους κόμβους αποτελέσματος. Αν χρησιμοποιούσαμε τον δεύτερο τρόπο τότε θα παίρναμε πιθανόν επιπλέον κόμβους ως αποτέλεσμα, αυτούς που θα είχαν ίδιο όνομα με τους κόμβους αποτελέσματος αλλά δεν θα ικανοποιούσαν τις σχέσεις του ερωτήματος.
- Στα αρχεία που αποθηκεύουμε την κωδικοποίηση θέσης του δέντρου, χρησιμοποιούμε το δεύτερο τρόπο (βλέπε Ενότητα 6.2.2).

Στο Σχήμα 7.1 φαίνεται ένα μικρό παράδειγμα κωδικοποίησης θέσης.



Σχήμα 7.1 : Κωδικοποίηση θέσης δέντρου σε δενδρική μορφή. Δίπλα από κάθε κόμβο υπάρχει η κωδικοποίηση θέσης του.

όνομα	κωδικός (1ος μετρητής)	κωδικοποίηση θέσης
root	0	0/17/0/0
a	1	1/10/1/0
b	2	2/3/2/0
c	3	4/5/2/0
d	4	6/9/2/0
b	5	7/8/3/0
b	6	11/16/1/0
a	7	12/13/2/0
e	8	14/15/2/0

Πίνακας 7.1 : Κωδικοποίηση θέσης δέντρου. Για κάθε κόμβο φαίνεται το όνομά του, ο κωδικός του σύμφωνα με τον 1^{ο} μετρητή και η κωδικοποίηση θέσης του.

όνομα	κωδικός (2ος μετρητής)	κωδικοποίηση θέσης
root	0	0/17/0/0
a	1	1/10/1/0 12/13/2/0
b	2	2/3/2/0 7/8/3/0 11/16/1/0
c	3	4/5/2/0
d	4	6/9/2/0
e	5	7/8/3/0

Πίνακας 7.2 : Κωδικοποίηση θέσης δέντρου. Για κάθε διαφορετικό κόμβο φαίνεται το όνομά του, ο κωδικός του σύμφωνα με τον 2^{ο} μετρητή και οι κωδικοποιήσεις θέσης του. Παρατηρούμε ότι ο κόμβος a εμφανίζεται στο δέντρο 2 φορές και ο b 3, γι'αυτό έχουν 2 και 3 κωδικοποιήσεις θέσης αντίστοιχα. Οι υπόλοιποι κόμβοι (root, c, d, e) εμφανίζονται από μία φορά στο δέντρο και έχουν μία κωδικοποίηση θέσης.

Πότε παράγουμε την κωδικοποίηση θέσης ενός δέντρου?

Κάθε φορά που αποθηκεύω ένα XML δέντρο σε ένα XML αρχείο, εξάγω και την κωδικοποίηση θέσης των κόμβων του δέντρου και την αποθηκεύω σε ένα txt αρχείο με την ίδια ονομασία που έχω δώσει στο XML αρχείο. Για παράδειγμα θέλω να δώσω στο δέντρο το όνομα mytree, τότε δημιουργώ δύο αρχεία : το mytree.xml όπου αποθηκεύω το XML δέντρο και το mytree.txt όπου αποθηκεύω την κωδικοποίηση θέσης του XML δέντρου.

Με αυτόν τον τρόπο, εξάγω την κωδικοποίηση θέσης μόνο αν χρειάζεται, αλλιώς παίρνω την επιθυμητή κωδικοποίση θέσης από τη συγκεκριμένη τοποθεσία..

Αντιγραφή κωδικοποίησης θέσης, όχι παραγωγή της

Για να γλυτώσω χρόνο και περιττή επεξεργασία, αν έχω ήδη εξάγει την κωδικοποίηση θέσης και δεν έχω κάνει κάποια αλλαγή στο XML δέντρο, τότε δεν ξαναπαράγω την κωδικοποίηση θέσης, αλλά εντοπίζω το txt αρχείο του δέντρου όπου έχω αποθηκεύσει την κωδικοποίηση θέσης και παίρνω από εκεί τα δεδομένα που χρειάζομαι.

Οι λειτουργίες που μπορεί να χρειαστούν την ήδη αποθηκευμένη κωδικοποίηση θέσης είναι η αποθήκευση του δέντρου με άλλη ονομασία (Save as) και η αποτίμηση ενός ερωτήματος σε ένα δέντρο (Evaluation)

Έτσι, αν θέλουμε να κάνουμε πολλά ερωτήματα σε ένα XML δέντρο δεν θα παράγουμε κάθε φορά την κωδικοποίηση θέσης αφού το δέντρο δεν έχει αλλάξει. Η κωδικοποίηση θέσης θα αποθηκεύεται την πρώτη φορά σε ένα txt αρχείο και μετά θα εντοπίζουμε αυτό το αρχείο για να παίρνουμε την κωδικοποίηση θέσης που χρειαζόμαστε για την αποτίμηση του ερωτήματος.

7.2.6 Αποτίμηση

Στην ενότητα αυτή παρουσιάζεται ο τρόπος υλοποίησης της αποτίμησης ενός ερωτήματος πάνω σε ένα δέντρο. Στο Σχήμα 7.2 υπάρχει ένα παράδειγμα.

Ακόμα παρουσιάζεται ο τρόπος σύνδεσης του κώδικα σε Java με τον κώδικα σε C++.

Αλγόριθμος Αποτίμησης

Βήμα 1 : Επιλογή αρχείων και αλγορίθμου

Επιλέγουμε το ερώτημα και το XML δέντρο πάνω στο οποίο θέλουμε να γίνει η αποτίμηση του ερωτήματος, από τα ήδη ανοιχτά αρχεία ή και από άλλα που μπορούμε να ανοίξουμε εκείνη τη στιγμή.

Επίσης, διαλέγουμε και τον αλγόριθμο (PathStack ή PartialPathStack) με τον οποίο θέλουμε να γίνει η αποτίμηση.

Βήμα 2 : Έλεγχοι

Αν επιλέξουμε τον PathStack, πριν κάνουμε οτιδήποτε, ελέγχουμε αν πρόκειται για πλήρως ορισμένο ερώτημα μονοπατιού. Αν είναι, τότε συνεχίζουμε κανονικά την αποτίμηση με τον αλγόριθμο PathStack. Αν δεν είναι, τότε ρωτάμε τον χρήστη αν θέλει να γίνει η αποτίμηση με τον PartialPathStack. Αν συμφωνήσει συνεχίζουμε την αποτίμηση με τον PartialPathStack, αλλιώς σταματάει η αποτίμηση.

Ένας δεύτερος βασικός έλεγχος που κάνουμε πριν προχωρήσουμε στην αποτίμηση είναι να δούμε αν έχουμε ορίσει κόμβο εξόδου στο ερώτημά μας. Αν όχι σταματάει η αποτίμηση.

Ένας άλλος έλεγχος που γίνεται αργότερα (μετά το βήμα 3) είναι να δούμε αν τα ονόματα των κόμβων του ερωτήματος υπάρχουν στο XML δέντρο μας. Αν έστω και ένα όνομα κόμβου ερωτήματος δεν υπάρχει στο XML δέντρο τότε η αποτίμηση τελειώνει με μηδενικό αποτέλεσμα, πριν φτάσουμε στην εκτέλεση του επιλεγμένου αλγορίθμου αποτίμησης.

Βήμα 3 : Κωδικοποίηση θέσης δέντρου & Ερώτημα

Για την αποτίμηση χρειαζόμαστε δύο αρχεία : την κωδικοποίηση θέσης του XML δέντρου και το ερώτημα στη μορφή που περιγράψαμε στην Ενότητα 6.2.4.

Δημιουργούμε δύο προσωρινά αρχεία σε συγκεκριμένο σημείο του δίσκου, τα οποία σβήνονται στην έξοδο της εφαρμογής.

- *Κωδικοποίηση θέσης δέντρου*
  Αν δεν έχουμε κάνει αλλαγές στο δέντρο και έχουμε ήδη εξάγει την κωδικοποίηση θέσης του παλαιότερα, τότε εντοπίζουμε το αρχείο αυτό και το αντιγράφουμε στο επιθυμητό αρχείο προορισμού. Αλλιώς παράγουμε εκείνη τη στιγμή την κωδικοποίηση θέσης.
- *Ερώτημα*
  Το ερώτημα αποθηκεύεται κάθε φορά στην αποτίμηση γιατί πρέπει οι κόμβοι του να αντιστοιχούν στους κόμβους του δέντρου πάνω στο οποίο θέλουμε να κάνουμε την ερώτηση. Μάλιστα διαφέρει από τη μορφή που περιγράψαμε στην Ενότητα 6.2.4 στο σημείο της γραμμής αποθήκευσης των κόμβων. Σε αυτό το σημείο αντί να αποθηκεύσουμε το όνομα του κόμβου του ερωτήματος, βάζουμε τον κωδικό του κόμβου που αντιστοιχεί στο δέντρο, γιατί οι αλγόριθμοι PathStack και PartialPathStack επεξεργάζονται μόνο κωδικούς και όχι ονόματα κόμβων.

Βήμα 4 : Εκτέλεση αλγορίθμου αποτίμησης

Αν τα αποτελέσματα των παραπάνω ελέγχων είναι τα επιθυμητά, συνεχίζουμε με την εκτέλεση του επιλεγμένου αλγορίθμου αποτίμησης. Σε αυτό το σημείο καλείται από τη Java εφαρμογή ένα εξωτερικό εκτελέσιμο υλοποιημένο σε C++ που εκτελεί τον κατάλληλο αλγόριθμο αποτίμησης και περνάει ως παράμετρο τον κωδικό του κόμβου εξόδου (βλέπε επόμενη ενότητα).

Βήμα 5 : Εξαγωγή Αποτελεσμάτων (C++)

Οι αλγόριθμοι PathStack και PartialPathStack που χρησιμοποιούμε εξάγουν τα αποτελέσματα σε ένα διάνυσμα από ζευγάρια { $id_{κόμβου\_ερωτήματος}$, κωδικοποίηση θέσης του στο δέντρο }. Κάθε ζευγάρι αντιστοιχεί σε έναν κόμβο του ερωτήματος, οπότε το διάνυσμα αποτελείται από τόσα στοιχεία όσοι είναι οι κόμβοι του ερωτήματος.

Εμείς χρειαζόμαστε μόνο την κωδικοποίηση θέσης που αντιστοιχεί στον κόμβο εξόδου.

Ακόμα επειδή οι τιμές start και end της κωδικοποίησης θέσης κάθε κόμβου είναι μοναδικές στο δέντρο, παίρνουμε μόνο την τιμή start και όχι όλη την κωδικοποίηση θέσης του κόμβου αποτελέσματος, εξοικονομώντας χώρο και επεξεργασία.

Τέλος επειδή ένας κόμβος του δέντρου μπορεί να εμφανιστεί στο αποτέλεσμα περισσότερες από μία φορές, εμείς θα τον κρατήσουμε μόνο μία φορά. Δεν έχει νόημα να εμφανίσουμε το ίδιο αποτέλεσμα δύο ή περισσότερες φορές.

Έτσι τυπώνω στην έξοδο τις μοναδικές start τιμές των αποτελεσμάτων διαχωρίζοντας τις με ένα κενό.

Γενικά γνωρίζουμε ότι η C++ είναι πιο γρήγορη και αποτελεσματική στην εκτέλεση αλγορίθμων ενώ η Java είναι καλύτερη στα γραφικά.

Προτιμάται έτσι αυτή η παραπάνω επεξεργασία να γίνει στον C++ κώδικα που είναι πιο αποδοτικός ενώ στον κώδικα με Java να αφήσει μόνο την εμφάνιση των αποτελεσμάτων. Επιπλέον με τον τρόπο αυτό δεν στέλνουμε περιττά στοιχεία στη Java για να τα κόψει μετά εκείνη, αλλά της δίνει μόνο ό,τι χρειάζεται για την απεικόνιση των αποτελεσμάτων.

Στο Σχήμα 7.3 φαίνεται ένα παράδειγμα των λειτουργιών του βήματος 5.

Βήμα 6 : Παρουσίαση αποτελεσμάτων (Java)

Εντοπίζουμε τους κόμβους αποτελέσματος από τις start τιμές που παίρνουμε από το προηγούμενο βήμα και παρουσιάζουμε τα αποτελέσματα.

- Αν δεν ικανοποιείται το ερώτημα, δηλαδή έχουμε μηδενικά αποτελέσματα, ενημερώνουμε κατάλληλα το χρήστη και δεν δημιουργείται καινούριο δέντρο-αποτέλεσμα
- Αν έχουμε μη-μηδενικά αποτελέσματα τότε δημιουργείται ένα καινούριο δέντρο-αποτέλεσμα όπου παιδιά της ρίζας του δέντρου είναι οι κόμβοι αποτελέσματος ενώ τους κόμβους αποτελέσματος ακολουθεί όλο το υποδέντρο τους (παιδιά και απόγονοι). Επιπλέον αν πρόκειται για δέντρο δενδρικής μορφής, εκτός από τη δημιουργία του καινούριου δέντρου-αποτέλεσμα, έχουμε και την εμφάνιση των κόμβων αποτελέσματος πάνω στο αρχικό δέντρο.

Ένωση κύριας εφαρμογής μας, υλοποιημένη με κώδικα Java με την αποτίμηση υλοποιημένη με C++

Όταν έχω πλέον στον επιθυμητό προορισμό την κωδικοποίηση θέσης του δέντρου και το ερώτημα, εκτελώ τον κώδικα του εργαστηρίου για την αποτίμηση του ερωτήματος πάνω στο δέντρο είτε με τον αλγόριθμο PathStack είτε με τον PartialPathStack.

Για να ενώσουμε την κύρια εφαρμογή μας υλοποιημένη με Java με την αποτίμηση υλοποιημένη σε C++ κοιτάξαμε τα εξής :

- Η ένωση να γίνει σύμφωνα με το JNI(Java Native Interface). Όμως μας περιόριζε να χρησιμοποιήσουμε το Microsoft Visual C++ compiler γιατί χρειαζόταν το εκτελέσιμο cl.exe, για τη δημιουργία της dll βιβλιοθήκης, που μόνο η πλατφόρμα αυτή παρέχει [4].
- Η ένωση να γίνει σύμφωνα με το JNI(Java Native Interface) ενώ η κλήση της dll να γίνει με Borland C++ Builder [5].
- Καλώντας από τον κώδικα σε Java μια εξωτερική εντολή που εκτελεί το αρχείο που κάνει την αποτίμηση σε C++ κώδικα.

Σύγκριση Μεθόδων

Με τους δύο πρώτους τρόπους περιορίζομαι στην πλατφόρμα, αλλά μπορώ να περάσω παραμέτρους από τη Java C++, αλλά και να επιστρέψω βασικούς τύπους παραμέτρων από τη C++ στη Java.

Η τρίτη περίπτωση όμως είναι ανεξάρτητη πλατφόρμας, μπορεί να περάσει παραμέτρους από τη Java στη C++, αλλά δεν μπορεί να περάσει παραμέτρους από τη C++ στη Java.

Επιλογή μας

Εμείς επιλέξαμε την τρίτη περίπτωση επειδή είναι ανεξάρτητη πλατφόρμας. Για να ξεπεράσουμε το πρόβλημα που προαναφέραμε, στον κώδικα C++ τυπώνουμε στην έξοδο τα στοιχεία που μας χρειάζονται και μετά έρχεται η Java να τα διαβάσει και να τα επεξεργαστεί.

Εφαρμογή μας

Συγκεκριμένα στην εφαρμογή μας, έχουμε στη C++ ένα διάνυσμα από ακέραιες τιμές. Τις τυπώνουμε λοιπόν στην έξοδο διαχωρίζοντας τις με ένα κενό. Έρχεται στη συνέχεια η Java να διαβάσει μία μία τις ακέραιες τιμές και να τις αποθηκεύσει σε ένα δικό της διάνυσμα απο ακέραιους.

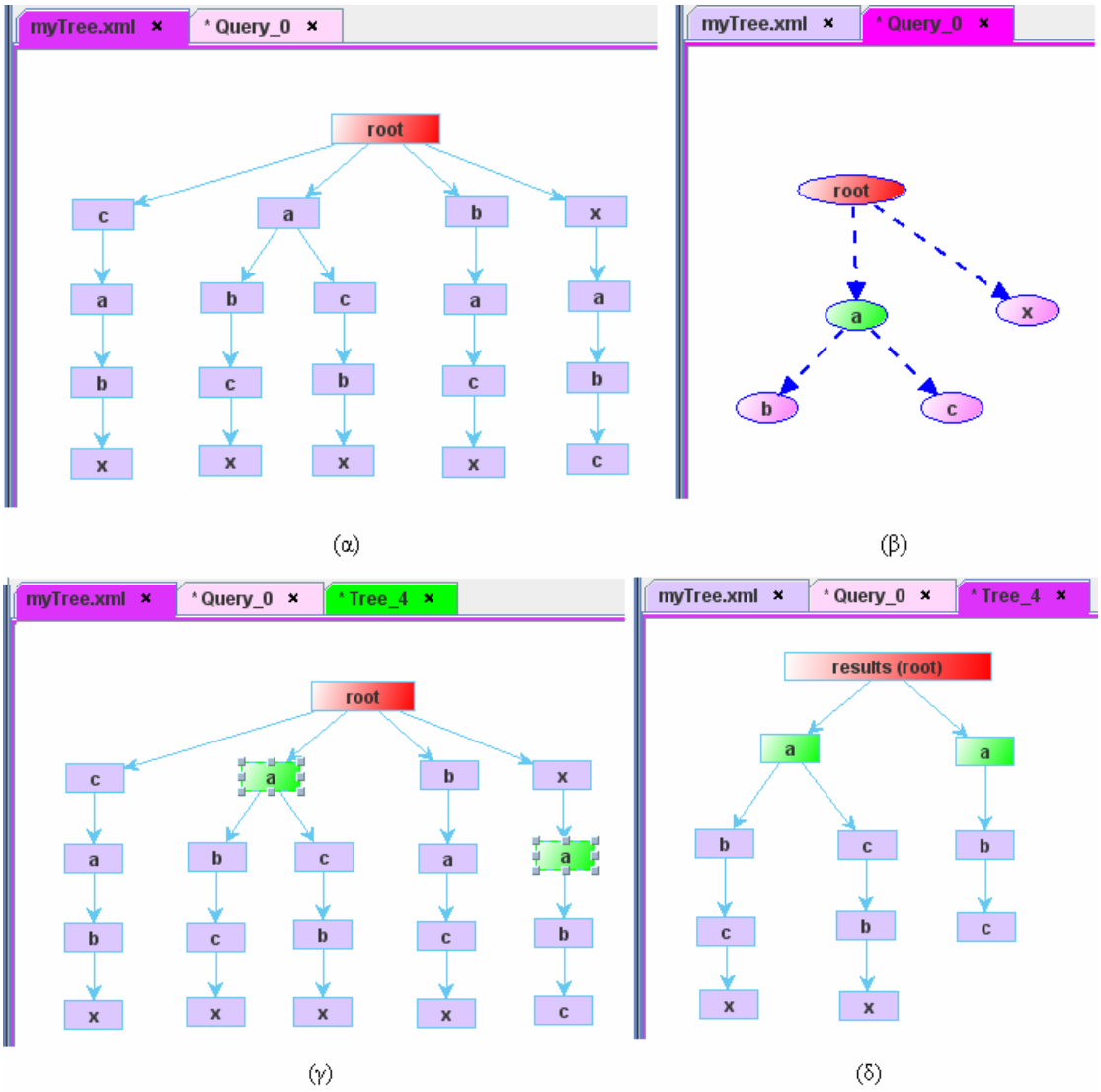
Κλήση εξερχόμενης εντολής και αποτελέσματα

Από την κυρίως εφαρμογή σε Java καλείται το εκτελέσιμο που έχει παραχθεί από κώδικα C++ με την εντολή :

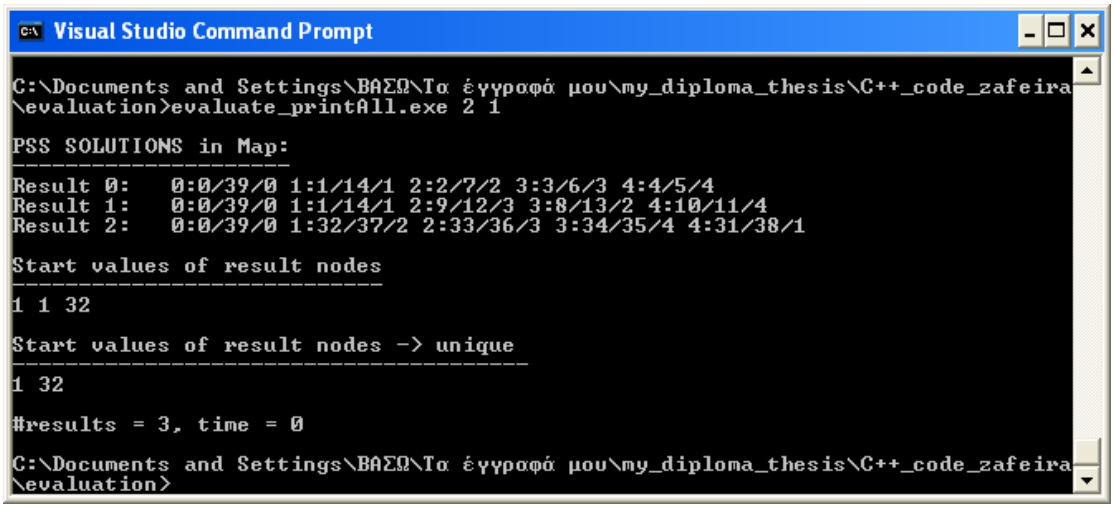
```
Process p = Runtime.getRuntime().exec(parameters);
```

Σαν παραμέτρους δίνω το όνομα του εκτελέσιμου και τον κωδικό του κόμβου εξόδου του ερωτήματος.

Το εκτελέσιμο γραμμένο σε C++ τυπώνει τις start τιμές της κωδικοποίησης θέσης των κόμβων του δέντρου που αντιστοιχούν στον κόμβο εξόδου του ερωτήματος (βλέπε Βήμα 5 προηγούμενης ενότητας). Μετά το πρόγραμμα επιστρέφει στην κυρίως εφαρμογή γραμμένη σε Java όπου διαβάζει τις start τιμές που τυπώθηκαν. Από αυτές εντοπίζει τους κόμβους αποτελέσματος του δέντρου και παρουσιάζει τα αποτελέσματα.



Σχήμα 7.2 : Αποτίμηση. (α) Το δέντρο (β) Το ερώτημα (γ) Τα αποτελέσματα στο αρχικό δέντρο (δ) Το δέντρο αποτελέσματος.



Σχήμα 7.3 : Εξαγωγή αποτελεσμάτων.

Στο Σχήμα 7.3 φαίνονται τα αποτελέσματα της αποτίμησης του ερωτήματος του Σχήματος 7.2β πάνω στο δέντρο του Σχήματος 7.2α. Αρχικά φαίνονται τα 3 μονοπάτια του δέντρου που ικανοποιούν το ερώτημα. Το ερώτημα έχει 5 κόμβους. Οπότε ο αλγόριθμος αποτίμησης επιστρέφει 3 διανύσματα με 5 ζευγάρια από { id : node encoding }. Τα 3 αυτά διανύσματα φαίνονται στις πρώτες γραμμές. Ο κόμβος εξόδου του ερωτήματος έχει κωδικό 1. Εντοπίζουμε λοιπόν στο κάθε διάνυσμα τον κόμβο με κωδικό 1 και παίρνουμε την κωδικοποίηση θέσης του. Έπειτα παίρνουμε μόνο το πεδίο start για κάθε κωδικοποίηση θέσης οπότε έχουμε τις τιμές 1 1 32. Παρατηρούμε ότι το 1 εμφανίζεται δύο φορές που δεν το θέλουμε. Έτσι στέλνουμε μόνο τις τιμές 1 32 όπως φαίνεται στο τέλος. Ακόμα εμφανίζονται το πλήθος των αποτελεσμάτων και ο χρόνος.

8

Επίλογος

8.1 Σύνοψη και Συμπεράσματα

Η παρούσα διπλωματική εργασία αφορούσε την ανάπτυξη μιας πλατφόρμας μοντελοποίησης και αναπαράστασης XML δέντρων και ερωτημάτων καθώς και την αποτίμηση μερικώς ορισμένων ερωτημάτων μονοπατιού σε XML δέντρα. Βασικό χαρακτηριστικό των ερωτήσεων είναι ότι επιτρέπουν μερικό προσδιορισμό δομής. Στηρίχθηκε κυρίως στο paper του Στέφανου Σουλδάτου και άλλων με τίτλο : **"Evaluation of Partial Path Queries on XML Data"** [**CIKM07**], όπου παρουσιάστηκαν οι έννοιες των μερικώς ορισμένων ερωτημάτων μονοπατιού, των συμπερασματικών κανόνων, της ικανοποιησιμότητας ενός ερωτήματος, των πλεοναζόντων κόμβων, της κανονικής μορφής ενός ερωτήματος καθώς και οι αλγόριθμοι αποτίμησης μερικώς ορισμένων μονοπατιών PartialMJ, PartialMJPlus και PartialPathStack.

Ασχοληθήκαμε με την υλοποίηση ενός σχεδιαστικού εργαλείου, του ***Partialist***, το οποίο δίνει τη δυνατότητα σχεδίασης και διαχείρισης XML δέντρων μορφής ετικετών, XML δέντρων δενδρικής μορφής και ερωτημάτων. Ακόμα δίνει τη δυνατότητα αποτίμησης ενός ερωτήματος πάνω σε ένα XML δέντρο και αναπαράσταση του δέντρου-αποτελέσματος.

Επίσης παρέχει βασικές λειτουργίες όπως προσθήκη κόμβου/κόμβων δέντρου ή ερωτήματος, ενημέρωση ενός κόμβου, προσθήκη κόμβου-παιδιού σε δέντρο, προσθήκη ακμής ερωτήματος που δηλώνει σχέση πατέρα-παιδιού, προσθήκη ακμής ερωτήματος που δηλώνει σχέση προγόνου-απογόνου, διαγραφή κόμβων και ακμών, αντικατάσταση κόμβου-πατέρα ενός κόμβου σε δέντρο. Συνάμα επιτρέπει, μέσω ενός συνόλου από άλλες λειτουργίες, μια πλειάδα επεμβάσεων πάνω στα ερωτήματα όπως εύρεση των άκυρων ακμών, έλεγχο πλήρως ορισμένου ερωτήματος μονοπατιού, έλεγχο ύπαρξης 'ξεκρέμαστων' κόμβων ερωτήματος.

8.2 Μελλοντικές επεκτάσεις

Στην παράγραφο αυτή παρουσιάζουμε κάποιες επεκτάσεις που θεωρούμε ότι μπορούν να γίνουν στην εφαρμογή που αναπτύξαμε, ώστε να ολοκληρωθούν κάποιες λειτουργίες που προσφέρει και να βελτιωθεί η ποιότητα των παρεχόμενων υπηρεσιών :

1. Επεξεργασία και αναπαράσταση όχι μόνο ετικετών, αλλά και κειμένων, μεταβλητών, σχολίων κτλ ενός XML εγγράφου
2. Αποθήκευση και επεξεργασία της θέσης των κόμβων πάνω στο panel κατά τη στιγμή που επιλέχθηκε η λειτουργία της αποθήκευσης.
3. Επέκταση για πολλαπλά ανεξάρτητα ερωτήματα σε ένα ερώτημα
4. Αποτίμηση όχι μόνο για ερωτήματα μονοπατιού, αλλά και για δενδρικώς ορισμένα ερωτήματα. Σε αυτή την περίπτωση πρέπει να αναθεωρήσουμε κάποιους περιορισμούς που θέτουμε τώρα στα ερωτήματα.
5. Εξαγωγή treeData και από δέντρο σε μορφή ετικετών

9

Βιβλιογραφία

[1] N.Bruno, N.Koudas, D.Srivastava, Holistic twig joins: optimal XML pattern matching, Proc. of ACM SIGMOD, 2002

[2] S.Souldatos, X.Wu, D.Theodoratos, T.Dalamagas, T.Sellis, Evaluation of Partial Path Queries on XML Data, Proc. of ACM CIKM, 2007

[3] S.Souldatos, X.Wu, D.Theodoratos, T.Dalamagas, T.Sellis, Evaluation of Partial Path Queries on XML Data, Journal of Latex Class Files, 1(8), Augoust 2002

[4] Sheng Liang, The Java Native Interface – Programmer's Guide and Specification, June 1999

[5] Dickman Luke, calling from a .dll using Java and JNI, Embarcadero Technologies, August 2005

[6] S.Al-Khalifa, H.V.Jaradish, N.Koudas, M.Patel, D.Srivastava, Y.Wu, Structural joins : A primitive for efficient XML query pattern matching, Proc. of ICDE, 2002

[7] L.Chen, A.Gupta, M.E.Kurul, Stack-based algorithms for pattern matching on dags, Proc. of VLDB, 2005

[8] S.Chen, H.G.Li, J.Tatemura, W.P.Hsiung, D.Agrawal, K.S.Candan, Twig2Stack : bottom-up processing of generalized-tree-pattern queries over XML documents, Proc. of VLDB, 2006

[9] S.Y.Chien, Z.Vagena, D.Zhang, V.J.Tsotras, C.Zaniolo, Efficient structural joins on indexed XML documents, Proc. of VLDM, 2002

[10] M.Fontoura, D.Kossman, I.Manolescu, Integrating keyword search into xml query processing, Computer Networks, 2000

[11] H.Jiang, H.Lu, W.Wang, Efficient processing of xml twig queries with or-predicates, Proc. of ACM SIGMOD, 2004

[12] H.Jiang, H.Lu, W.Wang, B.C.Ooi, Xr-tree : Indexing xml data for efficient structural joins, Proc. of ICDE, 2003

[13] H.Jiang, W.Wang, H.Lu, J.X.Yu, Holistic twig joins on indexed XML documents, Proc. of VLDB, 2003

[14] J.Lu, T.Chen, T.W.Ling, Efficien processing of XML twig patterns with parent child edges : A look-ahead approach, Proc. of CIKM, 2004

[15] B.Yang, M.Fontoura, E.Shekita, S.Rajagopalan, K.Beyer, Virtual cursors for XML joins, Proc. of ICDE, 2004

[16] C.Zhang, J.F.Naughton, D.J.DeWitt, Q.Luo, G.M.Lohman, On supporting [25] containment queries in relational database management systems, Proc. of ACM SIGMOD, 2001

[17] D.Oltenau, H.Meuss, T.Furche, F.Bry, Xpath: Looking forward, Proc. of XMLDM, MDDE, YRWS, 2002

[18] D.Oltenau, Forward node-selecting queries over trees, Proc. of ACM Trans, 2007

[19] G.Gottlob, C.Koch, R.Pichler, The complexity of xpath query evaluation, Proc. of PODS, 2003

[20] http//www.w3.org/TR/xml

[21] http//www.w3school.com

ΠΑΡΑΡΤΗΜΑ Α

Εγχειρίδιο Χρήσης

Στο παράρτημα αυτό αναλύουμε τις λειτουργίες του interface της εφαρμογής και εξηγούμε τον τρόπο χρήσης του προγράμματος. Αρχικά όμως παρουσιάζουμε τις οδηγίες εγκατάστασης.

Οδηγίες Εγκατάστασης

Χρησιμοποιήσαμε Java έκδοση jdk 1.6.0_02 , jre 1.6.0_02. Για την εκτέλεση της εφαρμογής χρειάστηκαν να προστεθούν οι βιβλιοθήκες jdom.jar και jgraph.jar για το άνοιγμα, τη διαχείριση και την αποθήκευση XML αρχείων και για την απεικόνιση των γράφων δέντρων δενδρικής μορφής και των γράφων ερωτημάτων αντιστοίχως.

Τοποθετούμε σε ένα φάκελο όλες τις κλάσεις που είναι υλοποιημένες σε C++ (υλοποίηση αποτίμησης του ερωτήματος) και μεταγλωττίζουμε τον κώδικα αυτό :

g++ -o evaluation *.cpp

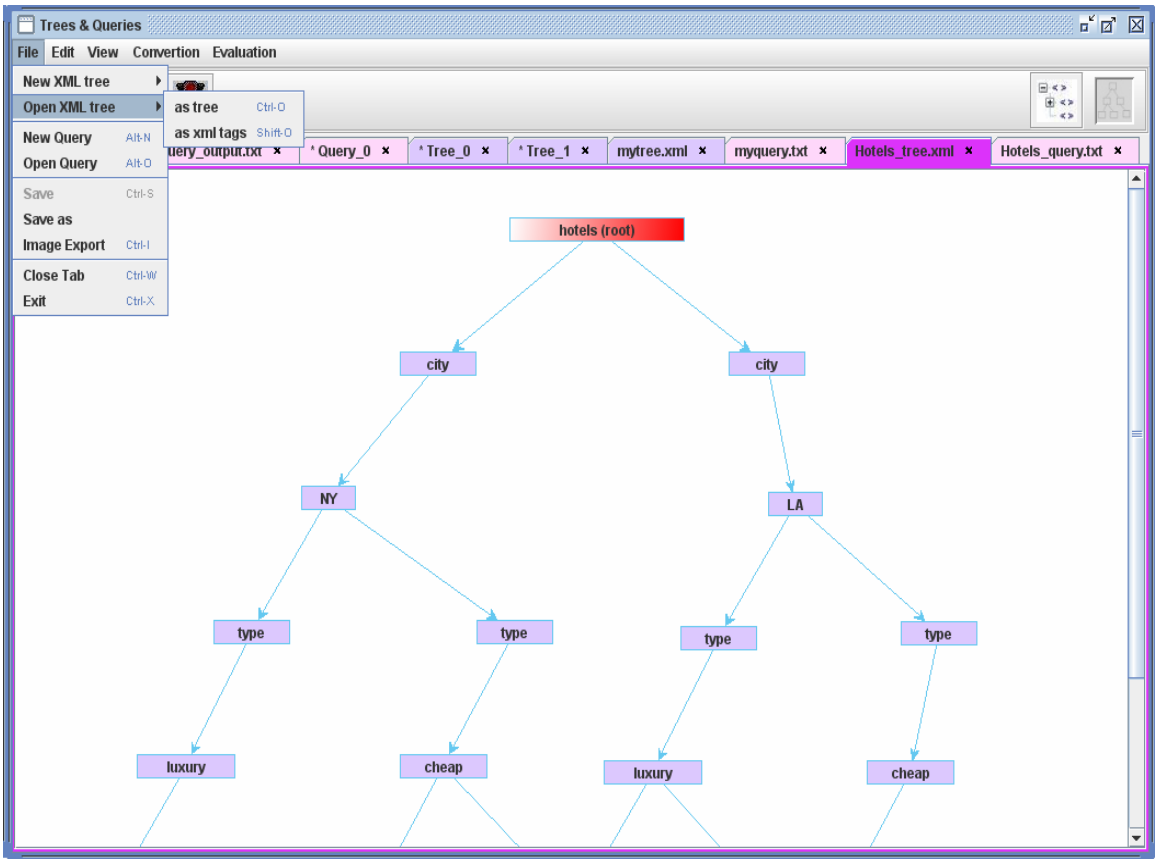
Τοποθετούμε σε ένα φάκελο όλες τις κλάσεις που είναι υλοποιημένες σε Java (υλοποίηση κύριας εφαρμογής) και μεταγλωττίζουμε τον κώδικα αυτό :

javac – classpath ./;./jgraph.jar;./jdom.jar trees_and_queries_package/Trees_and_Queries.java

Τρέχουμε την εφαρμογή μας :

java – classpath ./;./jgraph.jar;./jdom.jar trees_and_queries_package/Trees_and_Queries

Παραθέτουμε τη βασική οθόνη που περιγράφει τη διεπαφή με το χρήστη για να αποτελέσει οδηγό αναφοράς για το υπόλοιπο του παραρτήματος.



Σχήμα 1 : Το interface της εφαρμογής.

Περιγραφή Menu

Το μενού της εφαρμογής αποτελείται από 5 υπομενού που το καθένα περιέχει ένα σύνολο λειτουργιών. Αυτά είναι από αριστερά προς τα δεξιά : File, Edit, View, Conversion και Evaluation.

Το **μενού *File*** περιέχει τις ακόλουθες λειτουργίες :

- New XML tree
  - as xml tags → Shift + N
  - as tree → Ctrl + N
- Open XML tree
  - as xml tags → Shift + O
  - as tree → Ctrl + O
- New Query → Alt + N
- Open Query → Alt + O
- Save → Ctrl + S

- Save as
- Image Export → Ctrl + I
- Close Tab → Ctrl + W
- Exit → Ctrl + X

Ένα XML δέντρο μπορεί να απεικονιστεί είτε σε δενδρική μορφή είτε σε μορφή ετικετών. Γι'αυτό, οι λειτουργίες *New XML tree* και *Open XML tree* έχουν ένα υπομενού με το οποίο επιλέγεται η μορφή που θα αναπαρασταθεί το δέντρο που θέλουμε να δημιουργήσουμε ή να φορτώσουμε :

- as xml tags
- as tree

Επιλέγοντας *New XML tree* και μετά από το υπομενού επιλέξει *as xml tags*, ή πατώντας τα πλήκτρα *Shift + N*, ο χρήστης δημιουργεί μία νέα καρτέλα, στο panel της οποίας μπορεί να σχεδιάσει ένα XML δέντρο σε μορφή ετικετών. Επιλέγοντας *Open XML tree* και μετά από το υπομενού *as xml tags*, ή πατώντας τα πλήκτρα *Shift + O,* ο χρήστης φορτώνει στο περιβάλλον εργασίας ένα δέντρο μορφής ετικετών που έχει αποθηκευτεί σε αρχείο.

Επιλέγοντας *New XML tree* και μετά από το υπομενού επιλέξει *as tree*, ή πατώντας τα πλήκτρα *Ctrl + N*, ο χρήστης δημιουργεί μία νέα καρτέλα, στο panel της οποίας μπορεί να σχεδιάσει ένα XML δέντρο σε δενδρική μορφή. Επιλέγοντας *Open XML tree* και μετά από το υπομενού *as tree*, ή πατώντας τα πλήκτρα *Ctrl + O*, ο χρήστης φορτώνει στο περιβάλλον εργασίας ένα δέντρο δενδρικής μορφής που έχει αποθηκευτεί σε αρχείο.

Επιλέγοντας *New Query* ή πατώντας τα πλήκτρα *Alt + N* ο χρήστης δημιουργεί μία νέα καρτέλα, στο panel της οποίας μπορεί να σχεδιάσει το γράφο ενός ερωτήματος. Με *Open Query* ή πατώντας τα πλήκτρα *Alt + O*, επιλέγει να φορτώσει στο περιβάλλον εργασίας ένα γράφο ερωτήματος που έχει αποθηκευτεί σε αρχείο.

Την αποθήκευση σε αρχείο την πραγματοποιεί με τις λειτουργίες *Save* και *SaveAs* ή πατώντας τα πλήκτρα *Ctrl + S*.

Με την επιλογή *Image Export* ή πατώντας τα πλήκτρα *Ctrl + I* αποθηκεύει σε αρχείο εικόνας το γράφο και αφορά μόνο δέντρα δενδρικής μορφής και ερωτήματα..

Με *Close Tab* ή πατώντας τα πλήκτρα *Ctrl + W* κλείνει η τρέχουσα καρτέλα.

Τέλος με *Exit* ή πατώντας τα πλήκτρα *Ctrl + X* κλείνει η εφαρμογή.

Το **μενού *Conversion*** αφορά μόνο δέντρα, όχι ερωτήματα, και αποτελείται από δύο λειτουργίες, που μόνο μία είναι ενεργοποιημένη κάθε φορά, ανάλογα με τη μορφή που απεικονίζεται το δέντρο εκείνη τη στιγμή :

- Convert xml tags to tree
- Convert tree to xml tags

Με την επιλογή *Convert xml tags to tree* μετατρέπει ένα δέντρο μορφής ετικετών σε δενδρική μορφή.

Με την επιλογή *Convert tree to xml tags* μετατρέπει ένα δέντρο δενδρικής μορφής σε μορφή ετικετών.

Το **μενού *Evaluation*** αποτελείται από 6 λειτουργίες :

- Save Tree Data
- Save Region Encoding
- Check Path Query
- Canonical Form
- Clear Answers
- Evaluation

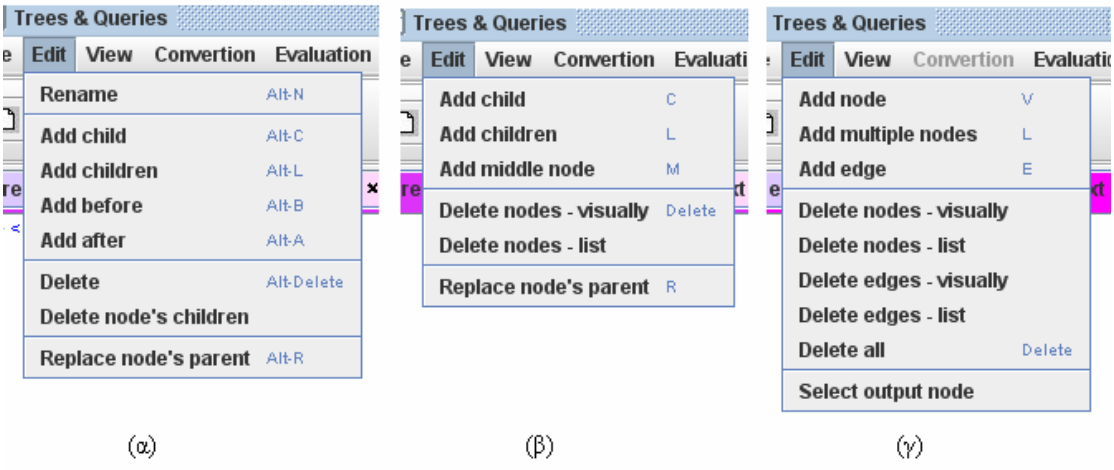
Οι λειτουργίες *Save Tree Data* και *Save Region Encoding* αφορούν μόνο XML δέντρα και αποθηκεύουν το δέντρο στη μορφή treeData και στη μορφή κωδικοποίησης θέσης αντίστοιχα.

Οι λειτουργίες *Check Path Query* και *Canonical Form* αφορούν μόνο ερωτήματα. Η $1^{η}$ ελέγχει αν το ερώτημα μονοπατιού είναι πλήρως ορισμένο ώστε να είναι δυνατή η αποτίμηση με τον αλγόριθμο PathStack. Η $2^{η}$ εντοπίζει αν υπάρχουν κόμβοι στο ερώτημα που να μην έχουν εισερχόμενες ακμές και τους συνδέει με τη ρίζα ή παιδί της και μετατρέπει το ερώτημα σε κανονική μορφή.

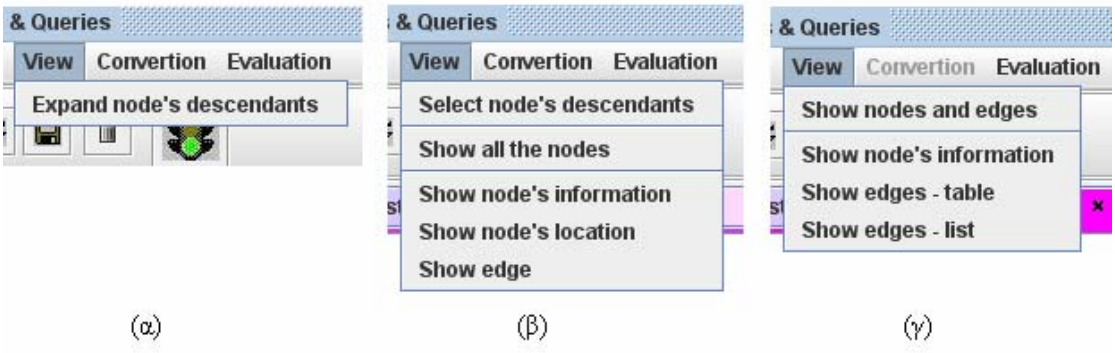
Η λειτουργία *Clear Answers* αφορά μόνο δέντρα δενδρικής μορφής από τα οποία ΄καθαρίζει΄ τους κόμβους αποτελέσματος που έχουν προέλθει από κάποια αποτίμηση.

Η λειτουργία *Evaluation* είναι η βασικότερη και κάνει την αποτίμηση ενός ερωτήματος πάνω σε ένα δέντρο.

Τα **μενού *Edit* και *View*** αλλάζουν τις λειτουργίες ανάλογα με το τί περιέχει κάθε φορά η τρέχουσα καρτέλα. Στα Σχήματα 2 και 3 φαίνονται οι λειτουργίες που διαθέτουν τα 2 μενού ανάλογα με το περιεχόμενο της καρτέλας.



Σχήμα 2 : Το μενού Edit για (α) δέντρο μορφής ετικετών, (β) δέντρο δενδρικής μορφής και (γ) ερώτημα.

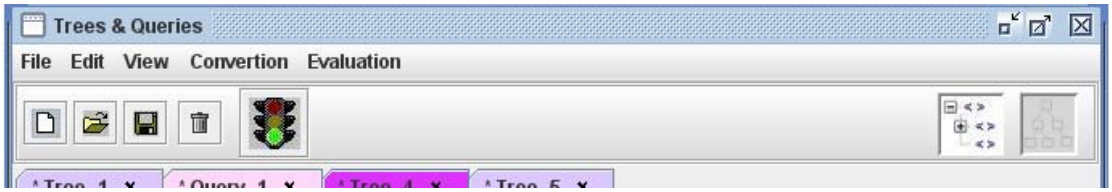


Σχήμα 3 : Το μενού View για (α) δέντρο μορφής ετικετών, (β) δέντρο δενδρικής μορφής και (γ) ερώτημα

Περιγραφή Toolbar.

Η Toolbar της εφαρμογής αποτελείται από επτά κουμπιά που υλοποιούν κάποιες από τις βασικότερες λειτουργίες. Οι λειτουργίες αυτές μπορούν να εκτελεστούν και από το μενού της εφαρμογής, απλά η toolbar προσφέρει έναν εύχρηστο τρόπο χρήσης του προγράμματος.

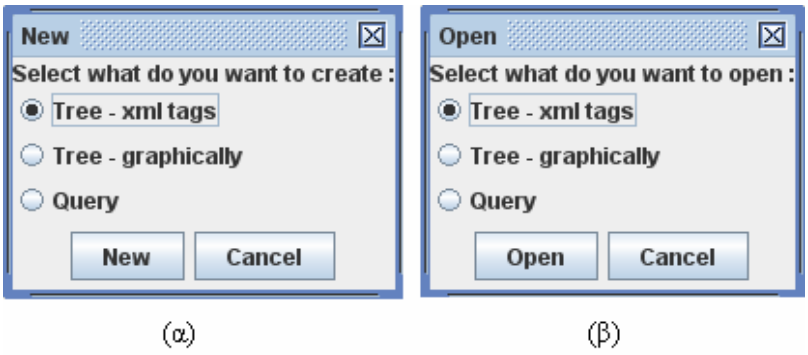
Ακολουθούν περιγραφές της εργασίας που επιτελεί το κάθε κουμπί και επιπλέον παραθέτουμε ένα σχήμα με την toolbar για την οποία μιλάμε.



Σχήμα 4 : Η toolbar της εφαρμογής.

Η πρώτη τριάδα κουμπιών αντιστοιχούν σε λειτουργίες του μενού *File*. Το πρώτο κουμπί είναι το αντίστοιχο του *New*, το δεύτερο αντιστοιχεί στο *Open* και το τρίτο στο *Save*.

Επιλέγοντας τα κουμπιά *New* και *Open* εμφανίζεται ένα παράθυρο για να επιλέξει ο χρήστης αν θέλει να δημιουργήσει ή φορτώσει αντίστοιχα ένα δέντρο μορφής ετικετών ή ένα δέντρο δενδρικής μορφής ή ένα ερώτημα.



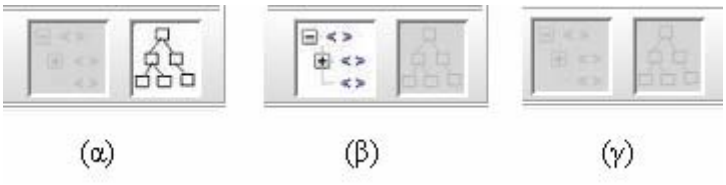
Σχήμα 5 : Επιλογή δέντρου ή ερωτήματος που (α) θα δημιουργηθεί ή (β) θα φορτωθεί από αρχείο.

Με το κουμπί *Save* σώζονται οι αλλαγές που έχουν γίνει στο γράφο που περιέχει η τρέχουσα καρτέλα. Αν δεν έχει γίνει κάποια αλλαγή τότε το κουμπί αυτό είναι απενεργοποιημένο.

Το τέταρτο κουμπί αντιστοιχεί στη λειτουργία *Delete* του μενού Edit. Αν πρόκειται για δέντρο μορφής ετικετών διαγράφονται οι επιλεγμένοι κόμβοι με τις συνδεδεμένες ακμές τους. Αν πρόκειται για δέντρο δενδρικής μορφής ζητείται από το χρήστη να επιλέξει αν θέλει να διαγράψει και τους απογόνους των επιλεγμένων κόμβων ή να συνδεθούν οι απόγονοι με τον παππού τους και μετά διαγράφονται όλοι οι επιλεγμένοι κόμβοι μαζί ή χωρίς αντιστοίχως τους απογόνους τους. Αν πρόκειται για ερώτημα διαγράφονται όλα τα στοιχεία που είναι επιλεγμένα στο γράφο, κόμβοι και ακμές, καθώς και οι ακμές που είναι συνδεδεμένες στους επιλεγμένους κόμβους.

Το πέμπτο κουμπί αντιστοιχεί στη λειτουργία *Evaluation*.

Τα τελευταία δύο κουμπιά αντιστοιχούν στις δύο λειτουργίες του μενού *Conversion*. Αν έχουμε δέντρο μόνο το ένα κουμπί είναι ενεργοποιημένο ανάλογα με τη μορφή που έχει εκείνη τη στιγμή, ενώ αν έχουμε ερώτημα κανένα από τα δύο κουμπιά δεν είναι ενεργοποιημένο.

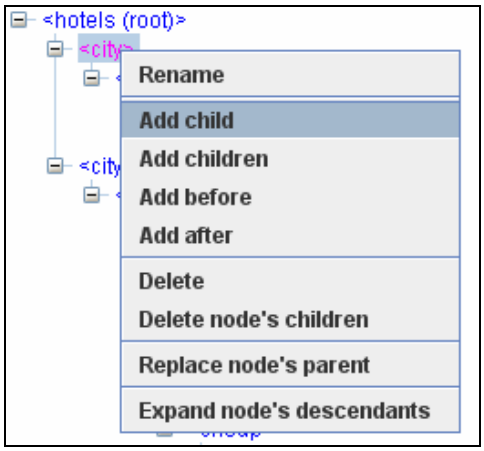


Σχήμα 6 : Τα κουμπιά μετατροπής μορφής δέντρου (α) για δέντρο μορφής ετικετών, (β) για δέντρο δενδρικής μορφής και (γ) για ερώτημα.

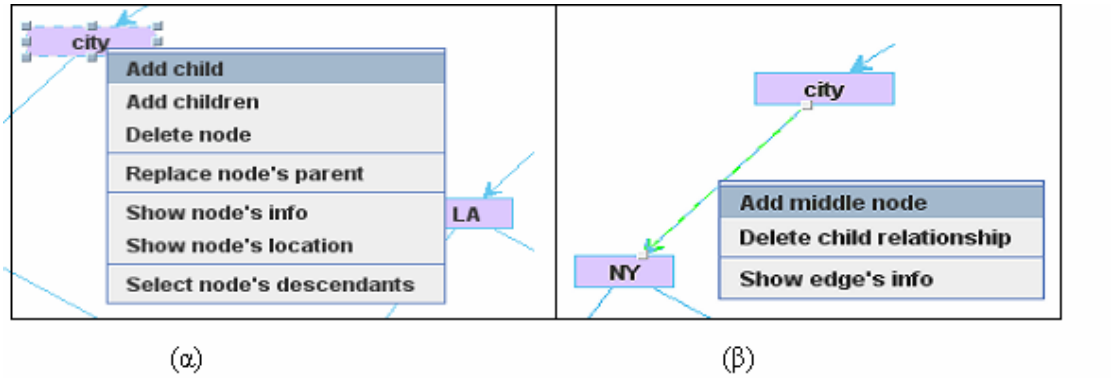
Αν η τρέχουσα καρτέλα περιέχει δέντρο μορφής ετικετών τότε είναι ενεργοποιημένο το κουμπί για μετατροπή από μορφή ετικετών σε δενδρική μορφή. Αν η τρέχουσα καρτέλα περιέχει δέντρο δενδρικής μορφής τότε είναι ενεργοποιημένο το κουμπί για μετατροπή από δενδρική μορφή σε μορφή ετικετών. Αν η τρέχουσα καρτέλα περιέχει ερώτημα και τα δύο κουμπιά είναι απενεργοποιημένα.

Περιγραφή Popup Menu

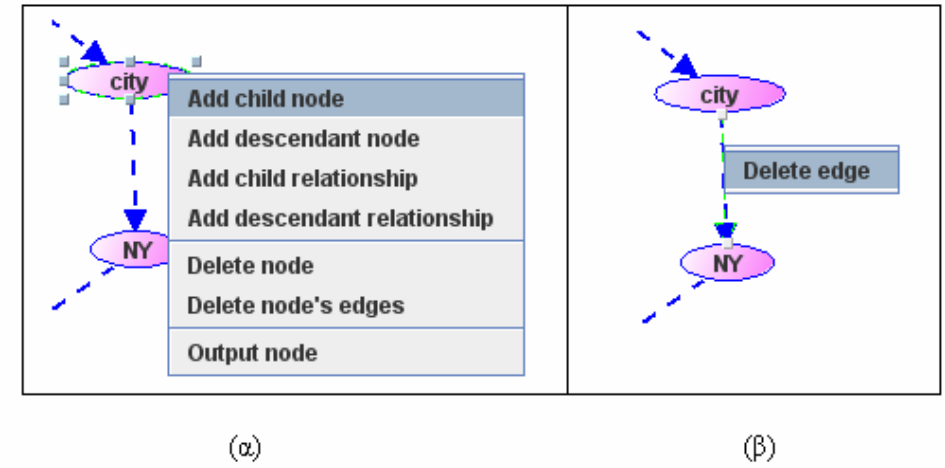
Τα popup μενού εμφανίζονται με το δεξί κλικ του ποντικιού. Στα επόμενα 3 σχήματα φαίνονται οι λειτουργίες που διαθέτουν τα popup μενού ανάλογα με το περιεχόμενο της καρτέλας και από το αντικείμενο που είναι επιλεγμένο.



Σχήμα 7 : Το popup menu για δέντρο μορφής ετικετών.



Σχήμα 8 : Το popup menu για δέντρο δενδρικής μορφής (α) όταν επιλέγεται κόμβος, (β) όταν επιλέγεται ακμή.



Σχήμα 9 : Το popup menu για ερώτημα (α) όταν επιλέγεται κόμβος, (β) όταν επιλέγεται ακμή.

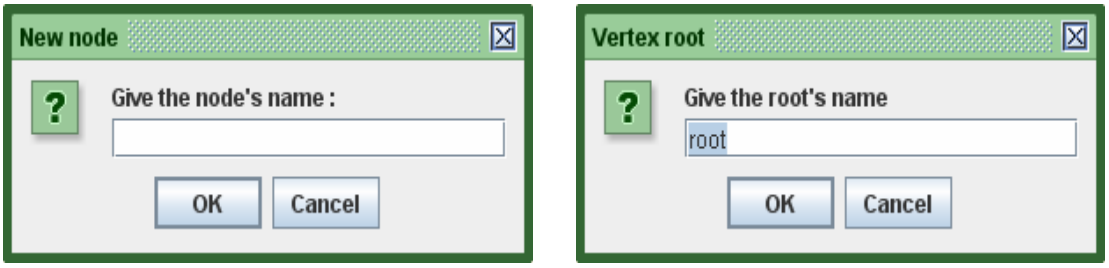
Περιγραφή Βασικών Λειτουργιών.

Προσθήκη ενός κόμβου.

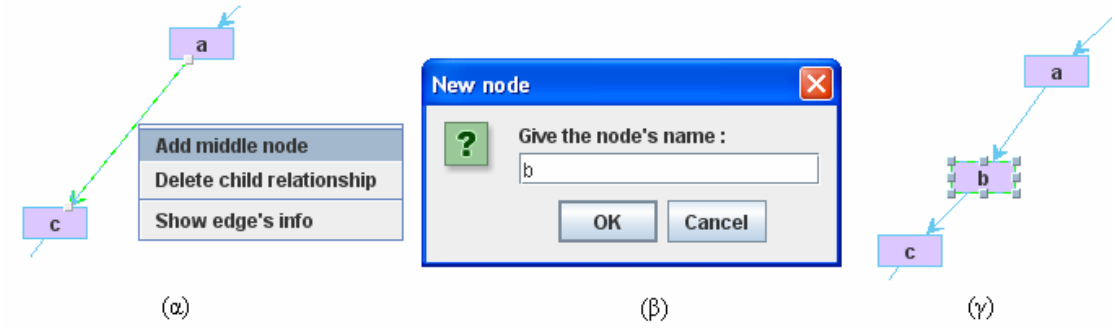
Η προσθήκη ενός κόμβου μπορεί να γίνει από το μενού Edit με τις επιλογές *Add child, Add before* και *Add after* ή πατώντας τα πλήκτρα *Alt + C, Alt + B, Alt + A* για δέντρο μορφής ετικετών, με τις επιλογές *Add child* και *Add middle node* ή πατώντας το πλήκτρο *C* ή *M* για δέντρο δενδρικής μορφής και με την επιλογή *Add node* ή πατώντας το πλήκτρο *V* για ερώτημα. Θα εμφανιστεί ένας διάλογος σαν αυτόν που φαίνεται στο Σχήμα 10(α), που θα επιτρέπει στο χρήστη να εισάγει το όνομα του κόμβου. Στο Σχήμα 11 παρουσιάζεται ένα παράδειγμα προσθήκης ενός ενδιάμεσου κόμβου σε δέντρο δενδρικής μορφής.

Ακόμα έχουμε προσθήκη κόμβου όταν δημιουργείται ένα νέο δέντρο ή ερώτημα, που μπορεί να γίνει από το μενού File με την επιλογή *New XML tree* ή *New Query* ή από το κουμπί *New*

που υπάρχει στο toolbar. Αν πρόκειται για την προσθήκη της ρίζας ενός δέντρου ή ενός ερωτήματος εμφανίζεται ο διάλογος του Σχήματος 10(β) που έχει σαν προεπιλεγμένο όνομα το 'root'.



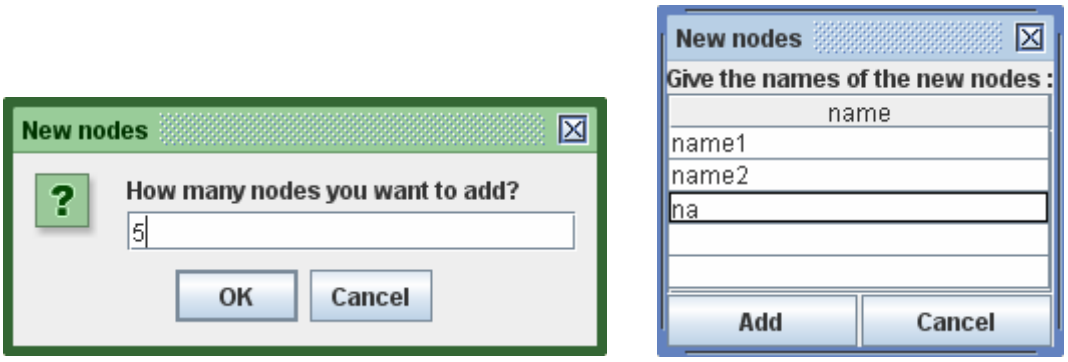
Σχήμα 10 : Προσθήκη νέου κόμβου (α) κοινός κόμβος, (β) ρίζα.



Σχήμα 11 : Προσθήκη ενδιάμεσου κόμβου σε δέντρο δενδρικής μορφής. (α) Αίτημα προσθήκης ενδιάμεσου κόμβου. (β) Όνομα νέου κόμβου (γ) Δημιουργία νέου ενδιάμεσου κόμβου με πηγή τον πατέρα της επιλεγμένης ακμής και προορισμό το παιδί της επιλεγμένης ακμής.

Προσθήκη πολλών κόμβων

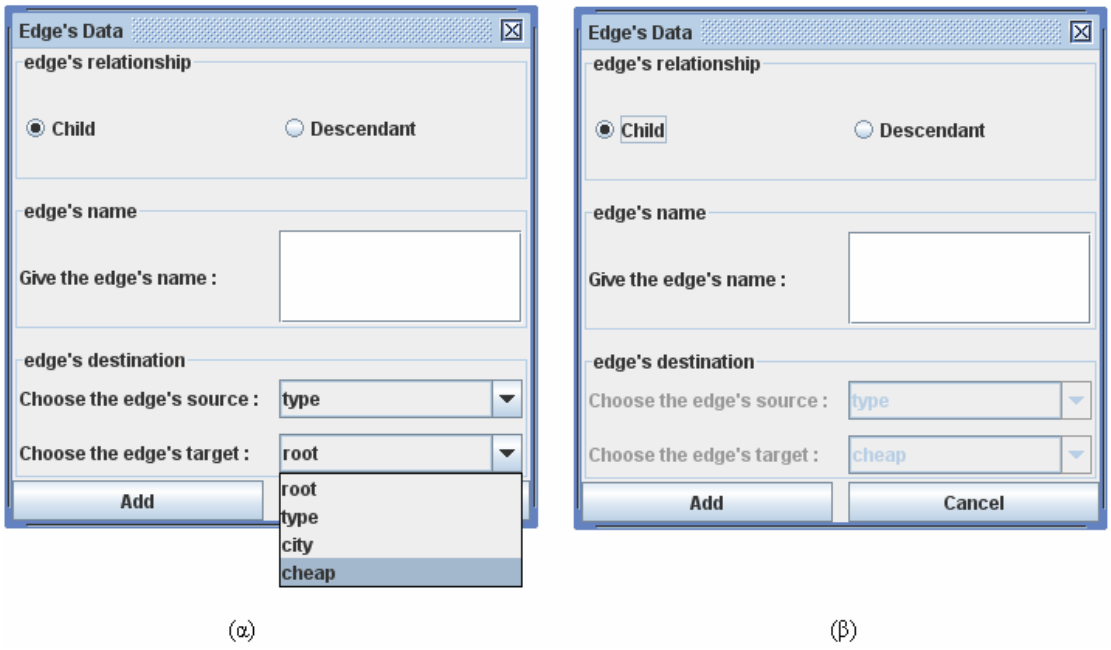
Η προσθήκη πολλών κόμβων μπορεί να γίνει από το μενού Edit με την επιλογή *Add children* ή πατώντας τα πλήκτρα *Alt + L* για δέντρο μορφής ετικετών, με την επιλογή *Add children* ή πατώντας το πλήκτρο *L* για δέντρο δενδρικής μορφής και με την επιλογή *Add multiple nodes* ή πατώντας το πλήκτρο *L* για ερώτημα. Θα εμφανιστεί ένας διάλογος σαν αυτόν που φαίνεται στο Σχήμα 12(α), που θα επιτρέπει στο χρήστη να εισάγει έναν ακέραιο που θα δηλώνει το πλήθος των νέων κόμβων που θέλει να προσθέσει. Αν εισάγει τον αριθμό 1 τότε θα εμφανιστεί ο διάλογος του Σχήματος 10(α). Αν εισάγει ακέραιο μεγαλύτερο του 1, τότε εμφανίζεται ένας πίνακας όπως του Σχήματος 12(β) με γραμμές όσες ο ακέραιος που δόθηκε όπου θα εισάγει τα ονόματα των νέων κόμβων.



Σχήμα 12 : Προσθήκη πολλών κόμβων (α) ορισμός πλήθους νέων κόμβων, (β) ορισμός ονομάτων των νέων κόμβων. Σε αυτό το παράδειγμα ο χρήστης έχει δώσει τον αριθμό 5, οπότε ο πίνακας των ονομάτων έχει 5 γραμμές.

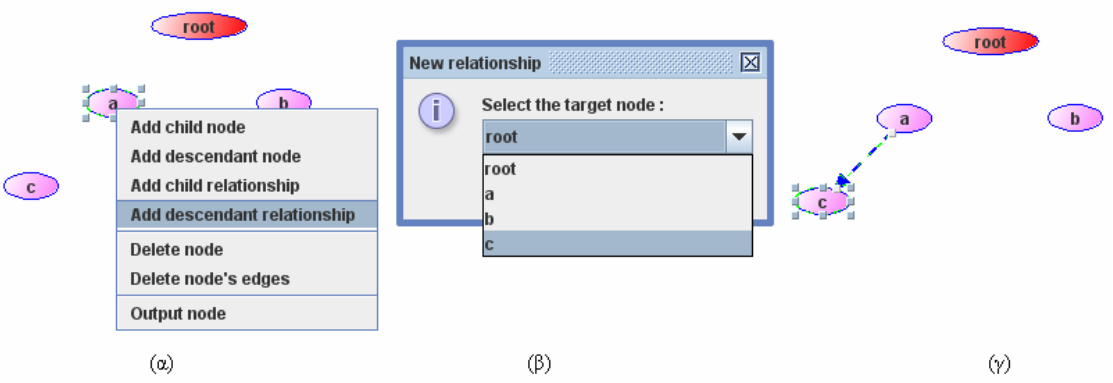
Προσθήκη ακμής ερωτήματος

Η προσθήκη ακμής ερωτήματος μπορεί να γίνει από το μενού Edit με την επιλογή *Add edge* ή πατώντας το πλήκτρο *E*. Θα εμφανιστεί ένας διάλογος σαν αυτόν που φαίνεται στο Σχήμα 13, όπου ο χρήστης θα επιλέγει το είδος της σχέσης, θα δίνει προαιρετικά ένα όνομα στην ακμή και θα ορίζει τον κόμβο-πηγή και τον κόμβο προορισμού. Οι δύο αυτοί κόμβοι ορίζονται είτε από τα δύο comboBoxes (Σχήμα 13α) ή από την επιλογή δύο κόμβων στο γράφο (Σχήμα 13β).



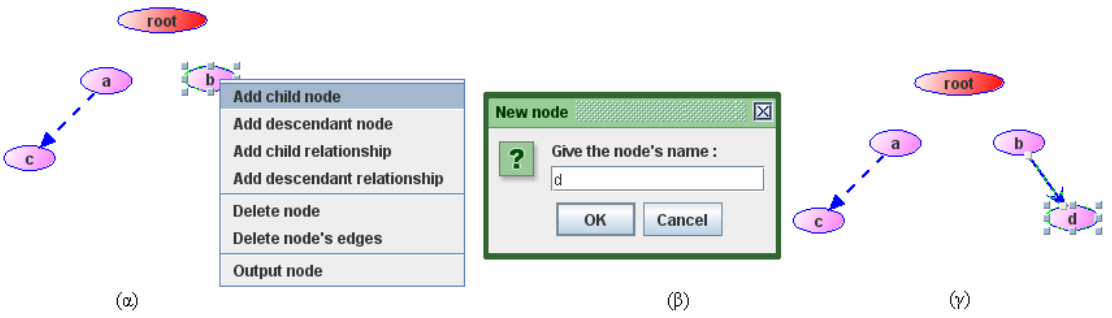
Σχήμα 13 : Προσθήκη ακμής ερωτήματος. (α) Οι κόμβοι πηγής και προορισμού επιλέγονται από τα δύο comboBoxes του παραθύρου. (β) Οι κόμβοι πηγής και προορισμού έχουν ήδη οριστεί από την επιλογή δύο κόμβων στο γράφο.

Επίσης, υπάρχει η δυνατότητα να δημιουργηθεί μια ακμή και από τις επιλογές *Add child relationship* και *Add descendant relationship* από το popup μενού στην περίπτωση που είναι επιλεγμένος ένας κόμβος. Έτσι αφού επιλεχθεί από μια λίστα ως κόμβος προορισμού ένας από τους υπάρχοντες κόμβους δημιουργείται μια νέα ακμή με πηγή τον επιλεγμένο στο γράφο κόμβο και προορισμό τον επιλεγμένο κόμβο από τη λίστα. Στο Σχήμα 14 φαίνεται ένα παράδειγμα.



Σχήμα 14 : Add descendant relationship. (α) Επιλέγεται στο γράφο ο κόμβος a και ζητείται να γίνει η λειτουργία Add descendant relationship. (β) Επιλέγεται ο κόμβος c από τη λίστα όπου υπάρχουν όλοι οι κόμβοι του ερωτήματος. Δημιουργείται μια σχέση προγόνου-απογόνου μεταξύ του επιλεγμένου στο γράφο κόμβου a και του επιλεγμένου στη λίστα κόμβου c.

Ακόμα υπάρχει η δυνατότητα ***προσθήκης κόμβου και ακμής ταυτόχρονα*** σε ένα ερώτημα με τις επιλογές *Add child node* και *Add descendant node* από το popup μενού στην περίπτωση που είναι επιλεγμένος ένας κόμβος. Έτσι αφού δοθεί το όνομα από το χρήστη δημιουργείται ο νέος κόμβος και μια ακμή με πηγή τον επιλεγμένο κόμβο και προορισμό τον κόμβο που μόλις δημιουργήθηκε. Στο Σχήμα 15 φαίνεται ένα παράδειγμα.

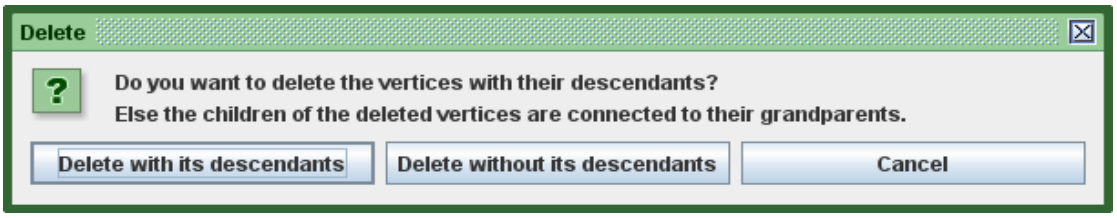


Σχήμα 15 : Add child node. (α) Επιλέγεται στο γράφο ο κόμβος b και ζητείται να γίνει η λειτουργία Add child node. (β) Δίνεται το όνομα d από τον χρήστη (γ) και δημιουργείται ο κόμβος d. Δημιουργείται μια σχέση πατέρα-παιδιού μεταξύ του επιλεγμένου στο γράφο κόμβου b και του μόλις δημιουργημένου κόμβου d.

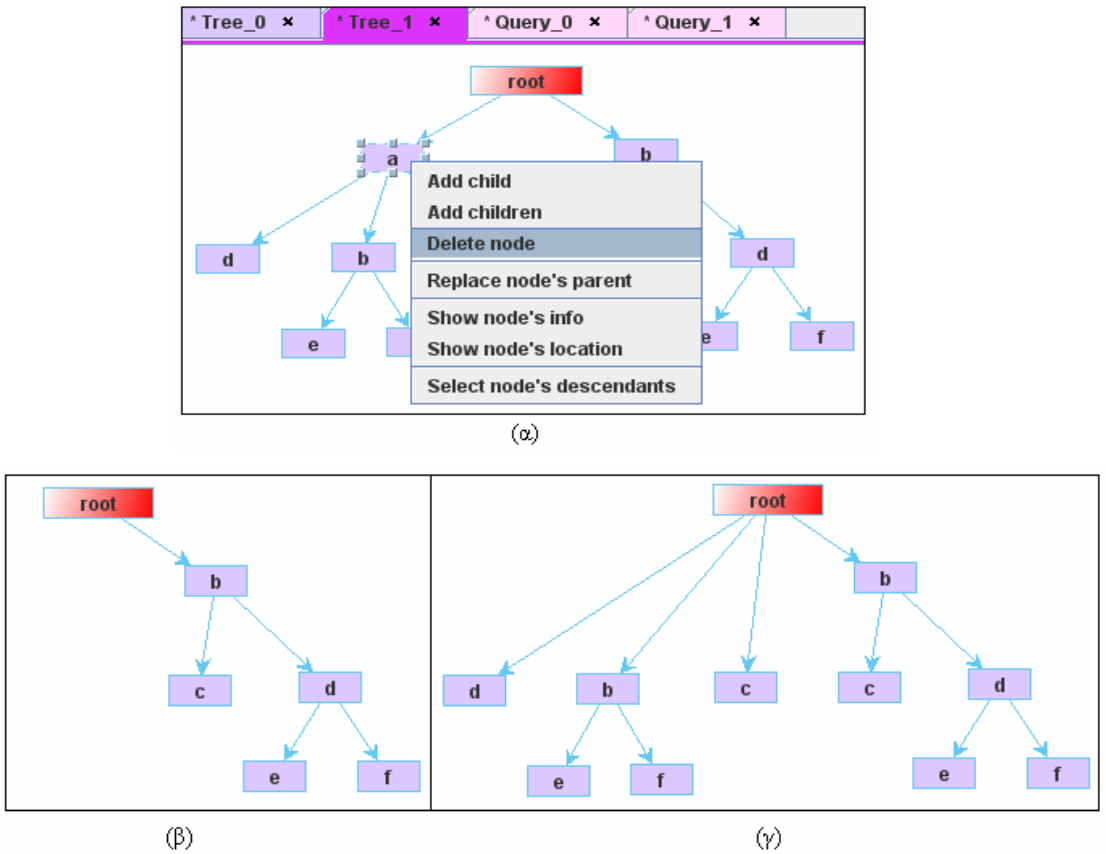
Διαγραφή κόμβων δέντρου δενδρικής μορφής

Η διαγραφή κόμβων ενός δέντρου δενδρικής μορφής μπορεί να γίνει από το μενού Edit με τις επιλογές *Delete nodes – graphically* και *Delete nodes – list* ή πατώντας το πλήκτρο *Delete*, καθώς και από το popup μενού που εμφανίζεται με το δεξί κλικ του ποντικιού με την επιλογή *Delete node* όταν επιλέγεται κόμβος και με την επιλογή *Delete child relationship* όταν επιλέγεται ακμή. Ακόμα από το *κουμπί Delete* στο toolbar.

Θα εμφανιστεί ένας διάλογος σαν αυτόν που φαίνεται στο Σχήμα 16, που θα ζητάει από το χρήστη να επιλέξει αν θέλει να διαγραφεί και το υποδέντρο του κάθε κόμβου προς διαγραφή ή αν θέλει το υποδέντρο να συνδεθεί με τον πατέρα του κάθε κόμβου που διαγράφεται. Στο Σχήμα 17 φαίνεται ένα παράδειγμα με τις 2 αυτές επιλογές.



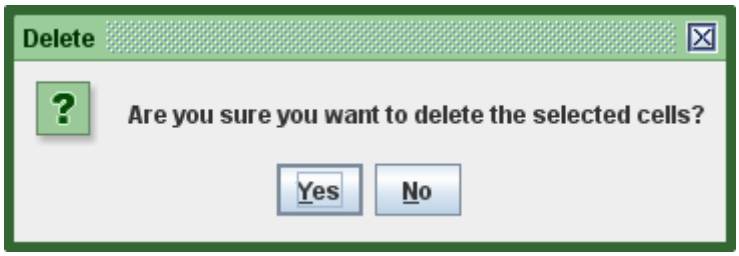
Σχήμα 16 : Διαγραφή κόμβων δέντρου δενδρικής μορφής.



Σχήμα 17 : Διαγραφή κόμβου δενδρικής μορφής. (α) Το αρχικό δέντρο πριν τη διαγραφή του κόμβου a. (β) Αφαίρεση του κόμβου a μαζί με τους απογόνους του. (β) Αφαίρεση του κόμβου a χωρίς τους απογόνους του, ενώ τα παιδιά του κόμβου a παίρνουν ως πατέρα τον παππού τους, δηλαδή οι κόμβοι b,c,d παίρνουν ως πατέρα τον root.

Διαγραφή κόμβων και ακμών ενός ερωτήματος

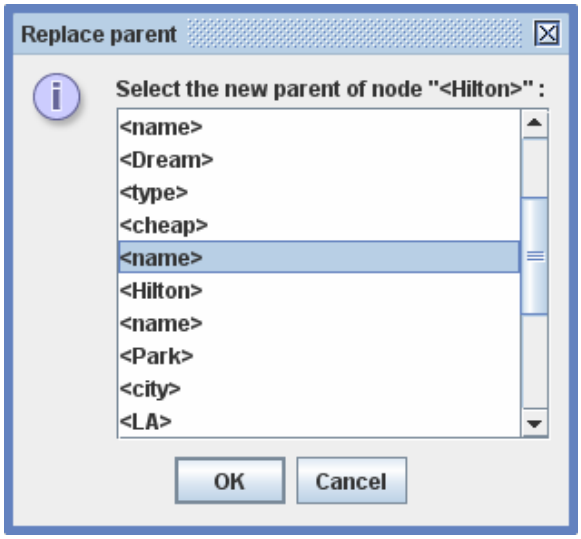
Η διαγραφή κόμβων και ακμών ενός ερωτήματος ως σύνολο μπορεί να γίνει από το μενού Edit με την επιλογή *Delete all*, από το κουμπί *Delete* που υπάρχει στο toolbar και από το πλήκτρο *Delete*. Πριν γίνει η διαγραφή ζητείται από το χρήστη η επιβεβαίωση μέσω του διαλόγου του Σχήματος 18.



Σχήμα 18 : Επιβεβαίωση διαγραφής κόμβων και ακμών ενός ερωτήματος

Αντικατάσταση πατέρα κόμβου

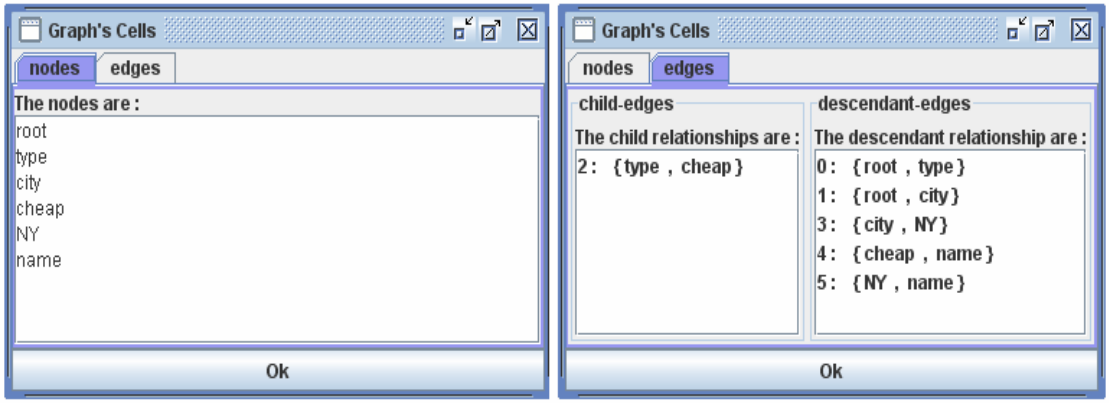
Η αντικατάσταση του πατέρα ενός κόμβου μπορεί να γίνει από το μενού Edit με την επιλογή *Replace node's parent* και πατώντας τα πλήκτρα *Alt + R* ή *R* αν πρόκειται για δέντρο μορφής ετικετών ή δενδρικής μορφής αντίστοιχα.



Σχήμα 19 : Αντικατάσταση πατέρα ενός κόμβου. Προεπιλεγμένος κόμβος είναι ο πατέρας που έχει εκείνη τη στιγμή.

Εμφάνιση κόμβων και ακμών ενός ερωτήματος

Η εμφάνιση κόμβων και ακμών ενός ερωτήματος μπορεί να γίνει από το μενού View με την επιλογή *Show nodes and edges*. Στο Σχήμα 20 παρουσιάζονται οι κόμβοι και οι ακμές για το ερώτημα του Σχήματος 25 πιο κάτω.

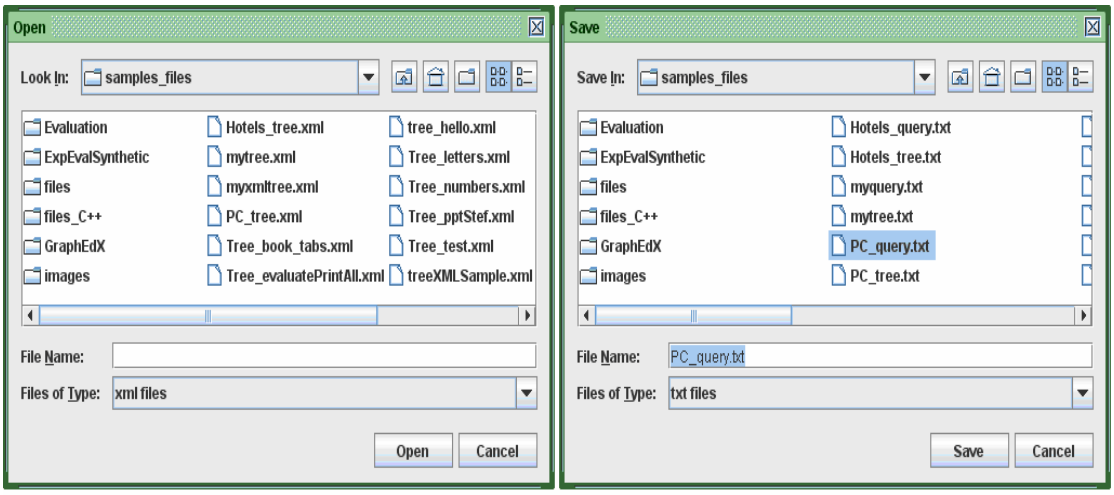


(α) (β)

Σχήμα 20 : Εμφάνιση κόμβων και ακμών ενός ερωτήματος. (α) οι κόμβοι του ερωτήματος (β) οι ακμές του ερωτήματος όπου διαχωρίζονται σε σχέσεις πατέρα-παιδιού και σχέσεις προγόνου-απογόνου.

Open - Save

Οι λειτουργίες *Open* και *Save* μπορούν να γίνουν από το μενού File ή πατώντας τα πλήκτρα που περιγράψαμε στην αρχή του παραρτήματος ή με τα αντίστοιχα κουμπιά που βρίσκονται στο toolbar. Στο Σχήμα 21 παρουσιάζονται τα παράθυρα που εμφανίζονται για τη λειτουργία Open και Save αντίστοιχα.



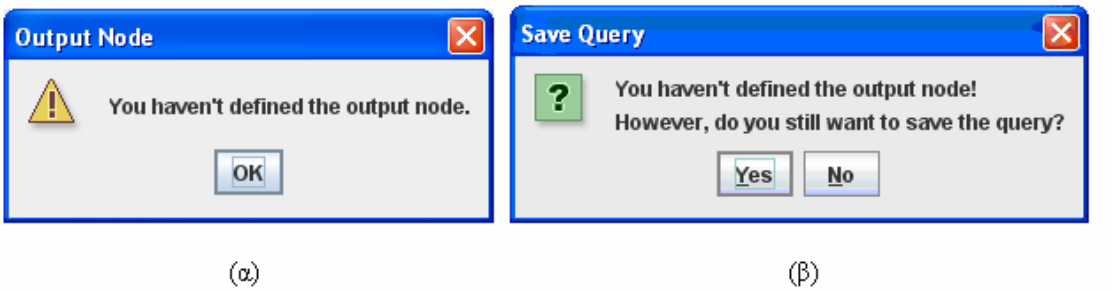
(α) (β)

Σχήμα 21 : Άνοιγμα και Αποθήκευση αρχείου. (α) Άνοιγμα αρχείου. (β) Αποθήκευση σε αρχείο. Προεπιλεγμένο αρχείο είναι αυτό που αποθήκευα μέχρι στιγμής τις αλλαγές.

Άνοιγμα – Αποθήκευση Ερωτήματος χωρίς ορισμένο κόμβο εξόδου

Αν θέλουμε να ανοίξουμε ένα txt αρχείο που περιέχει ένα ερώτημα, αλλά δεν έχει ορισμένο κόμβο εξόδου τότε το ερώτημα φορτώνεται και ενημερώνεται ο χρήστης με το μήνυμα του Σχήματος 22α.

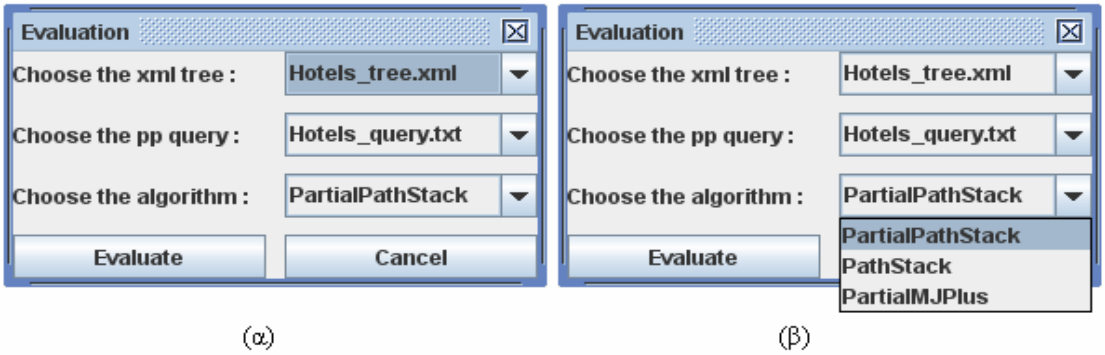
Αν θέλουμε να αποθηκεύσουμε ένα ερώτημα και δεν έχει οριστεί ο κόμβος εξόδου τότε ρωτάμε το χρήστη αν θέλει να συνεχίσει την αποθήκευση με το μήνυμα του Σχήματος 22β. Στην περίπτωση αποτίμησης, η αποθήκευση χωρίς κόμβο εξόδου απαγορεύεται.



Σχήμα 22 : Άνοιγμα και αποθήκευση ερωτήματος χωρίς κόμβο εξόδου. (α) Άνοιγμα. (β) Αποθήκευση.

Αποτίμηση

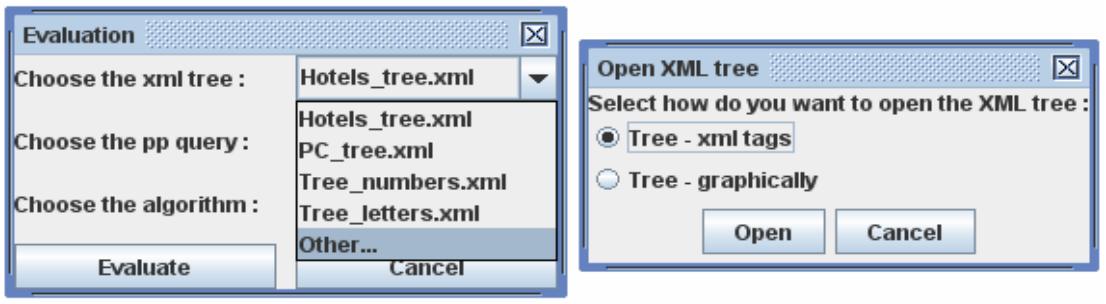
Κατά την αποτίμηση εμφανίζεται αρχικά ένας παράθυρο όπως του Σχήματος 23α όπου επιλέγει ο χρήστης το δέντρο, το ερώτημα και τον αλγόριθμο αποτίμησης.



Σχήμα 23 : Επιλογή δέντρου, ερωτήματος και αλγορίθμου αποτίμησης. (α) Έχει τρία comboBoxes από τα οποία επιλέγει αντίστοιχα το δέντρο, το ερώτημα και τον αλγόριθμο της αποτίμησης. (β) Φαίνονται οι διαθέσιμοι αλγόριθμοι αποτίμησης.

Σαν επιλογές δέντρου είναι όλα τα δέντρα, ανεξαρτήτως μορφής, που είναι ανοιχτά εκείνη τη στιγμή στην εφαρμογή. Ακόμα υπάρχει και η *επιλογή "Other..."* με την οποία μπορούμε να ανοίξουμε εκείνη τη στιγμή ένα νέο δέντρο. Τότε εμφανίζεται ένα παράθυρο για να επιλέξουμε σε ποια μορφή θέλουμε να εμφανιστεί το δέντρο που θα ανοίξουμε (Σχήμα 24β)

και μετά εμφανίζεται το παράθυρο από όπου θα επιλέξουμε το αρχείο που θα φορτώσουμε (Σχήμα 21α). Παρόμοια διαδικασία ακολουθείται και για τα ερωτήματα.



(α) (β)

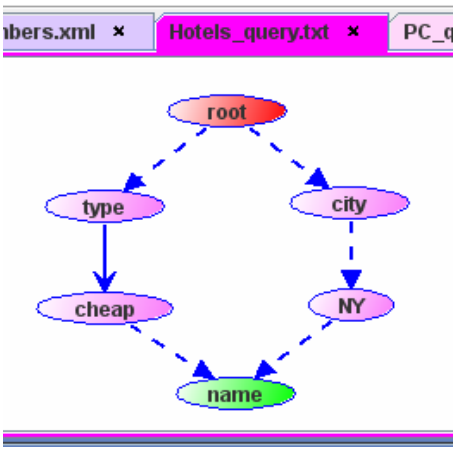
Σχήμα 24 : Άνοιγμα νέου δέντρου κατά την αποτίμηση. (α) Επιλογή φορτώματος νέου δέντρου κατά την αποτίμηση. (β) Επιλογή μορφής δέντρου που θα απεικονιστεί το δέντρο που θα ανοίξουμε.

Μόλις ολοκληρωθεί η αποτίμηση και υπάρχουν αποτελέσματα, τότε δημιουργείται ένα νέο δέντρο-αποτέλεσμα μορφής ίδιας με αυτή που έχει το δέντρο πάνω στο οποίο έγινε η αποτίμηση.

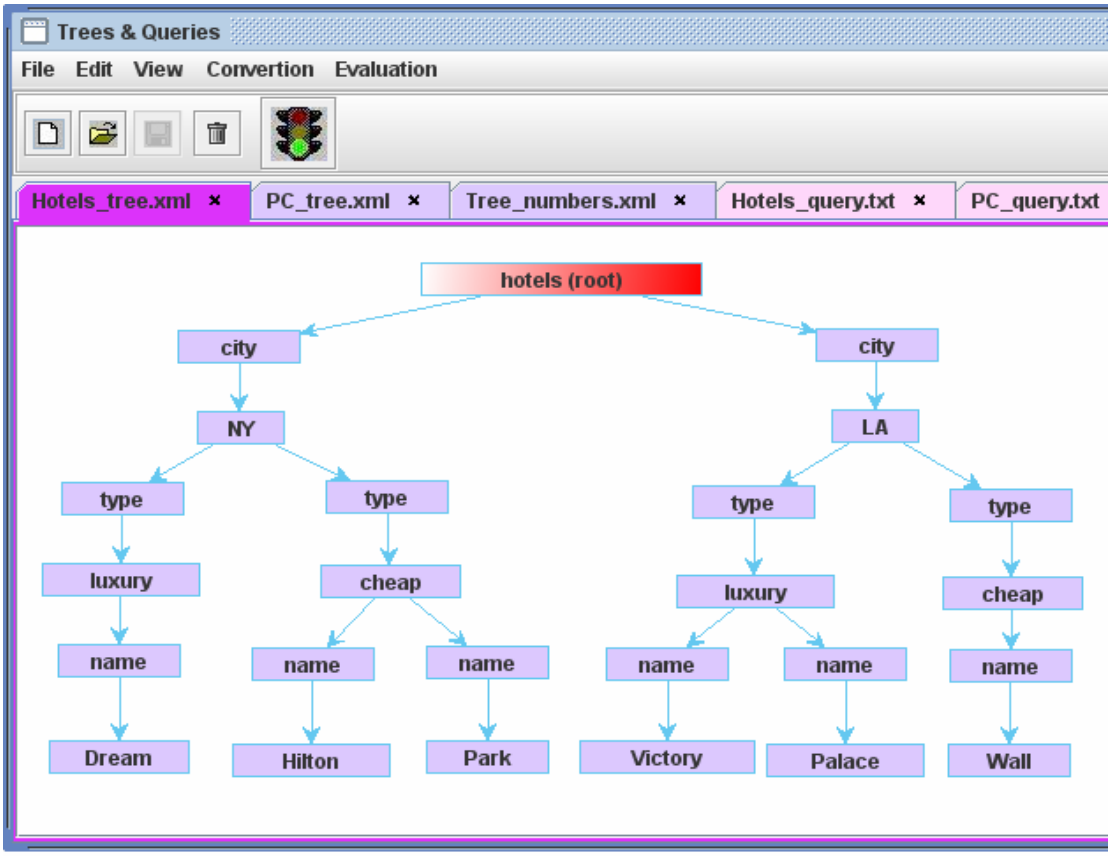
Ακολουθεί ένα παράδειγμα αποτίμησης. Στο Σχήμα 25 είναι το ερώτημα αποτίμησης.

Στο Σχήμα 26 είναι το δέντρο σε δενδρική μορφή πάνω στο οποίο θα γίνει η αποτίμηση. Σε αυτήν την περίπτωση το δέντρο αποτέλεσμα είναι και αυτό σε δενδρική μορφή. Επίσης πάνω στο αρχικό δέντρο σημειώνονται με πράσινο και οι κόμβοι αποτελέσματος, οι οποίοι μπορούν να επανέλθουν στην αρχική τους κατάσταση με την επιλογή *Clear Answers* από το μενού Evaluation.

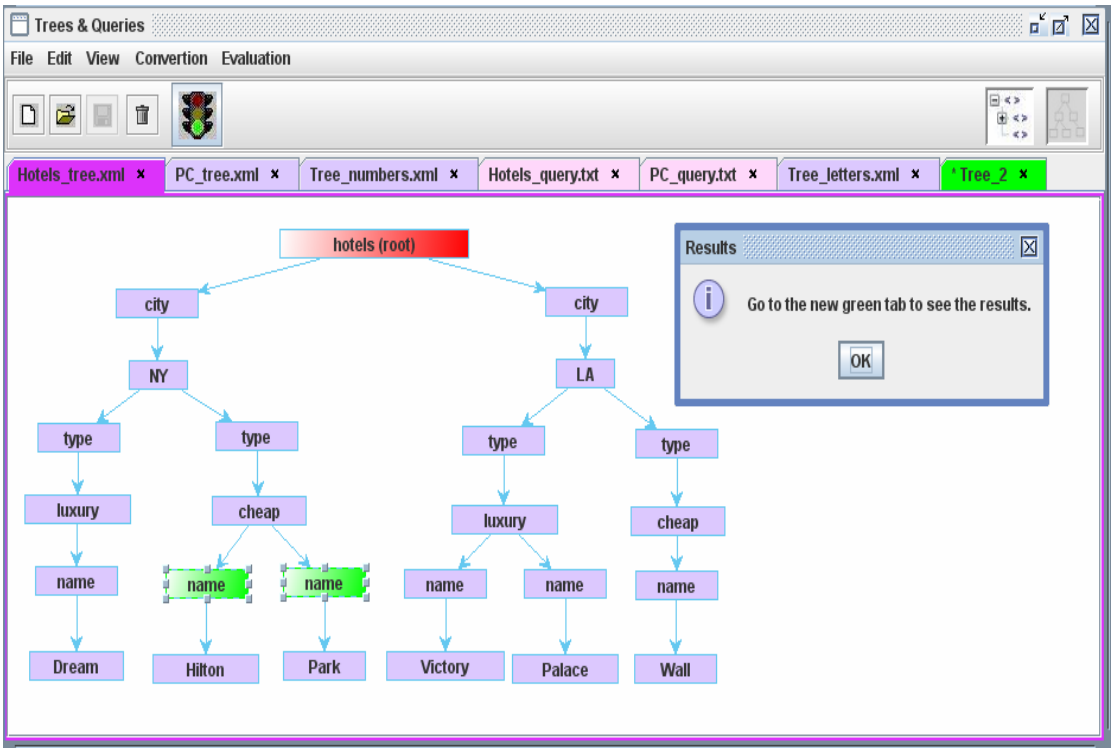
Στο Σχήμα 27 είναι το ίδιο δέντρο με το Σχήμα 26 αλλά σε μορφή ετικετών. Σε αυτήν την περίπτωση το δέντρο αποτέλεσμα είναι και αυτό σε μορφή ετικετών.



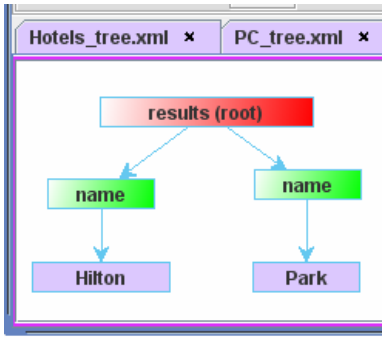
Σχήμα 25 : Ερώτημα αποτίμησης.


Trees & Queries
File Edit View Convertion Evaluation
Hotels_tree.xml
PC_tree.xml
Tree_numbers.xml
Hotels_query.txt
PC_query.txt
hotels (root)
city
city
NY
LA
type
type
type
type
luxury
cheap
luxury
cheap
name
name
name
name
name
name
Dream
Hilton
Park
Victory
Palace
Wall

(α)

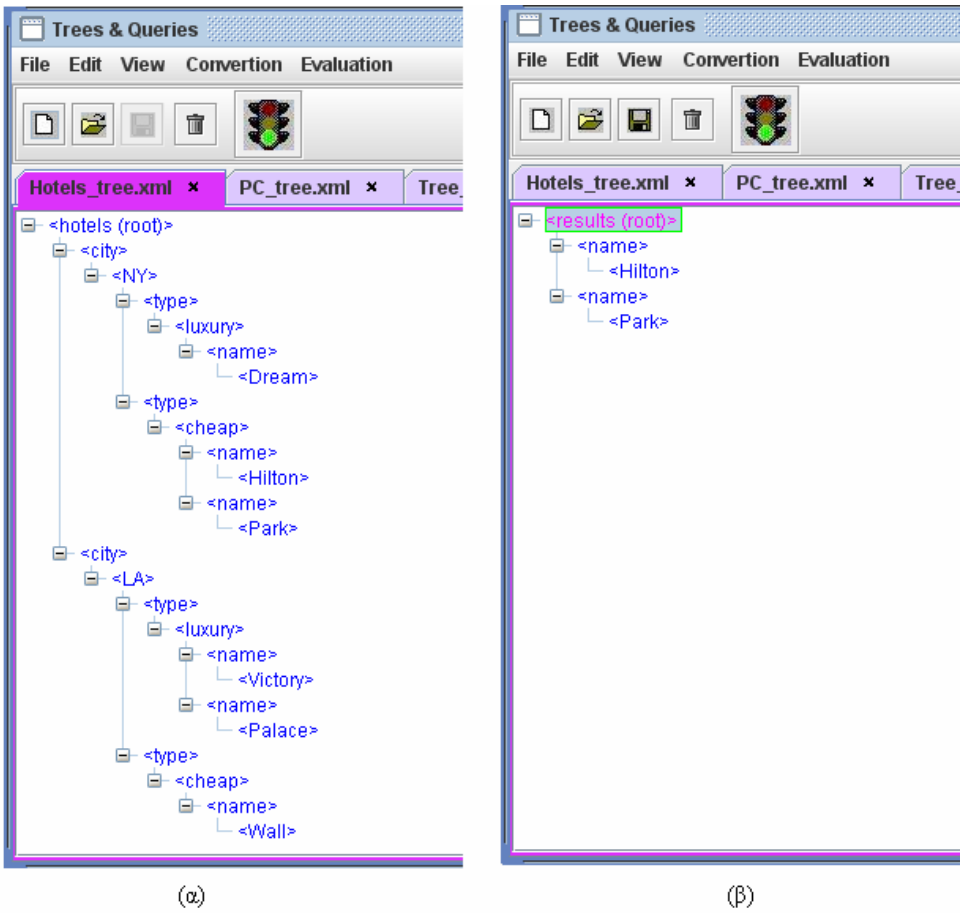

Trees & Queries
File Edit View Convertion Evaluation
Hotels_tree.xml
PC_tree.xml
Tree_numbers.xml
Hotels_query.txt
PC_query.txt
Tree_letters.xml
Tree_2
hotels (root)
Results
Go to the new green tab to see the results.
OK
city
city
NY
LA
type
type
type
type
luxury
cheap
luxury
cheap
name
name
name
name
name
name
Dream
Hilton
Park
Victory
Palace
Wall

(β)



(γ)

Σχήμα 26 : Δέντρα αποτίμησης σε δενδρική μορφή. (α) Δέντρο δενδρικής μορφής πάνω στο οποίο θα αποτιμηθεί το ερώτημα του Σχήματος 25. (β) Το αρχικό δέντρο με σημειωμένους τους κόμβους αποτελέσματος με πράσινο. (γ) Το δέντρο-αποτέλεσμα που είναι και αυτό δενδρικής μορφής.



(α) (β)

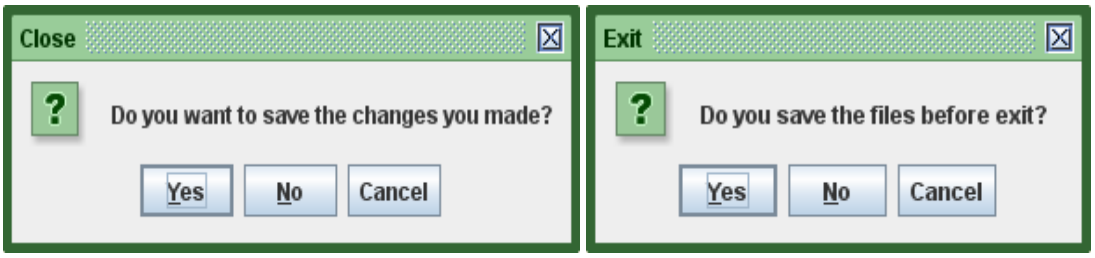
Σχήμα 27 : Δέντρα αποτίμησης σε μορφή ετικετών. (α) Δέντρο μορφής ετικετών πάνω στο οποίο θα αποτιμηθεί το ερώτημα του Σχήματος 25. (β) Το δέντρο-αποτέλεσμα που είναι και αυτό μορφής ετικετών.

Κλείσιμο καρτέλας

Πριν το κλείσιμο μιας καρτέλας, που γίνεται είτε από το μενού File με την επιλογή *Close Tab* είτε πατώντας τα πλήκτρα *Ctrl + W,* εμφανίζεται στο χρήστη ο διάλογος του Σχήματος 28α όπου ρωτάται αν θέλει να αποθηκευτούν οι αλλαγές που έχουν γίνει στο περιεχόμενο της καρτέλας.

Κλείσιμο εφαρμογής

Πριν το κλείσιμο μιας εφαρμογής, που γίνεται είτε από το μενού File με την επιλογή *Exit* είτε πατώντας τα πλήκτρα *Ctrl + X,* εμφανίζεται στο χρήστη ο διάλογος του Σχήματος 28β όπου ρωτάται αν θέλει να αποθηκευτούν οι αλλαγές που έχουν γίνει για όλες τις καρτέλες.

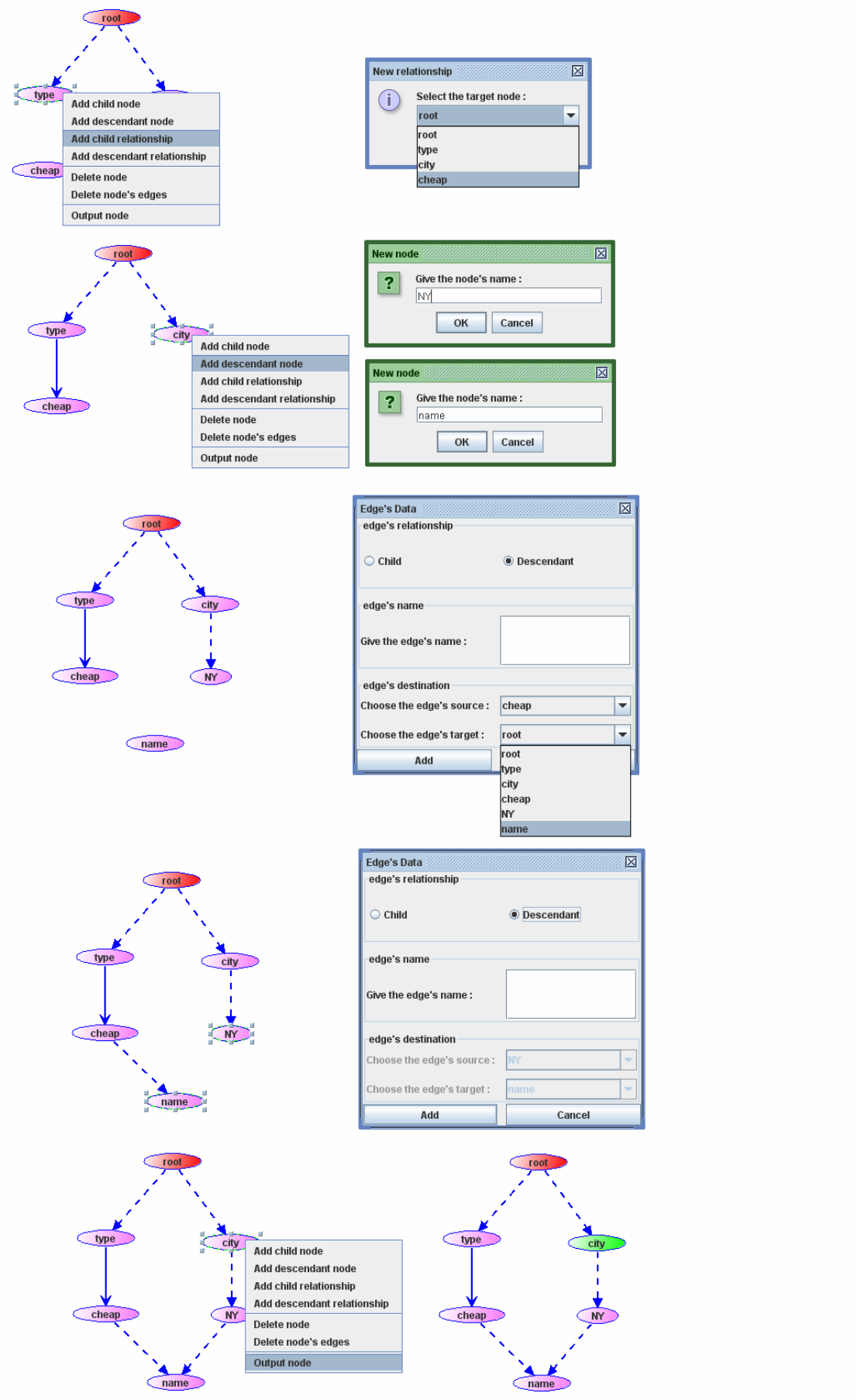


Σχήμα 28 : (α) Κλείσιμο καρτέλας, (β) Κλείσιμο εφαρμογής.

Κατασκευάζοντας ένα ερώτημα...

Στην παράγραφο αυτή δίνεται ένα παράδειγμα κατασκευής ενός ερωτήματος για να δούμε κάποιες βασικές του λειτουργίες.

Στο Σχήμα 30 παρουσιάζεται ένα παράδειγμα κατασκευής ενός ερωτήματος. Στην αρχή με την επιλογή Add child relationship δημιουργείται μία σχέση πατέρα-παιδιού μεταξύ του κόμβου type που είναι επιλεγμένος στο γράφο, όπως φαίνεται στην $1^{η}$ εικόνα, και του κόμβου cheap που επιλέγεται από τη λίστα σύμφωνα με τη $2^{η}$ εικόνα. Έπειτα με την επιλογή Add descendant node δημιουργείται μια σχέση προγόνου-απογόνου μεταξύ του κόμβου city που είναι επιλεγμένος στο γράφο, όπως φαίνεται στην $3^{η}$ εικόνα και του κόμβου NY που μόλις δημιουργήθηκε, όπως φαίνεται από την $4^{η}$ εικόνα. Μετά δημιουργείται ο κόμβος με όνομα name, όπως φαίνεται στην $5^{η}$ εικόνα. Στη συνέχεια δημιουργούνται 2 ακμές με πηγή τους κόμβους cheap και NY αντιστοίχως και προορισμό τον κόμβο name. Η $1^{η}$ ακμή ορίζει τους κόμβους από τα comboBoxes, ενώ η $2^{η}$ τους ορίζει από την επιλογή τους πάνω στο γράφο, όπως φαίνονται στις αντίστοιχες εικόνες. Τέλος ορίζεται ως κόμβος εξόδου ο κόμβος city και καταλήγουμε στην τελική εικόνα.



Σχήμα 30 : Κατασκευή ενός ερωτήματος.

ΠΑΡΑΡΤΗΜΑ Β

Περιγραφή των κλάσεων της εφαρμογής

Class NameTranslation

Ελέγχει τα ονόματα των κόμβων να είναι έγκυρα XML ονόματα ετικετών και τα μετατρέπει σε κατάλληλη μορφή ανάλογα με την περίσταση.

Fields

- static NameTranslation nameTranslation

Construstors

- private NameTranslation()

Methods

- public static NameTranslation getInstance()
- public String getRootName(boolean isTree) : Ζητάει από το χρήστη να δώσει ένα όνομα για τον κόμβο της ρίζας του δέντρου ή του ερωτήματος και ελέγχει αν είναι έγκυρο XML όνομα. Αν δεν είναι έγκυρο, ζητάει συνεχώς από το χρήστη ένα νέο όνομα μέχρι να δώσει ένα έγκυρο. Επιστρέφει το έγκυρο όνομα της ρίζας.
- public String getValidName() : Ζητάει από το χρήστη να δώσει ένα όνομα για τον νέο κόμβο και ελέγχει αν είναι έγκυρο XML όνομα. Αν δεν είναι επιστρέφει null, αλλιώς επιστρέφει το έγκυρο όνομα.
- public String setIntegerName(String childName) : Ελέγχει αν το όνομα του κόμβου ξεκινάει με ακέραιο, και αν ξεκινάει τότε προσθέτουμε τον χαρακτήρα '_' για την αποθήκευση του κόμβου. Επιστρέφει το όνομα.
- public String getIntegerName(String childName) : Ελέγχει αν το όνομα του κόμβου που ανοίγουμε από αρχείο ξεκινάει με τον χαρακτήρα '_' και μετά ακολουθεί ακέραιος, και αν ναι τότε αφαιρούμε τον χαρακτήρα '_' για την αναπαράστασή του
- public boolean checkValidName(String childName) : Ελέγχει αν είναι έγκυρο XML όνομα
- public String getNodeName(String name) : Αφαιρεί τους χαρακτήρες '<' και ΄>΄ από τον κόμβο του δέντρου μορφής ετικετών για την επεξεργασία ή την αποθήκευσή του

Class Save

Διαχειρίζεται 2 θέματα. Αν υπάρχουν αλλαγές στο δέντρο ή στο ερώτημα που δεν έχουν αποθηκευτεί και αν υπάρχει έγκυρο αρχείο κωδικοποίησης θέσης ενός δέντρου.

Fields

- static Save save

Construstors

- private Save()

Methods

- public static Save getInstance()
- public void isSaved(Trees_and_Queries gui, NewTab.DataTab dataTab) : Δηλώνει ότι τα δεδομένα της καρτέλας έχουν αποθηκευτεί
- public void isUnSaved(Trees_and_Queries gui, NewTab.DataTab dataTab) : Δηλώνει ότι τα δεδομένα της καρτέλας έχουν αλλάξει και ότι δεν έχουν αποθηκευτεί οι αλλαγές.

Αποθήκευση των δομών που αφορούν την κωδικοποοίηση θέσης μετά από αποθήκευση των αλλαγών του δέντρου.

- public void storeHash_JGraph(VerticesTree_JGraph ourVertices, SaveRegionEncoding_JGraph re)
- public void storeHash_JTree(VerticesTree_JTree ourVertices, SaveRegionEncoding_JTree re)

Άδειασμα των δομών που αφορούν την κωδικοποοίηση θέσης μετά από αλλαγή στο δέντρο.

- public void clearHash_JGraph(VerticesTree_JGraph ourVertices)
- public void clearHash_JTree(VerticesTree_JGraph ourVertices)

Αρχείο κωδικοποίησης θέσης

- public File getValidRegionEncoding_JGraph(File fileTree, VerticesTree_JGraph ourVertices) : Ελέγχει αν υπάρχει το αρχείο κωδικοποίησης θέσης του δέντρου δενδρικής μορφής που είναι αποθηκευμένο στο αρχείο fileTree και αν δεν έχουν γίνει αλλαγές εν τω μεταξύ στο δέντρο. Αν ισχύουν αυτά επιστρέφει το αρχείο όπου υπάρχει αποθηκευμένη η κωδικοποίηση θέσης, αλλιώς επιστρέφει null.
- public File getValidRegionEncoding_JTree(File fileTree, VerticesTree_JTree ourVertices) : Ελέγχει αν υπάρχει το αρχείο κωδικοποίησης θέσης του δέντρου μορφής ετικετών που είναι αποθηκευμένο στο αρχείο fileTree και αν δεν έχουν γίνει αλλαγές εν τω μεταξύ στο δέντρο. Αν ισχύουν αυτά επιστρέφει το αρχείο όπου υπάρχει αποθηκευμένη η κωδικοποίηση θέσης, αλλιώς επιστρέφει null.

Class Selection

Ελέγχει τα αντικείμενα που είναι επιλεγμένα στο γράφο ενός δέντρου δενδρικής μορφής ή στο γράφο ερωτήματος.

Fields

- static Selection selection

Construstors

- private Selection()

Methods

- public static Selection getInstance()
- public DefaultGraphCell getOneSelectedVertex(JGraph graph, GraphModel model String title) : Τα ορίσματα graph και model αφορούν το γράφο που εξετάζουμε και το μοντέλο του. Το όρισμα title δίνει τον τίτλο του μηνύματος που εμφανίζεται σε περίπτωση λάθους. Ελέγχει αρχικά αν έχω επιλέξει μόνο ένα αντικείμενο στο γράφο graph και έπειτα αν το αντικείμενο αυτό είναι κόμβος. Αν ναι, επιστρέφει αυτόν τον κόμβο. Αλλιώς επιστρέφει την τιμή null.
- public DefaultEdge getOneSelectedEdge(JGraph graph, GraphModel model String title) : Τα ορίσματα graph και model αφορούν το γράφο που εξετάζουμε και το μοντέλο του. Το όρισμα title δίνει τον τίτλο του μηνύματος που εμφανίζεται σε περίπτωση λάθους. Ελέγχει αρχικά αν έχω επιλέξει μόνο ένα αντικείμενο στο γράφο graph και έπειτα αν το αντικείμενο αυτό είναι ακμή. Αν ναι, επιστρέφει αυτήν την ακμή. Αλλιώς επιστρέφει την τιμή null.

Class Trees_and_Queries implements ActionListener, ChangeListener

Είναι η βασική κλάση. Αρχικοποιεί όλα τα πεδία της εφαρμογής που διαχειρίζονται γενικά δεδομένα. Δημιουργεί το παράθυρο/διαπροσωπεία της εφαρμογής με όλες τις λειτουργίες που προδιαγράφονται.

Fields

- JFrame general_frame: το βασικό παράθυρο της εφαρμογής
- JTabbedPane tabbed_pane : το σύνολο των καρτελών της εφαρμογής

- int num_tabs : το πλήθος των καρτελών της εφαρμογής
- int num_trees : το πλήθος των δέντρων της εφαρμογής
- int num_queries : το πλήθος των ερωτημάτων της εφαρμογής
- Vector<File> streamFiles : τα αρχεία που είναι ανοιχτά στην εφαρμογή

Δομές που σχετίζονται με τις καρτέλες και τι περιέχουν αυτές.

- Hashtable<String, NewTab.DataTab> componentName2dataTab
- Hashtable<NewTab.DataTab, NewTree_JTree.DataTree> dataTab2dataTree_JTree
- Hashtable<NewTab.DataTab, NewTree_JGraph.DataTree> dataTab2dataTree_JGraph
- Hashtable<NewTab.DataTab, NewQuery.DataQuery> dataTab2dataQuery

Πεδία που σχετίζονται με το interface (μενού, κουμπιά, κτλ).

- JMenuBar – JMenu – JMeuItem – JButton – JToolBar

Πεδία που αφορούν χρώματα της εφαρμογής

- Color defaultBackgroundColor, resultBackgroundColor;
- Color treeBackgroundColor, queryBackgroundColor;
- Color treeSelectedBackgroundColor, querySelectedBackgroundColor;

```
enum Application{Tree_jtree, Tree_jgraph, Query}
enum Create{New, Open}
```

Construstor

- Trees_and_Queries()

Methods

- public static void main(String[] args) : Εκκινεί το πρόγραμμα

Δημιουργία toolbar και κουμπιών

- public void createToolBar() : Δημιουργεί την toolbar της εφαρμογής
- protected void addButtons(JToolBar toolbar) : Προσθέτει στην toolbar τα κουμπιά
- public void addButtons_convertTrees(JToolar toolbar) : Προσθέτει στην toolbar τα κουμπιά της μετατροπής ενός δέντρου

Δημιουργία των μενού και ενεργοποίησή τους

- public JMenuBar JMenuBar_panel() : Δημιουργεί το menu επιλογών
- public void JMenuItems_File() : Δημιουργεί το menu File
- public void JMenuItems_Conversion() : Δημιουργεί το menu Conversion
- public void JMenuItems_Evaluation() : Δημιουργεί το menu Evaluation
- public void JMenuItems_JTree() : Δημιουργεί τις επιλογές των δέντρων μορφής ετικετών
- public void JMenuItems_JGraph() : Δημιουργεί τις επιλογές των δέντρων δενδρικής μορφής
- public void JMenuItems_Query() : Δημιουργεί τις επιλογές των ερωτημάτων
- public void enableMenus(Application application) : Ενεργοποιεί τα κατάλληλα menu
- public void menuEdit(Application application) : Δημιουργεί το menu Edit
- public void menuView(Application application) : Δημιουργεί το menu View
- public void enableMenuTree_JTree(boolean is_enabled) : Ενεργοποιεί τις επιλογές για τα δέντρα μορφής ετικετών
- public void enableMenuTree_JGraph(boolean is_enabled) : Ενεργοποιεί τις επιλογές για τα δέντρα δενδρικής μορφής
- public void enableMenuQuery(boolean is_enabled) : Ενεργοποιεί τις επιλογές για τα ερωτήματα

- public void selectApplication(Create create) : Επιλέγει τι είδος (δέντρο μορφής ετικετών ή δενδρικής μορφής) να δημιουργήσει ή να ανοίξει.

- public void actionPerformed(ActionEvent evt) : Καλείται όταν ο χρήστης εκτελέσει κάποια ενέργεια και αυτή με τη σειρά της καλεί την απαραίτητη μέθοδο.

- public void stateChanged(ChangeEvent evt) : Αλλάζει το μενού επιλογών κάθε φορά που αλλάζει η καρτέλα στο προσκήνιο.

Δημιουργία - Άνοιγμα

- public void newTree_JTree() : Δημιουργεί ένα δέντρο μορφής ετικετών
- public NewTab openTree_JTree() : Ανοίγει ένα δέντρο μορφής ετικετών. Επιστρέφει τα δεδομένα του δέντρου.
- public void newTree_JGraph() : Δημιουργεί ένα δέντρο δενδρικής μορφής
- public NewTab openTree_JGraph() : Ανοίγει ένα δέντρο δενδρικής μορφής. Επιστρέφει τα δεδομένα του δέντρου.
- public void newQuery () : Δημιουργεί ένα ερώτημα
- public NewTab openQuery () : Ανοίγει ένα ερώτημα. Επιστρέφει τα δεδομένα του ερωτήματος

Αποθήκευση

- public void save(NewTab.DataTab dataSaveTab) : Αποθηκεύει τα δεδομένα στο τρέχον αρχείο.
- public void saveAs(NewTab.DataTab dataSaveTab) : Αποθηκεύει τα δεδομένα σε νέο αρχείο.
- public void saveRegionEncoding(NewTab.DataTab dataSaveTab) : Αποθηκεύει την κωδικοποίηση θέσης του XML δέντρου.

Μετατροπή μορφής δέντρου

- public boolean JGraph2JTree() : Μετατρέπει ένα δέντρο δενδρικής μορφής σε μορφή ετικετών
- public boolean JTree2JGraph() : Μετατρέπει ένα δέντρο μορφής ετικετών σε δενδρική μορφή.

- public void exit() : Κλείνει την εφαρμογή, αφού πρώτα ρωτήσει αν θέλει ο χρήστης να σώσει τις αλλαγές που έχουμε κάνει σε όλα τα αρχεία.
- public void closeTab(int i) : Κλείνει την καρτέλα, αφού πρώτα ρωτήσει αν θέλει ο χρήστης να σώσει τις αλλαγές που έχουμε κάνει.
- public void removeHashtableData(Component component, NewTab.DataTab dataCloseTab) : Διαγράφει τα δεδομένα της καρτέλας από τις κατάλληλες δομές.
- public static String printHashtable(Hashtable<?,?> hashtable, String hashtable_name) : Εμφανίζει τα δεδομένα μιας δομής hashtable.

Class ImageExport

Αποθηκεύει σε αρχείο την εικόνα ενός γράφου. Αφορά μόνο δέντρα δενδρικής μορφής και ερωτήματα.

Fields

- Trees_and_Queries gui : το αντικείμενο της βασικής εφαρμογής
- JGraph graph : ο γράφος που θα εξαχθεί σε εικόνα

Construstor

- public ImageExport (Trees_and_Queries gui , JGraph graph)

Methods

- public void saveFile() : επιλέγει το αρχείο που θα αποθηκεύσουμε την εικόνα του γράφου
- public void imageExport(File file) : εξάγει την εικόνα του γράφου και την αποθηκεύει στο αρχείο file

Class NewTab

Δημιουργεί μία νέα καρτέλα στην εφαρμογή μας που περιέχει δέντρο μορφής ετικετών ή δέντρο δενδρικής μορφής ή ερώτημα ανάλογα με το πεδίο application.

inner class **DataTab**

Κρατάει τα δεδομένα μιας καρτέλας

Fields

- int id : κωδικός καρτέλας
- Application application : είδος δέντρου ή ερωτήματος που περιέχει η καρτέλα
- File file : αρχείο που είναι αποθηκευμένα τα περιεχόμενα της καρτέλας
- boolean isSaved : ελέγχει αν είναι σωσμένα τα δεδομένα του δέντρου ή του ερωτήματος που περιέχει η καρτέλα.

Construstor

- public DataTab()

Fields

- Trees_and_Queries gui : το αντικείμενο της βασικής εφαρμογής
- Application application : το είδος που περιέχει η καρτέλα (δέντρο μορφής ετικετών, δέντρο δενδρικής μορφής ή ερώτημα)
- DataTab dataTab : τα δεδομένα της καρτέλας.

Constructor

- public NewTab(Trees_and_Queries gui, Application application)

Methods

- public DataTab initDataTab() : Αρχικοποιεί τα δεδομένα της νέας καρτέλας.
- public DataTab CreateTab(String tabName, JTree tree) : Δημιουργεί μια νέα καρτέλα για το δέντρο μορφής ετικετών tree με τίτλο tabName
- public DataTab CreateTab(String tabName, JGraph graph, Boolean isTree) : Δημιουργεί μια νέα καρτέλα με τίτλο tabName για το γράφο του δέντρου δενδρικής μορφής αν το όρισμα isTree είναι true ή του ερωτήματος αν το όρισμα isTree είναι false.
- public void storeTab(String tabName, Component component_tab) : Αποθηκεύει τα δεδομένα της καρτέλας.
- public static JPanel PanelJGraph(JGraph graph) : Δημιουργεί το panel του γράφου είτε του δέντρου δενδρικής μορφής είτε του ερωτήματος
- public static JPanel PanelJTree(JTree tree) : Δημιουργεί το panel του δέντρου μορφής ετικετών
- public DataTab getDataTab() : Επιστρέφει τα δεδομένα της καρτέλας
- public void printTab() : Τυπώνει στοιχεία της καρτέλας

Class ButtonTabComponent

Επεξεργάζεται τη μορφή των καρτελών και τους δίνει τη δυνατότητα κλεισίματος

Class NewTree_JTree

Δημιουργεί ένα νέο δέντρο σε μορφή ετικέτων

inner class **DataTree**

Κρατάει τα δεδομένα ενός δέντρου μορφής ετικετών

Fields

- int id : κωδικός δέντρου μορφής ετικετών
- JTree tree : δέντρο μορφής ετικετών
- DefaultTreeModel treeModel : μοντέλο δέντρου μορφής ετικετών
- VerticesTree_JTree ourVertices : κόμβοι δέντρου μορφής ετικετών
- DefaultMutableTreeNode rootNode : κόμβος-ρίζα δέντρου μορφής ετικετών

Construstor

- public DataTree()

Fields

- Trees_and_Queries gui : το αντικείμενο της βασικής εφαρμογής
- JTree tree : το δέντρο μορφής ετικετών
- DefaultTreeModel treeModel : το μοντέλο δέντρου μορφής ετικετών
- VerticesTree_JTree ourVertices : οι κόμβοι δέντρου μορφής ετικετών
- DefaultMutableTreeNode rootNode : ο κόμβος-ρίζα δέντρου μορφής ετικετών
- DataTree dataTree : τα δεδομένα του δέντρου μορφής ετικετών

Construstor

- public NewTree_JTree(Trees_and_Queries gui) → new tree
- public NewTree_JTree(Trees_and_Queries gui, String rootName) → open tree

Methods

- public void putRoot() : Εισάγει τον κόμβο-ρίζα του δέντρου μορφής ετικετών
- public void initTree_JTree(String rootName) : Αρχικοποιεί το δέντρο μορφής ετικετών
- public DataTree initDataTree() : Αρχικοποιεί τα δεδομένα του δέντρου μορφής ετικετών και τα επιστρέφει
- public DataTree getDataTree() : Επιστρέφει τα δεδομένα του δέντρου μορφής ετικετών.

Class NewTree_JGraph

Δημιουργεί ένα νέο δέντρο σε δενδρική μορφή

inner class **DataTree**

Κρατάει τα δεδομένα ενός δέντρου δενδρικής μορφής

Fields

- int id : κωδικός δέντρου δενδρικής μορφής
- JGraph graph : γράφος δέντρου δενδρικής μορφής
- GraphModel model : μοντέλο δέντρου δενδρικής μορφής
- VerticesTree_JGraph ourVertices : κόμβοι δέντρου δενδρικής μορφής

Construstor

- public DataTree()

Fields

- Trees_and_Queries gui : το αντικείμενο της βασικής εφαρμογής
- JGraph graph : ο γράφος του δέντρου δενδρικής μορφής
- GraphModel model : το μοντέλο του δέντρου δενδρικής μορφής
- VerticesTree_JGraph ourVertices : οι κόμβοι του δέντρου δενδρικής μορφής
- DataTree dataTree : τα δεδομένα του δέντρου δενδρικής μορφής

Construstor

- public NewTree_JGraph(Trees_and_Queries gui, Create create)

Methods

- public void putRoot() : Εισάγει τον κόμβο-ρίζα του δέντρου δενδρικής μορφής
- public void initTree_JGraph() : Αρχικοποιεί το δέντρο δενδρικής μορφής
- public DataTree initDataTree() : Αρχικοποιεί τα δεδομένα του δέντρου δενδρικής μορφής και τα επιστρέφει
- public DataTree getDataTree() : Επιστρέφει τα δεδομένα του δέντρου δενδρικής μορφής

Class NewQuery

Δημιουργεί ένα νέο ερώτημα

inner class **DataQuery**

Κρατάει τα δεδομένα ενός ερωτήματος.

Fields

- int id : κωδικός ερωτήματος
- JGraph graph : γράφος ερωτήματος
- GraphModel model : μοντέλο ερωτήματος
- VerticesQuery ourVertices : κόμβοι ερωτήματος
- Edges_child_descendant ourEdges : ακμές ερωτήματος

Construstor

- public DataQuery()

Fields

- Trees_and_Queries gui : το αντικείμενο της βασικής εφαρμογής
- JGraph graph : ο γράφος του ερωτήματος
- GraphModel model : το μοντέλο του ερωτήματος
- VerticesQuery ourVertices : οι κόμβοι του ερωτήματος
- Edges_child_descendant ourEdges : οι ακμές του ερωτήματος
- DataQuery dataQuery : τα δεδομένα του ερωτήματος

Construstor

- public NewQuery (Trees_and_Queries gui, Create create)

Methods

- public void putRoot() : Εισάγει τον κόμβο-ρίζα του ερωτήματος
- public void initQuery_JGraph() : Αρχικοποιεί το ερώτημα
- public DataQuery initDataQuery() : Αρχικοποιεί τα δεδομένα του ερωτήματος και τα επιστρέφει
- public DataQuery getDataQuery() : Επιστρέφει τα δεδομένα του ερωτήματος

Class OpenTree_JTree

Ανοίγει ένα υπάρχον XML αρχείο σε δέντρο σε μορφή ετικέτων

Fields

- Trees_and_Queries gui : το αντικείμενο της βασικής εφαρμογής
- NewTree_JTree.DataTree dataTree : τα δεδομένα του δέντρου που ανοίγω
- File openFile : το αρχείο που ανοίγω

Construstors

- public OpenXML_JTree(Trees_and_Queries gui)

Methods

- public String openGraph() : Επιλέγει ποιο αρχείο θα ανοίξει από ένα JFileChooser. Το αρχείο πρέπει να έχει κατάληξη xml και να είναι ένα σωστά ορισμένο XML αρχείο, αλλιώς δεν κάνω τίποτα. Αν το αρχείο που επιλέγω είναι ήδη ανοιχτό, πάλι δεν κάνω τίποτα. Επιστρέφει το όνομα του αρχείου.
- public void generateXMLTree(File file) : Είναι η βασική συνάρτηση αυτής της κλάσης γιατί μετατρέπει το XML αρχείο που ανοίγουμε σε ένα δέντρο σε μορφή ετικετών. Διαβάζει λοιπόν το XML αρχείο με τη βοήθεια του SAX parser και διέρχεται σε όλα τα στοιχεία του με το JDOM. Δημιουργεί το νέο δέντρο και προσθέτει σαν ρίζα στο δέντρο μας το πρώτο στοιχείο του XML αρχείου. Μετά καλεί μια αναδρομική κλήση της συνάρτησης getChildren για κάθε ετικέτα που συναντάει στο αρχείο και έτσι στο τέλος θα έχουμε αναπαραστήσει όλο το XML αρχείο.
- public void getChildren(Element el, DefaultMutableTreeNode parentNode) : Για κάθε ετικέτα που συναντάω στο XML αρχείο δημιουργεί ένα DefaultMutableTreeNode κόμβο στο δέντρο μας με όνομα το όνομα της ετικέτας. Μετά γίνεται αναδρομική κλήση για όλα τα παιδιά του στοιχείου el και έτσι στο τέλος θα έχω περιηγηθεί σε όλα τα στοιχεία του XML αρχείου
- public List<?> getChildren_withAttributesText(Element el) : Είναι παρόμοια με τη συνάρτηση getChildren μόνο που επεξεργάζεται επιπλέον τις μεταβλητές και το κείμενο του κάθε στοιχείου. Επιστρέφει μια λίστα με τα παιδιά του στοιχείου.
- public NewTree_JTree.DataTree getDataTree() : Επιστρέφει τα δεδομένα του δέντρου
- public File getOpenFile() : Επιστρέφει το αρχείο που ανοίγω για να το παρουσιάσω σε δέντρο σε μορφή ετικετών.

Class OpenTree_JGraph

Ανοίγει ένα υπάρχον XML αρχείο σε δέντρο σε δενδρική μορφή. Αν είναι μεγάλο αρχείο, δεν το αναπαριστώ σε δενδρική μορφή (βλέπε Ενότητα 5.3.3).

Fields

- Trees_and_Queries gui : το αντικείμενο της βασικής εφαρμογής
- NewTree_JGraph.DataTree dataTree : τα δεδομένα του δέντρου που ανοίγω
- File openFile : το αρχείο που ανοίγω

Construstors

- public OpenXML_JGraph(Trees_and_Queries gui)

Methods

- public String openGraph() : Επιλέγει ποιο αρχείο θα ανοίξει από ένα JFileChooser. Το αρχείο πρέπει να έχει κατάληξη xml και να είναι ένα σωστά ορισμένο XML αρχείο, αλλιώς δεν κάνω τίποτα. Αν το αρχείο που επιλέγω είναι ήδη ανοιχτό, πάλι δεν κάνω τίποτα. Επιστρέφει το όνομα του αρχείου.
- public void generateXMLTree(File file) : Είναι η βασική συνάρτηση αυτής της κλάσης γιατί μετατρέπει το XML αρχείο που ανοίγουμε σε ένα δέντρο σε δενδρική μορφή. Αν είναι μεγάλο αρχείο (βλέπε Ενότητα 5.3.3), δεν γίνεται η αναπαράσταση και επιστρέφει στην κύρια εφαρμογή. Αν όχι, συνεχίζει για να γίνει η αναπαράσταση. Διαβάζει λοιπόν το XML αρχείο με τη βοήθεια του SAX parser και διέρχεται σε όλα τα στοιχεία του με το JDOM. Δημιουργεί το νέο δέντρο και προσθέτει σαν ρίζα στο δέντρο μας το πρώτο στοιχείο του XML αρχείου. Μετά καλεί μια αναδρομική κλήση της συνάρτησης getChildren για κάθε ετικέτα που συναντάει στο αρχείο και έτσι στο τέλος θα έχουμε αναπαραστήσει όλο το XML αρχείο.
- public void getChildren(Element el, DefaultGraphCell vertexParent) : Για κάθε ετικέτα που συναντάω στο XML αρχείο δημιουργεί ένα DefaultGraphCell κόμβο στο δέντρο μας με όνομα το όνομα της ετικέτας. Μετά γίνεται αναδρομική κλήση για όλα τα παιδιά του στοιχείου el και έτσι στο τέλος θα έχω περιηγηθεί σε όλα τα στοιχεία του XML αρχείου
- public List<?> getChildren_withAttributesText(Element el) : Είναι παρόμοια με τη συνάρτηση getChildren μόνο που επεξεργάζεται επιπλέον τις μεταβλητές και το κείμενο του κάθε στοιχείου. Επιστρέφει μια λίστα με τα παιδιά του στοιχείου.
- public NewTree_JGraph.DataTree getDataTree() : Επιστρέφει τα δεδομένα του δέντρου
- public File getOpenFile() : Επιστρέφει το αρχείο που ανοίγω για να το παρουσιάσω σε δέντρο σε δενδρική μορφή.

Class OpenQuery

Ανοίγει ένα υπάρχον ερώτημα

Fields

- Trees_and_Queries gui : το αντικείμενο της βασικής εφαρμογής
- NewQuery.DataQuery dataQuery : τα δεδομένα του ερωτήματος που ανοίγω
- File openFile : το αρχείο που ανοίγω

Πεδία που αποθηκεύουν τον κωδικό του κόμβο εξόδου

- int idOutputNode
- int get_idOutputNode

Βοηθητικά πεδία κλάσης που επεξεργάζονται τα περιεχόμενα του αρχείου για να εξάγουν τους κόμβους και τις ακμές του ερωτήματος

- private StringBuffer input
- String graphData
- String labels[]
- int nodelist[]
- String[] nodesNameList : περιέχει τα ονόματα όλων των κόμβων
- int nodesPerSet[]
- private String cross
- private float[] nodes
- private int[] edgeSet

Construstors

- public OpenQuery(Trees_and_Queries gui)

Methods

Διάβασμα του αρχείου ερωτήματος

- public String openGraph() : Επιλέγει ποιο αρχείο θα ανοίξει από ένα JFileChooser. Το αρχείο πρέπει να έχει κατάληξη txt και να έχει συγκεκριμένη μορφή όπως περιγράφεται στην Ενότητα 6.2.4, αλλιώς δεν κάνω τίποτα. Αν το αρχείο που επιλέγω είναι ήδη ανοιχτό, πάλι δεν κάνω τίποτα. Επιστρέφει το όνομα του αρχείου.
- public void pspHandler(File file) : Είναι η βασική συνάρτηση αυτής της κλάσης γιατί μετατρέπει το αρχείο που ανοίγουμε σε γράφο της εφαρμογής μας.
- public File inputFile(File file) : Ελέγχει αν το αρχείο είναι έγκυρο ερωτήματος όπως έχει περιγραφτεί στην Ενότητα 6.2.4. Αν είναι προχωράει στο διάβασμά του και εξετάζει αν έχει οριστεί κόμβος εξόδου. Αποθηκεύει σε ένα προσωρινό αρχείο μόνο τα δεδομένα που αφορούν τους κόμβους και τος σχέσεις τους σε κατάλληλη μορφή. Επιστρέφει αυτό το προσωρινό αρχείο.

- String readFile(File file) : Διαβάζει το προσωρινό αρχείο που δημιούργησε η συνάρτηση inputFile, επεξεργάζεται τα δεδομένα και διαχωρίζει κόμβους και σχέσεις.

Επεξεργασία δεδομένων αρχείου

- String makeLink(String inp) : Δημιουργεί τις συνδέσεις μεταξύ των κόμβων διαχωρίζοντας τις σχέσεις πατέρα-παιδιού από τις σχέσεις προγόνου-απογόνου. Επιστρέφει τις σχέσεις αυτές.
- public void uniqueNodes() : Αποθηκεύει όλους τους κόμβους που περιέχει το αρχείο και τους δίνει μοναδικό κωδικό.
- String removeRepeatedNodes(String inp): Αν υπάρχουν περιττές σχέσεις μεταξύ 2 κόμβων, αφήνει μόνο την απαραίτητη. Αν για παράδειγμα υπάρχει και σχέση πατέρα-παιδιού και προγόνου-απογόνου τότε αφήνω μόνο τη σχέση πατέρα-απογόνου που είναι πιο περιοριστική. Επιστρέφει τα δεδομένα.
- public void edgesPerGraph(String graphData) : Επεξεργάζεται τις ακμές.

Εμφάνιση των κόμβων και των ακμών

- public void showGraph(String graphData) : Επεξεργάζεται τα δεδμένα του αρχείου και απεικονίζει τους κόμβους και τις ακμές
- public DefaultGraphCell[] createVertices(int count) : Είναι η συνάρτηση που απεικονίζει τους κόμβους στο ερώτημά μας. Το όρισμα count της συνάρτησης ορίζει το πλήθος των κόμβων που έχει το ερώτημά μας. Με τον πίνακα nodesNameList της κλάσης παίρνουμε το όνομα του κάθε κόμβου. Παίρνουμε αρχικά το πρώτο στοιχείο του πίνακα που είναι και η ρίζα του ερωτήματος και την προσθέτουμε στο γράφο μας. Έπειτα παίρνουμε τα υπόλοιπα στοιχεία του πίνακα που είναι κοινοί κόμβοι και τους προσθέτουμε ένα-ένα στο γράφο μας. Τέλος, ελέγχουμε αν έχει οριστεί στο αρχείο κόμβος εξόδου και αν έχει, τον ορίζουμε και στο γράφο μας. Επιστρέφει το σύνολο των κόμβων που δημιουργήσαμε.
- public void createEdges(DefaultGraphCell[] v, String open_edges[]) : Είναι η συνάρτηση που απεικονίζει τις σχέσεις-ακμές μεταξύ των κόμβων στο ερώτημά μας. Το όρισμα DefaultGraphCell[] v της συνάρτησης είναι ένας πίνακας που περιέχει όλους τους κόμβους του ερωτήματος, στον οποίο θα εντοπίζουμε τον κόμβο-πηγή και τον κόμβο προορισμού της κάθε ακμής. Το όρισμα open_edges είναι ένα πίνακας που κάθε στοιχείο του περιέχει τον κωδικό του κόμβου-πηγή ένα βελάκι '→' αν πρόκειται για σχέση πατέρα-παιδιού ή ένα βελάκι '➔' αν πρόκειται για σχέση προγόνου-απόγονου και τέλος τον κωδικό του κόμβου-προορισμού. Παίρνουμε λοιπόν κάθε στοιχείο του πίνακα open_edges, εντοπίζουμε τους κόμβους πηγή και προορισμό από τον πίνακα v και ανάλογα με το είδος του βέλους απεικονίζουμε στο γράφο μας μια ακμή μεταξύ των παραπάνω δύο κόμβων που είναι συνεχής αν πρόκειται για σχέση πατέρα-παιδιού ή διακεκομμένη αν πρόκειται για σχέση προγόνου-απόγονου.

- public NewQuery.DataQuery getDataQuery() : Επιστρέφει τα δεδομένα του ερωτήματος
- public File getOpenFile() : Επιστρέφει το αρχείο που ανοίγω για να παρουσιάσω το ερώτημα σε γράφο.

Class SaveXML_JTree

Αποθηκεύει ένα δέντρο μορφής ετικετών σε ένα XML αρχείο.

Fields

- Trees_and_Queries gui : το αντικείμενο της βασικής εφαρμογής
- Document doc : η αναπαράσταση του όλου κειμένου XML και περιέχει συγχρόνως τα άλλα JDOM αντικείμενα που αντιστοιχούνται σύμφωνα με τις ονομασίες τους απ'ευθείας στις αντίστοιχες του XML.
- int num_elements : το πλήθος των στοιχείων του XML αρχείου
- Hashtable<Integer, DefaultMutableTreeNode> idElement2TreeNode : η δομή που αποθηκεύουμε για κάθε στοιχείο το ζευγάρι {κωδικός στοιχείου, κόμβος}
- DefaultMutableTreeNode rootNode : η ρίζα του δέντρου
- boolean doStoreTree : true για να κάνω την αποθήκευση από την αρχή, false για να κάνω αντιγραφή από το ήδη υπάρχον αποθηκευμένο αρχείο
- File saveFile_XMLTree : το αρχείο που αποθηκεύουμε το δέντρο
- File alreadySavedFile : το αρχείο στο οποίο είχαμε αποθηκεύσει την τελευταία φορά το δέντρο.

Construstors

- public SaveXML_JTree(Trees_and_Queries gui, DefaultMutableTreeNode rootNode) → καλείται από τη SaveRegionEncoding
- public SaveXML_JTree(Trees_and_Queries gui, DefaultMutableTreeNode rootNode, File alreadySavedFile)
- public SaveXML_JTree(Trees_and_Queries gui, File alreadySavedFile) → στην περίπτωση save as όταν υπάρχει ήδη το αρχείο και δεν έχουν γίνει αλλαγές, οπότε δεν διασχίζω το δέντρο για να εξάγω το XML αρχείο, αλλά το αντιγράφω από το υπάρχον.

Methods

- public boolean saveFile() : Επιλέγει σε ποιο XML αρχείο θα αποθηκεύσει το δέντρο μορφής ετικετών από ένα JFileChooser. Αν το αρχείο που επιλέγω είναι ήδη ανοιχτό και δεν είναι το αρχείο που αντιστοιχεί στην τρέχουσα καρτέλα τότε η αποθήκευση δεν γίνεται και ενημερώνω κατάλληλα τον χρήστη. Επιστρέφει true αν έγινε σωστά η αποθήκευση, αλλιώς false.
- public boolean storeTree(File file) : Είναι η βασική συνάρτηση αυτής της κλάσης γιατί αποθηκεύει το δέντρο μορφής ετικετών σε ένα XML αρχείο. Κατασκευάζεται αρχικά ένας XMLOutputter. Δημιουργούμε και ένα Document doc. Για κάθε κόμβο που συναντάμε στο δέντρο δημιουργούμε με την επομένη συνάρτηση showChildren_JDOM ένα νέο στοιχείο. Καλώντας αναδρομικά τη συνάρτηση αυτή στο τέλος θα έχουμε αποθηκεύσει όλο το δέντρο στο XML αρχείο. Τέλος, παραδίδουμε το doc και το επιθυμητό FileOutputStream στον XMLOutputter και έχουμε αποθηκεύσει το δέντρο στο XML αρχείο. Επιστρέφει true αν έγινε σωστά η αποθήκευση, αλλιώς false.
- public void showChildren_JDOM(Document doc, DefaultMutableTreeNode parentNode, Element parentElement) : Για κάθε κόμβο-παιδί του parentNode που υπάρχει στο δέντρο δημιουργεί ένα στοιχείο elementChild στο XML αρχείο με όνομα το όνομα του κόμβου και μπαίνει ως παιδί του parentElement με τη συνάρτηση addContent. Μετά γίνεται αναδρομική κλήση για όλα τα παιδιά του

parentNode και έτσι στο τέλος θα έχω περιηγηθεί σε όλους του κόμβους του δέντρου.
- public File getSaveFile() : Επιστρέφει το XML αρχείο που αποθήκευσα το δέντρο μορφής ετικετών
- public void printFileData(File file) : Τυπώνει τα δεδομένα του αρχείου file

Class SaveXML_JGraph

Αποθηκεύει ένα δέντρο δενδρικής μορφής σε ένα XML αρχείο.

Fields

- Trees_and_Queries gui : το αντικείμενο της βασικής εφαρμογής
- Document doc : η αναπαράσταση του όλου κειμένου XML και περιέχει συγχρόνως τα άλλα JDOM αντικείμενα που αντιστοιχούνται σύμφωνα με τις ονομασίες τους απ'ευθείας στις αντίστοιχες του XML.
- int num_elements : το πλήθος των στοιχείων του XML αρχείου
- Hashtable<Integer, Integer> idElement2idJGraph : η δομή που αποθηκεύουμε για κάθε στοιχείο το ζευγάρι {κωδικός στοιχείου, κωδικός κόμβου}
- VerticesTree_JGraph ourVertices : οι κόμβοι του δέντρου δενδρικής μορφής
- boolean doStoreTree : true για να κάνω την αποθήκευση από την αρχή, false για να κάνω αντιγραφή από το ήδη υπάρχον αποθηκευμένο αρχείο
- File saveFile_XMLTree : το αρχείο που αποθηκεύουμε το δέντρο
- File alreadySavedFile : το αρχείο στο οποίο είχαμε αποθηκεύσει την τελευταία φορά το δέντρο.

Construstors

- public SaveXML_JGraph(Trees_and_Queries gui, VerticesTree_JGraph ourVertices) → καλείται από τη SaveRegionEncoding
- public SaveXML_JGraph(Trees_and_Queries gui, VerticesTree_JGraph ourVertices, File alreadySavedFile)
- public SaveXML_JGraph(Trees_and_Queries gui, File alreadySavedFile) → στην περίπτωση save as όταν υπάρχει ήδη το αρχείο και δεν έχουν γίνει αλλαγές, οπότε δεν διασχίζω το δέντρο για να εξάγω το XML αρχείο, αλλά το αντιγράφω από το υπάρχον.

Methods

- public boolean saveFile() : Επιλέγει σε ποιο XML αρχείο θα αποθηκεύσει το δέντρο δενδρικής μορφής από ένα JFileChooser. Αν το αρχείο που επιλέγω είναι ήδη ανοιχτό και δεν είναι το αρχείο που αντιστοιχεί στην τρέχουσα καρτέλα τότε η αποθήκευση δεν γίνεται και ενημερώνω κατάλληλα τον χρήστη. Επιστρέφει true αν έγινε σωστά η αποθήκευση, αλλιώς false.
- public boolean storeTree(File file) : Είναι η βασική συνάρτηση αυτής της κλάσης γιατί αποθηκεύει το δέντρο δενδρικής μορφής σε ένα XML αρχείο. Κατασκευάζεται αρχικά ένας XMLOutputter. Δημιουργούμε και ένα Document doc. Για κάθε κόμβο που συναντάμε στο δέντρο δημιουργούμε με την επομένη συνάρτηση showChildren_JDOM ένα νέο στοιχείο. Καλώντας αναδρομικά τη συνάρτηση αυτή στο τέλος θα έχουμε αποθηκεύσει όλο το δέντρο στο XML αρχείο. Τέλος, παραδίδουμε το doc και το επιθυμητό FileOutputStream στον XMLOutputter και έχουμε αποθηκεύσει το δέντρο στο XML αρχείο. Επιστρέφει true αν έγινε σωστά η αποθήκευση, αλλιώς false.

- public void showChildren_JDOM(Document doc, DefaultGraphCell parentNode, Element parentElement) : Για κάθε κόμβο-παιδί του parentNode που υπάρχει στο δέντρο δημιουργεί ένα στοιχείο elementChild στο XML αρχείο με όνομα το όνομα του κόμβου και μπαίνει ως παιδί του parentElement με τη συνάρτηση addContent. Μετά γίνεται αναδρομική κλήση για όλα τα παιδιά του parentNode και έτσι στο τέλος θα έχω περιηγηθεί σε όλους του κόμβους του δέντρου.
- public File getSaveFile() : Επιστρέφει το XML αρχείο που αποθήκευσα το δέντρο δενδρικής μορφής.
- public void printFileData(File file) : Τυπώνει τα δεδομένα του αρχείου file

Class SaveQuery

Αποθηκεύει ένα ερώτημα σε ένα txt αρχείο στη μορφή που αναπτύχθηκε στην Ενότητα 6.2.4.

Fields

- Trees_and_Queries gui : το αντικείμενο της βασικής εφαρμογής
- VerticesQuery ourVertices : οι κόμβοι του ερωτήματος
- Edges_child_descendant ourEdges : οι ακμές του ερωτήματος
- boolean doStoreQuery : true για να κάνω την αποθήκευση από την αρχή, false για να κάνω αντιγραφή από το ήδη υπάρχον αποθηκευμένο αρχείο
- File saveFile_Query : το αρχείο που αποθηκεύουμε το δέντρο
- File alreadySavedFile : το αρχείο στο οποίο είχαμε αποθηκεύσει την τελευταία φορά το δέντρο.

Construstors

- public SaveQuery(Trees_and_Queries gui, NewQuery.DataQuery dataQuery, File alreadySavedFile) → Αποθηκεύει το ερώτημα σε ένα txt αρχείο. Αν δεν έχει οριστεί κόμβος εξόδου ενημερώνω το χρήστη και τον ρωτάω αν παρόλα αυτά θέλει να γίνει η αποθήκευση. Μετά συνδέω τους κόμβους χωρίς εισερχόμενες ακμές με τη ρίζα ή παιδί της (κανόνας IR1).
- public SaveQuery (Trees_and_Queries gui, File alreadySavedFile) → στην περίπτωση save as όταν υπάρχει ήδη το αρχείο και δεν έχουν γίνει αλλαγές, οπότε δεν διασχίζω το ερώτημα για να εξάγω το txt αρχείο, αλλά το αντιγράφω από το υπάρχον.

Methods

- public boolean saveFile() : Επιλέγει σε ποιο txt αρχείο θα αποθηκεύσει το ερώτημα από ένα JFileChooser. Αν το αρχείο που επιλέγω είναι ήδη ανοιχτό και δεν είναι το αρχείο που αντιστοιχεί στην τρέχουσα καρτέλα τότε η αποθήκευση δεν γίνεται και ενημερώνω κατάλληλα τον χρήστη. Επιστρέφει true αν έγινε σωστά η αποθήκευση, αλλιώς false.
- public boolean storeQuery(File file) : Είναι η βασική συνάρτηση αυτής της κλάσης γιατί αποθηκεύει το ερώτημα σε ένα txt αρχείο προσυμφωνημένης μορφής (βλέπε Ενότητα 6.2.4). Υπάρχουν 3 βασικές γραμμές, για τους κόμβους, για τις σχέσεις πατέρα-παιδιού και για τις σχέσεις προγόνου-απογόνου. Για κάθε κόμβο που συναντάμε στο ερώτημα προσθέτουμε τον κωδικό του και το όνομα του στη γραμμή κόμβων. Για κάθε ακμή που συναντάμε προσθέτουμε στην κατάλληλη γραμμή, ανάλογα με το είδος της σχέσης, τον κωδικό του κόμβου πηγής και τον κωδικό του κόμβου προορισμού. Τέλος, αν υπάρχει κόμβος εξόδου, βάζουμε τον κωδικό του

στην 1η γραμμή του αρχείου. Επιστρέφει true αν έγινε σωστά η αποθήκευση, αλλιώς false.

- public boolean storeQuery(File file, Hashtable<String, Integer> name2id) : Η συνάρτηση αυτή καλείται μόνο στην αποτίμηση ενός ερωτήματος. Κάνει την ίδια διαδικασία με την προηγούμενη συνάρτηση, με τη διαφορά ότι για κάθε κόμβο που συναντάει αντί να αποθηκεύει το όνομα, αποθηκεύει τον κωδικό που του αντιστοιχεί στο όρισμα name2id που προέρχεται από το δέντρο που ερωτάται. Επιστρέφει true αν έγινε σωστά η αποθήκευση, αλλιώς false.
- public File getSaveFile() : Επιστρέφει το txt αρχείο που αποθήκευσα το ερώτημα.

Class SaveRegionEncoding_JTree

Δημιουργεί την κωδικοποίηση θέσης του δέντρου σε μορφή ετικετών και την αποθηκεύει σε ένα txt αρχείο.

inner class *NodeEncoding*

Η κωδικοποίηση θέσης ενός στοιχείου

Fields

- int start : η πρώτη επίσκεψη του κόμβου
- int end : η τελευταία επίσκεψη του κόμβου
- int level : το επίπεδο του κόμβου

Construstor

- public NodeEncoding()

inner class *Stream*

Fields

- Vector<NodeEncoding> stream : το διάνυσμα που αποθηκεύω την κωδικοποίηση θέσης όλων των κόμβων του δέντρου
- public Stack<Integer> st : η στοίβα που κραταέι τη θέση στη λίστα των κόμβων που δεν έχουν τελειώσει ακόμα

Construstor

- public Stream()

Fields

- int num_tags : Το πλήθος όλων των στοιχείων που υπάρχουν. Προσδιορίζουν το μοναδικό κωδικό του κάθε κόμβου.
- int tagCounter : Το πλήθος όλων των διαφορετικών στοιχείων που υπάρχουν. Προσδιορίζουν το μοναδικό κωδικό του κάθε διαφορετικού κόμβου.

- int regCounter : Ο μετρητής που προσδιορίζει σε ποια επίσκεψη των στοιχείων του δέντρου βρισκόμαστε (κατά βάθος διάσχιση).
- int currentDepth : Ο μετρητής που προσδιορίζει σε ποιο επίπεδο του δέντρου βρισκόμαστε.

Βασικές δομές της κλάσης

- Hashtable<Integer, Stream> streams : Η δομή που αποθηκεύουμε τα ζευγάρια {κωδικός στοιχείου, διάνυσμα που περιέχει τις κωδικοποιήσεις θέσης του στοιχείου αυτού}

- Hashtable<String, Integer> tag2id : Η δομή που αποθηκεύουμε για κάθε ετικέτα το ζευγάρι {όνομα ετικέτας, κωδικός ετικέτας}
- Hashtable<Integer, String> id2tag : Η δομή που αποθηκεύουμε για κάθε ετικέτα το ζευγάρι {κωδικός ετικέτας, όνομα ετικέτας}

- Hashtable<Integer, Integer> start2id;
- Hashtable<Integer, Integer> startNodeEncoding2idElement : Η δομή που αποθηκεύουμε για κάθε ετικέτα το ζευγάρι {$1^η$ επίσκεψη του στοιχείου, κωδικός στοιχείου}

- Hashtable<Integer, DefaultMutableTreeNode> idElement2TreeNode : Η δομή που αποθηκεύουμε για κάθε στοιχείο το ζευγάρι {κωδικός στοιχείου, κόμβος στο δέντρο στη μορφή ετικετών}

Αρχεία

- File saveFile_XMLTree : το XML αρχείο που αποθηκεύω το δέντρο
- File saveFile_XMLTree_RE : το txt αρχείο που αποθηκεύω την κωδικοποίηση θέσης των κόμβων του δέντρου

Construstors

- public SaveRegionEncoding_JTree(Hashtable<Integer, DefaultMutableTreeNode> idElement2TreeNode, File saveXMLFile)
- public SaveRegionEncoding_JTree(Trees_and_Queries gui, DefaultMutableTreeNode rootNode)

Methods

Παραγωγή των κωδικοποιήσεων θέσης των κόμβων

- public void generateRegionEncoding() : Διαβάζει το XML αρχείο με τη βοήθεια του SAX parser και διέρχεται σε όλα τα στοιχεία του με το JDOM, καλώντας αναδρομικά την παρακάτω συνάρτηση.
- public void processElement(Element el) : Διασχίζει όλα τα στοιχεία του XML αρχείου.

- public void startElement(Element el) : Όταν εμφανίζεται ένα νέο στοιχείο el, βρίσκει σε ποια επίσκεψη και σε ποιο επίπεδο του δέντρου βρίσκεται το στοιχείο el και καταχωρεί τα δεδομένα αυτά στις τιμές start και level αντιστοίχως της κωδικοποίησης θέσης αυτού του στοιχείου.
- public void endElement(Element el) : Όταν κλείνει ένα στοιχείο el, βρίσκει σε ποια επίσκεψη του δέντρου βρίσκεται το στοιχείο el και καταχωρεί το δεδομένο αυτό στην τιμή end της κωδικοποίησης θέσης αυτού του στοιχείου.
- public void printstreams(Hashtable<Integer, Stream> streams) : Συγκεντρώνει όλες τις κωδικοποιήσεις θέσης των κόμβων του δέντρου.
- public String printStream(int tagid) : Τυπώνει την κωδικοποίηση θέσης κάθε κόμβου στη μορφή start/end/level/0. Η τελευταία τιμή είναι πάντα 0 γιατί πρόκειται για το ίδιο XML έγγραφο. Επιστρέφει την κωδικοποίηση θέσης.

Αρχείο

- public void saveFileRE(String dataStreams) : Επιλέγουμε από ένα JFileChooser σε ποιο αρχείο θα αποθηκεύουμε τις κωδικοποιήσεις θέσης των κόμβων
- public void saveFileRE_withoutJFileChooser(String dataStreams) : Ορίζουμε σε ποιο αρχείο θα αποθηκεύουμε τις κωδικοποιήσεις θέσης των κόμβων
- public void storeFileRE(File f, String dataStreams) : Αποθηκεύουμε τις κωδικοποιήσεις θέσης των κόμβων σε αρχείο
- public File getSaveFile_RE() : Επιστρέφει το αρχείο που αποθηκεύσαμε τις κωδικοποιήσεις θέσης των κόμβων

Επιστρέφουν βασικές δομές της κλάσης

- public Hashtable<String, Integer> getHash_tag2id()
- public Hashtable<Integer, Integer> getHash_startNodeEncoding2idElement()
- public Hashtable<Integer, DefaultMutableTreeNode> getHash_idElement2TreeNode()

Class SaveRegionEncoding_JGraph

Δημιουργεί την κωδικοποίηση θέσης του δέντρου σε δενδρική μορφή και την αποθηκεύει σε ένα txt αρχείο.

inner class *NodeEncoding*

Η κωδικοποίηση θέσης ενός στοιχείου

Fields

- int start : η πρώτη επίσκεψη του κόμβου
- int end : η τελευταία επίσκεψη του κόμβου
- int level : το επίπεδο του κόμβου

Construstor

- public NodeEncoding()

inner class ***Stream***

Fields

- Vector<NodeEncoding> stream : το διάνυσμα που αποθηκεύω την κωδικοποίηση θέσης όλων των κόμβων του δέντρου
- public Stack<Integer> st : η στοίβα που κραταέι τη θέση στη λίστα των κόμβων που δεν έχουν τελειώσει ακόμα

Construstor

- public Stream()

Fields

- int num_tags : Το πλήθος όλων των στοιχείων που υπάρχουν. Προσδιορίζουν το μοναδικό κωδικό του κάθε κόμβου.
- int tagCounter : Το πλήθος όλων των διαφορετικών στοιχείων που υπάρχουν. Προσδιορίζουν το μοναδικό κωδικό του κάθε διαφορετικού κόμβου.

- int regCounter : Ο μετρητής που προσδιορίζει σε ποια επίσκεψη των στοιχείων του δέντρου βρισκόμαστε (κατά βάθος διάσχιση).
- int currentDepth : Ο μετρητής που προσδιορίζει σε ποιο επίπεδο του δέντρου βρισκόμαστε.

Βασικές δομές της κλάσης

- Hashtable<Integer, Stream> streams : Η δομή που αποθηκεύουμε τα ζευγάρια {κωδικός στοιχείου, διάνυσμα που περιέχει τις κωδικοποιήσεις θέσης του στοιχείου αυτού}

- Hashtable<String, Integer> tag2id : Η δομή που αποθηκεύουμε για κάθε ετικέτα το ζευγάρι {όνομα ετικέτας, κωδικός ετικέτας}
- Hashtable<Integer, String> id2tag : Η δομή που αποθηκεύουμε για κάθε ετικέτα το ζευγάρι {κωδικός ετικέτας, όνομα ετικέτας}

- Hashtable<Integer, Integer> start2id
- Hashtable<Integer, Integer> startNodeEncoding2idElement : Η δομή που αποθηκεύουμε για κάθε ετικέτα το ζευγάρι {$1^η$ επίσκεψη του στοιχείου, κωδικός στοιχείου}

- Hashtable<Integer, Integer> idElement2idJGraph : Η δομή που αποθηκεύουμε για κάθε στοιχείο το ζευγάρι {κωδικός στοιχείου, κωδικός κόμβου στο δέντρο στη δενδρική μορφή}

Αρχεία

- File saveFile_XMLTree : το XML αρχείο που αποθηκεύω το δέντρο
- File saveFile_XMLTree_RE : το txt αρχείο που αποθηκεύω την κωδικοποίηση θέσης των κόμβων του δέντρου

Construstors

- public SaveRegionEncoding_JGraph(Hashtable<Integer, Integer> idElement2idJGraph, File saveXMLFile)

- public SaveRegionEncoding_JGraph(Trees_and_Queries gui, NewTab.DataTab dataSelectedTab)
- public SaveRegionEncoding_JGraph(File fileTree, Trees_and_Queries gui, NewTree_JGraph.DataTree dataSaveTree)

Methods

Παραγωγή των κωδικοποιήσεων θέσεις των κόμβων

- public void generateRegionEncoding() : Διαβάζει το XML αρχείο με τη βοήθεια του SAX parser και διέρχεται σε όλα τα στοιχεία του με το JDOM, καλώντας αναδρομικά την παρακάτω συνάρτηση.
- public void processElement(Element el) : Διασχίζει όλα τα στοιχεία του XML αρχείου.
- public void startElement(Element el) : Όταν εμφανίζεται ένα νέο στοιχείο el, βρίσκει σε ποια επίσκεψη και σε ποιο επίπεδο του δέντρου βρίσκεται το στοιχείο el και καταχωρεί τα δεδομένα αυτά στις τιμές start και level αντιστοίχως της κωδικοποίησης θέσης αυτού του στοιχείου.
- public void endElement(Element el) : Όταν κλείνει ένα στοιχείο el, βρίσκει σε ποια επίσκεψη του δέντρου βρίσκεται το στοιχείο el και καταχωρεί το δεδομένο αυτό στην τιμή end της κωδικοποίησης θέσης αυτού του στοιχείου.
- public void printstreams(Hashtable<Integer, Stream> streams) : Συγκεντρώνει όλες τις κωδικοποιήσεις θέσης των κόμβων του δέντρου.
- public String printStream(int tagid) : Τυπώνει την κωδικοποίηση θέσης κάθε κόμβου στη μορφή start/end/level/0. Η τελευταία τιμή είναι πάντα 0 γιατί πρόκειται για το ίδιο XML έγγραφο.Επιστρέφει την κωδικοποίηση θέσης.

Αρχείο

- public void saveFileRE(String dataStreams) : Επιλέγουμε από ένα JFileChooser σε ποιο αρχείο θα αποθηκεύουμε τις κωδικοποιήσεις θέσης των κόμβων
- public void saveFileRE_withoutJFileChooser(String dataStreams) : Ορίζουμε σε ποιο αρχείο θα αποθηκεύουμε τις κωδικοποιήσεις θέσης των κόμβων
- public void storeFileRE(File f, String dataStreams) : Αποθηκεύουμε τις κωδικοποιήσεις θέσης των κόμβων σε αρχείο
- public File getSaveFile_RE() : Επιστρέφει το αρχείο που αποθηκεύσαμε τις κωδικοποιήσεις θέσης των κόμβων

Επιστρέφουν βασικές δομές της κλάσης

- public Hashtable<String, Integer> getHash_tag2id()
- public Hashtable<Integer, Integer> getHash_startNodeEncoding2idElement()
- public Hashtable<Integer, Integer> getHash_idElement2idJGraph()

Class SaveTreeData

Δημιουργεί τη μορφή treeData του δέντρου δενδρικής μορφής (βλέπε Ενότητα 6.2.3) και την αποθηκεύει σε ένα txt αρχείο.

Fields

- Trees_and_Queries gui : το αντικείμενο της βασικής εφαρμογής
- GraphModel model : το μοντέλο του δέντρου δενδρικής μορφής
- VerticesTree_JGraph ourVertices : οι κόμβοι του δέντρου δενδρικής μορφής
- File alreadySavedFile : το αρχείο στο οποίο είχαμε αποθηκεύσει την τελευταία φορά το δέντρο.

Construstor

- public SaveTreeData (Trees_and_Queries gui, GraphModel model, VerticesTree_JGraph ourVertices, File file)

Methods

- public void saveFileTreeData() : Επιλέγουμε από ένα JFileChooser σε ποιο txt αρχείο θα αποθηκεύουμε τη μορφή treeData του δέντρου
- public void storeFileTreeData() : Αποθηκεύουμε τη μορφή treeData του δέντρου στο txt αρχείο.
- public String getTreeData() : Παίρνουμε τη μορφή treeData του δέντρου
- public String getNodes(DefaultGraphCell vertex, String treeData) : Καταχωρούμε τους κόμβους του δέντρου στη μορφή treeData. Για κάθε κόμβο που συναντάμε προσθέτουμε τον κωδικό και το όνομα του στην $1^{η}$ γραμμή του αρχείου. Επιστρέφει τη γραμμή αυτή.
- public String getEdges(DefaultGraphCell vertex, String treeData) : Καταχωρούμε τις ακμές του δέντρου στη μορφή treeData. Για κάθε κόμβο του δέντρου βάζουμε σε μια νέα γραμμή τον κωδικό του και μετά ακολουθούν οι κωδικοί των παιδιών του χωρισμένοι με έναν κενό χαρακτήρα. Αν δεν υπάρχουν παιδιά τότε στη τρέχουσα γραμμή θα υπάρχει μόνο ο κωδικός του κόμβου. Επιστρέφει τη μορφή treeData.

Class VerticesTree_JTree implements ActionListener

Διαχειρίζεται όλα τα δεδομένα ενός δέντρου σε μορφή ετικετών

Fields

- Trees_and_Queries gui : το αντικείμενο της βασικής εφαρμογής
- JTree tree : το δέντρο μορφής ετικετών
- DefaultTreeModel treeModel : το μοντέλο του δέντρου
- DefaultMutableTreeNode rootNode : η ρίζα του δέντρου

Δομές που σχετίζονται με την παραγωγή της κωδικοποίησης θέσης

- Hashtable<String, Integer> tag2id : Η δομή που αποθηκεύουμε για κάθε ετικέτα το ζευγάρι {όνομα ετικέτας, κωδικός ετικέτας}
- Hashtable<Integer, Integer> startNodeEncoding2idElement : Η δομή που αποθηκεύουμε για κάθε στοιχείο το ζευγάρι {$1^η$ επίσκεψη του στοιχείου, κωδικός στοιχείου}
- Hashtable<Integer, DefaultMutableTreeNode> idElement2TreeNode : Η δομή που αποθηκεύουμε για κάθε στοιχείο το ζευγάρι {κωδικός στοιχείου, κόμβος στο δέντρο στη μορφή ετικετών}
- Hashtable<Integer, Integer> idElement2idJGraph : Η δομή που αποθηκεύουμε για κάθε στοιχείο το ζευγάρι {κωδικός στοιχείου, κωδικός κόμβου στο δέντρο στη δενδρική μορφή}

Πεδία που αφορούν τον διάλογο που εμφανίζεται στο χρήστη κατά την προσθήκη πολλαπλών κόμβων

- JDialog dialog_addVertices : Το παράθυρο που εμφανίζεται για την προσθήκη πολλαπλών κόμβων
- JTable table : Ο πίνακας στον οποίο θα δώσουμε τα ονόματα των νέων κόμβων
- JButton button_ok_vertices : Το κουμπί για αποδοχή της προσθήκης των κόμβων με ονόματα αυτά που δόθηκαν στον πίνακα table
- JButton button_cancel_vertices : Το κουμπί για ματαίωση της προσθήκης

Πεδία που σχετίζονται με το μενού που εμφανίζεται με το δεξί κλικ του ποντικιού

- JPopupMenu popupMenu
- JMenuItem renameNode
- JMenuItem addChild
- JMenuItem addChildren
- JMenuItem addAfter
- JMenuItem addBefore
- JMenuItem deleteNode
- JMenuItem deleteChildren
- JMenuItem replaceParent
- JMenuItem expandDescendants

Construstor

- VerticesTree_JTree(Trees_and_Queries gui, JTree tree, DefaultTreeModel treeModel, DefaultMutableTreeNode) : Αρχικοποιώ τα πεδία της κλάσης και δημιουργώ το μενού που εμφανίζεται με το δεξί κλικ του ποντικιού

Methods

- public void renameNode() : Μετονομάζει έναν κόμβο. Ο κόμβος που θα μετονομαστεί είναι αυτός που είναι επιλεγμένος στο δέντρο. Αν δεν είναι μοναδικός ο κόμβος, τότε δεν γίνεται η μετονομασία. Έπειτα ζητάμε από το χρήστη να δώσει το νέο όνομα. Αν το νέο όνομα δεν είναι έγκυρο (βλέπε Ενότητα 5..3.4.3) ή αν είναι το ίδιο με το υπάρχον, η μετονομασία σταματάει.

Προσθήκη

- public boolean addNode() : Προσθέτει έναν κόμβο ως παιδί επιλεγμένου κόμβου. Ο νέος κόμβος θα προστεθεί σαν παιδί του κόμβου που είναι επιλεγμένος στο δέντρο. Αν δεν είναι μοναδικός ο επιλεγμένος κόμβος, τότε η προσθήκη δεν γίνεται. Αν πάλι δεν υπάρχει επιλεγμένος κόμβος τότε πατέρας του νέου κόμβου θα γίνει η ρίζα του δέντρου. Ζητάμε από το χρήστη να δώσει το όνομα του νέου κόμβου. Αν το όνομα δεν είναι έγκυρο (βλέπε Ενότητα 5.3.4.3) τότε η προσθήκη κόμβου δεν γίνεται. Επιστρέφει true αν γίνει η προσθήκη του κόμβου, αλλιώς false.
- public boolean addNode_before() : Προσθέτει έναν κόμβο ως προηγούμενο αδερφό του επιλεγμένου κόμβου. Αν ο επιλεγμένος κόμβος δεν είναι μοναδικός ή αν δεν υπάρχει επιλεγμένος κόμβος ή αν ο επιλεγμένος κόμβος είναι η ρίζα του δέντρου (αφού η ρίζα δεν μπορεί να έχει αδέρφια), τότε η προσθήκη δεν γίνεται. Αν δεν ισχύουν αυτά, ζητάμε από το χρήστη να δώσει το όνομα του νέου κόμβου. Αν το όνομα δεν είναι έγκυρο (βλέπε Ενότητα 5.3.4.3) τότε η προσθήκη κόμβου δεν γίνεται. Επιστρέφει true αν γίνει η προσθήκη του κόμβου, αλλιώς false.
- public boolean addNode_after(): Προσθέτει έναν κόμβο ως επόμενο αδερφό του επιλεγμένου κόμβου. Αν ο επιλεγμένος κόμβος δεν είναι μοναδικός ή αν δεν υπάρχει επιλεγμένος κόμβος ή αν ο επιλεγμένος κόμβος είναι η ρίζα του δέντρου (αφού η ρίζα δεν μπορεί να έχει αδέρφια), τότε η προσθήκη δεν γίνεται. Αν δεν ισχύουν αυτά, ζητάμε από το χρήστη να δώσει το όνομα του νέου κόμβου. Αν το όνομα δεν είναι έγκυρο (βλέπε Ενότητα 5.3.4.3) τότε η προσθήκη κόμβου δεν γίνεται. Επιστρέφει true αν γίνει η προσθήκη του κόμβου, αλλιώς false.

- public void addManyNodes() : Αρχικά ελέγχω αν στο δέντρο υπάρχει μοναδικός επιλεγμένος κόμβος. Αν όχι, η συνάρτηση επιστρέφει. Αν ναι, τότε ο μοναδικός αυτός κόμβος θα γίνει ο πατέρας των κόμβων που θα προσθέσει ο χρήστης. Έπειτα ζητείται από το χρήστη να δώσει το πλήθος των κόμβων που θέλει να προσθέσει. Αν η τιμή που θα δώσει δεν είναι ακέραιος τότε η συνάρτηση επιστρέφει. Αν δώσει τον αριθμό ένα τότε καλείται η συνάρτηση addNode() που είδαμε παραπάνω για την προσθήκη ενός κόμβου. Αλλιώς καλείται η συνάρτηση addNodes_table(num_vertices) (βλέπε παρακάτω)
- public void addNodes_table(int num_vertices) : Εμφανίζει έναν πίνακα με τόσες γραμμές όσες δηλώνει το όρισμα num_vertices και ζητάει από το χρήστη να τον γεμίσει με τα νέα ονόματα των κόμβων.
- public boolean addManyNodes_ok() : Προσθέτει τους νέους κόμβους ως παιδιά του επιλεγμένου κόμβου. Πατέρας των νέων κόμβων είναι ο μοναδικός επιλεγμένος κόμβος του δέντρου. Παίρνει μία-μία τις γραμμές του πίνακα και διαβάζει το όνομα της κάθε γραμμής. Αν το όνομα είναι κενό ή δεν είναι έγκυρο (βλέπε Ενότητα 5.3.4.3) τότε δεν προστίθεται ο κόμβος. Οπότε μπορεί να μην προστεθούν τόσοι κόμβοι όσοι ζήτησε αρχικά ο χρήστης γιατί δεν έδωσε έγκυρα ονόματα. Επιστρέφει true αν γίνει η προσθήκη των κόμβων, αλλιώς false.

Διαγραφή

- public boolean removeNode() : Διαγράφει τους επιλεγμένους κόμβους καθώς και τα υποδέντρα τους. Αν η ρίζα περιλαμβάνεται στους επιλεγμένους κόμβους, τότε διαγράφω τους υπόλοιπους κόμβους ενώ απαγορεύω να διαγράψω τη ρίζα. Επιστρέφει true αν γίνει διαγραφή, αλλιώς false.
- public boolean removeChildren() : Αν είναι επιλεγμένοι περισσότεροι του ενός κόμβοι ή αν δεν έχει επιλεγεί κόμβος τότε επιστρέφει false. Αλλιώς διαγράφει όλο το υποδέντρο του μοναδικού επιλεγμένου κόμβου του δέντρου. Επιστρέφει true αν γίνει διαγραφή, αλλιώς false

- public boolean replaceParent() : Αντικαθιστά τον πατέρα του επιλεγμένου κόμβου. Αν στο δέντρο δεν υπάρχει μοναδικός επιλεγμένος κόμβος, τότε η συνάρτηση επιστρέφει false. Αλλιώς εμφανίζεται μια λίστα στο χρήστη που περιέχει όλους τους κόμβους του δέντρου και του ζητείται να επιλέξει τον νέο πατέρα. Προεπιλεγμένος κόμβος στη ρίζα ορίζεται ο τωρινός πατέρας του κόμβου. Αν δεν επιλέχθηκε νέος πατέρας τότε επιστρέφει false. Αν ο κόμβος που επέλεξε είναι απόγονος ή ταυτίζεται με τον επιλεγμένο κόμβο του δέντρου τότε η αλλαγή πατέρα δεν επιτρέπεται. Επιστρέφει true αν γίνει η αντικατάσταση πατέρα, αλλιώς false

- public void expandDescendants(DefaultMutableTreeNode node) : Εμφανίζει όλα τα στοιχεία του υποδέντρου του κόμβου node αν είναι κρυμμένα.

- public void actionPerformed(ActionEvent evt) : Καλείται όταν ο χρήστης εκτελέσει κάποια ενέργεια και αυτή με τη σειρά της καλεί την απαραίτητη μέθοδο.

Χειρισμός αναδυόμενου μενού που εμφανίζεται με το δεξί κλικ του ποντικιού

- public void createPopupMenu()
- public void PopupMenuItems()

class PopupListener extends MouseAdapter

Fields

- JPopupMenu popup

Constructors

- PopupListener(JPopupMenu popupMenu)

Methods

- public void mousePressed(MouseEvent e)
- public void mouseReleased(MouseEvent e)

Class VerticesTree_JGraph implements ActionListener

Διαχειρίζεται όλα τα δεδομένα ενός δέντρου σε δενδρική μορφή.

inner class **DataVertex**

Κρατάει τα δεδομένα ενός κόμβου του δέντρου δενδρικής μορφής.

Fields

- int id : κωδικός κόμβου vertex
- String name : όνομα κόμβου vertex

- DefaultGraphCell vertex : ο τρέχων κόμβος
- int level : το επίπεδο του δέντρου που βρίσκεται ο κομβος vertex
- DefaultGraphCell vertexParent : ο κόμβος-πατέρας του κόμβου vertex.
- DefaultGraphCell vertexChildren : οι κόμβοι-παιδιά του κόμβου vertex.

Constructor

- public DataVertex()

Fields

- Trees_and_Queries gui : το αντικείμενο της βασικής εφαρμογής
- JGraph graph : ο γράφος του δέντρου δενδρικής μορφής
- GraphModel model : το μοντέλο του δέντρου δενδρικής μορφής

- DefaultGraphCell rootNode : ο κόμβος ρίζα του δέντρου δενδρικής μορφής
- DefaultGraphCell vectorOutputNodes : οι κόμβοι αποτελέσματος που εξάγονται μετά από την αποτίμηση ενός ερωτήματος στο δέντρο αυτό.

Βασικές δομές για την διαχείριση των στοιχείων των κόμβων

- Vector<DefaultGraphCell> streamVertices : Το διάνυσμα στο οποίο αποθηκεύουμε όλους τους κόμβους του δέντρου.
- Hashtable<Integer, DataVertex> id2dataVertex : Η δομή που αποθηκεύουμε για κάθε κόμβο το ζευγάρι {κωδικός κόμβου, δεδομένα κόμβου}
- Hashtable<DefaultGraphCell, DataVertex> vertex2dataVertex : Η δομή που αποθηκεύουμε για κάθε κόμβο το ζευγάρι {κόμβος, δεδομένα κόμβου}

- int num_nodes : μία αυξανόμενη μεταβλητή που δίνει σε κάθε κόμβο μοναδικό κωδικό. Αν δεν έχω διαγράψει κανένα κόμβο, τότε η μεταβλητή αυτή προσδιορίζει και το πλήθος των κόμβων του ερωτήματος

Δομές που σχετίζονται με την παραγωγή της κωδικοποίησης θέσης

- Hashtable<String, Integer> tag2id : Η δομή που αποθηκεύουμε για κάθε ετικέτα το ζευγάρι {όνομα ετικέτας, κωδικός ετικέτας}
- Hashtable<Integer, Integer> startNodeEncoding2idElement : Η δομή που αποθηκεύουμε για κάθε στοιχείο το ζευγάρι {$1^{η}$ επίσκεψη του στοιχείου, κωδικός στοιχείου}
- Hashtable<Integer, DefaultMutableTreeNode> idElement2TreeNode : Η δομή που αποθηκεύουμε για κάθε στοιχείο το ζευγάρι {κωδικός στοιχείου, κόμβος στο δέντρο στη μορφή ετικετών}
- Hashtable<Integer, Integer> idElement2idJGraph : Η δομή που αποθηκεύουμε για κάθε στοιχείο το ζευγάρι {κωδικός στοιχείου, κωδικός κόμβου στο δέντρο στη δενδρική μορφή}

Πεδία που σχετίζονται με τα χρώματα των κόμβων και των ακμών

- Color rootColor
- Color nodeColor
- Color edgeColor

Πεδία που αφορούν τον διάλογο που εμφανίζεται στο χρήστη κατά την προσθήκη πολλαπλών κόμβων

- JDialog dialog_addVertices : Το παράθυρο που εμφανίζεται για την προσθήκη πολλαπλών κόμβων
- JTable table : Ο πίνακας στον οποίο θα δώσουμε τα ονόματα των νέων κόμβων
- JButton button_ok_vertices : Το κουμπί για αποδοχή της προσθήκης των κόμβων με ονόματα αυτά που δόθηκαν στον πίνακα table
- JButton button_cancel_vertices : Το κουμπί για ματαίωση της προσθήκης

Πεδία που αφορούν τον διάλογο που εμφανίζεται στο χρήστη για την διαγραφή κόμβων

- JDialog dialog_deleteVertices : Το παράθυρο από όπου επιλέγουμε ποιους κόμβους θέλουμε να διαγράψουμε
- JList list_deleteVertices : Η λίστα από την οποία διαλέγουμε ποιους κόμβους θέλουμε να διαγράψουμε
- JButton button_ok_deleteVertices : Το κουμπί για αποδοχή της διαγραφής των κόμβων που επιλέξαμε από τη λίστα
- JButton button_cancel_deleteVertices : Το κουμπί για ματαίωση της διαγραφής

Πεδία που σχετίζονται με το μενού που εμφανίζεται με το δεξί κλικ του ποντικιού

- JPopupMenu popupMenu
- JMenuItem add_vertex_child, add_manyVertices_tree, add_middleVertex, delete_vertex, delete_edge, replace_parent;
- JMenuItem show_vertex, show_locationVertex, show_edgeTree, select_descendants;

Construstor

- VerticesTree_JGraph(Trees_and_Queries gui, JGraph graph, GraphModel model) : Αρχικοποιώ τα πεδία της κλάσης και δημιουργώ το μενού που εμφανίζεται με το δεξί κλικ του ποντικιού

Methods

Προσθήκη

- public DefaultGraphCell createRootVertex(String rootName): Δημιουργεί τον κόμβο-ρίζα του δέντρου δενδρικής μορφής. Επιστρέφει τη ρίζα.
- public DefaultGraphCell createVertex(String name, double x, double y, double w, double h, Color bg) : Δημιουργεί ένα νέο κόμβο με όνομα name, χρώματος bg, μήκους w, πλάτους h και τον τοποθετεί στην οριζόντια θέση y και στην κάθετη x. Επιστρέφει το νέο κόμβο
- public DefaultGraphCell storeDataVertex(DefaultGraphCell vertex, DefaultGraphCell vertexParent, int level) : Αποθηκεύει στις κατάλληλες δομές τα δεδομένα του κόμβου vertex. Επιστρέφει το κόμβο vertex.
- public boolean addVertexChild() : Προσθέτει ένα νέο κόμβο ως παιδί του επιλεγμένου κόμβου. Αν στο γράφο δεν έχει επιλεχθεί κόμβος ή δεν είναι μοναδικός τότε επιστρέφει false. Αν δεν δοθεί έγκυρο όνομα στον κόμβο επιστρέφει false. Αν γίνει η προσθήκη του κόμβου επιστρέφει true.

- public boolean addVertexMiddleChild() : Προσθέτει ένα νέο κόμβο ως παιδί της πηγής της επιλεγμένης ακμής και ως πατέρας του προορισμού. Δηλαδή επιλέγουμε μια ακμή και βάζουμε ενδιάμεσα ένα νέο κόμβο. Αν στο γράφο δεν έχει επιλεχθεί ακμή ή δεν είναι μοναδική τότε επιστρέφει false. Αν δεν δοθεί έγκυρο όνομα στον κόμβο επιστρέφει false. Αν γίνει η προσθήκη του κόμβου επιστρέφει true.
- public DefaultGraphCell createChildVertex(DefaultGraphCell vertexParent, String childName) : Προσδιορίζει τη θέση του νέου κόμβου σύμφωνα με τη θέση του πατέρα του. Μετά δημιουργεί το νέο κόμβο και τον αποθηκεύει.
- public DefaultGraphCell createChildVertex(DefaultGraphCell vertexSource, DefaultGraphCell vertexTarget, String childName, boolean isMiddleChild) : Προσδιορίζει τη θέση του νέου κόμβου σύμφωνα με τη θέση του πατέρα του και αν πρόκειται για ενδιάμεσος κόμβος σύμφωνα και με τη θέση του παιδιού του. Μετά δημιουργεί το νέο κόμβο.

- public void addMultipleVertices() : Προσθέτει πολλούς κόμβους ως παιδιά του επιλεγμένου κόμβου. Αρχικά ελέγχω αν στο δέντρο υπάρχει μοναδικός επιλεγμένος κόμβος. Αν όχι, η συνάρτηση επιστρέφει. Αν ναι, τότε ο μοναδικός αυτός κόμβος θα γίνει ο πατέρας των κόμβων που θα προσθέσει ο χρήστης. Έπειτα ζητείται από το χρήστη να δώσει το πλήθος των κόμβων που θέλει να προσθέσει. Αν η τιμή που θα δώσει δεν είναι ακέραιος τότε η συνάρτηση επιστρέφει. Αν δώσει τον αριθμό ένα τότε καλείται η συνάρτηση addVertex() που είδαμε παραπάνω για την προσθήκη ενός κόμβου. Αλλιώς καλείται η συνάρτηση addNodes_table(num_vertices) (βλέπε παρακάτω)
- public void addNodes_table(int num_vertices) : Εμφανίζει έναν πίνακα με τόσες γραμμές όσες δηλώνει το όρισμα num_vertices και ζητάει από το χρήστη να τον γεμίσει με τα νέα ονόματα των κόμβων.
- public boolean addMultipleVertices_ok() : Προσθέτει τους νέους κόμβους ως παιδιά του επιλεγμένου κόμβου. Πατέρας των νέων κόμβων είναι ο μοναδικός επιλεγμένος κόμβος του δέντρου. Παίρνει μία-μία τις γραμμές του πίνακα και διαβάζει το όνομα της κάθε γραμμής. Αν το όνομα είναι κενό ή δεν είναι έγκυρο (βλέπε Ενότητα 5.3.4.3) τότε δεν προστίθεται ο κόμβος. Οπότε μπορεί να μην προστεθούν τόσοι κόμβοι όσοι ζήτησε αρχικά ο χρήστης γιατί δεν έδωσε έγκυρα ονόματα. Επιστρέφει true αν γίνει η προσθήκη των κόμβων, αλλιώς false.

Διαγραφή

- public void deleteDataVertex(DataVertex dataRemovedVertex) : Διαγράφει τα στοιχεία του κόμβου dataRemovedVertex από τις κατάλληλες δομές. Αν ο κόμβος προς διαγραφή είναι η ρίζα τότε καλεί τη παρακάτω συνάρτηση deleteRoot.
- public void deleteRoot() : Διαγράφει όλο το δέντρο εκτός από τη ρίζα, αφού πάρει την έγκριση από το χρήστη. Αν ο χρήστης δεν θελήσει, επιστρέφει false, αλλιώς αν πει ναι και διαγραφεί όλο το υποδέντρο της ρίζας επιστρέφει true.
- public void dialogDeleteVertices_list() : Εμφανίζει το παράθυρο από όπου επιλέγονται από λίστα οι κόμβοι προς διαγραφή.
- public boolean deleteVertices(Object[] deleteCells) : Διαγράφει τους κόμβους deleteCells. Αρχικά ελέγχει αν έχει επιλεχθεί προς διαγραφή η ρίζα. Τότε καλεί τη συνάρτηση deleteRoot. Αλλιώς ρωτάει το χρήστη αν θέλει να διαγράψει τον κάθε επιλεγμένο κόμβο με όλο το υποδέντρο του ή αν θέλει να δια γράψει μόνο τον κάθε επιλεγμένο κόμβο ενώ στα παιδιά του ορίζεται ως πατέρας ο παππούς τους. Επιστρέφει true αν γίνει διαγραφή, αλλιώς false.
- public void deleteVertex_withDescendants(DataVertex dataRemovedVertex) : Διαγράφει τον κόμβο μαζί με όλο το υποδέντρο του καθώς και τις εισερχόμενες

ακμές του κόμβου για να μην μείνουν αιωρούμενες. Επιστρέφει true αν γίνει διαγραφή, αλλιώς false.

- public void deleteVertex_withoutDescendants(DataVertex dataRemovedVertex) : Διαγράφει μόνο τον κόμβο χωρίς το υποδέντρο του καθώς και τις εισερχόμενες ακμές του κόμβου για να μην μείνουν αιωρούμενες. Το υποδέντρο συνδέεται με τον πατέρα του κόμβου, δηλαδή όλα τα παιδιά του κόμβου παίρνουν ως πατέρα τον παππού τους. Επιστρέφει true αν γίνει διαγραφή, αλλιώς false.
- public void removeIncomingEdges(DefaultGraphCell vertex) : Διαγράφει τις εισερχόμενες ακμές ενός κόμβου, μετά από διαγραφή του κόμβου.
- public void fixLevel(DataVertex dataRemovedVertex, boolean increase) : Διορθώνει το πεδίο επίπεδο του κόμβου στο δέντρο. Αυξάνεται κατά ένα όταν προστίθεται ενδιάμεσος κόμβος, ενώ μειώνεται κατά ένα όταν διαγράφεται πρόγονος του κόμβου χωρίς να διαγραφεί το υποδέντρο του.

- public boolean replaceParent() : Αντικαθιστά τον πατέρα του επιλεγμένου κόμβου. Αν στο δέντρο δεν υπάρχει μοναδικός επιλεγμένος κόμβος, τότε η συνάρτηση επιστρέφει false. Αλλιώς εμφανίζεται μια λίστα στο χρήστη που περιέχει όλους τους κόμβους του δέντρου και του ζητείται να επιλέξει τον νέο πατέρα. Προεπιλεγμένος κόμβος στη ρίζα ορίζεται ο τωρινός πατέρας του κόμβου. Αν δεν επιλέχθηκε νέος πατέρας τότε επιστρέφει false. Αν ο κόμβος που επέλεξε είναι απόγονος ή ταυτίζεται με τον επιλεγμένο κόμβο του δέντρου τότε η αλλαγή πατέρα δεν επιτρέπεται. Επιστρέφει true αν γίνει η αντικατάσταση πατέρα, αλλιώς false.

- public DefaultGraphCell getSelectedVertex(DefaultGraphCell defaultSelectedVertex, String message, String title) : Εμφανίζει σε λίστα όλους τους κόμβους του δέντρου. Έχει σαν προεπιλεγμένο κόμβο τον defaultSelectedVertex. Το όρισμα message δίνει ένα μήνυμα στο χρήστη για το λόγο επιλογής του κόμβου. Το όρισμα title δίνει τον τίτλο του παραθύρου. Επιστρέφει null αν δεν επιλεχθεί κανένας κόμβος, αλλιώς επιστρέφει τον κόμβο που επιλέχθηκε.

- public void clearAnswers() : Καθαρίζει το δέντρο από τυχόν κόμβους αποτελέσματος προηγούμενης αποτίμησης.

Απόγονοι κόμβου

- public Vector<DefaultGraphCell> showDescendants(DefaultGraphCell parent) : Βρίσκει όλους τους απογόνους του κόμβου parent και τους τυπώνει. Επιστρέφει τους απογόνους σε ένα διάνυσμα.
- public Vector<DefaultGraphCell> selectDescendants(DefaultGraphCell parent) : Βρίσκει όλους τους απογόνους του κόμβου parent και τους επιλέγει στο γράφο. Επιστρέφει τους απογόνους σε ένα διάνυσμα
- public Vector<DefaultGraphCell> getDescendants(DefaultGraphCell parent) : Βρίσκει όλους τους απογόνους του κόμβου parent και τους επιστρέφει σε ένα διάνυσμα
- public boolean isDescendant(DefaultGraphCell vertex, DefaultGraphCell descendant, boolean showResult) : Ελέγχει αν ο κόμβος vertex έχει ως απόγονο τον κόμβο descendant. Επιστρέφει true αν ισχύει, αλλιώς false. Αν το όρισμα showResult είναι true τότε εμφανίζεται και κατάλληλο μήνυμα.
- public void checkRelationshipBetween2Nodes(DefaultGraphCell vertex1, DefaultGraphCell vertex2) : Ελέγχει ποια είναι η σχέση μεταξύ των 2 κόμβων

(ταυτίζονται, vertex1 πρόγονος ή απόγονος του vertex2) και εμφανίζει κατάλληλο μήνυμα.

Εμφάνιση δεδομένων

- public void showDataVertex(DefaultGraphCell vertex) : Εμφανίζει τα στοιχεία του κόμβου vertex.
- public void showLocationVertex(DefaultGraphCell vertex) : Εμφανίζει τη θέση του κόμβου vertex.
- public void showEdgeTree(DefaultEdge edge) : Εμφανίζει τους κόμβους πηγής και προορισμού της ακμής edge.

- public String showChildren(DefaultGraphCell parent) : Τυπώνει όλο το υποδέντρο (παιδιά και απογόνους) του κόμβου parent σε μορφή εμφωλιασμένη.
- public String showChildren_name(DefaultGraphCell parent) : Τυπώνει τα ονόματα όλων των κόμβων του υποδέντρου (παιδιά και απογόνους) του κόμβου parent.

- public void actionPerformed(ActionEvent evt) : Καλείται όταν ο χρήστης εκτελέσει κάποια ενέργεια και αυτή με τη σειρά της καλεί την απαραίτητη μέθοδο.

Χειρισμός μενού με δεξί κλικ ποντικιού

- public void createPopupMenu()
- public void createPopupMenu_vertex()
- public void createPopupMenu_edge()
- public void PopupMenuItems()

class PopupListener extends MouseAdapter

Fields

- JPopupMenu popup

Constructors

- PopupListener(JPopupMenu popupMenu)

Methods

- public void mousePressed(MouseEvent e)
- public void mouseReleased(MouseEvent e)
- private void maybeShowPopup(MouseEvent e) : Εμφανίζει το κατάλληλο μενού κάνοντας δεξί κλικ στο ποντίκι, ανάλογα με τί έχει επιλεχθεί. Αν έχει επιλεχθεί κόμβος, εμφανίζει το μενού για κόμβους. Αν έχει επιλεχθεί ακμή, εμφανίζει το μενού για ακμές.

Class Edges

Δημιουργεί τις ακμές πατέρα-παιδιού ενός δέντρου δενδρικής μορφής.

Field

- GraphModel model : το μοντέλο του δέντρου δενδρικής μορφής

Constructor

- Edges(GraphModel model)

Methods

- public DefaultEdge createEdge(String name, Color bg, DefaultGraphCell source, DefaultGraphCell target) : Δημιουργεί μια νέα ακμή στο γράφο του δέντρου δενδρικής μορφής με όνομα name, χρώματος bg και με κόμβο-πηγή source και κόμβο προορισμού target. Αν ο κόμβος-πηγή ή/και ο κόμβος προορισμού είναι κενοί επιστρέφει null. Αν ο κόμβος-πηγή και ο κόμβος προορισμού ταυτίζονται επιστρέφει null. Αλλιώς επιστρέφει την ακμή που δημιουργεί.

VerticesQuery implements ActionListener

Διαχειρίζεται όλα τα δεδομένα-κόμβους ενός ερωτήματος

inner class **DataVertex**

Κρατάει τα δεδομένα ενός κόμβου του ερωτήματος.

Fields

- int id : κωδικός κόμβου vertex
- String name : όνομα κόμβου vertex
- DefaultGraphCell vertex : ο τρέχων κόμβος
- DefaultGraphCell vertexChild : ο κόμβος-παιδί του κόμβου vertex. Αν ο κόμβος vertex δεν έχει παιδί, τότε το πεδίο αυτό είναι null.
- DefaultGraphCell vertexParent : ο κόμβος-πατέρας του κόμβου vertex. Αν ο κόμβος vertex δεν έχει πατέρα, τότε το πεδίο αυτό είναι null.

Constructor

- public DataVertex()

Fields

- Trees_and_Queries gui : το αντικείμενο της βασικής εφαρμογής
- JGraph graph : ο γράφος του ερωτήματος
- GraphModel model : το μοντέλο του ερωτήματος
- Edges_child_descendant ourEdges : οι ακμές του ερωτήματος

- DefaultGraphCell rootNode : ο κόμβος ρίζα του ερωτήματος
- DefaultGraphCell outputNode : ο κόμβος εξόδου του ερωτήματος. Αν δεν έχει οριστεί, τότε το πεδίο αυτό έχει την τιμή null

Βασικές δομές για την διαχείριση των στοιχείων των κόμβων

- Vector<DataVertex> streamVertices : Το διάνυσμα στο οποίο αποθηκεύουμε τα δεδομένα του κάθε κόμβου
- Hashtable<Integer, DataVertex> id2dataVertex : Η δομή που αποθηκεύουμε για κάθε κόμβο το ζευγάρι {κωδικός κόμβου, δεδομένα κόμβου}
- Hashtable<DefaultGraphCell, DataVertex> vertex2dataVertex : Η δομή που αποθηκεύουμε για κάθε κόμβο το ζευγάρι {κόμβος, δεδομένα κόμβου}

- int num_nodes : μία αυξανόμενη μεταβλητή που δίνει σε κάθε κόμβο μοναδικό κωδικό. Αν δεν έχω διαγράψει κανένα κόμβο, τότε η μεταβλητή αυτή προσδιορίζει και το πλήθος των κόμβων του ερωτήματος

Βοηθητικά πεδία για τον προσδιορισμό της θέσης των κόμβων

- int loc
- int nodes_y
- double locationY

Πεδία που σχετίζονται με τα χρώματα των κόμβων και των ακμών

- Color nodeColor
- Color edgeColor
- Color rootColor
- Color outputColor

Πεδία που αφορούν τον διάλογο που εμφανίζεται στο χρήστη κατά την προσθήκη πολλαπλών κόμβων

- JDialog dialog_addVertices : Το παράθυρο που εμφανίζεται για την προσθήκη πολλαπλών κόμβων
- JTable table : Ο πίνακας στον οποίο θα δώσουμε τα ονόματα των νέων κόμβων
- JButton button_ok_vertices : Το κουμπί για αποδοχή της προσθήκης των κόμβων με ονόματα αυτά που δόθηκαν στον πίνακα table
- JButton button_cancel_vertices : Το κουμπί για ματαίωση της προσθήκης

Πεδία που αφορούν τον διάλογο που εμφανίζεται στο χρήστη για την διαγραφή κόμβων

- JDialog dialog_deleteVertices : Το παράθυρο από όπου επιλέγουμε ποιους κόμβους θέλουμε να διαγράψουμε
- JList list_deleteVertices : Η λίστα από την οποία διαλέγουμε ποιους κόμβους θέλουμε να διαγράψουμε
- JButton button_ok_deleteVertices : Το κουμπί για αποδοχή της διαγραφής των κόμβων που επιλέξαμε από τη λίστα
- JButton button_cancel_deleteVertices : Το κουμπί για ματαίωση της διαγραφής

Πεδία που σχετίζονται με το μενού που εμφανίζεται με το δεξί κλικ του ποντικιού

- JPopupMenu popupMenu
- JMenuItem add_vertexChild

- JMenuItem add_vertexDescendant
- JMenuItem add_edgeChild
- JMenuItem add_edgeDescendant
- JMenuItem delete_vertex
- JMenuItem delete_edge
- JMenuItem delete_vertexEdges
- JMenuItem select_output_node

Construstor

- VerticesQuery(Trees_and_Queries gui, JGraph graph, GraphModel model, Edges_child_descendant ourEdges) : Αρχικοποιώ τα πεδία της κλάσης και δημιουργώ το μενού που εμφανίζεται με το δεξί κλικ του ποντικιού

Methods

Προσθήκη

- public DefaultGraphCell addVertex() : Προσθέτει ένα νέο κόμβο στο ερώτημα. Αν ο χρήστης δεν δώσει έγκυρο όνομα τότε δεν προστίθεται ο κόμβος και επιστρέφει null, αλλιώς το κόμβο που προστέθηκε.
- public DefaultGraphCell locationVertex() : Ορίζει σε ποια θέση του παραθύρου θα τοποθετήσει το νέο κόμβο τον οποίο και επιστρέφει.
- public DefaultGraphCell createRootVertex(String rootName) : Δημιουργεί το κόμβο ρίζα του ερωτήματος και τον επιστρέφει.
- public DefaultGraphCell createVertex(String name, double x, double y, double w, double h, Color bg) : Δημιουργεί ένα νέο κόμβο με όνομα name, χρώματος bg, μήκους w, πλάτους h και τον τοποθετεί στην οριζόντια θέση y και στην κάθετη x. Επιστρέφει το νέο κόμβο
- public DefaultGraphCell storeDataVertex(DefaultGraphCell vertex) : Αποθηκεύει στις κατάλληλες δομές τα δεδομένα του κόμβου vertex. Επιστρέφει το κόμβο vertex

- public void addMultipleVertices() : Προσθέτει πολλούς κόμβους. Αρχικά ζητείται από το χρήστη να δώσει το πλήθος των κόμβων που θέλει να προσθέσει. Αν η τιμή που θα δώσει δεν είναι ακέραιος τότε η συνάρτηση επιστρέφει. Αν δώσει τον αριθμό ένα τότε καλείται η συνάρτηση addVertex() που είδαμε παραπάνω για την προσθήκη ενός κόμβου. Αλλιώς καλείται η συνάρτηση addNodes_table(num_vertices) (βλέπε παρακάτω)
- public void addNodes_table(int num_vertices) : Εμφανίζει έναν πίνακα με τόσες γραμμές όσες δηλώνει το όρισμα num_vertices και ζητάει από το χρήστη να τον γεμίσει με τα νέα ονόματα των κόμβων.
- public boolean addMultipleVertices_ok() : Προσθέτει τους νέους κόμβους στο ερώτημα. Παίρνει μία-μία τις γραμμές του πίνακα και διαβάζει το όνομα της κάθε γραμμής. Αν το όνομα είναι κενό ή δεν είναι έγκυρο (βλέπε Ενότητα 5.3.4.3) τότε δεν προστίθεται ο κόμβος. Οπότε μπορεί να μην προστεθούν τόσοι κόμβοι όσοι ζήτησε αρχικά ο χρήστης γιατί δεν έδωσε έγκυρα ονόματα.

- public boolean addRelationship_newNode(boolean is_child) : Προσθέτει ένα κόμβο και μια ακμή όπως φαίνεται ακολούθως. Δημιουργεί ένα νέο κόμβο στον οποίο ορίζει πατέρα ή απόγονο (ανάλογα με την τιμή του ορίσματος is_child) τον επιλεγμένο

κόμβο. Αν είναι επιτυχής η προσθήκη του νέου κόμβου και της νέας ακμής, τότε επιστρέφει true αλλιώς false.

- public boolean addRelationship_existingNode(boolean is_child) : Προσθέτει μια ακμή όπως φαίνεται ακολούθως. Επιλέγει έναν κόμβο του ερωτήματος από μια λίστα στον οποίο ορίζει πατέρα ή απόγονο (ανάλογα με την τιμή του ορίσματος is_child) τον επιλεγμένο κόμβο. Αν είναι επιτυχής η προσθήκη της νέας ακμής, τότε επιστρέφει true αλλιώς false.

Διαγραφή

- public boolean deleteVertices(Object[] deleteCells, Edges_child_descendant ourEdges) : Διαγράφει τους επιλεγμένους κόμβους και τις συνδεδεμένες τους ακμές. Αν γίνει διαγραφή, επιστρέφει true, αλλιώς false.
- public boolean remove_connectedEdges(DefaultGraphCell vertex, Edges_child_descendant ourEdges) : Παίρνει τη λίστα από τις συνδεδεμένες ακμές του κόμβου vertex και τις διαγράφει από το μοντέλο. Αν υπάρχουν ακμές προς διαγραφή τότε επιστρέφει true, αλλιώς επιστρέφει false
- public List<DefaultEdge> connectedEdges(DefaultGraphCell vertex, Edges_child_descendant ourEdges, boolean remove) : Βρίσκει τις συνδεδεμένες ακμές του κόμβου vertex και τις επιστρέφει σε μια λίστα. Αν το όρισμα remove είναι true τότε διαγράφει τις ακμές από τις δομές που είναι αποθηκευμένες.
- public void deleteDataVertex(DataVertex dataRemovedVertex) : Διαγράφει τα στοιχεία του κόμβου dataRemovedVertex από τις κατάλληλες δομές.
- public void dialogDeleteVertices(Edges_child_descendant ourEdges) : Επιλέγει από μια λίστα ποιους κόμβους θέλει να διαγράψει.

Εμφάνιση δεδομένων

- public void showDataVertex(DefaultGraphCell vertex) : Εμφανίζει τα στοιχεία του κόμβου vertex.
- public void showLocationVertex(DefaultGraphCell vertex) : Εμφανίζει τη θέση του κόμβου vertex.

Κόμβος εξόδου

- public void outputVertex(DefaultGraphCell vertex) : Ορίζει ως κόμβο εξόδου τον κόμβο vertex.

- public static int getNumEdges(GraphModel, DefaultGraphCell vertex) : Επιστρέφει το πλήθος των συνολικών ακμών του κόμβου vertex.
- public static void checkCanonicalForm(GraphModel model, VerticesQuery ourVertices, Edges_child_descendant ourEdges) : Βρίσκει τους κόμβους που δεν έχουν εισερχόμενες ακμές, αν υπάρχουν, (εκτός της ρίζας που δεν μπορεί να έχει) και τους ενώνει με τη ρίζα ή με παιδί της.
- public static void checkCanonicalForm(JGraph graph, GraphModel model, VerticesQuery ourVertices, Edges_child_descendant ourEdges) : Βρίσκει τους κόμβους που δεν έχουν εισερχόμενες ακμές, αν υπάρχουν, (εκτός της ρίζας που δεν μπορεί να έχει) και τους ενώνει με τη ρίζα ή με παιδί της. Στο τέλος επιλέγονται στο γράφο αυτές οι νέες ακμές.
- public static boolean isPathQuery(GraphModel model, VerticesQuery ourVertices) : Επιστρέφει true αν πρόκειται για πλήρως ορισμένο ερώτημα μονοπατιού, αλλιώς επιστρέφει false.

- public void actionPerformed(ActionEvent evt) : Καλείται όταν ο χρήστης εκτελέσει κάποια ενέργεια και αυτή με τη σειρά της καλεί την απαραίτητη μέθοδο.

Χειρισμός μενού με δεξί κλικ ποντικιού

- public void createPopupMenu()
- public void createPopupMenu_vertex()
- public void createPopupMenu_edge()
- public void PopupMenuItems()

class PopupListener extends MouseAdapter

Fields

- JPopupMenu popup

Constructors

- PopupListener(JPopupMenu popupMenu)

Methods

- public void mousePressed(MouseEvent e)
- public void mouseReleased(MouseEvent e)
- private void maybeShowPopup(MouseEvent e) : Εμφανίζει το κατάλληλο μενού κάνοντας δεξί κλικ στο ποντίκι, ανάλογα με τί έχει επιλεχθεί. Αν έχει επιλεχθεί κόμβος, εμφανίζει το μενού για κόμβους. Αν έχει επιλεχθεί ακμή, εμφανίζει το μενού για ακμές.

Class GPCellViewFactory extends DefaultCellViewFactory

& Class JGraphEllipseView extends VertexView

Δίνουν στους κόμβους των ερωτημάτων κυκλικό σχήμα.

Class Edges_child_descendant implements ActionListener

Διαχειρίζεται όλες τις ακμές ενός ερωτήματος

inner class ***DataEdge***

Κρατάει τα δεδομένα μιας ακμής του ερωτήματος.

Fields

- int id : κωδικός edge
- String name : όνομα ακμής edge (συνήθως είναι null)
- DefaultEdge edge : ο τρέχων κόμβος

- boolean isChild : true αν είναι σχέση πατέρα-παιδιού, false αν είναι σχέση προγόνου-απόγονου.
- DefaultGraphCell source : ο κόμβος-πηγή της ακμής edge. Το πεδίο αυτό δεν μπορεί να είναι null.
- DefaultGraphCell target : ο κόμβος προορισμού της ακμής edge. Το πεδίο αυτό δεν μπορεί να είναι null.

Constructor

- public DataEdge()

Fields

- Trees_and_Queries gui : το αντικείμενο της βασικής εφαρμογής
- JGraph graph : ο γράφος του ερωτήματος
- GraphModel model : το μοντέλο του ερωτήματος
- VerticesQuery ourVertices : οι κόμβοι του ερωτήματος.

Βασικές δομές για την αποθήκευση των στοιχείων των ακμών

- Vector<DataEdge> streamEdges : Το διάνυσμα στο οποίο αποθηκεύουμε τα δεδομένα/στοιχεία της κάθε ακμής
- Hashtable<Integer, DefaultEdge> id2edge : Η δομή που αποθηκεύουμε για κάθε ακμή το ζευγάρι {κωδικός ακμής, ακμή}
- Hashtable<DefaultEdge, DataEdge> edge2dataEdge : Η δομή που αποθηκεύουμε για κάθε ακμή το ζευγάρι {ακμή, δεδομένα/στοιχεία ακμής}

- int num_edges : μία αυξανόμενη μεταβλητή που δίνει σε κάθε ακμή μοναδικό κωδικό. Αν δεν έχω διαγράψει καμία ακμή, τότε η μεταβλητή αυτή προσδιορίζει και το πλήθος των ακμών του ερωτήματος

- boolean relationshipChild : προσδιορίζει τη σχέση που παριστάνει η ακμή. Αν είναι true πρόκειται για σχέση πατέρα-παιδιού, αν είναι false πρόκειται για σχέση προγόνου-απογόνου.

Πεδία που αφορούν τον διάλογο που εμφανίζεται στο χρήστη για την προσθήκη ακμής

- JDialog dialogEdge
- JRadioButton child, descendant
- ButtonGroup relationship
- JTextField textfield_edgeName
- JComboBox comboBox_edgeSource, comboBox_edgeTarget
- JButton button_ok_EdgeInfo, button_cancel_EdgeInfo

Πεδία που αφορούν τον διάλογο που εμφανίζεται στο χρήστη για την διαγραφή ακμών

- JDialog dialogDeleteEdges
- JList list_deleteEdges, list_deleteEdges_list
- JButton button_ok_deleteEdges_list, button_cancel_deleteEdges_list

Πεδία που αφορούν το παράθυρο που εμφανίζεται στο χρήστη για την παρουσίαση των ακμών

- JDialog dialogShowEdges
- JTable table_showEdges
- JButton button_showEdges

Construstor

- public Edges_child_descendant(Trees_and_Queries gui, JGraph graph, GraphModel model, VerticesQuery ourVertices) : Αρχικοποιεί τα πεδία της κλάσης.

Methods

- public DefaultEdge createEdge (String name, Color bg, boolean relationshipChild, DefaultGraphCell source, DefaultGraphCell target) : Δημιουργεί τη νέα ακμή αφού κάνει πρώτα σημαντικούς ελέγχους (βλέπε Ενότητα 7.2.3). Επιστρέφει την ακμή που προσθέτει, αλλιώς null.

- public DefaultEdge addEdge(String name, Color bg, boolean relationshipChild, DefaultGraphCell source, DefaultGraphCell target) : Δημιουργεί και προσθέτει τη νέα ακμή με πηγή τον κόμβο source και προορισμό τον κόμβο target. Το είδος της σχέσης προσδιορίζεται από το όρισμα relationshipChild. Η ακμή έχει χρώμα bg και όνομα name που συνήθως όμως είναι null. Επιστρέφει την ακμή που προσθέτει.

- public DefaultEdge storeDataEdge(DefaultEdge edge, boolean isChild, DefaultGraphCell source, DefaultGraphCell target) : Αποθηκεύει τα στοιχεία της νέας ακμής στις βασικές δομές της κλάσης. Ακόμα αν πρόκειται για νέα σχέση πατέρα-παιδιού τότε επεξεργάζεται όλες τις σχέσεις προγόνου-απογόνου που συνδέονται στον πατέρα και το παιδί (βλέπε Ενότητα 7.2.3). Επιστρέφει την ακμή που αποθηκεύει.

- public void fixOutgoingDescendants(DefaultGraphCell source, DefaultGraphCell target) : Επεξεργάζεται τους απογόνους της πηγής-πατέρα της νέας σχέσης πατέρα-παιδιού.
- public void fixIncomingDescendants(DefaultGraphCell source, DefaultGraphCell target) : Επεξεργάζεται τους προγόνους του προορισμού-παιδιού της νέας σχέσης πατέρα-παιδιού.

Διαγραφή ακμών

- pubic boolean deleteDataEdge(DataEdge dataRemovedEdge) : Διαγράφει τα δεδομένα dataRemovedEdge μιας ακμής. Επιστρέφει true αν γίνει η διαγραφή, αλλιώς false.
- pubic boolean deleteDataEdge(DefaultEdge removedEdge) : Διαγράφει τα δεδομένα μιας ακμής removedEdge. Επιστρέφει true αν γίνει η διαγραφή, αλλιώς false
- pubic boolean deleteEdges(Object[] deleteEdges, boolean select) : Διαγράφει τις ακμές deleteEdges. Επιστρέφει true αν γίνει διαγραφή, αλλιώς false.
- pubic boolean deleteEdges_list(Object[] deleteEdges) : Διαγράφει τις ακμές deleteEdges από λίστα. Επιστρέφει true αν γίνει διαγραφή, αλλιώς false.

- public void dialogEdgeData() : Το παράθυρο που δίνω τα χαρακτηριστικά της νέας ακμής (πηγή, προορισμό, όνομα, είδος σχέσης)

- public void dialogDeleteEdges_list() : Το παράθυρο από όπου ο χρήστης επιλέγει από λίστα τις ακμές που θέλει να διαγράψει.

- public void dialogShowEdges_list() : Το παράθυρο όπου εμφανίζονται σε λίστα όλες οι ακμές του ερωτήματος.
- public void dialogShowEdges_table() : Το παράθυρο όπου εμφανίζονται σε πίνακα όλες οι ακμές του ερωτήματος.

- public void actionPerformed(ActionEvent evt) : Καλείται όταν ο χρήστης εκτελέσει κάποια ενέργεια και αυτή με τη σειρά της καλεί την απαραίτητη μέθοδο.

Class Cells implements ActionListener

Διαχειρίζεται τους κόμβους και τις ακμές ενός ερωτήματος. Παρουσιάζει και διαγράφει κόμβους και ακμές σαν σύνολο.

Fields

- Trees_nad_Queries gui : το αντικείμενο της βασικής εφαρμογής
- NewQuery.DataQuery dataQuery : οι κόμβοι του ερωτήματος.

Πεδία που αφορούν το παράθυρο που εμφανίζεται στο χρήστη για την παρουσίαση των κόμβων και των ακμών του ερωτήματος.

- JDialog dialogShowCells : το παράθυρο που παρουσιάζει τους κόμβους και τις ακμές του ερωτήματος.
- JTabbedPane tabbedPane : περιέχει μια καρτέλα για τους κόμβους και μια καρτέλα για τις ακμές
- JList list_showVertices : Η λίστα που πειέχει τους κόμβους
- JList list_showEdges_child : Η λίστα που περιέχει τις σχέσεις πατέρα-παιδιού
- JList list_showEdges_descendant : Η λίστα που περιέχει τις σχέσεις προγόνου-απογόνου
- JButton button_showCells

Construstor

- public Cells(Trees_and_Queries gui, NewQuery.DataQuery dataQuery)

Methods

- public void ShowCells() : Το παράθυρο όπου εμφανίζονται σε δύο καρτέλες οι κόμβοι και οι ακμές του ερωτήματος.
- public JPanel PanelShowCells() : Το panel που περιέχει τις 2 καρτέλες με τους κόμβους και τις ακμές του ερωτήματος.

- public JPanel PanelNodes() : Το panel που περιέχει τους κόμβους του ερωτήματος
- pubic JPanel PanelEdges() : Το panel που περιέχει τις ακμές του ερωτήματος σε 2 λίστες, μία για τις σχέσεις πατέρα-παιδιού και μία για τις σχέσεις προγόνου-απογόνου.
- pubic void showCells() : Τυπώνει τα στοιχεία κάθε κόμβου και κάθε ακμής.

- public boolean removeAll() : Διαγράφει τους επιλεγμένους αντικείμενα του γράφου του ερωτήματος. Εξετάζει κάθε φορά αν το αντικείμενο είναο κόμβος ή ακμή. Αν είναι κόμβος τότε διαγράφει και τις συνδεδεμένες ακμές του. Ακόμα αν έχει επιλεχθεί η ρίζα για διαγραφή τότε ο χρήστης ενημερωνέται ότι απαγορεύεται αυτή η διαγραφή. Επιστρέφει true αν γίνει διαγραφή, αλλιώς false.

- public void actionPerformed(ActionEvent evt) : Καλείται όταν ο χρήστης εκτελέσει κάποια ενέργεια και αυτή με τη σειρά της καλεί την απαραίτητη μέθοδο.

Class Evaluation implements ActionListener

Fields

Τα δεδομένα του δέντρου και του ερωτήματος που θα αποτιμηθούν

- NewTree_JGraph.DataTree dataEvaluateTree_JGraph : Τα δεδομένα του δέντρου δενδρικής μορφής που αποτιμάται
- NewTree_JTree.DataTree dataEvaluateTree_JTree : Τα δεδομένα του δέντρου μορφής ετικετών που αποτιμάται
- NewQuery.DataQuery dataEvaluateQuery : Τα δεδομένα του ερωτήματος προς αποτίμηση

Ποιος αλγόριθμος θα επιλεχθεί για την αποτίμηση

- Algorithm algorithm : Ο αλγόριθμος με τον οποίο θα γίνει η αποτίμηση
- enum Algorithm{*PartialPathStack*, *PathStack*} : Οι αλγόριθμοι για αποτίμηση

Δομές για τη βοήθεια διάκρισης του δέντρου και του ερωτήματος αποτίμησης

Ανοιχτά δέντρα

- Hashtable<Integer, DataTab> idTreeCombo2dataTab : Η δομή που αποθηκεύουμε για κάθε καρτέλα που περιέχει δέντρο το ζευγάρι {κωδικός ανοιχτού δέντρου, δεδομένα καρτέλας }
- int idTreeCombo : Ο κωδικός του ανοιχτού δέντρου
- Hashtable<Integer, Component> idTreeCombo2component : Η δομή που αποθηκεύουμε για κάθε καρτέλα που περιέχει δέντρο το ζευγάρι {κωδικός ανοιχτού δέντρου, component } → για να επιλέγεται η καρτέλα του δέντρου προς αποτίμηση

Ανοιχτά ερωτήματα

- Hashtable<String, DataTab> nameQuery2dataTab : Η δομή που αποθηκεύουμε για κάθε καρτέλα που περιέχει ερώτημα το ζευγάρι {όνομα ερωτήματος, δεδομένα καρτέλας}
- Hashtable<String, Component> nameQueryCombo2component : Η δομή που αποθηκεύουμε για κάθε καρτέλα που περιέχει ερώτημα το ζευγάρι {όνομα ερωτήματος, component } → για να επιλέγεται η καρτέλα του ερωτήματος προς αποτίμηση.

- Component componentEvaluateTree : Το component του δέντρου προς αποτίμηση
- Component componentEvaluateQuery: Το component του ερωτήματος προς αποτίμηση

Πεδία που αφορούν τον διάλογο που εμφανίζεται στο χρήστη για την επιλογή του δέντρου, του ερωτήματος και του αλγορίθμου αποτίμησης

- JDialog dialog_evaluation : Το παράθυρο που εμφανίζεται στο χρήστη για την επιλογή του δέντρου, του ερωτήματος και του αλγορίθμου αποτίμησης
- JComboBox combo_tree : Το combobox για την επιλογή του δέντρου
- JComboBox combo_query : Το combobox για την επιλογή του ερωτήματος
- JComboBox combo_algorithm : Το combobox για την επιλογή του αλγορίθμου
- JButton button_ok_evaluation : Το κουμπί για αποδοχή της αποτίμησης
- JButton button_cancel_evaluation : Το κουμπί για ματαίωση της αποτίμησης

Πεδία που αφορούν τον διάλογο που εμφανίζεται στο χρήστη για τη διάκριση της μορφής του δέντρου που θέλουμε να ανοίξουμε για αποτίμηση

- JDialog dialog_selectTree : Το παράθυρο που εμφανίζεται στο χρήστη για την επιλογή της μορφής του δέντρου που θέλουμε να ανοίξουμε
- ButtonGroup group : Οι επιλογές της μορφής με την οποία θέλω να ανοίξω το δέντρο. Οι επιλογές είναι οι δύο παρακάτω
- final String commandTree_JTree = "tree_jtree"
- final String commandTree_JGraph = "tree_jgraph"
- JButton button_openSelectTree : Το κουμπί για αποδοχή του ανοίγματος
- JButton button_cancelSelectTree : Το κουμπί για ματαίωση του ανοίγματος

- Trees_and_Queries gui : Το αντικείμενο της βασικής εφαρμογής
- String rootName_tree: Το όνομα της ρίζας του δέντρου

Construstor

- public Evaluation(Trees_and_Queries gui)

Methods

- public void chooseFiles_and_algorithm_combobox() : Εμφανίζει σε ένα παράθυρο τρία combobox από τα οποία επιλέγει ο χρήστης το δέντρο, το ερώτημα και τον αλγόριθμο αποτίμησης. Στο combobox για το δέντρο υπάρχουν όλα τα ανοιχτά δέντρα της εφαρμογής και η δυνατότητα ανοίγματος και άλλου δέντρου. Στο

combobox για το ερώτημα υπάρχουν όλα τα ανοιχτά ερωτήματα της εφαρμογής και η δυνατότητα ανοίγματος και άλλου ερωτήματος.

- public void selectTree() : Εμφανίζει ένα παράθυρο στο οποίο ο χρήστης επιλέγει σε ποια μορφή θέλει να ανοίξει το νέο δέντρο, σε μορφή ετικετών ή σε δενδρική μορφή
- public void openComboFile(Application application) : Διακρίνει τι είδους αρχείο έχει ζητήσει ο χρήστης να ανοίξει και καλεί αναλόγως μία από τις 3 παρακάτω μεθόδους
- public void openComboTree_jtree() : Ανοίγει ένα νέο δέντρο μορφής ετικετών
- public void openComboTree_jgraph() : Ανοίγει ένα νέο δέντρο δενδρικής μορφής
- public void openComboQuery () : Ανοίγει ένα νέο ερώτημα

Βλέπε Ενότητα 7.2.6 για λεπτομέρειες της διαδικασίας της αποτίμησης

- public void evaluation() : Κάνει τους πρώτους ελέγχους για την αποτίμηση και εντοπίζει το δέντρο προς αποτίμηση.
- public void evaluate_JGraph(File directory, DataTab dataEvaluateTab_tree) : Αρχικά καθαρίζει το δέντρο από τυχόν κόμβους αποτελέσματος προηγούμενης αποτίμησης. Μετά παίρνει την κωδικοποίηση θέσης του δέντρου. Ελέγχει αν υπάρχει ήδη το αρχείο με την κωδικοποίηση θέσης και αν δεν έχουν γίνει αλλαγές στο δέντρο και αν ισχύουν αυτά το εντοπίζει και το αντιγράφει στον επιθυμητό προορισμό, αλλιώς το παράγει εκείνη τη στιγμή. Έπειτα καλεί τη παρακάτω συνάρτηση getQueryResults για την αποτίμηση και τέλος παρουσιάζει τα αποτελέσματα της αποτίμησης με την συνάρτηση processResults_JGraph.
- public void evaluate_JTree(File directory, DataTab dataEvaluateTab_tree) : Παίρνει την κωδικοποίηση θέσης του δέντρου. Ελέγχει αν υπάρχει ήδη το αρχείο με την κωδικοποίηση θέσης και αν δεν έχουν γίνει αλλαγές στο δέντρο και αν ισχύουν αυτά το εντοπίζει και το αντιγράφει στον επιθυμητό προορισμό, αλλιώς το παράγει εκείνη τη στιγμή. Έπειτα καλεί τη παρακάτω συνάρτηση getQueryResults για την αποτίμηση και τέλος παρουσιάζει τα αποτελέσματα της αποτίμησης με την συνάρτηση processResults_JTree.

- public String getQueryResults(File directory, Hashtable<String, Integer> tag2id) : Παίρνει το ερώτημα και ελέγχει αν όλοι οι κόμβοι του υπάρχουν στο δέντρο. Αν όχι, τότε η αποτίμηση σταματάει. Αντιγράφει το ερώτημα στον επιθυμητό προορισμό οπού αντί για ονόματα βάζει τους κωδικούς που αντιστοιχούν στους κόμβους του δέντρου. Εκτελεί την αποτίμηση με τον αλγόριθμο που έχει επιλεχθεί και επιστρέφει τα αποτελέσματα.
- public boolean checkVerticesNames(Hashtable<String, Integer> hash_tag2id_tree, Vector<VerticesQuery.DataVertex> streamVertices_query) : Ελέγχει αν τα ονόματα όλων των κόμβων του ερωτήματος υπάρχουν στους κόμβους του δέντρου. Η ρίζα του ερωτήματος έχει πάρει το όνομα της ρίζας του δέντρου. Αν έστω και ένα όνομα κόμβου ερωτήματος δεν υπάρχει στο δέντρο επιστρέφει false, αλλιώς true.

Κλήση εξωτερικών εκτελέσιμων σε γλώσσα C++ για την αποτίμηση του αντίστοιχου αλγορίθμου

- public String executeCpp_PartialPathStack_inMap(int idNode) : Εκτελεί τον PartialPathStack. Τα αποτελέσματα που επιστρέφονται είναι οι κωδικοί των κόμβων αποτελέσματος του δέντρου.
- public String executeCpp_PathStack_inMap(int idNode) : Εκτελεί τον PathStack. Τα αποτελέσματα που επιστρέφονται είναι οι κωδικοί των κόμβων αποτελέσματος του δέντρου.

- public String executeCpp_PartialPathStack () : Εκτελεί τον PartialPathStack. Τα αποτελέσματα που επιστρέφονται είναι όλα τα μονοπάτια αποτελέσματος όπου οι κόμβοι εκφράζονται με την κωδικοποίηση θέσης τους. Στο τέλος δίνεται και το πλήθος των αποτελεσμάτων και ο χρόνος που χρειάστηκε για την αποτίμηση.
- public String executeCpp_PathStack () : Εκτελεί τον PathStack. Τα αποτελέσματα που επιστρέφονται είναι όλα τα μονοπάτια αποτελέσματος όπου οι κόμβοι εκφράζονται με την κωδικοποίηση θέσης τους. Στο τέλος δίνεται και το πλήθος των αποτελεσμάτων και ο χρόνος που χρειάστηκε για την αποτίμηση.

Επεξεργασία αποτελεσμάτων στην περίπτωση δέντρου δενδρικής μορφής

- public void processResults_JGraph(String results, Hashtable<Integer, Integer> startNodeEncoding2idElement, Hashtable<Integer, Integer> idElement2idJGraph) : Επεξεργάζεται τα αποτελέσματα results, εντοπίζει τους κόμβους αποτελέσματος και εμφανίζει τα αποτελέσματα αν υπάρχουν καλώντας την επόμενη συνάρτηση. Αν τα αποτελέσματα είναι μηδενικά τότε ενημερώνει το χρήστη και η διαδικασία σταματάει εδώ.
- public void showResults_JGraph(Vector<DefaultGraphCell> vector_outputVertices) : Καταρχήν δηλώνει τους κόμβους αποτελέσματος στο αρχικό δέντρο με πράσινο χρώμα. Μετά δημιουργεί ένα νέο δέντρο-αποτέλεσμα μορφής ίδιας με το αρχικό, οπότε δενδρικής μορφής, και ορίζει τη ρίζα. Τοποθετεί ως παιδιά της ρίζας τους κόμβους αποτελέσματος. Ακόμα καλώντας τη συνάρτηση storeResultVertex, τοποθετεί στο δέντρο-αποτέλεσμα κάτω από κάθε κόμβο αποτελέσματος το υποδέντρο που είχε στο αρχικό δέντρο. Τέλος χρωματίζει και στο δέντρο-αποτέλεσμα τους κόμβους αποτελέσματος με πράσινο χρώμα.
- public void storeSubtreeResultVertex_JGraph (NewTree_JGraph.DataTree dataResultTree, DefaultGraphCell vertexParent, DefaultGraphCell resultVertex) : Προσθέτει το resultVertex ως παιδί του vertexParent στο δέντρο του dataResultTree. Μετά παίρνει από το αρχικό δέντρο τα παιδιά του resultVertex και καλείται αναδρομικά η συνάρτηση αυτή. Οπότε στο τέλος θα υπάρχει όλο το υποδέντρο του κόμβου αποτελέσματος και στο δέντρο-αποτέλεσμα.

Επεξεργασία αποτελεσμάτων στην περίπτωση δέντρου μορφής ετικετών

- public void processResults_JTree(String results, Hashtable<Integer, Integer> startNodeEncoding2idElement, Hashtable<Integer, DefaultMutableTreeNode> idElement2TreeNode) : Επεξεργάζεται τα αποτελέσματα results, εντοπίζει τους κόμβους αποτελέσματος και εμφανίζει τα αποτελέσματα αν υπάρχουν καλώντας την επόμενη συνάρτηση. Αν τα αποτελέσματα είναι μηδενικά τότε ενημερώνει το χρήστη και η διαδικασία σταματάει εδώ.
- public void showResults_JTree(Vector<DefaultMutableTreeNode> vector_outputVertices) : Δημιουργεί ένα νέο δέντρο-αποτέλεσμα μορφής ίδιας με το αρχικό, οπότε μορφής ετικετών, και ορίζει τη ρίζα. Τοποθετεί ως παιδιά της ρίζας τους κόμβους αποτελέσματος. Ακόμα καλώντας τη συνάρτηση storeSubtreeResultVertex, τοποθετεί στο δέντρο-αποτέλεσμα κάτω από κάθε κόμβο αποτελέσματος το υποδέντρο που είχε στο αρχικό δέντρο.
- public DefaultMutableTreeNode storeSubtreeResultVertex_JTree(NewTree_JTree.DataTree dataResultTree, DefaultMutableTreeNode parentNode, DefaultMutableTreeNode answerNode) : Προσθέτει το answerNode ως παιδί του parentNode στο δέντρο του dataResultTree. Μετά παίρνει από το αρχικό δέντρο τα παιδιά του answerNode και καλείται αναδρομικά η συνάρτηση αυτή. Οπότε στο τέλος θα υπάρχει όλο το υποδέντρο του κόμβου αποτελέσματος και στο δέντρο-αποτέλεσμα.

- public static void copy(File source, File destination) : Αντιγράφει το αρχείο source στο αρχείο destination.

- public void actionPerformed(ActionEvent evt) : Καλείται όταν ο χρήστης εκτελέσει κάποια ενέργεια και αυτή με τη σειρά της καλεί την απαραίτητη μέθοδο.