Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Αποτίμηση Ερωτήσεων Μερικού Μονοπατιού για Δεδομένα XML

Διπλωματικη Εργασια

του

ΓΕΩΡΓΙΟΥ Α. ΤΡΙΜΠΟΝΙΑ

Επιβλέπων: Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Εργαστηριο Συστηματων Βασεων Γνωσεων και Δεδομενων
Αθήνα, Μάιος 2008

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Συστημάτων Βάσεων Γνώσεων και Δεδομένων

Αποτίμηση Ερωτήσεων Μερικού Μονοπατιού για Δεδομένα XML

Διπλωματικη Εργασια

του

ΓΕΩΡΓΙΟΥ Α. ΤΡΙΜΠΟΝΙΑ

Επιβλέπων: Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 12η Μαίου 2008.

.....	.....	.....
Τιμολέων Σελλής	Ιωάννης Βασιλείου	Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επίκ. Καθηγητής Ε.Μ.Π.

Αθήνα, Μάιος 2008

.....

Γεωργιος Α. Τριμπονιας

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

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Συστημάτων Βάσεων Γνώσεων και Δεδομένων

Ευχαριστίες

Θα ήθελα να εκφράσω τις εγκάρδιες ευχαριστίες μου προς τους γονείς μου, που με τις θυσίες και την πολυδιάστατη στήριξη τους με βοήθησαν καθ΄ όλη τη διάρκεια των προπτυχιακών σπουδών μου. Αισθάνομαι επίσης ευγνώμων απέναντι στον Καθηγητή κ. Τίμο Σελλή όχι μόνο για την επίβλεψη της διπλωματικής αλλά και για την ανθρώπινη προσέγγισή του σε μία σειρά ακαδημαϊκών θεμάτων που με απασχόλησαν κατά καιρούς. Ευχαριστώ, τέλος, τους Θοδωρή Δαλαμάγκα και Στέφανο Σουλδάτο για την υπομονετική καθοδήγηση και ανεκτίμητη συνεισφορά τους στη συγγραφή της παρούσας διπλωματικής εργασίας.

Περίληψη

Ως η de facto τυποποίηση για την αναπαράσταση και ανταλλαγή πληροφορίας πάνω από το Διαδίκτυο, η γλώσσα XML έχει χρησιμοποιηθεί εκτενώς σε πολλές εφαρμογές. Δυνατότητες για ερωτήσεις παρέχονται μέσα από από ερωτήσεις δενδρικού προτύπου, που είναι οι κύριες συνιστώσες των τυποποιημένων XML γλωσσών ερωτήσεων, όπως η XPath και η XQuery. Μία δενδρική ερώτηση μπορεί να μοντελοποιηθεί ως ένα δέντρο με σημασμένους κόμβους, όπου οι κόμβοι σημαίνονται με διαφορετικές επικεφαλίδες, και οι ακμές αντιπροσωπεύουν σχέσεις είτε γονέα-παιδιού ή προγόνου-απογόνου. Επιτρέποντας τη σχέση προγόνου-απογόνου, οι δενδρικές ερωτήσεις παρέχουν κάποια ελευθερία στον καθορισμό της δενδρικής δομής. Ωστόσο, εξακολουθούν να απαιτούν τη σειρά διάταξης των κόμβων της ερώτησης. Στην περίπτωση που είτε δεν ξέρουμε το σχήμα ή τη δομή της πληροφορίας του δοσμένου XML συνόλου δεδομένων είτε πρέπει να ενσωματώσουμε διαφορετικές πηγές δεδομένων με διαφορετικές δενδρικές δομές, η δομή ενός μονοπατιού στο δενδρικό πρότυπο πρέπει να αφεθεί πιο χαλαρή.

Η διπλωματική εργασία στοχεύει από τη μία στην αντιμετώπιση του ζητήματος μίας νέας γλώσσας ερωτήσεων που επιτρέπει μερικώς καθορισμένες XML ερωτήσεις μονοπατιού και από την άλλη στην εξερεύνηση αποδοτικών αλγόριθμων αποτίμησης για αυτήν την κλάση ερωτήσεων.

Λέξεις Κλειδιά

XML, ερωτήσεις μονοπατιού μερικής δομής, γλώσσα ερωτήσεων, αλγόριθμοι αποτίμησης για ερωτήσεις δενδρικού προτύπου και προτύπου μονοπατιού, PathStack, PartialMJ, PartialPathStack

Abstract

As a de facto standard for information representation and exchange over the internet, XML has been used extensively in many applications. Query capabilities are provided through twig queries, which are the core components for standard XML query languages, e.g. XPath and XQuery. A twig query can be modeled as a node labeled tree, where nodes are labeled with different tags, and edges represent either the Parent-Child or Ancestor-Descendant relationships. By allowing ancestor-descendant relationship, twig queries provide some freedom in the specification of a tree structure; however, they still require the precedence order between query nodes. In case we either don't know the schema or structure information of the given XML data set or we need to integrate multiple data sources with different tree structures, the structure of a path in a tree pattern has to be relaxed.

The current diploma thesis aims on the one hand at addressing the issue of a new query language that allows for partially structured path pattern XML queries and on the other hand at exploring efficient evaluation algorithms for this class of queries.

Keywords

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

Περιεχόμενα

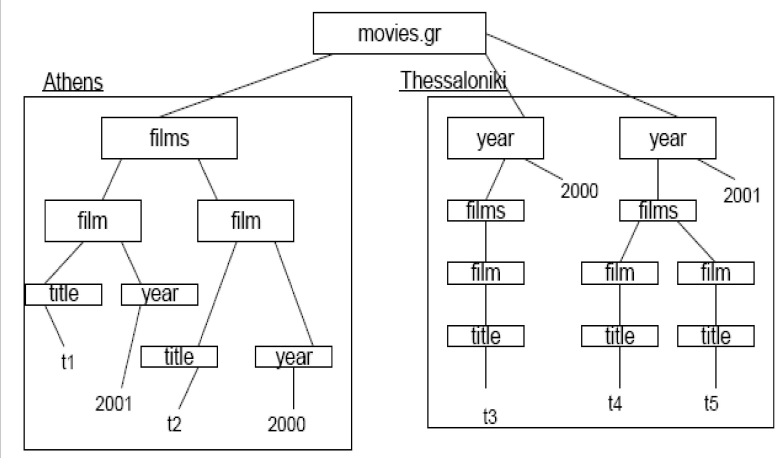
Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

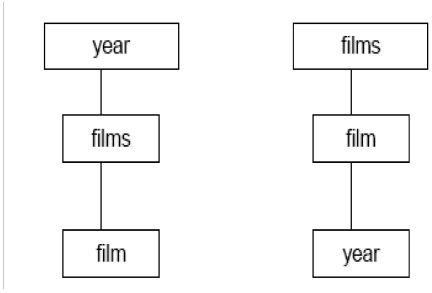
Εισαγωγή

Η γλώσσα XML (*Extensible Markup Language*) ([26]) έχει χρησιμοποιηθεί εκτενώς ως η de facto τυποποίηση/γλώσσα για την αναπαράσταση πληροφορίας και την ανταλλαγή δεδομένων πάνω από το Διαδίκτυο. Ένα έγγραφο XML περιέχει ιεραρχικά εμφωλευμένα στοιχεία και μπορεί να μοντελοποιηθεί κατά φυσικό τρόπο ως μία δεντρική δομή, όπου οι κόμβοι αντιπροσωπεύουν τα στοιχεία, ενώ οι ακμές τις δομικές σχέσεις ή σχέσεις συγγένειας μεταξύ των κόμβων του XML δέντρου. Τεράστιοι όγκοι δεδομένων οργανώνονται ή εξάγονται σε αυτή τη δενδρική μορφή και η επιθυμητή πληροφορία μπορεί να αποκτηθεί μέσω δενδρικών ερωτήσεων, που αποτελούν τον πυρήνα των γλωσσών έκφρασης XML ερωτήσεων, όπως η XPath ([27]) και η XQuery ([28]). Μία τέτοια ερώτηση μπορεί να μοντελοποιηθεί ως ένα δέντρο με σημασμένους κόμβους, όπου οι κόμβοι σημαίνονται με το αντίστοιχο στοιχείο προς αναζήτηση του XML εγγράφου, ενώ οι ακμές αναπαριστούν είτε δομικές σχέσεις γονέα-παιδιού ή δομικές σχέσεις προγόνου-απογόνου. Στη συνέχεια, η XML ερώτηση αποτιμάται με βάση κάποιον αλγόριθμο αποτίμησης XML ερωτήσεων, από τους πολλούς που έχουν προταθεί στη βιβλιογραφία, και επιστρέφεται το σύνολο των αποτελεσμάτων. Για παράδειγμα, οι δύο ερωτήσεις μονοπατιού του Σχήματος 1.2 μπορούν να χρησιμοποιηθούν για την εξαγωγή χρήσιμης πληροφορίας από ένα XML έγγραφο, όπου το σύμβολο | δηλώνει σχέση γονέα-παιδιού.



Σχήμα 1.1: Δύο XML έγγραφα από διαφορετικές πηγές με διαφορετική δομή.

Επιτρέποντας δομικές σχέσεις προγόνου-απογόνου και πατέρα-παιδιού, οι XML δενδρικές ερωτήσεις παρέχουν κάποια ελευθερία στον καθορισμό μίας δενδρικής δομής. Ένα περιοριστικό, ωστόσο, χαρακτηριστικό αυτών των ερωτήσεων είναι ότι απαιτούν μία ολική διάταξη μεταξύ των κόμβων κάθε μονοπατιού της δενδρικής ερώτησης. Για παράδειγμα, στην πρώτη ερώτηση του Σχήματος 1.2, ο κόμβος με τη σήμανση films πρέπει να είναι γονέας του κόμβου με τη σήμανση film, και ο δεύτερος κόμβος πρέπει με τη σειρά του να είναι γονέας του κόμβου με τη σήμανση year. Αυτός ο περιορισμός στον καθορισμό μίας δομής μονοπατιού μπορεί ωστόσο να δημιουργήσει σημαντικά προβλήματα στην πράξη.



Σχήμα 1.2: Δύο Ερωτήσεις Μονοπατιού.

Στην περίπτωση που δε γνωρίζουμε το σχήμα ή τη δομή της πληροφορίας του δοσμένου XML εγγράφου, τότε δεν έχει νόημα να γράψουμε μία ερώτηση καθορίζοντας την ολική διάταξη των κόμβων σε κάθε μονοπάτι της δενδρικής ερώτησης, αφού αυτή η ολική διάταξη μας είναι άγνωστη. Ακόμα όμως κι όταν το σχήμα ή η δομή της XML πληροφορίας είναι διαθέσιμα, αν θελήσουμε να πάρουμε όλη τη χρήσιμη πληροφορία από διάφορες πηγές με διαφορετική όμως δομή, τότε δεν μπορούμε να χρησιμοποιήσουμε μία μόνο XML ερώτηση με ολική διάταξη, αλλά πρέπει να γράψουμε μία τέτοια ερώτηση για κάθε τοποθεσία της κατανεμημένης XML βάσης δεδομένων. Για παράδειγμα, η πρώτη ερώτηση του Σχήματος 1.2 μπορεί να χρησιμοποιηθεί για να εξαγάγει χρήσιμη πληροφορία για την τοποθεσία Athens της κατανεμημένης XML βάσης δεδομένων του Σχήματος 1.1, ενώ η δεύτερη ερώτηση για την τοποθεσία Thessaloniki. Προφανώς, όταν ο αριθμός των διαφορετικών πηγών γίνεται πολύ μεγάλος, τότε είναι μη πρακτικό να υποβάλλουμε διαφορετικές ερωτήσεις σε κάθε πηγή δεδομένων, πόσο μάλλον όταν υπάρχουν και πηγές που το σχήμα ή η δομή της πληροφορίας τους είναι μη διαθέσιμα. Οι παραδοσιακοί κανόνες αντιστοίχισης μεταξύ ενός καθολικού σχήματος και του τοπικού σχήματος είναι πολύπλοκοι και μπορεί να οδηγήσουν σε λάθη, ενώ είναι μεγάλο βάρος να διατηρούνται οι κανόνες αντιστοίχισης μεταξύ του ολικού σχήματος και κάθε πηγής δεδομένων. Κοινός παρονομαστής και στις δύο παραπάνω περιπτώσεις είναι η επιθυμία μας για χαλάρωση του περιορισμού της ολικής διάταξης μεταξύ των κόμβων κάθε μονοπατιού της δενδρικής ερώτησης. Συγκεκριμένα, επιθυμούμε να γράφουμε XML ερωτήσεις όπου οι σχέσεις των κόμβων στα διάφορα μονοπάτια είναι μερικώς καθορισμένες και δεν υπάρχει ολική διάταξη, όπως προηγουμένως. Προς αυτήν την κατεύθυνση χρειαζόμαστε μία γλώσσα XML ερωτήσεων μονοπατιού με μερικώς καθορισμένη διάταξη, δηλαδή μία γλώσσα όπου τα υπό ερώτηση πρότυπα μονοπατιού δεν εισάγουν μία ολική σχέση συγγένειας μεταξύ των διαφόρων

κόμβων της ερώτησης, αλλά αντίθετα υπάρχει η ελευθερία να αφήνουμε ακαθόριστη τη δομική σχέση μεταξύ κόμβων της XML ερώτησης.

Βέβαια, η τυποποιημένη γλώσσα ερωτήσεων XML, XQuery, καθώς και η γλώσσα XPath που χρησιμοποιεί, επιτρέπουν πρότυπα που δε σχηματίζουν ένα πλήρες μονοπάτι ή δέντρο. Για παράδειγμα, ας θεωρήσουμε την επόμενη XPath έκφραση που περιλαμβάνει ανάστροφους άξονες και αστερίσκους: $//films/film[descendant-or-self::*[ancestor-or-self::year]]$. Αυτή η ερώτηση ρωτάει για τους film κόμβους σε μονοπάτια που επίσης περιέχουν year κόμβους, όπου ο film κόμβος είναι πάντα παιδί του films κόμβου, αλλά καμία καθορισμένη σχέση δεν απαιτείται μεταξύ του κόμβου year και των υπόλοιπων δύο κόμβων. Υπό αυτή την έννοια, πρόκειται για μία ερώτηση μονοπατιού μερικής δομής όπου η δομή του μονοπατιού είναι μόνο μερικώς κι όχι ολικώς καθορισμένη. Ο Oltenau και άλλοι ([19], [20]) έχουν δείξει οι XPath ερωτήσεις με ανάστροφους άξονες όπως η παραπάνω μπορούν ισοδύναμα να γραφούν σαν ένα σύνολο από δενρικές ερωτήσεις ολικής διάταξης. Ωστόσο, δείχνουν επίσης ότι αυτός ο μετασχηματισμός μπορεί να οδηγήσει σε εκθετική έκρηξη ως προς τον αριθμό των δενδρικών ερωτήσεων. Επίσης, ο Gottlob και άλλοι ([21]) έχουν αποδείξει ότι η πολυπλοκότητα της XPath είναι P-hard.

Είναι λοιπόν προφανές ότι 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. Δεδομένου μάλιστα ότι οι υπάρχουσες μέθοδοι αποτίμησης XML ερωτήσεων δε λαμβάνουν μέριμνα για την περίπτωση των ερωτήσεων μερικής διάταξης είναι σημαντικό να αναπτυχθούν κατάλληλες τεχνικές και αλγόριθμοι που να αποτιμούν αποδοτικά τέτοιες ερωτήσεις μονοπατιού μερικής δομής.

1.1 Αντικείμενο της Διπλωματικής Εργασίας

Η παρούσα διπλωματική εργασία ασχολείται με τις XML ερωτήσεις μονοπατιού μερικής δομής, δηλαδή με ερωτήσεις μονοπατιού όπου οι σχέσεις μεταξύ των διαφόρων κόμβων της ερώτησης μπορούν να είναι μη καθορισμένες. Βασίζεται κατά πολύ μεγάλο μέρος στις εργασίες που έχουν προηγηθεί στο Εργαστήριο Συστημάτων Βάσεων Γνώσεων και Δεδομένων και ιδιαίτερα στη δουλειά του Θοδωρή Δαλαμάγκα και του Στέφανου Σουλδάτου.

Συγκεκριμένα, στα πλαίσια αυτής της διπλωματικής εργασίας ορίζεται μία γλώσσα XML ερωτήσεων μονοπατιού μερικής δομής. Η γλώσσα αυτή κάνει χρήση ενός μοντέλου γράφου για τις ερωτήσεις και εισάγει ορισμένες πολύ σημαντικές έννοιες, όπως αυτές της πλήρους μορφής, της ικανοποιησιμότητας, του πλεονασμού κόμβων και της κανονικής μορφής. Χρησιμοποιεί ένα σύνολο από αληθείς και πλήρεις κανόνες παραγωγής δομικών σχέσεων μέσω των οποίων είναι δυνατό να μετασχηματίζουμε τη δοσμένη ερώτηση σε ισοδύναμες μορφές. Είναι μία αρκετά γενική γλώσσα ώστε να μπορεί να περιλαμβάνει από τη μία πλευρά ερωτήσεις μονοπατιού με ολική διάταξη και από την άλλη ερωτήσεις χωρίς δομικές σχέσεις (κόμβοι που βρίσκονται στο ίδιο μονοπάτι χωρίς όμως να προσδιορίζονται οι μεταξύ τους δομικές σχέσεις). Αυτή η γλώσσα μπορεί να εκφράσει διαφορετικούς τύπους XPath εκφράσεων σαν αυτή που

αναφέρθηκε πιο πάνω και έχει πλούσια εκφραστική ισχύ διατηρώντας όμως την απλότητα που πρέπει να χαρακτηρίζει κάθε γλώσσα ερωτήσεων.

Επίσης, στην εργασία αυτή αντιμετωπίζουμε το πρόβλημα της αποδοτικής αποτίμησης ερωτήσεων μονοπατιού μερικής δομής. Μία τέτοια ερώτηση μπορεί ισοδύναμα να εκφραστεί ως ένα σύνολο από ερωτήσεις μονοπατιού ολικής δομής/διάταξης. Υπάρχουν αρκετοί αλγόριθμοι στη βιβλιογραφία για την αποτίμηση ερωτήσεων μονοπατιού. Για παράδειγμα, ο Bruno και άλλοι [2] παρέχουν έναν αλγόριθμο που είναι ασυμπτωτικά βέλτιστος για ερωτήσεις μονοπατιού που δεν περιέχουν επαναλήψεις του ίδιου στοιχείου/σήμανσης στους κόμβους της ερώτησης. Ωστόσο, σύμφωνα και με όσα αναφέρθησαν πιο πάνω, αυτό το ισοδύναμο σύνολο μπορεί να περιέχει έναν αριθμό από ερωτήσεις μονοπατιού, που μπορεί να είναι εκθετικός ως προς τον αριθμό των κόμβων της ερώτησης, καθιστώντας το κόστος αυτής της τεχνικής απαγορευτικό. Επομένως, αντί να προσανατολιστούμε σε τεχνικές που δημιουργούν το ισοδύναμο σύνολο ερωτήσεων μονοπατιού ολικής δομής, εστιάζουμε σε μεθόδους που μπορούν άμεσα να επεξεργαστούν και να ταιριάξουν τη μερική ερώτηση στο XML δέντρο δεδομένων.

Προκειμένου να βγάλουμε χρήσιμα συμπεράσματα για τους διάφορους προτεινόμενους στα πλαίσια της διπλωματικής αλγόριθμους, προβαίνουμε και στην υλοποίησή τους στην αντικειμενοστρεφή γλώσσα προγραμματισμού C++, δίνοντας ιδιαίτερο βάρος στις τεχνικές λεπτομέρειες, ενώ αξιολογούμε πειραματικά τους αλγόριθμους αυτούς με τη βοήθεια διάφορων ερωτήσεων σε benchmarks XML αρχεία. Η πειραματική αξιολόγηση μας επιτρέπει να επαληθεύσουμε τα θεωρητικώς αναμενόμενα αποτελέσματα με πρακτικά αποτελέσματα, ενισχύοντας με αυτόν τον τρόπο την ισχύ των συμπερασμάτων μας.

Βέβαια, αν και το κυρίως αντικείμενο της παρούσας εργασίας σχετίζεται με τις XML ερωτήσεις μονοπατιού μερικής δομής, εξετάζονται μία σειρά από παρεμφερή ή βοηθητικά θέματα, που αποτελούν στην ουσία το υπόβαθρο για την ανάπτυξη των κύριων θεμάτων. Μεταξύ αυτών είναι εισαγωγικά στοιχεία για την XML γλώσσα καθώς και για τις γλώσσες ερωτήσεων XQuery και XPath, καθώς και μία κάπως εκτεταμένη βιβλιογραφική αναφορά πάνω σε υπάρχουσες τεχνικές αποτίμησης XML ερωτήσεων μονοματιού ή δενδρικών ερωτήσεων ολικής δομής.

1.2 Οργάνωση του Τόμου

Η διπλωματική εργασία είναι οργανωμένη σε επτά κεφάλαια.

Το παρόν κεφάλαιο αποτελεί την εισαγωγή.

Το Κεφάλαιο 2 είναι ένα σύντομο εγχειρίδιο της XML καθώς και των γλωσσών ερωτήσεων XQuery και XPath. Εξηγούνται οι βασικές αρχές των γλωσσών και δίδονται παραδείγματα για την ευκολότερη κατανόησή τους.

Το Κεφάλαιο 3 παρουσιάζει το θεωρητικό υπόβαθρο και τις κύριες ιδέες για τους αλγόριθμους αποτίμησης που προτείνονται στη διπλωματική εργασία, σε μία εκτεταμένη βιβλιογραφική αναφορά. Αρχικά, γίνεται αναφορά σε ορισμένες προκαταρκτικές έννοιες, όπως στο μοντέλο δεδομένων της γλώσσας XML, σε δύο σχήματα κωδικοποίησης θέσης σε XML έγγραφα, συγκεκριμένα στην κωδικοποίηση θέσης με χρήση ανεστραμμένων ευρετηρίων και στο σχήμα

εκτεταμένης κωδικοποίησης Dewey, και σε διάφορους τύπους ευρετηρίων για γρηγορότερη επεξεργασία ερωτήσεων. Έπειτα παρουσιάζονται μέθοδοι για αποτίμηση ερωτήσεων με δυαδικές δομικές σχέσεις: η μία προσέγγιση βασίζεται σε συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων, ενώ η άλλη σε αποδοτικές δυαδικές δομικές σχέσεις. Στη συνέχεια, αναλύονται εκτενώς οι αλγόριθμοι PathStack και TwigStack ([2]) για ερωτήσεις μονοπατιού και δενδρικής μορφής αντίστοιχα. Ειδικά ο αλγόριθμος PathStack αποτελεί το υπόβαθρο γύρω από τον οποίο αναπτύσσονται οι αλγόριθμοι του Κεφαλαίου 5 και δε θα ήταν υπερβολή να τον χαρακτηρίσουμε ως τον πυρήνα της παρούσας διπλωματικής εργασίας. Κατόπιν, αναφέρονται διάφορες τεχνικές για βελτιώσεις των αλγορίθμων PathStack και TwigStack, ενώ το Κεφάλαιο κλείνει με μία αναφορά σε τεχνικές που ακολουθούν το σχήμα εκτεταμένης κωδικοποίησης Dewey για την αναπαράσταση θέσης.

Το Κεφάλαιο 4 αποτελείται από δύο μέρη. Στο πρώτο μέρος, αναπτύσσεται μία γλώσσα XML ερωτήσεων μονοπατιού μερικής δομής, η οποία υπερβαίνει τις δυσκολίες που αναφέρθηκαν διεξοδικά στην αρχή του παρόντος Κεφαλαίου. Ορίζεται ένα σύνολο από αληθείς και πλήρεις κανόνες παραγωγής που χαρακτηρίζουν πλήρως την παραγωγή δομικών σχέσεων και που η επανειλημμένη εφαρμογή τους οδηγεί στην ισοδύναμη πλήρη μορφή της αρχικής ερώτησης. Εισάγεται η έννοια της μη ικανοποιησιμότητας, αφού σε αντίθεση με τις ερωτήσεις μονοπατιού ολικής διάταξης, οι ερωτήσεις μερικής δομής μπορούν να είναι μη ικανοποιήσιμες. Δίνονται μάλιστα και οι ικανές και αναγκαίες συνθήκες για μη ικανοποιησιμότητα. Επίσης, εξετάζεται η έννοια των πλεονάζοντων κόμβων, δηλαδή κόμβων που μπορούν να απομακρυνθούν από την ερώτηση χωρίς να επηρεάζεται η σημασία της ερώτησης, παρέχοντας ταυτόχρονα κάποια πρότυπα που απλοποιούν την εύρεση τέτοιων κόμβων. Τέλος, δείχνουμε πώς μπορούμε να μετασχηματίσουμε την αρχική ερώτηση μονοπατιού στην ισοδύναμη κανονική της μορφή, που είναι ένας κατευθυνόμενος ακυκλικός γράφος. Στο δεύτερο μέρος, αναπτύσσονται αλγόριθμοι αποτίμησης ερωτήσεων μονοπατιού μερικής δομής. Αρχικά αναπτύσσεται ο βασισμένος σε στοίβες αλγόριθμος PartialMJ. Ο PartialMJ εξάγει ένα επικαλύπτον δένδρο από την κανονική μορφή της ερώτησης. Χρησιμοποιεί μία επέκταση του PathStack για να υπολογίσει τα αποτελέσματα όλων των μονοπατιών από τη ρίζα προς τα φύλλα του επικαλύπτοντος δένδρου. Αυτά τα αποτελέσματα παράγονται ταξινομημένα ως προς τη ρίζα−προς−φύλλο διάταξη του XML δένδρου και συγχωνεύονται για να υπολογίσουμε την τελική απάντηση στην αρχική ερώτηση. Ο PartialMJ μπορεί όμως να δημιουργήσει πολλά ενδιάμεσα αποτελέσματα για τα διάφορα μονοπάτια από τη ρίζα προς τα φύλλα του μη επικαλυπτόμενου δένδρου που δε συνεισφέρουν στην τελική λύση. Για να ξεπεράσουμε αυτό το πρόβλημα, αναπτύσσουμε έναν καινούριο ολιστικό βασισμένο σε στοίβες αλγόριθμο, τον PartialPathStack, ο οποίος εκμεταλλεύεται μία τοπολογική διάταξη των κόμβων της ερώτησης στην κανονικής της μορφή, και προσπαθεί να ταιριάξει την κανονική μορφή συνολικά πάνω στο XML δένδρο. Αναλύουμε την πολυπλοκότητα του PartialPathStack και δείχνουμε ότι είναι ανεξάρτητος από ενδιάμεσα αποτελέσματα.

Στο Κεφάλαιο 5 αναφέρονται οι τεχνικές λεπτομέρειες των υλοποιημένων αλγορίθμων PathStack, PartialMJ, PartialPathStack.

Στο Κεφάλαιο 6 προβαίνουμε σε μία εκτενή πειραματική αξιολόγηση. Η πειραματική αξι-

ολόγηση δείχνει εκτός των άλλων την εξάρτηση του PartialMJ από τα ενδιάμεσα αποτελέσματα και την ανωτερότητα του PartialPathStack.

Στο Κεφάλαιο 7 επιχειρείται μία σύνοψη της διπλωματικής εργασίας και γίνεται αναφορά σε ενδιαφέρουσες μελλοντικές επεκτάσεις.

Στο τέλος του τόμου παρουσιάζεται η σχετική βιβλιογραφία.

Κεφάλαιο 2

Τεχνολογίες XML και Γλώσσα Ερωτήσεων XQuery

Σε αυτό το κεφάλαιο θα παρουσιάσουμε τη γλώσσα XML, τη μορφή των XML δεδομένων, γλώσσες ορισμού σχήματος, καθώς και τις γλώσσες XPath και XQuery. Υπάρχουν πολλά ακόμα σημαντικά θέματα στη γλώσσα XML, όπως η γλώσσα μετασχηματισμού XSL, τα οποία δε θίγουμε στα πλαίσια της παρούσας διπλωματικής εργασίας. Ο ενδιαφερόμενος αναγνώστης παραπέμπεται στο [26], το οποίο καλύπτει όλα τα επίκαιρα θέματα έρευνας και ανάπτυξης της γλώσσας XML, ενώ για μία συνοπτική ανασκόπηση της XML παραπέμπουμε στο [25] (Κεφάλαιο 10).

2.1 Τεχνολογίες XML

2.1.1 Υπόβαθρο

Η γλώσσα XML (*Extensible Markup Language*) είναι μία γλώσσα σήμανσης. Ο όρος σήμανση (*markup*) αναφέρεται σε οτιδήποτε υπάρχει σε ένα έγγραφο που δεν πρόκειται να τυπωθεί. Για παράδειγμα, μπορεί στο κείμενο που γράφουμε για την εφημερίδα του Πολυτεχνείου να υπάρχουν σημειώσεις για το πώς θα γίνει η στοιχειοθέτηση ή η εκτύπωση. Αυτές οι σημειώσεις είναι σημαντικό να μπορούν να ξεχωρίσουν από τα περιεχόμενα. Στην ηλεκτρονική επεξεργασία εγγράφων, μία γλώσσα σήμανσης είναι μία τυπική περιγραφή για το ποιο μέρος του εγγράφου είναι περιεχόμενο, ποιο είναι σήμανση και τι σημαίνει η σήμανση.

Οι γλώσσες σήμανσης αναπτύχθηκαν από την ανάγκη για τον ορισμό εντολών για τον τρόπο που θα τυπώνονται μέρη ενός εγγράφου, για να καθορίζεται η *λειτουργία* των περιεχομένων. Η λειτουργική σήμανση επιτρέπει στο έγγραφο να μορφοποιείται διαφορετικά, σε διαφορετικές περιστάσεις. Βοηθά επίσης διαφορετικά μέρη ενός εγγράφου να μορφοποιηθούν με ομοιόμορφο τρόπο. Ταυτόχρονα, η λειτουργική σήμανση επιτρέπει να αυτοματοποιηθεί η εξαγωγή βασικών τμημάτων των εγγράφων.

Για την οικογένεια των γλωσσών σήμανσης που περιλαμβάνουν τις HTML (*Hyper-Text Markup Language*), SGML (*Standard Generalized Markup Language*) και XML, η σήμανση

παίρνει τη μορφή ετικετών (*tag*) που περικλείονται σε γωνιώδεις αγκύλες, <>. Οι ετικέτες χρησιμοποιούνται σε ζευγάρια, με την <tag> και την </tag> να χωρίζουν την αρχή και το τέλος του τμήματος του εγγράφου στο οποίο αναφέρεται η ετικέτα.

Αντίθετα όμως με την HTML, η XML δεν περιγράφει το σύνολο των ετικετών που επιτρέπονται και το σύνολο μπορεί να εξειδικεύεται ανάλογα με την περίσταση. Αυτή η λειτουργία αποτελεί το κλειδί για το βασικό ρόλο της XML στην αναπαράσταση και ανταλλαγή των δεδομένων, όπου η HTML χρησιμοποιείται βασικά για μορφοποίηση εγγράφων.

Για παράδειγμα, ας θεωρήσουμε το XML έγγραφο του Σχήματος 2.1. Το έγγραφο αυτό περιέχει πληροφορίες για τους κινηματογράφους σε μία πόλη, καθώς και για τις ταινίες που προβάλλονται σε καθέναν από αυτούς. Επιπλέον παρέχονται πληροφορίες για διάφορα στοιχεία του κινηματογράφου ή των ταινιών που μπορεί να μας ενδιαφέρουν, όπως διεύθυνση ή βαθμολογία αντίστοιχα, ενώ υπάρχει ειδική μέριμνα για τις κρατήσεις ανά ταινία.

Σε σύγκριση με την αποθήκευση σε μια βάση δεδομένων, η αναπαράσταση XML μπορεί να είναι αναποτελεσματική, αφού τα ονόματα ετικετών επαναλαμβάνονται σε όλα τα έγγραφα, δημιουργώντας αυξημένες απαιτήσεις αποθηκευτικού χώρου. Ωστόσο, ανεξάρτητα από αυτό το μειονέκτημα, η αναπαράσταση XML έχει πολλά σημαντικά πλεονεκτήματα όταν χρησιμοποιείται για ανταλλαγή δεδομένων:

(α) Η παρουσία ετικετών κάνει το μήνυμα να *τεκμηριώνεται από μόνο του*, δηλαδή δε χρειάζεται να ανατρέξει κανείς σε ένα σχήμα για να καταλάβει τη σημασία του κειμένου.

(β) Η μορφή του εγγράφου δεν είναι αυστηρή. Θα μπορούσε κάποιος αποστολέας στο παράδειγμά μας να προσθέσει επιπλέον πληροφορία, όπως την ετικέτα <distribution-company> που σημειώνει την εταιρία διανομής της ταινίας. Ο παραλήπτης των δεδομένων μπορεί απλά να αγνοήσει την ετικέτα, αν δεν μπορεί να κατανοήσει το περιεχόμενό της. Η δυνατότητα να αναγνωρίζουμε και να αγνοούμε μη αναμενόμενες ετικέτες επιτρέπει να αναπτύσσεται η μορφή των δεδομένων με το χρόνο, χωρίς να κάνει άκυρες τις υπάρχουσες εφαρμογές.

(γ) Αφού η μορφή XML είναι ευρέως αποδεκτή, είναι διαθέσιμη μια μεγάλη ποικιλία από εργαλεία για να βοηθήσουν στην επεξεργασία, όπως το λογισμικό browser και τα εργαλεία βάσεων δεδομένων.

Για όλους τους παραπάνω λόγους η XML γίνεται η βασική μορφή για ανταλλαγή δεδομένων.

2.1.2 Δομή των XML Δεδομένων

Η βασική δομή σε ένα XML έγγραφο είναι το στοιχείο. Ένα στοιχείο είναι απλώς ένα ζευγάρι από ετικέτες αρχής και τέλους και όλο το κείμενο μεταξύ τους.

Τα XML έγγραφα πρέπει να έχουν ένα στοιχείο ρίζας, που περιέχει όλα τα άλλα στοιχεία του εγγράφου. Στο παράδειγμα του Σχήματος 2.1, το στοιχείο <cinemas> σχηματίζει το στοιχείο ρίζας. Επιπλέον, τα στοιχεία σε ένα XML έγγραφο πρέπει να έχουν μπει σωστά σε ένθεση, με ένα τρόπο που θυμίζει τη σωστή ένθεση των παρενθέσεων σε ένα κομμάτι κώδικα.

Η σωστή ένθεση μπορεί να οριστεί και πιο τυπικά. Συγκεκριμένα, λέμε ότι το κείμενο εμφανίζεται στο περιβάλλον ενός στοιχείου αν εμφανίζεται μεταξύ των ετικετών αρχής και

τέλους αυτού του στοιχείου. Οι ετικέτες είναι σωστά ένθετες αν κάθε ετικέτα αρχής έχει μια μοναδική, αντίστοιχη ετικέτα τέλους, που βρίσκεται στο περιβάλλον του ίδιου γονικού στοιχείου.

Το κείμενο μέσα στην εμβέλεια ενός στοιχείου μπορεί να αναμιχθεί με τα υποστοιχεία του στοιχείου, όπως φαίνεται και στο στοιχείο cinema με name="Apollon" στο Σχήμα 2.1. Αυτή η ελευθερία έχει περισσότερη έννοια σε περιβάλλον επεξεργασίας εγγράφων παρά σε περιβάλλον επεξεργασίας δεδομένων και δεν είναι ιδιαίτερα χρήσιμη για αναπαράσταση πιο δομημένων δεδομένων, όπως περιεχομένων βάσεων δεδομένων.

Φυσικά, μία πολύ σπουδαία ιδιότητα της XML είναι η δυνατότητα να ορίζουμε στοιχεία μέσα σε άλλα στοιχεία, όπως και στο XML έγγραφο που δώσαμε στο Σχήμα 2.1. Χρησιμοποιώντας αυτή τη δυνατότητα μπορούμε να ενθέσουμε όλες τις ταινίες που προβάλλονται σε ένα κινηματογράφο μέσα στο στοιχείο του κινηματογράφου, και όλες τις κρατήσεις που αντιστοιχούν σε μία ταινία που προβάλλεται σε ένα δεδομένο κινηματογράφο μέσα στο στοιχείο της ταινίας. Σε διαφορετική περίπτωση, θα έπρεπε να χρησιμοποιήσουμε κάποιο αναγνωριστικό για τις ταινίες και τους κινηματογράφους και να χρησιμοποιούσαμε κάποιο επιπλέον στοιχείο που να δείχνει ποια αναγνωριστικά ταινιών συνδυάζονται με ποια αναγνωριστικά κινηματογράφων, όπως το movies_in_cinema του Σχήματος 2.2, το οποίο χάριν έλλειψης χώρου δεν παρέχει πληροφορίες για τον κινηματογράφο "Apollon". Ομοίως και για τα στοιχεία ταινίες−κρατήσεις.

Οι ένθετες αναπαραστάσεις χρησιμοποιούνται ευρέως σε εφαρμογές XML ανταλλαγής δεδομένων, για να αποφεύγονται οι σύνδεσμοι (join). Για παράδειγμα, μία εφαρμογή αποστολής θα πρέπει να αποθηκεύει την πλήρη διεύθυνση του αποστολέα και του παραλήπτη επαναλαμβανόμενα σε ένα έγγραφο αποστολής που σχετίζεται με κάθε αποστολή, ενώ μία κανονικοποιημένη αναπαράσταση μπορεί να απαιτεί ένα σύνδεσμο στις εγγραφές αποστολής με μια σχέση $company - address$, από την οποία να παίρνει πληροφορίες διευθύνσεων.

Εκτός από στοιχεία, η γλώσσα XML καθορίζει και την έννοια της *ιδιότητας*. Για παράδειγμα, κάθε στοιχείο cinema έχει ως ιδιότητες τα name και region. Οι ιδιότητες ενός στοιχείου εμφανίζονται σε ζεύγη ονόματος - τιμής πριν από το '>' μιας ετικέτας αρχής. Οι ιδιότητες είναι συμβολοσειρές και δεν περιέχουν σήμανση. Επιπλέον, οι ιδιότητες μπορεί να επαναλαμβάνονται μόνο μία φορά σε μία ετικέτα, αντίθετα από τα υποστοιχεία, που μπορεί να επαναλαμβάνονται (π.χ. το υποστοιχείο movie μέσα στο στοιχείο movie). Είναι προφανές ότι αντί της ιδιότητας μπορούμε να χρησιμοποιήσουμε ένα υποστοιχείο με το ίδιο όνομα με την ιδιότητα και κείμενο την τιμή της ιδιότητας.

Σημειώνουμε ότι στο περιβάλλον κατασκευής ενός εγγράφου, η διαφορά μεταξύ μίας ιδιότητας κι ενός υποστοιχείου είναι σημαντική, όπου μία ιδιότητα είναι έμμεσο κείμενο που δεν εμφανίζεται στο έγγραφο που τυπώνεται ή εμφανίζεται. Πάντως, σε εφαρμογές βάσεων δεδομένων και ανταλλαγής δεδομένων της XML, αυτή η διαφοροποίηση δεν έχει τόση σημασία και η επιλογή της αναπαράστασης των δεδομένων ως μίας ιδιότητας ή υποστοιχείου είναι αυθαίρετη.

Μία τελευταία συντακτική σημείωση είναι ότι ένα στοιχείο της μορφής <element></element>, που δεν περιέχει υποστοιχεία ή κείμενο, μπορεί να συντομευτεί ως <element/>, όπως, για πα-

ράδειγμα, το <reservations/> στο δεύτερο στοιχείο movie του στοιχείου cinema με ιδιότητα name="Alexandra". Τα στοιχεία συντόμευσης μπορούν να περιέχουν ιδιότητες.

Αφού τα XML έγγραφα έχουν σχεδιαστεί να ανταλλάσσονται μεταξύ εφαρμογών, έχει παρουσιαστεί ένας μηχανισμός *χώρου ονομάτων* (*namespace*) που επιτρέπει σε οργανισμούς να καθορίζουν καθολικά μοναδικά ονόματα που θα χρησιμοποιούνται ως ετικέτες στοιχείων σε έγγραφα. Η ιδέα ενός χώρου ονομάτων είναι για να προτάσσει σε κάθε ετικέτα ή ιδιότητα ένα μοναδικό αναγνωριστικό πόρων (για παράδειγμα, μία Web διεύθυνση). Αν, λοιπόν, ένας μεγάλος οργανισμός με πολλά παραρτήματα και σχέσεις με πολλούς άλλους οργανισμούς ήθελε να διασφαλίσει ότι τα XML έγγραφα που δημιουργούσε δε θα είχαν διπλότυπες ετικέτες που να χρησιμοποιούνται από XML έγγραφα κάποιου συνεργάτη του, μπορεί να προτάξει κάποιο μοναδικό αναγνωριστικό με μία άνω-κάτω τελεία, σε κάθε όνομα ετικέτας. Ο οργανισμός μπορεί να χρησιμοποιήσει το Web URL ως μοναδικό αναγνωριστικό για όλα τα παραρτήματά του, αφού αυτό είναι εγγυημένα μοναδικό σε ολόκληρο το διαδίκτυο. Συνήθως, πάντως, χρησιμοποιείται κάποια συντόμευση για τα αναγνωριστικά, αφού μπορεί να έχουν μεγάλο και δύσχρηστο μέγεθος.

Αυτό μπορεί να επιτευχθεί εισάγοντας μία ιδιότητα xmlns:*short*=*value* στο στοιχείο ρίζας, η οποία βασικά δηλώνει ότι το *short* χρησιμοποιείται μέσα στο XML έγγραφο ως συντόμευση για το αναγνωριστικό *value* που δίδεται παραπάνω. Η συντόμευση μπορεί στη συνέχεια να χρησιμοποιηθεί και σε άλλες ετικέτες στοιχείων.

Ένα έγγραφο μπορεί να έχει περισσότερους από ένα χώρους ονομάτων, που δηλώνονται ως μέρος του στοιχείου ρίζας. Μετά μπορούν να συσχετιστούν διάφορα στοιχεία με διαφορετικούς χώρους ονομάτων. Ένας *προκαθορισμένος χώρος ονομάτων* μπορεί να οριστεί χρησιμοποιώντας την ιδιότητα xmlns αντί της xmlns:*short* στο στοιχείο ρίζα. Τα στοιχεία χωρίς άμεσο πρόθεμα χώρου ονομάτων θα ανήκουν μέσα στον προκαθορισμένο χώρο.

Τέλος, κάποιες φορές θέλουμε να αποθηκεύσουμε τιμές που περιέχουν ετικέτες, χωρίς να έχουμε τις ετικέτες που μεταφράζονται ως XML ετικέτες. Η γλώσσα XML επιτρέπει αυτό με χρήση της δομής:

<![CDATA[<element>...</element>]]>

Το κείμενο <element>, επειδή περικλείεται μέσα στο CDATA (character data), αντιμετωπίζεται ως κανονικό κείμενο κι όχι ως ετικέτα.

2.1.3 Σχήμα ενός Εγγράφου XML

Οι βάσεις δεδομένων έχουν σχήματα, που χρησιμοποιούνται για να περιορίζουν τις πληροφορίες που μπορούν να αποθηκευτούν στη βάση δεδομένων και να περιορίσουν τους τύπους των δεδομένων των αποθηκευμένων πληροφοριών. Σύμφωνα όμως με όσα προαναφέρθηκαν, τα XML έγγραφα μπορούν να δημιουργηθούν χωρίς κάποιο σχετικό σχήμα, δηλαδή ένα σύνολο από κανόνες που να περιορίζουν την ακολουθία στοιχείων με υποστοιχεία και ιδιότητες. Ένα στοιχείο μπορεί να ακολουθείται από οποιοδήποτε υποστοιχείο ή ιδιότητα. Ενώ μία τέτοια ελευθερία μπορεί να είναι αποδεκτή εξαιτίας της φύσης της εφαρμογής που περιγράφει από

μόνη της τη μορφή των δεδομένων, δεν είναι γενικά χρήσιμη όταν πρέπει να γίνει επεξεργασία των XML εγγραφών ατομικά ως μέρος μίας εφαρμογής ή ακόμα κι όταν μεγάλες ποσότητες σχετικών δεδομένων πρέπει να μορφοποιηθούν στην XML.

Την ανάγκη αυτή έρχεται να λύσει μερικώς ο έγγραφο-κεντρικός μηχανισμός σχήματος που είναι γνωστός ως (*Ορισμός Τύπου Εγγράφου Document Type Definition, DTD*), και ο οποίος περιλαμβάνεται και ως μέρος της τυποποίησης XML. Επειδή όμως ο μηχανισμός αυτός παρουσιάζει μία σειρά από προβλήματα, πρόσφατα έχουν προταθεί μία σειρά από γλώσσες περιγραφής του XML σχήματος που, αν και δεν έχουν γίνει ακόμη μέρος της τυποποίησης XML, είναι εντούτοις πολύ δημοφιλείς στους προγραμματιστές εφαρμογών.

Ορισμός Τύπου Εγγράφου

Ο ορισμός τύπου εγγράφου είναι ένα προαιρετικό μέρος ενός XML εγγράφου. Ο κύριος σκοπός του DTD είναι αντίστοιχος με το σκοπό του σχήματος στις βάσεις δεδομένων: να περιορίζει τον τύπο των πληροφοριών που παρουσιάζονται στο έγγραφο. Ωστόσο, το DTD δεν παρουσιάζει τους τύπους, σε σχέση με τους βασικούς τύπους, όπως ακέραιους ή συμβολοσειρές. Αντίθετα, περιορίζει μόνο την εμφάνιση υποστοιχείων και ιδιοτήτων μέσα σε ένα στοιχείο. Το DTD είναι βασικά μία λίστα από κανόνες για το ποιο μοτίβο υποστοιχείων θα εμφανίζεται μέσα σε ένα στοιχείο. Το Σχήμα 2.3 δείχνει μέρος ενός παραδείγματος DTD για πληροφορίες κινηματογράφων. Το έγγραφο XML στο Σχήμα 2.1 ανταποκρίνεται σε αυτό το DTD.

Κάθε δήλωση είναι με τη μορφή μίας κανονικής παράστασης για τα υποστοιχεία ενός στοιχείου. Έτσι, στο DTD του Σχήματος 2.3, ένα στοιχείο cinemas αποτελείται μηδέν ή περισσότερα στοιχεία cinema. Ο τελεστής | καθορίζει το διαζευκτικό τελεστή, ενώ ο τελεστής + καθορίζει το "ένα ή περισσότερα". Ο τελεστής ∗ χρησιμοποιείται για να καθορίζει "μηδέν ή περισσότερα", ενώ ο τελεστής ? χρησιμοποιείται για να καθορίσει ένα προαιρετικό στοιχείο (δηλαδή "μηδέν ή ένα").

Το στοιχείο cinema ορίζεται να περιέχει (μία φορά) το υποστοιχείο movies, το οποίο με τη σειρά του ορίζεται να περιέχει μηδέν ή περισσότερες φορές το υποστοιχείο movie. Επίσης, το movie περιέχει το υποστοιχείο director μία φορά, το υποστοιχείο actor μία ή περισσότερες φορές, καθώς και τα υποστοιχεία genre, rating, hour και reservations (με αυτή τη σειρά). Τέλος, το υποστοιχείο reservations ορίζεται να περιέχει μηδέν ή περισσότερες φορές το υποστοιχείο reservation, το οποίο με τη σειρά του πρέπει να περιέχει από μία φορά τα υποστοιχεία name και payment (με αυτή τη σειρά). Τα στοιχεία director, actor, genre, rating, hour name και payment ορίζονται να είναι τύπου #PCDATA. Η λέξη κλειδί PCDATA δείχνει δεδομένα κειμένου. Δύο άλλοι ειδικοί τύποι δηλώσεων είναι το empty, που λέει ότι το στοιχείο δεν έχει περιεχόμενα και το any, που λέει ότι δεν υπάρχει περιορισμός στα υποστοιχεία του υποστοιχείου. Δηλαδή, οποιαδήποτε στοιχεία, ακόμη κι αυτά που δεν αναφέρονται στο DTD, μπορούν να υπάρχουν ως υποστοιχεία του στοιχείου. Η απουσία μίας δήλωσης στοιχείου είναι ισοδύναμη με την άμεση δήλωση του τύπου any.

Όπως προκύπτει κι από το Σχήμα 2.3, και οι επιτρεπόμενες ιδιότητες για κάθε στοιχείο

δηλώνονται επίσης στο DTD. Αντίθετα με τα υποστοιχεία, δεν υπάρχει κάποια σειρά στις ιδιότητες. Οι ιδιότητες μπορούν να καθοριστούν να είναι τύπου CDATA, ID, IDREF ή IDREFS. Ο τύπος CDATA απλώς λέει ότι η ιδιότητα περιέχει χαρακτήρες δεδομένων, ενώ τα υπόλοιπα τρία εξηγούνται στη συνέχεια.

Οι ιδιότητες πρέπει να έχουν επίσης και μία προκαθορισμένη δήλωση. Η προκαθορισμένη δήλωση μπορεί να αποτελείται από μία προκαθορισμένη τιμή για την ιδιότητα ή το #REQUIRED, που σημαίνει ότι πρέπει να καθοριστεί αυτή η τιμή για την ιδιότητα κάθε στοιχείου, ή το #IMPLIED, που δηλώνει ότι δεν παρέχεται καμία προκαθορισμένη τιμή. Η προκαθορισμένη ιδιότητα μπορεί να είναι #FIXED, οπότε δεν επιτρέπεται καμία άλλη τιμή για την ιδιότητα πέρα από την προκαθορισμένη. Αν μία ιδιότητα δεν έχει προκαθορισμένη τιμή, τότε για κάθε στοιχείο που δεν καθορίζει μία τιμή για την ιδιότητα, η προκαθορισμένη ιδιότητα συμπληρώνεται αυτόματα όταν διαβάζεται το XML έγγραφο.

Για παράδειγμα, στο Σχήμα 2.3 δηλώνουμε ότι το στοιχείο cinema έχει τις ιδιότητες name και region, που περιέχουν χαρακτήρες δεδομένων, όπου η πρώτη πρέπει να συμπληρώνεται υποχρεωτικά, ενώ η δεύτερη όχι. Ομοίως και για τις ιδιότητες του στοιχείου movie.

Μία ιδιότητα τύπου ID παρέχει ένα μοναδικό αναγνωριστικό για το στοιχείο. Μια τιμή που υπάρχει σε μία ιδιότητα ID ενός στοιχείου δεν πρέπει να υπάρχει σε οποιοδήποτε άλλο στοιχείο του ίδιου εγγράφου. Το πολύ μία ιδιότητα ενός στοιχείου επιτρέπεται να είναι τύπου ID.

Μία ιδιότητα τύπου IRDEF είναι μία αναφορά σε ένα στοιχείο. Η ιδιότητα πρέπει να περιέχει μία τιμή που εμφανίζεται στην ιδιότητα ID κάποιου στοιχείου του εγγράφου. Ο τύπος IDREFS επιτρέπει μία λίστα από αναφορές, χωρισμένες με κενά. Οι ιδιότητες αυτές λειτουργούν ως μηχανισμός αναφοράς σε αντικειμενοστρεφείς βάσεις δεδομένων και σε σχεσιακές βάσεις δεδομένων, που επιτρέπουν την κατασκευή περίπλοκων τύπων. Επειδή όμως αυτές οι λειτουργίς ξεφεύγουν από τα πλαίσια της παρούσας διπλωματικής, δε θα εισέλθουμε σε περισσότερες λεπτομέρειες.

Οι ορισμοί τύπου εγγραφών είναι πολύ συνδεδεμένοι με τη μορφοποίηση της XML. Εξαιτίας αυτού, δεν είναι κατάλληλοι με πολλούς τρόπους να εξυπηρετήσουν ως δομή τύπων της XML, για εφαρμογές επεξεργασίας δεδομένων. Πάντως, ορίζεται ένας πολύ μεγάλος αριθμός μορφών ανταλλαγής δεδομένων σε σχέση με τα DTD, αφού ήταν μέρος της αρχικής τυποποίησης. Μερικοί από τους περιορισμούς των DTD ως μηχανισμοί σχήματος είναι οι παρακάτω:

(α) Τα περισσότερα μεμονωμένα στοιχεία κειμένου και οι ιδιότητες δεν μπορούν να έχουν τύπο. Για παράδειγμα το στοιχείο reservations δεν μπορεί να περιοριστεί να είναι ένας μη αρνητικός ακέραιος. Η έλλειψη τέτοιων περιορισμών είναι προβληματική για επεξεργασία δεδομένων και εφαρμογές ανταλλαγής δεδομένων, που μπορούν μετά να περιέχουν κώδικα για να επαληθεύσουν τους τύπους και ιδιότητες των στοιχείων.

(β) Είναι δύσκολο να χρησιμοποιήσουμε το μηχανισμό DTD για να καθορίσουμε μη διατεταγμένα σύνολα από υποστοιχεία. Η διάταξη δεν είναι πάντα σημαντική για ανταλλαγή δεδομένων. Ενώ ο συνδυασμός εναλλαγής (η πράξη |) και η πράξη $*$ επιτρέπουν προδιαγραφές μη διατεταγμένων συλλογών ετικετών, είναι πολύ πιο δύσκολο να καθορίσουμε ότι

κάθε ετικέτα μπορεί να εμφανίζεται μόνο μία φορά. Υπάρχει μάλιστα ένα γενικότερο πρόβλημα εκφραστικότητας για το DTD, αφού δεν μπορεί να συλλάβει πολλούς συντακτικούς περιορισμούς ενός εγγράφου.

(γ) Υπάρχει έλλειψη τύπων στα ID και IDREF. Έτσι, δεν υπάρχει τρόπος να καθορίσουμε τον τύπο του στοιχείου στον οποίο θα πρέπει να αναφέρεται μία ιδιότητα IDREF ή IDREFS, πράγμα που μπορεί να οδηγήσει σε σοβαρά λογικά λάθη.

(δ) Τέλος, δεν υπάρχει μέριμνα για νεότερα χαρακτηριστικά της XML, και κυρίως για τα namespaces.

Άλλες γλώσσες προσδιορισμού του σχήματος του XML εγγράφου

Δεδομένων των πολλών αδυναμιών του DTD για τον ορισμό του σχήματος ενός XML εγγράφου, μία σειρά από νέες γλώσσες καθορισμού σχήματος προτάθηκαν σταδιακά, με κάποιες από αυτές να είναι μάλιστα πολύ κοντά στο να λάβουν και τυποποίηση από τον οργανισμό W3C. Αναφέρουμε εν συντομία μερικές από αυτές:

(α) **XML Schema**: Είναι η γνωστότερη και πιο ευρέως χρησιμοποιούμενη γλώσσα ορισμού σχήματος (πέρα της DTD) και ήταν η πρώτη νέα γλώσσα XML σχήματος που έλαβε σύσταση από τον οργανισμό W3C. Μερικά από τα πλεονεκτήματά της είναι τα εξής:

- Επιτρέπει να δημιουργούνται τύποι ορισμένοι από το χρήστη, δίνοντας μάλιστα και πολλές αντικειμενοστρεφείς δυνατότητες για τις δηλώσεις τύπου, όπως κληρονομικότητα και αφηρημένοι τύποι.

- Επιτρέπει στο κείμενο που εμφανίζεται στα στοιχεία να περιορίζεται σε συγκεκριμένους τύπους, όπως αριθμητικούς τύπους σε συγκεκριμένες μορφές ή ακόμα πιο περίπλοκους τύπους, όπως λίστες ή ενώσεις. Περιέχει μάλιστα ποικιλία από πολλούς προκατασκευασμένους τύπους για δεδομένα κειμένου και τιμές ιδιοτήτων.

- Επιτρέπει σε τύπους να περιορίζονται για να δημιουργούνται εξειδικευμένοι τύποι, για παράδειγμα, καθορίζοντας μέγιστες και ελάχιστες τιμές, ή ορίζοντας έναν τύπο ως καθολικό ή τοπικό.

- Επιτρέπει να επεκτείνονται πολύπλοκοι τύποι χρησιμοποιώντας μία μορφή κληρονομικότητας.

- Είναι ένα γνήσιο υπερσύνολο των DTD.

- Επιτρέπει μοναδικότητα και περιορισμούς ξένων κλειδιών.

- Ενοποιείται με χώρους ονομάτων, για να επιτρέπει διαφορετικά μέρη ενός εγγράφου να συμβαδίζουν σε διαφορετικά σχήματα.

- Επιτρέπει κενές τιμές.

Σημειώνεται ότι λόγω των αυξημένων λειτουργιών και δυνατοτήτων, το κόστος της XML Schema είναι πολύ πιο περίπλοκο σε σχέση με τα DTD, μία πολύ σημαντική αδυναμία για τη γλώσσα.

(β) **RELAX NG (REgular LAnguage for XML Next Generation)**: Πρόκειται για μία σχετικά απλή γλώσσα ορισμού σχήματος. Αν και αρχικά δεν ήταν δημοφιλής, τα τελευταία χρόνια έχει διεισδύσει περισσότερο στο XML λογισμικό, και η αποδοχή της έχει βοηθηθεί από την υιοθέτησή της ως πρωτεύον σχήμα για γνωστές εγγραφο-κεντρικές γλώσσες σήμανσης όπως οι DocBook και OpenDocument. Η RELAX NG μοιράζεται με τη γλώσσα XML Schema μία σειρά κοινών χαρακτηριστικών όπως καθορισμό τύπου δεδομένων, υποστήριξη για κανονικές εκφράσεις, υποστήριξη χώρου ονομάτων, και ικανότητα να αναφέρεται σε σύνθετους ορισμούς.

(γ) **DSD (Document Structure Description)**: Πρόκειται για μία εναλλακτική γλώσσα ορισμού σχήματος με τη βοήθεια κανόνων. Σε αντίθεση με την XML Schema παρουσιάζει τα εξής πλεονεκτήματα:

- Πρόκειται για μία πολύ απλούστερη γλώσσα. Για παράδειγμα, η πλήρης τυποποίησή της χωρίς τα παραδείγματα είναι μόνο 15 σελίδες, οπότε η εκμάθησή της είναι σχετικά εύκολη.
- Δεν υπάρχει η έννοια του τύπου ή του υποτύπου, οι κανόνες είναι απευθείας σχετισμένοι με τα ονόματα των στοιχείων.
- Οι κανόνες μπορούν να είναι ιεραρχικοί όντας εξαρτώμενοι από τις τιμές των ιδιοτήτων και το περιεχόμενο των στοιχείων.
- Είναι 100% αυτοπεριγραφούμενη (δηλαδή υπάρχει ένα πλήρες "DSD για τα DSDs").
- Πολλά από τα μη απαραίτητα συστατικά της γλώσσας είναι απομακρυσμένα ή έχουν αντικατασταθεί από αντίστοιχα βασικά και γενικά.

Φυσικά ο κατάλογος των γλωσσών ορισμού σχήματος είναι πραγματικά ατελείωτος. Εμείς μιλήσαμε ακροθιγώς μόνο για τρεις από αυτές, με σκοπό να καταδείξουμε μερικές πλευρές του πολύ δύσκολου προβλήματος επιλογής κάποιας γλώσσας σχήματος. Ο ενδιαφερόμενος αναγνώστης μπορεί να βρει πολλές πληροφορίες στο Διαδίκτυο.

2.2 XPath και Γλώσσα Ερωτήσεων XQuery

Στις προηγούμενες ενότητες περιγράψαμε συνοπτικά τη γλώσσα XML, τη δομή των δεδομένων XML και γλώσσες ορισμού του σχήματος. Αν και αυτές οι ενότητες είναι απαραίτητες σε ένα οποιοδήποτε βασικό εγχειρίδιο XML, ο κεντρικός άξονας γύρω από τον οποίο κινείται η παρούσα διπλωματική δεν είναι άλλος από τις XML ερωτήσεις. Μπορούμε χάριν ευκολίας να φανταστούμε μία XML ερώτηση ως το ανάλογο μίας SQL ερώτησης σε μία σχεσιακή βάση δεδομένων, αν και όπως περιγράφουμε στο επόμενο κεφάλαιο υπάρχει τεράστιο χάσμα (αν και κάποιες αξιοσημείωτες κοινές ιδιότητες) στις μεθόδους αποτίμησης των δύο ειδών ερωτήσεων.

Η τυποποίηση αυτή τη στιγμή αναφέρεται σε ερωτήσεις με πλήρη καθορισμό των δομικών σχέσεων ανάμεσα σε κόμβους. Οι δύο κύριες γλώσσες που συμβάλλουν σε αυτήν την κατεύθυνση είναι οι παρακάτω:

- Η XPath είναι μία γλώσσα για παραστάσεις διαδρομών και είναι στην πραγματικότητα το λίθο πάνω στον οποίο χτίζεται η XQuery.
- Η XQuery έχει προταθεί ως τυποποίηση για ερωτήσεις σε XML δεδομένα. Συνδυάζει λειτουργίες από πολλές προηγούμενες προτάσεις για ερωτήσεις XML, ιδιαίτερα της γλώσσας Quilt.

Στη συνέχεια θα ασχοληθούμε με τις δύο γλώσσες και θα καταδείξουμε μερικά από τα βασικά τους σημεία. Η ανάλυσή μας βασίζεται κυρίως στα [26], [27], [28] και [25]. Τονίζουμε πάντως ότι λόγω της μεγάλου μεγέθους τυποποίησης, δεν μπορέσαμε παρά να θίξουμε εκείνα μόνο τα σημεία που θεωρήσαμε ότι παρουσιάζουν ενδιαφέρον στα πλαίσια της διπλωματικής εργασίας. Ο ενδιαφερόμενος αναγνώστης μπορεί να βρει αναλυτικά εγχειρίδια και τυποποιήσεις στην αναφερθείσα βιβλιογραφία.

Μοντέλο Δένδρου

Η δομή των XML δεδομένων που περιγράψαμε προηγουμένως δεν είναι εύκολο να βρει άμεσα εφαρμογή στην αποτίμηση ερωτήσεων με δενδρική μορφή. Εξαιτίας αυτού, τροποποιούμε ελαφρώς το μοντέλο δεδομένων της XML εισάγοντας ένα **μοντέλο δένδρου**. Ένα XML έγγραφο μοντελοποιείται από ένα δένδρο, με κάθε κόμβο του δένδρου να αντιστοιχεί σε κάποια δομική μονάδα του XML εγγράφου και να έχει μία σειρά από σχέσεις με άλλους κόμβους του μοντέλου δένδρου.

Περισσότερα για το μοντέλο δένδρου και τη σημασία του στο πλαίσιο της XPath (και της XQuery) αναπτύσσουμε παρακάτω στην περιγραφή της XPath. Ως ένα απλό παράδειγμα όμως των παραπάνω, το Σχήμα 2.4 δείχνει μέρος ενός XML εγγράφου, ενώ το Σχήμα 2.5 απεικονίζει το αντίστοιχο XML δέντρο.

2.2.1 XPath

Η XPath αποτελεί μία προσπάθεια να δοθεί μία κοινή σύνταξη και σημασιολογία για τη λειτουργικότητα του μετασχηματισμού XSL [XSLT] και του XPointer, ενώ πλέον παρέχεται και με τη μορφή τυποποίησης από τον οργανισμό W3C. Ο πρωταρχικός σκοπός της είναι η διευθυνσιοδότηση μερών ενός XML εγγράφου. Προς επίτευξη αυτού του σκοπού, παρέχει επίσης βασικές δυνατότητες για χειρισμό συμβολοσειρών, αριθμών και λογικών παραστάσεων. Η XPath χρησιμοποιεί μία πλήρη σύνταξη που δε βασίζεται στη γλώσσα XML για να διευκολυνθεί η χρήση της μέσα σε URIs και σε τιμές XML ιδιοτήτων, ενώ λειτουργεί πάνω στην αφηρημένη, λογική δομή ενός XML εγγράφου, παρά στην επιφανειακή σύνταξη. Το όνομά της προέρχεται από τη συστηματική της χρήση της έννοιας του μονοπατιού για να μπορεί να διασχίζει την ιεραρχική δομή ενός XML εγγράφου.

Εκτός από τη χρήση της για διευθυνσιοδότηση, η XPath είναι σχεδιασμένη έτσι ώστε να έχει ένα φυσικό υποσύνολο που μπορεί να χρησιμοποιηθεί για ταίριασμα (δηλαδή για έλεγχο εάν ένας κόμβος ταιριάζει με ένα πρότυπο). Η λειτουργία αυτή είναι χρήσιμη στην XSLT και επομένως δε θα τη συζητήσουμε επιπλέον.

Η XPath μοντελοποιεί ένα XML έγγραφο σαν ένα δέντρο από κόμβους. Υπάρχουν όμως διαφορετικά είδη κόμβων, συμπεριλαμβανομένων κόμβων στοιχείου, κόμβων ιδιότητας και κόμβων κειμένου. Η XPath ορίζει ένα τρόπο για τον υπολογισμό μιας τιμής συμβολοσειράς για κάθε είδος κόμβου. Σε μερικά είδη κόμβων δίνονται μάλιστα ονόματα. Υποστηρίζει, επίσης, χώρους ονομάτων XML. Επομένως, το όνομα ενός κόμβου μοντελοποιείται ως ένα ζευγάρι αποτελούμενο από ένα τοπικό μέρος και ένα (πιθανώς κενό) URI χώρου ονομάτων, το οποίο καλείται εκτεταμένο όνομα.

Η κύρια συντακτική δομή στην XPath είναι η έκφραση. Μία έκφραση αποτιμάται για να επιστρέψει ένα αντικείμενο, που έχει έναν από τους παρακάτω τέσσερις βασικούς τύπους:

- ένα σύνολο κόμβων (μία μη διατεταγμένη συλλογή από κόμβους χωρίς διπλότυπα)
- λογικός τύπος (αληθής ή ψευδής)
- αριθμός (ένας αριθμός κινητής υποδιαστολής)
- συμβολοσειρά (μία ακολουθία από χαρακτήρες UCS)

Η αποτίμηση μίας έκφρασης XPath γίνεται πάντα σε σχέση με ένα πλαίσιο. Ένα πλαίσιο αποτελείται από:

- έναν κόμβο (κόμβος του πλαισίου)
- ένα ζευγάρι από θετικούς ακεραίους (η θέση πλαισίου και το μέγεθος πλαισίου)
- ένα σύνολο από αντιστοιχίσεις μεταβλητών
- μία βιβλιοθήκη με συναρτήσεις
- το σύνολο των δηλώσεων χώρων ονομάτων στην εμβέλεια της έκφρασης

Η θέση πλαισίου είναι πάντα μικρότερη ή ίση του μεγέθους πλαισίου.

Οι αντιστοιχίσεις μεταβλητών αποτελούνται από μία αντιστοίχιση από ονόματα μεταβλητών σε τιμές μεταβλητών. Η τιμή μίας μεταβλητής είναι ένα αντικείμενο,που μπορεί να είναι οποιουδήποτε τύπου από τους τύπους που είναι δυνατοί για την τιμή μίας έκφρασης (και μπορεί να είναι ακόμη κι άλλων τύπων που δε διευκρινίζονται εδώ).

Η βιβλιοθήκη συναρτήσεων αποτελείται από μία αντιστοίχιση από ονόματα συναρτήσεων στις ίδιες τις συναρτήσεις. Κάθε συνάρτηση παίρνει μηδέν ή περισσότερα ορίσματα και επιστρέφει ένα μοναδικό αποτέλεσμα. Σηνήθως ενδιαφερόμαστε για συναρτήσεις από τη βιβλιοθήκη κορμού, όπου οι συναρτήσεις έχουν τόσο ορίσματα όσο και αποτέλεσμα που ανήκουν στους τέσσερις βασικούς τύπους, αν και γλώσσες όπως η XSL μπορούν να ορίσουν και συναρτήσεις πάνω σε άλλους τύπους δεδομένων.

Οι δηλώσεις χώρων ονομάτων αποτελούνται από μία αντιστοίχιση από προθέματα URIs χώρων ονομάτων.

Οι αντιστοιχίσεις μεταβλητών, η βιβλιοθήκη συναρτήσεων και οι δηλώσεις χώρων ονομάτων που χρησιμοποιούνται στην αποτίμηση μιας υποέκφρασης είναι πάντα οι ίδιες με αυτές της περιέχουσας έκφρασης. Αντίθετα, ο κόμβος πλαισίου, η θέση πλαισίου και το μέγεθος πλαισίου δεν είναι πάντα ίδιοι στις δύο περιπτώσεις. Κάποια είδη εκφράσεων αλλάζουν τον κόμβο πλαισίου, ενώ μόνο τα κατηγορήματα αλλάζουν τη θέση πλασίου και το μέγεθος πλαισίου. Αν η αποτίμηση ενός είδους έκφρασης περιγράφεται με σαφήνεια, θα δηλώνεται πάντα άμεσα αν ο κόμβος πλαισίου, η θέση πλαισίου και το μέγεθος πλαισίου αλλάζουν για την αποτίμηση των υποεκφράσεων. Αν όμως δεν αναφέρεται τίποτα, παραμένουν αμετάβλητα για την αποτίμηση των υποεκφράσεων αυτού του είδους έκφρασης.

Ένα πολύ βασικό είδος έκφρασης είναι η παράσταση διαδρομής.

Ορισμός 2.1. *Παράσταση Διαδρομής* είναι ένα σύνολο από έναν ή περισσότερους κόμβους – δομικά στοιχεία του XML εγγράφου, χωρισμένα με /. □

Η αρχική δείχνει τη ρίζα του εγγράφου. Μία παράσταση διαδρομής εκτελείται από αριστερά προς δεξιά και σε οποιοδήποτε σημείο το αποτέλεσμά της αποτελείται από ένα σύνολο από κόμβους σχετικούς με τον κόμβο πλαισίου. Οι παραστάσεις διαδρομής μπορούν τέλος να περιέχουν αναδρομικά εκφράσεις που χρησιμοποιούνται για να φιλτράρουμε σύνολα από κόμβους.

Σε σχέση και με το μοντέλο δένδρου που αναφέρθηκε πιο πάνω, η σύνταξη της XPath ορίζει το XML έγγραφο ως ένα δέντρο με τα εξής δομικά στοιχεία:

- ρίζα (root (or document) node)
- κόμβος στοιχείου (element node)
- κόμβος κειμένου (text node)
- κόμβος ιδιότητας (attribute node)
- κόμβος σχολίου (comment node)
- κόμβος επεξεργασίας (processing - instruction node)
- κόμβος χώρου ονομάτων (namespace node)

Επίσης, έχουμε μία σειρά ορισμών.

Ορισμός 2.2. *Οι ατομικές τιμές (atomic values)* είναι κόμβοι χωρίς παιδιά ή πατέρα. □

Ορισμός 2.3. *Τα αντικείμενα (items)* είναι είτε ατομικές τιμές ή κόμβοι. □

Σε ό,τι αφορά τις σχέσεις μεταξύ κόμβων, έχουμε τις εξής ιδιότητες:

- γονιός (parent): κάθε στοιχείο και ιδιότητα έχουν ένα ακριβώς γονιό

- παιδί (child): οι κόμβοι στοιχείου μπορεί να έχουν μηδέν, ένα ή περισσότερα παιδιά
- αδέρφια (siblings): οι κόμβοι που έχουν τον ίδιο γονιό
- πρόγονοι (ancestors): ο γονιός του κόμβου, ο γονιός του γονιού κ.ο.κ.
- απόγονοι (descendants): τα παιδιά του κόμβου, τα παιδιά των παιδιών κ.ο.κ.

Για να δούμε πώς δουλεύουν όλα αυτά στην πράξη, έστω στο έγγραφο της Εικόνας 2.2, η παράσταση XPath:

/cinemas-movies/movies/movie/director

Θα επιστραφούν τότε τα εξής στοιχεία:

<director>Steven Spielberg</director>
<director>Clint Eastwood</director>
<director>JamesBrooks</director>

Η παράσταση

/cinemas-movies/movies/movie/director/text()

θα επιστρέψει τα ίδια στοιχεία αλλά χωρίς τις ετικέτες.

Όταν ένα όνομα στοιχείου, όπως το movie, εμφανίζεται πριν από την επόμενη /, αναφέρεται σε όλα τα στοιχεία του συγκεκριμένου ονόματος που είναι παιδιά των στοιχείων στο τρέχον σύνολο στοιχείων. Αφού πολλά παιδιά έχουν το ίδιο όνομα, ο αριθμός των κόμβων στο σύνολο των κόμβων μπορεί να αυξηθεί ή να μειωθεί με κάθε βήμα.

Οι τιμές ιδιοτήτων μπορούν επίσης να προσπελαστούν χρησιμοποιώντας το σύμβολο @. Για παράδειγμα, η έκφραση /cinemas-movies/cinemas/cinema/@name επιστρέφει ένα σύνολο από όλες τις τιμές των ιδιοτήτων name των στοιχείων cinema, δηλαδή το σύνολο {"Alexandra", "Ideal Lux"} για το XML έγγραφο του Σχήματος 2.2.

Η γλώσσα XPath υποστηρίζει διάφορες άλλες λειτουργίες:

- Τα κατηγορήματα επιλογής μπορούν να ακολουθούν οποιοδήποτε βήμα σε μία διαδρομή και περιέχονται σε τετράγωνες αγκύλες. Για παράδειγμα, το

  /cinemas-movies/movies/movie[rating >8]

  επιστρέφει στοιχεία ταινιών με βαθμολογία άνω του 8, ενώ το

  /cinemas-movies/movies/movie[rating > 8]/@name

  επιστρέφει τα ονόματα αυτών των ταινιών.

  Μπορούμε να ελέγξουμε την ύπαρξη ενός υποστοιχείου αναφέροντάς το με οποιονδήποτε τελεστή σύγκρισης. Για παράδειγμα, αν αφαιρούσαμε μόνο το "> 8" από το παραπάνω, η παράσταση θα επέστρεφε τα ονόματα όλων των ταινιών που έχουν ένα υποστοιχείο βαθμολογίας ανεξάρτητα από την τιμή του.

- Η γλώσσα XPath παρέχει διάφορες συναρτήσεις που μπορούν να χρησιμοποιηθούν ως κατηγορήματα επιλογής, συμπεριλαμβανομένου του ελέγχου της θέσης του τρέχοντος κόμβου στη σειρά των αδερφών και μετρώντας τον αριθμό των κόμβων που ταιριάζουν. Για παράδειγμα, η παράσταση διαδρομής

/cinemas-movies/movies/movie/[actor/count() > 1]

επιστρέφει στοιχεία ταινιών με τουλάχιστον δύο αναγραφόμενους ηθοποιούς. Μπορούν να χρησιμοποιηθούν οι λογικές συνδέσεις and και or σε κατηγορήματα, ενώ η συνάρτηση not μπορεί να χρησιμοποιηθεί για άρνηση.

- Ο τελεστής | επιτρέπει να ενώνονται τα αποτελέσματα παραστάσεων. Ωστόσο, δεν μπορεί να είναι ένθετος μέσα σε άλλους τελεστές. Για παράδειγμα, η έκφραση /cinemas-movies/movies/movie/director | /cinemas-movies/movies/movie/actor επιλέγει όλα τα στοιχεία director και actor, που ικανοποιούν την παράσταση διαδρομής.

- Μία XPath παράσταση μπορεί να αγνοήσει πολλαπλά επίπεδα κόμβων χρησιμοποιώντας το "//". Για παράδειγμα, η παράσταση /cinemas-movies//director επιστρέφει στοιχεία director που βρίσκονται οπουδήποτε κάτω από το στοιχείο cinemas-movies. Μπορούν λοιπόν να βρίσκονται τα απαιτούμενα δεδομένα χωρίς πλήρη γνώση των σχέσεων πατέρα-παιδιού αλλά γνωρίζοντας μόνο σχέσεις προγόνου-απογόνου.

- Στην περίπτωση που έχουμε άγνωστα XML στοιχεία μπορούμε να χρησιμοποιήσουμε τα wildcards. Συγκεκριμένα, το wildcard μπορεί να είναι:

  - *: ταιριάζει με οποιονδήποτε κόμβο στοιχείου
  - @*: ταιριάζει με οποιονδήποτε κόμβο ιδιότητας
  - **node()**: ταιριάζει με οποιονδήποτε κόμβο (οποιουδήποτε είδους)

  Για παράδειγμα, η έκφραση //* επιλέγει όλα τα στοιχεία στο XML έγγραφο, η έκφραση /cinemas-movies/* επιλέγει όλους τους κόμβους στοιχείου που είναι παιδιά του κόμβου στοιχείου cinemas-movies, και η έκφραση //movie[@*] επιλέγει όλα τα στοιχεία movie που έχουν οποιαδήποτε ιδιότητα.

- Σε μία XPath έκφραση μπορεί να χρησιμοποιηθεί μία σειρά από τελεστές (μερικούς τους έχουμε αναφέρει ήδη). Οι πιο συχνά χρησιμοποιούμενοι είναι οι |, *+*, *-*, ***, *div*, *=*, *!=*, *<*, *<=*, *>*, *>=*, *or*, *and*, και *mod* με προφανή σημασία.

2.2.2 XQuery

Η γλώσσα XQuery είναι η κατεξοχήν γλώσσα ερωτήσεων πάνω σε XML δεδομένα. Προέρχεται από μία γλώσσα ερωτημάτων που ονομάζεται Quilt και απέκτησε τη σύσταση του οργανισμού W3C στις 23 Ιανουαρίου, 2007. Είναι συμβατή με πολλές W3C τυποποιήσεις, όπως XML, Namespaces, XSLT, XPath, και XML Schema. Μπορούμε να τη φανταστούμε ως το ανάλογο για XML δεδομένα της γλώσσας SQL για τις βάσεις δεδομένων.

Η γλώσσα XQuery χτίζεται πάνω στις εκφράσεις XPath που περιγράψαμε προηγουμένως και, πράγματι, η XQuery 1.0 και η XPath 2.0 μοιράζονται το ίδιο μοντέλο δεδομένων και υποστηρίζουν τους ίδιους τελεστές και τις ίδιες συναρτήσεις. Για παράδειγμα, χρησιμοποιεί παραστάσεις διαδρομής όπως και η XPath για να περιπλανηθεί στα στοιχεία ενός

XML εγγράφου, ενώ χρησιμοποιεί κατηγορήματα επιλογής για να περιορίσει τα δεδομένα από τα XML έγγραφα σε αυτά που πληρούν κάποια επιθυμητή ιδιότητα.

Μερικοί βασικοί κανόνες σύνταξης της XQuery:

- Η XQuery είναι case-sensitive.
- Τα XQuery στοιχεία, ιδιότητες και μεταβλητές πρέπει να είναι έγκυρα XML ονόματα.
- Μία συμβολοσειρά μπορεί να περικλείεται είτε από "" είτε από ′′, δηλαδή είτε από μονά είτε από διπλά εισαγωγικά.
- Μία μεταβλητή αναφέρεται με το $, το οποίο ακολουθείται από το όνομα της μεταβλητής. Για παράδειγμα, $movie-name.
- Τα σχόλια περικλείονται ανάμεσα σε (: και σε :). Για παράδειγμα, (: comment :).

Οι ερωτήσεις στην XQuery είναι οργανωμένες σε παραστάσεις που αποτελούνται από τέσσερις ενότητες: *for, let, where* και *return* (εν συντομία, αναφέρονται ως "FLWR"). Η ενότητα for δίνει μία σειρά από μεταβλητές που διαφέρουν από τα αποτελέσματα των παραστάσεων XPath. Όταν καθορίζονται περισσότερες από μία μεταβλητές, τα αποτελέσματα περιλαμβάνουν το καρτεσιανό γινόμενο των πιθανών τιμών που μπορούν να λάβουν οι μεταβλητές. Ο όρος let απλώς επιτρέπει να αντιστοιχούνται περίπλοκες παραστάσεις σε ονόματα μεταβλητών, για απλότητα της αναπαράστασης. Η ενότητα where κάνει επιπλεόν ελέγχους για τις συνδεδεμένες εγγραφές από την ενότητα for. Τέλος, η ενότητα return επιτρέπει την κατασκευή αποτελεσμάτων στην XML, τα οποία και επιστρέφει.

Μία απλή παράσταση FLWR που επιστρέφει τα ονόματα ταινιών με βαθμολογία μεγαλύτερη από 8 του Σχήματος 2.2 παρουσιάζεται παρακάτω:

for $x **in** /cinemas-movies/movies/movie
let $movie-name := $x/@name
where $x/rating > 8
return <movie-name>$movie-name</movie-name>

Θα επιστραφούν λοιπόν τα εξής στοιχεία:

<movie-name>"Million Dollar Baby"</movie-name>
<movie-name>"As Good As It Gets"</movie-name>

Αφού η ερώτηση αυτή είναι πολύ απλή, ο όρος let δεν είναι απαραίτητος και η μεταβλητή $movie-name στον όρο return θα μπορούσε να αντικατασταθεί με την $x/@name. Επίσης, αφού ο όρος for χρησιμοποιεί παραστάσεις XPath, οι επιλογές μπορούν να συμβούν μέσα στην παράσταση XPath με την εισαγωγή κάποιου κατηγορήματος επιλογής. Έτσι, μία ισοδύναμη ερώτηση μπορεί να έχει μόνο όρους for και return:

for $x **in** /cinemas-movies/movies/movie[rating > 8]
return <movie-name>$x/@name</movie-name>

Πάντως, ο όρος let απλοποιεί περίπλοκες ερωτήσεις.

Οι παραστάσεις διαδρομών στην XQuery μπορεί να επιστρέψουν ένα πολλαπλό σύνολο, με επαναλαμβανόμενους κόμβους. Η συνάρτηση distinct μπορεί να χρησιμοποιηθεί ακόμα

και με ένα όρο for. Η XQuery παρέχει επίσης συνοπτικές συναρτήσεις με sum και count, που μπορούν να εφαρμοστούν σε συλλογές όπως σύνολα και πολλαπλά σύνολα. Επίσης, οι μεταβλητές που παίρνουν όρους let μπορεί να είναι σύνολα ή πολλαπλά σύνολα, αν η παράσταση της διαδρομής στη δεξιά πλευρά επιστρέφει ένα σύνολο ή πολλαπλό σύνολο.

Τα αποτελέσματα μπορούν να ταξινομηθούν στην XQuery με έναν όρο order by. Ο όρος αυτός καθορίζει πώς θα πρέπει να ταξινομηθούν τα στιγμιότυπα της παράστασης. Για παράδειγμα, η προηγούμενη ερώτηση θα μπορούσε να γραφεί:

for $x **in** /cinemas-movies/movies/movie
let $movie-name := $x/@name
where $x/rating > 8
order by $x/rating **descending**
return <movie-name>$movie-name</movie-name>

Με αυτήν την τροποποίηση, οι ταινίες θα επιστραφούν με φθίνουσα σειρά βαθμολογίας. Αν θέλαμε με αύξουσα, απλά θα παραλείπαμε τον όρο descending. Η ταξινόμηση μπορεί να γίνει σε πολλαπλά επίπεδα ένθεσης.

Πρέπει επίσης να αναφέρουμε ότι η XQuery παρέχει και τη δυνατότητα έκφρασης υπό συνθήκη *if-then-else* με προφανή λειτουργικότητα. Για παράδειγμα, η ερώτηση:

for $x **in** /cinemas-movies/movies/movie
return if ($x/rating=10)
then <masterpiece-name>$x/@name</masterpiece-name>
else <moviename>$x/@name</moviename>

θα επιστρέψει τα εξής αποτελέσματα:

<masterpiece-name>"Million Dollar Baby"</masterpiece-name>
<movie-name>"As Good As It Gets"</movie-name>

Η γλώσσα XQuery παρέχει διάφορες ενσωματωμένες συναρτήσεις (πάνω από 100) και υποστηρίζει συναρτήσεις ορισμένες από το χρήστη. Για παράδειγμα, η ενσωματωμένη συνάρτηση document(name) επιστρέφει τη ρίζα ενός εγγράφου με όνομα name. Η ρίζα μπορεί μετά να χρησιμοποιηθεί σε μία παράσταση διαδρομής για πρόσβαση στα περιεχόμενα του εγγράφου. Μία κλήση σε μία συνάρτηση μπορεί να εμφανιστεί οπουδήποτε μπορεί να εμφανιστεί μία έκφραση. Για παράδειγμα, μπορεί να βρεθεί μέσα σε ένα κατηγόρημα επιλογής, όπως στο παράδειγμα /cinemas-movies/movies/movie[substring(director,1,5)='Clint'], το οποίο ελέγχει για στοιχεία director των οποίων οι 5 πρώτοι χαρακτήρες είναι 'Clint'.

2.2.3 Αποτίμηση Ερωτήσεων

Η γλώσσα XQuery αποτελεί μία τυποποιημένη προσπάθεια ορισμού μίας γλώσσας ερωτήσεων σε XML έγγραφα, τα οποία έχουν μοντελοποιηθεί με το μοντέλο δένδρου. Σαφώς, μία τέτοια δυνατότητα είναι εκ των ων ουκ άνευ, αφού δε θα είχε νόημα η ανάπτυξη μίας de facto γλώσσας ανταλλαγής δεδομένων όπως η XML χωρίς τη δυνατότητα να υποβάλλουμε ερωτήσεις. Θα ήταν σαν να οργανώναμε μία σχεσιακή βάση δεδομένων για την αποθήκευση των δεδομένων ενός τεράστιου οργανισμού, χωρίς όμως να έχουμε τη δυνατότητα να εκτελούμε

ερωτήσεις σε αυτή τη βάση δεδομένων μέσω της κάποιας γλώσσας ερωτήσεων όπως η SQL: αργά ή γρήγορα η βάση δεδομένων θα καθίστατο άχρηστη! Έτσι και με τη γλώσσα XML: για τη λειτουργικότητά της έχει ανάγκη από τις γλώσσες XPath και XQuery.

Ωστόσο, η γλώσσα XQuery, αν και εξαιρετικής σπουδαιότητας, αποτελεί το εξωτερικό περίβλημα διεπαφής του χρήστη του XML εγγράφου με το έγγραφο. Στην πραγματικότητα, παράλληλα με την ανάπτυξη γλωσσών ερωτήσεων διεξάγεται σημαντική έρευνα για το *πώς* μπορούν να αποτιμηθούν οι ερωτήσεις που εκφράζουμε μέσω μίας γλώσσας ερωτήσεων, όπως η XQuery (αντίστοιχα με την έρευνα στο χώρο των βάσεων δεδομένων για το *πώς* μπορούν να υπολογιστούν αποδοτικά SQL ερωτήσεις).

Ας θεωρήσουμε για παράδειγμα, εμπνευσμένοι από το έγγραφο του Σχήματος 2.1, την εκφρασμένη σε XQuery ερώτηση:

```
for $c in /cinemas/cinema[@name="Alexandra" or region="Stadiou"]
 for $m in $c/movies/movie
 return $m/@name
```

Σημασιολογικά, η ερώτηση αυτή επιστρέφει τους τίτλους των ταινιών που είτε παίζονται στον κινηματογράφο "Alexandra" ή στην περιοχή "Stadiou". Περιέχει τρεις XPath παραστάσεις διαδρομής, μία σε καθένα από τους όρους for, και μία στον όρο return. Μπορούμε εύκολα να φανταστούμε το προκύπτον δένδρο της ερώτησης. Υπάρχουν AND και OR κόμβοι διακλάδωσης, οι οποίοι καθορίζουν ότι όλοι ή τουλάχιστον ένα από τα (υπο)πρότυπα κάτω από τον κόμβο διακλάδωσης πρέπει να ταιριάζουν, αντίστοιχα.

Με την XQuery περιγράφουμε λοιπόν τη σημασιολογία της ερώτησης, αλλά δεν υπεισερχόμαστε στο σκέλος του πώς θα γίνει η αποτίμηση αυτή, το οποίο βρίσκεται στον πυρήνα του προβλήματος των ερωτήσεων. Θα γίνει αναδιάταξη του δέντρου ερώτησης για να καταλήξουμε σε ένα απλούστερο δενδρικό πρότυπο; Θα υπολογιστούν οι παραστάσεις διαδρομής με τη δεδομένη σειρά ή θα γίνει κάποια βελτιστοποίηση; Και, επίσης, πώς θα υπολογιστούν οι απαντήσεις στις διάφορες εκφράσεις XPath; Με έναν απλοϊκό αλγόριθμο διάσχισης του XML εγγράφου, μέσω μετάφρασης στο σχεσιακό μοντέλο, ή με μία νέα τεχνική;

Όπως βλέπουμε τα ερωτήματα που γεννώνται είναι πολλά και είναι ιδιαίτερα προκλητικό να τα αντιμετωπίσουμε αποδοτικά. Στα επόμενα δύο κεφάλαια, θα προσπαθήσουμε να απαντήσουμε στο γενικότερο ερώτημα αποτίμησης ερωτήσεων με πρότυπο δενδρικής μορφής αλλά και με πρότυπο μονοπατιού μερικώς καθορισμένο, προσπαθώντας να ορίσουμε ακροθιγώς μία γλώσσα ορισμού ερωτήσεων μονοπατιού μερικής δομής.

2.3 Σχετικές Εργασίες

Οι ερωτήσεις για δεδομένα XML περιλαμβάνουν δομικά πρότυπα αποτελούμενα από στοιχειακόμβους και δομικές σχέσεις μεταξύ τους. Το ταίριασμα (match) 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 δομικών σχέσεων. Στο [12] παρουσιάζεται ο αλγόριθμος Multi-Predicate Merge Join (MPMGJN) για την αποτίμηση αυτών των ερωτήσεων. Ο Al-Khalifa και άλλοι ([1]) εισάγουν μια οικογένεια αλγορίθμων στοίβας που είναι πιο αποτελεσματικοί από τον MPMGJN, καθώς δεν απαιτούν πολλαπλή διάσχιση του XML δέντρου. Αλγόριθμοι για αποτίμηση ερωτήσεων με διάσπασή τους σε δυαδικές δομικές σχέσεις μελετώνται στο [23] και προτείνονται τρόποι βελτιστοποίησης της σειράς συνένωσης των σχέσεων αυτών.

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, ο Bruno και άλλοι ([2]) παρουσίασαν δυο αλγορίθμους στοίβας, τον `PathStack` και τον `TwigStack`, για την αποτίμηση ερωτήσεων μονοπατιού και δέντρου αντίστοιχα. Ο `PathStack` είναι βέλτιστος για ερωτήσεις μονοπατιού, ενώ ο `TwigStack` είναι βέλτιστος για ερωτήσεις δέντρου χωρίς σχέσεις γονέα-παιδιού (parent-child relationships).

Πολλοί ερευνητές έχουν καταπιαστεί με διάφορες επεκτάσεις του `TwigStack`. Για παράδειγμα, στο [9], ο αλγόριθμος `TwigStackList` αποτιμά αποτελεσματικά δεντρικές ερωτήσεις με σχέσεις γονέα-παιδιού. Επιπλέον, στο [24], ο αλγόριθμος `Twig2Stack` αποτιμά γενικευμένες ερωτήσεις δέντρου με προαιρετικές δομικές σχέσεις. Ο Chen και άλλοι στο [3] προτείναν αλγορίθμους που χειρίζονται ερωτήσεις σε δεδομένα με δομή dag. Μέθοδοι αποτίμησης ερωτήσεων δέντρου με κατηγορήματα OR αναπτύσσονται στο [5]. Στο [7], το ευρετήριο XR-tree χρησιμοποιείται για την αποφυγή του χειρισμού δεδομένων που δε συμμετέχουν στην απάντηση της ερώτησης. Το [4] παρουσιάζει τον αλγόριθμο `TwigOptimal`, που χρησιμοποιεί τους virtual cursors [11] για να επιταχύνει τη διάσχιση του XML δέντρου.

Ερωτήσεις δενδρικού προτύπου μερικώς καθορισμένης δομής εισήχθησαν από το Θεοδωράτο και άλλους στα [15, 16]. Σε αυτές τις δημοσιεύσεις, οι ερωτήσεις μερικώς καθορισμένης δομής αποτιμώνται δημιουργώντας ένα σύνολο από πλήρεις δενδρικές ερωτήσεις με βάση τους γράφους ευρετηρίου (δομημένες περιλήψεις των δεδομένων). Τέλος, ο Σουλδάτος και άλλοι προτείνουν στο [17] μια νέα γλώσσα ερωτήσεων για ερωτήσεις προτύπου μονοπατιού μερικής δομής και εισάγουν τους αλγόριθμους PartialMJ και PartialPathStack για την αποτίμηση των ερωτήσεων αυτών.

```
<cinemas>
  <cinema name="Alexandra" region="Ampelokhpoi">
    <movies>
      <movie name="War of the Worlds" reservations="1" max_reservations="3">
        <director>Steven Spielberg</director>
        <actor>Tom Cruise</actor>
        <genre>Science Fiction</genre>
        <rating>6</rating>
        <hour>21:00</hour>
        <reservations>
          <reservation>
            <name>George Manolis</name>
            <payment>Cash</payment>
          </reservation>
        </reservations>
      </movie>
      <movie name="Million Dollar Baby" reservations="0" max_reservations="4">
        <director>Clint Eastwood</director>
        <actor>Hillary Swank</actor>
        <genre>Drama</genre>
        <rating>10</rating>
        <hour>20:00</hour>
        <reservations/>
      </movie>
    </movies>
  </cinema>
  <cinema name="Ideal Lux" region="Panepisthmiou">
    <movies>
      <movie name="As Good As It Gets" reservations="2" max_reservations="2">
        <director>James Brooks</director>
        <actor>Jack Nicholson</actor>
        <actor>Helen Hunt</actor>
        <genre>Comedy</genre>
        <rating>8.5</rating>
        <hour>22:00</hour>
        <reservations>
          <reservation>
            <name>Vasiliki Ntikou</name>
            <payment>Credit Card</payment>
          </reservation>
        </reservations>
      </movie>
    </movies>
  </cinema>
  <cinema name="Apollon" region="Stadiou">
    This cinema is one of the most historic in Athens.
    <movies>
      <movie name="Titanic" reservations="2" max_reservations="4">
        <director>James Cameron</director>
        <actor>Leonardo Di Caprio</actor>
        <actor>Kate Winslet</actor>
        <genre>Historic Drama</genre>
        <rating>9</rating>
        <hour>20:00</hour>
        <reservations>
          <reservation>
            <name>Antonis Aleksandrou</name>
            <payment>Cash</payment>
          </reservation>
        </reservations>
      </movie>
    </movies>
  </cinema>
</cinemas>
```

Σχήμα 2.1: Παράδειγμα XML Εγγράφου.

```
<cinemas-movies>
  <cinemas>
    <cinema name="Alexandra" region="Ampelokhpoi">
      <cinema-id>1</cinema-id>
    </cinema>
    <cinema name="Ideal Lux" region="Panepisthmiou">
      <cinema-id>2</cinema-id>
    </cinema>
  </cinemas>
  <movies>
    <movie name="War of the Worlds" reservations="1" max_reservations="3">
      <movie-id>1</movie-id>
      <director>Steven Spielberg</director>
      <actor>Tom Cruise</actor>
      <genre>Science Fiction</genre>
      <rating>6</rating>
      <hour>21:00</hour>
      <reservations>
        <reservation>
          <name>George Manolis</name>
          <payment>Cash</payment>
        </reservation>
      </reservations>
    </movie>
    <movie name="Million Dollar Baby" reservations="0" max_reservations="4">
      <movie-id>2</movie-id>
      <director>Clint Eastwood</director>
      <actor>Hillary Swank</actor>
      <genre>Drama</genre>
      <rating>10</rating>
      <hour>20:00</hour>
      <reservations/>
    </movie>
    <movie name="As Good As It Gets" reservations="2" max_reservations="2">
      <movie-id>3</movie-id>
      <director>James Brooks</director>
      <actor>Jack Nicholson</actor>
      <actor>Helen Hunt</actor>
      <genre>Comedy</genre>
      <rating>8.5</rating>
      <hour>22:00</hour>
      <reservations>
        <reservation>
          <name>Vasiliki Ntikou</name>
          <payment>Credit Card</payment>
        </reservation>
      </reservations>
    </movie>
  </movies>
  <movies_in_cinema>
    <cinema-id>1</cinema-id>
    <movie-id>1</movie-id>
    <movie-id>2</movie-id>
  </movies_in_cinema>
  <movies_in_cinema>
    <cinema-id>2</cinema-id>
    <movie-id>3</movie-id>
  </movies_in_cinema>
</cinemas-movies>
```

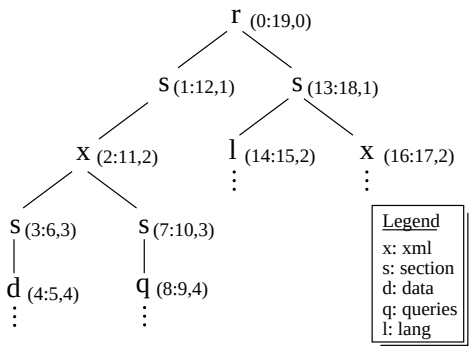
Σχήμα 2.2: Το έγγραφο του Σχήματος 2.1 χωρίς ένθεση μεταξύ των στοιχείων cinema και movies.

```
<!DOCTYPE cinemas [

<!ELEMENT cinemas( cinema* )>
<!ELEMENT cinema( movies )>
<!ELEMENT movies( (movie)* )>
<!ELEMENT movie( director, actor+, genre, rating, hour, reservations )>
<!ELEMENT reservations( reservation* )>
<!ELEMENT reservation( name, payment )>
<!ELEMENT director( #PCDATA )>
<!ELEMENT actor( #PCDATA )>
<!ELEMENT genre( #PCDATA )>
<!ELEMENT rating( #PCDATA )>
<!ELEMENT hour( #PCDATA )>
<!ELEMENT name( #PCDATA )>
<!ELEMENT payment( #PCDATA )>

<!ATTLIST cinema name CDATA #REQUIRED>
<!ATTLIST cinema region CDATA #IMPLIED>
<!ATTLIST movie name CDATA #REQUIRED>
<!ATTLIST movie reservations CDATA #IMPLIED>
<!ATTLIST movie max_reservations CDATA #IMPLIED>

]>
```

Σχήμα 2.3: Παράδειγμα ενός DTD.

```
<r>
  <section>
    <xml>
      <section>
        <data>...</data>
      </section>
      <section>
        <queries>...</queries>
      </section>
    </xml>
  </section>
  <section lang="...">
    <xml>...</xml>
  </section>
</r>
```

Σχήμα 2.4: XML Έγγραφο D.



Σχήμα 2.5: XML Δέντρο T.

Κεφάλαιο 3

Τεχνικές Επεξεργασίας Ερωτήσεων XML με Πρότυπα Μονοπάτια και Δέντρα

3.1 Προκαταρκτικές Έννοιες

Το κεφάλαιο αυτό στοχεύει στην ανάλυση των γνωστότερων και σπουδαιότερων τεχνικών επεξεργασίας ερωτήσεων XML με πρότυπα μονοπάτια και δέντρα που έχουν προταθεί στην ευρύτερη βιβλιογραφία. Προς αυτήν την κατεύθυνση είναι απαραίτητη η εισαγωγή κάποιων προκαταρκτικών εννοιών γύρω από τον ίδιο τον ορισμό του προβλήματος, αλλά και των διαφόρων δομών δεδομένων που διευκολύνουν την επίλυσή του.

Δεδομένου ότι το πρόβλημα των ερωτήσεων με πρότυπα μονοπάτια είναι στην ουσία γνήσια υποπερίπτωση του γενικότερου προβλήματος της ερώτησης με δενδρικά πρότυπα, θα ασχοληθούμε διεξοδικά μόνο με το πρόβλημα των δενδρικών ερωτήσεων, το οποίο περιλαμβάνει το ευκολότερο πρόβλημα των ερωτήσεων μονοπατιού. Μοναδική εξαίρεση σε αυτό είναι η επισταμένη ασχολία μας με το πρόβλημα των ερωτήσεων μονοπατιού στην ενότητα 3.3 που προτείνεται στην εργασία [2], αφού η κατανόησή του είναι ο θεμέλιος λίθος για την εμπέδωση των μεθόδων επεξεργασίας ερωτήσεων μονοπατιού μερικής δομής που αναφέρονται στο επόμενο κεφάλαιο.

3.1.1 Ταίριασμα Ερωτήσεων με Δενδρικά Πρότυπα

Μία ερώτηση με δενδρικό πρότυπο είναι στην ουσία ένα κατηγόρημα επιλογής πάνω σε πολλαπλά στοιχεία σε μία XML βάση δεδομένων, η οποία με τη σειρά της μπορεί να θεωρηθεί ως μία συλλογή από XML έγγραφα, σύμφωνα με όσα περιγράφησαν λεπτομερειακά στο προηγούμενο κεφάλαιο. Τέτοιες ερωτήσεις μπορούν γενικά να αναπαρασταθούν ως δέντρα με σημασμένους κόμβους. Το *ταίριασμα* μιας τέτοιας ερώτησης πάνω σε μία XML βάση δεδομένων ισοδυναμεί με την εύρεση όλων των εμφανίσεων του δενδρικού προτύπου μέσα στη βάση δεδομένων. Πιο τυπικά:

Ορισμός 3.4. Δοθεισών μίας ερώτησης με δενδρικό πρότυπο Q και μίας XML βάσης δεδομένων D, *ταίριασμα* του Q στη βάση D είναι μία αντιστοίχιση από τους κόμβους της ερώτησης Q στους κόμβους της βάσης δεδομένων D, έτσι ώστε: (i) τα κατηγορήματα των κόμβων της ερώτησης ικανοποιούνται από τους αντίστοιχους κόμβους της βάσης D, (ii) οι δομικές σχέσεις (πατέρα-παιδιού ή προγόνου-απογόνου) μεταξύ των κόμβων της Q ικανοποιούνται από τους αντίστοιχους κόμβους της D. □

Η *απάντηση* της Q με n κόμβους μπορεί να αναπαρασταθεί ως μία n-αδική σχέση, όπου κάθε εγγραφή (d_1,d_2, ..., d_n) αποτελείται από κόμβους της XML βάσης δεδομένων, οι οποίοι καθορίζουν ένα ξεχωριστό ταίριασμα της Q στη D.

Σημειώνουμε ότι αν και για την πληρότητα της θεωρίας αναφέρουμε το γενικό ορισμό, στην παρούσα εργασία θα ασχοληθούμε μόνο με το ειδικό πρόβλημα των ερωτήσεων πάνω σε XML δέντρα, και όχι σε XML βάσεις δεδομένων.

3.1.2 Αναπαράσταση της θέσης σε XML δεδομένα και Σχήματα Κωδικοποίησης

Ας σκεφτούμε για λίγο μία εφαρμογή, όπου μία σειρά από ερωτήσεις γίνεται σε μία XML βάση δεδομένων πολύ μεγάλου μεγέθους. Μία χονδροειδής λύση θα ήταν να διασχίζουμε τα διάφορα XML δέντρα με χρήση κάποιου από τους γνωστούς αλγόριθμους διάσχισης δένδρων ($post-order, pre-order, in-order$) και στη συνέχεια να βρίσκαμε την απάντηση στην αρχική ερώτηση εφαρμόζοντας κάποιον αλγόριθμο επεξεργασίας πάνω στα δεδομένα που συλλέξαμε από τον αλγόριθμο διάσχισης. Μία τέτοια απλοϊκή προσέγγιση θα είχε όμως το ανυπέρβλητο μειονέκτημα ότι για κάθε ερώτηση θα έπρεπε να εκτελούμε διάσχιση του δένδρου, ακόμη κι αν είχαμε μία σταθερή βάση δεδομένων όπου εκτελούμε επαναλαμβανόμενα ερωτήματα. Επιπλέον, η τεχνική αυτή θα μας καταδίκαζε στη διάσχιση ολόκληρου του δένδρου, το οποίο μπορεί να περιέχει κόμβους με πολλές διαφορετικές ετικέτες, ακόμη κι αν τα κατηγορήματα επιλογής της ερώτησής μας είναι πολύ λιγότερα.

Για τους παραπάνω λόγους είναι λοιπόν επιτακτική η ανάγκη προσδιορισμού ενός αποδοτικού σχήματος σήμανσης των κόμβων της βάσης δεδομένων, ώστε να καταφέρουμε να "συλλάβουμε" τη δομική πληροφορία των XML εγγράφων που είναι απαραίτητη για την επεξεργασία της ερώτησης. Κάθε ένα από τα σχήματα αυτά έχει διαφορετική *χωρική πολυπλοκότητα* ανάλογα με το μέγεθος της πληροφορίας που αποθηκεύει και διαφορετικό *πληροφοριακό περιεχόμενο*, το οποίο μπορούμε να φανταστούμε ως ένα δείκτη του πόση πληροφορία σχετικά με τα XML έγγραφα προσφέρει ο εν λόγω μηχανισμός. Τα δύο αυτά μεγέθη έχουν συνήθως αρνητική συσχέτιση για προφανείς λόγους.

Παρακάτω παρουσιάζουμε μερικούς από τους σημαντικότερους μηχανισμούς αναπράστασης θέσης που αναφέρονται στη βιβλιογραφία, ενώ στις επόμενες ενότητες θα φανεί η πρακτική χρησιμότητά τους.

Κωδικοποίηση θέσης με χρήση Ανεστραμμένων Ευρετηρίων

Η έννοια των *ανεστραμμένων ευρετηρίων* είναι στενά συνδεδεμένη με την ερευνητική περιοχή της ανάκλησης πληροφορίας ([25], κεφάλαιο 22.5). Στην περίπτωση των XML βάσεων δεδομένων, ένα ανεστραμμένο ευρετήριο αποτελείται από μία λίστα για κάθε ξεχωριστό στοιχείο σήμανσης που περιέχεται στο σύνολο δεδομένων, όπου το στοιχείο σήμανσης είναι συνήθως μία επικεφαλίδα, μία ιδιότητα ή ένα στοιχείο κειμένου. Η αντίστοιχη λίστα περιλαμβάνει ένα στοιχείο για κάθε εμφάνιση του στοιχείου σήμανσης στη βάση δεδομένων και είναι ταξινομημένη κατά τις πληροφορίες θέσης σύμφωνα με ένα μηχανισμό αναπαράστασης θέσης που ονομάζεται *κωδικοποίηση θέσης*.

Η κωδικοποίηση αυτή αναφέρεται στους XML κόμβους αποθηκεύοντας πληροφορία σχετικά με την αρχή, το τέλος και το επίπεδο ενός κόμβου στο XML δέντρο. Για λόγους απλότητας και δίχως βλάβη της γενικότητας, δε θεωρούμε την πληροφορία του αριθμού εγγράφου, υποθέτοντας ότι μόνο ένα XML έγγραφο είναι υπό επεξεργασία. Η ιδέα επεκτείνεται εύκολα και σε μεγαλύτερο αριθμό εγγράφων.

Η αρχή και το τέλος μπορούν να ευρεθούν απαριθμώντας όλα τα στοιχεία (επικεφαλίδες αρχής, επικεφαλίδες τέλους, μονάδες κειμένου) της βάσης δεδομένων. Οι αριθμοί που αποδίδονται στις επικεφαλίδες αρχής και τέλους ενός κόμβου συμπίπτουν με την αρχή και το τέλος αντίστοιχα. Για να αποφύγουμε ένα διαφορετικό μηχανισμό για τις μονάδες κειμένου, εδώ υποθέτουμε ότι η αρχή και το τέλος συμπίπτουν για αυτούς. Το επίπεδο αναπαριστά το βάθος του κόμβου στο XML δέντρο, με τη ρίζα να είναι στο επίπεδο 0. Είναι μάλιστα ίσος με τον αριθμό των επικεφαλίδων όπου φωλιάζεται ο κόμβος. Τελικά, κάθε στοιχείο θα χαρακτηρίζεται από την πληροφορία: ($start$:end,$level$), όπου όμως στις μονάδες κειμένου οι τιμές των $start$ και end συμπίπτουν. Εναλλακτικά, θα μπορούσαμε για τις μονάδες κειμένου να χρησιμοποιήσουμε την κωδικοποίηση ($start$,$level$) (να παραλείψουμε δηλαδή το end που ισούται με το $start$), με το πιθανό όμως μειονέκτημα της μη ομοιομορφίας στις κωδικοποίησης των στοιχείων επικεφαλίδας και των μονάδων κειμένου.

Το ανεστραμμένο ευρετήριο ταξινομείται κατά αύξουσα σειρά της πληροφορίας αρχής κάθε στοιχείου του XML εγγράφου, έτσι ώστε τα φωλιασμένα στοιχεία να βρίσκονται μετά τα εξωτερικά στοιχεία του συνόλου δεδομένων .

Η αναπαράσταση αυτή μας παρέχει τη δυνατότητα ελέγχου για ύπαρξη δομικών σχέσεων. Συγκεκριμένα:

(α)σχέση προγόνου-απογόνου: ένας κόμβος του δέντρου n_2, του οποίου η θέση κωδικοποιείται ως: ($start_2$:end_2,$level_2$), είναι απόγονος ενός κόμβου του δένδρου n_1, του οποίου η θέση κωδικοποιείται ως ($start_1$:end_1,$level_1$), αν και μόνο αν $level_1 < level_2$, και $end_2 < end_1$. Στην περίπτωση που η κωδικοποίηση θέσης περιλαμβάνει επίσης και τον αριθμό εγγράφου, δηλαδή είναι της μορφής: ($docnumber$,$start$:end,$level$), όπου $docNumber$ ο αριθμός εγγράφου, θα πρέπει επιπλέον να ισχύει στην προηγούμενη ισοδυναμία ότι $docnumber_1 = docnumber_2$.

(β)σχέση πατέρα-παιδιού: ένας κόμβος του δέντρου n_2, του οποίου η θέση κωδικοποιείται ως: ($start_2$:end_2,$level_2$), είναι παιδί ενός κόμβου του δένδρου n_1, του οποίου η θέση κωδικοποιείται ως ($start_1$:end_1,$level_1$), αν και μόνο αν $level_1 + 1 = level_2$, $end_2 < end_1$, και

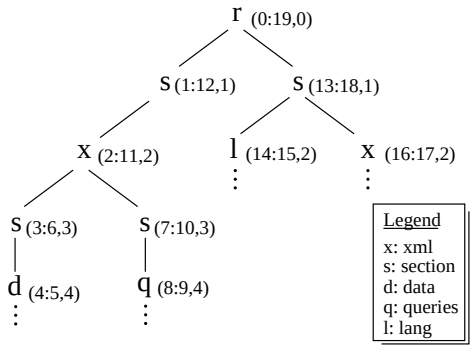
$start_1 < start_2$. Όπως και πριν, στην περίπτωση που η κωδικοποίηση θέσης περιλαμβάνει επίσης και τον αριθμό εγγράφου, δηλαδή είναι της μορφής: ($docnumber$,$start$:end,$level$), όπου $docNumber$ ο αριθμός εγγράφου, θα πρέπει επιπλέον να ισχύει στην προηγούμενη ισοδυναμία ότι $docnumber_1 = docnumber_2$.

(γ)κοντινότητα δύο κόμβων: δύο κόμβοι, n_1 και n_2 με αντίστοιχες κωδικοποιήσεις θέσης $docnumber_1$,$start_1$:end_1,$level_1$) και $docnumber_2$,$start_2$:end_2,$level_2$) έχουν κοντινότητα k, αν και μόνο αν $|start_1$ - $start_2| = k$ και $docnumber_1 = docnumber_2$. Προφανώς, αν υπάρχει κόμβος με μεγαλύτερο $level$, αυτός θα βρίσκεται πιο βαθειά μέσα στο XML έγγραφο.

Ένα σημαντικό σημείο που πρέπει να προσεχθεί σχετικά με την περιγραφείσα κωδικοποίηση θέσης των κόμβων του XML δέντρου είναι ότι ο έλεγχος για σχέση προγόνου-απογόνου είναι τόσο απλός όσο ο έλεγχος για σχέση πατέρα-παιδιού (μπορούμε να ελέγξουμε για την ύπαρξη σχέσης απογόνου-προγόνου) χωρίς να έχουμε γνώση των ενδιάμεσων κόμβων στο μονοπάτι), ενώ εξίσου απλή είναι η εύρεση της κοντινότητας μεταξύ δύο κόμβων.

Οι παραπάνω ιδιότητες μας δίνουν, τέλος, τη δυνατότητα να έχουμε μία ποικιλία από λειτουργίες πάνω στα ανεστραμμένα ευρετήρια. Για να επεξεργαστούμε την έκφραση "$a//b$", τα ανεστραμμένα ευρετήρια των a και b ανακαλούνται και οι διάφορες εμφανίσεις σε αυτά συγχωνεύονται, αν και μόνο αν ικανοποιούν τη σχέση προγόνου-απογόνου. Με τελείως ανάλογο τρόπο χειριζόμαστε και εκφράσεις της μορφής "a/b", με τη διαφορά ότι τώρα θέλουμε οι διάφορες εμφανίσεις στα δύο ευρετήρια να ικανοποιούν τη σχέση πατέρα-παιδιού που περιγράψαμε προηγουμένως.

Παράδειγμα 3.1. *Οι ιδιότητες αρχής, τέλους, επιπέδου των στοιχείων στην XML βάση δεδομένων του Σχήματος 3.1 αναγράφονται δίπλα σε κάθε στοιχείο. Για παράδειγμα, το πρώτο παιδί της ρίζας, που σημαίνεται από S έχει αναπαράσταση θέσης (1:12, 1), αφού (α) η επικεφαλίδα $< S >$ έχει δείκτη 1 στη βάση δεδομένων, (β) η επικεφαλίδα $< /S >$ έχει δείκτη 12 και (γ) S είναι στο επίπεδο 1, φωλιασμένο μόνο από τη ρίζα, δηλαδή ένα στοιχείο.* □



Σχήμα 3.1: XML Δέντρο T.

Σχήμα Εκτεταμένης Κωδικοποίησης Dewey

Αν και η κωδικοποίηση θέσης που αναφέρθηκε στην προηγούμενη ενότητα μπορεί να εκτελέσει ορισμένες σημαντικές λειτουργίες, όπως έλεγχος της σχέσης προγόνου-απογόνου

μεταξύ δύο κόμβων, έχει τον περιορισμό ότι η πληροφορία που περιέχεται σε κάθε εμφάνιση του ανεστραμμένου ευρετηρίου είναι περιορισμένη. Για παράδειγμα, η κωδικοποίηση αυτή δεν παρέχει πληροφορίες για το όνομα ενός στοιχείου ή για το ποιοι είναι οι πρόγονοί του. Μία τέτοια πληροφορία θα μπορούσε όμως να διευκολύνει πολύ την επεξεργασία κάποιων ερωτημάτων. Για παράδειγμα, αν ξέρουμε ότι η κωδικοποίηση "1.2.3.4" παριστάνει το μονοπάτι "$a/b/c/d$", τότε είναι πολύ εύκολο να προσδιορίσουμε αν το μονοπάτι αυτό ταιριάζει το ερώτημα $//c/d$. Η *εκτεταμένη κωδικοποίηση Dewey* που προτάθηκε από το Lu και άλλους στο [10] είναι ένα πολύ ισχυρό σχήμα κωδικοποίησης με πολύ μεγαλύτερο πληροφοριακό περιεχόμενο, που, όπως δηλώνει και το όνομά της, αποτελεί επέκταση του κλασικού σχήματος κωδικοποίησης Dewey που προτάθηκε από τον Tatarinov και άλλους στο [8]. Φυσικά, η αύξηση αυτή στο πληροφοριακό περιεχόμενο γίνεται με επιδείνωση της υπολογιστικής και χωρικής πολυπλοκότητας της κωδικοποίησης.

Στην κωδικοποίηση Dewey αυτή κάθε στοιχείο του XML εγγράφου αναπαριστάται από ένα διάνυσμα, και συγκεκριμένα:
(α) η ρίζα κωδικοποιείται από την κενή συμβολοσειρά ε (β) για κάθε κόμβο u που δεν είναι ρίζα, έχουμε: $label(u) = label(s).x$, όπου u είναι το x-αδικό παιδί του κόμβου s.

Προφανώς, το σχήμα αυτό προσφέρει αποδοτική αποτίμηση της δομικής σχέσης μεταξύ των στοιχείων του XML εγγράφου. Πράγματι, το στοιχείο u είναι πρόγονος του στοιχείου s, αν και μόνο αν το $label(u)$ είναι πρόθεμα του $label(s)$. Η κωδικοποίηση αυτή παρέχει επίσης μία πολύ σημαντική δυνατότητα: μπορούμε να εξάγουμε ποιοι είναι οι πρόγονοι του εξετάζοντας μόνο το *label* του. Έστω, για παράδειγμα, ότι ο κόμβος u έχει *label* "1.2.3.4", τότε ο γονιός του θα σημαίνεται ως "1.2.3", ο παππούς του ως "1.2" κ.ο.κ.

Η εκτεταμένη κωδικοποίηση Dewey βασίζεται στην αναφερθείσα τεχνική, αλλά την επεκτείνει ώστε να είναι σε θέση να ενσωματώσει πληροφορία για το όνομα των απογόνων ενός στοιχείου του εγγράφου - πέρα από τη δυνατότητα καθορισμού της σήμανσης των κόμβων.

Μία προφανής τεχνική για να επιτύχουμε αυτό θα ήταν να κάνουμε χρήση κάποιων επιπλέον bits, για να παρουσιάσουμε την ακολουθία των ονομάτων των προγόνων, ακολουθούμενους από την πρότυπη κωδικοποίηση Dewey. Ωστόσο, τα πειραματικά αποτελέσματα στο [10] δείχνουν ότι το φορτίο που προκύπτει από το μέγεθος του *label* είναι ιδιαίτερα αυξημένο, καθιστώντας το χωρικό κομμάτι της μεθόδου απαγορευτικά μεγάλο.

Η δυσκολία αυτή ξεπερνιέται, αν κωδικοποιήσουμε τα ονόματα των στοιχείων κατά μήκος ενός μονοπατιού σε μία μοναδική σήμανση Dewey και χρησιμοποιήσουμε στη συνέχεια μία αντίστοιχη μηχανή πεπερασμένων καταστάσεων ($FiniteStateTransducer$, FST), για να αποκωδικοποιήσουμε τα ονόματα των στοιχείων που είναι πρόγονοι ενός δεδομένου κόμβου του XML εγγράφου. Η βασική ιδέα πίσω από το εκτεταμένο σχήμα είναι να χρησιμοποιήσουμε μία συνάρτηση υπολοίπου ($modulo$) για να δημιουργήσουμε μία αντιστοίχιση από έναν ακέραιο στο όνομα του στοιχείου, έτσι ώστε δοθείσης μίας ακολουθίας ακεραίων να μπορούμε να τη μετατρέψουμε σε μία ακολουθία ονομάτων στοιχείων. Χωρίς να μπούμε σε περισσότερες τεχνικές λεπτομέρειες, απλά αναφέρουμε τις ιδιότητες της κωδικοποίησης αυτής:

1. Δοθείσης μίας εκτεταμένης κωδικοποίησης Dewey ενός στοιχείου, μπορούμε να βρούμε τα ονόματα όλων των προγόνων του (μέσω του FST).

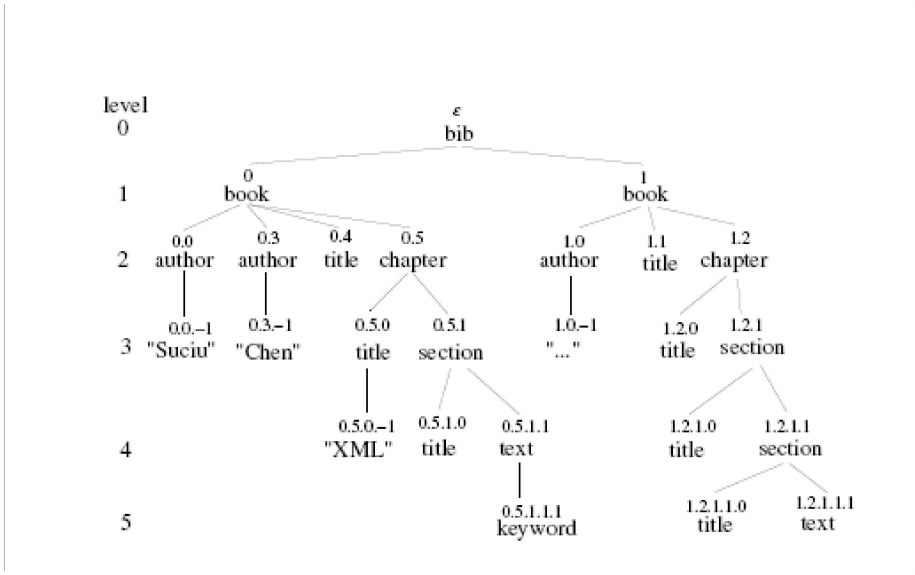
2. Δοθείσης μίας εκτεταμένης κωδικοποίησης Dewey ενός στοιχείου, μπορούμε να βρούμε τις κωδικοποιήσεις όλων των προγόνων του.

3. Δύο στοιχεία έχουν σχέση προγόνου-απογόνου, αν και μόνο η μία από τις αντίστοιχες κωδικοποιήσεις τους είναι πρόθεμα της άλλης.

4. Δύο στοιχεία έχουν σχέση πατέρα-παιδιού, αν και μόνο η μία από τις αντίστοιχες κωδικοποιήσεις τους είναι πρόθεμα της άλλης και τα μήκη τους διαφέρουν κατά τη μονάδα.

5. Το στοιχείο a έπεται (ή προηγείται) του στοιχείου b, αν και μόνο αν το $label(a)$ είναι μεγαλύτερο (ή μικρότερο) σε λεξικογραφική διάταξη από το $label(b)$.

Σημειώνουμε ότι αν και η κωδικοποίηση Dewey έχει σαφώς μεγαλύτερη χωρική πολυπλοκότητα από την κωδικοποίηση θέσης που περιγράφηκε προηγουμένως, έχει ωστόσο την ίδια ασυμπτωτική χωρική πολυπλοκότητα με την εκτεταμένη κωδικοποίηση Dewey, πράγμα που δίνει στην τελευταία ένα σημαντικό πλεονέκτημα για ενδεχόμενη χρήση. Ένα παράδειγμα ενός XML δέντρου με την αντίστοιχη κωδικοποίηση Dewey απεικονίζεται στο Σχήμα 3.2. Τέλος, στην ενότητα 3.9 εξηγείται και αναλύεται ο αλγόριθμος $TJFast$, που κάνει χρήση αυτής της κωδικοποίησης.

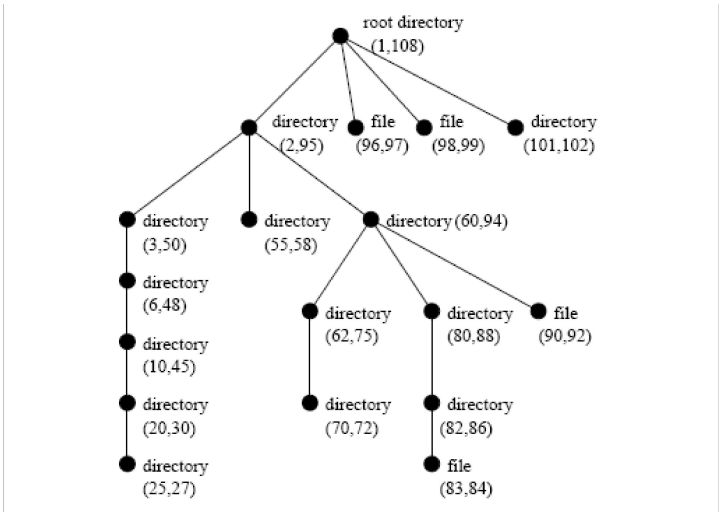


Σχήμα 3.2: Ένα XML δέντρο με την εκτεταμένη κωδικοποίηση Dewey.

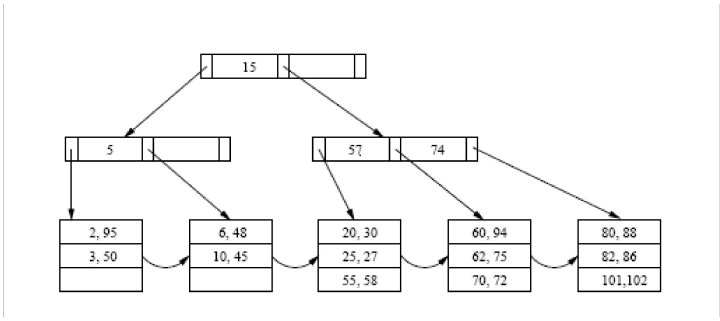
Διάφοροι Τύποι Ευρετηρίων για γρηγορότερη Επεξεργασία Ερωτημάτων

Ας επανέλθουμε στο αρχικό ερώτημα που θέσαμε στην ενότητα 3.1 και στην προσπάθειά μας να επεξεργαστούμε τις διάφορες ερωτήσεις στη βάση δεδομένων. Η κωδικοποίηση θέσης λύνει εν μέρει το πρόβλημα, αφού μας επιστρέφει μόνο τα στοιχεία που έχουν την ίδια σήμανση όπως και οι κόμβοι στην ερώτηση που θέτουμε. Ωστόσο, η βελτίωση αυτή δεν είναι αρκετή σε πολλές εφαρμογές. Για παράδειγμα, θα θέλαμε να είμαστε σε θέση να βρίσκουμε γρήγορα εκείνα τα στοιχεία με δεδομένη σήμανση που *είναι πιθανό* να συμμετέχουν στην απάντηση και να αγνοούμε όσα *εγγυημένα* δε συμμετέχουν.

Τα ανεστραμμένα ευρετήρια δεν αρκούν για να αντιμετωπίσουμε αυτό το πρόβλημα, έτσι πρέπει πολλές φορές να καταφύγουμε σε άλλους τύπους ευρετηρίων. Σε αυτό το σημείο δανειζόμαστε στοιχεία από την περιοχή των σχεσιακών βάσεων δεδομένων και συγκεκριμένα μιας πολύ γνωστής δομής ευρετηρίου, γνωστής κι ως *B+ δένδρο*. Η δομή αυτή αναλύεται στο [25](Κεφάλαιο 12) και εδώ δε θα υπεισέλθουμε στην ανάλυσή του. Ως παράδειγμα μπορούμε να δούμε ότι το B+ δένδρο για τα στοιχεία με σήμανση *directory* του XML δένδρου του Σχήματος 3.3(α΄) παρουσιάζεται στο Σχήμα 3.3(β΄).



(α΄) XML Έγγραφο



(β΄) B+ δένδρο

Σχήμα 3.3: XML Έγγραφο και B+ δένδρο.

Επειδή όμως η αναπαράσταση θέσης χρησιμοποιεί δύο στοιχεία και συγκεκριμένα τα *begin* και *end*, το κλασικό B+ δέντρο πρέπει να επεκταθεί ώστε να μπορεί να γίνει αποδοτικό ευρετήριο για τα στοιχεία ενός ανεστραμμένου ευρετηρίου. Δύο σημαντικοί τύποι τέτοιων ευρετηρίων έχουν προταθεί: τα XR-δένδρα[7] και τα XB-δένδρα[2].

Το XR-δέντρο είναι στην ουσία ένα B+-δέντρο που χτίζεται πάνω στο στοιχείο *start* των διαφόρων κόμβων ενός εγγράφου, όπου αυτοί οι κόμβοι έχουν κάποια δεδομένη σήμανση. Το Σχήμα 3.4(α΄) παρουσιάζει ένα τέτοιο ευρετήριο για τα στοιχεία *directory* του XML εγγράφου του Σχήματος 3.3(α΄). Κάθε εσωτερικός κόμβος του ευρετηρίου σχετίζεται με μία λίστα, η οποία αποθηκεύει τα διαστήματα των στοιχείων εισόδου που μπορούν να καλύψουν οποιοδήποτε κλειδί στον εσωτερικό κόμβο. Προς διευκόλυνση της διαδικασίας αναζήτησης

μέσα στις λίστες, κάθε κλειδί στους εσωτερικούς κόμβους σχετίζεται επίσης με το διάστημα του πρώτου στοιχείου στην αντίστοιχη λίστα που περιέχει το κλειδί, με την προϋπόθεση ότι δεν υπάρχει κλειδί σε κάποιον πρόγονο κόμβο που επίσης να περιέχει το εν λόγω διάστημα. Επιπλέον, κάθε στοιχείο ενός κόμβου φύλλου περιλαμβάνει και μία σημαία που δείχνει αν το στοιχείο αυτό περιέχεται σε κάποια λίστα των εσωτερικών κόμβων.

Πάνω σε αυτή τη δομή μπορεί να οριστεί μία σειρά πράξεων, όπως εύρεση προγόνων, απογόνων ή στοιχείων μέσα στη λίστα ενός κόμβου κατά τρόπο αποδοτικό, όπως περιγράφεται στο [7], ενώ στην ίδια εργασία προτείνεται και ένας αρκετά αποδοτικός αλγόριθμος που κάνει χρήση της δομής αυτής για γρήγορη αποτίμηση ερωτήσεων με πρότυπο δενδρικής μορφής.

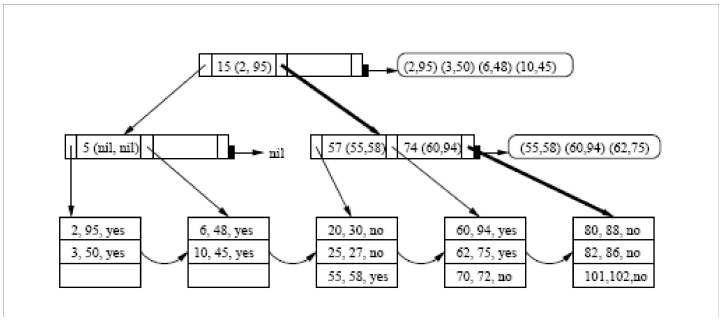
Στα μειονεκτήματά της περιλαμβάνεται αφενός η σχετικά ακριβή συντήρηση των λιστών στην περίπτωση εισαγωγής ή διαγραφής, καθώς και η αδυναμία της να χειριστεί XML δεδομένα με υψηλή αναδρομή, αφού σε εκείνη την περίπτωση οι λίστες μπορούν να έχουν μέγεθος πολλών σελίδων, με αποτέλεσμα μεγαλύτερα κόστη ενημερώσεων και ανάγκη για περισσότερο αποθηκευτικό χώρο.

Από την άλλη, το XB-δένδρο συνδυάζει τα δομικά χαρακτηριστικά τόσο του B+-δέντρου όσο και του R-δένδρου ([25], Ενότητα 23.3.5.3). Το ευρετήριο αυτό πρώτα δεικτοδοτεί τα διαστήματα (*start*, *end*) των διαφόρων στοιχείων σε μία δενδρική δομή, κατά έναν τρόπο που θυμίζει ένα μονοδιάστατο R-δένδρο. Στη συνέχεια, οργανώνει το *start* των διαστημάτων των στοιχείων κατά τον ίδιο τρόπο με το B+-δένδρο.

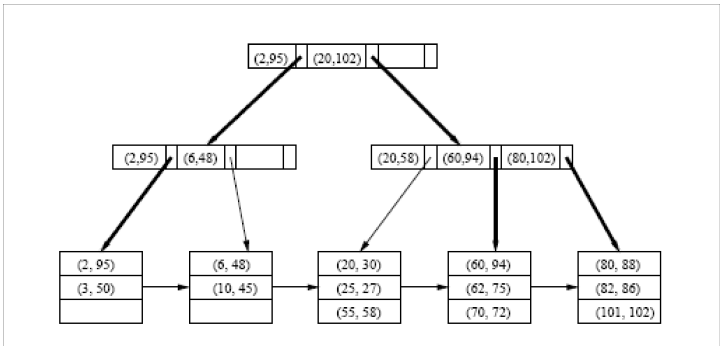
Το Σχήμα 3.3(β′) απεικονίζει το XB-δένδρο που κατασκευάζεται για το XML δένδρο του Σχήματος 3.3(α′). Κάθε εσωτερικός κόμβος διατηρεί ένα σύνολο από περιοχές (διαστήματα) που περιέχουν όλες τις περιοχές των κόμβων-παιδιών. Οι περιοχές των κόμβων στη δομή αυτή μπορούν να επικαλύπτονται μερικώς. Ωστόσο, διαφέρει από το R-δένδρο στο ότι το *start* των διαστημάτων ταξινομείται σε αυστηρά αύξουσα διάταξη. Σε αντίθεση με το XR-δένδρο, το ευρετήριο αυτό δεν περιέχει διπλά αντίγραφα των δεδομένων, πράγμα που οδηγεί σε χαμηλότερο κόστος ενημερώσεων και πιο αποδοτική χωρική χρησιμοποίηση.

Χρήση αυτής της δομής γίνεται στην εργασία [2] και συγκεκριμένα στον αλγόριθμο TwigStack-XB για την αποδοτική αποτίμηση δενδρικών ερωτήσεων. Ο αλγόριθμος αυτός είναι μία βελτίωση του αλγορίθμου TwigStack που προτείνεται στην ίδια εργασία και με τον οποίο ασχολούμαστε επισταμένως στην ενότητα 3.4.2. Προφανώς, η σημασία του έγκειται στη δυνατότητα που προσφέρει να αγνοούμε στοιχεία του ευρετηρίου που δεν μπορούν να συνεισφέρουν στην απάντηση της ερώτησης και στην αποδοτική υλοποίηση κάποιων λειτουργιών, όπως αναζήτηση προγόνων ενός στοιχείου.

Σύντομη Σύγκριση Δομών Ευρετηρίου Η παρούσα διπλωματική δεν ασχολείται με τους αλγόριθμους αποτίμησης ερωτήσεων που κάνουν χρήση των δομών ευρετηρίου που περιγράψαμε προηγουμένως, οπότε δεν υπεισερχόμαστε σε λεπτομέρειες. Αντίθετα, παραπέμπουμε στα αποτελέσματα του Li και άλλων [22], οι οποίοι συγκρίνουν όλα αυτά τα σχήματα με μετρική τον αριθμό των λειτουργιών εισόδου-εξόδου και ρυθμίζοντας τις παραμέτρους της επιλεκτικότητας προγόνων-απογόνων, του βαθμού εμφωλιάσματος και του μεγέθους του buffer της κύριας μνήμης. Λόγω έλλειψης χώρου, δε τα παραθέτουμε εδώ.



(α′) XR δένδρο



(β′) XB δένδρο

Σχήμα 3.4: Προχωρημένοι Τύποι Ευρετηρίων.

Έχοντας αναφερθεί εκτενώς στις σημαντικότερες εισαγωγικές έννοιες που θα μας απασχολήσουν στο κεφάλαιο αυτό, συνεχίζουμε στο υπόλοιπο μέρος του κεφαλαίου με μία ανασκόπηση στις έως τώρα προταθείσες μεθόδους αποτίμησης δενδρικών ερωτήσεων καθώς και ερωτήσεων μονοπατιού, δίνοντας τη μεγαλύτερη έμφαση στους αλγορίθμους $PathStack$ και $TwigStack$, αφού πάνω σε αυτούς βασίζονται οι αλγόριθμοι αποτίμησης ερωτήσεων μονοπατιού με μερική δομή που θα περιγραφούν σε επόμενα καφάλαια.

3.2 Αποτίμηση Ερωτήσεων με Δυαδικές Δομικές Σχέσεις

Αρχικά, το μεγαλύτερο μέρος της έρευνας πάνω στην αποτίμηση XML ερωτήσεων έριχνε το μεγαλύτερο βάρος στην εύρεση ταιριασμάτων δυαδικών δομικών σχέσεων. Πράγματι, μία ερώτηση, όσο σύνθετη κι αν είναι, μπορεί να αποσυντεθεί σε ένα σύνολο από δυαδικές σχέσεις, οπότε η αρχική ερώτηση προτύπου μετασχηματίζεται σε ένα σύνολο από ερωτήσεις δυαδικών σχέσεων, και στη συνέχεια πρέπει: (α) να ταιριάξουμε κάθε μία δομική σχέση πάνω στην XML βάση δεδομένων, και (β) να συγχωνεύσουμε σωστά αυτά τα επιμέρους ταιριάσματα. Η προσέγγιση αυτή κατά την οποία οι δυαδικές σχέσεις αντιμετωπίζονται ως ο θεμέλιος λίθος της ερώτησης προτύπου αναπτύχθηκαν κυρίως στα πλαίσια των εργασιών [12] και [1] και περιγράφονται συνοπτικά παρακάτω.

3.2.1 Προσέγγιση με Σύστημα Διαχείρισης Σχεσιακών Συστημάτων Βάσεων Δεδομένων

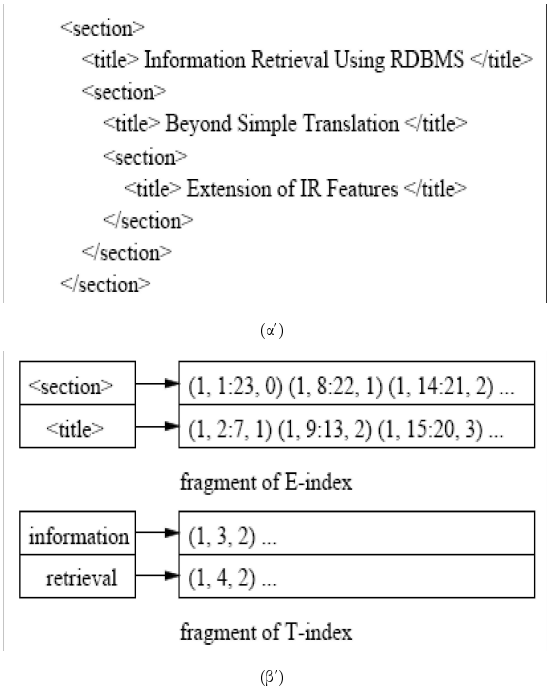
Όπως εξηγήσαμε στην ενότητα 3.1.2, οι λίστες ανεστραμμένου ευρετηρίου αποτελούν έναν πολύ αποδοτικό μηχανισμό για μία σειρά από ερωτήσεις

Οι Zhang και άλλοι στα πλαίσια της εργασίας [12] ασχολούνται με το θέμα του πώς μπορούν να υποστηριχθούν ερωτήσεις δομικών σχέσεων σε συστήματα σχεσιακών βάσεων δεδομένων. Πράγματι, αρχίζοντας με την παραδοχή ότι οι σχεσιακές βάσεις δεδομένων θα μπορούσαν να χρησιμοποιηθούν για την αποθήκευση XML βάσεων δεδομένων, εξετάζουν το πώς είναι καλύτερο να υποστηρίζονται δομικές XML ερωτήσεις πάνω σε τέτοια σχεσιακά συστήματα. Προφανώς, οι λίστες ανεστραμμένου ευρετηρίου της ενότητας 3.1.2 μοιάζουν ως ένας πολύ αποδοτικός μηχανισμός για την αποτίμηση αυτών των ερωτήσεων, παραμένει όμως το ερώτημα του πώς μπορεί να υλοποιηθεί αυτή η τεχνολογία στην πράξη. Στην εργασία αυτή, οι συγγραφείς διακρίνουν δύο περιπτώσεις: (α) την περίπτωση χρήσης μίας IR (*Information Retrieval*) μηχανής που συνδέεται χαλαρά με το RDBMS (*Relational DataBase Management System*), και (β) την περίπτωση που χρησιμοποιούνται οι εγγενείς πίνακες και μηχανισμοί αποτίμησης του RDBMS.

Επεξεργασία Ερωτήσεων με Λίστες Ανεστραμμένου Ευρετηρίου

Στην πρώτη περίπτωση, ουσιαστικά επεκτείνουμε την ιδέα που αναφέρθηκε στην ενότητα 3.1.2, χρησιμοποιώντας ένα T-Ευρετήριο (T-*Index*) για τις λέξεις κειμένου και ένα E-Ευρετήριο (E-*Index*) για τα στοιχεία της XML βάσης δεδομένων, όπως απεικονίζεται στο Σχήμα 3.5 για ένα τυχαίο XML έγγραφο. Σημειώνουμε βεβαίως πως οι *start* και *end* θέσεις της ανεστραμμένης λίστας διαφέρουν στην περίπτωση των στοιχείων, ενώ συμπίπτουν στην περίπτωση των λέξεων κειμένου, που είναι στην πραγματικότητα κόμβοι-φύλλα. Πράγματι, αυτές οι θέσεις προκύπτουν από μία κατά βάθος διάσχιση του δένδρου και αφού κάθε κόμβος που δεν είναι φύλλο διασχίζεται πάντα δύο φορές, μία πριν επισκεφτούμε τα παιδιά του και άλλη μία αφού τα επισκεφτούμε, σε κάθε εσωτερικό κόμβο αποδίδονται δύο θέσεις, ενώ σε κάθε κόμβο φύλλο αρκεί μόνο μία.

Έχοντας κατασκευάσει τις εν λόγω λίστες και εκμεταλλευόμενοι τις ιδιότητες της κωδικοποίησης θέσης που αναφέρθηκαν εκτενώς πιο πάνω (3.1.2), είναι πλέον δυνατόν να υλοποιήσουμε ένα σύστημα αποτίμησης δυαδικών δομικών σχέσεων. Κεντρική ιδέα γύρω από την οποία περιστρέφεται το σύστημα αυτό είναι η συγχώνευση δύο ανεστραμμένων λιστών. Συγκεκριμένα, αν θεωρήσουμε την ανεστραμμένη λίστα ως μία διατεταγμένη σχέση, η συγχώνευση των δύο λιστών μπορεί να θεωρηθεί ως μία πράξη συνδέσμου, όπου οι ιδιότητες που χρησιμοποιούνται για τη συγχώνευση είναι στην ουσία τα κατηγορήματα του συνδέσμου. Εφόσον αυτός ο τύπος συνδέσμου χρησιμοποιείται εκτεταμένα στην περίπτωση αυτή, η αποδοτικότητά του είναι εξέχουσας σημασίας.



Σχήμα 3.5: (α) XML Έγγραφο, και (β) τα T- και E- ευρετήριά του.

Επεξεργασία Ερωτήσεων με Συστήματα Διαχείρισης Σχεσιακών Βάσεων Δεδομένων

Στην περίπτωση αυτή, αρχικά αντιστοιχίζουμε τα ευρετήρια *E-Index* και *T-Index* στις παρακάτω δύο σχέσεις:
ELEMENTS(term, docno, begin, end, level)
TEXTS(term, docno, wordno, level)

Ο πίνακας ELEMENTS αποθηκεύει τις εμφανίσεις των στοιχείων του XML δένδρου, ενώ ο πίνακας TEXTS αποθηκεύει τις εμφανίσεις των λέξεων κειμένου. Κάθε εμφάνιση αποθηκεύεται ως μία γραμμή στον πίνακα. Το Σχήμα 3.6 δείχνει τις μεταφράσεις των συγχωνεύσεων λιστών ανεστραμμένου ευρετηρίου σε SQL πράξεις συνδέσμου. Η συγχώνευση δύο λιστών ανεστραμμένου ευρετηρίου μεταφράζεται σε ένα σύνδεσμο και η ιδιότητα που εφαρμόζεται κατά τη συγχώνευση γίνεται το κατηγόρημα του συνδέσμου. Λόγω έλλειψης χώρου, εδώ παρουσιάζουμε μόνο δομικές σχέσεις μεταξύ στοιχείου και λέξης κειμένου. Μία τέτοια ερώτηση περιλαμβάνει συνδέσμους μεταξύ του πίνακα ELEMENTS και του πίνακα TEXTS. Ομοίως μπορούν να οριστούν και ερωτήσεις μεταξύ στοιχείων, όπου εδώ ο δεύτερος πίνακας TEXTS αντικαθίσταται με έναν επιπλέον πίνακα ELEMENTS.

Σε αυτό το σημείο πρέπει να σημειώσουμε ότι η χρήση SQL ερωτήσεων αυξάνει την εκφραστική δύναμη των ερωτήσεων, αφού τώρα μπορούν αν τεθούν στο σύστημα ερωτήσεις που θα ήταν πολύ δύσκολο ή αδύνατο να επεξεργαστούμε μόνο με λίστες ανεστραμμένου ευρετηρίου. Ένα τέτοιο παράδειγμα είναι οι ερωτήσεις που δεσμεύουν πολλαπλές εκφράσεις μονοπατιού σε μία κοινή μεταβλητή. Τέτοιες ερωτήσεις είναι σχεδόν αδύνατο να εκφραστούν με ερωτήσεις που περιέχουν μόνο δομικές σχέσεις γονέα-παιδιού ή προγόνου-απογόνου ή σχέσεις

```
-- E//"T"
select *
from ELEMENTS e, TEXTS t
where e.term = 'E'
and   t.term = 'T'
and   e.docno = t.docno
and   e.begin < t.wordno
and   t.wordno < e.end
```

(α')

```
-- E / "T"
select *
from  ELEMENTS e, TEXTS t
where e.term = 'E'  and t.term = 'T'
and   e.docno = t.docno
and   e.begin < t.wordno
and   t.wordno < e.end
and   e.level = t.level -1
```

(β')

```
-- E="T"
select *
from ELEMENTS e, TEXTS t
where e.term = 'E'
and   t.term = 'T'
and   e.docno = t.docno
and   t.wordno = e.begin + 1
and   e.end = t.wordno + 1
```

(γ')

```
-- distance ("T1", "T2") <= n
select *
from TEXTS t1, TEXTS t2,
where t1.term = 'T1'
and   t2.term = 'T2'
and   t1.docno = t2.docno
and   t2.wordno > t1.wordno
and   t2.wordno <=  t1.wordno + n
```

(δ')

Σχήμα 3.6: Μεταφράσεις των βασικών Δομικών Σχέσεων σε SQL Ερωτήσεις.

κοντινότητας, διότι περιλαμβάνουν όρους που δεν είναι σταθεροί αλλά εξαρτώνται άμεσα από άλλες συνθήκες αναζήτησης. Για παράδειγμα, έστω ότι επιθυμούμε να κάνουμε την ερώτηση: ¨Βρείτε στοιχεία της βιβλιογραφίας που αναφέρουν την εργασία του Smith. Αυτή η ερώτηση υπονοεί δύο εκφράσεις μονοπατιού: "bib[author/Smith]/key" και "bib/cite", με τον επιπλέον περιορισμό ότι το στοιχείο key έχει το ίδιο περιεχόμενο όπως και το στοιχείο cite. Η ερώτηση είναι δυνατόν να εκφραστεί σχετικά εύκολα σε SQL ([13]).

Ο αλγόριθμος MPMGJN

Στην Ενότητα 3.2.1 επισημάνθηκε ότι ο αλγόριθμος συγχώνευσης των ανεστραμμένων λιστών είναι εξέχουσας σημασίας, αφού λόγω της εκτεταμένης χρήσης του ασκεί πολύ μεγάλη επίδραση στο συνολικό χρόνο της απάντησης. Στο πλαίσιο αυτό, ο Zhang και άλλοι [12] ανέπτυξαν ένα νέο αλγόριθμο για τη διαδικασία της συγχώνευσης, τον *Multi-Predicate Merge Join* (*MPMGJN*). Πρόκειται στην πραγματικότητα για έναν αλγόριθμο που δουλεύει με τον κλασικό τρόπο συγχώνευσης, αλλά διαφέρει από το συνήθη αλγόριθμο συγχώνευσης αλλά και από τον αλγόριθμο συγχώνευσης εμφωλιασμένου βρόχου που χρησιμοποιούνται ευρύτατα στα εμπορικά συστήματα βάσεων δεδομένων στην απόδοση.

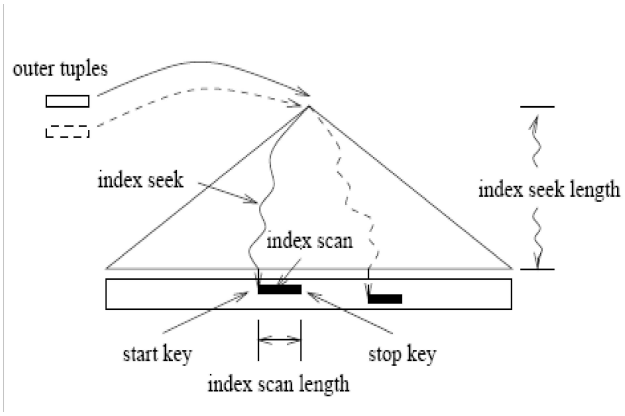
Σε σχέση με το συνήθη αλγόριθμο συγχώνευσης, α MPMGJN χρησιμοποιεί όλες τις στήλες (*docno*, *begin*, *end*, *worno*) για να καθοδηγήσει τη συγχώνευση. Με αυτόν τον τρόπο είναι δυνατόν να αποφευχθούν πολλές περιττές συγκρίσεις που θα γίνονταν με τον απλό αλγόριθμο συγχώνευσης, οπότε ο νέος αλγόριθμος σαφώς υπερέχει της απλής εκδοχής του. Πράγματι, ο απλός αλγόριθμος λειτουργεί σε δύο λογικά βήματα, και συγκεκριμένα: (α) δημιουργεί ζεύγη από ζεύγη γραμμών με το ίδιο *docno*, και (β) τα κατηγορήματα ανισότητας των δομικών ερωτήσεων εφαρμόζονται στα ταιριασθέντα ζεύγη του (α). Προφανώς, ο

MPMGJN συμπυκνώνει τα δύο βήματα σε μόλις ένα.

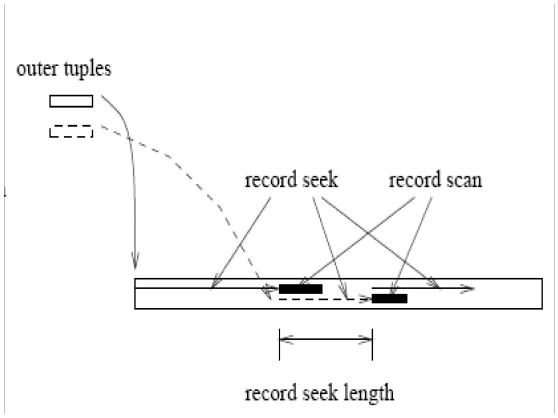
Σε σχέση με τον αλγόριθμο συγχώνευσης εμφωλιασμένου βρόχου, ο νέος αλγόριθμος και πάλι υπερέχει σε απόδοση. Αυτή τη φορά όμως, ο κυριότερος λόγος δεν είναι ο αριθμός των συγκρίσεων (αφού ο MPMGJN μπορεί να κάνει περισσότερες), αλλά λεπτομέρειες χρήσης της κρυφής μνήμης (*cache*). Συγκεκριμένα, ο αλγόριθμος εμφωλιασμένου βρόχου με ευρετήριο (B+-δένδρο) λειτουργεί ως εξής: για κάθε γραμμή του εξωτερικού πίνακα, τα στοιχεία της κωδικοποίησης θέσης χρησιμοποιούνται για την εύρεση μέσω του B+-δένδρου του εσωτερικού πίνακα του πρώτου κόμβου-φύλλου που ταιριάζει με αυτά (*index seek*). Στη συνέχεια, προχωράει σειριακά τους κόμβους φύλλα (*index scan*) μέχρι να φτάσει σε έναν κόμβο φύλλο με στοιχεία κωδικοποίησης θέσης που δεν ταιριάζουν με αυτά της εξωτερικής σχέσης. Τέλος, οι κόμβοι μεταξύ του πρώτου και του τελευταίου του εσωτερικού πίνακα συγκρίνονται με τη γραμμή του εξωτερικού πίνακα, για να εξάγουμε τα τελικά ταιριάσματα και αυτή η διαδικασία ακολουθείται για όλες τις εναπομείνασες εξωτερικές γραμμές. Από την άλλη, ο MPMGJN είναι μία μορφή αλγορίθμου εμφωλιασμένου βρόχου, με τη διαφορά ότι τώρα αναζήτηση της πρώτης εγγραφής δε γίνεται με τη βοήθεια κάποιου ευρετηρίου αλλά απευθείας στις εγγραφές δεδομένων κατά τρόπο σειριακό (*record seek*), χωρίς να ξεκινάμε κάθε φορά από την πρώτη εγγραφή αλλά από την αρχή του προηγούμενου *record scan*. Για καλύτερη κατανόηση όλων των παραπάνω, παραθέτουμε το Σχήμα 3.7, στο οποίο απεικονίζονται οι λειτουργίες seek και scan πάνω στον εσωτερικό πίνακα για την περίπτωση (α) του εμφωλιασμένου βρόχου με ευρετήριο, και (β) του MPMGJN.

Επομένως, για την ίδια ερώτηση αυτό που αλλάζει μεταξύ των δύο αλγορίθμων είναι η αναζήτηση της πρώτης εγγραφής προς ταίριασμα, που στην περίπτωση του εμφωλιασμένου βρόχου γίνεται με τη βοήθεια ευρετηρίου, ενώ στην περίπτωση του MPMGJN γίνεται σειριακά πάνω στις εγγραφές δεδομένων του εσωτερικού πίνακα. Επομένως, αν και ο πρώτος αλγόριθμος κάνει το index scan σε λιγότερα βήματα, κάθε κόμβος του B+-δένδρου που εξετάζεται σε αυτήν την φάση προκαλεί ένα cache miss στην κρυφή μνήμη με τον προηγούμενο κόμβο στο αμέσως πιο πάνω επίπεδο του B+-δένδρου και άρα η εκτέλεση του αλγορίθμου συνεπάγεται πολλά misses στην κρυφή μνήμη. Αντίθετα, ο MPMGJN αν και εξετάζει συνήθως περισσότερες εγγραφές κατά το record scan, έχει το πλεονέκτημα ότι λόγω της σειριακής προσπέλασης οι περισσότερες από τις εγγραφές που εξετάζει προκαλούν cache hit, και τα cache misses είναι σημαντικά λιγότερα. Αν μάλιστα αναλογιστούμε το τεράστιο κόστος των cache misses σε κύκλους επεξεργαστή, ο αλγόριθμος MPMGJN με την πολύ καλύτερη χρησιμοποίηση της ιεραρχίας της μνήμης μοιάζει ως καλύτερη επιλογή από τον αλγόριθμο εμφωλιασμένου βρόχου με ευρετήριο.

Προς επιβεβαίωση όλων των παραπάνω, μία σειρά από πειράματα στα πλαίσια της εργασίας [] έδειξε ότι πράγματι ο αλγόριθμος MPMGJN σε ευρετήρια ανεστραμμένων λιστών μπορεί να ξεπεράσει τουλάχιστον κατά μία τάξη μεγέθους τα συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων που στηρίζονται στον αλγόριθμο εμφωλιασμένου βρόχου με ευρετήριο ή τον απλό αλγόριθμο συγχώνευσης λόγω δύο κυρίως λόγων: (1) του νέου αλγορίθμου συγχώνευσης που χρησιμοποιήθηκε, και (2) της καλύτερης χρησιμοποίησης της ιεραρχίας της μνήμης.



(α′)



(β′)

Σχήμα 3.7: Η λειτουργία πάνω στον εσωτερικότερο πίνακα: (α) στον αλγόριθμο εμφωλιασμένου βρόχου με ευρετήριο, (β) στον αλγόριθμο MPMGJN.

3.2.2 Προσέγγιση με Αποδοτικούς Δομικούς Συνδέσμους

Παρά τη βελτιωμένη απόδοση του MPMGJN σε σχέση με τους συνήθεις αλγορίθμους συνδέσμου των συστημάτων διαχείρισης σχεσιακών βάσεων δεδομένων, ο αλγόριθμος αυτός μπορεί να κάνει πολλούς μη αναγκαίους υπολογισμούς και πράξεις εισόδου/εξόδου, ειδικά στην περίπτωση δομικής σχέσης γονέα-παδιού. Ορμώμενοι από αυτήν την αδυναμία του MPMGJN ο Al-Khalifa και άλλοι ([1]) πρότειναν μία νέα σειρά αλγορίθμων για την αποτίμηση δυαδικών δομικών σχέσεων, συγκεκριμένα την οικογένεια αλγορίθμων *συγχώνευσης δέντρου* ($tree - merge$) και την οικογένεια των αλγορίθμων *συγχώνευσης στοίβας* ($stack - merge$).

Εισαγωγικές Έννοιες

Ας θεωρήσουμε μία δομική σχέση προγόνου-απογόνου (e_1, e_2). Έστω AList = [a_1, a_2, ...] και DList = [d_1, d_2, ...] είναι οι λίστες των κόμβων του XML δένδρου που ταιριάζουν με τα κατηγορήματα e_1 και e_2 αντίστοιχα, όπου κάθε λίστα είναι ταξινομημένη κατά τις τιμές ($docno$, $start$) που προκύπτουν από τη διαδικασία της κωδικοποίησης θέσης. Για παράδειγμα, οι δύο λίστες θα μπορούσαν να ήταν οι λίστες ανεστραμμένου ευρετηρίου, σύμφωνα με όσα αναφέραμε και στην Ενότητα 3.1.2.

```
Algorithm Tree-Merge-Anc(AList, DList)
 1: // Ας υποθέσουμε χάριν απλότητας ότι όλοι οι κόμβοι στις λίστες AList και DList
 2: // έχουν το ίδιο DocNo.
 3: // Η AList είναι η λίστα των πιθανών προγόνων, ταξινομημένων κατά αύξουσα τιμή του StartPos.
 4: // Η DList είναι η λίστα των πιθανών απογόνων, ταξινομημένων κατά αύξουσα τιμή του StartPos.
 5:
 6: begin-desc = DList->firstNode; outputList = NULL;
 7: for (a = AList->firstNode; a != NULL; a = a->nextNode) do
 8:    for (d = begin-desc; (d != NULL && d.StartPos < a.StartPos); d = d->nextNode) do
 9:       // Αποφυγή κόμβων d που σίγουρα δεν ταιριάζουν με τον a.
10:    begin-desc = d;
11:    for (d = begin-desc; (d != NULL && d.EndPos < a.EndPos); d = d->nextNode) do
12:       if ( (a.StartPos < d.StartPos) && (d.EndPos < a.EndPos) && (d.LevelNum == a.LevelNum +
          1) ) then
13:          append(a, d) to OutputList;
```

Σχήμα 3.8: Ο αλγόριθμος Tree-Merge-Anc με έξοδο ταξινομημένη σειρά προγόνου/γονέα.

Δεδομένων των λιστών εισόδου, της AList των πιθανών προγόνων και της Dlist των πιθανών απογόνων, οι αλγόριθμοι κάθε οικογένειας δίδουν ως έξοδο μία λίστα OutputList = $[(a_i, D_j)]$ από τα αποτελέσματα του συνδέσμου των δύο λιστών. Η λίστα εξόδου μπορεί μάλιστα να είναι ταξινομημένη είτε κατά τις τιμές ($DocId$, $a_i.StartPos$, $d_j.StartPos$) είτε κατά τις τιμές ($DocId$, $d_j.StartPos$, $a_i.StartPos$). Και οι δύο δυνατότητες είναι χρήσιμες, και η κατάλληλη εκδοχή μπορεί να εξαρτάται από τη σειρά με την οποία ένας βελτιστοποιητής επιλέγει να συνθέσει τις επιμέρους δυαδικές σχέσεις προκειμένου να παράξει την απάντηση στην αρχική ερώτηση με πρότυπο δενδρικής μορφής.

Tree-Merge Αλγόριθμοι Συνδέσμου

Οι αλγόριθμοι της οικογένειας αυτής είναι μία φυσική επέκταση των παραδοσιακών σχεσιακών αλγορίθμων συνδέσμου-συγχώνευσης (οι οποίοι χρησιμοποιούν συνθήκη ισότητας για το σύνδεσμο) ώστε να αντιμετωπιστούν οι πολλαπλές ανισοτικές συνθήκες που χαρακτηρίζουν τις δομικές σχέσεις προγόνου-απογόνου ή γονέα-παιδιού και οι οποίες βασίζονται στο σχήμα κωδικοποίησης θέσης. Ο αλγόριθμος μάλιστα MPMGJN ανήκει σε αυτήν την κατηγορία.

Η βασική ιδέα είναι να εφαρμόσουμε έναν τροποποιημένο αλγόριθμο συγχώνευσης−συνδέσμου, πιθανώς εφαρμόζοντας πολλαπλές προσπελάσεις του εσωτερικού μέλους του συνδέσμου μέχρι του σημείου που χρειάζεται. Είτε η AList είτε η DList μπορούν να χρησιμοποιηθούν ως εσωτερικά μέλη για το σύνδεσμο. Τα αποτελέσματα θα ταξινομηθούν σε αυτήν την περίπτωση κατά το εξωτερικό μέλος. Στο Σχήμα 3.8 παρουσιάζεται ο αλγόριθμος tree-merge στην περίπτωση όπου το εξωτερικό μέλος του συνδέσμου είναι ο πρόγονος (αυτή η περίπτωση είναι όμοια με τον MPMGJN). Στο Σχήμα 3.9 παρουσιάζεται ο αντίστοιχος αλγόριθμος για την περίπτωση που το εξωτερικό μέλος του συνδέσμου είναι ο απόγονος της δυαδικής σχέσης. Σημειώνουμε ότι εδώ ασχολούμαστε μόνο με την περίπτωση που οι δύο λίστες εισόδου έχουν κόμβους με την ίδια τιμή για το $DocNo$, δηλαδή ανήκουν στο ίδιο XML έγγραφο.

```
Algorithm Tree-Merge-Desc(AList, DList)
 1: // Ας υποθέσουμε χάριν απλότητας ότι όλοι οι κόμβοι στις λίστες AList και DList
 2: // έχουν το ίδιο DocNo.
 3: // Η AList είναι η λίστα των πιθανών προγόνων, ταξινομημένων κατά αύξουσα τιμή του StartPos.
 4: // Η DList είναι η λίστα των πιθανών απογόνων, ταξινομημένων κατά αύξουσα τιμή του StartPos.
 5:
 6: begin-anc = AList->firstNode; outputList = NULL;
 7: for (d = DList->firstNode; d != NULL; d = d->nextNode) do
 8:    for (a = begin-anc; (a != NULL && a.EndPos < d.StartPos); a = a->nextNode) do
 9:       // Αποφυγή κόμβων a που σίγουρα δεν ταιριάζουν με τον d.
10:    begin-anc = a;
11:    for (a = begin-anc; (a != NULL && a.StartPos < d.StartPos); a = a->nextNode) do
12:       if ( (a.StartPos < d.StartPos) && (d.EndPos < a.EndPos) && (d.LevelNum == a.LevelNum +
          1) ) then
13:          append(a, d) to OutputList;
```

Σχήμα 3.9: Ο αλγόριθμος Tree-Merge-Desc με έξοδο ταξινομημένη σειρά απογόνου/παιδιού.

Ανάλυση των Tree-Merge Αλγορίθμων. Οι παραδοσιακοί αλγόριθμοι συνδέσμου-συγχώνευσης που χρησιμοποιούν μόνο μία ισοτική συνθήκη ως κατηγόρημα συνδέσμου αποδεικνύεται ότι έχουν χρονικές και χωρικές πολυπλοκότητες της τάξης $O(|input| + |output|)$ για ταξινομημένες εισόδους, ενώ η έξοδος είναι επίσης ταξινομημένη. Γενικά, δεν είναι σωστό να γενικεύσει αυτήν την πολυπλοκότητα, όταν το κατηγόρημα συνδέσμου περιλαμβάνει πολλές ισοτικές ή ανισοτικές συνθήκες. Παρακάτω εξετάζουμε κάτω από ποιες συνθήκες οι αλγόριθμοι συγχώνευσης δένδρου παρουσιάζουν ασυμπτωτικά βέλτιστη χρονική πολυπλοκότητα.

Το παρακάτω θεώρημα μας δίνει μία πρώτη απάντηση για τον αλγόριθμο Tree-Merge-Anc στην περίπτωση δομικών σχέσεων προγόνου-απογόνου.

Θεώρημα 3.1. Η χωρική και χρονική πολυπλοκότητα του αλγορίθμου *Tree-Merge-Anc* είναι $O(|AList| + |DList| + |OutputList|)$ για τις δομικές σχέσεις προγόνου-απογόνου. □

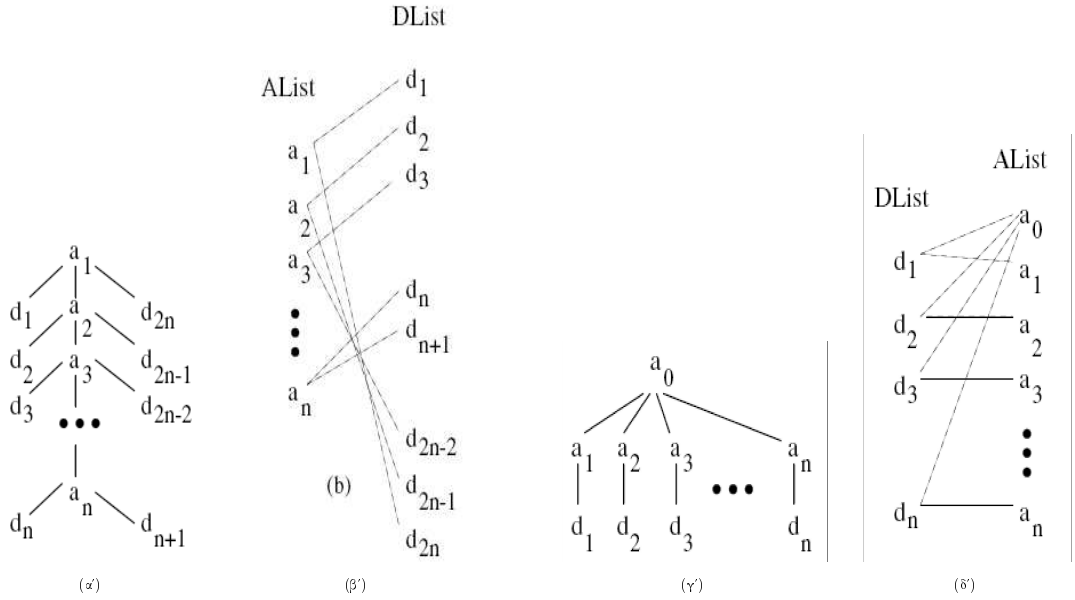
Η ιδέα πίσω από αυτήν την απόδειξη έχει ως εξής. Στην περίπτωση καταρχάς όπου δεν υπάρχουν κόμβοι στην AList που να έχουν κάποια σχέση προγόνου-απογόνου, το μέγεθος της OutputList είναι $O(|AList| + |DList|)$. Ο αλγόριθμος Tree-Merge-Anc κάνει ένα μοναδικό πέρασμα από την AList και το πολύ δύο περάσματα από τη λίστα DList. Άρα το παραπάνω θεώρημα ικανοποιείται. Ακόμη όμως και στην περίπτωση όπου πολλοί κόμβοι στην AList συνδέονται με σχέσεις προγόνου-απογόνου και μπορούμε κατ΄ επέκταση να έχουμε λίστα εξόδου μεγέθους της τάξης $O(|AList| * |DList|)$ μπορεί και πάλι να αποδειχθεί ότι ο αλγόριθμος έχει βέλτιστη χρονική πολυπλοκότητα ίση με $O(|AList| + |DList| + |OutputList|)$.

Παρ΄ όλα αυτά δεν μπορούμε να ισχυριστούμε ότι ο αλγόριθμος είναι βέλτιστος ως προς τις πράξεις εισόδου/εξόδου, αφού μπορεί να έχουμε επαναλαμβανόμενες εισαγωγές και εξαγωγές μπλοκ μνήμης, πράγμα που επιδεινώνει την απόδοση των πράξεων εισόδου/εξόδου.

Από την άλλη, όταν αποτιμούμε μία δυαδική δομική σχέση γονέα-παιδιού, η χρονική πολυπλοκότητα του αλγορίθμου είναι η ίδια με αυτήν του θα παίρναμε αν η ερώτηση ήταν προγόνου-απογόνου στα ίδια XML δεδομένα. Ωστόσο, στην πρώτη περίπτωση αυτή το μέγεθος της λίστας εξόδου OutputList μπορεί να είναι πολύ μικρότερο από τη δεύτερη περίπτωση. Ας

θεωρήσουμε ένα παράδειγμα όπου όλοι οι κόμβοι της AList σχηματίζουν μία αλυσίδα μήκους n, και κάθε κόμβος της AList έχει δύο παιδιά στην DList, ένα σε κάθε πλευρά του αντίστοιχου παιδιού του στην AList (Σχήμα 3.10(α′)). Μπορεί τότε να δειχθεί ότι το μέγεθος της OutputList είναι $O(|AList| + |DList|)$, αλλά η χρονική πολυπλοκότητα του αλγορίθμου είναι $O(\ (|AList| + |DList|)^2\ )$, δηλαδή τετραγωνική και όχι γραμμική ως προς το μέγεθος της εισόδου. Η αποτίμηση απεικονίζεται γραφικά στο Σχήμα 3.10(β′), όπου κάθε κόμβος της AList σχετίζεται με μία υπολίστα της DList, η οποία και πρέπει να εξεταστεί. Σημειώνουμε ότι και η πολυπλοκότητα των πράξεων εισόδου/εξόδου είναι τετραγωνική ως προς το μέγεθος των λιστών εισόδου.

Από την άλλη, δεν υπάρχει ανάλογο του θεωρήματος 3.1 για τον αλγόριθμο Tree-Merge-Desc, αφού τώρα η χρονική πολυπλοκότητα του αλγορίθμου μπορεί να είναι $O(\ (|AList| + |DList| + |OutputList|)^2\ )$ στη χειρότερη περίπτωση. Αυτό συμβαίνει, για παράδειγμα, στο Σχήμα 3.10(γ′), όταν ο πρώτος κόμβος στην AList είναι απόγονος κάθε κόμβου στην DList. Στο παράδειγμα αυτό κάθε κόμβος στην DList έχει μόνο δύο προγόνους στην AList, οπότε το μέγεθος της OutputList είναι $O(|AList| + |DList|)$. Ωστόσο η Alist προσπελαύνεται συνεχώς κι έτσι καταλήγουμε σε χρονική πολυπλοκότητα της τάξης $O(|AList| * |DList|)$. Η αποτίμηση απεικονίζεται γραφικά στο Σχήμα 3.10(δ′), όπου κάθε κόμβος της DList σχετίζεται με μία υπολίστα της AList, η οποία και πρέπει να εξεταστεί.



Σχήμα 3.10: (α), (β) Χειρότερες Περιπτώσεις για τον Tree-Merge-Anc, και (β), (γ) Χειρότερες Περιπτώσεις για τον Tree-Merge-Desc.

Ανάλυση των Stack-Merge Αλγορίθμων. Ο Al-Khalifa και οι υπόλοιποι πρότειναν στα πλαίσια της εργασίας την οικογένεια αλγορίθμων Stack-Tree ως μία καινοτόμα προσέγγιση στην αποτίμηση δυαδικών δομικών σχέσεων κατά τρόπο βέλτιστο ή τουλάχιστον καλύτερο των παραδοσιακών αλγορίθμων. Η κεντρική ιδέα των αλγορίθμων αυτών βασίζεται στην

παρατήρηση πως μία κατά βάθος διάσχιση του XML δένδρου δεδομένων μπορεί να γίνει σε γραμμικό χρόνο χρησιμοποιώντας απλά μία στοίβα με μέγεθος ίσο με το ύψος του δένδρου. Κατά τη διάρκεια της διάσχισης, κάθε σχέση προγόνου-απογόνου στο δέντρο υποδηλώνεται από την εμφάνιση του απόγονου κόμβου στη στοίβα σε θέση υψηλότερη από αυτήν του πρόγονου κόμβου. Φυσικά, σκοπός μας δεν είναι να διασχίσουμε όλη την XML βάση δεδομένων αλλά να περιοριστούμε στους υποψηφίους κόμβους που μας δίδονται στις λίστες εισόδου AList και DList.

Τονίζουμε ότι αυτή η κατηγορία παρουσιάζει μεγάλη σπουδαιότητα στα πλαίσια της παρούσας διπλωματικής, για το λόγο ότι ο αλγόριθμος PathStack, στον οποίο βασίζονται και οι αλγόριθμοι αποτίμησης ερωτήσεων μονοπατιού μερικής δομής που θα παρουσιαστούν αργότερα (Κεφάλαιο 5), βασίζεται σε αυτήν την οικογένεια αλγορίθμων, τόσο στη blocking όσο και non-blocking εκδοχή του.

Αλγόριθμος Stack-Tree-Desc. Ο αλγόριθμος Stack-Tree-Desc για την περίπτωση σχέσεων προγόνου-απογόνου παρουσιάζεται στο σχήμα 3.11. Ο αλγόριθμος αυτός ταξινομεί τη λίστα εξόδου $[(a_i, d_j)]$ κατά τις τιμές των (DocNo, d_j.StartPos, a_i.StartPos).Η βασική ιδέα είναι να πάρουμε τις δύο λίστες εισόδου, AList και DList, ταξινομημένες κατά τις τιμές των (DocNo, StartPos) και να τις συγχωνεύσουμε νοητά. Καθώς η συγχώνευση προχωρά, καθορίζουμε αν υπάρχει σχέση προγόνου-απογόνου μεταξύ της κορυφής της στοίβας και του επόμενου προς συγχώνευση κόμβου, που είναι ο κόμβος με τη μικρότερη τιμή του StartPos. Βασισμένοι σε αυτήν τη σύγκριση χειριζόμαστε τη στοίβα και παράγουμε την έξοδο.

Η στοίβα περιέχει σε κάθε χρονική στιγμή μία ακολουθία των προγόνων κόμβων, όπου κάθε κόμβος στη στοίβα είναι απόγονος του από κάτω του κόμβου. Όταν ένας νέος κόμβος από την AList βρίσκεται να είναι απόγονος της τρέχουσας κορυφής της στοίβας, τότε απλά προστίθεται στη στοίβα. Όταν ένας νέος κόμβος από τη DList βρίσκεται να είναι απόγονος της τρέχουσας κορυφής της στοίβας, γνωρίζουμε αυτόματα ότι θα είναι και απόγονος όλων των υπόλοιπων κόμβων στη στοίβα. Επομένως, παράγονται στην έξοδο τα αποτελέσματα του συνδέσμου αυτού του κόμβου από τη DList με κάθε κόμβο της AList στη στοίβα. Αν όμως ο νέος κόμβος της DList δεν είναι απόγονος της τρέχουσας κορυφής της στοίβας, τότε είναι εγγυημένο ότι κανένας άλλος κόμβος στη DList δεν μπορεί να είναι απόγονος της τρέχουσας κορυφής της στοίβας, οπότε μπορούμε να εξάγουμε την κορυφή της στοίβας και να συνεχίσουμε τον έλεγχο με τη νέα κορυφή της στοίβας. Σημειώνουμε ότι δεν παράγεται έξοδος όταν εξάγονται στοιχεία από τη στοίβα.

Η περίπτωση της δομικής σχέσης γονέα-παιδιού του αλγορίθμου Stack-Tree-Desc αντιμετωπίζεται ακόμα απλούστερα, αφού τώρα ο κόμβος της DList μπορεί να συνδεθεί (αν γίνεται) μόνο με την κορυφή της στοίβας και όχι με τους υπόλοιπους κόμβους της.

Τέλος, μπορεί αρκετά εύκολα να αποδειχθεί το παρακάτω πολύ σημαντικό θεώρημα που δείχνει ότι η χρονική και η χωρική πολυπλοκότητα καθώς και η πολυπλκότητα εισόδου/εξόδου του Stack-Tree-Desc είναι ασυμπτωτικά βέλτιστες.

Θεώρημα 3.2. Η χωρική και χρονική πολυπλοκότητα του αλγορίθμου *Stack-Tree-Desc* είναι $O(|AList| + |DList| + |OutputList|)$ τόσο για τη δομική σχέση προγόνου-απογόνου

```
Algorithm Stack-Tree-Desc(AList, DList)
 1: // Ας υποθέσουμε χάριν απλότητας ότι όλοι οι κόμβοι στις λίστες AList και DList
 2: // έχουν το ίδιο DocNo.
 3: // Η AList είναι η λίστα των πιθανών προγόνων, ταξινομημένων κατά αύξουσα τιμή του StartPos.
 4: // Η DList είναι η λίστα των πιθανών απογόνων, ταξινομημένων κατά αύξουσα τιμή του StartPos.
 5:
 6: a = AList->firstNode; d = DList->firstNode; OutputList = NULL;
 7: while the input lists are not empty or the stack list is not empty do
 8:    if ( (a.StartPos > stack->top.EndPos) && (d.StartPos > stack->top.EndPos) ) then
 9:       // Καιρός να εξάγουμε την κορυφή της στοίβας.
10:       tuple = stack->pop();
11:    else if (a.StartPos < d.StartPos) then
12:       stack->push(a);
13:       a = a->nextNode
14:    else
15:       for (a1 = stack->bottom; a1 != NULL; a1 = a1->up) do
16:          append (a1, d) to OutputList
17:       d = d->nextNode;
```

Σχήμα 3.11: Ο αλγόριθμος Stack-Tree-Desc με έξοδο ταξινομημένη κατά τη σειρά απόγονου.

όσο και για αυτή γονέα-παιδιού. Επιπλέον, ο αλγόριθμος Stack-Tree-Desc δεν μπλοκάρει την εμφάνιση των παραγόμενων αποτελεσμάτων στην έξοδο.

Τέλος, η πολυπλοκότητα εισόδου/εξόδου του αλγορίθμου αυτού είναι $O(\frac{|AList|}{B} + \frac{|DList|}{B} + \frac{|OutputList|}{B})$ τόσο για τη δομική σχέση προγόνου-απογόνου όσο και για αυτή γονέα-παιδιού, όπου B είναι το μέγεθος μπλοκ της μνήμης. □

Αλγόριθμος Stack-Tree-Anc. Εξετάζουμε την περίπτωση όπου η λίστα εξόδου $[(a_i, d_j)]$ πρέπει να ταξινομηθεί κατά τις τιμές των (DocNo, a_i.StartPos, d_j.StartPos).

Το κύριο πρόβλημα που αντιμετωπίζουμε στην περίπτωση αυτή είναι ότι αν ένας κόμβος a από την AList βρεθεί να είναι πρόγονος κάποιου κόμβου d στη DList, τότε κάθε κόμβος a' από την AList που είναι πρόγονος του a (και επομένως βρίσκεται κάτω από τον κόμβο a στη στοίβα) θα είναι επίσης πρόγονος του d. Αφού η StartPos του a' προηγείται της StartPos του a, πρέπει να καθυστερήσουμε την παραγωγή του ζεύγους (a, d) μέχρις ότου η λύση (a', d) έχει παραχθεί. Παραμένει όμως το ενδεχόμενο ένα νέο στοιχείο d' μετά το d στη DList να μπορεί να συνδεθεί με το a', εφόσον το a' είναι στη στοίβα, επομένως δεν μπορούμε να παράξουμε τη λύση (a, d) μέχρι ο πρόγονος κόμβος a' να εξαχθεί από τη στοίβα. Εν τω μεταξύ, μπορούμε να κατασκευάσουμε μεγάλα ενδιάμεσα αποτελέσματα που δεν μπορούν ακόμη να εμφανιστούν στην έξοδο. Η λύση που πρότειναν οι Al-Khalifa είναι ο αλγόριθμος Stack-Tree-Anc στο Σχήμα 3.8.

Η κύρια διαφορά με τον προηγούμενο αλγόριθμο είναι ότι τώρα συσχετίζουμε με κάθε κόμβο στη στοίβα δύο λίστες: η πρώτη, self-list, είναι μία λίστα από αποτελέσματα που έχουν προκύψει από τη σύνδεσμο του κόμβου αυτού με κατάλληλα στοιχεία από τη DList, ενώ η δεύτερη, inherit-list, είναι μία λίστα με τα αποτελέσματα συνδέσμου που περιελάμβαναν κόμβους της AList που ήταν απόγονοι του τρέχοντος κόμβου στη στοίβα. Όπως και πριν,

```
Algorithm Stack-Tree-Anc(AList, DList)
 1: // Ας υποθέσουμε χάριν απλότητας ότι όλοι οι κόμβοι στις λίστες AList και DList
 2: // έχουν το ίδιο DocNo.
 3: // Η AList είναι η λίστα των πιθανών προγόνων, ταξινομημένων κατά αύξουσα τιμή του StartPos.
 4: // Η DList είναι η λίστα των πιθανών απογόνων, ταξινομημένων κατά αύξουσα τιμή του StartPos.
 5:
 6: a = AList->firstNode; d = DList->firstNode; OutputList = NULL;
 7: while the input lists are not empty or the stack list is not empty do
 8:    if ( (a.StartPos > stack->top.EndPos) && (d.StartPos > stack->top.EndPos) ) then
 9:       // Καιρός να εξάγουμε την κορυφή της στοίβας.
10:       tuple = stack->pop();
11:       if ( stack->size == 0 ) then
12:          append tuple.inherit-list to OutputList
13:       else
14:          append tuple.inherit-list to tuple.self-list
15:          append the resulting tuple.self-list to stack->top.inherit-list
16:    else if (a.StartPos < d.StartPos) then
17:       stack->push(a);
18:       a = a->nextNode
19:    else
20:       for (a1 = stack->bottom; a1 != NULL; a1 = a1->up) do
21:          if (a1 == stack->bottom) then
22:             append (a1, d) to OutputList
23:          else
24:             append (a1, d) to the self-list of a1
25:       d = d->nextNode;
```

Σχήμα 3.12: Ο αλγόριθμος Stack-Tree-Anc με έξοδο ταξινομημένη κατά τη σειρά προγόνου.

όταν ένας νέος κόμβος από τη DList είναι απόγονος της τρέχουσας κορυφής της στοίβας, απλά προστίθεται στη στοίβα. Όταν ένας νέος κόμβος από τη DList βρίσκεται πως είναι απόγονος της τρέχουσας κορυφής της στοίβας, προστίθεται στις self-lists των κόμβων στη στοίβα. Αν κανένας κόμβος (από οποιαδήποτε λίστα) δεν είναι απόγονος της κορυφής της στοίβας, τότε είναι εγγυημένο ότι ότι κανένας μελλοντικός κόμβος από τη DList δε θα είναι απόγονος της κορυφής της στοίβας, οπότε μπορούμε να εξάγουμε την κορυφή της στοίβας. Όταν εξάγεται το στοιχείο στον πυθμένα της στοίβας, εμφανίζουμε στην έξοδο πρώτα τη self-list του και μετά την inherit-list του. Για οποιοδήποτε άλλο στοιχείο της στοίβας πλην του πυθμένα της δεν εμφανίζεται κάτι στην έξοδο. Αντίθετα, συνενώνουμε την inherit-list του στη self-list του, και στη συνέχεια συνενώνουμε το προκύπτον αποτέλεσμα στη νέα κορυφή της στοίβας.

Τέλος, αν και δεν είναι τόσο προφανές, μπορεί ωστόσο να αποδειχθεί το παρακάτω πολύ σημαντικό θεώρημα που δείχνει ότι η χρονική και η χωρική πολυπλοκότητα καθώς και η πολυπλοκότητα εισόδου/εξόδου του Stack-Tree-Desc είναι ασυμπτωτικά βέλτιστες. Λόγω έλλειψης χώρου δεν παρατίθεται εδώ η απόδειξη του θεωρήματος, η βασική της ιδέα θα χρησιμοποιηθεί όμως και στην ανάλυση του αλγορίθμου PathStack με blocking στην Ενότητα 3.3.3.

Θεώρημα 3.3. Η χωρική και χρονική πολυπλοκότητα του αλγορίθμου *Stack-Tree-Anc* είναι $O(|AList| + |DList| + |OutputList|)$ τόσο για τη δομική σχέση προγόνου-απογόνου όσο

και για αυτή γονέα-παιδιού. Επιπλέον, ο αλγόριθμος Stack-Tree-Desc δεν μπλοκάρει την εμφάνιση των παραγόμενων αποτελεσμάτων στην έξοδο.

Τέλος, η πολυπλοκότητα εισόδου/εξόδου του αλγορίθμου αυτού είναι $O(\frac{|AList|}{B} + \frac{|DList|}{B} + \frac{|OutputList|}{B})$ τόσο για τη δομική σχέση προγόνου-απογόνου όσο και για αυτή γονέα-παιδιού, όπου B είναι το μέγεθος μπλοκ της μνήμης. □

3.3 Αποτίμηση Ερωτήσεων Μονοπατιού

3.3.1 Ερωτήσεις Μονοπατιού

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 3.13 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*.

(α) $PQ_1 = r//x/s$
(β) $PQ_2 = r//s/s//x/d$

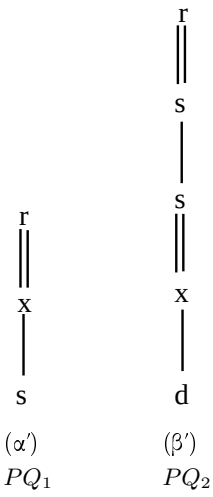
Σχήμα 3.13: Παραδείγματα Ερωτήσεων Μονοπατιού

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 3.13 απεικονίζεται στο Σχήμα 3.14.

3.3.2 Ορολογία

Έστω q η υπό εξέταση ερώτηση σε μορφή μονοπατιού πάνω στο XML δένδρο, καθώς και ο κόμβος ρίζα της εν λόγω ερώτησης. Στον αλγόριθμο στην εργασία [;] και υλοποιήθηκε στα πλαίσια της παρούσας αναφοράς, κάνουμε χρήση των παρακάτω λειτουργιών (με προφανή σημασία) επί των κόμβων της ερώτησης: isLeaf: $Node \rightarrow Bool$, isRoot: $Node \rightarrow Bool$, και parent: $Node \rightarrow Node$, children: $Node \rightarrow Node$.

Συσχετισμένο με κάθε κόμβο q μιας ερώτησης είναι μία ροή T_q. Η ροή περιέχει τις αναπαραστάσεις θέσης των κόμβων της βάσης δεδομένων (επί της οποίας γίνεται η ερώτηση) οι οποίοι ικανοποιούν το κατηγόρημα του κόμβου q (εναλλακτικά στη δική μας απλή περίπτωση: που σημαίνονται από το ίδιο στοιχείο με τον κόμβο q). Η ροή αυτή στην περίπτωσή μας αποκτάται κάνοντας preorder διάσχιση του δέντρου και τοποθετώντας στην ίδια ροή κόμβους

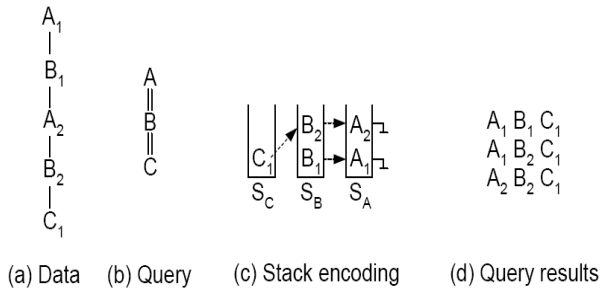


Σχήμα 3.14: Γραφική Αναπαράσταση Ερωτήσεων Μονοπατιού.

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 (*docnumber*, *start*). Οι δυνατές λειτουργίες πάνω στις ροές (με επίσης προφανή σημασία) είναι: eof, advance, next, nextS, και nextE. Οι τελευταίες δύο λειτουργίες επιστρέφουν τις τιμές *start* και *end* της κωδικοποίησης θέσης του επόμενου στοιχείου στη ροή, αντίστοιχα.

Στο βασισμένο σε στοίβα αλγόριθμο που θα αναπτύξουμε στη συνέχεια, πρέπει επίσης να συσχετίσουμε με κάθε κόμβο του ερωτήματος μία στοίβα S_q. Κάθε κόμβος στη στοίβα αποτελείται από το ζεύγος: (αναπαράσταση θέσης από έναν κόμβο από την T_q, δείκτης σε έναν κόμβο στη στοίβα $S_{parent(q)}$). Οι λειτουργίες πάνω στις στοίβες είναι: empty, pop, push, topS, και topE. Οι τελευταίες δύο λειτουργίες επιστρέφουν τις τιμές *start* και *end* της κωδικοποίησης θέσης του πρώτου στοιχείου στη στοίβα, αντίστοιχα. Σε *κάθε* σημείο κατά τη διάρκεια του αλγορίθμου, (α) όλοι οι κόμβοι στη στοίβα S_q (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.

Παράδειγμα 3.2. *Το Σχήμα 3.15 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*



Σχήμα 3.15: Συμπαγής κωδικοποίηση των λύσεων χρησιμοποιώντας στοίβες.

στη στοίβα S_A, όπου δείχνει ο B_1. □

3.3.3 PathStack

Ο αλγόριθμος PathStack παρουσιάζεται σε ψευδοκώδικα στο Σχήμα 3.16, για την ειδική περίπτωση που οι ροές περιλαμβάνουν κόμβους μόνο από ένα XML έγγραφο. Στην περίπτωση που οι ροές περιέχουν κόμβους από περισσότερα XML έγγραφα, ο αλγόριθμος μπορεί εύκολα να επεκταθεί, ώστε να ελέγχει και την ισότητα του αριθμού εγγράφου πριν συνεχίσει με την επεξεργασία των κόμβων στις ροές και στις στοίβες.

Η κύρια ιδέα του συγκεκριμένου 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 5-7). 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 δε γίνεται να ταιριάξουν με δεδομένα από τις ροές οι κόμβοι της ερώτησης μονοπατιού που είναι πάνω από τον τρέχοντα κόμβο στην ερώτηση μονοπατιού) (Σειρά 8). Αν ο τρέχων κόμβος προστέθηκε στη στοίβα $S_{q_{min}}$, 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 9-11).

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

```
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(qmin) ≠ null) then
10:       moveStreamToStack(T_qmin, S_qmin, pointer to top(S_parent(qmin)))
11:       if isLeaf(q_min) then
12:          showSolutions(q_min, 1)
13:          pop(S_qmin)
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 3.16: Αλγόριθμος PathStack.

χρησιμοποίηση αυτού του σχήματος είναι η συμβατότητά του με την έξοδο της διαδικασίας μπλοκαρίσματος των απαντήσεων, η οποία αναπτύσσεται και αναλύεται διεξοδικά στην υποενότητα 3.3.3. Ο αλγόριθμος showSolutions για την περίπτωση που η ερώτηση περιέχει μόνο ακμές προγόνου-απογόνου φαίνεται στο σχήμα 3.17.

Όταν ακμές πατέρα-παιδιού υπάρχουν στην ερώτηση, πρέπει να λάβουμε υπόψη και το στοιχείο *επίπεδο* της κωδικοποίησης θέσης. Ο PathStack δε χρειάζεται να αλλάξει, αλλά χρειάζεται να διασφαλίσουμε ότι κάθε φορά που καλείται η showSolutions, δεν προβάλλει λάθος απαντήσεις. Αυτό μπορεί να γίνει εύκολα
τροποποιώντας την αναδρομική κλήση για να ελέγχει για ακμές πατέρα-παιδιού, στην οποία περίπτωση πρέπει να προκληθεί μόνο μία αναδρομική κλήση, αφού πρώτα επιβεβαιωθεί ότι το στοιχείο *επίπεδο* των δύο κόμβων διαφέρει μόνο κατά ένα. Με αυτόν τον τρόπο αποφεύγουμε και επιπλέον αχρείαστη εργασία, αφού κάνουμε μόνο τη μία κλήση που είναι απαραίτητη.

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.

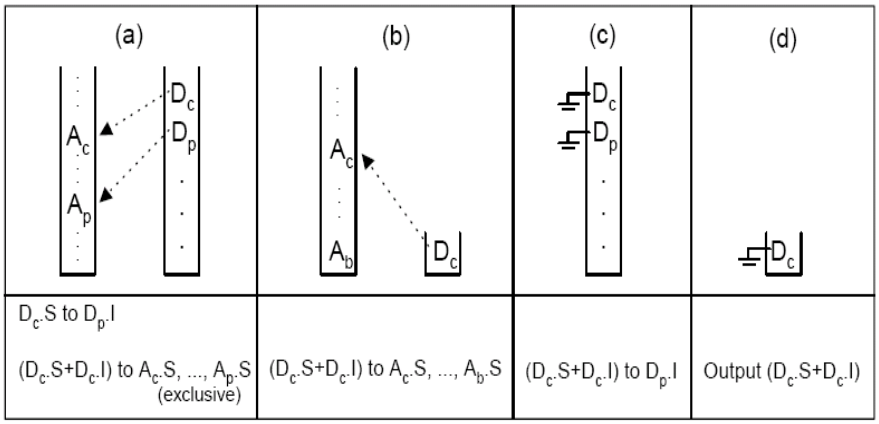
```
Procedure showSolutions(q, index)
 1: // Ας υποθέσουμε χάριν απλότητας ότι οι στοίβες των κόμβων της ερώτησης από τη ρίζα έως
 2: // και το τρέχον φύλλο μπορούν να προσπελαστούν ως S_1,..,S_n.
 3: // Επίσης, έστω ένας global πίνακας outputIndex[1,...,n] δεικτών στα στοιχεία των στοιβών.
 4: // Ο index[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[SN] = 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 3.17: Διαδικασία showSolutions.

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 a' από τη στοίβα S_A που είναι πρόγονος του a πρέπει να είναι και πρόγονος του d. Αφού $a'.start < a.start$, 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 παραμάνει παρόμοια με πριν, αλλά τώρα δε γράφουμε μία πλήρη απάντηση στη λίστα εξόδου αμέσως μόλις την εντοπίσουμε, αλλά συσσωρεύουμε



Σχήμα 3.18: Δυνατές διατάξεις των στοιβών κατά το μπλοκάρισμα αποτελεσμάτων.

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:

(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 3.18(a)). 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).

(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 3.18(b)). 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 3.18(c)). 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 3.18(d)). Πρώτα γράφουμε στην έξοδο τα στοιχεία της self-list και στη συνέχεια τα αποτελέσματα της inherit-list του στοιχείου D_C.

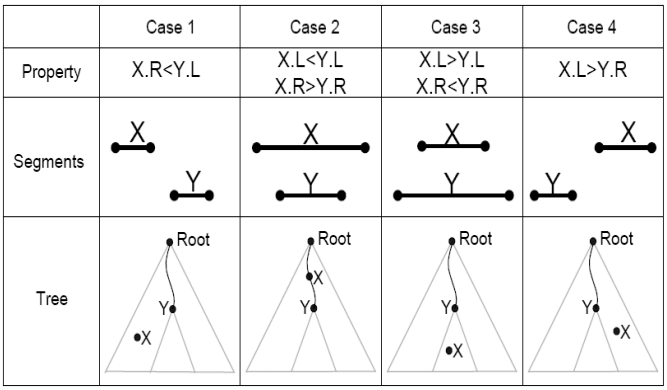
Είναι αρκετά εύκολο να δούμε ότι στις περιπτώσεις (α), (β), και (γ) διατηρούμε όλες τις λύσεις, όταν αναδιατάσσουμε τις συνδεδεμένες λίστες. Επίσης, κάθε φορά που συνενώνουμε μία λίστα με μία άλλη, είναι εγγυημένο ότι κανένα μελλοντικό στοιχείο δε θα συνενωθεί εκτός σειράς. Άρα, στην περίπτωση (δ) γράφουμε 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 τη λίστα self-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 δε χάσαμε κάποια μερική λύση όσο χειριζόμασταν τις λίστες. Επίσης, όταν εξάγουμε το στοιχείο D_C, γνωρίζουμε ότι 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.

3.3.4 Ανάλυση του PathStack

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

Πρόταση 3.1. *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 3.19).* □



Σχήμα 3.19: Περιπτώσεις για τον PathStack.

Λήμμα 3.1. *Έστω ότι για έναν τυχαίο κόμβο q στην υπό εξέταση ερώτηση έχουμε πως getMinSource(q)=q_N. Επίσης, έστω ότι t_{q_N} είναι το επόμενο στοιχείο στη ροή T_{q_N}. Τότε, αφού το στοιχείο t_{q_N} τοποθετηθεί στη στοίβα S_{q_N}, η αλυσίδα των στοιβών από την S_{q_N} ως την S_{q_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 $t_{q_{min}}$ που τοποθετείται στη στοίβα $S_{q_{min}}$, είναι εύκολο να διαπιστώσουμε από το παραπάνω λήμμα και από την επαναληπτική φύση του αλγορίθμου ShowSolutions ότι όλες οι απαντήσεις, στις οποίες το $t_{q_{min}}$ είναι ταίριασμα για τον κόμβο q_{min} της ερώτησης, θα εμφανιστούν. Αυτό μας οδηγεί στο παρακάτω αποτέλεσμα ορθότητας:

Θεώρημα 3.4. *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 3.5. *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: (a)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.*

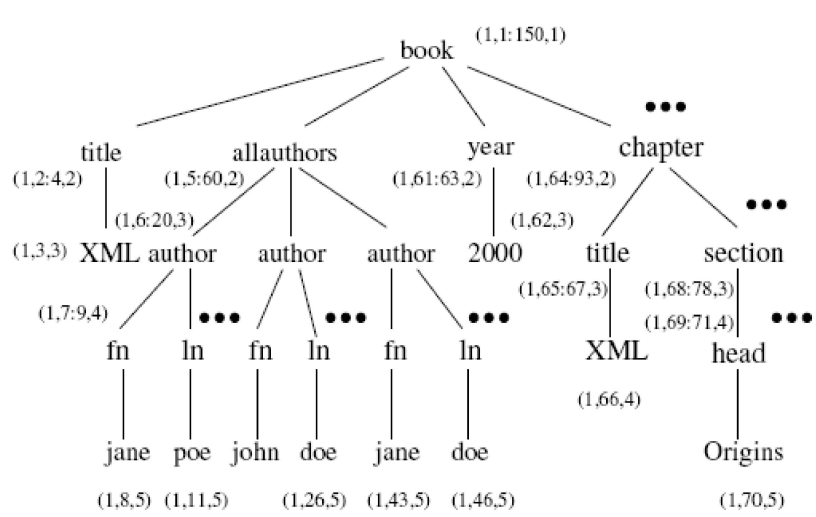
3.4 Ο αλγόριθμος TwigStack

3.4.1 Περιορισμοί Χρήσης του PathStack

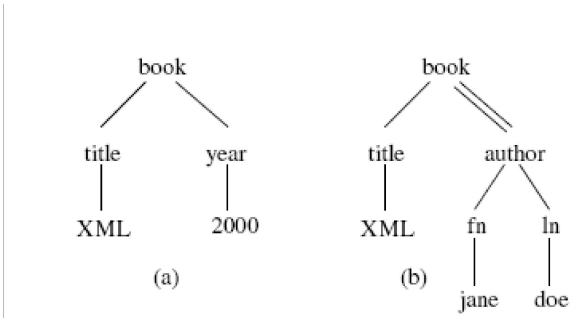
Έχοντας περιγράψει τον αλγόριθμο PathStack ένας πολύ απλός τρόπος για να αποτιμήσουμε μία ερώτηση με πρότυπο δενδρικής μορφής θα ήταν να αποσυνθέσουμε το δενδρικό πρότυπο σε πολλαπλά πρότυπα μορφής μονοπατιού, στη συνέχεια να χρησιμοποιήσουμε τον PathStack για να προσδιορίσουμε την απάντηση σε καθένα από αυτά τα μονοπάτια και, τέλος, να συγχωνεύσουμε με συνένωση όλες τις απαντήσεις ώστε να υπολογίσουμε την τελική απάντηση στο αρχικό ερώτημα. Αυτή η προσέγγιση, όμως, έχει το ίδιο μειονέκτημα θεμελειώδες με αυτό που αντιμετωπίζουν οι τεχνικές που βασίζονται σε δυαδικούς δομικούς συνδέσμους: πολλά ενδιάμεσα αποτελέσματα μπορεί να μην είναι μέρος της τελικής απάντησης. Πράγματι, υπάρχει η ακραία αλλά όχι απίθανη περίπτωση τα αποτελέσματα που προκύπτουν από τα διάφορα μονοπάτια να μην μπορούν να συνδυαστούν με κανένα τρόπο κι έτσι το σύνολο της τελικής απάντησης να είναι κενό, ενώ το σύνολο των ενδιάμεσων αποτελεσμάτων να περιέχει πολύ μεγάλο αριθμό ενδιάμεσων αποτελεσμάτων, στοιχείο που καθιστά την προσέγγιση τελείως αναποτελεσματική.

Παράδειγμα 3.3. *Θεωρούμε την ερώτηση που αντιστοιχεί στο υποδένδρο που έχει ρίζα τον κόμβο author του Σχήματος 3.21. Σε σχέση με την XML βάση δεδομένων του Σχήματος 3.20, τα δύο μονοπάτια της ερώτησης: $author - fn - jane$ και $author - ln - doe$, έχουν δύο λύσεις το καθένα, αλλά η τελική απάντηση στην αρχική ερώτηση έχει μόνο μία απάντηση.* □

Γενικά, αν οι ερωτήσεις μονοπατιού στις οποίες αποσυντίθεται η αρχική ερώτηση έχουν πολλές λύσεις που δε συνεισφέρουν στην τελική απάντηση, τότε ο PathStack δεν είναι βέλτιστος, με την έννοια ότι το συνολικό κόστος υπολογισμού είναι ανάλογο όχι μόνο του μεγέθους των ροών εισόδου και του τελικού αποτελέσματος, αλλά και των ενδιάμεσων αποτελεσμάτων. Ο αλγόριθμος TwigStack μας βοηθά να ξεπεράσουμε το πρόβλημα της μη βελτιστότητας.



Σχήμα 3.20: Δείγμα XML Δέντρου με Κωδικοποίηση Θέσης.



Σχήμα 3.21: Ερωτήσεις με Πρότυπα Δενδρικής Μορφής.

3.4.2 TwigStack

Ο αλγόριθμος TwigStack, ο οποίος υπολογίζει την απάντηση σε ερωτήσεις με πρότυπα δενδρικής μορφής, παρουσιάζεται στο Σχήμα 3.22, για την περίπτωση που οι ροές εισόδου περιέχουν κόμβους από ένα μόνο έγγραφο XML. Αν οι ροές περιέχουν κόμβους από πολλαπλά XML έγγραφα, τότε ο αλγόριθμος μπορεί εύκολα να επεκταθεί για να ελέγχει και ομοιότητα της ταυτότητας εγγράφου, πριν να χειριστεί τους κόμβους στις ροές εισόδου και στις στοίβες.

Ο αλγόριθμος TwigStack λειτουργεί σε δύο φάσεις. Στην πρώτη φάση (γραμμές 2-12), μερικές (αλλά όχι όλες) απαντήσεις στις διάφορες ερωτήσεις μονοπατιού στις οποίες αποσυντίθεται η ερώτηση. Στη δεύτερη φάση (γραμμή 14), αυτές οι λύσεις συγχωνεύονται με συνένωση για τον υπολογισμό της τελικής απάντησης στην αρχική δενδρική ερώτηση.

Η *κύρια διαφορά* μεταξύ του PathStack και της πρώτης φάσης του TwigStack έγκειται στο ότι πριν ένας κόμβος h_q από τη ροή εισόδου T_q τοποθετηθεί στη στοίβα S_q, ο TwigStack (μέσω της κλήσης στη συνάρτηση getNext) έχει εξασφαλίσει δύο πολύ σημαντικά στοιχεία:

(α) ο κόμβος h_q έχει έναν απόγονο h_{q_i} σε κάθε μία από τις ροές T_{q_i}, όπου $q_i \in \text{children}(q)$, και

(β) καθένας από τους κόμβους h_{q_i} ικανοποιεί αναδρομικά την πρώτη ιδιότητα.

```
Algorithm TwigStack(q)
 1: // Φάση 1
 2: while ¬end(q) do
 3:    if (¬isRoot(q_act)) then
 4:       cleanStack(parent(q_act), nextL(q_act))
 5:    if (isRoot(q_act) ∧ ¬empty(S_parent(q_act))) then
 6:       cleanStack(q_act, nextL(q_act))
 7:       moveStreamToStack(T_q_act, S_q_act, pointer to top(S_parent(q_act)))
 8:       if isLeaf(q_act) then
 9:          showSolutionsWithBlocking(S_q_act, 1)
10:          pop(S_q_act)
11:    else
12:       advance(T_q_act)
13: // Φάση 2
14: mergeAllPathSolutions()

Procedure cleanStack(S, actL)
 1: while (¬empty(S) ∧ (topR(S) < actL)) do
 2:    pop(S)
```

Σχήμα 3.22: Αλγόριθμος TwigStack.

Ο αλγόριθμος PathStack από την άλλη δεν ικανοποιεί αυτές τις ιδιότητες και δε χρειάζεται μάλιστα να τις ικανοποιεί για να εγγυηθεί την (ασυμπτωτική) βελτιστότητα για ερωτήσεις με πρότυπα μονοπατιού.

Επομένως, όταν η δενδρική ερώτηση έχει μόνο σχέσεις προγόνου-απογόνου, κάθε μία από τις λύσεις στις διάφορες ερωτήσεις μονοπατιού είναι εξασφαλισμένη να μπορεί να συνδυαστεί και να συγχωνευθεί με τουλάχιστον μία λύση σε καθεμία από τις υπόλοιπες ερωτήσεις μονοπατιού. Αυτό με τη σειρά του εγγυάται ότι καμία ενδιάμεση λύση δεν είναι μεγαλύτερη από την τελική απάντηση στη δενδρική μας ερώτηση.

Η δεύτερη φάση συγχώνευσης του αλγορίθμου είναι γραμμική ως προς το άθροισμα των μεγεθών εισόδου (οι λύσεις στις διάφορες ερωτήσεις μονοπατιού) και εξόδου (η τελική απάντηση), μόνο όταν οι είσοδοι είναι ταξινομημένες σε αύξουσα διάταξη ως προς τα κοινά προθέματα στις ενδιάμεσες λύσεις. Πράγματι, σε αυτήν την περίπτωση μπορούμε να χρησιμοποιήσουμε έναν οποιονδήποτε αλγόριθμο συγχώνευσης από αυτούς που προτείνονται στη βιβλιογραφία, όπως ο merge-join που προτείνεται στο [25], ενότητα 13.5.4. Αυτό όμως προϋποθέτει οι ενδιάμεσες λύσεις στις διάφορες ερωτήσεις μονοπατιού να είναι ταξινομημένες από τη ρίζα προς το φύλλο και κατ΄ επέκταση μπλοκάρισμα. Μπορεί επομένως να χρησιμοποιηθεί μία συνάρτηση όπως η showSolutionsWithBlocking (ενότητα 3.3.3). Σε αντίθετη περίπτωση, θα έπρεπε να προβούμε σε μία ταξινόμηση των ενδιάμεσων αποτελεσμάτων, η οποία έχει ασημπτωτική πολυπλοκότητα $O(nlogn)$, καθιστώντας την προσέγγιση αυτή αναποτελεσματική.

Παράδειγμα 3.4. *Ας θεωρήσουμε εκ νέου την ερώτηση του Παραδείγματος 3.3, που είναι η ερώτηση που αντιστοιχεί στο υποδένδρο με ρίζα τον κόμβο author του δενδρικού προτύπου του Σχήματος 3.21, καθώς και την XML βάση δεδομένων του Σχήματος 3.20. Πριν ο TwigStack*

```
Function getNext(q)
 1: if (isLeaf(q) then
 2:     return q
 3: for q_i in children(q) do
 4:     n_i = getNext(q_i)
 5:     if (n_i ≠ q_i) then
 6:         return n_i
 7: n_min = minarg_{n_i} nextL(T_{n_i})
 8: n_max = maxarg_{n_i} nextL(T_{n_i})
 9: while (nextR(T_q) < nextL(T_{n_max}) do
10:     advance(T_q)
11: if (nextL(T_q) < nextL(T_{n_min}) then
12:     return q
13: else
14:     return n_min
```

Σχήμα 3.23: Αλγόριθμος getNext.

τοποθετήσει ένα στοιχείο με σήμανση author στη στοίβα S_{author}, πρώτα εξασφαλίζει ότι ο κόμβος αυτός έχει: (α) έναν απόγονο κόμβο με σήμανση fn στη ροή εισόδου T_{f_n}, και (β) έναν απόγονο κόμβο με σήμανση ln στη ροή εισόδου T_{l_n} (που με τη σειρά του έχει έναν απόγονο κόμβο doe στη ροή T_{doe}). Επομένως, μόνο το ένα από τα τρία στοιχεία της XML βάσης δεδομένων με τη σήμανση author τοποθετείται στην αντίστοιχη στοίβα. Τα ακόλουθα βήματα του αλγορίθμου εξασφαλίζουν ότι μόνο μία λύση σε καθεμία από τις δύο ερωτήσεις μονοπατιού θα υπολογιστεί, και συγκεκριμένα: $author - fn - jane$ και $author - ln - doe$. Τελικά, η φάση συγχώνευσης υπολογίζει τη μία επιθυμητή απάντηση. □

3.4.3 Ανάλυση του TwigStack

Στην ενότητα αυτή αποδεικνύεται η ορθότητα και βελτιστότητα υπό συνθήκες του αλγορίθμου TwigStack. Οι αποδείξεις στα διάφορα θεωρήματα και λήμματα παραλείπονται στα πλαίσια της διπλωματικής εργασίας. Ο ενδιαφερόμενος αναγνώστης παραπέμπεται στο [2].

Ορισμός 3.5. *Έστω μία ερώτηση με πρότυπο δενδρικής μορφής Q. Για κάθε κόμβο $q \in$ subtreeNodes(Q) ορίζουμε ως* κεφαλή *του q και το συμβολίζουμε με h_q το πρώτο στοιχείο στη ροή εισόδου T_q που συμμετέχει σε μία λύση για την υποερώτηση που έχει ως ρίζα τον κόμβο q. Λέμε ότι ο κόμβος q έχει* ελάχιστη επέκταση απογόνου *αν υπάρχει μία λύση για την υποερώτηση με ρίζα τον κόμβο q που αποτελείται αποκλειστικά από τις κεφαλές των κόμβων subtreeNodes(q).* □

Η Πρόταση 3.1, που βασίζεται στο Σχήμα 3.19, είναι ο θεμέλιος λίθος πάνω στον οποίο χτίζεται το επόμενο λήμμα:

Λήμμα 3.2. *Έστω ότι για έναν τυχαίο κόμβο q στη δενδρική ερώτηση έχουμε ότι getNext(q)= q_N. Τότε ισχύουν οι παρακάτω ιδιότητες:*

1. Ο q_N έχει ελάχιστη επέκταση απογόνου.

2. Για κάθε κόμβο $q' \in subtreeNodes(q_N)$, το πρώτο στοιχείο της ροής εισόδου $T_{q'}$ είναι η κεφαλή $h_{q'}$ του κόμβου q'.

3. Είτε (α) $q = q_N$ ή (β) ο κόμβος $parent(q_N)$ δεν έχει ελάχιστη επέκταση απογόνου, εξαιτίας του q_N (και πιθανώς κι άλλων κόμβων. Με άλλα λόγια, η λύση της υποερώτησης με ρίζα τον κόμβο $p = parent(q_N)$ που χρησιμοποιεί το στοιχείο h_p δε χρησιμοποιεί το στοιχείο h_q για τον κόμβο q αλλά κάποιο άλλο στοιχείο του οποίου η start τιμή στην κωδικοποίηση θέσης είναι μεγαλύτερη από αυτή του h_q. □

Με χρήση του παραπάνω λήμματος μπορεί να αποδειχθεί ότι όταν κάποιος κόμβος q_N επιστρέφεται από τη συνάρτηση getNext, τότε η κεφαλή h_{q_N} θα έχει επέκταση απογόνου σε όλους τους κόμβους στο σύνολο subtreeNodes(q_N). Μπορούμε επίσης να αποδείξουμε ότι οποιοδήποτε στοιχείο στους προγόνους του q_N χρησιμοποιεί το h_{q_N} στην ελάχιστη επέκταση απογόνου του, πρέπει να επιστραφεί από τη getNext *πριν* από το h_{q_N}. Επομένως, μπορούμε να διατηρήσουμε για κάθε κόμβο q στην ερώτηση τα στοιχεία που είναι μέρος μιας λύσης που περιλαμβάνει άλλα στοιχεία στις ροές των κόμβων που ανήκουν στο subtreeNodes(q). Τότε, κάθε φορά που ο κόμβος q_N = getNext(q) είναι ένας κόμβος φύλλο, εμφανίζουμε όλες τις λύσεις που χρησιμοποιούν την κεφαλή h_{q_N}. Το τελευταίο είναι δυνατό, αν διατηρήσουμε μία στοίβα για κάθε κόμβο στη ερώτηση, κατά τρόπο παρόμοιο με όσα κάναμε στον αλγόριθμο PathStack.

Θεώρημα 3.6. *Δοθέντων μιας ερώτησης με πρότυπο δενδρικής μορφής q και μιας XML βάσης δεδομένων D, ο αλγόριθμος TwigStack ορθά επιστρέφει όλες τις απαντήσεις για το q πάνω στη D.* □

Αν και η ορθότητα ισχύει για δενδρικές ερωτήσεις που περιέχουν τόσο σχέσεις προγόνου-απογόνου όσο και πατέρα-παιδιού, μπορούμε ωστόσο να αποδείξουμε τη βελτιστότητα μόνο στην περίπτωση που η ερώτηση περιέχει μόνο ακμές προγόνου-απογόνου. Η βασική ιδέα πίσω από αυτό είναι απλή. Αφού τοποθετούμε στις στοίβες μόνο στοιχεία που έχουν επεκτάσεις προγόνου και απογόνου, είναι εγγυημένο ότι κανένα στοιχείο που δε συμμετέχει στην τελική απάντηση δεν τοποθετείται στις στοίβες. Επομένως, η δεύτερη φάση της συγχώνευσης είναι βέλτιστη κι έχουμε το παρακάτω αποτέλεσμα.

Θεώρημα 3.7. *Έστω μία ερώτηση με πρότυπο δενδρικής μορφής q με n κόμβους που περιέχει μόνο σχέσεις προγόνου-απογόνου και μία XML βάση δεδομένων D. Ο αλγόριθμος TwigStack 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)n φορές το μέγιστο μήκος ενός μονοπατιού από τη ρίζα σε ένα φύλλο στο D.* □

Είναι ιδιαίτερης σημασίας να σημειώσουμε ότι η χρονική πολυπλοκότητα του αλγορίθμου TwigStack σε ερωτήσεις με μόνο σχέσεις προγόνου-απογόνου είναι *ανεξάρτητη των μεγεθών των λύσεων σε οποιδήποτε μονοπάτι από ρίζα προς φύλλο του προτύπου.*

Υποβελτιστότητα για Σχέσεις Πατέρα Παιδιού

Το Θεώρημα 3.7 ισχύει μόνο για την περίπτωση ακμών προγόνου-απογόνου. Στην περίπτωση που η ερώτηση περιέχει επίσης και ακμές πατέρα-παιδιού, τότε ο αλγόριθμος 3.22 δεν είναι πλεόν απαραίτητα βέλτιστος ως προς τις λειτουργίες I/O και ως προς το χρόνο CPU. Πιο συγκεκριμένα, ο αλγόριθμος είναι πιθανό να παραγάγει μία λύση για κάποιο ρίζα−προς−φύλλο μονοπάτι, η οποία δεν ταιριάζει με καμία λύση στην αρχική δενδρική ερώτηση. Αποτέλεσμα είναι τα πολύ μεγάλα ενδιάμεσα αποτελέσματα, τα οποία δε βρίσκουν βήμα στην τελική απάντηση.

Παράδειγμα 3.5. *Ας θεωρήσουμε μία δενδρική ερώτηση με τρεις κόμβους: A, B και C, καθώς και τις ακμές πατέρα-παιδιού (A,B) και (A,C). Έστω επίσης ότι τα XML δεδομένα αποτελούνται από από τον κόμβο A_1, με παιδιά (κατά σειρά) A_2, B_2, C_2, έτσι ώστε ο A_2 έχει παιδιά τους κόμβους B_1, C_1. Οι τρεις ροές εισόδου T_A, T_B και T_C έχουν ως πρώτα στοιχεία τα A_1, B_1 και C_1 αντίστοιχα. Σε αυτήν την περίπτωση,* δεν *μπορούμε να πούμε ότι κάποιος από αυτούς τους κόμβους συμμετέχει σε μία λύση χωρίς να προχωρήσουμε στις υπόλοιπες ροές, και δεν μπορούμε να προχωρήσουμε καμία ροή πριν μάθουμε αν το στοιχείο της συμμετέχει σε κάποια λύση. Ως αποτέλεσμα, η βελτιστότητα δεν είναι πλέον εγγυημένη.* □

3.5 TwigStackList

Αν και ο αλγορίθμος TwigStack δείξαμε στην προηγούμενη ενότητα ότι είναι CPU και I/O βέλτιστος όταν η δενδρική ερώτηση έχει μόνο σχέσεις προγόνου-απογόνου, στην περίπτωση που η ερώτηση έχει επίσης σχέσεις γονέα-παιδιού, τότε ο αλγόριθμος δεν είναι βέλτιστος αφού μπορεί να σχηματιστούν ενδιάμεσα αποτελέσματα για τα μονοπάτια της ερώτησης, τα οποία δεν μπορούν στην συνέχεια να συγχωνευτούν με άλλα αποτελέσματα. Ο αλγόριθμος TwigStack δεν μπορεί να ελέγξει το μέγεθος αυτών των ενδιάμεσων αποτελεσμάτων, έτσι το ποσοστό των άχρηστων ενδιάμεσων αποτελεσμάτων μπορεί να γίνει όσο μεγάλο θέλουμε. Για να αντιμετωπίσουν αποτελεσματικά το πρόβλημα αυτό ο Lu και άλλοι ([9]) πρότειναν μία επέκταση του κλασικού αλγορίθμου TwigStack, ο οποίος είναι σημαντικά πιο αποδοτικός σε δενδρικές ερωτήσεις με σχέσεις γονέα-παιδιού.

Ο νέος αλγόριθμος, TwigStackList, κατορθώνει αυτό με χρήση δύο δομών δεδομένων: μίας *στοίβας* και μίας *λίστας* για κάθε κόμβο της ερώτησης. Μία λίστα συνδεδεμένων στοιβών χρησιμοποιείται όπως και στον TwigStack για τη συμπαγή αναπαράσταση αποτελεσμάτων ατομικών μονοπατιών από τη ρίζα προς τα φύλλα. Τώρα όμως διαβάζουμε εκ των προτέρων μερικά στοιχεία στις ροές δεδομένων της εισόδου και αποθηκεύουμε στη λίστα έναν περιορισμένο αριθμό από αυτά. Ο αριθμός των στοιχείων σε μία τέτοια λίστα φράσσεται από το μήκος του μακρύτερου μονοπατιού στο XML έγγραφο. Τα στοιχεία στις λίστες μας βοηθούν να καθορίσουμε εάν ένα στοιχείο μπορεί όντως να συνεισφέρει σε τελικές απαντήσεις και οδηγούν σε σημαντικά μικρότερο αριθμό άχρηστων αποτελεσμάτων.

Με τη χρήση της δομής της λίστας ο νέος αλγόριθμος εκμεταλλεύεται όχι μόνο τη γνώση για σχέσεις προγόνου-απογόνου μεταξύ κόμβων της ερώτησης αλλά χρησιμοποιεί και την

πληροφορία για το επίπεδο των κόμβων, οδηγώντας έτσι σε μεγαλύτερη αποδοτικότητα. Πράγματι, όπως θα προκύψει και από την ανάλυση του TwigPathList, ο νέος αλγόριθμος μπορεί να εγγυηθεί τη βελτιστότητα των πράξεων εισόου/αξόδου, ακόμα και στην παρουσία σχέσεων πατέρα-παιδιού κάτω από μη διακλαδιζόμενους κόμβους, ενώ ακόμα και όταν υπάρχουν σχέσεις γονέα-παιδιού κάτω από διακλαδιζόμενους κόμβους της ερώτησης, τα ενδιάμεσα αποτελέσματα που εξάγει είναι εγγυημένα ένα υποσύνολο των αποτελεσμάτων που εξάγει ο PathStack.

3.5.1 Ορολογία

Η ορολογία που χρησιμοποιείται στη διατύπωση του αλγορίθμου TwigStackList έχει πολλά κοινά σημεία με την ορολογία του PathStack στην ενότητα 3.3.2, παρουσιάζει όμως και κάποιες επιπλέον προσθήκες. Η συνάρτηση $children(n)$ επιστρέφει όλα τα παιδιά του κόμβου n, και οι $PCRchildren(n)$, $ADRchildren(n)$ επιστρέφουν όλους τους κόμβους παιδιά του κόμβου n με τα οποία έχει σχέση γονέα-παιδιού ή προγόνου-απογόνου αντίστοιχα στο δενδρικό πρότυπο της ερώτησης. Δηλαδή, $PCRchildren(n) \cup ADRchildren(n) = children(n)$.

Σε σύγκριση με τον TwigStack, ο TwiStackList κάνει χρήση και μίας δομής λίστας, πέρα από τη γνωστή δομή στοίβας. Συγκεκριμένα, δοθείσης μιας ερώτησης σχετίζουμε με κάθε κόμβο της μία λίστα L_n και μία στοίβα S_n. Για τη στοίβα S_n ισχύουν όσα αναφέρθηκαν στην ενότητα 3.3.2. Όσο για τη λίστα L_n, δηλώνουμε μία ακέραια μεταβλητή p_n, ως ένα δρομέα που δείχνει σε κάποιο στοιχείο μέσα στη λίστα L_n. Χρησιμοποιούμε τη $L_n.elementAt(p_n)$ για να δηλώσουμε το στοιχείο στο οποίο δείχνει ο p_n. Μπορούμε να προσπελάσουμε τις πληροφορίες θέσης του στοιχείου $L_n.elementAt(p_n)$ με τις συναρτήσεις $L_n.elementAt(p_n).start$, $L_n.elementAt(p_n).end$ και $L_n.elementAt(p_n).level$. Σε *κάθε* σημείο του υπολογισμού τα στοιχεία σε κάθε λίστα L_n είναι αυστηρά εμφωλευμένα από το πρώτο προς το τελευταίο, δηλαδή κάθε στοιχείο είναι απόγονος του στοιχείου που το ακολουθεί στη λίστα. Οι επιτρεπτές πράξεις πάνω στη λίστα είναι οι delete(p_n) και append(e). Η πρώτη διαγράφει το στοιχείο $L_n.elementAt(p_n)$ από τη λίστα, ενώ η δεύτερη παραθέτει το στοιχείο e στο τέλος της L_n.

3.5.2 Αλγόριθμοι

Ο αλγόριθμος TwigStackList λειτουργεί σε δύο φάσεις, όπως και ο TwigStack. Στην πρώτη φάση, καλεί επανειλημμένα τον αλγόριθμο getNext με τη ρίζα του κόμβου ως παράμετρο για να βρει τον επόμενο προς επεξεργασία κόμβο. Σε αυτή τη φάση εξάγουμε τις λύσεις στα ατομικά μονοπάτια της ερώτησης από τη ρίζα προς τα φύλλα. Στη δεύτερη φάση, αυτές οι (μερικές) λύσεις συχγωνεύονται για να υπολογιστεί η απάντηση σε όλο το δενδρικό πρότυπο της ερώτησης.

Αλγόριθμος getNext

Η διαδικασία getNext(n) (Σχήμα 3.24) καλείται από τον κύριο αλγόριθμο του TwigStackList. Επιστρέφει έναν κόμβο n' (πιθανώς $n' = n$) με τις εξής τρεις σημαντικές ιδιότητες: έστω το στοιχείο $e_{n'} = getElement(n')$, τότε

- το $e_{n'}$ έχει έναν απόγονο e_{n_i} σε κάθε ροή δεδομένων T_{n_i}, όπου $n_i \in children(n')$,

- εάν ο n' δεν είναι κόμβος διακλάδωσης στην ερώτηση, το στοιχείο $e_{n'}$ έχει ένα παιδί e_{n_i} στην T_{n_i}, όπου $n_i \in PCRchildren(n')$ (αν υπάρχει)

- εάν ο n' είναι κόμβος διακλάδωσης στην ερώτηση, τότε υπάρχει ένα στοιχείο e_i (με σήμανση n') στο μονοπάτι από το $e_{n'}$ στο $e_{n_{max}}$ που είναι γονέας του e_{n_i}, όπου $n_i \in PCRchildren(n')$ (αν υπάρχουν) και $e_{n_{max}}$ είναι το στοιχείο με τη μέγιστη *start* τιμή για όλα τα $children(n')$.

Οι πρώτες 9 γραμμές του getNext(n) είναι γνωστές από τον TwigStack και δεν πρόκειται να σχολιαστούν. Η γραμμή 10, όμως, αποτελεί ένα πολύ σημαντικό βήμα. Εδώ διαβάζουμε εκ των προτέρων κάποια στοιχεία της T_n ροής και φυλάμε στη λίστα L_n όσα είναι απόγονοι του στοιχείου $C_{n_{max}}$. Όποτε ένα στοιχείο n_i δεν μπορεί να βρει το γονέα του στη λίστα L_n όπου $n_i \in children(n)$, ο αλγόριθμος getNext επιστρέφει τον κόμβο n_i. Αυτή είναι μία πολύ σημαντική διαφορά μεταξύ των TwigStackList και TwigStack, αφού ο τελευταίος θα επέστρεφε σε αυτήν την περίπτωση τον κόμβο n αντί του n_i, πράγμα που μπορεί να οδηγήσει σε πολλά άχρηστα ενδιάμεσα αποτελέσματα. Αντίθετα, ο νέος αλγόριθμος επιστρέφει τον κόμβο n_i που δεν έχει γονέα στη λίστα L_n, αφού ο κόμβος n_i δε συνεισφέρει σε τελικά αποτελέσματα που περιλαμβάνουν στοιχεία από τα υπόλοιπα μέρη των ροών. Έτσι, η κύρια διαφορά μεταξύ των δύο μεθόδων getNext στους δύο αλγόριθμους συνοψίζεται στο εξής: στον TwigStack η getNext(n) επιστρέφει τον κόμβο n' αν το τρέχον στοιχείο $e_{n'}$ στη ροή $T_{n'}$ έχει έναν απόγονο σε κάθε ροή T_{n_i}, για $n_i \in children(n')$, ενώ στον TwigStackList ο επιστρεφόμενος κόμβος πρέπει να ικανοποιεί και τις τρεις ιδιότητες που αναφέρονται πιο πάνω.

Κύριος Αλγόριθμος

Ο κύριος αλγόριθμος του TwigStackList υπάρχει στο [9] και δεν περιγράφεται εδώ λόγω της έντονης ομοιότητάς του με τον κύριο αλγόριθμο του TwigStack. Ο αλγόριθμος αυτός καλεί επανειλημμένα τη συνάρτηση getNext($root$) για να πάρει τον επόμενο προς επεξεργασία κόμβο n. Απομακρύνει μερικές λύσεις που δεν μπορούν να συμμετάσχουν σε πλήρεις λύσεις από τις στοίβες του n και του $parent(n)$. Αν ο n είναι κόμβος φύλλο, τοποθετούμε το getElement(n) στη στοίβα S_n, αλλιώς παράγουμε τη (μερική) λύση που περιλαμβάνει το στοιχείο getElement(n). Οι (μερικές) λύσεις που εξάγονται για τα μονοπάτια του δενδρικού προτύπου πρέπει να είναι ταξινομημένες από τη ρίζα προς τα φύλλα έτσι ώστε η διαδικασία της συγχώνευσης να γίνει αποδοτικά. Για αυτό το λόγο, χρησιμοποιείται η τεχνική του μπλοκαρίσματος που αναλύθηκε στην ενότητα 3.3.3. Η μόνη διαφορά που σημειώνεται από τον TwigStack είναι ότι η νέα έκδοση του κύριου αλγόριθμου χρησιμοποιεί διαφορετικές συναρτήσεις για το καθάρισμα της στοίβας του κόμβου που επιστρέφει η getNext και του γονέα του, έτσι ώστε να διατηρείται η ορθότητα του αλγορίθμου.

```
Algorithm getNext(n)
 1: if (isLeaf(n) then
 2:     return n
 3: for all n_i in children(n) do
 4:     g_i = getNext(n_i)
 5:     if (g_i ≠ n_i) then
 6:         return g_i
 7: n_min = minarg_{n_i} getStart(n_i)
 8: n_max = maxarg_{n_i} getStart(n_i)
 9: while (getEnd(n) < getStart(n_max) do
10:     proceed(n)
11: if (getStart(n) > getStart(n_min) then
12:     return n_min
13: MoveStreamToList(n, n_max)
14: for all node n_i in children(n) do
15:     if there is an element e_i in List L_n such that e_i is the parent of getElement(n_i) then
16:         if n_i is the only child of n) then
17:             move the cursor p_n of list L_n to point to e_i
18:     else
19:         return n_i
20: return n
```

```
Procedure moveStreamToList(n, g)
1: while C_n.start < getStart(g) do
2:     if C_n.end > getEnd(g) then
3:         L_n.append(C_n)
4:     advance(T_n)
```

```
Procedure proceed(n)
1: if empty(L_n) then
2:     advance(T_n)
3: else
4:     L_n.delete(p_n)
5:     p_n = 0 //Move p_n to point the beginning of L_n
```

Σχήμα 3.24: Αλγόριθμος getNext.

3.5.3 Ανάλυση του TwigStackList

Σε αυτήν την ενότητα παρουσιάζουμε τα θεωρητικά αποτελέσματα της εργασίας [9], που φανερώνουν την ανωτερότητα του TwigStackList έναντι του TwigStack. Λόγω περιορισμένου χώρου, αναφέρονται μόνο τα θεωρήματα, χωρίς να μπούμε σε λεπτομέρειες των αποδείξεων.

Θεώρημα 3.8. *Δοθέντων μιας ερώτησης με πρότυπο δενδρικής μορφής q και μιας XML βάσης δεδομένων D, ο αλγόριθμος* TwigStackList *ορθά επιστρέφει όλες τις απαντήσεις για το q πάνω στη D.* □

Θεώρημα 3.9. *Θεωρούμε μία ερώτηση δενδρικού προτύπου με m κόμβους, όπου υπάρχουν μόνο σχέσεις προγόνου-απογόνου κάτω από διακλαδιζόμενους κόμβους, και μία XML βάση δεδομένων D. Ο αλγόριθμος* TwigStackList *έχει χειρότερη πολυπλοκότητα εισόδου/εξόδου*

γραμμική ως προς το άθροισμα των μεγεθών των m λιστών εισόδου και της λίστας εξόδου. □

Αφού όμως το μέγεθος οποιασδήποτε στοίβας και λίστας του TwigStackList είναι ανάλογη του μέγιστου μήκους ενός μονοπατιού από τη ρίζα προς φύλλο του XML δένδρου, έχουμε το παρακάτω αποτέλεσμα σχετικά με τη χωρική πολυπλοκότητα του TwigStackList.

Θεώρημα 3.10. *Θεωρούμε μία ερώτηση δενδρικού προτύπου με m κόμβους, όπου υπάρχουν μόνο σχέσεις προγόνου-απογόνου κάτω από διακλαδιζόμενους κόμβους, και μία XML βάση δεδομένων D. Ο αλγόριθμος* TwigStackList *έχει χειρότερη πολυπλοκότητα εισόδου/εξόδου που είναι ανάλογη m φορές το μέγιστο μήκος ενός μονοπατιού από τη ρίζα προς φύλλο στη D.* □

Εδώ τελειώνει η παρουσίαση του TwigStackList. Σημειώνουμε ότι οι συγγραφείς στο [9] προσφέρουν και πειραματική επαλήθευση των παραπάνω θεωρητικών αποτελεσμάτων.

3.6 Ολιστικοί δενδρικοί σύνδεσμοι σε δεικτοδοτημένα XML έγγραφα

Αν και ο αλγόριθμος TwigStack αποδείχθηκε πως είναι CPU και I/O βέλτιστος σε όρους των μεγεθών εισόδου και εξόδου για ερωτήσεις δενδρικού προτύπου που περιέχουν μόνο δομικές σχέσεις προγόνου-απογόνου και όχι γονέα-παιδιού, το ενδεχόμενο όφελος από την προσπέραση στοιχείων που δε συνεισφέρουν σε μία τελική λύση χρησιμοποιώντας ευρετήρια δεν είχε εξερευνηθεί πλήρως στα πλαίσια της εργασίας αυτής. Παρά ταύτα, το όφελος από την προσπέραση στοιχείων δίχως ταίριασμα μπορεί να είναι τεράστιο όταν οι λίστες εισόδου είναι μεγάλες και μόνο ελάχιστα στοιχεία εμφανίζονται στην έξοδο.

Έχοντας ως σημείο εκκίνησης αυτήν τη διαπίστωση ο Jiang και άλλοι προτείνουν στην εργασία [6] ένα νέο γενικό αλγόριθμο, τον TSGeneric+, ο οποίος κάνοντας χρήση διαθέσιμων ευρετηρίων πάνω στα στοιχεία του XML εγγράφου συμβάλλει στον πιο αποδοτικό υπολογισμό της τελικής λύσης. Ο αλγόριθμος αυτός στην ουσία επεκτείνει τον γενικό αλγόριθμο δενδρικού συνδέσμου TSGeneric, όπως θα περιγρεαφεί στη συνέχεια.

3.6.1 Ο γενικός αλγόριθμος δενδρικού συνδέσμου

Οι έννοιες που χρησιμοποιήθηκαν στον αλγόριθμο TwigStack συνεχίζουν να έχουν ισχύ και εδώ. Έτσι, θα θεωρήσουμε ότι οι αλγόριθμοι συνδέσμου κάνουν χρήση δύο τύπων δομών δεδομένων: δρομέων και στοιβών. Δοθείσης μίας δενδρικής ερώτησης T, σχετίζουμε ένα δρομέα C_q και μία στοίβα S_q σε κάθε κόμβο $q \in T$.

Κάθε δρομέας C_q δείχνει σε κάποιο στοιχείο στην αντίστοιχη ροή δεδομένων του κόμβου q. Επομένως, οι όροι "C_q" ή "στοιχείο C_q" θα αναφέρονται στο στοιχείο στο οποίο ο δρομέας δείχνει. Ο δρομέας μπορεί να μετακινηθεί στο επόμενο στοιχείο (αν υπάρχει) του στοιχείου C_q. Η συνάρτηση C_q→advance() προκαλεί αυτή τη συμπεριφορά. Ομοίως, μπορούμε να προσπελάσουμε τις τιμές των ιδιοτήτων της κωδικοποίησης θέσης του στοιχείου C_q μέσω των

```
Algorithm TSGeneric(root)
 1: while ¬end(root) do
 2:    q = getNext(root)
 3:    if ¬isRoot(q) then
 4:       cleanStack(S_parent(q), C_q)
 5:    if isRoot(q) ∨ ¬empty(S_parent(q)) then
 6:       cleanStack(S_q, C_q)
 7:       if ¬isLeaf(q) then
 8:          push(S_q, C_q, top(S_parent(q))
 9:       outputPathSolutionsWithBlocking(C_q)
10:    C_q→advance()
11: mergeAllPathSolutions()

Procedure cleanStack(S_p, C_p)
 1: pop all elements from S_q that are not ancestors of C_p

Procedure getStart(n)
 1: return the start attribute of getElement(n)

Procedure end(q)
 1: return ∀q_i ∈ ssubtreeNodes(q): isLeaf(q_i) ⇒ end(C_q_i)
```

Σχήμα 3.25: Αλγόριθμος TSGeneric.

C_q→*start*, C_q→*end* και C_q→*level*. Αρχικά, όλοι οι δρομείς δείχνουν στο πρώτο στοιχείο της αντίστοιχης ροής δεδομένων.

Αρχικά, οι στοίβες είναι άδειες. Κατά τη διάρκεια της εκτέλεσης, κάθε στοίβα S_q μπορεί να αποθηκεύσει μερικά στοιχεία πριν το δρομέα C_q και αυτά τα στοιχεία είναι αυστηρά εμφωλευμένα από τον πυθμένα προς την κορυφή της στοίβας, δηλαδή κάθε στοιχείο είναι απόγονος του από κάτω στοιχείου του. Επίσης σχετίζουμε με κάθε στοιχείο e στη στοίβα S_q ένα δείκτη στο χαμηλότερο απόγονο της στοίβας $Sparent(q)$. Με αυτόν τον τρόπο μπορούμε να προσπελάσουμε αποδοτικά όλους τους απόγονους του e στη $Sparent(q)$. Πράγματι, τα αποθηκευμένα στοιχεία στις στοίβες αντιπροσωπεύουν τα μερικά αποτελέσματα που μπορούν να επεκταθούν σε πλήρη αποτελέσματα καθώς ο αλγόριθμος προχωρά. Στη συνέχεια, ορίζουμε μία πολύ σημαντική έννοια, που είναι κλειδί στην κατανόηση του TSGeneric αλγορίθμου.

Ορισμός 3.6. *Λέμε ότι ένας κόμβος q έχει μια* επέκταση λύσης *αν υπάρχει μία λύση για την υποερώτηση με ρίζα τον κόμβο q που αποτελείται αποκλειστικά από τα στοιχεία στους δρομείς όλων των κόμβων της υποερώτησης.* □

Ο γενικός αλγόριθμος TSGeneric

Ο αλγόριθμος TSGeneric εμπνέεται κατά κύριο λόγο από τον TwigStack και παρουσιάζεται στο Σχήμα 3.25. Σημειώνουμε πως, σε αναλογία με τον TwigStack, και ο αλγόριθμος αυτός χρησιμοποιεί τη συνάρτηση getNext, η οποία επιστρέφει κάθε φορά τον επόμενο κόμβο της ερώτησης προς επεξεργασία. Η συνάρτηση αυτή απεικονίζεται στο Σχήμα 3.26.

Η getNext(q) επιστρέφει έναν κόμβο της ερώτησης q_x έτσι ώστε να ικανοποιούνται τα

Algorithm getNext(q)
1: **if** (isLeaf(q) **then**
2: **return** q
3: **for** q_i **in** children(q) **do**
4: n_i = getNext(q_i)
5: **if** ($n_i \neq q_i$) **then**
6: **return** n_i
7: $n_{min} = minarg_{n_i}(C_{n_i} \rightarrow start)$
8: $n_{max} = maxarg_{n_i}(C_{n_i} \rightarrow start)$
9: **while** $C_q \rightarrow end < C_{n_{max}} \rightarrow start$ **do**
10: $C_q \rightarrow advance()$
11: **if** $C_q \rightarrow start < C_{n_{min}} \rightarrow start$ **then**
12: **return** q
13: **else**
14: **return** n_{min}

Σχήμα 3.26: Αλγόριθμος getNext.

παρακάτω τρία κριτήρια: (1) ο q_x έχει μία επέκταση λύσης, (2) εάν ο q_x έχει αδέρφια, τότε $C_{q_x} \rightarrow start < C_{q_j} \rightarrow start$, όπου q_j τα αδέρφια του (προφανώς όλα τα αδέρφια του πρέπει να έχουν επέκταση λύσης, αλλιώς ένας χαμηλότερος κόμβος στην ερώτηση θα είχε επιστραφεί από τις γραμμές 5-6 του αλγορίθμου), και (3) εάν $q_x \neq q$, τότε $C_{q_x} \rightarrow start < C_{parent(q_x)} \rightarrow start$.

Κατά τα άλλα, η λειτουργία των TSGeneric και getNext είναι προφανής μετά την ανάλυση που προηγήθηκε στον TwigStack και για αυτό το λόγο δε θα την επαναλάβουμε εδώ. Ουσιαστικά, πρόκειται για μία πιο γενική/γενικευμένη μορφή του αλγορίθμου αυτού και δεν προσθέτει κάτι παραπάνω.

3.6.2 Διαπροσωπεία δρομέα για τον αλγόριθμο TSGeneric

Προφανώς το ερώτημα που παραμένει είναι το κατά πόσο μπορούμε, εκμεταλλευόμενοι δομές ευρετηρίου, να επιταχύνουμε την επεξεργασία των ερωτήσεων. Η λύση που προτείνουν ο Jiang και οι υπόλοιποι στο [6] είναι η επέκταση της διαπροσωπείας του δρομέα έτσι ώστε να αντανακλά τις νέες ικανότητες προσπέλασης στοιχείων μέσω των ευρετηρίων. Επιπρόσθετα στην advance() ορίζουμε δύο ακόμη μεθόδους:

- η $C_q \rightarrow$fwdBeyond(C_p) προωθεί το δρομέα C_q στο πρώτο στοιχείο e, έτσι ώστε $e.start > C_p.start$
- η $C_q \rightarrow$fwdBToAncestorOf(C_p) προωθεί το δρομέα C_q στον πρώτο πρόγονο του C_p και επιστρέφει TRUE. Αν δεν υπάρχει κανένα τέτοιο στοιχείο, τότε σταματά στο πρώτο στοιχείο e, έτσι ώστε $e.start > C_p.start$, και επιστρέφει FALSE.

Η υλοποίηση των νέων αυτών μεθόδων μπορεί να γίνει μέσω χρήσης ευρετηρίων πάνω στα στοιχεία του XML εγγράφου που αντιστοιχούν στους διάφορους κόμβους της ερώτησης. Για παράδειγμα, ο Jiang και η ομάδα του προτείνουν μία υλοποίηση που βασίζεται στα XR-δέντρα

Algorithm getNextCursor(q)
1: **if** (isLeaf(q) **then**
2: **return** q
3: **for** q_i **in** children(q) **do**
4: n_i = getNext(q_i)
5: **if** ($n_i \neq q_i$) **then**
6: **return** n_i
7: $n_{min} = minarg_{n_i}(C_{n_i} \rightarrow start)$
8: $n_{max} = maxarg_{n_i}(C_{n_i} \rightarrow start)$
9: **if** $C_q \rightarrow$fwdToAncestorOf($C_{n_{max}}$) == TRUE **then**
10: **if** C_q is an ancestor of $C_{n_{min}}$ **then**
11: **return** q
12: **else**
13: **return** n_{min}

Σχήμα 3.27: Αλγόριθμος getNextCursor.

που αναπτύχθηκαν στην ενότητα 3.1.2. Η υλοποίηση αυτή δεν αναφέρεται εδώ για λόγους οικονομίας χώρου.

Με τις επιπρόσθετες αυτές μεθόδους μπορούμε να πάρουμε μία βελτιωμένη έκδοση του αλγόριθμου TSGeneric. Στην πραγματικότητα, αρκεί να αλλάξουμε την υλοποίηση της getNext, που πλέον θα ονομάζεται getNextCursor (Σχήμα 3.27). Οι γραμμές 10-13 του αλγορίθμου αυτού ουσιαστικά αντιστοιχούν στις γραμμές 10-16 του getNext.

Τελικά, η σημασία του βελτιωμένου TSGeneric αλγόριθμου έγκειται αφενός στο ότι είναι γενικός, αφού μπορεί να αντιμετωπίσει όλα τα σύνολα δεδομένων με ευρετήρια, και αφετέρου στο ότι είναι ελαστικός, αφού μπορεί να χρησιμοποιηθεί με οποιοδήποτε ευρετήριο για XML δεδομένα που έχουν αναπαρασταθεί με κωδικοποίηση θέσης.

3.6.3 Ο αλγόριθμος TSGeneric+

Αν και ο αλγόριθμος TSGeneric είναι ικανός να αποφύγει την προσπέλαση ορισμένων αχρείαστων δεδομένων, μπορούμε να εκμεταλλευτούμε ακόμα περισσότερο το πλεονέκτημα του να αποφεύγουμε την προσπέλαση ορισμένων στοιχείων. Αυτό γίνεται κυρίως μέσω ενός αλγόριθμου δομικού συνδέσμου που βασίζεται σε δρομείς, του SJCursor. Ο αλγόριθμος αυτός, ο οποίος αποτιμάται πάνω σε μία ακμή της ερώτησης, βρίσκει το πρώτο ζεύγος στοιχείων με σχέση προγόνου-απογόνου αρχίζοντας από την τρέχουσα θέση των δρομέων των δύο κόμβων που συνδέονται από την ακμή. Ο αλγόριθμος αυτός παρουσιάζεται στο Σχήμα 3.28. Μία ακμή (p, c) ορίζεται ως χαλασμένη εάν τα στοιχεία C_p και C_c δεν έχουν σχέση προγόνου-απογόνου (συνάρτηση isBroken). Ο αλγόριθμος αυτός δουλεύει ως εξής. Αν η ακμή δεν είναι χαλασμένη ή έχουμε φτάσει στο τέλος των στοιβών, τότε τελειώνει τον υπολογισμό. Αλλιώς, αν η τιμή $C_p \rightarrow start$ είναι μικρότερη της $C_c \rightarrow start$, καλούμε την $C_p \rightarrow$fwdToAncestorOf(C_c) ώστε να μετακινήσουμε το δρομέα C_p στο πρώτο πρόγονο στοιχείο του C_c (ή πιο πέρα αν δεν υπάρχει ένα τέτοιο στοιχείο). Αλλιώς, προωθούμε το δρομέα C_c στο πρώτο στοιχείο του οποίου η $start$ τιμή είναι μεγαλύτερη της $C_p \rightarrow start$, επειδή ένα απόγονο στοιχείο πρέπει να

Algorithm SJCursor(p, c)
1: **while** (**not** end(C_p)) **and** (**not** end(C_c)) and isBroken(p, c **do**
2: **if** $C_p \rightarrow start < C_c \rightarrow start$ **then**
3: $C_p \rightarrow$fwdToAncestorOf(C_c)
4: **else**
5: $C_c \rightarrow$fwdBeyond(C_p)

Function isBroken(p, c)
1: **return not** ($C_p \rightarrow start < C_c \rightarrow start$ **and** $C_c \rightarrow start < C_p \rightarrow end$)

Σχήμα 3.28: Αλγόριθμος SJCursor.

έχει μεγαλύτερη *start* τιμή από τον πρόγονό του.

Το παρακάτω λήμμα θέτει το πλαίσιο μέσα στο οποίο μπορεί να χρησιμοποιείται ο αλγόριθμος SJCursor, όπως θα φανεί και παρακάτω.

Λήμμα 3.3. *Έστω ότι η συνάρτηση getNextCursor(root) επιστρέφει τον κόμβο q. Αν η στοίβα S_{q_a} ενός οποιουδήποτε προγόνου κόμβου q_a του κόμβου q είναι άδεια, τότε η τρέχουσα επέκταση λύσης του κόμβου q δε συνεισφέρει σε υπόλοιπες λύσεις και το στοιχείο C_q μπορεί να παραληφθεί.* □

Σύμφωνα με το παραπάνω λήμμα, αν η στοίβα κάποιου κόμβου q είναι άδεια, τότε δεν έχει νόημα η getNextCursor να επιστρέψει ένα κόμβο q' που είναι απόγονος του q στη δενδρική ερώτηση. Ισοδύναμα, εφόσον διαπιστώνουμε ότι η στοίβα ενός κόμβου q είναι άδεια στην αναδρομική κλήση της getNextCursor, δε χρειάζεται να ξανακαλέσουμε τη getNextCursor για τα παιδιά του q. Απεναντίας, πρέπει να ψάξουμε για μία επέκταση λύσης του κόμβου q. Βασισμένοι σε αυτήν την παρατήρηση, ο Jiang και οι άλλοι βελτιώνουν τον αλγόριθμο getNextCursor ενσωματώνοντας μία διαδικασία για εύρεση επέκτασης λύσης. Ο νέος αλγόριθμος ονομάζεται getNextExt και παρουσιάζεται στο Σχήμα 3.29. Οι γραμμές 3-5 είναι ο νεός κώδικας: αν η στοίβα του κόμβου q είναι άδεια, καλείται η διαδικασία LocateExtension (Σχήμα 3.30), η οποία βρίσκει την πρώτη επέκταση λύσης του κόμβου q, και στη συνέχεια απλά επιστρέφεται ο κόμβος q. Καλούμε τον αλγόριθμο TSGeneric ως TSGeneric+, αν καλεί τη getNextExt(root), αντί της getNextCursor(root).

Σε ό,τι αφορά τον αλγόριθμο LocateExtension, κάθε φορά επιλέγει μία χαλασμένη ακμή και τη διορθώνει με τη βοήθεια του αλγορίθμου SJCursor, μέχρις ότου ο κόμβος q αποκτά μία επέκταση λύσης ή ο δρομέας στο υποδέντρο της ερώτησης φτάνει στο τέλος. Σημειώνουμε ότι το επιπλέον κόστος για τον έλεγχο των χαλασμένων ακμών είναι ελάχιστο, αφού όλες οι λειτουργίες πραγματοποιούνται στα στοιχεία των δρομέων των κόμβων της ερώτησης, όπου το CPU κόστος είναι αμελητέο. Τέλος, το ποια ακμή θα επιλεχθεί για έλεγχο πραγματοποιείται με τη βοήθεια ευριστικών συναρτήσεων, οι οποίες προσπαθούν να διαλέξουν την ακμή με τέτοιο τρόπο, ώστε το ταίριασμά της να οδηγήσει τους δρομείς όσο πιο μακριά γίνεται, αποφεύγοντας έτσι να προσπελάσουμε το μεγαλύτερο δυνατό αριθμό στοιχείων.

Για τον αλγόριθμο TSGeneric+ ισχύει το παρακάτω σημαντικό θεώρημα:

Θεώρημα 3.11. *Δοθέντων μιας ερώτησης με πρότυπο δενδρικής μορφής q και μιας XML*

```
Algorithm getNextExt(q)
 1: if (isLeaf(q) then
 2:     return q
 3: if empty(S_q) then
 4:     LocateExtension(q)
 5:     return q
 6: for q_i in children(q) do
 7:     n_i = getNext(q_i)
 8:     if (n_i ≠ q_i) then
 9:         return n_i
10: n_min = minarg_{n_i}(C_{n_i}→start)
11: n_max = maxarg_{n_i}(C_{n_i}→start)
12: if C_q→fwdToAncestorOf(C_{n_max}) == TRUE then
13:     if C_q is an ancestor of C_{n_min} then
14:         return q
15: else
16:     return n_min
```

Σχήμα 3.29: Αλγόριθμος getNextExt.

```
Algorithm LocateExtension(q)
1: while (not end(q)) and (not hasExtension(q)) do
2:     (p, c) = PickBrokenEdge(q)
3:     SJCursor (p, c)
Algorithm hasExtension(q)
1: for each edge (p, c) in the sub query tree q do
2:     if isBroken(p, c) then
3:         return FALSE
4: return TRUE
```

Σχήμα 3.30: Αλγόριθμος LocateExtension.

βάσης δεδομένων D, ο αλγόριθμος TSGeneric+ ορθά επιστρέφει όλες τις απαντήσεις για το q πάνω στη D. □

Επίσης, μπορεί να αποδειχθεί ότι ο TSGeneric+ με ευρετήρια μέσω των διαπροσωπειών των δρομέων είναι τόσο αποδοτικός όσο και ο TwigStack σε όρους χειρότερης CPU και I/O πολυπλοκότητας.

3.7 Αποδοτική Επεξεργασία Δενδρικών XML Ερωτήσεων με OR-κατηγορήματα

Οι αλγόριθμοι TwigStack και TwigStackList που αναφέραμε προηγουμένως μπορούν να χρησιμοποιηθούν για το ταίριασμα ενός δενδρικού προτύπου σε μία XML βάση δεδομένων, όταν όλες οι ακμές των αδελφών κόμβων στο γράφο της ερώτησης συνδέονται μέσω AND λογικής. Σε πολλές όμως πραγματικές εφαρμογές οι ερωτήσεις που τίθενται από τους χρήστες του συστήματος μπορούν να περιέχουν λογικούς OR-τελεστές. Γενικά, θα θέλαμε μία

XQuery έκφραση να περιέχει λογικούς−AND ή −OR τελεστές κατά μία διάταξη που εξυπηρετεί την εκάστοτε εφαρμογή. Καλούμε τέτοιες γενικές ερωτήσεις δενδρικού προτύπου AND/OR-δενδρικές ερωτήσεις και δηλώνουμε ερωτήσεις δίχως λογικούς OR-τελεστές ως AND-δενδρικές ερωτήσεις.

Μία αφελής μέθοδος για τον υπολογισμό μίας τέτοιας ερώτησης θα ήταν η αποσύνθεσή της σε πολλές AND-δενδρικές ερωτήσεις, η επεξεργασία κάθε προκύπτουσας AND-ερώτησης με κάποια υπάρχουσα τεχνική, και τέλος ο συνδυασμός όλων των αποτελεσμάτων. Ωστόσο, αυτή η προσέγγιση έχει το ανυπέρβλητο μειονέκτημα του ενδεχόμενου διαβάσματος των ίδιων δεδομένων, καταλήγοντας έτσι σε υψηλό CPU και κυρίως I/O κόστος. Πράγματι, η διαδικασία της αποσύνθεσης είναι ανάλογη του μετασχηματισμού μίας τυχαίας λογικής έκφρασης στη διαζευκτική κανονική μορφή της. Στη χειρότερη περίπτωση, ο αριθμός των προκύπτοντων AND-ερωτήσεων μπορεί να είναι εκθετικός ως προς το μέγεθος της αρχικής ερώτησης και, αν και κάποια βελτίωση είναι εφικτή, είναι αναπόφευκτο πως ορισμένα δεδομένα θα πρέπει να διαβαστούν πολλαπλές φορές.

Εμπνευσμένοι από τους αποδοτικούς ολιστικούς αλγόριθμους για τον υπολογισμό AND-δενδρικών ερωτήσεων, ο Jiang και άλλοι προτείνουν στο [5] με τη σειρά τους έναν ολιστικό αλγόριθμο για τον αποδοτικό υπολογισμό AND/OR-δενδρικών ερωτήσεων χωρίς να χρησιμοποιείται αποσύνθεση. Στοιχεία της εργασίας τους μελετώνται στην παρούσα ενότητα.

3.7.1 Ορολογία-Αναπαράσταση Δέντρου για AND/OR-Ερωτήσεις

Αναπαριστούμε μία AND/OR-δενδρική ερώτηση με τρεις τύπους κόμβων: κόμβους βήματος θέσης (QNode), λογικούς−AND κόμβους (ANode), και λογικούς−OR κόμβους (ONode):

- QNode: Ένας κόμβος βήματος θέσης αντιπροσωπεύει μία σχέση προγόνου-απογόνου ή γονέα-παιδιού (εναλλακτικά: ένα βήμα θέσης) στην αρχική μας ερώτηση. Ένας τέτοιος κόμβος λοιπόν θα έχει το περιεχόμενο $/tag$ ή $//tag$.

- ANode: Ένας λογικός−AND κόμβος πάντα παίρνει το κείμενο $''and''$ στο δενδρικό πρότυπο της ερώτησης. Συνδέει δύο ή περισσότερα υποδέντρα-παιδιά με AND λογική.

- ONode: Ένας λογικός−OR κόμβος πάντα παίρνει το κείμενο $''or''$ στο δενδρικό πρότυπο της ερώτησης. Συνδέει δύο ή περισσότερα υποδέντρα-παιδιά με OR λογική.

Μία τέτοια ερώτηση μπορεί να περιέχει και πλεονάζοντας κόμβους. Συγκεκριμένα, ορίζουμε δύο κανόνες απλοποίησης ιδιαίτερης σπουδαιότητας (αν και υπάρχουν και άλλοι κανόνες απλοποίησης ερωτήσεων):

- Αν ένας ANode ή ένας ONode n έχει έναν κόμβο παιδί n_i του ίδιου τύπου, τότε μπορούμε να αφαιρέσουμε τον κόμβο n_i και να συνδέσουμε του κόμβους παιδιά του n_i απευθείας στον κόμβο n.

- Αν ένας QNode n έχει ένα ANode παιδί n_i, τότε μπορούμε να αφαιρέσουμε τον κόμβο n_i και να συνδέσουμε του κόμβους παιδιά του n_i απευθείας στον κόμβο n.

Τέλος, εκτός από τις συναρτήσεις children(n) και parent(n) που συζητήσαμε στους προηγούμενους αλγόριθμους, εισάγονται και οι συναρτήσεις Qchildren(n) και Qparent(n). Η πρώτη επιστρέφει το σύνολο όλων των κόμβων τύπου QNode στο υποδέντρο n, που είναι προσπελάσιμοι από τον κόμβο n χωρίς να διασχίσουμε άλλους QNode κόμβους, ενώ η δεύτερη τον κοντινότερο QNode πρόγονο του n.

3.7.2 Ταιριάσματα σε AND/OR-Ερωτήσεις

Στα επόμενα θεωρούμε ότι κάθε QNode κόμβος q_i σχετίζεται με ένα στοιχείο e_i με την ίδια σήμανση. Ορίζουμε:

Ορισμός 3.7 (**edgeTest(q) ή edgeTest(e', e)**). *Έστω q ένας QNode σε μία AND/OR-ερώτηση και q' ο κόμβος Qparent(q). Η λογική συνάρτηση edgeTest(q) ή edgeTest(e', e) αποτιμάται σε αληθής αν και μόνο αν το στοιχείο e' είναι πρόγονος (αντίστοιχα, γονέας) του στοιχείου e, αν ο κόμβος q είναι ένας QNode προγόνου-απογόνου (αντίστοιχα, γονέα-παιδιού).* □

Πριν προχωρήσουμε στον ορισμό του ταιριάσματος για μία AND/OR-δενδρική ερώτηση, ορίζουμε μία ειδική δομή, το OR-κατηγόρημα:

Ορισμός 3.8 (**OR-κατηγόρημα**). *Δοθείσης μίας δενδρικής ερώτησης Q, ένα OR-κατηγόρημα είναι ένα υποδέντρο στο Q τέτοιο ώστε η ρίζα του υποδέντρου να είναι ένας ONode κόμβος και ο κόμβος parent(n) να είναι QNode.* □

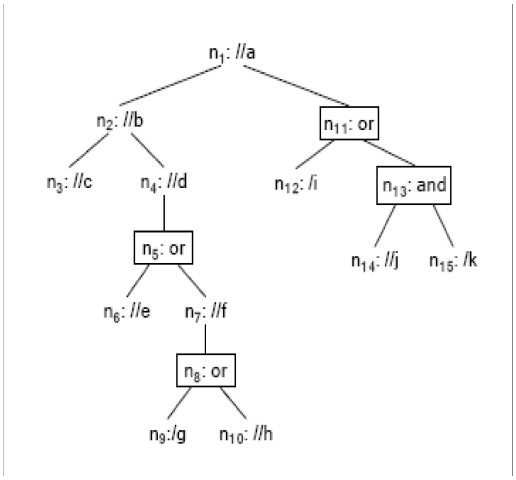
Με βάση το OR-κατηγόρημα μπορούμε τώρα να προχωρήσουμε στον ορισμό του ταιριάσματος για μία AND/OR-δενδρική ερώτηση:

Ορισμός 3.9 (**Ταίριασμα για AND/OR-δενδρική ερώτηση**). *Έστω Q μία δενδρική ερώτηση με N κόμβους n_1, n_2, ..., n_N, όπου ο n_1 είναι η QNode ρίζα. Λέμε ότι το στοιχείο e_1 έχει ένα ταίριασμα για το δέντρο n_1 αν τα παρακάτω ισχύουν για κάθε παιδί-υποδέντρο n_{k_i} του n_1: αν ο n_{k_i} είναι ONode κόμβος, τότε το e_1 ικανοποιεί το OR-κατηγόρημα n_{k_i}, αλλιώς (δηλαδή ο n_{k_i} είναι QNode) η συνάρτηση edgeTest(n_{k_i}) είναι αληθής και το στοιχείο e_{k_i} έχει ένα ταίριασμα για το υποδέντρο n_{k_i}, αν ο κόμβος n_{k_i} δεν είναι κόμβος φύλλο.* □

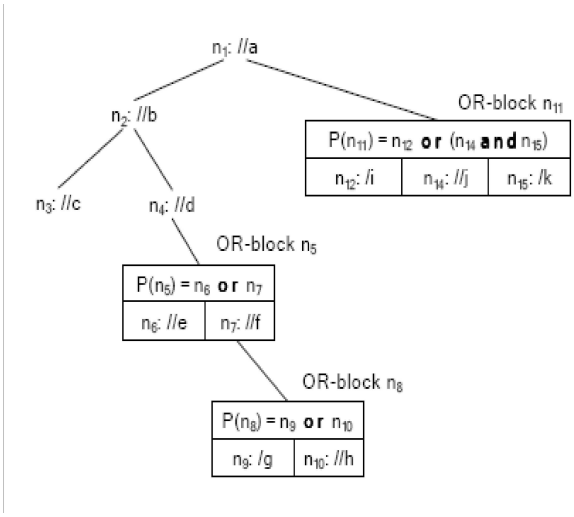
Η πρόκληση στην αποτίμηση ενός OR-κατηγορήματος συνίσταται στο ότι δε χρειάζεται όλες οι συνιστώσες ενός OR-κατηγορήματος να είναι αληθείς, προκειμένου το κατηγόρημα να είναι αληθές. Για να αποτιμήσουμε λοιπόν ένα τέτοιο κατηγόρημα, πρέπει να θεωρήσουμε κατάλληλο λογικό συνδυασμό των edgeTest(q_i) τιμών για όλους τους QNode κόμβους q_i στο OR-κατηγόρημα. Στην κατεύθυνση της αποτίμησης ενός OR-κατηγορήματος βοηθά ο επόμενος ορισμός:

Ορισμός 3.10 (**OR-μπλοκ**). *Δοθείσης μίας δενδρικής ερώτησης Q, ένα OR-μπλοκ είναι ένα υποδέντρο t εμβαπτισμένο στο Q έτσι ώστε η ρίζα του t να είναι ένας QNode κόμβος n, ο κόμβος parent(n) να είναι ένας QNode κόμβος και οι κόμβοι φύλλα του t είναι το σύνολο Qchildren(n).* □

Σημειώνουμε ότι σε αντίθεση με τα OR-κατηγορήματα, τα OR-μπλοκ σε μία δενδρική ερώτηση είναι ξένα μεταξύ τους. Αν θεωρήσουμε ένα OR-μπλοκ ως ένα σύνθετο κόμβο της δενδρικής ερώτησης, μία AND/OR-δενδρική ερώτηση μπορεί να αναπαρασταθεί ως ένα δέντρο με μόνο QNode κόμβους και OR-μπλοκ. Για παράδειγμα, η ερώτηση του Σχήματος 3.31 μπορεί να αναπαρασταθεί ισοδύναμα όπως στο Σχήμα 3.32. Κάθε OR-μπλοκ στο Σχήμα 3.32 αποτελείται από δύο μέρη. Το κάτω μέρος παραθέτει όλους τους QNode κόμβους που ανήκουν στο OR-μπλοκ, ενώ το άνω μέρος είναι μία έκφραση P που δείχνει το λογικό συνδυασμό των QNode κόμβων στο OR-μπλοκ.



Σχήμα 3.31: Παράδειγμα μίας AND/OR-δενδρικής ερώτησης.



Σχήμα 3.32: Αναπαράσταση της ερώτησης του Σχήματος 3.31 χρησιμοποιώντας QNode κόμβους και OR-μπλοκς.

Με βάση το OR-μπλοκ μπορούμε πλέον να δώσουμε τον ορισμό της αποτίμησης ενός OR-κατηγορήματος:

Ορισμός 3.11 (Αποτίμηση OR-κατηγορήματος). *Έστω πως ο ONode κόμβος n*

Algorithm ORBlockMax(n)
1: **if** n is a QNode **then**
2: **return** n
3: **else**
4: **for each** n_i **in** children(n) **do**
5: q_i = ORBlockMax(n_i)
6: **if** n is an ANode **then**
7: **return** $argmax_{q_i} e_i.start$, for q_i initialized at line 5
8: **else**
9: **return** $argmin_{q_i} e_i.start$, for q_i initialized at line 5

Σχήμα 3.33: Αλγόριθμος ORBlockMax.

είναι η ρίζα ενός OR-κατηγορήματος που συνδέεται στον QNode κόμβο q, του οποίου το σχετιζόμενο στοιχείο είναι το e. Λέμε ότι το στοιχείο e ικανοποιεί το OR-κατηγόρημα n αν το $P(n)$ είναι αληθές αντικαθιστώντας κάθε QNode κόμβο n_i στο $P(n)$ με μία λογική συνάρτηση ως εξής: αν ο n_i είναι κόμβος φύλλο, αντικαθιστούμε το n_i με το edgeTest(n_i), αλλιώς αντικαθιστούμε το n_i με τη λογική τιμή (edgeTest(n_i) $\wedge$ το e_i έχει ένα ταίριασμα για το υποδέντρο n_i). □

3.7.3 Ο λογικός–μέγιστος QNode κόμβος σε ένα OR-μπλοκ

Θεωρούμε έναν QNode κόμβο q σε μία δενδρική ερώτηση και ένα OR-μπλοκ n συνδεδεμένο στον κόμβο q. Έστω ότι το OR-μπλοκ n περιέχει k QNode κόμβους q_1, ..., q_k και έστω πως ανάμεσα σε αυτούς τους k κόμβους, το στοιχείο e_{min} του κόμβου q_{min} έχει τη μικρότερη *start* τιμή της κωδικοποίησης θέσης. Μας ενδιαφέρει να βρούμε τη *μεγαλύτερη δυνατή* τιμή κατωφλίου u ($\geq e_{min}.start$) έτσι ώστε αν $e.end < u$ τότε ο u δεν μπορεί να ικανοποιήσει το OR-μπλοκ όσο κι αν προχωρήσουν οι δρομείς στις ροές στοιχείων που αντιστοιχούν στους κόμβους της ερώτησης. Η κύρια εφαρμογή της εύρεσης μίας τέτοιας τιμής κατωφλίου είναι ότι μας επιτρέπει να παραλείψουμε διάφορα στοιχεία από τις ροές των κόμβων T_q στον αλγόριθμο που θα παρουσιαστεί στη συνέχεια. Μπορεί να δειχθεί ότι ο αλγόριθμος ORBlockMax που παρουσιάζεται στο Σχήμα 3.33 επιστρέφει έναν QNode κόμβο q_{max} στο OR-μπλοκ n έτσι ώστε το $e_{max}.start$ να είναι η επιθυμητή τιμή κατωφλίου u.

Η εξήγηση πίσω από τον αλγόριθμο αυτό είναι πως για QNode κόμβους που συνδέονται με AND λογική, διαλέγουμε τον QNode κόμβο με τη *μέγιστη start* τιμή, ενώ για QNode κόμβους που συνδέονται με OR λογική, διαλέγουμε τον QNode κόμβο με τη *μικρότερη start* τιμή.

3.7.4 Αλγόριθμος GTwigMerge

Σε αυτήν την ενότητα παρουσιάζεται ο αλγόριθμος GTwigMerge, για να βρίσκουμε όλα τα ταιριάσματα μίας AND/OR-δενδρικής ερώτησης πάνω σε ένα XML έγγραφο. Πρέπει να σημειώσουμε ότι ο αλγόριθμος αυτός μοιράζεται πολλές ομοιότητες με τον TSGeneric (Σχήμα 3.25 της ενότητας 3.6.1, κάνει όμως σημαντικές επεκτάσεις για να μπορεί να χειριστεί και

Algorithm Κύριος αλγόριθμος του QTwigMerge

```
 1: while ¬end(root) do
 2:    q = GetQNode(root)
 3:    if ¬isRoot(q) then
 4:       cleanStack(S_Qparent(q), C_q)
 5:    cleanStack(S_q, C_q)
 6:    if isRoot(q) ∧ ¬empty(S_Qparent(q)) then
 7:       if ¬isLeaf(q) then
 8:          push(S_q, C_q, isRoot?-1 : top(S_Qparent(q)))
 9:       if q is inside OR-predicate(s) then
10:          Reevaluate OR-predicates
11:       else if q has no QNode children then
12:          outputPathSolutions(C_q)
13:    C_q→advance()
14: mergePathSolutions()
```

Σχήμα 3.34: Αλγόριθμος GTwigMerge.

AND/OR-ερωτήσεις.

QTwigMerge

Ο κύριος αλγόριθμος για τον GtwigMerge παρουσιάζεται στο Σχήμα 3.34 και η διαδικασία GetQNode στο Σχήμα 3.35.

Σε ό,τι αφορά τη διαδικασία QetQNode, η διαδικασία αυτή επιστρέφει έναν QNode κόμβο q_x με τρεις σημαντικές ιδιότητες: (1) ο q_x έχει μία έκταση (hasExtension) και $C_{q_x}\rightarrow start < C_{q_i}\rightarrow start$, $q_i \in$ Qchildren(q_x), (2) εάν $q_x \neq q$, τότε $C_{q_x}\rightarrow start < C_{q_j}\rightarrow start$, $q_j \in$ Qsibling(q_x), και (3) εάν $q_x \neq q$, τότε $C_{q_x}\rightarrow start < C_{Qparent(q_x)}\rightarrow start$. Αυτές οι τρεις ιδότητες εγγυώνται την ορθότητα του κύριου αλγόριθμου κατά την επεξεργασία του C_{q_x}. Σημειώνουμε επίσης ότι ο κόμβος q_{max} βρίσκεται μέσω της συνάρτησης getMaxQChild, η οποία λαμβάνει μέριμνα και για τους OR κόμβους και βασίζεται στην ανάλυση του λογικού-μέγιστου QNode κόμβου.

Σε ό,τι αφορά τον κυρίως αλγόριθμο του QTwigStack, ο κώδικας στις γραμμές 9-12 είναι καινούριος και αποτελεί το μόνο σημείο στο οποίο ο παρών αλγόριθμος διαφέρει από τον TSGeneric, επειδή τώρα χρειάζεται να επαναπροσδιορίσουμε την έννοια του δενδρικού στιγμιότυπου εξόδου. Πράγματι, στους προηγούμενους αλγόριθμους που εξετλασαμε, ένα στιγμιότυπο εξόδου περιελάμβανε στοιχεία από *όλους* τους QNode κόμβους στην ερώτηση. Η υιοθέτηση αυτής της προσέγγισης γίνεται όμως προβληματική στην περίπτωση των AND/OR-δενδρικών ερωτήσεων, αφού σε ένα ταίριασμα στην ερώτηση μπορεί να συνεισφέρουν μόνο μερικοί QNode κόμβοι της ερώτησης. Έτσι υιοθετούμε την εξής προσέγγιση: κάθε δενδρικό στιγμιότυπο εξόδου για μία AND/OR-δενδρική ερώτηση περιλαμβάνει στοιχεία από τους QNode κόμβους που δεν είναι μέσα σε οποιοδήποτε OR-κατηγόρημα και τα OR-κατηγορήματα απλά δρουν ως φίλτρα. Οι QNode κόμβοι που συμμετέχουν στην έξοδο καλούνται *κόμβοι εξόδου*.

Έτσι, αν ο q είναι ένας κόμβος−φύλλο εξόδου, όλες οι (μερικές) λύσεις μονοπατιού για το

Algorithm GetQNode(q)

```
1: if (isLeaf(q) then
2:     return q
3: for q_i in Qchildren(q) do
4:     q' = getNext(q_i)
5:     if (q' ≠ q_i) then
6:         return q'
7: q_max = getMaxQChild(q)
8: while C_q→end < C_{q_max}→start do
9:     C_q→advance()
10: q_min = argmin_{q_i}(C_{q_i}→start), q_i in Qchildren(q)
11: if hasExtension(q) and C_q→start < C_{q_min}→start then
12:     return q
13: else
14:     return q_min
```

Function getMaxQChild(q)

```
1: for each n_i in children(q) do
2:     if n_i is a QNode then
3:         q_i = n_i
4:     else
5:         q_i = ORBlockMax(n_i)
6: return argmax_{q_i}(C_{q_i}→start), for q_i initialized at lines 3, 5
```

Function hasExtension(q)

1: επιστρέφει true αν το στοιχείο στο C_q έχει ταίριασμα θεωρώντας όλους τους QNode κόμβους ως κόμβους προγόνου-απογόνου, αλλιώς επιστρέφει false.

Σχήμα 3.35: Αλγόριθμος GetQNode.

στοιχείο C_q εξάγονται (γραμμή 12). Ένας QNode κόμβος είναι κόμβος−φύλλο εξόδου αν είναι κόμβος εξόδου και δεν έχει QNode παιδιά. Σημειώνουμε ακόμα δύο βασικές λεπτομέρειες που είναι γνωστές από τους προηγούμενους αλγόριθμους. Πρώτον, αν υπάρχουν QNode κόμβοι γονέα-παιδιού σε μία μερική λύση μονοπατιού, τότε πρέπει να απορρίψουμε αυτή τη λύση αν οι αντίστοιχες ακμές δε σέβονται τη δομική σχέση γονέα-παιδιού. Δεύτερον, οι μερικές λύσεις μονοπατιού πρέπει να είναι ταξινομημένες από τη ρίζα προς τα φύλλα, έτσι ώστε η συγχώνευση να μπορέσει να γίνει αποδοτικά. Και πάλι χρησιμοποιείται η τεχνική του μπλοκαρίσματος των αποτελεσμάτων (3.3.3).

OR-κατηγορήματα με QNode κόμβους γονέα-παιδιού

Υποθέτουμε ότι η συνάρτηση QetQNode επιστρέφει έναν QNode κόμβο με k OR-κατηγορήματα n_1, ..., n_k. Αν κάποιο υποδέντρο n_x περιέχει QNode κόμβους γονέα-παιδιού, τότε είναι δυνατόν το στοιχείο C_q να μην ικανοποιεί το OR-κατηγόρημα n_x. Είναι μάλιστα δυνατόν το συγκεκριμένο στοιχείο C_q να μην ικανοποιεί το n_x καθόλου. Αυτό δημιουργεί ένα ενδεχόμενο πρόβλημα αν ο κόμβος q είναι ένας κόμβος εξόδου, επειδή το C_q μπορεί να συμμετέχει σε (μερικές) λύσεις μονοπατιού.

Αντιμετωπίζουμε το πρόβλημα αυτό ως εξής: για κάθε στοιχείο e_i σε μία στοίβα, όλα τα πιθανά extensions στα οποία το e_i συμμετέχει πρέπει να έχουν επιστραφεί από τον GetQNode(*root*) και να έχουν εξεταστεί από τον κύριο αλγόριθμο GTwigMerge, πριν το στοιχείο e_i εξαχθεί από τη λίστα. Βασισμένοι σε αυτήν την παρατήρηση, για κάθε κόμβο εξόδου q_j με OR-κατηγορήματα, μπορούμε να συνεχίζουμε να αποτιμούμε τα OR-κατηγορήματα για κάθε στοιχείο e_i στη στοίβα S_{q_j} και να ξέρουμε αν το e_i ικανοποιεί όλα τα OR-κατηγορήματα, όταν το στοιχείο αυτό εξάγεται από τη στοίβα.

Χωρίς να μπούμε σε επιπλέον λεπτομέρειες, είναι δυνατόν με χρήση πινάκων κατακερματισμού να υλοποιήσουμε την παραπάνω τεχνική. Οι πίνακες αυτοί δημιουργούνται κάθε φορά που εισάγεται ένα στοιχείο σε μία στοίβα, εφόσον πρόκειται είτε για QNode κόμβο μέσα σε ένα OR-κατηγόρημα ή για έναν κόμβο εξόδου με OR-κατηγορήματα (γραμμή 8 του GTwigMerge), ενώ η συντήρησή τους πραγματοποιείται στη γραμμή 10 του ίδιου αλγορίθμου.

Κλείνουμε την ενότητα 3.5.2 με το παρακάτω βασικό θεώρημα:

Θεώρημα 3.12. *Δοθέντων μιας AND/OR-δενδρικής ερώτησης q και μιας XML βάσης δεδομένων D, ο αλγόριθμος* GTwigMerge *ορθά επιστρέφει όλα τα δενδρικά στιγμιότυπα εξόδου για το q πάνω στη D.* □

Τέλος, αν η ερώτηση q περιέχει QNode κόμβους που είναι μόνο προγόνου-απογόνου, τότε αποδεικνύεται ότι ο QTwigMerge έχει χειρότερη I/O πολυπλοκότητα ανάλογη του $|input|$ + $|output|$ και χειρότερη CPU πολυπλοκότητα ανάλογη του $|Q|$ * $|input|$ + $|output|$, όπου $|Q|$ είναι το μέγεθος της ερώτησης, $|input|$ το συνολικό μέγεθος των λιστών των στοιχείων εισόδου, και $|output|$ το μέγεθος της εξόδου. Το κόστος αυτό δεν ισχύει στην περίπτωση που υπάρχουν και QNode κόμβοι γονέα-παιδιού.

3.8 Βέλτιστη κίνηση δρομέα σε ολιστικούς δενδρικούς συνδέσμους

Από τις ενότητες που προηγήθηκαν και κυρίως από την ενότητα 3.6 έχει καταστεί εμφανές ότι επειδή κάθε κίνηση του δρομέα σε μία λίστα ανεστραμμένου ευρετηρίου μπορεί να προκαλέσει είσοδο/έξοδο, η απόδοση του ολιστικού δενδρικού συνδέσμου καθορίζεται εν πολλοίς από τον αριθμό των μετακινήσεων των δρομέων. Παρά την παρατήρηση αυτή, οι προηγούμενοι αλγόριθμοι που εξετάστηκαν δεν έχουν βελτιστοποιηθεί ως προς αυτό το χαρακτηριστικό. Έχει υπάρξει περισσότερο ενδιαφέρον για την ελαχιστοποίηση των απαιτήσεων μνήμης των ενδιάμεσων αποτελεσμάτων παρά στην ελαχιστοποίηση των αριθμού των κινήσεων των δρομέων. Έτσι, αυτοί οι αλγόριθμοι παρουσιάζουν το πρόβλημα ότι κάνουν μία τοπική και άρα υποβέλτιστη επιλογή για το ποιο δρομέα θα μετακινήσουν κάθε φορά και το εύρος της μετακίνησης αυτής.

Αυτό το πρόβλημα αντιμετωπίζεται επιτυχώς στην εργασία [4] του Fontoura και άλλων. Στην εργασία τους, περιγράφεται ένας νέος ολιστικός αλγόριθμος δενδρικού συνδέσμου με βέλτιστη μετακίνηση δρομέα, ο αλγόριθμος TwigOptimal, ο οποίος σε αντίθεση με τους προηγηθέντες αλγόριθμους κοιτάζει πιο σφαιρικά την κατάσταση της ερώτησης για να καθορίσει

ποιο δρομέα να μετακινήσει στη συνέχεια. Το υπόλοιπο μέρος της ενότητας αυτής αναφέρεται στα σημαντικότερα αποτελέσματά της εργασίας αυτής.

3.8.1 Ο αλγόριθμος TwigOptimal

Δομές Δεδομένων

Όπως και στους υπόλοιπους ολιστικούς αλγόριθμους συνδέσμου, η κατάσταση του TwigOptimal σε κάθε χρονική στιγμή είναι μία τριπλέτα $< Q, C, S >$, όπου Q είναι η προς αποτίμηση δενδρική ερώτηση, C είναι το σύνολο των δρομέων με τους οποίους προσπελάζουμε τα δεδομένα των ανεστραμμένων ευρετηρίων, και S είναι το σύνολο των στοιβών για την κατασκευή των λύσεων. Οι κόμβοι δεδομένων που είναι μέρος μιας λύσης τοποθετούνται στις στοίβες S και εμφανίζονται στην έξοδο όταν βρίσκεται μία πλήρης λύση.

Ένας δρομέας C_q και μία στοίβα S_q σχετίζονται με κάθε κόμβο της ερώτησης q. Ο C_q δείχνει στο τρέχον στοιχείο του κόμβου q, ενώ η στοίβα S_q χρησιμοποιείται για να θυμόμαστε τα στοιχεία του q που είναι μέρος μίας λύσης. Κάθε είσοδος στη στοίβα έχει επίσης και ένα δείκτη σε μία άλλη είσοδο μίας στοίβας πρόγονου κόμβου.

Η θέση ενός δρομέα C_q προσπελάζεται μέσω των τιμών $C_q.start$, $C_q.end$ και $C_q.level$, και ομοίως για το στοιχείο της στοίβας S_q. Οι δομικοί περιορισμοί προγόνου-απογόνου ή γονέα-παιδιού ελέγχονται εξετάζοντας τις τιμές αυτές. Λέμε ότι ένας δρομέας C_p περικλείει έναν άλλο δρομέα C_q αν και μόνο αν $C_p.start \leq C_q.start$ και $C_p.end \geq C_q.end$. Ομοίως, λέμε ότι μία στοίβα S_p περικλείει το δρομέα C_q αν υπάρχει μία είσοδος στη στοίβα S_p που περικλείει τον C_q.

Για κάθε δρομέα C_q, η μέθοδος C_q.forwardTo(pos) μετακινεί τον C_q από την τρέχουσα θέση του στην πρώτη θέση που είναι μεγαλύτερη ή ίση της θέσης pos. Αυτό μπορεί να προκαλέσει λειτουργίες εισόδου/εξόδου καθώς ο C_q ψάχνει για τη θέση pos. Για να βελτιστοποιήσει τις μετακινήσεις των δρομέων, ο TwigOptimal χρησιμοποιεί επίσης εικονικούς δρομείς, οι οποίοι σε αντίθεση με τους φυσικούς δρομείς δεν προκαλούν λειτουργίες εισόδου/εξόδου. Απεναντίας, μία εικονική μετακίνηση του C_q απλά θέτει την τιμή $C_q.start$ χωρίς να μετακινεί φυσικά το δρομέα C_q. Αυτό θα γίνει κατανοητό σε λίγο. Η ιδιότητα C_q.virtual τίθεται σε TRUE κάθε φορά που ο C_q μετακινείται εικονικά και επανατίθεται σε FALSE όταν ο C_q μετακινείται φυσικά. Σε αντίθεση με τους μη διακλαδιζόμενους κόμβους, κάθε διακλαδιζόμενος κόμβος στην ερώτηση Q δε σχετίζεται με μία λίστα ανεστραμμένου ευρετηρίου κατά τα γνωστά. Ωστόσο, κάθε διακλαδιζόμενος κόμβος έχει ένα δρομέα, που χρησιμοποιείται για να περάσει τη θέση του δρομέα του γονέα του ή των παιδιών του, όταν ο TwigOptimal αποφασίζει για τον ποιο δρομέα να μετακινήσει. Συνεπώς, ο δρομέας ενός διακλαδιζόμενου κόμβου είναι πάντα εικονικός. Διατηρώντας ένα δρομέα για κάθε διακλαδιζόμενο κόμβο, ο αλγόριθμος δε χρειάζεται να ξεχωρίζει μεταξύ του τύπου των κόμβων, πράγμα που με τη σειρά του απλοποιεί τον αλγόριθμο.

```
ExecuteQuery()
1: αρχικοποίησε όλους τους δρομείς και τις στοίβες
2: while not done do
3:    q = ο κόμβος στο Q που σχετίζεται με τον ελάχιστο δρομέα
4:    while Extension(q == FALSE do
5:       MoveCursors(q)
6:       q = ο κόμβος στο Q που σχετίζεται με τον ελάχιστο δρομέα
7: OutputAndPush(q)
8: C_q.forwardTo(C_q.start + 1)
```

Σχήμα 3.36: Ο κύριος βρόχος.

Ο κύριος βρόχος

Η ExecuteQuery(), η οποία παρουσιάζεται στο Σχήμα 3.36, αποτελεί το σημείο εισόδου και το κύριο βρόχο του TwigOptimal. Αρχικοποιεί κάθε δρομέα στο πρώτο στοιχείο της αντίστοιχης λίστας ανεστραμμένου ευρετηρίου (γραμμή 1), και στη συνέχεια ψάχνει επανειλημμένα για τον (μη διακλαδιζόμενο) κόμβο της ερώτησης που έχει τον ελάχιστο δρομέα (γραμμή 3), δηλαδή το δρομέα με τη μικρότερη τιμή $start$, μέχρις ότου να βρει και να εμφανίσει όλες τις λύσεις.

Για να βρει και να εμφανίσει τις λύσεις, ο δρομείς μετακινούνται μέχρις ότου να βρεθεί μία επέκταση λύσης (Ορισμός 3.6) για τον κόμβο q (γραμμές 4-7). Όταν βρεθεί μία επέκταση λύσης, οι θέσεις των δρομέων στο υποδέντρο με ρίζα τον κόμβο q είναι εγγυημένο πως θα είναι μέρος της λύσης. Επιπρόσθετα, οι στοίβες των προγόνων του q περιέχουν τη θέση των στοιχείων που, όταν συνδυαστούν με την επέκταση λύσης για τον κόμβο q, σχηματίζουν την πλήρη λύση.

Αφότου βρεθεί μία επέκταση λύσης, η διαδικασία OutputAndPush καλείται (γραμμή 8) για να εμφανίσει νέες λύσεις που είναι κωδικοποιημένες στις στοίβες και για να τοποθετήσει το στοιχείο C_q στην αντίστοιχη στοίβα. Το στοιχείο C_q τοποθετείται στη στοίβα μόνο όταν είναι μέρος της λύσης. Τέλος, ο δρομέας C_q προχωρά στην επόμενη φυσική του τοποθεσία (γραμμή 9) για να αρχίσει την αναζήτηση μίας άλλης λύσης. Ο κύριος βρόχος τερματίζει όταν συναντάμε το τέλος μίας ή περισσότερων λιστών ανεστραμμένου ευρετηρίου, αφού τότε μπορούμε να συμπεράνουμε ότι δεν μπορούν να βρεθούν άλλες λύσεις.

Ελέγχοντας μία επέκταση λύσης

Η διαδικασία Extension(), η οποία απεικονίζεται στο Σχήμα 3.37, ελέγχει εάν οι θέσεις των δρομέων για τους κόμβους του υποδέντρου με ρίζα τον κόμβο q αποτελούν μία επέκταση λύσης. Το σύνολο $\{C\}$ είναι το σύνολο όλων των δρομέων των μη διακλαδιζόμενων κόμβων του υποδέντρου με ρίζα τον κόμβο q. Προκειμένου το υποδέντρο με ρίζα τον κόμβο q να σχηματίζει μία επέκταση λύσης, το στοιχείο C_q πρέπει να περικλείεται από τη στοίβα S_p του γονέα του p, όλοι οι δρομείς στο σύνολο $\{C\}$ να είναι πραγματικοί (μη εικονικοί), και όλοι οι δρομείς στο $\{C\}$ πρέπει να ικανοποιούν τους δομικούς περιορισμούς της ερώτησης Q (γραμμή 3). Η τελευταία συνθήκη συνεπάγεται ότι ένας δρομέας που αντιστοιχεί σε κόμβο με άλλον

Extension(q)
1: p = parent(q)
2: {C} = το σύνολο των δρομέων των κόμβων του υποδένδρου με ρίζα τον q
3: **if** (η στοίβα S_p περικλείει το στοιχείο C_q) **and** (όλοι οι δείκτες στο {C} είναι πραγματικοί) **and** (το {C} ικανοποιεί τους δομικούς περιορισμούς του Q) **then**
4: **return** TRUE
5: **else**
6: **return** FALSE

Σχήμα 3.37: Έλεγχος για επέκταση λύσης.

κόμβο AND από κάτω του πρέπει να περικλείει όλους τους δρομείς των παιδιών του, ενώ ένας δρομέας με έναν κόμβο OR από κάτω του πρέπει να περικλείει τουλάχιστον έναν από τους δρομείς των παιδιών του.

Μετακινώντας τους δρομείς

Ο αλγόριθμος TwigOptimal καλεί τη διαδικασία MoveCursors(q) στον κύριο βρόχο του καθώς ψάχνει για μία επέκταση λύσης, όπου q ο κόμβος με την ελάχιστη $start$ τιμή δρομέα. Η διαδικασία αυτή βασικά προσπαθεί να μετακινήσει τους δρομείς στο υποδέντρο με ρίζα τον κόμβο q στην επόμενη επέκταση λύσης για τον κόμβο q, αν βέβαια αυτή υπάρχει. Προκειμένου να αποφύγει λειτουργίες εισόδου/εξόδου, αυτό γίνεται μετακινώντας τους εικονικούς και όχι τους φυσικούς δρομείς. Ένας φυσικός δρομέας μετακινείται μόνο όταν δεν είναι δυνατές περαιτέρω μετακινήσεις του εικονικού δρομέα.

Η διαδικασία MoveCursors παρουσιάζεται στο Σχήμα 3.38. Πραγματοποιούνται δύο περάσματα πάνω στο υποδέντρο της Q με ρίζα τον κόμβο q για να μετακινηθούν εικονικά οι δρομείς, ένα πέρασμα από κάτω προς τα πάνω (γραμμή 6) και ένα πέρασμα από πάνω προς τα κάτω (γραμμή 7). Τα δύο περάσματα ανακαλύπτουν συνολικά και όχι απλά τοπικά το περισσότερο δυνατό που ο κάθε δρομέας μπορεί να μετακινηθεί χωρίς να χαθεί κάποια επέκταση λύσης για τον κόμβο q. Αυτό το στάδιο έρχεται σε ευθεία αντίθεση με τους προηγούμενος ολιστικούς αλγόριθμους συνδέσμου που εξετάστηκαν στα προηγούμενα κεφάλαια, αφού οι τελευταίοι μετακινούνται εξετάζοντας κάθε φορά μόνο τις θέσεις ενός ζεύγους δρομέων με σχέση γονέα-παιδιού και δεν υιοθετούν την πιο σφαιρική προσέγγιση του TwigOptimal.

Η TeigOptimal ξεκινά ελέγχοντας αν το στοιχείο C_q περικλείεται από τη στοίβα του γονέα του (γραμμή 2). Αν όχι, τότε ο C_q μετακινείται εικονικά στο μέγιστο των θέσεων $C_q.start$ και $C_p.start + 1$, αφού αυτό είναι το περισσότερο που ο C_q μπορεί να μετακινηθεί χωρίς να χαθεί κάποια επέκταση λύσης για τον q. Έπειτα, οι δρομείς μετακινούνται εικονικά σε δύο περάσματα (γραμμές 6-7). όπως περιγράφηκε νωρίτερα. Τέλος, ελέγχουμε αν μετά τα δύο περάσματα ο q εξακολουθεί να αντιστοιχεί στον ελάχιστο δρομέα. Αν αυτό ισχύει, τότε δεν είναι δυνατό να μετακινήσουμε άλλο τους εικονικούς δρομείς, αλλά πραγματοποιούμε πλέον φυσική κίνηση δρομέα (γραμμές 9-11).

Όταν η MoveCursors εξαναγκάζεται να κάνει μία φυσική μετακίνηση δρομέα, διαλέγει τον καλύτερο δυνατό εικονικό δρομέα προς μετακίνηση στο υποδέντρο με ρίζα τον q (γραμμή 9).

MoveCursors(q)
1: p = parent(q)
2: **if** (η στοίβα S_p δεν περικλείει το στοιχείο C_q) **then**
3: $C_q.start$ = max($C_q.start$, $C_p.start$ + 1)
4: C_q.virtual = TRUE
5: MoveCursorsBottomUp
6: MoveCursorsTopDown
7: **if** ο κόμβος q ακόμα αντιστοιχεί στον ελάχιστο δρομέα **then**
8: C_b = ο καλύτερος εικονικός δρομέας για να μετακινήσουμε μεταξύ του C_q και των απογόνων του
9: C_b.forwardTo($C_b.start$)
10: C_q.virtual = FALSE

Σχήμα 3.38: Μετακίνηση των δεικτών.

Στην ουσία, πρόκειται για το δρομέα που προβλέπεται να μετακινηθεί το περισσότερο δυνατό. Η επιλογή του καλύτερου δρομέα βασίζεται κατά κύριο λόγο σε ευριστικές συναρτήσεις, σαν αυτές που χρησιμοποιεί ο TSGeneric+ της ενότητας 3.6.3, και δεν εξηγείται αναλυτικά εδώ.

Η διαδικασία MoveCursorsBottomUp, η οποία απεικονίζεται στο Σχήμα 3.39, εκτελεί με αναδρομικό τρόπο ένα πέρασμα από κάτα προς τα πάνω στο υπό θεώρηση υποδέντρο. Ο σκοπός αυτού του περάσματος είναι να προσπαθήσει να μετακινήσει κάθε γονικό δρομέα μπροστά έτσι ώστε να περικλείει τους δρομείς των παιδιών του. Αν ο q είναι ένας AND κόμβος (γραμμή 4), τότε ο C_q πρέπει να περικλείει το δρομέα του μέγιστου παιδιού του έτσι ώστε να μπορεί να υπάρξει επέκταση λύσης. Ομοίως, αν ο q είναι ένας OR κόμβος (γραμμή 8), τότε ο C_q πρέπει να περικλείει το δρομέα του ελάχιστου παιδιού του. Τέλος, αν ο q είναι μη διακλαδιζόμενος κόμβος (γραμμή 12), τότε ο C_q πρέπει να περικλείει το δρομέα αυτού του μοναδικού του παιδιού.

Σε αυτό το σημείο υπενθυμίζεται ότι ο δρομέας ενός διακλαδιζόμενου κόμβου χρησιμοποιείται για το πέρασμα της θέσης του γονικού του δρομέα ή του δρομέα κάποιου παιδιού του. Εδώ, στην περίπτωση ενός AND κόμβου, ο δρομέας C_q χρησιμοποιείται για το πέρασμα προς τα πάνω της $start$ τιμής του μεγαλύτερου δρομέα μεταξύ των δρομέων των παιδιών του (γραμμές 5-6). Παρόμοια δράση λαμβάνεται στην περίπτωση ενός OR κόμβου αλλά αυτή τη φορά με το μικρότερο δρομέα (γραμμές 9-10). Τέλος, αν ο q είναι μη διακλαδιζόμενος κόμβος με ένα παιδί και η τιμή $C_q.end$ είναι μικρότερη από το δρομέα του παιδιού του, τότε ο C_q μετακινείται εικονικά στο μέγιστο των $C_q.start$ και $C_q.end$ + 1 (γραμμές 15-16), που είναι το μέγιστο που ο C_q μπορεί να μετακινηθεί χωρίς να χαθεί κάποια επέκταση λύσης. Η χρήση του μεγίστου είναι απαραίτητη (γραμμή 15) για την περίπτωση που μία προηγούμενη κλήση στη διαδικασία MoveCursors έχει ήδη μετακινήσει το $C_q.start$ μετά το $C_q.end$.

Αφού η MoveCursorsBottomUp τελειώσει, κάθε δρομέας θα έχει μετακινηθεί όσο το δυνατόν πιο μακριά του επιτρέπουν οι δρομείς των παιδιών του, χωρίς να χαθεί κάποια επέκταση λύσης. Η διαδικασία MoveCursorsTopDown, η οποία απεικονίζεται στο Σχήμα 3.40, καλείται τότε για να εκτελέσει αναδρομικά το από πάνω προς τα κάτω πέρασμα στο υπό θεώρηση υποδέντρο. Ο σκοπός αυτού του περάσματος είναι να προσπαθήσει να μετακινήσει το δρομέα κάθε παιδιού έτσι ώστε να περικλείεται από το δρομέα του γονέα του.

```
MoveCursorsBottomUp(q)
 1: for c ∈ children(q) do
 2:     MoveCursorsBottomUp(c)
 3: if ο q είναι AND κόμβος then
 4:     m = το παιδί του q με το μέγιστο δρομέα
 5:     C_q.start = C_m.start
 6: else if ο q είναι OR κόμβος then
 7:     m = το παιδί του q με τον ελάχιστο δρομέα
 8:     C_q.start = C_m.start
 9: else if ο q έχει ένα παιδί then
10:     c = το μοναδικό παιδί του q
11:     if C_q.end < C_c.start then
12:         C_q.start = max(C_q.start, C_q.end + 1)
13:         C_q.virtual = TRUE
```

Σχήμα 3.39: Πέρασμα από κάτω προς τα πάνω για τη μετακίνηση των δεικτών.

```
MoveCursorsTopDown(q)
1: for c ∈ children(q) do
2:     if ο c είναι AND ή OR κόμβος then
3:         C_c.start = C_q.start
4:     else if (C_c.start < C_q.start) and (το στοιχείο C_c δεν περικλείεται από τη στοίβα S_q) then
5:         C_c.start = C_q.start + 1
6:         C_c.virtual = TRUE
7:     MoveCursorsTopDown(c)
```

Σχήμα 3.40: Πέρασμα από πάνω προς τα κάτω για τη μετακίνηση των δεικτών.

Στη MoveCursorsTopDown, τα c και q αντιστοιχούν στους τρέχοντες κόμβους παιδιού και γονέα που εξετάζονται, αντίστοιχα. Αν ο c είναι ένας διακλαδιζόμενος κόμβος, τότε ο δρομέας C_c του παιδιού χρησιμοποιείται για να περάσει προς τα κάτω τη θέση του γονικού δρομέα C_q (γραμμή 3). Διαφορετικά, αν το $C_c.start$ είναι μικρότερο του $C_q.start$ και δεν περικλείεται από τη γονική στοίβα S_q, τότε ο δρομέας C_c μετακινείται εικονικά στη θέση $C_q.begin$ (γραμμές 5-8), που είναι το μέγιστο που μπορεί να μετακινηθεί χωρίς να χαθεί μία επέκταση λύσης, Ο λόγος που πρέπει να ελέγξουμε για το αν η στοίβα S_q περικλείει το C_c έχει να κάνει με την αποφυγή χαμένων λύσεων όταν υπάρχει κάποιο αναδρομικό δεδομένο για τον κόμβο q.

Εμφανίζοντας μία λύση

Οι λύσεις εμφανίζονται και τοποθετούνται στις στοίβες από τη διαδικασία OutputAndPush, η οποία παρουσιάζεται στο Σχήμα 3.41. Πριν το στοιχείο δρομέα τοποθετηθεί στην αντίστοιχη στοίβα (γραμμή 8), πραγματοποιείται ένας έλεγχος για να δούμε αν ο κόμβος q αντιστοιχεί στη ρίζα της ερώτησης Q. Αν συμβαίνει αυτό, τότε μία η περισσότερες λύσεις εξάγονται πριν ο νέος δρομέας ρίζας τοποθετηθεί στη στοίβα (γραμμές 2-6).

Σημειώνουμε επίσης για λόγους πληρότητας ότι η παρούσα μέθοδος είναι απλοϊκή και δεν

OutputAndPush(q)
1: **if** $q ==$ root$Q$ **then**
2: **while** το στοιχείο S_q.top() δεν είναι πρόγονος του C_q **do**
3: εμφάνισε τις λύσεις με το στοιχείο S_q.top()
4: S_q.pop()
5: από όλες τις στοίβες απομάκρυνε τα στοιχεία εκείνα που δεν έχουν ρίζα
6: S_q.push(C_q)

Σχήμα 3.41: Εξάγοντας μία λύση.

είναι αποδοτική χωρικά. Μπορεί να μετατραπεί σε μία πιο κατάλληλη και χωρικά αποδοτική μορφή με τη βοήθεια της μεθόδου που περιγράφεται στο [6].

3.8.2 Βελτιστότητα του TwigOptimal

Το παρακάτω θεώρημα βρίσκεται στην καρδιά του παραπάνω αλγορίθμου και αποτελεί ένα πολύ σπουδαίο αποτέλεσμα. Η απόδειξή του δεν παρουσιάζεται εδώ για λόγους εξοικονόμησης χώρου.

Θεώρημα 3.13. *Δοθείσης μίας κατάστασης αποτίμησης $< Q, C, S >$, όποτε ένας δρομέας μετακινείται φυσικά στον αλγόριθμο twigOptimal:*

1. Είναι αναγκαίο να μετακινηθεί ένας από τους δρομείς που εξετάζονται από τον TwigOptimal.

2. Ο δρομέας που μετακινείται φυσικά μετακινείται όσο το δυνατόν περισσότερο δίχως να χάνεται κάποια λύση. □

Τέλος, θεωρώντας ότι μόνο μετακινήσεις προς τα εμπρός είναι επιτρεπτές για τους δρομείς, και με βάση το θεώρημα 3.13, προκύπτει το παρακάτω σημαντικό θεώρημα:

Θεώρημα 3.14. *Υποθέτοντας πως οι μετακινήσεις δρομέα είναι μόνο προς τα εμπρός, ο αλγόριθμος TwigOptimal εκτελεί τον ελάχιστο αριθμό φυσικών μετακινήσεων δρομέα για να αποτιμήσει την ερώτηση Q.* □

Τέλος, ο Fontoura και οι υπόλοιποι προβαίνουν σε πειραματική αξιολόγηση, η οποία επαληθεύει έμπρακτα την ανωτερότητα του TwigOptimal έναντι των άλλων ολιστικών αλγορίθμων συνδέσμου, και ιδιαίτερα του TSGeneric+.

3.9 Από την κωδικοποίηση θέσης στο εκτεταμένο σχήμα Dewey: μία νέα προσέγγιση αποδοτικής επεξεργασίας ερωτήσεων δενδρικού προτύπου

Όλοι οι αλγόριθμοι που εξετάστηκαν στις προηγούμενες ενότητες έχουν ως σημείο εκκίνησης την κωδικοποίηση θέσης για την αναπαράσταση θέσης των στοιχείων του XML εγγράφου.

Πράγματι, θεωρώντας ότι η XML βάση δεδομένων έχει αναπαρασταθεί με αυτήν την κωδικοποίηση και εκμεταλλευόμενοι τις διάφορες ιδιότητες που συνεπάγεται μία τέτοια κωδικοποίηση για τις δομικές σχέσεις των στοιχείων της βάσης δεδομένων, ο προηγηθέντες αλγόριθμοι αποπειρώνται να δώσουν αποδοτική λύση στο πρόβλημα του ταιριάσματος ερώτησης δενδρικού προτύπου σε XML δεδομένα.

Η κωδικοποίηση θέσης, σε συνδυασμό με τις λίστες ανεστραμμένου ευρετηρίου, αποτέλεσε όντως ένα πολύ σημαντικό εργαλείο στην προσπάθεια αποδοτικής επεξεργασίας των ερωτήσεων με τους ολιστικούς αλγόριθμους. Ωστόσο, δεν παύει να παρουσιάζει μία σειρά από αδυναμίες. Για παράδειγμα, δοθεισών των κωδικοποιήσεων δύο στοιχείων, μπορούμε να βρούμε τη μεταξύ τους δομική σχέση με χρήση του σχήματος κωδικοποίησης θέσης, αλλά αν μας δοθεί μόνο η κωδικοποίηση ενός στοιχείου, δεν μπορούμε μόνο από αυτή να βρούμε όλους τους απογόνους του ή, ακόμα περισσότερο, τις σημάνσεις όλων των απογόνων του.

Το εναλλακτικό σχήμα εκτεταμένης κωδικοποίησης Dewey, τα κύρια σημεία του οποίου αναφέρθηκαν στην ενότητα 3.1.2, επιχειρεί να λύσει τις αδυναμίες αυτές της κωδικοποίησης θέσης, αυξάνοντας βέβαια τη χωρική πολυπλοκότητα της κωδικοποίησης. Βασιζόμενοι σε αυτήν την αναπαράσταση, ο Lu και άλλοι στην εργασία [10] προτείνουν ένα νέο αποδοτικό αλγόριθμο επεξεργασίας, τον TJFast, και μελετούν την πολυπλοκότητά του, παρέχοντας και πειραματική επαλήθευση.

3.9.1 Δομές Δεδομένων και Ορολογία

Έστω q η ερώτηση δενδρικού προτύπου και p_n το πρότυπο μονοπατιού από τη ρίζα μέχρι τον κόμβο $n \in q$. Χρησιμοποιούνται οι παρακάτω λειτουργίες πάνω στους κόμβους: isLeaf: Node → Bool, isBranching: Node → Bool, leafNodes: Node → {Node}, directBranchingOrLeafNodes: Node → {Node}. Η leafNodes(n) επιστρέφει το σύνολο των κόμβων φύλλων στο υποδέντρο με ρίζα τον κόμβο n. Η directBranchingOrLeafNodes(n) (εν συντομία, dbl(n)), επιστρέφει το σύνολο όλων των διακλαδιζόμενων κόμβων b και των κόμβων φύλλων f στο υποδέντρο με ρίζα τον κόμβο n, έτσι ώστε στο μονοπάτι από τον n στον b ή τον f (εξαιρουμένων των κόμβων στην άκρη) να μην υπάρχουν διακλαδιζόμενοι κόμβοι.

Σχετιζόμενη με κάθε κόμβο φύλλο f της ερώτησης είναι μία ροή δεδομένων T_f. Η ροή περιέχει τις εκτεταμένες κωδικοποιήσεις Dewey όλων των στοιχείων που ταιριάζουν τον κόμβο f. Τα στοιχεία στις ροές ταξινομούνται κατά αύξουσα λεξικογραφική διάταξη. Οι λειτουργίες σε μία ροή είναι οι current(T_f), advance(T_f), eof(T_f), και η λειτουργία τους είναι ακριβώς η ίδια με αυτήν που συναντήσαμε στους προηγούμενους αλγόριθμους. Επίσης, σε κάθε στοιχείο e του XML εγγράφου μπορούμε να εφαρμόσουμε τις λειτουργίες ancestors(e), descendants(e), οι οποίες επιστρέφουν τους προγόνους και τους απογόνους του στοιχείου e, αντίστοιχα.

Κατά τη διάρκεια της εκτέλεσής του. ο TJFast κρατάει μία δομή δεδομένων: ένα σύνολο S_b για κάθε διακλαδιζόμενο κόμβο b. Κάθε δύο στοιχεία στο σύνολο S_b πρέπει να έχουν δομική συγγένεια προγόνου-απογόνου (ή, προφανώς, γονέα-παιδιού). Επομένως, το μέγιστο μέγεθος του συνόλου S_b δεν είναι μεγαλύτερο από το μήκος του μεγαλύτερου μονοπατιού στο έγγραφο όπου γίνεται το ταίριασμα. Κάθε στοιχείο του συνόλου μπορεί εν δυνάμει συμμετέχει

σε απαντήσεις στην ερώτηση. Αρχικά, το S_b είναι άδειο.

3.9.2 TJFast

Ο αλγόριθμος TJFast παρουσιάζεται στο Σχήμα 3.42 και λειτουργεί σε δύο φάσεις. Στην πρώτη φάση (γραμμές 1-9), υπολογίζονται κάποιες λύσεις στα πρότυπα μονοπατιού ρίζας−προς−φύλλο της αρχικής ερώτησης. Στη δεύτερη φάση, αυτές οι λύσεις συγχωνεύονται για να υπολογιστούν οι τελικές λύσεις στην ερώτηση δενδρικού προτύπου.

Δοθείσης της εκτεταμένης κωδικοποίησης Dewey ενός στοιχείου, λόγω των ιδιοτήτων της κωδικοποίησης αυτής μπορούμε να ελέγξουμε αν το μονοπάτι του ταιριάζει με τα πρότυπα μονοπατιού ρίζας−προς−φύλλο. Επομένως, η ουσία του TJFast είναι να καθορίσουμε εάν μία λύση μονοπατιού μπορεί να συνεισφέρει σε λύσεις όλης της δενδρικής ερώτησης. Στην καλύτερη περίπτωση, εμφανίζουμε μόνο εκείνες τις λύσεις μονοπατιού που μπορούν να συγχωνευτούν με τουλάχιστον μία λύση από τα άλλα μονοπάτια ρίζας−προς−φύλλο. Δύο λύσεις μονοπατιού μπορούν να συγχωνευθούν, αν ταιριάζουν τους κοινούς διακλαδιζόμενους κόμβους της ερώτησης με το ίδιο στοιχείο. Επομένως, για να καθορίσει ο αλγόριθμος TJFast αν μία λύση μονοπατιού συνεισφέρει σε τελικές απαντήσεις, προσπαθεί να βρει τα πιο πιθανά στοιχεία που ταιριάζουν τους διακλαδιζόμενους κόμβους b και τα αποθηκεύει στο σύνολο S_b.

Η λειτουργία του TJFast είναι αρκετά απλή. Στις γραμμές 1-3, για κάθε ροή δεδομένων βρίσκουμε το πρώτο στοιχείο του οποίου το μονοπάτι ταιριάζει το πρότυπο μονοπατιού ρίζας−προς−φύλλο. Στη γραμμή 5, προσδιορίζουμε την επόμενη προς επεξεργασία ροή $T_{f_{act}}$ χρησιμοποιώντας τον αλγόριθμο getNext($topBranchingNode$), όπου ο κόμβος $topBranchingNode$ ορίζεται ως ο διακλαδιζόμενος κόμβος της δενδρικής ερώτησης που είναι πρόγονος όλων των υπόλοιπων διακλαδιζόμενων κόμβων (αν υπάρχουν). Στη γραμμή 6, εμφανίζουμε στην έξοδο μερικές ταιριασμένες λύσεις μονοπατιού στις οποίες κάθε στοιχείο που ταιριάζει ένα διακλαδιζόμενο κόμβο b μπορεί να βρεθεί στο αντίστοιχο σύνολο S_b. Προχωράμε στο επόμενο στοιχείο της ροής στη γραμμή 7 και βρίσκουμε το επόμενο στοιχείο της ροής $T_{f_{act}}$ που ταιριάζει το πρότυπο μονοπατιού στη γραμμή 8.

Ο αλγόριθμος getNext, που απεικονίζεται στο Σχήμα 3.43, βρίσκεται στην καρδιά του TJFast, και μας βοηθά σε δύο πράγματα. Το πρώτο είναι πως ανακαλύπτει την επόμενη προς επεξεργασία ροή δεδομένων, ενώ το δεύτερο πως ενημερώνει τα σύνολα S_b τα σχετιζόμενα με τους διακλαδιζόμενους κόμβους b.

Προκειμένου να βρει την επόμενη προς επεξεργασία ροή στοιχείων, ο αλγόριθμος getNext(n) επιστρέφει έναν κόμβο φύλλο f σύμφωνα με τα εξής τρία αναδρομικά κριτήρια: (α) αν ο n είναι κόμβος φύλλο, τότε επέστρεψε τον ίδιο τον n (γραμμή 2), αλλιώς (β) ο n είναι διακλαδιζόμενος κόμβος και για κάθε κόμβο $n_i \in \text{dbl}(n)$, (1) αν τα τρέχοντα στοιχεία δεν μπορούν να σχηματίσουν ένα ταίριασμα για το υποδέντρο με ρίζα τον κόμβο n_i, επιστρέφουμε αμέσως τον f_i (γραμμή 7), αλλιώς, (2) αν το τρέχον στοιχείο της ροής T_{f_i} δε συμμετέχει στη λύση που περιλαμβάνει μελλοντικά στοιχεία από άλλες ροές, επιστρέφουμε τον κόμβο f_i (γραμμή 14), αλλιώς, (3) επιστρέφουμε τον κόμβο f_{min} έτσι ώστε το τρέχον στοιχείο e_{min} να έχει την ελάχιστη σήμανση από όλα τα άλλα e_i κατά λεξικογραφική διάταξη (γραμμή 20).

Algorithm TJFast

1: **for each** f **in** leafNodes($root$) **do**
2: locateMatchedLabel(f)
3: **while** ¬end($root$) **do**
4: f_{act} = getNext($topBranchingNode$)
5: outputSolutions(f_{act})
6: advance($T_{f_{act}}$))
7: locateMatchedLabel(f_{act})
8: mergeAllPAthSolutions

Procedure locateMatchedLabel(f)

/* Υποθέτουμε ότι το μονοπάτι από τη ρίζα στο τρέχον στοιχείο της ροής T_f είναι $n_1/n_2/.../n_k$ και p_f είναι το πρότυπο μονοπατιού από τη ρίζα στον κόμβο φύλλο f */

1: **while** ¬($n_1/n_2/.../n_k$ ταιριάζει το πρότυπο p_f) ∧ (n_k ταιριάζει τον κόμβο f) **do**
2: advance(T_f)

Function end(n)

1: **return** $\forall f \in$ leafNodes(n) ⇒ eof(T_f)

Procedure outputSolutions(f)

1: εμφάνισε στην έξοδο λύσεις μονοπατιού του τρέχοντος στοιχείου T_f στο πρότυπο μονοπατιού p_f έτσι ώστε σε κάθε λύση s, $\forall e \in s \Rightarrow$ (το στοιχείο e ταιριάζει έναν διακλαδιζόμενο κόμβο $b \Rightarrow e \in S_b$)

Σχήμα 3.42: Αλγόριθμος TJFast.

Η δεύτερη λειτουργία του αλγορίθμου, η ενημέρωση του συνόλου S_b, είναι ιδιαίτερα σημαντική, αφού τα στοιχεία στο σύνολο S_b καθορίζουν ποιες λύσεις μονοπατιού θα εξαχθούν στην έξοδο από τη διαδικασία outputSolutions. Στη γραμμή 18, πριν ένα στοιχείο e εισαχθεί στο σύνολο S_b, βεβαιωνόμαστε ότι το e είναι πρόγονος (ή ταυτίζεται) με κάθε άλλο στοιχείο e_{b_i} που ταιριάζει τον κόμβο b στις αντίστοιχες λύσεις μονοπατιού.

Σημειώνουμε ότι η δεύτερη φάση του αλγορίθμου TJFast μπορεί να εκτελεσθεί αποδοτικά, μόνο όταν οι ενδιάμεσες λύσεις μονοπατιού εμφανίζονται σε ταξινομημένη διάταξη. Αυτό μπορεί να γίνει χρησιμοποιώντας την τεχνική του μπλοκαρίσματος, που εξετάσθηκε σε βάθος στην ενότητα 3.3.3.

3.9.3 Ανάλυση του TJFast

Παραθέτουμε τα παρακάτω δύο σημαντικά θεωρήματα της ορθότητας και της πολυπλοκότητας του TJFast δίχως απόδειξη.

Θεώρημα 3.15. *Δοθέντων μιας ερώτησης δενδρικού προτύπου q και μιας XML βάσης δεδομένων D, ο αλγόριθμος* TJFast *ορθά επιστρέφει όλες τις απαντήσεις για το q πάνω στη D.* □

Θεώρημα 3.16. *Έστω μία XML βάση δεδομένων D και μία ερώτηση δενδρικού προτύπου Q με μόνο δομικές σχέσεις προγόνου-απογόνου μεταξύ των διακλαδιζόμενων κόμβων και των παιδιών τους. Η χειρότερη περίπτωση I/O πολυπλοκότητας για τον TJFast είναι γραμμική ως προς το άθροισμα των μεγεθών των λιστών εισόδου και της λίστας εξόδου. Η χειρότερη περίπτωση για τη χωρική πολυπλοκότητα είναι $O(d^2 * |b| + d * |f|)$, όπου $|f|$ ο αριθμός*

Algorithm getNext(n)

1: **if** isLeaf(n) **then**
2: **return** n
3: **for all** n_i **in** dbl(n) **do**
4: f_i = getNext(n_i)
5: **if** isBranching(n_i) $\wedge$ empty(S_{n_i}) **then**
6: **return** f_i
7: $e_i = \max\{p|\text{p} \in \text{MB}(n_i, n)\}$
8: $n_{min} = minarg_{n_i}\{e_i\}$
9: $n_{max} = maxarg_{n_i}\{e_i\}$
10: **for all** n_i **in** dbl(n) **do**
11: **if** $\forall e \in$ MB(n_i, n): $\neg(e \in$ ancestors(e_{max})) **then**
12: **return** f_i
13: **for all** e **in** MB(n_{min}, n) **do**
14: **if** $e \in$ ancestors(e_{max}) **then**
15: updateSet(S_n, e)
16: **return** f_{min}

Function MB(n, b)

1: **if** isBranching(n) **then**
2: έστω e το μέγιστο στοιχείο στο σύνολο S_n
3: **else**
4: έστω e = current(T_n)
5: επέστρεψε το σύνολο από στοιχεία a που είναι απόγονοι του e έτσι ώστε το a να ταιριάζει με τον κόμβο b στη λύση μονοπατιού του e στο πρότυπο μονοπάτι p_n

Procedure clearSet(S, e)

1: διάγραψε οποιοδήποτε στοιχείο a στο σύνολο S έτσι ώστε: $\neg(a \in$ ancestors(e)) $\wedge \neg(a \in$ descendants(e))

Procedure updateSet(S, e)

1: clearSet(S, e)
2: $S = S \bigcup e$

Σχήμα 3.43: Αλγόριθμος getNext.

των κόμβων φύλλων της ερώτησης, $|b|$ ο αριθμός των διακλαδιζόμενων κόμβων της και d το μήκος της μακρύτερης σήμανσης στις λίστες εισόδου. □

Τονίζουμε ότι στην περίπτωση που η ερώτηση περιέχει και δομικές σχέσεις παιδιού-γονέα μεταξύ των διακλαδιζόμενων κόμβων και των παιδιών τους, ο TJFast δεν είναι πια βέλτιστος ως προς τις πράξεις εισόδου/εξόδου.

Με την παρουσίαση και του αλγορίθμου TJFast κλείνει το παρόν κεφάλαιο, που στόχο είχε την παρουσίαση των σημαντικότερων τεχνικών αποτίμησης ερωτήσεων δενδρικού προτύπου και προτύπου μονοπατιού. Έχοντας ως αφετηρία τον ολιστικό αλγόριθμο αποτίμησης ερωτήσεων μονοπατιού PathStack, θα αναπτυχθούν στο κεφάλαιο 5 αλγόριθμοι αποτίμησης ερωτήσεων μονοπατιού μερικής δομής. Πρώτα, όμως, θα αναπτυχθεί στο επόμενο κεφάλαιο μία νέα γλώσσα ερωτήσεων μονοπατιού μερικής δομής.

Κεφάλαιο 4

Γλώσσα και Τεχνικές Επεξεργασίας Ερωτήσεων XML με Πρότυπα Μονοπάτια Μερικής Δομής

4.1 Γλώσσα Ερωτήσεων XML με Πρότυπα Μονοπάτια Μερικής Δομής

4.1.1 Ερωτήσεις με Πρότυπα Μονοπάτια Μερικής Δομής

Σύνταξη. Μία ερώτηση μονοπατιού μερικής δομής (*PartialPathQuery*, PPQ) είναι μία ερώτηση μονοπατιού, όπου όμως η δομή δεν είναι πάντα αυστηρά καθορισμένη. Τυπικά:

Ορισμός 4.12. Μία *Ερώτηση Μονοπατιού Μερικής Δομής* (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. □

Η Εικόνα 4.1 απεικονίζει τρία τέτοια ερωτήματα. Το πρώτο, PPQ_1, για παράδειγμα, δύο σχέσεις προγόνου-απογόνου από τον κόμβο ρίζα στους κόμβους που σημαίνονται από τα στοιχεία x και d, καθώς και μία σχέση προγόνου-απογόνου από τον κόμβο που σημαίνεται από το στοιχείο x στον κόμβο που σημαίνεται από το s.

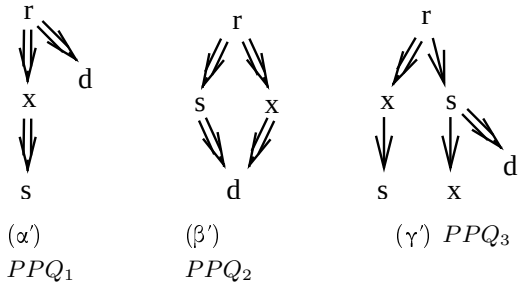
Είναι προφανές ότι ο χρήστης έχει τη δυνατότητα να καθορίσει τη δομή μίας τέτοιας ερώτησης, πλήρως, μερικώς ή ακόμη και καθόλου. Παραδείγματος χάρη, μία γλώσσα ερωτήσεων μερικής δομής επιτρέπει το σχηματισμό και τη διατύπωση ερωτήσεων που είναι πλήρως δομημένα μονοπάτια με σαφώς ορισμένες σχέσεις διαδοχής. Από την άλλη πλευρά, ένα PPQ μπορεί να μην περιέχει καμία απολύτως τέτοια σχέση διαδοχής, στην οποία περίπτωση εκ-

(α) PPQ_1=$\{r//x_0, x_0//s_0, r//d_0\}$
(β) PPQ_2=$\{r//s_0, s_0//d_0, r//x_0, x_0//d_0\}$
(ς) PPQ_3=$\{r//x_0, x_0/s_0, r/s_1, s_1/x_1, s_1//d_0\}$

Σχήμα 4.1: Παραδείγματα Ερωτήσεων Μονοπατιού με Πρότυπα Μερικής Δομής

φυλίζεται σε ένα σύνολο από λέξεις κλειδιά. Μεταξύ των δύο ακραίων περιπτώσεων, υπάρχουν ερωτήσεις που παρέχουν μερική περιγραφή της δομής χωρίς ωστόσο να καθορίζουν ένα μόνο συγκεκριμένο μονοπάτι.

Παριστάνουμε γραφικά τις ερωτήσεις αυτές χρησιμοποιώντας ορολογία γράφων, όπως κάναμε και στο προηγούμενο κεφάλαιο. 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 τριών ερωτήσεων της Εικόνας 4.1 παρουσιάζεται στην Εικόνα 4.2.



Σχήμα 4.2: Γραφική Αναπαράσταση των Τριών Ερωτήσεων.

Μία ερώτηση μονοπατιού μερικής δομής μπορεί να περιλαμβάνει περισσότερους από έναν κόμβους χωρίς εισερχόμενες ακμές (δηλαδή πηγές). Από υπόθεση, θεωρούμε όμως ότι κάθε XML δέντρο έχει ως ρίζα έναν κόμβο που σημαίνεται από r, μπορούμε να προσθέσουμε (αν δεν υπάρχει ήδη) έναν κόμβο r και διπλάβέλη 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.

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 PQ. Ορίζουμε με μεγαλύτερη αυστηρότητα την έννοια της απάντησης σε μία ερώτηση μονοπατιού μερικής δομής με την εισαγωγή κάποιων νέων ορισμών:

Ορισμός 4.13. *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)$\forall\ a_i/b_j$ (αντ. $a_i//b_j$) $\in Q$, $M(b_j)$ είναι παιδί (αντ. απόγονος) του $M(a_i) \in T$. □

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 4.14. 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 την ερώτηση PPQ_1 στην Εικόνα 4.1. Θα επιστρέψει όλους τους κόμβους του XML δένδρου T που σημαίνονται από τα στοιχεία r, x, s και d, αντίστοιχα, έτσι ώστε (α) όλοι τους κείνται στο ίδιο μονοπάτι του δέντρου T, (β) ο κόμβος που σημαίνεται από r είναι πρόγονος αυτών που σημαίνονται από x και d, (γ) ο κόμβος που σημαίνεται από x είναι πρόγονος αυτού που σημαίνεται από s, και (δ) η σχέση διαδοχής του κόμβου που σημαίνεται από d σε σχέση με τους κόμβους που σημαίνονται από x και s δεν είναι καθορισμένη.

Σημειώνουμε ότι και οι ερωτήσεις μονοπατιού μερικής δομής μπορεί να περιέχουν διακριτούς κόμβους που σημαίνονται όμως από το ίδιο στοιχείο, για παράδειγμα οι κόμβοι x_0 και x_1 στην ερώτηση PPQ_3. Οι εικόνες δύο τέτοιων κόμβων μπορεί να συμπέσουν, εκτός κι αν υπάρχει σχετίζονται μέσα από μία ακολουθία σχέσεων διαδοχής μεταξύ τους, οπότε και αντιστοιχίζονται σε διαφορετικούς κόμβους του δέντρου T.

Παρατηρούμε επίσης ότι και η ερώτηση PPQ_1 στην Εικόνα 4.2(α′) έχει δομή παρόμοια με αυτή ενός κλαδιού σε μία ερώτηση με πρότυπο δενδρικής δομής. Ωστόσο, στο πλάισιο των ερωτήσεων μονοπατιού μερικής δομής, ο d πρέπει να βρίσκεται στο ίδιο μονοπάτι με τους x και s 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.

4.1.2 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 που άμεσα καθορίζονται στην ερώτηση. Η αποτίμηση μίας τέτοιας ερώτησης μπορεί να επιταχυνθεί, αν τροποποιηθεί κατάλληλα με βάση τις παραγόμενες σχέσεις διαδοχής.

Συμπερασματική Εξαγωγή Δομικών Σχέσεων

Ας θεωρήσουμε την ερώτηση PPQ_3 της Εικόνας 4.1. Αφού ο κόμβος s_1 είναι γονέας του κόμβου x_1 και πρόγονος του d_0, μπορούμε να συμπεράνουμε ότι ο x_1 είναι επίσης πρόγονος του d_0. Με άλλα λόγια, ο d_0 οφείλει να βρίσκεται κάτω και από τους δύο κόμβους s_1 και x_1, οι δύο τελευταίοι κόμβοι συνδέονται μέσα από σχέση πατέρα-παιδιού και κανείς άλλος κόμβος δεν μπορεί να μεσολαβήσει ενδιάμεσα τους. Έχουμε επομένως:

Ορισμός 4.15. 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 $\mathcal{P}$ είναι το σύνολο που περιλαμβάνει τις σχέσεις διαδοχής στο $\mathcal{P}$, καθώς και όλες τις παραγόμενες σχέσεις διαδοχής από το σύνολο $\mathcal{P}$. □

Για να υπολογίσουμε το σύνολο ενός τέτοιου συνόλου σχέσεων $\mathcal{P}$, εισάγουμε ένα σύνολο από συμπερασματικούς κανόνες, οι οποίοι απεικονίζονται στην Εικόνα 4.3. 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 $\vdash$ για να δηλώσουμε ότι οι σχέσεις που προηγούνται του συμβόλου παράγουν τη σχέση που έπεται αυτού. Η απουσία σχέσεων που έπονται του συμβόλου $\vdash$ δηλώνει αξίωμα.

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

Σχήμα 4.3: Συμπερασματικοί Κανόνες

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

Θεώρημα 4.17. Έστω $\mathcal{R}$ ένα σύνολο από σχέσεις διαδοχής μίας ερώτησης μονοπατιού μερικής δομής και r μία δομική έκφραση που δεν ανήκει στο σύνολο $\mathcal{R}$. Το σύνολο των συμπερασματικών κανόνων της Εικόνας 4.3 είναι *συνεπές* (αν η r μπορεί να παραχθεί από το $\mathcal{R}$ χρησιμοποιώντας τους συμπερασματικούς κανόνες, τότε η r είναι πάντα μία παραγόμενη σχέση από το $\mathcal{R}$), και *πλήρες* (αν η r μπορεί να παραχθεί από το $\mathcal{R}$, τότε η r εμφανίζεται στο $\mathcal{R}$, ή μπορεί να παραχθεί από το $\mathcal{R}$ χρησιμοποιώντας τους συμπερασματικούς κανόνες). □

Αν το σύνολο $\mathcal{R}$ των σχέσεων διαδοχής του Q ισούται με το κλείσιμο του $\mathcal{R}$, λέμε ότι το Q είναι σε *πλήρη μορφή*. Για παράδειγμα, οι ερωτήσεις PPQ_1 και PPQ_2 της Εικόνας 4.1 είναι σε πλήρη μορφή. Η πλήρης μορφή της ερώτησης PPQ_3 απεικονίζεται στην Εικόνα 4.6(α′).

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 4.16. 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 μα είναι μη ικανοποιήσιμες. Ας θεωρήσουμε, για παράδειγμα, την ερώτηση q_5 της Εικόνας 4.4(a). 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 4.18. 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 $b//a$. □

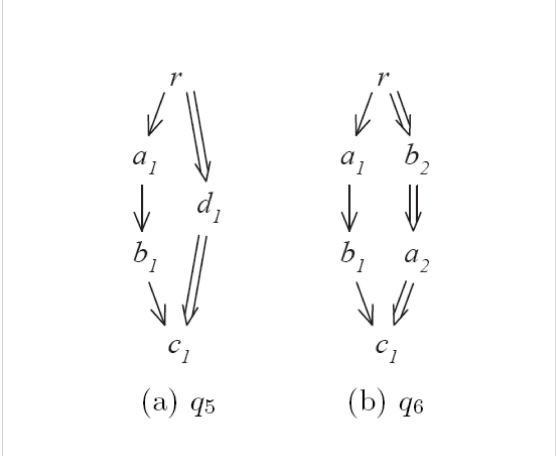
Για παράδειγμα, ας θεωρήσουμε, τις ερωτήσεις q_5 και q_6 της Εικόνας 4.4. Μπορεί εύκολα να δει κανείς πως η πλήρης μορφή και των δύο ερωτήσεων περιέχει τον τετριμμένο κύκλο $r//a_1$ και $a_1//r$

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 σε Ερωτήσεις

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

Ορισμός 4.17. Ένας κόμβος σε μία ερώτηση μονοπατιού μερικής δομής είναι *περιττός*, 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. □

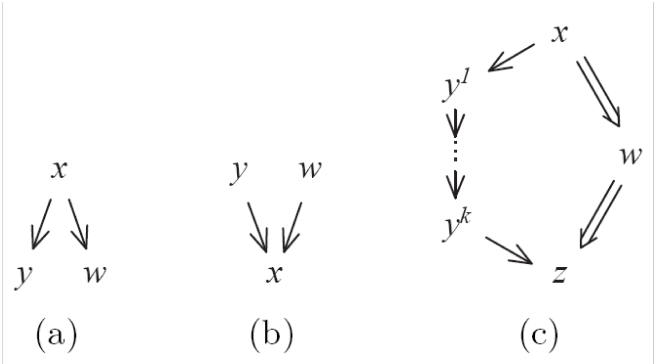


Σχήμα 4.4: Δύο μη ικανοποιήσιμες ερωτήσεις.

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

Θεώρημα 4.19. Ένας κόμβος w σε μία ερώτηση μονοπατιού μερικής δομής είναι περιττός, αν και μόνο αν η πλήρης μορφή της ερώτησης περιλαμβάνει ένα από τα παρακάτω σύνολα δομικών σχέσεων:
(α) x/w και x/y, όπου οι x και y είναι κόμβοι της ερώτησης και οι w και y έχουν την ίδια σήμανση.
(β) w/x και y/x, όπου οι x και y είναι κόμβοι της ερώτησης και οι w και y έχουν την ίδια σήμανση.
(γ)x/y^1, x/y^2, ..., x/y^k, $x//w$, $w//z$, $k \geq 1$, όπου x, y^1, y^2, ..., y^k, z και w είναι κόμβοι της ερώτησης και η σήμανση του w είναι η ίδια με τη σήμανση ενός από τους κόμβους y^1, y^2, ..., y^k. □

Οι Εικόνες 4.5(a), 4.5(b) και 4.5(c) απεικονίζουν τις 3 συνθήκες του Θεωρήματος 4.19.



Σχήμα 4.5: Ερωτήσεις Μονοπατιού Μερικής Δομής με περιττό κόμβο w.

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

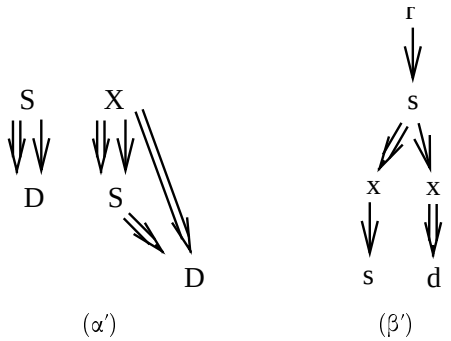
Κανονική Μορφή Ερωτήσεων

Τώρα, ας θεωρήσουμε μία ερώτηση Q και το σύνολο $\mathcal{R}$ 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 IP2). 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 4.18. Μία ερώτηση μονοπατιού με πρότυπο μερικής δομής Q είναι σε *κανονική μορφή*, αν το σύνολο $\mathcal{P}$ των σχέσεων διαδοχής του Q ισούται με το κλείσιμο του $\mathcal{P}$ αφαιρουμένου του συνόλου των σχέσεων διαδοχής που μπορούν να εξαχθούν από το $\mathcal{P}$ 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.

Οι ερωτήσεις PPQ_1 και PPQ_2 της Εικόνας 4.1 είναι σε κανονική μορφή . Η συμπαγής μορφή της ερώτησης PPQ_3 φαίνεται στην Εικόνα 4.6(β′).



Σχήμα 4.6: (α) Πλήρης Μορφή και (β) Συμπαγής Μορφή της Ερώτησης PPQ_3.

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 $IR1$ και $IR2$, 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 περιττεύοντες 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.

4.2 Τεχνικές Επεξεργασίας Ερωτήσεων XML με Πρότυπα Μονοπάτια Μερικής Δομής

Έχοντας ορίσει στο προηγούμενο κεφάλαιο την έννοια των ερωτήσεων με πρότυπα μονοπάτια μερικής δομής, είμαστε πλέον σε θέση να μπούμε στο πολύ ενδιαφέρον κομμάτι των τεχνικών επεξεργασίας αυτού του είδους ερωτήσεων. Στα πλαίσια αυτής της διπλωματικής θα ασχοληθούμε με τους βασισμένους σε στοίβα αλγορίθμους *PartialMJ* και *PartialPathStack*. Οι αλγόριθμοι αυτοί προτάθηκαν από το Σουλδάτο και άλλους στην εργασία [17] και αποτελούν την πρώτη ολοκληρωμένη προσπάθεια να αντιμετωπιστεί αποδοτικά το πρόβλημα της αποτίμησης ερωτήσεων μονοπατιού μερικής δομής.

4.2.1 Εισαγωγικά

Έστω q μία ερώτηση μονοπατιού μερικής δομής σε κανονική μορφή και n ο αριθμός των κόμβων σε αυτή. Η συνάρτηση nodes(q) επιστρέφει όλους τους κόμβους στην q. Η λογική συνάρτηση isRoot(n) είναι αληθής όταν ο κόμβος n δεν έχει εισερχόμενες ακμές στην q, αλλιώς είναι ψευδής. Η λογική συνάρτηση isSink(n) είναι αληθής αν ο κόμβος n δεν έχει εξερχόμενες ακμές στην q, αλλιώς είναι ψευδής. Η συνάρτηση parents(n) επιστρέφει όλους τους κόμβους στην ερώτηση q με εξερχόμενες ακμές προς τον κόμβο n.

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. Από τις διάφορες λειτουργίες που μπορούμε να υλοποιήσουμε με τους κέρσορες, εμείς περιοριζόμαστε στην advance(C_n), η οποία μετακινεί τον κέρσορα στο επόμενο στοιχείο της ροής. Η συνάρτηση eos(C_n) είναι αληθής αν ο C_n έχει φτάσει στο τέλος της T_n. Το C_n.begin δηλώνει το πεδίο *start* της κωδικοποίησης θέσης του C_n.

Μία στοίβα S_n σχετίζεται με κάθε κόμβο n της ερώτησης q. Στην περίπτωση του αλγορίθμου PartialMJ, κάθε είσοδος στη S_n είναι ένα ζευγάρι από έναν κόμβο από τη ροή T_n και από ένα δείκτη σε μία κατάλληλη θέση της στοίβας ενός γονέα του n στην ερώτηση. 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.

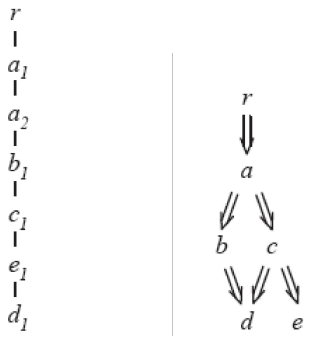
Η λογική συνάρτηση empty(S_n) είναι αληθής αν η στοίβα S_n είναι κενή, αλλιώς είναι ψευδής. Η λειτουργία push(S_n, *entry*) τοποθετεί το στοιχείο *entry* στην κορυφή της στοίβας S_n, ενώ η λειτουργία pop(S_n) εξάγει το στοιχείο που βρίσκεται στην κορυφή της στοίβας S_n. Οι συναρτήσεις bottom(S_n) και top(S_n) επιστρέφουν τη θέση του πυθμένα και της κορυφής της στοίβας S_n αντίστοιχα. Σε κάθε σημείο κατά τη διάρκεια της εκτέλεσης των αλγορίθμων: (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.

Είναι φανερό ότι πολλές από τις έννοιες που εισάγουμε σε αυτήν την ενότητα είναι ήδη γνωστές από το Κεφάλαιο 3, όπου αναλύσαμε τις τεχνικές αποτίμησης ερωτήσεων με πλήρη καθορισμό της δομής. Υπάρχουν όμως και σημαντικές διαφορές/επεκτάσεις, όπως γίνεται σαφές στις ενότητες που ακολουθούν.

4.2.2 Αλγόριθμος PartialMJ

Δοθείσης μίας ερώτησης μονοπατιού μερικής δομής, 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, που είδαμε με λεπτομέρεια στην ενότητα 3.3.3. 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 απάιτηση μπορεί να διασφαλιστεί με τη διαδικασία μπλοκαρίσματος που περιγράφηκε στον PathStack. Οι εγγραφές αυτές συνδυάζονται και συγχωνεύονται στην πορεία έτσι ώστε: (α) όλοι οι κόμβοι στις διάφορες μερικές λύσεις βρίσκονται στο ίδιο μονοπάτι, και (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.



(α′) XML Δέντρο (β′) Ερώτηση q

Σχήμα 4.7: Παράδειγμα ενός μονοδιάστατου XML δένδρου και μίας ερώτησης μονοπατιού μερικής δομής.

Το Σχήμα 4.7(β′) δείχνει το γράφο μίας ερώτησης q και το Σχήμα 4.9(α′) απεικονίζει ένα επικαλύπτον δένδρο q_s της ερώτησης q. Η ακμή $c//d$ της q λείπει από το q_s. Οποιαδήποτε μερικά αποτελέσματα των δύο μονοπατιών του q_s από τη ρίζα προς τα φύλλα που βρίσκονται στο ίδιο μονοπάτι του XML δένδρου μπορούν να συγχωνευθούν για να παράξουν ένα αποτέλεσμα για το q αν ικανοποιούν τις ταυτοτικές συνθήκες στους κόμβους r και a της ερώτησης q και επιπλέον τη δομική σχέση $c//d$.

Ο αλγόριθμος PartialMJ παρουσιάζεται στο Σχήμα 4.8. Σε αυτόν τον αλγόριθμο, κάθε θέση στη στοίβα S_n είναι ένα ζεύγος από (α) έναν κόμβο από τη ροή εισόδου T_n, και (β) ένα δείκτη ptr στη θέση του χαμηλότερου προγόνου του στη στοίβα S_m, όπου m ο γονέας

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

Algorithm PartialMJ(q)
1: **while** ¬end(q) **do**
2: n = getNextQueryNode()
3: cleanStacks(C_n)
4: **if** (isRoot(n) or ∀ $m \in$ parents(n): ¬empty(S_m)) **then**
5: moveToStack(n)
6: **if** isLeaf(n) **then**
7: showResults(S_n, top(S_n))
8: advance(C_n)
9: joinPathSolutions()

Function end()
return ∀ $n \in$ nodes(q): isSink(n) ⇒ eos(C_n)

Function getNextQUeryNode()
return $n \in$ nodes(q) such that C_n.begin is minimal

Procedure cleanStacks(C_n)
1: **for** m in nodes(q) **do**
2: pop all entries in S_m whose nodes are not ancestors of C_n in the XML tree

Procedure moveToStack(n)
1: ptr = pointer to top of S_m, where m is the parent of n in q_s
2: push(S_n, (C_n, ptr))

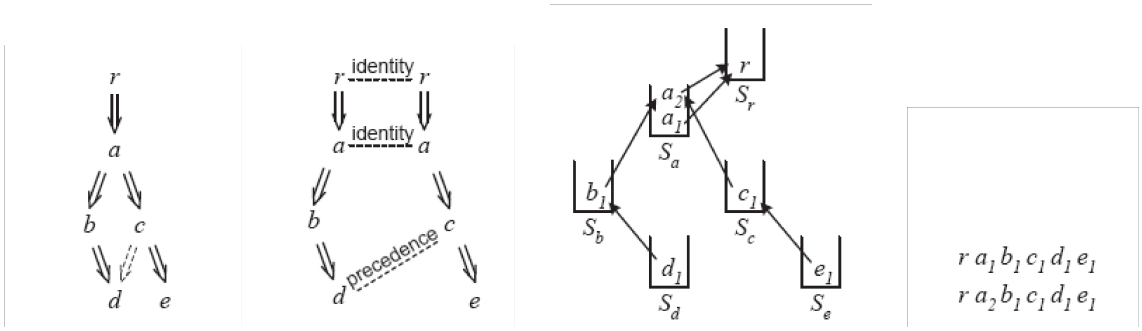
Procedure joinPathSolutions()
1: order the root-to-leaf paths of q_s in descending order of the level of their lowest branching node
2: merge-join the solutions of the root-to-leaf paths of q_s that are on the same path of the XML tree and satisfy the structural relationships in E

Σχήμα 4.8: Αλγόριθμος PartialMJ.

του κόμβου n στο επικαλύπτον δένδρο της ερώτησης. Η λογική συνάρτηση isLeaf(n) είναι αληθής αν ο κόμβος n είναι κόμβος φύλλο στο q_s, αλλιώς είναι ψευδής.

Στις γραμμές 1-8, 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 2 καθορίζει ποιος είναι ο επόμενος κόμβος της ερώτησης προς επεξεργασία. Η γραμμή 3 εξάγει από την κορυφή των στοιβών όλους τους κόμβους που δε 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 4-5). Με αυτόν τον τρόπο, αποφεύγουμε να αποθηκεύουμε στη στοίβα και να επεξεργαζόμαστε κόμβους που εγγυημένα δε συνεισφέρουν στο τελικό αποτέλεσμα. Όταν τοποθετούμε τον κόμβο C_n στην κορυφή της στοίβας S_n, προσθέτουμε επίσης κι 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 6 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 7 καλεί τη διαδικασία showResults, 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 πριεγράφη στον αλγόριθμο PathStack. Ο συνδυασμός και η συγχώνευση των τελικών αποτελεσμάτων γίνονται στη γραμμή 9. Αυτός ο σύνδεσμος περιλαμβάνει ότι (α) τα μερικά αποτελέσματα βρίσκονται στο ίδιο μονοπάτι του XML δένδρου, (β) τα ταιριάσματα των κοινών κόμβων στα μονοπάτια είναι ταυτόσημα (ταυτοτική σηνθήκη), και (γ) οι δομικές σχέσεις της ερώτησης που δεν εμφανίζονται στο επικαλύπτον δένδρο ικανοποιούνται. Όλες αυτές οι συνθήκες μπορούν να ελεγχθούν άμεσα χρησιμοποιώντας την αναπαράσταση θέσης των κόμβων στο XML δένδρο.



(α′) Επικαλύπτον Δένδρο q_s (β′) Συνθήκες Συνδέσμου (γ′) Στοίβες (δ′) Αποτελέσματα

Σχήμα 4.9: Παράδειγμα του PartialMJ.

Παράδειγμα 4.6. *Το Σχήμα 4.9(γ′) δείχνει την κατάσταση των στοιβών μετά την αποτίμηση της ερώτησης q του Σχήματος 4.7(β′) πάνω στα μονοδιάστατα XML δεδομένα του Σχήματος 4.7(α′). Το Σχήμα 4.9(α′) δείχνει το επικαλύπτον δένδρο q_s της q, το οποίο χρησιμοποιείται από τον αλγόριθμο αποτίμησης. Εφόσον η δομική σχέση $c//d$ της q δεν εμφανίζεται στο q_s, δεν υπάρχουν δείκτες από τη στοίβα S_d στη στοίβα S_c. Τα αποτελέσματα για το αριστερό μονοπάτι του q_s είναι $\{ra_1b_1d_1, ra_2b_1d_1\}$, ενώ αυτά για το δεξί μονοπάτι είναι $\{ra_1c_1e_1, ra_2c_1e_1\}$. Μπορεί τώρα εύκολα να δει κανείς πως από τα τέσσερα δυνατά ζεύγη αποτελεσμάτων μόνο δύο μπορούν να συνδυαστούν και να συγχωνευτούν, τα οποία παρουσιάζονται στο Σχήμα 4.9(δ′).* □

Σχέσεις Πατέρα-Παιδιού. Ο αλγόριθμος που παρουσιάστηκε προηγουμένως είναι σχεδιασμένος για την αποτίμηση ερωτήσεων που δεν περιέχουν σχέσεις πατέρα-παιδιού. Στην παρουσία των σχέσεων αυτών, δύο αλλαγές πρέπει να γίνουν. Πρώτον, 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) στη στοίβα S_a. Δεύτερον, 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 στα αποτελέσματα για το μονοπάτι μόνο αυτά περιέχουν επίσης από τη στοίβα S_a το γονέα του στο XML δένδρο.

Ανάλυση του PartialMJ. Ο αλγόριθμος PartialMJ γεμίζει τις στοίβες σε ένα μοναδικό πέρασμα των ροών εισόδου. Επιπλεόν, χρησιμοποιεί τη διαδικασία showResults για να παράγει αποτελέσματα για κάθε μονοπάτι από τη ρίζα προς ένα φύλλο του επικαλύπτοντος δένδρου της ερώτησης. Αυτή η διαδικασία έχουμε ήδη αποδείξει στην Ενότητα ;; 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.

4.2.3 Αλγόριθμος PartialPathStack

Για να αποφύγουμε το πρόβλημα των ενδιάμεσων αποτελεσμάτων του PartialMJ, η εργασία [17] προτείνει έναν καινούριο ολικό αλγόριθμο βασισμένο σε στοίβες για την αποτίμηση ερωτήσεων μονοπατιού μερικής δομής. Σε αντίθεση με τον PartialMJ, 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, ήτοι μία γραμμική διάταξη των κόμβων που σέβεται τη μερική διάταξη, όπως αυτή προκύπτει από τις δομικές σχέσεις στο γράφο της ερώτησης. Ο αλγόριθμος PartialPathStack απεικονίζεται στο Σχήμα 4.10.

Ο νέος αλγόριθμος διαχειρίζεται τις ροές εισόδου και τις στοίβες όπως ο προηγούμενος αλγόριθμος PartialMJ. Η μόνη διαφορά συνίσταται στη χρήση ενός συνόλου από δείκτες προς τις στοίβες όλων των γονέων ενός κόμβου που προστίθεται (στην κορυφή) σε μία στοίβα, αντί της χρήσης ενός μόνο δείκτη στο μοναδικό γονέα που καθορίζεται από το επικαλύπτον δένδρο της ερώτησης. Όποτε ένας κόμβος C_n της ροής T_n τοποθετείται στη στοίβα, αν αυτός ο κόμβος αντιστοιχεί σε κόμβο-καταβόθρα (χωρίς εξερχόμενες ακμές) της ερώτησης, τότε ο αλγόριθμος ελέγχει αν μπορούν να παραχθούν αποτελέσματα. Η έξοδος παράγεται 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. Για να αποφύγουμε την περιττή παραγωγή αποτελεσμάτων, ο αλγόριθμος εξάγει σε αυτό το σημείο μόνο αποτελέσματα που περιέχουν τον κόμβο C_n. Η διαδικασία outputResults συνδυάζει κόμβους από όλες τις στοίβες για να σχηματίσει τα αποτελέσματα. Κανένας άλλος κόμβος της στοίβας S_n δε χρησιμοποιείται σε αυτό το σημείο για να δημιουργήσει νέα αποτελέσματα για τη ερώτηση (γραμμές 7-8). Όλοι οι κόμβοι από στοίβες που δεν αντιστοιχούν σε κόμβους−καταβόθρες της ερώτησης μποτούν να χρησιμοποιηθούν αν αυτοί (ή κόμβοι πιο ψηλά στη στοίβα) δεικ-

τοδοτούνται από κόμβους στις στοίβες όλων των παιδιών του κόμβου n στην ερώτηση (γραμμές 12-15).

Παράδειγμα 4.7. *Το Σχήμα 4.11(β΄) δείχνει την κατάσταση των στοιβών μετά την αποτίμηση της ερώτησης q του Σχήματος 4.7(β΄) πάνω στα μονοδιάστατα XML δεδομένα του Σχήματος 4.7(α΄). Όταν ο κόμβος d_1 τοποθετείται στη στοίβα S_d, νέα αποτελέσματα που περιλαμβάνουν τον κόμβο d_1 μπορούν να παραχθούν σύμφωνα και με την τοπολογική διάταξη του Σχήματος 4.11(α΄). Τα αποτελέσματα της ερώτησης βρίσκονται στο Σχήμα 4.11(γ΄).* □

Για να επεξεργαστούμε ερωτήσεις με σχέσεις πατέρα-παιδιού, πρέπει να τροποποιήσουμε τη διαδικασία αποθήκευσης στις στοίβες και τη συνένωση/συγχώνευση των αποτελεσμάτων, όπως εξηγήσαμε 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 4.4. 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 4.20. Δοθέντων μιας ερώτησης με πρότυπο μονοπάτι μερικής δομής q και ενός XML δέντρου T, ο αλγόριθμος PathPathStack *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 4.21. 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 κι όχι απλά γράφος, μόνο ο

βαθμός-εξόδου χρειάζεται να φραχθεί από κάποια σταθερά για να επιτύχουμε ασυμπτωτική βελτιστότητα. Σε κάθε περίπτωση, ο PartialPathStack δε δημιουργεί άχρηστα ενδιάμεσα αποτελέσματα.

q: partial path query with N nodes

Algorithm PartialPathStack(q)

```
1: extract a topological order 1..N of the query nodes
2: while ¬end(q) do
3:    n = getNextQueryNode()
4:    cleanStacks(C_n)
5:    if (isRoot(n) or ∀ m ∈ parents(n): ¬empty(S_m)) then
6:       moveToStack(n)
7:       if isSink(n) and ∀ m ∈ nodes(q): isSink(m) ⇒ ¬empty(S_m) then
8:          if (n == N then
9:             outputResults(n, N, top(S_N))
10:         else
11:            for i = bottom(S_N) to top(S_N) do
12:               outputResults(n, N, i)
13:   advance(C_n)
```

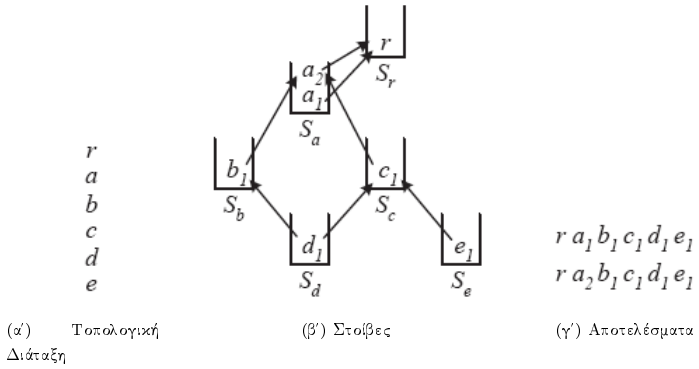
Procedure moveToStack(n)

```
1: ptrs = pointers to top of all parents(n) in q
2: push(S_n, (C_n, ptrs))
```

Procedure outputResults($sinkNode$, m, $stackPos$)

```
1: solution[m] = stackPos
2: if (m = 1) then
3:    output (S_1[solution[1]], ..., S_N[solution[N]])
4: else
5:    for a in parents(m) do
6:       S_a.pointerFrom[m] = S_m[stackPos].pointer_to_a
7:    if (m-1 = sinkNode) then
8:       outputResults(sinkNode, m-1, top(S_{m-1}))
9:    else if (isSink(m-1)) then
10:      for i = bottom(S_{m-1}) to top(S_{m-1}) do
11:         outputResults(sinkNode, m-1, i)
12:   else
13:      maxHeight = minarg_d{S_{m-1}.pointerFrom[d]}
14:      for i = bottom(S_{m-1}) to maxHeight do
15:         outputResults(sinkNode, m-1, i)
```

Σχήμα 4.10: Αλγόριθμος PartialPathStack.



(α′) Τοπολογική Διάταξη (β′) Στοίβες (γ′) Αποτελέσματα

Σχήμα 4.11: Παράδειγμα του PartialPathStack.

Κεφάλαιο 5

Τεχνικές Λεπτομέρειες

Το κεφάλαιο αυτό αποπειράται να δώσει μία σύνοψη των τεχνικών λεπτομερειών των υλοποιημένων αλγορίθμων, συγκεκριμένα του PathStack με αλλά και χωρίς μπλοκάρισμα αποτελεσμάτων, του PartialMJ και του PartialPathStack. Δεδομένου ότι στα πλαίσια της διπλωματικής δεν υλοποιήθηκε κάποιο πολύπλοκο σύστημα, αλλά ο προσανατολισμός ήταν ολοφάνερα προς τη μελέτη και σύγκριση των εν λόγω αλγορίθμων, οι τεχνικές λεπτομέρειες έχουν κυρίως να κάνουν με τα χρησιμοποιηθέντα προγραμματιστικά εργαλεία και με την περιγραφή των κλάσεων που απαρτίζουν το σύστημα. Ακολουθεί στη συνέχεια η περιγραφή τους.

5.1 Προγραμματιστικά εργαλεία

Το σύστημα υλοποιήθηκε στη γλώσσα προγραμματισμού C++. Η επιλογή της συγκεκριμένης γλώσσας προγραμματισμού για το σύστημά μας οφείλεται στους εξής λόγους:

- Ταχύτητα, την οποία και κληρονομεί από τη γλώσσα C. Το υλοποιημένο σύστημα δημιουργήθηκε για τη μελέτη και σύγκριση διαφορετικών αλγορίθμων, όπου αυτό που κυρίως αναμένουμε από τη γλώσσα προγραμματισμού είναι να παράγει κώδικα που τρέχει γρήγορα. Άλλα στοιχεία, όπως το GUI ή η διασύνδεση συστημάτων πάνω από το διαδίκτυο, δε μας ενδιαφέρουν στο πλαίσιο της παρούσας διπλωματικής.

- Προσανατολισμός προς το αντικείμενο. Η C++ προσφέρει στοιχεία αντικειμενοστρεφούς προγραμματισμού, όπως οι κλάσεις ή η κληρονομικότητα, οι οποίες μπορούν να διευκολύνουν σημαντικά την οργάνωση και ανάπτυξη του συστήματος.

- Επαναχρησιμοποίηση. Διάφορες κλάσεις που χρησιμοποιήθηκαν στο σύστημα είχαν ήδη γραφεί σε C++, οι οποίες και χρησιμοποιήθηκαν αυτούσιες ή ελαφρώς αλλαγμένες.

- Βιβλιοθήκες. Η τυποποίηση της C++ έρχεται με μία σειρά από βιβλιοθήκες που ορίζουν μία σειρά από δομές δεδομένων και συναρτήσεις/λειτουργίες πάνω στις δομές αυτές. Στοιχεία από τις βιβλιοθήκες χρησιμοποιήθηκαν εκτενώς στο παρόν σύστημα και συνέβαλαν σημαντικά στην απλοποίηση του κώδικα.

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 Vista, τρέξαμε τον εν λόγω μεταγλωττιστή μέσα από το λογισμικό Cygwin, το οποίο επιτρέπει σε λειτουργικά τύπου Microsoft Windows να συμπεριφέρονται με χαρακτηριστικά συστημάτων τύπου UNIX.

Τέλος, σημειώνουμε ότι χρησιμοποιήθηκε και ένα αρχείο γραμμένο στη γλώσσα Java, το RegionEncoding.java, του οποίου η λειτουργία θα εξηγηθεί στην επόμενη ενότητα.

5.2 Περιγραφή των κλάσεων

Στην ενότητα αυτή αναφέρονται οι κλάσεις που απαρτίζουν το σύστημα που υλοποιήσαμε, με πληροφορίες για τα πεδία τους, τις μεθόδους τους και τα αντίστοιχα αρχεία. Επίσης, αναφέρονται και τα επιπλέον αρχεία του συστήματος μαζί με τη χρησιμότητά τους.

5.2.1 public class Tag

Η κλάση αυτή περιγράφει έναν κόμβο του XML δένδρου πάνω στο οποίο κάνουμε την ερώτηση. Περιέχονται πεδία για την κωδικοποίηση θέσης καθώς και για τη σήμανση του στοιχείου που περιέχει, καθώς και αντίστοιχες μεθόδους.

Πεδία

- private unsigned int element

  Το στοιχείο (η σήμανση του στοιχείου) που περιέχει ο κόμβος.

- private unsigned int start

  Η ιδιότητα *start* της κωδικοποίησης θέσης.

- private unsigned int end

  Η ιδιότητα *end* της κωδικοποίησης θέσης.

- private unsigned int level

  Η ιδιότητα *level* της κωδικοποίησης θέσης.

- private unsigned int docID

  Η ιδιότητα *docID* της κωδικοποίησης θέσης.

Μέθοδοι

- public void setElement(unsigned int element)

  Μέθοδος που θέτει τη σήμανση του στοιχείου του κόμβου στην τιμή element.

- public void setStart(unsigned int start)

  Μέθοδος που θέτει την ιδιότητα *start* του κόμβου στην τιμή start.

- public void setEnd(unsigned int end)

  Μέθοδος που θέτει την ιδιότητα *end* του κόμβου στην τιμή end.

- public void setLevel(unsigned int level)

  Μέθοδος που θέτει την ιδιότητα *level* του κόμβου στην τιμή level.

- public void setDocID(unsigned int docID)

  Μέθοδος που θέτει την ιδιότητα *docID* του κόμβου στην τιμή docID.

- public unsigned int getElement()

  Μέθοδος που επιστρέφει τη σήμανση του στοιχείου που περιέχει ο κόμβος.

- public unsigned int getStart()

  Μέθοδος που επιστρέφει την τιμή της ιδιότητας *start* του κόμβου.

- public unsigned int getEnd()

  Μέθοδος που επιστρέφει την τιμή της ιδιότητας *end* του κόμβου.

- public unsigned int getLevel()

  Μέθοδος που επιστρέφει την τιμή της ιδιότητας *level* του κόμβου.

- public unsigned int getDocID()

  Μέθοδος που επιστρέφει την τιμή της ιδιότητας *docID* του κόμβου.

C++ Αρχεία

- Tag.h (διαπροσωπεία)
- Tag.cpp (υλοποίηση)

5.2.2 public class Tree

Η κλάση αυτή περιγράφει το XML δένδρο πάνω στο οποίο κάνουμε την ερώτηση. Περιέχονται πεδία και αντίστοιχες μέθοδοι για τους κόμβους που περιέχει το δένδρο, τις δομικές σχέσεις προγόνου-απογόνου και γονέα-παιδιού μεταξύ των στοιχείων του, καθώς τις ροές που αντιστοιχούν σε κάθε σήμανση στοιχείου που περιέχει. Χρησιμοποιεί την κλάση Tag.

Πεδία

- private unsigned int nNodes

  Ο αριθμός των κόμβων που περιέχει το δέντρο.

- private unsigned int nElements

  Ο αριθμός των διαφορετικών σημάνσεων στοιχείων που περιέχει το δέντρο.

- private unsigned int height

  Το ύψος του δέντρου.

- private unsigned int id

  Η ταυτότητα του δέντρου.

- private vector <unsigned int> treeNodes

  Το σύνολο των κόμβων του δέντρου.

- private map <unsigned int, unsigned int> node2element

  Η αντιστοίχιση των κόμβων στη σήμανση των στοιχείων που περιέχουν.

- private multimap <unsigned int, unsigned int> parent2Child

  Η αντιστοίχιση των κόμβων γονέων στους κόμβους παιδιά.

- private map <unsigned int, unsigned int> child2Parent

  Η αντιστοίχιση των κόμβων παιδιών στους κόμβους γονείς.

- private map <unsigned int, deque <Tag> > streams

  Η αντιστοίχιση της σήμανσης ενός στοιχείου στους κόμβους του δέντρου με αυτή τη σήμανση (οι ροές του περιγράψαμε στο κεφάλαιο 3).

Μέθοδοι

- private void createTag(int currentNode, unsigned int *index, unsigned int level)

  Εσωτερική μέθοδος που χρησιμοποιείται στην κατασκευή των ροών.

- private void printTag(int currentNode, int level, ofstream *file)

  Εσωτερική μέθοδος που χρησιμοποιείται στην κατασκευή των ροών.

- public void setNElements(unsigned int nElements)

  Μέθοδος που θέτει τον αριθμών των διαφορετικών σημάνσεων στοιχείων του δέντρου στην τιμή nElements.

- public void setNNodes(unsigned int nNodes)

  Μέθοδος που θέτει τον αριθμό των κόμβων του δέντρου στην τιμή nNodes.

- public void setHeight(unsigned int height)

  Μέθοδος που θέτει το ύψος του δέντρου στην τιμή height.

- public void setID(unsigned int id)

  Μέθοδος που θέτει την ταυτότητα του δέντρου στην τιμή id.

- public unsigned int getNElements()

  Μέθοδος που επιστρέφει τον αριθμό των διαφορετικών σημάνσεων των στοιχείων του δέντρου.

- public unsigned int getNNodes()

  Μέθοδος που επιστρέφει τον αριθμό των κόμβων του δέντρου.

- public unsigned int getHeight()

  Μέθοδος που επιστρέφει το ύψος του δέντρου.

- public unsigned int getID()

  Μέθοδος που επιστρέφει την ταυτότητα του δέντρου.

- public void addNode(unsigned int node, unsigned int element)

  Μέθοδος που προσθέτει στο δέντρο τον κόμβο node που σημαίνεται από το στοιχείο element.

- public unsigned int countNodes()

  Μέθοδος που μετράει τον αριθμό των κόμβων του δέντρου και επιστρέφει την τιμή της μέτρησης.

- public const vector <unsigned int> getNodes()

  Μέθοδος που επιστρέφει τους κόμβους του δέντρου.

- public unsigned int getElement(unsigned int node)

  Μέθοδος που επιστρέφει το στοιχείο από το οποίο σημαίνεται ο κόμβος node.

- public void addEdge(unsigned int node1, unsigned int node2)

  Μέθοδος που προσθέτει στο δέντρο μία ακμή από τον κόμβο node1 στον κόμβο node2.

- public const vector <unsigned int> getChildren(unsigned int node)

  Μέθοδος που επιστρέφει τα παιδιά του κόμβου node.

- public int getParent(unsigned int node)

  Μέθοδος που επιστρέφει το γονέα του κόμβου node.

- public void extractStreams()

  Μέθοδος που εξάγει τις ροές, μεταφράζοντας τους κόμβους σε ροές δεδομένων.

- public void extractNodes()

  Μέθοδος που εξάγει τους κόμβους, μεταφράζοντας τις ροές δεδομένων σε κόμβους.

- public void streamPushBack(unsigned int element, Tag tag)

  Μέθοδος που τοποθετεί στη ροή δεδομένων που αντιστοιχεί στο στοιχείο (σήμανση) element τον κόμβο που περιγράφεται με tag.

- public const deque<Tag> getStream(unsigned int element)

  Μέθοδος που επιστρέφει τη ροή δεδομένων που αντιστοιχεί στη σήμανση element.

C++ Αρχεία

- Tree.h (διαπροσωπεία)
- Tree.cpp (υλοποίηση)

5.2.3 public class PP

Η κλάση αυτή περιγράφει την ερώτηση μονοπατιού, ολικής ή μερικής δομής, που προσπαθούμε να ταιριάξουμε στο XML δένδρο. Περιέχονται πεδία και αντίστοιχες μέθοδοι για τους κόμβους που περιέχει η ερώτηση, καθώς και για τις δομικές σχέσεις προγόνου-απογόνου και γονέα-παιδιού μεταξύ των κόμβων της.

Πεδία

- private unsigned int id

  Η ταυτότητα της ερώτησης.

- private vector <unsigned int> ppNodes

  Το διάνυσμα των που απαρτίζουν την ερώτηση.

- private map <unsigned int, unsigned int> node2element

  Η αντιστοίχιση μεταξύ των κόμβων της ερώτησης και των στοιχείων από τα οποία σημαίνονται.

- private static const unsigned int maxNodes = 101

  Ο μέγιστος αριθμός των κόμβων σε μία ερώτηση.

- private int parentMatrix[maxNodes]

  Ο πίνακας που για κάθε κόμβο της ερώτησης δίνει τον κόμβο γονέα. Χρησιμοποιείται μόνο για δομικές σχέσεις γονέα-παιδιού.

- private int childMatrix[maxNodes]

  Ο πίνακας που για κάθε κόμβο της ερώτησης δίνει τον κόμβο γονέα. Χρησιμοποιείται μόνο για δομικές σχέσεις γονέα-παιδιού.

- private map <unsigned int, unsigned int> parent2Child

  Η αντιστοίχιση των κόμβων της ερώτησης στα παιδιά τους. Χρησιμοποιείται μόνο για δομικές σχέσεις γονέα-παιδιού.

- private map <unsigned int, unsigned int> child2Parent

  Η αντιστοίχιση των κόμβων της ερώτησης στους γονείς τους. Χρησιμοποιείται μόνο για δομικές σχέσεις γονέα-παιδιού.

- private bool ancDescMatrix[maxNodes][maxNodes]

  Ο πίνακας που για κάθε ζεύγος κόμβων της ερώτησης έχει τιμή TRUE αν οι κόμβοι έχουν σχέση προγόνου-απογόνου, αλλιώς FALSE. Χρησιμοποιείται μόνο για δομικές σχέσεις προγόνου-απογόνου.

- private vector<unsigned int> ancestorVector[maxNodes]

  Το διάνυσμα που περιέχει τους προγόνους κόμβους κάθε κόμβου της ερώτησης. Χρησιμοποιείται μόνο για δομικές σχέσεις προγόνου-απογόνου.

- private vector<unsigned int> descendantVector[maxNodes]

  Το διάνυσμα που περιέχει τους απογόνους κόμβους κάθε κόμβου της ερώτησης. Χρησιμοποιείται μόνο για δομικές σχέσεις προγόνου-απογόνου..

- private multimap <unsigned int, unsigned int> anc2Desc

  Η αντιστοίχιση μεταξύ των κόμβων της ερώτησης και των απογόνων κόμβων τους. Χρησιμοποιείται μόνο για δομικές σχέσεις προγόνου-απογόνου.

- private multimap <unsigned int, unsigned int> desc2Anc

  Η αντιστοίχιση μεταξύ των κόμβων της ερώτησης και των προγόνων κόμβων τους. Χρησιμοποιείται μόνο για δομικές σχέσεις προγόνου-απογόνου.

Μέθοδοι

- public unsigned int getID()

  Μέθοδος που επιστρέφει την ταυτότητα της ερώτησης μονοπατιού.

- public void addNode(unsigned int node, unsigned int element)

  Μέθοδος που προσθέτει στη δομή της ερώτησης τον κόμβο node που σημαίνεται από το στοιχείο element.

- public unsigned int countNodes()

  Μέθοδος που επιστρέφει τον αριθμό των κόμβων της ερώτησης.

- public const vector <unsigned int> getNodes()

  Μέθοδος που επιστρέφει σε ένα διάνυσμα τους κόμβους που απαρτίζουν την ερώτηση.

- public unsigned int getElement(unsigned int node)

  Μέθοδος που επιστρέφει το στοιχείο από το οποίο σημαίνεται ο κόμβος node.

- public bool isSink(unsigned int node)

  Μέθοδος που επιστρέφει TRUE, αν ο κόμβος node είναι κόμβος καταβόθρα, αλλιώς FALSE.

- public bool isSource(unsigned int node)

  Μέθοδος που επιστρέφει TRUE, αν ο κόμβος node είναι κόμβος πηγή, αλλιώς FALSE.

- public bool thereIsSingleArrow(unsigned int parent, unsigned int child)

  Μέθοδος που επιστρέφει TRUE, αν ο κόμβος parent και ο κόμβος child συνδέονται με σχέση γονέα-παιδιού, αλλιώς FALSE.

- public bool addSingleArrow(unsigned int parent, unsigned int child)

  Μέθοδος που προσθέτει μία σχέση γονέα-παιδιού μεταξύ των κόμβων parent και child.

- public void removeSingleArrow(unsigned int parent, unsigned int child)

  Μέθοδος που αφαιρεί τη σχέση γονέα-παιδιού μεταξύ των κόμβων parent και child.

- public unsigned int countSingleArrows()

  Μέθοδος που επιστρέφει τον αριθμό των δομικών σχέσεων γονέα-παιδιού της ερώτησης.

- public const map<unsigned int, unsigned int>::iterator getSingleArrow(unsigned int index)

  Μέθοδος που επιστρέφει την index-οστή σχέση γονέα-παιδιού μέσα στο πεδίο parent2Child.

- public const map<unsigned int, unsigned int> getSingleArrows()

  Μέθοδος που επιστρέφει όλες τις δομικές σχέσεις γονέα-παιδιού στην ερώτηση σε μορφή αντιστοίχισης από τους κόμβους γονείς στους κόμβους παιδιά.

- public int getChild(unsigned int node)

  Μέθοδος που επστρέφει το παιδί του κόμβου node.

- public int getParent(unsigned int node)

  Μέθοδος που επιστρέφει το γονέα του κόμβου node.

- public bool thereIsDoubleArrow(unsigned int ancestor, unsigned int descendant)

  Μέθοδος που επιστρέφει TRUE, αν ο κόμβος ancestor και ο κόμβος descendant συνδέονται με σχέση προγόνου-απογόνου, αλλιώς FALSE.

- public bool addDoubleArrow(unsigned int ancestor, unsigned int descendant)

  Μέθοδος που προσθέτει μία σχέση προγόνου-απογόνου μεταξύ των κόμβων ancestor και descendant.

- public void removeDoubleArrow(unsigned int ancestor, unsigned int descendant)

  Μέθοδος που αφαιρεί τη σχέση προγόνου-απογόνου μεταξύ των κόμβων ancestor και descendant.

- public unsigned int countDoubleArrows()

  Μέθοδος που επιστρέφει τον αριθμό των δομικών σχέσεων προγόνου-απογόνου της ερώτησης.

- public const multimap <unsigned int, unsigned int>::iterator getDoubleArrow(unsigned int index)

  Μέθοδος που επιστρέφει την index-οστή αντιστοίχιση σχέσεων προγόνου-απογόνου μέσα στο πεδίο anc2Desc.

- public const multimap <unsigned int, unsigned int> getDoubleArrows()

  Μέθοδος που επιστρέφει όλες τις δομικές σχέσεις προγόνου-απογόνου στην ερώτηση σε μορφή αντιστοίχισης από τους κόμβους προγόνους στους κόμβους απογόνους.

- public const vector<unsigned int> getDescendants(unsigned int node)

  Μέθοδος που επιστρέφει σε ένα διάνυσμα τους απογόνους κόμβους του κόμβου node.

- public const vector<unsigned int> getAncestors(unsigned int node)

  Μέθοδος που επιστρέφει σε ένα διάνυσμα τους προγόνους κόμβους του κόμβου node.

- public bool hasDescendants(unsigned int node)

  Μέθοδος που επιστρέφει TRUE, αν ο κόμβος node έχει απογόνους, αλλιώς FALSE.

- public bool hasAncestors(unsigned int node)

  Μέθοδος που επιστρέφει TRUE, αν ο κόμβος node έχει προγόνους, αλλιώς FALSE.

C++ Αρχεία

- PP.h (διαπροσωπεία)

- PP.cpp (υλοποίηση)

5.2.4 public class FileParser

Η κλάση αυτή περιέχει μεθόδους (αλλά όχι πεδία) που παίρνουν ως είσοδο τις διευθύνσεις των αρχείων που περιέχουν τα δεδομένα εισόδου, τα οποία μπορούν να περιγράφουν ένα από τα εξής τρία πράγματα:

1. Το δέντρο πάνω στο οποίο θέλουμε να ταιριάξουμε την ερώτηση μονοπατιού. Σε αυτήν την περίπτωση τα δεδομένα εισόδου περιέχουν τους κόμβους του δέντρου, τα αντίστοιχα στοιχεία από τα οποία σημαίνονται, καθώς και τη δενδρική διάταξη των εν λόγω κόμβων. Λόγω του ότι οι εξεταζόμενοι αλγόριθμοι απαιτούν τα δεδομένα σε οργάνωση λιστών ανεστραμμένου ευρετηρίου, η μορφή αυτή εισόδου χρειάζεται επιπλέον επεξεργασία για να πάρουμε τις ροές δεδομένων που αντιστοιχούν σε κάθε στοιχείο σήμανσης.

2. Τις ροές δεδομένων, οι οποίες για κάθε στοιχείο σήμανσης περιέχουν την πληροφορία της κωδικοποίησης θέσης (σε αύξουσα διάταξη της τιμής *start*) όλων των κόμβων του δέντρου που σημαίνονται από το στοιχείο αυτό. Στην περίπτωση αυτή τα δεδομένα εισόδου έχουν τη μορφή των λιστών ανεστραμμένου ευρετηρίου, έτσι δε χρειάζονται περαιτέρω επεξεργασία για να πάρουμε τις ροές εισόδου, που αποτελούν βασικές δομές δεδομένων των εξεταζόμενων αλγορίθμων. Πρόκειται λοιπόν για την πλέον βολική μορφή αναπαράστασης των δεδομένων του δένδρου.

3. Την ερώτηση μονοπατιού που θέλουμε να ταιριάξουμε στο δέντρο δεδομένων. Σε αυτήν την περίπτωση τα δεδομένα εισόδου περιέχουν τους κόμβους της ερώτησης, τα αντίστοιχα στοιχεία από τα οποία σημαίνονται, καθώς και τις δομικές σχέσεις μεταξύ των κόμβων.

Χρησιμοποιεί τις κλάσεις Tree και PP.

Μέθοδοι

- public Tree* readTree(char* folder, unsigned int nElements, unsigned int nNodes, unsigned int height, unsigned int id)

  Μέθοδος που διαβάζει τα δενδρικά δεδομένα, όταν είναι στη μορφή (1). Οι παράμετροι folder, nElements, nNodes, height, id καθορίζουν τη διεύθυνση του αρχείου με τα δεδομένα εισόδου. Επιστρέφει ένα δείκτη σε ένα αντικείμενο τύπου Tree, αφού ο τύπος αυτός χρησιμοποιείται από το σύστημά μας για την περιγραφή των δενδρικών δεδομένων εισόδου.

- public Tree* readStreamFile(char* folder, unsigned int nElements, unsigned int nNodes, unsigned int height, unsigned int id)

  Μέθοδος που διαβάζει τα δενδρικά δεδομένα, όταν είναι στη μορφή (2). Οι παράμετροι folder, nElements, nNodes, height, id καθορίζουν τη διεύθυνση του αρχείου με τα δεδομένα εισόδου. Επιστρέφει ένα δείκτη σε ένα αντικείμενο τύπου Tree. Επιστρέφει

ένα δείκτη σε ένα αντικείμενο τύπου Tree, αφού ο τύπος αυτός χρησιμοποιείται από το σύστημά μας για την περιγραφή των δενδρικών δεδομένων εισόδου.

- public PP* readPTPQ(char* folder, unsigned int tID, unsigned int qID, unsigned int nPP, unsigned int nNodes, unsigned int nSArrows, unsigned int nDArrows)

  Μέθοδος που διαβάζει δεδομένα που αφορούν την ερώτηση μονοπατιού, και τα οποία είναι πάντα στη μορφή (3). Οι παράμετροι folder, tID, qID, nPP, nNodes, nSArrows, nDArrows καθορίζουν τη διεύθυνση του αρχείου με τα δεδομένα εισόδου. Επιστρέφει ένα δείκτη σε ένα αντικείμενο τύπου PP, αφού ο τύπος αυτός χρησιμοποιείται από το σύστημά μας για την περιγραφή των ερωτήσεων μονοπατιού που θέλουμε να ταιριάξουμε πάνω στα δενδρικά δεδομένα εισόδου.

C++ Αρχεία

- FileParser.h (διαπροσωπεία)
- FileParser.cpp (υλοποίηση)

5.2.5 public class PathStack

Η κλάση αυτή αφορά τον αλγόριθμο PathStack. Περιέχονται πεδία για τις δομές δεδομένων που χρησιμοποιεί, καθώς και μέθοδοι που περιγράφουν τα διάφορα αλγοριθμικά βήματα. Χρησιμοποιεί την κλάση FileParser.

Πεδία

- private list < list <Tag> > results

  Σε αυτή τη δομή αποθηκεύονται τα αποτελέσματα της ερώτησης μονοπατιού που εξάγει ο αλγόριθμος.

- private PP * pp

  Δείκτης στην ερώτηση μονοπάτι που θέτουμε στο σύστημα.

- private list<Tag> outputVector

  Η δομή αυτή χρησιμοποιείται βοηθητικά στην εκδοχή του αλγόριθμου χωρίς μπλοκάρισμα αποτελεσμάτων για την αποθήκευση των αποτελεσμάτων στα οποία συμμετέχει κάθε κόμβος φύλλο που εισάγεται στη λίστα.

- private map <unsigned int, deque <Tag> > Tq

  Η αντιστοίχιση μεταξύ των κόμβων της ερώτησης μονοπατιού και των ροών δεδομένων των αντίστοιχων στοιχείων από τα οποία σημαίνονται οι εν λόγω κόμβοι.

- private map < unsigned int, vector < pair < pair < unsigned int, unsigned int>, pair < Tag, pair < list < list < Tag > >, list < list < Tag > > > > > > > Sq

  Η αντιστοίχιση μεταξύ των στοιχείων από τα οποία σημαίνονται οι κόμβοι των δενδρικών δεδομένων και των αντίστοιχων στοιβών.

- unsigned int leafNode

  Ο μοναδικός κόμβος φύλλο της ερώτησης.

- private unsigned int rootNode

  Ο μοναδικός κόμβος ρίζα της ερώτησης.

Μέθοδοι

- private bool moveStreamToStack(unsigned int qmin, unsigned int parent, PP * pp)

  Εσωτερική μέθοδος που αντιστοιχεί στη συνάρτηση moveStreamToStack του αλγορίθμου όπως περιγράφεται στην εργασία [2]. Επιστρέφει TRUE στην περίπτωση που όντως είχαμε μετακίνηση στοιχείου στη στοίβα, αλλιως FALSE.

- private unsigned int getMinSource(vector < unsigned int > ppNodes)

  Εσωτερική μέθοδος που αντιστοιχεί στη συνάρτηση getMinSource του αλγορίθμου όπως περιγράφεται στην εργασία [2].

- private void showSolutions(unsigned int qmin, unsigned int index, PP * pp)

  Εσωτερική μέθοδος που αντιστοιχεί στη συνάρτηση showSolutions του αλγορίθμου όπως περιγράφεται στην εργασία [2].

- private void showSolutionsWithBlocking(unsigned int currentNode, unsigned int index, PP * pp, bool inputFinished)

  Εσωτερική μέθοδος που αντιστοιχεί στη συνάρτηση showSolutionsWithBlocking του αλγορίθμου όπως περιγράφεται στην εργασία [2]. Η λογική μεταβλητή inputFinished είναι TRUE, αν και μόνο αν έχουμε φτάσει στο τέλος των δεδομένων εισόδου.

- public list < list <Tag> > runPathStack(PP *pp, Tree *tree, bool blocking)

  Μέθοδος που αντιστοιχεί στον αλγόριθμο PathStack όπως περιγράφεται στην εργασία [2]. Η λογική μεταβλητή blocking καθορίζει εάν θα έχουμε μπλοκάρισμα αποτελεσμάτων ή όχι.

C++ Αρχεία

- PathStack.h (διαπροσωπεία)
- PathStack.cpp (υλοποίηση)

5.2.6 public class PartialMJ

Η κλάση αυτή αφορά τον αλγόριθμο PartialMJ. Περιέχονται πεδία για τις δομές δεδομένων που χρησιμοποιεί, καθώς και μέθοδοι που περιγράφουν τα διάφορα αλγοριθμικά βήματα. Χρησιμοποιεί την κλάση FileParser.

Πεδία

- private list < list <Tag> > results

  Σε αυτή τη δομή αποθηκεύονται τα αποτελέσματα της ερώτησης μονοπατιού που εξάγει ο αλγόριθμος.

- private PP * pp

  Δείκτης στην ερώτηση μονοπάτι που θέτουμε στο σύστημα.

- private map <unsigned int, deque <Tag> > Tq

  Η αντιστοίχιση μεταξύ των κόμβων της ερώτησης μονοπατιού και των ροών δεδομένων των αντίστοιχων στοιχείων από τα οποία σημαίνονται οι εν λόγω κόμβοι.

- private map < unsigned int, vector < pair < pair < unsigned int, unsigned int>, pair < Tag, map < unsigned int, pair < list < list < Tag > >, list < list < Tag > > > > > > > > > Sq

  Η αντιστοίχιση μεταξύ των στοιχείων από τα οποία σημαίνονται οι κόμβοι των δενδρικών δεδομένων και των αντίστοιχων στοιβών.

- private set < unsigned int > leafNodes

  Το σύνολο από κόμβους φύλλα της ερώτησης.

- private vector < unsigned int > sinkNodes

  Το διάνυσμα από τους κόμβους καταβόθρες.

- private vector < unsigned int > examinedNodes

  Το διάνυσμα των κόμβων της ερώτησης σε τοπολογική διάταξη.

- private map < unsigned int, Tag > storeNodes

  Η αντιστοίχιση μεταξύ των κόμβων φύλλων και του τελευταίου στοιχείου που αποθηκεύτηκε στην αντίστοιχη στοίβα.

- private unsigned int rootNode

  Ο μοναδικός κόμβος ρίζα της ερώτησης μονοπατιού.

- private map < unsigned int, unsigned int > st

  Η αντιστοίχιση μεταξύ των κόμβων της ερώτησης και των γονέων τους στο επικαλύπτον δένδρο της ερώτησης.

- private map < unsigned int, vector < unsigned int > > STchildren

  Η αντιστοίχιση μεταξύ των κόμβων της ερώτησης και του διανύσματος των παιδιών τους στο επικαλύπτον δένδρο της ερώτησης.

- private map < unsigned int, vector < unsigned int > > qe

  Η αντιστοίχιση μεταξύ των κόμβων της ερώτησης και του διανύσματος των παιδιών τους αν από την αρχική ερώτηση αφαιρέσουμε τις ακμές του επικαλύπτοντος δένδρου.

- private map < unsigned int, list < list < Tag > > > pathSolutions

  Η αντστοίχιση μεταξύ των από ρίζα προς φύλλα μονοπατιών με τις αντίστοιχες μερικές λύσεις.

- private map < unsigned int, list < unsigned int > > pathIndexMap

  Αντιστοίχιση μεταξύ των κόμβων της ερώτησης και των ρίζα προς φύλλα μονοπατιών του ελάχιστου επικαλύπτοντος δένδρου της ερώτησης στα οποία συμμετέχουν.

Μέθοδοι

- private bool moveToStack(unsigned int qmin)

  Εσωτερική μέθοδος που αντιστοιχεί στη συνάρτηση moveToStack του αλγορίθμου όπως περιγράφεται στην εργασία [17]. Επιστρέφει TRUE στην περίπτωση που όντως είχαμε μετακίνηση στοιχείου στη στοίβα, αλλιως FALSE.

- private unsigned int getMinSource(vector < unsigned int > ppNodes)

  Εσωτερική μέθοδος που αντιστοιχεί στη συνάρτηση getNextQueryNode του αλγορίθμου όπως περιγράφεται στην εργασία [17].

- private bool end()

  Εσωτερική μέθοδος που αντιστοιχεί στη συνάρτηση end του αλγορίθμου όπως περιγράφεται στην εργασία [17].

- private void cleanStacks(unsigned int qmin)

  Εσωτερική μέθοδος που αντιστοιχεί στη συνάρτηση cleanStacks του αλγορίθμου όπως περιγράφεται στην εργασία [17].

- private bool checkNotEmptyParents(unsigned int qmin)

  Εσωτερική μέθοδος που επιστρέφει TRUE αν όλοι οι γονείς του κόμβου qmin έχουν μη κενές στοίβες, αλλιώς FALSE.

- private bool checkNotEmptySinks()

  Εσωτερική μέθοδος που επιστρέφει TRUE αν όλες οι καταβόθρες της ερώτησης έχουν μη κενές στοίβες, αλλιώς FALSE.

- private void initialize(Tree *tree, vector < unsigned int > ppNodes, set < unsigned int > *visitedNodes, unsigned int index)

  Εσωτερική μέθοδος που είναι υπεύθυνη για τις αρχικοποιήσεις των πεδίων της κλάσης.

- private void showSolutionsWithBlocking(unsigned int currentNode, unsigned int index, PP * pp, bool inputFinished)

  Εσωτερική μέθοδος που αντιστοιχεί στη συνάρτηση showSolutionsWithBlocking του αλγορίθμου όπως περιγράφεται στην εργασία [2]. Η λογική μεταβλητή inputFinished είναι TRUE, αν και μόνο αν έχουμε φτάσει στο τέλος των δεδομένων εισόδου.

- private void outputPathSolutions()

  Εσωτερική μέθοδος που εξάγει τις μερικές λύσεις που αντιστοιχούν στα ρίζα προς φύλλα μονοπάτια του δένδρου.

- private list < list < Tag > > mergeAllPathSolutions(unsigned int currentNode, unsigned int)

  Αναδρομική εσωτερική μέθοδος που συνενώνει τις μερικές λύσεις των ρίζα προς φύλλα μονοπατιών στην πλήρη λύση.

- private list < list < Tag > > merge_proc(list < list < Tag > >, list < list < Tag > >, unsigned int)

  Εσωτερική μέθοδος που καλείται από τη mergeAllPathSolutions και συνενώνει τις δύο λίστες που παίρνει ως όρισμα. Επιστρέφει τη συνενωμένη λίστα.

- private bool checkIfSamePath(Tag tag1, Tag tag2)

  Εσωτερική μέθοδος που επιστρέφει TRUE αν και μόνο αν οι κόμβοι tag1 και tag2 των δενδρικών δεδομένων βρίσκονται πάνω στο ίδιο μονοπάτι.

- private void filterSolutions()

  Εσωτερική μέθοδος που φιλτράρει τις εξαχθείσες πλήρεις λύσεις ώστε να αποκόψει τις λύσεις εκείνες που δεν ικανοποιούν τις δομικές σχέσεις του τμήματος της αρχικής ερώτησης που παραλείφθηκε από το επικαλύπτον δένδρο.

- public list < list < Tag > > runPartialMJ(PP *pp, Tree *tree)

  Μέθοδος που αντιστοιχεί στον αλγόριθμο PartialMJ όπως περιγράφεται στην εργασία [17].

C++ Αρχεία

- PartialMJ.h (διαπροσωπεία)
- PartialMJ.cpp (υλοποίηση)

5.2.7 public class PartialPathStack

Η κλάση αυτή αφορά τον αλγόριθμο PartialPathStack. Περιέχονται πεδία για τις δομές δεδομένων που χρησιμοποιεί, καθώς και μέθοδοι που περιγράφουν τα διάφορα αλγοριθμικά βήματα. Χρησιμοποιεί την κλάση FileParser.

Πεδία

- private list < list <Tag> > results

  Σε αυτή τη δομή αποθηκεύονται τα αποτελέσματα της ερώτησης μονοπατιού που εξάγει ο αλγόριθμος.

- private PP * pp

  Δείκτης στην ερώτηση μονοπάτι που θέτουμε στο σύστημα.

- private map < unsigned int, deque < Tag > > Tq

  Η αντιστοίχιση μεταξύ των κόμβων της ερώτησης μονοπατιού και των ροών δεδομένων των αντίστοιχων στοιχείων από τα οποία σημαίνονται οι εν λόγω κόμβοι.

- private map < unsigned int, vector < pair < map < unsigned int, unsigned int >, Tag > > > Sq

  Η αντιστοίχιση μεταξύ των στοιχείων από τα οποία σημαίνονται οι κόμβοι των δενδρικών δεδομένων και των αντίστοιχων στοιβών.

- private unsigned int nodesNumber

  Ο αριθμός των κόμβων της ερώτησης.

- private vector < unsigned int > sinkNodes

  Το διάνυσμα από τους κόμβους καταβόθρες.

- private vector < unsigned int > orderedNodes

  Το διάνυσμα των κόμβων της ερώτησης σε τοπολογική διάταξη.

- private unsigned int rootNode

  Ο μοναδικός κόμβος ρίζα της ερώτησης μονοπατιού.

- private vector < unsigned int > solution

  Βοηθητική δομή που χρησιμοποιείται από τη μέθοδο outputResults την οποία εξετάζουμε σε λίγο.

Μέθοδοι

- private bool moveToStack(unsigned int qmin)

  Εσωτερική μέθοδος που αντιστοιχεί στη συνάρτηση moveToStack του αλγορίθμου όπως περιγράφεται στην εργασία [17]. Επιστρέφει TRUE στην περίπτωση που όντως είχαμε μετακίνηση στοιχείου στη στοίβα, αλλιως FALSE.

- private unsigned int getMinSource(vector < unsigned int > ppNodes)

  Εσωτερική μέθοδος που αντιστοιχεί στη συνάρτηση getNextQueryNode του αλγορίθμου όπως περιγράφεται στην εργασία [17].

- private bool end()

  Εσωτερική μέθοδος που αντιστοιχεί στη συνάρτηση end του αλγορίθμου όπως περιγράφεται στην εργασία [17].

- private void cleanStacks(unsigned int qmin)

  Εσωτερική μέθοδος που αντιστοιχεί στη συνάρτηση cleanStacks του αλγορίθμου όπως περιγράφεται στην εργασία [17].

- private bool checkNotEmptyParents(unsigned int qmin)

  Εσωτερική μέθοδος που επιστρέφει TRUE αν όλοι οι γονείς του κόμβου qmin έχουν μη κενές στοίβες, αλλιώς FALSE.

- private bool checkNotEmptySinks()

  Εσωτερική μέθοδος που επιστρέφει TRUE αν όλες οι καταβόθρες της ερώτησης έχουν μη κενές στοίβες, αλλιώς FALSE.

- private void initialize(Tree *tree, vector < unsigned int > ppNodes, set < unsigned int > *visitedNodes, unsigned int index)

  Εσωτερική μέθοδος που είναι υπεύθυνη για τις αρχικοποιήσεις των πεδίων της κλάσης.

- private void outputResults(unsigned int n, unsigned int m, unsigned int stackPos)

  Εσωτερική μέθοδος που αντιστοιχεί στη συνάρτηση outputResults του αλγορίθμου όπως περιγράφεται στην εργασία [17].

- public list < list < Tag > > runPartialPathStack(PP *pp, Tree *tree)

  Μέθοδος που αντιστοιχεί στον αλγόριθμο PartialPathStack όπως περιγράφεται στην εργασία [17].

C++ Αρχεία

- PartialPathStack.h (διαπροσωπεία)
- PartialPathStack.cpp (υλοποίηση)

5.2.8 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 PathStack, PartialMJ, PartialPathStack, και FileParser.

5.2.9 Μορφή δεδομένων εισόδου - RegionEncoding.java

Κατά την πειραματική αξιολόγηση των υλοποιημένων αλγορίθμων τρέξαμε ερωτήσεις μονοπατιού διαφορετικής κάθε φορά δομής πάνω σε δύο benchmark XML αρχεία, το treebank.xml και xmark.xml. Ωστόσο, τα δενδρικά δεδομένα που χρησιμοποιούνται ως είσοδος στους τρεις αλγόριθμους του συστήματος πρέπει να είναι αντικείμενα της κλάσης Tree και όχι xml αρχεία. Από την άλλη, αντικείμενα της κλάσης Tree δημιουργούνται όπως είδαμε από την κλάση FileParser, η οποία παίρνει ως είσοδο ένα αρχείο που περιέχει είτε την περιγραφή του δένδρου (μορφή (1)) είτε τις ροές δεδομένων (μορφή (2)) και επιστρέφει ένα δείκτη σε αντικείμενο τύπου Tree. Έπρεπε λοιπόν να ακολουθήσει ένα στάδιο μετασχηματισμού, κατά το οποίο τα αρχεία xml θα μετασχηματίζονταν σε αρχεία που θα περιείχαν τα δενδρικά δεδομένα είτε στη μορφή (1) είτε στη μορφή (2), ώστε να μπορέσουν στη συνέχεια να χρησιμοποιηθούν από τους αλγόριθμους.

Υπεύθυνο ακριβώς για αυτόν το μετασχηματισμό είναι το αρχείο RegionEncoding.java, γραμμένο στη γλώσσα java. Συγκεκριμένα, παίρνει ως είσοδο ένα XML αρχείο και στην έξοδο δίνει τις ροές δεδομένων, που αντιστοιχούν στις σημάνσεις των κόμβων του xml αρχείου, δηλαδή μετασχηματίζει τα xml δεδομένα στη μορφή (2). Επίσης, επειδή το σύστημά μας αναπαριστά τις σημάνσεις τόσο των κόμβων του δέντρου όσο και των κόμβων της ερώτησης με μη προσημασμένους ακεραίους (unsigned integer) όπως φαίνεται στο πεδίο map <unsigned int, unsigned int> node2element των κλάσεων Tree και PP, το αρχείο αυτό κάνει και μία μετάφραση των στοιχείων που σημαίνουν τους κόμβους σε μοναδικούς μη προσημασμένους ακεραίους. Επομένως, θα μπορούσαμε να πούμε ότι το αρχείο RegionEncoding.java καταφέρνει να εξασφαλίσει τη συνέπεια στη μορφή μεταξύ των αρχικών δεδομένων εισόδου (xml αρχεία) και των δεδομένων όπως τα δέχονται ως είσοδο οι αλγόριθμοι του συστήματος.

Συνέπεια του παραπάνω μετασχηματισμού είναι ότι πριν απευθύνουμε κάποια ερώτηση στο σύστημα, πρέπει να βρούμε σε ποιους μη προσημασμένους ακεραίους αντιστοιχούν τα στοιχεία σήμανσης που περιέχει η αρχική xml ερώτηση και να επαναδιατυπώσουμε την ερώτηση σε μορφή που αντικαθιστά τις αρχικές σημάνσεις με τις μετασχηματισμένες μη προσημασμένες ακέραιες μορφές τους. Αυτή ήταν και η διαδικασία που ακολουθήθηκε στη φάση της πειραματικής αξιολόγησης/επαλήθευσης.

Κεφάλαιο 6

Πειραματική Αξιολόγηση

Το παρόν κεφάλαιο επιχειρεί μία σύνοψη των πειραματικών αποτελεσμάτων που πήραμε από την εκτέλεση των υλοποιημένων αλγορίθμων PathStack, PathStackWithBlocking, PartialMJ, και PartialPathStack, οι τεχνικές λεπτομέρειες των οποίων εδόθησαν στο προηγούμενο κεφάλαιο. Μέσω της παράθεσης του πειραματικού ελέγχου θα επιχειρήσουμε μία σύγκριση της απόδοσης των υλοποιημένων αλγορίθμων για ερωτήσεις μονοπατιού τόσο ολικής όσο και μερικής δομής, ενώ ταυτόχρονα θα επιβεβαιώσουμε τα θεωρητικά/αναλυτικά αποτελέσματα που αφορούν τους αλγόριθμους αυτούς.

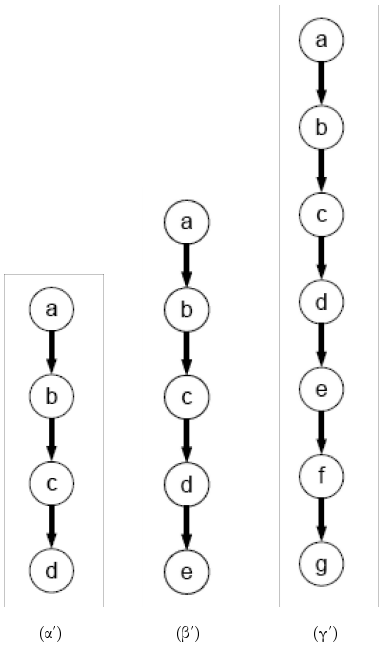
6.1 Μεθοδολογία Ελέγχου

Όλα τα πειράματά μας έτρεξαν σε 2 GHz Core 2 Duo επεξεργαστή Intel με 2 GB RAM και σε λειτουργικό σύστημα Windows Vista. Οι ερωτήσεις έγιναν πάνω σε δύο XML αρχεία με πολύ διαφορετικά χαρακτηριστικά, τα treebank.xml και xmark.xml, τα κύρια χαρακτηριστικά των οποίων αναφέρουμε παρακάτω. Επίσης, παρακάτω αναφέρουμε και τον τύπο (δομή) των διάφορων ερωτήσεων που έγιναν πάνω στα δύο XML αρχεία.

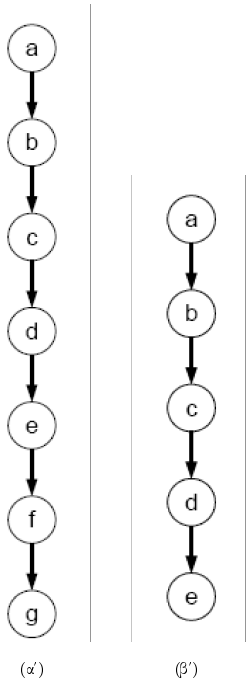
6.1.1 XML Αρχεία Ελέγχου

treebank.xml

Ένα treebank είναι ένα μεγάλο και δομημένο σύνολο κειμένων, όπου η κάθε πρόταση έχει σημανθεί με στοιχεία συντακτικής δομής. Η συντακτική δομή αναπαριστάται ως μία δενδρική δομή, εξ ου και το δενδρικό XML αρχείο, ενώ βρίσκει εκτεταμένη χρήση στην επιστήμη της γλωσσολογίας και της υπολογιστικής γλωσσολογίας. Εμείς χρησιμοποιήσαμε το treebank του πανεπιστημίου της Pennsylvania ([35]), στο οποίο σημαίνεται η δομή των φράσεων που απαρτίζουν το κείμενο. Πρόκειται για ένα βαθύ έγγραφο με έντονο το στοιχείο της αναδρομής. Το σύνολο δεδομένων αποτελείται από περίπου 2.5 εκατομμύρια κόμβους που σημαίνονται από 250 διαφορετικά στοιχεία, ενώ το μέγιστο βάθος είναι 36. Το μέγεθος του εν λόγω αρχείου είναι 82 MB. Περιέχει δεδομένα του πραγματικού κόσμου.



Σχήμα 6.1: (α) Ο τύπος της ερώτησης $Q_{t,1}$, (β) ο τύπος της ερώτησης $Q_{t,2}$, και (γ) ο τύπος της ερώτησης $Q_{t,3}$.



Σχήμα 6.2: (α) Ο τύπος των ερωτήσεων $Q_{x,1}$ και $Q_{x,2}$, και (β) ο τύπος της ερώτησης $Q_{x,3}$.

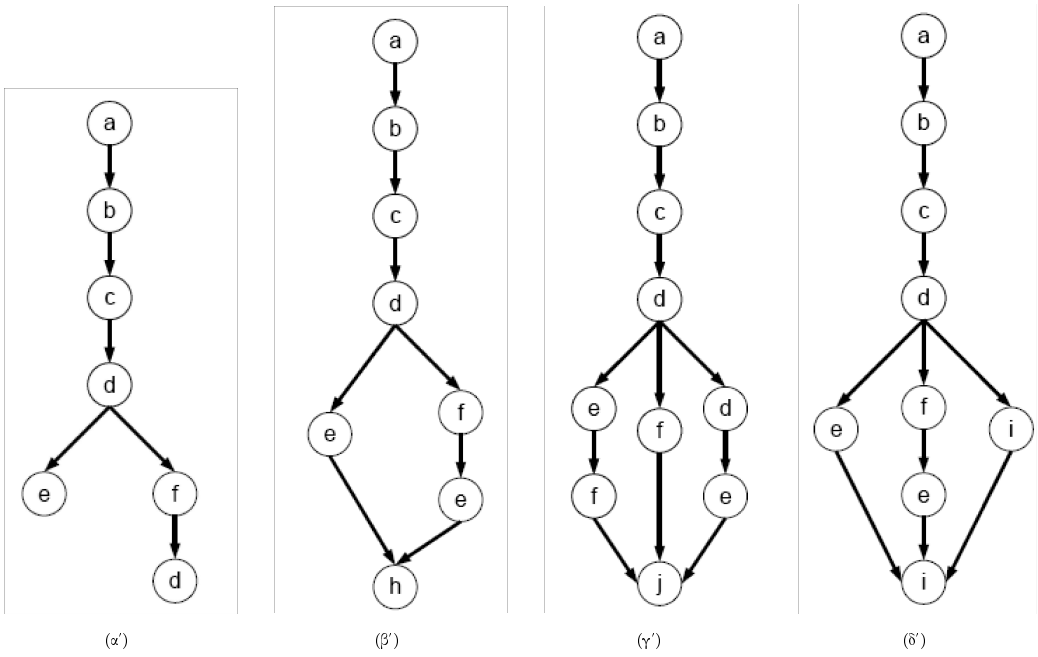
xmark.xml

Το αρχείο xmark.xml είναι ένα benchmark αρχείο ([36]), το οποίο χρησιμοποιείται για το συγκριτικό έλεγχο της απόδοσης και των ικανοτήτων διάφορων XML εγγράφων, επιτρέποντας στους χρήστες να βγάλουν χρήσιμα συμπεράσματα για τα χαρακτηριστικά των δικών τους

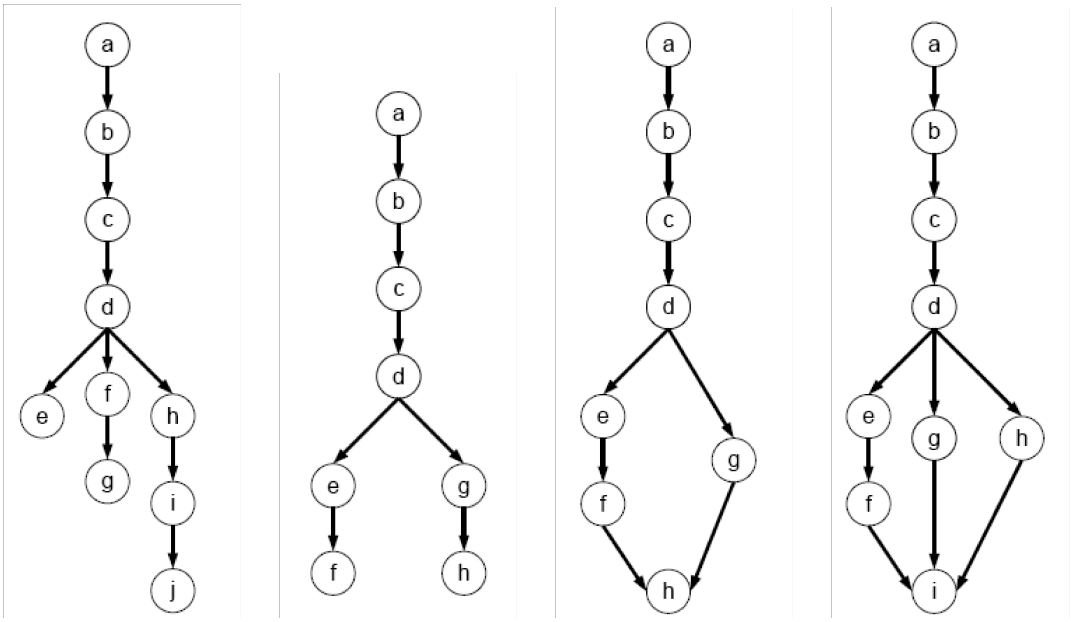
συνόλων δεδομένων. Το μέγεθος του εγγράφου είναι 111 MB, ενώ οι κόμβοι του, που ο αριθμός τους είναι περίπου 1 εκατομμύριο, σημαίνονται από 74 διαφορετικά στοιχεία. Σε αντίθεση με το αρχείο ελέγχου treebank.xml, έχει βάθος μόλις 7, ενώ δεν περιέχει καθόλου αναδρομή στοιχείων. Επίσης, περιέχει συνθετικά δεδομένα, τα οποία μπορούν να δημιουργηθούν με βάση κάποιο DTD, και όχι δεδομένα του πραγματικού κόσμου. Πρόκειται προφανώς για ένα αρχείο ελέγχου με τελείως διαφορετικά χαρακτηριστικά από το treebank.xml.

6.1.2 Τύπος των Ερωτήσεων Ελέγχου

Στο πλαίσιο της πειραματικής αξιολόγησης τρέξαμε από 7 ερωτήσεις προτύπου μονοπατιού τόσο ολικής όσο και μερικής δομής πάνω στα δύο αρχεία ελέγχου που περιγράψαμε προηγουμένως. Οι ερωτήσεις στο αρχείο treebank.xml αναφέρονται και ως $Q_{t,1}$ - $Q_{t,7}$, ενώ οι ερωτήσεις στο αρχείο xmark.xml αναφέρονται και ως $Q_{x,1}$ - $Q_{x,7}$. Οι ερωτήσεις $Q_{t,1}$ - $Q_{t,3}$ και $Q_{x,1}$ - $Q_{x,3}$ είναι ολικώς καθορισμένης δομής, ενώ οι ερωτήσεις $Q_{t,4}$ - $Q_{t,7}$ και $Q_{x,4}$ - $Q_{x,7}$ είναι μερικώς καθορισμένης δομής. Το Σχήμα 6.1 περιέχει τους τύπους των ερωτήσεων $Q_{t,1}$ - $Q_{t,3}$, το Σχήμα 6.2 τους τύπους των ερωτήσεων $Q_{x,1}$ - $Q_{x,3}$, το Σχήμα 6.3 τους τύπους των ερωτήσεων $Q_{t,4}$ - $Q_{t,7}$, και το Σχήμα 6.4 τους τύπους των ερωτήσεων $Q_{x,4}$ - $Q_{x,7}$. Οι σημάνσεις των κόμβων των ερωτήσεων επιλέχθηκαν κατάλληλα ώστε οι προκύπτουσες ερωτήσεις να έχουν νόημα στο αντίστοιχο ερωτώμενο αρχείο. Είναι προφανές ότι το σύνολο των ερωτήσεων καλύπτει το πλήρες φάσμα των μερικών ερωτήσεων μονοπατιού, από τις (εκφυλισμένες) ερωτήσεις μονοπατιού μερικής δομής μέχρι και τις μερικώς προσδιορισμένες ερωτήσεις που έχουν μη δενδρική μορφή (μορφή γράφου).



Σχήμα 6.3: (α) Ο τύπος της ερώτησης $Q_{t,4}$, (β) ο τύπος της ερώτησης $Q_{t,5}$, (γ) ο τύπος της ερώτησης $Q_{t,6}$, και (δ) ο τύπος της ερώτησης $Q_{t,7}$.



Σχήμα 6.4: (α) Ο τύπος της ερώτησης $Q_{x,4}$, (β) ο τύπος της ερώτησης $Q_{x,5}$, (γ) ο τύπος της ερώτησης $Q_{x,6}$, και (δ) ο τύπος της ερώτησης $Q_{x,7}$.

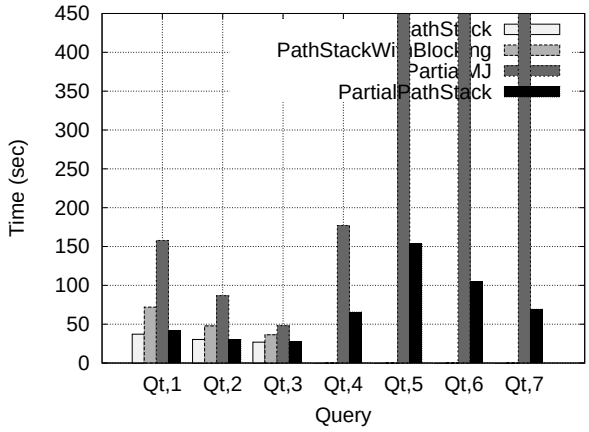
6.2 Παράθεση των Αποτελεσμάτων του Πειραματικού Ελέγχου

6.2.1 Χρονική Διάρκεια Εκτέλεσης των Αλγορίθμων

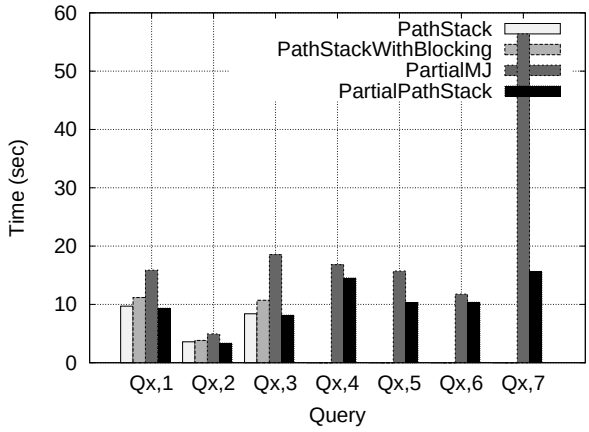
Για καθεμία από τις επτά ερωτήσεις $Q_{t,1}$ - $Q_{t,7}$ πάνω στο treebank.xml και τις επτά ερωτήσεις $Q_{x,1}$ - $Q_{x,7}$ πάνω στο αρχείο xmark.xml μετρήσαμε τη χρονική διάρκεια (σε δευτερόλεπτα) εκτέλεσης των αλγορίθμων PartialMJ και PartialPathStack, από τη χρονική στιγμή που καλείται ο κώδικας του αλγορίθμου μέχρι τη χρονική στιγμή που επιστρέφεται το σύνολο των αποτελεσμάτων. Ειδικά για τις ερωτήσεις $Q_{t,1}$ - $Q_{t,3}$ και $Q_{x,1}$ - $Q_{x,3}$ των οποίων η δομή είναι ολικώς προσδιορισμένη μετρήθηκε και η χρονική διάρκεια εκτέλεσης των αλγορίθμων PathStack και PathStackWithBlocking. Στο διάγραμμα του Σχήματος 6.5 παρουσιάζονται συνοπτικά τα αποτελέσματα αυτά για τις επτά ερωτήσεις πάνω στα δεδομένα του treebank.xml, ενώ στο διάγραμμα του Σχήματος 6.6 παρουσιάζονται συνοπτικά τα αποτελέσματα αυτά για τις επτά ερωτήσεις πάνω στα δεδομένα του xmark.xml.

6.2.2 Συνολικός αριθμός κόμβων εισόδου

Επιπρόσθετα στη χρονική διάρκεια εκτέλεσης των υλοποιημένων αλγορίθμων μετρήθηκε και το μέγεθος της εισόδου, δηλαδή ο συνολικός αριθμός των κόμβων εισόδου που βρίσκονται αρχικά στις ροές T_q. Η αξία ενός τέτοιου μέτρου σύγκρισης έγκειται στη συνεισφορά του στην πολυπλοκότητα των εξεταζόμενων αλγορίθμων, όπως θα δείξουμε στη συνέχεια. Έτσι, ο Πίνακας 6.1 περιέχει το συνολικό αριθμό κόμβων εισόδου των ερωτήσεων ολικής και μερικής



Σχήμα 6.5: Χρόνος (σε δευτερόλεπτα) για την αποτίμηση των επτά ερωτήσεων στα TreeBank δεδομένα με χρήση των διάφορων αλγορίθμων.



Σχήμα 6.6: Χρόνος (σε δευτερόλεπτα) για την αποτίμηση των επτά ερωτήσεων στα XMark δεδομένα με χρήση των διάφορων αλγορίθμων.

δομής $Q_{t,1}$ - $Q_{t,7}$ πάνω στο treebank.xml, ενώ ο Πίνακας 6.2 περιέχει το συνολικό αριθμό κόμβων εισόδου των ερωτήσεων ολικής και μερικής δομής $Q_{x,1}$ - $Q_{x,7}$ πάνω στο xmark.xml.

Ερώτηση	**Συνολικός Αριθμός Κόμβων Εισόδου**
$Q_{t,1}$	641,811
$Q_{t,2}$	727,531
$Q_{t,3}$	900,380
$Q_{t,4}$	1,317,518
$Q_{t,5}$	1,317,518
$Q_{t,6}$	1,625,086
$Q_{t,7}$	1,403,238

Πίνακας 6.1: Συνολικός αριθμός κόμβων εισόδου για τις ερωτήσεις μονοπατιού ολικής και μερικής δομής πάνω στα δεδομένα του TreeBank.

Ερώτηση	Συνολικός Αριθμός Κόμβων Εισόδου
$Q_{x,1}$	231,848
$Q_{x,2}$	85,395
$Q_{x,3}$	181,115
$Q_{x,4}$	256,759
$Q_{x,5}$	252,661
$Q_{x,6}$	252,661
$Q_{x,7}$	262,410

Πίνακας 6.2: Συνολικός αριθμός κόμβων εισόδου για τις ερωτήσεις μονοπατιού ολικής και μερικής δομής πάνω στα δεδομένα του XMark.

6.2.3 Αριθμός τελικών και ενδιάμεσων αποτελεσμάτων

Τέλος, μετρήθηκε και ο συνολικός αριθμός των επιστρεφομένων αποτελεσμάτων για κάθε μία από τις επτά ερωτήσεις πάνω στα δύο αρχεία ελέγχου. Ειδικά για τον αλγόριθμο PartialMJ, ο οποίος πρώτα βρίσκει τις απαντήσεις σε καθένα μονοπάτι του επικαλύπτοντος δένδρου της ερώτησης, έπειτα συγχωνεύει αυτές τις απαντήσεις και τέλος τις φιλτράρει, μετρήθηκε και ο συνολικός αριθμός των ενδιάμεσων αποτελεσμάτων, που ορίζεται ως το άθροισμα των απαντήσεων που επιστρέφονται από τα διάφορα μονοπάτια του επικαλύπτοντος δένδρου. Προφανώς, αυτό το μέτρο ελέγχου έχει νόημα μόνο για μερικώς και όχι ολικώς καθορισμένες ερωτήσεις, ώστε να υπάρχουν περισσότερα του ενός μονοπάτια στο επικαλύπτον δένδρο της ερώτησης.

Ερώτηση	Αριθμός Αποτελεσμάτων
$Q_{t,1}$	586,204
$Q_{t,2}$	129,734
$Q_{t,3}$	0

Πίνακας 6.3: Αριθμός αποτελεσμάτων για τις ερωτήσεις μονοπατιού ολικής δομής πάνω στα δεδομένα του TreeBank.

Ερώτηση	Αριθμός Αποτελεσμάτων
$Q_{x,1}$	9,990
$Q_{x,2}$	1,941
$Q_{x,3}$	18,579

Πίνακας 6.4: Αριθμός αποτελεσμάτων για τις ερωτήσεις μονοπατιού ολικής δομής πάνω στα δεδομένα του XMark.

Ο Πίνακας 6.3 περιέχει τον αριθμό των επιστρεφόμενων αποτελεσμάτων των ερωτήσεων ολικής δομής $Q_{t,1}$ - $Q_{t,3}$ πάνω στο treebank.xml, ενώ ο Πίνακας 6.4 περιέχει τον αριθμό των επιστρεφόμενων αποτελεσμάτων των ερωτήσεων ολικής δομής $Q_{x,1}$ - $Q_{x,3}$ πάνω στο xmark.xml. Τέλος, ο Πίνακας 6.5 περιέχει τον αριθμό των επιστρεφόμενων καθώς και των ενδιάμεσων αποτελεσμάτων των ερωτήσεων μερικής δομής $Q_{t,4}$ - $Q_{t,7}$ πάνω στο treebank.xml, ενώ ο Πίνακας 6.6 περιέχει τον αριθμό των επιστρεφόμενων καθώς και των ενδιάμεσων αποτε-

λεσμάτων των ερωτήσεων μερικής δομής $Q_{x,4}$ - $Q_{x,7}$ πάνω στο xmark.xml.

Ερώτηση	Αριθμός Αποτελεσμάτων	Αριθμός Ενδιάμεσων Αποτελεσμάτων
$Q_{t,4}$	2,190	137,154
$Q_{t,5}$	241,419	930,476
$Q_{t,6}$	0	930,746
$Q_{t,7}$	13	1,035,242

Πίνακας 6.5: Αριθμός αποτελεσμάτων και αριθμός ενδιάμεσων αποτελεσμάτων για τις ερωτήσεις μονοπατιού μερικής δομής πάνω στα δεδομένα του TreeBank.

Ερώτηση	Αριθμός Αποτελεσμάτων	Αριθμός Ενδιάμεσων Αποτελεσμάτων
$Q_{x,4}$	0	2,207
$Q_{x,5}$	5,922	4,746
$Q_{x,6}$	1,764	1,937
$Q_{x,7}$	26,655	36,702

Πίνακας 6.6: Αριθμός αποτελεσμάτων και αριθμός ενδιάμεσων αποτελεσμάτων για τις ερωτήσεις μονοπατιού μερικής δομής πάνω στα δεδομένα του XMark.

6.3 Σχολιασμός Πειραματικών Αποτελεσμάτων

Ακολουθεί ο σχολιασμός των πειραματικών αποτελεσμάτων για ερωτήσεις τόσο ολικής όσο και μερικής δομής.

6.3.1 Ερωτήσεις Μονοπατιού Ολικώς Καθορισμένης Δομής

Οι ερωτήσεις $Q_{t,1}$ - $Q_{t,3}$ πάνω στο treebank.xml καθώς και οι ερωτήσεις $Q_{x,1}$ - $Q_{x,3}$ πάνω στο xmark.xml είναι όπως προαναφέρθηκε ολικώς καθορισμένης δομής και ως εκ τούτου μπορούν να απαντηθούν με οποιονδήποτε από τους τέσσερις αλγορίθμους PathStack, PathStackWithBlocking, PartialMJ και PartialPathStack. Μετά την εκτέλεσή τους προέκυψε μία σειρά από συμπεράσματα.

- *Εξάρτηση του χρόνου εκτέλεσης όλων των αλγορίθμων από (α) το μέγεθος της εισόδου, δηλαδή το συνολικό αριθμό των κόμβων στις ροές εισόδου της ερώτησης, και (β) το μέγεθος της εξόδου, δηλαδή τον αριθμό των απαντήσεων που επιστρέφονται.*

  Μία τέτοια εξάρτηση προβλέπεται από την ανάλυση όλων των αλγορίθμων, όπου αποδεικνύεται ότι η CPU πολυπλοκότητα των αλγορίθμων είναι ανάλογη του αθροίσματος του μεγέθους εισόδου με το μέγεθος εξόδου. Πράγματι, όταν το άθροισμα αυτό είναι μεγάλο, τότε ο χρόνος εκτέλεσης ακόμα και για το σχετικά απλό αλγόριθμο PathStack είναι σημαντικά μεγάλος (άνω των 30 sec) όπως προκύπτει από τους χρόνους εκτέλεσης των ερωτήσεων $Q_{t,1}$ - $Q_{t,3}$. Αντίθετα, για τις ερωτήσεις $Q_{x,1}$ - $Q_{x,3}$, όπου το εν λόγω άθροισμα είναι πολύ μικρότερο, οι χρόνοι εκτέλεσης για τον PathStack μειώνονται σημαντικά

και μπορούν να φτάσουν ακόμα και τα 3.5 sec. Τελείως όμοια αποτελέσματα ισχύουν και για τους άλλους τρεις αλγορίθμους.

- *Χειρότερη συμπεριφορά του αλγόριθμου PathStackWithBlocking από τον αλγόριθμο PathStack.*

  Ο αλγόριθμος PathStackWithBlocking πέρα του ότι κάνει χρήση πιο πολύπλοκων δομών έχει το μειονέκτημα ότι κάθε φορά που κάποιος κόμβος εξάγεται από μία στοίβα, μία σειρά από ενέργειες γίνονται στις λίστες SList και IList των κόμβων της γονεϊκής του στοίβας ακόμα και αν ο απομακρυνόμενος κόμβος δε συνεισφέρει καθόλου στις τελικές απαντήσεις. Έτσι, υπάρχει παρατηρείται μία αύξηση στη διάρκεια εκτέλεσης του αλγορίθμου PathStackWithBlocking. Στην περίπτωση των ερωτήσεων $Q_{x,1}$ - $Q_{x,3}$ παρατηρούμε ότι η επιπλέον χρονική επιβάρυνση λόγω του μπλοκαρίσματος δεν είναι ιδιαίτερα σημαντική και κυμαίνεται στο 10-20% της χρονικής διάρκειας εκτέλεσης δίχως μπλοκάρισμα. Αντίθετα, στην περίπτωση των ερωτήσεων $Q_{t,1}$ - $Q_{t,3}$ πάνω στα δεδομένα treebank.xml παρατηρούμε μία σαφώς χειρότερη επιβάρυνση που μπορεί να φτάσει ακόμα και το 100% για την περίπτωση της ερώτησης $Q_{t,1}$. Η κακή αυτή συμπεριφορά μπορεί να αποδοθεί σε δύο κυρίως λόγους:

  1. Επειδή το αρχείο treebank.xml παρουσιάζει πολύ μεγάλο βάθος, κάθε χρονική στιγμή πολλοί κόμβοι μπορούν να βρίσκονται στις στοίβες, οπότε η προηγηθείσα διαδικασία με τις λίστες SList και IList μπορεί να αφορά πολύ περισσότερους κόμβους σε σχέση με την αντίστοιχη διαδικασία στο αρχείο xmark.xml που έχει μικρότερο βάθος.
  2. Ειδικά για τις ερωτήσεις $Q_{t,1}$ και $Q_{t,2}$ μπορούμε να ισχυριστούμε ότι ένας σημαντικός παράγοντας επιβάρυνσης είναι το ίδιο το μέγεθος της λίστας των αποτελεσμάτων. Πράγματι, σε αντίθεση με τον PathStack, ο PathStackWithBlocking κρατάει στην μνήμη τη λίστα με όλα τα αποτελέσματα και όταν αυτά γίνουν πάρα πολλά για να τα χωρέσει η κύρια μνήμη, έχουμε πρόσθετες πράξεις εισόδου/εξόδου μεταξύ μνήμης και σκληρού δίσκου. Έτσι, παρατηρούμε ότι για την ερώτηση $Q_{t,1}$ που έχει ως λύση 586,204 απαντήσεις ο χρόνος εκτέλεσης με μπλοκάρισμα είναι σχεδόν διπλάσιος από το χρόνο εκτέλεσης δίχως μπλοκάρισμα.

- *Πολύ παρόμοια συμπεριφορά των αλγορίθμων PathStack και PartialPathStack.*

  Παρατηρούμε ότι και για τις έξι ερωτήσεις $Q_{t,1}$ - $Q_{t,3}$ και $Q_{x,1}$ - $Q_{x,3}$ ο χρόνος εκτέλεσης των δύο αλγορίθμων είναι σχεδόν ταυτόσημος. Αυτό είναι απολύτως αναμενόμενο λόγω του ότι για ερωτήσεις μονοπατιού ολικής δομής ο αλγόριθμος PartialPathStack εκφυλίζεται στον PathStack.

- *Χειρότερη συμπεριφορά του αλγόριθμου PartialMJ από τον αλγόριθμο PathStackWithBlocking.*

  Ο αλγόριθμος PartialMJ έχει ως βάση τον αλγόριθμο PathStackWithBlocking για το λόγο ότι πρέπει τα αποτελέσματα στα διάφορα μονοπάτια του επικαλύπτοντος δέν-

δρου της ερώτησης να είναι ταξινομημένα για τη διαδικασία της συγχώνευσης που θα ακολουθήσει. Μετά το στάδιο της συγχώνευσης ακολουθεί και το στάδιο του φιλτραρίσματος όπου τα αποκτηθέντα αποτελέσματα ελέγχονται για το αν ικανοποιούν τις δομικές σχέσεις που δεν υπάρχουν στο επικαλύπτον δένδρο αλλά υπάρχουν στην αρχική ερώτηση. Όταν ο αριθμός των αποτελεσμάτων του στάδιου της συγχώνευσης είναι μεγάλος, τότε αναμένουμε μεγάλες χρονικές καθυστερήσεις στο στάδιο του φιλτραρίσματος και κατ΄ επέκταση μεγαλύτερους χρόνους εκτέλεσης. Έτσι, για τις ερωτήσεις $Q_{t,1}$, $Q_{t,2}$, $Q_{x,3}$ όπου ο αριθμός των αποτελεσμάτων είναι (σχετικά) μεγάλος το στάδιο του φιλτραρίσματος καταλαμβάνει αρκετό χρόνο, εξ ου και η σαφώς χειρότερη συμπεριφορά του PartialMJ.

6.3.2 Ερωτήσεις Μονοπατιού Μερικώς Καθορισμένης Δομής

Οι ερωτήσεις $Q_{t,4}$ - $Q_{t,7}$ πάνω στο treebank.xml καθώς και οι ερωτήσεις $Q_{x,4}$ - $Q_{x,7}$ πάνω στο xmark.xml είναι όπως προαναφέρθηκε μερικώς καθορισμένης δομής και ως εκ τούτου μπορούν να απαντηθούν μόνο από τους δύο αλγορίθμους PartialMJ και PartialPathStack. Μετά την εκτέλεσή τους προέκυψε μία σειρά από συμπεράσματα.

- *Εξάρτηση του χρόνου εκτέλεσης όλων των αλγορίθμων από (α) το μέγεθος της εισόδου, δηλαδή το συνολικό αριθμό των κόμβων στις ροές εισόδου της ερώτησης, και (β) το μέγεθος της εξόδου, δηλαδή τον αριθμό των απαντήσεων που επιστρέφονται.*

  Αν και η εξάρτηση δεν είναι ακριβώς γραμμική ως προς το άθροισμα των μεγεθών εισόδου και εξόδου, είναι ωστόσο σαφές ότι για μεγάλα μεγέθη εισόδου και/ή εξόδου ο χρόνος εκτέλεσης μεγαλώνει. Έτσι, για τις ερωτήσεις $Q_{t,4}$ - $Q_{t,7}$ οι χρόνοι εκτέλεσης και για τους δύο αλγόριθμους είναι μεγάλοι και ξεπερνούν τα 50 sec, ενώ για τις ερωτήσεις $Q_{x,4}$ - $Q_{x,7}$ όπου τα μεγέθη εισόδου/εξόδου είναι πιο μικρά οι χρόνοι εκτέλεσης κυμαίνονται γύρω στα 10-20 sec.

- *Εξάρτηση του PartialMJ από αριθμό ενδιάμεσων αποτελεσμάτων.*

  Ένα μειονέκτημα του PartialMJ έχει να κάνει με το ότι δημιουργούνται πολλά ενδιάμεσα αποτελέσματα, από τα οποία μόνο κάποια συνεισφέρουν στις τελικές λύσεις. Στην περίπτωση των ερωτήσεων $Q_{t,4}$ - $Q_{t,7}$ αλλά και της ερώτησης $Q_{x,7}$ όπου ο αριθμός των ενδιάμεσων αποτελεσμάτων υπερβαίνει τον αριθμό των τελικών αποτελεσμάτων παρατηρούμε μία σημαντική υποβάθμιση της συμπεριφοράς του PartialMJ σε σχέση με τον PartialPathStack που οφείλεται στην επιπλέον χρονική επιβάρυνση για τη δημιουργία των ενδιάμεσων αποτελεσμάτων από τα οποία μόνο ένας μικρός αριθμός συμμετέχει στην τελική λύση. Αντίθετα, ο PartialPathStack δε δημιουργεί τέτοιες αχρείαστες λύσεις και ως εκ τούτου αποφεύγει αυτές τις καθυστερήσεις.

- *Χειρότερη συμπεριφορά του αλγορίθμου PartialMJ από τον αλγόριθμο PartialPathStack.*

  Πράγματι, για όλες τις ερωτήσεις παρατηρούμε ότι ο χρόνος εκτέλεσης του PartialMJ είναι μεγαλύτερος από το χρόνο εκτέλεσης του PartialPathStack, φτάνοντας σε ορισμένες περιπτώσεις ακόμα και το 500%. Παράγοντες που παίζουν ρόλο είναι αφενός ο

αριθμός των ενδιάμεσων αποτελεσμάτων, όπως αναφέρθηκε και προηγουμένως, αλλά και ο αριθμός των τελικών αποτελεσμάτων, αφού στο τελικό στάδιο του φιλτραρίσματος τα διερχόμαστε όλα για να ελέγξουμε τις δομικές σχέσεις που δεν αφαιρέθηκαν από το γράφο της ερώτησης για το σχηματισμό του επικαλύπτοντος δένδρου. Μάλιστα, όσο πιο πολλές είναι οι δομικές σχέσεις που έχουν απομακρυνθεί τόσο περισσότερο θα διαρκέσει το στάδιο του φιλτραρίσματος και τόσο μεγαλύτερη θα είναι κατ΄ επέκταση η χρονική επιβάρυνση. Επίσης, όσο περισσότερα είναι τα αποτελέσματα τόσο περισσότερες πράξεις εισόδου/εξόδου αναμένονται μεταξύ μνήμης και σκληρού δίσκου λόγω της αδυναμίας της κύριας μνήμης να χωρέσει όλα τα αποτελέσματα. Κάτι τέτοιο είναι φανερό από τις ερωτήσεις $Q_{t,5}$ - $Q_{t,7}$ όπου ο αριθμός των ενδιάμεσων αποτελεσμάτων ξεπερνά τις 900,000 και ως εκ τούτου η κύρια μνήμη αδυνατεί να διατηρήσει όλο αυτό το φορτίο συμμετέχοντας σε συχνές ανταλλαγές (swap) δεδομένων με το σκληρό δίσκο.

Κεφάλαιο 7

Επίλογος

Στην πρώτη ενότητα αυτού του κεφαλαίου επιχειρείται μία σύνοψη της διπλωματικής εργασίας και δίνονται τα γενικά συμπεράσματα. Στη δεύτερη ενότητα αναφέρονται ορισμένες προκλήσεις/κατευθύνσεις για τη μελλοντική έρευνα.

7.1 Σύνοψη και Συμπεράσματα

Στα πλαίσια της παρούσας διπλωματικής εργασίας μελετήθηκαν οι ερωτήσεις μονοπατιού με μερικό καθορισμό της δομής, δηλαδή ερωτήσεις μονοπατιού όπου οι δομικές σχέσεις μεταξύ των διάφορων κόμβων της ερώτησης μπορούν να είναι μη προσδιορισμένες. Βασικό κίνητρο για τη μελέτη αυτής της κατηγορίας ερωτήσεων ήταν αφενός η περίπτωση όπου η δομή ενός XML εγγράφου είναι μερικώς προσδιορισμένη κι αφετέρου η περίπτωση όπου θέλουμε να εξάγουμε πληροφορία από μία κατανεμημένη XML βάση δεδομένων, όπου το σχήμα κάθε επιμέρους εγγράφου της βάσης δεδομένων είναι διαφορετικό. Και στις δύο αυτές περιπτώσεις είναι απαραίτητο να χαλαρώσουμε το περιοριστικό στοιχείο της ολικής διάταξης των κόμβων της ερώτησης και να επιτρέψουμε το μερικό καθορισμό της δομής.

Συγκεκριμένα, στα πλαίσια αυτής της διπλωματικής εργασίας ορίζεται μία γλώσσα XML ερωτήσεων μονοπατιού μερικής δομής. Ορίστηκε ένα σύνολο από αληθείς και πλήρεις κανόνες παραγωγής που χαρακτηρίζουν πλήρως την παραγωγή δομικών σχέσεων και που η επανειλημμένη εφαρμογή τους οδηγεί στην ισοδύναμη πλήρη μορφή της αρχικής ερώτησης. Εισήχθηκε η έννοια της μη ικανοποιησιμότητας, αφού σε αντίθεση με τις ερωτήσεις μονοπατιού ολικής διάταξης, οι ερωτήσεις μερικής δομής μπορούν να είναι μη ικανοποιήσιμες. Δόθηκαν μάλιστα και οι ικανές και αναγκαίες συνθήκες για μη ικανοποιησιμότητα. Επίσης, εξετάστηκε η έννοια των πλεονάζοντων κόμβων, δηλαδή κόμβων που μπορούν να απομακρυνθούν από την ερώτηση χωρίς να επηρεάζεται η σημασία της ερώτησης, παρέχοντας ταυτόχρονα κάποια πρότυπα που απλοποιούν την εύρεση τέτοιων κόμβων. Τέλος, δείξαμε πώς μπορούμε να μετασχηματίσουμε την αρχική ερώτηση μονοπατιού στην ισοδύναμη κανονική της μορφή, που είναι ένας κατευθυνόμενος ακυκλικός γράφος, και την οποία εκμεταλλευόμαστε στους αλγόριθμους αποτίμησης.

Επίσης, στην εργασία αυτή αντιμετωπίσαμε το πρόβλημα της αποδοτικής αποτίμησης ερωτή-

σεων μονοπατιού μερικής δομής. Αρχικά αναπτύχθηκε ο βασισμένος σε στοίβες αλγόριθμος PartialMJ. Ο PartialMJ εξάγει ένα μη επικαλυπτόμενο δένδρο από την κανονική μορφή της ερώτησης. Χρησιμοποιεί μία επέκταση του PathStack για να υπολογίσει τα αποτελέσματα όλων των μονοπατιών από τη ρίζα προς τα φύλλα του μη επικαλυπτόμενου δένδρου. Αυτά τα αποτελέσματα παράγονται ταξινομημένα ως προς τη ρίζα−προς−φύλλο διάταξη του XML δένδρου και συγχωνεύονται για να υπολογίσουμε την τελική απάντηση στην αρχική ερώτηση. Ο PartialMJ μπορεί όμως να δημιουργήσει πολλά ενδιάμεσα αποτελέσματα για τα διάφορα μονοπάτια από τη ρίζα προς τα φύλλα του μη επικαλυπτόμενου δένδρου που δε συνεισφέρουν στην τελική λύση. Για να ξεπεράσουμε αυτό το πρόβλημα, αναπτύξαμε έναν καινούριο ολιστικό βασισμένο σε στοίβες αλγόριθμο, τον PartialPathStack, ο οποίος εκμεταλλεύεται μία τοπολογική διάταξη των κόμβων της ερώτησης στην κανονικής της μορφή, και προσπαθεί να ταιριάξει την κανονική μορφή συνολικά πάνω στο XML δένδρο. Αναλύσαμε την πολυπλοκότητα του PartialPathStack και δείξαμε ότι είναι ανεξάρτητος από ενδιάμεσα αποτελέσματα.

Προκειμένου να βγάλουμε χρήσιμα συμπεράσματα για τους διάφορους προτεινόμενους στα πλαίσια της διπλωματικής αλγόριθμους, προβήκαμε και στην υλοποίησή τους στην αντικειμενοστρεφή γλώσσα προγραμματισμού C++, δίνοντας ιδιαίτερο βάρος στις τεχνικές λεπτομέρειες, ενώ αξιολογήσαμε πειραματικά τους αλγόριθμους αυτούς με τη βοήθεια διάφορων ερωτήσεων σε benchmarks XML αρχεία. Η πειραματική αξιολόγηση έδειξε την εξάρτηση του PartialMJ από τα ενδιάμεσα αποτελέσματα και την ανωτερότητα του PartialPathStack, επιβεβαιώνοντας στην ουσία τα θεωρητικά αποτελέσματα.

Τέλος, αν και το κυρίως αντικείμενο της παρούσας εργασίας σχετίζεται με τις XML ερωτήσεις μονοπατιού μερικής δομής, εξετάστηκαν μία σειρά από παρεμφερή ή βοηθητικά θέματα, που αποτέλεσαν στην ουσία το υπόβαθρο για την ανάπτυξη των κύριων θεμάτων. Έτσι από το μία πλευρά δόθηκε ένα σύντομο εγχειρίδιο της XML καθώς και των γλωσσών ερωτήσεων XQuery και XPath. Από την άλλη πλευρά παρουσιάστηκε το θεωρητικό υπόβαθρο και οι κύριες ιδέες για τους αλγόριθμους αποτίμησης που προτείνονται στη διπλωματική εργασία, κάνοντας ταυτόχρονα μία εκτεταμένη βιβλιογραφική αναφορά πάνω σε αυτούς. Αρχικά, έγινε αναφορά σε ορισμένες προκαταρκτικές έννοιες, όπως στο μοντέλο δεδομένων της γλώσσας XML, σε δύο σχήματα κωδικοποίησης θέσης σε XML έγγραφα, συγκεκριμένα στην κωδικοποίηση θέσης με χρήση ανεστραμμένων ευρετηρίων και στο σχήμα εκτεταμένης κωδικοποίησης Dewey, και σε διάφορους τύπους ευρετηρίων για γρηγορότερη επεξεργασία ερωτήσεων. Έπειτα παρουσιάστηκαν μέθοδοι για αποτίμηση ερωτήσεων με δυαδικές δομικές σχέσεις: η μία προσέγγιση βασίστηκε σε συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων, ενώ η άλλη σε αποδοτικές δυαδικές δομικές σχέσεις. Στη συνέχεια, παρουσιάστηκαν και αναλύθηκαν εκτενώς οι αλγόριθμοι PathStack και TwigStack για ερωτήσεις μονοπατιού και δενδρικής μορφής αντίστοιχα. Ειδικά ο αλγόριθμος PathStack αποτέλεσε το υπόβαθρο γύρω από τον οποίο αναπτύχθηκαν οι αλγόριθμοι αποτίμησης ερωτήσεων μονοπατιού μερικής δομής και δε θα ήταν υπερβολή να τον χαρακτηρίσουμε ως τον πυρήνα της παρούσας διπλωματικής εργασίας. Κατόπιν, αναφέρθηκαν διάφορες τεχνικές για βελτιώσεις των αλγορίθμων PathStack και TwigStack, ενώ έγινε και μία αναφορά σε τεχνικές που ακολουθούν το σχήμα εκτεταμένης κωδικοποίησης Dewey για την αναπαράσταση θέσης.

Ελπίζουμε ότι η συγκεκριμένη οργάνωση του τόμου καταφέρνει να παρουσιάσει με σαφήνεια, πληρότητα και απλότητα το θέμα των ερωτήσεων μονοπατιού μερικής δομής σε κάθε ενδιαφερόμενο αναγνώστη, αρχάριο και μη.

7.2 Μελλοντικές Προκλήσεις

Στα πλαίσια της παρούσας διπλωματικής εργασίας ρίξαμε το βάρος στις ερωτήσεις μονοπατιού μερικής δομής, δηλαδή σε ερωτήσεις όπου όλοι οι κόμβοι τους ανήκουν στο ίδιο μονοπάτι του XML δένδρου και οι δομικές σχέσεις μεταξύ των κόμβων μπορεί να είναι μη καθορισμένες. Οι ερωτήσεις αυτές αποτελούν μία φυσική επέκταση των ερωτήσεων μονοπατιού ολικής δομής, τις οποίες προφανώς και περιλαμβάνουν. Αν και μία πολύ χρήσιμη κλάση ερωτήσεων, στις περισσότερες πραγματικές εφαρμογές μας ενδιαφέρουν ερωτήσεις δενδρικού προτύπου και όχι απλά προτύπου μονοπατιού. Είναι λοιπόν πολύ λογικό κατά τρόπο ανάλογο των ερωτήσεων μονοπατιού να θέλουμε να επεκτείνουμε την κλάση των δενδρικών ερωτήσεων ολικής δομής στην ευρύτερη κλάση των δενδρικών ερωτήσεων με μερικώς καθορισμένη δομή για να αντιμετωπίσουμε την περίπτωση που είτε δεν έχουμε πλήρη γνώση του XML εγγράφου στο οποίο αναφέρεται η ερώτηση ή που ρωτάμε μία κατανεμημένη XML βάση δεδομένων όπου το κάθε έγγραφο έχει διαφορετικό σχήμα ή οργάνωση της πληροφορίας.

Πάνω σε αυτό το πρόβλημα έχουν ήδη εκδώσει δημοσιεύσεις οι Θοδωρής Δαλαμάγκας και Στέφανος Σουλδάτος του Εργαστηρίου Συστημάτων Βάσεων Γνώσεων και Δεδομένων μαζί με άλλους συνάδελφούς τους ([15], [14], [16]). Η προσέγγισή τους συνίσταται στο σχεδιασμό μίας γλώσσας ερωτήσεων που επιτρέπει το μερικό καθορισμό των δομικών σχέσεων σε δενδρικές ερωτήσεις (*ερωτήσεις δενδρικού προτύπου μερικής δομής*). Ο σχηματισμός και η αποτίμηση τέτοιων ερωτήσεων βασίζεται στους *γράφους διαστάσεων*, οι οποίοι συμπυκνώνουν τη δομή των XML δέντρων πάνω στα οποία θέλουμε να κάνουμε ερωτήσεις, χρησιμοποιώντας *διαστάσεις*, δηλαδή σύνολα από σημασιολογικά παρόμοιους κόμβους του XML δένδρου. Οι γράφοι διαστάσεων παρέχουν την απαραίτητη σημασιολογία για το σχηματισμό ερωτήσεων, θέτουν το πλαίσιο για την ερώτηση πηγών με δομικές διαφορές ή ακόμα και ασυνέπειες, και υποστηρίζουν την αποτίμηση και βελτιστοποίηση των ερωτήσεων. Επίσης, οι παραπάνω ερευνητές εισάγουν την έννοια των*δέντρων διαστάσεων*, τα οποία είναι ισοδύναμα με τη μερική ερώτηση σε ένα καθορισμένο γράφο διαστάσεων. Συνοπτικά, η διαδικασία που περιγράφουν είναι αρχικά η εφαρμογή κανόνων παραγωγής πάνω στην ερώτηση ώστε να παραχθεί η πλήρης μορφή του, στη συνέχεια η αναζήτηση μονοπατιών στο γράφο διαστάσεων που ταιριάζουν με τα μονοπάτια της ερώτησης (οπότε και σχηματίζονται τα δέντρα διαστάσεων), και η μετάφραση των δέντρων διαστάσεων σε XPath ερωτήσεις. Τέλος, ασχολούνται και με την έννοια της *συμπερίληψης (containment)* μίας ερώτησης από μία άλλη, και φτάνουν σε ορισμένα πολύ χρήσιμα θεωρήματα για τον έλεγχο της συμπερίληψης.

Η προσπάθεια των παραπάνω ερευνητών είναι σημαντική, αφού υιοθετεί μία κατεύθυνση για τις δενδρικές ερωτήσεις μερικής δομής, η οποία τυποποιεί το πρόβλημα της συγκεκριμένης κλάσης ερωτήσεων και οδηγεί σε σημαντικά συμπεράσματα και θεωρήματα. Μία μελλοντική κατεύθυνση έρευνας που μέχρι τη συγγραφή της παρούσας διπλωματικής δεν έχει διερευνηθεί

επαρκώς (και στην οποία συμμετέχει ενεργά το Εργαστήριο Συστημάτων Βάσεων Γνώσεων και Δεδομένων) είναι η ανάπτυξη αποδοτικών αλγόριθμων αποτίμησης ερωτήσεων δενδρικού προτύπου μερικής δομής. Κατ΄ αναλογία με τον PartialPathStack για το πρόβλημα των ερωτήσεων μονοπατιού μερικής δομής, ένας τέτοιος αλγόριθμος θα ήταν ολιστικός, βασισμένος σε στοίβες και λογικά θα αποτελούσε μία επέκταση του TwigStack, έτσι ώστε να λαμβάνει μέριμνα για τις περιπτώσεις όπου οι δομικές σχέσεις των κόμβων της δενδρικής ερώτησης δεν είναι πλήρως καθορισμένες.

Η εύρεση ενός τέτοιου αποδοτικού αλγόριθμου θα ερχόταν ως φυσική συνέχεια του PartialPathStack για δενδρικές ερωτήσεις και θα ολοκλήρωνε την οικογένεια των ολιστικών αλγόριθμων των βασισμένων σε στοίβα για την αποτίμηση γενικής μορφής XML ερωτήσεων.

Βιβλιογραφία

[1] S. Al-Khalifa, H. Jagadish, N. Koudas, J.M. Patel. D. Srivastava και Y. Wu. Structural Joins: A primitive for efficient XML query pattern matching. Στο *Proc. of ICDE*, 2002.

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

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

[4] M. Fontoura, V. Josifovski, E.Shekita και B. Yang. Optimizing cursor movement in holistic twig joins. Στο *Proceedings of CIKM*, 2005.

[5] H. Jiang, H.Lu και W. Wang. Efficient processing of xml twiq queries with or-predicates. Στο *Proc. of ACM SIGMOD*, 2004.

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

[7] H. Jiang, H.Lu, W. Wang και B.C.Ooi. XR-Tree: Indexing XML data for efficient structural joins. Στο *ICDE*, 2003.

[8] I. Tatarinov, S.Viglas, K.S. Beyer, J. Shanmugasundaram, E.J. Shekita και C.Zhang. Storing and querying ordered ordered XML using a relational database system. Στο *Proc. of SIGMOD*, 2002.

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

[10] J. Lu, T.W. Ling, C.Y. Chan και T. Chen. From region encoding to extended Dewey: On efficient processing of XML twig pattern matching. Στο *Proc. of VLDB*, 2005.

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

[12] C. Zhang, J.F. Naughton, D.J. Dewitt, Q. Luo και G.M. Lohman. On supporting containment queries in relational database management systems. Στο *Proc. of ACM SIGMOD*, 2001.

[13] C. Zhang, J.F. Naughton, D.J. Dewitt, Q. Luo και G.M. Lohman. On supporting containment queries in relational database management systems. http://www.cs.wisc.edu/niagara/papers/ZND+01full.pdf, 2001.

[14] D. Theodoratos, T. Dalamagas, P. Placek, S. Souldatos και T. Sellis. Containment of partially specified tree-pattern-queries. Στο *Proc. of SSDBM*, 2006.

[15] D. Theodoratos, S. Souldatos, T. Dalamagas, P. Placek και T. Sellis. Heuristic containment check of partial tree-pattern queries in the presence of index graphs. Στο *Proc. of ACM CIKM*, 2006.

[16] D. Theodoratos, T. Dalamagas, A. Koufopoulos και N. Gehani. Semantic querying of tree-structured data sources using partially specified tree patterns. Στο *Proc. of ACM CIKM*, 2006.

[17] S. Souldatos, X. Wu, D. Theodoratos, T. Dalamagas και T. Sellis. Evaluation of partial path queries on XML Data. Στο *Proc. of ACM CIKM*, 2007.

[18] J. Zhou. Efficient processing of partially specified twig queries. Στο *Proc. of SIGMOD*, 2007.

[19] D. Oltenau, H. Meuss, T. Furche και F. Bry. XPath: Looking forward. Στο *Proc. of XMLDM, MDDE, YRWS*, 2002.

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

[21] G. Gottlob, C. Koch και R. Pichler. The complexity of xpath query evaluation. Στο *Proc. of PODS*, 2003.

[22] H. Li, M.L. Lee, W. Hu και C. Chen. An Evaluation of XML Indexes for Structural Join.

[23] Y. Wu, J.M. Patel και H.V. Jagadish. Structural join order selection for XML query optimization. Στο *Proc. of ICDE*, 2003.

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

[25] A. Silberschatz, H. Korth και S. Sudarshan. Database System Concepts. McGraw-Hill.

[26] http://www.w3.org/TR/xml/.

[27] http://www.w3.org/TR/xpath/.

[28] http://www.w3.org/TR/xquery/.

[29] http://www.w3.org/TR/xslt/.

[30] http://www.w3schools.com/xml/.

[31] http://www.w3schools.com/xsl/.

[32] http://www.w3schools.com/xpath/.

[33] http://www.w3schools.com/xquery/.

[34] http://www.w3schools.com/schema/.

[35] http://www.cis.upenn.edu/∼treebank/.

[36] http://monetdb.cwi.nl/xml/.