Project - Part I, II & III

Μανωλάκη-Σεμπάγιος Μαρία-Ισαβέλλα 1115201300093
Σταματοπούλου Παναγιώτα 1115201300167

Γλώσσα Υλοποίησης : C++

Compile:
Με χρήση make.

Run:
./graph <creationFile> <workloadFile> > out.txt

Υλοποίηση:

- ListNode: Περιέχει πίνακα γειτόνων σταθερού μεγέθους και δείκτη (int index του Buffer) σε επόμενο ListNode του συγκεκριμένου κόμβου. Επίσης το head ListNode του κάθε κόμβου περιέχει πληροφορίες που αφορούν το σύνολο των γειτόνων, όπως το πλήθος τους (totalNodeCount) και το index του σχετικού ListNode που χρησιμοποιήθηκε τελευταίο.
- Buffer: Πίνακας από ListNodes που δημιουργείται έχοντας σταθερό (defined) μέγεθος και διπλασιάζεται κάθε φορά που ένα στοιχείο δεν χωράει σε κάποιο ListNode. Σε περίπτωση που χρησιμοποιήσει καινούργιο ListNode για κάποιο κόμβο έχει την ευθύνη της σύνδεσης με το προήγουμενο ListNode (του ίδιου κόμβου).
- NodeIndex: Πίνακας από integers που δημιουργείται έχοντας σταθερό (defined) μέγεθος και διπλασιάζεται μέχρι το μέγεθός του να είναι μεγαλύτερο ή ίσο με το id του στοιχείου που πρέπει να εισαχθεί. Κάθε θέση αντιπροσωπεύει έναν κόμβο και περιέχει το index του head ListNode του στον Buffer. Αρχικοποιείται με -1.
- Graph: Περιέχει διπλές τις δομές που αναφέρθηκαν, ώστε να αποθηκεύονται οι εισερχόμενες και οι εξερχόμενες ακμές.
- CC: Περιέχει πίνακα από integers στον οποίο αποθηκέυονται τα weakly connected components των κόμβων μετά την εκτέλεση του DFS για την εύρεση της συνεκτικότητας του γράφου. Επίσης περιέχει τη δομή UpdateIndex στην οποία αποθηκεύονται οι συνεκτικές συνιστώσεις που ενώνονται μετά τις εισαγωγές νέων κόμβων. Η δομή αυτή ανανεώνεται ανάλογα με την τιμή του metric.
- SCC: Υλοποιημένο με ίδια λογική με τον Buffer και το NodeIndex. Περιέχει τις ισχυρά συνεκτικές συνιστώσες (components_), τους δείκτες σε καθεμία (sccIndex_), πίνακα των κόμβων όπου κάθε κελί περιέχει το SCC στο οποίο ανήκει ο κόμβος (belongsToComponent_) και ComponentCursor. Για τον καθορισμό των SCCs εκτελείται ο επαναληπτικός αλγόριθμος του Tarjan.
- Grail: Στατικός πίνακας από πίνακες που περιέχουν GrailNodes, καθένας από τους οποίους αντιστοιχεί σε μία εκτέλεση του αλγορίθμου της δημιουργίας του GrailIndex. Χρησιμοποιείται για την αποθήκευση των πολλαπλών labels.
- Hypergraph: Υλοποιημένος με ίδια λογική με τον Buffer και το NodeIndex, με τη διαφορά ότι τα integers που αποθηκεύει αντιστοιχούν σε strongly connected components και όχι σε κόμβους. Περιέχει συνάρτηση DFS για την εύρεση πιθανής σύνδεσης μεταξύ των SCCs.

Μετά την υλοποίηση του δευτέρου part η δομή Graph διαχωρίστηκε σε SGraph/DGraph ώστε να εξοικονμηθεί μνήμη με την δημιουργία μόνο των δομών που χρειάζεται ο γράφος ανάλογα με το είδος (Grail, CC etc.).

Μετά την υλοποίηση του τρίτου part οι δομές Buffer, Listnode, NodeIndex διαχωρίστηκαν σε, SBuffer/DBuffer, SListNode/DListNode, SNodeIndex/DNodeIndex ώστε το edge property να δημιουργείται μόνο στους δυναμικούς γράφους εφόσον στους στατικούς δεν υπάρχει versioning.
Για την υλοποίηση των αλγορίθμων προσπέλασης του γράφου χρησιμοποιούνται επιπλέον στατικές δομές:

- Queue: Ουρά υλοποιημένη με στατικό πίνακα που διπλασιάζεται στην περίπτωση της υπερχείλισης. Περιέχει structs με το id ενός γείτονα και το επίπεδο στο οποίο εισήχθη. Αρχικοποιείται με -1.
- Visited: Array όπου κάθε κελί αντιστοχεί σε ένα node και ο αριθμός μέσα σε αυτό δείχνει την καταστασή του (εάν είναι visited κλπ). Για να αποφευχθούν οι αρχικοποιήσεις χρησιμοποιείται versioning.
- Stack: Στοίβα υλοποιημένη με στατικό πίνακα που διπλασιάζεται στην περίπτωση υπερχείλισης. Περιέχει structs με το id ενός κόμβου και δύο επιπλεόν integers που αξιοποιούνται ανάλογως σε κάθε αλγόριθμο.
- TarjanNodes & GrailNodes: Δομές που χρησιμοποιούνται κατά την εύρεση Strongly Connected Components και την δημιουργία GrailIndex αντίστοιχα και διατηρούν χρήσιμες πληροφορίες για το κάθε node (πχ parent).

Σχεδιαστικές Επιλογές:
Αρχικά, υλοποιήθηκε HashTable το οποίο χρησιμοποιήθηκε για το μαρκάρισμα των visited nodes στο Bidirectional BFS, αλλά όπως αναφέρθηκε παραπάνω αντικαταστάθηκε με τη δομή Visited, με αποτέλεσμα τη μείωση του χρόνου. Επίσης το HashTable χρησιμοποιήθηκε δοκιμαστικά ώστε να αποφευχθεί η γραμμική αναζήτηση της insertEdge στον γράφο. Η αρχική ιδέα ήταν κάθε ListNode να έχει το δικό του HashTable με fixed μέγεθος ανάλογο με το πλήθος των Neighbors. Η αλλαγή αυτή δεν μείωσε τον χρόνο αλλά αντιθέτως τον αύξησε, μάλλον λόγω αύξησης του απαιτούμενου χώρου. Επιπλέον παρατηρήθηκε ότι ο χρόνος που απαιτείται για την γραμμική αναζήτηση αποτελεί ένα πολύ μικρό (σχεδόν αμελητέο) ποσοστό του συνολικού. (Ο κώδικας του Hashtable, παρόλα αυτά υπάρχει στον φάκελο του παραδοτέου.)
Η περίπτωση που η γραμμική αναζήτηση επιβάρυνε πράγματι την εκτέλεση ήταν όταν γινόταν κλήση από την createHypergaph, λόγω της εμφώλευσης σε loops. Στην περίπτωση αυτή χρησιμοποίηθηκαν arrays με versioning για γρήγορο έλεγχο για το αν ένα SCC έχει ήδη εισαχθεί στο Hypergraph αντί για γραμμική search.
Η ουρά που χρησιμοποιείται στο Bidirectional BFS ήταν αρχικά κυκλική για εξοικονόμηση χώρου, αλλά λόγω της μεταφοράς δεδομένων στο σωστό σημείο σε κάποιες περιπτώσεις διπλασιασμού, αποδείχθηκε ότι ήταν αρκετά ακριβή υλοποιήση και για αυτό αντικαταστάθηκε από ένα απλό array.
Μια αλλαγή η οποία μείωσε πολύ τον χρόνο ήταν η δημιουργία δομών έξω από τις συναρτήσεις στις οποίες χρησιμοποιούνται (πχ BFS) και η αποφυγή επαναδημιουργίας και αρχικοποίησης εκ νέου.
Άλλη μια σημαντική αλλαγή ήταν η απόφαση για το ποια πλευρά του γράφου θα αναπτυχθεί κάθε φορά στο Bidirectional BFS όχι μόνο με βάση την μικρότερη ουρά αλλά και το πλήθος των γειτόνων που θα έχουν τα nodes συνολικά στο επόμενο στάδιο.
Επιπλέον, πολύ σημαντική μείωση του χρόνου παρατηρήθηκε με την αλλαγή της υλοποίησης του Tarjan και της κατασκευής του Grail Index. Αρχικά, γινόταν αναζήτηση των γειτόνων του γράφου ή του υπεργράφου ξεκινώντας κάθε φορά από τον πρώτο αλλά τελικά χρησιμοποιήθηκαν tarjanNodes και grailNodes που αποθηκεύουν τον τελευταίο γείτονα που έχουμε επισκεφθεί, ώστε να γίνεται άμεση ανάκτηση του επόμενου unvisited. Ο χρόνος μειώθηκε και με τη χρήση μίας στοίβας αντί για δύο στους παραπάνω αλγορίθμους.

Παρατηρήθηκε, επίσης, ότι στο BFS του δυναμικού γράφου είναι προτιμότερο να γίνεται αναζήτηση σε ολόκληρο το γράφο από το να ελέγχουμε το component στο οποίο ανήκει κάθε γείτονας. Συγκεκριμένα, ο έλεγχος του component ενώ μείωσε κατά μισό λεπτό το χρόνο στο large dataset, τον αύξησε κατά αρκετά δευτερόλεπτα στο medium.
Για την πρόσβαση στα strongly connected components παρότι έχει υλοποιηθεί iterator χρησιμοποιείται array της μορφής NodeIndex (κάθε θέση κρατάει listHead) διότι θεωρήσαμε ότι αυτή η επιλογή θα οδηγούσε σε καλύτερους χρόνους (άμεση πρόσβαση αντί για iteration πάνω σε όλα τα components - listNodes, συμπεριλαμβανομένων και των listNodes που αποτελούν extensions).
Όσον αφορά την εκτύπωση ο χρόνος βελτιώθηκε αρκετά με χρήση stream σε σχέση με τον χρόνο που χρειαζόταν για εκτύπωση στο stdout και λίγο περισσότερο με χρήση FILE* αλλά εν τέλει ο καλύτερος χρόνος επιτεύχθηκε με ανακατεύθυνση σε αρχείο.
Επίσης, σε μερικά σημεία οι αρχικοποιήσεις μεγάλων πινάκων σε loops αντικαταστάθηκαν με memset , με αποτέλεσμα μια μικρή μείωση στον χρόνο. Αρχικοποιήσεις που δεν χρειάζονταν αποφεύχθηκαν.
Έγινε η δοκιμή να μειωθούν οι κλήσεις συναρτήσεων αλλά δεν είχε ως αποτέλεσμα μείωση του χρόνου. Παρόλα αυτά σε μερικά σημεία έγινε προσπάθεια να αποφευχούν σε πολύ συχνά χρησιμοποιούμενες συναρτήσεις (πχ BFS).
Επίσης στα περισσότερα σημεία τα αντικείμενα δημιουργούνται στατικά αντί να δεσμεύονται δυναμικά, πράγμα που οδήγησε σε χαμηλότερο χρόνο.
Για το complilation χρησιμοποιήθηκε flag για optimization, κάτι που οδήγησε σε σημαντική μείωση του χρόνου. Αυτό που παρατηρήθηκε ότι είχε το καλύτερο αποτέλεσμα για τα μηχανήματα στα οποία έγινε το testing είναι το -O3. Επίσης, ενώ το testing έγινε σε μηχανήματα με δυπήρινο επεξεργαστή, ο ιδανικός αριθμός threads ήταν τα 6.

Χαρακτηριστικά Μηχανημάτων

- MacBook Pro (Late 2012)
  Ram 8 GB 1600 MHz DDR3
  2,5 GHz Intel Core i5
  64-bit
  Disk 128GB
  Supports hyperthreading

  OS X El Capitan 10.11.6

- Dell Inspiron 5558
  Ram 8GB
  Intel Core i5 5200U 2.2 GHz x 4
  64-bit
  Disk 1TB
  Supports hyperthreading

  Ubuntu 14.04 LTS

Χρόνοι

Static	Medium	Large
Mac OS	0m7.74s	5m3.60s

Linux	0m5.77s	5m42.01s

Dynamic	Medium	Large
Mac OS	0m6.71s	11m18.35s
Linux	0m5.58s	7m40.06s

Styleguide:
http://geosoft.no/development/cppstyle.html