Άγγελος Πλεύρης
03115038
ΗΜΜΥ 7ο Εξάμηνο
Χανής Γεώργιος
03115116
ΗΜΜΥ 7ο Εξάμηνο

Τεχνητή Νοημοσύνη

1ο Θέμα

Γενική Περιγραφή:
Υλοποιήσαμε ενα σύστημα το οποίο απαρτίζεται από 5 κλάσεις: Taxi, node, Client, Graph , AIask1, και την κλάση Driver. Στην κλάση Taxi αποθηκεύουμε τα δεδομένα που αφορούν τα ταξί, όπως τις συντεταγμένες τους(σε ξεχωριστές double μεταβλητές), το id τους καθώς και τον πλησιέστερο κόμβο του χάρτη, καθώς είναι πιθανό η τοποθεσία των ταξί να μην αντιστοιχεί σε κόμβο. Η κλάση node αναφέρεται στους κόμβους που αποτελούν το γράφο. Κάθε τέτοιος κόμβος έχει τις συντεταγμένες του,το id του ώστε να αναγνωρίζουμε αν βρίσκεται στον ίδιο δρόμο με άλλο κόμβο. Η κλάση Client περιέχει τα δεδομένα που αφορούν τον πελάτη, δηλαδή τις συντεταγμένες του, τον πλησιέστερο σε αυτόν κόμβο σε αντίστοιχη λογική με τα Taxi. Οι κλάσεις αυτές συνοδεύονται με τους απαραίτητους getters ώστε να έχουμε πρόσβαση στις τιμές των μεταβλητών τους που είναι private καθώς και άλλες μεθόδους για διάφορους υπολογισμούς, αποτιμήσεις ισότητας και constructors. Η κλάση Driver είναι βοηθητική και χρησιμοποιείται στο διάβασμα των taxi από το αρχείο εισόδου καθώς και τη δημιουργία της λίστας των taxi. Η κλάση Aiask1 περιέχει τη main συνάρτηση του προγράμματος η οποία παράγει ένα αντικείμενο τύπου Graph και καλώντας τις μεθόδους του Graph τυπώνει στην κονσόλα το καλύτερο ταξί καθώς και παράγει το αρχείο map.kml το οποίο έχει τις εναλλακτικές διαδρομές. Από τα παραπάνω γίνεται σαφές ότι η πιο σημαντική κλάση του προγραμματός μας είναι η Graph. Η κλάση αυτή θα έχει ένα αντικείμενο τύπου Client , την λίστα των ταξί τύπου ArrayList<Taxi>, ένα hashmap το οποίο θα κωδικοποιεί το γράφο μας τύπου HashMap<String,ArrayList<node>>(το string αυτό είναι οι συντεταγμένες του κόμβου σε μορφή "X,Y"), μια λίστα από λιστες τύπου ArrayList<ArrayList<node>> που θα αποθηκεύουμε τις εναλλακτικές διαδρομές του καλύτερου ταξί και μια μεταβλητή int BestTaxi που θα κρατάει το id του καλύτερου taxi. Με το hashmap κωδικοποιούμε το γράφο ως εξής: θεωρούμε ως κλειδί τις συντεταγμένες των κόμβων του (ο λόγος που είναι string και δεν είναι Node είναι για να παράγουμε κοινό hash για κόμβους με διαφορετικά id αλλά ίδιες συντεταγμένες) και ως τιμή μια λίστα από κόμβους που ουσιαστικά είναι οι γείτονες αυτού του κόμβου, δηλαδή οι ακμές του γράφου.Η κλάση Graph περιέχει ακόμα τις μεθόδους private ArrayList<ArrayList<node>> A_star_implementation, public double A_star και public void write_path. Η πρώτη μέθοδος υλοποιεί τον αλγόριθμο A* με ορίσματα τον κόμβο έναρξης καθώς και τον κόμβο στόχο και επιστρέφει μια λίστα από λίστες με κόμβους, δηλαδή μια λίστα από όλα τα εναλλακτικά μονοπάτια που βρήκε ο αλγόριθμος A* για τη συγκεκριμένη είσοδο που του δόθηκε. Η μέθοδος A_star καλεί επαναληπτικά την A_star_implementation με άλλο ταξί κάθε φορά ως κόμβο έναρξης και βρίσκει το ταξί με την μικρότερη διανυόμενη απόσταση καθώς και πόσο είναι αυτή. Ενημερώνει και αποθηκεύει την μικρότερη απόσταση καθώς και το id του ταξί το οποίο τελικά θα κληθεί. Η write_path παράγει το map.kml αρχείο στο οποίο εμφανίζει με πράσινο χρώμα τη βέλτιστη διαδρομή και με κόκκινο όλες τις υπόλοιπες ισοδύναμες για το ίδιο ταξί που έχει επιλεγεί ως καλύτερο.

Λεπτομέρειες Υλοποίησης:

- Για πραγματική και για ευριστική συνάρτηση χρησιμοποιούμε την ευκλείδια απόσταση , δηλαδή πόσο απέχουν δύο κόμβοι. Προφανώς για την ευριστική παίρνουμε την απόσταση του κόμβου από τον κόμβο στόχο. Η ευριστική μας συνάρτηση είναι η καταλληλότερη μέγιστη εκτιμήτρια συνάρτηση που θα μπορούσαμε να χρησιμοποιήσουμε και οι τιμές που δίνει δεν υπερεκτιμούν την απόσταση, αλλά αντίθετα την υποτιμούν λόγω τριγωνικής ανισότητας και επομένως είναι αποδεκτές (Admissible) και άρα σίγουρα η ευριστική τιμή είναι μικρότερη η ίση της πραγματικής τους απόστασης.

- Για την υλοποίηση του A* χρησιμοποιούμε μια ουρά προτεραιότητας PriorityQueue καθώς και δύο σύνολα HashSet openset και closedset για να ελέγχουμε αν κάτι βρίσκεται μέσα στην ουρά σε γρήγορο χρόνο(O(1)) και για να ελέγχουμε εάν ένα κόμβο τον έχουμε επισκεφτεί. Ο A* μας βγάζει ένα στοιχείο από την ουρά (το μικρότερο στοιχείο του μετώπου αναζήτησης) το current_node, και βάζει στην ουρά τους γείτονες του αρχικού κόμβου που δεν έχουν ξαναμπεί στην ουρά καθώς και ενημερώνει τους γείτονες του ότι έχουν έρθει από τον current_node. Αυτό γίνεται γιατί ο current node αφού έχει βγει από το PriorityQueue θα έχει το καλύτερο f και άρα για όλους τους γείτονες του θα είναι ο καλύτερος τρόπος να φτάσουμε σε αυτούς. Αν κάποιος γείτονας είναι ήδη μέσα στην ουρά ελέγχει αν αυτό που υπάρχει ήδη μέσα στην ουρά έχει μικρότερο ή ίσο κόστος από το στοιχείο που πάμε να εισάγουμε. Αν αυτό που πάμε να βάλουμε είναι καλύτερο τότε πετάμε αυτό που υπάρχει στην ουρά και το αντικαθιστούμε, αν είναι χειρότερο τότε η ουρά παραμένει ως έχει. Η περίπτωση της ισότητας είναι που μας δίνει και τα διαφορετικά μονοπάτια. Αρχικά να επισημάνουμε ότι δεν εξετάζουμε ακριβή ισότητα των floats αλλά βρίσκουμε την ποσοστιαία διαφορά τους και θέλουμε αυτή να μην υπερβαίνει το 1% ώστε να τα θεωρούμε ίσα. Στην περίπτωση αυτή αντί να πούμε ότι πατέρας τους είναι ο current node θα πάρουμε τον μέχρι τώρα πατέρα τους και θα προσθέσουμε τον current node ως πατέρα, φτιάχνοντας έτσι μια λίστα. Δηλαδή για κάποιους κόμβους θα υπάρχουν παραπάνω από έναν βέλτιστους τρόπους άφιξης σε αυτούς, δημιουργώντας έτσι εναλλακτικές διαδρομές. Επίσης στον A* μας δεν διακόπτουμε το άδειασμα του μετώπου αναζήτησης μόλις βρούμε τον κόμβο στόχο για αφήσουμε τον αλγόριθμο να βρει τυχόν εναλλακτικές διαδρομές που έχει στην ουρά αλλά δεν εξέτασε μέχρι εκείνη τη στιγμη. Τέλος, αφού αδειάσει και η ουρά ανακατασκευάζουμε το μονοπάτι με το οποίο φτάσαμε στον κόμβο στόχο. Για να το κάνουμε αυτό έχουμε ένα HashMap<String,ArrayList<node>> που λέγεται came_from και χρησιμοποιεί ως κλειδιά το hash που αναφέραμε παραπάνω (string με συντεταγμένες "X,Y") και ως τιμές τη λίστα των πατεράδων κάθε κόμβου. Για την ανακατασκευή των μονοπατιών όσο υπάρχουν εναλλακτικά μονοπάτια τα βρίσκει και τα αποθηκεύει. Ως εναλλακτικά μονοπάτια θεωρεί ότι κάθε φορά που βρίσκει έναν κόμβο με παραπάνω από έναν πατέρες παίρνει μια απόφαση να πάρει τη μία εναλλακτική και την αφαιρει από τη λίστα των πατέρων δημιουργώντας έτσι ενα μονοπάτι. Στην επόμενη επανάληψη έχοντας αφαιρέσει τη μία εναλλακτική θα πάρει την άλλη και τελικά θα σταματήσει να βρίσκει εναλλακτικά μονοπάτια όταν κάθε κόμβος του μονοπατιού θα έχει μόνο έναν πατέρα.

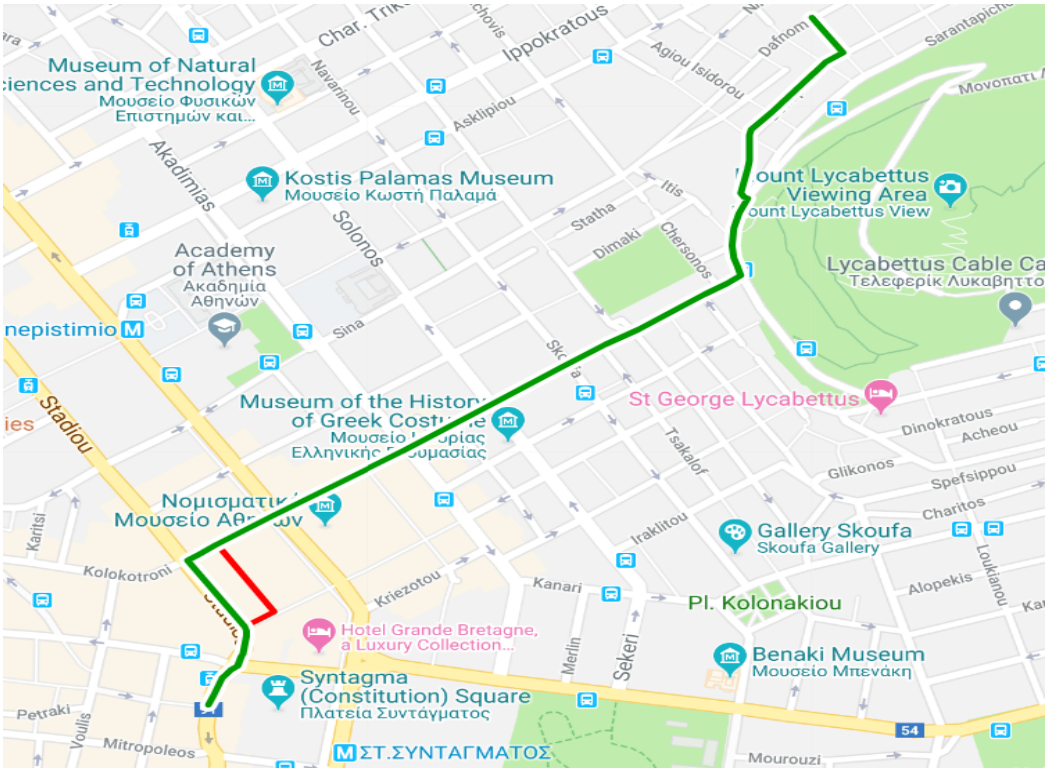
- Στα δεδομένα εισόδου χρειάστηκε να τροποποιήσουμε το αρχείο με τις οδούς καθώς τα ελληνικά στα ονόματα των δρόμων δημιουργούσαν πρόβλημα και γι αυτό δεν αποθηκεύουμε κίολας κάπου αυτήν την πληροφορία. Κατά τα άλλα τα δεδομένα διαβάζονται με την βοήθεια ενός scanner και αποθηκεύονται στα κατάλληλα πεδία των κλάσεων που αναφέρθηκαν παραπάνω για να μπορούμε να τα χρησιμοποίησουμε και να τα επεξεργαστούμε
- Για τον πελάτη και για ταξί χρησιμοποιούμε τα closest_node για να βρούμε ποιος κόμβος του χάρτη είναι πιο κοντά στις τοποθεσίες τους ώστε να μπορούμε να έχουμε αναφορά σε αυτά.

Γενικά Σχόλια

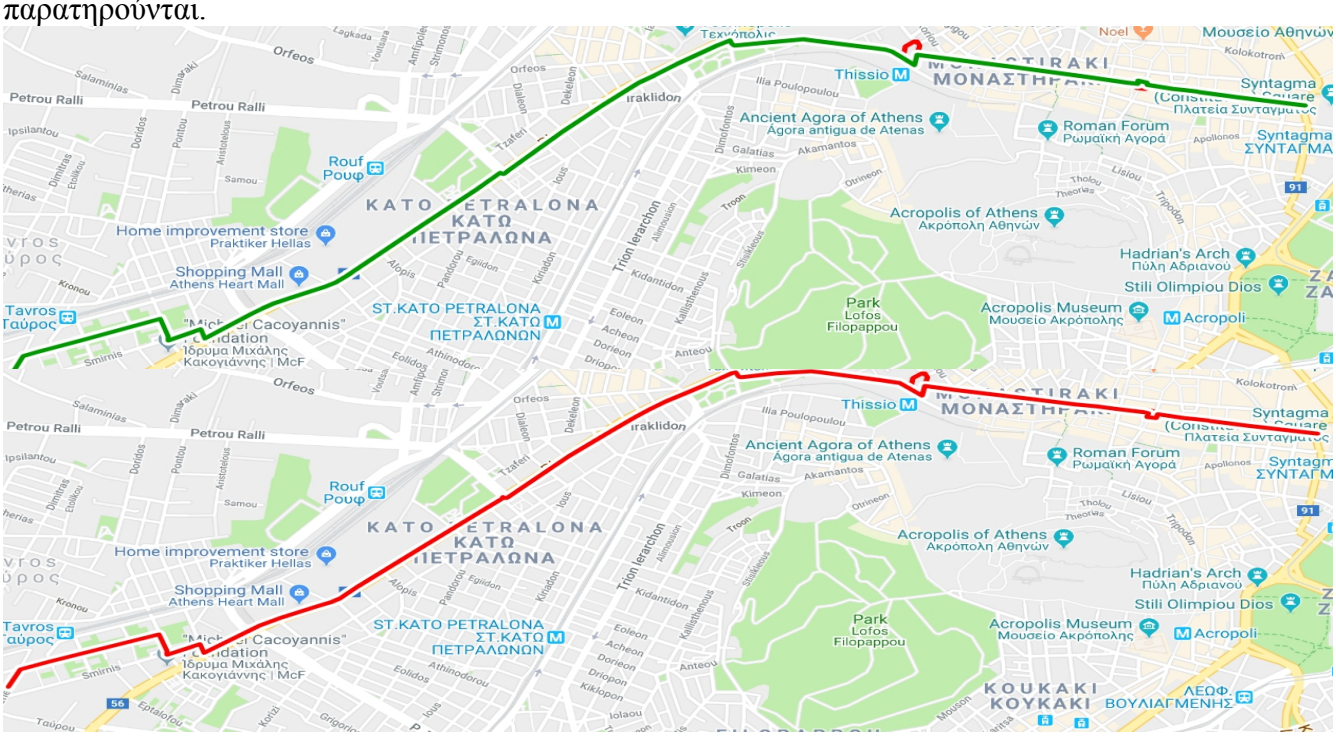
Ο κώδικας μας σε γενικές γραμμές παράγει αρκετές εναλλακτικές διαδρομές, οι οποίες όμως πρακτικά δεν έχουν μεγάλες διαφορές μεταξύ τους. Με τον τρόπο που τον έχουμε υλοποιήσει, η προυπόθεση για να βγάζει πραγματικά εναλλακτικές διαδρομές , είναι ,σχηματικά, ο στόχος και το ταξί να ενώνονται με έναν νοητό ρόμβο όπου οι δεξιές και οι αριστερές ακμές είναι περίπου ίσες. Δηλαδή ο κώδικας μας δεν βρίσκει εναλλακτικές διαδρομές όπως θα θέλαμε πχ για να αποφύγουμε ένα δρόμο, αλλά βρίσκει ουσιαστικά στη βέλτιστη διαδρόμη "ρόμβους" οι οποίοι είναι πολύ κοντά στο βέλτιστο μήκος διαδρομής. Ακόμη, με τις μεταβλητές που έχουμε είτε float ,πόσο μάλλον double, ισότητα δύο μονοπατιών είναι πολύ δύσκολο να υπάρξει, πρακτικά απίθανο. Για αυτό απαιτείται κάποιου είδους στρογγυλοποίηση. Εμείς υλοποιήσαμε την ποσοστιαία μετάβολη, αλλά κατά τη διάρκεια της ανάπτυξης δοκιμάσαμε τη στρογγυλοποίηση των τιμών της f(n) και ο αλγόριθμος έδινε παρόμοια αποτελέσματα. Επίσης, το γεγονός ότι οι δρόμοι όλοι θεωρούνται διπλής κατεύθυνσης απλοποιεί εξαιρετικά το πρόβλημα σε σημείο να είναι πολύ λίγες οι ελάχιστες διαδρομές αφού το ταξί μπορεί να κινηθεί σχεδόν ευθεία για να φτάσει τον στόχο. Επομένως είναι αρκετά λογικό να μην είναι πολύ κοντά σε μία αληθινή υλοποίηση για καθημερινή χρήση. Ακόμη, επειδή θεωρούμε ως εναλλακτική διαδρομή ,κάθε διαδρομή που διαφοροποιείται έστω και σε έναν κόμβο, σε κάποια στιγμιότυπα έχουμε πολλες εναλλακτικές διαδρομές που το μεγαλύτερο κομμάτι τους είναι αλληλοεπικαλυπτόμενο. Τέλος, η υλοποίηση μας εμφανίζει την πραγματικά βέλτιστη διαδρομή με πράσινο, δηλαδή αυτή που θα επέλεγε ο απλός A*, όλες οι υπόλοιπες είναι λίγο μεγαλύτερες με διαφορά μερικών δεκάδων μέτρων οπότε πρακτικά θα λέγαμε ότι είναι εξ΄ ίσου βέλτιστες. Ο αλγόριθμος μας πάντα θα βρίσκει τη βέλτιστη διαδρομή και υπάρχει και ποθανότητα να μην βρίσκει εναλλακτικές εάν η θέση πελάτη και ταξί είναι τέτοια.

Παραδείγματα της εκτέλεσης του κώδικα μας

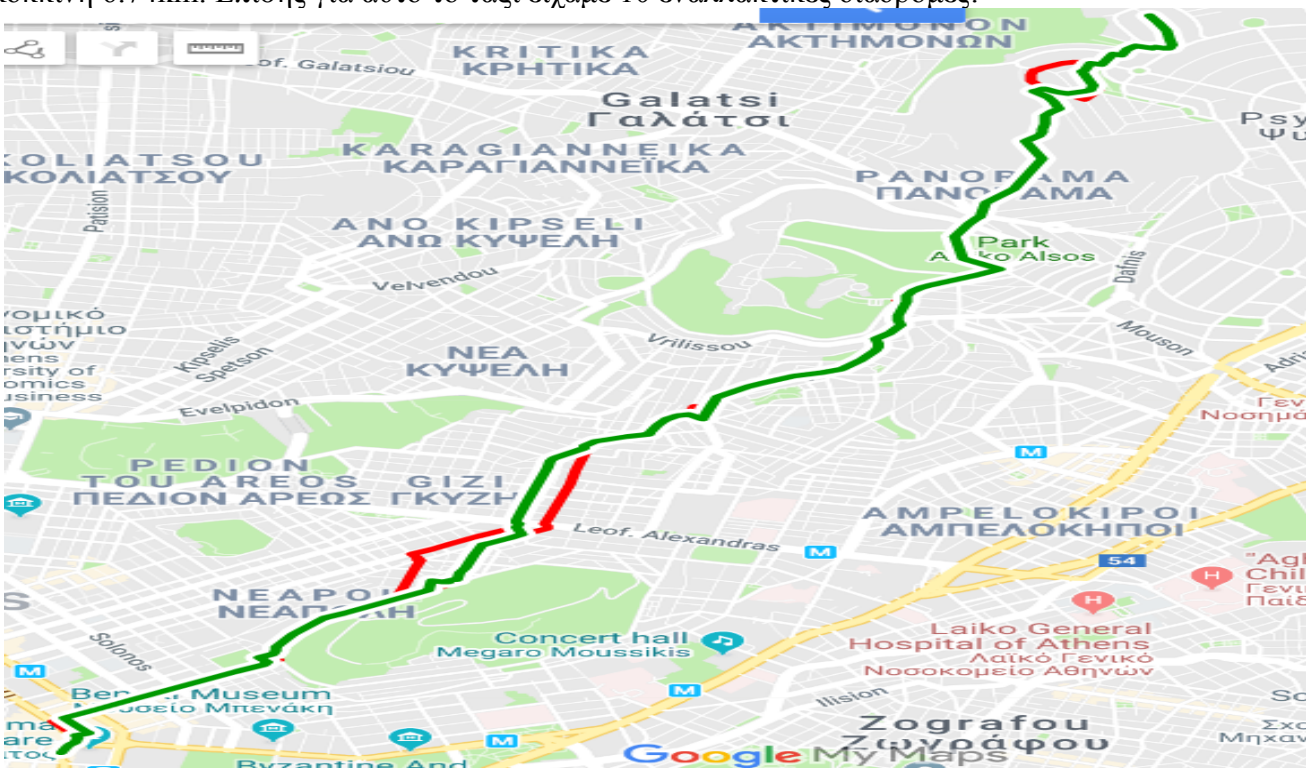
Το αρχικό παράδειγμα:
Αρχικά όπως είπαμε, ο κώδικας επιστρέφει τη διαδρομή για ένα ταξί στην περίπτωση μας το ταξί με id 100, και με πράσινο συμβολίζει τη βέλτιστη και με κόκκινο τις εναλλακτικές της. Αλλά παρακάτω φαίνονται οι διαδρομές για όλα τα ταξί, μαζί με τις εναλλακτικές τους.
Ο πελάτης βρίσκεται στο Σύνταγμα. Με κόκκινο,βλέπουμε ότι η εναλλακτική διαδρομή είναι ο "ρόμβος" που περιγράψαμε παραπάνω.



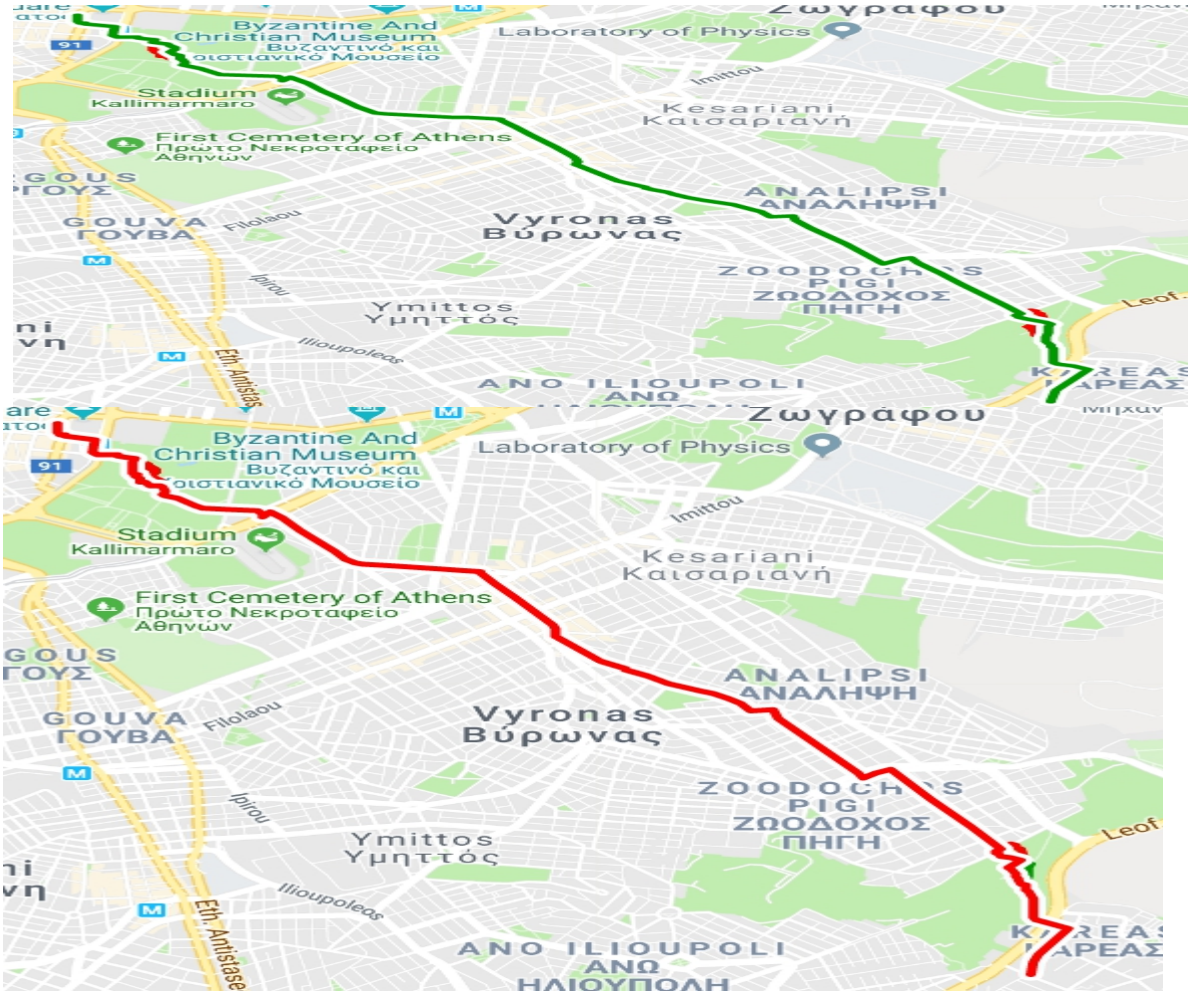
Για το ταξί με id 110 βλέπουμε τις 3 εναλλακτικές διαδρομές και τις μικρές αλλαγές που παρατηρούνται.



Για το ταξί με id 140 παρατηρούμε ότι οι εναλλακτικές διαδρομές είναι πιο κοντά σε πραγματικές εναλλακτικές. Συγκεκριμένα η πράσινη διαδρομή είναι σύμφωνα με το google maps 6.72km ενώ η κόκκινη 6.74km. Επίσης για αυτό το ταξί είχαμε 10 εναλλακτικές διαδρομές!

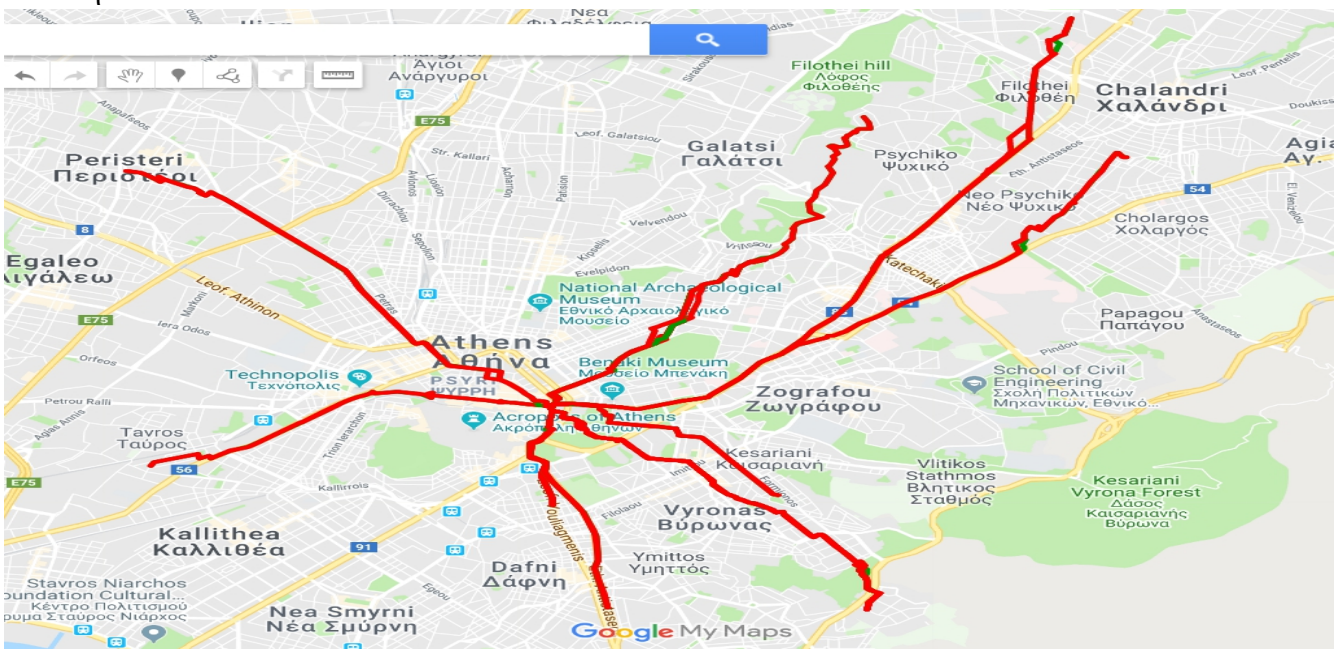


Για το ταξί με id 180 έχουμε 9 εναλλακτικές και φαίνεται το πρόβλημα της απλοποίησης της υλοποίησης καθώς το ταξί μας πηγαίνει μέσα από τον εθνικό κήπο και μάλιστα εντός του έχουμε εναλλακτικές διαδρομές:

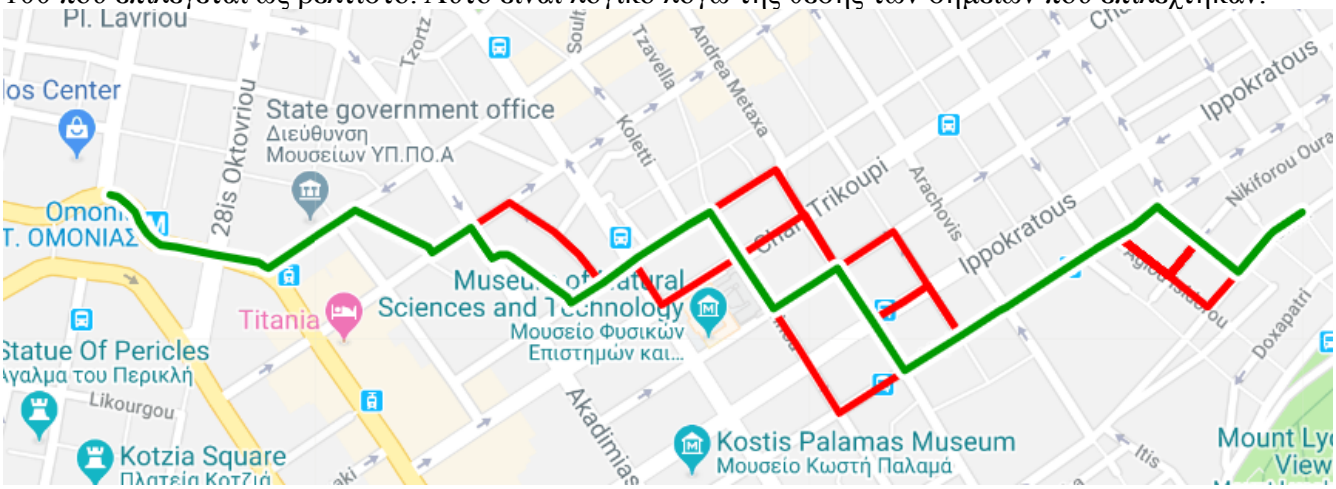


Ας δούμε γενικά όλες τις διαδρομές των ταξί εκτός από το πρώτο που ήταν και το καλύτερο.Φαίνονται έντονα τα κόκκινα γιατί επικαλύπτουν τις πράσινες διαδρομές.

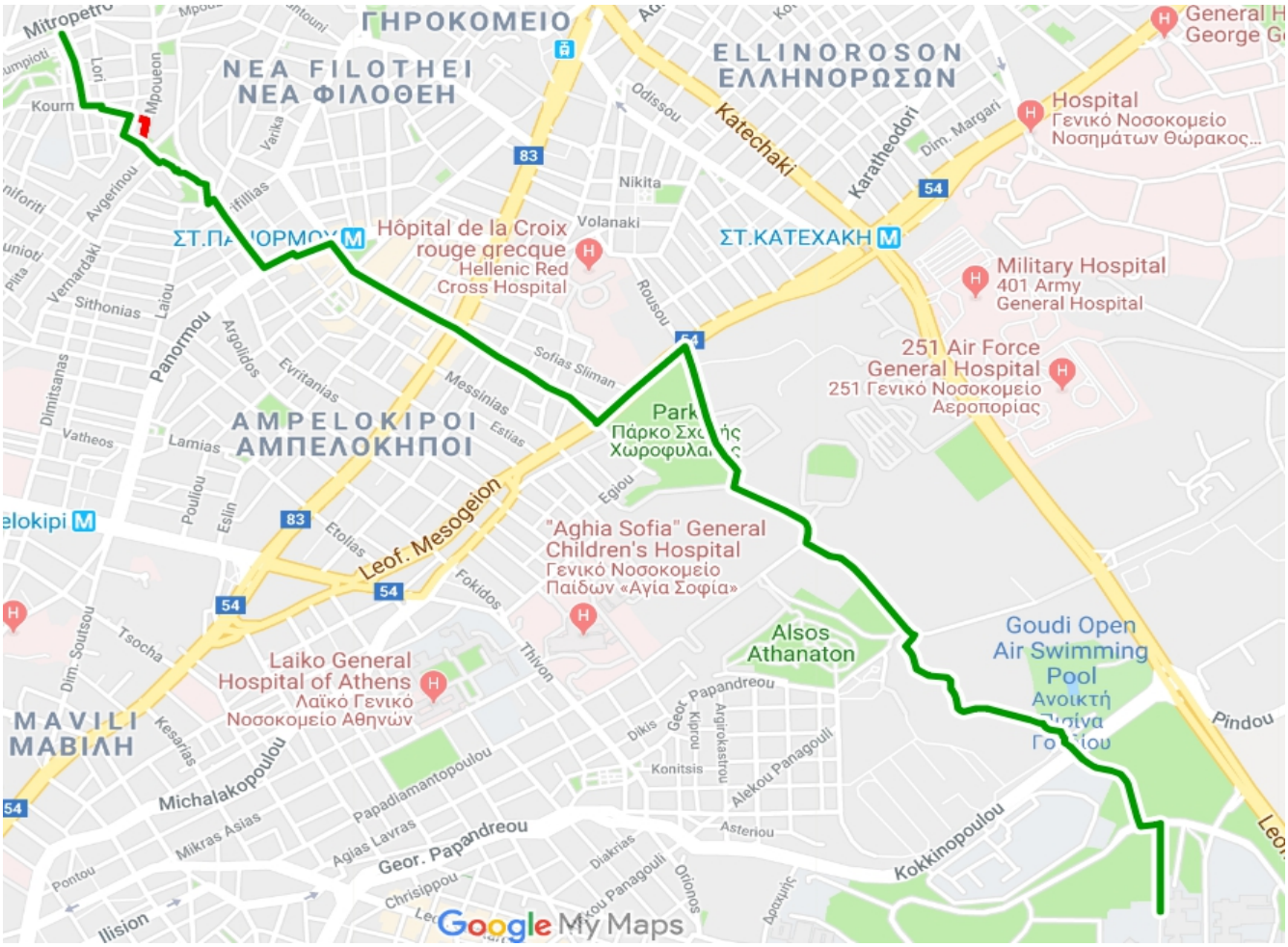
Παρακάτω θα δείξουμε μόνο τη βέλτιστη και τις εναλλακτικές της για διαφορετικές τοποθεσίες του πελάτη.



Με τον πελάτη στην πλ. Ομονοίας ο κώδικας μας παράγει 9 εναλλακτικές διαδρομές για το ταξί με id 100 που επιλέγεται ως βέλτιστο. Αυτό είναι λογικό λόγω της θέσης των σημείων που επιλέχτηκαν.



Πελάτης στους ΗΜΜΥ. Επιλέγεται το ταξί με id 170.



Πελάτης στου Ψυρρή: