Αλγόριθμοι αναζήτησης – υλοποιήσεις σε MATLAB

## Περιγραφή

Η εργασία αυτή αφορά την υλοποίηση και την εφαρμογή των αλγορίθμων αναζήτησης Α\* και Ευριστικής αναζήτησης σε προγραμματιστικό περιβάλλον MATLAB. Οι δύο αλγόριθμοι θα εφαρμόζονται πάνω σε δύο παραδείγματα όπου το ένα είναι κατευθυνόμενος γράφος και το δεύτερο είναι το παράδειγμα με τις πόλεις της Ρουμανίας που είναι μη κατευθυνόμενος γράφος.

## Πρώτο Μέρος

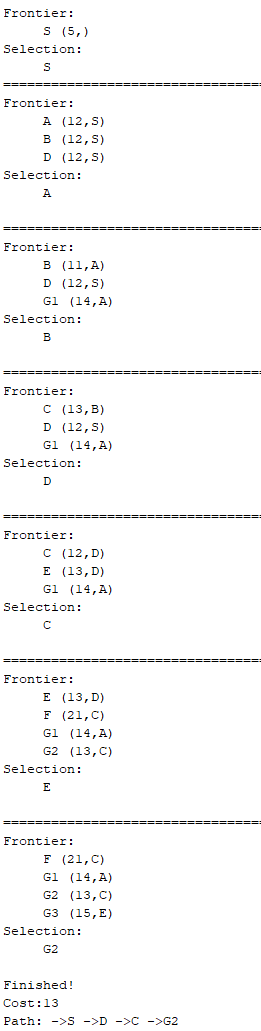
Στο πρώτο μέρος μας δίνεται ο κώδικας που αφορά τον αλγόριθμο αναζήτησης A\* και συμπληρώνεται ώστε να εμφανίζει τα δεδομένα με την μορφή που ζητούνται. Στον αλγόριθμο αυτό αρχικοποιούνται:

1. η λίστα με του κόμβους,
2. η λίστα με τις συνδέσεις του κάθε κόμβου και το κόστος που έχει με έναν άλλο,
3. και η λίστα με τις τιμές της ευριστικής συνάρτησης εκτίμησης απόστασης.

Χρησιμοποιήθηκε:

1. η συνάρτηση fprintf() του MATLAB για την εμφάνιση των δεδομένων,
2. η συνάρτηση table2cell για να μετατρέψουμε πίνακες σε μορφή cell
3. και η συνάρτηση string() ώστε να μετατρέψουμε σε κείμενο μεταβλητές που ήταν τύπου cell για να μπορούν να παρουσιαστούν.

Τα αποτελέσματα που παίρνουμε στο Command Window είναι ότι με 7 βήματα επιλέγεται η πιο σύντομη διαδρομή με κόστος 13 από το S ->D ->C ->G2.



## Δεύτερο Μέρος

Για το δεύτερο μέρος της εργασίας εφαρμόστηκε ο κώδικας του πρώτου μέρους στο παράδειγμα με τις πόλεις της Ρουμανίας από τις διαφάνειες του μαθήματος. Αντικαταστάθηκαν τα στοιχεία των λιστών με δεδομένα από το παράδειγμα αυτό.

NodesList={'Arad';'Bucharest';'Craiova';'Dobreta';'Eforie';'Fagaras';'Giurgiu';'Hirsova';'Iasi';'Lugoj';'Mehadia';'Neamt';'Oradea';'Pitesti';'Rimnicu Vilcea';'Sibiu';'Timisoara';'Urziceni';'Vaslui';'Zerind'};

% --- Heuristic function values

hList=[366;0;160;242;161;178;77;151;226;244;241;234;380;100;193;253;329;80;199;374];

% --- Link weights between nodes

LinksList={

'Arad','Zerind',75

'Arad','Sibiu',140

'Arad','Timisoara',118

'Bucharest','Fagaras',211

'Bucharest','Urziceni',85

'Bucharest','Giurgiu',90

'Bucharest','Pitesti',101

'Craiova','Dobreta',120

'Craiova','Rimnicu Vilcea',146

'Craiova','Pitesti',138

'Dobreta','Mehadia',75

'Eforie','Hirsova',86

'Fagaras','Sibiu',99

'Hirsova','Urziceni',98

'Iasi','Neamt',87

'Iasi','Vaslui',92

'Lugoj','Mehadia',70

'Lugoj','Timisoara',111

'Oradea','Sibiu',151

'Oradea','Zerind',71

'Pitesti','Rimnicu Vilcea',97

'Rimnicu Vilcea','Sibiu',80

'Urziceni','Vaslui',142

};

% --- Starting node

Start={'Arad'};

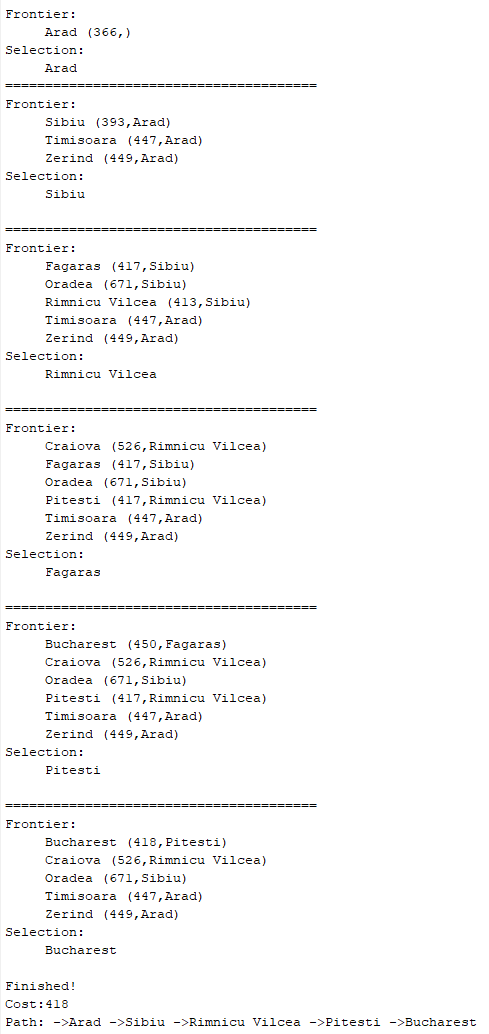
Σε αυτό το παράδειγμα όμως ο γράφος δεν είναι κατευθυνόμενος. Δηλαδή οι ακμές του από έναν κόμβο σε έναν άλλον είναι αμφίδρομες. Για παράδειγμα από το ARAD πρέπει να μπορούμε να πάμε στο ZERIND με κόστος διαδρομής 75 αλλά και το αντίστροφο με το ίδιο κόστος, αρκεί βέβαια να μην έχει επισκεφτεί μία πόλη ήδη μια φορά. Προγραμματιστικά υπάρχουν δύο λύσεις:

1. Είτε να περάσουμε τις συνδέσεις ξεχωριστά με το χέρι, για παράδειγμα στην LinksList να γράψουμε 'Arad','Zerind',75 και από κάτω 'Zerind','Arad',75
2. Είτε, όταν γίνεται η σύγκριση της πρώτης στήλης της LinksList με την πόλη (κόμβο) που βρισκόμαστε την δεδομένη στιγμή και επιστρέφει το κόστος και τις πόλεις της δεύτερης στήλης που συνδέονται με την πόλη αυτή, να προσθέσουμε παρόμοια σύγκριση με την δεύτερη στήλη της λίστας και να επιστρέφει το κόστος και τις πόλεις της πρώτης στήλης:



Με αυτόν τον τρόπο ο πίνακας AdjNodesG περιέχει σε κάθε βήμα όλες τις πόλεις και το κόστος τους από και προς την πόλη που εξετάζουμε.

Ο κώδικας που χρησιμοποιήθηκε για αυτή την εργασία ακολουθεί τον δεύτερο τρόπο για να εξετάζει όλες τις διαδρομές μέσα στην λίστα LinksList. Το αποτελέσμα που προκύπτει είναι ότι με 6 βήματα βρίσκει την βέλτιστη διαδρομή με κόστος 418 ακολουθώντας την διαδρομή από Arad ->Sibiu ->Rimnicu Vilcea ->Pitesti ->Bucharest.



## Τρίτο Μέρος

Στο τρίτο μέρος τροποποιείται ο κώδικας από τα δύο προηγούμενα μέρη αυτής της εργασίας ώστε να υλοποιείται ο αλγόριθμος αναζήτησης ομοιόμορφου κόστους. Η διαφορά του αλγορίθμου A\* με τον αλγόριθμο ομοιόμορφου κόστους είναι ότι στον δεύτερο επιλέγεται ο επόμενος κόμβος με βάση μόνο το κόστος και όχι με βάση το άθροισμα του κόστους g με μία τιμή της ευριστικής συνάρτησης εκτίμησης απόστασης h. Έτσι στον κώδικα αντικαθιστούμε την σχέση f=g+h με f=g όπου υπάρχει. Όμως η λίστα hList υπάρχει ακόμα, μόνο που όλες οι τιμές αντικαταστάθηκαν με 1 και οι τιμές των κόμβων που είναι οι στόχοι έμειναν 0 για να ελέγχεται ποιοι κόμβοι είναι οι στόχοι. Εφαρμόζοντας λοιπόν την αναζήτηση ομοιόμορφου κόστους στα δεδομένα των προηγούμενων μερών καταλήγουμε στα εξής αποτελέσματα:

Στα δεδομένα του πρώτου μέρους ακολουθείται ο ίδιος αριθμός βημάτων (7), βρίσκει το ίδιο κόστος (13) και την ίδια διαδρομή. Όμως στο 3ο με το 4ο βήμα, στον Α\* επιλέγεται ο D κόμβος και μετά ο B, ενώ στην αναζήτηση ομοιόμορφου κόστους γίνεται το αντίστροφο. Αυτό συμβαίνει διότι σε κάθε βήμα ο αλγόριθμος ομοιόμορφου κόστους ελέγχει μόνο ποιος είναι ο κοντινότερος επόμενος κόμβος χωρίς να υπολογίζει εάν αυτός αποτελεί μέρος της συντομότερης διαδρομής.

A picture containing scatter chart

Description automatically generated Graphical user interface

Description automatically generated with low confidence

Στα δεδομένα του δεύτερου μέρους με τις πόλεις της Ρουμανίας, επειδή ο γράφος είναι μη κατευθυνόμενος η αναζήτηση ομοιόμορφου κόστους ελέγχει πολλούς περισσότερους κόμβους που μπορεί να είναι πολύ πιο μακρία από τον τελικό στόχο και οδηγείται στο ίδιο αποτέλεσμα κόστους και διαδρομής αλλά με διπλάσιο αριθμό βημάτων (από 6, ελέγχει 13 ).

A picture containing table

Description automatically generated 