Εθνικο Μετσοβιο Πολυτεχνειο

Τεχνητή Νοημοσύνη

Αναφορά 2ης Προγραμματιστικής Άσκησης

Γεωργίου Δημήτριος (03115106)
<el15106@central.ntua.gr>

Σαρμάς Ελισσαίος (03115111)
<el15111@central.ntua.gr>

Φεβρουάριος 2019

Γενικός Σχεδιασμός

Compile Run

- Η εφαρμγή γίνεται compile με την εντολή: *javac -classpath jipconsole.jar *.java*
- Για την εκτέλεση του προγράμματος απαιτούνται 1 command argumentm το directory με το αρχείο εισόδου. Ενδεικτικά *java -classpath jipconsole.jar Main*
- Το πρόγραμμα αφού εμφανίσει την διαθεσιμότητα των ταξί, στην συνέχεια βρίσκει τις συντομότερες διαδρομές για τα διαθέσιμα, και παράγει ως έξοδο το kml αρχείο που ζητείται για το πλέον κατάλληλο, με όνομα **"taxis.kml"**

Περιγραφή Προβλήματος

Στο παρόν Θέμα καλούμαστε να υλοποιήσουμε μια ευφυή υπηρεσία εξυπηρέτησης πελατών ταξί. Πιο συγκεκριμένα, θεωρούμε ότι υπάρχει ένας πελάτης που βρίσκεται σε μια ορισμένη τοποθεσία, ο οποίος επιθυμεί να καλέσει ένα ταξί δίνοντας στην υπηρεσία της συντεταγμένες του μέσω GPS κινητού τηλεφώνου. Η υπηρεσία διαθέτει μια βάση δεδομένων με 11 διαθέσιμα ταξί και τη γεωγραφική θέση στην οποία βρίσκονται (θεωρητικά η θέση των ταξί θα πρέπει να ανανεώνεται συνεχώς κάθε χρονική στιγμή). Η ευφυής υπηρεσία **σε πρώτο στάδιο** θα πρέπει να βρίσκει τα διαθέσιμα ταξί που μπορούν να εξυπηρετήσουν τον πελάτη βάσει διαθεσιμότητας, χωρητικότητας και γλώσσας που μιλάει οδηγός ταξί. **Σε δεύτερο στάδιο** θα πρέπει να υπολογίζει την συντομότερη διαδρομή για κάθε διαθέσιμο ταξί προς τον πελάτη, να ειδοποιεί το ταξί που βρίσκεται πιό κοντά στον πελάτη και συνεπώς που μπορεί να μεταβεί πιο γρήγορα κοντά στη θέση του πελάτη για να τον εξυπηρετήσει. **Σε τρίτο στάδιο** θα υπολογίζει και την συντομότερη διαδρομής του επιλεγμένου ταξί από την αφετηρία του πελάτη προς τον προορισμό του. Για τις επιλογές συντομότερων διαδρομών λαμβάνονται υπόψιν χαρακτηριστικά των δρόμων όπως η κατεύθυνση, η κίνηση, το είδος του δρόμου κλπ.

Δεδομένα Εισόδου

Λαμβάνουμε ως δεδομένα τις συντεταγμένες X,Υ (γεωγραφικό μήκος και γεωγραφικό πλάτος αντίστοιχα) για τον πελάτη, για τα ελεύθερα ταξί, για διάφορα σημεία του χάρτη, πως αυτά ορίζουν εν τέλει δρόμους (χωρίς την απλοποίηση του 1ου Θέματος) και πληροφορίες σχετικά με την κίνηση σε αυτούς.
Κατά το στάδιο του σχεδιασμού λαμβάνουμε υπ'όψιν πως πλέον τα σημεία του χάρτη έχουν δικό τους αντιπροσωπευτικό id, ενώ ανήκουν στην ίδια οδό αν έχουν ίδιο line id, ενώ όπως και στο 1ο Θέμα κάποια ανιστοιχούν σε σημεία τομής με άλλες οδούς.
Τα δεδομένα δίνονται σε 5 αρχεία τύπου .csv, στα αρχεία client, taxis, nodes, lines και traffic τα οποία περιέχουν τις συντεταγμένες της τοποθεσίας του πελάτη, τις συντεταγμένες και το id του κάθε ταξί, τις συντεταγμένες το όνομα της οδού και το line id κάθε κόμβου, τις γεωγραφικές γραμμές οι οποίες ορίζονται βάσει κόμβων και ενσωματώνουν χαρακτηριστικά για αυτές (εδώ έγκειται μεγάλ μέρος της διαφοροποίησης 2ου και 1ου Θέματος), και των ωραρίων κίνησης αντίστοιχα.
Τα δεδομένα εντός των αρχείων .csv δεν υπέστησαν κάποια προεπεξεργασία. Ωστόσο, οι συντεταγμένες X,Υ ήταν εισαγμένες σε ανάποδη σειρά. Δηλαδή αν εισαχθούν στο Google Maps αντιστοιχούν σε πολυ μακρινές τοποθεσίες, ενώ αν εισαχθούν ανάποδα αντισοτιχούν σε κόμβους εντός της Αθήνας το οποίο είναι και το επιθυμητό.

Δεδομένα Εξόδου

Το πρόγραμμα δημιουργεί ως έξοδο ένα αρχείο *.kml* με την ζητούμενη μορφή. Το αρχείο αυτό είναι κατάλληλο για την οπτική απεικόνιση των αποτελεσμάτων σε χάρτη, με τρόπο ώστε η ζητούμενη διαδρομή από τις αφετηρίες των ταξί προς τον πελάτη να απεικονίζεται με Χ χρώμα, η διαδρομή που θα ακολουθήσει το επιλεγμένο ταξί για να πάει στον πελάτη με Υ χρώμα και οι υπόπλοιπες διαδρομές με άλλα χρώματα

Σκιαγράφηση Λύσης

Σε πρώτο στάδιο:
Για την επίλυση του προβλήματος και την υλοποίηση της ευφυούς υπηρεσίας δεν αρκεστήκαμε στην εκτέλεση της παραλλαγής του αλγορίθμου A* που ορίσαμε στο 1ο Θέμα. Συκγκεκριμέμα, αρχικά το πρόγραμμα "μαθαίνει" τους κανόνες που ορίζουν τι είναι επιτρεπτό και τι όχι στον κόσμο μας από τους αντίστοιχους Parsers (των nodes, lines και traffic). Αυτοί κάνουν χρήση της μηχανής του JIProlog, η οποία στέλνει τα δεδομένα στην prolog για την δημιουργία των κατάλληλων κανόνων (με κατάλληλα queries, γεγονότα και κατηγορήματα) και τους αποθηκεύει στα αρχεία *nodes.pl, lines.pl, traffic.pl* αντίστοιχα.
Σε δεύτερο στάδιο:
Εφόσον έχει οριστεί ο κόσμος μας, στην συνέχεια υλοποιήσαμε τον αλγόριθμο A*. Στην πραγματικότητα έγινε τροποποιήση της παραλλαγής που είχαμε ορίσει στο 1ο Θέμα, ώστε να μην λαμβάνονται υπόψιν ισοδύναμες εναλλακτικές διαδρομές, από πλευράς κόστους, και να μην γίνει αναδρομική γραφή αυτών σε αρχείο .txt, καθώς δεν θα βρεθεί καμία. Συγκεκριμένα, αν λάβουμε υπόψιν μας ότι οι παραδοχές του 1ου Θέματος δεν ισχύουν, το πρόβλημα ανάγεται στην εύρεση του κοντινότερου **διαθέσιμου** (ώστε τα requirements πελάτη και να καλύπτονται από τα capabilities του εκάστοτε ταξί) ταξί και στην μετέπειτα αναζήτηση της συντομότερης διαδρομής για κάθε διαθέσιμου ταξί προς τον πελάτη και της συντομότερης διαδρομής του επιλεγμνένου ταξί από την αφετερεία του πελάτη προς τν προορισμό του. Επομένως, η έννοια της εναλλακτικής διαδρομής θα είχε φυσικό νόημα κυρίως κατά την διαδικασία εύρεσης συντομότερης διαδρομής στην δεύτερη περίπτωση, παρόλα αυτά αν δεν οριστεί ένα threshold διαφοράς μήκους πιθανών συντομότερων της τάξης των 30m, δεν θα βρεθεί ποτέ κάποια εναλλακτική.
Παρατηρήσεις:
Η υλοποίηση του αλγορίθμου A* έγινε στη γλώσσα προγραμματισμού Java. Για την υλοποίηση του αλγορίθμου χρησιμοποιούμε τις έννοιες του κλειστού συνόλου και του μετώπου αναζήτησης, καθώς και άλλα χαρακτηρηστικά τα οποία έχουν αναλυθεί στο 1ο Θέμα. Αξίζει να σημειώσουμε, πως στην συγκεκριμένη υλοποίηση, A* βρίσκει τα σημεία στα οποία μπορούμε να μεταβούμε από το σημείο που εξετάζουμε, κάνοντας queries στην Prolog.
Παραδοχές:
• Όλα τα σημεία τομής είναι σημεία διασταύρωσης.
Παραδοχές που **δεν ισχύουν** πλέον:
• Όλες οι οδοί είναι διπλής κατεύθυνσης.
• Όλες οι οδοί επιτρέπουν την ίδια ταχύτητα κίνησης.
• Όλα τα δεδομένα που σας δίνονται είναι οδοί όπου μπορούν να κινηθούν τα ταξί παρά το γεγονός ότι λόγω του απλουστευμένου τρόπου λήψης των δεδομένων, σε αυτά μπορεί να περιέχονται και άλλες γεωγραφικές γραμμές που δεν αντιστοιχούν σε οδούς (πχ. μονοπάτια, υδάτινες οδοί, όρια περιοχών, κλπ).

Περιγραφή Υλοποίησης

1ο Στάδιο - Γεγονότα Prolog

Η μορφή των κλάσεων είναι όμοια με αυτή του 1ου Θέματος, με την κύρια διαφορά ότι στο πρόγραμμα Java αποθηκεύονται μόνο τα IDs των ταξί, ενώ τα υπόλοιπα δεδομένα σε κατάλληλα αρχεία **.pl** ώστε να χρησιμοποιούνται ως βάση δεδομένων, στην οποία θα κάνουμε queries σε prolog.

1. Δημιουργία του αρχείου client.pl που βασίζεται στο client.csv με τα ακόλουθα γεγονότα:
    - client_coor(X,Y). όπου X, Y double για τις συντεταγμένες του πελάτη.
    - client_dest(X,Y). όπου X,Υ double για τις συντεταγμένες του προορισμού του.
    - client_time(X),όπου X string για την ώρα παραλαβής του πελάτη
    - client_persons(X). όπου X integer για το πλήθος των επιβατών.
    - client_language(X). όπου X string για την ώρα παραλαβής του πελάτη.
    - client_luggage(X). όπου X integer για το πλήθος των αποσκευών.

2. Δημιουργία του αρχείου taxis.pl που βασίζεται στο taxis.csv με τα ακόλουθα γεγονότα:

   - taxis_coor(Z,X,Y). όπου X,Υ double για τις συντεταγμένες του ταξί
   - taxis_available(Z,X). όπου X string για την διαθεσιμότητα του ταξί
   - taxis_capacity(Z,X). όπου X interger για το μέγιστο πλήθος επιβατών.
   - taxis_language(Z,X). όπου X string για την γλώσσα που μιλάει ο οδηγός.
   - taxis_rating(Z,X). όπου X double για την βαθμολογία του ταξί.
   - taxis_long(Z,X). όπου X string για το αν το ταξί εκτελεί μεγάλες αποστάσεις. Ως μεγάλη θεωρείται μία απόσταση πάνω από 10 km.
   - taxis_luggage(Z,X). όπου X integer για το μέγιστο πλήθος αποσκευών.

3. Δημιουργία του αρχείου lines.pl που βασίζεται στο lines.csv με τα ακόλουθα γεγονότα:

   - line(Z). για όσα lines είναι δρόμοι για αυτοκίνητα. Όσα δεν είναι, τα προσπερνάμε και δεν αναφέρονται καν στο αρχείο lines.pl. Αγνοούνται όσοι δρόμοι είναι railway, boundary, access, natural, barrier ή waterway.
   - line_oneway(Z). αν ο δρόμος είναι μονόδρομος.
   - line_oneway_back(Z). αν ο δρόμος είναιθ μονόδρομος, αλλά με αντίθετη φορά από αυτή που διαβάζονται τα nodes.
   - line_speed(Z,X). αν ο δρόμος έχει ανώτατο όριο ταχύτητας και ποιον είναι αυτό (X).
   - line_lanes(Z,X). αν γνωρίζουμε πόσες λωρίδες έχει ο δρόμος και πόσες είναι αυτές (X). Μάλιστα, αν υπάρχει γραμμεία λεωφορείων αφαιρείται ένα lane.
   - line_lit(Z,X). όπου X string για το αν ο δρόμος Z έχει φώτα.
   - line_tolls(Z,X). όπου X string για το αν ο δρομός Z απαιτεί διόδια
   - line_heuristic(Z,X). όπου X double για τον συντελεστή της ευριστικής, ο οποίος υπολογίζεται λαμβάνοντας υπόψιν τα παραπάνω και την τιμή των πεδίων highway και incline, όταν διαβάζεται το αρχείο lines.csv. Ο συντελεστής πολλαπλασιάζεται με την ευκλίδεια απόσταση κάθε node της γραμμής προς τον πελάτη και το γινόμενο εκφράζει την συνολική ευριστική του.

4. Δημιουργία του αρχείου nodes.pl που βασίζεται στο nodes.csv με τα ακόλουθα γεγονότα, όπου Z εκφράζει το ID του node.

   - nodes_line(Z,X). όπου X το ID της γραμμής που ανήκει το node. Ελέγχεται πρώτα αν το line είναι δρόμος. Ο έλεγχος γίνεται μέσω του query road(X) που βρίσκεται στο αρχείο prolog.pl. Όσα δεν ικανοποιούν την συνθήκη αυτή, τα προσπερνάμε και δεν αναφέρονται καν στο αρχείο nodesl.pl.
   - nodes_coor(Z,X,Y). όπου X,Υ double για τις συντεταγμένες του node
   - nodes_neighbors(Z,X). όπου X το ID ενός node στο οποίο μπορούμε να μεταβούμε από το Z. ο έλεγχος γίνεται από τα querries, oneway(X) και onewayBack(X) στο αρχείο prolog.pl

5. Δημιουργία του αρχείου traffic.pl που βασίζεται στο traffic.csv με τα ακόλουθα γεγονότα:

   - traffic(Z,X). όπου Z το line id και X ένα string με το είδος της κίνησης για εκείνο το δίωρο αν υπάρχει.

Επίσης αξίζει να αναφερθεί ότι μέσω της συνάρτησης *select_taxi()* , αφαιρούνται τα ταξί που δεν ικανοποιούν τα αιτήματα του πελάτη. Μάλιστα, τα δεδομένα που χρειάζονται γις τους ελέγχους και την ολοκλήρωση του 1ου σταδίου του προβλήματος προέρχονται από κατάλληλα Queries στα αρχεία που παρασκευάστηκαν.

Ανάλυση Ευριστικής

Σε αυτό το πρόγραμμα έχουν οριστεί οι εξής κανόνες που ρυθμίζουν τον κόσμο μας, συγκεκριμένα αυτοί αφορούν το lines.pl και το traffic.pl, και λαμβάνονται υπόψιν για τον υπολογισμό της προαναφερθείσας ευριστικής, που υποδηλώνει την καταλληλότητα ενός δρόμου γενικότερα. Ξεκινάμε με υποθετική αρχική ευριστική τιμή *heuristic = 1.3*, η οποία συμβολίζει μέτρο ακαταλλήτοτας, όσο πιο κοντά είναι στο 0 τόσο πιο κατάλληλος θεωρείται ο δρόμος, έτσι τα χαρακτηρστικά τον δρόμων είτε ενισχύουν είτε μειώνουν το κόστος αυτό.

- Αν ο δρόμος είναι motowray, τότε υπάρχει μεγάλο flow και έτσι μειώνουμε το κόστος κατά 1.2, αν είναι trunk κατά 0.9, αν είναι primary κατά 0.7, αν είναι secondary κατά 0.5, αν είναι tertiary αυξάνουμε κατά 0.3, αν είναι residential κατά 0.5, ανάλογα το πως ευνοεί σ
- Αν δεν υπάρχει φωτισμός στους δρόμους, αυξάνουμε την ευριστική τιμή κατά 0.1
- Ανάλογα των αριθμό των γραμμών, η μείωση που υφίσταται η heuristic είναι ανάλογη των διαθέσιμων λωρίδων που έχει ο δρόμος (λωρίδες * 0.1) (γίνεται exclude της λεωφορειογραμμής αν υπάρχει), καθώς σε περισσότερες λωρίδες υπάρχει μεγαλύτερος βαθμός ελευθερίας, ελίγμών και προσπεράσεων.
- Αν υπάρχει κλίση ανηφοράς αυξάνουμε το heuristic κατά 0.2 λόγω της εξτρα χρονικής καθυστέρησης συγκριτικά με έναν επίπεδο, ενώ αν υπάρχει κλίση κατηφόρας μειώνουμε κατά 0.2.
- Αν πρέπει να πληρώσουμε διόδια, το κόστος οφείλεται στην αναμονή και έτσι αυξάνουμε το heuristic κατά 0.3
- Στο σημείο αυτό εκτελούμε query προς το traffic.pl ώστε να ελέγξουμε το είδος της κίνησης που υπάρχει αν υπάρχει για την ζητούμενη ώρα μετακίνησης του πελάτη, και αν είναι high αυξάνουμε την ευριστική κατά 2, αν είναι medium κατά 1 και αν είναι low κατά 0.5. Παρατηρούμε ότι παρά τ γεγονός ότι οι δρόμοι για τους οποίους υπάρχει κίνηση είναι λίγοι, ο παράγοντας αυτός παίζει πολύ μεγάλο ρόλο, καθώς δυσχαιρένει την ποιότητα της εφαρμογής μας και δεν ευνοεί τον πελάτη.

2ο Στάδιο - Αλγόριθμος A*

- Σε αυτό το στάδιο καλείται ο κλασσικός αλγόριθμος A* (τροποποιημένος μετά το πέρας του 1ο Θέματος δεν απαιτούνται εναλλακτικές διαδρομές) για τα ταξί που ικανοποιούν τα αιτήματα του πελάτη. Ο A* χρησιμοποιεί τα Queries node_line(Z,X), nodes_neighbors(Z,X), nodes_corr(Z,X,Y,) και line_heuristic(Z,X) με σκοπό την ανάκτηση των απαραίτητων πληροφορίων. Μάλιστα επιλέγονται τα 5 καλύτερα βάσει απόστασης από αυτά και ταξινομούνται ως προς την μετρική rating (άρα την προτίμηση του πελάτη) για να επιλεχθεί το καλύτερο για αυτόν.
- Γίνεται κλήση ακόμα μια φορά του αλγορίθμου A*, αυτή την φορά για την εύρεση της συντομότερης διαδρομής από τον πελάτη πρός τον προορισμό, με το επιλεγμένο πλέον ταξί.

Υλοποίηση του Αλγορίθμου A*

Η υλοποίηση του αλγορίθμου A* έγινε μέσω της κλάσης Astar.

- Δίνουμε ως παραμέτρους το nodes_map που είναι τύπου NodesParser, και το οποίο αποτελεί το hashMap των κόμβων-δρόμων, ενώ ταυτόχρονα, τα endx endy που έχει ως πεδία υποδηλώνουν τον προορισμό μας, όταν αφετηρία θεωρείται οι συντεταγμένες του εκάστοτε ταξί.
- Ως ευριστική χρησιμοποιούμε την πραγματική απόσταση των σημειών, θα επεξηγηθεί παρακάτω. Στην close list καταγράφονται τα σημεία που έχουν ήδη εξερευνηθεί, ενώ στην queue εισάγουμε τα στοιχεία που εμπρόκειτο να εξευνηθούν στα επόμενα βήματα. Οι τιμές G,H,F αποτελούν χαρακτηριστικά κάθε κόμβου με προφανή ρόλο για τον αλγόριθμο, όπως εχει αναλυθεί στο 1ο Θέμα.

- Για την υλοποίηση του αλγορίθμου κάναμε queries στην Prolog για την εύρεση επόμενων σημείων, καθώς και για τον έλεγχο των μετατροπών που πρέπει να γίνουν στην F κάθε σημείου για να αυξομειώσουμε την προτεραιότητα του, πράγμα το οποίο επιτυγχάνεται με αύξηση ή μείωση του G κατά ένα ποσοστό της απόστασης μεταξύ 2 σημείων
- Για κάθε επόμενο στοιχείο που βρίσκουμε μέσω του next, ελέγχουμε σε ποιον δρόμο - γραμμή ανήκει, ώστε να μπορέσουμε να κάνουμε queries που αφορούν την συγκεκριμένη γραμμή.

Υλοποίηση Συναρτήσεων Εκτίμησης Απόστασης

Ως Ευριστική Συνάρτηση στην υλοποίηση μας επιλέξαμε την Ευκλίδεια απόσταση στο δισδιάστατο χώρο. Επομένως για κάθε σημείο του γράφου επιλέγουμε ως ευριστική συνάρτηση την απόσταση μεταξύ αυτού του σημείου και του σημείου στόχου που βρίσκεται ο πελάτης. Η επιλογή έγινε λόγω της απλότητας της στην υλοποίηση και του υψηλού ποσοστού επιτυχίας που έχει, όπως φαίνεται και στα αποτελέσματα παρακάτω.

Δομές Δεδομένων

Ο γράφος-χάρτης λοιπόν υλοποιείται με HashMap λόγω της ιδιότητας του να επιτελεί την αναζήτηση για τυχαίο σημείο σε **χρόνική πολυπλοκότητα Θ(1)**. Αυτή η ιδιότητα είναι πολύ χρήσιμη διότι κατά το διάβασμα των συντεταγμένων των σημείων πρέπει να γίνει γρήγορος έλεγχος για το αν υπάρχει ένα σημείο-node, αφού με τον τρόπο αυτό δηλώνονται οι διασταυρώσεις.
Εκτός του HashMap, όλες οι λίστες που απαιτούνται για την ανάπτυξη της εφαρμογής υλοποιούνται με την επιλογή ArrayList της Java, χάριν απλότητας και αποτελεσματικότητας της υλοποίησης της.

Περιγραφή Κλάσεων που χρησιμοποιήθηκαν

Για την υλοποίηση του ευφυούς συστήματος υλοποιήθηκαν συνολικά 11 κλάσεις με σκοπό την σωστή ανάλυση του προβλήματος και την απλούστευση της κατανόησης του κώδικα. Η λειτουργία κάθε κλάσης δεν θα παρουσιαστεί αναλυτικά παρακάτω για όλες, καθώς κάποιες υπέστησαν μικροαλλαγές

(1) **Point:** Η κλάση Point αντιπροσωπεύει ένα node του χάρτη.

(2) **ClientParser:** Χρησιμποιείται για την δημιουργία των γεγονότων που αφορών των πελάτη μας.

(3) **TaxisParser:** Χρησιμποιείται για την δημιουργία των γεγονότων που αφορούν τα διαφορετικά ταξί μας.

(4) **NodesParser:** Χρησιμποιείται για την δημιουργία των γεγονότων που αφορούν τους κόμβους του χάρτη, καθώς επίσης και την δημιουργία του hashMap, που περιέχει ως κλειδιά όλα τα σημεία του χάρτη και ως value όλους τους γείτονες του. Σημειώνεται ότι εδώ συγκριτικά με το 1ο Θέμα, υπάρχουν δρόμοι μονής και διπλής κατηγορίας, και βάσει της γνώσης αυτών μέσω κατάλληλων query στο nodes.pl μπορεί να περάσει και ως πληροφορία στο hasmap.

(5) **LinesParser:** Χρησιμποιείται για την δημιουργία των γεγονότων που αφορούν τους διαφορετικούς δρόμους και για τον υπολογισμό μιας εκτίμησης heuristic για την καταλλήλοτητα του καθενός.

(6) **TrafficParser:** Χρησιμποιείται για την δημιουργία των γεγονότων που αφορούν την κίνηση σε κάθε δρόμο

(7) **Astar, AstarSearch, AstarResult:** Συνολικά αφορούν την εκτέλεση του A*. Η Astar χρησιμοποιείται για να καλεί για κάθε ένα ταξί τον αλγόριθμο A* της AstarSearch και να αποδηκεύει το αποτέλεσμα της συντομότερης διαδρομής στην AstarResult.

(8) **Main:** Η κλάση Main αποτελεί την κεντρική κλάση της υλοποίησης μας.

Σημαντικές Παρατηρήσεις

(1) Κατά την διάρκεια οποιουδήποτε open από αρχείου ή read/write σε αρχείου τοποθετούμε try-catch για την καταπολέμηση πιθανών Exceptions που μπορεί να προκύψουν (**IOException** ή **FileNotFound** αντίστοιχα)

Αποτελέσματα - Οπτικές Αναπαραστάσεις

Παρακάτω επισυνάπτονται τα αποτελέσμα της εφαρμογής τόσο για το δοθέν Input της εκφώνησης. Το ID 0 αντιστοιχεί στον A* από τον πελάτη προς τον προορισμό του.

Για το Customised Input μας (client2.pl, taxis2.pl):