Δημήτρης Κελέσης A.M.: 03115037

Γενικός σχεδιασμός Συστήματος

Για τον σχεδιασμό του εν λόγω συστήματος με είσοδο τα δεδομένα όπως αυτά περιγράφονται στα αρχεία .csv χρησιμοποιήσαμε την μεθοδολογία εργασίας που ενδείκνυται στον αλγόριθμο A*. Συγκεκριμένα αρχικά διαβάστηκαν τα δεδομένα εισόδου από τα εν λόγω αρχεία και αποθηκεύτηκαν σε αντικείμενα που ονομάζονται Quad και έχουν ως πεδία τους τέσσερις μεταβλητές, όσα δηλαδή και τα δεδομένα που θα έπρεπε να αντλήσουμε από κάθε γραμμή εισόδου (x, y, id, Όνομα οδού). Τα πεδία αυτά διαμορφώθηκαν κατάλληλα ώστε να συμφωνούν με τις αντίστοιχες τιμές που θα δεχθούν. Στη συνέχεια κατασκευάστηκε η κλάση Graph¡T¿ η οποία δημιουργεί ένα γράφο με κορυφές τύπου T που καθορίζεται από τον προγραμματιστή. Στην συγκεκριμένη περίπτωση κατασκευάσαμε γράφο αποτελούμενο απο Quad(s). Ο αρχικός αυτός γράφος είχε πλήθος κορυφών όσο και το πλήθος των κόμβων αλλά χωρίς να του προσθέσουμε καμία ακμή. Στη συνέχεια καλώντας την μέθοδο CreateGraph με όρισμα τον παραπάνω γράφο αρχίζουμε να εντοπίζουμε τα duplicates, δηλαδή κορυφές που εμφανίζονται δύο η περισσότερες φορές καθώς αποτελούν σημεία που βρίσκονται σε διασταυρώσεις δρόμων. Και κατόπιν να προσθέτουμε ακμές κατάλληλα ώστε τα σημεία που βρίσκονται σε μία οδό να έχουν ως γειτονικά τους μόνο δύο άλλα, αυτά που βρίσκονται δηλαδή μπροστά και πίσω τους καθώς επίσης και τα σήμεία που βρίσκονται πάνω σε κάποια διασταύρωση να έχουν ως γείτονες όλα τα σημεία που βρίσκοντα εκατέροθεν του σε όσους δρόμους αυτά ανήκουν. Δηλαδή ένα σημείο που βρίσκεται στην διασταύρωση δύο δρόμων θα πρέπει να έχει τέσσερις γείτονες, δύο σε κάθε δρόμο.

Για να το επιτύχουμε αυτό κάναμε χρήση της δομής Hashmap της Java. Καθώς θέλαμε να χρησιμοποιήσουμε ως κλειδί της συγκεκριμένης δομής όχι κάποιον τύπο της Java (Integer, Double κλπ.) αλλά ένα αντικείμενο δημιουργήσαμε την κλάση Klidi η οποία έχει ως πεδία της τα x, y και σκοπός της είναι να παίξει τον ρόλο μοναδικού κλειδιού στην δομή Hashmap. Για να το επιτύχουμε αυτό κάναμε Override της μεθόδους equals και hashCode. Έτσι με την χρήση αυτού του κλειδιού μπορούσαμε σε κάθε πιθανή είσοδο αντικειμένου στην Hashmap να ελέγχουμε αν αυτό υπάρχει ήδη και αν υπάρχει σε χρόνο O(1) να το ανακτούμε από την δομή η να έχουμε πρόσβαση σε διάφορα στοιχεία του. Έτσι χρησιμοποιώντας όλα τα παραπάνω κατασκευάσαμε τον τελικό γράφο του προβλήματος. Αυτός αποτελούνταν από μοναδικές κορυφές καθώς χρησιμοποιώντας την δομή Hashmap μπορούσαμε να ελέγχουμε αν μία κορυφή υπο ανήκει σε δύο οδούς έχει ήδη προστεθεί στο γράφημα και έτσι να μην την ξαναπροσθέσουμε. Ταυτόχρονα τις γειτονικές τις κορυφές τις προσθέταμε ως γείτονες στην ήδη υπάρχουσα κορυφή. Αυτό είχε ως συνέπεια αν μία κορυφή ανήκει στην οδό A και B και διαβαστεί και τοποθετηθεί στον γράφο με όνομα οδού το A τότε θα αποκτήσει τους γείτονες των A και B αλλά όχι και το όνομα B, γεγονός που δεν επηρεάζει την λύση μας.

Για την διαχείρηση των αποστάσεων λάβαμε υπόψη μας όταν τα δεδομένα περιέρχουν γεωγραφικές συντεταγμένες και συνεπώς θα έπρεπε να λάβουμε υπόψη μας ότι η απόσταση τους δεν δίνεται απευθείας από τα x y αλλά θα έπρεπε να ληφθούν υπόψη η ακτίνα της Γης και διάφορες γωνίες μετασχηματισμού που ανάγουν την απόσταση σε χιλιομετρική και αντικατοπτρίζουν την πραγματικότητα. Για τον λόγο αυτό δημιουργήθηκε η μέθοδος LatLon που κάνει

την εν λόγω μετατροπή και δίνει ρεαλιστικά αποτελέσματα και ταυτόχρονα τροποποιήθηκε η ευριστική συνάρτηση για τους ίδιους λόγους.
Ως ευριστική ο αλγόριθμος μας χρησιμοποιεί την απόσταση Manhattan τροποποιημένη ώστε να δίνει έγκυρα αποτελέσματα για είσοδο με γεωγραφικά δεδομένα. Η συγκεκριμένη μέθοδος έχει γίνει και Override για να δέχεται τόσο δεδομένα συντεταγμένων απευθείας όσο και αντικείμενα της κλάσης Quad.
Φτάνοντας τώρα στην υλοποίηση του A* παρατηρούμε ότι δεν έχουν τοποθετηθεί βάρη στις αποστάσεις μεταξύ των ακμών ακόμη καθώς επίσης και δεν έχουν οριστεί τα σημεία έναρξης και λήξης του αλγορίθμου. Για τον λόγο αυτό αρχικά βρίσκουμε για κάθε ένα από τα ταξί το κοντινότερο σημείο του από όλες τις κορυφές του γράφου και την απόσταση που έχει να διανύσει για να φτάσει σε αυτό. Στη συνέχεια κάνουμε το ίδιο για την θέση του πελάτη. Ορίζουμε λοιπόν ως έναρξη τον κοντινότερο κόμβο στη θέση του πελάτη και ως στόχο κάθε φορά τον κοντινότερο κόμβο στη θέση του εξεταζόμενου ταξι. Προαφανώς αναζητούμε την ελάχιστη απόσταση που προκείπτει ως άθροισμα της απόστασης μεταξύ των δύο παραπάνω κόμβων και της απόστασης που πρέπει να διανύσει το ταξί για να φτάσει στον προορισμό του, καθώς η απόσταση που πρέπει να κάνει ο πελάτης μέχρι το κοντινότερο σημείο είναι σταθερή για όλα τα ταξί.
Για να υλοποιήσουμε τον A* δημιουργούμε μία ακόμη κλασή την κλάση Node. Αυτή περιέχει τα πεδία Quad,g,h,Klidi,Node που συμβολίζουν το Quad-κορυφή που θα περιέχει ο κόμβος αυτός τα g h όπως αυτά ορίζονται στον A* (το f υπολογίζεται ως το άθροισμα τους εσωτερικά στην κλάση για κάθε κόμβο). Το χαρακτηριστικό κλειδί περιέχοντας τα x, y για να είναι αναγνριστικό στην αναζήτηση του εκάστοτε Node. Και το Node γονέα που υποδηλώνει από ποιο Node προέρχεται. Για τον A* χρησιμοποιήσαμε την δομή PriorityQueue κάνοντας και πάλι Override κάποιες μεθόδους της ώστε να προσαρμοστούν στα δεδομένα του A* και να ελέγχουμε τις ζητούμενες σχετικά με τα f συνθήκες. Ξεκινώντας τον αλγόριθμο εισάγουμε την αρχική θέση υπολογίζοντας στη συνέχεια για κάθε επόμενη τα g, h με χρήση των συναρτήσεων που περιγράφηκαν παραπάνω. Σε κάθε βήμα χαλαρώνουμε την καλύτερη δυνατή κορυφή στο μέτωπο και εισάγουμε στο ανοιχτό τα παιδία της, τοποθετώντας την ίδια στο κλειστό. Όταν βρεθέι ο στόχος τερματίζουμε την διαδικασία και αναζητούμε την βέλτιστη διαδρομή στο κλειστό. Για να το επιτύχουμε αυτό κάνουμε χρήση κυρίως του πεδίου Node parent που υποδηλώνει από που προήλθε ο κάθε κόμβος στην λύση. Τέλος υπολογίζουμε την απαιτούμενη απόσταση για να φτάσουμε στο στόχο. Επαναλαμβάνουμε την διαδικασία για όλα τα ταξί και στο τέλος κρατάμε το μικρότερο δυνατό που είναι και το ζητούμενο. Τέλος γράφουμε σε αρχεία τις συντεταγμένες των σημείων που θα πρέπει να ακολουθηθούν με συγκεκριμένη σείρα.

Σχετικά με τα δεδομένα εισόδου αυτά δεν χρειάστηκαν κάποια προεπεξεργασία καθώς το πρόγραμμα τα διάβασε απευθείας από τα αρχεία εισόδου με τη μόνη λεπτομέρεια ότι η ζητούμενη κωδικοποίηση ηταν greek και όχι UTF-8. Μοντελοποιήθηκαν με την χρήση της προαναφερθείσας κλάσης Quad κυρίως και για όσα από αυτά είχαμε τρεις εισόδους αντι για τέσσερις αφέθηκε ως κενό το τελευταίο πεδίο που κανονικά θα αντιστοιχούσε στο όνομα του δρόμου. Για την θέση του πελάτη αυτή διαβάστηκε απευθείας σε δύο μεταβλητές που υποδήλωναν τα x y αντίστοιχα. Στη συνέχεια μετασχηματίστηκαν σε γράφημα με κορυφές από Quad και τελίκά σε Nodes με επιπλέον χαρακτηριστικά για να εξυπηρετούν τις απαιτήσεις του προβλήματος. Επίσης λαμβάνοντας υπόψη ότι οι δρόμοι είναι διπλής κατευθύνσεως δεν υπήρξαν κάποιοι περιορισμοί σχετικά με την επιτρεπτή κίνηση κάθε οχήματος και τα κόστη σε κάθε ακμή ισοδυναμούσαν με την μεταξύ των σημείων απόσταση πάνω στον χάρτη.

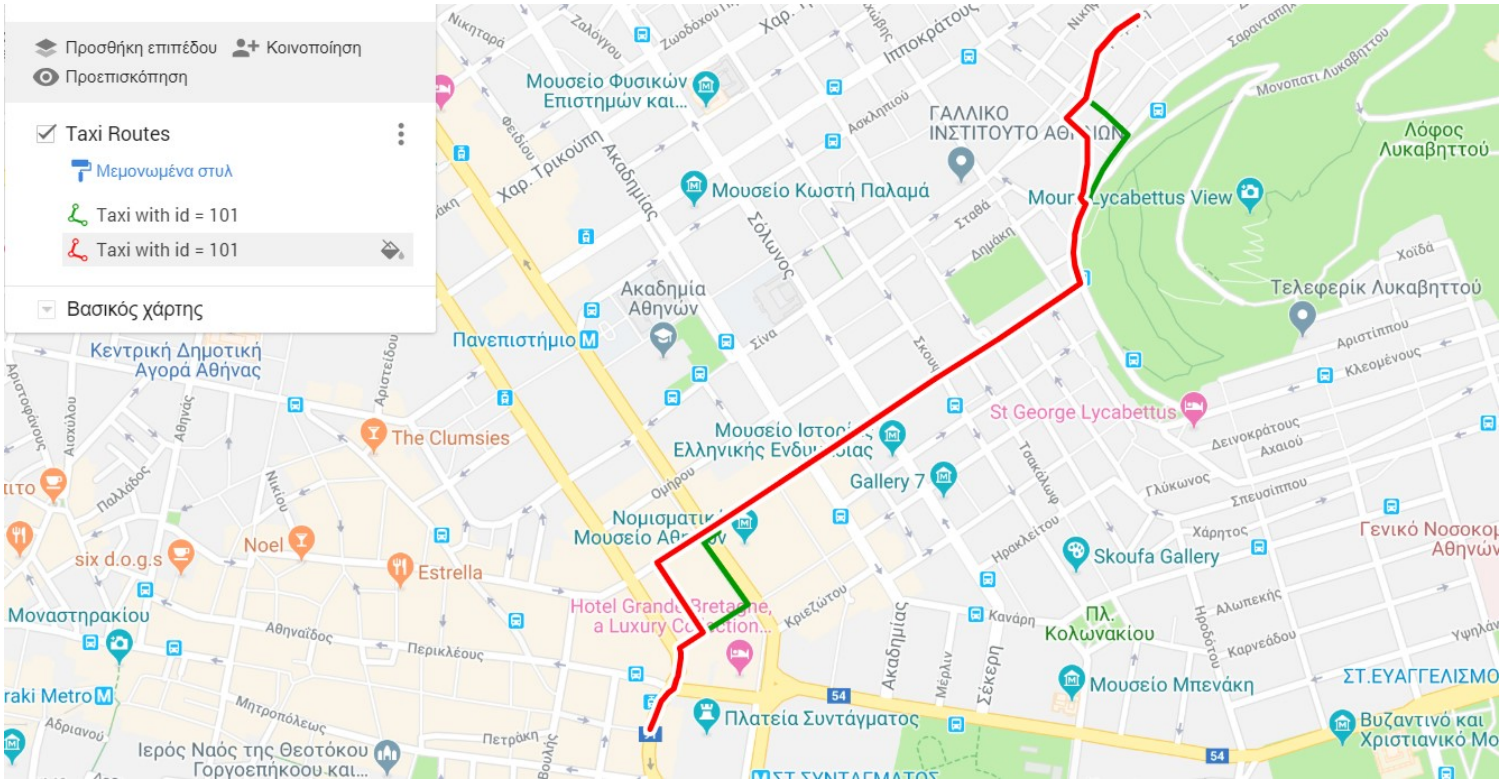
Για την εκτίμηση απόστασης επιλέχθηκε η Manhattan καθώς θεωρήθηκε αποτελεσματικότερη από την Ευκλείδια. Η τροποποίηση της έγινε κατάλληλα για να δέχεται γεωγραφικές συντεταγμένες ακολουθώντας προτάσεις απο πηγές που βρέθηκαν στο διαδίκτυο.

Για τις γεωγραφικές συντεταγμένες αξιοποιήθηκε η μετατροπή σε μετρικό σύστημα καθώς δεν βρισκόμαστε στο Καρτεσιανό και έγινε χρήση του τύπου Haversine. Η συγκεκριμένη φόρμουλα βρίσκεται σε αρκετές βιβλιογραφίες και η ανάπτυξη της φαίνεται στον τρόπο δόμησης της μεθόδου υπολογισμού της απόστασης. Γενικά γίνεται μετατροπή των μοιρών με χρήση σφαιρικών συντεταγμένων σε απόσταση πάνω στον χάρτη και έτσι ανάγουμε το πρόβλημα στο απλό πρόβλημα λαβυρίνθου χωρίς εμπόδια που λαμβάνει χώρα σε ένα grid.

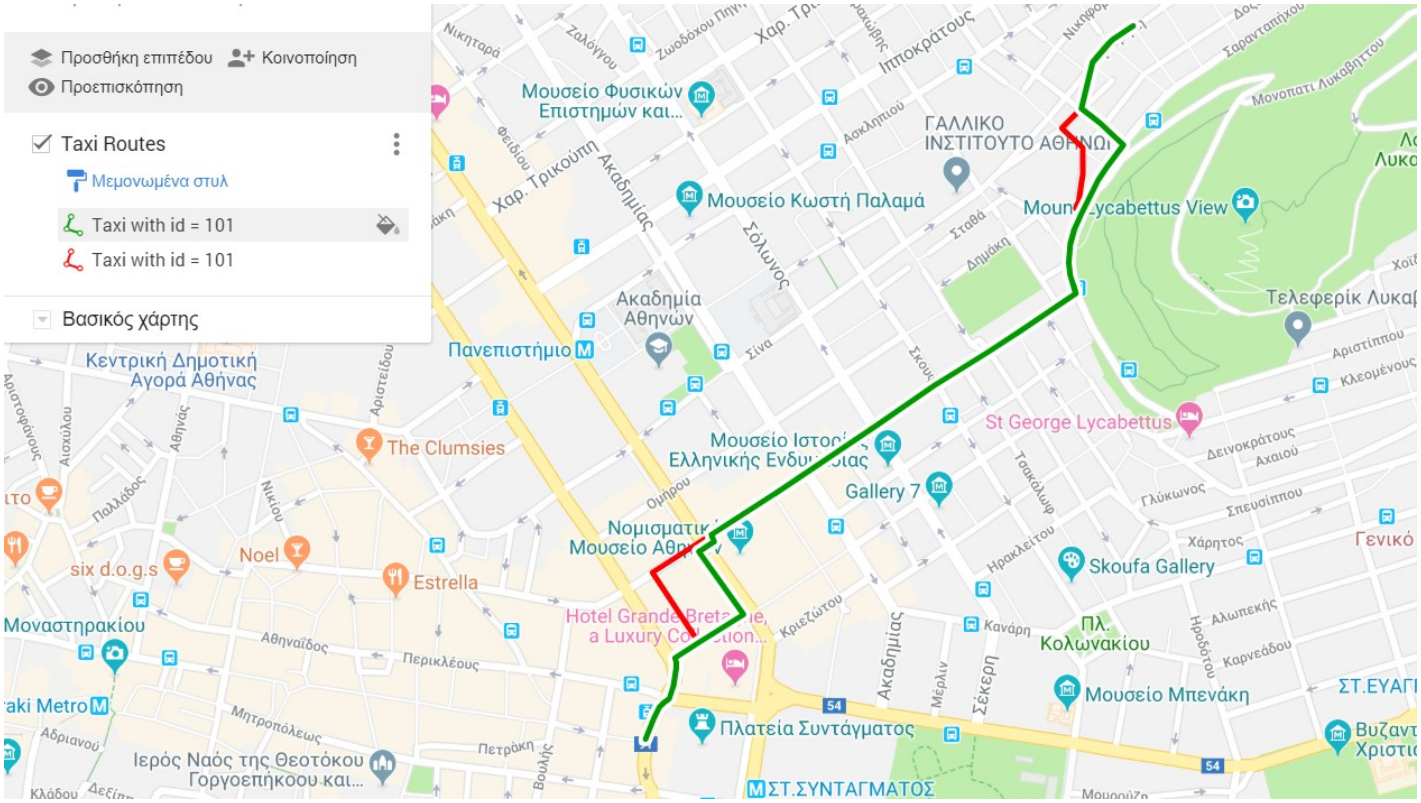
Σημειώνεται ότι για κάθε συνδυασμό τοποθεσίας ταξί και θέσης πελάτη ο αλγόριθμος βρίσκει την συντομότερη διαδρομή και τυχόν ισοδύναμες αυτής. Στην διάρκεια του παραπάνω υπολογισμού υπολογίζονται και όλες οι συντομότερες διαδρομές για τα υπόλοιπα ταξί αλλά δεν εγγράφονται σε κάποιο αρχείο καθώς δεν αποτελούν ζητούμενο του προβλήματος. Έτσι στην επεικόνιση του τελικού αποτελέσματος εμφανίζονται μόνο οι καλύτερες ισοδύναμες διαδρομές από το κοντινότερο ταξί. Το πρόγραμμα παράγει αρχεία .txt με τις συγκεκριμένες συντεταγμένες τις οποίες τοποθετούμε στο διδόμενο αρχείο .kml στην θέση ¡coordinates¿. Τέλος σημειώνεται ότ για να εξετάσουμε διαφορετικό συνδυασμό δεδεμένων εισόδου τροποποιούμε τις αντίστοιχες γραμμές ανάγνωσης αυτών.

Η τροποποίηση του A* φαίνεται και στο ζητούμενο παράδειγμα αλλά και σε αυτό που δημιουργήσαμε καθώς εμφανίζονται δύο διαφορετικές διαδρομές για κάθε βέλτιστη διαδρομή ταξί, οι οποίες έχουν ίδιο κόστος.

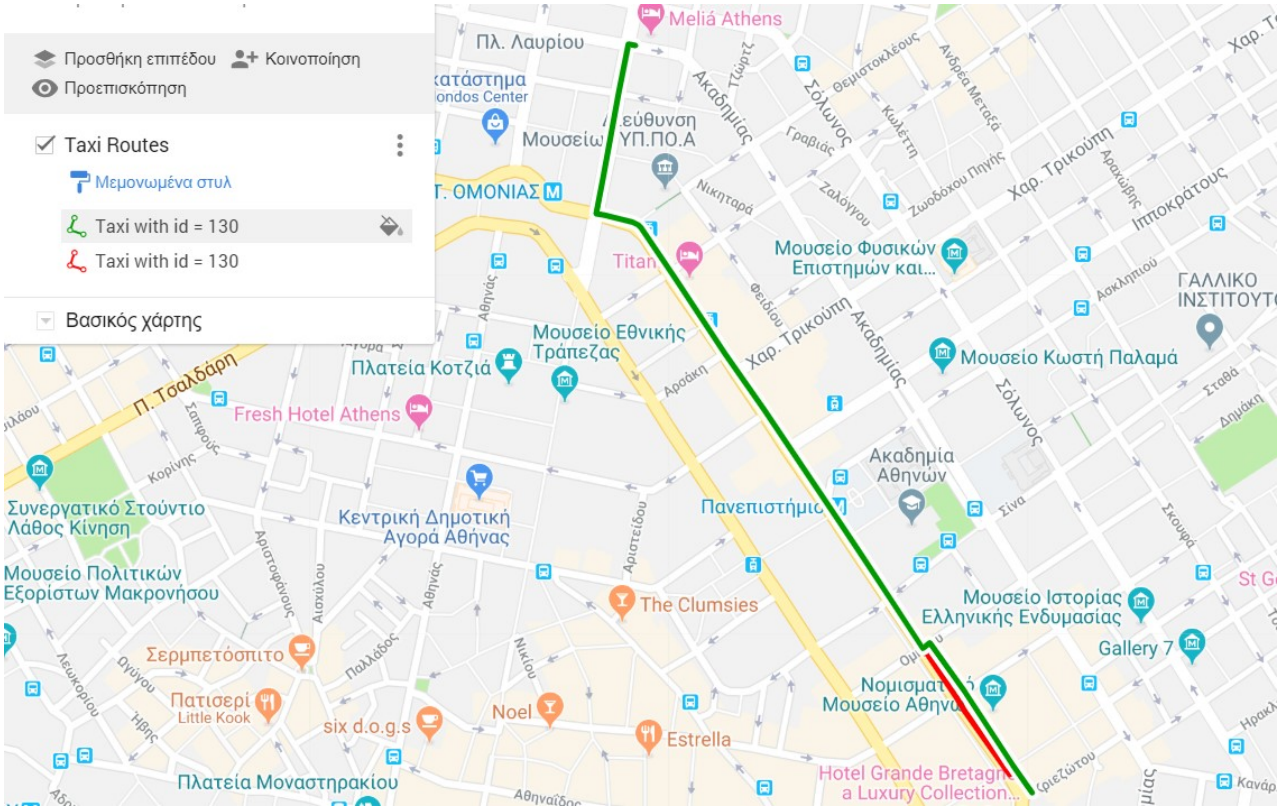
Για τον συνδυασμό των τοποθεσιών ταξί και πελάτη έχουμε τις παρακάτω δύο συντομότερες διαδρομές από το ταξι με id = 101 καθώς αυτό προέκυψε από τον αλγόριθμο ως η βέλτιστη λύση του προβλήματος. Με κόκκινο και πράσινο χρώμα εμφανίζονται αυτές και υποδηλώνουν τόσο τα κοινά όσο και τα διαφορετικά τμήματα τους.

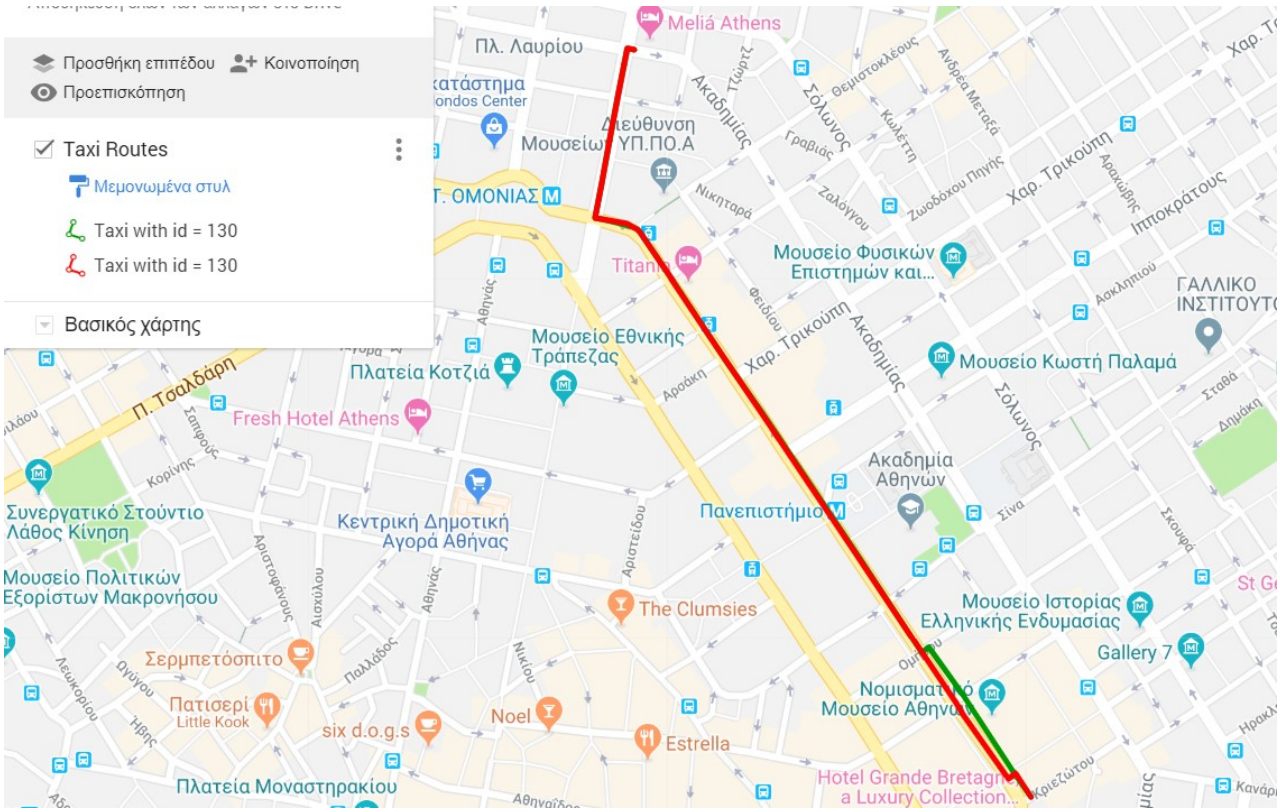


Παρατηρούμε όπως αναμέναμε από τον αλγόριθμο μας ότι δημιουργεί δύο δυνατές, ισοδύναμες από άποψη κόστους, διαδρομές που μπορεί να ακολουθήσει το πιο κοντινό ταξί. Με κόκκινο και πράσινο χρώμα αντίστοιχα φαίνονται οι ζητούμενες διαδρομές θεωρώντας πάντα δρόμους διπλής κατεύθυνσης.



Για το αρχείο που δημιουργήσαμε και είναι διαφορετικό από αυτό που δίδεται στην εκφώνηση σχετικά με την τοποθεσία πελάτη και ταξί θα έχουμε τα ακόλουθα αποτελέσματα για τις συντομότερες διαδρομές.





Όμοια όπως και στο παράδειγμα της εκφώνησης ο τροποποιημένος A* αλγόριθμος παράγει δύο ισοδύναμες διαφορετικές διαδρομές που σε κάποιο τμήμα τους επικαλύπτονται ενώ σε

κάποιο άλλο είναι διαφορετικές. Οι παραπάνω οπτικές απεικονίσεις το επιβεβαιώνουν αυτό και ειδικότερα στην πρώτη φαίνεται καλύτερα καθώς συμβαίνει να υπάρχουν δύο διαφορετικά σημεία στα οποία το ταξί μπορεί να πάρει άλλη διαδρομή.

Ο αλγόριθμος όπως έγινε φανερό μπορεί να προτείναι διαφορετικές διαδρομές ωστόσο στην πράξη που θα πρέπει να λάβουμε υπόψη την κατεύθυνση των οδών και την ύπαρξη ή μη φυσικών εμποδίων ή την ανυπαρξία οδικού δικτύου σε κάποια τμήματα ο συγκεκριμένος αλγόριθμος δεν θα μπορούσε να ανταποκριθεί. Θα έπρεπε να τροποποιηθεί ώστε να απαγορευτούν κάποιες συγκεκριμένες κατευθύνσεις και να δωθεί ελαστικότητα στο τι θεωρούμε ισοδύναμο σε μία διαδρομή. Το συγκεκριμένο πρόβλημα αποτελέι μια αφαιρετική προσέγγιση του πραγματικού καθώς δεν περιλαμβάνει τα παραπάνω αλλά ουτε και όλα τα δυνατά κριτήρια που θα πρέπει να λαμβάνει υπόψη του ένας αλγόριθμος εύρεσης συντομότερων διαδρομών. Οι ισοδύναμες διαδρομές με την αυστηρή έννοια του κόστους θα προταθούν. Η βλετισότητα εξαρτάται κυρίως από τον αλγόριθμο A* και τον χρόνο που θέλει αυτός να λειτουργήσει αλλά και κυρίως από τα ευριστικά κριτήρια και τον τρόπο μέτρησης των αποστάσεων που χρησιμοποιούμε. Στα παραπάνω γίνονται κάποιες παραδοχές που θεωρούνται ανεκτές σε αυτό το επίπεδο, ωστόσο για μεγαλύτερη ακρίβεια θα χρειαστούν τροποποιήσεις. Τέλος όπως κάθε ευριστικός αλγόριθμος έτσι και ο A* μπορεί να αποτύχει σε κάποιες περιπτώσεις είτε να δώσει χειρότερο αποτέλεσμα συγκριτικά με άλλους αλγορίθμους που εφαρμόζονται σε γραφήματα και πραγματοποιούν προεπεξεργασία των δεδομένων για να πετύχουν καλύτερη απόδοση.