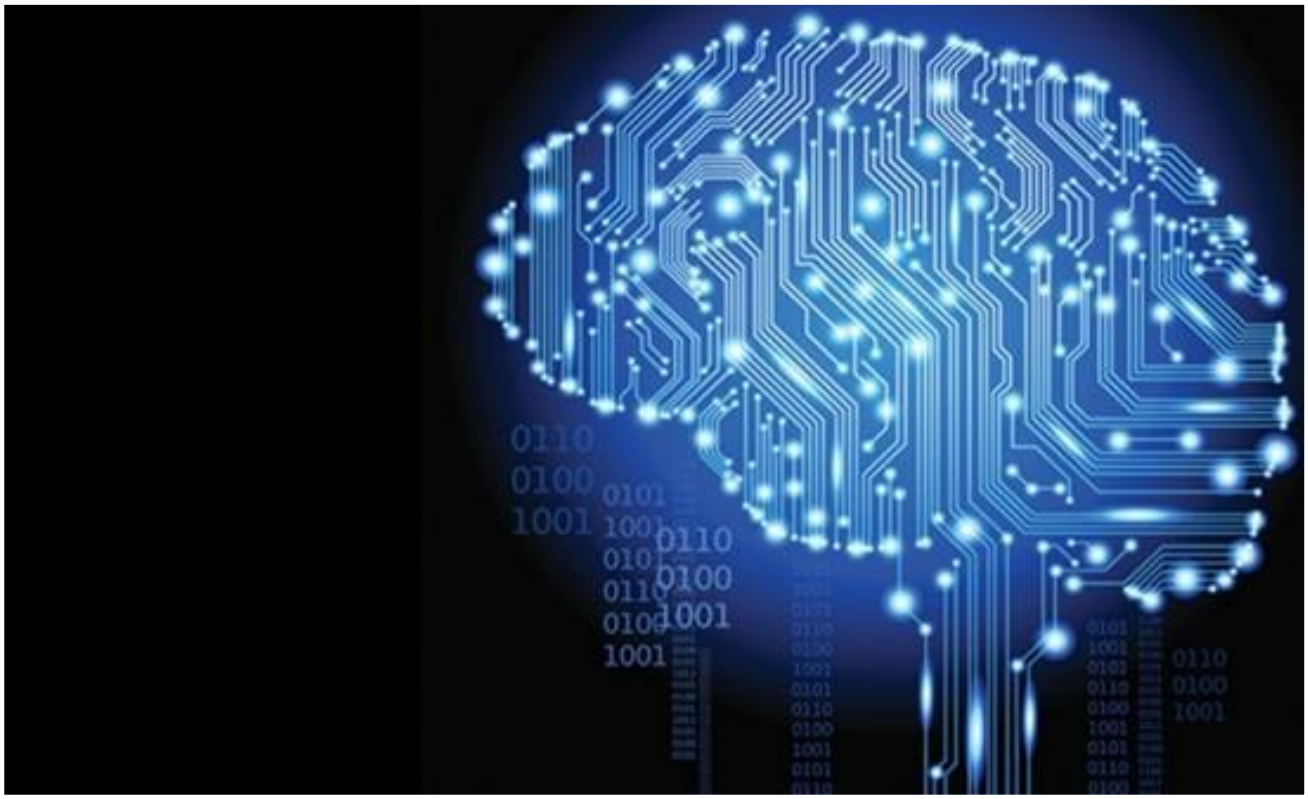
Τεχνητή Νοημοσύνη

ΘΕΜΑ 1: Υλοποίηση ευφυούς υπηρεσίας εξυπηρέτησης πελατών ταξί



Στοιχεία Φοιτητών:

Ονοματεπώνυμο	ΑΜ	Εξάμηνο	Email
Αντωνιάδης Παναγιώτης	03115009	7ο	el15009@central.ntua.gr
Μπαζώτης Νικόλαος	03115739	7ο	el15739@central.ntua.gr

Github url of the project: https://github.com/PanosAntoniadis/AI-ntua (ήταν private όσο ήταν ενεργή η εργασία)

- ## Περιγραφή εργασίας

Ο σκοπός της συγκεκριμένης εργασίας είναι η υλοποίηση του βασικού κορμού μιας ευφυούς υπηρεσίας εξυπηρέτησης πελατών ταξί. Συγκεκριμένα, θεωρούμε ότι υπάρχει ένας πελάτης που βρίσκεται σε μια ορισμένη τοποθεσία, ο οποίος διαθέτει κινητό τηλέφωνο με GPS και επιθυμεί να καλέσει ένα ταξί. Η υπηρεσία διαθέτει μια βάση δεδομένων με όλα τα διαθέσιμα ταξί και τη γεωγραφική θέση στην οποία βρίσκονται κάθε χρονική στιγμή, η οποία θεωρητικά θα πρέπει να ανανεώνεται συνεχώς. Η υπηρεσία εντοπίζει και να ειδοποιεί το ταξί που μπορεί να μεταβεί πιο γρήγορα στη θέση του πελάτη ώστε να τον εξυπηρετήσει. Για να το επιτύχει αυτό διαθέτει έναν χάρτη με πληροφορίες για την περιοχή ενδιαφέροντος. Επίσης θα έχει και την δυνατότητα να προτείνει βέλτιστες διαδρομές αλλά και ισοδύναμες αυτών αν εφόσον είναι εφικτό.

- ## Γενικός σχεδιασμός του συστήματος

Η υλοποίηση της εφαρμογής έγινε σε Java 1.8 και βασίστηκε στις αρχές του αντικειμενοστραφούς προγραμματισμού. Έχει δοκιμαστεί σε περιβάλλον linux ubuntu 18.04. Στη συνέχεια θα περιγραφεί η διαδικασία μοντελοποίησης των δεδομένων και η υλοποίηση του αλγορίθμου. Αρχικά, οι κλάσεις που δημιουργήθηκαν είναι οι ακόλουθες:

i. **Point:** Αναπαριστά ένα οποιοδήποτε σημείο στον χάρτη και ορίζεται από το γεωγραφικός μήκος και γεωγραφικό πλάτος του.

ii. **Street:** Αναπαριστά μία οδό στον χάρτη και ορίζεται από το id της οδού που είναι ένας μοναδικός κωδικός για κάθε μία και το όνομα της οδού (εφόσον αυτό δίνεται). Σε περίπτωση που το όνομα της οδού δεν δίνεται έχουμε ορίσει συμβολικά το αντίστοιχο πεδία να είναι None.

iii. **Node:** Κληρονομεί την κλάση Point και ουσιαστικά εκφράζει ένα σημείο του χάρτη nodes.csv που μας έχει δοθεί. Για τον λόγο αυτό εκτός από τις συντεταγμένες του, έχει και ένα πεδίο τύπου Street που αντιστοιχεί στην οδό που βρίσκεται το σημείο.

iv. **Taxi:** Κληρονομεί την κλάση Point και αντιστοιχεί σε ένα ταξί το οποίο περιγράφετε από τις γεωγραφικές συντεταγμένες του, τον μοναδικό κωδικό του (id) και το σημείο του χάρτη στο οποίο βρίσκεται πιο κοντά.

v. **Client:** Κληρονομεί την κλάση Point και αντιστοιχεί σε ένα πελάτη. Περιγράφετε από τις γεωγραφικές συντεταγμένες του και το κοντινότερο σημείο του χάρτη.

vi. **State:** Κληρονομεί την κλάση Node και εκφράζει μία συγκεκριμένη κατάσταση στον αλγόριθμό μας. Ουσιαστικά, είναι ένας κόμβος στον συνολικό γράφο του μοντέλου μας ή νοηματικά είναι μία συγκεκριμένη κατάσταση στην οποία βρίσκεται ο κόσμος μας. Περιγράφεται (πέρα από τα πεδία που κληρονομούνται από την Node) από την τιμή της ευριστικής σε αυτή την κατάσταση, την απόσταση από την αρχική κατάσταση (ρίζα του γράφου) και από μία λίστα στην οποία κρατιούνται η ακολουθία των καταστάσεων που χρειάστηκε για να φτάσουμε στην συγκεκριμένη. Το τελευταίο σχολιάζεται αναλυτικά στην περιγραφή του αλγορίθμου.

vii. **StateComparator:** Η κλάση αυτή υλοποιεί έναν comparator για τα αντικείμενα τύπου State και δηλώνει ότι ένα State είναι μεγαλύτερο από ένα άλλο όταν το άθροισμα της ευριστικής και της πραγματικής απόστασης του ενός είναι μεγαλύτερο από το αντίστοιχο άθροισμα του άλλου.

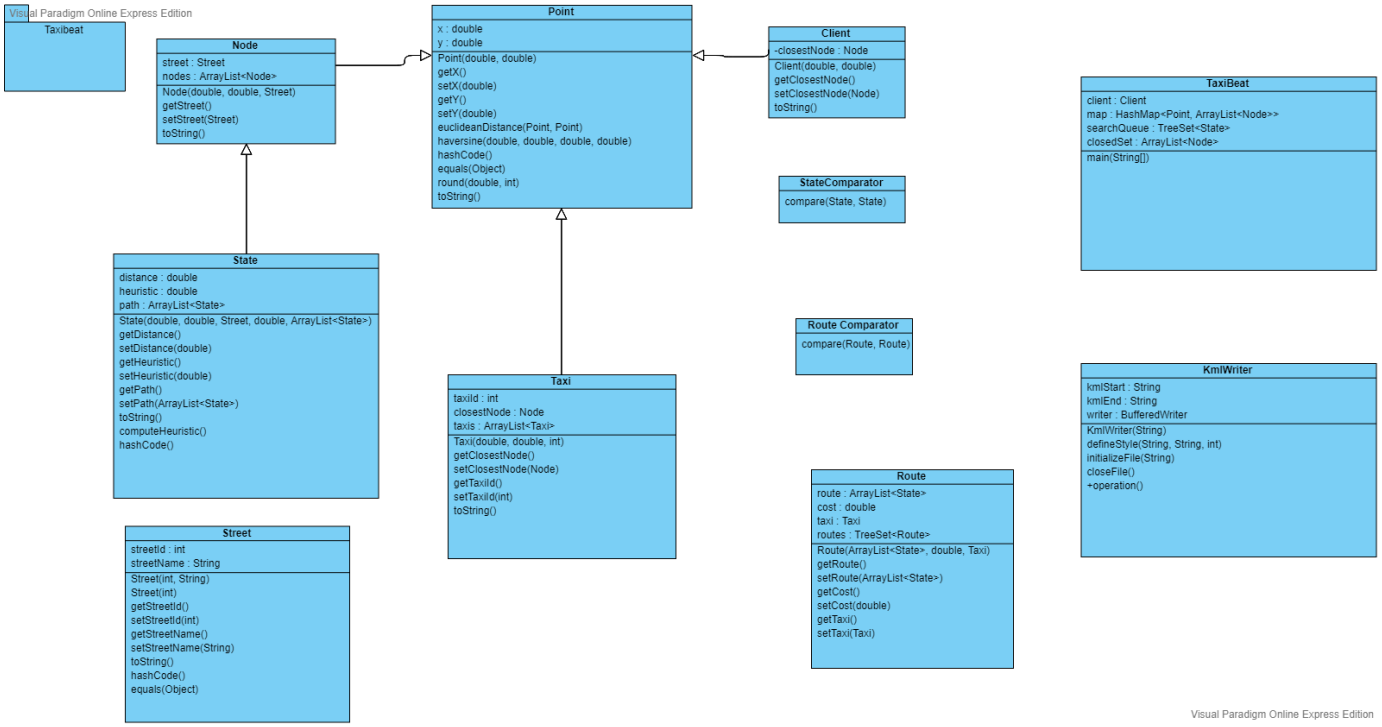
viii. **Route:** Η κλάση αυτή εκφράζει μία ελάχιστη διαδρομή που έχει βρει ο αλγόριθμος από ένα ταξί στον πελάτη. Περιέχει μία λίστα από States που είναι ουσιαστικά οι καταστάσεις της διαδρομής, τον συνολική απόσταση της διαδρομής και το ταξί από το οποίο ξεκινά.

ix. **RouteComparator:** Η κλάση αυτή υλοποιεί έναν comparator για τα αντικείμενα τύπου Route και δηλώνει ότι η διάταξη των διαδρομών καθορίζεται από την συνολική τους απόσταση.

x. **KmlWriter:** Η κλάση αυτή αναλαμβάνει την δημιουργία του αρχείου kml που αποτελεί μία συγκεκριμένη μορφοποίηση για την αναπαράσταση των διαδρομών σε κάποιον χάρτη.

xi. **TaxiBeat:** Η κλάση αυτή αποτελεί την main του προγράμματος και αναλαμβάνει το διάβασμα των δεδομένων, την δημιουργία κατάλληλων δομών δεδομένων για τις οποίες θα μιλήσουμε στη συνέχεια, την υλοποίηση του αλγορίθμου A* πάνω στον γράφο του προβλήματος και την παραγωγή του kml αρχείου μέσω κλήσεων στην KmlWriter.

Παρακάτω βλέπουμε το διάγραμμα uml του προγράμματός μας όπου φαίνονται συνοπτικά όλες οι κλάσεις με τα αντίστοιχα πεδία και τις μεθόδους τους.



Όσων αφορά τώρα την υλοποίηση του αλγορίθμου εφαρμόστηκε ο αλγόριθμος A* ο οποίος βρίσκει την βέλτιστη διαδρομή για κάθε ταξί (όταν η ευριστική είναι πάντα μικρότερη από την πραγματική απόσταση όπου στην περίπτωσή μας ισχύει) και στη συνέχεια παρουσιάζουμε στον πελάτη την ελάχιστη αυτών. Θεωρούμε δηλαδή τον πελάτη ως τελικό στόχο και κάθε φορά το εκάστοτε ταξί ως αρχικό. Έγινε μία παραλλαγή στον αλγόριθμο με σκοπό να παρουσιάζονται εναλλακτικές διαδρομές σύμφωνα με την οποία στο μέτωπο αναζήτησης κρατιούνται και διαδρομές με ίδιες συντεταγμένες αλλά με διαφορετικό μονοπάτι.

Δομές Δεδομένων

Στην ενότητα αυτή θα παρουσιαστούν συγκεντρωμένες όλες οι δομές δεδομένων που χρησιμοποιήθηκαν και οι λόγοι επιλογής αυτών. Έχουμε, λοιπόν:

1. Λίστα taxis που κρατάει όλα τα αντικείμενα τύπου Taxi (στατική στην κλάση Taxi). Χρησιμοποιήθηκε λίστα διότι είναι εύκολη στον χειρισμό που η μόνη λειτουργία που γίνεται σε αυτήν την λίστα είναι να διατρέχουμε τα στοιχεία της.

2. Λίστα nodes που κρατάει όλα τα αντικείμενα τύπου Node (στατική στην κλάση Node). Οι λόγοι είναι οι ίδιοι με παραπάνω.

3. Λίστα route που κρατάει για κάθε αντικείμενο τύπου Route τις καταστάσεις από τις οποίες αποτελείτε η διαδρομή (πεδίο της κλάσης Route).

4. Λίστα path που κρατάει για κάθε αντικείμενο της κατάστασης State όλες τις καταστάσεις που αντιστοιχούν στην διαδρομή μέχρι να φτάσουμε στην ίδια.

5. TreeSet με όνομα routes που κρατάει τις ελάχιστες διαδρομές που έχει εντοπίσει ο αλγόριθμος για κάθε ταξί (στατικό στην κλάση Route). Χρησιμοποιήθηκε η συγκεκριμένη δομή που ουσιαστικά είναι ένα σύνολο ταξινομημένο σύμφωνα με ένα comparator διότι θέλουμε οι διαδρομές να είναι ταξινομημένες σύμφωνα με την συνολική τους απόσταση (για αυτό τον λόγο υλοποιήθηκε και ο αντίστοιχος comparator) και δεν μας ενδιαφέρει το index της κάθε διαδρομής.

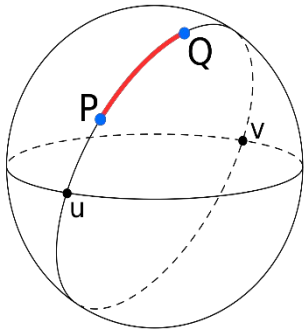
6. HashMap με όνομα map που αντιπροσωπεύει τον γράφο του προβλήματος μας και χρησιμοποιείται από τον αλγόριθμο A* για να βρούμε την ελάχιστη διαδρομή από ένα ταξί. Χρησιμοποιήθηκε η συγκεκριμένη δομή γιατί έχει σταθερό χρόνο πρόσβασης στα στοιχεία κάποιου κλειδιού και αυτή είναι η λειτουργία η οποία εφαρμόζεται συχνότερα πάνω στη συγκεκριμένη δομή. Τα κλειδιά είναι σημεία στα χάρτη και η τιμή είναι μία λίστα που περιέχει τα παιδιά του συγκεκριμένου σημείου. Ως παιδιά για ένα σημείο ορίζονται: Το προηγούμενο και το επόμενη στην οδό του αν το σημείο δεν είναι διασταύρωση. Το προηγούμενο και το επόμενο όλων των οδών που διασταυρώνονται στο συγκεκριμένο σημείο αν είναι διασταύρωση.

7. TreeSet με όνομα searchQueue που αποτελεί το μέτωπο αναζήτησης του κόσμου μας. Περιέχει, δηλαδή, τις καταστάσεις στις οποίες έχουμε φτάσει αλλά δεν τις έχουμε επεκτείνει με μία εξαίρεση βέβαια που προέκυψε για τον εντοπισμό εναλλακτικών ισοδύναμων διαδρομών και παρουσιάζεται στην αντίστοιχη ενότητα. Χρησιμοποιήθηκε η συγκεκριμένη δομή διότι οι καταστάσεις μέσω στο μέτωπο αναζήτησης πρέπει να είναι πάντα ταξινομημένες.

8. Μία ακόμα λίστα η οποία αναπαριστά το κλειστό σύνολο με όνομα closedSet.

Συνάρτηση εκτίμησης απόσταση

Η ευριστική συνάρτηση που χρησιμοποιήθηκε για την υλοποίηση του αλγόριθμου είναι η **Haversine formula** καθώς ήταν η καταλληλότερη για εφαρμογές πλοήγησης, για το γεγονός ότι υπολογίζει και την καμπυλότητα της γης, δίνοντας μόνο γεωγραφικό πλάτος και μήκος των σημείων, ούτως ώστε να εκτιμάει την απόσταση 2 σημείων πάνω σε σφαίρα και όχι στο επίπεδο. Επίσης κάτι άλλο που την καθιστά καταλληλότερη είναι το γεγονός ότι υποεκτιμά την πραγματική απόσταση από τον στόχο εγγυημένα ,διότι



υπολογίζει την καμπύλη που ενώνει απευθείας τα 2 σημεία κάτι που καθιστά κατά συνέπεια τον αλγόριθμό αποδεκτό (admissible).

Haversine formula: Έστω δύο σημεία $P(X_1, Y_1), Q)$
,όπου X_1, X_2 το γεωγραφικό μήκος των 2 σημείων και
Y_1, Y_2 το γεωγραφικό πλάτος των 2 σημείων αντίστοιχα

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{X_2 - X_1}{2}\right) + \cos(X_1)\cos(X_2)\sin^2\left(\frac{Y_2 - Y_1}{2}\right)}\right)$$

, όπου d η απόσταση που ενώνει απευθείας τα σημεία P,Q πάνω στην σφαίρα .

- ## Χειρισμός γεωγραφικών συντεταγμένων

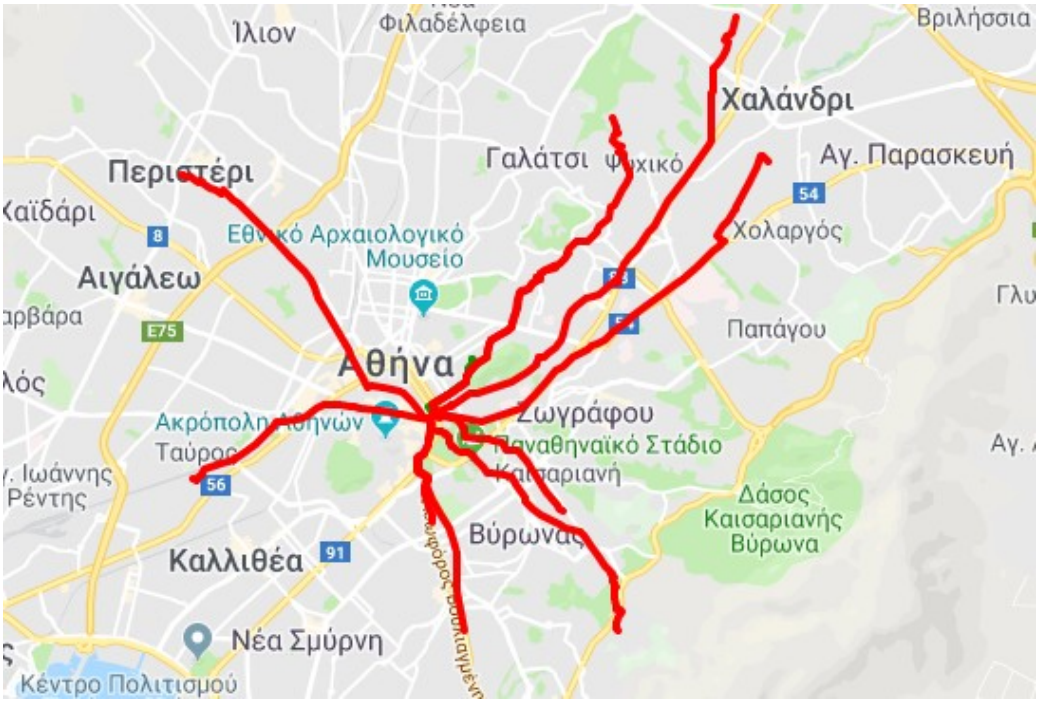
Οι γεωγραφικές συντεταγμένες που περιέχονταν στα αρχεία client.csv, nodes.csv και taxis.csv. Αφότου διαβάστηκαν σε αντίστοιχους πίνακες (για ταξί και nodes) και στην μεταβλητή client, δεν έγινε κάποια ιδιαίτερη προεπεξεργασία τους. Το μόνο που χρειάστηκε είναι η στρογγυλοποίηση αυτών σε μέτρα ώστε να είναι δυνατή η εύρεση ισοδύναμων εναλλακτικών διαδρομών. Ως προεπεξεργασία θα μπορούσε να θεωρηθεί το γεγονός ότι επειδή για τα ταξί και τον πελάτη δεν γνωρίζουμε την οδό τους βρίσκουμε πάντα το κοντινότερο τους node θεωρούμε ότι βρίσκονται εκεί και στο τέλος απλά προσθέτουμε και τις κατάλληλες αποστάσεις από το πραγματικό σημείο.

- ## Παραδείγματα χρήσης της υπηρεσίας:

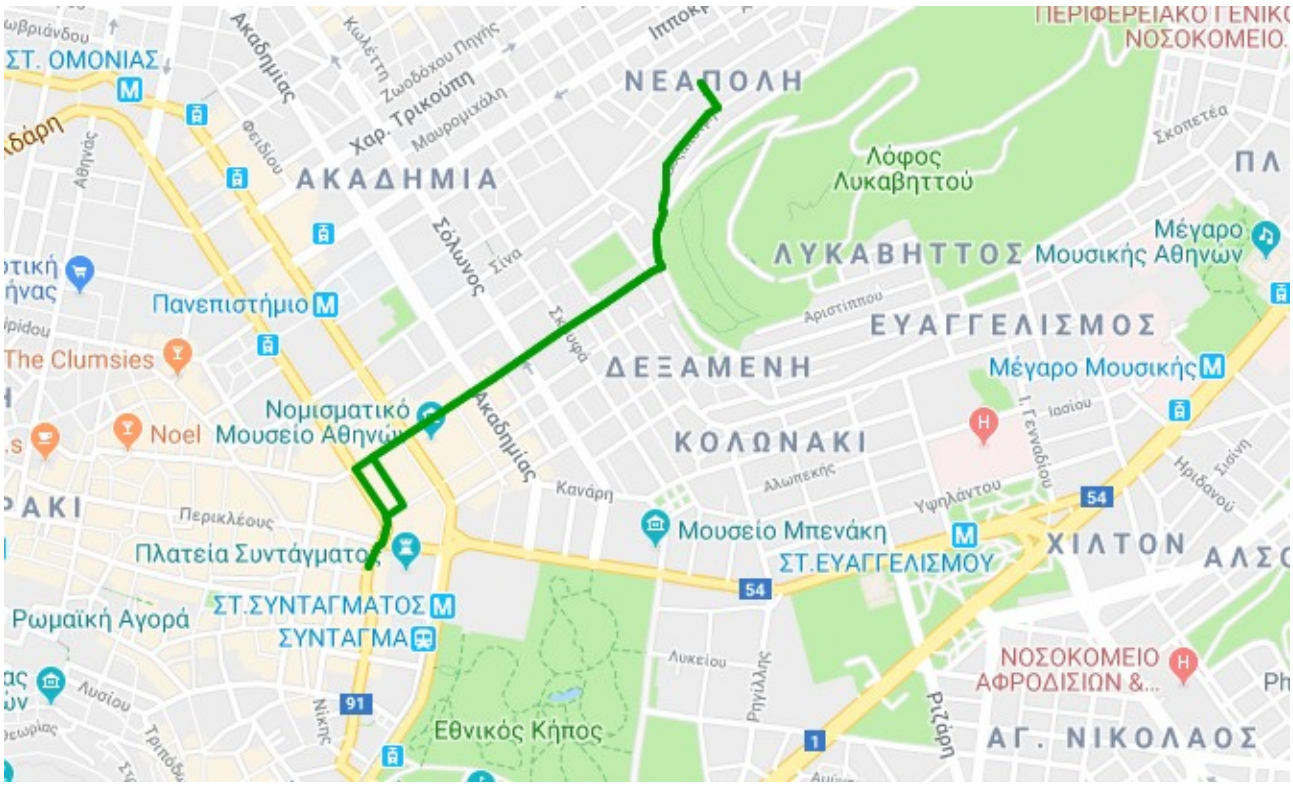
Αρχικά, τρέχοντας το πρόγραμμα με το δεδομένα που μας δόθηκαν παράγεται το routes.kml αρχείο στο οποίο έχουν αποθηκευτεί σε κατάλληλη μορφοποίηση όλες οι ελάχιστες διαδρομές για κάθε ταξί. Κανονικά πρέπει να κρατιούνται μόνο η ελάχιστες (μία ή πολλές ανάλογα αν υπάρχουν εναλλακτικές) αλλά για λόγους παρουσίασης της άσκησης βλέπουμε όλες τις διαδρομές. Αρχικά στον παρακάτω πίνακα βλέπουμε τις ελάχιστες διαδρομές για κάθε ταξί όπως τις εκτυπώνει το πρόγραμμα:

ΤΑΞΊ	ΚΌΣΤΟΣ
100	**1.402**
100	**1.402**
150	**1.864**
120	**3.045**
170	**3.532**
160	**3.662**
110	**4.351**
180	**5.313**
190	**5.997**
140	**6.66**
200	**7.542**
130	**8.92**

Στη συνέχεια βλέπουμε τον χάρτη μέσω της υπηρεσίας myMaps όπου με πράσινο παρουσιάζεται η ελάχιστη διαδρομή. Στην συγκεκριμένη περίπτωση είναι δύο οι ελάχιστες απλά δεν φαίνεται η δεύτερη γιατί επικαλύπτεται με μία άλλη διαδρομή προς άλλο ταξί.

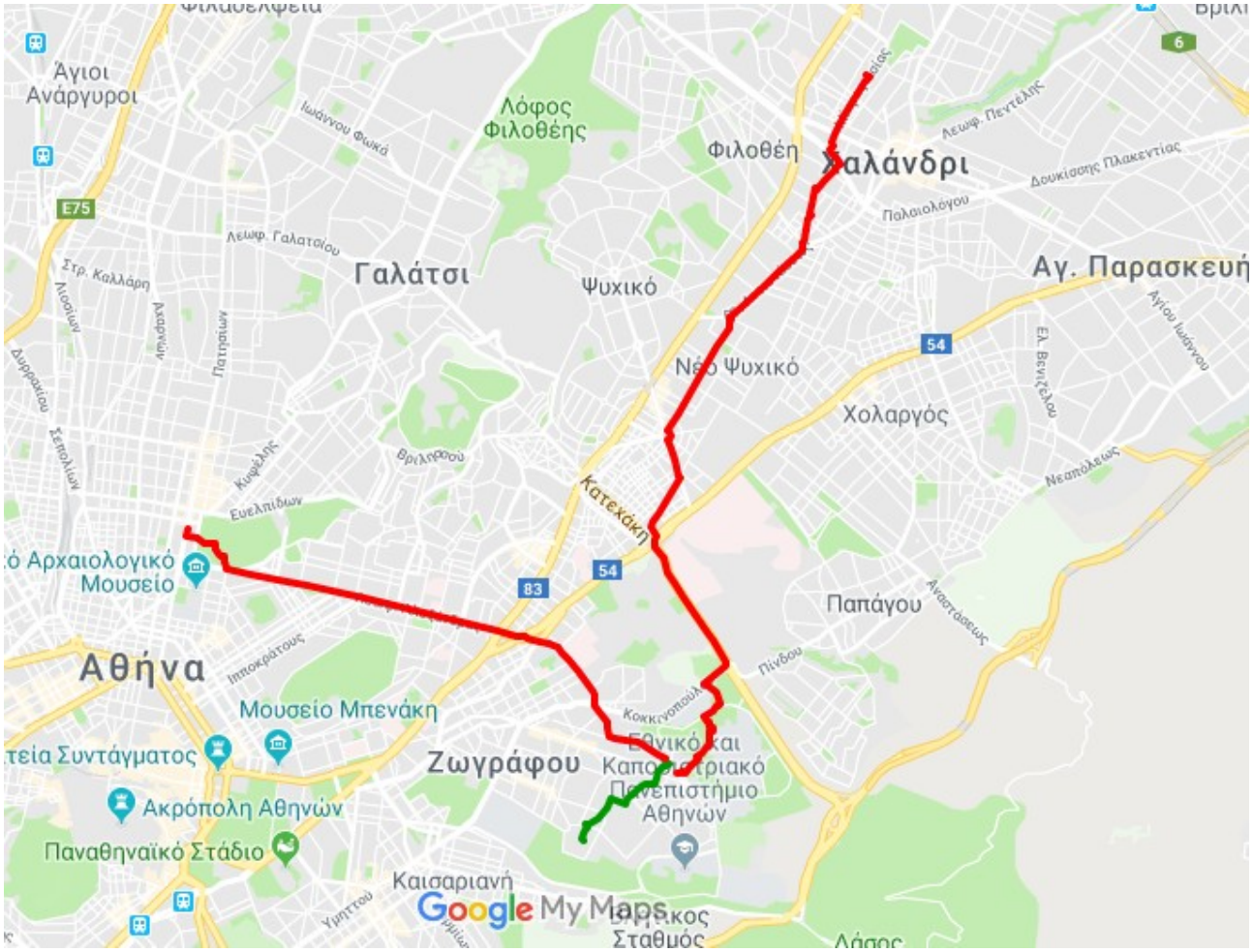


Και παρακάτω η ίδια εικόνα με τις ελάχιστες διαδρομές:



Στη συνέχεια, χρησιμοποιώντας τα αρχεία myclient.csv και mytaxis.csv δημιουργούμε τις δικές μας διαδρομές οι οποίες φαίνονται παρακάτω:

Ταξί	Κόστος (km)
100	**1.161**
120	**4.885**
110	**7.29**



- ## Σχολιασμός αποτελεσμάτων

Γενικά παρατηρούμε ότι το πρόγραμμα λειτουργεί σωστά βρίσκοντας πάντα την ελάχιστη διαδρομή προς κάθε ταξί και μετά επιλέγοντας την μικρότερη από αυτές βρίσκει την συνολική ελάχιστη. Σε περίπτωση που υπάρχουν εναλλακτικές διαδρομές τις εντοπίζει κανονικά. Βέβαια αυτό κάθε φορά εξαρτάται από τις στρογγυλοποίηση που θα κάνουμε στις αποστάσεις μας. Από άποψη ταχύτητας είναι αρκετά γρήγορο κάνοντας 4-5 sec για να εντοπίσει τις διαδρομές και να δημιουργήσει κατάλληλα το kml αρχείο, πράγμα που οφείλεται στις δομές δεδομένων που χρησιμοποιήθηκαν όπου η συνήθης λειτουργία σε καθεμία από αυτές απαιτεί τον ελάχιστο δυνατό χρόνο. Παρακάτω αναλύονται διάφορα κρίσιμα σημεία της λειτουργίας του προγράμματος:

- ✓ Ικανότητα αλγορίθμου να προτείνει εναλλακτικές διαδρομές: Στα δεδομένα που μας δίνονται παρατηρούμε ότι η ελάχιστη απόσταση του πελάτη από το ταξί 1 προκύπτει ακολουθώντας δύο διαφορετικές διαδρομές. Έτσι, σωστά το πρόγραμμα μας επιστρέφει και τις δύο αυτές διαδρομές. Αυτό το καταφέρνει ακολουθώντας την παρακάτω λογική: Κάθε φορά που επεκτείνουμε μία κατάσταση για κάθε παιδί βρίσκουμε τις καταστάσεις του μετώπου αναζήτησης που περιέχουν το ίδιο σημείο με αυτό. Σβήνουμε αυτές που έχουν μεγαλύτερη απόσταση αφού πλέον δεν χρειάζονται και κρατάμε πάντα αυτές με την μικρότερη απόσταση από την αρχή. Αν μείνουν παραπάνω από μία με την ελάχιστη απόσταση αυτό σημαίνει ότι έχουμε εναλλακτική διαδρομή το οποίο το βλέπουμε και από το πεδίο path της κάθε κατάστασης. Αφού γίνει αυτό, λοιπόν, γνωρίζουμε ότι κάθε στιγμή στο μέτωπο αναζήτησης βρίσκονται μόνο καταστάσεις με διαφορετικά σημεία και , αν υπάρχουν με ίδια, να εκφράζουν εναλλακτικές διαδρομές με ίδιο κόστος. Όταν τώρα πάμε να βάλουμε μία κατάσταση στο κλειστό σύνολο ελέγχουμε αν υπάρχει κάποια ισοδύναμή τις (ίδιο σημείο, ίδιο κόστος, διαφορετικό μονοπάτι) στο μέτωπο. Αν υπάρχει τότε δεν την βάζουμε στο κλειστό ακόμα έτσι ώστε όταν έρθει η ώρα να πάρουμε την ισοδύναμη της κατάσταση να ελέγξουμε τα παιδιά της κανονικά.
- ✓ Συνθήκες κάτω από τις οποίες μπορεί να προτείνει το πρόγραμμα εναλλακτικές ισοδύναμες διαδρομές: Αυτό εξαρτάται από τα nodes που έχουν δοθεί και από την ανεκτικότητα που θα δείξουμε στην έννοια “ισοδύναμη διαδρομή” μέσα από την στρογγυλοποίηση που θα κάνουμε κάθε φορά στις αποστάσεις. Δηλαδή, όσων αφορά το πρώτο, όσο πιο πυκνά είναι τα nodes που δίνονται τόσο πιο πιθανό είναι να βρεθεί μία ισοδύναμή διαδρομή διότι θα είναι πιο ρεαλιστική η απεικόνιση. Για το δεύτερο, στην περίπτωση μας θέλαμε να έχουμε ακρίβεια τάξης του 1 μέτρου που σημαίνει αν δύο διαδρομές έχουν απόσταση ίση στρογγυλοποιημένη στο μέτρο είναι ισοδύναμες. Θα μπορούσαμε προφανώς (και θα ήταν πιο ρεαλιστικό) να θεωρήσουμε ακρίβεια τάξης των 10 μέτρων διότι σε μία τέτοια εφαρμογή θα θέλαμε να προταθούν ισοδύναμη διαδρομή στον πελάτη ακόμα και αν το κόστος της διαφέρει κατά 10 μέτρα από την άλλη.
- ✓ Βελτιστότητα ισοδύναμων διαδρομών: Οι ισοδύναμες διαδρομές που υπολογίζει το πρόγραμμα είναι βέλτιστες αν δούμε και σχηματικά τις διαδρομές στον χάρτη.