Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα

Χειμερινό Εξάμηνο 2016-2017
Project Report
"Εύρεση συντομότερων μονοπατιών σε γράφο"

Ομάδα

Πουλοπούλου Παναγιώτα **1115201100110**
Παπαρρηγόπουλος Παναγιώτης **1115201100125**
Κωσταγιόλας Νικόλαος-Στέφανος **1115201100039**

Α' ΜΕΡΟΣ

1. Δομές για την αποθήκευση του γράφου

Ο Buffer

Η δομή αυτή αποθηκεύει τους γείτονες κάθε κόμβου. Εφόσον κάθε κόμβος του γράφου έχει την δική του λίστα γειτνίασης, κάθε κόμβος της λίστας περιέχει ένα πίνακα σταθερού μεγέθους με τους γείτονες και τις ιδιότητες των ακμών. Το μέγεθος του πίνακα είναι τέτοιο ώστε να ελαχιστοποιούμε τις μετακινήσεις στο χώρο για να προσπελάσουμε τους γείτονες ενός κόμβου αλλά και την σπατάλη χώρου. Η δομή αυτή δεσμεύει ένα συνεχόμενο χώρο μνήμης και αποθηκεύει τις λίστες γειτνίασης σε αυτόν. Με αυτό τον τρόπο μπορούμε να εκμεταλλευτούμε τις ιδιότητες της τοπικής χωρητικότητας κατά τους υπολογισμούς των επερωτήσεων.

Η κάθε θέση του Buffer όπως προαναφέρθηκε, αντιστοιχεί σε μια λίστα γειτνίασης. Σε περίπτωση που ένας κόμβος έχει παραπάνω γείτονες από όσους χωράει η λίστα γειτνίασης η οποία του αντιστοιχεί (δηλαδή 50+), τότε εκχωρείται σε αυτόν μια δεύτερη λίστα γειτνίασης. Η σύνδεση της νέας λίστας γειτνίασης με την λίστα γειτνίασης της οποίας η χωρητικότητα εξαντλήθηκε γίνεται αλυσιδωτά, με την τελευταία να δείχνει στην πρώτη. Σε περίπτωση που ο ίδιος ο Buffer υπερχειλίσει, δηλαδή απαιτούνται παραπάνω λίστες γειτνίασης από όσες μπορεί να χωρέσει, τότε ακολουθεί διπλασιασμός του Buffer.

Ως σύμβαση, αρχικοποιήσαμε το μέγεθος του Buffer σε 1024.

Το ευρετήριο κόμβων (Index)

Το ευρετήριο των κόμβων επιτρέπει την ανάκτηση των γειτόνων από την δομή του **buffer**,
για κάθε κόμβο το ευρετήριο διατηρεί ένα δείκτη προς την αρχή της λίστας γειτνίασης του.

Όπως και στις υπόλοιπες δομές, έτσι και εδώ όταν υπερχειλίζει το ευρετήριο, διπλασιάζεται με τρόπο παρόμοιο με τον διπλασιασμό του Buffer. Ως σύμβαση επιλέξαμε το αρχικό μέγεθος του ευρετηρίου να είναι ίσο με 10000.

Για την ασφαλέστερη σύνδεση του κάθε κόμβου της δομής Index με την αντίστοιχη αλυσίδα πινάκων γειτόνων της δομής Buffer αποφασίσαμε να μη χρησιμοποιούμε δείκτες από τη δομή Index στη δομή Buffer, λόγω των πολλών περιπτώσεων ασύγχρονης κλήσης της συνάρτησης realloc. Αντίθετα, χρησιμοποιήσαμε ακέραιο offset αρχικοποιημένο στην τιμή -1 το οποίος αντιστοιχεί στη θέση του Buffer στην οποία βρίσκεται η επιθυμητή λίστα. Για παράδειγμα Στην περίπτωση όπου οι γείτονες του κόμβου Α βρίσκονται στη θέση 3 τότε η τιμή του offset στη θέση Index[A] θα ισούται με 3. Στην περίπτωση που υπάρχουν συνεχόμενες λίστες

γειτόνων για τον Α, τότε η τιμή του offset στην θέση Buffer[3] θα ισούται με την θέση στο Buffer της επόμενης λίστας στην αλυσίδα γειτόνων του Α.

2. Υλοποίηση αλγορίθμου εύρεσης συντομότερου μονοπατιού.

Για την εύρεση του συντομότερου μονοπατιού θα χρησιμοποιηθεί ο αλγόριθμος **Bidirectional Breadth-First-Search**. Ο αλγόριθμος αυτός χρησιμοποιεί απο δύο κατευθύνσεις (από τον αρχικό προς τον τελικό και απο τον τελικό προς τον αρχικό κόμβο) τον αλγόριθμο BFS για να βρει το κοντινότερο μονοπάτι ανάμεσα σε δύο κόμβους. Σε κάθε βήμα του BFS, ξεκινώντας απ' τους αρχικούς κόμβους κάθε αναζήτησης, γίνονται expand τα παιδιά του κόμβου που βρίσκεται στην αρχή της ουράς. Κατά το expand τα παιδιά μπαίνουν στο τέλος της ουράς. Όταν υπάρχει ένας κόμβος και στις δύο ουρές των κατευθύνσεων τότε σημαίνει πως οι εξαπλώσεις των δύο αναζητήσεων ενώθηκαν στον κοινό κόμβο και προστίθενται οι αποστάσεις από τις δύο πλευρές για να βρεθεί το ελάχιστο μονοπάτι.

3. Unit testing

Στο πλαίσιο του πρώτου μέρους της άσκησης υλοποιήθηκαν ανάλογα unit tests. Τα unit tests μας επέτρεψαν να ελέγξουμε τις βασικές λειτουργίες που αναπτύχθηκαν τόσο για την δομή αποθήκευσης του γράφου και για την αναζήτηση συντομότερου μονοπατιού. Το πρόγραμμά μας, βάσει των unit tests, θα πρέπει για συγκεκριμένη είσοδο το πρόγραμμα να παράγει την αναμενόμενη έξοδο.
Αναλυτικότερα υλοποιήθηκαν:

- Δύο tests για τη δομή:
    - testStructureSizes: ελέγχει, μετά την είσοδο από το testGraph.txt ότι τα μεγέθη των δομών αντιστοιχούν στις απαιτούμενες τιμές.
    - testNeighborsOfNode : ελέγχει, μετά την είσοδο από το testGraph.txt ότι οι γείτονες των κόμβων αντιστοιχούν στους αναμενόμενους γείτονες μετά τις είσοδο.
- Ένα test για τον bi-directional BFS:
    - tinyGraphBFSTest: εισάγει τα δεδομένα του κόμβου που δίνονται στο αρχείο tinyGraph.txt και στη συνέχεια καλεί όλα τα ερωτήματα του αρχείου tinWorkload_FINAL.txt και ελέγχει εάν οι απαντήσεις στα ερωτήματα Q είναι οι αναμενόμενες με βάση το αρχείο tinyWorkload_RESULTS.txt

Β' ΜΕΡΟΣ

Στο δεύτερο μέρος της άσκησης προστέθηκαν κάποιες λειτουργικότητες οι οποίες έκαναν την απάντηση των ερωτημάτων ταχύτερη.

Hash Table
Σε κάθε node του Index προστέθηκε μία δομή ευρετηρίου HashTable στην οποία αποθηκεύονται ξανά όλοι οι κόμβοι του κάθε node. Με τη βοήθεια της δομής αυτής δεν χρειάζεται κάθε φορά που θέλουμε να ελέγξουμε αν κάποια ακμή υπάρχει ήδη στο index μας να κοιτάμε ολόκληρο το πίνακα γειτνίασης.

Δομές πίνακα και αλλαγές στον bi-directional BFS
Οι δομές λίστας που χρησιμοποιούσε ο αλγόριθμος BFS αντικαταστάθηκαν από δομές πίνακα οι οποίες αρχικοποιούνται μόνο μία κατά τη δημιουργία της κλάσης BFS και καταστρέφονται στο τέλος μαζί της. Αφού αυτές οι δομές αρχικοποιούνται μόνο μία φορά και ουσιαστικά δεν αδειάζουν μετά το πέρας ενός ερωτήματος πρέπει ο αλγόριθμος να μπορεί να διαχωρίσει τις τιμές που περιέχονται και αφορούν το τρέχον ερώτημα που εκτελεί. Γι' αυτό το λόγο κάθε ερώτημα διαχωρίζεται από τα υπόλοιπα με έναν αναγνωριστικό

αριθμό version. Η δομή της ουράς αντικαταστάθηκε με έναν πίνακα που κρατάει δύο δείκτες για να ξέρουμε τα front και rear της ουράς. Σε κάθε νέο ερώτημα, αντί να διαγράφεται η δομή, απλώς αρχικοποιούνται οι δείκτες front και rear. Επίσης, κάθε φορά που επισκέπτεται ο αλγόριθμος έναν κόμβο αντί να σημειώνει true σε έναν boolean πίνακα (inQueueInternal, inQueueExternal) αυξάνει τον μετρητή του κόμβου για αυτον τον πίνακα. Έτσι, ένας κόμβος θεωρείται visited μόνο εάν ο μετρητής του είναι ίδιος με την έκδοση (version) του ερωτήματος που εκτελείται.
Μία άλλη αλλαγή που προστέθηκε στον αλγόριθμο είναι σε κάθε στάδιο να μην γίνονται expand και οι δύο πλευρές του γράφου αλλά μόνο η πλευρά όπου το σύνολο των παιδιών των κόμβων που περίμενουν στην ουρά για να γίνουν expand είναι το μικρότερο. Αυτή η αλλαγή σε συνδιασμό με την αποφυγή της συνεχής δέσμευσης μνήμης για κάθε ερώτημα είχαν ως αποτέλεσμα πολυ μεγάλη βελτιωση στην απόδοση του προγράμματος.

Δομή Connected Components για δυναμικούς γράφους

Για τους δυναμικούς γράφους μετά την εισαγωγή τους δημιουργείται η δομή ccindex η οποία είναι ένας πίνακας που κάθε θέση του αντιστοιχεί σε κόμβους του γράφου και το περιεχόμενο αντιστοιχεί στο id του connected component που ανήκει ο κόμβος. Η δημιουργία αυτού του πίνακα γίνεται με έναν απλό αλγόριθμο BFS.
Κάθε φορά που έρχεται μία νέα ακμή στον γράφο κάποια connected components πιθανόν να ενώνονται. Αυτή η πληροφορία αποθηκεύεται στη δομή updateIndex η οποία είναι ένας πίνακας και κάθε θέση της αντιστοιχεί σε ένα connected component. Το περιεχόμενο κάθε θέσης του πίνακα έχει έναν αριθμό uint32_t. Τα connected components που έχουν τον ίδιο αριθμό σημαίνει πως έχουν ενωθεί μεταξύ τους με κάποιο ερώτιμα Add. Αυτός ο αριθμός είναι ένας μετριτής που αυξάνεται για κάθε Add. Κάθε φορά που κάποιο Add ενώνει δύο components ο μετριτής αυξάνεται και ενημερώνει όλες τις θέσεις του πίνακα που είχανε τον ίδιο αριθμό με τα components που ενώθηκαν.

Δομές Strongly Connected Components και Grail Index για στατικούς γράφους

Για την εύρεση των ισχυρά συνδεδεμένων συνιστωσών του γράφου χρησιμοποιήθηκε ο αλγόριθμος του Tarjan ο οποίος αποτελεί μια παραλλαγή του αλγορίθμου DFS με τη χρήση τριών πινάκων (visited, lowlink και disc) και μιας στοίβας. Ο αλγόριθμος αυτός λαμβάνει ως είσοδο το γράφημα και επιστρέφει τα Strongly Connected Components του γραφήματος αυτού. Χρησιμοποιήσαμε μια παρόμοια δομή με αυτή που αποθηκεύουμε τα Connected Components η οποία επίσης διπλασιάζεται δυναμικά πριν από κάθε υπερχείλιση. Η δομή αυτή αντιστοιχίζει το id του κάθε κόμβου του γραφήματος με το id του component στο οποίο ανήκει. Τέλος, υλοποιήθηκαν και συναρτήσεις οι οποίες υλοποιούν βοηθητικές λειτουργίες όπως π.χ. ο υπολογισμός της απόστασης μεταξύ δύο κόμβων του γραφήματος που ανήκουν στο ίδιο Strongly Connected Component και η διάσχιση της δομής με τη χρήση του αντικειμένου ComponentCursor.
Το ευρετήριο grails αποτελείτε από έναν γράφο με κόμβους οι οποίοι περιέχουν την πληροφορία για τα rank βάση της οποίας στο τέλος γίνεται η απάντηση των ερωτημάτων. Ο υπολογισμός των rank κάθε κόμβου γίνεται με την αναδρομική κλήση της συνάρτησης expandNode. Ως είσοδος το ευρετήριο δέχεται ένα index από τα strongly connectly components του γράφου το οποίο δημιουργείται διατρέχοντας τον αρχικό γράφο και εισάγοντας για καθε ακμή A -> B την ακμή sccOf(A) -> sccOf(B).

Γ' ΜΕΡΟΣ

Πολυνυματισμός

Για την παραλληλοποίηση του υπολογισμού χρησιμοποιήθηκε η δομή JobScheduler. Ο JobScheduler αποτελείτε από μια ουρά από Jobs τα οποία δημιουργούνται κατά το διάβασμα του workload. Το κάθε job αποτελείτε από τις δυο ακμές για τις οποίες ψάχνουμε μονοπάτι σύνδεσης καθώς και πληροφορίες σχετικά με το version του καθε query. Το version αυτό χρησιμοποιείτε για να λαμβάνονται υπόψιν μονό οι ακμές που υπήρχαν στον γραφο όταν ήρθε το query. Ο JobScheduler περιέχει επίσης ένα pool από threads τα οποια

αναλμβανουν να τραβανε και να εκτελούναι Jobs απο την ουρά. Ο συγχρονισμός των νημάτων αυτών γίνεται με την χρήση δύο σημαφόρων και δύο conditional variables, ένα ζευγαρι σημαφόρου και conditional variable που ελέγχει τη λειτουργία των νημάτων worker και άλλο ένα ζευγάρι για το main νήμα. Ο αριθμός των νημάτων γίνεται defined στο main.cpp αρχείο.

Πίνακας με Χρόνους

	Small dynamic	**Medium dynamic**	**Medium static**
Part 1	2.5 hours	∞	-
Part 2	15 seconds	35.77 seconds	12.45 seconds
Part 3	2.17 seconds	40.77 seconds	15.21 seconds