



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

Τεχνητή Νοημοσύνη
Χειμερινό Εξάμηνο 2016-2017

Θέμα 2

Ημερομηνία: 9/2/2017

Καλογερόπουλος Δημήτριος (03111179)

Χαρδούβελης Παναγιώτης Ιάσων (03111082)

ΣΗΜΕΙΩΣΗ :

Αλλαγές στα δεδομένα εισόδου: Στα αρχεία εισόδου που μας δόθηκαν με entry 7ο δημοτικό,8ο δημοτικό.....,16ο δημοτικό αλλάξαμε το “,” σε “/” καθώς το “,” χρησιμοποιείται ως separator για τα διαφορετικά attributes. Στο αρχείο που παραδώσαμε έχουμε συμπεριλάβει τα νέα αρχεία για τα δεδομένα εισόδου.(φάκελος *input-new*)

Για να **τρέξει το εκτελέσιμο** Main δίνουμε τα ορίσματα με την εξής σειρά: {clients.csv, taxis.csv, lines.csv, nodes.csv, traffic.csv}

Επίσης, σημαντικό είναι το αρχείο rules.pl με τους κανόνες της Prolog να βρίσκεται στο φάκελο ../src , όπου ο φάκελος source περιέχει όλα τα αρχεία .java με τις κλάσεις μας.

Τα αρχεία της Prolog με τα facts κατασκευάζονται δυναμικά, καθώς τρέχει το πρόγραμμα και βρίσκονται και αυτά στο directory ../src . Παρόλ’ αυτά έχουμε συμπεριλάβει τα αρχεία αυτά στο επισυναπτόμενο αρχείο(φάκελος facts).

Επίσης, στην αρχή ο χρήστης πρέπει να δώσει μέσω του stdin αν η διαδρομή που θα ακολουθήσει το ταξί θα αποφύγει διόδια ή όχι (yes/no).

Διάβασμα αρχείων:

Όλη η “γνώση” του κόσμου κρατιέται στο knowledge base της Prolog. Συγκεκριμένα προσπελαίνουμε τα αρχεία εισόδου και δημιουργούμε καινούργια αρχεία prolog όπου κωδικοποιούμε τη γνώση σε facts (συγκεκριμένα τα αρχεία clientFacts.pl, taxiFacts.pl, nodeFacts.pl,nodeLinksFacts.pl, lineFacts.pl, trafficFacts.pl).

Αναλυτικότερα:

- **ClientFacts.pl**

Ένα fact της μορφής client(Time,Persons,Language,Luggage).

- **taxiFacts.pl**

Κάθε fact αντιπροσωπεύει ένα ταξί. Είναι της μορφής

taxi(Id,Available,Capacity,Languages_spoken,Rating,Long_distance,Type,Description).

όπου Capacity λίστα της μορφής [min,max] το min k max των ατόμων που μπορεί να μεταφέρει, και Languages_spoken μια λίστα με τις γλώσσες που μιλάει ο οδηγός.

- **lineFacts.pl**

Περιέχει την κωδικοποίηση του αρχείου lines.csv με facts της μορφής

line(Id,Oneway,Highway,[PrioList],[PermissionList]).

Η prioList περιέχει τα χαρακτηριστικά των lines που καθορίζουν την προτεραιότητα-βαρύτητά του ενώ η PermissionList τα χαρακτηριστικά που ορίσαμε ως ακατάλληλα για να περάσει το taxi απο εκεί.

PrioList = [lit,lanes,maxspeed,tunnel,bridge,incline,busway,toll]
PermissionList=[railway,boundary,access,natural,barrier,waterway]
Περαιτέρω εξήγηση παρακάτω στις σχεδιαστικές επιλογές.

- **nodeFacts.pl**

Facts της μορφής belongsTo(Nodeid,Lineid). Κατεσκευάστηκε όταν προσπελαύναμε το αρχείο nodes.csv

- **trafficFacts.pl**

Περιέχει facts της μορφής traffic(Lineid,[List of intervals]).
όπου ListofIntervals μια λίστα απο λίστες της μορφής
[[9.0,11.0,high],[13.0,15.0,medium],[17.0,19.0,high]] που κωδικοποιούν την κίνηση.

- **nodeLinksFacts.pl**

Περιέχει facts της μορφής next(Nodeid1,Nodeid2,X2,Y2)., που μεταφράζεται ότι απο τον κόμβο με id1 μπορώ να πάω στον nid2 και οι συντεταγμένες του 2 είναι οι X2,Y2. Αυτά τα facts τα δημιουργήσαμε κατευθείαν καθώς διαβάζαμε το αρχείο nodes (οπου διαβάζαμε ανα δυο nodes). Κοιτάζαμε αν δυο κόμβοι ανοίκουν στο ίδιο line και μετα με ένα query στο συστημα prolog (έχοντας ολοκληρώσει το parsing του αρχείου lines και τη δημιουργία των facts) μαθαίναμε τη τιμή του πεδίου oneway = {yes,no,-1,nh}, κατασκευάζαμε το αντίστοιχο fact next(). Δηλαδή για 2 διαδοχικούς κόμβους σε line διπλής κατεύθυνσης έχουμε δυο next facts. Στην περίπτωση το πεδίο oneway δεν είχε πληροφορία (nh) θεωρήσαμε δρόμο διπλής κατεύθυνσης.

Το αρχείο rules.pl περιέχει όλους τους κανόνες prolog που χρησιμοποιήσαμε.
Παρουσιάζονται οι κανόνες που κάνουμε query απο τη java και η χρησιμότητά τους.

Κανόνες σχετικοι με την καταλληλότητα των ταξι και τις ζητούμενες κατατάξεις τους,

- isSuitable(Taxid,Distance).

Έχοντας υπολογίσει την απόσταση απο το σημείο παραλαβής μέχρι τον προορισμό του πελάτη κάνουμε το query isSuitable(Tid,<distance>) και το σύστημα prolog μας επιστρέφει τα ids των ταξι που είναι κατάλληλα γι αυτή τη διαδρομή. Για να είναι ένα ταξι κατάλληλο θα πρέπει να είναι available, να είναι διατεθειται για μεγάλες διαδρομές στη περίπτωση που η υπολογισμένη απόσταση είναι άνω των 30km, ο

οδηγός να μιλάει τη γλώσσα του πελάτη, να μπορεί να μεταφέρει όλους τους επιβάτες (κανόνες iffits) καθώς και τον αριθμό των αποσκευών τους (canscurry). Για τις αποσκευές υποθέσαμε ότι taxi τύπου minivan δεν έχουν πρόβλημα χώρου, τύπου subcompact μπορούν να μεταφέρουν μέχρι 2 αποσκευές, τύπου compact μέχρι 3 και large μέχρι 4.

- Για τα taxis που πληρούν τις παραπάνω προϋποθέσεις εκτελούμε τον A* με προορισμό τον πελάτη και παίρνουμε τις διαδρομές καθώς και την απόσταση taxi - πελάτη. Έπειτα εκτελούμε το query: `ranking1([[tid1,dist1],[tid2,dist2],[tid3,dist3]....],R)` και στο R επιστρέφεται μια λίστα με τα ids των taxi ταξινομημένα σε αύξουσα σειρά από την απόσταση τους από τον πελάτη (κατάταξη νο1). Στον πελάτη παρουσιάζονται τα 5 πρώτα taxi αυτής της κατάταξης
- Στη συνέχεια ταξινομούμε τα παραπάνω 5 taxi βάσει μιας δικίας μας αξιολόγησης. Για το σύστημα αξιολόγησης αυτό συνυπολογίσαμε το rating του κάθε taxi και τον τύπο του οχήματος, σκεπτόμενοι ότι όλοι προτιμούμε ένα πολυτελές αμάξι για τις μετακινήσεις (large, compact) μας σε σχέση με ένα μικρό στενόχωρο (subcompact).

Γι αυτό το σκοπό εκτελούμε το query `ranking2([taxi_ids], R)`. το οποίο επιστρέφει λίστα με τα ids που του δόθηκαν ταξινομημένα κατά φθίνουσα σειρά ως προς το σύστημα αξιολόγησης που χρησιμοποιήσαμε. Ο κανόνας `final2()` αναλαμβάνει να υπολογίσει τη βαθμολογία αυτή και να κατασκευάσει μια λίστα ζευγαριών `[[tid, rating],....]` για την ταξινόμηση από τον κανόνα `sortids2`.

Κανόνες σχετικοί με την καταλληλότητα και προτεραιότητα δρόμων.

Σχετικά με τις σχεδιαστικές μας επιλογές σε ότι αφορά την καταλληλότητα και την προτεραιότητα δρόμων θεωρήσαμε τα εξής.

- **Καταλληλότητα δρόμου για μετακίνηση**

Παραπάνω μιλήσαμε για την

`PermissionList=[railway,boundary,access,natural,barrier,waterway]` που περιέχει τα χαρακτηριστικά ενός line που εμείς αποφασίσαμε, αφού συμβουλευτήκαμε το site `openstreetmap.org` ότι αν έχουν τιμή καθιστούν το line ακατάλληλο για μετακίνηση. Αν δεν υπάρχει τιμή για κάποιο στοιχείο της λίστας δώσαμε προκαθορισμένη τιμή `nn`.

Ένας δεύτερος παράγοντας είναι τιμή του πεδίου `highway` του κάθε line.

Συγκεκριμένα με το όρισμα `permit_list` ορίζουμε μια λίστα με τις αποδεκτές τιμές για μετακίνηση.

Λαμβάνοντας υπ' όψη τα παραπάνω ο κανόνας που χρησιμοποιούμε για να μάθουμε τις δυνατές μεταβάσεις μας απο τον current κόμβο είναι ο

canMoveFromTo(X,Y,X1,Y1) που καλείται απο την java ως
canMoveFromTo(currentNodeId,_,X1,Y1) και μας επιστρέφει τις συντεταγμένες των γειτόνων που επιτρέπεται η μετακίνηση.

- Προτεραιότητα δρόμων

Για τον ορισμό της προτεραιότητας θεωρήσαμε ότι κάθε ένα απο τα παρακάτω χαρακτηριστικά αποδίδει διαφορετική βαρύτητα σε ένα δρόμο.
Τα κριτήρια που συνυπολογίσαμε για να δώσουμε στον αλγόριθμο προτίμηση σε συγκεκριμένους δρόμους είναι τα εξής:

lit: αν η μετακίνηση γίνεται μετα τις 19.00 τότε Prio +=0.0002

lanes: Lines με 1 η 2 lanes τότε Prio +=0.0001

Lines με 3 η 4 lanes τότε Prio +=0.0002

Lines με πάνω απο 5 lanes τότε Prio +=0.0003

maxspeed: (30,60] → Prio +=0.0002

>=60 → Prio +=0.0003

traffic: low→ Prio +=0.0006 , medium → Prio +=0.0003 , High → Prio -=0.0006

είδος δρόμου (τιμή πεδίου highway): Αν το line έχει τιμή μια εκ των ορισμένων στην prio_list([motorway,trunk,primary,secondary,tertiary,motorway_link,trunk_link,primary_link,secondary_link,tertiary_link]). που είναι υποσύνολο της permitlist τότε Prio+=0.0002

Ο κανόνας priority(R,T,Z) μας υπολογίζει την προτεραιότητα ενός δρόμου με lineid R, τη χρονική στιγμή T με βάσει τα παραπάνω.

Στην υλοποίηση μας επιλέξαμε το priority να είναι της ίδιας τάξης μεγέθους με την απόσταση μεταξύ δυο διαδοχικών κόμβων και να αφαιρείται απο αυτό σε κάθε μετάβαση, μικραίνοντας έτσι ουσιαστικά το βάρος ακμής.

Σημείωση: Για τα διόδια ο χρήστης ερωτάται απο την αρχή άμα θέλει η διαδρομή του να περάσει ενδεχομένως και απο δρόμο που έχει διόδια , γεγονός που θα επιφέρει μεγαλύτερη χρέωση.

Σημαντικές διαφοροποιήσεις με την προηγούμενη υλοποίηση

Το γεγονός ότι πλέον οι ακμές του γράφου μοντελοποιούνται στην prolog μας επιτρέπει να αφαιρέσουμε τα πεδία previous και next απο την κλάση position που είχαμε στην προηγούμενη υλοποίηση. Ακόμα έπρεπε να διαφοροποιηθεί η απόσταση απο το συνδιασμένο βάρος ακμής που έχουμε τώρα, γι αυτό κρατάμε και την απόσταση όπως προκύπτει μόνο απο το edge weight.

Τέλος, απο τις ανάγκες του νέου προγράμματος, εκτελούμε τον A* μια φορά για την διαδρομή Αφετηρία πελάτη → Προορισμος πελάτη , όπου υπολογίζουμε και την απόσταση και στην συνέχεια εκτελούμε τον A* για τις διαδρομές κατάλληλο ταξι → Αφετηρία πελάτη.

Η οπτική απεικόνιση των διαδρομών για τα δοθέντα αρχεία εισόδου και ένα δικό μας πείραμα φαίνεται στο παρακάτω link:

<https://drive.google.com/open?id=10NNU33bAZodOIFcfr6n8aC9l5LQ&usp=sharing>

Τα αποτελέσματα για τα δεδομένα αρχεία εισόδου είναι (με avoid tolls == yes):

***** RANKING 1 *****

***** distance to client *****

- 1)210 || ~2.0 km away
- 2)230 || ~2.08 km away
- 3)220 || ~2.13 km away
- 4)170 || ~2.66 km away
- 5)120 || ~2.83 km away

***** RANKING 2 *****

***** rating and vehicle type *****

***** according ranking 1 *****

- 1) 230
- 2) 210
- 3) 120
- 4) 220
- 5) 170

Γενικά, στον κώδικα υπάρχουν πολλά σχόλια που βοηθύν στην κατανόηση του κώδικα.

Τρέχοντας τον αλγόριθμο για δικά μας αρχεία εισόδου, προέκυπταν μεγάλες αποστάσεις και μεγάλος όγκος δεδομένων, οπότε το google maps δεν το δεχόταν. Επισυνάπτουμε το αρχείο my_solution.kml όπου φαίνεται μια δικιά μας υλοποίηση.