Τεχνητή Νοημοσύνη

Άσκηση 1η

Παντερής Γεώργιος Α.Μ.: 03107076
Παπανικήτας Αθανάσιος Α.Μ.: 03107778

1.

Σκοπός της άσκησης, είναι η κατασκευή ενός προγράμματος, το οποίο κάνοντας χρήση ενός αλγορίθμου εύρεσης βέλτιστης λύσης με ευριστικές τεχνικές, θα καθοδηγεί τις κινήσεις ενός ρομπότ ώστε να συναντήσει με τον ελάχιστο αριθμό κινήσεων ,ξεκινώντας από μία αρχική θέση του χώρου, ένα δεύτερο ρομπότ το οποίο κινείται τυχαία. Την επίλυση του προβλήματος, δυσχεραίνει το γεγονός, ότι στον χώρο υπάρχουν εμπόδια τα οποία τα ρομπότ πρέπει να αποφύγουν. Επίσης πρέπει να ληφθεί υπόψην το γεγονός ότι το Ρομπότ 1 κινείται με τριπλάσια ταχύτητα από το Ρομπότ 2 με αποτέλεσμα να συναντηθούν (αν τα εμπόδια το επιτρέπουν).

Για την επίλυση του προβλήματος, αρχικά έπρεπε να δημιουργήσουμε ένα μοντέλο για τα δεδομένα εισόδου. Επιλέχτηκε, αυτό να είναι ένα αρχείο εισόδου, στο οποίο δίνονται σε καρτεσιανές συντεταγμένες, οι διαστάσεις NxM του χώρου στον οποίο κινούνται τα ρομπότ και οι αρχικές τους θέσεις. Έπειτα ακολουθεί ένα κείμενο N γραμμών και M στηλών που αναπαριστά κάθε σημείο του χώρου και σε αυτό σημειώνεται X αν υπάρχει κάποιο εμπόδιο ή O αν η θέση είναι κενή. Οι συντεταγμένες x και y αριθμούνται από 1..N και 1 έως 1..M αντίστοιχα και η πρώτη γραμμή του κειμένου αντιστοιχεί στα σημεία με x = 1 και y από 1 έως M.

Ο αλγόριθμός εύρεσης βέλτιστης λύσης που χρησιμοποιείται είναι ο A* σε δύο διαφορετικές του εκδόσεις, μία με υπό-εκτιμητή και μία με υπέρ-εκτιμητή. Για το δέντρο των πιθανών κινήσεων, το οποίο κάνει χρήση ο αλγόριθμος, έχει δημιουργηθεί μια ειδική δομή η οποία θα αναλυθεί λεπτομερώς παρακάτω. Το πρόβλημα λειτουργεί ως εξής:

Το Ρομπότ 2 κάνει μία τυχαία κίνηση χρησιμοποιώντας τη συνάρτηση random και ανάλογα με τη θέση στην οποία θα βρεθεί, το Ρομπότ 1 εκτελεί τον A* μέχρι αυτή τη θέση και κρατάει τα 3 πρώτα βήματα κάνοντας έτσι την κινησή του. Μετά έρχεται η σειρά του Ρομπότ 2 να κινηθεί και μετά ξανά το Ρομπότ 1 κ.ο.κ. Αυτή η διαδικασία συνεχίζεται μέχρι μέχρι το Ρομπότ 1 να συναντήσει το 2. Για την υλοποίηση του A*, έγινε χρήση μιας ουράς προτεραιότητας ελαχίστου ώστε σε κάθε βήμα να εξετάζεται η κίνηση που ελαχιστοποιεί τον εκτιμητή και χρήση ενός πίνακα M*N στον οποίο αποθηκεύονται οι βέλτιστες διαδρομές προς όλους τους κόμβους και 5 καθολικών δεικτών (Start, Top, Active, Last , Exported).

2.

Η ουρά προτεραιότητας του δέντρου αναζήτησης του A*, είναι ένας δυαδικός σωρός οι κόμβοι του οποίου είναι δομές δεδομένων που περιέχουν τα απαραίτητα στοιχεία για τη λύση του προβλήματος. Οι κόμβοι είναι συνδεδεμένοι και σα διπλά συνδεδεμένη λίστα που βοηθάει στην ταχύτερη επεξεργασία του σωρού. Στον σωρό ισχύει ότι κάθε πατέρας έχει μικρότερη τιμή απόστασης(εκτίμησης και ήδη διανυθείσας απόστασης) από τα παιδία του.

Για τους κόμβους του διαδυκού σωρού έγινε χρήση της ακόλουθης δομής:

```
typedef struct str1 node;

struct str1
{
	int C[2];
	int eval;
	int step;
	node *path;
	node *parent;
	node *previous;
	node *next;
	node *right;
	node *left;
};
```

Κάθε κόμβος αναπαριστά ένα σημείο του χώρου. Οι τιμές C[0] και C[1] αντιστοιχούν στις συντεταγμένες του σημείου αυτού. Οι τιμές eval και step είναι η εκτίμηση του κόμβου και η απόσταση που έχει ήδη διανυθεί αντίστοιχα. Ο δείκτης path δείχνει τον προηγούμενο κόμβο του μονοπατιού. Ο δείκτης parent δείχνει στον πατέρα του κάθε κόμβου στο δέντρο. Οι δείκτες right και left δείχνουν τα δύο παιδιά του κόμβου αυτού (δεξί και αριστερό αντίστοιχα). Τέλος, οι δείκτες previous και next υλοποιούν τη διπλά συνδεδεμένη λίστα.

Ο δισδιάστατος πίνακας δεικτών Grid χρησιμοποιείται για να γίνεται το prouning (κλάδεμμα) στην υλοποίηση του A*. Συγκεκριμένα, κρατάει την καλύτερη τιμή απόστασης προς κάθε κόμβο κρατώντας και το βέλτιστο μονοπάτι προς αυτόν.

Ο Start δείχνει τον κόμβο από τον οποίο ξεκινάει το Ρομπότ 1 με συντεταγμένες τη θέση του και κατάλληλα αρχικοποιημένες τιμές. Ο Top δείχνει τη ρίζα του σωρού. Ο Active δείχνει τον κόμβο στον οποίο θα εισαχθεί ένας νέος κόμβος. Ο Last δείχνει τον τελευταίο κόμβο του σωρού. Τέλος, ο Exported δείχνει τον τελευταίο κόμβο που έχει εξαχθεί από το σωρό.

3.

Για την υλοποίηση του A* που θα τρέξει το Ρομπότ 1 χρησιμοποιήθηκαν οι εξής συναρτήσεις:

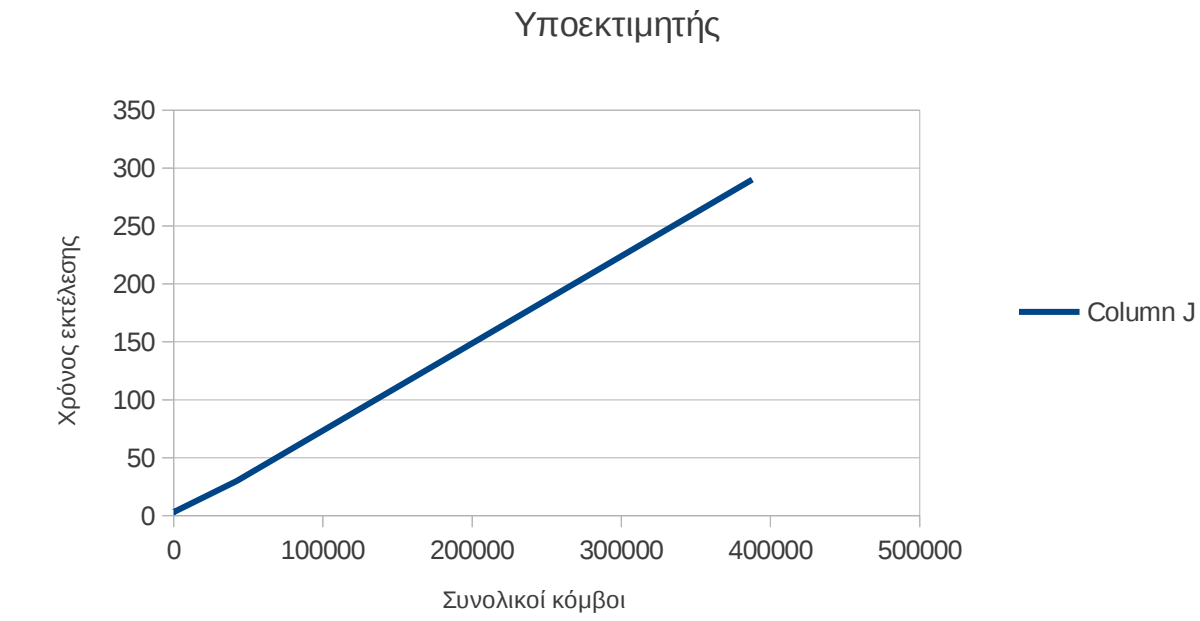
- initialisation() : κάνει αρχικοποίση του διαδικού σωρού των 4 γειτονικών κόμβων της θέσης του R1 και τις μεταθέτει κατάλληλα.
- export() : βγάζει τη ρίζα του σωρού καθώς αυτή έχει τη μικρότερη τιμή απόστασης και βάζει στη θέση της τον τελευταίο κόμβο (Last).
- allocatetop() : γίνεται μετά το export και μεταθέτει κατάλληλα από πάνω προς τα κάτω τους κόμβους ώστε να ισχύει ότι η τιμή κάθε πατέρα να είναι μικρότερη της τιμής των παιδιών του.
- newnodes() : βρίσκει τα γειτονικά τετράγωνα του κόμβου που εξάχθηκε (Exported) στα οποία μπορεί να μεταβεί το Ρομπότ και δημιουργεί ανάλογους κόμβους για να εισαχθούν στη συνέχεια.
- insert() : εισάγει ένα νέο κόμβο στην τελευταία θέση του σωρού ανάλογα με το αποτέλεσμα της newnodes.
- allocatelast() : μεταθέτει τον κόμβο του ορίσματός της στη σωστή θέση στο σωρό ανάλογα με την τιμή της απόστασης κάνωντας συγκρίσεις από κάτω προς τα πάνω, ώστε στην τελική κατάσταση η τιμή κάθε πατέρα να είναι μικρότερη της τιμής των παιδιών του.

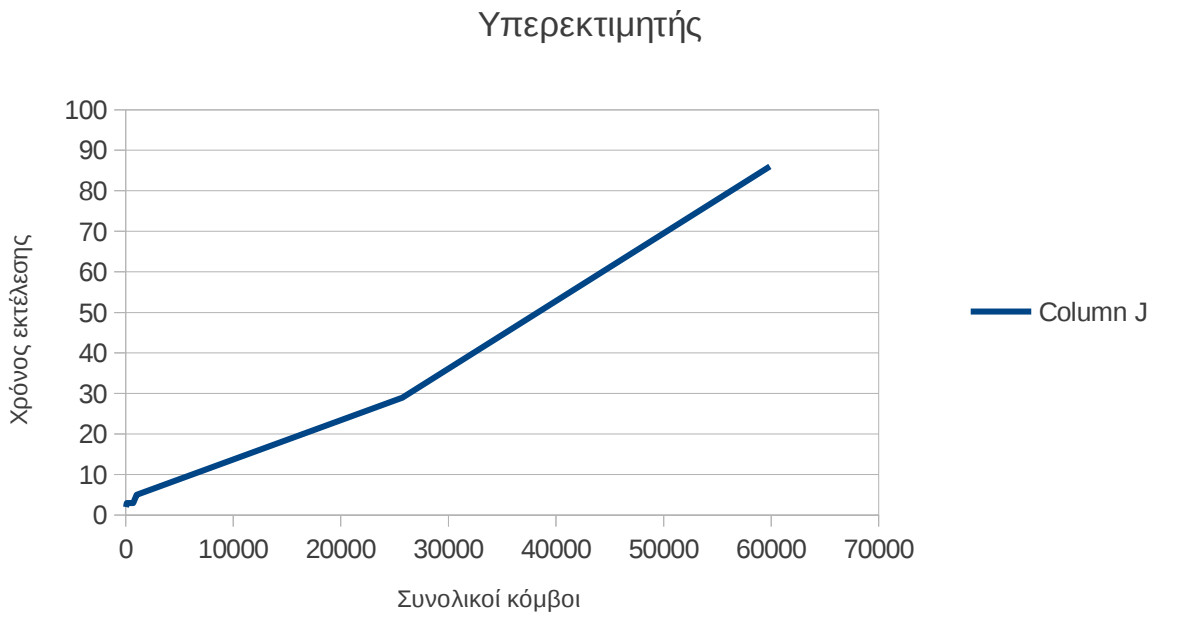
4.

Ως υπό-εκτιμητής χρησιμοποιήθηκε η απόσταση Manhattan καθώς είναι η μικρότερη απόσταση που απαιτείται για να φτάσει το Ρομπότ 1 στο 2. Ως υπέρ-εκτιμητή χρησιμοποιήσαμε το δεκαπλάσιο του τετραγώνου της απόστασης Manhattan. Η τιμή αυτή θα είναι πάντα μεγαλύτερη από της υποεκτίμησης εκτός από την περίπτωση γειτονικών τετραγώνων ή του ίδιου, όπου οι δύο τιμές είναι ίσες.

5.

Παρακάτω παραθέτονται τα ζητούμενα διαγράμματα συνολικών κόμβων - χρόνου εκτέλεσης για τις 2 περιπτώσεις(υποεκτιμητής, υπερεκτιμητής) :





6.

Υποεκτιμητής					
	Διαστάσεις χάρτη	Συνολικά βήματα	Κόμβοι για 1 κίνηση	Συνολικοί κόμβοι	Χρόνος εκτέλεσης(msec)
test case 1	20 * 13	2	22	32	3
test case 2	20 * 13	6	37	147	3
test case 3	20 * 13	11	132	835	4
test case 4	20 * 13	12	122	1363	4
test case 5	50 * 30	49	1347	42195	30
test case 6	180 * 180	118	10244	387813	290

Υπερεκτιμητής					
	Διαστάσεις χάρτη	Συνολικά βήματα	Κόμβοι για 1 κίνηση	Συνολικοί κόμβοι	Χρόνος εκτέλεσης
test case 1	20 * 13	3	22	48	2
test case 2	20 * 13	6	30	100	3
test case 3	20 * 13	12	117	702	3
test case 4	20 * 13	18	174	1023	5
test case 5	50 * 30	51	1070	25710	29
test case 6	180 * 180	153	826	59882	86

Ο παραπάνω πίνακας παρουσιάζει τα στατιστικά που λαμβάνουμε κατά την εκτέλεση του προγράμματος και σύμφωνα με τα οποία προέκυψαν τα γραφήματα.

Ο χρόνος μετρήθηκε με την συνάρτηση time του λειτουργικού συστήματος linux. Τα αρχεία εισόδου κατευθύνθηκαν στο standard input με την εντολή:

time ./project1-final.out < test180.txt

Ανάλυση και σχολιασμός:

- Παρατηρούμε ότι αύξηση του αριθμού των κόμβων του δέντρου, συνοδεύεται από αύξηση του χρόνου, πολλές φορές και γραμμική και στις 2 περιπτώσεις.
- Βλέπουμε ότι με τον υπέρ-εκτιμητή, το πρόγραμμα εξετάζει λιγότερους κόμβους και τελειώνει πιο γρήγορα, αυτό συμβαίνει γιατί η μεγάλη τιμή του εκτιμητή, οξύνει τις διαφορές και εισάγει λιγότερους κόμβους στο δέντρο. Αυτό θα μπορούσε να αλλάξει κάνοντας χρήση ενός μεταβλητού εκτιμητή, οποίο όμως δεν θα είχε σχέση με την απόσταση, όπως π.χ. το τετράγωνο του αριθμού των βημάτων που έχουν γίνει.
- Παρατηρούμε ότι ο υποεκτιμητής βρίσκει πάντα το βέλτιστο μονοπάτι, ενώ ο υπερεκτιμητής όχι γεγονός που φαίνεται από τον αριθμό των βημάτων. Συγκεκριμένα, ο υπερεκτιμητής βρίσκει πιο γρήγορα μονοπάτι αλλά δεν είναι πάντα το βέλτιστο.