Data-Structure I_CS122

MAZE SOLVER

Name :Mohamed Mashaal Mohamed Ali Alagha

Id : 60

The implementation :

PDF Containing the Implementation :

https://drive.google.com/file/d/0B0Wq-MQ9sQITUUF1R193Sk0yV2c/view?usp=sharing

BitBucket Repo :

https://bitbucket.org/MohamedMashaal/data-structure/src/4c1ec41af7e05b3da5dda7f34c7d1472c36d7d8e/src/eg/edu/alexu/csd/datastructure/maze/

Data-Structure and Algorithms Used :

- 2D Array (-TheM-) :

It was used in the first step of the both methods for scanning the Grid representing the maze from the File provided .

- 2D Array (-visited-) :

It was used in the both methods for checking through the procedure if the cell has been already visited or not .

- 1D Array of Object (-temp-)

It was used in the both methods to provide a way of retrieving the path , as i had 2 elements , the first pointing the coordinate of the current point and the second points to the coordinate of the previous point .

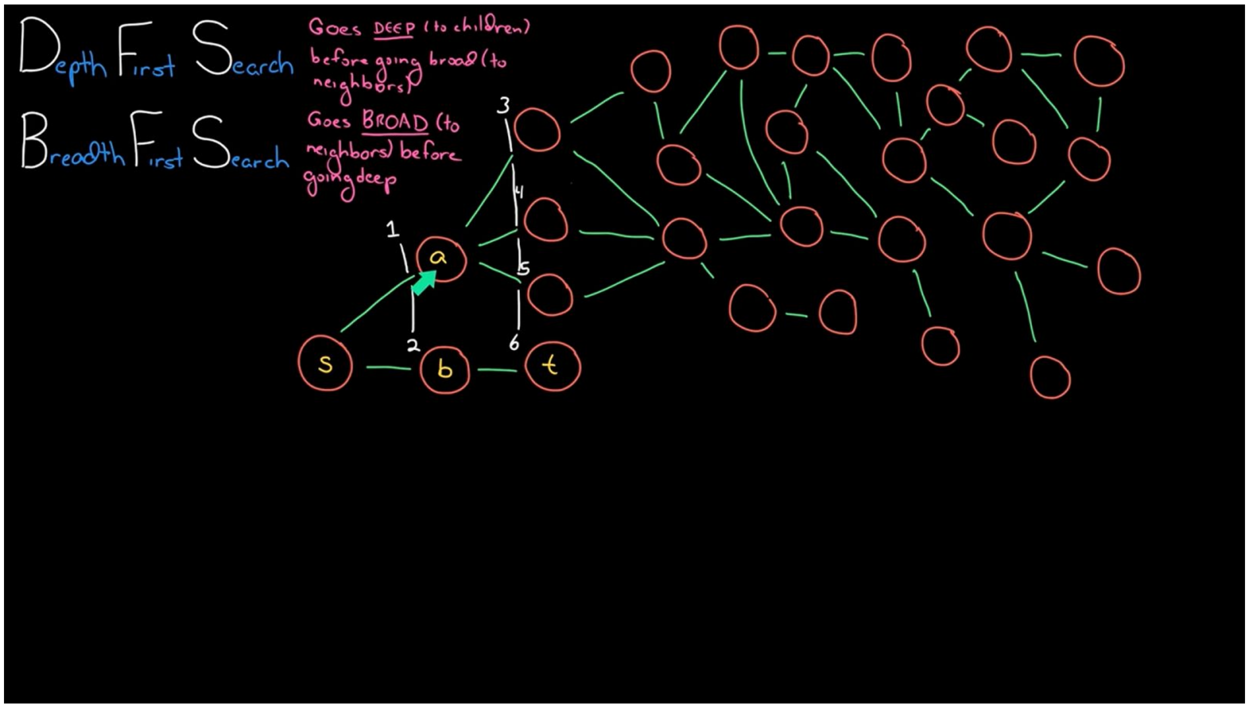
- Queue :

It was used in the implementation of the BFS as it provides the required functionality needed as the BFS Depends on going level by level unlike the DFS which keeps searching till the end of each node

- Stack :

It was used in the implementation of the DFS as it provides the same functionality as recursion does which is required for the DFS that keeps searching each node till it finds the required node or reaches a dead end , unlike the BFS Which goes level by level which is faster in case of getting the Shortest path .

Comparison :


Depth First Search
Goes DEEP (to children) before going broad (to neighbors)
Breadth First Search
Goes BROAD (to neighbors) before going deep
1
2
3
4
5
6
a
s
b
t

The image up Just summarize the whole comparison .

As for DFS it works by searching each node till it reaches the required node or reaches a dead end in a recursive manner which could lead to a waste of time as shown in the example above , and that behavior can be implemented using a recursion including Stacks .

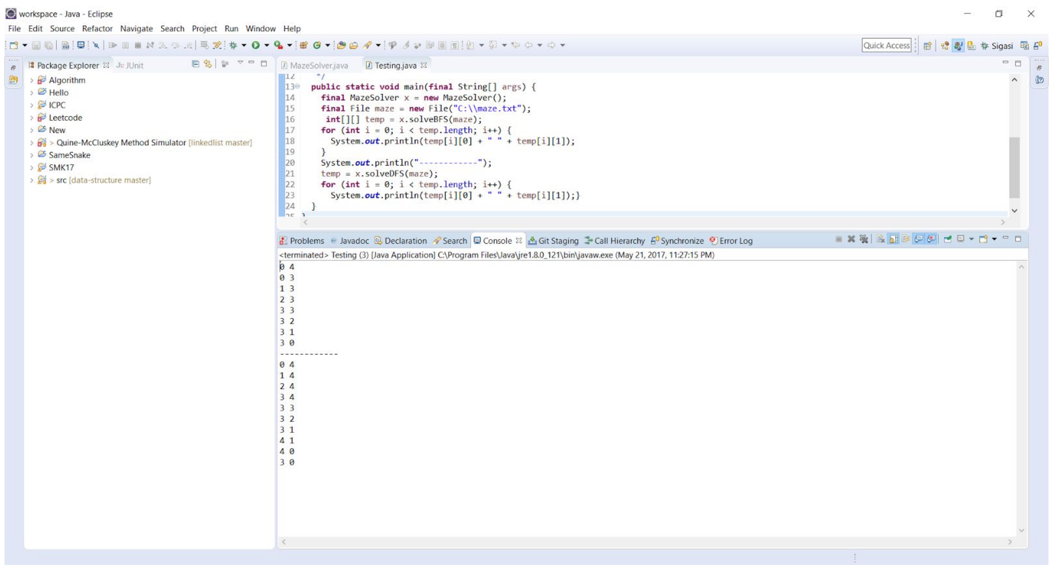
Unlike that BFS works level by level as it searches each level nodes which naturally leads to finding the shortest path as well as eliminating the time wasted in cases like the one mentioned above , and that behavior can be implemented using a queue .

Test Samples :

Sample 1 :

```
# # . .  S
.  . # .  .
. # # .   .
E .  . .  .
.  . # # #
```

The Solution , a difference is noticed in The DFS Compared to BFS .



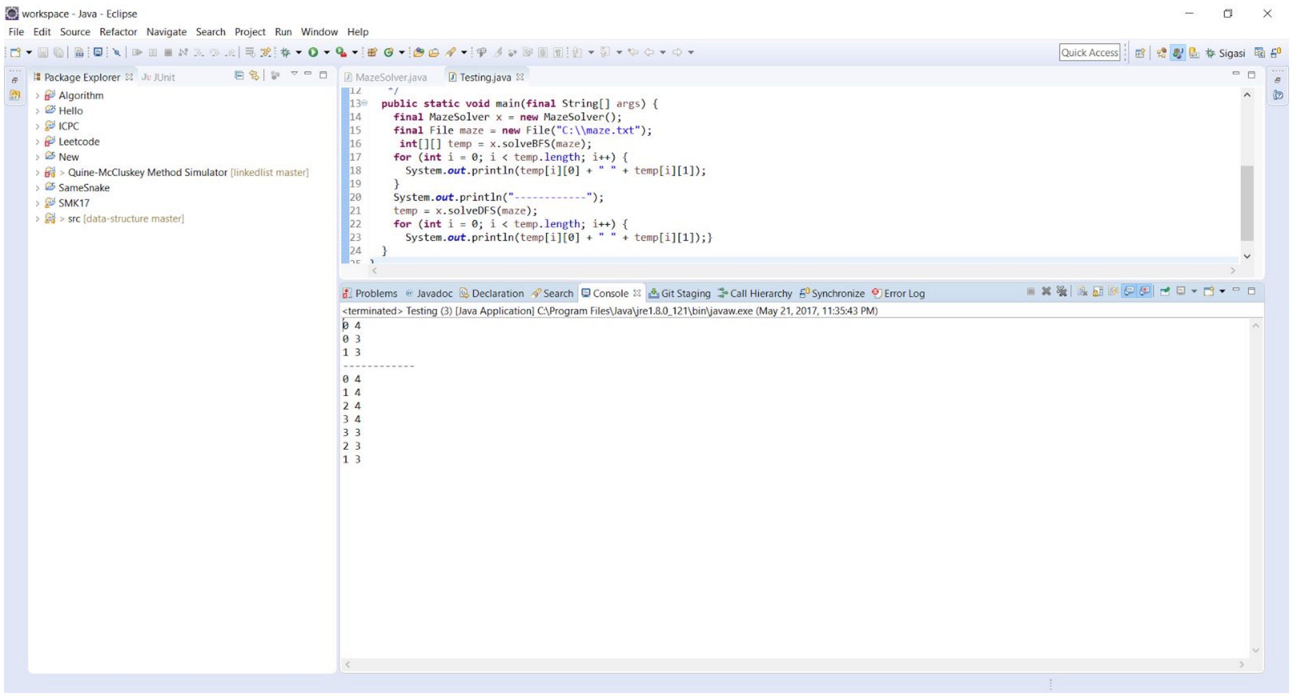
```java
  public static void main(final String[] args) {
    final MazeSolver x = new MazeSolver();
    final File maze = new File("C:\\maze.txt");
     int[][] temp = x.solveBFS(maze);
    for (int i = 0; i < temp.length; i++) {
      System.out.println(temp[i][0] + " " + temp[i][1]);
    }
    System.out.println("------------");
    temp = x.solveDFS(maze);
    for (int i = 0; i < temp.length; i++) {
      System.out.println(temp[i][0] + " " + temp[i][1]);}
  }
```

```
<terminated> Testing (3) [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (May 21, 2017, 11:27:15 PM)
0 4
0 3
1 3
2 3
3 3
3 2
3 1
3 0
------------
0 4
1 4
2 4
3 4
3 3
3 2
3 1
4 1
4 0
3 0
```

Sample 2 :

```
. . . . S
. . . E .
. # # . .
. . # . .
. . # # #
```

The Solution , A More clear and obvious sample showing the difference between the both search methods .



workspace - Java - Eclipse

```java
  */
public static void main(final String[] args) {
  final MazeSolver x = new MazeSolver();
  final File maze = new File("C:\\maze.txt");
   int[][] temp = x.solveBFS(maze);
  for (int i = 0; i < temp.length; i++) {
    System.out.println(temp[i][0] + " " + temp[i][1]);
  }
  System.out.println("-----------");
  temp = x.solveDFS(maze);
  for (int i = 0; i < temp.length; i++) {
    System.out.println(temp[i][0] + " " + temp[i][1]);}
}
```

<terminated> Testing (3) [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (May 21, 2017, 11:35:43 PM)

```
0 4
0 3
1 3
-----------
0 4
1 4
2 4
3 4
3 3
2 3
1 3
```