ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Τεχνητή Νοημοσύνη 2018-2019

Θέμα 1

Στυλιανίδης Χρήστος - 03113130

Στο παρόν θέμα στόχος ήταν να σχεδιαστεί ο κορμός μιας ευφυούς υπηρεσίας εξυπηρέτησης πελατών ταξί βασισμένος στον αλγόριθμο A* (στη γλώσσα Java). Τα στοιχεία των ταξί, τα στοιχεία του πελάτη και οι συντεταγμένες των σημείων των οδών παρέχονται σε csv αρχεία. Το πρόγραμμα διαβάζει τα αρχεία αυτά, επεξεργάζεται τα δεδομένα τους, αποφασίζει ποιο ταξί είναι το πλέον κατάλληλο και κατασκευάζει αρχεία kml τα οποία εφόσον μεταφορτωθούν στην ιστοσελίδα https://www.google.com/maps/d απεικονίζουν την διαδρομή κάθε ταξί στο χάρτη. Λόγω δυσκολιών που προέκυψαν στον κώδικα ο αλγόριθμος A* εκτελείται στην συνηθισμένη του μορφή, δηλαδή προτείνει μία βέλτιστη διαδρομή για κάθε ταξί.

Παρακάτω παρουσιάζονται οι κλάσεις του προγράμματος.

Point: Μία κλάση που αναπαριστά ένα σημείο, περιέχει τις μεταβλητές τύπου double x και y καθώς και την μέθοδο distance ώστε να υπολογίζει την απόσταση μεταξύ δύο points σε μέτρα.

Taxi: Η κλάση στην οποία αποθηκεύουμε τις πληροφορίες κάθε ταξί. Περιέχει μία μεταβλητή τύπου Point (coords) και μία τύπου int (id). Με τη μέθοδο findTaxiNode βρίσκουμε το κοντινότερο στο ταξί node. Δεν μπορούμε να θεωρήσουμε το ίδιο το ταξί σαν ξεχωριστό node διότι σε περίπτωση που αυτό βρίσκεται μακριά από τα υπόλοιπα nodes η διαδρομή από το ταξί στο πλησιέστερο node δεν θα είναι σωστή (πιθανώς να περνάει μέσα από κτίρια π.χ.).

Node: Η κλάση που αντιπροσωπεύει έναν κόμβο πάνω στον χάρτη. Σε τέτοιους κόμβους θα τρέξει ο αλγόριθμος A*. Περιέχει μια μεταβλητή τύπου Point (coords), μία τύπου int (id) και δύο τύπου double (fScore, gScore) οι οποίες χρησιμοποιούνται από τον αλγόριθμο A* για την εύρεση της βέλτιστης διαδρομής. Περιέχει τη μεθόδο distance που λειτουργεί σαν αυτήν της κλάσης Point και κάνει override τις μεθόδους hashCode και equals προκειμένου να μπορούμε να φτιάξουμε HashMaps με κλειδιά Nodes.

NodeComparator: Μία κλάση που που μας προσφέρει δυνατότητα compare σε αντικείμενα τύπου Node ώστε να φτιάξουμε μια priority queue με αυτά.

Pair: Μία κλάση που πακετάρει μαζί μία μεταβλητή τύπου HashMap<Node, Node> (hashmap) και μία τύπου double (distance). Την χρησιμοποιούμε επειδή θέλουμε να επιστρέφουμε δύο αποτελέσματα από τον A*.

InputReader: Η κλάση αυτή εκτελεί το διάβασμα των αρχείων csv και την προεπεξεργασία των δεδομένων μέσω της συνάρτησης runInputReader (που καλεί άλλες συναρτήσεις της κλάσης). Πρώτα διαβάζει την θέση του χρήστη, έπειτα τις θέσεις των ταξί και στη συνέχεια τις θέσεις των κόμβων που αντιπροσωπεύουν τις οδούς. Για την τελευταία αυτή εργασία χρησιμοποιεί ένα HashMap<Node, ArrayList<Node>>(hashmap) με τον εξής τρόπο: Ο χάρτης απεικονίζει κάθε κόμβο στη λίστα γειτόνων του. Για κάθε κόμβο που διαβάζει ελέγχει καταρχάς αν αυτός ο κόμβος υπάρχει ήδη στο hashmap, δηλαδή πρόκειται για διασταύρωση. Αν υπάρχει τότε αφαιρείται από το hashmap, ενημερώνεται με νέους γείτονες και έπειτα ξαναπροστίθεται, διαφορετικά απλώς προστίθεται στο hashmap. Ο έλεγχος για το αν δύο διαδοχικοί κόμβοι είναι γειτονικοί γίνεται συγκρίνοντας τα id τους. Αν αυτά είναι ίδια τότε θεωρούνται γείτονες. Καθώς διαβάζει τους κόμβους η συνάρτηση ψάχνει και για τον πλησιέστερο στον πελάτη κόμβο τον οποίο κρατάει στην μεταβλητή client τύπου Node. Η χρήση HashMap επιλέχτηκε προκειμένου

να έχουμε σχεδόν σταθερό χρόνο αναζήτησης και εισαγωγής ενός Node και επομένως περίπου γραμμικό χρόνο για την προεπεξεργασία όλων των Nodes.
Σημείωση: Παρατήρησα πως τα csv αρχεία μερικές φορές παρουσιάζουν προβλήματα κατά την ανάγνωση. Επειδή τα αρχεία από το mycourses δεν είχαν newline στην τελευταία γραμμή έκανα το ίδιο και για τα δικά μου csv και διαβάστηκαν κανονικά, επίσης χρησιμοποίησα κωδικοποίηση UTF-8.

AStar: Η κλάση στην οποία εκτελείται ο αλγόριθμος A* για ένα ζευγάρι ταξί-πελάτη. Η κλάση περιέχει μία μεταβλητή τύπου HashMap<Node, ArrayList<Node>> η οποία είναι αυτή που κατασκευάζουμε στην κλάση InputReader, μία μεταβλητή τύπου Node (client - τον πελάτη), μία μεταβλητή τύπου HashMap<Node,Node> (το closedSet) και μία τύπου PriorityQueue<Node> (το OpenSet). Επιλέξαμε HashMap για το κλειστό σύνολο διότι θέλουμε να μπορούμε σε σταθερό χρόνο να δούμε αν ένας κόμβος βρίσκεται στο κλειστό σύνολο (κάθε κόμβος είναι mapped στον εαυτό του). Το PriorityQueue επιλέχθηκε για το ανοιχτό σύνολο προκειμένου να έχουμε πρόσβαση στο στοιχείο με το μικρότερο f σε σταθερό χρόνο και εισαγωγή στοιχείων σε λογαριθμικό. Η εύρεση τυχαίου στοιχείου στην PriorityQueue παίρνει γραμμικό χρόνο όπως και σε ένα ArrayList, επομένως δεν έχουμε κέρδος χρόνου σε αυτή την λειτουργία. Ο αλγόριθμος ο οποίος τρέχει με τη συνάρτηση runAStar είναι ίδιος με αυτόν που παρουσιάζεται π.χ. στην Βικιπέδια και κρατάει για κάθε κόμβο τον γονέα του σε ένα χάρτη HashMap<Node, Node> parents. Επιστρέφει τον χάρτη αυτό μαζί με την απόσταση του ταξί από τον πελάτη.

Taxis_Main: Η main κλάση του προγράμματος. Στην main ορίζεται ένας InputReader που επεξεργάζεται την είσοδο όπως περιγράψαμε. Στη συνέχεια για κάθε ταξί η main δημιουργεί ένα στιγμιότυπο AStar και καλεί τη συνάρτηση runAStar για κάθε ταξί. Κρατάει για κάθε ένα τα αποτελέσματα τόσο της διαδρομής όσο και της απόστασης. Όταν τελειώσει καλεί τη συνάρτηση kml η οποία κατασκευάζει το kml αρχείο από τα HashMaps των αποτελεσμάτων (κόμβοι mapped στους γονείς τους). Προκειμένου να περάσουμε τα αποτελέσματα για όλα τα ταξί μαζί φυλάμε τις διαδρομές (δηλαδή τα HashMaps κόμβων - γονέων) σε ένα ArrayList<HashMap<Node,Node>> (results).

Χάρη στην επιλογή των κατάλληλων δομών δεδομένων το πρόγραμμα χρειάστηκε περίπου 2 δευτερόλεπτα για να ολοκληρωθεί για 1 πελάτη, 10 ταξί και περίπου 154000 αρχικούς κόμβους. Οι εκτίμηση της απόστασης δύο κόμβων έγινε μετατρέποντας τα x και y που δίνονταν στα αρχεία csv σε μέτρα και στη συνέχεια παίρνοντας την Ευκλείδεια απόστασή τους. Το πρόγραμμα εκτελέστηκε για δύο σύνολα εισόδου: Το σύνολο που βρίσκεται στο Mycourses (client.csv, nodes.csv, taxis.csv) από τα οποία παράχθηκε το results.kml και ένα σύνολο δικής μου κατασκευής στο οποίο έχουν αλλάξει η θέση του πελάτη και οι θέσεις των περισσότερων ταξί (client2.csv, nodes.csv, taxis2.csv) από το οποίο παράχθηκε το results2.kml. Προσπάθησα να κάνω τον αλγόριθμο να παρέχει εναλλακτικές διαδρομές επιτρέποντας σε έναν κόμβο να έχει παραπάνω από έναν γονείς όπου ένας νέος γονέας έμπαινε στη λίστα γονέων αν το κόστος μετάβασης είχε μια πολύ μικρή διαφορά από το φθηνότερο (π.χ. 0.1) μιας και ίσο με το φθηνότερο δεν θα μπορούσε να είναι λόγω της ακρίβειας των μεταβλητών τύπου double. Επιπλέον αφού έβρισκα τον κόμβο client συνέχιζα τον A* μέχρι η διαφορά της απόστασης της κεφαλής του PiorityQueue από τον client με την αρχική βέλτιστη απόσταση να ξεπεράσει ένα όριο π.χ. τα 10 μέτρα. Στη συνέχεια διέσχιζα τον γράφο που προέκυπτε με την τεχνική DFS. Παρόλα αυτά κάτι δεν δούλευε καλά στην εκτέλεση επομένως κράτησα την αρχική μορφή του

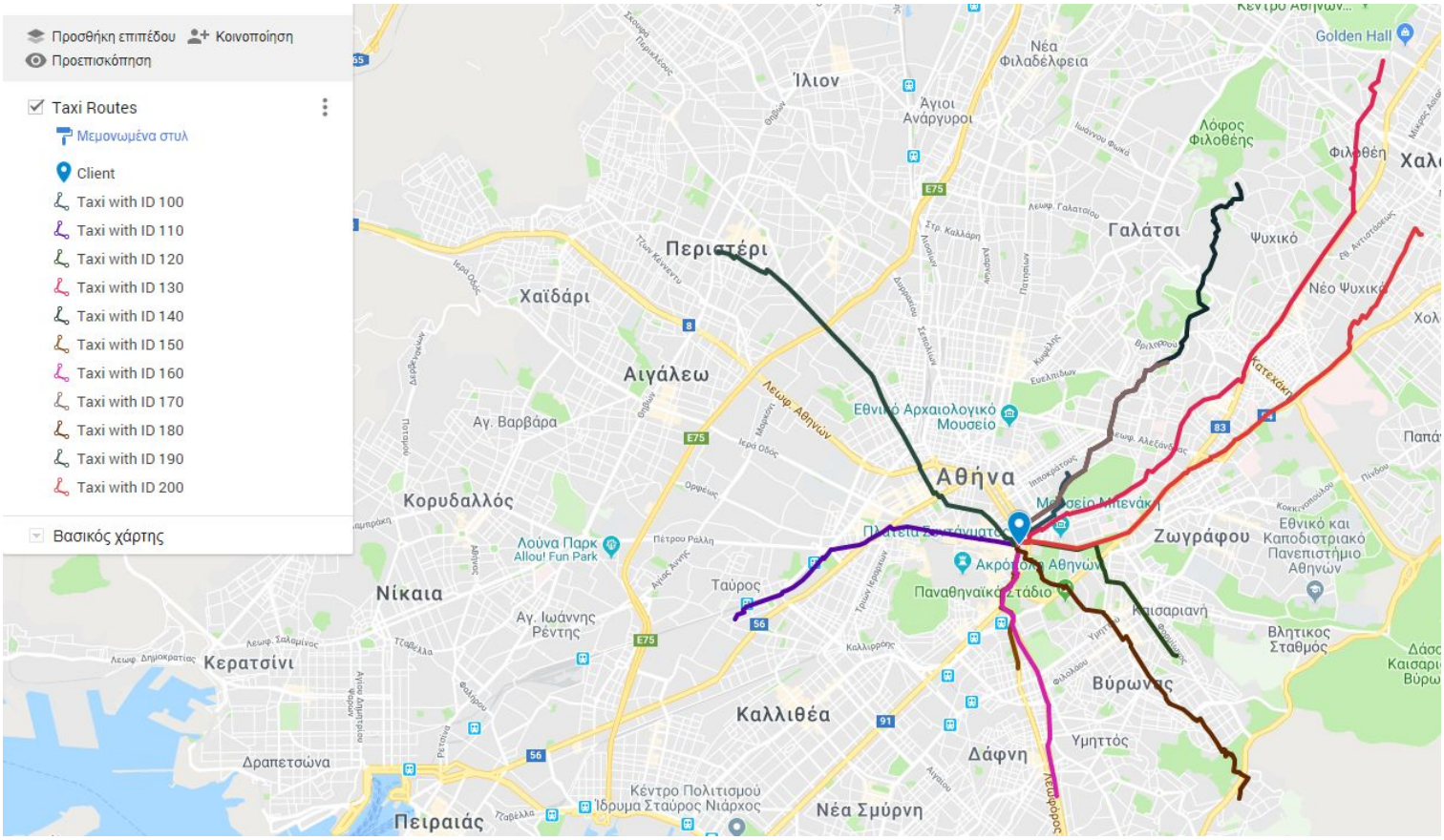
A*. Εάν η προσέγγιση μου είχε δουλέψει τότε ο αλγόριθμος θα μπορούσα να ανακαλύπτει βέλτιστες διαδρομές των οποίων το κόστος θα διέφερε ελάχιστα, το πολύ λίγα μέτρα. Με δεδομένη την ύπαρξη κίνησης στους δρόμους σε πραγματικές συνθήκες, αυτό θα ήταν ικανοποιητικό. Προφανώς στο συγκεκριμένο πρόβλημα λόγο της απλότητάς του δεν χρειαζόμαστε κάτι παραπάνω. Παρακάτω παρουσιάζονται τα αποτελέσματα τόσο στο παράθυρο του IDE όσο και στην υπηρεσία του Google Maps από το τρέξιμο του αλγορίθμου.

Σύνολο 1:

```
run:
InputReader: Started reading client position.
InputReader: Done with client position.
InputReader: Started reading taxi positions.
InputReader: Done with taxi positions.
InputReader: Started reading node positions.
  Client node is 23.733998, 37.9756885, id:23182767
  Starting number of nodes: 154404.
  Final number of nodes after removing duplicates: 104487.
  Distance between client and client node: 9.563982557966932 meters.
InputReader: Done with node creation.
Main: Matching taxis with nodes.
  Taxi with id 100 has distance from closest node: 21.490143946344133 meters.
  Taxi with id 110 has distance from closest node: 61.42093182934069 meters.
  Taxi with id 120 has distance from closest node: 20.205290122846062 meters.
  Taxi with id 130 has distance from closest node: 21.16385530775671 meters.
  Taxi with id 140 has distance from closest node: 11.306579581156173 meters.
  Taxi with id 150 has distance from closest node: 23.047154080013858 meters.
  Taxi with id 160 has distance from closest node: 38.8115335606753 meters.
  Taxi with id 170 has distance from closest node: 9.244787573957808 meters.
  Taxi with id 180 has distance from closest node: 51.507344887208625 meters.
  Taxi with id 190 has distance from closest node: 39.17977533718625 meters.
  Taxi with id 200 has distance from closest node: 13.805256290444373 meters.
Main: Finished matching taxis with nodes.
Main: Beginning initialization of A* Algorithm for all taxis.
Main: A* Algorithm initialized successfully for taxi with ID 100.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 100. - SUCCESS
  Nodes explored: 506, distance to client: 1509.217183943086 meters.
Main: A* Algorithm initialized successfully for taxi with ID 110.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 110. - SUCCESS
  Nodes explored: 2671, distance to client: 5157.299666466182 meters.
Main: A* Algorithm initialized successfully for taxi with ID 120.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 120. - SUCCESS
  Nodes explored: 2260, distance to client: 3418.4964114152494 meters.
Main: A* Algorithm initialized successfully for taxi with ID 130.
```

```
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 130. - SUCCESS
  Nodes explored: 10404, distance to client: 9604.426367385086 meters.
Main: A* Algorithm initialized successfully for taxi with ID 140.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 140. - SUCCESS
  Nodes explored: 9004, distance to client: 7127.832252038256 meters.
Main: A* Algorithm initialized successfully for taxi with ID 150.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 150. - SUCCESS
  Nodes explored: 922, distance to client: 1815.3193320769083 meters.
Main: A* Algorithm initialized successfully for taxi with ID 160.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 160. - SUCCESS
  Nodes explored: 1691, distance to client: 3526.7134513925666 meters.
Main: A* Algorithm initialized successfully for taxi with ID 170.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 170. - SUCCESS
  Nodes explored: 2264, distance to client: 3848.8534824264725 meters.
Main: A* Algorithm initialized successfully for taxi with ID 180.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 180. - SUCCESS
  Nodes explored: 3854, distance to client: 5742.89474270112 meters.
Main: A* Algorithm initialized successfully for taxi with ID 190.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 190. - SUCCESS
  Nodes explored: 2550, distance to client: 6558.459718787603 meters.
Main: A* Algorithm initialized successfully for taxi with ID 200.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 200. - SUCCESS
  Nodes explored: 5772, distance to client: 8490.039643915426 meters.
Main: Exiting A* Algorithm.
  ***Best taxi was taxi with ID 100 with a distance of 1509.217183943086 meters from the client.
Main: Began creation of kml file.
KML: Began writing taxi with ID 100.
KML: Began writing taxi with ID 110.
KML: Began writing taxi with ID 120.
KML: Began writing taxi with ID 130.
KML: Began writing taxi with ID 140.
KML: Began writing taxi with ID 150.
KML: Began writing taxi with ID 160.
KML: Began writing taxi with ID 170.
KML: Began writing taxi with ID 180.
KML: Began writing taxi with ID 190.
KML: Began writing taxi with ID 200.
Main: Finished creation of kml file.
```

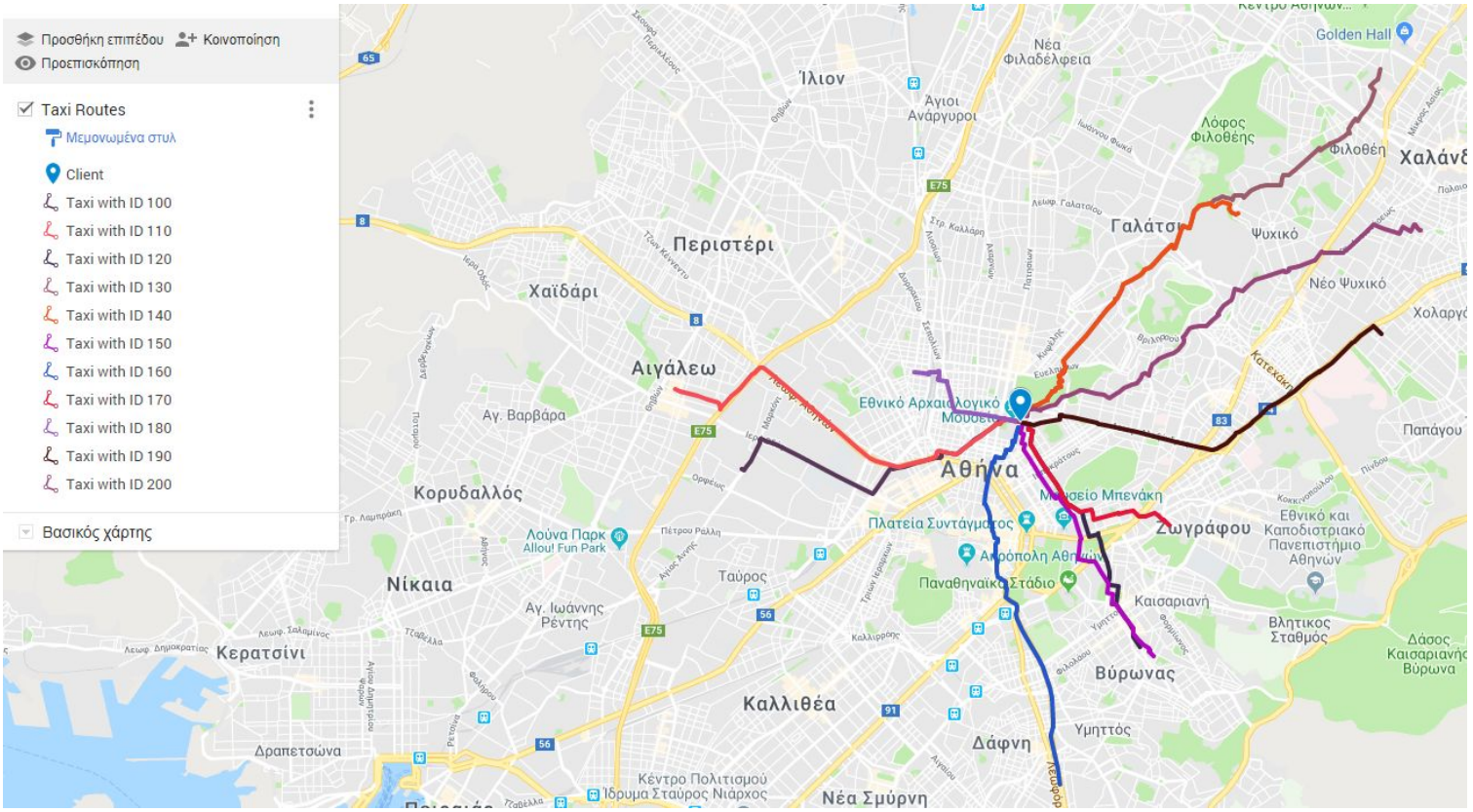
Main: Repeating: ***Best taxi was taxi with ID 100 with a distance of 1509.217183943086 meters from the client.
Main: Exiting now...
BUILD SUCCESSFUL (total time: 2 seconds)



<u>Σύνολο 2:</u>
run:
InputReader: Started reading client position.
InputReader: Done with client position.
InputReader: Started reading taxi positions.
InputReader: Done with taxi positions.
InputReader: Started reading node positions.
  Client node is 23.733645, 37.9895051, id:10971761
  Starting number of nodes: 154404.
  Final number of nodes after removing duplicates: 104487.
  Distance between client and client node: 35.66231813383918 meters.
InputReader: Done with node creation.
Main: Matching taxis with nodes.
  Taxi with id 100 has distance from closest node: 31.062218170594154 meters.
  Taxi with id 110 has distance from closest node: 25.769338606701293 meters.
  Taxi with id 120 has distance from closest node: 26.441364199073057 meters.
  Taxi with id 130 has distance from closest node: 97.30442752368994 meters.
  Taxi with id 140 has distance from closest node: 13.441027288045222 meters.

```
  Taxi with id 150 has distance from closest node: 13.715873708212213 meters.
  Taxi with id 160 has distance from closest node: 38.8115335606753 meters.
  Taxi with id 170 has distance from closest node: 15.960611238670529 meters.
  Taxi with id 180 has distance from closest node: 7.89937807281273 meters.
  Taxi with id 190 has distance from closest node: 45.096005046502526 meters.
  Taxi with id 200 has distance from closest node: 13.805256290444373 meters.
Main: Finished matching taxis with nodes.
Main: Beginning initialization of A* Algorithm for all taxis.
Main: A* Algorithm initialized successfully for taxi with ID 100.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 100. - SUCCESS
  Nodes explored: 2373, distance to client: 5339.37125097079 meters.
Main: A* Algorithm initialized successfully for taxi with ID 110.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 110. - SUCCESS
  Nodes explored: 4913, distance to client: 6592.764908955053 meters.
Main: A* Algorithm initialized successfully for taxi with ID 120.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 120. - SUCCESS
  Nodes explored: 2672, distance to client: 4004.379424720756 meters.
Main: A* Algorithm initialized successfully for taxi with ID 130.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 130. - SUCCESS
  Nodes explored: 10300, distance to client: 8679.233450066724 meters.
Main: A* Algorithm initialized successfully for taxi with ID 140.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 140. - SUCCESS
  Nodes explored: 3677, distance to client: 5434.101517597729 meters.
Main: A* Algorithm initialized successfully for taxi with ID 150.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 150. - SUCCESS
  Nodes explored: 2790, distance to client: 4293.468109525546 meters.
Main: A* Algorithm initialized successfully for taxi with ID 160.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 160. - SUCCESS
  Nodes explored: 6993, distance to client: 5418.138694830825 meters.
Main: A* Algorithm initialized successfully for taxi with ID 170.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 170. - SUCCESS
  Nodes explored: 2080, distance to client: 3435.0696963764017 meters.
Main: A* Algorithm initialized successfully for taxi with ID 180.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 180. - SUCCESS
  Nodes explored: 476, distance to client: 2179.117664221115 meters.
Main: A* Algorithm initialized successfully for taxi with ID 190.
A*: Algorithm started running.
```

A*: Algorithm finished running for taxi with ID 190. - SUCCESS
  Nodes explored: 2971, distance to client: 6690.448666457031 meters.
Main: A* Algorithm initialized successfully for taxi with ID 200.
A*: Algorithm started running.
A*: Algorithm finished running for taxi with ID 200. - SUCCESS
  Nodes explored: 6405, distance to client: 8060.426961825043 meters.
Main: Exiting A* Algorithm.
  ***Best taxi was taxi with ID 180 with a distance of 2179.117664221115 meters from the client.
Main: Began creation of kml file.
KML: Began writing taxi with ID 100.
KML: Began writing taxi with ID 110.
KML: Began writing taxi with ID 120.
KML: Began writing taxi with ID 130.
KML: Began writing taxi with ID 140.
KML: Began writing taxi with ID 150.
KML: Began writing taxi with ID 160.
KML: Began writing taxi with ID 170.
KML: Began writing taxi with ID 180.
KML: Began writing taxi with ID 190.
KML: Began writing taxi with ID 200.
Main: Finished creation of kml file.
Main: Repeating: ***Best taxi was taxi with ID 180 with a distance of 2179.117664221115 meters from the client.
Main: Exiting now...
BUILD SUCCESSFUL (total time: 2 seconds)



Στο παραδοτέο zip περιέχονται επιπλέον ο πηγαίος κώδικας σε Java, τα αρχεία kml που παράχθηκαν, 2 εικόνες από την απεικόνισή τους στο Google Maps και τα αρχεία csv.