BBM405

Fundamentals of Artificial Intelligence

HW1

Name – Surname: Ozar Ömer Uncu

Student Id: 21727815

Programming Language: Java

Introduction

Aim of this experiment is learning some search algorithms work process. Also compare different algorithms behaviors at random generated maze.

At this experiment we will see:

How can we create maze with search algorithm?

Which algorithm can efficiently find path from start destination to goal destination?

What is the length of chosen path from selected search algorithm?

How much time passed by algorithm for complete maze?

How many nodes extended by algorithm for solution at maze?

1st Part generate maze with use randomized DFS

In this part I took given example of randomized depth first search pseudocode to java. You can see implementation at Maze.java class as **randomizedDFS(Vertex V,Maze a)**. I need to implement some functions for complete this function like **chooseUnvisitedNeighbor.** It basically looks coordinates around our current vertex for move. It randomly chooses another vertex with Math. Random function. I need to connect chosen vertexes for maze. So I connect them at **connectCells(Vertex cur, Vertex next, Maze a)**. I deploy connections of vertexes at Boolean 2d Array as adjacency matrix later I need to see connection between graph for add them to vertex neighbor's arrays for see next vertexes for search algorithms. From that I can deploy neighbors of vertices and walk around at maze and discover paths.

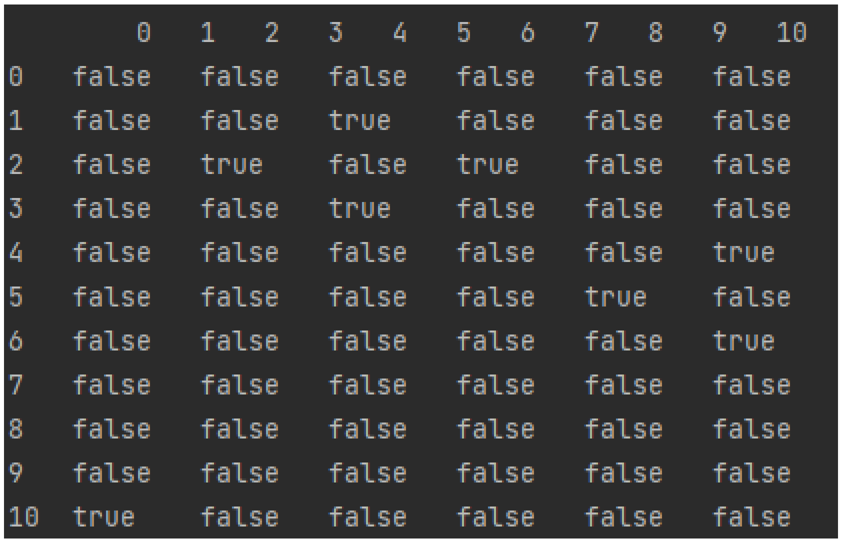
I explain how I deploy my maze to program. So, lets talk about how our algorithm process.

Algorithm 1: Randomzied Depth-First-Search

```
function createMaze()
    $startVertex \leftarrow Vertex(0,0)$
    randomizedDFS(startVertex)
end
function randomizedDFS(vertex)
    markVisited($vertex$)
    $nextVertex \leftarrow randomUnvisitedNeighbour(vertex)$
    while $nextVertex$ != $null$ do
        connectCells(vertex, nextVertex);
        randomizedDFS($nextVertex$)
        $nextVertex \leftarrow randomUnvisitedNeighbour(vertex)$
    end
    return
end
```

From pseudocode of generator our program selects start vertex as 0 point by coordinates 0,0. It pass start vertex to the randomizedDfs method. First it mark our starter vertex. Then it needs next vertex which will be connect to the our first Vertex. It will be return by chooseUnvisitedNeighbor method. At this method we check (x-1 y), (x+1, y), (x, y-1), (x, y+1) vertices by their coordinates. If coordinates will be x < 0 or y < 0 we have limits. We cannot choose this coordinated vertex for our path. We deploy next vertices to temp array than we will choose randomly from it. Get back to our randomizedDfs method we start loop to nextVertex has null. We need to vertices for maze paths. So go to connectCells at connectCells we fill adjacency matrix for see which vertex is connected to each other.

Like:

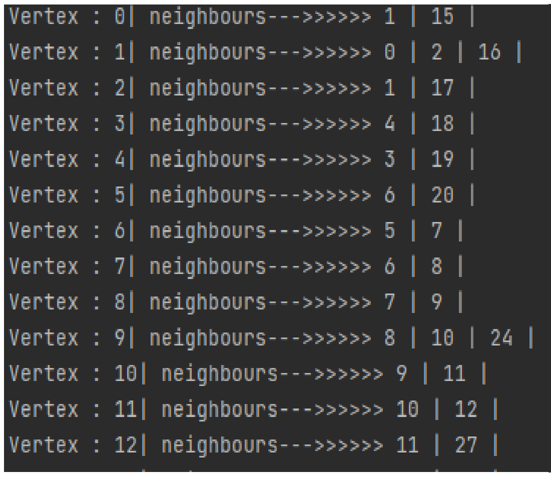


```
        0   1   2   3   4   5   6   7   8   9   10
0   false   false   false   false   false   false
1   false   false   true    false   false   false
2   false   true    false   true    false   false
3   false   false   true    false   false   false
4   false   false   false   false   false   true
5   false   false   false   false   true    false
6   false   false   false   false   false   true
7   false   false   false   false   false   false
8   false   false   false   false   false   false
9   false   false   false   false   false   false
10  true    false   false   false   false   false
```

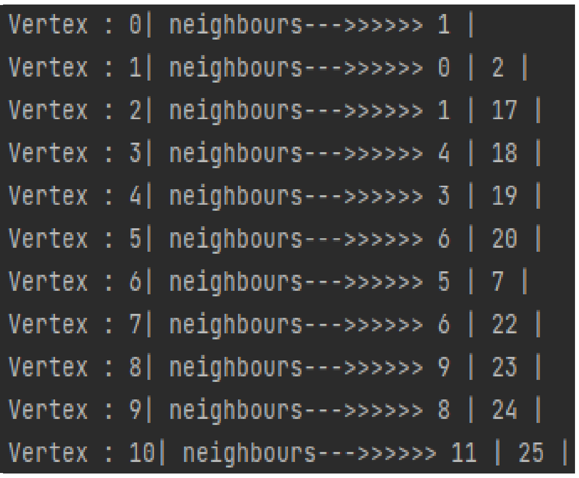
Then our method call itself again and our process begin again but take current vertex as new randomly selected vertex. After we choose again next element randomly for backtracking of recursive function. That is all about us generate maze algorithm.

Next, I need a structure for algorithms. I fill vertices neighbors array by this adjacency matrix. So, I created our structure very well.

Example:



```
Vertex : 0| neighbours--->>>>>> 1 | 15 |
Vertex : 1| neighbours--->>>>>> 0 | 2 | 16 |
Vertex : 2| neighbours--->>>>>> 1 | 17 |
Vertex : 3| neighbours--->>>>>> 4 | 18 |
Vertex : 4| neighbours--->>>>>> 3 | 19 |
Vertex : 5| neighbours--->>>>>> 6 | 20 |
Vertex : 6| neighbours--->>>>>> 5 | 7 |
Vertex : 7| neighbours--->>>>>> 6 | 8 |
Vertex : 8| neighbours--->>>>>> 7 | 9 |
Vertex : 9| neighbours--->>>>>> 8 | 10 | 24 |
Vertex : 10| neighbours--->>>>>> 9 | 11 |
Vertex : 11| neighbours--->>>>>> 10 | 12 |
Vertex : 12| neighbours--->>>>>> 11 | 27 |
```

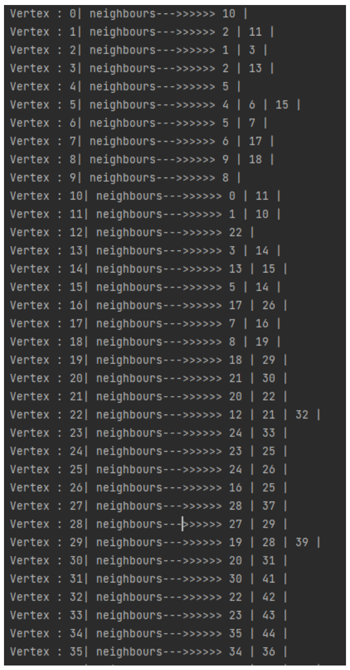


```
Vertex : 0| neighbours--->>>>>> 1 |
Vertex : 1| neighbours--->>>>>> 0 | 2 |
Vertex : 2| neighbours--->>>>>> 1 | 17 |
Vertex : 3| neighbours--->>>>>> 4 | 18 |
Vertex : 4| neighbours--->>>>>> 3 | 19 |
Vertex : 5| neighbours--->>>>>> 6 | 20 |
Vertex : 6| neighbours--->>>>>> 5 | 7 |
Vertex : 7| neighbours--->>>>>> 6 | 22 |
Vertex : 8| neighbours--->>>>>> 9 | 23 |
Vertex : 9| neighbours--->>>>>> 8 | 24 |
Vertex : 10| neighbours--->>>>>> 11 | 25 |
```

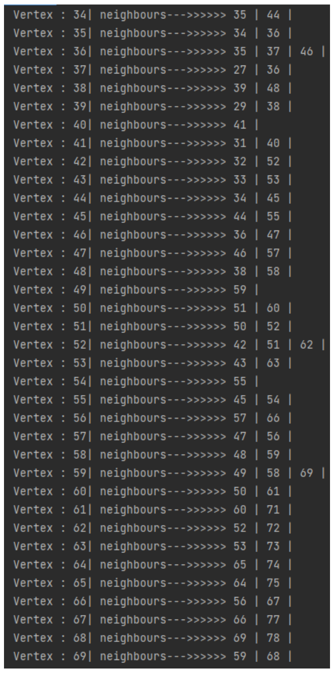
I want to print maze with javaFx but I could not complete visualization.

For objectives, I give start point as 0,0 and call our class with same size two times. From the memory heap I could not create 1000x1000 adj matrix or 815x815. I could create biggest maze as 152x152.

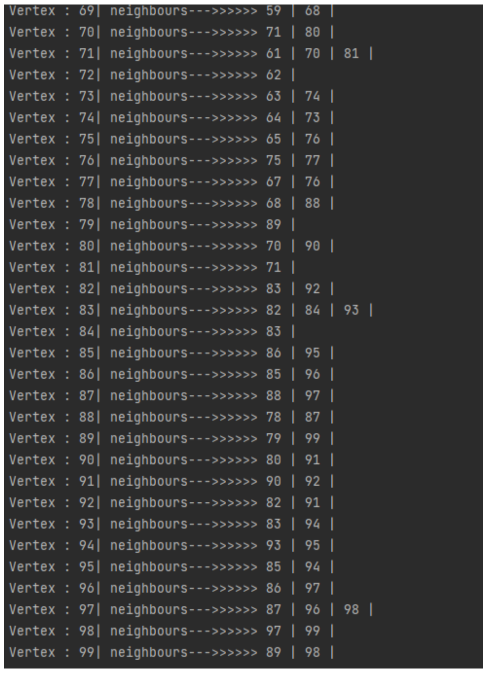
Also, I could not visualize my maze but from 10x10 example it will be look like below.



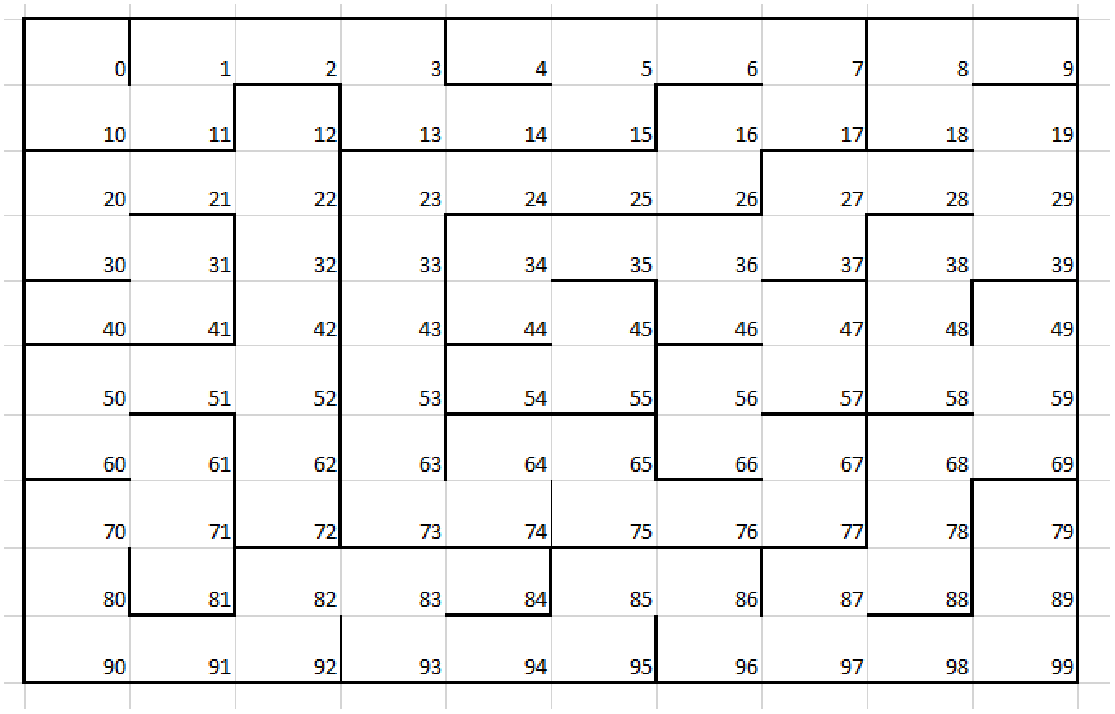
```
Vertex : 0| neighbours--->>>>>> 10 |
Vertex : 1| neighbours--->>>>>> 2 | 11 |
Vertex : 2| neighbours--->>>>>> 1 | 3 |
Vertex : 3| neighbours--->>>>>> 2 | 13 |
Vertex : 4| neighbours--->>>>>> 5 |
Vertex : 5| neighbours--->>>>>> 4 | 6 | 15 |
Vertex : 6| neighbours--->>>>>> 5 | 7 |
Vertex : 7| neighbours--->>>>>> 6 | 17 |
Vertex : 8| neighbours--->>>>>> 9 | 18 |
Vertex : 9| neighbours--->>>>>> 8 |
Vertex : 10| neighbours--->>>>>> 0 | 11 |
Vertex : 11| neighbours--->>>>>> 1 | 10 |
Vertex : 12| neighbours--->>>>>> 22 |
Vertex : 13| neighbours--->>>>>> 3 | 14 |
Vertex : 14| neighbours--->>>>>> 13 | 15 |
Vertex : 15| neighbours--->>>>>> 5 | 14 |
Vertex : 16| neighbours--->>>>>> 17 | 26 |
Vertex : 17| neighbours--->>>>>> 7 | 16 |
Vertex : 18| neighbours--->>>>>> 8 | 19 |
Vertex : 19| neighbours--->>>>>> 18 | 29 |
Vertex : 20| neighbours--->>>>>> 21 | 30 |
Vertex : 21| neighbours--->>>>>> 20 | 22 |
Vertex : 22| neighbours--->>>>>> 12 | 21 | 32 |
Vertex : 23| neighbours--->>>>>> 24 | 33 |
Vertex : 24| neighbours--->>>>>> 23 | 25 |
Vertex : 25| neighbours--->>>>>> 24 | 26 |
Vertex : 26| neighbours--->>>>>> 16 | 25 |
Vertex : 27| neighbours--->>>>>> 28 | 37 |
Vertex : 28| neighbours--->>>>>> 27 | 29 |
Vertex : 29| neighbours--->>>>>> 19 | 28 | 39 |
Vertex : 30| neighbours--->>>>>> 20 | 31 |
Vertex : 31| neighbours--->>>>>> 30 | 41 |
Vertex : 32| neighbours--->>>>>> 22 | 42 |
Vertex : 33| neighbours--->>>>>> 23 | 43 |
Vertex : 34| neighbours--->>>>>> 35 | 44 |
Vertex : 35| neighbours--->>>>>> 34 | 36 |
```



```
Vertex : 34| neighbours--->>>>>> 35 | 44 |
Vertex : 35| neighbours--->>>>>> 34 | 36 |
Vertex : 36| neighbours--->>>>>> 35 | 37 | 46 |
Vertex : 37| neighbours--->>>>>> 27 | 36 |
Vertex : 38| neighbours--->>>>>> 39 | 48 |
Vertex : 39| neighbours--->>>>>> 29 | 38 |
Vertex : 40| neighbours--->>>>>> 41 |
Vertex : 41| neighbours--->>>>>> 31 | 40 |
Vertex : 42| neighbours--->>>>>> 32 | 52 |
Vertex : 43| neighbours--->>>>>> 33 | 53 |
Vertex : 44| neighbours--->>>>>> 34 | 45 |
Vertex : 45| neighbours--->>>>>> 44 | 55 |
Vertex : 46| neighbours--->>>>>> 36 | 47 |
Vertex : 47| neighbours--->>>>>> 46 | 57 |
Vertex : 48| neighbours--->>>>>> 38 | 58 |
Vertex : 49| neighbours--->>>>>> 59 |
Vertex : 50| neighbours--->>>>>> 51 | 60 |
Vertex : 51| neighbours--->>>>>> 50 | 52 |
Vertex : 52| neighbours--->>>>>> 42 | 51 | 62 |
Vertex : 53| neighbours--->>>>>> 43 | 63 |
Vertex : 54| neighbours--->>>>>> 55 |
Vertex : 55| neighbours--->>>>>> 45 | 54 |
Vertex : 56| neighbours--->>>>>> 57 | 66 |
Vertex : 57| neighbours--->>>>>> 47 | 56 |
Vertex : 58| neighbours--->>>>>> 48 | 59 |
Vertex : 59| neighbours--->>>>>> 49 | 58 | 69 |
Vertex : 60| neighbours--->>>>>> 50 | 61 |
Vertex : 61| neighbours--->>>>>> 60 | 71 |
Vertex : 62| neighbours--->>>>>> 52 | 72 |
Vertex : 63| neighbours--->>>>>> 53 | 73 |
Vertex : 64| neighbours--->>>>>> 65 | 74 |
Vertex : 65| neighbours--->>>>>> 64 | 75 |
Vertex : 66| neighbours--->>>>>> 56 | 67 |
Vertex : 67| neighbours--->>>>>> 66 | 77 |
Vertex : 68| neighbours--->>>>>> 69 | 78 |
Vertex : 69| neighbours--->>>>>> 59 | 68 |
```

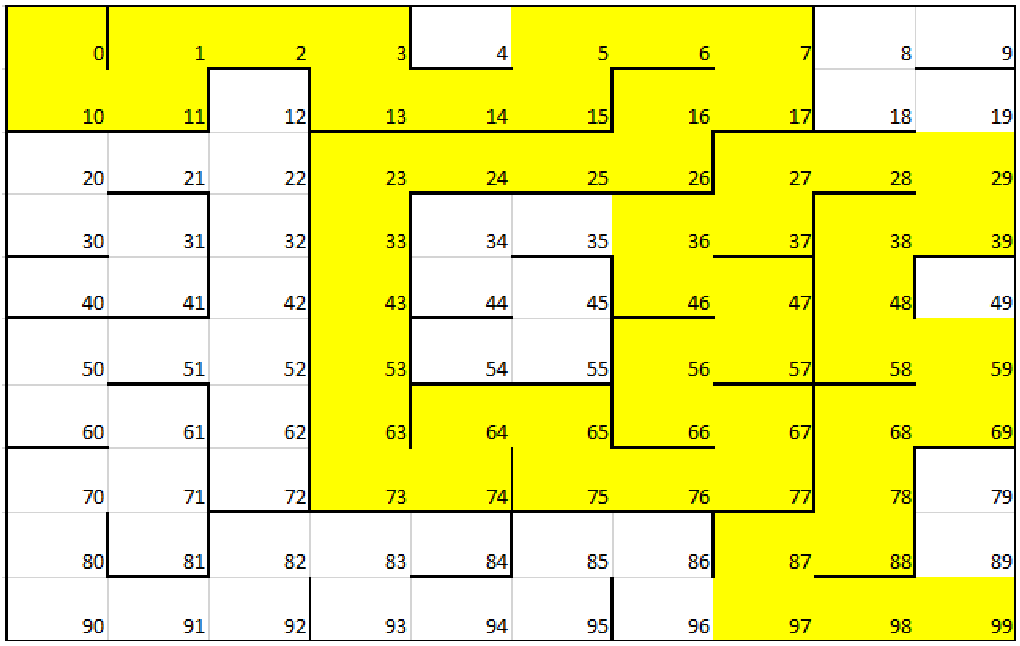


```
Vertex : 69| neighbours--->>>>>> 59 | 68 |
Vertex : 70| neighbours--->>>>>> 71 | 80 |
Vertex : 71| neighbours--->>>>>> 61 | 70 | 81 |
Vertex : 72| neighbours--->>>>>> 62 |
Vertex : 73| neighbours--->>>>>> 63 | 74 |
Vertex : 74| neighbours--->>>>>> 64 | 73 |
Vertex : 75| neighbours--->>>>>> 65 | 76 |
Vertex : 76| neighbours--->>>>>> 75 | 77 |
Vertex : 77| neighbours--->>>>>> 67 | 76 |
Vertex : 78| neighbours--->>>>>> 68 | 88 |
Vertex : 79| neighbours--->>>>>> 89 |
Vertex : 80| neighbours--->>>>>> 70 | 90 |
Vertex : 81| neighbours--->>>>>> 71 |
Vertex : 82| neighbours--->>>>>> 83 | 92 |
Vertex : 83| neighbours--->>>>>> 82 | 84 | 93 |
Vertex : 84| neighbours--->>>>>> 83 |
Vertex : 85| neighbours--->>>>>> 86 | 95 |
Vertex : 86| neighbours--->>>>>> 85 | 96 |
Vertex : 87| neighbours--->>>>>> 88 | 97 |
Vertex : 88| neighbours--->>>>>> 78 | 87 |
Vertex : 89| neighbours--->>>>>> 79 | 99 |
Vertex : 90| neighbours--->>>>>> 80 | 91 |
Vertex : 91| neighbours--->>>>>> 90 | 92 |
Vertex : 92| neighbours--->>>>>> 82 | 91 |
Vertex : 93| neighbours--->>>>>> 83 | 94 |
Vertex : 94| neighbours--->>>>>> 93 | 95 |
Vertex : 95| neighbours--->>>>>> 85 | 94 |
Vertex : 96| neighbours--->>>>>> 86 | 97 |
Vertex : 97| neighbours--->>>>>> 87 | 96 | 98 |
Vertex : 98| neighbours--->>>>>> 97 | 99 |
Vertex : 99| neighbours--->>>>>> 89 | 98 |
```



I also deploy walls at vertices as up, down, left and right as Boolean.

At our maze we have just one path 0 to 99. That is result of perfect maze. Perfect maze has one path to goal vertex, so our algorithm generate maze correctly.



Iterative Deepening Search

Iterative deep search has similarities with BFS and DFS algorithms. It uses less memory. And travel same as DFS but it has cumulative order in which nodes are first visited same as BFS algorithm.

I implement Iterative deepening search algorithm at IDS.java class. I could not implement properly but it processes work correct. For the iterative deepening search, we call IDS method from main pass start vertex and goal vertex. We loop over map with depth. It passes process to DLS method. At DLS method we if it accesses to goal or depth 0(one node path) returns current node. It will be last vertex at path. But other situations as depth > 0 we need to traverse over neighbors of the map. We call DLS recursively with child vertex and decrease depth too. And process this recursive call with new passed values. For the backtracking of recursive method, we check can we complete path or cannot find result.

At my implementation I start IDS algorithm. I observe it takes lots of time to complete path with correct path.

I could not observe other algorithms.

References

1- Iterative deepening depth-first search - Wikipedia
2- Maze generation - Rosetta Code
3- PathFinder: The Amazing Maze Algorithm Demonstrator! (emmilco.github.io)
4- Algorithm to Generate a Maze | Baeldung on Computer Science
5- Minimum Spanning vs Shortest Path Trees | Baeldung on Computer Science
6- Kruskal's vs Prim's Algorithm | Baeldung on Computer Science
7- Maze generation algorithm - Wikipedia
8- Drawing a border around a JavaFX Text node - Stack Overflow
9- Java Program to Implement Iterative Deepening - Sanfoundry
10- Calculating the difference between two Java date instances - Stack Overflow
11- java - Uniform Cost Search Implementation - Stack Overflow
12- A* Algorithm (With Java Example) | HappyCoders.eu
13- java.lang.OutOfMemoryError: Java heap space while initialising an array - Stack Overflow
14- Java Data Types (w3schools.com)
15- Iterative Deepening Search(IDS) or Iterative Deepening Depth First Search(IDDFS) - GeeksforGeeks
16- java - Two dimensional array list - Stack Overflow
17- Sabit Maliyet Araması (Uniformed Cost Search , UCS) - YouTube
18- java.lang.OutOfMemoryError: Java heap space while initialising an array - Stack Overflow
19- Multi Dimensional ArrayList in Java | Baeldung
20- How to Use Borders (The Java™ Tutorials > Creating a GUI With JFC/Swing > Using Swing Components) (oracle.com)