Τεχνολογίες Υλοποίησης Αλγορίθμων

2η εργασία

Ανάλυση Κώδικα

Αλγόριθμος Dijkstra-SP (void DijkstraSP(BGraph &BG, Vertex s, Vertex t)):

Ξεκινώντας με μια συνθήκη if, ελέγχουμε αν η υπάρχει η κορυφή t. Αν υπάρχει συνεχίζουμε κανονικά με τον αλγόριθμο, αλλιώς εμφανίζεται μήνυμα. Στην συνέχεια δημιουργούμε δύο heaps. Ένα για την αποθήκευση των κορυφών και ένα για την διαχείρισή τους. Αρχικοποιούμε την απόσταση της αρχικής κορυφής s με μηδέν και την εισάγουμε στο heap των κορυφών. Με μια συνθήκη while, όσο η ουρά έχει μέσα κορυφές και δεν έχει βρεθεί ακόμα η κορυφή t, τότε βρίσκει την κορυφή με την μικρότερη απόσταση και την προσθέτει στην ουρά. Αν η κοντινότερη αυτή κορυφή είναι η t, τότε τελειώνει ο αλγόριθμος, αλλιώς παίρνουμε πληροφορίες για κάθε εξερχόμενη ακμή και ελέγχουμε αν ισχύει η τριγωνική ανισότητα. Αν βλέπουμε την κορυφή για πρώτη φορά τότε την εισάγουμε στην ουρά, αλλιώς ενημερώνουμε τις αποστάσεις.

Αλγόριθμος A* (void aStar(Vertex s, Vertex t, BGraph &BG)):

Αρχικά, με 2 for loops γίνεται η αρχικοποίηση των αποστάσεων και των κοστών των κορυφών. Έπειτα, γίνεται η κλήση της συνάρτησης ht_process(s,t,BG), η οποία αποτελεί την προεπεξεργασία του αλγορίθμου A* και εξηγείται παρακάτω. Μετά, ξεκινά η μέτρηση χρόνου, καλούμε την συνάρτηση DijkstraSP και παίρνουμε τον χρόνο εκτέλεσης του αλγορίθμου.

Προεπεξεργασία A* (void ht_process(Vertex s, Vertex t, BGraph &BG)):

Χωρίζεται σε 2 μέρη κατά κύρια βάση. Ένα για τον κόμβο L1 και ένα για τον κόμβο L2. Αρχικά, επιλέγουμε ως L1 έναν τυχαίο κόμβο και αρχικοποιούμε αποστάσεις με άπειρο. Ύστερα, καλούμε την συνάρτηση DijkstraS για να βρούμε τις αποστάσεις από την L1 και αποθηκεύουμε τις αποστάσεις που βρέθηκαν σε ένα vector. Ακολουθούμε την ίδια διαδικασία για να βρούμε και τις αποστάσεις προς την L1. Όμοια διαδικασία ακολουθούμε και για την κορυφή L2, βρίσκοντας με τον αλγόριθμο Dijkstra τις αποστάσεις από και προς αυτήν. Τέλος, βρίσκουμε την τιμή της ht.

Αντιγραφή Γραφήματος (void copy_graph(const LGraph &LG, BGraph &BG, edge_array<int> &l_weight, node_array<NodeInfo>&nodeInfo)):

Αρχικά, δημιουργούμε ένα array από κορυφές για να τις αποθηκεύσουμε. Μετά, για κάθε κορυφή την αντιγράφουμε στο καινούριο array και προσθέτουμε και τις αποστάσεις. Όμοια, κάθε ακμή την αντιγράφουμε στο νέο γράφημα.

Δημιουργία Γραφήματος (void randomGraph(BGraph &BG, int a, int b)):

Το γράφημα που δημιουργείται είναι ένα τυχαίο γράφημα. Συνδέουμε τις κορυφές με ακμές και αρχικοποιούμε τα κόστη των ακμών. Ακόμα, αρχικοποιούμε τις αποστάσεις στο άπειρο και έπειτα καλούμε την συνάρτηση αντιγραφής copy_graph, μετατρέποντας το γράφημα από boost σε leda. Τέλος, αποθηκεύουμε τα νέα βάρη.

Main():

Στην συνάρτηση main() αρχικοποιούμε τις αποστάσεις και τα βάρη και καλούμε μια μια τις συναρτήσεις που υλοποιήθηκαν.

Σημειώσεις:

- Το grid Graph δεν έχει υλοποιηθεί
- Λόγω segmentation fault ο κώδικας δεν τρέχει, οπότε δεν μπορεί να υλοποιηθεί η σύγκριση των αλγορίθμων και να εξαχθούν διαγράμματα.
- Όπου οι αποστάσεις τείνουν στο άπειρο έχω θέσει μια τυχαία μεγάλη τιμή (1.000.000).