ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Graphit-DB:Πρότυπο Σύστημα Διαχείρισης Δεδομένων Γράφων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΛΕΜΟΝΙΑΣ Μ. ΜΠΟΥΛΑ

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Αθήνα, Δεκέμβριος 2007

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Graphit-DB:Πρότυπο Σύστημα Διαχείρισης Δεδομένων Γράφων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΛΕΜΟΝΙΑΣ Μ. ΜΠΟΥΛΑ

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 17η Δεκεμβρίου 2007.

.....	.....	.....
Τιμολέων Σελλής	Ιωάννης Βασιλείου	Ανδρέας-Γεώργιος Σταφυλοπάτης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Δεκέμβριος 2007

.....

ΛΕΜΟΝΙΑ Μ. ΜΠΟΥΛΑ

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

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Τίμο Σελλή για το ενδιαφέρον που έδειξε για την εκπόνηση της διπλωματικής μου εργασίας, όσο και για την πολύτιμη βοήθεια που μου έχει προσφέρει. Θα ήθελα, επίσης, να ευχαριστήσω τους συνεπιβλέποντες της εργασίας Θοδωρή Δαλαμάγκα, Παναγιώτη Μπούρο και Σπύρο Σκιαδόπουλο για τις εποικοδομητικές συμβουλές που μου έδωσαν για την εργασία και για το χρόνο που αφιέρωσαν. Ευχαριστώ, επίσης, το φίλο και συμφοιτητή Λευτέρη Κρητικό για τη συμπαράσταση και τις παρατηρήσεις του σχετικά με την εργασία αυτή. Τέλος, ευχαριστώ την οικογένειά μου που ήταν δίπλα μου όλο αυτόν τον καιρό.

Περίληψη

Οι γράφοι ως δομή χρησιμοποιούνται από πολύ παλιά για την μοντελοποίηση περίπλοκων δεδομένων. Η πλειοψηφία των γνωστών αλγορίθμων γράφων μπορεί να εφαρμοστεί αποδοτικά είτε σε γράφους με μικρό μέγεθος, είτε σε μη-μεταβαλλόμενους γράφους. Η παρούσα διπλωματική ασχολείται με ερωτήματα πάνω σε δεδομένα που είναι οργανωμένα σε γράφους. Η διαφορά των μεθόδων που θα παρουσιάσουμε στη συνέχεια από τις γνωστές βρίσκεται στον τρόπο αναπαράστασης των γράφων. Οι μέθοδοι που προτείνουμε, χρησιμοποιούν μια αναπαράσταση γράφων, στην οποία κάθε γράφος δίνεται ως σύνολο μονοπατιών. Για την αναπαράσταση αυτή χρησιμοποιούμε μια δομή, την οποία ονομάζουμε PATHINDEX και η οποία περιέχει για κάθε κόμβο στοιχεία, όπως τα μονοπάτια στα οποία εμφανίζεται, η θέση του σ'αυτά και οτιδήποτε άλλο μπορεί να χρειάζεται στον κάθε αλγόριθμο. Υλοποιήσαμε αλγορίθμους που χρησιμοποιούν τη δομή PATHINDEX και συγκρίναμε την απόδοσή τους με γνωστούς από τη βιβλιογραφία αλγορίθμους. Στην περίπτωση των πυκνών γράφων, η πλειοψηφία των μεθόδων που προτείνουμε είναι καλύτερες από τις γνωστές.

Λέξεις Κλειδιά: γράφος, μονοπάτι, ακμή, κόμβος, PATHINDEX, ερώτημα, λίστα γειτνίασης, αναπαράσταση γράφου, αναζήτηση κατά πλάτος, αναζήτηση κατά βάθος, κοινωνικό δίκτυο

Abstract

The graph data structure is commonly used in complex data modeling. The majority of well-known algorithms using graphs can be used efficiently in small-scale or static graphs. The purpose of this thesis is to answer queries on data represented as graphs. The major difference between our algorithms and already used algorithms is graph representation. Our proposed algorithms use graphs which are represented by a set of paths. For this representation, we use a new structure called PATHINDEX. This structure contains information about the graph nodes, such as the paths in which they appear or the node's position in each path. We implemented algorithms using PATHINDEX and compared their performance against that of already used algorithms. We noticed that most of our algorithms perform better, especially on dense graphs.

Keywords: graph, path, edge, node (or vertex), PATHINDEX, query, adjacency list, graph representation, breadth-first search, depth-first search, social network

Πίνακας περιεχομένων

1

Εισαγωγή

1.1 Ερωτήματα πάνω σε δεδομένα που είναι οργανωμένα σε γράφους

Οι γράφοι ως δομή χρησιμοποιούνται από πολύ παλιά για την μοντελοποίηση περίπλοκων δεδομένων. Στην βιβλιογραφία υπάρχουν πάρα πολλοί αλγόριθμοι που απαντούν ερωτήσεις για δεδομένα οργανωμένα σε γράφους, όπως: «Ποιος είναι ο πιο σύντομος δρόμος μεταξύ δύο κόμβων;», «Ποιά είναι τα συνεκτικά κομμάτια του γράφου;», κ.λ.π.

Τον τελευταίο καιρό, λόγω της εξάπλωσης των εφαρμογών Σημασιολογικού Ιστού (Semantic Web), όπως τα Κοινωνικά Δίκτυα (Social Networks), δίκτυα από blogs (Blog Networks), Δίκτυα Εννοιών (Concept Networks), έχουν προκύψει νέες κατηγορίες ερωτήσεων για δεδομένα οργανωμένα σε γράφους. Για παράδειγμα, είναι πολύ χρήσιμο σε ένα Κοινωνικό Δίκτυο που έχει τη μορφή ενός γράφου να μπορεί κάποιος να βρεί, δοθέντων τριών κόμβων v1, v2, v3 που συμβολίζουν τρία διαφορετικά πρόσωπα, τον αριθμό των συντομότερων μονοπατιών που ξεκινούν από τον v1 και καταλήγουν στον v3, διαμέσου του v2. Αν ο αριθμός αυτός είναι μεγάλος, τότε υπάρχει ένδειξη ισχυρής φιλίας του v2 με τους v1 και v3. Επιπλέον, οι γράφοι των εφαρμογών αυτών είναι συνήθως πολύ μεγάλοι.

Η πλειοψηφία των γνωστών αλγορίθμων γράφων μπορεί να εφαρμοστεί αποδοτικά είτε σε γράφους με μικρό μέγεθος (δηλαδή σε γράφους που είναι δυνατό να αποθηκευθούν στην κύρια μνήμη) είτε σε μη-μεταβαλλόμενους γράφους (δηλαδή σε γράφους στους οποίους το σύνολο των κόμβων, των ακμών αλλά και των βαρών των ακμών δεν μεταβάλλονται).

1.2 Αντικείμενο διπλωματικής

Η παρούσα διπλωματική ασχολείται με ερωτήματα πάνω σε δεδομένα που είναι οργανωμένα σε γράφους. Στα πλαίσια της εργασίας γίνεται μια προσπάθεια να απαντηθούν κάποια ερωτήματα με διαφορετικό τρόπο από αυτούς που υπάρχουν στη βιβλιογραφία. Η διαφορά των μεθόδων που θα παρουσιάσουμε στη συνέχεια από τις γνωστές βρίσκεται στον τρόπο αναπαράστασης των γράφων. Συνήθως, οι γράφοι αναπαρίστανται με τη βοήθεια πινάκων ή λιστών που περιέχουν τις ακμές ή τους κόμβους του κάθε γράφου. Έτσι, κάθε βήμα των αλγορίθμων που βρίσκουμε στη βιβλιογραφία έχει μήκος μόνο μία ακμή και επισκέπτεται μόνο τα παιδιά ενός κόμβου. Αυτό που θα θέλαμε είναι να μειώσουμε τον αριθμό των βημάτων που εκτελεί ο κάθε αλγόριθμος. Για να το κάνουμε αυτό θα πρέπει σε κάθε βήμα να μπορούμε να μελετάμε περισσότερες από μία ακμές, δηλαδή, να μην εξετάζουμε μόνο τα παιδιά ενός κόμβου, αλλά τους απογόνους του. Οι μέθοδοι που προτείνουμε χρησιμοποιούν μια διαφορετική αναπαράσταση των γράφων, στην οποία έχουμε κάθε γράφο ως σύνολο μονοπατιών. Επιλέγουμε αυτή την αναπαράσταση, διότι πιστεύουμε ότι μπορεί να μας παρέχει πληροφορίες που θα μας οδηγήσουν πιο γρήγορα στην απάντηση του κάθε ερωτήματος. Με την αναπαράσταση που προτείνουμε ομαδοποιούμε τους κόμβους σε μονοπάτια και αποθηκεύουμε πληροφορία, όχι μόνο για ζευγάρια, αλλά για «οικογένειες» κόμβων. Για την αναπαράσταση αυτή χρησιμοποιούμε μια δομή, την οποία ονομάζουμε PATHINDEX και η οποία περιέχει για κάθε κόμβο στοιχεία όπως τα μονοπάτια στα οποία εμφανίζεται, η θέση του σ'αυτά και οτιδήποτε άλλο μπορεί να χρειάζεται στον κάθε αλγόριθμο.

Τα ερωτήματα με τα οποία ασχολούμαστε είναι τα εξής:

α) Δίνεται ένας κόμβος Α και ζητούνται όλα τα παιδιά του.

β) Δίνονται δύο κόμβοι Α και Β και ζητείται να βρούμε αν υπάρχει μονοπάτι ανάμεσα στους κόμβους Α και Β(και αν ναι, ποιό είναι αυτό).

γ) Δίνονται δύο κόμβοι Α και Β και ζητείται να βρούμε αν υπάρχει μονοπάτι ανάμεσα στους κόμβους Α και Β που να ικανοποιεί κάποιο συγκεκριμένο περιορισμό, για παράδειγμα να έχει μήκος το πολύ 100 κόμβους (και αν ναι, ποιο είναι αυτό).

Η γενική ιδέα των αλγορίθμων που χρησιμοποιήσαμε είναι η εξής:

- Έχουμε ένα σύνολο κόμβων προς μελέτη, το οποίο ονομάζουμε μέτωπο αναζήτησης. Το μέτωπο αναζήτησης περιέχει αρχικά τον κόμβο Α.
- Σε κάθε επανάληψη παίρνουμε ένα κόμβο από το μέτωπο αναζήτησης και ελέγχουμε αν πρόκειται για τον κόμβο Β.
    - Αν ναι, τότε, είτε δίνουμε θετική απάντηση (για το Ερώτημα β), είτε ελέγχουμε αν ικανοποιείται ο περιορισμός που έχουμε θέσει (για το Ερώτημα γ).
    - Αν όχι, τότε προσθέτουμε στο μέτωπο αναζήτησης τα κομμάτια των μονοπατιών που ξεκινούν από τον κόμβο αυτό και μετά.

Δοκιμάσαμε διάφορες εκδοχές αυτής της κεντρικής ιδέας. Για το ερώτημα β) έχουμε τις εξής εκδοχές:

- αλγόριθμοι που τοποθετούν τα κομμάτια των μονοπατιών στο μέτωπο αναζήτησης, με υποεκδοχές:
  - αλγόριθμοι των οποίων το μέτωπο αναζήτησης είναι μια ουρά, τους λέμε bfs-l.
  - αλγόριθμοι των οποίων το μέτωπο αναζήτησης είναι μια στοίβα τους λέμε dfs-l.
- αλγόριθμοι που τοποθετούν χωριστά τους κόμβους κάθε κομματιού στο μέτωπο αναζήτησης, με υποεκδοχές:
  - αλγόριθμοι των οποίων το μέτωπο αναζήτησης είναι μια ουρά, τους λέμε bfs-l.
  - αλγόριθμοι των οποίων το μέτωπο αναζήτησης είναι μια στοίβα τους λέμε dfs-l.

Για το Ερώτημα γ) οι αλγόριθμοι ανήκουν σε μία από τις δύο εκδοχές που είπαμε πιο πάνω, αλλά χρησιμοποιούν δύο δομές, μία στην οποία κρατάμε το μονοπάτι που κατασκευάζουμε και μία στοίβα με τους προς εξέταση κόμβους. Θα μπορούσαμε να πούμε ότι είναι παραλλαγές των dfs-l αλγορίθμων, εμπλουτισμένες με τους απαραίτητους ελέγχους για το μήκος του ζητούμενου μονοπατιού.

Αφού υλοποιήσαμε τους παραπάνω αλγορίθμους πραγματοποιήσαμε πειράματα για να εξετάσουμε την απόδοσή τους. Οι παράμετροι για τα πειράματα αυτά είναι διάφορες, όπως το πλήθος των κόμβων των γράφων ή η αναπαράσταση του γράφου που δίνουμε στο πρόγραμμά μας. Τα ίδια πειράματα έγιναν και για αλγόριθμους που είναι ήδη γνωστοί από τη βιβλιογραφία. Οι αλγόριθμοι που χρησιμοποιήσαμε ως μέτρο σύγκρισης είναι οι αλγόριθμοι αναζήτησης κατά βάθος (dfs) και κατά πλάτος (bfs). Οι αλγόριθμοι αυτοί χρησιμοποιούν λίστες γειτνίασης για την αναπαράσταση των γράφων.

1.2.1 Συνεισφορά

Η συνεισφορά της παρούσας διπλωματικής εργασίας συνοψίζεται στα εξής:

- Προτείνεται μια μέθοδος αναπαράστασης των γράφων με τη μορφή συνόλου μονοπατιών. Περιγράφουμε συνοπτικά την κατασκευή του αρχείου που περιέχει το γράφο σαν μονοπάτια και μιας δομής με την οποία αποθηκεύουμε την πληροφορία που προέρχεται από το αρχείο αυτό κατά τη διάρκεια εκτέλεσης των προγραμμάτων μας.
- Περιγράφονται και υλοποιούνται αλγόριθμοι που κάνουν χρήση της παραπάνω πληροφορίας. Οι αλγόριθμοι αυτοί υλοποιούν ερωτήματα:
  - εύρεσης των παιδιών ενός δοθέντος κόμβου Α.
  - εύρεσης μονοπατιού ανάμεσα σε δύο κόμβους Α και Β και εκτύπωσης του μονοπατιού αυτού.
  - εύρεσης μονοπατιού ανάμεσα σε δύο κόμβους Α και Β, το οποίο πρέπει να ικανοποιεί

κάποιο περιορισμό, και εκτύπωσης του μονοπατιού αυτού.

- Πραγματοποιούνται πειράματα για την αξιολόγηση των αλγορίθμων που προτείνονται. Τα πειράματα αυτά εξετάζουν τη συμπεριφορά των αλγορίθμων αυτών και τη συγκρίνουν με τη συμπεριφορά των αλγορίθμων που είναι γνωστοί από τη σχετική βιβλιογραφία.
- Με βάση τα παραπάνω πειράματα αναφέρονται συμπεράσματα, σχόλια και προτάσεις.
    - Προτείνεται η συμπίεση του γράφου με ομαδοποίηση των κόμβων που έχουν κοινές ιδιότητες για να επιταχυνθεί η αναζήτηση.

1.3 Οργάνωση κειμένου

Στο Κεφάλαιο 2 αναφέρονται οι σχετικές με το αντικείμενο εργασίες που υπάρχουν στη διεθνή βιβλιογραφία. Το Κεφάλαιο 3 δίνει ορισμούς εννοιών που θα χρησιμοποιήσουμε σε όλο το κείμενο για να περιγράψουμε το πρόβλημα και τους αλγόριθμους. Στο Κεφάλαιο 4 γίνεται μια τυπική περιγραφή του προβλήματος με το οποίο ασχολείται η παρούσα διπλωματική εργασία. Οι αλγόριθμοι που υλοποιήθηκαν, καθώς και παραδείγματα για την κατανόησή τους, περιέχονται στο Κεφάλαιο 5. Στο Κεφάλαιο 6 παρουσιάζονται τα πειράματα που εκτελέστηκαν για να αξιολογηθεί η απόδοση των αλγορίθμων που υλοποιήθηκαν. Στο Κεφάλαιο 7 γίνεται αναφορά στις κλάσεις και τις δομές που χρησιμοποιήθηκαν κατά την υλοποίηση. Στο Κεφάλαιο 8 καταγράφονται τα συμπεράσματα στα οποία μπορεί κανείς να καταλήξει από την προηγούμενη μελέτη. Επιπλέον, γίνεται περιγραφή μεθόδων που θα μπορούσαν να βελτιώσουν την απόδοση των μεθόδων που αναφέρθηκαν παραπάνω, όπως η συμπίεση των γράφων. Στο Κεφάλαιο 9 παρουσιάζεται η βιβλιογραφία στην οποία ανατρέξαμε κατά την εκπόνηση της διπλωματικής εργασίας.

2

Σχετικές Εργασίες

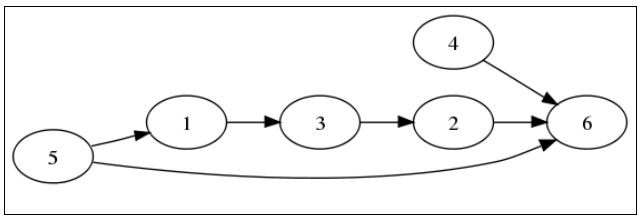
2.1 Ερωτήματα που απαντώνται στη βιβλιογραφία

Προκειμένου να ενημερωθούμε σχετικά με το πώς αντιμετωπίζονται διάφορα προβλήματα σχετικά με το αντικείμενο της παρούσας διπλωματικής, ανατρέξαμε στη διεθνή βιβλιογραφία. Δημοφιλή προβλήματα στην επεξεργασία ερωτήσεων σε γράφους με τα οποία ασχολούνται οι ερευνητές είναι τα παρακάτω:

1. Εύρεση κόμβων - παιδιών

Δίνεται ένας κόμβος K και ένας γράφος G (V,E) και ζητούνται όλα τα παιδιά του, δηλαδή όλοι οι κόμβοι με τους οποίους συνδέεται με μία μόνο ακμή. Αν ο γράφος είναι κατευθυνόμενος θα πρέπει ο δεδομένος κόμβος να είναι η πηγή (source) της ακμής.

Ένα παράδειγμα αυτού του είδους των ερωτημάτων είναι η αναζήτηση των παιδιών του κόμβου 3 στο γράφο του Σχήματος 2.1.



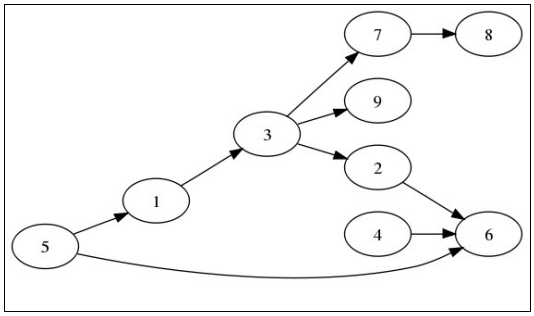
Σχήμα 2.1

Η απάντηση σε αυτό το ερώτημα είναι το σύνολο {2,4}.

2. Εύρεση κόμβων – απογόνων

Δίνεται ένας κόμβος K και ένας γράφος G (V,E) και ζητούνται όλοι οι απόγονοί του, δηλαδή όλοι οι κόμβοι με τους οποίους συνδέεται με κάποιο μονοπάτι. Αν ο γράφος είναι κατευθυνόμενος θα πρέπει ο δεδομένος κόμβος να είναι η πηγή (source) του μονοπατιού. Το σύνολο αυτό των κόμβων προκύπτει από το μεταβατικό κλείσιμο (transitive closure) του δοθέντος γράφου.

Ένα παράδειγμα αυτού του είδους των ερωτημάτων είναι να βρούμε τους απογόνους του κόμβου 3 στο γράφο του Σχήματος 2.2.



Σχήμα 2.2

Η απάντηση σε αυτό το ερώτημα είναι το σύνολο {2,4,5,6}.

3. Εύρεση συντομότερου μονοπατιού

Δίνεται ένας γράφος G(V,E) και ένα ζευγάρι κόμβων A, B από το σύνολο V. Ζητείται να βρεθεί το συντομότερο μονοπάτι ανάμεσα στους δύο αυτούς κόμβους. Αν ο γράφος είναι κατευθυνόμενος, ο A πρέπει να είναι η πηγή (source) και ο B ο τελευταίος κόμβος του μονοπατιού.

Στη συνέχεια παρουσιάζουμε εργασίες που προτείνουν τεχνικές επεξεργασίας των παραπάνω ερωτήσεων.

2.2 Εύρεση κόμβων-παιδιών

Μέθοδος προσπέλασης με βάση τη συνεκτικότητα για δίκτυα και υπολογισμούς σε δίκτυα. [SL97]

Η μέθοδος αυτή βασίζεται στην ιδιότητα που έχουν ζευγάρια κόμβων ενός γράφου να «ανακτώνται» μαζί, με μεγάλη συχνότητα. Σαν στόχο έχει να μειώσει το κόστος εισόδου/εξόδου για τα ερωτήματα σε δίκτυα (γενικότερα: γράφους). Η μορφή στην οποία δίνεται ο γράφος (στο spatial network) είναι οι λίστες γειτνίασης (adjacency lists). Ο γράφος διαμερίζεται δυναμικά σε συστάδες κόμβων που αποθηκεύονται στις σελίδες του δίσκου, με τη χρήση ευριστικής. Η μετρική που μας ενδιαφέρει εδώ είναι το WCRR (Weighted Connectivity Residue Ratio), το οποίο είναι. ένας τρόπος εκτίμησης του κόστους εισόδου/εξόδου των ερωτημάτων. Το WCRR εξαρτάται από την πιθανότητα να

προσπελαστούν μαζί δύο κόμβοι του γράφου, οι οποίοι βρίσκονται και στην ίδια σελίδα του δίσκου. Η πιθανότητα αυτή εκφράζεται δίνοντας ένα «βάρος» σε κάθε ακμή του γράφου.

Το WCRR δίνεται από τον τύπο (1):

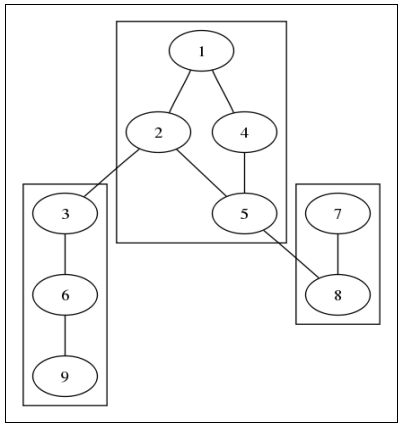
$$WCRR = \frac{\sum w(u,v) \; such \; that \; Page(u) = Page(v)}{\sum weights \; of \; all \; edges} \quad (1)$$

Σε πολλές περιπτώσεις χρησιμοποιούμε B^+ δέντρα με Z-order λόγω της εφαρμογής η οποία έχει να κάνει με γεωγραφικά συστήματα και η οποία απαιτεί διάταξη των κόμβων με τοπολογική σειρά. Η μέθοδος αυτή εφαρμόζεται στο ATIS/IVHS [IVHS94]. Το σύστημα αυτό είναι μια προσπάθεια να χρησιμοποιηθεί η τεχνολογία πληροφορίας και επικοινωνιών στο σύστημα μεταφορών. Έχει σαν στόχο να διαχειριστεί παράγοντες που συγκρούονται μεταξύ τους όπως το πλήθος των οχημάτων, οι διαδρομές και τα φορτία προκειμένου να προωθηθεί η ασφάλεια, οι χρόνοι μετακίνησης και η κατανάλωση καυσίμων.

Παράδειγμα 2.1:

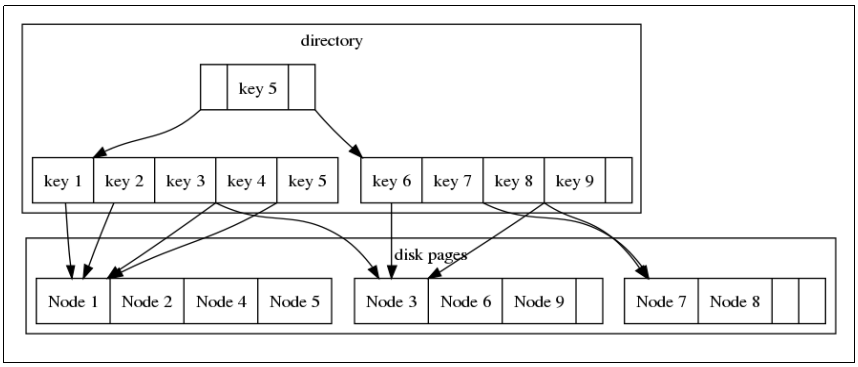
Έστω ότι έχουμε ένα γράφο, ο οποίος αναπαριστά, για παράδειγμα, μια γεωγραφική περιοχή. Αρχικά, θα πρέπει να ομαδοποιήσουμε τους κόμβους σε συστάδες με βάση κάποια ευριστική. Εδώ, χρησιμοποιούμε την ευριστική των Cheng και Wei [CW91], που ονομάζεται two-way ratio-cut. Στόχος μας είναι να μειώσουμε την τιμή του W.C.R.R. Η έξοδος της ευριστικής είναι ένα σύνολο από σελίδες του δίσκου, κάθε μία από τις οποίες περιέχει κόμβους που έχουν μεγάλη πιθανότητα να προσπελαστούν μαζί.

Έστω ο γράφος του Σχήματος 2.3:



Σχήμα 2.3

Η διαμέριση του γράφου φαίνεται στο Σχήμα 2.3. Στο δίσκο, τώρα, η εικόνα θα είναι όπως στο Σχήμα 2.4:



Σχήμα 2.4

Αν θέλουμε να προσπελάσουμε τα παιδιά του κόμβου 3, θα φέρουμε τη σελίδα στην οποία περιέχεται. Από τη σελίδα αυτή βρίσκουμε τη λίστα με τα παιδιά του 3. Παιδιά του 3 είναι ο 2 και ο 6. Ο 6 Είναι στην ίδια σελίδα, άρα δεν χρειάζεται να φέρουμε άλλη σελίδα από το δίσκο. Για τον 2, όμως, θα πρέπει να φέρουμε την αντίστοιχη σελίδα, που τον περιέχει, από το δίσκο.

2.3 Εύρεση κόμβων απογόνων

Τεχνική οργάνωσης αρχείου που υποστηρίζει την αναδρομική διάσχιση [LD89]

Η μέθοδος αυτή προτείνει μια δομή αποθήκευσης που αποτελείται από δύο αρχεία, ένα που είναι το κυρίως αρχείο και ένα που έχει το ρόλο του αρχείου δεικτοδότησης. Η μέθοδος αυτή εφαρμόζεται σε μεγάλους, κατευθυνόμενους και ακυκλικούς γράφους. Η ταξινόμηση των κόμβων γίνεται με τοπολογική σειρά και η αποθήκευση γίνεται σε B-δέντρα. Με τον τρόπο αυτό οι πρόγονοι/απόγονοι προσπελαύνονται ψάχνοντας προς τα εμπρός/πίσω στο αρχείο. Βασικό στοιχείο είναι οι ετικέτες των κόμβων/ακμών, που περιέχουν πληροφορίες για τους κόμβους/ακμές του γράφου. Η έξοδος που προκύπτει είναι ένας νέος γράφος που περιέχει κόμβους και ακμές από τον αρχικό και οι ετικέτες των ακμών έχουν υπολογιστεί κατά τη διάσχιση του γράφου.

Παράδειγμα 2.2:

Έστω ότι ο γράφος αναπαρίσταται όπως στον Πίνακα 2.1:

Nodes

Node Key	Node Data
A	100
B	220
C	300
D	150
E	400
F	150
G	130
H	290
I	160
J	235
K	100

Edges

Start Node	End Node	Edge Data
A	B	3
A	C	4
D	C	6
D	E	4
G	H	1
G	I	3
C	F	5
H	J	10
I	J	12
I	K	3

Πίνακας 2.1

Node#	NodeData
1	100
2	220
3	300
4	150
5	400
6	150
7	130
8	290
9	160
10	235
11	100

Πίνακας 2.2

Node#	NodeData
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I
10	J
11	K

Πίνακας 2.3

Για τη δημιουργία του αρχείου έχουμε:

Φάση1

Χωρίζουμε τον πίνακα με τους κόμβους σε δύο πίνακες V' και VS, καθένας από τους οποίους είναι ταξινομημένος ως προς το Node#.

Ο πίνακας V είναι ο Πίνακας 2.2 και ο πίνακας VS είναι ο Πίνακας 2.3.

Φάση2

Ταξινομούμε το αρχείο Edges ως προς τον τελικό κόμβο, οπότε δημιουργείται το αρχείο T. Εκτελείται ένα merge-join μεταξύ των αρχείων VS και T (ως προς το όνομα του κόμβου και τον τελικό κόμβο), οπότε δημιουργείται ο αρχείο T’, το οποίο ταξινομείται ως προς τον αρχικό κόμβο. Εκτελείται ακόμα ένα merge-join μεταξύ των VS και S και κατασκευάζεται έτσι το αρχείο S’. Μετά από ταξινόμηση του S’ ως προς τον τελικό κόμβο, φτιάχνεται το αρχείο S’’.

Start Node	End Node	Edge Data
A	B	3
A	C	4
D	C	6
D	E	4
C	F	5
G	H	1
G	I	3
H	J	10
I	J	12
I	K	3

Πίνακας 2.4(T)

Π.χ. για το T': Η γραμμή του VS με Node# = 1 συνδέεται με τις δύο πρώτες γραμμές του T διότι έχουν την ίδια τιμή στα πεδία NodeKey και EndNode και δημιουργούνται οι δύο πρώτες εγγραφές στον πίνακα T'. Το πεδίο EndNode# προκύπτει από συνδυασμό των πεδίων ως προς τα γίνεται ο σύνδεσμος. Για το S': Η γραμμή του VS με Node# = 1 συνδέεται με τις δύο πρώτες γραμμές του S διότι έχουν την ίδια τιμή στα πεδία NodeKey και StartNode και δημιουργούνται δύο εγγραφές στον πίνακα S'. Το πεδίο StartNode# προκύπτει από συνδυασμό των πεδίων ως προς τα γίνεται ο σύνδεσμος.

Start Node	End Node#	Edge Data
A	B2	3
A	C3	4
D	C3	6
D	E5	4
C	F6	5
G	H8	1
G	I9	3
H	J10	10
I	J10	12
I	K11	3

Πίνακας 2.5(T')

Start Node	End Node#	Edge Data
A	B2	3
A	C3	4
C	F6	5
D	C3	6
D	E5	4
G	H8	1
G	I9	3
H	J10	10
I	J10	12
I	K11	3

Πίνακας 2.6(S)

Start Node#	End Node#	Edge Data
A1	B2	3
A1	C3	4
C3	F6	5
D4	C3	6
D4	E5	4
G7	H8	1
G7	I9	3
H8	J10	10
I9	J10	12
I9	K11	3

Πίνακας 2.7(S')

Φάση3

Για την ταξινόμηση χρησιμοποιούμε τους Πίνακες 2.8, οι οποίοι περιέχουν μια συμπιεσμένη μορφή των αρχικών αρχείων.

Φάση4

Με βάση τους παραπάνω πίνακες κατασκευάζουμε το κυρίως αρχείο…(Πίνακας 2.9)

Nodes

	Edge Count	First Edge
A	0	1
B	1	-1
C	2	3
D	0	4
E	1	-1
F	1	-1
G	0	6
H	1	8
I	1	9
J	2	-1
K	1	-1

Edges

	End Node	Stop Flag
A→B	2	0
A→C	3	1
C→F	6	1
D→C	3	0
D→E	5	1
G→H	8	1
G→I	9	1
H→J	10	1
I→J	10	0
I→K	11	1

Πίνακας 2.8

Order Key	Node	Successors	Predecessors
#10	A 100	(#40,3),(#50,4)	
#20	D 150	(#50,6),(#60,4)	
#30	G 130	(#70,1),(#80,3)	
#40	B 220		(#10,3)
#50	C 300	(#90,5)	(#10,4),(#20,6)
#60	E 400		(#20,4)
#70	H 290	(#100,10)	(#30,1)
#80	I 160	(#100,12),(#110,3)	(#30,3)
#90	F 150		(#50,5)
#100	J 235		(#70,10),(#80,12)
#110	K 100		(#80,3)

Πίνακας 2.9

Φάση5

…και το αρχείο index(Πίνακας 2.10).

Node Key	Order Key
A	#10
B	#40
C	#50
D	#20
E	#60
F	#90
G	#30
H	#70
I	#80
J	#100
K	#110

Πίνακας 2.10

Για να βρούμε π.χ. τους απογόνους του κόμβου G, κάνουμε τα εξής:

- Βρίσκουμε από το αρχείο index το αντίστοιχο Order Key. Είναι #30.
- Αναζητούμε στο κυρίως αρχείο την εγγραφή με Order Key = #30.

  Έξοδος G.
- Παίρνουμε τη λίστα με τα παιδιά (H - #70 και I - #80).

  Η ουρά τώρα είναι: {I,H}
- Αναζητούμε στο κυρίως αρχείο την εγγραφή με Order Key = #70.

  Έξοδος H.
- Παίρνουμε τη λίστα με τα παιδιά (J - #100).

  Η ουρά τώρα είναι: {J,I}.
- Αναζητούμε στο κυρίως αρχείο την εγγραφή με Order Key = #80.

  Έξοδος I.
- Παίρνουμε τη λίστα με τα παιδιά (J - #100 και K - #110).

  Η ουρά τώρα είναι: {K,J}.
- Αναζητούμε στο κυρίως αρχείο την εγγραφή με Order Key = #100.

  Έξοδος J.
- Η λίστα με τα παιδιά είναι κενή.
- Αναζητούμε στο κυρίως αρχείο την εγγραφή με Order Key = #110.

  Έξοδος K.
- Η λίστα με τα παιδιά είναι κενή.

Άρα, οι απόγονοι του G είναι οι {H,I,J,K}

Μέθοδος Αποδοτικής Διαχείρισης Μεταβατικών Σχέσεων σε Μεγάλες Βάσεις Δεδομένων και Γνώσης [ABJ89]

Πρόκειται για μια τεχνική συμπίεσης που βασίζεται στο χαρακτηρισμό ακμών του ζευγνύοντος δέντρου ενός γράφου, με χρήση αριθμητικών διαστημάτων. Η μέθοδος αυτή χρησιμοποιείται σε εφαρμογές που έχουν να κάνουν με ιεραρχικές δομές και κληρονομικότητα. Η ιδέα είναι να υπολογίσουμε το σύνολο των κόμβων που μπορούμε να προσπελάσουμε από κάποιο κόμβο, λαμβάνοντας υπ' όψη τα αντίστοιχα σύνολα των παιδιών του.

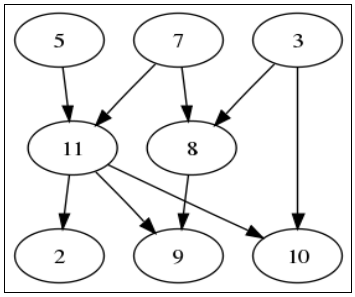
Το πεδίο εφαρμογής αυτής της μεθόδου είναι οι δυικές σχέσεις που προκύπτουν από βάσεις δεδομένων και οι οποίες μπορούν να αναπαρασταθούν με γράφους.

Παράδειγμα 2.3:

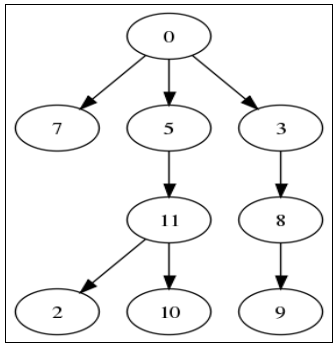
Ο αρχικός γράφος φαίνεται στο Σχήμα 2.5.

1. Το ζευγνύον δέντρο φαίνεται στο Σχήμα 2.6.

Εδώ βλέπουμε ότι έχουν αφαιρεθεί ακμές και έχει διατηρηθεί ο ελάχιστος αριθμός ακμών που συνδέουν τις κορυφές, ώστε να υπάρχει μονοπάτι ανάμεσα σε όλα τα ζευγάρια κορυφών. Επίσης, έχει προστεθεί η ρίζα, με την οποία ενώσαμε όλες τις συνεκτικές συνιστώσες του γράφου σε μία.



Σχήμα 2.5



Σχήμα 2.6

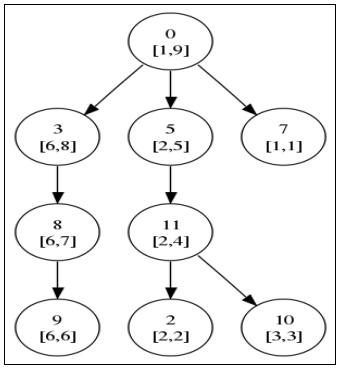
2.Ο ορισμός των διαστημάτων φαίνεται στο Σχήμα 2.7.

Έτσι, υπάρχει ένα και μοναδικό διάστημα που περιέχει πληροφορίες σχετικά με την προσβασιμότητα των κόμβων που μπορούν να προσπελαστούν με μια σειρά ακμών του δέντρου ξεκινώντας από ένα συγκεκριμένο κόμβο. Το διάστημα αυτό λέγεται διάστημα δέντρου. Στην ουσία, με αυτόν τον τρόπο αποθηκεύεται η κλειστότητα του γράφου.

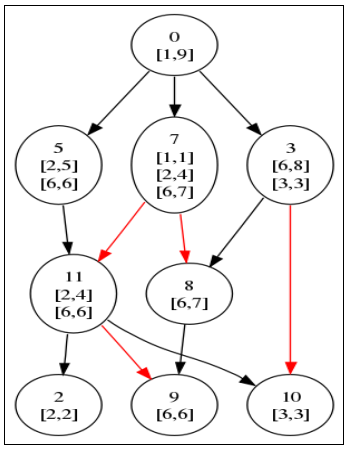
3.Κατασκευή του τελικού γράφου με όλες τις πληροφορίες, ο οποίος έχει τη μορφή που φαίνεται στο Σχήμα 2.8.

Με κόκκινο παριστάνονται οι ακμές του γράφου που δεν ανήκουν στο ζευγνύον δέντρο. Μέσω των ακμών αυτών οι κόμβοι κληρονομούν τα λεγόμενα «διαστήματα που δεν ανήκουν στο δέντρο» (non-tree intervals), όπως π.χ. το διάστημα [6,6] για τον κόμβο 11. Στο διάγραμμα αυτό φαίνεται το αποτέλεσμα της μεθόδου.

Για να βρούμε τους απογόνους του κόμβου 5, θα πρέπει να ψάξουμε ανάμεσα στους κόμβους με postorder τιμές 2,3,4 και 6. Οι κόμβοι αυτοί είναι οι 2,9,11 και 8.



Σχήμα 2.7



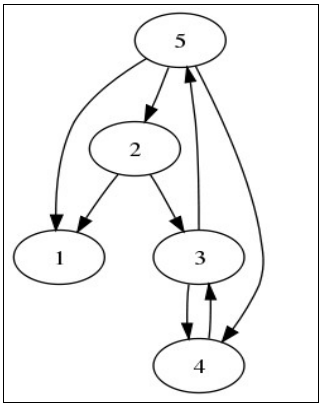
Σχήμα 2.8

Νέες Στρατηγικές για τον Υπολογισμό της Κλειστότητας μιας Σχέσης Βάσης Δεδομένων [L87]

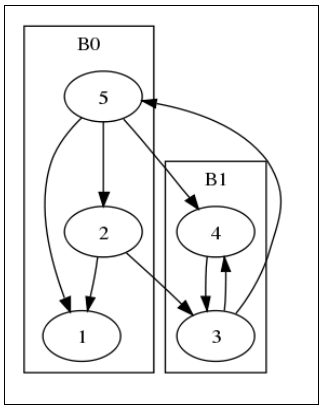
Πρόκειται για ένα σύνολο από μεθόδους που προσπαθούν να υπολογίσουν την κλειστότητα ενός γράφου. Οι μέθοδοι αυτές στοχεύουν στην αποφυγή των περιττών επαναλήψεων κατά τον υπολογισμό της κλειστότητας ενός γράφου και στην απαλοιφή εκείνου του μέρους της εισόδου κάθε επανάληψης που δεν συνεισφέρει τίποτα στο τελικό αποτέλεσμα. Η πρώτη μέθοδος προτείνει να αφαιρέσουμε τις πλειάδες που δεν δίνουν καινούργιες πλειάδες στις επόμενες επαναλήψεις. Η μέθοδος αυτή αποτελεί μια βελτίωση άλλων, παραδοσιακών, αλγορίθμων. Η δεύτερη μέθοδος προτείνει να μειώσουμε τις επαναλήψεις που χρειάζονται για τον υπολογισμό της κλειστότητας. Η βασική δομή, πάνω στην οποία δουλεύει η στρατηγική αυτή, είναι η δομή bucket, που είναι χώρος στη μνήμη όπου αποθηκεύονται οι πλειάδες. Ο αλγόριθμος που συγκεντρώνει τα χαρακτηριστικά των δύο παραπάνω είναι μια μέθοδος βασισμένη στον κατακερματισμό που ονομάζεται HYBRIDTC.

Παράδειγμα 2.4:

Έστω ο γράφος (ο οποίος μπορεί να περιέχει και κύκλους!) του Σχήματος 2.9. Τα buckets φαίνονται στο Σχήμα 2.10. Ο χωρισμός των κόμβων σε buckets γίνεται ως εξής: Στο bucket B0 ανήκουν οι κόμβοι {1,2,5} ενώ στο bucket B1 ανήκουν οι κόμβοι {3,4}. Η διαμέριση του γράφου θα μπορούσε να γίνει και με πολλούς άλλους τρόπους.



Σχήμα 2.9



Σχήμα 2.10

Για την πρώτη στρατηγική έχουμε:

Στην πρώτη επανάληψη, η πλειάδα (2,3) στην πρώτη στήλη θα ταιριάξει με τις πλειάδες (3,4) και (3, 5). Προκύπτουν έτσι οι πλειάδες (2,4) και (2,5), που εντάσσονται στο ΔR1, το οποίο θα χρησιμοποιηθεί για την επόμενη επανάληψη. Με γκρι χρώμα σημαδεύονται οι πλειάδες που δεν δίνουν καινούριες πλειάδες. Όλα αυτά φαίνονται στον Πίνακα 2.11.

Iteration1		Iteration2		Iteration3	
Ro	Ro	ΔR1	Ro1	ΔR2	Ro2
(2 , 1)	(2 , 1)	(2 , 4)	(2 , 3)	(2 , 2)	(2 , 3)
(2 , 3)	(2 , 3)	(2 , 5)	(3 , 4)	(4 , 2)	(3 , 4)
(3 , 4)	(3 , 4)	(3 , 3)	(3 , 5)	(5 , 5)	(3 , 5)
(3 , 5)	(3 , 5)	(3 , 1)	(4 , 3)	(3 , 3)	(5 , 2)
(4 , 3)	(4 , 3)	(3 , 2)	(5 , 2)		
(5 , 1)	(5 , 1)	(4 , 4)			
(5 , 2)	(5 , 2)	(4 , 5)			
(5 , 4)	(5 , 4)	(5 , 3)			

Πίνακας 2.11

Για τη δεύτερη στρατηγική έχουμε:

Η πλειάδα (3,5) από το ΔR^0_i και η πλειάδα (5,2) από το R^1_{0i} συνδυάζονται και δίνουν την πλειάδα (3,2). Η πλειάδα αυτή καταχωρείται στο bucket 0, αφού το δεύτερο χαρακτηριστικό της είναι το 2, που ανήκει στο B0.

Η πλειάδα (5,2) από το ΔR^0_i και η πλειάδα (2,3) από το R^1_{0i} συνδυάζονται και δίνουν την πλειάδα (5,3). Η πλειάδα αυτή καταχωρείται στο bucket 1, αφού το δεύτερο χαρακτηριστικό της είναι το 3, που ανήκει στο B1.

Με γκρι χρώμα σημαδεύονται οι πλειάδες που δεν δίνουν καινούριες πλειάδες. Αυτά φαίνονται στον Πίνακα 2.12.

				Iteration1		
	ΔR^0_i	R^1_{0i}	R_i^1	ΔR^1_i	R^2_{0i}	R_1^2
Bucket 0	(3 , 5)	(5 , 4)	(3 , 2)	(4 , 5)	(5 , 4)	(4 , 2)
	(5 , 2)	(2 , 3)	(3 , 1)	(2 , 5)	(2 , 3)	(4 , 1)
	(5 , 1)	(5 , 2)			(5 , 2)	(2 , 2)
	(2 , 1)	(2 , 1)			(2 , 1)	
		(5 , 1)			(5 , 1)	
Bucket 1	(5 , 4)	(4 , 3)	(5 , 3)	(4 , 4)	(4 , 3)	
	(4 , 3)	(3 , 4)	(4 , 4)	(2 , 4)	(3 , 4)	
	(3 , 4)	(3 , 5)	(3 , 3)		(3 , 5)	
	(2 , 3)		(2 , 4)			

Πίνακας 2.12

Το παράδειγμα για τον HYBRIDTC είναι παρόμοιο με αυτό της δεύτερης στρατηγικής, με τη διαφορά ότι σε κάθε επανάληψη μειώνουμε το μέγεθος της αρχικής σχέσης, όπως στην πρώτη στρατηγική.

Για να βρούμε τους απογόνους του κόμβου 4, θα ψάξουμε όλες τις πλειάδες με πρώτο χαρακτηριστικό ίσο με 4. Δηλαδή, έχουμε το σύνολο:

{(4,1),(4,2),(4,5),(4,1)}. Συνεπώς, οι απόγονοι του κόμβου 4 είναι οι {1,2,3,5}.

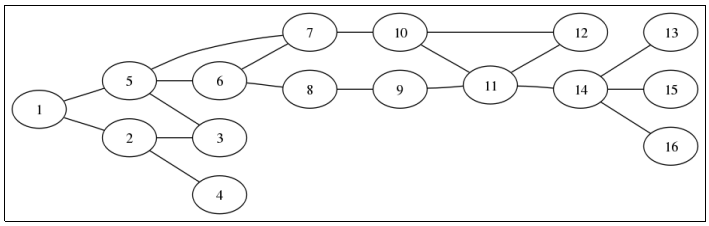
Αποδοτική Αναζήτηση σε πολύ μεγάλες βάσεις δεδομένων [AJ88]

Η βασική ιδέα της μεθόδου αυτής είναι να προϋπολογίζονται κάποιες «πληροφορίες» σχετικές με τους κόμβους ενός γράφου, οι οποίες χρησιμοποιούνται στη συνέχεια από τις επόμενες επαναλήψεις της αναζήτησης για να περιοριστεί το πεδίο αναζήτησης, δηλαδή το σύνολο των κόμβων που πρέπει να επισκεφθούμε για να φτάσουμε στο στόχο μας. Η παρούσα μέθοδος χρησιμοποιεί την έννοια της περιοχής (domain). Μια περιοχή είναι ένα σύνολο κόμβων, που συνδέονται μεταξύ τους με μονοπάτια. Κάθε τέτοιο σύνολο είναι συνεκτικό. Κάθε περιοχή έχει το κέντρο της και την ακτίνα της. Η ακτίνα μιας περιοχής είναι η ελάχιστη απόσταση μεταξύ του κέντρου της και του κόμβου που βρίσκεται στη μέγιστη απόσταση από αυτό.

Η μέθοδος αυτή βρίσκει εφαρμογή κατά την επίλυση του *«Προβλήματος Ακραίου Μονοπατιού» ("Extremal Path Problem")*. Το πρόβλημα αυτό περιλαμβάνει τον ορισμό ενός μονοπατιού ανάμεσα σε δύο κόμβους το γράφου, το οποίο έχει ακραία τιμή (μέγιστη ή ελάχιστη με βάση κάποια κριτήρια ταξινόμησης) στην ετικέτα του. Χαρακτηριστικό παράδειγμα είναι αυτό της εύρεσης του φθηνότερης πτήσης ανάμεσα σε δύο πόλεις.

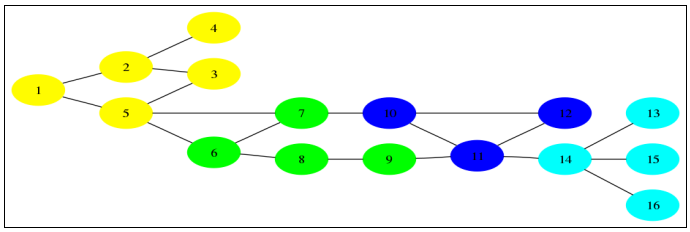
Παράδειγμα 2.5:

Έστω ο γράφος που δίνεται στο σχήμα 2.11.



Σχήμα 2.11

Χρησιμοποιώντας κάποια από τις δοθείσες ευριστικές, έχουμε:



Σχήμα 2.12

Αν ξεκινήσουμε από τον κόμβο 1, τότε, αφού εξερευνήσουμε όλους τους κόμβους της κίτρινης περιοχής, θα προχωρήσουμε σε κάποιον από τους κόμβους της μπλε περιοχής. Για να βρούμε σε ποιον από τους πράσινους κόμβους θα πάμε, το βρίσκουμε από το ανώτερο και το κατώτερο όριο της απόστασης που θα πρέπει να έχουν δύο κόμβοι που ανήκουν σε δύο διαφορετικές περιοχές. Τα όρια αυτά βελτιώνονται όσο εξελίσσεται η αναζήτηση. Τα μονοπάτια που βρίσκουμε υποδεικνύουν και τους κόμβους που είναι απόγονοι του κόμβου 1.

2.4 Εύρεση συντομότερου μονοπατιού

Αναζήτηση συσχετίσεων σε κοινωνικά δίκτυα [TU]

Ένα κοινωνικό δίκτυο είναι η αναπαράσταση των σχέσεων μεταξύ ατόμων που ανήκουν σε ομάδες ή οργανισμούς με τη μορφή γράφου. Στο γράφο αυτόν οι κόμβοι είναι τα άτομα που συμμετέχουν στην ομάδα και οι ακμές είναι οι σχέσεις που τα άτομα αυτά έχουν μεταξύ τους. Με τον όρο συσχέτιση εννοούμε ένα μονοπάτι από γνωριμίες μέσω των οποίων ένα άτομο "πηγή" μπορεί να έρθει σε επαφή με κάποιο άλλο άτομο. Αν, για παράδειγμα, ο Α γνωρίζει το Β και ο Β γνωρίζει το Γ, μέσα από τη διαδοχή των γνωριμιών, ο Α μπορεί να έρθει σε επαφή με το Γ, αφού ο Β θα τον έχει "συστήσει" στο Γ.

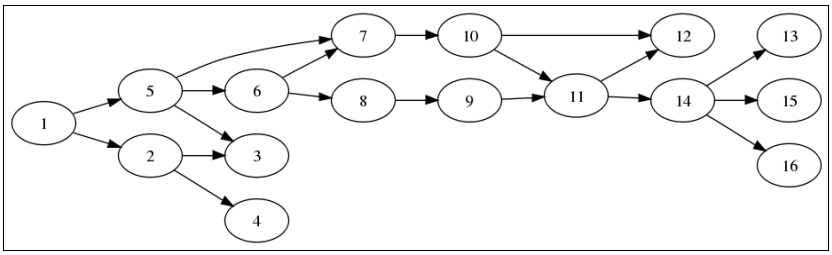
Η μέθοδος που περιγράφουμε έχει σαν στόχο να βρει και να ταξινομήσει τις διάφορες συσχετίσεις σε πολύ μεγάλα κοινωνικά δίκτυα. Για το λόγο αυτό μοντελοποιεί το πρόβλημα ως πρόβλημα εύρεσης συντομότερου μονοπατιού και χαρακτηρίζει τις γνωριμίες των ατόμων που συμμετέχουν στο δίκτυο με τη βοήθεια «βαρών». Ο αλγόριθμος που προτείνεται λύνει το πρόβλημα σε δύο στάδια: α)Το

πρώτο χρησιμοποιεί τον αλγόριθμο του Dijkstra για να βρει το συντομότερο μονοπάτι από όλους τους κόμβους του δικτύου προς τον κόμβο Β. β)Το δεύτερο στάδιο χρησιμοποιεί μια διαδικασία κατά βάθος αναζήτησης για να βρει γρήγορα μονοπάτια των οποίων το μήκος δεν είναι παραπάνω από το όριο $(1+\beta)L_{min}$, όπου β είναι ένα όριο που ορίζεται από το χρήστη και L_{min} είναι το μήκος του συντομότερου μονοπατιού ανάμεσα στον κόμβο Α και στον κόμβο Β.

Υλοποιήσεις αυτής της μεθόδου εφαρμόζονται πάνω σε πραγματικά δεδομένα, όπως το Personal Network Search (PNS) και χρησιμοποιούνται στην προσπάθεια επαλήθευσης της υπόθεσης ότι όλοι οι άνθρωποι στον κόσμο "συνδέονται"/γνωρίζονται μεταξύ τους με ένα μονοπάτι που δεν περιέχει παραπάνω από 5 ενδιάμεσους κόμβους. Πράγματι, ένα πολύ μεγάλο ποσοστό των ερωτημάτων που εκτελέστηκαν πάνω στα δεδομένα του PNS έδωσαν αποτέλεσμα με λιγότερους από 6 ενδιάμεσους στο μονοπάτι άτομοΑ – άτομοΒ.

Παράδειγμα 2.6:

Έστω ο γράφος του Σχήματος 2.13:



Σχήμα 2.13

Θα δείξουμε πώς δουλεύει ο παραπάνω αλγόριθμος αναζητώντας το σύνολο των συντομότερων μονοπατιών ανάμεσα στους κόμβους 1 και 14. Θεωρούμε ότι το βάρος κάθε ακμής είναι ίσο με 1(Πίνακας 2.13). Η πρώτη επανάληψη θα έχει $v_{min} = 14$. Ακολουθώντας τα βήματα του αλγορίθμου ο Πίνακας 2.13 θα μετατραπεί στον Πίνακα 2.14. Ο επόμενος «τρέχων κόμβος» είναι ο 11. Η επανάληψη έχει ως αποτέλεσμα τον Πίνακα 2.15. Στην περίπτωση αυτή θα βάλουμε δύο κόμβους στη σωρό, αφού έχουμε δύο ακμές που καταλήγουν στον κόμβο 11. Έστω ότι ο επόμενος κόμβος που παίρνουμε από τη στοίβα είναι ο 9. Ο Πίνακας 2.16 δείχνει τη συνέχεια.

Συνεχίζοντας με αυτόν τον τρόπο, καταλήγουμε στο αποτέλεσμα που είναι ένα σύνολο από μονοπάτια, όπως αυτό που φαίνεται στον Πίνακα 2.17. Με x σημειώνουμε τις περιπτώσεις όπου δεν υπάρχει μονοπάτι. Συνεχίζουμε τώρα με το δεύτερο στάδιο της εκτέλεσης.

κόμβος	d'	επόμενος στο μονοπάτι	#εμφανίσεων στο μονοπάτι
1	∞	-1	0
2	∞	-1	0
3	∞	-1	0
4	∞	-1	0
5	∞	-1	0
6	∞	-1	0
7	∞	-1	0
8	∞	-1	0
9	∞	-1	0
10	∞	-1	0
11	∞	-1	0
12	∞	-1	0
13	∞	-1	0
14	∞	-1	0
15	∞	-1	0
16	∞	-1	0

Πίνακας 2.13

κόμβος	d'	επόμενος στο μονοπάτι	#εμφανίσεων στο μονοπάτι
1	∞	-1	0
2	∞	-1	0
3	∞	-1	0
4	∞	-1	0
5	∞	-1	0
6	∞	-1	0
7	∞	-1	0
8	∞	-1	0
9	∞	-1	0
10	∞	-1	0
11	1	14	0
12	∞	-1	0
13	∞	-1	0
14	∞	-1	0
15	∞	-1	0
16	∞	-1	0

Πίνακας 2.14

Να σημειώσουμε ότι το L_{min} είναι 5, όσο δηλαδή το μήκος του μονοπατιού 1 →14 που βρήκαμε παραπάνω. Έστω ότι β = 0.2. Συνεπώς, θέλουμε το μήκος του μονοπατιού να είναι μικρότερο από 6. Επίσης, θεωρούμε ότι αν ένας κόμβος έχει κενό σύνολο εξερχόμενων ακμών εξέρχεται από τη στοίβα, μαζί με την ακμή που οδήγησε σε αυτόν.

κόμβος	d'	επόμενος στο μονοπάτι	#εμφανίσεων στο μονοπάτι
1	∞	-1	0
2	∞	-1	0
3	∞	-1	0
4	∞	-1	0
5	∞	-1	0
6	∞	-1	0
7	∞	-1	0
8	∞	-1	0
9	**2**	**11**	**0**
10	**2**	**11**	0
11	1	14	**1**
12	∞	-1	0
13	∞	-1	0
14	∞	-1	0
15	∞	-1	0
16	∞	-1	0

Πίνακας 2.15

κόμβος	d'	επόμενος στο μονοπάτι	#εμφανίσεων στο μονοπάτι
1	∞	-1	0
2	∞	-1	0
3	∞	-1	0
4	∞	-1	0
5	∞	-1	0
6	∞	-1	0
7	∞	-1	0
8	**3**	**9**	0
9	2	11	**1**
10	2	11	0
11	1	14	1
12	∞	-1	0
13	∞	-1	0
14	∞	-1	0
15	∞	-1	0
16	∞	-1	0

Πίνακας 2.16

κόμβος πηγή	**μονοπάτι**
1	1→5→7→10→11→14
2	x
3	x
4	x
5	5→7→10→11→14
6	6→8→9→11→14
7	7→10→11→14
8	8→9→11→14
9	9→11→14
10	10→11→14
11	11→14
12	x
13	x
14	x
15	x
16	x

Πίνακας 2.17

Ξεκινάμε από τον κόμβο πηγή, δηλαδή τον 1. Τον τοποθετούμε στο μονοπάτι.

στοίβα κόμβων	στοίβα ακμών	κόμβος στην κορυφή
1	-	1
1,2,5	1→2,1→5	5
1,2,5,3,6,7	1→2,1→5,5→3,5→6,5→7	7
1,2,5,3,6,7,10	1→2,1→5,5→3,5→6,5→7,7→10	10
1,2,5,3,6,7,10,11,12	1→2,1→5,5→3,5→6,5→7,7→10,10→11,10→12	12
1,2,5,3,6,7,10,11	1→2,1→5,5→3,5→6,5→7,7→10,10→11	11
1,2,5,3,6,7,10,11,**14**	1→2,1→5,5→3,5→6,5→7,7→10,10→11,11→14	14

Πίνακας 2.18

Στο σημείο αυτό, προσθέτουμε στο σύνολο των συσχετίσεων ένα ακόμα στοιχείο. Ο αλγόριθμος δεν το ξεκαθαρίζει, αλλά θεωρούμε ότι δεν βάζουμε αυτούσια τη στοίβα με τις ακμές στις συσχετίσεις, αλλά κρατάμε μόνο την αλληλουχία των ακμών που οδηγούν από το 1 (κόμβος Α) στο 14 (κόμβος Β).

Συνεχίζοντας, βρίσκουμε όλα τα μονοπάτια από το 1 στο 14 που έχουν μήκος μικρότερο από 6.

Τα μονοπάτια αυτά είναι:

1→5→7→10→11→14 (μήκος 5)

1→5→6→7→10→11→14 (μήκος 7)

1→5→6→8→9→11→14 (μήκος 7)

Από αυτά τα μονοπάτια μόνο ένα έχει μήκος μικρότερο από 6, γι'αυτό δεν τα δεχόμαστε όλα, αλλά κρατάμε μόνο το μονοπάτι που ικανοποιεί τη συνθήκη που αναφέραμε.

Εύρεση συντομότερων μονοπατιών υπό συνθήκη και παραλλαγές [CW03]

Σκοπός της μεθόδου που θα περιγράψουμε εδώ είναι να βρει όλα τα μονοπάτια από ένα κόμβο Α σε ένα κόμβο Β, τα οποία έχουν τις εξής ιδιότητες: α) δεν έχουν κύκλους, β) έχουν θετικά βάρη στις ακμές τους και γ) το μήκος των μονοπατιών αυτών είναι μικρότερο από το (1+ε)*(μήκος_συντομότερου_μονοπατιού), όπου το ε είναι ορισμένο από το χρήστη και ε μεγαλύτερο ή ίσο με το 0.

Ο βασικός αλγόριθμος στον οποίο στηρίζεται η μέθοδος είναι ο αλγόριθμος των Byers&Waterman [BW84] και ο οποίος θα περιγραφεί στη συνέχεια. Πάνω σε αυτό τον αλγόριθμο έχουν υλοποιηθεί δύο βασικές κατηγορίες αλγορίθμων:

1) ο αλγόριθμος εύρεσης όλων των συντομότερων μονοπατιών που ικανοποιούν τη συνθήκη

(μήκος_μονοπατιού) < (1+ε)*(μήκος_συντομότερου_μονοπατιού)

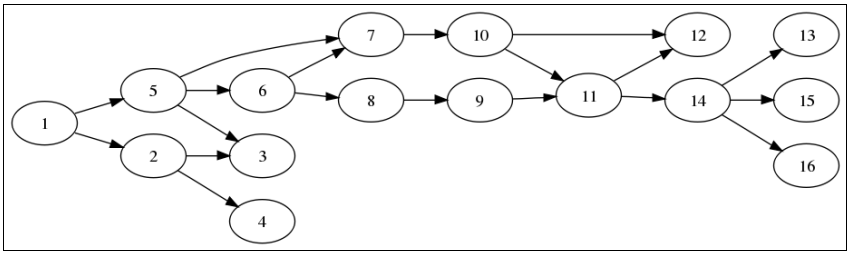
και

2) ο αλγόριθμος εύρεσης των Κ συντομότερων από τα μονοπάτια που βρίσκει η κατηγορία 1). Ο

αλγόριθμος αυτός χρησιμοποιεί δυαδική αναζήτηση για να βρει τα Κ μονοπάτια που θέλουμε.

Παράδειγμα 2.7:

Έστω ο γράφος του Σχήματος 2.14:



Σχήμα 2.14

Θέλουμε να βρούμε τα μονοπάτια από 1 μέχρι 12 που το μήκος τους είναι μικρότερο από

$$(1+\varepsilon)*(\text{μήκος\_συντομότερου\_μονοπατιού}).$$

Επιλέγουμε ε = 0.4. Έστω ότι έχουμε βρει τα συντομότερα μονοπάτια και τα μήκη τους. Το μήκος_συντομότερου_μονοπατιού είναι 5. Συνεπώς η συνθήκη είναι:

$$(\text{μήκος\_μονοπατιού}) < (1+0.4)*5 = 7$$

συνεχίζοντας με αυτόν τον τρόπο, βρίσκουμε το μονοπάτι

$$1 \rightarrow 5 \rightarrow 7 \rightarrow 10 \rightarrow 11 \rightarrow 14$$

ενώ δεν βρίσκουμε τα μονοπάτια που έχουν μήκος μεγαλύτερο από το δοθέν όριο.

στοίβα	τί ισχύει	τι κάνουμε
1	non-visited children του 1 ={2,5}, αν προσθέσουμε το 2 στο μονοπάτι δεν παραβιάζεται η συνθήκη	προσθέτουμε το 2 στο μονοπάτι και σημειώνουμε την παρουσία του σ'αυτό
1,2	non-visited children του 2 ={3,4}, αν προσθέσουμε το 3 στο μονοπάτι δεν παραβιάζεται η συνθήκη	προσθέτουμε το 3 στο μονοπάτι και σημειώνουμε την παρουσία του σ'αυτό
1,2,3	το 3 δεν έχει παιδιά	βγάζουμε το 3 από το μονοπάτι
1,2	non-visited children του 2 ={4}, αν προσθέσουμε το 4 στο μονοπάτι δεν παραβιάζεται η συνθήκη	προσθέτουμε το 4 στο μονοπάτι και σημειώνουμε την παρουσία του σ'αυτό
1,2,4	το 4 δεν έχει παιδιά	βγάζουμε το 4 από το μονοπάτι
1,2	το 2 δεν έχει άλλα παιδιά που να μην έχουμε επισκευθεί	βγάζουμε το 2 από το μονοπάτι
1	non-visited children του 1 ={5}, αν προσθέσουμε το 5 στο μονοπάτι δεν παραβιάζεται η συνθήκη	προσθέτουμε το 5 στο μονοπάτι και σημειώνουμε την παρουσία του σ'αυτό
1,5	non-visited children του 5 ={3,6,7}, αν προσθέσουμε το 3 στο μονοπάτι δεν παραβιάζεται η συνθήκη	προσθέτουμε το 3 στο μονοπάτι και σημειώνουμε την παρουσία του σ'αυτό
1,5,3	το 3 δεν έχει παιδιά	βγάζουμε το 3 από το μονοπάτι
1,5	non-visited children του 5 ={6,7}, αν προσθέσουμε το 6 στο μονοπάτι δεν παραβιάζεται η συνθήκη	προσθέτουμε το 6 στο μονοπάτι και σημειώνουμε την παρουσία του σ'αυτό
1,6,7	non-visited children του 6 ={7,8}, αν προσθέσουμε το 7 στο μονοπάτι δεν παραβιάζεται η συνθήκη	προσθέτουμε το 7 στο μονοπάτι και σημειώνουμε την παρουσία του σ'αυτό
κλπ...	κλπ...	κλπ...

Πίνακας 2.19

2.5 Συμπεράσματα

Συγκεντρώσαμε τα πλεονεκτήματα και τα μειονεκτήματα της κάθε μεθόδου και τα εμφανίζουμε στον Πίνακα 2.10:

Κάθε μέθοδος αναφέρεται σε συγκεκριμένα προβλήματα. Συνεπώς, κάθε μία είναι αποτελεσματική για την αντίστοιχη εφαρμογή. Ωστόσο, καλό θα ήταν να έχουμε πιο γενικές και ευέλικτες μεθόδους, προκειμένου να μπορούμε να επιλύσουμε και τις νέες εφαρμογές που θα αναπτυχθούν.

Μέθοδος	εύρεση παιδιών	εύρεση απογόνων	εύρεση συντομότερου μονοπατιού	πλεονεκτήματα	μειονεκτήματα
[SL97]	ναι	όχι	όχι	1.Πολύ καλή για συναθροιστικά ερωτήματα και για το ερώτημα αποτίμησης διαδρομής 2.Δίνει σημασία στην απόδοση προσπέλασης κόμβων	1. δεν λαμβάνει υπόψη το κόστος για τη διαμέριση 2. εξειδικευμένη μέθοδος
[LD89]	Ναι(εμμέσως)	ναι	όχι	1. εύκολη αναζήτηση των κόμβων λόγω της τοπολογικής διάταξης 2. δυνατότητες επέκτασης της μεθόδου 3.ασχολείται με τον τρόπο αποθήκευσης των δεδομένων	1. απαιτείται προεπεξεργασία για την κατασκευή των αρχείων 2. απαιτείται περαιτέρω πειραματική ανάλυση για περισσότερες παραμέτρους και και καλυτερες ευριστικές
[ABJ89]	Ναι(εμμέσως)	ναι	όχι	1.χαμηλές απαιτήσεις σε χώρο 2.ασχολείται με τον τρόπο αποθήκευσης των δεδομένων	έχουν γίνει αρκετές παραδοχές, οι οποίες ίσως αγνοούν επεξεργασία που επιφέρει υψηλό κόστος σε χώρο και χρόνο.
[L87]	Ναι(εμμέσως)	ναι	όχι	1. μείωση χρόνου που χρειάζεται για την εύρεση του αποτελέσματος 2. μείωση των I/O στο δίσκο 3. μείωση της αρχικής σχέσης 4. δυνατότητα επέκτασης σε κατανεμημένα ή παράλληλα συστήματα	Έχουν γίνει αρκετές παραδοχές οι οποίες καθιστούν μη πλήρη την ανάλυση
[AJ88]	Ναι(εμμέσως)	ναι	όχι	1.χαμηλές απαιτήσεις σε χώρο 2.Δίνει σημασία στην απόδοση προσπέλασης κόμβων	1. η απόδοση εξαρτάται από τις τιμές των παραμέτρων που επιλέγουμε 2. απαιτούνται πιό έξυπνες ευριστικές
[TU]	όχι	όχι	ναι	1.πολύ καλή απόδοση για μεγάλους γράφους και όχι για μικρούς. 2.απαιτεί περίπου τον ίδιο χρόνο για τα διάφορα ερωτήματα. 3.ταξινομημένα αποτελέσματα	1.δεν γνωρίζουμε τί γίνεται όταν το μονοπάτι να έχει πολύ μεγάλο μήκος ή όταν δεν υπάρχει συσχέτιση ανάμεσα σε δύο κόμβους. 2.δεν ασχολείται καθόλου με γράφους που έχουν κύκλους.
[CW03]	όχι	όχι	ναι	μπορεί να επεκταθεί και σε μη κατευθυνόμενους γράφους, ή γράφους με κύκλους και αρνητικά βάρη στις ακμές τους.	1.εξάρτηση της απόδοσης από τη δομή του γράφου που δεχόμαστε ως είσοδο. 2. η απόδοση δεν είναι πολύ καλή, αφού συχνά χάνεται χρόνος στην εξερεύνηση διαδρομών που έχουν ήδη εξερευνηθεί προηγουμένως και οι οποίες δεν οδηγούν σε κάποιο αποτέλεσμα.

Πίνακας 2.20

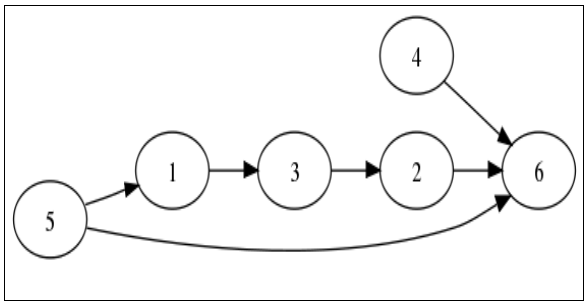
3

Θεωρητικό υπόβαθρο

Στο κεφάλαιο αυτό θα περιγράψουμε τις έννοιες και τις τεχνικές που είναι σχετικές με το αντικείμενο της διπλωματικής εργασίας. Η γνώση αυτών των εννοιών είναι απαραίτητη για την κατανόηση των όσων θα ακολουθήσουν.

3.1 Γράφοι και σχετικές έννοιες

Ορισμός 3.1. *Γράφος είναι ένα σύνολο G= (V,E) όπου V είναι ένα πεπερασμένο, μη κενό σύνολο από κόμβους / κορυφές και E είναι ένα σύνολο από ακμές (σύνδεσμοι ανάμεσα σε ζευγάρια κόμβων).*

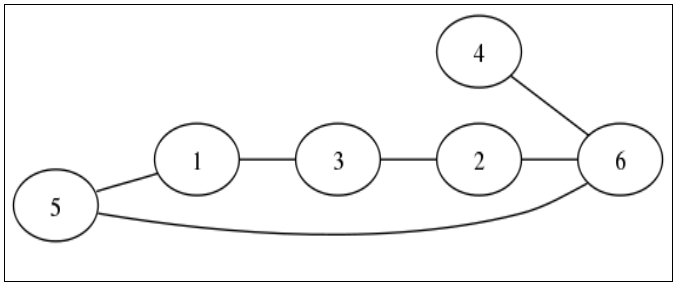


Σχήμα 3.1

Στο Σχήμα 3.1 παρουσιάζεται ένα παράδειγμα γράφου. Το σύνολο κόμβων V αποτελείται από τους κόμβους 1, 2, 3, 4, 5 και 6, ενώ το σύνολο ακμών E περιέχει τα ζευγάρια κόμβων (1,3), (3,2), (2,6), (5,1), (5,6), (4,6).

Ορισμός 3.2. *Όταν οι ακμές σε ένα γράφο δεν έχουν κατεύθυνση, ο γράφος ονομάζεται μη κατευθυνόμενος, αλλιώς λέγεται κατευθυνόμενος*.

Ο γράφος του Σχήματος 3.1 είναι κατευθυνόμενος. Αντίθετα, ο γράφος του Σχήματος 3.2 είναι μη κατευθυνόμενος.



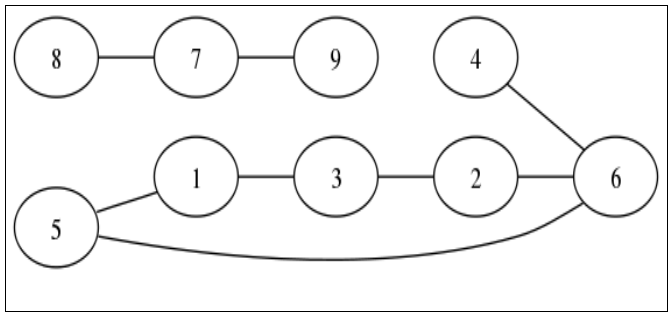
Σχήμα 3.2

Ορισμός 3.3. *Μονοπάτι είναι μια ακολουθία από κόμβους, τέτοια ώστε για κάθε κόμβο να υπάρχει μια ακμή προς τον επόμενο κόμβο της ακολουθίας. Ο πρώτος κόμβος λέγεται αρχικός κόμβος και ο τελευταίος λέγεται τελικός κόμβος. Και οι δύο αυτοί κόμβοι λέγονται ακραίοι κόμβοι του μονοπατιού. Οι υπόλοιποι λέγονται ενδιάμεσοι κόμβοι.*

Στο γράφο του Σχήματος 3.1 το μονοπάτι που συνδέει τους κόμβους 1 και 6 είναι 1->3->2->6

Ορισμός 3.4. *Ένας μη κατευθυνόμενος γράφος στον οποίο δεν υπάρχει ζευγάρι κόμβων που να συνδέονται με παραπάνω από ένα μονοπάτι, ονομάζεται ακυκλικός. Ένας κατευθυνόμενος γράφος είναι ακυκλικός, αν δεν υπάρχει μονοπάτι του με αρχή και τέλος τον ίδιο κόμβο."*

Ο γράφος του Σχήματος 3.2 περιέχει κύκλους. Ο γράφος του Σχήματος 3.1 δεν περιέχει κύκλους, αφού δεν μπορούμε να καταλήξουμε σε κάποιο κόμβο ξεκινώντας από τον ίδιο.



Σχήμα 3.3

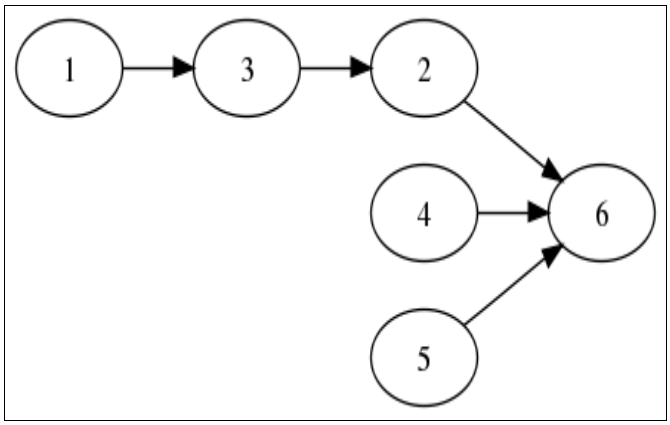
Ο γράφος του Σχήματος 3.3 είναι μη συνεκτικός αφού π.χ. για το ζευγάρι κόμβων 1-8 δεν υπάρχει μονοπάτι που να τους συνδέει. Οι συνεκτικές συνιστώσες του είναι οι ομάδες κόμβων {1,2,3,4,5,6} και {7,8,9}.

Ορισμός 3.5. *Αν σε ένα γράφο, για κάθε ζευγάρι κόμβων υπάρχει ένα μονοπάτι που τους συνδέει, τότε ο γράφος ονομάζεται συνεκτικός. Ένας μη συνεκτικός γράφος αποτελείται από τμήματα που ονομάζονται συνεκτικές συνιστώσες.*

Ορισμός 3.6. *Ζευγνύον δέντρο (spanning tree) ενός συνεκτικού γράφου είναι το μέγιστο σύνολο από ακμές του γράφου που δεν περιέχει κύκλους. Ισοδύναμα, ζευγνύον δέντρο ενός γράφου είναι το*

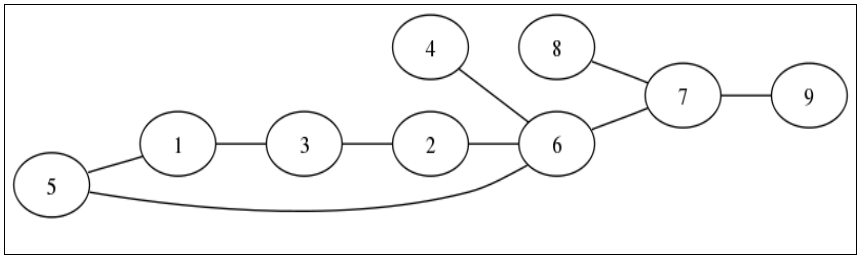
μικρότερο σύνολο από ακμές του γράφου που δεν συνδέουν όλες τις κορυφές του.

Ένα πιθανό ζευγνύον δέντρο του γράφου του Σχήματος 3.1 φαίνεται στο Σχήμα 3.4:

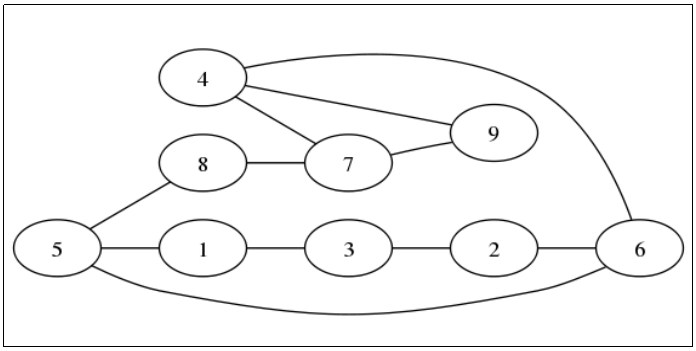


Σχήμα 3.4

Ορισμός 3.7. *Συνεκτικότητα (connectivity) ενός γράφου είναι ο ελάχιστος αριθμός ακμών που απαιτείται να αφαιρεθούν ώστε ο γράφος να πάψει να είναι συνεκτικός.*



Σχήμα 3.5

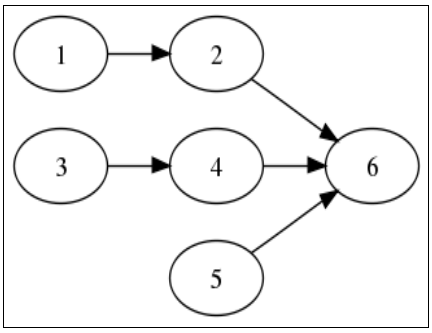


Σχήμα 3.6

Η συνεκτικότητα του γράφου του Σχήματος 3.5 είναι ίση με 1, αφού αν βγάλουμε μία ακμή (π.χ. την 6-7) ο γράφος θα μετατραπεί σε μη συνεκτικό. Αντίθετα, ο γράφος του Σχήματος 3.6 έχει συνεκτικότητα ίση με 2, αφού αρκεί να αφαιρεθούν 2 ακμές (π.χ. 1-5 και 2-6 για να γίνει ο γράφος μη συνεκτικός.

Ορισμός 3.8. *Τοπολογική σειρά (topological order) ενός γράφου είναι μια γραμμική ταξινόμηση όλων των κόμβων του, έτσι ώστε αν ο γράφος περιέχει μια ακμή (u,v), τότε ο u εμφανίζεται πριν από τον v*

στην ταξινόμηση.



Σχήμα 3.7

Στο γράφο του Σχήματος 3.7 οι κόμβοι διατεταγμένοι με τοπολογική σειρά είναι οι εξής:

{1,2,3,4,5,6}

Ορισμός 3.9. *Ένα δίκτυο στο οποίο οι ακμές που συνδέουν τους κόμβους σχετίζονται με τη θέση των κόμβων στο χώρο και παίρνουν τιμές με βάση αυτή, λέγεται χωρικό δίκτυο (Spatial Network).*

Στο Σχήμα 3.8 παραθέτουμε ένα παράδειγμα χωρικού δικτύου. Το δίκτυο αυτό βρίσκεται στο γεωγραφικό χώρο της Ελλάδας και οι κόμβοι του αντιστοιχούν σε μεγάλες πόλεις.

Σχήμα 3.8

Ορισμός 3.10. *Η κλειστότητα (Transitive closure) ενός γράφου είναι το σύνολο όλων των απογόνων όλων των κόμβων του.*

Αξίζει να επισημάνουμε ότι η έννοια της κλειστότητας γενικά συνδέεται με ένα γράφο και όχι με ένα μεμονωμένο κόμβο του. Αντιμετωπίζει το γράφο ως σχέση (relation: σύνολο από ζευγάρια κόμβων) και είναι στην ουσία, όλα τα ζευγάρια κόμβων που συνδέονται μεταξύ τους με μία ή περισσότερες ακμές.

Για παράδειγμα, ο γράφος του Σχήματος 3.1 αναπαρίσταται ως:

{(1,3),(2,6),(3,2),(4,6),(5,1),(5,6)}

Η κλειστότητα του γράφου του Σχήματος 3.1 είναι το σύνολο:

{(1,3),(2,6),(3,2),(4,6),(5,1),(5,6),(1,2),(1,6),(3,6),(5,2),(5,3)}

Ορισμός 3.11. *Ερώτημα (query) σε γράφο είναι η αναζήτηση κόμβων (μεμονωμένων ή ως σύνολα) με συγκεκριμένα χαρακτηριστικά.*

Ένα παράδειγμα είναι η αναζήτηση στο γράφο του Σχήματος 3.1 των κόμβων με i περιττό. Η απάντηση σε αυτό είναι {1,3,5}.

Ορισμός 3.12. *Το ερώτημα σχετικά με το αν δύο κόμβοι που ανήκουν σε ένα γράφο συνδέονται με κάποιο μονοπάτι ονομάζεται προσβασιμότητα (reachability).*

Ορισμός 3.13. *Ένα ερώτημα κατά το οποίο η επεξεργασία κάθε εγγραφής ή κόμβου είναι το αποτέλεσμα της επεξεργασίας των παιδιών του, ονομάζεται αναδρομικό ερώτημα (recursive query).*

Παράδειγμα αναδρομικού ερωτήματος είναι η αναζήτηση των απογόνων ενός κόμβου. Στην αρχή βρίσκουμε τους άμεσους απογόνους του κόμβου, δηλ. τα παιδιά του, στη συνέχεια τα παιδιά των παιδιών του κ.ο.κ.

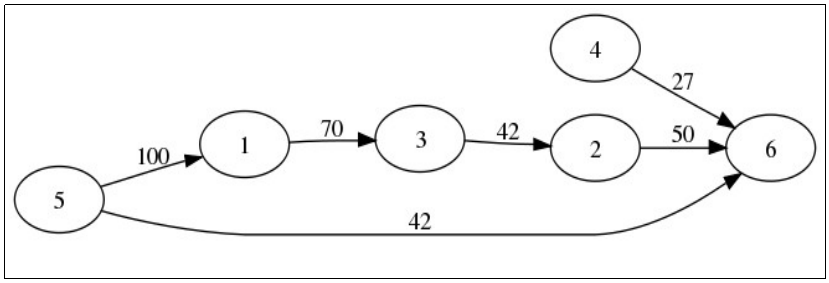
Ορισμός 3.14. *Η αναδρομική διάσχιση (Traversal Recursion) είναι μια κλάση αναδρομικών ερωτημάτων που περιλαμβάνουν διάσχιση ενός γράφου ή δέντρου.*

Ένας άλλος τύπος ερωτημάτων σχετικά με γράφους είναι τα λεγόμενα συναθροιστικά ερωτήματα (Aggregate queries).
Ορισμός 3.15. *Συναθροιστικό είναι το ερώτημα που αναφέρεται σε ομάδες κόμβων που έχουν μια επιθυμητή τιμή για κάποιο χαρακτηριστικό.*

Για παράδειγμα, στο γράφο του Σχήματος 3.9 θέλουμε να βρούμε το μέσο κόστος ανά ακμή του μονοπατιού από τον κόμβο 1 στον κόμβο 6. Η απάντηση είναι (70 + 42 + 50) / 3 = 162 / 3 = 54 .

Ορισμός 3.16. *Ένα ερώτημα αποτίμησης διαδρομής (route evaluation), είναι το ερώτημα το οποίο συνίσταται στον υπολογισμό της τιμής ενός χαρακτηριστικού των ακμών ενός κόμβου που συμμετέχει σε κάποιο μονοπάτι του γράφου.*

Για το γράφο του Σχήματος 3.9, ένα παράδειγμα ερωτήματος αποτίμησης διαδρομής είναι ο υπολογισμός του συνολικού κόστους της διαδρομής 3 – 6. Η απάντηση εδώ είναι 92.



Σχήμα 3.9

3.2 Τεχνικές αναπαράστασης γράφων

Αναπαράσταση ενός γράφου είναι ο τρόπος με τον οποίο αποθηκεύουμε το γράφο στη μνήμη ή το δίσκο του υπολογιστή, χρησιμοποιώντας δομές, όπως λίστες, πίνακες κλπ. Ένας γράφος μπορεί να δίνεται ως ένα σύνολο ακμών και ένα σύνολο κόμβων με συγκεκριμένες ιδιότητες, ως σύνολο από μονοπάτια, ως ομάδες κόμβων ή ακμών, οι οποίες προκύπτουν με βάση κάποιο χαρακτηριστικό τους. Ανάλογα με την αναπαράσταση ενός γράφου, χρησιμοποιούνται τα κατάλληλα εργαλεία και αναπτύσσονται τεχνικές προσαρμοσμένες στη δοθείσα μορφή.

Οι πιο γνωστές τεχνικές αναπαράστασης γράφων είναι:

- οι λίστες γειτνίασης (adjacency lists). Οι λίστες γειτνίασης είναι ιδιαίτερα αποδοτική για αραιούς γράφους.
- ο πίνακας γειτνίασης (adjacency matrix). Ο πίνακας γειτνίασης είναι μια πολύ χρήσιμη αναπαράσταση όταν θέλουμε να βρούμε αν υπάρχει ακμή ανάμεσα σε δύο κόμβους. Επιπλέον, χρησιμοποιώντας πίνακα γειτνίασης κερδίζουμε χώρο, ιδιαίτερα για πυκνούς γράφους. Οι πίνακες γειτνίασης είναι πολύ καλή αναπαράσταση για μικρούς γράφους.
- ο πίνακας πρόσπτωσης (incidence matrix).

Έστω ένας γράφος G(V,E) με $V = \{u_1,u_2,...,u_N\}$ και $E = \{l_1,l_2,...,l_M\}$.

Ορισμός 3.17. *(Αναπαράσταση με λίστες γειτνίασης). Σε κάθε κόμβο του γράφου αντιστοιχεί μία λίστα με τους γειτονικούς του κόμβους*

Οι λίστες γειτνίασης για το γράφο του Σχήματος 3.10 είναι οι παρακάτω:

1→{3,5}

2→{3,6}

3→{1,2}

4→{6,7,9}

5→{1,6,8}

6→{4,5}

7→{4,,9}

8→{5,7}

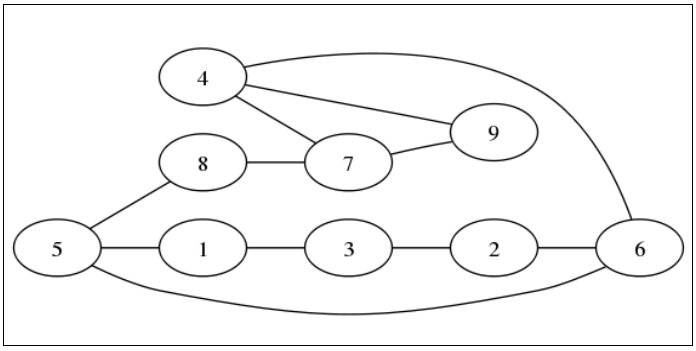
Ορισμός 3.18. *Ο πίνακας γειτνίασης είναι ένας n x n πίνακας A(G), όπου*

$$A(G)=[a_{ij}], a_{ij}=\begin{cases}1, \alpha\nu (u_i, u_j)\in E\\ 0, \alpha\lambda\lambda\iota\acute{\omega}\varsigma\end{cases}$$

και είναι συμμετρικός ($a_{ij} = a_{ji}$).

Για το γράφο του Σχήματος 3.10 ο πίνακας γειτνίασης είναι ο:

$$A(G)=\begin{bmatrix}0&0&1&0&1&0&0&0&0\\0&0&1&0&0&1&0&0&0\\1&1&0&0&0&0&0&0&0\\0&0&0&0&0&1&1&0&1\\1&0&0&0&0&1&0&1&0\\0&1&0&1&1&0&0&0&0\\0&0&0&1&0&0&0&1&1\\0&0&0&0&1&0&1&0&0\\0&0&0&1&0&0&1&0&0\end{bmatrix}$$



Σχήμα 3.10

Ορισμός 3.19. *Ο πίνακας πρόσπτωσης είναι ένας n x m πίνακας B(G), όπου*

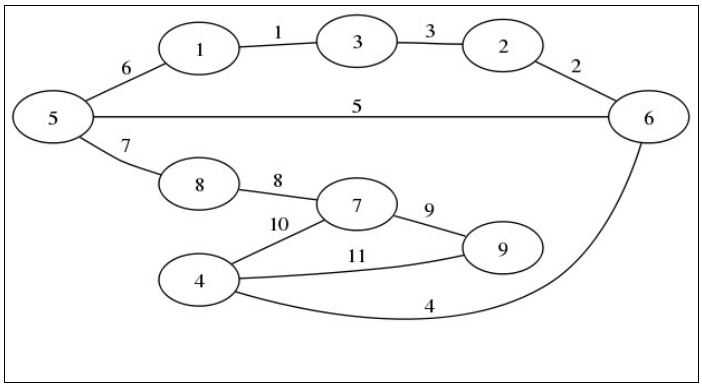
$$B(G)=[b_{ij}], b_{ij}=\begin{cases}1, \alpha\nu\, l_j \pi\rho o\sigma\pi\acute{\iota}\pi\tau\epsilon\iota \sigma\tau o\, u_i\\ 0, \alpha\lambda\lambda\iota\acute{\omega}\varsigma\end{cases}$$

Για το γράφο του Σχήματος 3.11 ο πίνακας πρόσπτωσης είναι παρακάτω:

$$B(G)=\begin{bmatrix}1&0&0&0&0&1&0&0&0&0&0\\0&1&1&0&0&0&0&0&0&0&0\\1&0&1&0&0&0&0&0&0&0&0\\0&0&0&1&0&0&0&0&0&1&1\\0&0&0&0&1&1&1&0&0&0&0\\0&1&0&1&1&0&0&0&0&0&0\\0&0&0&0&0&0&0&1&1&1&0\\0&0&0&0&0&0&1&1&0&0&0\\0&0&0&0&0&0&0&0&1&0&1\end{bmatrix}$$

Άλλες αναπαραστάσεις γράφων είναι

- οι λίστες πρόσπτωσης, που είναι η αντίστοιχη με τον πίνακα πρόσπτωσης αναπαράσταση και περιλαμβάνει λίστες από ακμές που καταλήγουν/ξεκινούν από τον κάθε κόμβο του γράφου.
- ο πίνακας εισόδου ή πίνακας του Kirchhoff, που χρησιμοποιείται στην ανάλυση κυκλωμάτων.
- ο πίνακας αποστάσεων, ο οποίος περιέχει την απόσταση ανάμεσα σε κάθε κόμβο του γράφου και σε κάθε έναν από τους υπόλοιπους κόμβους του γράφου.



Σχήμα 3.11

3.3 Τεχνικές αναζήτησης σε γράφους (dfs και bfs)

Στη συνέχεια θα περιγράψουμε δύο πολύ βασικούς αλγόριθμους αναζήτησης, τους οποίους χρησιμοποιήσαμε σαν βάση για την απάντηση των ερωτημάτων με τα οποία ασχοληθήκαμε. Οι αλγόριθμοι αυτοί είναι ο αλγόριθμος αναζήτησης κατά βάθος και ο αλγόριθμος αναζήτησης κατά πλάτος.

Η γενική ιδέα των αλγορίθμων αναζήτησης είναι η εξής:

Έχουμε ένα σύνολο κόμβων προς μελέτη, το οποίο ονομάζουμε μέτωπο αναζήτησης. Το μέτωπο αναζήτησης περιέχει αρχικά μόνο ένα κόμβο, έστω τον κόμβο Α. Σε κάθε επανάληψη αφαιρούμε ένα κόμβο από το μέτωπο αναζήτησης και προσθέτουμε σ' αυτό όσα από τα παιδιά του δεν έχουμε επισκευθεί. Σημειώνουμε με κάποιο τρόπο ότι έχουμε επισκεφθεί τον κόμβο που βγάζουμε κάθε φορά από το μέτωπο αναζήτησης. Μπορούμε, είτε να ζητάμε ένα συγκεκριμένο κόμβο-στόχο, έστω Β, είτε να εξερευνήσουμε όλο το γράφο για να υπολογίσουμε την τιμή κάποιας ιδιότητας των κόμβων.

Αλγόριθμος Αναζήτησης κατά Βάθος (dfs)

Ο αλγόριθμος dfs χρησιμοποιεί ως μέτωπο αναζήτησης μια στοίβα, όπου τοποθετούμε τους κόμβους στην κορυφή της και τους παίρνουμε από την αρχή. Για να γνωρίζουμε αν έχουμε επισκευθεί κάποιο κόμβο χρησιμοποιούμε χαρακτηριστικά «χρώματα», τα οποία χαρακτηρίζουν τους κόμβους. Έτσι, χρωματίζουμε έναν κόμβο ΑΣΠΡΟ, αν δεν τον έχουμε επισκεφθεί, ΓΚΡΙ, αν τον έχουμε επισκευθεί και ΜΑΥΡΟ, αν έχουμε επισκεφθεί όλα τα παιδιά του.

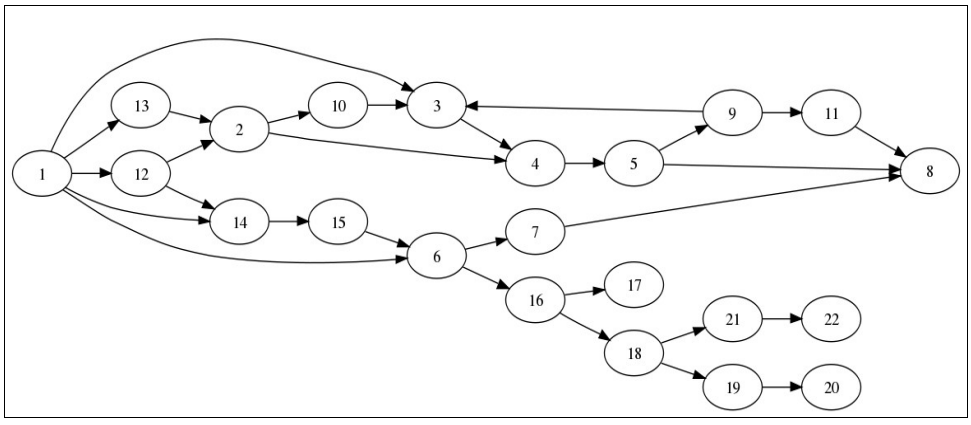
```
χρωματίζουμε όλους τους κόμβους ΑΣΠΡΟΥΣ/*δεν τους έχουμε επισκεφθεί*/
βάζουμε τον πρώτο κόμβο στη στοίβα
όσο η στοίβα δεν είναι άδεια
{
      u = κορυφή της στοίβας
      αφαιρούμε το u από τη στοίβα και το χρωματίζουμε ΓΚΡΙ
      για κάθε παιδί του u
      {
            αν δεν το έχουμε επισκεφθεί(χρώμα:ΑΣΠΡΟ)
                  το βάζουμε στη στοίβα
            και σημειώνουμε το u ως πατέρα του
      }
      αν έχουμε επισκεφθεί όλα τα παιδιά του
            το χρωματίζουμε ΜΑΥΡΟ
}
```

Ο αλγόριθμος αυτός μπορεί να εμπλουτιστεί με κατάλληλους ελέγχους ώστε να βρίσκει κόμβους με διάφορες ιδιότητες και να εκτυπώνει το μονοπάτι που οδηγεί στη λύση.

Ο dfs έχει μικρές απαιτήσεις σε χώρο. Η πολυπλοκότητα χώρου είναι O(L), όπου L είναι το μεγαλύτερο σε μήκος απλό (δηλαδή χωρίς κύκλους) μονοπάτι του γράφου, ενώ η πολυπλοκότητα χρόνου είναι O(V+E), όπου V και E είναι το πλήθος των κόμβων και των ακμών του γράφου αντίστοιχα.

Παράδειγμα.

Έστω ο γράφος του Σχήματος 3.12. Ξεκινάμε από τον κόμβο 1 και θέλουμε να εξερευνήσουμε όλους τους κόμβους του γράφου. Βλέπουμε τη συνέχεια στον Πίνακα 3.1. Επειδή το 6 είναι ΜΑΥΡΟ, δεν θα βάλουμε τα παιδιά του στη στοίβα. Συνεχίζουμε έτσι και με τους υπόλοιπους κόμβους, μέχρι να τους επισκεφθούμε όλους.



Σχήμα 3.12

κόμβος u[χρώμα]	στοίβα	κόμβοι που έχουμε επισκευθεί[πατέρας]
1[A]	3,13,12,14,6	-
6[A]	3,13,12,14,7,16	1[-]
16[A]	3,13,12,14,7,18,17	1[-],6[1]
17[A]	3,13,12,14,7,18	1[-],6[1],16[6]
18[A]	3,13,12,14,7,21,19	1[-],6[1],16[6],17[16]
19[A]	3,13,12,14,7,21,20	1[-],6[1],16[6],17[16],18[17]
20[A]	3,13,12,14,7,21	1[-],6[1],16[6],17[16],18[17],19[18]
21[A]	3,13,12,14,7,22	1[-],6[1],16[6],17[16],18[17],19[18],20[19]
22[A]	3,13,12,14,7	1[-],6[1],16[6],17[16],18[17],19[18],20[19],21[20]
7[A]	3,13,12,14,8	1[-],6[1],16[6],17[16],18[17],19[18],20[19],21[20],22[21]
8[A]	3,13,12,14	1[1],6[1],16[6],17[16],18[17],19[18],20[19],21[20],22[21],7[6]
14[A]	3,13,12,15	1[-],6[1],16[6],17[16],18[17],19[18],20[19],21[20], 22[21],7[6],8[7]
15[A]	3,13,12,6	1[-],6[1],16[6],17[16],18[17],19[18],20[19],21[20], 22[21],7[6],8[7],14[1]
6[M]	3,13,12	1[-],6[1],16[6],17[16],18[17],19[18],20[19],21[20],22[21],7[6],8[7],15[1]
...	...	κλπ...

Πίνακας 3.1

Αλγόριθμος Αναζήτησης κατά Πλάτος (bfs)

Ο αλγόριθμος bfs χρησιμοποιεί ως μέτωπο αναζήτησης μια ουρά, όπου τοποθετούμε τους κόμβους στο τέλος και τους παίρνουμε από την αρχή της.

```
χρωματίζουμε όλους τους κόμβους ΑΣΠΡΟΥΣ/*δεν τους έχουμε επισκευθεί*/
βάζουμε τον πρώτο κόμβο στην ουρά
όσο η ουρά δεν είναι άδεια
{
      u = πρώτο στοιχείο της ουράς

      για κάθε παιδί του u
      {
            αν δεν το έχουμε επισκεφθεί(χρώμα:ΑΣΠΡΟ)
                  το βάζουμε στην ουρά και το χρωματίζουμε ΓΚΡΙ
                  και σημειώνουμε το u ως πατέρα του
      }
      αφαιρούμε το u από την ουρά και το χρωματίζουμε ΜΑΥΡΟ
}
```

Ο bfs βρίσκει πάντα τη μικρότερη σε μήκος λύση, δηλαδή το μονοπάτι είναι το μικρότερο. Ωστόσο, η ουρά μπορεί να μεγαλώσει υπερβολικά, οπότε προκύπτει πρόβλημα λόγω μεγάλων απαιτήσεων σε μνήμη. Τα χρώματα που χρησιμοποιούμε είναι τα ίδια με αυτά που χρησιμοποιούμε και στην dfs. Η πολυπλοκότητα χώρου είναι O(V+E), ενώ η πολυπλοκότητα χρόνου είναι O(V+E).

Παράδειγμα 3.1.

Έστω και πάλι ο γράφος του Σχήματος 3.12.

Ξεκινάμε από τον κόμβο 1. Βλέπουμε τη συνέχεια στον Πίνακα 3.2.

κόμβος u[χρώμα]	ουρά	κόμβοι που έχουμε επισκευθεί[πατέρας]
1[A]	3,13,12,14,6	-
3[A]	13,12,14,6,4	1[-]
13[A]	12,14,6,4,2	1[-],3[1]
12[A] (*2[Γ]->δεν μπαινει στην ουρά*)	14,6,4,2	1[-],3[1],13[1]
14[A]	6,4,2,15	1[-],3[1],13[1],12[1]
6[A]	4,2,15,7,16	1[-],3[1],13[1],12[1],14[1]
4[A]	2,15,7,16,5	1[-],3[1],13[1],12[1],14[1],6[1]
...	...	κλπ...

Πίνακας 3.2

Συνεχίζοντας με αυτόν τον τρόπο, εξερευνούμε τελικά όλους τους κόμβους μέχρι να βρούμε τον κόμβο που ψάχνουμε ή μέχρι να τους επισκεφθούμε όλους.

4

Ορισμός Προβλήματος

Στο Κεφάλαιο αυτό δίνουμε μια παρουσίαση του προβλήματος το οποίο διαπραγματευτήκαμε κατά την εκπόνηση της διπλωματικής εργασίας. Στην Εισαγωγή κάνουμε μια γενική περιγραφή όσων κάναμε. Στη συνέχεια, δίνουμε τυπικούς ορισμούς για την αναπαράσταση των γράφων και για τα ερωτήματα που απαντήσαμε.

4.1 Εισαγωγή

Το πρόβλημα που μας απασχόλησε στην παρούσα διπλωματική εργασία είναι η απάντηση ερωτημάτων πάνω σε δεδομένα που είναι οργανωμένα σε γράφους. Ασχοληθήκαμε με την αποτίμηση ερωτημάτων εύρεσης μονοπατιών μεταξύ δύο κόμβων ενός γράφου. Μας ενδιαφέρουν και οι περιπτώσεις ερωτημάτων που εισάγουν περιορισμούς όσο αναφορά το μήκος των μονοπατιών. Σε κάθε περίπτωση θέλουμε και να υπολογίσουμε το μονοπάτι, δηλαδή να βρούμε ποιο είναι, εφ'όσον, φυσικά υπάρχει. Πιο συγκεκριμένα, τα ερωτήματα τα οποία απαντήσαμε είναι τα παρακάτω:

α) Δίνεται ένας κόμβος Α και ζητούνται όλα τα παιδιά του.

β) Δίνονται δύο κόμβοι Α και Β και ζητείται να βρούμε μονοπάτι ανάμεσα στους κόμβους Α και Β καθώς επίσης και να το υπολογίσουμε, εφ' όσον αυτό υπάρχει.

γ) Δίνονται δύο κόμβοι Α και Β και ζητείται να βρούμε μονοπάτι ανάμεσα στους κόμβους Α και Β που να ικανοποιεί κάποιο συγκεκριμένο περιορισμό, για παράδειγμα να έχει μήκος το πολύ 100 κόμβους καθώς επίσης και να το υπολογίσουμε, εφ' όσον αυτό υπάρχει.

Οι περισσότεροι από τους αλγόριθμους που συναντάμε στη βιβλιογραφία χρησιμοποιούν αναπαραστάσεις που περιγράφουν κάθε ακμή χωριστά. Έτσι, κάθε βήμα τους έχει μήκος μόνο μία ακμή και επισκέπτεται μόνο τα παιδιά ενός κόμβου. Αυτό που θα θέλαμε είναι να μειώσουμε τον

αριθμό των βημάτων που εκτελεί ο κάθε αλγόριθμος. Για να το κάνουμε αυτό θα πρέπει σε κάθε βήμα να μπορούμε να μελετάμε περισσότερες από μία ακμές, δηλαδή, να μην εξετάζουμε μόνο τα παιδιά ενός κόμβου, αλλά τους απογόνους του. Τους απογόνους ενός κόμβου μπορούμε να τους βρούμε εξετάζοντας τα μονοπάτια στα οποία ανήκει ο κόμβος αυτός. Καταλήγουμε, λοιπόν, στο συμπέρασμα ότι αν αναπαραστήσουμε το γράφο με μονοπάτια και όχι με μεμονωμένους κόμβους/ακμές, μπορούμε να πάρουμε περισσότερες πληροφορίες και να λύσουμε πιο γρήγορα το πρόβλημά μας. Η ιδέα μας, λοιπόν, είναι να αναπαραστήσουμε το γράφο που μας δίνεται με μονοπάτια, τα οποία αν συνδεθούν μας δίνουν ακριβώς το γράφο από τον οποίο προήλθαν.

Ο γράφος που αποτελεί είσοδο για τις μεθόδους που προτείνουμε είναι ένα αρχείο, στο οποίο τα μονοπάτια έχουν την εξής μορφή:

K1 K2 K3

K3 K4 K7

...

όπου K1, K2, K3,...,Kν είναι κόμβοι του γράφου.

Για το καθένα από τα ερωτήματα που απαντήσαμε:

α) Επιλέξαμε και υλοποιήσαμε κάποιους από τους ήδη υπάρχοντες αλγόριθμους που βρήκαμε από τη βιβλιογραφία

β) Προτείναμε και υλοποιήσαμε νέους αλγόριθμους

γ) Πραγματοποιήσαμε πειράματα προκειμένου να αξιολογήσουμε την επίδοση των νέων αλγορίθμων σε σχέση με τους κλασικούς και καταλήξαμε σε σχετικά συμπεράσματα.

Οι κλασικοί (ήδη υπάρχοντες αλγόριθμοι) που υλοποιήσαμε βασίζονται στη χρήση κατά βάθους και κατά πλάτους αναζήτησης πάνω σε αναπαραστάσεις γράφων με λίστες γειτνίασης.

4.2 Αναπαράσταση Γράφων

Όπως είπαμε και στην Ενότητα 4.1, η ιδέα στην οποία βασιζόμαστε είναι η ομαδοποίηση των κόμβων σε ομάδες, προκειμένου να απαντήσουμε πιο γρήγορα στα ερωτήματα που διαπραγματευόμαστε. Οι ομάδες που χρησιμοποιούμε είναι τα μονοπάτια του γράφου.

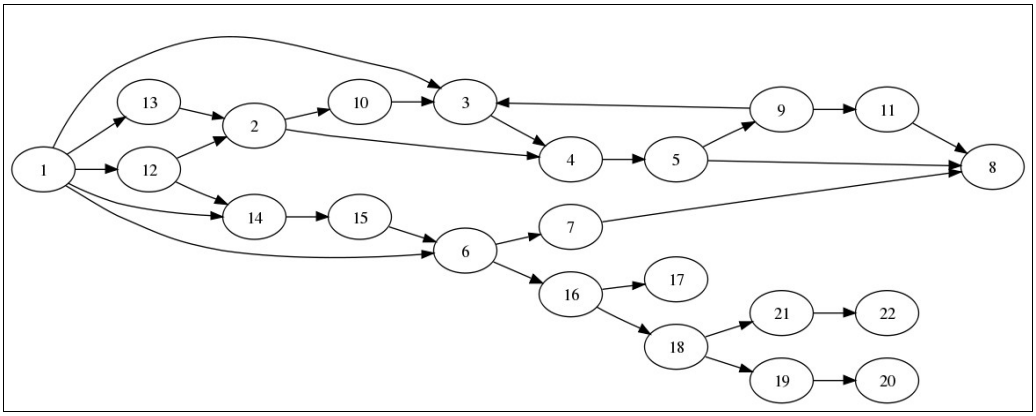
Ορισμός 4.1. *Έστω γράφος G(V,E). Η αναπαράσταση με μονοπάτια είναι ένα σύνολο G_P της μορφής*

$$G_P=\{P_1, P_2, P_3, \ldots, P_N\}$$

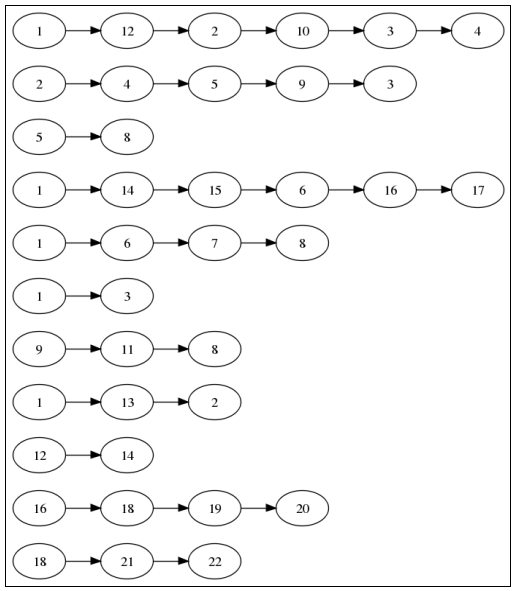
όπου $P_i = \{N_1,N_2,...,N_M\}$ είναι ένα μονοπάτι και N_i είναι ένας κόμβος του γράφους. Ισχύει $\bigcup_{i \in I} P_i$,

δηλαδή η ένωση των μονοπατιών μας δίνει το γράφο που αναπαρίσταται και οι κόμβοι κάθε μονοπατιού εμφανίζονται μία φορά σε αυτό.

Παράδειγμα 4.1. Έστω ο γράφος του Σχήματος 4.1. Μια πιθανή αναπαράστασή του σε μονοπάτια είναι αυτή του Σχήματος 4.2.



Σχήμα 4.1



Σχήμα 4.2

Στην ενότητα αυτή περιγράφουμε μια δομή η οποία συμπυκνώνει την πληροφορία που παίρνουμε από τα μονοπάτια και η οποία μας διευκολύνει στην αναζήτηση. Τη δομή αυτή την ονομάσαμε PATHINDEX. Ο PATHINDEX είναι ένα είδος ευρετηρίου, το οποίο για κάθε κόμβο του γράφου περιέχει όλα τα μονοπάτια στα οποία εμφανίζεται ο κόμβος, μαζί με οποιαδήποτε πληροφορία είναι χρήσιμη σε κάθε αλγόριθμο, όπως για παράδειγμα, η θέση του, ο κόμβος πατέρας του κλπ.

Ορισμός 4.2. *Ο PATHINDEX είναι ένα σύνολο από στοιχεία της μορφής (K→PathList), όπου* $K \in G_P$ *είναι ένας κόμβος του γράφου και PathList είναι μια λίστα με τα αναγνωριστικά των*

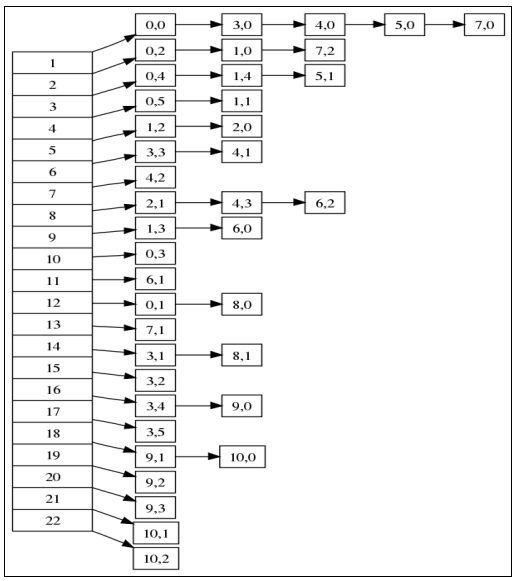
μονοπατιών στα οποία εμφανίζεται ο κόμβος Κ. Ο PATHINDEX περιέχει ακριβώς ένα στοιχείο για κάθε κόμβο Κ του γράφου.

Ο PATHINDEX έχει πλήθος εγγραφών:

(πλήθος ακμών G) + (πλήθος μονοπατιών G)

Αυτό σημαίνει ότι γενικά το μέγεθός του δεν είναι πολύ μεγάλο, επομένως κατασκευάζεται σχετικά σύντομα και δεν καταλαμβάνει πολύ χώρο.

Παράδειγμα 4.2. Ο PATHINDEX για το γράφο του Σχήματος 4.1 φαίνεται στο Σχήμα 4.3.



Σχήμα 4.3

4.3 Ερωτήματα

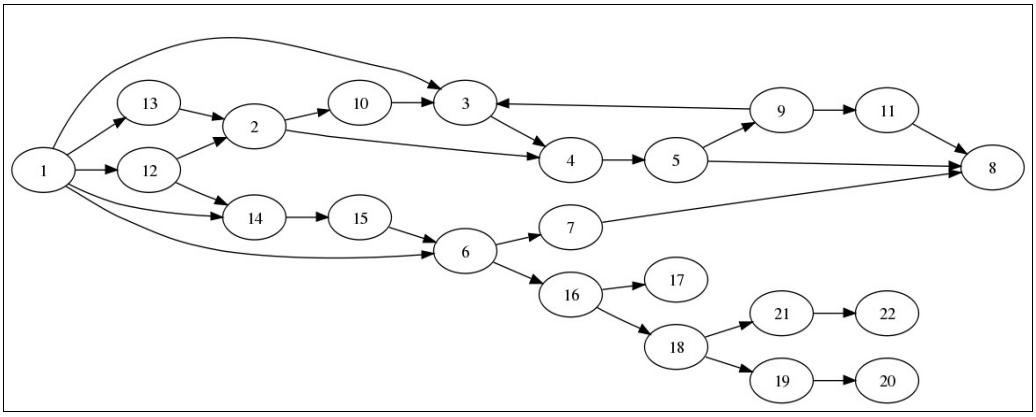
Ακολουθούν οι ορισμοί των ερωτημάτων τα οποία απαντήσαμε:

Εύρεση παιδιών ενός κόμβου Α

Στην περίπτωση του ερωτήματος αυτού ζητάμε να βρούμε όλα τα παιδιά, δηλαδή όλους τους άμεσους απογόνους ενός δοθέντος κόμβου Α.

Ορισμός 4.3. *Δίνεται ένας γράφος* $G=(V,E)$ *. Έστω Α κόμβος του G. Να βρεθούν όλοι οι κόμβοι Κ του G που είναι παιδιά του Α, δηλαδή για κάθε ζευγάρι (Α,Κ) υπάρχει ακμή* $e \in E$ *που να συνδέει τους κόμβους Α και Κ.*

Παράδειγμα 4.3. Τα παιδιά του κόμβου 1 στο Σχήμα 4.4 είναι οι κόμβοι { 3, 6, 12, 13, 14}.



Σχήμα 4.4

Εύρεση μονοπατιού ανάμεσα σε δύο κόμβους Α και Β

Στο ερώτημα αυτό ζητάμε να βρούμε μονοπάτι που να συνδέει δύο κόμβους Α και Β ενός γράφου. Μας ενδιαφέρει, επίσης, και ο υπολογισμός αυτού του μονοπατιού, δηλαδή, η απαρίθμηση ή/και εκτύπωση των κόμβων που το αποτελούν.

Ορισμός 4.4. *Δίνεται ένας γράφος* $G=(V,E)$ *. Έστω Α και Β κόμβοι του G. Ζητείται μονοπάτι* $P_R = \{A,N_{R1}, N_{R2},..., N_{RL},B\}$ *που να συνδέει τους Α και Β.*

Παράδειγμα 4.4. Έστω ο γράφος του Σχήματος 4.4. Η απάντηση στην ερώτηση «Υπάρχει μονοπάτι ανάμεσα στους κόμβους 1 και 18» είναι ΝΑΙ! Επίσης, για την αντίστοιχη ερωτηση για τους κόμβους 2 και 17, η απάντηση είναι ΟΧΙ.

Ορισμός 4.5. *Δίνεται ένας γράφος* $G=(V,E)$ *. Έστω Α και Β κόμβοι του G. Ζητείται να βρεθεί μονοπάτι* $P_R = \{A,N_{R1}, N_{R2},..., N_{RL},B\}$ *που να συνδέει τους Α και Β και να τυπωθούν οι κόμβοι του σε αρχείο ή στην οθόνη.*

Παράδειγμα 4.5. Για το γράφο του Σχήματος 4.4, στην ερώτηση «Υπάρχει μονοπάτι ανάμεσα στους κόμβους 1 και 18 και αν ναι, ποιο είναι αυτό;» η απάντηση είναι το μονοπάτι {1→12→14→15→6→16→18}. Όταν η απάντηση είναι αρνητική, απλώς απαντάμε ΟΧΙ.

Εύρεση μονοπατιού ανάμεσα σε δύο κόμβους Α και Β υπό συνθήκη

Στο ερώτημα αυτό ζητάμε να βρούμε μονοπάτι που να συνδέει δύο κόμβους Α και Β ενός γράφου. Το μονοπάτι αυτό θα πρέπει να ικανοποιεί κάποια συνθήκη. Μας ενδιαφέρει, επίσης, και ο υπολογισμός αυτού του μονοπατιού, δηλαδή, η απαρίθμηση ή/και εκτύπωση των κόμβων που το αποτελούν.

Ορισμός 4.6. *Δίνεται ένας γράφος* $G=(V,E)$ *. Έστω Α και Β κόμβοι του* G_P*. Υπάρχει μονοπάτι* P_R *= {A,*N_{R1}*,* N_{R2}*,..,* N_{RL}*,B} που να συνδέει τους Α και Β; Θα πρέπει να ισχύει:*

$$Συνθήκη(Μήκος(P_R))=TRUE$$

Παράδειγμα 4.6. Για το γράφο του Σχήματος 4.4 θέλουμε το μήκος του μονοπατιού που θα πάρουμε σαν απάντηση να είναι το πολύ 8. Η ερώτηση «Υπάρχει μονοπάτι ανάμεσα στον κόμβο 1 και στον κόμβο 8» στο γράφο του Σχήματος 4.1 δίνει απάντηση ΝΑΙ!. Αντίθετα, η ίδια ερώτηση με μήκος μονοπατιού το πολύ 3 δίνει αρνητική απάντηση.

Ορισμός 4.7. *Δίνεται ένας γράφος* $G=(V,E)$ *. Έστω Α και Β κόμβοι του* G_P*. Υπάρχει μονοπάτι* P_R *= {A,*N_{R1}*,* N_{R2}*,..,* N_{RL}*,B} που να συνδέει τους Α και Β και να τυπωθούν οι κόμβοι του σε αρχείο ή στην οθόνη. Θα πρέπει να ισχύει:* $Συνθήκη(Μήκος(P_R))=TRUE$.

Παράδειγμα 4.7. Για το γράφο του Σχήματος 4.4, θέλουμε το μήκος του μονοπατιού που θα πάρουμε σαν απάντηση να είναι το πολύ 8. Η ερώτηση «Υπάρχει μονοπάτι ανάμεσα στον κόμβο 1 και στον κόμβο 8» στο γράφο του Σχήματος 4.1 δίνει ως απάντηση το μονοπάτι {1→3→4→5→8}. Αντίθετα, η ίδια ερώτηση με μήκος μονοπατιού το πολύ 3 δίνει αρνητική απάντηση.

5

Τεχνικές για την απάντηση ερωτημάτων πάνω σε γράφους

Παρακάτω, θα περιγράψουμε το κύριο κομμάτι της διπλωματικής εργασίας, που είναι στην ουσία η ανάπτυξη μεθόδων και αλγορίθμων για την επίλυση του προβλήματος που ορίσαμε στο προηγούμενο κεφάλαιο. Αρχικά θα δώσουμε μια σύντομη περιγραφή της κατασκευής των γράφων και της αναπαράστασής τους ως σύνολα μονοπατιών. Στη συνέχεια, θα περιγράψουμε τους αλγόριθμους που υλοποιήσαμε.

5.1 Οι γράφοι ως σύνολο μονοπατιών

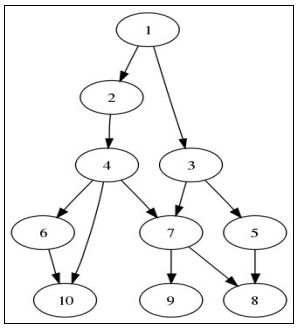
Παρακάτω, περιγράφουμε τη διαδικασία που παράγει αναπαραστάσεις βασιζόμενη σε κατά βάθος διάσχιση γράφου. Δεν μας απασχολεί η κατασκευή των αναπαραστάσεων μονοπατιών με αποδοτικό τρόπο.

Έστω γράφος G(V,E).

```
μονοπάτι p = κενό                    /*current path*/
Όσο υπάρχουν κόμβοι στο γράφο
      A = τυχαίος κόμβος του G
      Όσο ο A δεν είναι φύλλο
            N = τυχαία, ένα από τα παιδιά του N
            Αν το p είναι άδειο τότε πρόσθεσε A
            Πρόσθεσε N
            Αφαίρεσε από το G την ακμή A->N
            Αν ο A δεν έχει άλλα non-visited παδιά τότε αφαίρεσε τον A από
το G
            A = N       /*Για να συνεχιστεί η κατασκευή του μονοπατιού*/
      Αφαίρεσε τον A από το G
      Αν p δεν είναι κενό, τότε τύπωσε μονοπάτι p
      p = κενό
```

Παράδειγμα 5.1.

Έστω ο γράφος του Σχήματος 5.1. Έστω ότι ξεκινάμε από τον κόμβο 1. Τα βήματα για την κατασκευή του αρχείου φαίνονται στον Πίνακα 5.1.



Σχήμα 5.1

τρέχων κόμβος	παιδί που επιλέγουμε	παιδιά	μονοπάτια	visited edges	κόμβοι που έχουν αφαιρεθεί
1	3	2, 3(visited)	{1->3}	(1->3)	-
3	7	5,7(v)	{1->3->7}	{1->3},{3->7}	-
7	8	8(v),9	{1->3->7->8}	{1->3},{3->7},{7->8}	-
8(φύλλο)	-	-	{1->3->7->8}	{1->3},{3->7},{7->8}	8
2	4	4(v)	{1->3->7->8},{2->4}	{1->3},{3->7},{7->8},{2->4}	8
4	6	6(v),7,10	{1->3->7->8},{2->4->6}	{1->3},{3->7},{7->8},{2->4},{4->6}	2,8
6	10	10(v)	{1->3->7->8},{2->4->6->10}	{1->3},{3->7},{7->8},{2->4},{4->6},{6->10}	2,8
10(φύλλο)	-	-	{1->3->7->8},{2->4->6->10}	{1->3},{3->7},{7->8},{2->4},{4->6},{6->10}	2,6,8
5	8	8(v)	{1->3->7->8},{2->4->6->10},{5->8}	{1->3},{3->7},{7->8},{2->4},{4->6},{6->10},{5->8}	2,6,8
8(φύλλο)	-	-	{1->3->7->8},{2->4->6->10},{5->8}	{1->3},{3->7},{7->8},{2->4},{4->6},{6->10},{5->8}	2,5,6,8
4	7	6(v),7(v), 10	{1->3->7->8},{2->4->6->10},{5->8},{4->7}	{1->3},{3->7},{7->8},{2->4},{4->6},{6->10},{5->8},{4->7}	2,5,6,8
7	9	8(v),9(v)	{1->3->7->8},{2->4->6->10},{5->8},{4->7->9}	{1->3},{3->7},{7->8},{2->4},{4->6},{6->10},{5->8},{4->7},{7->9}	2,5,6,8
9(φύλλο)	-	-	{1->3->7->8},{2->4->6->10},{5->8},{4->7->9}	{1->3},{3->7},{7->8},{2->4},{4->6},{6->10},{5->8},{4->7},{7->9}	2,5,6,7,8
4	10	6(v),7(v), 10(v)	{1->3->7->8},{2->4->6->10},{5->8},{4->7->9},{4->10}	{1->3},{3->7},{7->8},{2->4},{4->6},{6->10},{5->8},{4->7},{7->9},{4->10}	2,5,6,7,8,9
10(φύλλο)	-	-	{1->3->7->8},{2->4->6->10},{5->8},{4->7->9},{4->10}	{1->3},{3->7},{7->8},{2->4},{4->6},{6->10},{5->8},{4->7},{7->9},{4->10}	2,4,5,6,7,8,9
1	2	2(v),3(v)	{1->3->7->8},{2->4->6->10},{5->8},{4->7->9},{4->10},{1->2}	{1->3},{3->7},{7->8},{2->4},{4->6},{6->10},{5->8},{4->7},{7->9},{4->10},{1->2}	1,2,4,5,6,7,8,9
3	5	5(v),7(v)	{1->3->7->8},{2->4->6->10},{5->8},{4->7->9},{4->10},{1->2},{3->5}	{1->3},{3->7},{7->8},{2->4},{4->6},{6->10},{5->8},{4->7},{7->9},{4->10},{1->2},{3->5}	1,2,4,5,6,7,8,9
-	-	-	{1->3->7->8},{2->4->6->10},{5->8},{4->7->9},{4->10},{1->2},{3->5}	{1->3},{3->7},{7->8},{2->4},{4->6},{6->10},{5->8},{4->7},{7->9},{4->10},{1->2},{3->5}	1,2,3,4,5,6,7,8,9

Πίνακας 5.1

5.2 Εύρεση παιδιών ενός κόμβου A

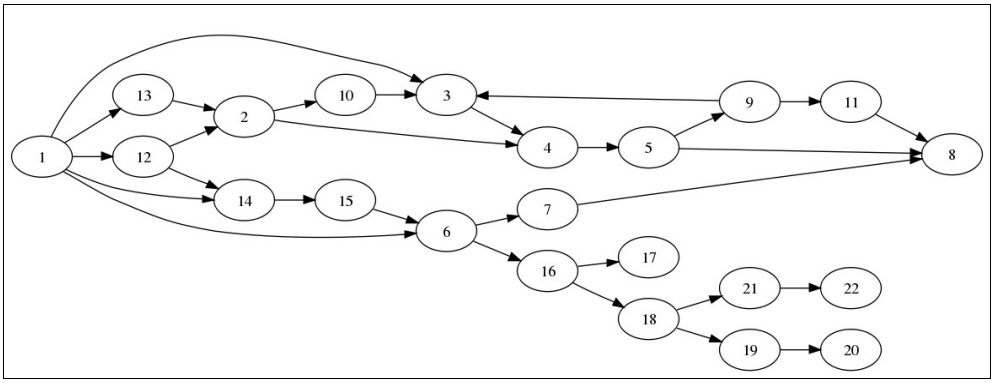
Έχουμε δώσει στην ομάδα αλγορίθμων που απαντούν στο ερώτημα αυτό την ονομασία GETNEXT. Ο αλγόριθμος της Ενότητας 5.2.1 ονομάζεται GETNEXT-TRAD, ενώ ο αλγόριθμος της Ενότητας 5.2.2 ονομάζεται GETNEXT-PINDEX.

5.2.1 Αλγόριθμος που χρησιμοποιεί την αναπαράσταση με λίστες γειτνίασης

Ο αλγόριθμος αυτός αναζητά μέσα στο σύνολο των λιστών γειτνίασης τη λίστα που αντιστοιχεί στον κόμβο A και επιστρέφει ως αποτέλεσμα αυτή τη λίστα.

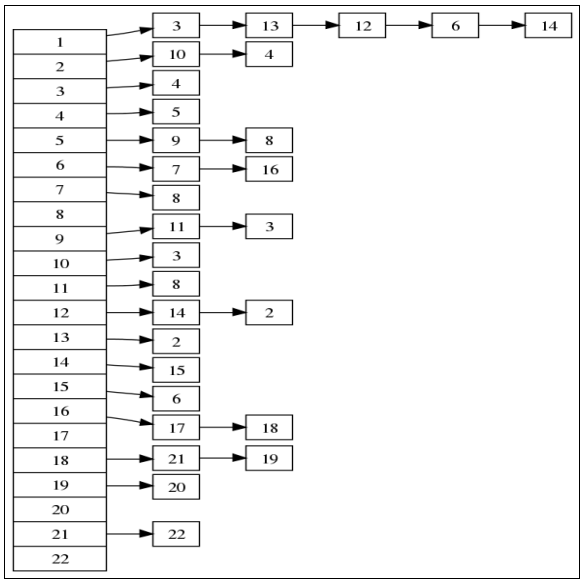
Παράδειγμα 5.2.

Έστω ο γράφος του Σχήματος 5.2.



Σχήμα 5.2

Το σύνολο των λιστών γειτνίασης φαίνεται στο Σχήμα 5.3.



Σχήμα 5.3

Αν θέλουμε, για παράδειγμα, να βρούμε τους απογόνους του κόμβου 1, θα πάρουμε τη λίστα

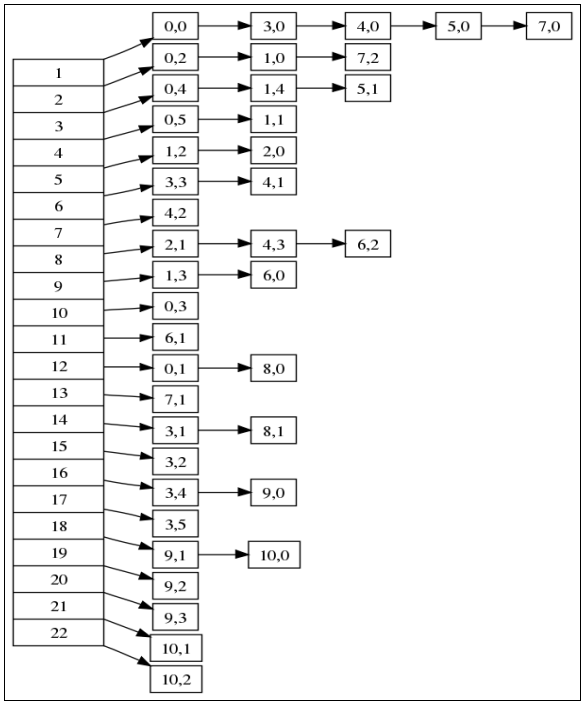
{3→13→12→6→14}

5.2.2 Αλγόριθμος που χρησιμοποιεί την αναπαράσταση με PATHINDEX

Ο αλγόριθμος αυτός δέχεται ως είσοδο το αναγνωριστικό του κόμβου Α και μια αναφορά στη δομή PATHINDEX και επιστρέφει ένα σύνολο από κόμβους που είναι τα παιδιά του Α. Παρακάτω δίνεται ο ψευδοκώδικας:

```
σύνολο_κόμβων getNextPIndex(id_κόμβου_Α, αναφορά_PATHINDEX)
{
      βρες την εγγραφή του PATHINDEX που αναφέρεται στον κόμβο Α
      για κάθε στοιχείο της εγγραφής του PATHINDEX
            αν το στοιχείο δεν είναι στο τέλος του μονοπατιού
            {
                  βρες τον επόμενο κόμβο στο μονοπάτι;
                  πρόσθεσε τον επόμενο στο αποτέλεσμα;
            }
}
```

Παράδειγμα 5.3. Για το γράφο του Σχήματος 5.2 ο PATHINDEX φαίνεται στο Σχήμα 5.4.



Σχήμα 5.4

Η αναπαράσταση με μονοπάτια φαίνεται στο Σχήμα 5.5. Για να βρούμε τα παιδιά του κόμβου 1 ακολουθούμε τα εξής βήματα:

- Βρίσκουμε την εγγραφή με αναγνωριστικό 1. Η εγγραφή είναι η λίστα:

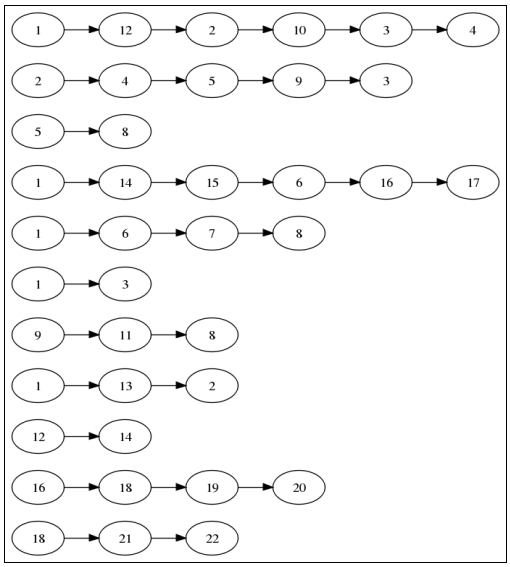
  {(0,0),(3,0),(4,0),(5,0),(7,0)}

- Για κάθε στοιχείο της λίστας πάμε στο μονοπάτι που δίνεται από τον πρώτο νούμερο του

ζεύγους και παίρνουμε το στοιχείο που βρίσκεται στη θέση δεύτερο νούμερο + 1.

- Καταλήγουμε λοιπόν στο αποτέλεσμα

{12,14,6,3,13}



Σχήμα 5.5

5.3 Εύρεση μονοπατιού ανάμεσα σε δύο κόμβους A και B

Έχουμε δώσει στην ομάδα αλγορίθμων που απαντούν στο ερώτημα εύρεσης μονοπατιού το όνομα EXISTPATH, ενώ στους αλγόριθμους που βρίσκουν και ποιο είναι το μονοπάτι δώσαμε το όνομα SHOWPATH. Οι αλγόριθμοι των ενοτήτων που ακολουθούν είναι:

- Ενότητα 5.3.1: BFS-TRAD-EXISTPATH/SHOWPATH
- Ενότητα 5.3.2: DFS-TRAD-EXISTPATH/SHOWPATH
- Ενότητα 5.3.3: BFS-MERGE-EXISTPATH/SHOWPATH
- Ενότητα 5.3.4: DFS-MERGE-EXISTPATH/SHOWPATH
- Ενότητα 5.3.5: BFS-SEP-EXISTPATH/SHOWPATH
- Ενότητα 5.3.6: DFS-SEP-EXISTPATH/SHOWPATH

Η διαφορά ανάμεσα στους αλγόριθμους DFS και BFS είναι η δομή που χρησιμοποιούν ως μέτωπο αναζήτησης, δηλαδή το πώς αποθηκεύουν τους προς εξέταση κόμβους. Οι αλγόριθμοι DFS χρησιμοποιούν στοίβα στην κορυφή της οποίας τοποθετούν και παίρνουν τους κόμβους που πρόκειται να εξεταστούν. Οι αλγόριθμοι BFS χρησιμοποιούν ουρά. Οι αλγόριθμοι αυτοί παίρνουν κόμβους από την αρχή της ουράς και τους τοποθετούν στο τέλος της. Οι αλγόριθμοι με το

αναγνωριστικό MERGE προσθέτουν ολόκληρα κομμάτια μονοπατιών στο μέτωπο αναζήτησης, ενώ οι αλγόριθμοι με το αναγνωριστικό SEP προσθέτουν τους κόμβους προς εξέταση χωριστά.

5.3.1 Αλγόριθμος που χρησιμοποιεί την αναπαράσταση με λίστες γειτνίασης(bfs)

Ο αλγόριθμος αυτός αναζητά μονοπάτι ανάμεσα στους κόμβους Α και Β ενός γράφου που δίνεται ως σύνολο λιστών γειτνίασης. Η ιδέα στην οποία στηρίζεται είναι η κατά πλάτος εξερεύνηση των κόμβων του γράφου. Προκειμένου να μην επισκεφθούμε τον ίδιο κόμβο πολλές φορές, σημειώνουμε ποιούς κόμβους έχουμε επισκεφθεί, δίνοντάς τους χρώματα. Τα χρώματα αυτά είναι:

- ΑΣΠΡΟ για τους κόμβους που δεν έχουμε επισκεφθεί ακόμα
- ΓΚΡΙ για τους κόμβους που έχουμε επισκεφθεί
- ΜΑΥΡΟ για τους κόμβους, των οποίων έχουμε επισκεφθεί όλα τα παιδιά.

Ψευδοκώδικας:

```
ελέγχουμε αν οι Α και Β υπάρχουν στο γράφο
χρωματίζουμε όλους τους κόμβους ΑΣΠΡΟΥΣ
τοποθετούμε τον Α στην ουρά και τον χρωματίζουμε ΓΚΡΙ
όσο η ουρά δεν είναι άδεια
{
      t←πρώτο στοιχείο της ουράς
      βρίσκουμε τη λίστα γειτνίασης που αντιστοιχεί στον κόμβο t
      Για κάθε στοιχείο e της λίστας
      {
            αν είναι ΑΣΠΡΟ
            {
                  το χρωματίζουμε ΓΚΡΙ;
                  σημειώνουμε ότι ο πατέρας του e είναι ο t;
                  εισάγουμε το e στην ουρά;
                  αν (e == B)
                  {
                        τέλος;//αυτό σημαίνει ΝΑΙ! για το existPath
                              //ή εκτύπωση μονοπατιού για το showPath
                  }
            }
      }
      βγάζουμε το t από την ουρά;
      χρωματίζουμε το t ΜΑΥΡΟ;
}
```

Παράδειγμα 5.4. Το αποτέλεσμα που θα προκύψει αν τρέξουμε τον παραπάνω αλγόριθμο φαίνεται στον Πίνακα 5.1. Θέλουμε να βρούμε αν υπάρχει μονοπάτι ανάμεσα στους κόμβους 1 και 20 του γράφου του Σχήματος 5.2. Στον Πίνακα 5.2 σημειώνουμε με κόκκινο χρώμα τους κόμβους που εισάγονται στην ουρά. Αν θέλουμε να βρούμε και ΠΟΙΟ είναι το μονοπάτι που συνδέει τους κόμβους 1 και 20, θα πρέπει να προσθέσουμε τον εξής επαναληπτικό βρόχο:

```
τρέχων κόμβος = κόμβος Β;
όσο τρέχων κόμβος != κόμβος Α
{
      τρέχων κόμβος = κόμβος πατέρας του τρέχοντος κόμβου;
      τύπωσε ή αποθήκευσε τον τρέχοντα κόμβο;
}
```

Την ίδια διαδικασία ακολουθούμε και στις υπόλοιπες μεθόδους. Τρέχον στοιχείο και στοιχείο πατέρας, μπορεί να είναι, είτε ένας κόμβος, είτε ένα κομμάτι μονοπατιού, ανάλογα με το τί προσθέτουμε στο μέτωπο αναζήτησης.

Βήμα	κόμβος u[χρώμα]	ουρά[πατέρας]
1	-	1[-]
2	1[A]	12[1],14[1],6[1],3[1],13[1]
3	12[A]	14,6,3,13,2[12],14[12]
	...	...
N-1	5[A]	19,21,...,9[5]
N	19[A]	21,...,9,20[19] ←Βρέθηκε!

Πίνακας 5.2

5.3.2 Αλγόριθμος που χρησιμοποιεί την αναπαράσταση με λίστες γειτνίασης(dfs)

Ο αλγόριθμος αυτός αναζητά μονοπάτι ανάμεσα στους κόμβους Α και Β ενός γράφου που δίνεται ως σύνολο λιστών γειτνίασης. Η ιδέα στην οποία στηρίζεται είναι η κατά βάθος εξερεύνηση των κόμβων του γράφου. Οι κόμβοι σημειώνονται με τα εξής «χρώματα»:

- ΑΣΠΡΟ για τους κόμβους που δεν έχουμε επισκεφθεί ακόμα
- ΓΚΡΙ για τους κόμβους που έχουμε επισκεφθεί
- ΜΑΥΡΟ για τους κόμβους, των οποίων έχουμε επισκεφθεί όλα τα παιδιά.

Ψευδοκώδικας:

```
ελέγχουμε αν οι Α και Β υπάρχουν στο γράφο
χρωματίζουμε όλους τους κόμβους ΑΣΠΡΟΥΣ
τοποθετούμε τον Α στη στοίβα
όσο η στοίβα δεν είναι άδεια
{
      t←κορυφή της στοίβας;
      βγάζουμε το t από τη στοίβα και τον χρωματίζουμε ΓΚΡΙ;
      βρίσκουμε τη λίστα γειτνίασης που αντιστοιχεί στον κόμβο t;
      Για κάθε στοιχείο e της λίστας
      {
            αν είναι ΑΣΠΡΟ
            {
                  σημειώνουμε ότι ο πατέρας του e είναι ο t;
                  εισάγουμε το e στη στοίβα;
```

```
                    αν (e == B)
                    {
                          τέλος;//αυτό σημαίνει NAI! για το existPath
                                //ή εκτύπωση μονοπατιού για το showPath
                    }
              }
        }
        βγάζουμε το t από τη στοίβα;
        Αν έχουμε επισκεφθεί όλα τα παιδιά του t
              χρωματίζουμε το t ΜΑΥΡΟ;
}
```

Παράδειγμα 5.5. Το αποτέλεσμα που θα προκύψει αν τρέξουμε τον παραπάνω αλγόριθμο φαίνεται στον Πίνακα 5.2. Θυμίζουμε ότι ψάχνουμε μονοπάτι ανάμεσα στους κόμβους 1 και 20 του Σχήματος 5.2.

Βήμα	κόμβος u[χρώμα]	στοίβα[πατέρας]
1	-	1[-]
2	1[A]	12[1],14[1],6[1],3[1],13[1]
3	13[A]	12,14,6,3,2[13]
4	2[A]	12,14,6,3,10[13],4[13]
	...	...
N-3	18[A]	...,19[18],21[18]
N-2	21[A]	...,19,22[21]
N-1	22[A]	...,19
N	19[A]	...,20[19] ←Βρέθηκε!

Πίνακας 5.3

Αν θέλουμε να βρούμε και ΠΟΙΟ είναι το μονοπάτι που συνδέει τους κόμβους 1 και 20, θα πρέπει να προσθέσουμε τον εξής επαναληπτικό βρόχο:

```
τρέχων κόμβος = κόμβος B;
όσο τρέχων κόμβος != κόμβος A
{
      τρέχων κόμβος = κόμβος πατέρας του τρέχοντος κόμβου;
      τύπωσε ή αποθήκευσε τον τρέχοντα κόμβο;
}
```

5.3.3 Αλγόριθμος που τοποθετεί κομμάτια μονοπατιών στο μέτωπο αναζήτησης(bfs-l)

Ο αλγόριθμος αυτός αναζητά μονοπάτι ανάμεσα στους κόμβους A και B ενός γράφου που δίνεται ως σύνολο μονοπατιών. Η ιδέα στην οποία στηρίζεται είναι η κατά πλάτος εξερεύνηση των κόμβων του γράφου. Οι κόμβοι σημειώνονται με τα εξής «χρώματα»:

- ΑΣΠΡΟ για τους κόμβους που δεν έχουμε επισκεφθεί ακόμα

- ΓΚΡΙ για τους κόμβους που έχουμε επισκεφθεί
- ΜΑΥΡΟ για τους κόμβους, των οποίων έχουμε επισκεφθεί όλα τα παιδιά.

Στον αλγόριθμο που προτείνουμε βάζουμε στην ουρά ολόκληρα τα κομμάτια των μονοπατιών στα οποία ανήκει ο τρέχων κόμβος και όχι κάθε κόμβο χωριστά.

Ψευδοκώδικας:

```
ελέγχουμε αν οι A και B υπάρχουν στο γράφο;
προσθέτουμε τον A στην ουρά;
όσο η ουρά δεν είναι άδεια
{
      u←πρώτο στοιχείο της ουράς;
      αν (u == B)
            τέλος;
      αν έχουμε επισκευθεί τον u, τον βγάζουμε από την ουρά και
      συνεχίζουμε;
      αλλιώς, σημειώνουμε ότι τον έχουμε επισκευθεί;
      βρίσκουμε την εγγραφή του PATHINDEX που αντιστοιχεί στον u;
      //θα αναλύσουμε το existCommonPath παρακάτω...
      found = existCommonPath(εγγραφηPATHINDEX_u, εγγραφηPATHINDEX_B);
      Αν (found == TRUE)
            τέλος;
      Για κάθε στοιχείο e της  εγγραφηPATHINDEX_u
      {
            αν έχουμε βάλει το κομμάτι του μονοπατιού που δίνεται από το e
            {
                  συνεχίζουμε στο επόμενο στοιχείο e;
            }
            αλλιώς
            {
                  προσθέτουμε το κομμάτι e στο τέλος της ουράς;
                  σημειώνουμε ότι πατέρας του e είναι το κομμάτι στο οποίο
                  είμαστε τώρα;
            }
      }
      αφαιρούμε το u από την ουρά;
}
```

Η συνάρτηση existCommonPath παρουσιάζεται με τον παρακάτω ψευδοκώδικα. Η συνάρτηση αυτή αναζητά το πρώτο κοινό στοιχείο δύο λιστών. Το αποτέλεσμά της είναι TRUE μόνο όταν η θέση του u είναι πριν τον κόμβο B στο μονοπάτι. Η existCommonPath πραγματοποιεί merge-join μεταξύ δύο λιστών.

```
έχουμε δύο λίστες με μονοπάτια, τη λίστα του κόμβου B και τη λίστα του
κόμβου u;
ορίζουμε δύο δείκτες δείκτης_u και δείκτης_B, που δείχνουν ο καθένας στην
αρχή της αντίστοιχης λίστας;
όσο ( δείκτης_u != τέλος λίστας) και (δείκτης_B != τέλος λίστας)
{
      αν(στοιχείο_που_δείχνει_δείκτης_u < στοιχείο_που_δείχνει_δείκτης_B)
      {
            δείκτης_u++;
```

```
            συνεχίζουμε;
      }
      αν(στοιχείο_που_δείχνει_δείκτης_u > στοιχείο_που_δείχνει_δείκτης_B)
      {
            δείκτης_B++;
            συνεχίζουμε;
      }
      αν((στοιχείο_που_δείχνει_δείκτης_u ==
            στοιχείο_που_δείχνει_δείκτης_B)
       &&(η θέση του u μετά το B στο μονοπάτι))
      {
            δείκτης_u++;
            συνεχίζουμε;
      }
      αν((στοιχείο_που_δείχνει_δείκτης_u                ==
            στοιχείο_που_δείχνει_δείκτης_B)
       &&(η θέση του u πριν το B στο μονοπάτι))
      {
            βρήκαμε μονοπάτι;
            found_ = TRUE;
      }
}
```

Παράδειγμα 5.6.

Το αποτέλεσμα που θα προκύψει αν τρέξουμε τον αλγόριθμο εύρεσης μονοπατιού φαίνεται στον Πίνακα 5.3. Θυμίζουμε ότι ψάχνουμε μονοπάτι ανάμεσα στους κόμβους 1 και 20 του Σχήματος 5.2.

Βήμα	κόμβος u [χρώμα]	ουρά[κομμάτι πατέρας]	Παρατήρηση
1	-	1[-]	
2	1[A]	(0,1,5)[(-1,0,0)]→(3,1,5)[(-1,0,0)]→(4,1,3)[(-1,0,0)]→(5,1,1)[(-1,0,0)]→(7,1,2)[(-1,0,0)] 12,2,10,3,4,14,15,6,16,17,6,7,8,3,13,2	
3	12[A]	(0,1,5)[(-1,0,0)]→(3,1,5)[(-1,0,0)]→(4,1,3)[(-1,0,0)]→(5,1,1)[(-1,0,0)]→(7,1,2)[(-1,0,0)]→(8,1,1)[(0,1,5)] 2,10,3,4,14,15,6,16,17,6,7,8,3,13,2,14	Το κομμάτι (2,5) του μονοπατιού 0 έχει ήδη προστεθεί!
4	2[A]	…	…
…	…	…	…
N	16[A]	…	Ο κόμβος 20 ανήκει στο κομμάτι (1,3) του μονοπατιού 9 ←Βρέθηκε!

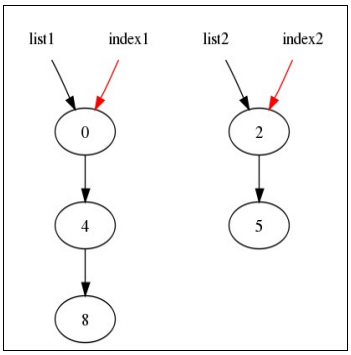
Πίνακας 5.4

Αν θέλουμε να βρούμε και ΠΟΙΟ είναι το μονοπάτι που συνδέει τους κόμβους 1 και 20, θα πρέπει να προσθέσουμε τον εξής επαναληπτικό βρόχο:

```
τρέχον κομμάτι = κομμάτι στο οποίο ανήκει ο B;
όσο τρέχον κομμάτι != κενό κομμάτι
{
      τρέχον κομμάτι = κομμάτι πατέρας του τρέχοντος κόμβου;
      τύπωσε ή αποθήκευσε το τρέχον κομμάτι;
}
```

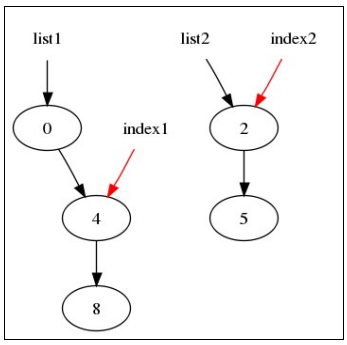
Παράδειγμα 5.7. Το παραδειγμα αυτό αφορά στη συνάρτηση existCommonPath.

Έστω οι λίστες τους Σχήματος 5.6. Αρχικά, οι δύο δείκτες δείχνουν στην αρχή των δύο λιστών.



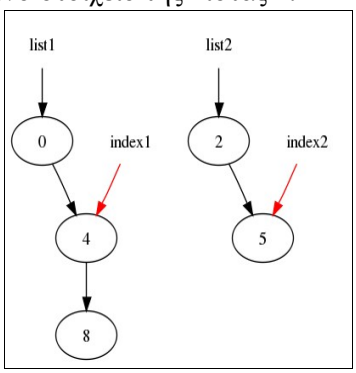
Σχήμα 5.6

Επειδή 0 < 2, προχωράμε στο επόμενο στοιχείο της λίστας 1.



Σχήμα 5.7

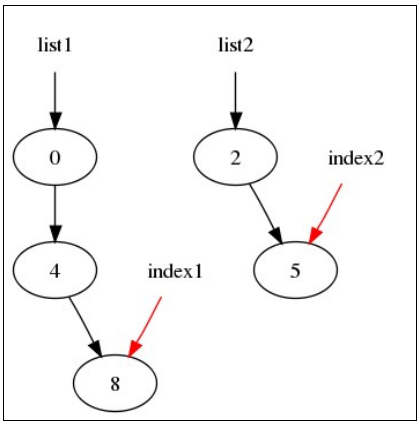
Επειδή 2 < 4, προχωράμε στο επόμενο στοιχείο της λίστας 2.



Σχήμα 5.8

Επειδή 4 < 5, προχωράμε στο επόμενο στοιχείο της λίστας 1.(Σχήμα 5.9)

Εφόσον και οι δύο δείκτες δείχνουν στο τέλος των λιστών, έχουμε τελειώσει. Αν είχαμε βρει κάποιο κοινό στοιχείο, θα είχαμε απάντηση TRUE.



Σχήμα 5.9

5.3.4 Αλγόριθμος που τοποθετεί κομμάτια μονοπατιών στο μέτωπο αναζήτησης(dfs-l)

Ο αλγόριθμος αυτός αναζητά μονοπάτι ανάμεσα στους κόμβους Α και Β ενός γράφου που δίνεται ως σύνολο μονοπατιών. Η ιδέα στην οποία στηρίζεται είναι η κατά βάθος εξερεύνηση των κόμβων του γράφου. Οι κόμβοι σημειώνονται με τα εξής «χρώματα»:

- ΑΣΠΡΟ για τους κόμβους που δεν έχουμε επισκευθεί ακόμα
- ΓΚΡΙ για τους κόμβους που έχουμε επισκευθεί
- ΜΑΥΡΟ για τους κόμβους, των οποίων έχουμε επισκευθεί όλα τα παιδιά.

Στον αλγόριθμο που προτείνουμε βάζουμε στην ουρά ολόκληρα τα κομμάτια των μονοπατιών στα οποία ανήκει ο τρέχων κόμβος και όχι κάθε κόμβο χωριστά.

Ο ψευδοκώδικας είναι:

```
ελέγχουμε αν οι Α και Β υπάρχουν στο γράφο;
προσθέτουμε τον Α στη στοίβα;
όσο η στοίβα δεν είναι άδεια
{
      u←στοιχείο στην κορυφή της στοίβας;
      αφαιρούμε το u από τη στοίβα;
      αν (u == B)
            τέλος;
      αν έχουμε επισκεφθεί τον u, τον βγάζουμε από τη στοίβα και
      συνεχίζουμε;
      αλλιώς, σημειώνουμε ότι τον έχουμε επισκεφθεί;
      βρίσκουμε την εγγραφή του PATHINDEX που αντιστοιχεί στον u;
      //η existCommonPath έχει περιγραφεί στην Ενότητα 5.3.3.
      found = existCommonPath(εγγραφηPATHINDEX_u, εγγραφή_PATHINDEX_B);
      Αν (found == TRUE)
            τέλος;
      Για κάθε στοιχείο e της  εγγραφηPATHINDEX_u
      {
            αν έχουμε βάλει το κομμάτι του μονοπατιού που δίνεται από το e
            {
                  συνεχίζουμε στο επόμενο στοιχείο e;
            }
            αλλιώς
```

```
                προσθέτουμε το κομμάτι e στην κορυφή της στοίβας;
        }
}
```

Παράδειγμα 5.8. Θυμίζουμε ότι ψάχνουμε μονοπάτι ανάμεσα στους κόμβους 1 και 20 του Σχήματος 5.2.

Βήμα	κόμβος u [χρώμα]	στοίβα[πατέρας]	Παρατήρηση
1	-	1[-]	
2	1[A]	(0,1,5)[(-1,0,0)]→(3,1,5)[(-1,0,0)]→(4,1,3)[(-1,0,0)]→(5,1,1)[(-1,0,0)]→(7,1,2)[(-1,0,0)]	
		12,2,10,3,4,14,15,6,16,17,6,7,8,3,13,2	
3	2[A]	(0,1,5)[(-1,0,0)]→(3,1,5)[(-1,0,0)]→(4,1,3)[(-1,0,0)]→(5,1,1)[(-1,0,0)]→(7,1,2)[(-1,0,0)]→(1,1,4)[(7,1,2)]	Το κομμάτι (2,5) του μονοπατιού 0 έχει ήδη προστεθεί!
		12,2,10,3,4,14,15,6,16,17,6,7,8,3,13,4,5,9,3	Το κομμάτι (2,2) του μονοπατιού 7 έχει ήδη προστεθεί!
4	3[A]	...	...
...	...	...	...
N	16[A]	...	Ο κόμβος 20 ανήκει στο κομμάτι (1,3) του μονοπατιού 9 ←Βρέθηκε!

Πίνακας 5.5

Αν θέλουμε να βρούμε και ΠΟΙΟ είναι το μονοπάτι που συνδέει τους κόμβους 1 και 20, θα πρέπει να προσθέσουμε τον εξής επαναληπτικό βρόχο:

```
τρέχον κομμάτι = κομμάτι στο οποίο ανήκει ο B;
όσο τρέχον κομμάτι != κενό κομμάτι
{
      τρέχον κομμάτι = κομμάτι πατέρας του τρέχοντος κόμβου;
      τύπωσε ή αποθήκευσε το τρέχον κομμάτι;
}
```

5.3.5 Αλγόριθμος που τοποθετεί χωριστά τους κόμβους στο μέτωπο αναζήτησης(bfs-l)

Ο αλγόριθμος αυτός αναζητά μονοπάτι ανάμεσα στους κόμβους Α και Β ενός γράφου που δίνεται ως σύνολο μονοπατιών. Η ιδέα στην οποία στηρίζεται είναι η κατά πλάτος εξερεύνηση των κόμβων του γράφου. Οι κόμβοι σημειώνονται με τα εξής «χρώματα»:

- ΑΣΠΡΟ για τους κόμβους που δεν έχουμε επισκεφθεί ακόμα
- ΓΚΡΙ για τους κόμβους που έχουμε επισκεφθεί
- ΜΑΥΡΟ για τους κόμβους, των οποίων έχουμε επισκεφθεί όλα τα παιδιά.

Στον αλγόριθμο που προτείνουμε βάζουμε στην ουρά ξεχωριστά κάθε έναν από τους κόμβους που ανήκουν στα κομμάτια των μονοπατιών στα οποία ανήκει ο τρέχων κόμβος.

Ψευδοκώδικας:

```
ελέγχουμε αν οι A και B υπάρχουν στο γράφο;
προσθέτουμε τον A στην ουρά;
όσο η ουρά δεν είναι άδεια
{
      u←πρώτο στοιχείο της ουράς;
      αν (u == B)
            τέλος;//βρήκαμε τον κόμβο B
```

```
        αν έχουμε επισκεφθεί τον u, τον βγάζουμε από την ουρά και
        συνεχίζουμε;
        αλλιώς, σημειώνουμε ότι τον έχουμε επισκεφθεί;
        βρίσκουμε την εγγραφή του PATHINDEX που αντιστοιχεί στον u;
        Για κάθε στοιχείο e της  εγγραφή_PATHINDEX_u
        {
                αν έχουμε βάλει το κομμάτι του μονοπατιού που δίνεται από το e
                {
                        συνεχίζουμε στο επόμενο στοιχείο e;
                }
                αλλιώς
                {
                        για κάθε στοιχείο n του μονοπατιού
                        {
                                αν το n δεν έχει μπει στην ουρά
                                        αν (n == B)
                                        {
                                                τέλος;
                                        }
                                        αλλιώς
                                        {
                                                προσθέτουμε το n στο τέλος της ουράς;
                                        }
                        }
                }
        }
        αφαιρούμε το u από την ουρά;
}
```

Παράδειγμα 5.9. Θυμίζουμε ότι ψάχνουμε μονοπάτι ανάμεσα στους κόμβους 1 και 20 του Σχήματος 5.2. Αν θέλουμε να βρούμε και ΠΟΙΟ είναι το μονοπάτι που συνδέει τους κόμβους 1 και 20, θα πρέπει να προσθέσουμε τον εξής επαναληπτικό βρόχο:

```
τρέχων κόμβος = κόμβος B;
όσο τρέχων κόμβος != κόμβος A
{
        τρέχων κόμβος = κόμβος πατέρας του τρέχοντος κόμβου;
        τύπωσε ή αποθήκευσε τον τρέχοντα κόμβο;
}
```

Βήμα	κόμβος u [χρώμα]	ουρά[πατέρας]	Παρατήρηση
1	-	1[-]	
2	1[A]	12[1],2[12],10[2],3[10],4[3],14[1],15[14],6[15],16[6],17[16],6[1],7[6],8[7],3[1],13[3],2[13]	
3	12[A]	2,10,3,4,14,15,6,16,17,6,7,8,3,13,2,14[12]	Το κομμάτι (2,5) του μονοπατιού 0 έχει ήδη προστεθεί!
4	2[A]	...	...
...	...	...	...
N	16[A]	...,18[16],19[18],20[19]	Ο κόμβος 20 ανήκει στο κομμάτι (1,3) του μονοπατιού 9 ←Βρέθηκε!

Πίνακας 5.6

5.3.6 Αλγόριθμος που τοποθετεί χωριστά τους κόμβους στο μέτωπο αναζήτησης(dfs-l)

Ο αλγόριθμος αυτός αναζητά μονοπάτι ανάμεσα στους κόμβους Α και Β ενός γράφου που δίνεται ως σύνολο μονοπατιών. Η ιδέα στην οποία στηρίζεται είναι η κατά βάθος εξερεύνηση των κόμβων του γράφου. Οι κόμβοι σημειώνονται με τα εξής «χρώματα»:

- ΑΣΠΡΟ για τους κόμβους που δεν έχουμε επισκεφθεί ακόμα
- ΓΚΡΙ για τους κόμβους που έχουμε επισκεφθεί
- ΜΑΥΡΟ για τους κόμβους, των οποίων έχουμε επισκεφθεί όλα τα παιδιά.

Στον αλγόριθμο που προτείνουμε βάζουμε στη στοίβα ξεχωριστά κάθε έναν από τους κόμβους που ανήκουν στα κομμάτια των μονοπατιών στα οποία ανήκει ο τρέχων κόμβος.

Ψευδοκώδικας:

```
ελέγχουμε αν οι Α και Β υπάρχουν στο γράφο;
προσθέτουμε τον Α στη στοίβα;
όσο η στοίβα δεν είναι άδεια
{
      u←στοιχείο στην κορυφή της στοίβας;
      αν (u == Β)
            τέλος;//βρήκαμε τον κόμβο Β
      αν έχουμε επισκεφθεί τον u, τον βγάζουμε από τη στοίβα και
      συνεχίζουμε;
      αλλιώς, σημειώνουμε ότι τον έχουμε επισκεφθεί;
      βρίσκουμε την εγγραφή του PATHINDEX που αντιστοιχεί στον u;
      αφαιρούμε το u από τη στοίβα;
      Για κάθε στοιχείο e της  εγγραφή_PATHINDEX_u
      {
            αν έχουμε βάλει το κομμάτι του μονοπατιού που δίνεται από το e
            {
                  συνεχίζουμε στο επόμενο στοιχείο e;
            }
            αλλιώς
            {
                  για κάθε στοιχείο n του μονοπατιού
                  {
                        αν το n δεν έχει μπει στη στοίβα
                              αν (n == Β)
                              {
                                    τέλος;
                              }
                              αλλιώς
                              {
                                    προσθέτουμε το n στην κορυφή της
                                    στοίβας;
                              }
                  }
            }
      }
}
```

Παράδειγμα 5.10.

Αν τρέξουμε τον παραπάνω αλγόριθμο, θα έχουμε το αποτέλεσμα που φαίνεται στον Πίνακα 5.6. Θυμίζουμε ότι ψάχνουμε μονοπάτι ανάμεσα στους κόμβους 1 και 20 του Σχήματος 5.2.

Βήμα	κόμβος u [χρώμα]	στοίβα[πατέρας]	Παρατήρηση
1	-	1[-]	
2	1[A]	12[1],2[12],10[2],3[10],4[3],14[1],15[14],6[15],16[6],17[16],6[1],7[6],8[7],3[1],13[1],2[13]	
3	2[A]	12,2,10,3,4,14,15,6,16,17,6,7,8,3,13,4[2],5[4],9[5],3[9]	Το κομμάτι (2,5) του μονοπατιού 0 έχει ήδη προστεθεί! Το κομμάτι (2,2) του μονοπατιού 7 έχει ήδη προστεθεί!
4	3[A]	...	...
...	...	...	...
N	16[A]	...,18[16],19[18],20[19]	Ο κόμβος 20 ανήκει στο κομμάτι (1,3) του μονοπατιού 9 ←Βρέθηκε!

Πίνακας 5.7

Αν θέλουμε να βρούμε και ΠΟΙΟ είναι το μονοπάτι που συνδέει τους κόμβους 1 και 20, θα πρέπει να προσθέσουμε τον εξής επαναληπτικό βρόχο:

```
τρέχων κόμβος = κόμβος B;
όσο τρέχων κόμβος != κόμβος A
{
      τρέχων κόμβος = κόμβος πατέρας του τρέχοντος κόμβου;
      τύπωσε ή αποθήκευσε τον τρέχοντα κόμβο;
}
```

5.4 Εύρεση μονοπατιού ανάμεσα σε δύο κόμβους A και B υπό συνθήκη

Έχουμε δώσει στην ομάδα αλγορίθμων που απαντούν στο ερώτημα εύρεσης μονοπατιού το όνομα EXISTPATHCONSTRAINT, ενώ στους αλγόριθμους που βρίσκουν και ποιο είναι το μονοπάτι δώσαμε το όνομα SHOWPATHCONSTRAINT. Οι αλγόριθμοι των ενοτήτων που ακολουθούν είναι:

- Ενότητα 5.4.1: TRAD-EXISTPATHCONSTRAINT/SHOWPATHCONSTRAINT
- Ενότητα 5.4.2: MERGE-EXISTPATHCONSTRAINT/SHOWPATHCONSTRAINT
- Ενότητα 5.4.3: SEP-EXISTPATHCONSTRAINT/SHOWPATHCONSTRAINT

Το μέτωπο αναζήτησης στις παρακάτω μεθόδους, δηλαδή η δομή όπου αποθηκεύονται οι προς εξέταση κόμβοι, είναι μια στοίβα. Οι κόμβοι τοποθετούνται και αφαιρούνται από την κορυφή της στοίβας. Ο αλγόριθμος με το αναγνωριστικό MERGE προσθέτει ολόκληρα κομμάτια μονοπατιών στο μέτωπο αναζήτησης, ενώ ο αλγόριθμος με το αναγνωριστικό SEP προσθέτει τους κόμβους προς εξέταση χωριστά.

5.4.1 Αλγόριθμος που χρησιμοποιεί την αναπαράσταση με λίστες γειτνίασης

Ο αλγόριθμος αυτός εκτελεί κατά βάθος αναζήτηση στους κόμβους του γράφου, ο οποίος δίνεται ως λίστες γειτνίασης. Κρατάμε πληροφορία σε δύο δομές, τη στοίβα, όπου βάζουμε τους κόμβους τους οποίους συναντήσαμε και το μονοπάτι, όπου σχηματίζουμε το αποτέλεσμα. Το μονοπάτι είναι και αυτό μια στοίβα.

Ψευδοκώδικας:

```
ελέγχουμε αν οι Α και Β υπάρχουν στο γράφο;
προσθέτουμε τον κόμβο Α στο μονοπάτι;
όσο το μονοπάτι δεν είναι άδειο
{
      u→κορυφή του μονοπατιού;
      αν δεν έχω βάλει τα παιδιά του u στη στοίβα, προσθέτω όσα δεν είναι
      ήδη στο μονοπάτι;
      αν το e στην κορυφή της στοίβας είναι παιδί του u
      {
            αν ικανοποιείται ο περιορισμός
            {
                  αν (e == B)
                  {
                        τέλος;
                  }
                  αλλιώς, προσθέτω το e στο μονοπάτι;
            }
            βγάζουμε το e από τη στοίβα;
      }
      αλλιώς, βγάζουμε το u από το μονοπάτι;
}
```

Παράδειγμα 5.11. Έστω ότι θέλουμε το μονοπάτι που θα βρούμε να έχει το πολύ 6 κόμβους. Το αποτέλεσμα από μία εκτέλεση θα είναι όπως στον Πίνακα 5.8. Θυμίζουμε ότι ψάχνουμε μονοπάτι ανάμεσα στους κόμβους 1 και 20 του Σχήματος 5.2.

Βήμα	κορυφή μονοπατιού	μονοπάτι	κορυφή στοίβας	στοίβα
1	-	1	-	-
2	1	1,13	13	12,14,6,3,13(βγαίνει)
3	13	1,13,2	2	12,14,6,3,2(βγαίνει)
...	...	...	...	...
N	19	1,...,19	20	...,20←Βρέθηκε!

Πίνακας 5.8

Αν θέλουμε να βρούμε και ΠΟΙΟ είναι το μονοπάτι που συνδέει τους κόμβους 1 και 20, θα πρέπει να προσθέσουμε τον εξής επαναληπτικό βρόχο:

```
τρέχων κόμβος = κόμβος Β;
όσο τρέχων κόμβος != κόμβος Α
{
      τρέχων κόμβος = κόμβος πατέρας του τρέχοντος κόμβου;
      τύπωσε ή αποθήκευσε τον τρέχοντα κόμβο;
}
```

5.4.2 Αλγόριθμος που τοποθετεί κομμάτια μονοπατιών στο μέτωπο αναζήτησης

Ο αλγόριθμος αυτός εκτελεί κατά βάθος αναζήτηση στους κόμβους του γράφου, ο οποίος δίνεται ως σύνολο μονοπατιών. Κρατάμε πληροφορία σε δύο δομές, τη στοίβα, όπου βάζουμε τους κόμβους

τους οποίους συναντήσαμε και το μονοπάτι, όπου σχηματίζουμε το αποτέλεσμα. Το μονοπάτι είναι και αυτό μια στοίβα. Η στοίβα περιέχει αναφορές σε κομμάτια μονοπατιών.

Ψευδοκώδικας:

```
ελέγχουμε αν οι Α και Β υπάρχουν στο γράφο;
προσθέτουμε τον κόμβο Α στο μονοπάτι;
όσο το μονοπάτι δεν είναι άδειο
{
      u→κορυφή του μονοπατιού;
      αν δεν έχω βάλει τα παιδιά του u στη στοίβα, προσθέτω όσα δεν είναι
      ήδη στο μονοπάτι;
      αν το e στην κορυφή της στοίβας προέρχεται από το u
      {
            αν ικανοποιείται ο περιορισμός
            {
                  αν (e == B)
                  {
                        τέλος;
                  }
                  αλλιώς, προσθέτω το κομμάτι e του μονοπατιού στη στοίβα;
                  //ελέγχοντας αν κάθε φορά αν ικανοποιείται ο περιορισμός
            }
            βγάζουμε το e από τη στοίβα;
      }
      αλλιώς, βγάζουμε το u από το μονοπάτι;
}
```

Παράδειγμα 5.12. Θυμίζουμε ότι ψάχνουμε μονοπάτι ανάμεσα στους κόμβους 1 και 20 του Σχήματος 5.2.

Βήμα	κορυφή μονοπατιού	μονοπάτι	κορυφή στοίβας	στοίβα[θέση έναρξης]
1	-	1	-	-
2	1	1,13,2		0[1]→3[1]→4[1]→5[1]→7[1] 12,2,10,3,4,14,15,6,16,17,6,7,8,3,13,2
3	2	1,13,2,4,5,9,3		0[1]→3[1]→4[1]→5[1]→0[3]→1[1] 12,2,10,3,4,14,15,6,16,17,6,7,8,3,13,2,10,3,4,4,5,9,3
…	…	…	…	…
N	16	1,…,16,18,19,20←ΒΡΕΘΗΚΕ		…→4[5]→9[1] …,6,7,8,18,19,20

Πίνακας 5.9

Αν θέλουμε να βρούμε και ΠΟΙΟ είναι το μονοπάτι που συνδέει τους κόμβους 1 και 20, θα πρέπει να προσθέσουμε τον εξής επαναληπτικό βρόχο:

```
τρέχον κομμάτι = κομμάτι στο οποίο ανήκει ο Β;
όσο τρέχον κομμάτι != κενό κομμάτι
{
      τρέχον κομμάτι = κομμάτι πατέρας του τρέχοντος κόμβου;
      τύπωσε ή αποθήκευσε το τρέχον κομμάτι;
}
```

5.4.3 Αλγόριθμος που τοποθετεί χωριστά τους κόμβους στο μέτωπο αναζήτησης

Ο αλγόριθμος αυτός εκτελεί κατά βάθος αναζήτηση στους κόμβους του γράφου, ο οποίος δίνεται ως σύνολο μονοπατιών. Κρατάμε πληροφορία σε δύο δομές, μια στοίβα, όπου βάζουμε τους κόμβους

τους οποίους συναντήσαμε και το μονοπάτι, όπου σχηματίζουμε το αποτέλεσμα. Το μονοπάτι είναι και αυτό μια στοίβα. Η στοίβα περιέχει αναφορές σε κομμάτια μονοπατιών.

Ψευδοκώδικας:

```
ελέγχουμε αν οι Α και Β υπάρχουν στο γράφο;
προσθέτουμε τον κόμβο Α στο μονοπάτι;
όσο το μονοπάτι δεν είναι άδειο
{
      u→κορυφή του μονοπατιού;
      αν δεν έχω βάλει τα παιδιά του u στη στοίβα, προσθέτω όσα δεν είναι
      ήδη στο μονοπάτι;
      αν το e στην κορυφή της στοίβας προέρχεται από το u
      {
            αν ικανοποιείται ο περιορισμός
            {
                  αν (e == B)
                  {
                        τέλος;
                  }
                  αλλιώς
                        για κάθε κόμβο n από το κομμάτι e
                              αν ικανοποιείται ο περιορισμός
                                    αν (n  == B)
                                          τέλος;
                                    αλλιώς, αν n δεν υπάρχει ήδη στο
                                    μονοπάτι
                                          προσθέτω το n στο μονοπάτι;
            }
            βγάζουμε το e από τη στοίβα;
      }
      αλλιώς, βγάζουμε το u από το μονοπάτι;
}
```

Παράδειγμα 5.13. Θυμίζουμε ότι ψάχνουμε μονοπάτι ανάμεσα στους κόμβους 1 και 20 του Σχήματος 5.2.

Βήμα	κορυφή μονοπατιού	μονοπάτι	κορυφή στοίβας	στοίβα
1	-	1	-	-
2	1	1,13,2		12,2,10,3,4,14,15,6,16,17,6,7,8,3,13,2
3	2	1,13,2,4,5,9,3		12,2,10,3,4,14,15,6,16,17,6,7,8,3,13,2,10,3,4,4,5,9,3
…	…	…	…	…
N	16	1,…,16,18,19,20←ΒΡΕΘΗΚΕ		…,6,7,8,18,19,20

Πίνακας 5.10

Αν θέλουμε να βρούμε και ΠΟΙΟ είναι το μονοπάτι που συνδέει τους κόμβους 1 και 20, θα πρέπει να προσθέσουμε τον εξής επαναληπτικό βρόχο:

```
τρέχων κόμβος = κόμβος Β;
όσο τρέχων κόμβος != κόμβος Α
{
      τρέχων κόμβος = κόμβος πατέρας του τρέχοντος κόμβου;
      τύπωσε ή αποθήκευσε τον τρέχοντα κόμβο;
}
```

6

Αξιολόγηση

Στο κεφάλαιο αυτό παρουσιάζουμε την πειραματική αξιολόγηση των τεχνικών μας.

6.1 Σύστημα αξιολόγησης

Πριν προχωρήσουμε στην περιγραφή των πειραμάτων και την ανάλυση των αποτελεσμάτων τους, παρουσιάζουμε συνοπτικά τις μεθόδους που προτείνουμε καθώς και τις μεθόδους που χρησιμοποιούν λίστες γειτνίασης με τις οποίες τις συγκρίνουμε. Οι μέθοδοι παρουσιάζονται στους Πίνακες 6.1 και 6.2.

Οι αλγόριθμοι με το συνθετικό «OUR» στο όνομά τους είναι ακριβώς οι ίδιοι με τους αντίστοιχους «TRAD», με τη διαφορά ότι χρησιμοποιούν τον PATHINDEX και τον αλγόριθμο GETNEXT για να βρούμε τα παιδιά κάθε κόμβου.

Στο πλαίσιο των πειραμάτων χρησιμοποιήσαμε 6 συνθετικούς γράφους και 2 γράφους που προέρχονται από πραγματικά συστήματα. Οι συνθετικοί γράφοι ήταν των 10000 κόμβων ή των 100000 κόμβων και είχαν 2, 10 και 20 ακμές ανά κόμβο. Σε όλα τα πειράματα αναφέρουμε τους συνθετικούς γράφους με ονόματα NkEk, όπου N και E είναι το πλήθος των κόμβων και των ακμών, αντίστοιχα, σε χιλιάδες. Οι πραγματικοί γράφοι έχουν περίπου 450000 κόμβους και 2 ή 3 ακμές ανά κόμβο. Για κάθε γράφο είχαμε 3 διαφορετικές αναπαραστάσεις. Οι αναπαραστάσεις δημιουργούνται με τον αλγόριθμο που περιγράψαμε στην Ενότητα 5.1.

Για κάθε γράφο δημιουργήσαμε ένα τυχαίο σύνολο από ζευγάρια κόμβων. Αυτό το σύνολο το χωρίσαμε σε δύο υποσύνολα. Το ένα περιέχει τα ζευγάρια κόμβων μεταξύ των οποίων υπάρχει μονοπάτι και το άλλο τα ζευγάρια που δεν έχουν μονοπάτι μεταξύ τους. Το πρώτο υποσύνολο είναι το σύνολο των «NAI-queries» και το δεύτερο το σύνολο των «OXI-queries».

Όνομα Αλγορίθμου	Ερώτημα που Απαντάει	Περιγραφή
GETNEXT-TRAD	Εύρεση παιδιών του κόμβου A	Εύρεση άμεσων απογόνων σε γράφο με τη μορφή λιστών γειτνίασης
GETNEXT-PINDEX	Εύρεση παιδιών του κόμβου A	Εύρεση άμεσων απογόνων σε γράφο με χρήση PATHINDEX.
DFS-TRAD-EXISTPATH	Εύρεση μονοπατιού ανάμεσα στους κόμβους A και B	Αναζήτηση κατά βάθος σε γράφο με τη μορφή λιστών γειτνίασης.
DFS-TRAD-SHOWPATH	Υπολογισμός μονοπατιού ανάμεσα στους κόμβους A και B	Αναζήτηση κατά βάθος σε γράφο με τη μορφή λιστών γειτνίασης.
BFS-TRAD-EXISTPATH	Εύρεση μονοπατιού ανάμεσα στους κόμβους A και B	Αναζήτηση κατά πλάτος σε γράφο με τη μορφή λιστών γειτνίασης.
BFS-TRAD-SHOWPATH	Υπολογισμός μονοπατιού ανάμεσα στους κόμβους A και B	Αναζήτηση κατά πλάτος σε γράφο με τη μορφή λιστών γειτνίασης.
DFS-SEP-EXISTPATH	Εύρεση μονοπατιού ανάμεσα στους κόμβους A και B	Αναζήτηση κατά βάθος σε γράφο με χρήση PATHINDEX. Τοποθέτηση κάθε κόμβου χωριστά στη στοίβα.
DFS-SEP-SHOWPATH	Υπολογισμός μονοπατιού ανάμεσα στους κόμβους A και B	Αναζήτηση κατά βάθος σε γράφο με χρήση PATHINDEX. Τοποθέτηση κάθε κόμβου χωριστά στη στοίβα.
BFS-SEP-EXISTPATH	Εύρεση μονοπατιού ανάμεσα στους κόμβους A και B	Αναζήτηση κατά πλάτος σε γράφο με χρήση PATHINDEX. Τοποθέτηση κάθε κόμβου χωριστά στη στοίβα.
BFS-SEP-SHOWPATH	Υπολογισμός μονοπατιού ανάμεσα στους κόμβους A και B	Αναζήτηση κατά πλάτος σε γράφο με χρήση PATHINDEX. Τοποθέτηση κάθε κόμβου χωριστά στη στοίβα.
DFS-MERGE-EXISTPATH	Εύρεση μονοπατιού ανάμεσα στους κόμβους A και B	Αναζήτηση κατά βάθος σε γράφο με χρήση PATHINDEX. Τοποθέτηση ολόκληρων κομματιών από μονοπάτια στη στοίβα.
DFS-MERGE-SHOWPATH	Υπολογισμός μονοπατιού ανάμεσα στους κόμβους A και B	Αναζήτηση κατά βάθος σε γράφο με χρήση PATHINDEX. Τοποθέτηση ολόκληρων κομματιών από μονοπάτια στη στοίβα.
BFS-MERGE-EXISTPATH	Εύρεση μονοπατιού ανάμεσα στους κόμβους A και B	Αναζήτηση κατά πλάτος σε γράφο με χρήση PATHINDEX. Τοποθέτηση ολόκληρων κομματιών από μονοπάτια στην ουρά.
BFS-MERGE-SHOWPATH	Υπολογισμός μονοπατιού ανάμεσα στους κόμβους A και B	Αναζήτηση κατά πλάτος σε γράφο με χρήση PATHINDEX. Τοποθέτηση ολόκληρων κομματιών από μονοπάτια στην ουρά.
DFS-OUR-EXISTPATH	Εύρεση μονοπατιού ανάμεσα στους κόμβους A και B	Αναζήτηση κατά βάθος σε γράφο με χρήση PATHINDEX.
DFS-OUR-SHOWPATH	Υπολογισμός μονοπατιού ανάμεσα στους κόμβους A και B	Αναζήτηση κατά βάθος σε γράφο με χρήση PATHINDEX.
BFS-OUR-EXISTPATH	Εύρεση μονοπατιού ανάμεσα στους κόμβους A και B	Αναζήτηση κατά πλάτος σε γράφο με χρήση PATHINDEX.
BFS-OUR-SHOWPATH	Υπολογισμός μονοπατιού ανάμεσα στους κόμβους A και B	Αναζήτηση κατά πλάτος σε γράφο με χρήση PATHINDEX.

Πίνακας 6.1

Στα πειράματά μετρήσαμε το χρόνο που χρειάζεται η κάθε μέθοδος να απαντήσει ένα ερώτημα. Οι μετρήσεις έγιναν σε χτύπους ρολογιού (CLOCKTICKS). Για το EXIST/SHOWPATH τα αποτελέσματα αναπαρίστανται σε δευτερόλεπτα. Η μετατροπή βασίζεται στην ισότητα:

$$\chi\rho\acute{o}\nu o\varsigma\ \sigma\varepsilon\ sec = \frac{CLOCKTICKS}{CLOCKSPERSEC}$$

όπου CLOCKSPERSEC είναι μια σταθερά που εξαρτάται από το ρολόι του υπολογιστή. Στον υπολογιστή όπου έγιναν τα πειράματα ισχύει: CLOCKSPERSEC = 1.000.000 CLOCKTICKS/sec. Η σύγκρισή των μεθόδων έγινε με υπολογισμό του ποσοστού των περιπτώσεων που μια μέθοδος είναι καλύτερη (δηλαδή δίνει απάντηση σε μικρότερο χρόνο) από μια άλλη, αλλά και του ποσοστού

του χρόνου που η καλύτερη μέθοδος κερδίζει ή χάνει ως προς την άλλη μέθοδο.

Οι παράμετροι που μας ενδιαφέρουν είναι ο λόγος του πλήθους των ακμών προς το πλήθος των κόμβων, καθώς και το πλήθος των κόμβων.

Όνομα Αλγορίθμου	Ερώτημα που Απαντάει	Περιγραφή
TRAD-EXISTPATHCONSTRAINT	Εύρεση μονοπατιού ανάμεσα στους κόμβους A και B υπό συνθήκη	Αναζήτηση κατά βάθος σε γράφο με τη μορφή λιστών γειτνίασης. Δεκτά μόνο τα μονοπάτια των οποίων το μήκος ικανοποιεί κάποια συνθήκη.
TRAD-SHOWPATHCONSTRAINT	Υπολογισμός μονοπατιού ανάμεσα στους κόμβους A και B υπό συνθήκη	Αναζήτηση κατά βάθος σε γράφο με τη μορφή λιστών γειτνίασης. Δεκτά μόνο τα μονοπάτια των οποίων το μήκος ικανοποιεί κάποια συνθήκη.
PARTS-EXISTPATHCONSTRAINT	Εύρεση μονοπατιού ανάμεσα στους κόμβους A και B υπό συνθήκη	Αναζήτηση κατά βάθος σε γράφο με χρήση PATHINDEX. Τοποθέτηση ολόκληρων κομματιών από μονοπάτια στη στοίβα. Δεκτά μόνο τα μονοπάτια των οποίων το μήκος ικανοποιεί κάποια συνθήκη.
PARTS-SHOWPATHCONSTRAINT	Υπολογισμός μονοπατιού ανάμεσα στους κόμβους A και B υπό συνθήκη	Αναζήτηση κατά βάθος σε γράφο με χρήση PATHINDEX. Τοποθέτηση ολόκληρων κομματιών από μονοπάτια στη στοίβα. Δεκτά μόνο τα μονοπάτια των οποίων το μήκος ικανοποιεί κάποια συνθήκη.
NODES-EXISTPATHCONSTRAINT	Εύρεση μονοπατιού ανάμεσα στους κόμβους A και B υπό συνθήκη	Αναζήτηση κατά βάθος σε γράφο με χρήση PATHINDEX. Τοποθέτηση κάθε κόμβου χωριστά στη στοίβα. Δεκτά μόνο τα μονοπάτια των οποίων το μήκος ικανοποιεί κάποια συνθήκη.
NODES-SHOWPATHCONSTRAINT	Υπολογισμός μονοπατιού ανάμεσα στους κόμβους A και B υπό συνθήκη	Αναζήτηση κατά βάθος σε γράφο με χρήση PATHINDEX. Τοποθέτηση κάθε κόμβου χωριστά στη στοίβα. Δεκτά μόνο τα μονοπάτια των οποίων το μήκος ικανοποιεί κάποια συνθήκη.

Πίνακας 6.2

6.1.1 Εύρεση/Υπολογισμός μονοπατιού ανάμεσα στους κόμβους A και B

Για να εξετάσουμε την απόδοση των αλγορίθμων μας κάναμε τα εξής πειράματα:

1. Συγκρίναμε τις μεθόδους DFS-SEP και DFS-MERGE μεταξύ τους, όπως και τις μεθόδους BFS-SEP και BFS-MERGE μεταξύ τους. Με τον τρόπο αυτό βρήκαμε την καλύτερη DFS και BFS μέθοδο.
2. Συγκρίναμε την κάθε μία από τις παραπάνω με την αντίστοιχη TRAD μέθοδο.
3. Συγκρίναμε την καλύτερη BFS με την καλύτερη DFS μέθοδο από αυτές που προτείνουμε εμείς.
4. δώσαμε ραβδογράμματα που δείχνουν πώς μεταβάλλεται ο χρόνος που χρειάζεται η κάθε μέθοδος ανάλογα με τον τρόπο αναπαράστασης του γράφου σε μονοπάτια.

Εκτελέσαμε κάθε ερώτημα (query) 10 φορές (ή 1, σε περίπτωση χρονοβόρας εκτέλεσης). Για τα OXI-queries των γράφων 10k200k, 100k1m και 100k2m χρησιμοποιήσαμε δύο αναπαραστάσεις.

6.1.2 Εύρεση/Υπολογισμός μονοπατιού ανάμεσα στους κόμβους A και B με περιορισμούς

Η διαδικασία που ακολουθήσαμε για τα πειράματα σχετικά με την εύρεση μονοπατιού ανάμεσα στους κόμβους A και B με κάποιο περιορισμό είναι λίγο διαφορετική από τη

διαδικασία που ακολουθήσαμε στην Ενότητα 6.2.1.

1. Συγκρίναμε τις μεθόδους που προτείνουμε εμείς μεταξύ τους για να βρούμε την καλύτερη.
2. Συγκρίναμε την κάθε μία από τις παραπάνω με τη μέθοδο που χρησιμοποιεί λίστα γειτνίασης(TRAD).
3. Κατασκευάσαμε διαγράμματα που δείχνουν πώς συμπεριφέρονται οι μέθοδοι σε διάφορα σενάρια που θεωρήσαμε σκόπιμο να μελετήσουμε.

Τα πειράματα που εκτελέσαμε έγιναν πάνω στους συνθετικούς γράφους. Επαναλάβαμε τα OXI-queries 20 φορές για τους γράφους 10k20k και 100k200k και 10 φορές για τους υπόλοιπους. Για τα OXI-queries χρησιμοποιήθηκαν δύο αναπαραστάσεις κάθε γράφου.

Για τα πειράματά μας εκτελέσαμε το ερώτημα απαιτώντας τα μονοπάτια που βρίσκουμε να έχουν μήκος το πολύ 100, 200, 500 και 1000 κόμβους, για τα NAI-queries και 3, 5 και 10 για τα OXI-queries.

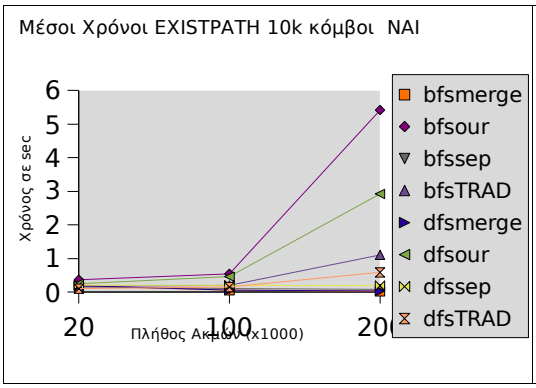
6.2 Αποτελέσματα

6.2.1 Εύρεση/Υπολογισμός μονοπατιού ανάμεσα στους κόμβους A και B

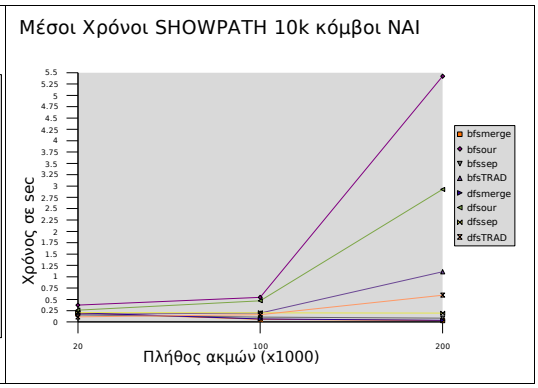
6.2.1.1 Συνθετικοί Γράφοι

Αρχικά πήραμε το μέσο όρο των χρόνων από τις 3 αναπαραστάσεις για κάθε query και στη συνέχεια πήραμε το μέσο χρόνο που χρειάζεται η κάθε μέθοδος για να εκτελέσει τα queries. Επαναλάβαμε τη διαδικασία για όλους τους γράφους και χωριστά για τα NAI-queries και τα OXI-queries.

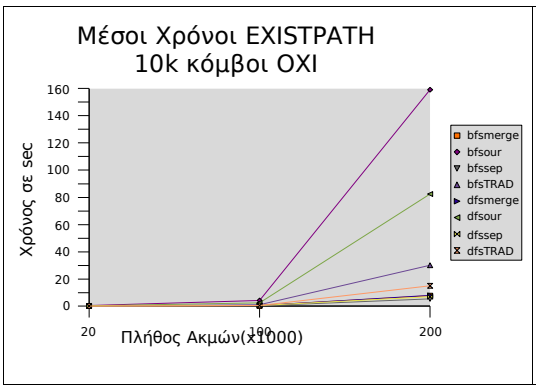
Τα διαγράμματα που προέκυψαν είναι τα Διαγράμματα 6.1 ως 6.8. Τα διαγράμματα έγιναν για γράφους με 10k και 100k κόμβους (1k = 1000).



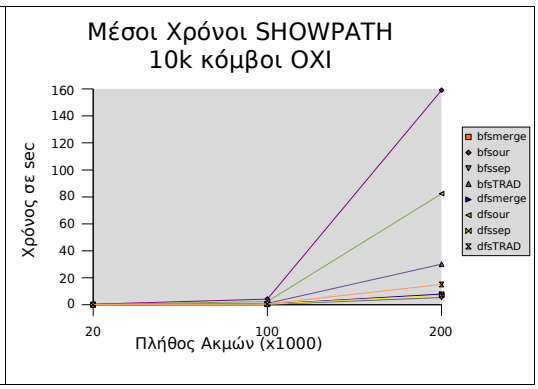
Διάγραμμα 6.1



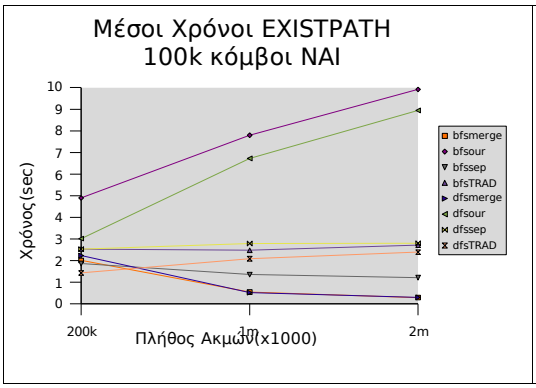
Διάγραμμα 6.2



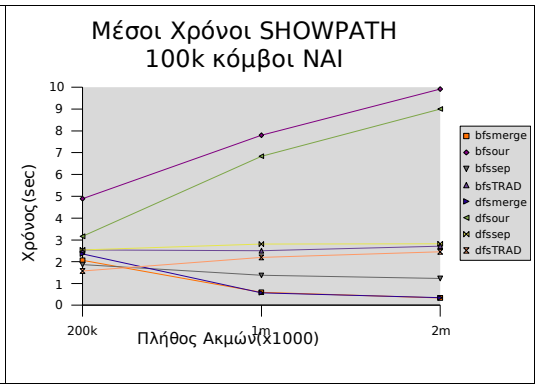
Διάγραμμα 6.3



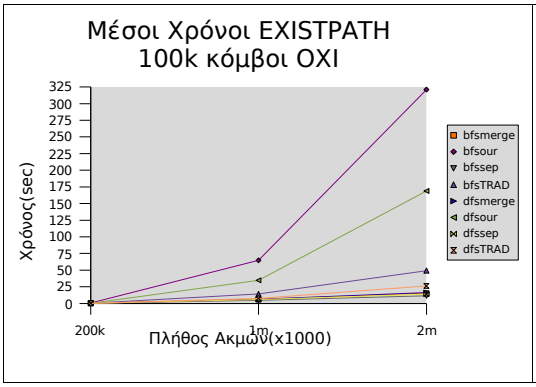
Διάγραμμα 6.4



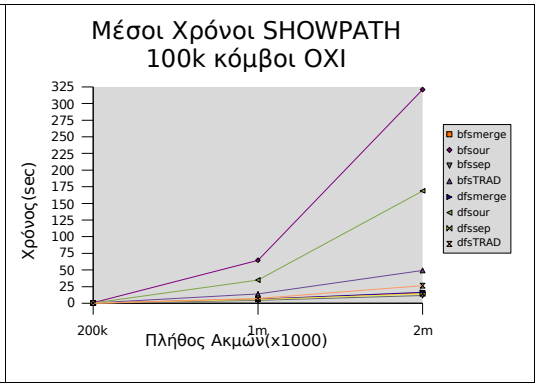
Διάγραμμα 6.5



Διάγραμμα 6.6



Διάγραμμα 6.7



Διάγραμμα 6.8

Παρατηρώντας τα Διαγράμματα 6.1 ως 6.8 διαπιστώνουμε ότι:

Γενικά, οι μέθοδοι που χρησιμοποιούν αναπαραστάσεις με μονοπάτια είναι καλύτερες από τις TRAD μεθόδους, διότι, λόγω της προσθήκης ολόκληρων κομματιών από μονοπάτια οι επαναλήψεις στα βήματα της αναζήτησης είναι λιγότερες και επομένως ο χρόνος που χρειάζεται για την εκτέλεση του κάθε query είναι μικρότερος.

Για τα OXI-queries έχουμε πολύ μικρούς χρόνους για τους πολύ αραιούς γράφους, οι οποίοι αυξάνονται, όσο οι γράφοι γίνονται πιο πυκνοί. Επίσης, όσο πιο πυκνοί γίνονται οι γράφοι, οι διαφορές στους χρόνους μεγαλώνουν. Οι μέθοδοι που χρησιμοποιούν αναπαραστάσεις με μονοπάτια διατηρούν πολύ καλούς χρόνους, τους οποίους ανταγωνίζονται οι DFS-TRAD και BFS-TRAD,

διατηρώντας χαμηλά τους αντίστοιχους χρόνους.

Από την άλλη, για τα NAI-queries έχουμε διαφορετική συμπεριφορά για τους μικρούς (10k) και τους μεγάλους (100k) γράφους.

Για τους μικρούς γράφους, έχουμε μια διαφοροποίηση των χρόνων των μεθόδων BFSSEP, DFSMERGE και BFSMERGE. Συγκεκριμένα, παρατηρούμε ότι οι χρόνοι που χρειάζονται αυτές οι μέθοδοι μειώνονται όσο προχωράμε προς πιο πυκνούς γράφους. Αυτό οφείλεται στο ότι φέρνουμε ολόκληρα κομμάτια μονοπατιών στο μέτωπο αναζήτησης. Με την κίνηση αυτή και αποκλείοντας την προσθήκη ή εξερεύνηση κόμβων που έχουμε ήδη επεξεργαστεί, μειώνουμε το χρόνο εκτέλεσης.

Για τα NAI-queries των πιο μεγάλων γράφων έχουμε σταθεροποίηση των χρόνων και για τις TRAD μεθόδους. Αυτό είναι αναμενόμενο, διότι οι γράφοι πλέον είναι αρκετά μεγάλοι και πυκνοί, με αποτέλεσμα, τα queries να εκτελούνται στον ίδιο χρόνο για όλους τους γράφους.

Τέλος, όπως είναι αναμενόμενο, οι μέθοδοι OUR έχουν τους χειρότερους χρόνους, αφού στην ουσία κάνουν ότι και οι TRAD αλλά με περισσότερο χρονοβόρες διαδικασίες.

Στη συνέχεια, θα περιγράψουμε τα βήματα που απαιτούνται για τον εντοπισμό της καλύτερης μεθόδου.

Βήμα 1.

Αρχικά, συγκρίναμε μεταξύ τους τις μεθόδους που ακολουθούν την ίδια τεχνική τοποθέτησης στο μέτωπο αναζήτησης. Συγκρίναμε, δηλαδή, πρώτα τις BFS και μετά τις DFS μεθόδους που στηρίζονται στην αναπαράσταση μονοπατιών. Από τη σύγκριση αυτή προέκυψε ο Πίνακας 6.3 για τα NAI-queries και ο Πίνακας 6.4 για τα OXI-queries, όπου φαίνονται οι επικρατέστερες από τις μεθόδους για κάθε γράφο.

	NAI			
	BEST BFS με PATHINDEX		BEST DFS με PATHINDEX	
	bfsmerge-bfssep		dfsmerge-dfssep	
	ExistPath	ShowPath	ExistPath	ShowPath
10k20k	SEP		MERGE	
10k100k	MERGE		MERGE	
10k200k	MERGE		MERGE	
100k200k	MERGE		MERGE	
100k1m	MERGE		MERGE	
100k2m	MERGE		MERGE	

Πίνακας 6.3

	OXI			
	BEST BFS με PATHINDEX		BEST DFS με PATHINDEX	
	bfsmerge-bfssep		dfsmerge-dfssep	
	ExistPath	ShowPath	ExistPath	ShowPath
10k20k	SEP		MERGE	
10k100k	MERGE		MERGE	
10k200k	SEP		SEP	
100k200k	SEP		MERGE	
100k1m	SEP		SEP	
100k2m	SEP		SEP	

Πίνακας 6.4

Οι Πίνακες 6.5-6.6 περιέχουν για κάθε μέθοδο το λόγο

(διαφορά χρόνων)/(χρόνος καλύτερης μεθόδου)

για όλους τους γράφους, δηλαδή το πόσο κερδίζει ή χάνει η καλύτερη μέθοδος σε κάθε περίπτωση. Επιπλέον, αναφέρεται και η καλύτερη μέθοδος από κάθε σύγκριση.

	NAI							
	BEST BFS με PATHINDEX				BEST DFS με PATHINDEX			
	ExistPath		CalcPath		ExistPath		CalcPath	
	κερδίζει	χάνει	κερδίζει	χάνει	κερδίζει	χάνει	κερδίζει	χάνει
10k20k	SEP				MERGE			
10k20k	0.28	0.05	0.3	0.04	0.3	0.05	0.28	0.06
10k100k	MERGE				MERGE			
10k100k	0	0.83	0	0.7	0.44	2.46	0.43	2.2
10k200k	MERGE				MERGE			
10k200k	0	2.07	0	1.8	0.21	5.19	0.28	4.55
100k200k	MERGE				MERGE			
100k200k	0.13	0.03	0.15	0.02	0.34	0.32	0.35	0.28
100k1m	MERGE				MERGE			
100k1m	0.06	1.49	0.5	1.31	0	4.31	0	3.88
100k2m	MERGE				MERGE			
100k2m	0	3.08	0	2.64	0	8.23	0	7.14

Πίνακας 6.5

	OXI							
	BEST BFS με PATHINDEX				BEST DFS με PATHINDEX			
	ExistPath		CalcPath		ExistPath		CalcPath	
	κερδίζει	χάνει	κερδίζει	χάνει	κερδίζει	χάνει	κερδίζει	χάνει
10k20k	SEP				MERGE			
10k20k	0.34	0.02	0.37	0.04	0.05	0.06	0.06	0.04
10k100k	MERGE				MERGE			
10k100k	0.34	0	0.34	0	0.15	0	0.17	0
10k200k	SEP				SEP			
10k200k	0.41	0	0.43	0	0.12	0	0.13	0
100k200k	SEP				MERGE			
100k200k	0.19	0	0.22	0	0.22	0	0	0.08
100k1m	SEP				SEP			
100k1m	0.3	0	0.32	0	0.05	0	0.06	0
100k2m	SEP				SEP			
100k2m	0.36	0	0.37	0	0.16	0	0.17	0

Πίνακας 6.6

Βλέπουμε ότι κατά μέσο όρο οι καλύτερες DFS και BFS μέθοδοι κερδίζουν ένα 20%, ενώ χάνουν αρκετά σε ορισμένες περιπτώσεις, όπως πχ η DFSMERGE στα NAI-queries. Σε αρκετές περιπτώσεις χάνουν ένα πολύ μεγάλο ποσοστό, ακόμα και 200%. Αυτό προφανώς μας κάνει σκεπτικούς για το κατά πόσο πρέπει να θεωρηθεί η καλύτερη μέθοδος σε αυτές τις περιπτώσεις. Παρατηρούμε ότι αυτό συμβαίνει ειδικά στους πυκνούς γράφους. Το γεγονός αυτό οφείλεται στη συνάρτηση existCommonPath (Ενότητα 5.3.3), διότι το πλήθος των μονοπατιών είναι πολύ μεγάλο και μπορεί να διαρκεί αρκετά. Στα OXI-queries έχουμε και πάλι κέρδος 20% κατά μέσο όρο, ενώ οι το ποσοστό που χάνουν είναι πολύ μικρό (μικρότερο του 10%). Στα OXI-queries επικρατούν οι SEP μέθοδοι και πάλι λόγω της existCommonPath.

Βήμα 2.

Συγκρίναμε την κάθε μία από τις μεθόδους των Πινάκων 6.3 και 6.4 με την αντίστοιχη TRAD μέθοδο. Τα ποσοστά στους Πίνακες 6.7 και 6.8 δείχνουν σε τί ποσοστό των queries ήταν καλύτερη η δική μας μέθοδος. Γενικά, οι δικές μας μέθοδοι είναι καλύτερες. Είναι πολύ καλύτερες για τα OXI-

queries. Για τα NAI-queries, οι DFS μέθοδοι δεν είναι και τόσο καλές σε σχέση με τις TRAD μεθόδους για τους αραιούς γράφους. Το ποσοστό των ερωτημάτων στα οποία απαντάνε καλύτερα είναι μικρότερο από το 50%, οπότε θεωρούμε ότι οι μέθοδοι αυτές δεν είναι καλύτερες από την αντίστοιχη TRAD μέθοδο. Εδώ χάνεται χρόνος από τις συνεχόμενες προσθήκες κόμβων στο μέτωπο αναζήτησης. Η συμπεριφορά αυτή είναι παρόμοια και στα OXI-queries. Στον Πίνακα 6.10 φαίνεται επίσης ότι η σχετική απώλεια είναι μεγάλη εκεί που η TRAD μέθοδος είναι καλύτερη.

NAI	BFS		DFS	
	ExistPath	ShowPath	ExistPath	ShowPath
10k20k	0.57	0.57	0.28	0.3
10k100k	0.8	0.8	0.8	0.79
10k200k	0.84	0.83	0.82	0.8
100k200k	0.57	0.56	0.38	0.39
100k1m	0.85	0.83	0.83	0.83
100k2m	0.86	0.84	0.91	0.9

Πίνακας 6.7

OXI	BFS		DFS	
	ExistPath	CalcPath	ExistPath	CalcPath
10k20k	0.98	0.97	0.62	0.62
10k100k	0.98	0.98	1	1
10k200k	1	1	1	1
100k200k	0.96	0.94	0.5	0.5
100k1m	1	1	1	1
100k2m	1	1	1	1

Πίνακας 6.8

Βήμα 3.

Έπειτα, συγκρίναμε τις καλύτερες μεθόδους μεταξύ τους και προέκυψαν τα αποτελέσματα που φαίνονται στον Πίνακα 6.9 (αναγράφονται οι επικρατέστερες μέθοδοι):

	Σύγκριση των καλύτερων μεταξύ τους			
	NAI		OXI	
	ExistPath	ShowPath	ExistPath	ShowPath
10k20k	bfssep		bfssep	
10k100k	bfsmerge		bfsmerge	
10k200k	bfsmerge		bfssep	
100k200k	bfsmerge		bfssep	
100k1m	dfsmerge		bfssep	
100k2m	bfsmerge		bfssep	

Πίνακας 6.9

Στον Πίνακα 6.10 φαίνονται τα ποσοστά απώλειας ή κέρδους της καλύτερης μεθόδου, όπως προκύπτουν από τα αποτελέσματα των συγκρίσεων των Βημάτων 2 και 3. Από αυτούς βλέπουμε ότι κυριαρχούν οι BFS αλγόριθμοι.

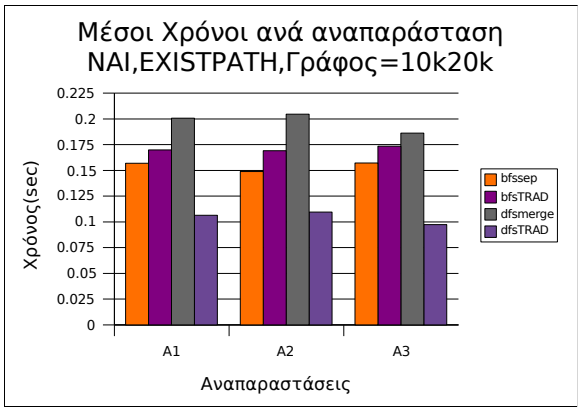
		BFS vs TRAD		DFS vs TRAD		BFS vs DFS	
10k20kNAI	ΚΑΛΥΤΕΡΗ	BFSSEP		ADJ		bfssep	
	χάνει	0.41	0.4	1.11	1.07	0.31	0.27
	κερδίζει	0.56	0.57	0.44	0.41	0.58	0.64
10k20kOXI	ΚΑΛΥΤΕΡΗ	BFSSEP		DFSMERGE		BFSSEP	
	κερδίζει	0	0	0	0	1.01	1.08
	χάνει	1.13	1.13	1.15	1.18	0	0
10k100kNAI	ΚΑΛΥΤΕΡΗ	bfsmerge		DFSMERGE		bfsmerge	
	κερδίζει	3.4	3.02	2.26	2.1	0.51	0.5
	χάνει	0.69	0.74	0.98	1.02	0.5	0.5
10k100kOXI	ΚΑΛΥΤΕΡΗ	bfsmerge		DFSMERGE		bfsmerge	
	κερδίζει	2.18	2.13	0.27	0.25	0.06	0.06
	χάνει	0	0	0	0	0	0
10k200kNAI	ΚΑΛΥΤΕΡΗ	bfsmerge		DFSMERGE		bfsmerge	
	κερδίζει	7.3	6.76	4.3	3.79	1.17	1.08
	χάνει	0.82	0.64	0.8	0.62	0.61	0.62
10k200kOXI	ΚΑΛΥΤΕΡΗ	BFSSEP		DFSMERGE		BFSSEP	
	κερδίζει	4.5487	4.55	0.93	0.9	0.45	0.46
	χάνει	0	0	0	0	0	0
100k200kNAI	ΚΑΛΥΤΕΡΗ	bfsmerge		DFSMERGE		bfsmerge	
	κερδίζει	0.83	0.78	0.48	0.48	0.5	0.47
	χάνει	0.33	0.34	0.98	0.92	0.44	0.4
100k200kOXI	ΚΑΛΥΤΕΡΗ	BFSSEP		DFSSEP		BFSSEP	
	κερδίζει	1.09	0.67	0	0	0.48	0.1
	χάνει	0	0	0.62	0.65	0	0
100k1mNAI	ΚΑΛΥΤΕΡΗ	bfsmerge		DFSMERGE		dfsmerge	
	κερδίζει	4.45	4.08	3.71	3.53	0.66	0.63
	χάνει	1.26	1.07	0.95	0.97	0.85	0.85
100k1mOXI	ΚΑΛΥΤΕΡΗ	BFSSEP		DFSSEP		BFSSEP	
	κερδίζει	2.1	2.09	0.26	0.26	0.22	0.22
	χάνει	0	0	0	0	0	0
100k2mNAI	ΚΑΛΥΤΕΡΗ	bfsmerge		DFSMERGE		bfsmerge	
	κερδίζει	1.38	1.34	0.2	0.19	0.03	0.03
	χάνει	0	0	0	0	0	0

Πίνακας 6.10

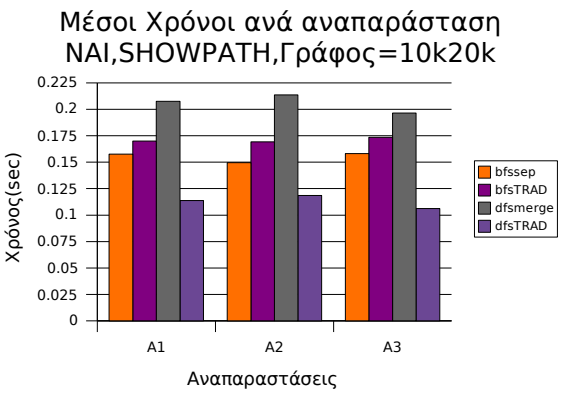
Βήμα 4.

Τέλος, παρουσιάζουμε σε ραβδογράμματα (Διαγράμματα 6.9 ως 6.32) τους χρόνους των καλύτερων μεθόδων (DFS και BFS) καθώς και των αντίστοιχων TRAD μεθόδων.

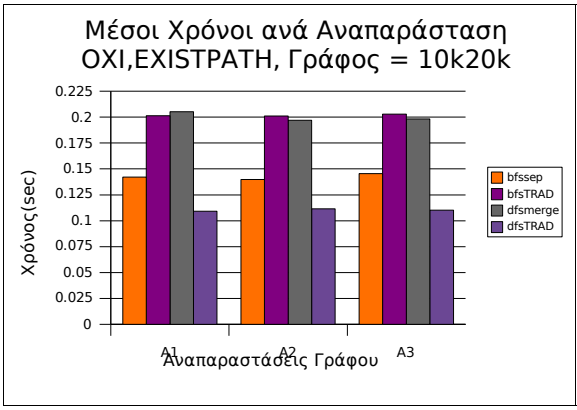
Οι μετρήσεις έγιναν, όπως και παραπάνω σε όλους τους γράφους και σε όλες τις αναπαραστάσεις. Τονίζουμε εδώ ότι, λόγω της μεγάλης καθυστέρησης κατά την πραγματοποίηση των πειραμάτων, τα OXI-queries των γράφων 10k200k, 100k1m και 100k2m έγιναν από μία φορά το καθένα και μόνο για 2 από τις 3 αναπαραστάσεις.



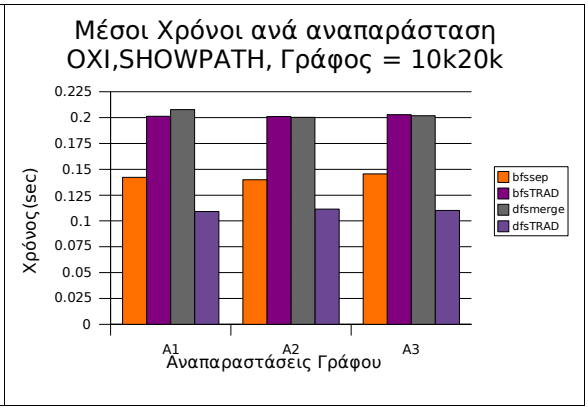
Διάγραμμα 6.9



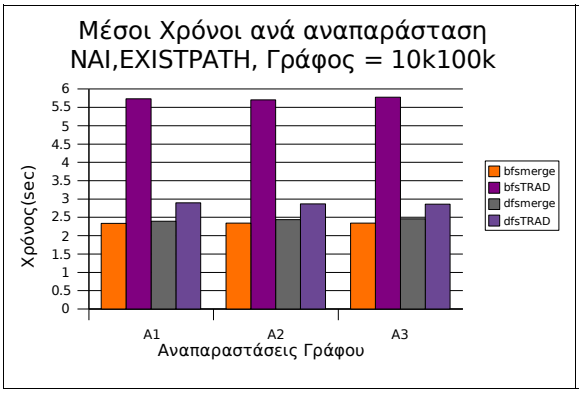
Διάγραμμα 6.10



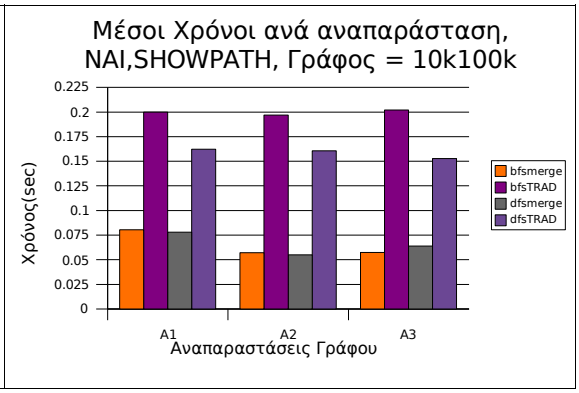
Διάγραμμα 6.11



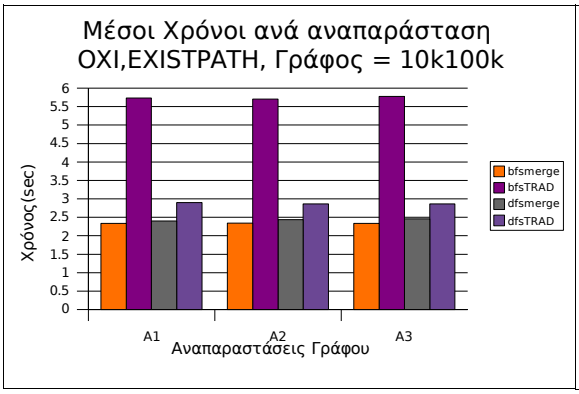
Διάγραμμα 6.12



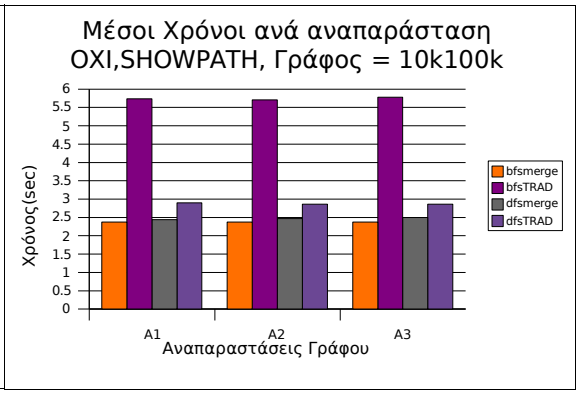
Διάγραμμα 6.13



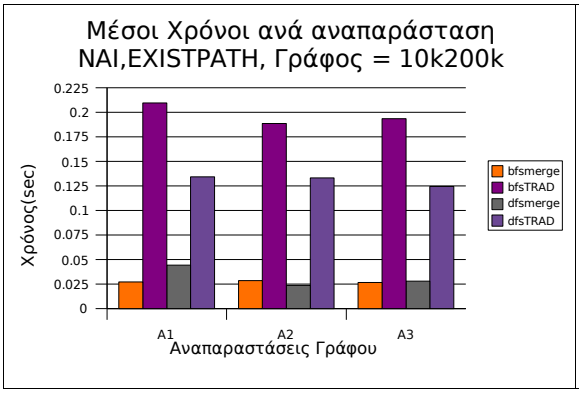
Διάγραμμα 6.14



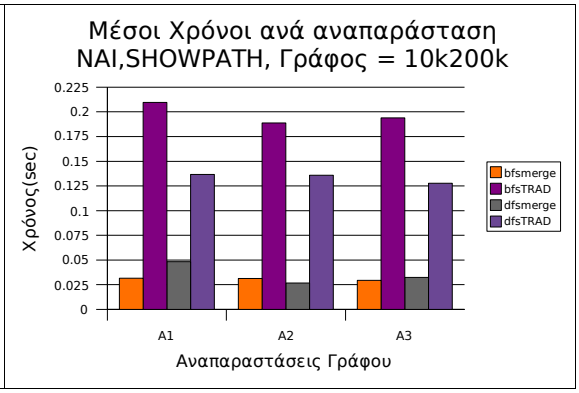
Διάγραμμα 6.15



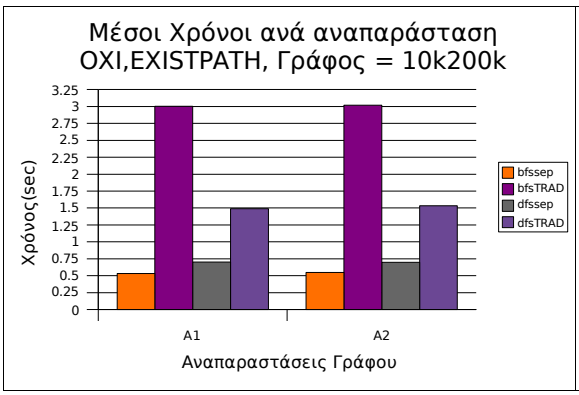
Διάγραμμα 6.16



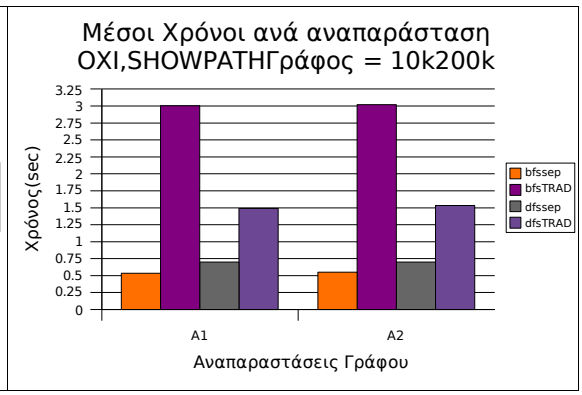
Διάγραμμα 6.17



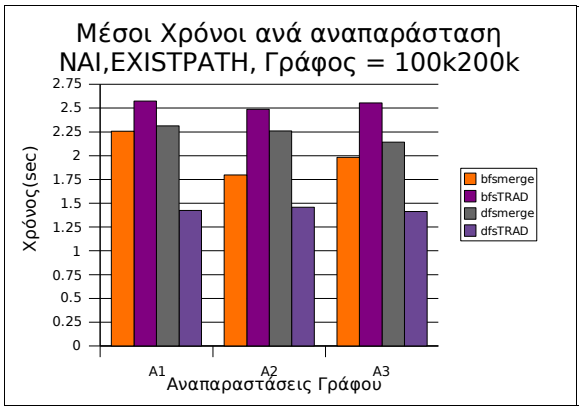
Διάγραμμα 6.18



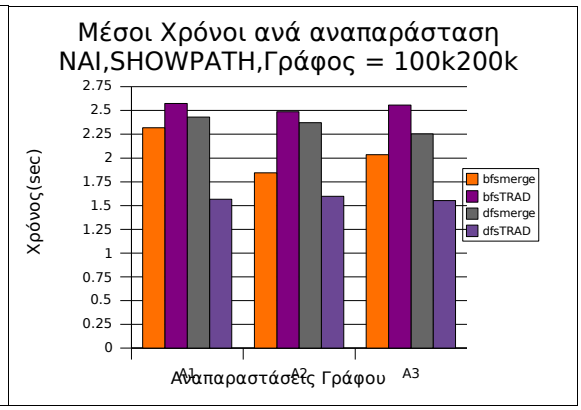
Διάγραμμα 6.19



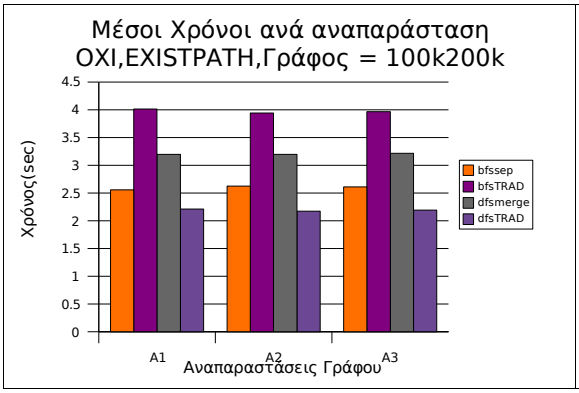
Διάγραμμα 6.20



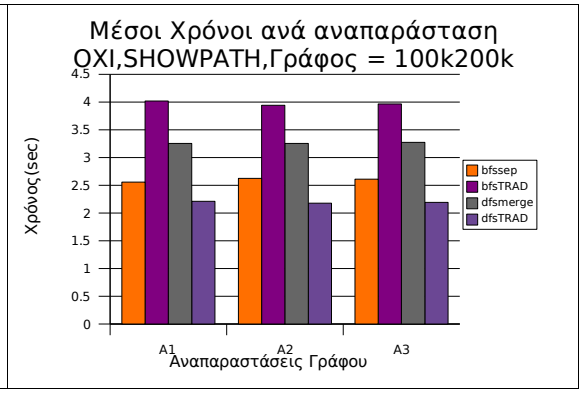
Διάγραμμα 6.21



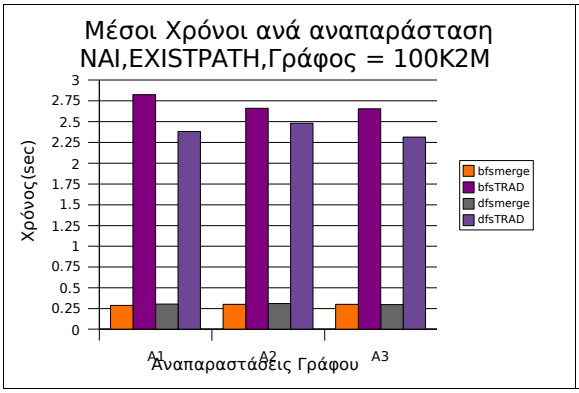
Διάγραμμα 6.22



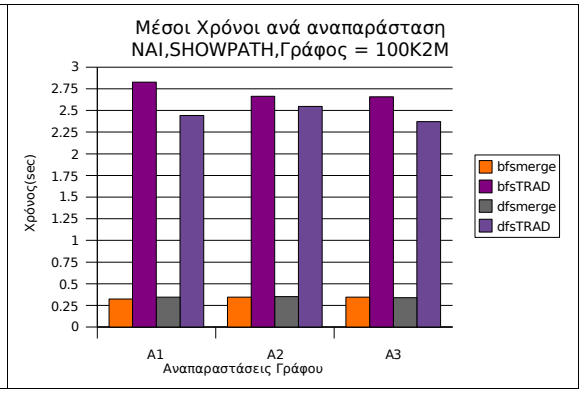
Διάγραμμα 6.23



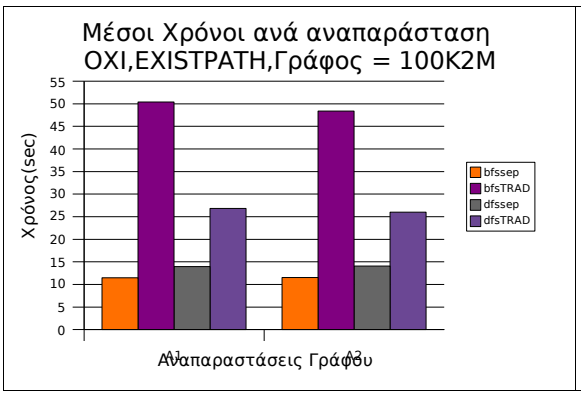
Διάγραμμα 6.24



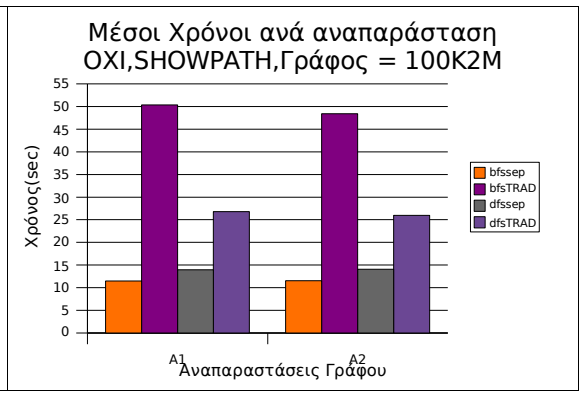
Διάγραμμα 6.25



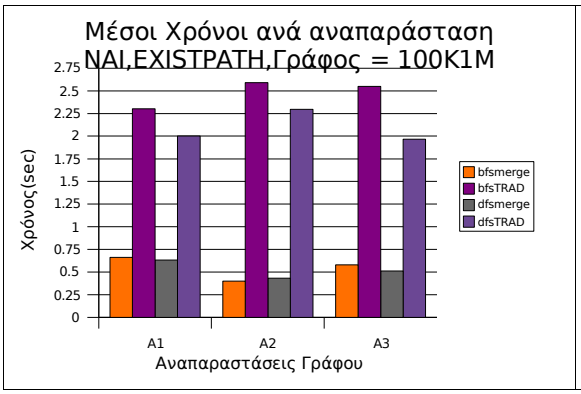
Διάγραμμα 6.26



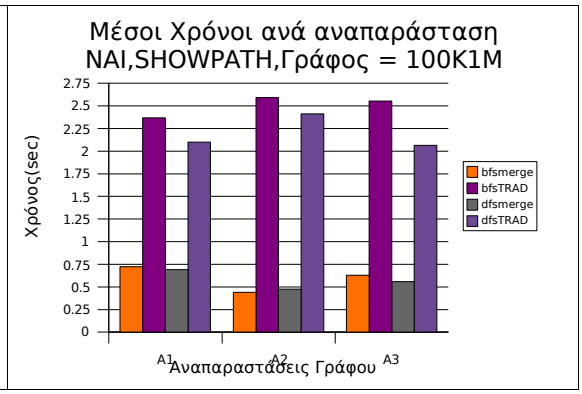
Διάγραμμα 6.27



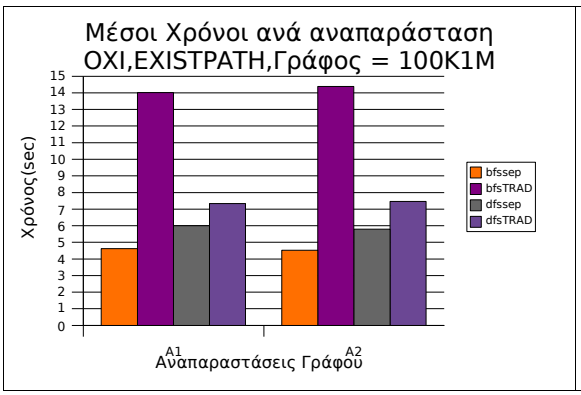
Διάγραμμα 6.28



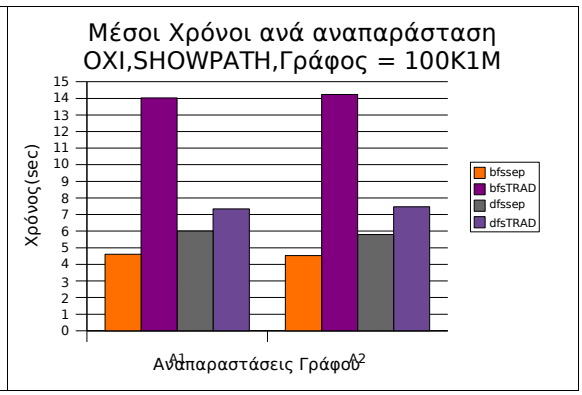
Διάγραμμα 6.29



Διάγραμμα 6.30



Διάγραμμα 6.31



Διάγραμμα 6.32

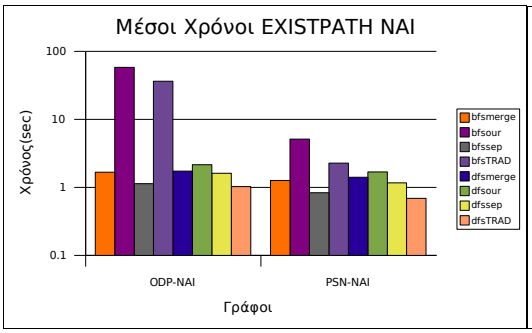
Παρατηρούμε ότι για τους αραιούς μας γράφους η DFS-TRAD είναι καλύτερη από τις καλύτερες δικές μας. Αυτό αλλάζει στους υπόλοιπους γράφους.

Στους γράφους 10k20k, 10k100k(για τον αλγόριθμο ShowPath), 10k200k, 100k200k και 100k1m και για τα NAI-queries έχουμε αυξομειώσεις των χρόνων ανάλογα με την αναπαράσταση στην οποία κάνουμε τα queries. Αυτό έχει να κάνει με τη μορφή των μονοπατιών στις αναπαραστάσεις, όπου έχουμε μεγάλα μονοπάτια στην αρχή και μικρά στο τέλος.

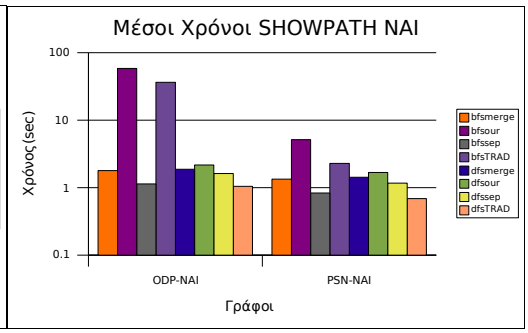
6.2.1.2 Πραγματικοί Γράφοι

Την ίδια διαδικασία ακολουθήσαμε και για τα πειράματα που αφορούν τους πραγματικούς γράφους.

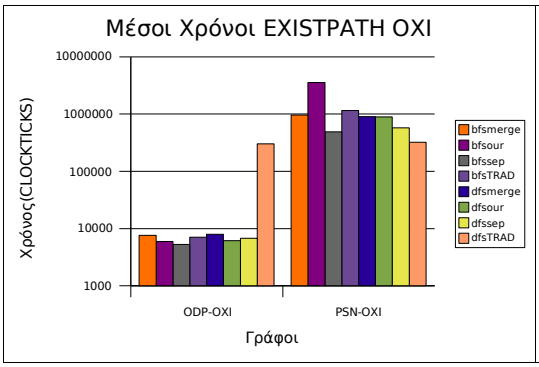
Τα συγκριτικά διαγράμματα των χρόνων φαίνονται στα Διαγράμματα 6.33 ως 6.36. Δυστυχώς, τα διαγράμματα αυτά δεν είναι ενδεικτικά της συμπεριφοράς των αλγορίθμων. Στην ουσία , η συμπεριφορά των αλγορίθμων είναι η ίδια και για τους δύο γράφους, αλλά λόγω της εμφάνισης ενός query που χρειάζεται υπερβολικά μεγάλο χρόνο για να εκτελεστεί, οι αλγόριθμοι BFSTRAD και BFSOUR φαίνεται να επιβαρύνονται. Πρέπει να τονίσουμε εδώ ότι ο ODP έχει λόγο ακμές/κόμβοι 2,3, ενώ ο PSN 3.



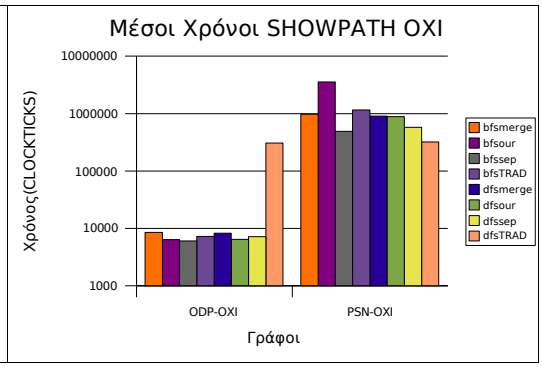
Διάγραμμα 6.33



Διάγραμμα 6.34



Διάγραμμα 6.35



Διάγραμμα 6.36

Συνεχίζουμε με τα βήματα που περιγράψαμε στην προηγούμενη ενότητα. Τα αναφέρουμε ξανά για να τα θυμηθούμε:

1. Βρήκαμε την καλύτερη DFS και BFS μέθοδο συγκρίνοντας μεταξύ τους τις μεθόδους που προτείναμε εμείς από κάθε κατηγορία.
2. Συγκρίναμε την κάθε μία από τις παραπάνω με την αντίστοιχη TRAD μέθοδο.
3. Συγκρίναμε την καλύτερη BFS με την καλύτερη DFS μέθοδο από αυτές που προτείνουμε εμείς.
4. δώσαμε ραβδογράμματα που δείχνουν πώς μεταβάλλεται ο χρόνος που χρειάζεται η κάθε μέθοδος ανάλογα με τον τρόπο αναπαράστασης του γράφου σε μονοπάτια.

Βήμα 1.

Οι Πίνακες 6.11 και 6.12 δείχνουν ότι, σε κάθε περίπτωση, η καλύτερη μέθοδος είναι η SEP. Η συμπεριφορά αυτή είναι παρόμοια με τη συμπεριφορά των αλγορίθμων στους αραιούς γράφους (10k20k) της προηγούμενης ενότητας.

	NAI			
	BEST BFS με PATHINDEX		BEST DFS με PATHINDEX	
	bfsmerge-bfssep		dfsmerge-dfssep	
	ExistPath	ShowPath	ExistPath	ShowPath
ODP	SEP		SEP	
PSN	SEP		SEP	

Πίνακας 6.11

	OXI			
	BEST BFS με PATHINDEX		BEST DFS με PATHINDEX	
	bfsmerge-bfssep		dfsmerge-dfssep	
	ExistPath	CalcPath	ExistPath	CalcPath
ODP	SEP		SEP	
PSN	SEP		SEP	

Πίνακας 6.12

Παρουσιάζουμε και τον Πίνακα 6.13 με το λόγο διαφορά_χρόνων/χρόνος_καλύτερης.

NAI							
BEST BFS με PATHINDEX				BEST DFS με PATHINDEX			
ExistPath		ShowPath		ExistPath		ShowPath	
κερδίζει	χάνει	κερδίζει	χάνει	κερδίζει	χάνει	κερδίζει	χάνει
SEP				SEP			
0.48	0.01	0.57	0.01	0.57	0.7	0.64	0.69
SEP				SEP			
0.53	0	0.61	0	0.24	0.29	0.26	0.29
OXI							
BEST BFS με PATHINDEX				BEST DFS με PATHINDEX			
ExistPath		ShowPath		ExistPath		ShowPath	
κερδίζει	χάνει	κερδίζει	χάνει	κερδίζει	χάνει	κερδίζει	χάνει
SEP				SEP			
1.23	2.37	1.19	1.97	0.8	1.6	0.79	1.16
SEP				SEP			
1.23	0.02	1.27	0.03	0.49	0.02	0.78	0.02

Πίνακας 6.13

Βήμα 2.

	BFS		DFS			BFS		DFS	
	ExistPath	ShowPath	ExistPath	ShowPath		ExistPath	ShowPath	ExistPath	ShowPath
ODP – bfssep – dfssep	0.96	0.96	0.3	0.32	ODP – bfssep – dfssep	0.17	0.17	0.23	0.23
PSN – bfsmerge – dfssep	0.93	0.93	0.06	0.06	PSN – bfsmerge – dfssep	0.19	0.19	0.17	0.17

Πίνακας 6.14

Βήμα 3

	NAI		OXI	
	ExistPath	ShowPath	ExistPath	ShowPath
ODP	bfssep		dfssep	
PSN	bfssep		dfssep	

Πίνακας 6.15

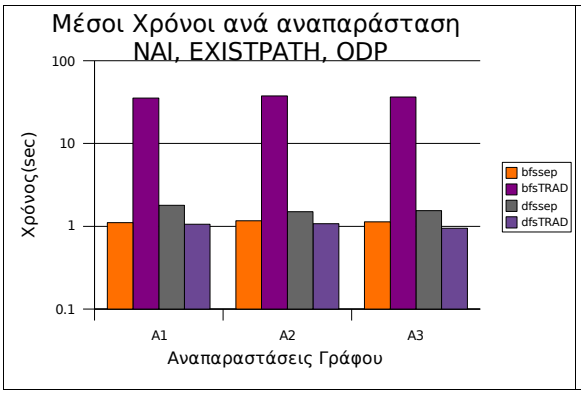
		BFS vs TRAD		DFS vs TRAD		BFS vs DFS	
ODP-NAI	**ΚΑΛΥΤΕΡΗ**	**BFSSEP**		**ADJ**		**BFSSEP**	
	χάνει	0.33	0.33	0.4	0.38	0.56	0.56
	κερδίζει	32.31	32.24	0.99	0.98	1.07	1.07
ODP-OXI	**ΚΑΛΥΤΕΡΗ**	**ADJ**		**ADJ**		**DFSSEP**	
	κερδίζει	0.51	0.6	0.02	0.02	0.46	0.52
	χάνει	5.21	4.2	3.15	3.14	1.8	1.61
PSN-NAI	**ΚΑΛΥΤΕΡΗ**	**BFSSEP**		**ADJ**		**BFSSEP**	
	χάνει	0.03	0.03	0.2	0.2	0.48	0.48
	κερδίζει	1.83	1.83	0.75	0.75	0.83	0.82
PSN-OXI	**ΚΑΛΥΤΕΡΗ**	**ADJ**		**ADJ**		**DFSSEP**	
	κερδίζει	0	0	0.96	0.97	0.01	0.01
	χάνει	3.16	3.32	0.05	0.05	0.53	0.51

Πίνακας 6.16

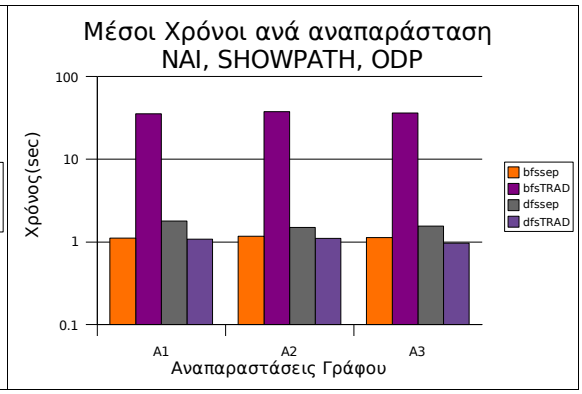
Όπως ίσως θα φανεί και παρακάτω, οι χρόνοι της BFS-SEP είναι καλύτεροι από της αντίστοιχης TRAD, παρ' όλο που για τα OXI η TRAD απαντάει πιο γρήγορα στα περισσότερα queries. Το ανάποδο ισχύει για την DFS. Επιπλέον, στη σύγκριση των καλύτερων μεταξύ τους η BFS είναι καλύτερη, ακόμα και στην περίπτωση που δεν απαντάει τα περισσότερα queries πιο γρήγορα.

Βήμα 4.

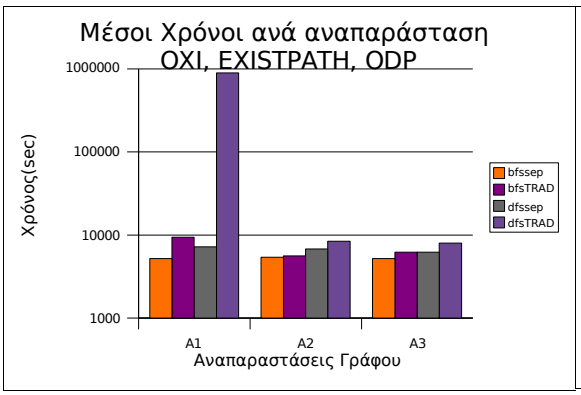
Η απόδοση των μεθόδων για κάθε αναπαράσταση φαίνεται στα Διαγράμματα 6.37 ως 6.44.



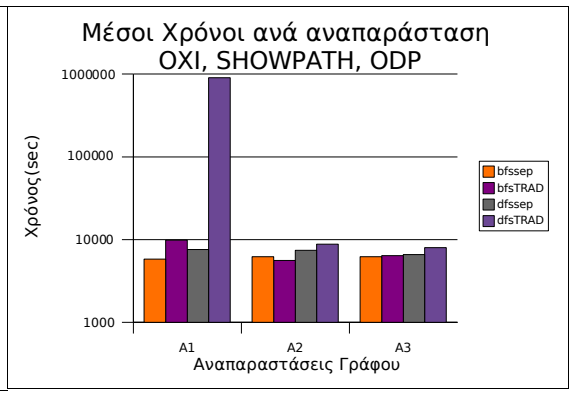
Διάγραμμα 6.37



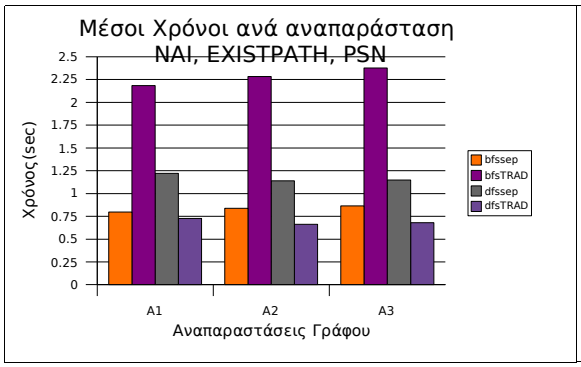
Διάγραμμα 6.38



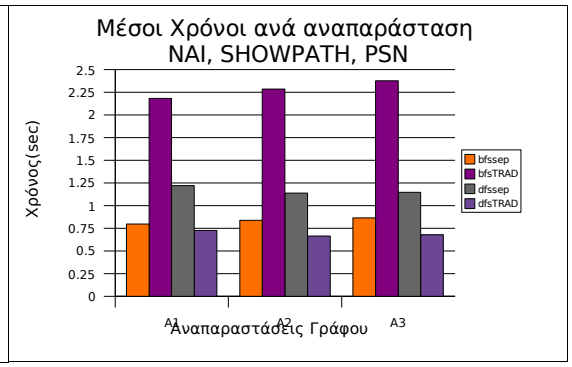
Διάγραμμα 6.39



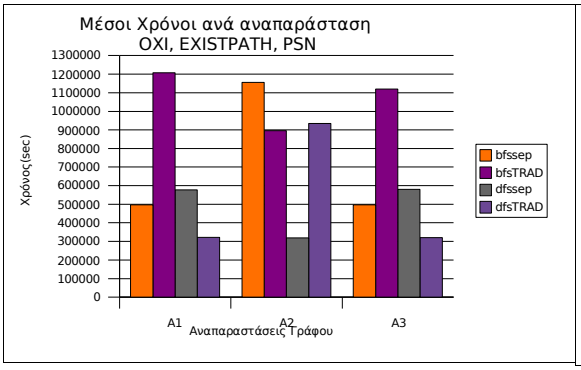
Διάγραμμα 6.40



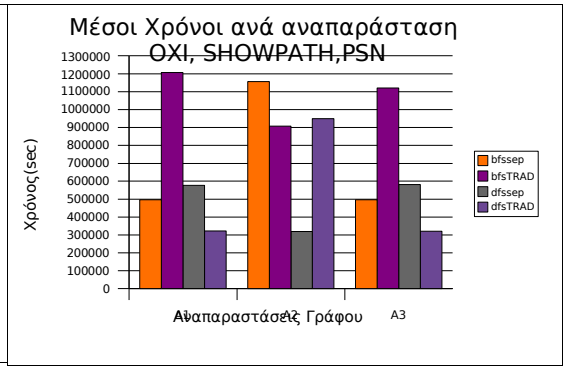
Διάγραμμα 6.41



Διάγραμμα 6.42



Διάγραμμα 6.43



Διάγραμμα 6.44

Παρατηρείται διακύμανση για τα OXI queries του PSN ανάλογα με τις αναπαραστάσεις. Επειδή οι γράφοι είναι πραγματικοί δεν έχουμε ίδιο πλήθος ακμών ανά κόμβο, για όλους τους κόμβους, επομένως είναι λογικές και οι διακυμάνσεις που παρατηρούνται, τόσο στους χρόνους απάντησης των queries, όσο και στους χρόνους της ίδιας μεθόδου για τις διάφορες αναπαραστάσεις.

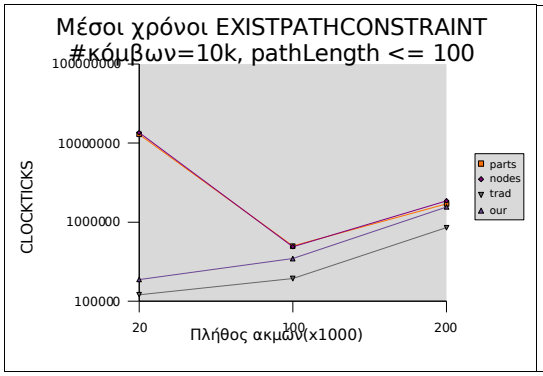
Τονίζουμε ότι όπου οι διαφορές στους μέσους όρους ήταν πολύ πολύ μεγάλες χρησιμοποιήθηκε λογαριθμική κλίμακα.

6.2.2 Εύρεση/Υπολογισμός μονοπατιού ανάμεσα στους κόμβους Α και Β με περιορισμούς

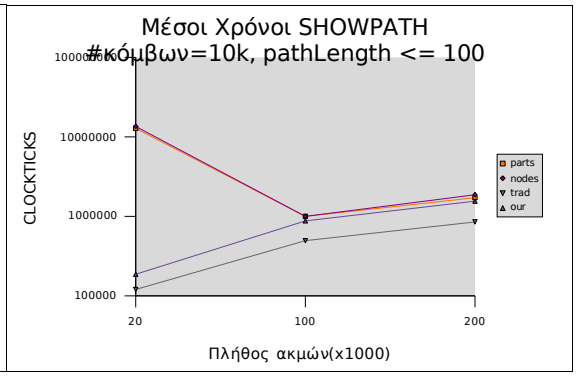
Αρχικά πήραμε το μέσο όρο των χρόνων από τις 3 αναπαραστάσεις για κάθε query και στη συνέχεια

πήραμε το μέσο χρόνο που χρειάζεται η κάθε μέθοδος για να εκτελέσει τα queries. Επαναλάβαμε τη διαδικασία για όλους τους γράφους και χωριστά για τα NAI-queries και τα OXI-queries.

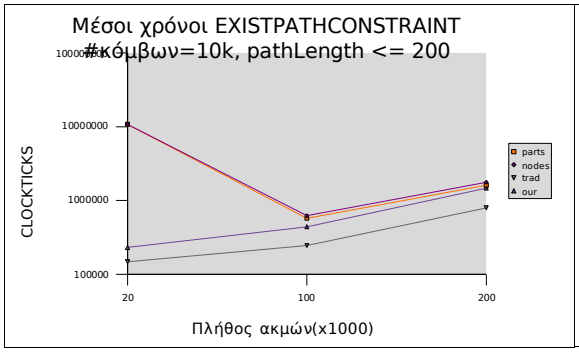
Τα αποτελέσματα για τα NAI-queries φαίνονται στα Διαγράμματα 6.45 ως 6.60.



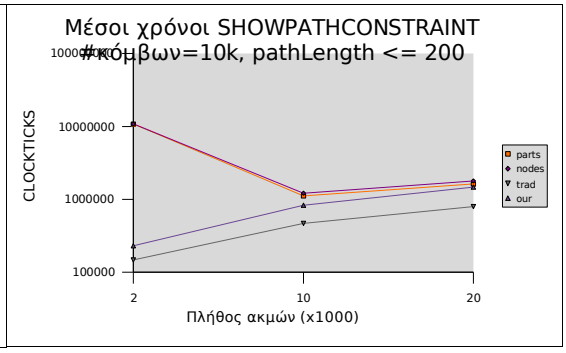
Διάγραμμα 6.45



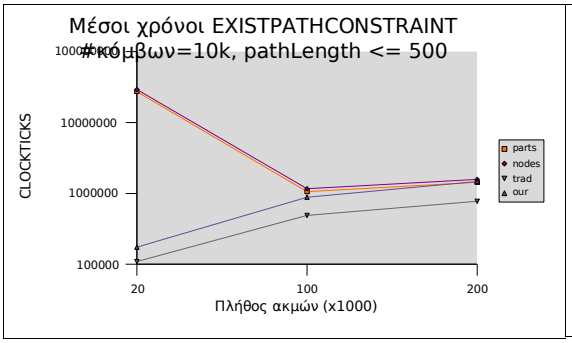
Διάγραμμα 6.46



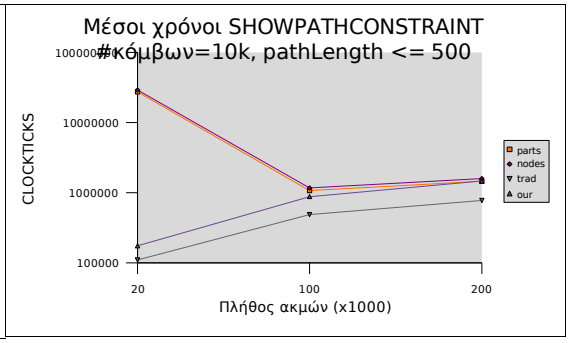
Διάγραμμα 6.47



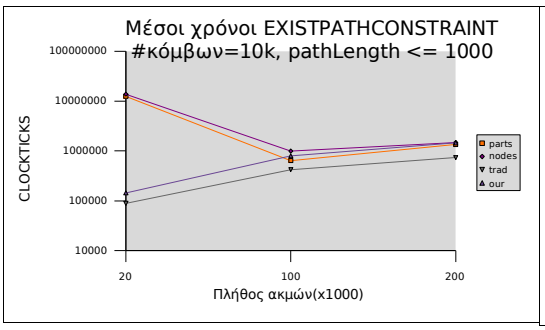
Διάγραμμα 6.48



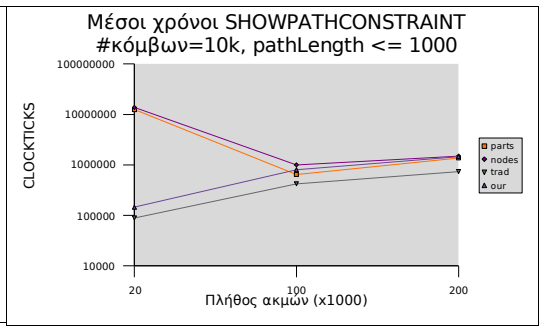
Διάγραμμα 6.49



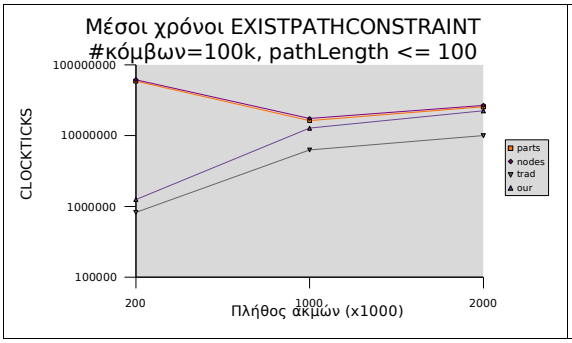
Διάγραμμα 6.50



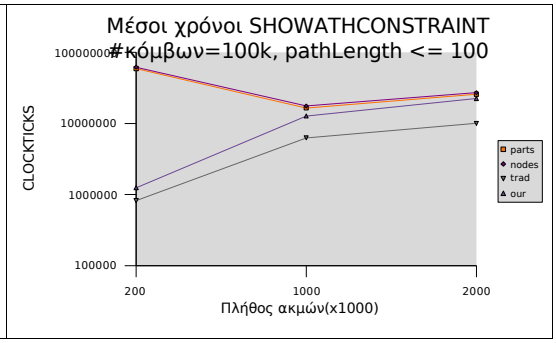
Διάγραμμα 6.51



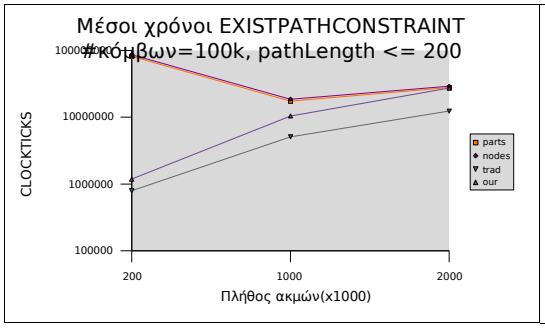
Διάγραμμα 6.52



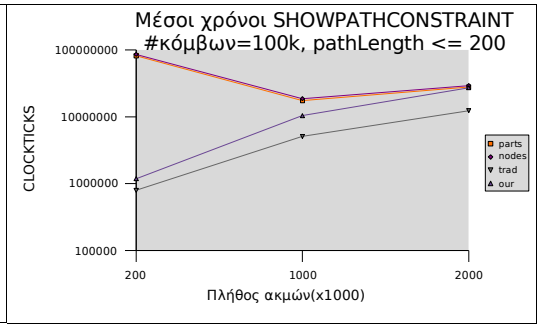
Διάγραμμα 6.53



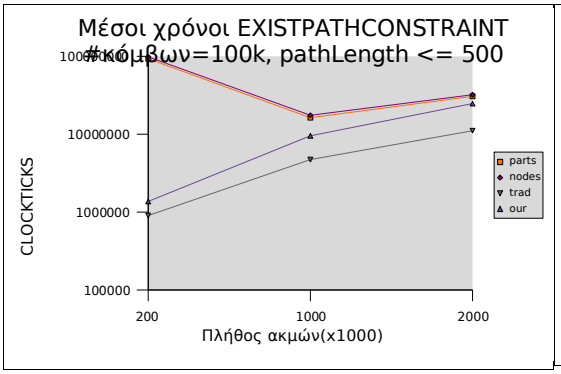
Διάγραμμα 6.54



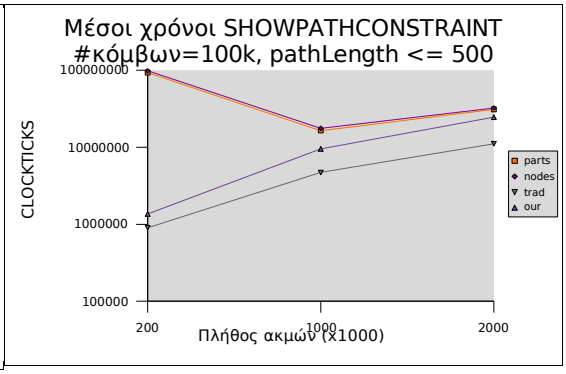
Διάγραμμα 6.55



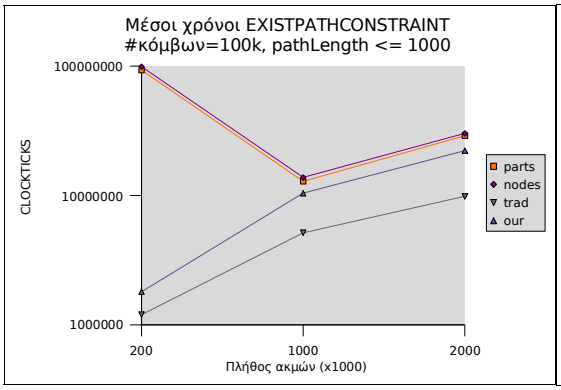
Διάγραμμα 6.56



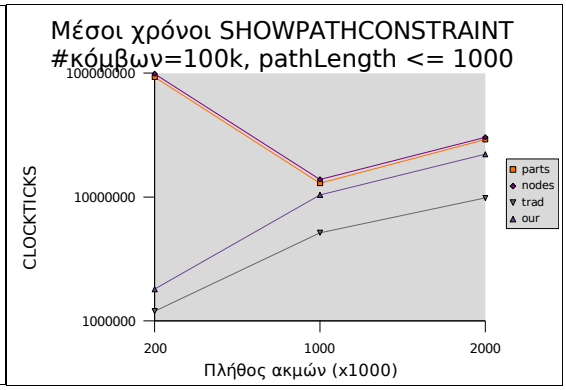
Διάγραμμα 6.57



Διάγραμμα 6.58



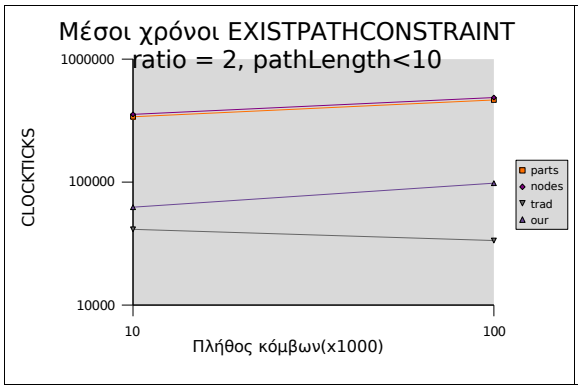
Διάγραμμα 6.59



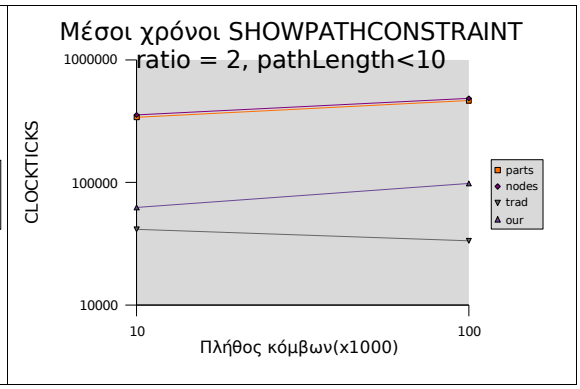
Διάγραμμα 6.60

Από τα Διαγράμματα 6.1 ως 6.60 βλέπουμε ότι γενικά οι μέθοδοί μας δεν καταφέρνουν να γίνουν καλύτερες από την TRAD. Για αραιούς γράφους η TRAD μέθοδος είναι πολύ καλύτερη, ενώ για πιο πυκνούς γράφους η δικές μας μέθοδοι την πλησιάζουν αρκετά. Η μικρότερη χρονική απόσταση εντοπίζεται για μεγάλο μήκος μονοπατιού και για τους γράφους με 10.000 κόμβους. Τα αποτελέσματα αυτά οφείλονται στην ιδιαιτερότητα του ερωτήματος, για την απάντηση του οποίου χρειάζεται να περάσουμε από κάποιους κόμβους αρκετές φορές και άρα η προσθήκη μεγάλων κομματιών από μονοπάτια ή αρκετών κόμβων χωριστά σε κάθε επανάληψη επιβαρύνει αρκετά την απόδοση των αλγορίθμων μας.

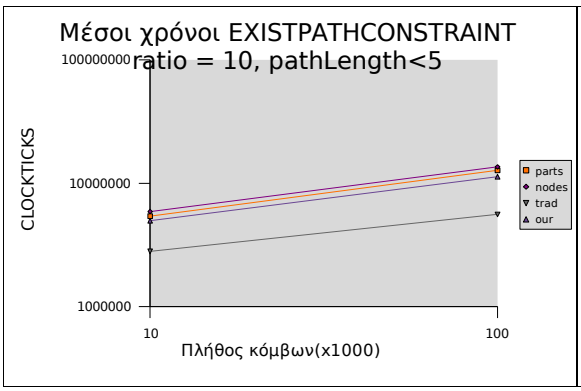
Τα αποτελέσματα για τα OXI-queries φαίνονται στα Διαγράμματα 6.61 ως 6.66, όπου έχουμε ratio = ακμές/κόμβοι. Στα διαγράμματα αυτά βλέπουμε πώς μεταβάλλεται ο μέσος χρόνος που χρειάζεται η κάθε μέθοδος, διατηρώντας σταθερό το μέγιστο μήκος μονοπατιού και το ratio και μεταβάλλοντας το πλήθος των κόμβων.



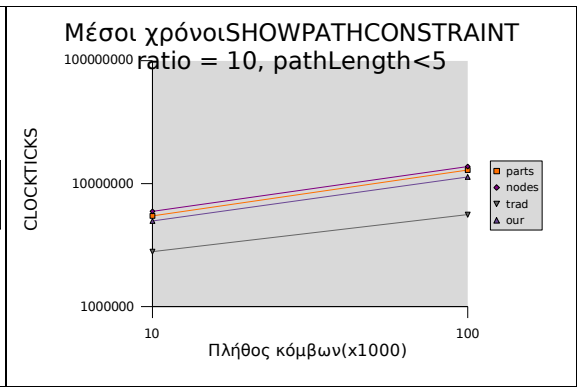
Διάγραμμα 6.61



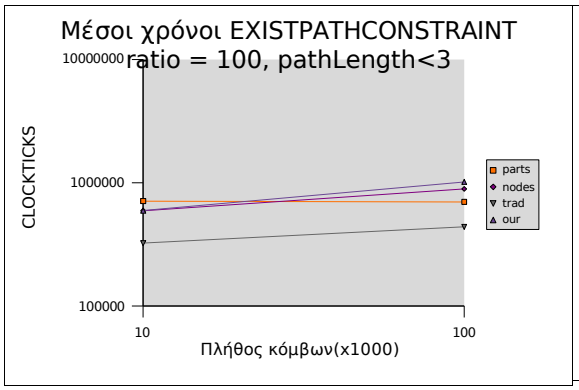
Διάγραμμα 6.62



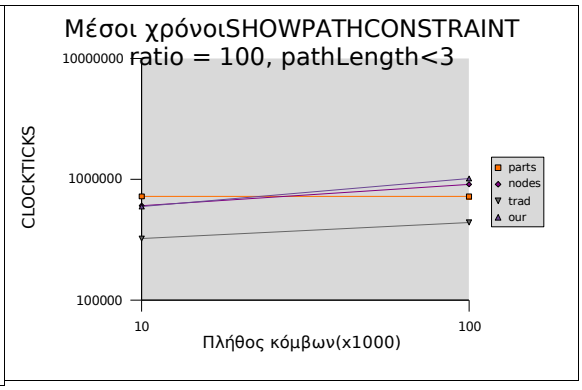
Διάγραμμα 6.63



Διάγραμμα 6.64



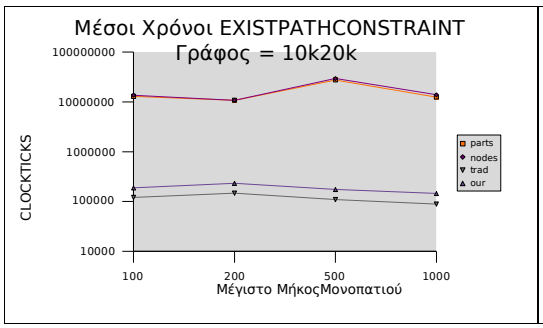
Διάγραμμα 6.65



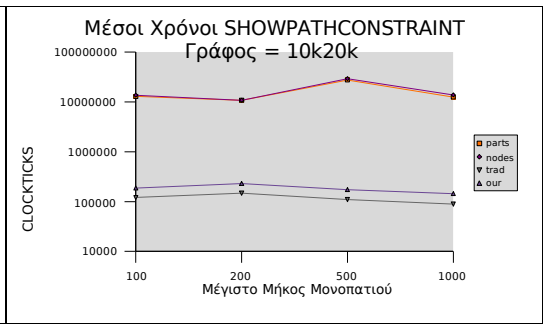
Διάγραμμα 6.66

Για τα OXI-queries βλέπουμε και πάλι ότι οι μέθοδοί μας πλησιάζουν την TRAD μέθοδο για τους πιο πυκνούς γράφους.

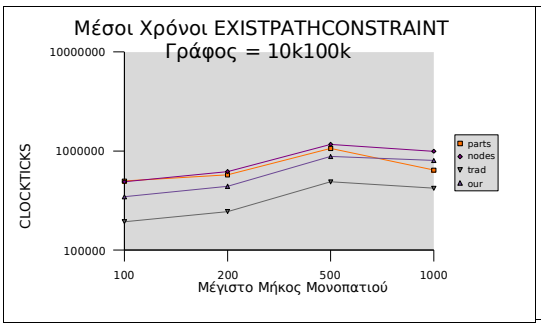
Στη συνέχεια, θέλουμε να μελετήσουμε τη συμπεριφορά των μεθόδων καθώς μεταβάλλεται το μέγιστο μήκος μονοπατιού. Αυτό το κάνουμε για τα NAI-queries. Τα αποτελέσματα φαίνονται στα Διαγράμματα 6.67 ως 6.78.



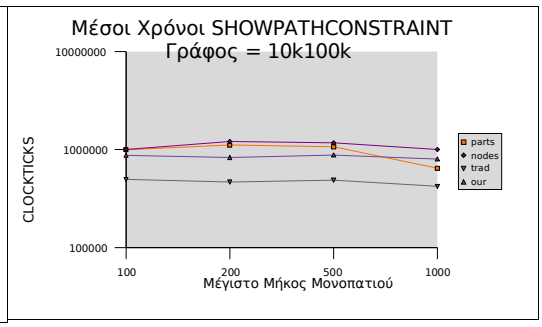
Διάγραμμα 6.67



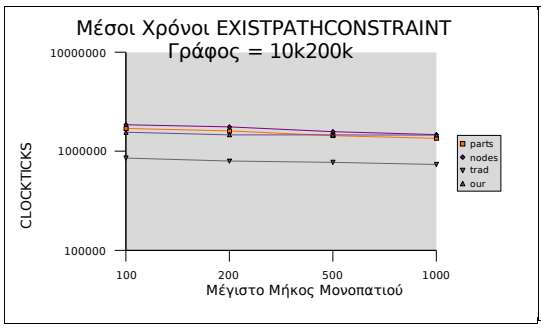
Διάγραμμα 6.68



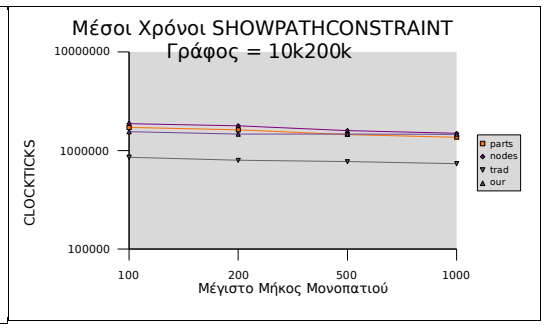
Διάγραμμα 6.69



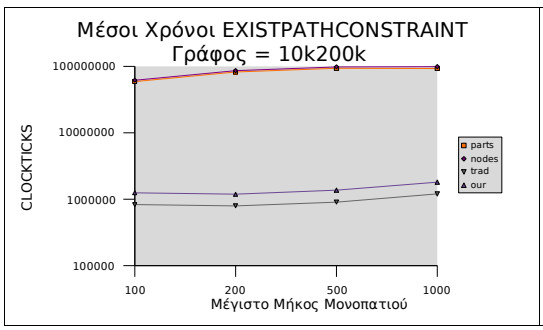
Διάγραμμα 6.70



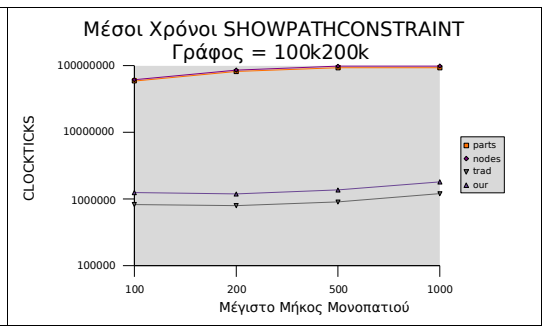
Διάγραμμα 6.71



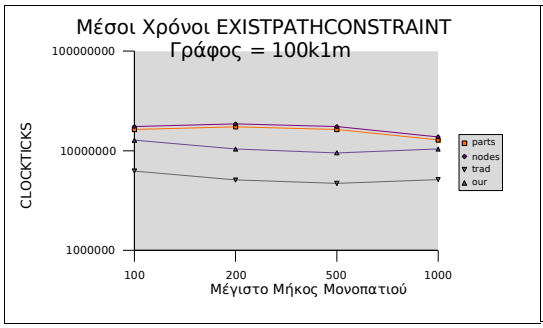
Διάγραμμα 6.72



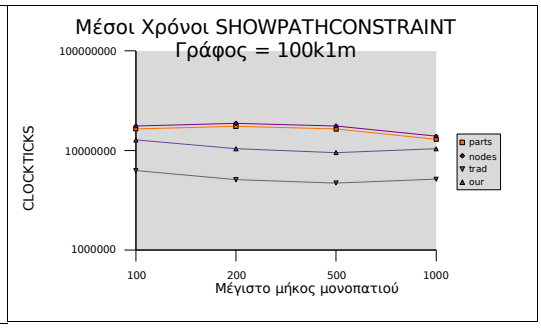
Διάγραμμα 6.73



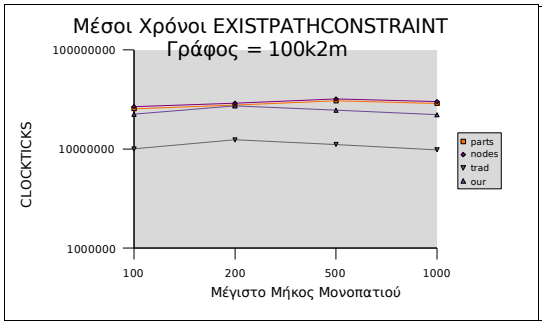
Διάγραμμα 6.74



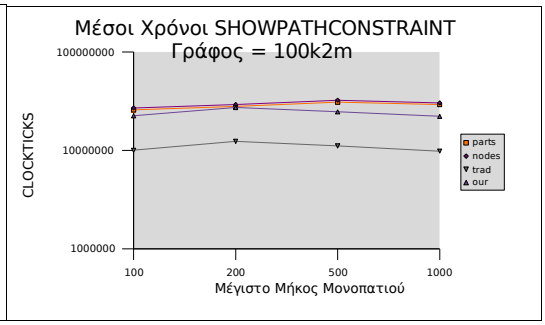
Διάγραμμα 6.75



Διάγραμμα 6.76



Διάγραμμα 6.77



Διάγραμμα 6.78

Οι μέθοδοί μας πλησιάζουν την TRAD για πυκνούς γράφους. Για μεγάλα μήκη, οι δικές μας μέθοδοι έχουν μια τάση να μειώνουν τον απαιτούμενο χρόνο, ενώ η TRAD τείνει να χρειάζεται περισσότερο χρόνο για να λύσει το πρόβλημα.

Το παραπάνω μπορεί να εξηγηθεί ως εξής:
Κάθε φορά που μελετάμε ένα κομμάτι μονοπατιού, εξετάζουμε πρώτα αν ισχύει:
τρέχον_μήκος_μονοπατιού + μήκος_κομματιού <= μέγιστο_μήκος_μονοπατιού
Αν δεν ισχύει κάτι τέτοιο, θα πρέπει να πάρουμε μόνο το μέρος του κομματιού που ικανοποιεί τον παραπάνω περιορισμό, ακόμα και αν ο κόμβος B βρίσκεται στο υπόλοιπο. Στις περιπτώσεις που το μέγιστο μήκος μονοπατιού είναι μικρό οι μέθοδοί μας χάνουν αρκετό χρόνο με τον παραπάνω έλεγχο σε σχέση με την TRAD. Αντίθετα, όταν ζητάμε το μέγιστο μήκος μονοπατιού να είναι μεγαλύτερο, βρίσκουμε πιο γρήγορα τη λύση, διότι ο περιορισμός είναι λιγότερο αυστηρός και μπορούμε να εξετάζουμε μεγαλύτερα κομμάτια μονοπατιών, στα οποία υπάρχει μεγαλύτερη πιθανότητα να βρεθεί ο κόμβος στόχος.

Βήμα 1.
Συγκρίναμε τις μεθόδους που προτείνουμε εμείς μεταξύ τους για να βρούμε την καλύτερη. Για τα NAI-queries δείχνουμε τα ποσοστά (%) που η NODES είναι καλύτερη (Πίνακας 6.17), αλλά και πόσο χάνει/κερδίζει σε σχέση με την άλλη μέθοδο, η οποία βάζει ολόκληρα κομμάτια μονοπατιών (Πίνακας 6.18). Εκτός από το συμβολισμό NkEk (N=κόμβοι, E=ακμές), στην πρώτη στήλη περιλαμβάνεται και ο περιορισμός που θέτουμε στο μήκος του μονοπατιού.

Από τους Πίνακες 6.17 και 6.18 καταλαβαίνουμε ότι η καλύτερη μέθοδος είναι η PARTS. Για αρκετά πυκνούς γράφους, ιδιαίτερα, δεν χάνει σχεδόν καθόλου, ενώ κερδίζει χρόνο που αποτελεί το 10% του χρόνου που χρειάζεται η ίδια για να απαντήσει στο κάθε query.

Για τα OXI-queries δείχνουμε τα ποσοστά (%) που η μέθοδος NODES είναι καλύτερη (Πίνακας 6.19), αλλά και πόσο χάνει/κερδίζει σε σχέση με τη PARTS (Πίνακας 6.20).

Για τα OXI-queries η κατάσταση είναι διαφορετική από τα NAI-queries. Εδώ. για πυκνούς γράφους η PARTS χάνει κατά μέσο όρο ένα 20% σε σχέση με τη NODES.

	ExistPath	ShowPath
10k20k100	0	0
10k20k200	4.69	4.69
10k20k500	8.2	8.2
10k20k1000	3.51	3.51
10k100k100	41.18	40
10k100k200	1.18	0
10k100k500	0	0
10k100k1000	5.95	5.95
10k200k100	0	0
10k200k200	0	0
10k200k500	0	0
10k200k1000	0	0
100k200k100	0	0
100k200k200	0	0
100k200k500	0	0
100k200k1000	0	0
100k1m100	0	0
100k1m200	0	0
100k1m500	0	0
100k1m1000	0	0
100k2m100	0	0
100k2m200	0	0
100k2m500	0	0
100k2m1000	0	0

Πίνακας 6.17

	Κερδίζει		Χάνει	
	ExistPath	ShowPath	ExistPath	ShowPath
10k20k100	0.0546	0.0545	0.0000	0.0000
10k20k200	0.0507	0.0507	1.0309	1.0309
10k20k500	0.2003	0.2003	1.4325	1.4325
10k20k1000	0.1078	0.1036	0.0129	0.0129
10k100k100	0.5522	0.0043	0.8690	0.4045
10k100k200	0.1711	0.0874	7.3214	0.0000
10k100k500	0.1000	0.0991	0.0000	0.0000
10k100k1000	0.6089	0.6083	0.1365	0.1368
10k200k100	0.0925	0.0920	0.0000	0.0000
10k200k200	0.0998	0.0989	0.0000	0.0000
10k200k500	0.0964	0.0954	0.0000	0.0000
10k200k1000	0.0927	0.0919	0.0000	0.0000
100k200k100	0.0464	0.0464	0.0000	0.0000
100k200k200	0.0464	0.0464	0.0000	0.0000
100k200k500	0.0464	0.0464	0.0000	0.0000
100k200k1000	0.0628	0.0628	0.0000	0.0000
100k1m100	0.0678	0.0674	0.0000	0.0000
100k1m200	0.0678	0.0674	0.0000	0.0000
100k1m500	0.0678	0.0674	0.0000	0.0000
100k1m1000	0.0678	0.0674	0.0000	0.0000
100k2m100	0.0498	0.0497	0.0000	0.0000
100k2m200	0.0498	0.0497	0.0000	0.0000
100k2m500	0.0498	0.0497	0.0000	0.0000
100k2m1000	0.0407	0.0403	0.0000	0.0000

Πίνακας 6.18[(διαφορά χρόνων)/(χρόνος καλύτερης μεθόδου)]

	ExistPath	ShowPath
10k20k10	11.76	11.76
10k100k5	0	0
10k200k3	76	76
100k200k10	7.89	7.89
100k1m5	0	0
100k2m3	10	14

Πίνακας 6.19

	Κερδίζει		Χάνει	
	ExistPath	ShowPath	ExistPath	ShowPath
10k20k10	0.09	0.04	0	0
10k100k5	0.13	0.13	0	0
10k200k3	0.12	0.13	0.25	0.25
100k200k10	0.08	0.08	0	0
100k1m5	0.07	0.07	0	0
100k2m3	0.32	0.26	0.11	0.11

Πίνακας 6.20[(διαφορά χρόνων)/(χρόνος καλύτερης μεθόδου)]

Βήμα 2.

Συγκρίναμε την κάθε μία από τις παραπάνω με την TRAD. Για τα NAI-queries τα ποσοστά (%) των ερωτημάτων στα οποία η TRAD είναι καλύτερη από τις δικές μας φαίνεται στον Πίνακα 6.21, ενώ το πόσο χάνει ή κερδίζει φαίνεται στον Πίνακα 6.22.

	PARTS		NODES	
	ExistPath	ShowPath	ExistPath	ShowPath
10k20k100	100	100	100	100
10k20k200	100	100	100	100
10k20k500	98.36	98.36	98.36	98.36
10k20k1000	96.49	96.49	96.49	96.49
10k100k100	76.47	76.47	74.12	75.29
10k100k200	80	80	80	81.18
10k100k500	74.12	75.29	76.47	76.47
10k100k1000	59.52	59.52	75	75
10k200k100	82.47	82.47	85.57	85.57
10k200k200	84.54	85.57	87.63	87.63
10k200k500	75.26	75.26	78.35	79.38
10k200k1000	74.23	75.26	77.32	78.35
100k200k100	97.3	97.3	97.3	97.3
100k200k200	100	100	100	100
100k200k500	100	100	100	100
100k200k1000	97.3	97.3	97.3	97.3
100k1m100	81	81	83	84
100k1m200	92	92	92	92
100k1m500	88	88	90	90
100k1m1000	86	86	88	88
100k2m100	78	80	80	82
100k2m200	80	80	80	82
100k2m500	92	92	92	92
100k2m1000	86	88	88	90

Πίνακας 6.21

Η TRAD είναι, όπως αναφέραμε και προηγουμένως, καλύτερη από τις δικές μας. Για αραιούς γράφους η διαφορά των μεθόδων είναι πολύ μεγάλη για τους πολύ αραιούς γράφους, ενώ οι δικές μας μέθοδοι πλησιάζουν πολύ την TRAD για τους υπόλοιπους. Πρέπει να επισημάνουμε το γεγονός ότι για τους γράφους με λόγο ακμές/κόμβοι = 10 ή 20, οι δικές μας μέθοδοι είναι καλύτερες από την TRAD σε ένα 20-40% των περιπτώσεων.

	PARTS-Κερδίζει		NODES-Κερδίζει		PARTS-Χάνει		NODES-Χάνει	
	ExistPath	ShowPath	ExistPath	ShowPath	ExistPath	ShowPath	ExistPath	ShowPath
10k20k100	105.64	105.64	111.46	111.46	0	0	0	0
10k20k200	72.08	72.05	72.07	72.05	0	0	0	0
10k20k500	253.23	252.98	270.12	269.85	0.49	0.49	0.36	0.36
10k20k1000	144.59	144.6	159.68	159.68	0.17	0.17	0.17	0.17
10k100k100	3.95	1.56	4.11	1.59	6.15	0.75	6.06	0.74
10k100k200	3.37	1.79	3.93	2.08	7.14	0.41	9.64	0.37
10k100k500	1.79	1.77	2	2.01	0.6	0.62	0.58	0.57
10k100k1000	1.4	1.41	2.01	2.02	0.74	0.74	0.6	0.59
10k200k100	1.34	1.36	1.49	1.51	0.65	0.64	0.73	0.71
10k200k200	1.28	1.3	1.46	1.48	0.45	0.44	0.46	0.44
10k200k500	1.35	1.38	1.51	1.53	0.63	0.62	0.67	0.67
10k200k1000	1.39	1.41	1.51	1.53	0.68	0.67	0.67	0.66
100k200k100	70.57	70.57	73.89	73.89	0	0	0	0
100k200k200	102.15	102.15	107.33	107.33	0	0	0	0
100k200k500	102.38	102.38	107.86	107.85	0	0	0	0
100k200k1000	78.61	78.6	83.61	83.6	0.26	0.26	0.25	0.25
100k1m100	2.12	2.14	2.26	2.28	0.72	0.7	0.59	0.57
100k1m200	2.67	2.69	2.92	2.95	0.58	0.57	0.51	0.5
100k1m500	2.8	2.82	3.06	3.09	0.44	0.43	0.45	0.44
100k1m1000	1.72	1.74	1.93	1.95	0.24	0.23	0.19	0.18
100k2m100	2.08	2.1	2.22	2.25	0.53	0.52	0.52	0.51
100k2m200	1.65	1.67	1.84	1.86	0.67	0.66	0.63	0.62
100k2m500	1.96	1.98	2.1	2.12	0.55	0.54	0.52	0.51
100k2m1000	2.34	2.37	2.41	2.44	0.51	0.49	0.49	0.48

Πίνακας 6.22[(διαφορά χρόνων)/(χρόνος καλύτερης μεθόδου)]

Για τα OXI-queries παραθέτουμε τα αποτελέσματα στους Πίνακες 6.23 και 6.24.

	PARTS		NODES	
	ExistPath	ShowPath	ExistPath	ShowPath
10k20k10	44.12	47.06	55.88	52.94
10k100k5	62.5	62.5	75	75
10k200k3	100	100	100	100
100k200k10	55.26	55.26	57.89	57.89
100k1m5	100	100	100	100
100k2m3	100	100	100	100

Πίνακας 6.23

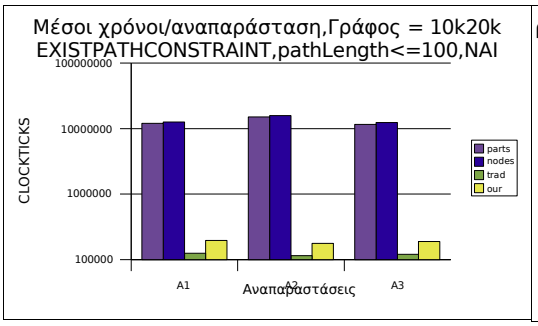
	PARTS-Κερδίζει		NODES-Κερδίζει		PARTS-Χάνει		NODES-Χάνει	
	ExistPath	ShowPath	ExistPath	ShowPath	ExistPath	ShowPath	ExistPath	ShowPath
10k20k10	16.38	14.45	13.59	14.34	0	0	0	0
10k100k5	1.62	1.65	1.58	1.61	0.22	0.22	0.33	0.33
10k200k3	1.19	1.23	0.83	0.87	0	0	0	0
100k200k10	23.34	23.37	23.35	23.38	0	0	0	0
100k1m5	1.27	1.31	1.43	1.46	0	0	0	0
100k2m3	0.59	0.64	1.03	1.07	0	0	0	0

Πίνακας 6.24[(διαφορά χρόνων)/(χρόνος καλύτερης μεθόδου)]

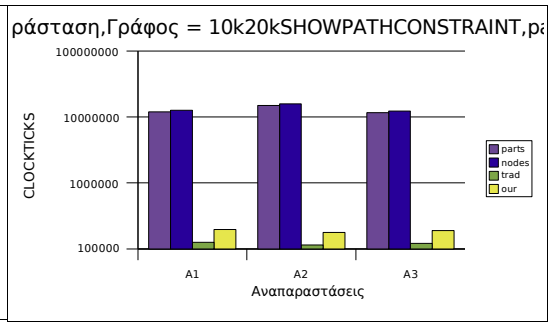
Για τα OXI-queries η μέθοδος TRAD είναι πολύ καλύτερη από τις δικές μας για πιο πυκνούς γράφους. Αυτό οφείλεται στο γεγονός ότι για τους πυκνούς γράφους απαιτούμε πολύ μικρό μήκος μονοπατιού, πράγμα που, όπως εξηγήσαμε και πιο πάνω, επιβαρύνει τις δικές μας μεθόδους.

Βήμα 3

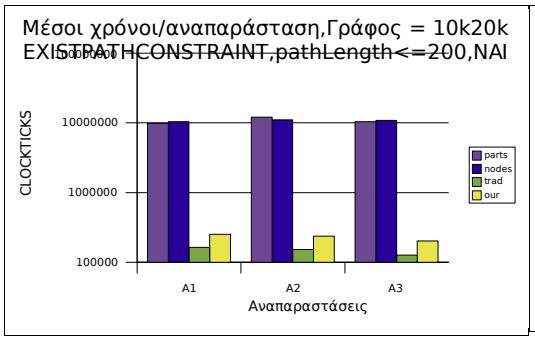
Κατασκευάσαμε ιστογράμματα που δείχνουν πώς συμπεριφέρονται οι μέθοδοι σε σχέση με τις διαφορετικές αναπαραστάσεις που έχουμε για τους γράφους. Για τα NAI-queries παραθέτουμε τα Διαγράμματα 6.79 ως 6.126.



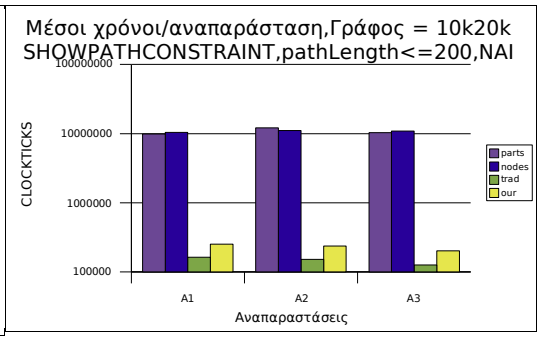
Διάγραμμα 6.79



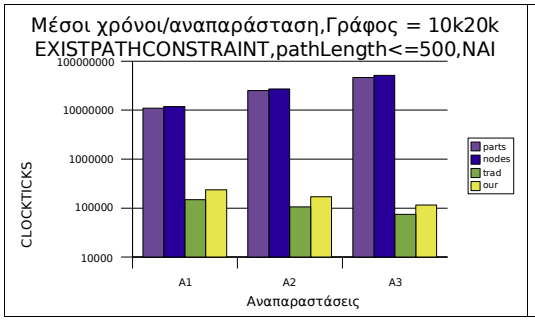
Διάγραμμα 6.80



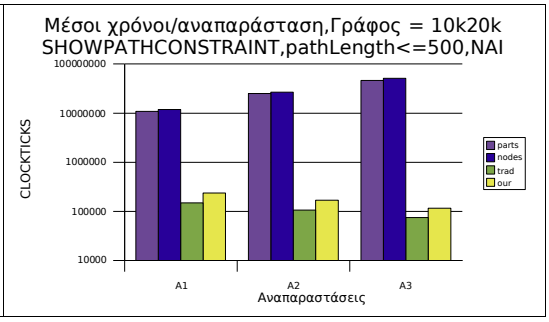
Διάγραμμα 6.81



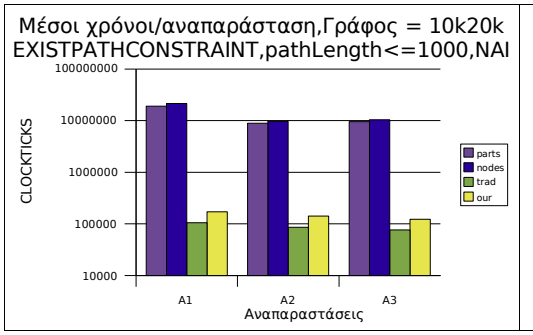
Διάγραμμα 6.82



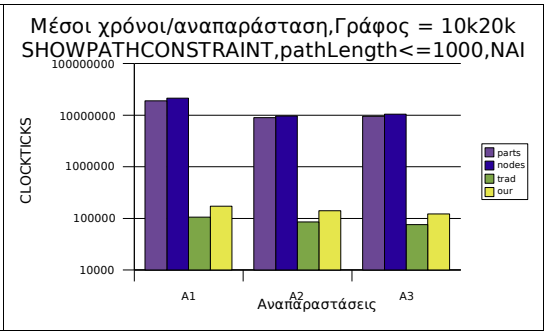
Διάγραμμα 6.83



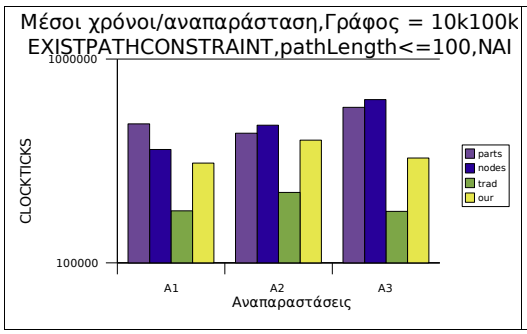
Διάγραμμα 6.84



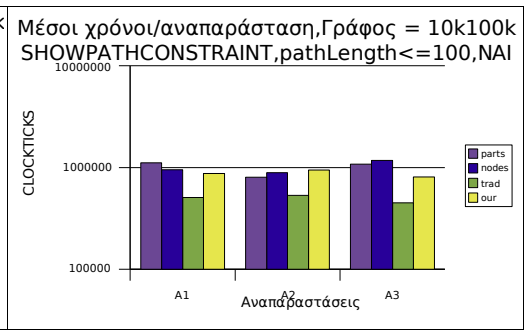
Διάγραμμα 6.85



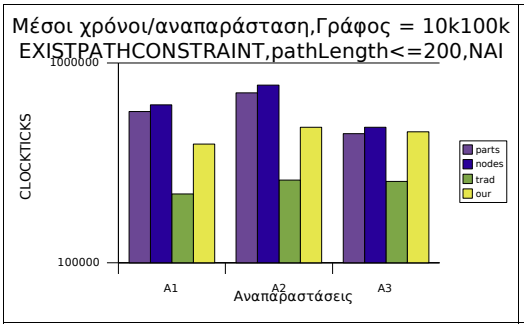
Διάγραμμα 6.86



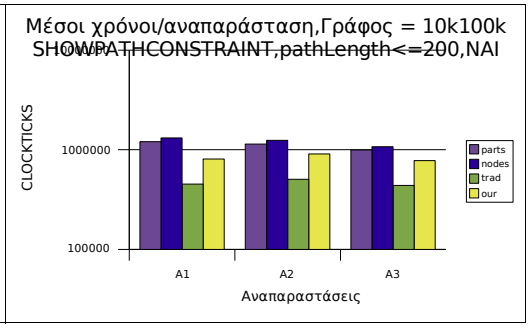
Διάγραμμα 6.87



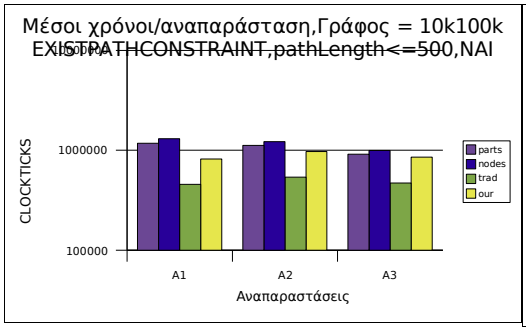
Διάγραμμα 6.88



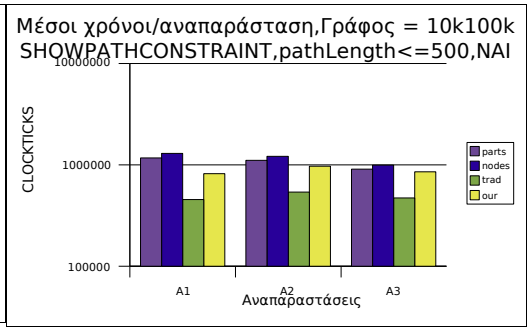
Διάγραμμα 6.89



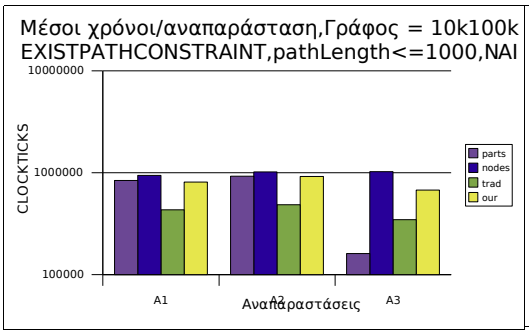
Διάγραμμα 6.90



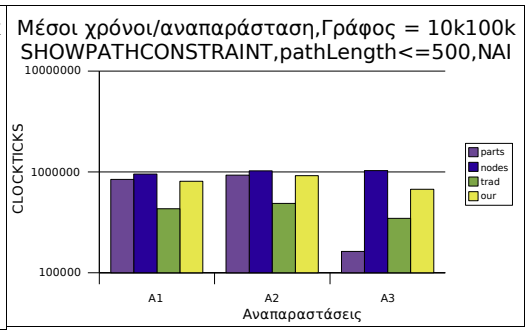
Διάγραμμα 6.91



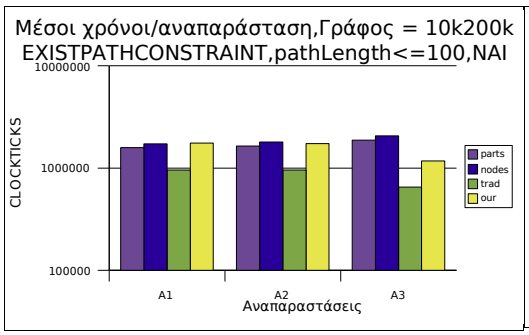
Διάγραμμα 6.92



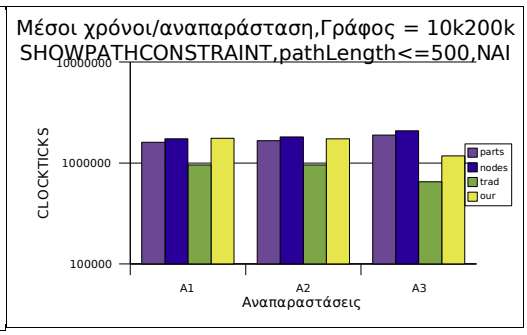
Διάγραμμα 6.93



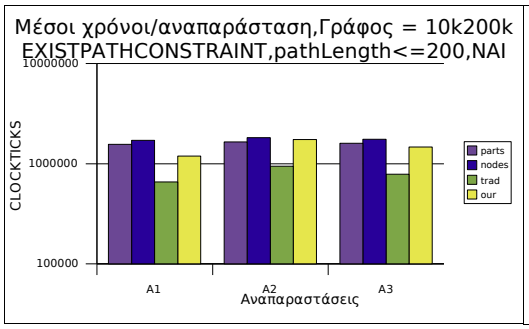
Διάγραμμα 6.94



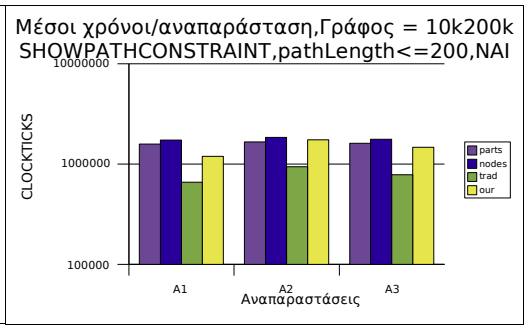
Διάγραμμα 6.95



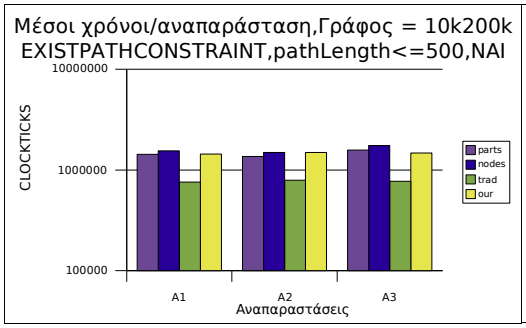
Διάγραμμα 6.96



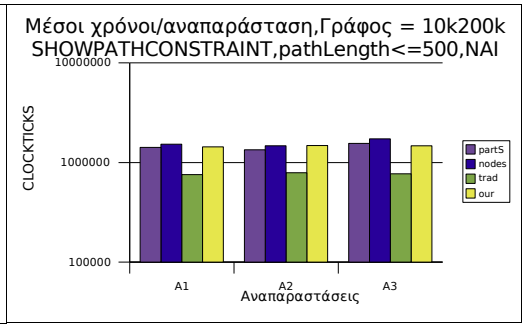
Διάγραμμα 6.97



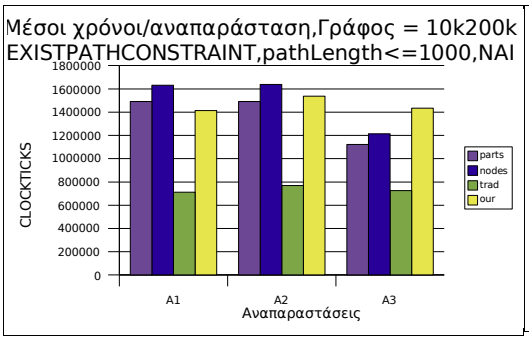
Διάγραμμα 6.98



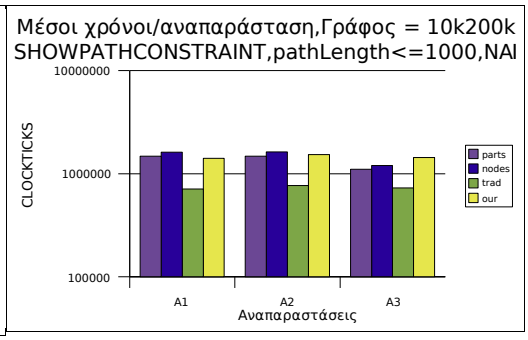
Διάγραμμα 6.99



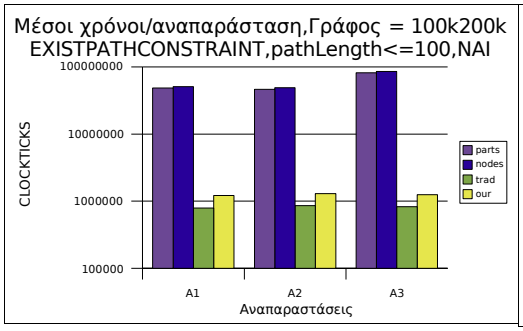
Διάγραμμα 6.100



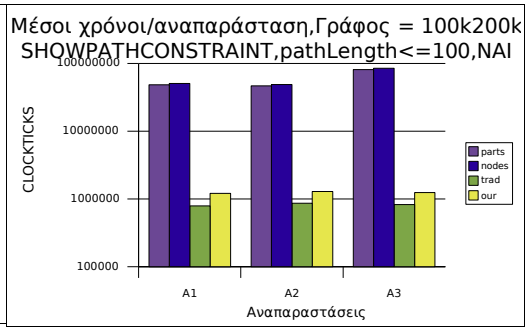
Διάγραμμα 6.103



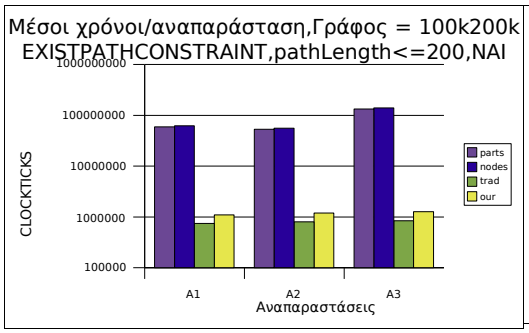
Διάγραμμα 6.102



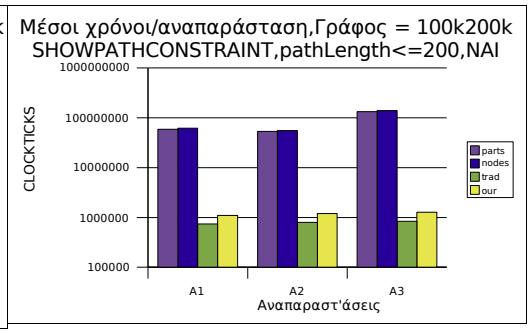
Διάγραμμα 6.103



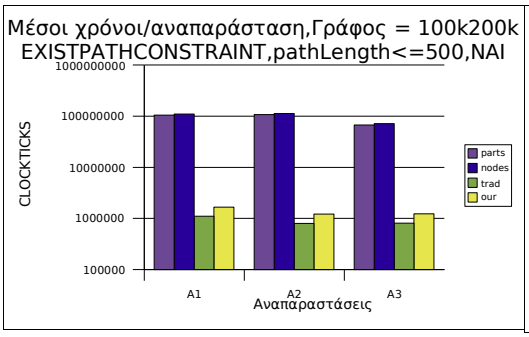
Διάγραμμα 6.104



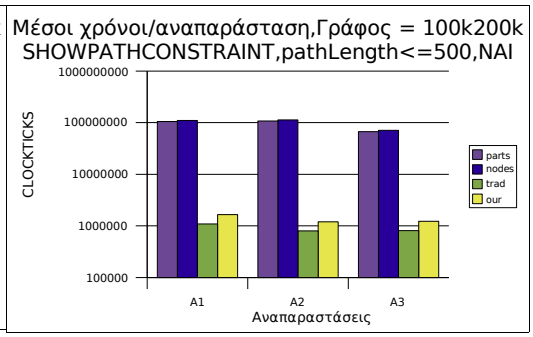
Διάγραμμα 6.105



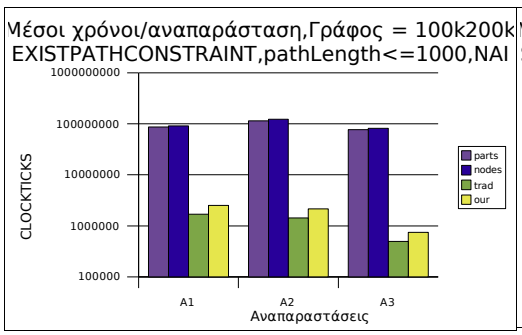
Διάγραμμα 6.106



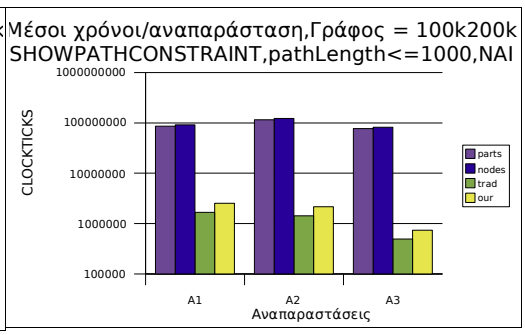
Διάγραμμα 6.107



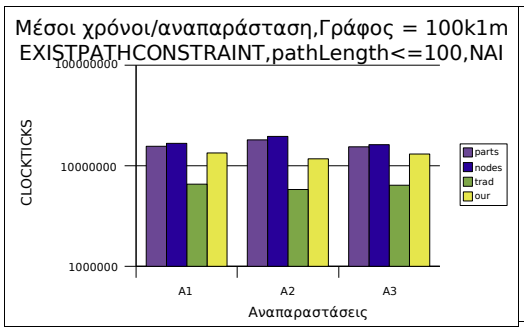
Διάγραμμα 6.108



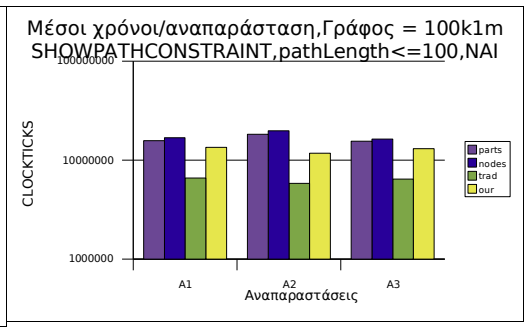
Διάγραμμα 6.109



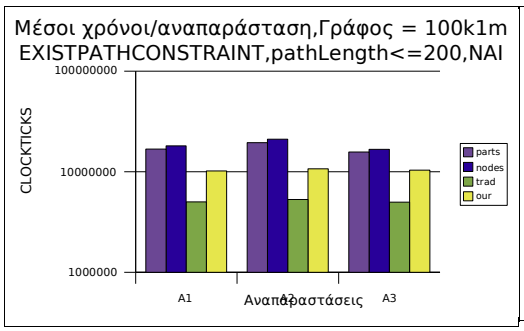
Διάγραμμα 6.110



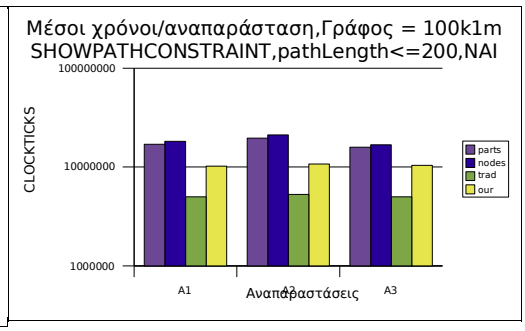
Διάγραμμα 6.111



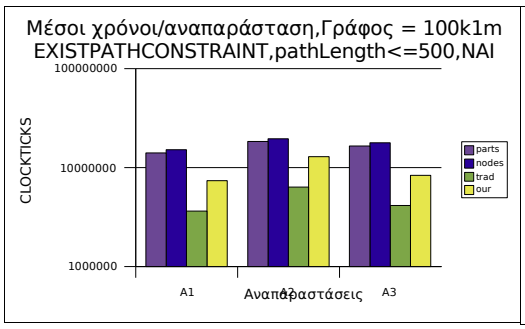
Διάγραμμα 6.112



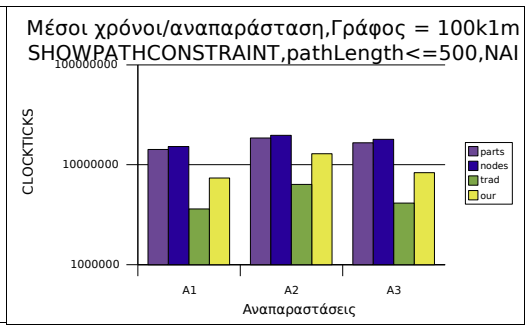
Διάγραμμα 6.113



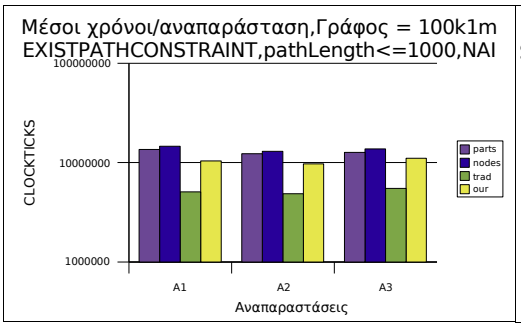
Διάγραμμα 6.114



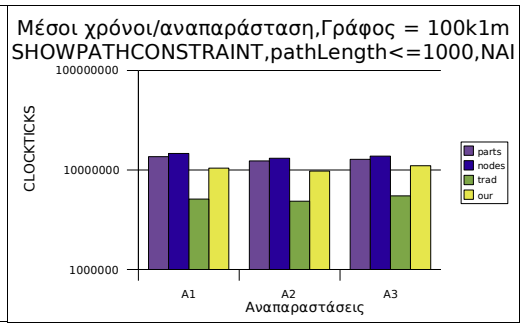
Διάγραμμα 6.115



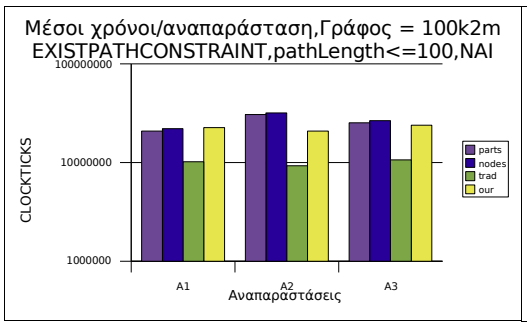
Διάγραμμα 6.116



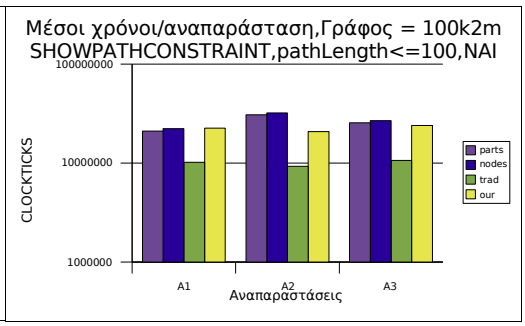
Διάγραμμα 6.117



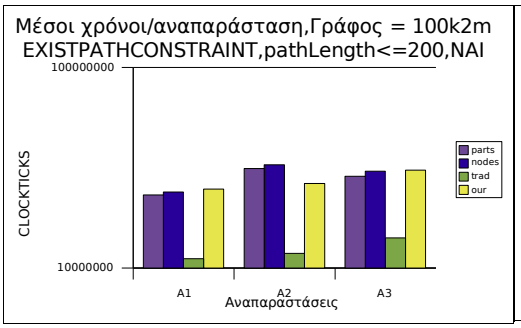
Διάγραμμα 6.118



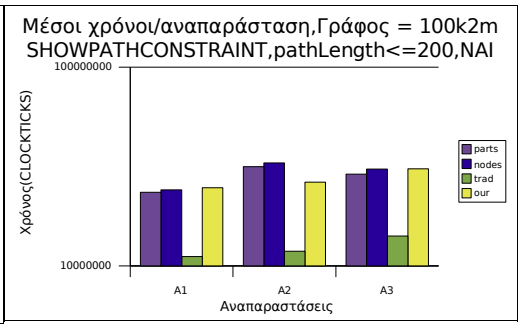
Διάγραμμα 6.119



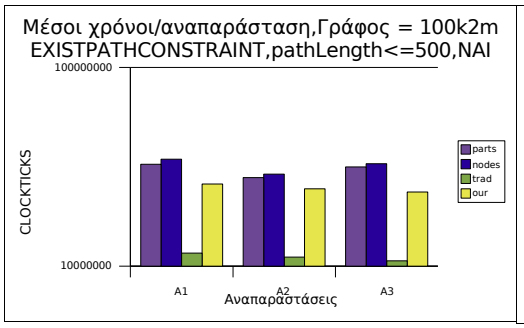
Διάγραμμα 6.120



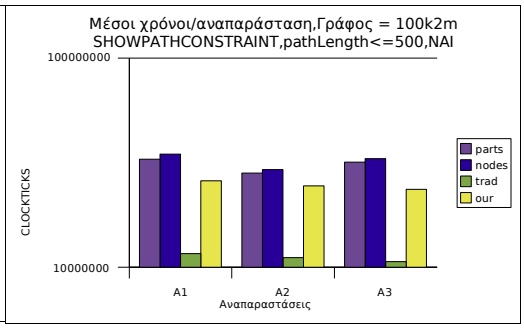
Διάγραμμα 6.121



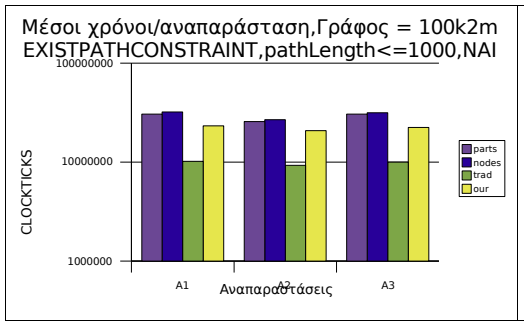
Διάγραμμα 6.122



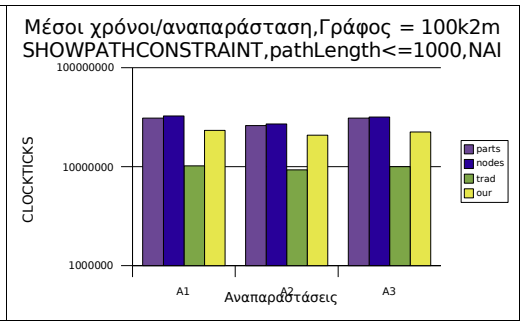
Διάγραμμα 6.125



Διάγραμμα 6.124

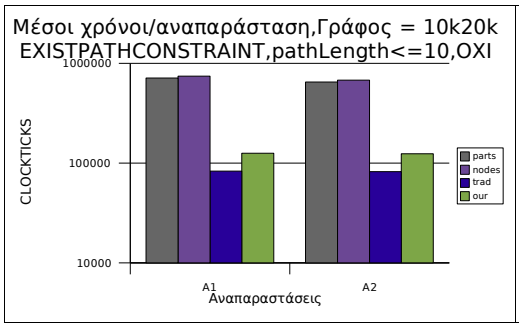


Διάγραμμα 6.125

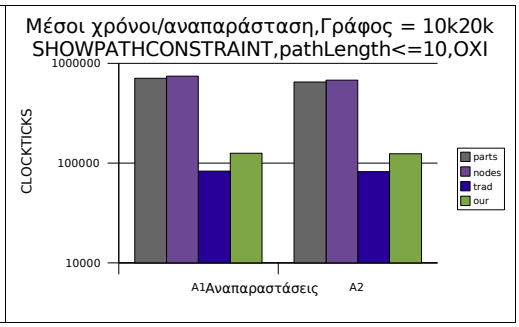


Διάγραμμα 6.126

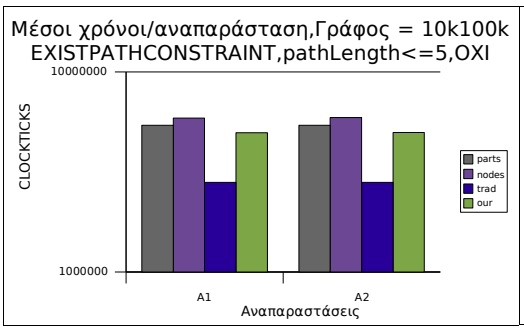
Για τα OXI-queries παραθέτουμε τα Διαγράμματα 6.127 ως 6.38.



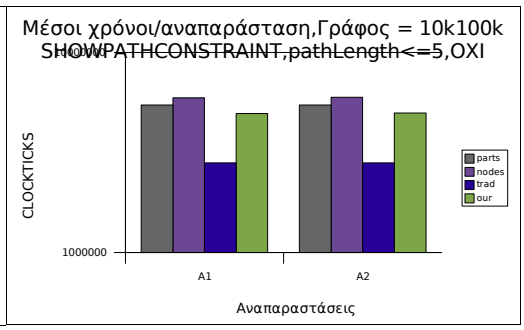
Διάγραμμα 6.127



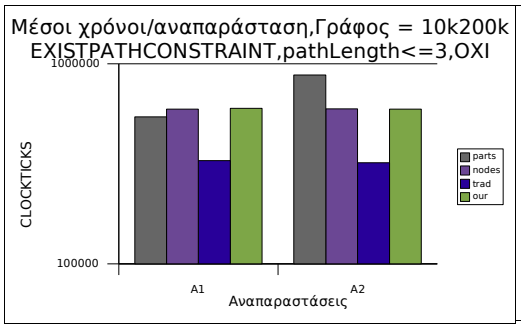
Διάγραμμα 6.128



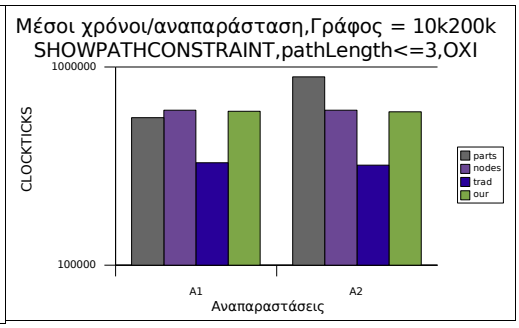
Διάγραμμα 6.129



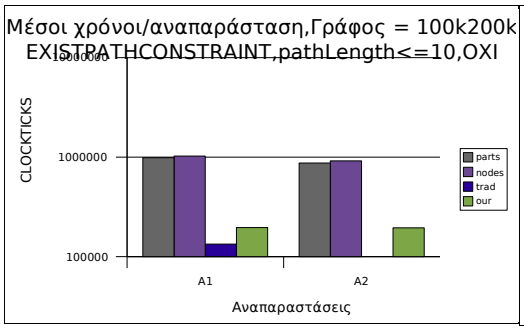
Διάγραμμα 6.130



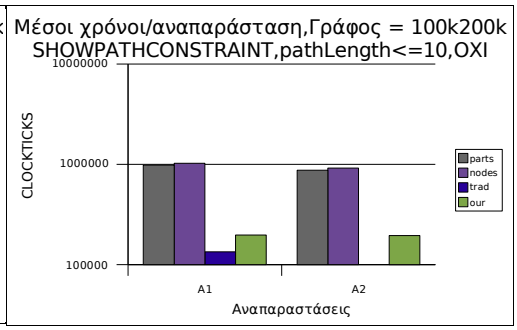
Διάγραμμα 6.131



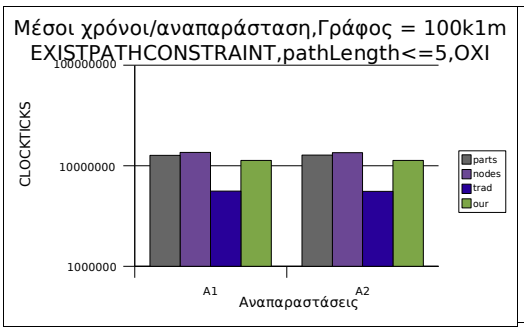
Διάγραμμα 6.132



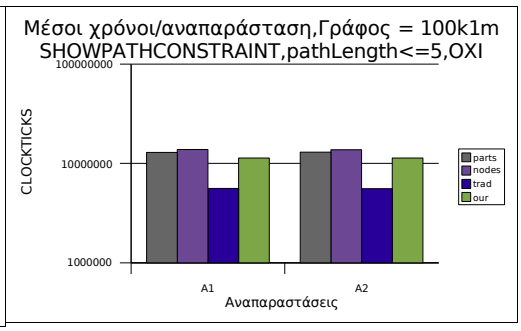
Διάγραμμα 6.133



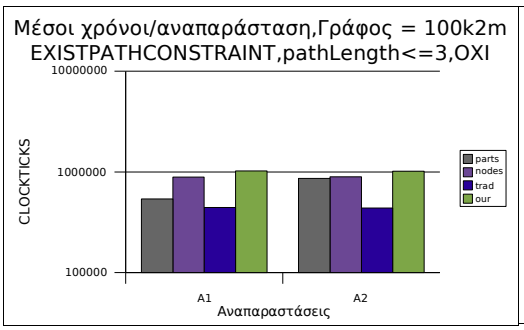
Διάγραμμα 6.134



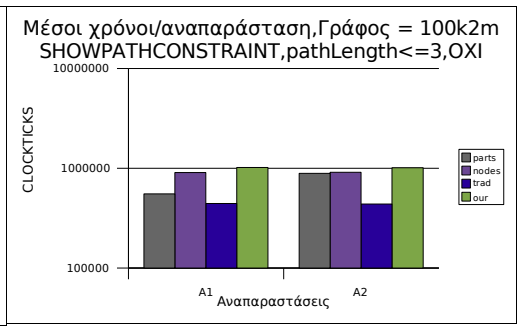
Διάγραμμα 6.135



Διάγραμμα 6.136



Διάγραμμα 6.137



Διάγραμμα 6.138

Τα ιστογράμματα 6.79 ως 6.138 μας οδηγούν στο συμπέρασμα ότι οι μέθοδοί μας εξαρτώνται πολύ από την αναπαράσταση του γράφου που δεχόμαστε ως είσοδο.

6.3 Σύνοψη συμπερασμάτων αξιολόγησης

Μετά την ολοκλήρωση των πειραμάτων και την παρουσίαση των αποτελεσμάτων τους, μπορούμε να συνοψίσουμε τα συμπεράσματά μας στα εξής:

- Για το ερώτημα εύρεσης μονοπατιού ανάμεσα σε δύο κόμβους Α και Β:
  - Σχεδόν όλες οι μέθοδοι που προτείνουμε είναι καλύτερες από τις TRAD μεθόδους. Αυτό είναι ιδιαίτερα εμφανές για πυκνούς γράφους.
  - Για τα NAI-queries ξεχωρίζουν σε απόδοση οι μέθοδοι που προσθέτουν ολόκληρα κομμάτια μονοπατιών στο μέτωπο αναζήτησης(MERGE), ενώ για τα OXI-queries είναι καλύτερες οι μέθοδοι που προσθέτουν χωριστά τους κόμβους(SEP).
  - Σε όλες σχεδόν τις περιπτώσεις είναι καλύτερες οι μέθοδοι που χρησιμοποιούν ουρά ως μέτωπο αναζήτησης, δηλαδή συμπεριφέρονται όπως η μέθοδος αναζήτησης κατά πλάτος (BFS).
  - Μια πιθανή βελτίωση των αλγορίθμων μας είναι να θυμόμαστε ποιοι κόμβοι οδηγούν σε λύση και ποιοι όχι, ώστε να αποφεύγουμε επανειλημμένες επισκέψεις σε κόμβους που δεν οδηγούν στον κόμβο Β.
- Για το ερώτημα εύρεσης μονοπατιού ανάμεσα σε δύο κόμβους Α και Β με περιορισμό ως

προς το μήκος του μονοπατιού:

- Οι μέθοδοι που χρησιμοποιούν αναπαραστάσεις με μονοπάτια είναι γενικά πιο αργές από αυτές που χρησιμοποιούν λίστες γειτνίασης. Αυτό είναι πολύ πιθανό να οφείλεται στους συχνούς ελέγχους που κάνουμε, καθώς και στο ότι αγνοούμε λύσεις, λόγω παραβίασης του περιορισμού, παρ' ότι καταλήγουμε σε αυτές νωρίτερα.
- Οι μέθοδοί μας πλησιάζουν πολύ τη μέθοδο με τις λίστες γειτνίασης για πυκνούς γράφους και μεγάλα μήκη μονοπατιών.
- Οι αναπαραστάσεις επηρεάζουν την απόδοση των αλγορίθμων που χρησιμοποιούν αναπαραστάσεις με μονοπάτια. Για το λόγο αυτό, θα ήταν πολύ χρήσιμο να βρούμε διάφορα χαρακτηριστικά των αναπαραστάσεων και να προσπαθήσουμε να κατασκευάσουμε αναπαραστάσεις που γενικά θα βοηθούν τους αλγορίθμους αυτούς.
- Μια πιθανή βελτίωση είναι να αποθηκεύουμε το αν ένας κόμβος μπορεί να οδηγήσει στον κόμβο B και έτσι να αποφεύγουμε επανειλημμένες επισκέψεις σε κόμβους που δεν οδηγούν σε αποτέλεσμα.
- Απάντηση στο ίδιο ερώτημα θα μπορούσαν να δώσουν και αλγόριθμοι που είναι παραλλαγές των αλγορίθμων του προηγούμενου ερωτήματος. Ωστόσο, οι αλγόριθμοι αυτοί δεν είναι τόσο ευέλικτοι όσο αυτοί που προτείνουμε εδώ, ως προς την οπισθοδρόμηση της αναζήτησης, δηλαδή δεν μπορούμε εύκολα να αλλάξουμε το μονοπάτι που οδηγεί στη λύση και συχνά προκύπτουν κύκλοι.

7

Τεχνικές λεπτομέρειες

Στα προηγούμενα κεφάλαια περιγράψαμε την υλοποίηση μεθόδων αποτίμησης ερωτημάτων εύρεσης μονοπατιών σε αναπαραστάσεις μονοπατιών γράφων. Στο κεφάλαιο αυτό παρουσιάζουμε τις τεχνικές λεπτομέρειες της υλοποίησης. Αρχικά, αναφέρονται με λίγα λόγια τα προγραμματιστικά εργαλεία που χρησιμοποιήθηκαν. Στη συνέχεια παρουσιάζουμε τις κλάσεις και τις δομές που υλοποιήσαμε.

7.1 Πλατφόρμες και προγραμματιστικά εργαλεία

Για την υλοποίηση των μεθόδων της εργασίας χρησιμοποιήθηκε η γλώσσα προγραμματισμού C++. Ως περιβάλλον εργασίας και προγραμματισμού χρησιμοποιήθηκε το Eclipse Europa σε λειτουργικό Linux (διανομή Ubuntu 7.04). Ο υπολογιστής είχε μνήμη 2GB και Intel Core2Duo επεξεργαστή στα 2GHz. Οι μετρήσεις έγιναν εκτός του Eclipse και παραστάθηκαν με το λογιστικό φύλλο του OpenOffice. Για τη σχεδίαση των γράφων χρησιμοποιήθηκε το πρόγραμμα Graphviz.

7.2 Λεπτομέρειες υλοποίησης

Πριν ξεκινήσουμε, παρουσιάζουμε τα ονόματα των ερωτημάτων στα οποία απαντήσαμε στον Πίνακα 7.1. Θα χρησιμοποιήσουμε τα ονόματα αυτά για να αναφερόμαστε στα αντίστοιχα ερωτήματα.

Ερώτημα που απαντάμε	Όνομα
Εύρεση παιδιών ενός κόμβου A	getNext
Εύρεση μονοπατιού ανάμεσα στους κόμβους A και B	existPath
Εύρεση μονοπατιού ανάμεσα στους κόμβους A και B υπό συνθήκη	existPathConstraint

Πίνακας 7.1

7.2.1 Δομές της C++ που χρησιμοποιήσαμε

Για την υλοποίηση χρησιμοποιήσαμε και τις παρακάτω δομές της C++ και τις συναρτήσεις που αυτές παρέχουν.

- map : Ταξινομημένο ευρετήριο που περιέχει μοναδικά ζευγάρια κλειδί-τιμή.
- stack : Υλοποίηση της στοίβας.
- vector : Υλοποίηση διανύσματος – συνόλου από στοιχεία.
- queue : Υλοποίηση της ουράς.
- deque : Χρησιμοποιείται ως ουρά όταν βάζουμε κομμάτια μονοπατιών, διότι παρέχει τη συνάρτηση insert.

7.2.2 Κλάση Graph

Η κλάση αυτή υλοποιεί το γράφο που αποτελεί την είσοδο για τους αλγορίθμους που υλοποιήσαμε. Ο γράφος είναι ένα σύνολο μονοπατιών, όπως ακριβώς και στο αρχείο από το οποίο τον διαβάζουμε.

Οι βασικές μεταβλητές που περιέχονται στην κλάση Graph είναι:

- vector<Path> g : Είναι το σύνολο των μονοπατιών που αποτελούν το γράφο.
- int graphSize : Περιέχει το μέγεθος σε bytes του γράφου.

Στην κλάση αυτή έχουμε και διάφορες άλλες δομές, οι οποίες κρατούν πληροφορίες για το αν ένας κόμβος ή ένα κομμάτι μονοπατιού έχει εξερευνηθεί ή όχι.

Οι συναρτήσεις της κλάσης Graph είναι οι εξής:

- Graph(ifstream& fileName): Κατασκευάζει το γράφο από ένα αρχείο. Ο αντίστοιχος ψευδοκώδικας είναι:

```
αριθμός_μονοπατιού = 0

όσο υπάρχουν μονοπάτια στο αρχείο που δεν έχουμε διαβάσει
{
      χωρίζουμε τη γραμμή που περιέχει το μονοπάτι με βάση τα
            κενά (" ");//χρήση της συνάρτησης strtok(NULL, " ")
      θέση_κόμβου = 0;
      όσο δεν έχουμε τελειώσει με τα κομμάτια της γραμμής
      {
            addToPath(κομμάτι, αριθμός_μονοπατιού, θέση_κόμβου);
            παίρνουμε το επόμενο κομμάτι;
```

```
            θέση_κόμβου++;
        }
        προσθέτουμε το μονοπάτι στο g του γράφου;
        graphSize += μήκος μονοπατιού*sizeof(pathAux);
        αριθμός_μονοπατιού++;
    }
```

Παράδειγμα 7.1. Έστω ότι το αρχείο που δίνεται είναι το παρακάτω:

1 12 2 10 3 4

2 4 5 9 3

5 8

1 14 15 6 16 17

1 6 7 8

1 3

9 11 8

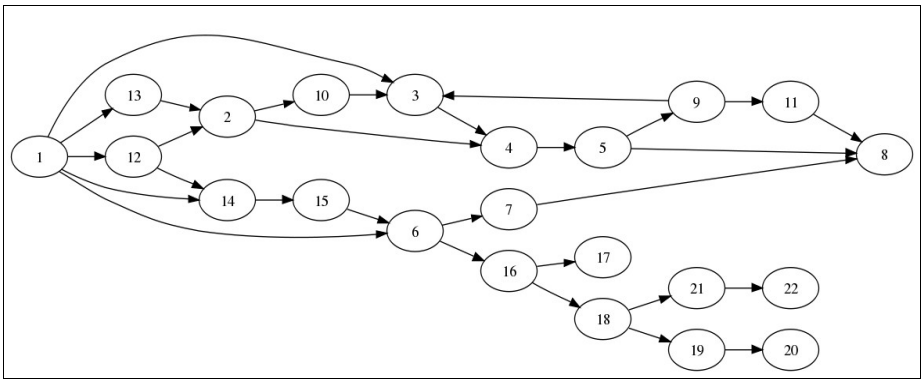
1 13 2

12 14

16 18 19 20

18 21 22

Ο γράφος είναι αυτός που δίνεται στο Σχήμα 7.1.

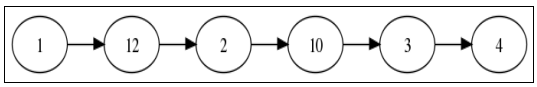


Σχήμα 7.1

Διαβάζουμε γραμμή – γραμμή το αρχείο. Ξεκινάμε από την πρώτη γραμμή:

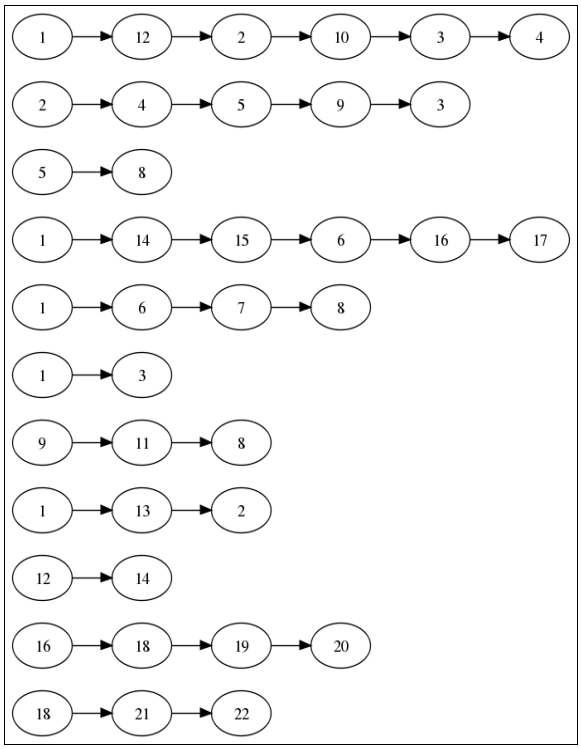
1 12 2 10 3 4

Τα στοιχεία που περιέχονται στη γραμμή είναι τα 1,12,2,10,3 και 4. Φτιάχνουμε λοιπόν το πρώτο μονοπάτι, το οποίο προσθέτουμε στο vector του γράφου. Το μονοπάτι αυτό φαίνεται στο Σχήμα 7.2.



Σχήμα 7.2

Συνεχίζουμε με τα υπόλοιπα μονοπάτια μέχρι να φτάσουμε στο τέλος του αρχείου. Στο τέλος, γράφος θα είναι όπως στο Σχήμα 7.3.



Σχήμα 7.3

- void addToGraph(Path newPath): Προσθέτει ένα μονοπάτι στο γράφο. Αυτό γίνεται με προσθήκη του newPath στο vector g του γράφου.

- void printGraph(): Εκτυπώνει το γράφο, είτε στην οθόνη, είτε σε κάποιο αρχείο. Για κάθε στοιχείο του vector g καλεί την printPath η οποία τυπώνει το αντίστοιχο μονοπάτι.

- vector<Node> getNext(int Node, PathIndex& pI): Υλοποιεί το getNext, βρίσκει, δηλαδή τα παιδιά ενός κόμβου με χρήση πληροφορίας από τη δομή PATHINDEX. Αυτό γίνεται με την κλήση της συνάρτησης getNextInPath της κλάσης Path.

- int existPath_bfs_sep(int NodeA, int NodeB, PathIndex& pI): Βρίσκει αν υπάρχει μονοπάτι ανάμεσα στους κόμβους A και B, απαντάει δηλαδή στο existPath. Ανάλογα με το τί θέλουμε, μπορεί να εκτυπώνει και το μονοπάτι. Η συνάρτηση αυτή υλοποιεί την αναζήτηση με χρήση του PATHINDEX και βάζει χωριστά τους κόμβους στην ουρά.

- int existPath_bfs_merge(int NodeA, int NodeB, PathIndex& pI): Βρίσκει αν υπάρχει μονοπάτι ανάμεσα στους κόμβους A και B, απαντάει δηλαδή στο existPath. Ανάλογα με το τί θέλουμε, μπορεί να εκτυπώνει και το μονοπάτι. Η συνάρτηση αυτή υλοποιεί την αναζήτηση με χρήση του PATHINDEX και βάζει ολόκληρα κομμάτια μονοπατιών στην ουρά.

- int existPath_dfs_sep(int NodeA, int NodeB, PathIndex& pI): Βρίσκει αν υπάρχει μονοπάτι ανάμεσα στους κόμβους Α και Β, απαντάει δηλαδή στο existPath. Ανάλογα με το τί θέλουμε, μπορεί να εκτυπώνει και το μονοπάτι. Η συνάρτηση αυτή υλοποιεί την αναζήτηση με χρήση του PATHINDEX και βάζει χωριστά τους κόμβους στη στοίβα.

- int existPath_dfs_merge(int NodeA, int NodeB, PathIndex& pI): Βρίσκει αν υπάρχει μονοπάτι ανάμεσα στους κόμβους Α και Β, απαντάει δηλαδή στο existPath. Ανάλογα με το τί θέλουμε, μπορεί να εκτυπώνει και το μονοπάτι. Η συνάρτηση αυτή υλοποιεί την αναζήτηση με χρήση του PATHINDEX και βάζει ολόκληρα κομμάτια μονοπατιών στη στοίβα.

- int existPath_trad_dfs(int NodeA, int NodeB, map<int, vector<Node> >& neighbours): Βρίσκει αν υπάρχει μονοπάτι ανάμεσα στους κόμβους Α και Β, απαντάει δηλαδή στο existPath. Ανάλογα με το τί θέλουμε, μπορεί να εκτυπώνει και το μονοπάτι. Η συνάρτηση αυτή υλοποιεί την αναζήτηση με χρήση των λιστών γειτνίασης και τοποθετεί τους κόμβους σε στοίβα.

- int existPath_trad_bfs(int NodeA, int NodeB, map<int, vector<Node> >& neighbours): Βρίσκει αν υπάρχει μονοπάτι ανάμεσα στους κόμβους Α και Β, απαντάει δηλαδή στο existPath. Ανάλογα με το τί θέλουμε, μπορεί να εκτυπώνει και το μονοπάτι. Η συνάρτηση αυτή υλοποιεί την αναζήτηση με χρήση των λιστών γειτνίασης και τοποθετεί τους κόμβους σε ουρά.

- int existPath_trad_constraint(int NodeA, int NodeB, map<int, vector<Node> >& neighbours, int constraint): Βρίσκει αν υπάρχει μονοπάτι ανάμεσα στους κόμβους Α και Β που να ικανοποιεί κάποιο περιορισμό, απαντάει δηλαδή στο existPathConstraint. Ανάλογα με το τί θέλουμε, μπορεί να εκτυπώνει και το μονοπάτι. Η συνάρτηση αυτή υλοποιεί την αναζήτηση με χρήση των λιστών γειτνίασης.

- int existPath_sep_constraint(int NodeA, int NodeB, PathIndex& pI, int constraint): Βρίσκει αν υπάρχει μονοπάτι ανάμεσα στους κόμβους Α και Β που να ικανοποιεί κάποιο περιορισμό, απαντάει δηλαδή στο existPathConstraint. Ανάλογα με το τί θέλουμε, μπορεί να εκτυπώνει και το μονοπάτι. Η συνάρτηση αυτή υλοποιεί την αναζήτηση με χρήση του PATHINDEX και τοποθετεί τους κόμβους χωριστά στο μέτωπο αναζήτησης.

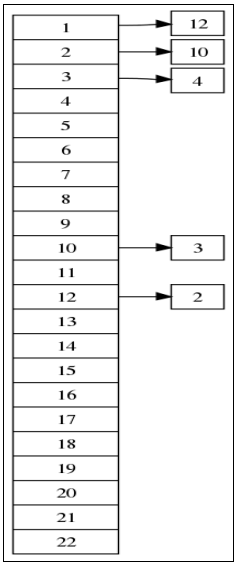
- int existPath_merge_constraint(int NodeA, int NodeB, PathIndex& pI, int constraint): Βρίσκει αν υπάρχει μονοπάτι ανάμεσα στους κόμβους Α και Β που να ικανοποιεί κάποιο περιορισμό, απαντάει δηλαδή στο existPathConstraint. Ανάλογα με το τί θέλουμε, μπορεί να εκτυπώνει και το μονοπάτι. Η συνάρτηση αυτή υλοποιεί την αναζήτηση με χρήση του PATHINDEX και τοποθετεί ολόκληρα κομμάτια μονοπατιών στο μέτωπο αναζήτησης.

- map<int, vector<Node> > getAdjList(): Κατασκευάζει το σύνολο των λιστών γειτνίασης.

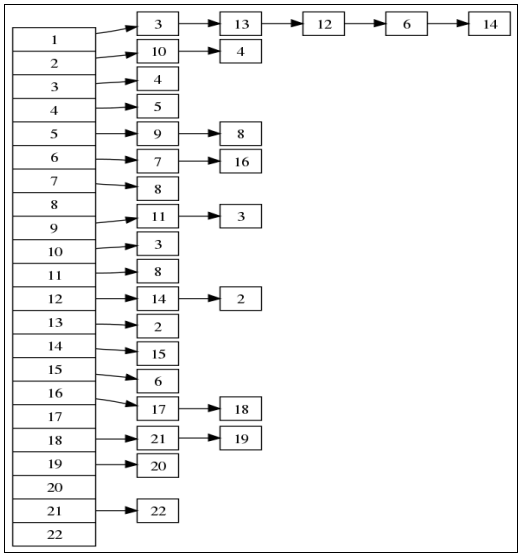
```
για κάθε στοιχείο του g

{

  aPath = g.at(i); //μονοπάτι στη θέση i του g

  για κάθε στοιχείο του aPath

  {

        προσθέτω στο αποτέλεσμα, στην εγγραφή που αντιστοιχεί στο
        συγκεκριμένο στοιχείο, μια αναφορά στον επόμενο κόμβο του
        μονοπατιού;

        αυξάνω το μέγεθος του αποτελέσματος;

  }

}

επιστρέφω το αποτέλεσμα;
```

Παράδειγμα 7.2. Έχουμε το γράφο του Σχήματος 7.3. Ξεκινώντας από το πρώτο μονοπάτι, δηλαδή αυτό του Σχήματος 7.2, βάζουμε για κάθε κόμβο του τον επόμενο στην αντίστοιχη λίστα γειτνίασης. Για το πρώτο μονοπάτι έχουμε το αποτέλεσμα που φαίνεται στο Σχήμα 7.4. Συνεχίζοντας, καταλήγουμε στο αποτέλεσμα που φαίνεται στο Σχήμα 7.5.

Πρέπει να σημειώσουμε ότι τόσο οι λίστες γειτνίασης όσο και οι λίστες στο PATHINDEX είναι ταξινομημένες σε αύξουσα σειρά.



Σχήμα 7.4



Σχήμα 7.5

7.2.3 Κλάση PathIndex

Η κλάση αυτή υλοποιεί τη δομή PATHINDEX. Οι βασικές μεταβλητές που περιέχει είναι οι παρακάτω:

- map<int, vector<nodePathPair> > pIndex: Είναι ένα ευρετήριο που για κάθε κόμβο περιέχει μια λίστα με όλα τα μονοπάτια στα οποία συμμετέχει μαζί με την απαραίτητη πληροφορία.
- int pathIndexSize: Είναι το μέγεθος σε bytes του PATHINDEX.

Πολύ σημαντική είναι και η structure:

- struct nodePathPair: Είναι το δομικό στοιχείο κάθε εγγραφής του PATHINDEX. Τα πεδία της είναι:
  - int pathId: Το αναγνωριστικό του μονοπατιού στο οποίο ανήκει ο κόμβος
  - int positionInPath: Η θέση του κόμβου στο μονοπάτι pathId
  - int predecessor: Ο κόμβος που προηγείται στο μονοπάτι pathId.

Οι συναρτήσεις της κλάσης PathIndex είναι οι εξής:

- PathIndex(Graph& graph): Κατασκευάζει τον PATHINDEX από ένα δοσμένο γράφο.

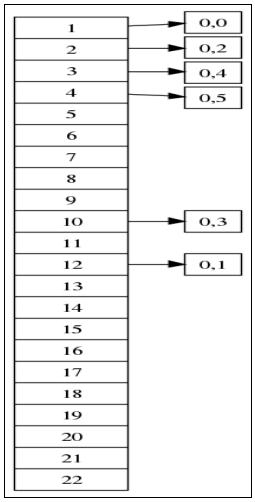
  Ο ψευδοκώδικας είναι ο εξής:

```
κάνουμε τις απαραίτητες αρχικοποιήσεις;
```

```
για κάθε στοιχείο του vector g
{
    aPath = το στοιχείο του g;
    για κάθε στοιχείο του aPath
    {
      προσθέτω στην αντίστοιχη εγγραφή του pathIndex ένα στοιχείο
      nodePathPair με τις απαραίτητες πληροφορίες;
    }
}
```

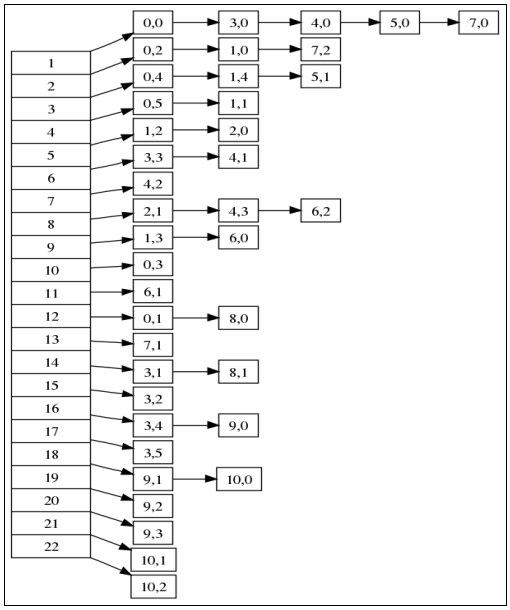
Παράδειγμα 7.3.

Έχουμε και πάλι το γράφο του Σχήματος 7.3. Ξεκινώντας από το πρώτο μονοπάτι, δηλαδή αυτό του Σχήματος 7.2, βάζουμε για κάθε κόμβο του ένα στοιχείο στην αντίστοιχη λίστα. Για το πρώτο μονοπάτι έχουμε το αποτέλεσμα που φαίνεται στο Σχήμα 7.6.



Σχήμα 7.6

Συνεχίζοντας, καταλήγουμε στο αποτέλεσμα που φαίνεται στο Σχήμα 7.7.



Σχήμα 5.7

- void printPathIndex(): Εκτυπώνει τον PATHINDEX, είτε στην οθόνη, είτε σε κάποιο αρχείο. Για κάθε εγγραφή του PATHINDEX τυπώνει ένα ένα τα στοιχεία της στην οθόνη ή σε αρχείο.

7.2.4 Κλάση Path

Η κλάση αυτή υλοποιεί ένα μονοπάτι. Η βασική μεταβλητή της κλάσης αυτής είναι:

- vector<pathAux> p: Μονοπάτι, υλοποιημένο ως vector από «κόμβους».
- struct pathAux : περιέχει πληροφορίες για τον κόμβο ενός μονοπατιού.
- int nodeId: όνομα του κόμβου.
    - int pathId: όνομα του μονοπατιού στο οποίο ανήκει.
    - int position: θέση στο μονοπάτι.

Οι συναρτήσεις της κλάσης Path είναι οι εξής:

- void addToPath(int newNode): Προσθέτει ένα κόμβο στο μονοπάτι.
- void printPath(int id): Εκτυπώνει ένα μονοπάτι, είτε στην οθόνη, είτε σε αρχείο.
- Node getNextInPath(struct nodePathPair npp, int node): Βρίσκει τον επόμενο ενός κόμβου σε ένα συγκεκριμένο μονοπάτι.

  Ο αντίστοιχος κώδικας είναι:

```
Node next;//δημιουργία επόμενου κόμβου
int newPosition = npp.positionInPath + 1;
                                        //βρίσκω την επόμενη θέση
next.name = p.at(newPosition).nodeId;
                                  //βρίσκω το όνομα του επόμενου
next.position = newPosition;
return next;
```

Παράδειγμα 7.4.

Έστω ότι για το γράφο του Σχήματος 7.3 θέλουμε να βρούμε τον επόμενο του κόμβου 1 με δεδομένο το πρώτο στοιχείο της αντίστοιχης εγγραφής του PATHINDEX. Η εγγραφή αυτή λέει ότι ο 1 είναι το πρώτο στοιχείο του πρώτου μονοπατιού. Άρα ο επόμενος κόμβος θα είναι το δεύτερο στοιχείο του πρώτου μονοπατιού, δηλαδή ο κόμβος 12.

7.2.5 Κλάση Node

Η κλάση αυτή υλοποιεί ένα κόμβο του γράφου. Περιέχει τις μεταβλητές:

- int name: Όνομα του κόμβου.

- int position: Θέση του κόμβου.

7.2.6 Κλάση PathName

Η κλάση αυτή υλοποιεί ένα κομμάτι μονοπατιού. Οι μεταβλητές τις οποίες περιέχει είναι οι εξής:

- int pathId: Αναγνωριστικό μονοπατιού.
- int begin: Αρχή κομματιού.
- int end: Τέλος κομματιού.
- int pId: Αναγνωριστικό κόμβου στον οποίο γίνεται η σύνδεση των κομματιών.

Για τη συγκεκριμένη κλάση ορίζουμε και ένα συγκριτή (comparator), ο οποίος δίνεται από τον εξής κώδικα:

```
struct classcomp
{
	bool operator() (const PathName& lhs, const PathName& rhs) const
	{
		/*Compare MyClass members.*/
		if (lhs.pathId == rhs.pathId)
			return (lhs.begin < rhs.begin);
		else
			return (lhs.pathId < rhs.pathId);
	}
};
```

Ο κώδικας αυτός ορίζει πώς θα συγκρίνουμε δύο αντικείμενα της κλάσης Με τον τρόπο αυτό μπορούμε να χρησιμοποιήσουμε την κλάση PathName ως δείκτη σε πίνακες, όπως θα χρησιμοποιούσαμε πχ. έναν ακέραιο.

7.2.7 Κλάση main

Η κλάση αυτή χρησιμοποιεί όλες τις παραπάνω κλάσεις για να απαντήσει στα ερωτήματα με τα οποία ασχολούμαστε. Συγκεκριμένα, ακολουθεί την εξής διαδικασία:

- Κατασκευάζει το γράφο από το δεδομένο αρχείο.
- Κατασκευάζει τον PATHINDEX ή το σύνολο με τις λίστες γειτνίασης, ανάλογα με το τί μας χρειάζεται κάθε φορά.
- Εκτελεί για όσες επαναλήψεις θέλουμε τα εξής:
  - καλεί τις συναρτήσεις που υλοποιούν τις απαντήσεις στα διάφορα ερωτήματα με τα οποία ασχολούμαστε.
  - υπολογίζει και εκτυπώνει (στην οθόνη ή σε αρχείο) τους χρόνους για την εύρεση και/ή την εκτύπωση του μονοπατιού.
  - αρχικοποιεί τις δομές που χρησιμοποιούμε για να «θυμόμαστε» ποιούς κόμβους/κομμάτια μονοπατιών έχουμε επισκευθεί.

8

Επίλογος

Στο κεφάλαιο αυτό θα κάνουμε μια σύνοψη των όσων αναφέραμε στη διπλωματική εργασία και θα αναφέρουμε τα συμπεράσματα στα οποία καταλήξαμε. Επιπλέον, θα προτείνουμε διάφορες ιδέες που μπορούν να επεκτείνουν τη μέχρι τώρα εργασία μας.

8.1 Σύνοψη και συμπεράσματα

Στη διπλωματική εργασία αυτή ασχοληθήκαμε με την απάντηση ερωτημάτων πάνω σε δεδομένα οργανωμένα σε γράφους. Συγκεκριμένα, υλοποιήσαμε τα ερωτήματα:

α) Εύρεση παιδιών ενός κόμβου Α.

β) Εύρεση/υπολογισμός μονοπατιού ανάμεσα στους κόμβους Α και Β.

γ) Εύρεση/υπολογισμός μονοπατιού ανάμεσα στους κόμβους Α και Β υπό συνθηκη.

τόσο με κάποιες από τις ήδη υπάρχουσες μεθόδους, όσο και με αυτές που προτείνουμε εμείς. Στη συνέχεια πραγματοποιήσαμε διάφορες ομάδες πειραμάτων προκειμένου να αξιολογήσουμε την επίδοση των αλγορίθμων μας σε σχέση με τους ήδη υπάρχοντες αλγορίθμους.

Τα συμπεράσματά μας μπορούν να συνοψιστούν στα παρακάτω:

- Για το ερώτημα εύρεσης μονοπατιού ανάμεσα σε δύο κόμβους Α και Β:
    - Συμπεράσματα: Σχεδόν όλες οι μέθοδοι που προτείνουμε είναι καλύτερες από τις TRAD μεθόδους. Αυτό είναι ιδιαίτερα εμφανές για πυκνούς γράφους. Για τα NAI-queries ξεχωρίζουν σε απόδοση οι μέθοδοι που προσθέτουν ολόκληρα κομμάτια μονοπατιών στο μέτωπο αναζήτησης(MERGE), ενώ για τα OXI-queries είναι καλύτερες οι μέθοδοι που προσθέτουν χωριστά τους κόμβους(SEP). Σε όλες σχεδόν τις περιπτώσεις είναι καλύτερες οι μέθοδοι που χρησιμοποιούν ουρά ως μέτωπο αναζήτησης, δηλαδή συμπεριφέρονται όπως η μέθοδος αναζήτησης κατά πλάτος (BFS).

- Βελτιώσεις: Μια πιθανή βελτίωση των αλγορίθμων μας είναι να θυμόμαστε ποιοι κόμβοι οδηγούν σε λύση και ποιοι όχι, ώστε να αποφεύγουμε επανειλημμένες επισκέψεις σε κόμβους που δεν οδηγούν στον κόμβο Β.

- Για το ερώτημα εύρεσης μονοπατιού ανάμεσα σε δύο κόμβους Α και Β με περιορισμό ως προς το μήκος του μονοπατιού:
  - Συμπεράσματα: Οι μέθοδοι που προτείνουμε είναι γενικά πιο αργές από αυτές που χρησιμοποιούν λίστες γειτνίασης για την αναπαράσταση του γράφου. Αυτό είναι πολύ πιθανό να οφείλεται στους συχνούς ελέγχους που κάνουμε, καθώς και στο ότι αγνοούμε λύσεις, λόγω παραβίασης του περιορισμού, παρ'ότι καταλήγουμε σε αυτές νωρίτερα. Οι μέθοδοί μας πλησιάζουν πολύ τη μέθοδο με τις λίστες γειτνίασης για πυκνούς γράφους και μεγάλα μήκη μονοπατιών.
  - Βελτιώσεις - Παραλλαγές:
    - Οι αναπαραστάσεις επηρεάζουν την απόδοση, των αλγορίθμων μας. Για το λόγο αυτό, θα ήταν πολύ χρήσιμο να βρούμε διάφορα χαρακτηριστικά των αναπαραστάσεων και να προσπαθήσουμε να κατασκευάσουμε αναπαραστάσεις που γενικά θα βοηθούν τους αλγορίθμους αυτούς.
    - Μπορούμε να προτείνουμε διάφορες βελτιώσεις για τους αλγορίθμους μας. Μια από αυτές είναι να αποθηκεύουμε το αν ένας κόμβος μπορεί να οδηγήσει στον κόμβο Β και έτσι να αποφεύγουμε επανειλημμένες επισκέψεις σε κόμβους που δεν οδηγούν σε αποτέλεσμα.
    - Απάντηση στο ίδιο ερώτημα θα μπορούσαν να δώσουν και αλγόριθμοι που είναι παραλλαγές των αλγορίθμων του προηγούμενου ερωτήματος. Ωστόσο, οι αλγόριθμοι αυτοί δεν είναι τόσο ευέλικτοι όσο αυτοί που προτείνουμε εδώ, ως προς την οπισθοδρόμηση της αναζήτησης, δηλαδή δεν μπορούμε εύκολα να αλλάξουμε το μονοπάτι που οδηγεί στη λύση και συχνά προκύπτουν κύκλοι.

8.2 Μελλοντικές επεκτάσεις

Στην ενότητα αυτή θα προτείνουμε μερικές ιδέες για επέκταση της μέχρι τώρα δουλειάς μας. Οι ιδέες αυτές αφορούν δύο θέματα:

1. Συμπίεση των γράφων.
2. Απάντηση περισσότερων ερωτημάτων.

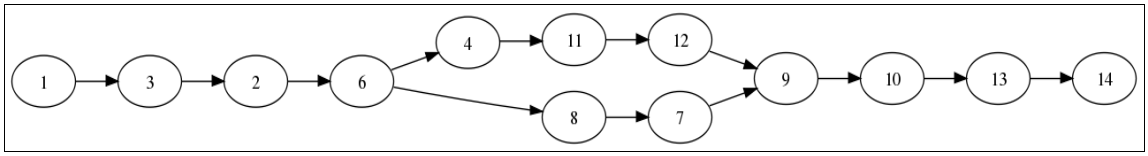
8.2.1 Συμπίεση γράφων

Από τους διάφορους γράφους που χρησιμοποιήσαμε στα πειράματά μας, παρατηρήσαμε ότι υπάρχουν ομάδες κόμβων που θα μπορούσαν να αντιμετωπιστούν ως ενιαία σύνολα έτσι ώστε να

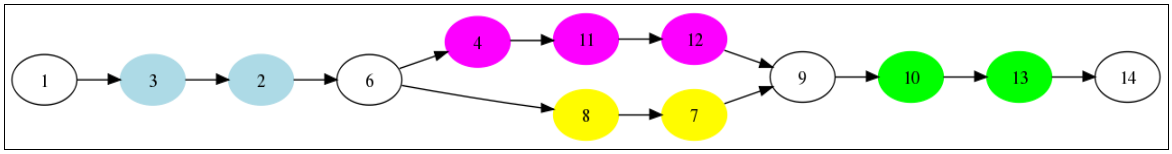
μειωθεί ο απαιτούμενος χρόνος απάντησης ενός ερωτήματος και συνεπώς να αυξηθεί η απόδοση των μεθόδων μας. Αντικαθιστώντας ομάδες κόμβων με κόμβους που «αντιπροσωπεύουν» τις ομάδες αυτές (τους ονομάζουμε υπερκόμβους), στην ουσία συμπιέζουμε τους γράφους. Για να συμπιέσουμε ένα γράφο θέτουμε κάποιους κανόνες με βάση τους οποίους ομαδοποιούνται οι κόμβοι του.

Κάποιοι από τους κανόνες είναι:

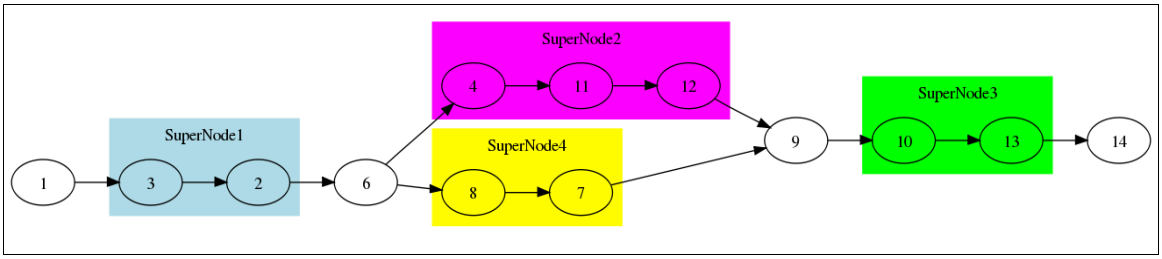
1. Ομαδοποίησε σε έναν υπερκόμβο τους κόμβους που σχηματίζουν μια αλυσίδα, στην οποία κάθε κόμβος έχει ένα πατέρα και ένα παιδί. Εναλλακτικά, κανένας κόμβος της αλυσίδας δεν πρέπει να συμμετέχει σε διακλάδωση (Σχήματα 8.1 – 8.3).
2. Ομαδοποίησε σε έναν υπερκόμβο τους κόμβους που σχηματίζουν μια αλυσίδα, στην οποία κάθε κόμβος έχει ένα πατέρα και ένα παιδί, ή μόνο ένα πάτερα, ή μόνο ένα παιδί. Ο κανόνας αυτός είναι επέκταση του κανόνα 1, έτσι ώστε να διαχειρίζεται επιπλέον τους κόμβους-φύλλα και τους κόμβους-ρίζες (Σχήματα 8.4 – 8.5).
3. Ομαδοποίησε σε έναν υπερκόμβο τους κόμβους/υπερκόμβους που έχουν τον ίδιο πατέρα (μοναδικό) και το ίδιο παιδί(μοναδικό), μαζί με τον κόμβο-πατέρα και τον κόμβο-παιδί (Σχήμα 8.6).



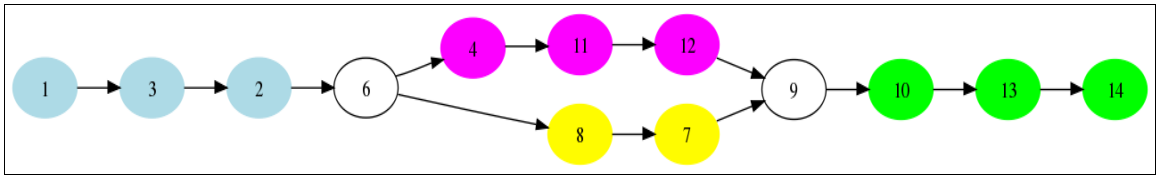
Σχήμα 8.1(Αρχικός γράφος)



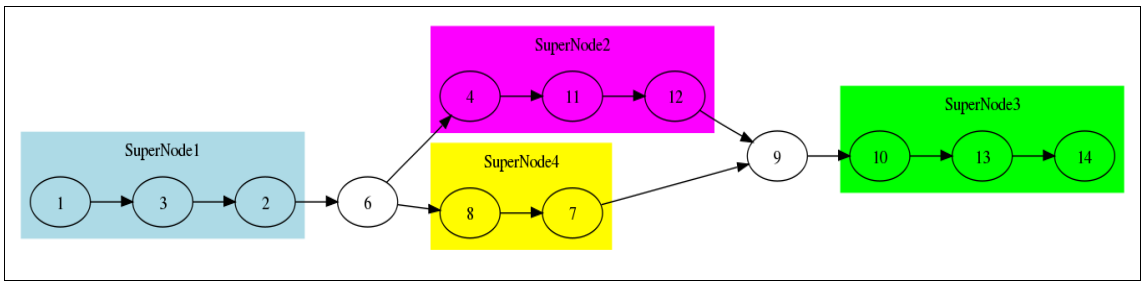
Σχήμα 8.2(Εφαρμογή κανόνα 1 – κατασκευή υπερκόμβων)



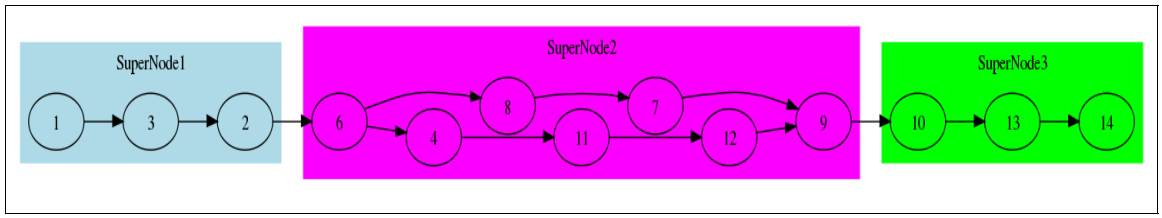
Σχήμα 8.3(Εφαρμογή κανόνα 1 - αποτέλεσμα)



Σχήμα 8.4(Εφαρμογή κανόνα 2- κατασκευή υπερκόμβων)

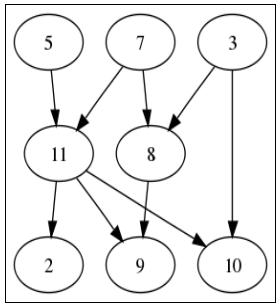


Σχήμα 8.5(Εφαρμογή κανόνα 2 - αποτέλεσμα)



Σχήμα 8.6(Εφαρμογή κανόνα 3 – αποτέλεσμα)

Από τα Σχήματα 8.1 – 8.6 είναι προφανές ότι αν αντικαταστήσουμε ομάδες κόμβων με υπερκόμβους, το μέγεθος του γράφου μειώνεται. Φυσικά, ο βαθμός συμπίεσης εξαρτάται και από τη δομή του γράφου. Έτσι, ενώ για τον παραπάνω γράφο πετύχαμε συμπίεση περίπου 80%, σε ένα γράφο όπως αυτός του Σχήματος 8.7. δεν θα είχαμε καθόλου συμπίεση. Επομένως, θα πρέπει να εχουμε περισσότερους κανόνες, προκειμένου να μπορούμε να συμπιέσουμε μεγαλύτερο σύνολο γράφων.



Σχήμα 8.7

Από πλευράς υλοποίησης, αυτό που θα πρέπει να κάνουμε είναι να διαβάσουμε το αρχείο που περιέχει το γράφο και να φτιάξουμε έναν PATHINDEX ο οποίος, όμως, θα περιέχει και πληροφορίες για τους υπερκόμβους. Ο γράφος που δεχόμαστε ως είσοδο θα πρέπει να αναπαρίσταται ως σύνολο από λίστες γειτνίασης και όχι ως σύνολο μονοπατιών για να μην έχουμε εξάρτηση του υπεργράφου από την αναπαράσταση.

8.2.2 Απάντηση περισσότερων ερωτημάτων

Η αναπαράσταση των γράφων ως σύνολα μονοπατιών μπορεί να κάνει αποδοτικότερη την απάντηση πολλών ερωτημάτων. Εκτός από τα ερωτήματα με τα οποία ασχοληθήκαμε στη διπλωματική εργασία, υπάρχουν πολλά άλλα που θα μπορούσαν να αποτελέσουν αντικείμενο μελέτης. Παραδείγματα είναι η εύρεση του συντομότερου μονοπατιού ανάμεσα σε δύο κόμβους ή των συνεκτικών συνιστωσών ενός γράφου και η απάντηση περισσότερο σύνθετων ερωτημάτων.

9

Βιβλιογραφία

[CW03] *Near-Shortest and K-Shortest Simple Paths*, W. Matthew Carlyle, R. Kevin Wood, Operations Research Dept. Naval Postgraduate School Monterey, CA 93943, 2 June 2003

[TU] *Association Search in Social Networks* ,Tsinghua University,Department of Computer Science and Technology

10-201, East Main Building, Tsinghua University, Beijing, China, 100084

[SL97] *CCAM: A Connectivity-Clustered Access Method for Networks and Network Computations*, Shashi Shekhar, Senior Member, IEEE, and Duen-Ren Liu, Associate Member, IEEE, IEEE Transactions On Knowledge And Data Engineering, Vol. 9, No. 1, January-February 1997

[LD89] *A File Structure Supporting Traversal Recursion*, P.-A. Larson and V. Deshpande, Department of Computer Science,University of Waterloo,Waterloo, Ontario,Canada,N2L 3Gl, 1989

[ABJ89] *Efficient Management Of Transitive Relationships In Large Data And Knowledge Bases*, Rakesh Agrawal*,Alexander Borgida **,H. V. Jagadish *, 1989

* AT&T Bell Laboratorie.s, Murray Hill, New Jersey 07974,** Rutgers University, New Brunswick, New Jersey 08903

[L87] *New Strategies for Computing the Transitive Closure of a Database Relation*, Hongjun Lu, Honeywell Inc., Corporate Systems Development Division,Goden Valley, MN 55427, Proceedings of the 13th VLDB Conference, Brighton 1987

[AJ88] *Efficient Search In Very Large Databases*, Rakesh Agrawal, H. V. Jagadish, AT&T Bell Laboratories, Murray Hill, New Jersey 07974, Pmceedings of

the 14th VLDB Confemce, Los Angeles, California 1988

[CW91] *An Improved Two-Way Partitioning Algorithm with Stable Performance*, C.K. Cheng and Y.C. Wei, IEEE Trans. Computer-Aided Design, vol. 10, no. 12, pp. 1,502–1,511, Dec. 1991.

[BR86] *An amateur's introduction to recursive query processing strategies*, Bancilhon, F. and Ramakrishnan, R., Proceedings of the 1986 ACM SIGMOD international conference, Washington, D.C, May 1986.

[CLR01] *Introduction to Algorithms, Second Edition*, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, The MIT Press, Cambridge ,Massachusetts London, England McGraw-Hill Book Company Boston Burr Ridge , IL Dubuque , IA Madison , WI New York San Francisco St. Louis Montreal Toronto

[IVHS94] Intelligent Vehicle Highway Systems Projects, Dept. of Transportation, Minnesota Document, Mar. 1994.

[BW84] *Determining all optimal and near-optimal solutions when solving shortest path problems by dynamic* programming, Byers, T.H. and Waterman, M.S., Operations Reasearch, 32, pp. 1381-1384, 1984.