# Τεχνητή Νοημοσύνη

## Θέμα 1: Προγραμματιστικό θέμα ανάπτυξης ευφυούς υπηρεσίας με χρήση αλγορίθμων αναζήτησης λύσης

**Χατζηχαραλάμπους Γεώργιος 03114709**
**Χρήστου Ανδρέας 03114705**

## Πρόβλημα

Χρησιμοποιούμε σαν είσοδο τα δεδομένα από τα αρχεία nodes.csv, client.csv, taxis.csv και κατασκευάζοντας ένα πρόγραμμα Java το οποίο θα υπολογίζει το καταλληλότερο ταξί το οποίο θα πρέπει να κατευθυνθεί προς τον πελάτη, επιλέγουμε αυτό που θα διανύσει τη μικρότερη διαδρομή για να φτάσει στον πελάτη. Για τον υπολογισμό της διαδρομής που θα πρέπει να ακολουθήσει το ταξί, θα χρησιμοποιήσουμε τον αλγόριθμο A* με ακτινωτή αναζήτηση, ο οποίος είναι μία παραλλαγή του κλασικού A*, ο οποίος όμως θέτει ένα μέγιστο όριο στο μέγεθος του μετώπου αναζήτησης που διατηρεί, παίρνοντας αυτό το όριο από τον χρήστη.

## Σχεδιασμός του συστήματος

Το πρόγραμμα μας εισάγει από το αρχείο client.csv τις συντεταγμένες του πελάτη και δημιουργεί ένα νέο αντικείμενο **Client** στο οποίο δεν καταχωρεί τις πραγματικές του συντεταγμένες, αλλά τις συντεταγμένες που βρίσκονται κοντινότερα σε αυτές, αφού οι πραγματικές του συντεταγμένες δεν αποτελούν συνήθως σημείο που βρίσκεται στο αρχείο nodes.csv. Στη συνέχεια διαβάζοντας το αρχείο taxis.csv διαβάζουμε τις συντεταγμένες όλων των διαθέσιμων ταξί και δημιουργούμε ένα αντικείμενο για το καθένα, καταχωρώντας τα σε ένα ArrayList, με τον ίδιο τρόπο που το κάνουμε και για το αντικείμενο Client όσον αφορά τις συντεταγμένες. Στη συνέχεια διαβάζουμε το αρχείο nodes.csv και καταχωρούμε όλα τα σημεία του χάρτη σε ένα ArrayList και ελέγχουμε ταυτόχρονα αν έχει ξαναεισαχθεί. Αν ναι, τότε στο αντικείμενο με τις ίδιες συντεταγμένες καταχωρούμε στο πεδίο OdoiL, το οποίο είναι μια λίστα με όλες τις οδούς στις οποίες ανήκει το συγκεκριμένο σημείο. Συνεπώς, επιτυγχάνουμε σύνδεση μεταξύ σημείων και οδών. Επίσης η κάθε οδός (**Odoi**) περιέχει όλα τα σημείου που ανήκουν σε αυτή.

Ο αλγόριθμος A* χρησιμοποιεί μια ευριστική συνάρτηση με υλοποίηση ευκλείδιας απόστασης από το τρέχον σημείο μέχρι το σημείο στόχος (πελάτης) και υπολογισμό του πραγματικού συνολικού κόστους απόστασης μέχρι το σημείο που έχουμε φτάσει. Αθροίζουμε την ευριστική αποστάση με την πραγματική απόσταση ώστε να βρούμε μια εκτιμήτρια απόστασης βάσει της οποίας θα δουλέψει ο αλγόριθμος A*. Υλοποιήσαμε ένα Priority Queue στο οποίο αποθηκεύουμε τους κόμβους, οι οποίοι είναι έτοιμοι προς διερεύνηση. Αυτό το Priority Queue είναι το μέτωπο αναζήτησης. Κάθε φορά βγαίνει από το Priority Queue ο κόμβος με τη μικρότερη εκτιμήτρια, ώστε να διερευνηθεί. Για κάθε κόμβο που βγαίνει από το Priority Queue βλέπουμε όλες στις οδούς στις οποίες ανήκει το συγκεκριμένο σημείο, και βάζουμε στο μέτωπο αναζήτησης όλα τα σημεία αυτών των οδών, υπολογίζοντας για το καθένα την εκτιμήτρια του. Σε κάθε σημείο που επισκεπτόμαστε-διερευνούμε χρησιμοποιούμε με μεταβλητή boolean την οποία χρησιμοποιούμε για να δηλώσουμε πως έχουμε ξαναεπισκεφτεί το συγκεκριμένο σημείο. Όταν βρεθεί σημείο το οποίο έχουμε ξαναεπιστεφτεί, ελέγχουμε αν την προηγούμενη φορά που το είχαμε επισκεφτεί, αυτό είχε γίνει με μικρότερη πραγματική απόσταση. Αν την είχαμε ξαναεπισκεφτεί με μεγαλύτερη πραγματική απόσταση, τότε ανανεώνουμε την πραγματική απόσταση του

σημείου με τη μικρότερη. Αυτό γίνεται ώστε να αποφεύγει ο αλγόριθμος να κάνει κύκλους γύρω από ένα σημείο.
Ο συνδιασμός του A* με ακτινωτή αναζήτηση γίνεται κόβοντας στο τέλος κάθε επανάληψης, το Priority Queue σε μέγεθος που δίνεται από τον χρήστη. Αυτό βοηθάει ώστε να μειώνεται αισθητά το μέγεθος του μετώπου αναζήτησης και να έχουμε πιο γρήγορο αλγόριθμο.
Ο αλγόριθμος τερματίζει όταν το σημείο που βγήκε από το Priority Queue, δηλαδή το σημείο που πρέπει να διερευνηθεί, είναι ο κόμβος στον οποίο βρίσκεται ο πελάτης. Στον κλασικό αλγόριθμο A* θα βρίσκαμε σίγουρα λύση για όλα τα διαθέσιμα ταξί, όμως ο συνδυασμός με ακτινωτή αναζήτηση, παρόλο που κάνει πιο γρήγορο το πρόγραμμα μας, κάποιες φορές δεν βρίσκει λύση για κάποια ταξί. Ο αλγόριθμος τρέχει για όλα τα διαθέσιμα ταξί, βρίσκοντας στο τέλος το καταλληλότερο που θα πρέπει να κινηθεί προς τον πελάτη.
Όταν βρεθεί λύση για το κάθε ταξί, τυπώνεται η διαδρομή που θα πρέπει να ακολουθήσει ώστε να φτάσει στον στόχο και βρίσκει ταυτόχρονα το καταλληλότερο ταξί αλλά και την πραγματική απόσταση που θα πρέπει να διανύσει.
Σημαντικό να αναφέρουμε πως χρησιμοποιούμε τις συντεταγμένες X,Y σαν κανονικά σημεία σε ένα επίπεδο, αφού παρατηρήσαμε πως πληρούνται τα κριτήρια ώστε να γίνει αυτό και να οδηγηθούμε επαγωγικά σε ορθή λύση. Βρίσκοντας λύση-διαδρομή για το κάθε ταξί, χρησιμοποιούμε τον τύπο Haversine ώστε να μετατρέψουμε την τελική διαδρομή σε πραγματικές αποστάσεις (km), ώστε να πάρουμε μια πιο εποπτική απεικόνιση της λύσης.

## Δομές Δεδομένων – Κλάσεις

Αρχικά έχουμε δημιουγήσει δύο κλάσεις **Simia** και **Odoi.**
Στη πρώτη αποθηκεύουμε όλα τα ξεχωριστά σημεία που διαβάζουμε από τη είσοδο. Στη δεύτερη αποθηκεύουμε όλες τις ξεχωριστές οδούς του χάρτη. Παράλληλα, συνδέουμε κάθε σημείο με όλες τις οδούς που ανήκει, με χρήση μιας λίστας από Odoi, αλλά και όλες τις οδούς,με χρήση μιας λίστας από Simia. Στις κλάσεις αυτές έχουμε διάφορες συναρτήσεις για αρχικοποιήσεις αλλά και ενημέρωση των στοιχείων της κλάσης.
Σημαντικό να αναφέρουμε ότι στη κλάση Simia, έχουμε δύο πεδία τύπου double για αποθήκευση των συντεταγμένων του σημείου X,Y, όπως και ένα πεδίο flag, τύπου boolean, όπου αρχικά είναι false και πρακτικά μας ενημερώνει αν έχουμε ήδη επισκεφθεί το συγκεκριμένο σημείο. Επίσης έχουμε ένα πεδίο Rdist (double) όπου είναι η πραγματική απόσταση που διανύσαμε από το συγκεκριμένο ταξί για να φτάσουμε στο συγκεκριμένο σημείο. Η συγκεκριμένη απόσταση αρχικοποιείται στο μηδέν. Συνάμα σημαντική η συνάρτηση euclid, όπου έχει σαν είσοδο ένα σημείο και επιστρέφει την ευκλείδια απόσταση του σημείου που στέλνουμε από αυτό που βρισκόμαστε τώρα.
Στη κλάση Odoi,απλά αποθηκεούμε το id της κάθε οδού μαζί με τη λίστα με τα συνδεδεμένα σημεία της οδού.
Επιπλέον, έχουμε δημιουργήσει μια κλάση **Taxi**, που χρησιμεύει στο να δημιουργούμε αντικείμενα ταξί και να αποθηκεύουμε όλα τα στοιχεία που χρειαζόμαστε. Περιεχεί δύο πεδία τύπου double για τις συντεταγμένες του ταξί όπως και ένα πεδίο τύπου int για το id του ταξί. Καθώς οι συντεταγμένες των ταξί μπορεί να μην αντιστοιχούν σε πραγματικό σημείο του χάρτη μας, βρίσκουμε στο κυρίως πρόγραμμα για κάθε ταξί το συντομότερο σημείο που βρίσκεται στον χάρτη από το σημείο που βρισκόμαστε και αντιστοιχίζουμε εκείνο το σημείο σαν τη πραγματική τοποθεσία του ταξί (με χρήση της συνάρτησης changeXY). Για αυτό το σκοπό ήταν χρήσιμο να συμπεριλάβουμε πάλι

μια συνάρτηση euclid,που βρίσκει την ευκλείδια απόσταση μεταξύ του ταξί και ενός σημείου που καλείται η συνάρτηση.
Επίσης, δημιουργήσαμε μια κλάση όπου αντιστοιχεί στο πελάτη (**Client**). Σε αυτήν έχουμε τις συντεταγμενες του πελάτη όπου πάλι μπορεί να μην αντιστοιχούν σε πραγματικές συντεταγμένες του χάρτη μας και χρειάζεται πάλι αντιστοίχηση με πραγματικό σημείο. Αυτό γίνεται πάλι από το κυρίως μας πρόγραμμα και με τη βοήθεια της συνάρτησης changeXY όπως και πριν.
Επιπλέον αναγκαία είναι η δημιουργία της κλάσης **Tnode**, όπου πρακτικά είναι ο κόμβος όπου θα εισάγουμε στη δομή δεδομένων μας (priority queue). Αυτή η κλάση όπως είναι λογικό πρέπει να έχει τη πραγματική απόσταση μέχρι τον κόμβο-σημείο που βρισκόμαστε (Rdist), την ευριστική απόσταση από το συγκεκριμένο σημείο μέχρι το πελάτη (Hdist), αλλά και το άθροισμα αυτών των δύο όπου λειτουργεί σαν εκτιμήτρια για τον αλγόριθμο μας (A*), καθώς χρησιμοποιεί το άθροισμα ευριστικής και πραγματικής ώστε να πάει σε επόμενο κόμβο. Επίσης, σ'αυτή την κλάση κρατάμε κα τις συντεταγμένες του πελάτη, έτσι ώστε να μπορούμε να ελέγχουμε αν έχουμε φτάσει στον τελικό προορισμό. Αυτό γίνεται με τη βοήθεια της συνάρτησης isFinal, όπου επιστέφει false/true αναλόγως. Εξίσου σημαντικού και το πεδίο path τύπου ArrayList<Simia> , όπου είναι μια λίστα από τα σημεία που περνούμε για να φτάσουμε στο σημείο που βρισκόμαστε τώρα. Όταν βρεθεί λύση για το κάθε ταξή αυτή η λίστα θα δηλώνει την διαδρομή που θα πρέπει να ακολουθήσει το κάθε ταξί.
Όσο αφορά τη δομή δεδομένων, έχουμε χρησιμοποιήσει την έτοιμη δομή ουράς προτεραιότητας της Java (Priority Queue) τροποποιώντας τον τρόπο με τον οποίο γίνεται η σύγκριση. Για τη υλοποίηση αυτής της σύγκρισης, έχουμε δημιουργήσει-τροποποιήσει της κλάση **Stepscomparator** (Comparator). Εκεί ουσιαστικά στέλνουμε δύο κόμβους Tnode και συγκρίνουμε τις SumDist κάθε Tnode.
Επίσης, δημιουργήσαμε την κλάση **Haversine**, η οποία χρησιμοποιείται ώστε να μετατρέπονται αποστάσεις μεταξύ συντεταγμένων σε πραγματικές αποστάσεις σε km. Υλοποιήσαμε τη συγκεγκριμένη συνάρτηση ώστε να βρίσκουμε την πραγματική απόσταση που θα διανύσει ένα ταξί μέχρι να φτάσει στον προορισμό του (πελάτης).
Τέλος, η κλάση του κυρίως προγράμματος **Main.** Εκεί πρακτικά γίνεται το διάβασμα της εισόδου και η δημιουργία αντίστοιχων αντικειμένων για τοποθέτηση των σημείων, των οδών , των ταξί και του πελάτη στα αντίστοιχα αντικείμενα κλάσεων, αλλά και οι κατάλληλες ενέργειες για σωστή σύνδεση τους και ενημέρωση των σωστών σημείων για ταξί και πελάτη. Έπειτα γίνεται η εκτέλεση του αλγόριμου και εκτύπωση των αποτελεσμάτων.

## Αποτελέσματα – Συμπεράσματα

Το πρόγραμμα μας τυπώνει τη διαδρομή που θα ακολουθήσει το κάθε ταξί ώστε να φτάσει στον προορισμό του (πελάτης), καθώς και την πραγματική απόσταση που θα διανύσει μέχρι να φτάσει εκεί. Στο τέλος τυπώνει επίσης και το καταλληλότερο ταξί, δηλαδή το ταξί που βρίσκεται πιο κοντά στον πελάτη. Επίσης μας τυπώνει πόσες φορές θα κάνει τον αλγόριθμο, δηλαδή πόσες φορές έβγαλε κάποιο σημείο από το μέτωπο αναζήτησης (Priority Queue), καθώς και το μέγιστο μέγεθος του Priority Queue πριν το κόψιμο του (ώστε ο αλγόριθμός μας να γίνει συνδιασμός με ακτινωτή αναζήτηση).

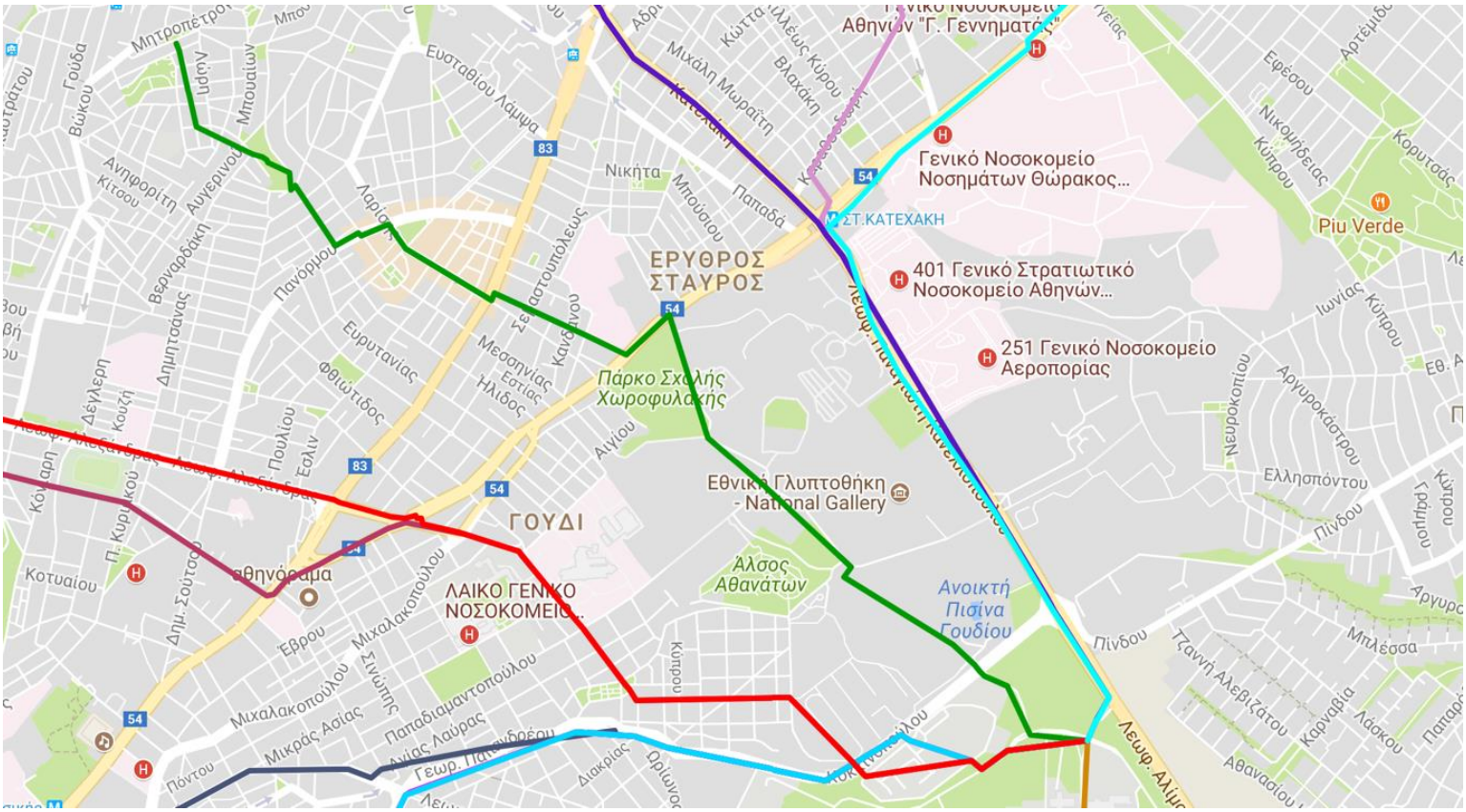
Παρατηρήσαμε πως όσο μικρότερο μέγεθος δίνεται από τον χρήστη, ως το μέγεθος του μετώπου αναζήτησης (ακτινωτή αναζήτηση), τόσο πιο γρήγορα εκτελείται το πρόγραμμα, αφού ο αλγόριθμος τρέχει λιγότερες φορές. Αυτό όμως δεν είναι κατ' ανάγκη καλό αφού παρατηρήσαμε πως για αρκετά μικρά w (είσοδος – μέγεθος μετώπου αναζήτησης) για κάποια ταξί δεν βρίσκει λύση, αφού εξαντλείται το Priority Queue. Επίσης για μικρά w παρατηρούμε πως η λύση δεν είναι βέλτιστη, αφού όπως φαίνεται και στα αποτελέσματα πιο κάτω, όταν έχουμε w=50 η απόσταση που θα πρέπει να διανύσει το καταλληλότερο ταξί θα είναι μεγαλύτερη από την απόσταση που θα διανύσει όταν έχουμε w=500, αφού οι διαδρομές είναι διαφορετικές. Έτσι η επιλογή του w πρέπει να γίνεται βάσει κάποιων κριτηρίων, ώστε να δημιουργούμε ένα γρήγορο αλγόριθμο, αλλά ταυτόχρονα να μην χάνουμε κάποιες λύσεις. Στο συγκεκριμένο πρόβλημα, αυτό ίσως να μην μας επηρεάζει αφού ψάχνουμε μόνο το καλύτερο ταξί, σε διαφορετικές περιπτώσεις όμως, αυτό πιθανόν να επιφέρει καταστροφικά αποτελέσματα. Επιπλέον, η επιλογή ενός πάρα πολύ μικρού w πιθανόν να αποκλείσει και τη βέλτιστη λύση, γεγονός που καθιστά την «καλή» επιλογή w αναγκαία.

## Οπτικοποίηση Αποτελεσμάτων

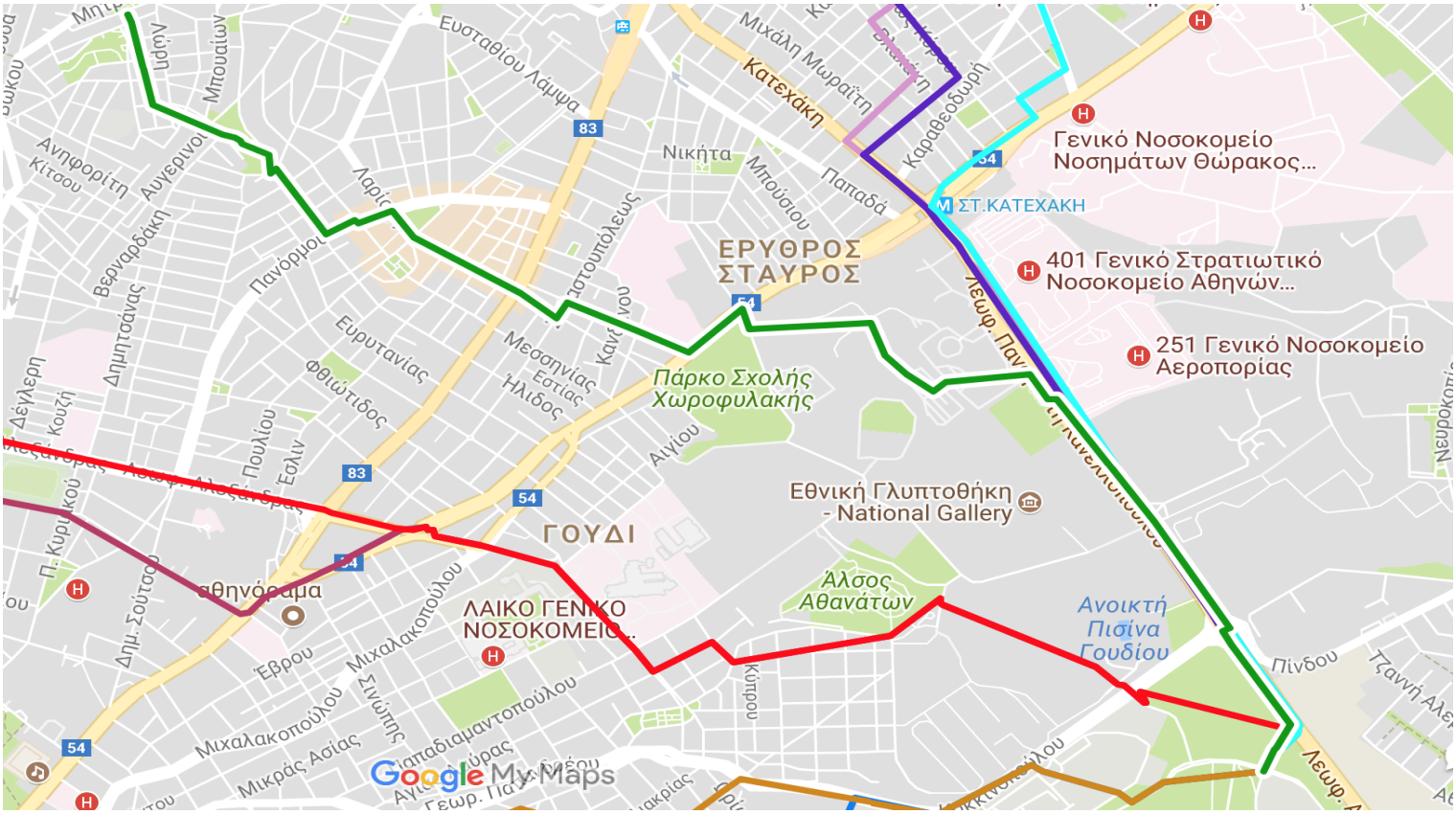
Για την οπτική παρουσίαση των αποτελεσμάτων, μεταφέρουμε τα αποτελέσματα του προγράμματος Java στο αρχείο KML που έχουμε δημιουργήσει, ώστε να παρουσιαστούν οπτικά χρησιμοποιώντας το Google Maps API. Στην οπτική παρουσίαση των αποτελεσμάτων χρησιμοποιήσαμε πράσινο χρώμα για να παρουσιάσουμε το καταλληλότερο ταξί και κάποια άλλα χρώματα για τα υπόλοιπα. Στις περιπτώσεις που ο αλγόριθμος μας δεν καταλήγει σε λύσεις για κάποια ταξί, αυτά τα ταξί δεν φαίνονται στον χάρτη, αφού ουσιαστικά είναι μη διαθέσιμα.

Παρακάτω παρουσιάζουμε οπτικά τα αποτελέσματα για τα αρχεία που δόθηκαν, καθώς και για αρχεία που φτιάξαμε εμείς, δίνοντας στο πρόγραμμα μας 2 διαφορετικές τιμές για το w (μέγιστο μέγεθος μετώπου αναζήτησης).

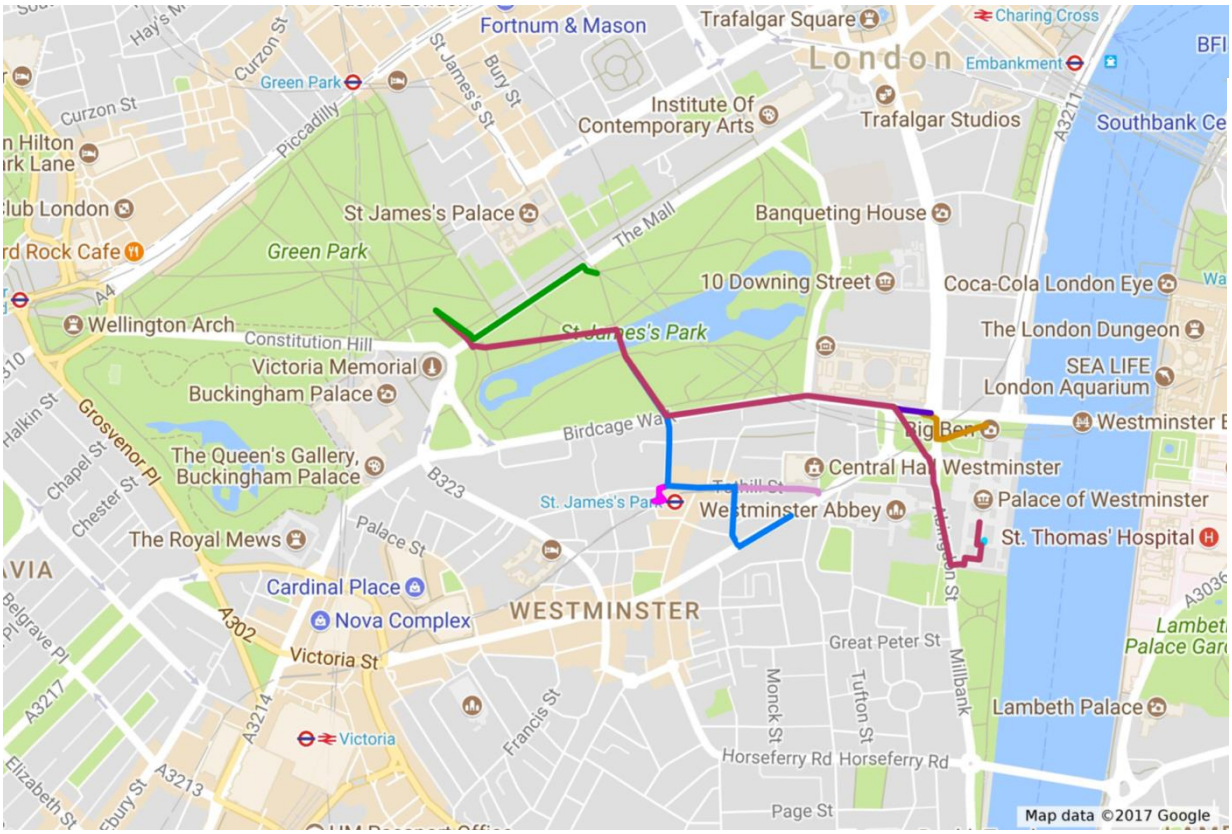
Για **w=500** το πρόγραμμα βρίσκει λύση για όλα τα διαθέσιμα ταξί και λαμβάνουμε τα παρακάτω αποτελέσματα:
Καταλληλότερο ταξί: #170
Απόσταση που θα διανύσει το καταλληλότερο ταξί: 3.8971003916 km
Μέγιστο μέγεθος μετώπου αναζήτησης: 698
Πόσες φορές έχει τρέξει ο αλγόριθμος: 57113

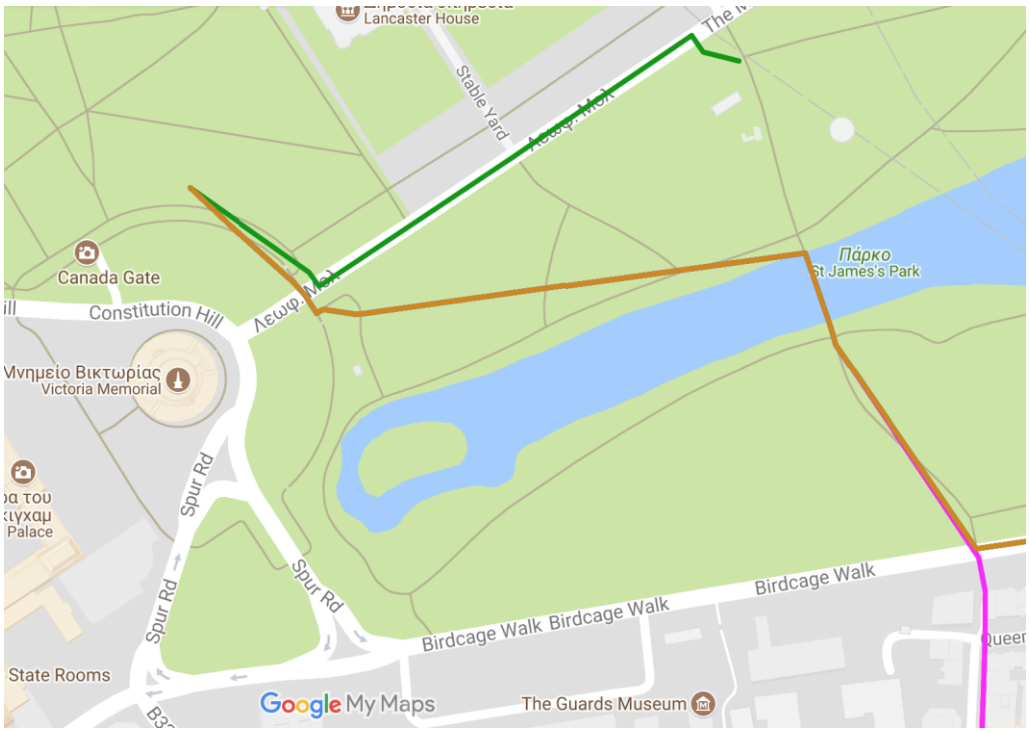



Για **w=50** το πρόγραμμα βρίσκει λύση για 8 από τα 11 διαθέσιμα ταξί και λαμβάνουμε τα παρακάτω αποτελέσματα:
Καταλληλότερο ταξί: #170
Απόσταση που θα διανύσει το καταλληλότερο ταξί: 4.1299983 km
Μέγιστο μέγεθος μετώπου αναζήτησης: 171
Πόσες φορές έχει τρέξει ο αλγόριθμος: 16429

Για τον χάρτη που φτιάξαμε με **w=500** το πρόγραμμα βρίσκει λύση για όλα τα διαθέσιμα ταξί και λαμβάνουμε τα παρακάτω αποτελέσματα:
Καταλληλότερο ταξί: #6
Απόσταση που θα διανύσει το καταλληλότερο ταξί: 0.559466600km
Μέγιστο μέγεθος μετώπου αναζήτησης: 657
Πόσες φορές έχει τρέξει ο αλγόριθμος: 7985



Για τον χάρτη που φτιάξαμε με **w=50** το πρόγραμμα βρίσκει λύση για 8 από τα 9 διαθέσιμα ταξί και λαμβάνουμε τα παρακάτω αποτελέσματα:
Καταλληλότερο ταξί: #6
Απόσταση που θα διανύσει το καταλληλότερο ταξί: 0.559466600 km
Μέγιστο μέγεθος μετώπου αναζήτησης: 387
Πόσες φορές έχει τρέξει ο αλγόριθμος: 4927