Σωτήριος Παναγιώτου, 4456

Διαχείριση Σύνθετων Δεδομένων

Assignment 3

Μέρος 1: Αποθήκευση και δεικτοδότηση στη μνήμη

Input: 2 txt αρχεια cal.cnode.txt και cal.cedge.txt hard-set στον κωδικα

Παράδειγμα εκτέλεσης: python3 ask1.py

Output: 1 txt αρχειο ask1out.txt που περιεχει όλα τα nodes μαζι με τους γειτονες τους

Εξηγηση κωδικα:

Διαβαζουμε τα 2 txt αρχεια και τα αποθηκευουμε σε λιστες.

Επειτα, για κάθε κομβο αποθηκευουμε τους γειτονες του μεσα σε ένα dictionary “dict”

Πχ: dict[10] = [9, 0.01294, 11, 0.013238]
Δηλαδη το node 10 εχει γειτονες το 9, με αποσταση 0.01294 και το 11 με αποσταση 0.013238

Ετσι, με άλλη μια for loop, φτιαχνουμε το αρχειο ask1out.txt «κολλωντας» τα πρωτα 3 στοιχεια του cal.cnode.txt (node id, x, y) με τους γειτονες που βρηκαμε στο dictionary “dict”

Ετσι για παραδειγμα για το node 10 στο ask1out.txt θα εχουμε «10 -122.542992 41.903084 9 0.01294 11 0.013238»

Μέρος 2: Dijkstra και A*

Input: 2 txt αρχεια cal.cnode.txt και cal.cedge.txt hard-set στον κωδικα, 2 node ids σαν arguments στο command line

Παράδειγμα εκτέλεσης: python3 ask2.py 1 10

Output: Μονοπατι, length, distance & iterations των αλγοριθμων Dijkstra & Astar

Εξηγηση κωδικα:

Φτιαχνουμε το graph από τα 2 txt input αρχεια.

Για τον Dijkstra, του δινουμε το graph, το βαζει σε ένα set και μετα αρχιζει την αναζητηση. Ανανεωνει το distance οποτε χρειαζεται μεσα σε dictionary και στο τελος, μολις βρεθει ο goal node επιστρεφει μια λιστα με το dictionary “prev”, και 2 αριθμους: το counter και το distance.

Ακριβως η ιδια λογικη ακολουθειται και στον Astar (copy paste του dijkstra), με την μονη διαφορα ότι αυτή τη φορα εκει που υπολογιζουμε το distance κανουμε χρηση heuristic

συναρτησης, στην συγκεκριμενη περιπτωση χρησιμοποιουμε την ευκλιδεια αποσταση μεταξυ 2 κομβων, που την υπολογιζουμε με τις συντεταγμενες των κομβων αυτων.

Το dictionary “prev” μας δινει τελικα το shortest path του κάθε αλγοριθμου. Ξεκιναμε αναποδα. Ξεκιναμε από το target node και βλεπουμε ποιο είναι το προηγουμενο node στο path. Ετσι φτανουμε τελικα στο starting node και βρηκαμε το path.
Για παραδειγμα:
target node = 10
prev[10] = 9. Prev[9]=37. Prev[37]=38 … prev[0]=1

Και ετσι εχουμε πως το μονοπατι είναι reversed([10,9,37,38…0,1])

Το counter είναι ο αριθμος των iterations που εκανε κάθε αλγοριθμος.
Και τελος το distance είναι το distance από το start node μεχρι το goal node.

Μέρος 3: Βέλτιστο σημείο συνάντησης

Input: 2 txt αρχεια cal.cnode.txt και cal.cedge.txt hard-set στον κωδικα, n node ids σαν arguments στο command line (n>=2)

Παράδειγμα εκτέλεσης: python3 ask3.py 1 6 10

Output: best meeting point id, Μονοπατι, distance του κάθε starting node μεχρι το best meeting point & shortest path distance

Εξηγηση κωδικα:
Φτιαχνουμε το graph από τα 2 txt input αρχεια.

Τροποποιω το dijkstra ετσι ώστε να μην είναι αναγκαστικο να δεχεται goal node (για να τρεξει «ολοκληρος» ο αλγοριθμος)

Ο Dijkstra τωρα επιστρεφει:

1. Αν εχουμε δωσει goal node: Έναν πινακα [prev, counter, distance], δηλαδη ακριβως τα ιδια που επεστρεφε και ο dijkstra στο part 2
2. Αν δεν εχουμε δωσει goal node: Έναν πινακα [prev, counter, sortedDistances] οπου sortedDistances είναι το dictionary “dist” που εχει αποθηκευμενα τα nodes με τα distances τους από τον starting node αλλα sorted από το μικροτερο στο μεγαλυτερο.

Για κάθε starting node καλω τον dijkstra χωρις goal node και αποθηκευω το result του σε μια λιστα tmpList.

Μετα για κάθε node που υπαρχει στο graph, παιρνω το max(dist(n1, m), dist(n2, m),..., dist(nv, m)) οπου n1,n2…nv είναι τα starting nodes. Αποθηκευω αυτό το max σε μια λιστα. Αυτή η λιστα θα εχει υπολιστες της μορφης [nodeId, maxDistance]

Ετσι με αυτή τη λιστα θα εχουμε υπολιστες/pairs με όλα τα nodes-maxDistance.

Για να βρουμε το best meeting point απλα παιρνουμε το nodeId που εχει το μικροτερο maxDistance.

Επειτα τυπωνουμε το best meeting point, το distance του το οποιο θα είναι και το shortest path distance

Και τελος καλουμε το dijkstra για κάθε starting point, αλλα αυτή τη φορα με goal node το best meeting point που βρηκαμε.