Αναφορά Υλοποίησης Τελικής Άσκησης

Στοιχεία Φοιτητή

Ονοματεπώνυμο: Δημήτριος Κωστορρίζος

Αριθμός Μητρώου: 1054419

Έτος Σπουδών: Ε’

Email Επικοινωνίας: kostorrizos@ceid.upatras.gr

Θέμα Τελικής Άσκησης: Ευρετικές μέθοδοι εύρεσης συντομότερων διαδρομών ALT

Περιγραφή Υλοποίησης

Αρχικά, αποφάσισα τι είδους γραφήματα θα χρησιμοποιήσω για να αξιολογήσω την συνάρτηση ALT. Επέλεξα να χρησιμοποιήσω τυχαία γραφήματα, γραφήματα τύπου πλέγματος(grid) και πλήρη γραφήματα. Για αν δημιουτγήσω τα γραφήματα αυτά χρησιμοποίησα τις συναρτήσεις δημιουργίας γραφημάτων της LEDA, random_graph, grid_graph και complete_graph.

Για τα προσανατολισμένα γραφήματα σε LEDA, επέλεξα την κλάση graph() με διπλής ακρίβειας αριθμητική τιμή(double) ως κόστη ακμών.

Για τα προσανατολισμένα γραφήματα σε Boost, επέλεξα την κλάση adjacency_list() με διπλής ακρίβειας αριθμητική τιμή(double) ως κόστη ακμών. Η υλοποίηση της ιδιότητας «κόστος ακμής», έγινε χρησιμοποιώντας την εντολή: **property<edge_weight_t, double>** . Για την υλοποίηση των κόμβων του γραφήματος, χρησιμοποιήθηκε το vertex_descriptor του adjacency_list, ενώ για την υλοποίηση των ακμών του γραφήματος χρησιμοποιήθηκε το edge_descriptor. Προκειμένου να έχω την δυνατότητα να διατρέχω την Boost υλοποίηση των γραφημάτων, χρησιμοποίησα το edge_iterator ως iterator για να διατρέχω τις ακμές του γραφήματος, out_edge_iterator ως iterator για να διατρέχω τις εξερχόμενες ακμές ενός κόμβου του γραφήματος και vertex_iterator ως iterator για να διατρέχω τους κόμβους του γραφήματος. Τέλος, χρησιμοποιήθηκε η εντολή **property_map<DirectedGraph, edge_weight_t>** ώστε να ορισθεί η σχέση μεταξύ του γραφήματος και των τιμών κόστους για τις ακμές που περιέχει.

Αναφορά Υλοποίησης Τελικής Άσκησης

Για τα τυχαία γραφήματα, επέλεξα να μην περιέχουν παράλληλες, αντί-παράλληλες και αυτό-κυκλικές ακμές. Ως αριθμό ακμών στα τυχαία γραφήματα, όρισμα την τιμή N * (N - 1), όπου N ο αριθμός των κόμβων στο γράφημα, η οποία αποτελεί τον μέγιστο αριθμό ακμών.

Για τα πλήρη γραφήματα και τα γραφήματα τύπου πλέγματος, χρησιμοποιήθηκε η προκαθορισμένο υλοποίηση της συνάρτησης της LEDA.

Για το κόστος της κάθε ακμής, ορίστηκε μία τυχαία τιμή μεταξύ του 1 και του 101.

Οι μετρήσεις για τα προαναφερθέντα γραφήματα εκτελέστηκαν για τα εξής πλήθη κόμβων: 10, 100, 1000 κόμβοι. Προσπάθησα να εκτελέσω τις δοκιμές για 10000 και 100000 κόμβους, αλλά ο Διογένης δεν επέτρεπε την δέσμευση περαιτέρω μνήμης, οπότε και δεν εκτελέστηκαν μετρήσεις για τους συγκεκριμένους αριθμούς κόμβων.

Έχοντας δημιουργήσει τα γραφήματα, χρησιμοποίησα μία συνάρτηση για να «αντιγράψω» τα γραφήματα από την υλοποίηση σε LEDA, στην υλοποίηση σε Boost. Η συγκεκριμένη συνάρτηση δημιουργεί ένα γράφημα σε Boost με ίσο αριθμό κόμβων με το αντίστοιχο γράφημα σε LEDA. Έπειτα, κάθε ακμή του γραφήματος σε LEDA, δημιουργείται στην Boost υλοποίηση του γραφήματος, με βάση τον αρχικό κόμβο, τον τελικό κόμβο και το κόστος της ακμής. Μετά την ολοκλήρωση της αντιγραφής των ακμών, ο δείκτης της Boost υλοποίησης του γραφήματος ενημερώνεται ώστε να επιστραφεί ως out παράμετρος.

Για την υλοποίηση της συνάρτησης ALT σε Boost, χρησιμοποιήθηκε ως βάση η υλοποίηση της ευρετικής αναζήτησης A*. Αρχικά, ορίζονται τα κόστη των ακμών, καθώς και δύο χάρτες που περιέχουν ζευγάρια κόμβων-αριθμιτικών τιμών διπλής ακρίβειας(double). Ο πρώτος χάρτης θα περιέχει τα κόστη ανακάλυψης των κόμβων, ενώ ο δεύτερος θα περιέχει τα ευρετικά κόστη ανακάλυψης των κόμβων. Οι δύο αυτοί χάρτες αρχικοποιούνται στην μέγιστη αριθμητική τιμή διπλής ακρίβειας, DBL_MAX. Για την υλοποίηση της ουράς των «ανοιχτών» κόμβων χρησιμοποιήθηκε ο τύπος **std::vector** ενώ για την υλοποίηση των χαρτών ο τύπος **std::map**.

Έπειτα, ορίζονται οι αρχικές τιμές ευρετικού κόστους και κόστους ανακάλυψης για το αρχικό κόμβο και ο κόμβος αυτός τοποθετείται στο διάνυσμα «ανοιχτών» κόμβων. Μέχρις ότου το διάνυσμα των ανοιχτών κόμβων να είναι άδειο, γεγονός που δηλώνει ότι ο τελικός κόμβος δεν μπορεί να προσεγγιστεί, εκτελείται ο παρακάτω αλγόριθμος.

Αναφορά Υλοποίησης Τελικής Άσκησης

Επιλέγεται ο κόμβος με το μικρότερο ευρετικό κόστος. Ελέγχεται αν αυτός ο κόμβος είναι ο τελικός κόμβος. Αν είναι, τότε ο αλγόριθμος τερματίζει. Αν δεν είναι, ο συγκεκριμένος κόμβος αφαιρείται από το διάνυσμα «ανοιχτών» κόμβων. Έπειτα, ελέγχεται κάθε ακμή που έχει ως αρχή τον επιλεγμένο κόμβο. Αν το κόστος της δεδομένης ακμής συν το κόστος ανακάλυψης του επιλεγμένου κόμβου είναι μικρότερο από το κόστος ανακάλυψης του κόμβου στο πέρας της ακμής, ο κόμβος που βρίσκεται το πέρας της ακμής τοποθετείται στην λίστα «ανοιχτών» κόμβων αν δεν υπάρχει ήδη. Σε αυτό το σημείο ενημερώνονται και οι εγγραφές των χαρτών για το συγκεκριμένο κόμβο, το κόστος ανακάλυψης του ορίζεται ως το κόστος ανακάλυψης του προηγούμενο κόμβου προσαυξημένο από το κόστος της επιλεγμένης ακμής και υπολογίζεται το ευρετικό κόστος του.

Για την υλοποίηση της συνάρτησης υπολογισμού ευρετικού κόστους, έγινε προσπάθεια να χρησιμοποιηθεί η τεχνική των Landmarks και της τριγωνικής ανισότητας. Μελέτησα την βιβλιογραφική αναφορά και κατάληξα στα εξής συμπεράσματα:

- Ως Landmark, θα όριζα ορισμένους κόμβους του γραφήματος.
- Κάθε Landmark θα πρέπει να ικανοποιήσει την σχέση της τριγωνικής ανισότητας για τον εκάστοτε κόμβο, δηλαδή η Ευκλείδεια απόσταση του επιλεγμένου κόμβου θα πρέπει να είναι μικρότερη ή ίση από την απόσταση του επιλεγμένου κόμβου από τον γειτονικό του κόμβο.
- Κάθε Landmark θα πρέπει να απέχει το πολύ από τον επιλεγμένο κόμβο όσο ο επιλεγμένος κόμβος από τον γειτονικό του.
- Τα Landmark πρέπει να προσαρμόζονται ανάλογα την δομή του γραφήματος.

Προσπάθησα να χρησιμοποιήσω τα παραπάνω συμπεράσματα ως την βάση για την υλοποίηση της συνάρτησης υπολογισμού τους ευρετικού κόστος. Ωστόσο, λόγω έλλειψης εμπειρίας με την συγκεκριμένη διαδικασία, η συνάρτηση επέστρεφε λανθασμένα ευρετικά κόστη, με αποτέλεσμα να οδηγεί την αναζήτηση A* σε ατέρμων επαναλήψεις. Έχω υλοποιήσει την συνάρτηση αυτή στο παρελθόν, δεδομένης της συνάρτησης υπολογισμού του ευρετικού κόστους. Για την υλοποίηση που σας παραδίδω, έχει χρησιμοποιηθεί η συνάρτηση υπολογισμού τους ευρετικού κόστους, η απόλυτη τιμή της Manhattan απόστασης τον κόμβων στην adjacency_list που έχει χρησιμοποιηθεί για την υλοποίηση του προσανατολισμένου γραφήματος σε Boost.

Κατά την εκτέλεση του προγράμματος, ο χρήστης μπορεί να επιλέξει μεταξύ των επιλογών:

- grid για γραφήματα τύπου πλέγματος
- complete για πλήρη γραφήματα
- random για τυχαία γραφήματα

Αναφορά Υλοποίησης Τελικής Άσκησης

Επιλέγοντας μία από της παραπάνω επιλογές, ο χρήστης εισάγει τον αριθμό των κόμβων που επιθυμεί να έχει το γράφημα. Αν η επιλογή του είναι τυχαίο γράφημα, τότε χρησιμοποιείται η συνάρτηση Make_Connected() της LEDA προκειμένου να γίνει συνεκτικό το τυχαίο γράφημα πριν αντιγραφεί στην βιβλιοθήκη Boost.

Έπειτα, αρχικοποιούνται οι λίστες με τους κόμβους, τις ακμές και τα κόστη των ακμών προκειμένου να χρησιμοποιηθούν στην συνάρτηση της LEDA SHORTEST_PATH_T(). Η συγκεκριμένη συνάρτηση υπολογίζει τα κόστη τις συντομότερες διαδρομές από τον αρχικό κόμβο προς όλου τους υπόλοιπους στο γράφημα. Ο αρχικός και ο τελικό κόμβος για την αναζήτηση, επιλέγονται τυχαία από το σύνολο κόμβων που περιέχονται στο γράφημα. Έχοντας υπολογίσει τα διανύσματα που περιέχουν τα μονοπάτια από τον αρχικό κόμβο προς όλους τους κόμβους, καλείται η συνάρτηση της LEDA, Compute_Shortest_Path(), προκειμένου να υπολογιστεί το συνολικό κόστος τους μονοπατιού σε συνδυασμό με την λίστα των ακμών που εμπεριέχονται στο ελάχιστο μονοπάτι.

Η πειραματική αξιολόγηση εκτελέστηκε στο σύστημα Διογένης, καθώς η δωρεάν έκδοση της βιβλιοθήκης LEDA, δεν περιλαμβάνει τις συναρτήσεις που ανήκουν στην κατηγορία των Shortest_Path algorithms.

Από τις παρακάτω μετρήσεις, απουσιάζουν οι μετρήσεις για γραφήματα τύπου πλέγματος για την υλοποίηση σε LEDA, καθώς η υλοποίηση της SHORTEST_PATH_T() στην βιβλιοθήκη LEDA, δεν υποστηρίζει γραφήματα τύπου grid.

Για τον υπολογισμό του χρόνου εκτέλεσης της μεθόδου χρησιμοποιήθηκε η συνάρτηση used_time() της βιβλιοθήκης LEDA.

Για το compile του πηγαίου κώδικα, χρησιμοποιήθηκε το Makefile που έχει παρουσιαστεί στο μάθημα, εισάγοντας τις επιπλέον εξαρτήσεις για την βιβλιοθήκη της Boost. Για την εκτέλεση της πειραματικής αξιολόγησης, στον εκτελέσιμο κώδικα έχει εφαρμοστεί βελτιστοποίηση τάξης O3. Ως compiler έχει χρησιμοποιηθεί η έκδοση του g++ που είναι εγκατεστημένη στον Διογένη. Η έκδοση της C++ που χρησιμοποιήθηκε είναι η C++ 98, που είναι εγκατεστημένη στον Διογένη.

Παρακάτω παρουσιάζονται οι μετρήσεις για τα τυχαία και μη τυχαία γραφήματα εισόδου. Θεώρησα ότι η υλοποίηση της συνάρτησης ALT στην βιβλιοθήκη LEDA, αντιστοιχεί στην συνάρτηση Compute_Shortest_Path. Οι παρακάτω μετρήσεις έχουν υπολογιστεί ως η μέση τιμή δέκα εκτελέσεων για το κάθε είδος γραφήματος και πλήθος κόμβων.

Αναφορά Υλοποίησης Τελικής Άσκησης

Πειραματικές Μετρήσεις

Grid Graphs

Αριθμός Κόμβων / Μέθοδος	User Defined ALT	Leda Compute Shortest Path
10	0 seconds	-
100	0 seconds	-
1000	1.78 seconds	-

Complete Graphs

Αριθμός Κόμβων / Μέθοδος	User Defined ALT	Leda Compute Shortest Path
10	0 seconds	0 seconds
100	0 seconds	0 seconds
1000	10.34 seconds	0.11 seconds

Random Graphs

Αριθμός Κόμβων / Μέθοδος	User Defined ALT	Leda Compute Shortest Path
Κόμβοι 10, Ακμές 90	0 seconds	0 seconds
Κόμβοι 100, Ακμές 9.900	0 seconds	0 seconds
Κόμβοι 1000, Ακμές 909.000	368.91 seconds	0.29001 seconds