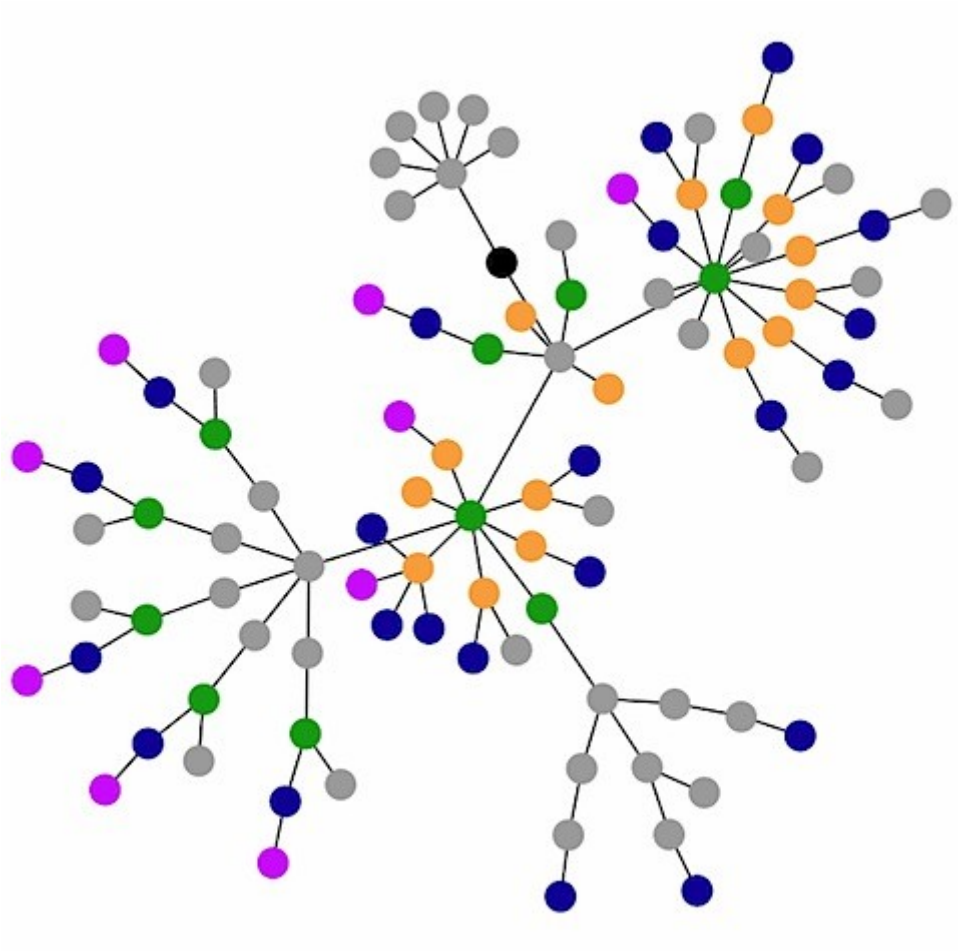
Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Ανάπτυξη Λογισμικού Πληροφορικής
Χειμερινό Εξάμηνο 2016-2017
“Εύρεση συντομότερων μονοπατιών σε γράφο”

Υπεύθυνος Καθηγητής:
Ιωάννης Ιωαννίδης

Συνεργάτες Μαθήματος:
Μαίλης Θεόφιλος, Γιαννακόπουλος Αναστάσιος, Γαβανάς Χρήστος, Γαλούνη Κωνσταντίνα, Τσαμπίκος Λιβισιανός



Μέλη Ομάδας

Κατούνας Γεώργιος	1115201200061
Σπάχου Ευαγγελία	1115201200169
Σταμούλου Κωνσταντίνα	1115201200173

Α’ Μέρος της Άσκησης

Δομές αναπαράστασης γράφου:

- Graph: η δομή graph στην οποία ομαδοποιούμε τα 2 buffer, τα 2 index και τα 2 bufferInfo.
- Buffer: είναι ένας πίνακας από listnode για τον οποίο διατηρούμε μία παράλληλη δομή BufferInfo στην οποία κρατάμε το μέγεθός του(size) και το πρώτο διαθέσιμο ListNode(available). Επίσης όταν γίνονται οι εισαγωγές από το workload ελέγχουμε σειριακά αν υπάρχει ήδη ο γείτονας για να μην υπάρχουν διπλότυπα στη λίστα των γειτόνων.
- Index: ένας πίνακας με μεγέθος όσοι είναι οι κόμβοι και σε κάθε θέση του κραταμε κάποια στοιχεία που μας βοηθούν να κάνουμε πιο γρήγορα τις εισαγωγές. Πιο συγκεκριμένα έχουμε το first που είναι το πρώτο listnode των γειτόνων του κόμβου, το last που είναι το τελευταίο και το pos που είναι η πρώτη διαθέσιμη θέση για εισαγωγή.

Οι παραπάνω δομές είναι δυναμικές και η κάθεμία δεσμεύει ένα συνεχόμενο χώρο μνήμης αφού σε περίπτωση που γεμίσουν αυξάνουμε το μέγεθός τους με realloc. Έτσι εκμεταλλευόμαστε τις ιδιότητες της τοπικής χωρητικότητας.

Επιπλέον βοηθητικές δομές:

- HashEntry: το περιεχόμενο ενός hash table που δημιουργήσαμε για να ελέγχουμε γρήγορα αν δύο κόμβοι είναι γείτονες αλλά τελικά διαπιστώσαμε ότι η διαδικασία δημιουργίας του hash table ήταν περισσότερο χρονοβόρα από το να ελέγξουμε σειριακά αν οι δύο κόμβοι συνορεύουν.
- Id_nodes: η δομή στην οποία αποθηκεύουμε τους κόμβους που πρέπει να εξετάσουμε και με τους δείκτες first και last της δομής δεν χρειάζεται την αρχικοποιούμε κάθε φορά.

Αλγόριθμος εύρεσης συντομότερου μονοπατιού:

- BBFS: επιλέγει να ξεκινήσει από την πλευρά που έχει τα λιγότερα παιδιά για να εξετάσει, αφού έχουμε ήδη κρατήσει τον αριθμό των παιδιών. Οι πίνακες με τους κόμβους που έχουμε επισκεφτεί αρχικοποιούνται με 0. Κάθε φορά που καλείται ο αλγόριθμος χρησιμοποιείται ένας διαφορετικός αύξοντας αριθμός προκειμένου να τον βάλουμε στη θέση του κόμβου στον πίνακα όταν τον επισκεφτούμε. Αυτό γίνεται για να μην δημιουργούμε ή αρχικοποιούμε κάθε φορά τον πίνακα αλλά να τον αυξάνουμε αν χρειαστεί.

Β’ Μέρος της Άσκησης

Για τους στατικούς γράφους, αφού καλέσουμε τον αλγόριθμο του Tarjan και βρει τα Strongly Connected Components, δημιουργούμε έναν υπεργράφο με τα Components με τις ήδη υπάρχουσες δομές του part1 και στη συνέχεια δημιουργούμε τα ευρετήρια Grail NUM_OF_GRAILS σε αριθμό.

Επιπλέον βοηθητικές δομές:

- StaticData: περιλαμβάνει τον κύριο γράφο με τους κόμβους, τα ευρετήρια grail και τη δομή SCC.
- grailStack: η στοίβα αυτή χρησιμοποιείται στη συνάντηση buildGrailIndex για να αποθηκεύονται οι κόμβοι που δεν έχουμε επισκεφτεί ακόμη και δεν έχουν τιμή στα πεδία min_rank και rank. Δημιουργείται μια φορά στην αρχή και για τη δημιουργία κάθε GrailIndex χρησιμοποιείται η ίδια, επαναορίζοντας απλά τον counter της στοίβας, γλιτώνοντας έτσι χρόνο για την αρχικοποίηση ή τα malloc-free κάθε φορά.
- grailVisited: δημιουργείται μια φορά στην αρχή και αρχικοποιείται με 0. Σε κάθε κλήση της συνάρτησης buildGrailIndex στην οποία χρησιμοποιείται ελέγχει αν έχει ελεγχθεί ο κόμβος στην αντίστοιχη θέση του πίνακα. Για να γλιτώσουμε την αρχικοποίηση ή τα malloc-free κάθε φορά που καλείται η συνάρτηση, κάθε φορά περνάμε ένα διαφορετικό αριθμό που δηλώνει την επίσκεψη του κόμβου.

Για του δυναμικούς γράφους κάνουμε DFS, βρίσκουμε όλα τα Weekly Connected Components και στη συνέχεια δημιουργούμε έναν γράφο με όλα τα components. Όταν μας έρθει ένα query κάνουμε bbfs στον γράφο με τα components και αν ενώνονται τότε κάνουμε bbfs και στο γράφο με τους κόμβους, αλλιώς επιστρέφει -1.

Επιπλέον βοηθητικές δομές:

- DynamicData: περιλαμβάνει τον κύριο γράφο με τους κόμβους, το γράφο με τα components και τον πίνακα ccindex που δείχνει σε ποιο component ανήκει ο κάθε κόμβος.

Χρόνοι εκτέλεσης:

Static Input	1 thread
medium	2.17min
large	6.38min

Dynamic Input	1 thread
medium	3.942s
large	26.34min

Γ’ Μέρος της Άσκησης

Χρησιμοποιούμε 3 σημαφόρους, έναν για το thread_id, καθώς δίνουμε στα threads μια τιμή από το 0 μέχρι το NUM_OF_THREADS-1, έτσι ώστε να έχει το καθένα τις δικές του δομές που χρησιμοποιούνται στις συναρτήσεις. Ο δεύτερος είναι για την πρόσβαση στην ουρά με τα jobs και ο τρίτος είναι για μία global μεταβλητή στην οποία έχουν πρόσβαση όλα τα threads και τη μειώνουν κάθε φορά που τελειώνουν το job τους. Όταν γίνει 0 αυτή η μεταβλητή, ξυπνάει η main και γεμίζει την ουρά.

Χρόνοι εκτέλεσης:

Static Input	2 threads	4 threads	6 threads	8 threads	10 threads	12 threads	20 threads
medium	1.34min	1.33min	1.38min	1.33min	1.34min	1.31min	1.32min
large	4.7min	3.3min	3.57min	2.48min	3.40min	3.2min	3.3min

Dynamic Input	2 threads	4 threads	6 threads	8 threads	10 threads	12 threads	20 threads
medium	3.298s	3.118s	3.118s	3.032s	2.917s	3.043s	2.976s
large	20.25min	14.53min	14.25min	13.45min	13.35min	14.12min	16.4min

Προδιαγραφές συστήματος:

- RAM: 6GB
- Επεξεργαστής:Intel® Core™ i5-4210U CPU @ 1.70GHz × 4

Τέλος, έχουμε υλοποιήσει τα 2 πηγαία αρχεία utest.c και utest2.c, στα οποία ελέγχουμε τη λειτουργικότητα των δομών που υλοποιήσαμε. Το αρχείο testGraph.txt περιέχει τον γράφο που χρησιμοποιούμε για τα τεστ. Υπάρχουν λεπτομερή σχόλια που περιγράφουν τι κάνει το κάθε αρχείο και τι πρέπει να εμφανίσουν αυτά που εκτελεί.
Η μεταγλώττιση γίνεται με τις ακόλουθες εντολές:

- gcc -o utest utest.c index.c graph.c buffer.c bbfs.c
- gcc -o utest2 utest2.c graph.c index.c buffer.c bbfs.c SCC.c stack.c grail.c