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

Τεχνητή Νοημοσύνη

Χειμερινό Εξάμηνο 2016-2017

Θέμα 1

Ημερομηνία: 5/12/2016
Καλογερόπουλος Δημήτριος (03111179)
Χαρδούβελης Παναγιώτης Ιάσων (03111082)

- # ΓΕΝΙΚΟΣ ΣΧΕΔΙΑΣΜΟΣ -ΤΡΟΠΟΣ ΜΟΝΤΕΛΟΠΟΙΗΣΗΣ ΔΕΔΟΜΕΝΩΝ

Επεξεργασία αρχείων

Τα αρχεία δίνονται σε csv(Comma Seperated Value) μορφή. Έτσι, διαβάζουμε κάθε γραμμή του αρχείου και αποθηκεύεται σε ένα string line. Έπειτα, το string αυτό το αποθηκέυουμε σε ένα πίνακα String[] lineArray, όπου ουσιαστικά “σπάμε” το line σε επιμέρους strings με διαχωριστικό το κόμμα(cvsSplitBy). Στη συνέχεια, χειριζόμαστε ξεχωριστά κάθε στοιχείο του πίνακα αναθέτοντάς το στο αντίστοιχο πεδίο με χρήση των setters των κλάσεων.

Κάθε γραμμή του αρχείου nodes.csv μοντελοποιήθηκε σαν ένα αντικείμενο street.

Η κλάση Street

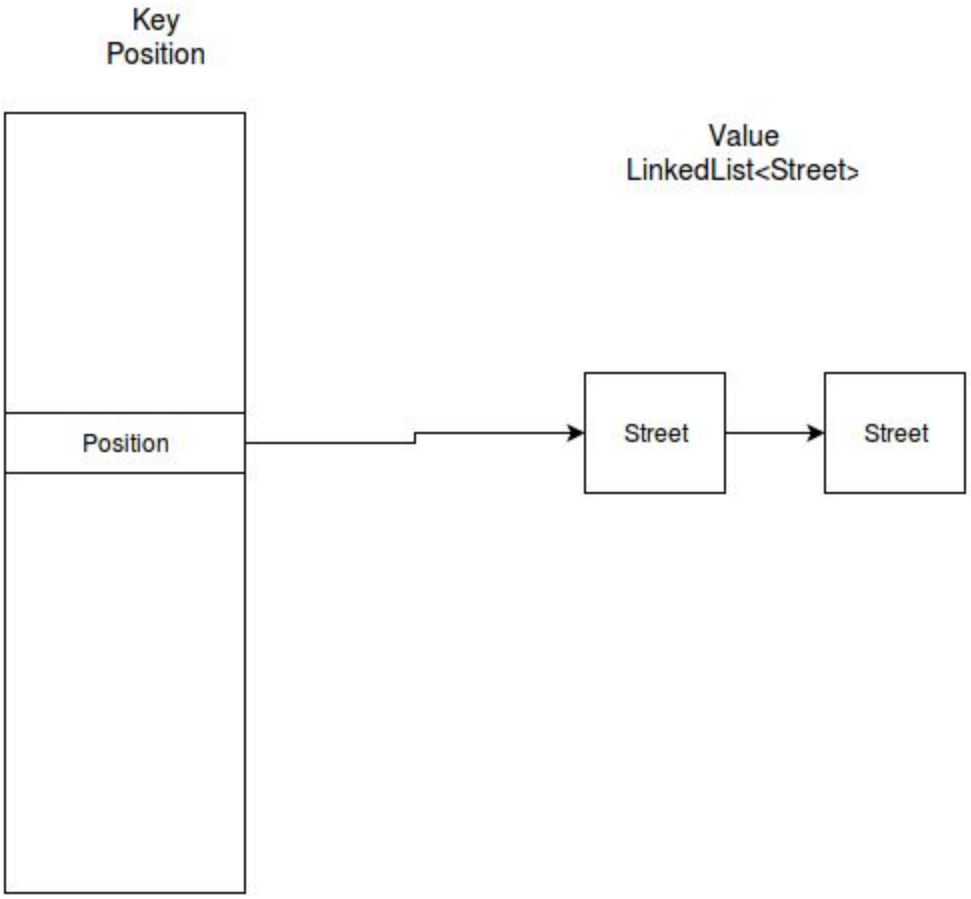
```
public class Street {
    private int id;
    private String name;
    private Position prev;
    private Position next;
}
```

Το αντικείμενο αυτό περιέχει το id και το όνομα του κόμβου καθώς και δυο references που αναφέρονται στις συντεταγμένες του προηγούμενου και επόμενου γείτονα του(τύπου Position-βλέπε παρακάτω). Κόμβοι που δεν είναι διασταύρωση έχουν απλα 2 (το πολύ) γείτονες. Αντίθετα κόμβοι-διασταυρώσεις έχουν γείτονες ανάλογους με το βαθμό της διασταύρωσης. Πιο συγκεκριμένα, ο κόμβος που βρίσκεται πάνω στη διασταύρωση δυο δρόμων έχει (το πολύ) 4 γείτονες.

Λέμε “το πολύ”, καθώς για παράδειγμα ο πρώτος κόμβος κάθε δρόμου (αν δεν είναι διασταύρωση) έχει μόνο εναν γείτονα, τον next που είναι ο επόμενος κόμβος στο αρχείο. Αντίστοιχα, ο τελευταίος κόμβος ενός δρόμου έχει μόνο prev γείτονα

Το αρχείο εισόδου nodes.csv είναι οργανωμένο σε blocks κόμβων με ίδιο id δρόμου. Έτσι υπάρχουν κόμβοι με τις ίδιες συντεταγμένες <X,Y> και διαφορετικό id, που σηματοδοτούν κόμβο που βρίσκεται πάνω σε μια διαστάυρωση, οι οποίοι έπρεπε να εντοπιστούν για τους λόγους που αναφέραμε παραπάνω.

Γι αυτό το λόγο, κατα τη διαδικασία του διαβάσματος του αρχείου nodes.csv, δημιουργούμε ένα καινούργιο αντικείμενο Street το οποίο το βάζουμε σε ένα αντικείμενο HashMap<Position, LinkedList<Street> > .



Έτσι με ένα απλό map.get(myPosition) παίρνουμε κατευθείαν (προσεγγιστικά σε O(1)) μια λίστα αντικειμένων Street, ενός (αν μιλάμε για έναν απλό κόμβο) ή περισσοτέρων αντικείμενων street αν στις συγκεκριμένες συντεταγμένες έχουμε διασταύρωση.
Να σημειωθεί οτι διαβάζουμε το αρχείο κατά ζεύγη γραμμών και χρησιμοποιούμε μια στοίβα μιας θέσης, ώστε να μπορούμε να καταχωρούμε τους next, prev γείτονες κατά την ώρα του διαβάσματος.

Η κλάση Position

```
public class Position {
    private double X;
    private double Y;
    private double gScore = Double.MAX_VALUE;
    private double fScore = Double.MAX_VALUE;
    private Position cameFrom;
.
.
.
}
```

Αντικείμενα αυτη της κλάσης μοντελοποιούν ουσιαστικά τους πραγματικούς κόμβους του χάρτη μας.
Περιέχει τις συντεταγμένες ενός κόμβου, τις τιμές gScore και fScore που χρησιμοποιούνται στον A*, καθώς και το reference σε αντικείμενα του ίδιου τύπου cameFrom, όπου κατα την εκτέλεση του A* κρατάμε τον αμέσως προηγούμενο κόμβο απο τον οποίο προσπελάστηκε αυτός που εξετάζουμε τώρα, ώστε να μπορούμε να ανακατασκευάσουμε τη τελική διαδρομή που δίνει ο αλγόριθμος.
Το πεδίο gScore περιέχει το κόστος της διαδρομής απο την αφετηρία μέχρι αυτόν τον κόμβο.
Το πεδίο fScore δηλώνει το εκτιμώμενο κόστος της διαδρομής απο την αφετηρία στο στόχο περνώντας απο αυτόν τον κόμβο. Η τιμή αυτή είναι εν μέρει γνωστή και εν μέρει ευριστική.

Διάβασμα του αρχείου client.csv

Πρόκειται για αρχείο μιας γραμμής που περιέχει τις συντεταγμένες του πελάτη.
Κατα το δίαβασμα του αρχείου δημιουργούμε και αρχικοποιούμε ένα καινούργιο αντικείμενο τύπου Client.

Η κλάση Client

```
public class Client {
    private Position myCoord;
    private Position near;
    private double curDistance = Double.MAX_VALUE;


    /*each time we process a new node line
     * we call this method to update the near field
    */
    public void calculateNear(Position other) {
       double dx = Math.abs(other.getX() - myCoord.getX());
       double dy = Math.abs(other.getY() - myCoord.getY());

       if ( (dx < 0.001) && (dy < 0.001) ) {
             if (Math.sqrt(dx*dx + dy*dy) < (this.curDistance)) {
                   this.curDistance = Math.sqrt(dx*dx + dy*dy);
                   this.setNear(other);
             }
       }

    }

}
```

Η κλάση client περιέχει τις συντεταγμένες του πελάτη και το reference σε αντικείμενο Position near, το οποίο δείχνει στον πλησιέστερο στον πελάτη κόμβο (εξήγηση παρακάτω).

Διάβασμα του αρχείου taxis.csv

Πρόκειται για το αρχείο που περιέχει τις συντεταγμένες των διαθέσιμων taxi.
Για το δίαβασμα του αρχείου καλούμε την readTaxisFile(...); , που επιστρέφει μια λίστα αντικειμένων Taxi. Κάθε κόμβος αυτης της λίστας περιέχει όλες τις απαραίτητες πληροφορίες για ενα taxi.

Η κλάση Taxi

```
/*
 * Taxi class corresponding to a taxi
 * contains its coordinates, its id,
 * the nearest node of the map closest to the taxi,
 * finalscore indicating the cost of the taxi-cient route
 * And a list of Coordinates object which models the path
 * the taxi followed to the client
*/
import java.util.ArrayList;

public class Taxi {
    private Position myCoord;
    private int id;
    private Position near;
    private double curDistance = Double.MAX_VALUE;
    private double finalScore = Double.MAX_VALUE;
    private ArrayList<Coordinates> totalPath;

    public void calculateNear(Position other) {
       double dx = Math.abs(other.getX() - myCoord.getX());
       double dy = Math.abs(other.getY() - myCoord.getY());

       if ( (dx < 0.001) && (dy < 0.001) ) {
             if (Math.sqrt(dx*dx + dy*dy) < (this.curDistance)) {
                   this.curDistance = Math.sqrt(dx*dx + dy*dy);
                   this.setNear(other);
             }
       }

    }

}
```

Η κλάση αυτή είναι πανομοιότυπη με την Client με εξαίρεση το πεδίο totalPath, που χρησιμοποιείται για την αποθήκευση του μονοπατιού που υπολόγισε ο A*, στην reconstructPath(..);

Εξήγηση της μεθόδου calculateNear(..) των κλάσεων Taxi και Client

Όπως συμβαίνει και στην πραγματικότητα, οι συντεταγμένες των ταξι αλλά και του πελάτη δεν είναι απαραίτητο να συμπίπτουν ακριβώς με έναν κόμβο-δρόμο του του χάρτη μας.Για παράδειγμα ο πελάτης μπορεί να μην βρίσκεται σε σημείο προσβάσιμο απο δρόμο.

Με τη μέθοδο calculateNear υπολογίζουμε τον κοντινότερο actual κόμβο στις δοθείσες συντεταγμένες του πελάτη και του ταξι αντίστοιχα.

Συγκεκριμένα όταν διαβάζουμε το αρχείο nodes.csv, για κάθε καινούργιο κόμβο που φτιάχνουμε, καλούμε την caclulateNear για το ταξι και τον πελάτη, η οποία ανανεώνει, αν χρειάζεται το πεδίο near της αντίστοιχης κλάσης. Έτσι με το πέρας του διαβάσματος του αρχείου nodes.csv το πεδίο near αναφέρεται στον κοντινότερο actual κόμβο.

Επεξήγηση της κλάσης Main

Η κλάση Main είναι η κύρια κλάση της υλοποίησής μας, η οποία αρχικά διαβάζει τα αρχεία client.csv , taxi.csv και nodes.csv , τα οποία αποθηκεύει στις δομές που περιγράψαμε. Έπειτα, για κάθε ταξί καλούμε τη μέθοδο Astar(), που παίρνει ως είσοδο το hashMap που περηγράψαμε παραπάνω κατα το διάβασμα του αρχείου nodes.csv, ένα αντικείμενο τύπου Taxi, από όπου παίρνουμε τον κόμβο αφετηρία και ένα αντικείμενο client που μας δίνει το κόμβο- στόχο. Η μέθοδος επιστρέφει ένα reference σε αντικέιμενο τύπου position, και συγκεκριμένα αυτό του fully evaluated goal κόμβου αν επιτύχει.

```
private static Position Astar(HashMap<Position, LinkedList<Street>>
map,Taxi taxi,Client client)
```

Η μέθοδος υπολογίζει το βέλτιστο μονοπάτι ξεκινώντας από το taxi.getNear() (προσεγγίζοντας τις πλησιέστερες διαθέσιμες συντεταγμένες από το nodes.csv) προς τον client.getNear() (προσεγγίζοντας τις πλησιέστερες διαθέσιμες συντεταγμένες από το nodes.csv).

Σχετικά με την υλοποίηση του A*, οι δομές που χρησιμοποιήσαμε είναι οι εξής:

- Priority Queue για το openSet, το οποίο περιέχει δεδομένα τύπου Position

Στη δομή αυτή αποθηκεύουμε κάθε φορά τους υπό εξέταση κόμβους, δηλαδή τους γείτονες του current κόμβου που βρισκόμαστε, και αυτό γίνεται με τη βοήθεια του Hashmap. Με ένα map.get(curr) παίρνουμε τη λίστα LinkedList<Street> που μας επιτρέπει γρήγορη εύρεση των γειτόνων του υπο εξέταση κόμβου

Έχουμε 4 περιπτώσεις καθώς συναντάμε έναν κόμβο:

1) Δεν έχουμε ξαναεπισκεφθεί τον κόμβο-γείτονα, οπότε αυτός μπαίνει στο openSet και καταχωρούμε τα πεδία gScore, fScore, cameFrom.
2) Ο κόμβος υπάρχει ήδη στο openSet και βρίσκουμε καλύτερο μονοπάτι προς τον κόμβο αυτό(δλδ μικρότερο tentativeScore από το gScore του γείτονα), τότε ανανεώνουμε τα πεδία gScore, fScore και cameFrom με τις καινούριες τιμές. Δηλαδή, βρίσκουμε καλύτερο μονοπάτι από ότι είχαμε πριν.
3) Ο κόμβος υπάρχει ήδη στο openSet και βρίσκουμε χειρότερο μονοπάτι προς τον κόμβο αυτό(δλδ μεγαλύτερο tentativeScore από το gScore του γείτονα), τότε συνεχίζουμε την εκτέλεση με τον επόμενο γείτονα, αφού το υπάρχων μονοπάτι είναι το βέλτιστο μέχρι στιγμής.
4) Στην περίπτωση που ο κόμβος βρίσκεται στο closedSet, είναι fully evaluated και δεν χρειάζεται να ασχοληθούμε μαζί του ξανά.

Αξίζει να σημειώσουμε πως για την υλοποίηση της priority queue χρειάζεται να ορίσουμε μια κλάση MyComparator, που κάνει implement to interface Comparator της JAVA. Συγκεκριμένα, κάνουμε override τη μέθοδο compare() ώστε η μετρική σύγκρισης να είναι το fScore κατά την εισαγωγή ενός στοιχείου στο priority queue. Στην κορυφή της queue βρίσκεται κάθε φορά ο κόμβος με το μικρότερο fScore και στο επόμενο βήμα είναι αυτός που θα επισκεφθούμε, δηλαδή θα μεταφερθεί στο closedSet.

- HashSet για το closedSet, το οποίο περιέχει τους κόμβους που αποτελούν κομμάτι της λύσης μας. Χρησιμοποιήθηκε το hashSet επειδή δεν υπάρχουν διπλότυποι κόμβοι(1 φορά ο καθένας) και έχουμε γρήγορα inserts. Όπως αναφέραμε παραπάνω, στο hashSet κάθε φορά βάζουμε το πρώτο στοιχείο του openSet, μέχρι ο current κόμβος να είναι ο κόμβος-στόχος.

Ανακατασκευή του μονοπατιού

Η μέθοδος Astar() επιστρέφει έναν κόμβο τύπου Position, ο οποίος είναι ο κόμβος στόχος. Σε περίπτωση αποτυχίας, επιστρέφει null. Χρειάζεται, επομένως, να κατασκευάσουμε το μονοπάτι-λύση από το ταξί στον πελάτη. Αυτό επιτυγχάνεται με τη μέθοδο reconstructPath(), η οποία δέχεται τον κόμβο που επέστρεψε η Astar() και σε μια ArrayList<Coordinates> αποθηκεύουμε τους διαδοχικούς κόμβους που θα επισκεφθεί το ταξί. Αυτό υλοποιείται διασχίζοντας διαδοχικά το cameFrom πεδίο κάθε κόμβου μέχρι να φτάσουμε στον αρχικό κόμβο.

Η διαδρομή αυτή αποθηκεύεται στο πεδίο totalPath για κάθε ταξί.

Η κλάση Coordinates είναι μια βοηθητική κλάση που έχει πεδία X,Y, δηλαδή τις συντεταγμένες κάθε κόμβου που χρειαζόμαστε για την τελική απεικόνιση της διαδρομής στο google maps.

Τέλος, όπως ζητείται, τυπώνουμε το id του ταξί που σύμφωνα με τον A* αλγόριθμο βρίσκεται πιο κοντά στον πελάτη. Παράλληλα, για λόγους διευκόλυσης της κατασκευής του kml αρχείου, φτιάχνουμε ένα φάκελο με το όνομα *results* στο workspace, ο οποίος περιέχει αρχεία .txt με τις διαδρομές για κάθε ταξί(συντεταγμένες που θα διεσχίσουμε).

Επεξήγηση της επιλογής heuristic function
Αξίζει να σχολιάσουμε την επιλογή της ευριστικής μας συνάρτησης, η οποία είναι η ευκλείδια απόσταση του τρέχοντα κόμβου από το κόμβο goal. Επιλέχθηκε η ευκλείδια απόσταση, επειδή καθώς πλησιάζουμε προς τον πελάτη, η ευριστική τιμή πάντα θα μειώνεται. Με αυτόν τον τρόπο διατηρείται η απαραίτητη μονοτονία και ικανοποιείται το κριτήριο αποδοχής(admissible). Αυτά μας οδηγούν στο συμπέρασμα ότι η διαδρομή που υπολογίστηκε είναι η βέλτιστη για κάθε ταξί.

Απεικόνιση αποτελεσμάτων στο χάρτη
Η απεικόνιση των αποτελεσμάτων τόσο για το αρχείο nodes.csv που μας δίνεται όσο για το αρχείο nodes-kipseli.csv που δημιουργήσαμε εμείς φαίνονται εδώ. Υπάρχουν 2 layers:

1) **Taxi Routes :** Οι διαδρομές των ταξί με βάση τα αρχεία csv που μας δίνονται και το αρχείο solution.kml που δημιουργήσαμε. Το αρχείο solutions.pdf περιέχει τα ίδια οπτικά αποτελέσματα με το παραπάνω link για το layer Taxi Routes μόνο.
2) **Taxi Routes-my_map :** Οι διαδρομές των ταξί με βάση τα αρχεία csv που δημιουργήσαμε εμείς και το αρχείο solution-my_map.kml .

Στα παραπάνω με πράσινο απεικονίζεται η βέλτιστη διαδρομή για όλα τα διαθέσιμα ταξί, με βάση το id που απεικονίζει το πρόγραμμά μας στην έξοδο.

Προσπαθήσαμε να εξάγουμε σε csv αρχείο μεγαλύτερο κομμάτι του χάρτη, αλλά δεν τα καταφέραμε αφού λαμβάναμε μήνυμα λάθους από το openstreetmap.

Σημείωση: Ο αλγόριθμος A* που χρησιμοποιήσαμε είναι βασισμένος στον αντίστοιχο αλγόριθμο της wikipedia.