Τεχνητή Νοημοσύνη

Προγραμματιστικό Θέμα 1

Ονοματεπώνυμο: Σταύρος Σταύρου
Εξάμηνο: 7ο
Σχολή: Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
ΑΜ: 03115701

Πρόβλημα/Θέμα

Μας δίδονται σε αρχεία csv οι συντεταγμένες μιας αλληλουχίας δρόμων, οι συντεταγμένες που βρίσκεται ένας πελάτης και 11 ταξί. Μας ζητείται να φτιάξουμε μια υλοποίηση του αλγορίθμου A* σε Java, ώστε να υπολογίζεται το κοντινότερο ταξί στον πελάτη, όπως επίσης και η διαδρομή που θα πρέπει το ταξί να εκτελέσει μέχρι να φτάσει σε αυτόν.

Σχεδιασμός

Για υλοποίηση του αλγορίθμου χρειάστηκε η υλοποίηση 7 κλάσεων.

TaxiFinder:

Η κλάση TaxiFinder είναι η κλάση που περιέχει την main. Σε αυτήν δημιουργείται ένα αντικείμενο της κλάσης Astar το οποίο λύνει το πρόβλημα. Επίσης, στη μέθοδο main γίνεται η δημιουργία των 2 αρχείων μας “result.txt” και “result.kml” τα οποία περιέχουν την λύση του προβλήματός μας.

Astar:

Η κλάση Astar είναι η κυριότερη κλάση της υλοποίησής μας. Συγκεκριμένα, περιέχει 2 μεθόδους (void readInputs() και LinkedList<TaxiRoute> solve()) και 7 πεδία (LinkedList<Taxi> taxis, Node goal, HashMap<Integer, Odos> streets, HashMap<Node, ArrayList<Integer>> nodeStreets, HashMap<Node, Node> simia, LinkedList<Node> nodes, LinkedList<TaxiRoute> liseis).

Με την κλήση της μεθόδου readInputs() το αντικείμενο διαβάζει την είσοδο από τα 3 αρχεία και γεμίζει τα 6 από τα 7 πεδία του με την είσοδο κατάλληλα. Συγκεκριμένα, ο goal είναι πλέον ένας Node με τις συντεταγμένες του πελάτη, η Taxis είναι μια λίστα που περιέχει πληροφορίες για όλα τα ταξί και η nodes περιέχει πληροφορίες για όλους τους δοθέντες κόμβους. Τέλος, τα 3 HashMap χρησιμοποιούνται για γρήγορη πρόσβαση στα δεδομένα μας. Με τον simia έχουμε γρήγορη πρόσβαση σε κάθε κόμβο στο σύμπαν του προβλήματος, ενώ με τους nodeStreets και streets, έχουμε άμεση πρόσβαση στα σημεία που αποτελούν κάθε οδό, αλλά και στις οδούς που ανήκει κάθε σημείο, με μία N:M αντιστοιχία.

Με την solve() εκτελείται ο αλγόριθμός μας και επιστρέφουμε μια λίστα από TaxiRoute που περιέχει το συντομότερο μονοπάτι για καθένα από τα ταξί. Ο αλγόριθμος εκτελείται ως εξής:

1. Θέσε ως current ένα νέο αντικείμενο της κλάσης Taxidi που έχει ως αρχικές συντεταγμένες τις συντεταγμένες του κάθε ταξί και τα υπόλοιπα πεδία, ανάλογα με το που βρίσκεται ο πελάτης. Πρόσθεσε το current σε μια συνδεδεμένη λίστα metopo, ως μέτωπο αναζήτησης.
2. Ταξινόμησε τη λίστα metopo και πάρε το Taxidi με τη χαμηλότερη ευρυστική (απόσταση που καλύφθηκε + εκτίμηση για τον στόχο). Αν το Taxidi που πήρες έχει τις ίδιες συντεταγμένες με τον κόμβο στόχο ανακοίνωσε επιτυχία και βάλε τη διαδρομή στη liseis.
3. Με χρήση των HashMap βρες τους κόμβους που μπορεί να ταξιδέψει μετά το ταξί και πρόσθεσε ισάριθμα αντικείμενα Taxidi στη λίστα metopo.
4. Επανέλαβε το βήμα 2.

Node:
Η κλάση Node περιέχει μόνο 2 πεδία, τα X και Y, όπως δίνονται στα δεδομένα μας. Κάθε αντικείμενο της κλάσης υποδεικνύει ένα σημείο στο χάρτη. Ακόμη, πέραν των constructors, getters και getters για τα πεδία της κλάσης, έχουμε τις μεθόδους real findDistance(node n), void setCloser(LinkedList<Node> nodes) και boolean sameSpot(Node n). Με την setCloser γίνεται ανάθεση των τιμών X και Y του αντικειμένου στο κοντινότερο σημείο που περιέχεται στη λίστα nodes. Η κλάση αυτή χρησιμοποιείται για αντιστοίχιση του πελάτη και των ταξί σε σημεία που ανήκουν στο σύμπαν του προβλήματός μας. Η sameSpot δίνει true αν ο n βρίσκεται στην ίδια θέση με το αντικείμενο και false διαφορετικά. Τέλος, με την findDistance βρίσκουμε την απόσταση του αντικειμένου από το αντικείμενο n. Για εύρεση της απόστασης γίνεται χρήση της Haversine function.
Επίσης, επειδή χρησιμοποιούμε αντικείμενα της κλάσης Node σαν key σε HashMap γίνεται Override των equals και hashCode για να ανταποκρίνεται στις ανάγκες μας.

Taxi:
Η κλάση Taxi χρησιμοποιείται για αναπαράσταση των ταξί στην υλοποίησή μας. Κάνει extend την Node και το μόνο επιπλέον πεδίο που περιέχει είναι το int id, που αντιπροσωπεύει την ξεχωριστή ταυτότητα για κάθε ταξί.

Taxidi:
Η κλάση Taxidi χρησιμοποιείται για να κρατάει διαδρομές που μας οδήγησαν στον κάθε κόμβο. Κάνει extend τη Node και στα πεδία x και y περιέχεται σε κάθε στιγμιότυπο ο τελευταίος κόμβος που φτάσαμε. Ακόμη, κάνει Implement το interface Comparable, γιατί θέλουμε να έχουμε τη δυνατότητα να ταξινομούμε αντικείμενα της κλάσης αυτής για τις ανάγκες του αλγορίθμου μας και ως εκ τούτου γίνεται Override της μεθόδου int compareTo(Object obj).
Ακόμη περιέχονται τα πεδία distance (απόσταση που καλύφθηκε, μέχρι να φτάσουμε στον τρέχοντα κόμβο), estimation (εκτιμώμενη απόσταση μέχρι τον κόμβο-στόχο), heuristic (άθροισμα distance + estimation), goal (αναπαράσταση του κόμβου στόχου) και stathmoi (διαδρομή μέχρι την τρέχουσα θέση).
Η μόνη επιπλέον μέθοδος (σε σχέση με την Node) είναι η addNode(Node n), η οποία προσθέτει τον n στη διαδρομή που ακολουθήθηκε, κάνει τη θέση του n, την τρέχουσα θέση του αντικειμένου και ενημερώνει τα πεδία distance, estimation, heuristic.

Odos:
Η κλάση Odos περιέχει απλά μια ArrayList<Node> streetNodes. Κάθε id οδού, που περιέχεται στο αρχείο nodes.csv έχει ένα δικό του αντικείμενο της κλάσης αυτής και στην streetNodes περιέχονται όλα τα σημεία που την απαρτίζουν.

TaxiRoute:
Τα αντικείμενα της κλάσης TaxiRoute χρησιμοποιούνται για να αποθηκεύσουμε λύσεις που βρήκαμε. Περιέχει τα πεδία id (id του κάθε ταξί), distance (απόσταση που καλύφθηκε) και route (διαδρομή μέχρι το σημείο αυτό). Επειδή, θέλουμε να έχουμε τη δυνατότητα ταξινόμησης των αντικειμένων της κλάσης αυτής βάση του distance, η κλάση κάνει implement το interface Comparable και έχουμε Override της μεθόδου compareTo.

Μοντελοποίηση Δεδομένων/Παραδοχές:

Για την αποδοτική υλοποίηση του αλγορίθμου μας έχουμε κάνει μια σειρά παραδοχών:

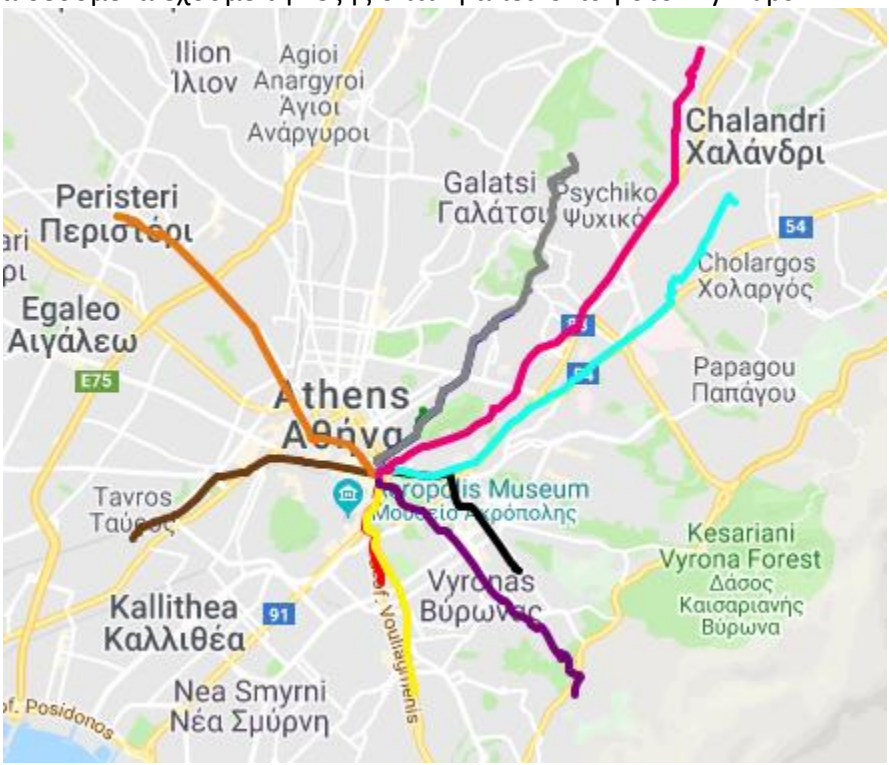
- Η θέση για κάθε πελάτη και ταξί αντιστοιχίζεται σε κάποια από τις διαθέσιμες στο αρχείο nodes.csv. Αυτό γίνεται, διότι θέλουμε να περιορίσουμε το πρόβλημα στα στοιχεία του αρχείου

nodes.csv, καθώς δεν έχουμε επαρκή στοιχεία για το πώς συνδέονται οι κόμβοι πελάτη και ταξί με τους υπόλοιπους.

- Για εύρεση αποστάσεων χρησιμοποιήθηκε η Haversine Formula (https://en.wikipedia.org/wiki/Haversine_formula). Επειδή, ωστόσο οι αποστάσεις είναι πολύ μικρές στο δοθέν σύμπαν (Αθήνα) θα μπορούσαμε να θεωρήσουμε τα σημεία X και Y ως σημεία στο επίπεδο και τις αποστάσεις μεταξύ των σημείων σαν αποστάσεις σημείων στο επίπεδο με πολύ καλή ακρίβεια στα αποτελέσματά μας.
- Δυστυχώς δεν καταφέραμε να υλοποιήσουμε τη δυνατότητα ο αλγόριθμός μας να μας δίνει περισσότερα από 1 μονοπάτια, εφόσον αυτά είναι ισοδύναμα. Αντ' αυτού επισυνάπτονται τα αποτελέσματα, ενός δεύτερου παραδείγματος με δίκες μας αρχικές συντεταγμένες για τον πελάτη και τα ταξί.
- Ως εκτιμήτρια απόστασης για κάθε τρέχων κόμβο, χρησιμοποιείται η ευκλείδεια απόσταση του από τον κόμβο-στόχο (με χρήση της Haversine Formula και πάλι). Ως εκ τούτου, η εκτιμήτρια αποτελεί πάντα υποεκτίμηση της πραγματικής απόστασης από τον κόμβο-στόχο και η λύση που προτείνεται για κάθε ταξί είναι η βέλτιστη.
- Στην τελική απεικόνιση στο MyMaps οι λύσεις απεικονίζονται με την εξής σειρά (Βέλτιστο Ταξί → Χείριστο Ταξί): Πράσινο, Κόκκινο, Μαύρο, Κίτρινο, Μπλε, Καφέ, Μοβ, Πορτοκαλί, Γκρι, Γαλάζιο και Ροζ.

Αποτελέσματα:

Με τα δοθέντα δεδομένα έχουμε την εξής οπτική απεικόνιση στο myMaps:



Με τα δικά μας δεδομένα (client2.csv και taxis2.csv, τα οποία επισυνάπτονται), έχουμε τα εξής: