Τεχνητή Νοημοσύνη
Θέμα 1 - Αναφορά
7ο Εξάμηνο 2018
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών - ΕΜΠ

Βρούτση Νίκη 03115639
Κυριάκου Αθηνά 03117405

Γενικός Σχεδιασμός Συστήματος:

Σκοπός της συγκεκριμένης εργασίας ήταν η υλοποίηση του βασικού κορμού μιας ευφυούς υπηρεσίας εξυπηρέτησης πελατών ταξί. Συγκεκριμένα, χρησιμοποιώντας τον χάρτη που κατασκευάστηκε βάσει του αρχείου nodes.csv και των δοθέντων συντεταγμένων των ταξί (taxis.csv) και της τοποθεσίας του πελάτη (client.csv), υλοποιήθηκε αλγόριθμος παρόμοιος με τον A* για την επιλογή του ταξί που απέχει λιγότερο από τη θέση του πελάτη. Κύριο χαρακτηριστικό του αλγορίθμου που αναπτύχθηκε είναι η ικανότητά του να βρίσκει και να επιστρέφει όχι μόνο τη συντομότερη διαδρομή, αλλά ενδεχομένως περισσότερες αλλά ισοδύναμες από πλευράς κόστους διαδρομές.

1. Βασικά συστατικά της εργασίας:

Η υλοποίηση αποτελείται από τα παρακάτω 4 αρχεία κώδικα σε Java:

1. **Main.java:** η public class του project στην οποία δημιουργείται ο χάρτης σε μορφή γράφου με χρήση του nodes.csv
2. **Astar.java**: private class για την υλοποίηση της παραλλαγής του Astar
3. **FindGoal.java:** private class για την αντιστοίχιση των δοθέντων συντεταγμένων της θέσης του πελάτη και κάθε ταξί σε σημεία του χάρτη που δημιουργήθηκε. Συγκεκριμένα, εάν τα σημεία δεν υπάρχουν ήδη στον χάρτη, αντιστοιχίζονται στα κοντινότερα υπάρχοντα σε αυτά με χρήση της Havershine Formula
4. **Havershine.java**: private class που υλοποιεί τη συνάρτηση εκτίμησης απόστασης βάσει της Havershine Formula

Οι βασικές classes που δημιουργήθηκαν είναι:

1. **Coordinates**: private class για κάθε σημείο του χάρτη. Αποτελείται από τις συντεταγμένες x,y και τις τιμές f,g,h.
2. **Node**: private class που αποτελείται από δύο Arraylist με αντικείμενα τύπου Coordinates. Η Arraylist neighbors περιέχει τους γείτονες κάθε σημείου, όπως καθορίστηκαν από τη διαδικασία δημιουργίας του χάρτη και η Arraylist fathers από τους γονείς κάθε σημείου κατά την προσπέλαση του χάρτη από την παραλλαγή του Astar.

2. Προεπεξεργασία των δεδομένων:

Για τη δημιουργία του χάρτη χρησιμοποιήθηκαν τα **δεδομένα του αρχείου nodes.csv.** Για κάθε γεωγραφικό σημείο δόθηκαν οι συντεταγμένες x,y του και η διεύθυνσή του μέσω id (και προαιρετικά το ονόματός της).

Τα δεδομένα αυτά διαθέτουν τα παρακάτω χαρακτηριστικά:

1. Τα σημεία που ανήκουν στον ίδιο δρόμο βρίσκονται εμφανίζονται ομαδοποιημένα και ταξινομημένα.
2. Ένα σημείο μπορεί να εμφανίζεται περισσότερες από μία φορές στο αρχείο nodes.csv με διαφορετικό id, άρα σε διαφορετικό δρόμο. Τα σημεία με αυτή την ιδιότητα αποτελούν σημεία διασταύρωσης.

Βάσει αυτών των χαρακτηριστικών, η προεπεξεργασία των γεωγραφικών συντεταγμένων ώστε να αναπαρασταθεί ο χάρτης σε γράφο τον οποίο θα διατρέξουμε μετέπειτα με τον Astar έγινε με χρήση της δομής δεδομένων HashMap της Java. Κάθε κόμβος του γράφου, δηλαδή κάθε γεωγραφικό σημείο, έχει ως key ένα αντικείμενο τύπου Coordinates και ως value ένα αντικείμενο τύπου Nodes. Η διαδικασία που ακολουθήθηκε για κάθε σημείο που διαβάζεται από το αρχείο nodes.csv που αναπαρίσταται σε κόμβο (data,values) είναι η εξής:

- Αν το σημείο υπάρχει στο HashMap που έχει δημιουργηθεί μέχρι τώρα, θεωρώ το υπάρχον ως κόμβο (data, old_values). Συγκρίνω το id του σημείου που εξετάζω με το last_checked_addr του προηγούμενου σημείου με κόμβο (last_check_data, prev_values) που εξετάστηκε. Εάν είναι ίσα, και άρα τα σημεία ανήκουν στον ίδιο δρόμο, δημιουργώ ακμή ανάμεσά τους, προσθέτοντας το data στους neighbors του (last_check_data, prev_values) και το last_checked_data στους neighbors του (data, old_values).

- Αν το σημείο δεν υπάρχει, το προσθέτω στο HashMap και συγκρίνω το id του με το last_checked_addr. Αν είναι ίσα και άρα ανήκουν στον ίδιο δρόμο, δημιουργώ ακμή ανάμεσά τους προσθέτοντας το data στους neighbors του (last_check_data, prev_values) και το last_checked_data στους neighbors του (data, values).

- Ανανεώνω τις τιμές των μεταβλητών last_checked_data και last_checked_addr θέτοντας last_checked_data=data και last_checked_addr=id.

Έτσι, κάθε διακριτό γεωγραφικό σημείο εισάγεται στον χάρτη μία φορά και οι γείτονές του εισάγονται στην ArrayList του neighbors.

Επίσης, στο στάδιο της προεπεξεργασίας, αντιστοιχήθηκε η θέση του πελάτη από το αρχείο client.csv και κάθε ταξί από το αρχείο taxis.csv στα **κοντινότερα δυνατά σημεία του γράφου**. Η αντιστοίχιση αυτή έγινε από τη συνάρτηση findGoal της class FindGoal με χρήση της Havershine Formula.

3. Υλοποίηση του αλγορίθμου:

Ο αλγόριθμος που αναπτύχθηκε ακολουθεί τα ίδια βήματα με τον Astar με τη διαφορά να παρατηρείται όταν ελέγχουμε, για κάθε neighbor/successor του σημείου q που επιλέχθηκε από την OpenList, αν ανήκει ή όχι στην OpenList και χρησιμοποιώντας τη **Arraylist fathers** του. Συγκεκριμένα:

- Αν ο successor βρίσκεται ήδη στην OpenList με ίδια τιμή g (μετρική της απόστασης του σημείο από το αρχικό σημείο start) με τον εξεταζόμενο για εισαγωγή (και άρα έχουν την ίδια τιμή f αφού έχουν την ίδια τιμή h), τότε το σημείο q προστίθεται στη **fathers** του, καθώς από αυτόν τον πατέρα μπορεί να οδηγηθούμε σε άλλο ένα ελάχιστο μονοπάτι με το ελάχιστο κόστος.Στην υλοποίηση μας ,συγκεκριμένα, καθώς χρησιμοποιούμε μεταβλητές τύπου double με αποτέλεσμα η απόλυτη ισότητα των g να είναι σχεδόν απίθανη, θεωρούμε αποδεκτή την διαφορά εως και 1 μέτρο.
- Αν ο successor βρίσκεται ήδη στην OpenList με μεγαλύτερη τιμή g από τον εξεταζόμενο για εισαγωγή, τότε αδειάζουμε τη λίστα fathers του και εισάγουμε το q καθώς βρέθηκε τρόπος να οδηγηθούμε στον successor από μονοπάτι με μικρότερο κόστος από αυτά που είχαμε βρει προηγουμένως (κόμβοι που βρίσκονται στο πεδίο **fathers**).
- Αν ο successor δε βρίσκεται στην OpenList, τότε τον εισάγουμε σε αυτήν αφού πρώτα προσθέσουμε τον κόμβο q στους **fathers** του.

Οι **δομές δεδομένων** που χρησιμοποιήθηκαν είναι:

1. Για την Open List χρησιμοποιήθηκε η δομή **ArrayList** καθώς σε αντίθεση με ένα Array δίνει τη δυνατότητα δυναμικής δέσμευσης μνήμης και χρησιμοποιείται με στοιχεία αντικείμενα. Στην υλοποίησή μας, τα στοιχεία της είναι αντικείμενα της class Coordinates. Επίσης, μας δίνει τη δυνατότητα εύκολης ταξινόμησης μέσω της Collections.sort(), βήμα που είναι απαραίτητο για την επιλογή του σημείου που θα εξεταστεί από το μέτωπο αναζήτησης σε κάθε επανάληψη του αλγορίθμου μέχρι τον τερματισμό του.
2. Για την Closed List χρησιμοποιήθηκε μία υλοποίηση HashTable, το **HashSet** της Java. Εφόσον για κάθε στοιχείο της Open List, όταν ελέγχουμε τους neighbors/ successors του μας ενδιαφέρει απλώς αν υπάρχουν ή όχι στην Closed List για να τους παρακάμψουμε ή να τους εισάγουμε στην Open List, χρησιμοποιήθηκε αυτή η δομή δεδομένων καθώς μας επιτρέπει γρήγορη αναζήτηση της παρουσίας ή όχι στοιχείου σε O(1). Αποτελείται και αυτή από αντικείμενα τύπου Coordinates.

Όλα τα δυνατά μονοπάτια που προκύπτουν μετά την ολοκλήρωση του αλγορίθμου για κάθε ταξί προς τον πελάτη τυπώνονται στο αρχείο **output.txt** που δημιουργείται.

Για τη **συνάρτηση εκτίμησης απόστασης** χρησιμοποιήθηκε η ευθεία απόσταση σημείου από σημείο εξετάζοντας κάθε φορά την απόσταση του εκάστοτε κόμβου neighbor/successor από το σημείο στόχο που είναι η θέση του πελάτη. Η υλοποίηση στηρίζεται στην Haversine Formula που υπολογίζει την απόσταση μεταξύ γεωγραφικών συντεταγμένων.

Εκτέλεση του Συστήματος και Σχολιασμός Αποτελεσμάτων:

Η οπτική αναπαράσταση του χάρτη των διαδρομών βρίσκεται στο https://drive.google.com/open?id=1hzdJWB8B4N1u1s-TkXWNEkc_4y9eM1R9&usp=sharing

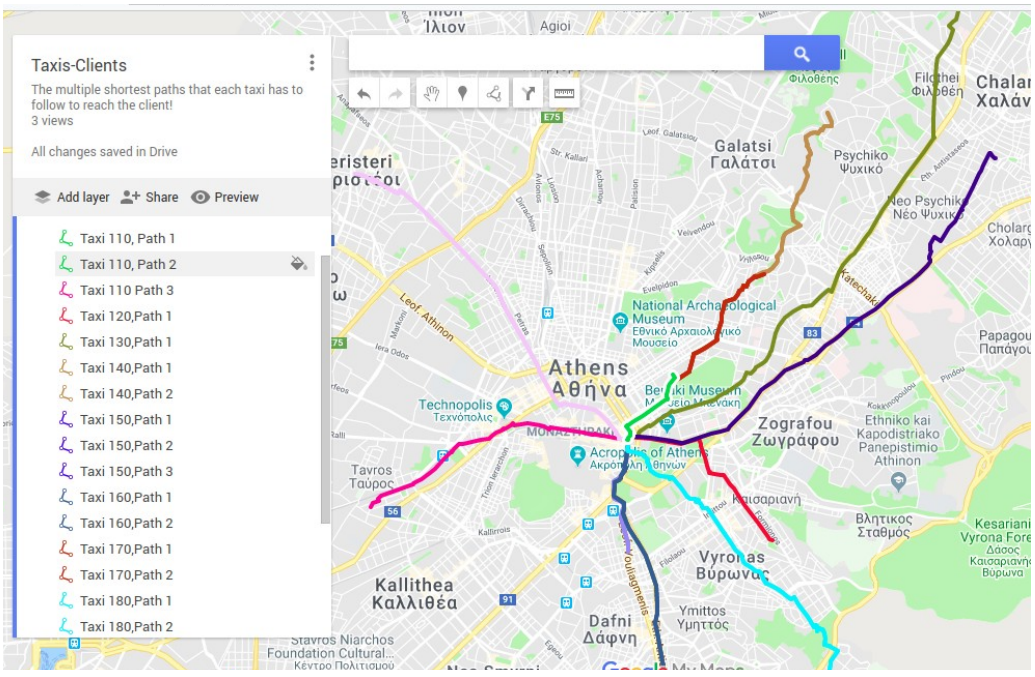
1. Εκτέλεση για τον χάρτη, την τοποθεσία πελάτη και τις τοποθεσίες ταξί που δίνονται:

Τοποθεσία πελάτη (client.csv):
23.733912,37.975687

Τοποθεσία κάθε ταξί (taxis.csv):
23.741587,37.984125,100
23.691133,37.967339,110
23.757838,37.962517,120
23.789114,38.032908,130
23.766993,38.018644,140
23.734153,37.961316,150
23.740078,37.945631,160
23.756566,37.997309,170
23.767763,37.945429,180
23.689228,38.010435,190
23.795044,38.012519,200

Το κοντινότερο ταξί που επιλέγεται για να εξυπηρετήσει τον πελάτη και εκτελώντας τη συντομότερη διαδρομή σε απόσταση 1.38 χιλιομέτρων, είναι το ταξί με **id 110** το οποίο όπως φαίνεται μπορεί να ακολουθήσει **δύο διαφορετικές διαδρομές ελάχιστου κόστους** για να φτάσει στον πελάτη βάσει των συντεταγμένων που δόθηκαν και με την παραδοχή ότι όλοι οι δρόμοι είναι διπλής κυκλοφορίας και προσπελάσιμοι από αυτοκίνητο.

Οι ισοδύναμες διαδρομές του σημειώνονται με πράσινο χρώμα στην οπτική αναπαράσταση μέσω του MyMaps (routesfinal.KML). Οι πολλαπλές ισοδύναμες διαδρομές ελάχιστου κόστος που βρίσκει ο αλγόριθμος για κάθε ταξί, εάν υπάρχουν, εμφανίζονται με αρίθμηση των paths.



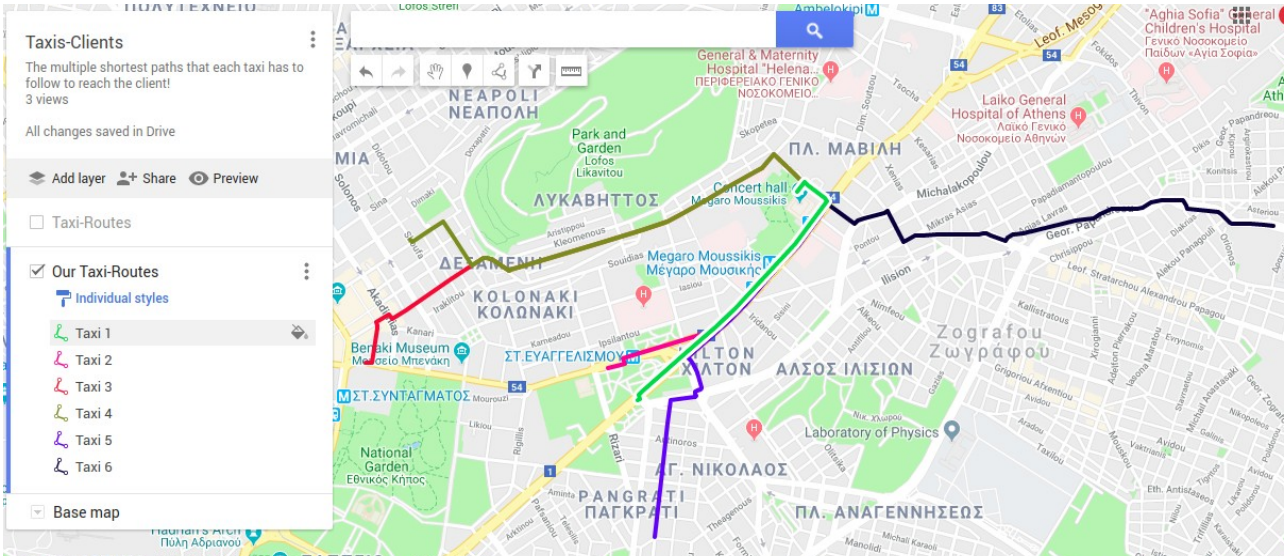
2. Εκτέλεση για τον χάρτη που δίνεται, και διαφορετικές επιλεγμένες τοποθεσίες πελάτη και ταξί:

Τοποθεσία πελάτη (ourclient.csv):
23.754123,37.981351

Τοποθεσία κάθε ταξί (ourtaxis.csv):
23.747458,37.974903,1
23.746342,37.976011,2
23.736564,37.976123,3
23.738262,37.980193,4
23.748210,37.970406,5
23.773307,37.980528,6

Το κοντινότερο ταξί που επιλέγεται για να εξυπηρετήσει τον πελάτη και εκτελώντας τη συντομότερη διαδρομή σε απόσταση 1.13 χιλιομέτρων, είναι το ταξί με **id 1** το οποίο όπως φαίνεται μπορεί να ακολουθήσει **μόνο μία διαδρομή ελάχιστου κόστους** για να φτάσει στον πελάτη.

Η διαδρομή που εκτελεί σημειώνεται με πράσινο χρώμα στην οπτική αναπαράσταση μέσω του MyMaps (MEGARO.KML). Σε αυτήν την περίπτωση ο αλγόριθμος δεν εμφανίζει εναλλακτικά μονοπάτια ελάχιστου κόστους για κανένα ταξί.



3. Σχολιασμός της δυνατότητας του αλγορίθμου να προτείνει όλες τις εναλλακτικές διαδρομές ελαχίστου κόστους για κάθε ταξί:

Όπως φαίνεται από την εκτέλεση για τα δεδομένα που δίνονται, ο αλγόριθμος έχει τη δυνατότητα να βρίσκει για κάθε ταξί **περισσότερες από μια ελαχίστου κόστους διαδρομές** και άρα ισοδύναμες **σε κάθε περίπτωση**. Αυτό επιτυγχάνεται λόγω του γεγονότος ότι κάθε σημείο εισάγεται στην OpenList μόνο μία φορά. Από την εισαγωγή του και μετά μέχρι να βγει από την OpenList, τροποποιείται μόνο η ArrayList fathers του. Σε αυτήν εισάγονται μόνο τα σημεία-πατεράδες που μπορούν να οδηγήσουν στο σημείο με το συντομότερο δυνατό μονοπάτι, μέχρι αυτή τη χρονική στιγμή εκτέλεσης του αλγορίθμου.

Οι διαδρομές που προκύπτουν είναι οι **βέλτιστες δυνατές** εφόσον ο αλγόριθμος σταματάει όταν από την OpenList αφαιρεθεί το σημείο-στόχος (η θέση του client) και άρα, καθώς από την OpenList κάθε φορά επιλέγουμε το σημείο με την μικρότερη τιμή g, έχουν εισαχθεί στην ArrayList fathers του σημείου-στόχου όλοι εκείνοι οι πατεράδες που μπορούν να οδηγήσουν σε αυτό με τις συντομότερες δυνατές διαδρομές. Η εκτύπωση όλων των βέλτιστων μονοπατιών γίνεται διαδοχικά για κάθε μονοπάτι με αναδρομή στην ArrayList fathers του σημείου-στόχου (οι πατεράδες των πατεράδων κλπ) μέχρι να φτάσουμε στο σημείο έναρξης του μονοπατιού (η αρχική θέση του ταξί).

Η προϋπόθεση για να είναι **στην πραγματικότητα** οι διαδρομές που προκύπτουν βέλτιστες, είναι οι θέσεις του ταξί και του πελάτη να βρίσκονται όσο το δυνατόν πιο κοντά γίνεται στα σημεία του σχηματιζόμενου χάρτη, δηλαδή να μην υπάρχει μεγάλη απόκλιση της πραγματικής τοποθεσίας από αυτή που αντιστοιχίζεται στον χάρτη.