Διδάσκων : Α. Λύκας
ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2022
ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

Εργαστηριακή Άσκηση 1 (Αναζήτηση σε λαβύρινθο)

Στοιχεία μελών ομάδας :

Ονομ/πώνυμο	Α.Μ	Email
Μοδιώτης Αθανάσιος	4736	cs04736@uoi.gr
Μπονίτσης Παντελής	4742	cs04742@uoi.gr
Σιδηρόπουλος Γεώργιος	4789	cs04789@uoi.gr

1. Γενικές πληροφορίες για τον κώδικα της άσκησης

Η εργαστηριακή άσκηση έχει υλοποιηθεί σε γλώσσα προγραμματισμού **Java**. Στον κατάλογο της άσκησης περιέχονται τρία αρχεία κώδικα. Αυτά είναι τα **Labyrinth.java** , **Cell.java** και **MinPQ.java**. Το **εκτελέσιμο αρχείο** (αυτό που περιέχει την μέθοδο main) είναι το **Labyrinth.java**.

MinPQ.java

Η κλάση MinPQ υλοποιεί μία ουρά προτεραιότητας ελάχιστου (min priority queue) η οποία αποτελεί το μέτωπο αναζήτησης των αλγορίθμων.

Περιέχει μεθόδους που εκτελούν βασικές λειτουργίες μερικές από τις οποίες είναι η εισαγωγή αντικειμένου με συγκεκριμένο κλειδί (void insert(Item v, Key k)) , η επιστροφή του αντικειμένου με το ελάχιστο κλειδί (Item minItem()) καθώς η διαγραφή του από την ουρά (void delMin()).

Στην συγκεκριμένη περίπτωση τα αντικείμενα θα είναι τύπου Cell (κελί) και τα κλειδιά τους τύπου double τα οποία θα αποτελούν τα κόστη.

Labyrinth.java

Στην κλάση Labyrinth ουσιαστικά δημιουργείται ο λαβύρινθος ο οποίος αποτελείται από έναν NxN πίνακα ο οποίος περιέχει αντικείμενα Cell (κελιά). Επιπλέον στην κλάση αυτή υλοποιούνται οι αναζητήσεις ομοιόμορφου κόστους (UCS) και A*.

Βασικότερες μέθοδοι:

- boolean **isExpandedUcs**(Cell newCell): Επιβεβαιώνει αν μία κατάσταση (ένα κελί) είναι μέλος του κλειστού συνόλου. Η μέθοδος υπάρχει για τον αλγόριθμο UCS και τον αλγόριθμο A*.

- void **createLabyrinth**(): Δημιουργεί τον λαβύρινθο, αποφασίζει τυχαία για κάθε κελί αν περιέχει εμπόδιο και δίνει τυχαία σε κάθε κελί μία τιμή val (1 έως 4).

- ArrayList<Cell> **algorithm**(String nameAlgorithm, Cell S, Cell G1, Cell G2): Εκτελεί τους αλγορίθμους UCS και A*.

- ArrayList<Cell> **createPath**(Cell S, ArrayList<Cell> wrongPath): Δημιουργεί το τελικό μονοπάτι της πλησιέστερης διαδρομής σε τελική κατάσταση.

Cell.java

Η κλάση Cell ουσιαστικά υλοποιεί ένα κελί του λαβύρινθου το οποίο κρατεί πληροφορίες ως πεδία όπως οι συντεταγμένες του, οι συντεταγμένες του κελιού το οποίο το έβαλε στο μέτωπο αναζήτησης καθώς και τα οριζόντια, κάθετα και διαγώνια γειτονικά κελιά του.

Βασικότερες μέθοδοι:

- boolean **isParent**(Cell other): Επιβεβαιώνει αν ένα κελί έβαλε στο μέτωπο αναζήτησης ένα άλλο.

- double **calculateH**(int rowG, int columnG): Υπολογίζει την τιμή της ευρετικής συνάρτησης για μία κατάστασης (κελί).

- void **insertNeighboursUCS**(MinPQ<Cell,Double> pq, int N): Βάζει στο μέτωπο αναζήτησης τα γειτονικά κελιά (καταστάσεις-παιδιά) από ένα κελί (κατάσταση). Η μέθοδος υπάρχει για τον αλγόριθμο UCS και τον αλγόριθμο A*.

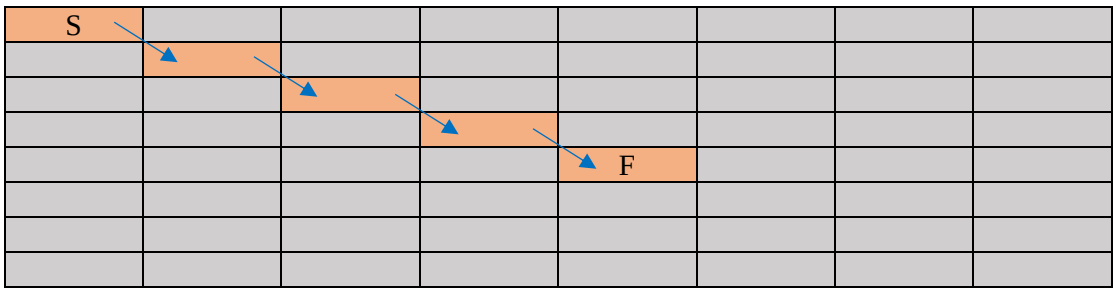
- MinPQ<Cell,Double> **fixPQUcs**(MinPQ<Cell,Double> pq, Cell cellToInsert, int N) : Όταν πρόκειται να μπει στο μέτωπο αναζήτησης ένα κελί (κατάσταση) ενώ υπάρχει ήδη μέσα σε αυτό ένα ίδιο τότε η μέθοδος φροντίζει να μπει ή να διατηρηθεί το κελί με το μικρότερο κόστος διαδρομής. Η μέθοδος υπάρχει για τον αλγόριθμο UCS και τον αλγόριθμο A*.

2. Ευρετική συνάρτηση h(n)

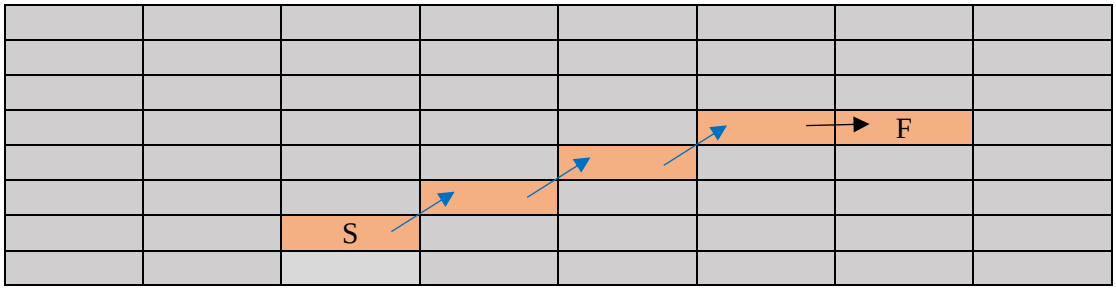
Με βάση την εκφώνηση της άσκησης γίνεται κατανοητό ότι το κόστος για μία διαγώνια μετακίνηση (Δ + 0.5) είναι μικρότερο από αυτό μίας οριζόντιας ή κάθετης (Δ + 1).

Για αυτόν τον λόγο από ένα αρχικό κελί έως και ένα τελικό χρησιμοποιούνται όσο το δυνατό περισσότερες διαγώνιες μετακινήσεις για τις οποίες προστίθεται η τιμή 0.5 και αν είναι αναγκαίο χρησιμοποιείται επιπλέον μία οριζόντια ή κάθετη για τις οποίες προστίθεται η τιμή 1. Έτσι προκύπτει η τιμή της ευρετικής συνάρτησης h(n) για κάθε κελί (κατάσταση).

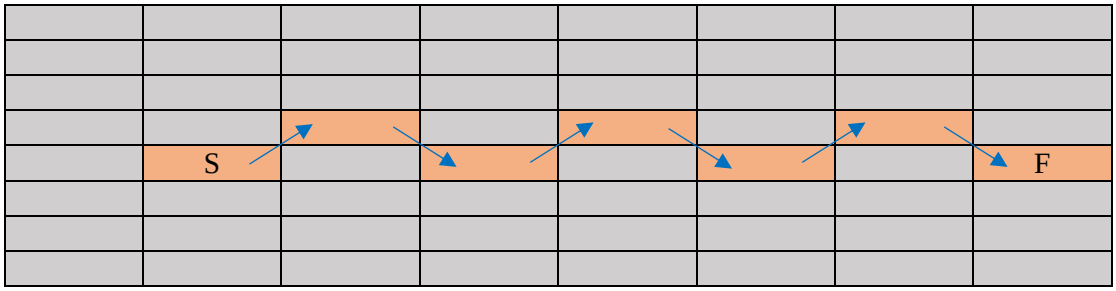
Παρακάτω παρατίθενται μερικά σχηματικά παραδείγματα σε έναν 8x8 λαβύρινθο.



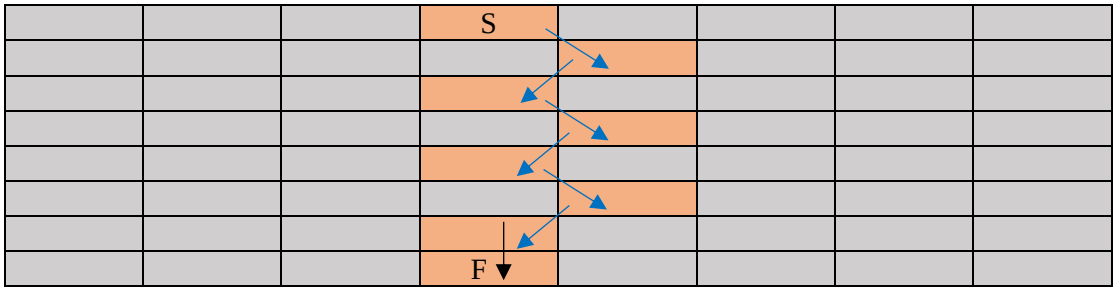
h(S) = 0.5 * 4 = 2



h(S) = (0.5 * 3) + 1 = 2.5



h(S) = 0.5 * 6 = 3



h(S) = (0.5 * 6) + 1 = 4

Με βάση την θεωρία είναι γνωστό ότι μία ευρετική συνάρτηση είναι αποδεκτή αν για κάθε κατάσταση η τιμή της είναι μικρότερη ή το πολύ ίση με την πραγματική απόσταση της κατάστασης από την πλησιέστερη τελική.

Στην χείριστη περίπτωση το ελάχιστο κόστος μίας διαγώνιας μετακίνησης είναι 0.5 ενώ το ελάχιστο κόστος μίας οριζόντιας ή κατακόρυφης μετακίνησης είναι 1. Αυτό συμβαίνει όταν τα δύο γειτονικά κελιά έχουν ίδιες τυχαίες τιμές val (πχ 3 και 3).

Επομένως στην χείριστη περίπτωση η τιμή της ευρετικής συνάρτησης είναι ίση με την πραγματική απόσταση από την πλησιέστερη τελική κατάσταση, ενώ για όλες τις υπόλοιπες είναι μικρότερη. Έτσι η ευρετική συνάρτηση είναι **αποδεκτή**.

3. Συμπεράσματα για την αποδοτικότητα των αλγορίθμων μέσω πειραμάτων.

Πείραμα πρώτο: N = 3 , p = 0.1

Λαβύρινθος:

```
3x3 Labyrinth (p = 0.1):

[0][0] Val:4        [0][1] Val:4        [0][2] Val:4

   (Blocked)       [1][1] Val:2        [1][2] Val:2

[2][0] Val:2        [2][1] Val:4        [2][2] Val:2
```

Αποτελέσματα αλγορίθμων:

```
--------- UCS algorithm ---------

 S cell: [0][0] Val:4
G1 cell: [1][2] Val:2
G2 cell: [2][2] Val:2

----Path----
[0][0] Val:4        Cost: 0.0
[1][1] Val:2        Cost: 2.5
[2][2] Val:2        Cost: 3.0

Total cost of path: 3.0
Number of expansions: 6.0

Running time: 2 ms
```

```
-------- Astar algorithm --------

 S cell: [0][0] Val:4
G1 cell: [1][2] Val:2
G2 cell: [2][2] Val:2

----Path----
[0][0] Val:4        Cost: 0.0
[1][1] Val:2        Cost: 2.5
[2][2] Val:2        Cost: 3.0

Total cost of path: 3.0
Number of expansions: 5.0

Running time: 1 ms
```

Γίνεται κατανοητό ότι οι δύο μέθοδοι έχουν βρει λύση η οποία είναι και βέλτιστη. Βέβαια η αναζήτηση A* είναι ελάχιστα γρηγορότερη ενώ έχει πραγματοποιήσει και λιγότερες επεκτάσεις από την αναζήτηση ομοιόμορφου κόστους (UCS).

Πείραμα δεύτερο: N = 5 , p = 0.25

Λαβύρινθος:

```
5x5 Labyrinth (p = 0.25):

   (Blocked)       [0][1] Val:3       [0][2] Val:1       [0][3] Val:3       [0][4] Val:3

   (Blocked)       [1][1] Val:3          (Blocked)       [1][3] Val:4          (Blocked)

   (Blocked)          (Blocked)       [2][2] Val:3       [2][3] Val:2          (Blocked)

   (Blocked)       [3][1] Val:2          (Blocked)       [3][3] Val:4       [3][4] Val:2

[4][0] Val:4        [4][1] Val:2       [4][2] Val:4        [4][3] Val:1       [4][4] Val:1
```

Αποτελέσματα αλγορίθμων:

```
--------- UCS algorithm ---------

 S cell: [3][4] Val:2
G1 cell: [0][1] Val:3
G2 cell: [4][0] Val:4

----Path----
[3][4] Val:2        Cost: 0.0
[2][3] Val:2        Cost: 0.5
[2][2] Val:3        Cost: 2.5
[1][1] Val:3        Cost: 3.0
[0][1] Val:3        Cost: 4.0

Total cost of path: 4.0
Number of expansions: 11.0

Running time: 6 ms
```

```
-------- Astar algorithm --------

 S cell: [3][4] Val:2
G1 cell: [0][1] Val:3
G2 cell: [4][0] Val:4

----Path----
[3][4] Val:2        Cost: 0.0
[2][3] Val:2        Cost: 0.5
[2][2] Val:3        Cost: 2.5
[1][1] Val:3        Cost: 3.0
[0][1] Val:3        Cost: 4.0

Total cost of path: 4.0
Number of expansions: 7.0

Running time: 1 ms
```

Γίνεται κατανοητό ότι όπως και στο προηγούμενο πείραμα οι δύο μέθοδοι βρίσκουν λύση η οποία είναι η βέλτιστη. Η αναζήτηση A* παραμένει πιο γρήγορη και με λιγότερες επεκτάσεις από την αναζήτηση ομοιόμορφου κόστους (UCS). Βέβαια η διαφορά στον χρόνο και στον αριθμό των επεκτάσεων έχει αυξηθεί.

Πείραμα τρίτο: N = 5 , p = 0.6

Λαβύρινθος:

```
5x5 Labyrinth (p = 0.6):

  (Blocked)        (Blocked)        (Blocked)        (Blocked)      [0][4] Val:2

  (Blocked)        (Blocked)        (Blocked)        (Blocked)        (Blocked)

  (Blocked)      [2][1] Val:2         (Blocked)      [2][3] Val:2     [2][4] Val:4

  (Blocked)      [3][1] Val:1       [3][2] Val:1     [3][3] Val:2         (Blocked)

  (Blocked)      [4][1] Val:1         (Blocked)        (Blocked)        (Blocked)
```

Αρχικά λόγω της αυξημένης πιθανότητας p παρατηρείται ραγδαία αύξηση των κελιών που περιέχουν εμπόδια.

Αποτελέσματα αλγορίθμων:

```
--------- UCS algorithm ---------

 S cell: [2][1] Val:2
G1 cell: [3][3] Val:2
G2 cell: [2][4] Val:4

----Path----
[2][1] Val:2        Cost: 0.0
[3][2] Val:1        Cost: 1.5
[3][3] Val:2        Cost: 3.5

Total cost of path: 3.5
Number of expansions: 6.0

Running time: 0 ms
```

```
-------- Astar algorithm --------

 S cell: [2][1] Val:2
G1 cell: [3][3] Val:2
G2 cell: [2][4] Val:4

----Path----
[2][1] Val:2        Cost: 0.0
[3][2] Val:1        Cost: 1.5
[3][3] Val:2        Cost: 3.5

Total cost of path: 3.5
Number of expansions: 5.0

Running time: 0 ms
```

Γίνεται κατανοητό ότι όπως και στα προηγούμενα πειράματα οι δύο μέθοδοι βρίσκουν λύση η οποία είναι η βέλτιστη. Επιπλέον λόγω των αυξημένων εμποδίων το μήκος του τελικού μονοπατιού έχει μειωθεί. Τέλος η αναζήτηση A* έχει τον ίδιο μηδενικό χρόνο και ελάχιστα λιγότερες επεκτάσεις από την αναζήτηση ομοιόμορφου κόστους (UCS).

Πείραμα τέταρτο: N = 8 , p = 0.25

Λαβύρινθος:

```
8x8 Labyrinth (p = 0.25):

   (Blocked)        (Blocked)      [0][2] Val:3        (Blocked)        (Blocked)        (Blocked)        (Blocked)      [0][7] Val:1
[1][0] Val:3      [1][1] Val:4         (Blocked)        (Blocked)        (Blocked)      [1][5] Val:1      [1][6] Val:1      [1][7] Val:1
[2][0] Val:2         (Blocked)        (Blocked)      [2][3] Val:4      [2][4] Val:1         (Blocked)        (Blocked)      [2][7] Val:1
[3][0] Val:2      [3][1] Val:1      [3][2] Val:1      [3][3] Val:4      [3][4] Val:2      [3][5] Val:3         (Blocked)        (Blocked)
[4][0] Val:2      [4][1] Val:2      [4][2] Val:3        (Blocked)      [4][4] Val:2      [4][5] Val:1         (Blocked)      [4][7] Val:4
[5][0] Val:4         (Blocked)        (Blocked)      [5][3] Val:2      [5][4] Val:3      [5][5] Val:4         (Blocked)      [5][7] Val:2
[6][0] Val:3      [6][1] Val:4      [6][2] Val:1      [6][3] Val:4        (Blocked)      [6][5] Val:4      [6][6] Val:1      [6][7] Val:2
   (Blocked)        (Blocked)      [7][2] Val:1      [7][3] Val:2        (Blocked)      [7][5] Val:2      [7][6] Val:3      [7][7] Val:3
```

Αποτελέσματα αλγορίθμων:

```
--------- UCS algorithm ---------

 S cell: [1][1] Val:4
G1 cell: [6][5] Val:4
G2 cell: [7][6] Val:3

----Path----
[1][1] Val:4        Cost: 0.0
[2][0] Val:2        Cost: 2.5
[3][0] Val:2        Cost: 3.5
[4][1] Val:2        Cost: 4.0
[4][2] Val:3        Cost: 6.0
[5][3] Val:2        Cost: 7.5
[5][4] Val:3        Cost: 9.5
[6][5] Val:4        Cost: 11.0

Total cost of path: 11.0
Number of expansions: 28.0

Running time: 17 ms
```

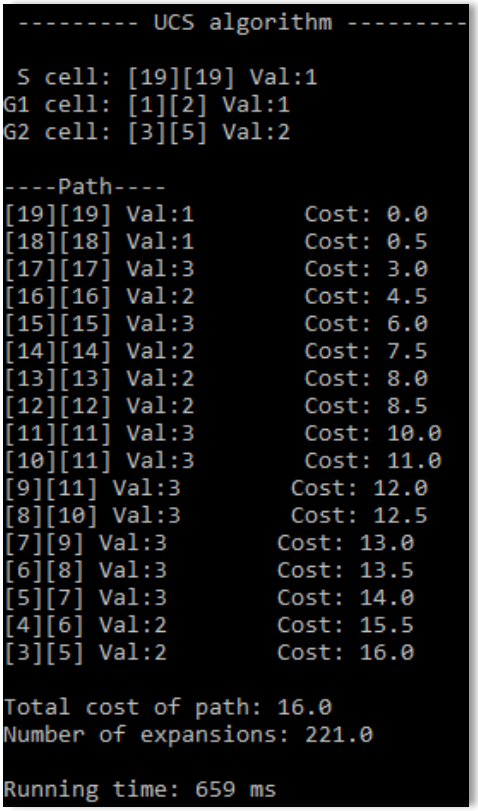
```
-------- Astar algorithm --------

 S cell: [1][1] Val:4
G1 cell: [6][5] Val:4
G2 cell: [7][6] Val:3

----Path----
[1][1] Val:4        Cost: 0.0
[2][0] Val:2        Cost: 2.5
[3][0] Val:2        Cost: 3.5
[4][1] Val:2        Cost: 4.0
[4][2] Val:3        Cost: 6.0
[5][3] Val:2        Cost: 7.5
[5][4] Val:3        Cost: 9.5
[6][5] Val:4        Cost: 11.0

Total cost of path: 11.0
Number of expansions: 21.0

Running time: 13 ms
```

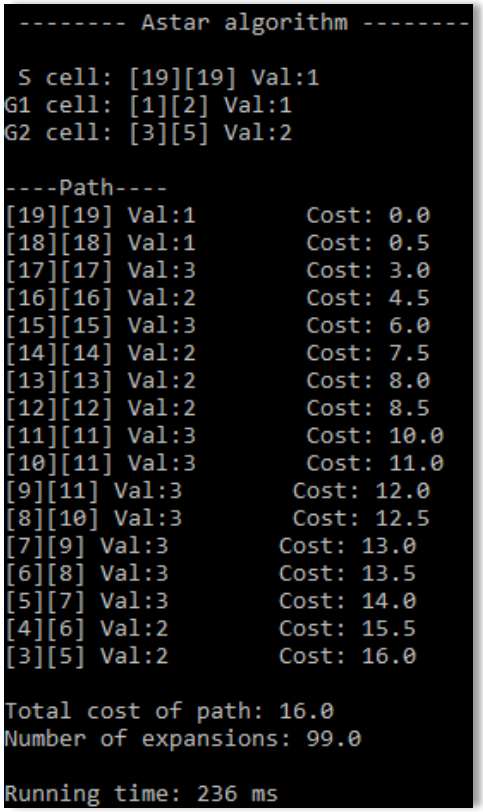
Γίνεται κατανοητό ότι όπως και στα προηγούμενα πειράματα οι δύο μέθοδοι βρίσκουν λύση η οποία είναι η βέλτιστη. Επιπλέον λόγω του αυξημένου μεγέθους του λαβύρινθου το μήκος του τελικού μονοπατιού καθώς και ο αριθμός του συνολικού κόστους και των επεκτάσεων έχει αυξηθεί και στις δύο αναζητήσεις. Τέλος η αναζήτηση A* παραμένει πιο γρήγορη και με λιγότερες επεκτάσεις από την αναζήτηση ομοιόμορφου κόστους (UCS).

Πείραμα πέμπτο: N = 20 , p = 0.1

Ο λαβύρινθος σε αυτό το πείραμα παραλείπεται διότι δεν είναι αρκετά ευκρινής λόγω του αρκετά μεγάλου μεγέθους του. Το αρχικό καθώς και τα δύο τελικά κελιά που επιλέγονται από τον χρήστη δεν περιέχουν εμπόδια.

Αποτελέσματα αλγορίθμων:



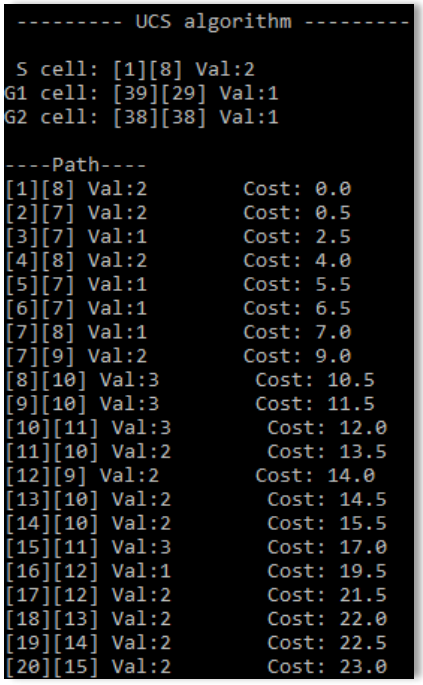


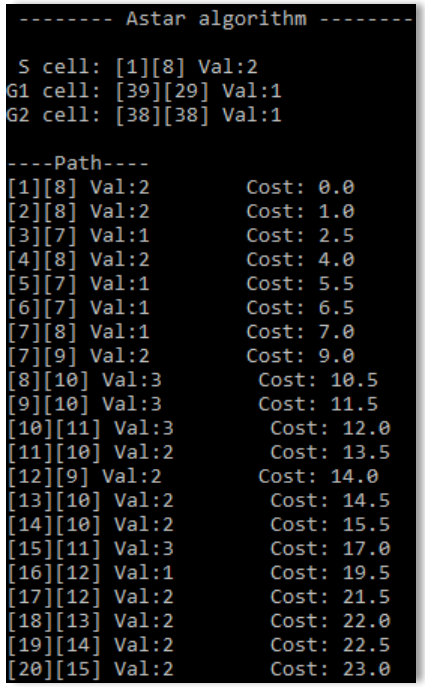
Γίνεται κατανοητό ότι όπως και στα προηγούμενα πειράματα οι δύο μέθοδοι βρίσκουν λύση η οποία είναι η βέλτιστη. Επιπλέον λόγω του μεγάλου μεγέθους του λαβύρινθου το μήκος του τελικού μονοπατιού καθώς και ο αριθμός του συνολικού κόστους και των επεκτάσεων είναι αρκετά αυξημένος και στις δύο αναζητήσεις. Η αναζήτηση A* παραμένει πιο γρήγορη και με λιγότερες επεκτάσεις από την αναζήτηση ομοιόμορφου κόστους (UCS). Βέβαια η διαφορά στον χρόνο και στον αριθμό των επεκτάσεων είναι αρκετά μεγάλη.

Πείραμα έκτο: N = 40 , p = 0.2

Ο λαβύρινθος όπως και στο προηγούμενο πείραμα παραλείπεται διότι δεν είναι αρκετά ευκρινής λόγω του αρκετά μεγάλου μεγέθους του. Το αρχικό καθώς και τα δύο τελικά κελιά που επιλέγονται από τον χρήστη δεν περιέχουν εμπόδια.

Αποτελέσματα αλγορίθμων:





Γίνεται κατανοητό ότι όπως και στα προηγούμενα πειράματα οι δύο μέθοδοι βρίσκουν λύση η οποία είναι η βέλτιστη. Επιπλέον λόγω του μεγάλου μεγέθους του λαβύρινθου το μήκος του τελικού μονοπατιού καθώς και ο αριθμός του συνολικού κόστους και των επεκτάσεων είναι αρκετά μεγάλος και στις δύο αναζητήσεις. Η αναζήτηση A* παραμένει πιο γρήγορη και με λιγότερες επεκτάσεις από την αναζήτηση ομοιόμορφου κόστους (UCS). Βέβαια η διαφορά στον χρόνο και στον αριθμό των επεκτάσεων είναι αρκετά μεγαλύτερη από όλα τα προηγούμενα πειράματα.