ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ 2016 **ΘΕΜΑ 1**

ΠΟΛΙΤΗΣ ΗΛΙΑΣ 03111135
ΣΤΑΘΟΠΟΥΛΟΥ ΜΑΡΙΑ 03111904

ΠΕΡΙΓΡΑΦΉ

Στόχος της εργασίας είναι η υλοποίηση μιας ευφυούς υπηρεσίας εξυπηρέτησης πελατών ταξί. Πιο συγκεκριμένα, θεωρούμε ότι υπάρχει ένας πελάτης που βρίσκεται σε μια ορισμένη τοποθεσία και επιθυμεί να καλέσει ένα ταξί. Η υπηρεσία διαθέτει μια βάση δεδομένων με όλα τα διαθέσιμα ταξί και τη γεωγραφική θέση στην οποία βρίσκονται κάθε χρονική στιγμή, η οποία ανανεώνεται συνεχώς. Η υπηρεσία θα πρέπει να εντοπίζει και να ειδοποιεί το ταξί που μπορεί να μεταβεί πιο γρήγορα στη θέση του πελάτη.

ΔΕΔΟΜΈΝΑ

1. Γεωγραφική θέση Πελάτη (client.csv)
2. Γεωγραφικές συντεταγμένες και κωδικοί των Ταξί (taxis.csv)
3. Κατάλογος με γεωγραφικές συντεταγμένες διαφόρων σημείων των οδών (nodes.csv)
    - Κάθε οδός αναγνωρίζεται από έναν κωδικό, συνεπώς σημεία με τον ίδιο κωδικό ανήκουν στην ίδια οδό
    - Τα σημεία κάθε οδού δίνονται ακολουθιακά το ένα μετά το άλλο
    - Τα σημεία κάθε οδού ορίζουν ευθύγραμμα τμήματα που είτε προσεγγίζουν καμπύλες διαδρομές, είτε αντιστοιχούν σε σημεία τομής με άλλες οδούς (διασταύρωση)

ΖΗΤΟΎΜΕΝΑ

Κατασκευή ενός προγράμματος Java, που υπολογίζει το καταλληλότερο ταξί, δηλαδή το ταξί που πρέπει να διανύσει τη μικρότερη διαδρομή για να φτάσει στον πελάτη. Για τον υπολογισμό της διαδρομής να χρησιμοποιηθεί ο αλγόριθμος A*. Πιο συγκεκριμένα, το πρόγραμμα να υπολογίζει με τον A* την συντομότερη διαδρομή για κάθε ταξί προς τον πελάτη και να επιλέγει το ταξί που βρίσκεται πιο κοντά.

ΥΛΟΠΟΊΗΣΗ ΣΕ JAVA

ΕΠΕΞΕΡΓΑΣΊΑ ΔΕΔΟΜΈΝΩΝ ΕΙΣΌΔΟΥ

Στην κλάση Preprocess υλοποιείται η προεπεξεργασία των δεδομένων εισόδου, ώστε να είναι εύκολα διαχειρίσιμες οι πληροφορίες που μας δίνονται

1. nodes.csv

   Από το αρχείο nodes.scv μπορούμε να συλλέξουμε δυο πληροφορίες. Πρώτον, τις διαφορετικές οδούς και τα σημεία που ανήκουν σε αυτές και δεύτερον τις διασταυρώσεις.

   ➢ Μια οδός χαρακτηρίζεται από το μοναδικό id της (ενδεχομένως και από το όνομά της) και το σύνολο των σημείων που βρίσκονται σε αυτή. Οπότε για την περιγραφή της χρησιμοποιήσαμε τη δομή :

   ```
   public class Road {
      public int id;
      public String name;
      public ArrayList<double[]> coordinates = new ArrayList<>();
   }
   ```

   ➢ Οι διαφορετικές οδοί αποθηκεύονται σε δομή HashMap<Integer, Road>, όπου το key (Integer) είναι η id της οδού.

   ➢ Μια διασταύρωση χαρακτηρίζεται από το σημείο στο χάρτη (x,y) και το σύνολο των οδών στις οποίες ανήκει αυτό το σημείο. Οπότε για την περιγραφή τους χρησιμοποιούμε τη δομή:

   HashMap< Pair <Double,Double>, ArrayList<Integer>> Junctions

2. client.csv

   Από το αρχείο client.csv παίρνουμε πληροφορία για τον σημείο - στόχο που επιθυμούμε να φτάσουμε. Επειδή ο πελάτης μπορεί να μην βρίσκεται ακριβώς πάνω σε κάποιο σημείο από τον κατάλογο με τις γεωγραφικές συντεταγμένες των διαφόρων σημείων των οδών που παρέχεται από το αρχείο nodes.csv, βρίσκουμε το κοντινότερο στον πελάτη σημείο που ανήκει στον κατάλογο και αναθέτουμε αυτό ως κόμβο - στόχο. Το διάβασμα αυτού του αρχείου επιστρέφει μια δομή AStarNode που περιέχει τις συντεταγμένες του σημείου, την οδό που ανήκει και άλλες πληροφορίες που θα αναλύσουμε στη συνέχεια.

3. taxis.csv

   Από το αρχείο taxis.csv λαμβάνουμε τις γεωγραφικές συντεταγμένες (x,y) και τους κωδικούς των ταξί, δηλαδή τα σημεία έναρξης του A* αλγόριθμου. Για κάθε ταξί ακολουθούμε την ίδια διαδικασία που υλοποιήσαμε για τον πελάτη και το αντιστοιχούμε στο κοντινότερο σημείο που ανήκει στον κατάλογο των nodes.csv. Παρομοίως, κάθε ταξί είναι μια δομή AStarNode, που περιέχει (εκτός των άλλων που θα περιγράψουμε στη συνέχεια) τις συντεταγμένες του σημείου απ' όπου θα ξεκινήσει ο αλγόριθμος A* και την οδό που ανήκει. Εν τέλει το διάβασμα αυτού του αρχείου θα επιστρέψει μια δομή

   HashMap< Integer, AStarNode > Taxis

   όπου το key (Integer) είναι το id του ταξί.

ΑΛΓΌΡΙΘΜΟΣ Α*

(Εφαρμόζουμε τον Α αλγόριθμο για κάθε ταξί ξεχωριστά).*

Χρησιμοποιούμε 2 λίστες:

- ➔ Την **openList** της οποίας τα στοιχεία είναι ταξινομημένα σύμφωνα με τη συνάρτηση F(S) = g(S) + h(S), όπου η g(S) δίνει την πραγματική απόσταση του κόμβου S από τον αρχικό κόμβο και η h(S) δίνει την εκτίμηση της απόστασης του κόμβου S από τον κόμβο - στόχο, μέσω μιας ευριστικής συνάρτησης.
  Υλοποιήσαμε την openList με τη δομή *PriorityQueue<AStarNode>* και για ευριστική συνάρτηση χρησιμοποιήσαμε την ευκλείδεια απόσταση.
- ➔ Και την **closedList** που περιέχει “εξερευνημένους” κόμβους, οι οποίοι όμως μπορούν να τοποθετηθούν πίσω στην openList αν βρεθεί συντομότερο μονοπάτι προς αυτούς
  Υλοποιήσαμε την closedList με τη δομή *LinkedList<AStarNode>*

Καθώς οι κόμβοι στην openList είναι ταξινομημένοι σε αύξουσα σειρά (ως προς την f) ο αλγόριθμος επεκτείνει τους κόμβους που έχουν το μικρότερο εκτιμώμενο κόστος, δηλαδή αυτούς που είναι πιο πιθανό να είναι προς την κατεύθυνση του κόμβου - στόχου.

ΔΟΜΈΣ ΓΙΑ ΤΟΝ Α* ΑΛΓΌΡΙΘΜΟ

- ➢ AStarNode
  Η δομή AStarNode περιγράφει έναν κόμβο στον γράφο αναζητήσεων του Α* αλγορίθμου.

```
public class AStarNode implements Comparable<AStarNode> {
    public AStarNode pathParent;
    public double g;
    public double h;
    public double x;
    public double y;
    public int id;

 public double getf(){
     return this.g+this.h;
}

@Override
public int compareTo(AStarNode object) {
    return (object.getf() < this.getf())?-1:1;
}
 //euclidean
public double getEstimatedCost(AStarNode goalNode){
    return sqrt(pow((this.x - goalNode.x),2) + pow(this.y - goalNode.y,2));
}

 public ArrayList<AStarNode> getNeighbors(HashMap<Pair<Double, Double>,
ArrayList<Integer>> Junctions, HashMap<Integer, Road> Roads);
```

Η κλάση αυτή περιέχει:

- τις γεωγραφικές συντεταγμένες του σημείου - κόμβου (x,y)
- την οδό στην οποία ανήκει (id),
- την πραγματική απόσταση από τον αρχικό κόμβο (g)
- την εκτιμώμενη απόσταση από τον κόμβο στόχο (h)

και τις συναρτήσεις:

- getf, που επιστρέφει την τιμή της F που περιγράψαμε παραπάνω
- getEstimatedCost, που επιστρέφει την εκτιμώμενη απόσταση από έναν άλλο κόμβο
- getNeighbors, που επιστρέφει μια λίστα από AStarNode που είναι γειτονικά του˙ αν δεν είναι διασταύρωση τότε γείτονες είναι μόνο το προηγούμενο και το επόμενο σημείο στην οδό που ανήκει (αν υπάρχουν) , αλλιώς αν είναι διασταύρωση, γείτονες αποτελούν το προηγούμενο και το επόμενο σημείο όλων των οδών στις οποίες ανήκει

➢ AStar

Η κλάση AStar υπολογίζει το καλύτερο μονοπάτι προς τον κόμβο - στόχο με τη συνάρτηση:

public static LinkedList<AStarNode> findPath(AStarNode startNode, AStarNode goalNode, HashMap<Pair<Double, Double>, ArrayList<Integer>> Junctions, HashMap<Integer, Road> Roads)

που δέχεται σαν ορίσματα ένα ταξί, τον πελάτη, τις οδούς και τις διασταυρώσεις. Αν βρει μονοπάτι τότε το δημιουργεί - επιστρέφει με τη συνάρτηση

public static LinkedList constructPath(AStarNode node)

διαφορετικά επιστρέφει null.

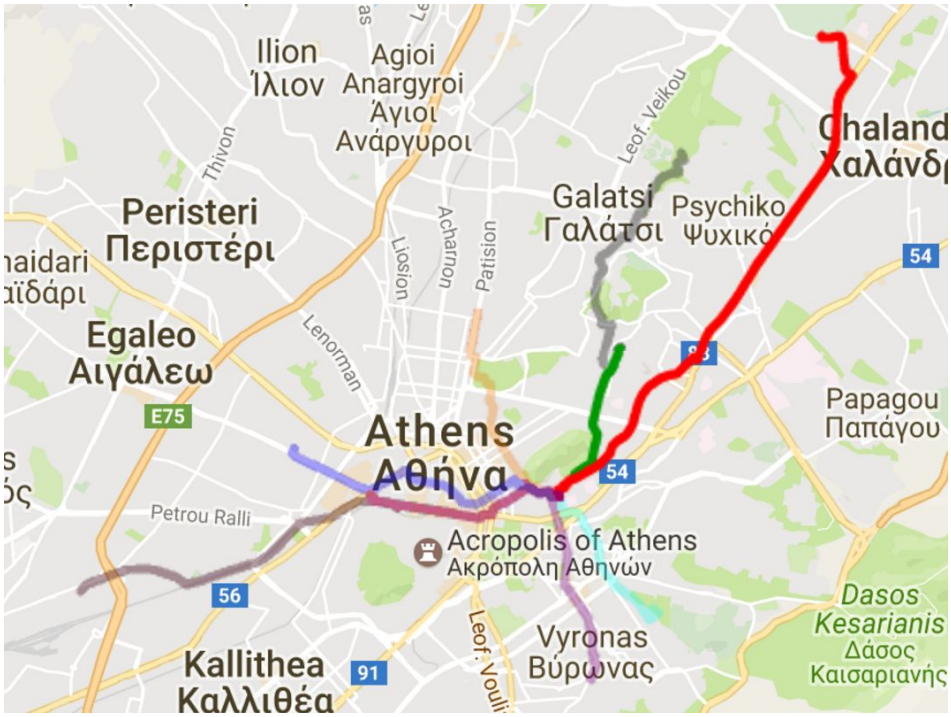
Το πρόγραμμά μας στο τέλος έχει συγκεντρώσει στη δομή:

HashMap<Integer, Pair<Double, LinkedList<AStarNode>>> Costs

για κάθε ταξί (Integer key) ένα ζεύγος που περιέχει το κόστος από το ταξί στον πελάτη και το μονοπάτι που ακολουθήθηκε, επομένως το ταξί με το μικρότερο κόστος είναι η καλύτερη επιλογή.

ΟΠΤΙΚΈΣ ΑΠΕΙΚΟΝΊΣΕΙΣ

ΔΟΘΈΝ ΑΡΧΕΊΟ:



ΔΙΚΟ ΜΑΣ ΑΡΧΕΙΟ: