ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τεχνητή Νοημοσύνη

Αναφορά 1ου Θέματος

ΟΜΑΔΑ

Γεώργιος-Ορέστης Χαρδούβελης, ΑΜ: 03115100

Ιάσων Λάζαρος Παπαγεωργίου, ΑΜ: 03114034

Στην άσκηση αυτή καλούμαστε να υλοποιήσουμε μία ευφυή υπηρεσία εξυπηρέτησης πελατών ταξί. Μας παρέχονται 3 δεδομένα εισόδου τύπου .csv:

- Το αρχείο **client.csv** που περιέχει τις γεωγραφικές συντεταγμένες του σημείου στο οποίο βρίσκεται ο πελάτης.
- Το αρχείο **taxis.csv** με τον κατάλογο συντεταγμένων και κωδικών των ελεύθερων ταξί.
- Το αρχείο **nodes.csv** το οποίο περιέχει τις γεωγραφικές συντεταγμένες σημείων που ανήκουν σε οδούς, τον κωδικό της κάθε οδούς (όλα τα σημεία της ίδιας οδού έχουν κοινό κωδικό) και σε κάποιες περιπτώσεις το όνομα του δρόμου.

Λαμβάνοντας υπ'όψιν τα παραπάνω δεδομένα καλούμαστε να προσδιορίσουμε το ταξί που χρειάζεται να διανύσει τη μικρότερη απόσταση προκειμένου να φτάσει στον πελάτη. Το βέλτιστο αυτό ταξί θα βρεθεί εφαρμόζοντας τον αλγόριθμο A* για κάθε ένα ταξί πάνω στον γράφο των σημείων του χάρτη, έχοντας ως στόχο την τοποθεσία του πελάτη.

Ανάλυση λύσης

Η εφαρμογή της λύσης του προβλήματος έγινε προγραμματιστικά χρησιμοποιώντας τη γλώσσα JAVA.

Ο κορμός του προγράμματος μας είναι οι εξής κλάσεις:

- **client:** Η κλάση του πελάτη. Όπως αναφέρθηκε αποτελεί τον τελικό προορισμό των taxi οπότε και τον στόχο του A*. Στα πεδία της κλάσης αυτής αποθηκεύονται οι συντεταγμένες της θέσης του πελάτη.
- **taxi:** Η κλάση των taxi. Στα πεδία της κλάσης αυτής αποθηκεύονται οι συντεταγμένες και ο κωδικός του κάθε ταξί όπως επίσης και η ευκλείδεια απόστασή του από τη θέση του πελάτη.
- **point:** Η κλάση για τα σημεία των δρόμων που περιέχονται στο αρχείο εισόδου. Στα πεδία της αποθηκεύονται οι συντεταγμένες των σημείων και ο κωδικός του δρόμου στον οποίο ανήκουν. Η κλάση **point** περιλαμβάνει επίσης:
    - Δύο δομές τύπου ArrayList για τα σημεία που αποτελούν διασταυρώσεις,
    - Μία μεταβλητή τύπου boolean, η χρησιμότητα της οποίας εντοπίζεται στην εκτέλεση του αλγορίθμου A*.

- **freader:** Η κλάση που χρησιμοποιείται για ανάγνωση από τα τρία αρχεία εισόδου. Περιέχει:
    - Ένα πεδίο τύπου **client** για τον πελάτη,
    - Ένα **ArrayList<taxi>** για τα δεδομένα των ταξί,
    - Ένα **ArrayList<point>** για τα δεδομένα των σημείων,
    - Ένα **ArrayList<point>** στο οποίο αποθηκεύονται τα σημεία - διασταυρώσεις.

Εδώ πρέπει να αλλαχθεί και το path των αρχείων client, taxi και nodes αναλόγως σε ποιον υπολογιστή τρέχει κανείς το πρόγραμμα ή στην περίπτωση που δίνουμα τα δικά μας αρχεία.

- **search_front:** Περιέχει τις ακολουθίες κόμβων προς εξέταση από τον αλγόριθμο A*.
- **Astar:** Η κλάση για την υλοποίηση του A*. Χρησιμοποιεί τις οντότητες τύπου **search_front**.
- **Solution_Best:** Η main μέθοδος του προγράμματος, καλεί τις υπόλοιπες και τυπώνει την βέλτιστη διαδρομή του βέλτιστου ταξί.
- **Solution_All:** Άλλη main μέθοδος του προγράμματος, καλεί τις υπόλοιπες και τυπώνει την βέλτιστη διαδρομή για όλα τα ταξί.

Οι δύο τελευταίες κλάσεις λειτουργούν με όμοιο τρόπο και έχουν ελάχιστες διαφορές. Παρακάτω στην αναφορά αναφερόμαστε κυρίως στην Solution_Best που εντοπίζεται και η βέλτιστη λύση. Στην Solution_All απλά δεν γίνεται σύγκριση με τις αποστάσεις και κάθε φορά που τρέχει ο A για ένα ταξί τυπώνεται το αποτέλεσμα.*
Η λύση ουσιαστικά περιέχεται στην Solution_Best αφου μας δίνει την βέλτιστη διαδρομή στο καταλληλότερο ταξί ενώ η Solution_All είναι απαραίτητη για την αναπαράσταση και των υπόλοιπων ταξί στον χάρτη.

Μεθοδολογία

Ξεκινάμε δημιουργώντας μία οντότητα freader την **state** προκειμένου να διαβάσουμε τα αρχεία και να αποθηκεύσουμε τα δεδομένα στις κατάλληλες δομές για την μετέπειτα επεξεργασία.

Στη συνέχεια, προκειμένου να βρούμε και να αποθηκεύσουμε στο Hashmap **crossnodes** τις διασταυρώσεις, χρησιμοποιούμε τη μέθοδο **checknodes.** Κάθε οντότητα **point** που ανήκει στον πίνακα των διασταυρώσεων, διαθέτει μια λίστα με τους κωδικούς των δρόμων στους οποίους ανήκει, καθώς και μια λίστα με τις θέσεις στις οποίες βρίσκεται το αντίστοιχο σημείο στο γενικό πίνακα των σημείων.

Κατόπιν, ταξινομούμε τον πίνακα των ταξί σε σχέση με την **ευκλείδεια (ευριστική) απόστασή τους** από τη θέση του πελάτη. Η ταξινόμηση αυτή γίνεται για λόγους αποδοτικότητας του αλγορίθμου A* και για εξοικονόμηση χρόνου κατά την εκτέλεσή του, καθώς τα πιο πιθανά ταξί εξετάζονται πρώτα.

Έπειτα, καθώς η θέση του πελάτη δεν είναι απαραίτητο να ταυτίζεται με κάποιο σημείο των δρόμων που μας δίνονται στο αρχειο nodes.csv, βρίσκουμε και στη συνέχεια θέτουμε ώς στόχο το σημείο που απέχει την ελάχιστη απόσταση από τη θέση αυτή. Ομοίως λειτουργούμε για τα ταξί.

Προσδιορίζουμε την απόσταση αφετηρίας – στόχου και, στην περίπτωση που αυτή είναι μικρότερη από τη μέχρι στιγμής βέλτιστη λύση, εκτελούμε τον αλγόριθμο A* για το συγκεκριμένο ταξί.

Ως έξοδος του προγράμματος τυπώνεται για τη βέλτιστη επιλογή, ο κωδικός του ταξί, η απόσταση που θα διανύσει και η διαδρομή που θα ακολουθηθεί.

Περιγραφή υλοποίησης A*

Ο αλγόριθμος **A*** ξεκινά από τον αρχικό κόμβο του προβλήματος και προσθέτει στο μέτωπο αναζήτησης τους γειτονικούς του.

Για κάθε κόμβο υπολογίζει την ποσότητα **f = g + h**, όπου:

- **g** είναι η πραγματική απόσταση του τρέχοντος κόμβου από την αφετηρία
- **h** η ευριστική απόσταση του τρέχοντος κόμβου από τον κόμβο στόχο **(ευκλείδεια απόσταση)**.

Κατόπιν, επιλέγει από το μέτωπο αναζήτησης τον κόμβο με το ελάχιστο f και ακολουθεί την ίδια διαδικασία μέχρι την προσέγγιση του στόχου.

Κατά την εκτέλεση του αλγορίθμου A* γίνεται χρήση της κλάσης **search_front**, η ανάλυση της οποίας έχει ως εξής:

- ❏ Πεδίο τύπου point cur_node: ο τρέχων κόμβος υπό εξέταση, του οποίου οι γειτονικοί του πρόκειται να προστεθούν στο μέτωπο αναζήτησης
- ❏ Πεδίο τύπου point goal_node: ο τελικός κόμβος - στόχος του προβλήματος

- ❏ Πεδίο τύπου ArrayList<point> all_nodes: λίστα με όλα τα σημεία των δρόμων που περιέχονται στο αρχείο εισόδου
- ❏ Πεδίο τύπου Map<point,point> crosses: αποτελεί την υλοποίηση του Hashmap στο οποίο αποθηκεύονται τα σημεία που αποτελούν διασταυρώσεις δρόμων. Μας επιτρέπει τον γρήγορο έλεγχο του κατά πόσο ο κόμβος υπό εξέταση αποτελεί διασταύρωση, καθώς και τον προσδιορισμό των οδών που διασταυρώνονται εκεί.
- ❏ Πεδίο τύπου double **g_distance:** η πραγματική απόσταση μεταξύ του τρέχοντος κόμβου και της αφετηρίας
- ❏ Πεδίο τύπου double **h_distance:** η ευριστική τιμή της απόστασης του τρέχοντος κόμβου από τον στόχο, προκειμένου να αξιοποιηθεί από τον αλγόριθμο A*. Για την εκτίμηση της συγκεκριμένης τιμής χρησιμοποιήθηκε η ευκλείδεια απόσταση μεταξύ των δύο κόμβων, καθώς υπολογίζεται εύκολα και ικανοποιεί τον περιορισμό της μικρότερης ή ίσης τιμής από την πραγματική, προκειμένου να εκτελείται απρόσκοπτα ο αλγόριθμος A*.
- ❏ Πεδίο τύπου int **pos:** η θέση(index) του τρέχοντος κόμβου στον πίνακα all_nodes
- ❏ Πεδίο τύπου **search_front previous:** περιέχει τον κόμβο από τον οποίο έχουμε οδηγηθεί στον τρέχοντα, προκειμένου να είναι εύκολη η αναθεώρηση της διαδρομής, σε περίπτωση που αποδειχθεί λανθασμένη.
- ❏ Μέθοδος **check_neighbours():** η συνάρτηση που επεκτείνει τον τρέχων κόμβο. Αυτό γίνεται εξετάζοντας τα σημεία που ανήκουν στον ίδιο δρόμο με τον τρέχοντα κόμβο και βρίσκονται μια θέση πριν και μια θέση στον πίνακα. Αν αυτά τα σημεία δεν έχουν ακόμη επιλεγεί βέλτιστα, δημιουργεί για το καθένα μια οντότητα **search_front** και τα προσθέτει στη λίστα.
- ❏ Πεδίο τύπου boolean **susan**: ελέγχει αν είμαστε σε σημείο από το οποίο ξεκινάνε πάνω από ένα διαφορετικά βέλτιστα μονοπάτια
- ❏ Πεδίο τύπου boolean **jason**: ελέγχει αν είμαστε σε σημείο όπου τελειώνουν / συμπίπτουν τα δύο παραπάνω βέλτιστα μονοπάτια.

Στην περίπτωση που ο τρέχων κόμβος είναι διασταύρωση, τότε ελέγχονται και τα σημεία που έχουν τις ίδιες συντεταγμένες με αυτό αλλά διαφορετικό κωδικό δρόμου. Τελικά επιστρέφει μια λίστα, η οποία περιέχει όλα τα πιθανά επόμενα βήματα από τον τρέχοντα κόμβο.

- ❏ Μέθοδος **calculate_f():** συνάρτηση που υπολογίζει κι επιστρέφει την συνολική τιμή της απόστασης, ως άθροισμα της πραγματικής και της ευριστικής.
- ❏ Μέθοδος **success():** επιστρέφει true, έχουμε φτάσει στον κόμβο- στόχο, διαφορετικά false
- ❏ Μέθοδος **printit():** συνάρτηση που τυπώνει τη διαδρομή από την αφετηρία έως τον τρέχοντα κόμβο. Καλείται αναδρομικά για τον previous κόμβο κάθε οντότητας

και κατόπιν τυπώνει τα στοιχεία του cur_node. Καλείται από τη Solution_Best() και Solution_All() στο τέλος του προγράμματος για να εμφανίσει τη βέλτιστη διαδρομή έως τον πελάτη.
Ακόμη, με βάση ορισμένες boolean μεταβλητές (susan και jason) τυπώνει πότε ξεκινάει ένα εναλλακτικό μονοπάτι (βρισκεται σε σχόλιο αλλά μπορεί να μπει για να φανεί πότε ξεκινάνε τα διαφορετικά μονοπάτια και τυπώνει DIFFERENT WAYS COMING UP) καθώς και κάθε πότε τελειώνει ένα εναλλακτικό μονοπάτι (και τυπώνει END OF WAY).

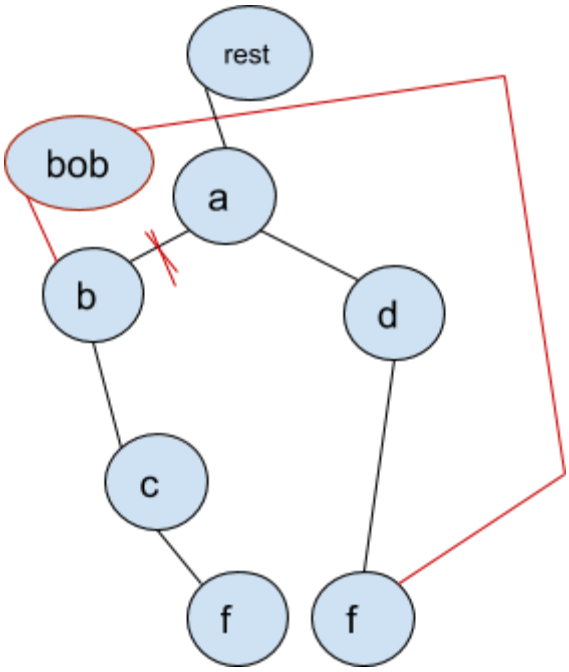
Ο αλγόριθμος **A*** υλοποιείται από την κλάση **Astar** με τον ακόλουθο τρόπο:

- Πεδίο **to_do:** Το μέτωπο αναζήτησης. Είναι τύπου ArrayList<search_front> και αρχικοποιείται στο search_front του κοντινότερου σε κάθε ταξί κόμβου.
- Πεδίο **best_so_far:** δηλωτικό της βέλτιστης μέχρι στιγμής λύσης. Κατά την εκκίνηση του προγράμματος αρχικοποιείται στην μέγιστη δυνατή τιμή.

Ο αλγόριθμος εκτελείται επιλέγοντας από τη λίστα to_do το στοιχείο με τη μικρότερη συνολική απόσταση και διαγράφοντας το, μέχρι να μην υπάρχει κάποιος κόμβος προς εξέταση, ενώ παράλληλα ελέγχουμε για κάθε κόμβο αν υπάρχουν εναλλακτικές διαδρομές μέχρι αυτόν, όπως αναλύεται παρακάτω.
Αν έχει προσεγγιστεί ο κόμβος – στόχος, τότε επιστρέφεται η ακολουθία κόμβων που οδηγούν σ' αυτόν. Σε αντίθετη περίπτωση, ελέγχεται αν ο τρέχων κόμβος έχει ήδη επεκταθεί, οπότε η εκτέλεση συνεχίζεται στο επόμενο στοιχείο του μετώπου αναζήτησης. Εάν ο κόμβος δεν έχει επεκταθεί, θέτει το πεδίο του, dev, στην τιμή true, και αν η απόσταση του τρέχοντος στοιχείου είναι μικρότερη από τη μέχρι στιγμής βέλτιστη λύση, καλεί τη μέθοδο **check_neighbours**, προκειμένου να προσθέσει τους γειτονικούς κόμβους στο μέτωπο αναζήτησης και επαναλαμβάνει. Εάν η απόσταση ξεπερνά τη βέλτιστη λύση, δεν υπάρχει λόγος περαιτέρω επέκτασης και άρα επιστρέφεται η τιμή null.

Εναλλακτικές διαδρομές

Για να μπορέσουμε να προτείνουμε εναλλακτικές διαδρομές, σε κάθε επανάληψη του A*, όταν επιλέγουμε από το μέτωπο αναζήτησης τον κόμβο με την μικρότερη τιμή της συνάρτηση f, ελέγχουμε και τις υπόλοιπες οντότητες search front στο μέτωπο αναζήτησης και εντοπίζουμε αυτές που τελικά δείχνουν στον ίδιο κόμβο με το ίδιο κόστος. Έπειτα, βρίσκουμε την πρώτη οντότητα search front που είναι απόγονος των παραπάνω. Τότε δημιουργούμε ένα πανομοιότυπο search front αντικείμενο, τον bob, και συνδέουμε το παιδί του κοινού απογόνου με τον bob, όπου έχει πατέρα το αμέσως



προηγούμενο search front από τον κοινό απόγονο σε μία από τις διαδρομές και παιδί το τελευταίο search front στην άλλη διαδρομή που έχει ίδια τιμή με αυτό που επιλέξαμε να εξερευνήσουμε από το μέτωπο αναζήτησης.
Έτσι, έχουμε σειριακά και όλα τα διαφορετικά μονοπάτια. Με boolean μεταβλητές στην κλάση search nodes (jason, susan) μπορούμε και αναγνωρίζουμε τα σημεία από όπου ξεκινάνε παραπάνω από μια διαφορετικές διαδρομές και που τελειώνει η κάθε μια.
Αφού εντοπιστεί το εναλλακτικό μονοπάτι και ενσωματωθεί στο “μονοπάτι προηγούμενων κόμβων”, συνεχίζεται ο αλγόριθμος κανονικά όπως περιγράφηκε παραπάνω.

Στην πράξη για να παραχθούν ισοδύναμες διαδρομές πρέπει να υπάρχουν παραπάνω από ένας τρόποι να φτάσεις στον ίδιο κόμβο με το ίδιο κόστος. Η τιμή αυτή εξαρτάται βέβαια και από την χαρτογράφηση με nodes που έχει γίνει (πόσο πυκνή ή αραιή είναι) καθώς και ο τύπος αριθμού που είναι η απόσταση (πχ άμα είναι αποθηκευμένο σε float δύσκολα θα παραχθούν ακριβώς ίσες τιμές).
Ο αλγόριθμος όπως έχει δομηθεί βρίσκει όλες τις βέλτιστες διαδρομές αφού ψάχνει όλους τους ίσους κόμβους που έχουν ίσα μέχρι στιγμής μονοπάτια. Παράλληλα, ο αλγόριθμοσμας βρίσκει πάντα την βέλτιστη / βέλτιστες λύσεις αφού η ευριστική μας συνάρτηση (ευκλείδεια απόσταση) είναι συνεπής (πάντα μικρότερη ή ίση της πραγματικής απόστασης).

Βέβαια, στο συγκεκριμένο πρόβλημα δεν υπολογίζουμε παράγοντες όπως το ρεύμα των δρόμων, την κίνηση καθώς και κάποια φυσικά εμπόδια, οπότε μία ρεαλιστική εφαρμογή θα ήταν αρκετά διαφορετική.

Αποτελέσματα

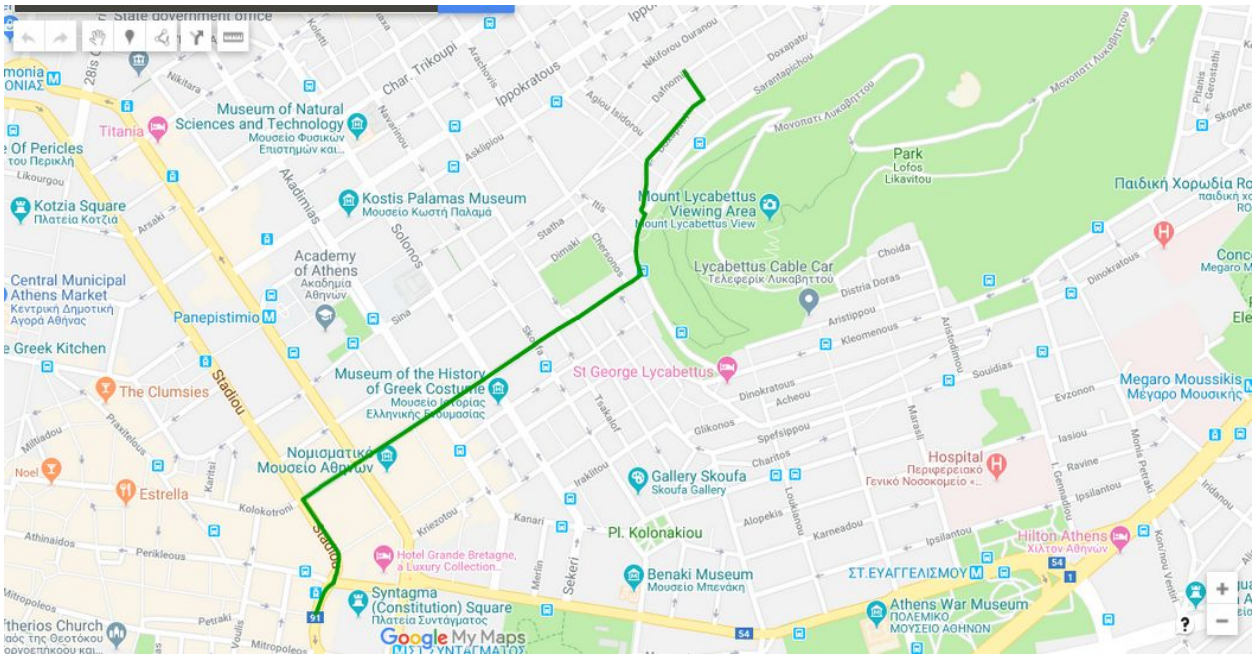
- **Δοθέν παράδειγμα**

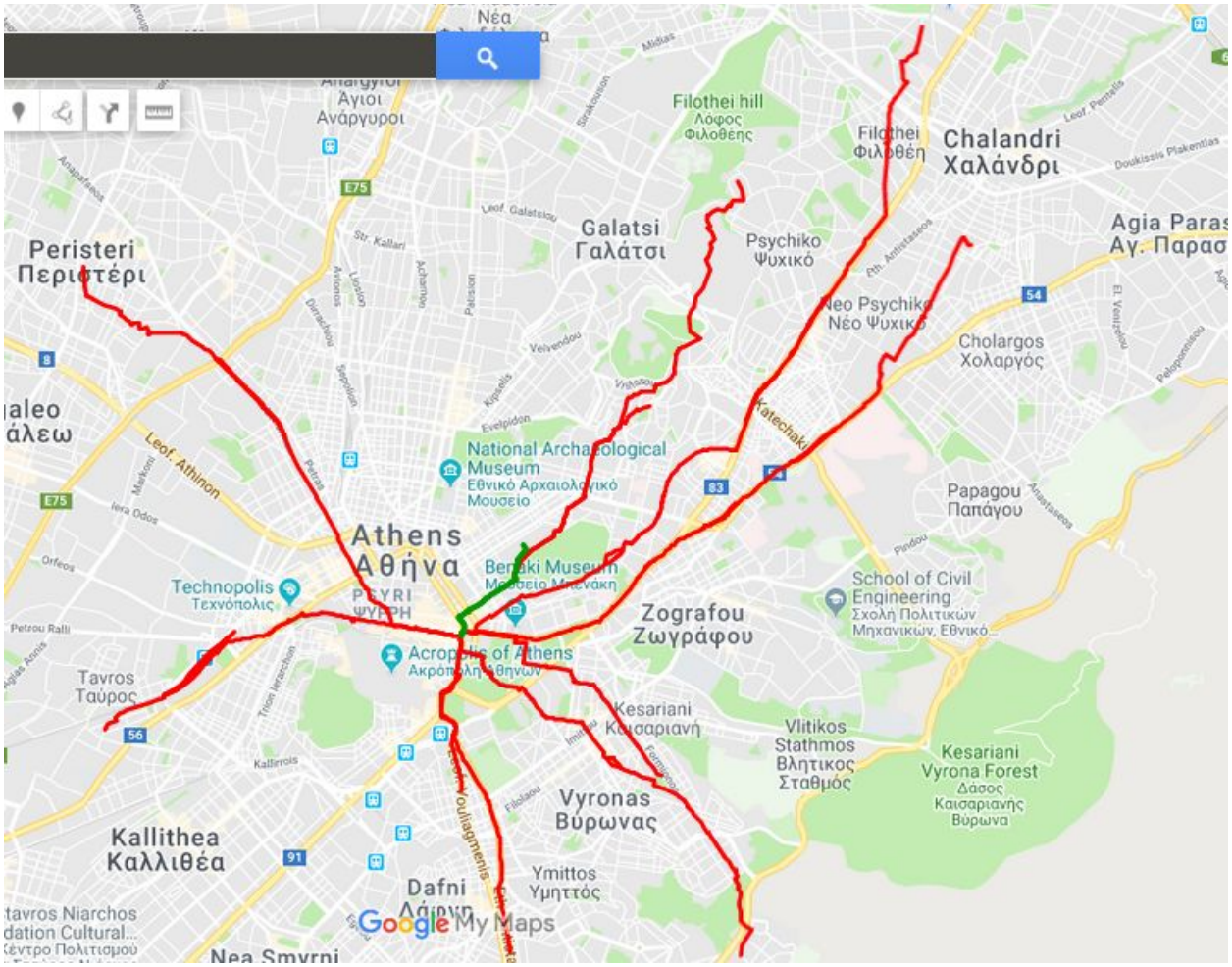
Εδώ παραθέτουμε τα αποτελέσματα που προέκυψαν από το δοθέν παράδειγμα.

Τα kml αρχεία για την απεικόνιση της βέλτιστης και όλων των διαδρομών βρίσκονται στα αρχεία best route και all routes αντίστοιχα. Η βέλτιστη απεικονίζεται με πράσινο χρώμα ενώ οι υπόλοιπες με κόκκινο. Για την απεικόνιση στον χάρτη, στα σημεία όπου τυπώνεται END OF WAY, δηλαδή τελειώνει ένα διαφορετικό βέλτιστο μονοπάτι (σε κομμάτια διαδρομής που υπήρχαν πάνω από ένα) βάλαμε να απεικονίζεται θεωρητικά άλλο ταξί, για να μην ενώνει τους τα άκρα των 2 διαφορετικών μονοπατιών στον χάρτη.

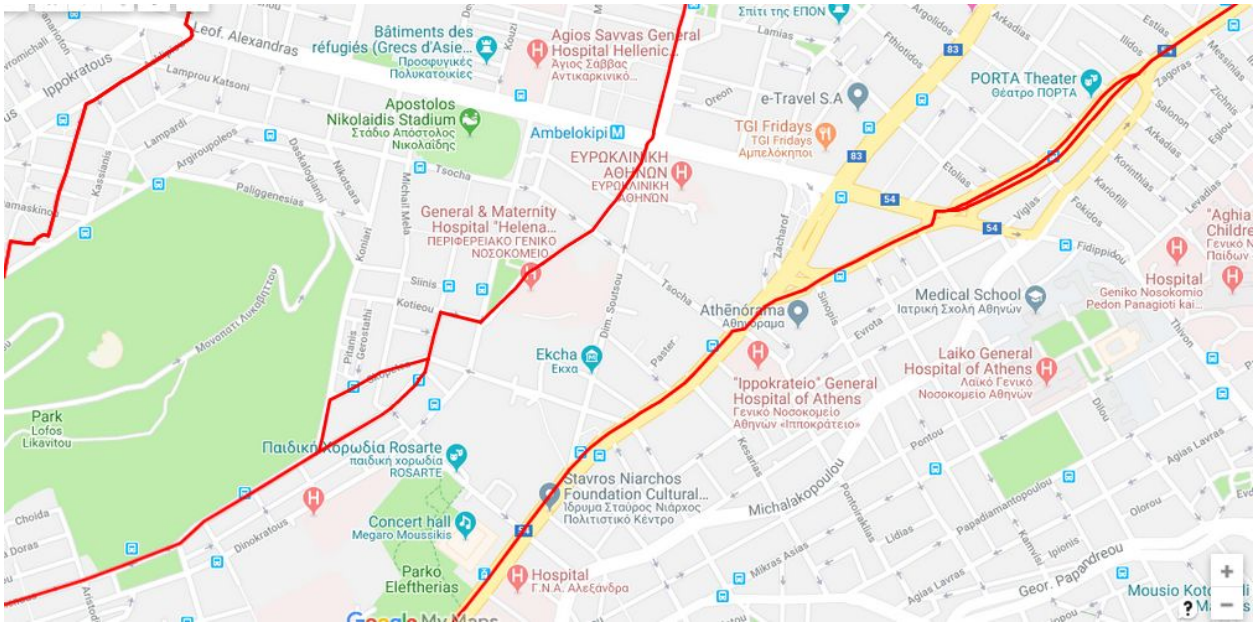
Παράλληλα κάποιοι κόμβοι στο kml αρχείο εμφανίζονται πάνω από μία φορές. Αυτό συμβαίνει αφού όταν υπάρχει διασταύρωση ο κόμβος εμφανίζεται παραπάνω από μία φορές στον χάρτη όποτε τυπώνεται και τόσες φορές.

Ως καταλληλότερο ταξί επιλέγεται αυτό με id 170.





Βλέπουμε πως σε ορισμένα σημεία κάποιων διαδρομών έχουν τυπωθεί παραπάνω εναλλακτικές διαδρομές, όπως εδώ.

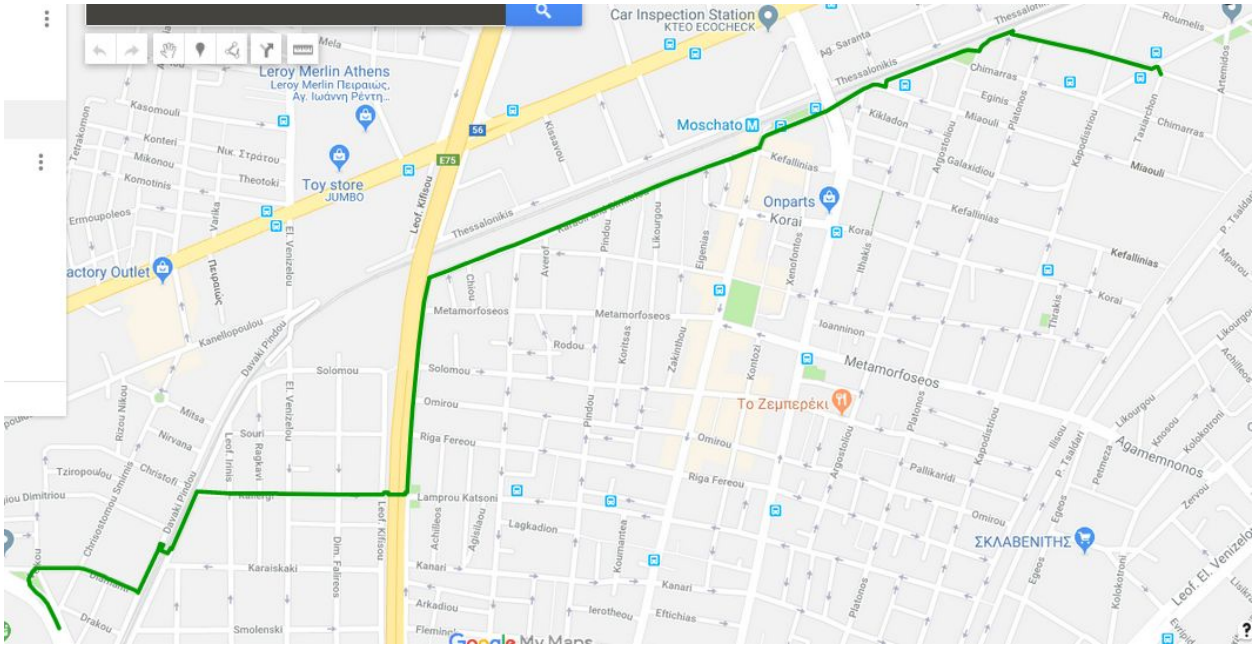


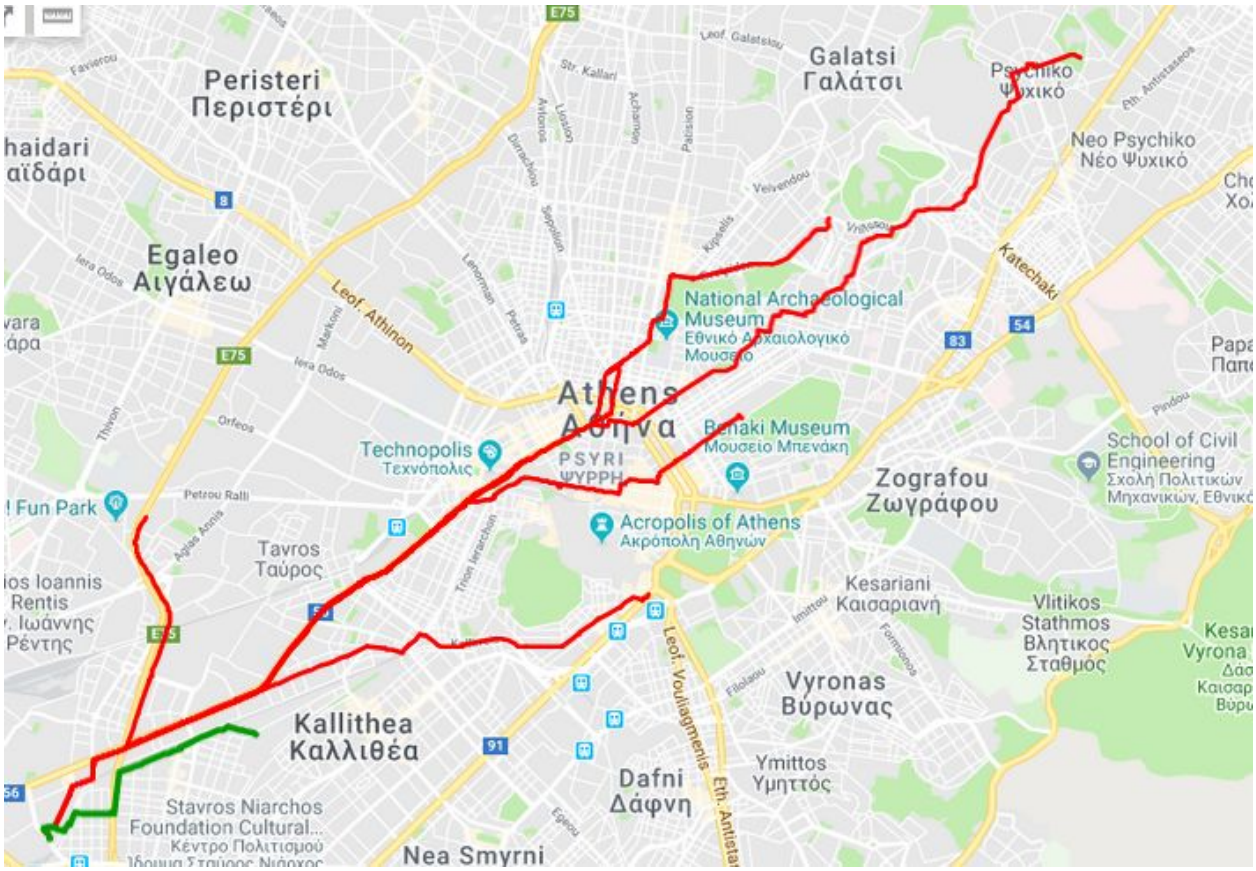
- **Δικό μας παράδειγμα**

Εδώ έχουμε τα αποτελέσματα από το παράδειγμα που δημιουργήσαμε εμείς.
Τα csv αρχεία που αλλάξαμε και αφορούν την θέση του πελάτη και τον ταξί είναι τα client2.csv και taxis2.csv αντίστοιχα.

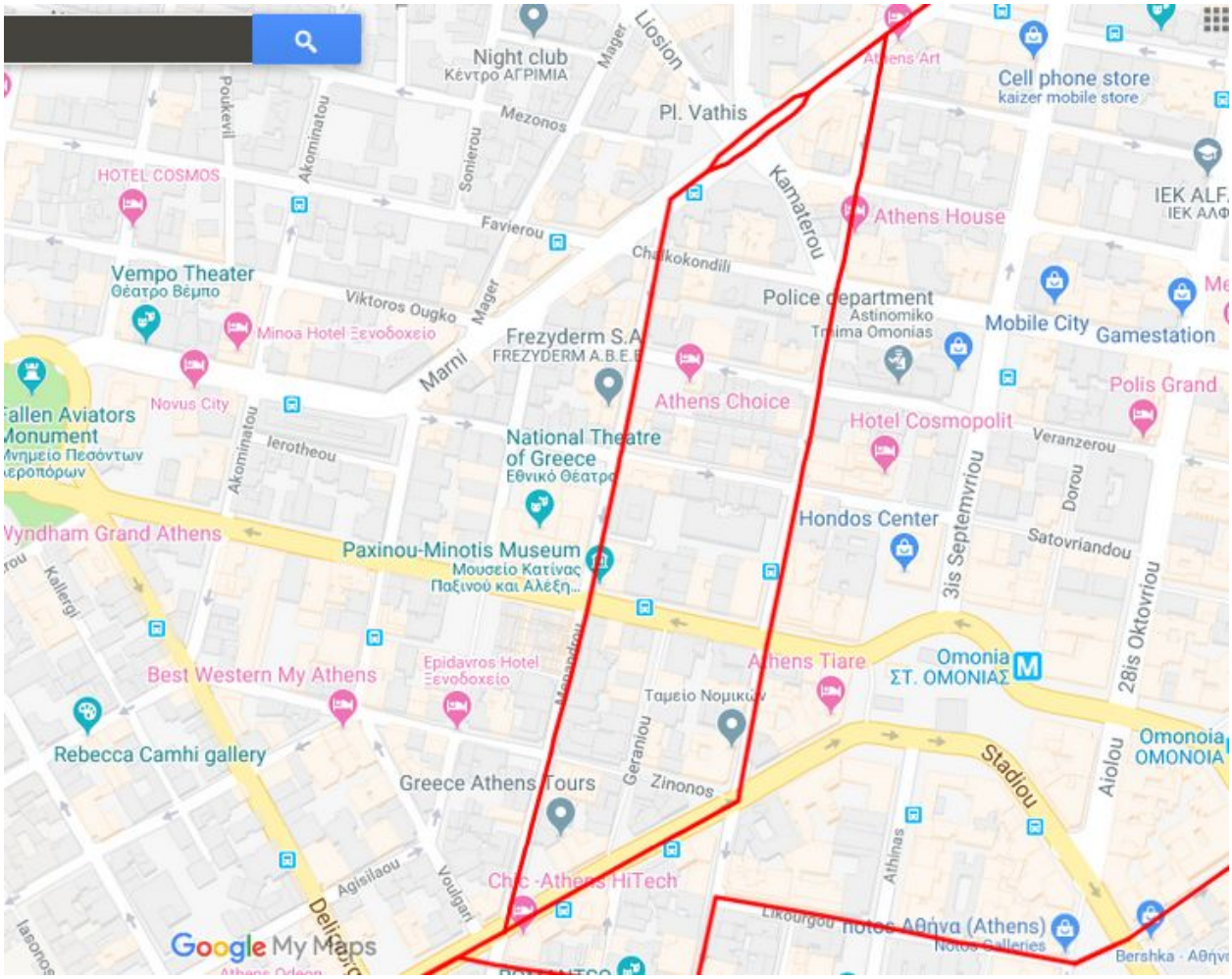
Τα kml αρχεία για την απεικόνιση της βέλτιστης και όλων των διαδρομών βρίσκονται στα αρχεία best route (new) και all routes (new) αντίστοιχα. Η βέλτιστη απεικονίζεται με πράσινο χρώμα ενώ οι υπόλοιπες με κόκκινο.

Ως καταλληλότερο ταξί επιλέγεται αυτό με id 170.





Και εδώ βλέπουμε πως σε ορισμένα σημεία δίνονται εναλλακτικές ίσου κόστους διαδρομές.


Night club
Κέντρο ΑΓΡΙΜΙΑ
Mager
Liosion
Pl. Vathis
Athens Art
Cell phone store
kaizer mobile store
Mezonos
Polukevil
Akominatou
Sonierou
HOTEL COSMOS
Favierou
Kamaterou
Athens House
IEK ALF
Chalkokondili
Vempo Theater
Θέατρο Βέμπο
Viktoros Ougko
Mager
Police department
Astinomiko
Tmima Omonias
Mobile City
Gamestation
Minoa Hotel Ξενοδοχείο
Frezyderm S.A.
FREZYDERM A.B.E.E.
Marni
Novus City
Athens Choice
Polis Grand
Hotel Cosmopolit
Veranzerou
Fallen Aviators
Monument
Μνημείο Πεσόντων
Αεροπόρων
Akominatou
Ierotheou
National Theatre
of Greece
Εθνικό Θέατρο
Dorou
Hondos Center
Wyndham Grand Athens
Paxinou-Minotis Museum
Μουσείο Κατίνας
Παξινού και Αλέξη...
Satovriandou
3is Septemvriou
Kallergi
Best Western My Athens
Epidavros Hotel
Ξενοδοχείο
Athens Tiare
Omonia
ΣΤ. ΟΜΟΝΙΑΣ
28is Oktovriou
Μενάνδρου
Ταμείο Νομικών
Rebecca Camhi gallery
Geraniou
Zinonos
Omonoia
OMONOIA
Stadiou
Aiolou
Greece Athens Tours
Agisilaou
Voulgari
Athinas
Chic -Athens HiTech
Likourgou
notos Αθήνα (Athens)
Notos Galleries
Bershka · Αθήνα
Google My Maps