Τεχνολογίες Υλοποίησης Αλγορίθμων
Version 2

Αλ-Μπαμπούλ Γεώργιος 5ο έτος
1059631

Σεπτέμβριος 2021

Contents

1 Εισαγωγή

Το παρόν έγγραφο αποτελεί την τεχνική αναφορά της Εργασίας που ανατέθηκε στον μαθητή Γεώργιο Αλ-Μπαμπούλ με ΑΜ 1059631 με τίτλο:

Υλοποίηση με χρήση της βιβλιοθήκης BOOST του αλγορίθμου εύρεσης μέγιστης ροής για διμερή δίκτυα [AMO93, Κεφ. 8.3], και εκτενής πειραματική αξιολόγησή του με τυχαία και συνθετικά δίκτυα σε σύγκριση με την υλοποίηση MAX_FLOW της LEDA.

Στην συνέχεια παρατίθενται διάφορες παραδοχές που έγιναν για την σωστή λειτουργία του προγράμματος και εξηγείται το σετ δεδομένων που χρησιμοποιήθηκε για την εξέταση της σωστής λειτουργίας του στην υλοποίηση της boost. Έπειτα εξηγείται το πρόγραμμα και το σετ δεδομένων για την υλοποίηση της LEDA. Τέλος παρατίθενται σε μορφή πίνακα οι μετρήσεις που λήφθηκαν για τις δύο υλοποιήσεις καθώς και συμπεράσματα πάνω σε αυτά.

2 Changelog

Από την εκδοχή 1 στην 2 έχουν αλλάξει 2 βασικά πράγματα. Αρχικά, πλέον χρησιμοποιούμε μία ουρά για τον βρόγχο while του βασικού κομματιού του αλγορίθμου. Δεύτερον πλέον τα γραφήματα που παράγονται και εξετάζονται είναι τυχαία με την χρήση της γεννήτριας της Leda.

3 Test Set

Για την δημιουργία ενός test set για να εξετάσουμε τον αλγόριθμο επιλέξαμε να δημιουργήσουμε γράφους με τυχαία capacity στις ακμές. Συγκεκριμένα, καθώς ο αλγόριθμος υλοποιείται για διμερείς γράφους επιλέξαμε να χρησιμοποιήσουμε τους "τετράγωνους" γράφους καθώς αυτοί ήταν διμερείς σύμφωνα με την λύση της πρώτης άσκησης. Οπότε έπρεπε αρχικά να "μεταφράσουμε" τον αλγόριθμο που τα δημιουργούσε σε Boost και να χρησιμοποιήσουμε το ίδιο στην LEDA. Ένα θέμα που αντιμετωπίσαμε στην υλοποιήση της LEDA ήταν το invalidation των edge descriptors που γινόταν κάθε φορά που δημιουργούσαμε ένα καινούριο edge, το οποίο αντιμετωπίστηκε με το να κάνουμε assign τα capacity στο τέλος της δημιουργίας των γράφων. Όσον αφορά την υλοποιήση της Boost, σε κάθε δημιουργία ενός edge βάζαμε και ένα τυχαίο βάρος απο το εύρος [0,100] αλλά δημιουργούσαμε και την αντίστροφή της με residual flow 0 καθώς δεν μπορούσαμε να φτιάξουμε καινούρια edge στην μέση του αλγορίθμου γιατί ο

γράφος είναι τύπου ορισμένου με vectors. Οπότε αν αλλάζαμε κάτι σε loop στο οποίο χρησιμοποιείται edge descriptor ή node descriptor γίνονταν invalidated και προκαλούσε segmentation fault.

4 Καινούριο Test Set

Πλέον το πρόγραμμα τρέχει με τυχαία διμερή γραφήματα. Αυτό επιτεύχθηκε με την χρήση της γεννήτριας της leda complete_bigraph. Ένα πρόβλημα το οποίο όμως αντιμετωπίσαμε με αυτή την συνάρτηση ήταν ότι δημιουργεί ακμές μόνο από το Σετ Α προς το Σετ Β. Για να το λύσουμε, για κάθε κόμβο στο Σετ Β δημιουργήσαμε έναν τυχαίο αριθμό από ακμές από τον κόμβο πρός έναν τυχαίο κόμβο του Σετ Α. Ένα άλλο πρόβλημα που προκύπτει από αυτή την υλοποίηση είναι το γεγονός ότι έχουμε περισσότερες από μία ακμές προς μία κατεύθυνση. Δεν έχει βρεθεί λύση για το συγκεκριμένο πρόβλημα. Το καινούριο τυχαίο διμερές γράφημα, αφού εξεταστεί για MAX_FLOW στην leda, μετατρέπεται σε αντίστοιχο γράφημα στην boost και μετά τρέχει ο δικός μας αλγόριθμος.

5 Υλοποίηση Boost

Η υλοποίηση της Boost έγινε ακολουθώντας τον ψευδοκώδικα που δώθηκε από το paper.

5.1 Αρχικοποίηση

Ο αλγόριθμος ξεκινάει με το βήμα της αρχικοποίησης, στο οποίο θέτουμε αρχικά το id του κάθε κόμβου, το excess του και το ύψος του. Έπειτα θέτουμε το ύψος του S σε $2N_1+1$ σύμφωνα με το paper. Στην συνέχεια "σπρώχνουμε" από το S το μέγιστο residual capacity (θεωρούμε ότι το S έχει άπειρο excess). Με τον όρο "σπρώχνουμε", εννοούμε τον μηδενισμό του στοιχείου residual των ακμών που είναι εξερχόμενες από το S καθώς και την μεγιστοποίησή του από την "ανάποδη ακμή". Θέτουμε επίσης το excess του target ίσο με το μέγιστο capacity. Excess εννοούμε το υπόλοιπο μετά από σπρώξιμο το οποίο ακόμα δεν έχει διοχετευθεί αλλού. Τέλος, εκτελούμε ανάποδο BFS, από τον T πρός τους υπόλοιπους κόμβους για να ορίσουμε το ύψος τους σε σχέση με το T.

5.2 Κύριο Κομμάτι

Αρχικά τα επόμενα βήματα περικλείονται σε μία while loop η οποία ως συνθήκη έχει το να μην υπάρχει κόμβος με στοιχείο excess διάφορο του μηδενός.

Το οποίο ελέγχεται από μία συνάρτηση με όνομα overFlowVertex ή οποία εξετάζει όλους τους κόμβους του γράφου και επιστρέφει τον πρώτο κόμβο που έχει excess. Έπειτα, ελέγχουμε αν ο συγκεκριμένος κόμβος έχει ένα πρώτο admissable edge.Admissable edge είναι μία ακμή η οποία είναι εξερχόμενη, έχει residual διάφορο του μηδενός και το ύψος του κόμβου στόχου είναι μικρότερο από το ύψος της πηγής. Αν βρεθεί, ελέγχουμε αν ο στόχος του πρώτου admissable edge έχει και αυτός ένα δεύτερο admissable edge 2. Αν ναι, αυτό σημαίνει ότι μπορούμε να κάνουμε ένα push μήκους 2, το οποίο είναι η βασική ιδέα αυτού του αλγορίθμου. Με άλλα λόγια, μειώνουμε το residual των ακμών της διαδρομής που ακολουθεί το push, αυξάνουμε τις ανάποδες ακμές, μειώνουμε το excess του αρχικού κόμβου και αυξάνουμε του τελικού. Είναι σημαντικό να αναφέρουμε ότι ο ενδιάμεσος κόμβος δεν δέχεται αλλαγές στο excess του. Σε κάθε περίπτωση που ελέγχουμε για ένα admissable edge, αν δεν βρεθεί κανένα, τότε αυξάνουμε το ύψος του κόμβου κατά το ελάχιστο ύψος των γειτόνων συν ένα. Μία περίπτωση για την οποία δεν βρήκαμε κάτι ήταν η περίπτωση το πρώτο node μετά τον S να μην μπορεί να σπρώξει πουθενά στους γείτονές του γιατί έχει γεμίσει το capacity των εξερχόμενων. Υποθέσαμε ότι ό,τι Excess μένει με αυτό τον τρόπο, θα επιστραφεί στο S με push μήκους ένα. Σε κάθε επανάληψη, ανεξαρτήτως του βήματος τυπώνουμε τον γράφο σε μορφή DOT του εργαλείου graphviz. Όταν τελειώσει η επανάληψη παίρνουμε μετρήσεις για τον χρόνο που χρειάστηκε και σημειώνουμε το excess του κόμβου T το οποίο είναι και το maximum flow.

6 Υλοποίηση LEDA

Όσον αφορά την υλοποίηση της LEDA, δημιουργούμε τον γράφο στο ίδιο αρχείο που καλούμε και την συνάρτηση της MAX_FLOW. Πλέον ο γράφος που δημιουργείται είναι τυχαίος με την χρήση της συνάρτησης complete_bigraph της Leda. Έπειτα θέτουμε το capacity της κάθε ακμής ίση με έναν τυχαίο αριθμό μεταξύ 0 και 100. Τέλος αρχικοποιούμε τους μετρητές χρόνου και μετά όπως στο παράδειγμα του documentation της LEDA κάνουμε χρήση της συνάρτησης MAX_FLOW για να βρούμε την μέγιστη ροή του γράφου και εκτυπώνουμε τον τελικό χρόνο που απαιτήθηκε για την διεκπεραίωση του αλγορίθμου.

7 Εκτέλεση Προγράμματος

Για να εκτελεστεί πρέπει να εισέλθετε στον φάκελο και να χρησιμοποιήσετε το Makefile που παρέχεται με τις εντολές make και make run. Το main.cpp

είναι η υλοποίηση του αλγορίθμου ενώ μέσα στον φάκελο leda βρίσκεται το πρόγραμμα που τρέχει τον αντίστοιχο αλγόριθμο για την leda και έχει υλοποιηθεί ως .h βιβλιοθήκη για να γίνει include στην main.cpp και να τρέξουν και οι δύο υλοποιήσεις. Στον φάκελο boosthelper περιέχεται μία .h που περιέχει τον αλγόριθμο για την υλοποίηση ενός square graph σε boost όπως αυτός που ζητήθηκε στην πρώτη εργασία, καθώς αποτελεί διμερές γράφημα και χρησιμοποιήθηκε για την εξαγωγή των δεδομένων.
Να σημειωθεί ότι ο αλγόριθμός τρέχει μέσα και μερικά print statements για την οπτικοποίηση που ζητήθηκε όταν του δωθεί τέτοια εντολή μέσω της αρχικής οθόνης (logging = 1). Συγκεκριμένα μετά από κάθε εξέταση ενός ενεργού κόμβου εκτυπώνει την τωρινή μορφή του γράφου. Η μορφή αυτή είναι αναπαράσταση σε dot language του graphviz. Μπορείτε να δείτε τους γράφους κάνοντας copy paste τον dot κώδικα εδώ.

8 Δεδομένα

N	LEDA	BOOST
5	52	9
10	55	33
40	84	65
100	100	268
500	336	1200
1000	450	11445
2000	953	48239
5000	2933	349519
10000	5755	1619722

Οι αριθμοί που βλέπουμε στην αριστερή στήλη είναι το πλήθος των τετραγώνων που δημιουργήθηκαν, ενώ οι υπόλοιπες στήλες είναι ο χρόνος σε microseconds που χρειάστηκε η κάθε υλοποίηση για να υπολογίσει την μέγιστη ροή. Παρατηρούμε ότι η LEDA είναι πιο γρήγορη σε σύγκριση με την δικιά μας υλοποιήση σε Boost, κάτι το οποίο είναι λογικό καθώς η LEDA αποτελεί εμπορικό προϊόν κατασκευασμένο από ειδήμονες του πεδίου. Τα δεδομένα χρόνου καταγράφηκαν χωρίς εκτυπώσεις των μορφών graphviz για την boost και καταγράφηκαν στον Διογένη της Σχολής.

9 Νέα Δεδομένα

A	B	LEDA	BOOST
2	3	58	5
2	10	66	9
5	20	82	205
10	20	128	3002
50	100	1146	8856
100	300	10542	683964
200	500	32827	9213121
500	100	15804	264278
1000	100	31630	608812

Παρατηρούμε περίπου τα ίδια αποτελέσματα με πριν με την μόνη διαφορά ότι αφού πλέον μπορούμε να ελέγξουμε το πλήθος των κόμβων στο A Σετ καθώς και στο B, μπορούμε να δούμε την αποδοτικότητα του αλγορίθμου όντως να είναι καλύτερη όταν το B είναι αρκετά μικρότερο του A. Επίσης όπως πρίν, η υλοποιήση της Leda είναι σημαντικά πιο γρήγορη.