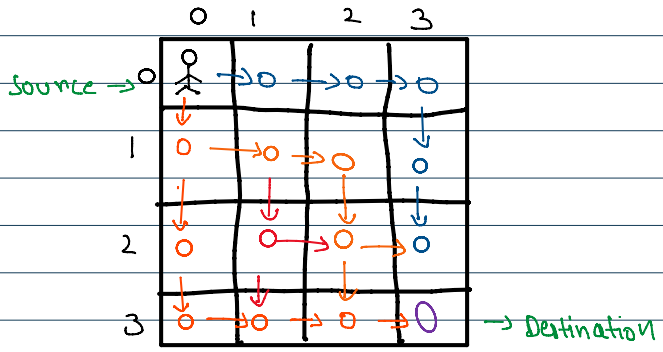
Backtracking

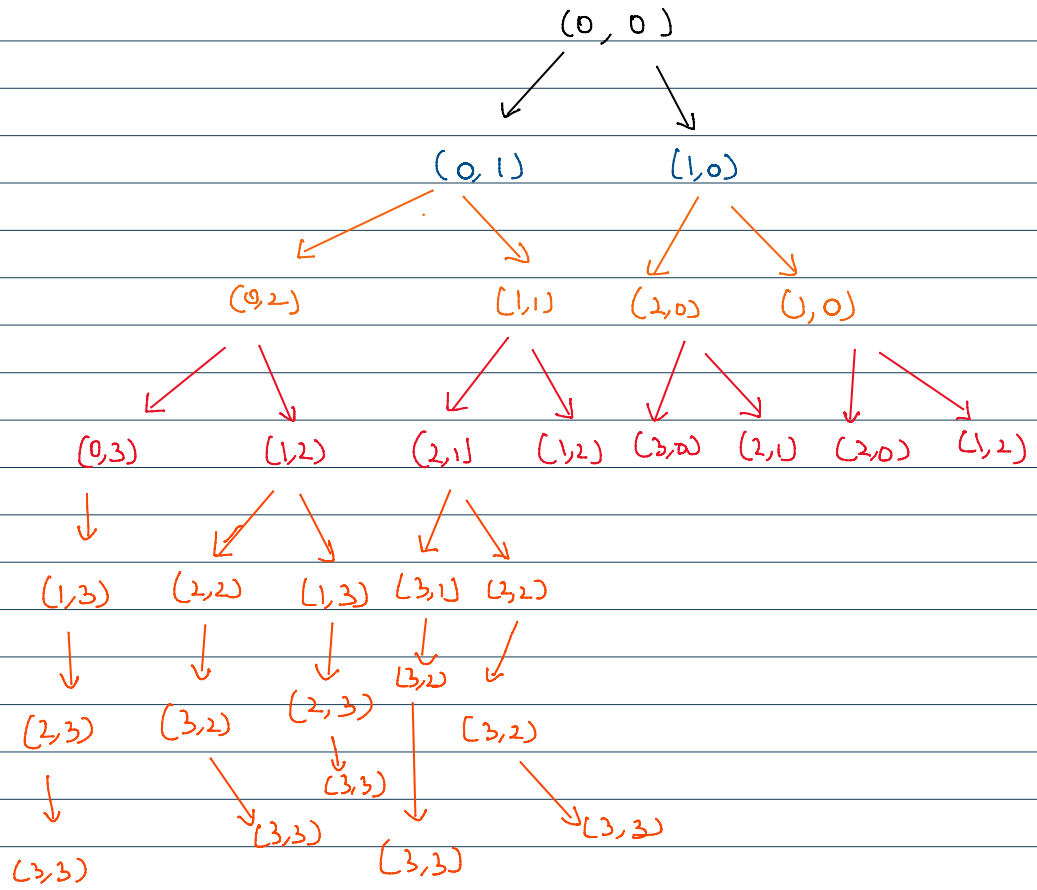
30 May 2022 19:59

Basic Maze Questions.

1.g Reach from source to destination. you can go either right or downward



⇒ Now let's make a recursive call tree for it



CPH JUDGE: RESULTS

Local: maze

Testcase 1 Failed 364ms

Input: Copy
src:- 0 0 dest:- 2 1

Expected Output: Copy

Received Output: Copy
[[0, 0], [1, 0], [2, 0], [2, 1]]
[[0, 0], [1, 0], [1, 1], [2, 1]]
[[0, 0], [0, 1], [1, 1], [2, 1]]

+ New Testcase

Set ONLINE_JUDGE

Run All + New
Stop Help Delete

Phone_Number.java diceroll.java maze.java 1 Settings

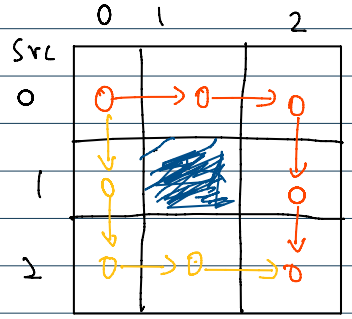
maze.java > maze > Paths(ArrayList<ArrayList<Integer>>, int[], int[], int[])

```java
15      static ArrayList<ArrayList<ArrayList<Integer>>> Paths
        (ArrayList<ArrayList<Integer>> list,  int[] dest,
16              int[] index, int[] size) {
17          ArrayList<ArrayList<ArrayList<Integer>>> ans = new ArrayList<>();
18                  // If index goes out of boudary return empty list
19                  if (index[0] >= size[0] || index[1] >= size[1]) {
20                      return ans;
21                  }
22              ArrayList<ArrayList<Integer>> temp = new ArrayList<>();
23              temp.addAll(list);
24              ArrayList<Integer> idx = new ArrayList<>();
25              // add index to temp
26              idx.add(index[0]);
27              idx.add(index[1]);
28              temp.add(idx);
29              // if index == dest return the path
30              if (index[0] == dest[0] && index[1] == dest[1]) {
31                  ans.add(temp);
32                  return ans;
33              }
34              int[] right = {index[0]+1, index[1]};
35              int[] down = { index[0], index[1] + 1 };
36              // go right
37              ans.addAll(Paths(temp, dest, right, size));
38              // go down
39              ans.addAll(Paths(temp, dest, down, size));
40              return ans;
41      }
42  }
```

Run Testcases ⊗ 0 ⚠ 4 Ln 36, Col 24 Spaces: 4 UTF-8 CRLF { } Java Go Live

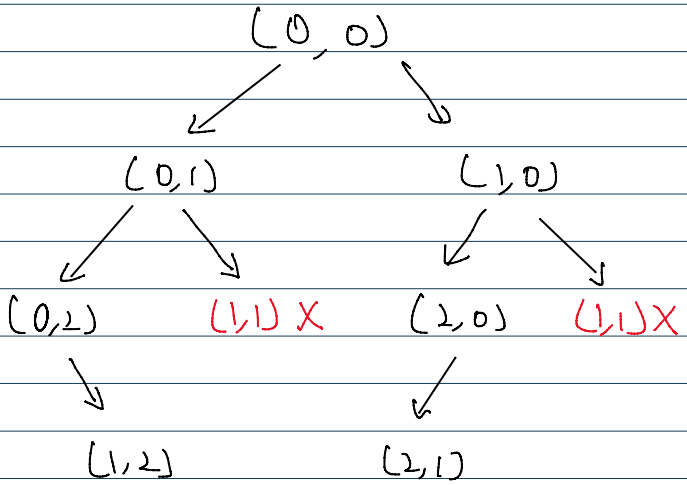
Maze with Obstacle

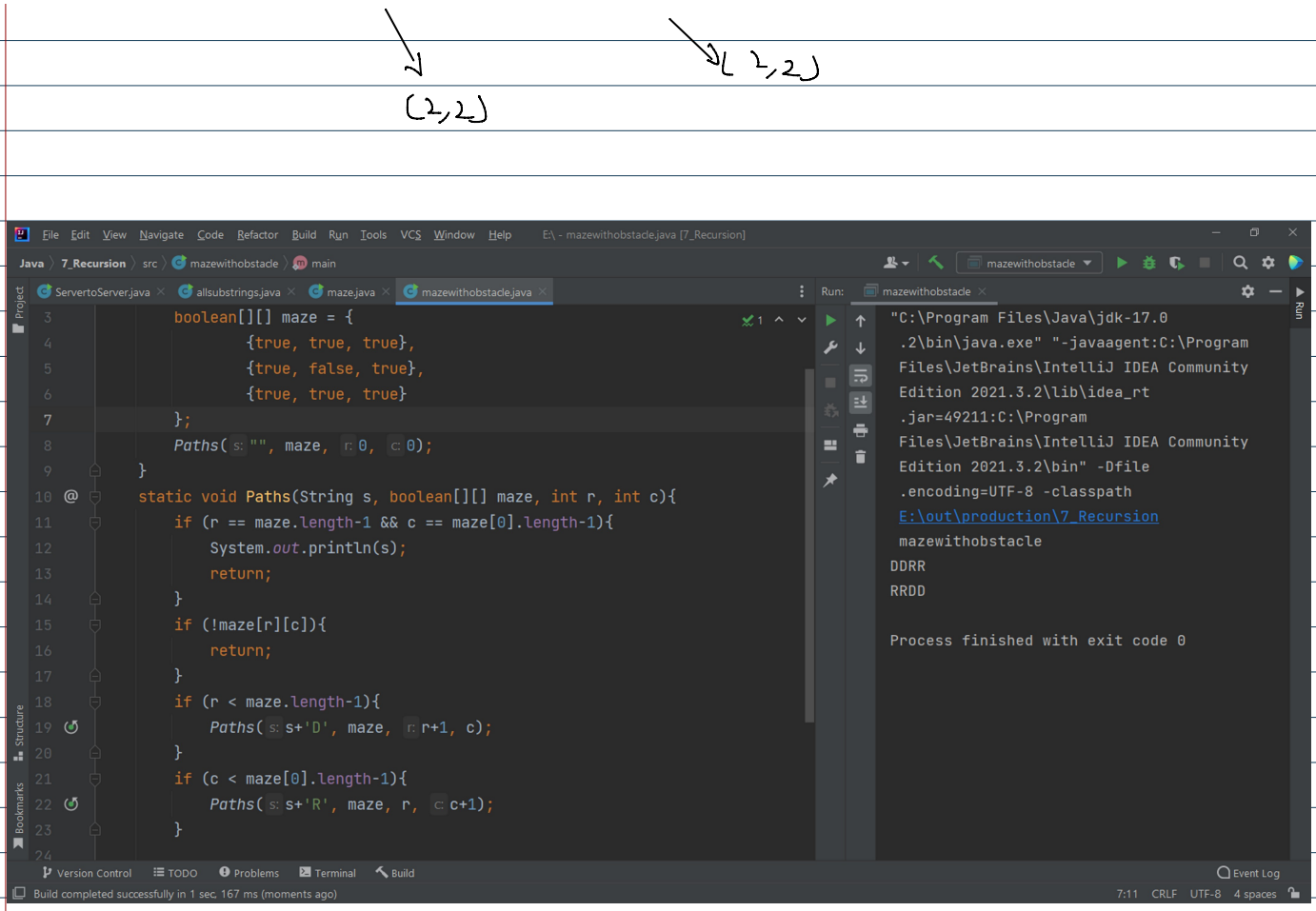
⇒ In these types of problems we have obstacles i.e we cannot go forward from this point.



⇒ Here we have a boolean matrix and false means obstacle.
mat[i][j] = false

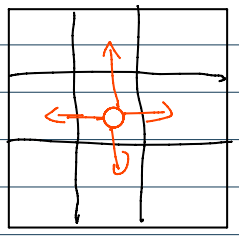
⇒ Recursive Call Tree



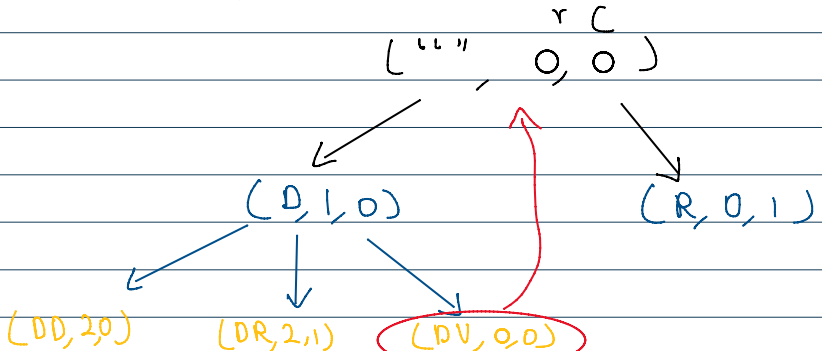


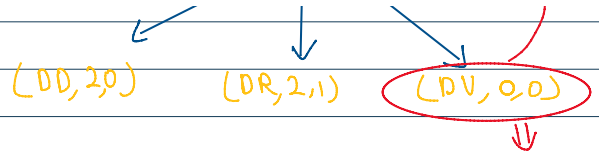
Include All Paths

⇒ Till now we were going only Down and Right Let's try to go left and up as well.



⇒ Recursive Call Tree





From here it will again go to (0,0) and it will repeat the steps endlessly.

⇒ So, How to Solve this Problem

⇓

By not allowing pointer to go that section again

⇓

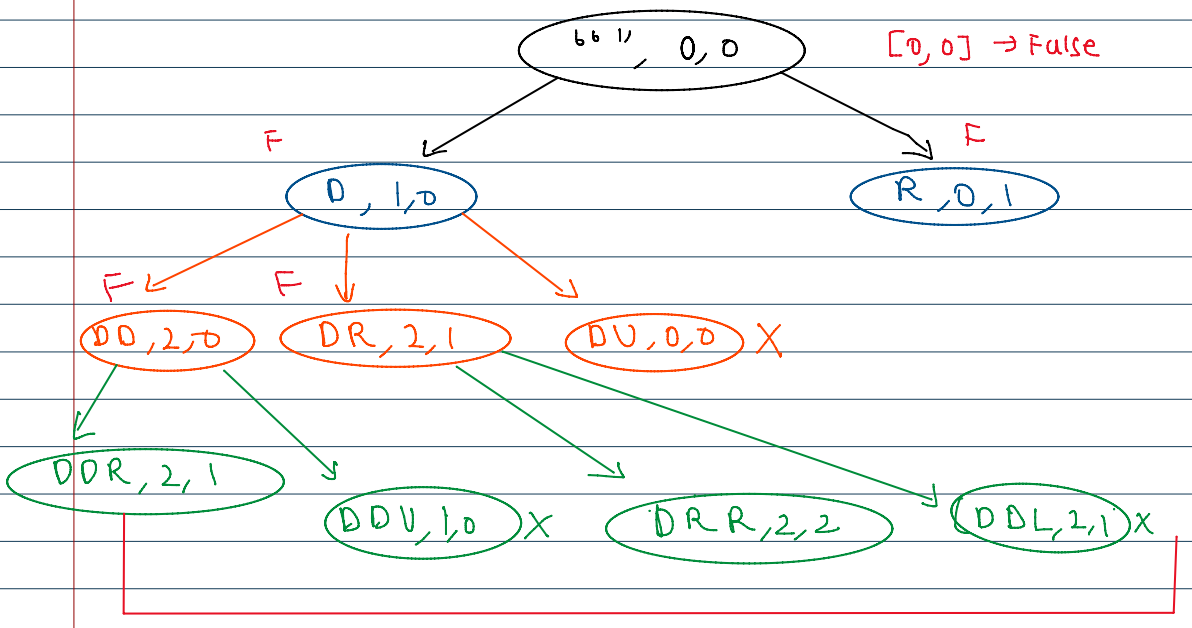
By making (0,0) as false

⇒ we Have to mark all the co-ordinates as false that are visited by us.

mark the place that are visited

F	T	T
F	T	T
F → F		T

⇒ Recursive Call Tree



⇓

But here we have a problem, when we mark all these coordinates false, we will be not able to use those sections in future recursive calls, as we have to update main matrix

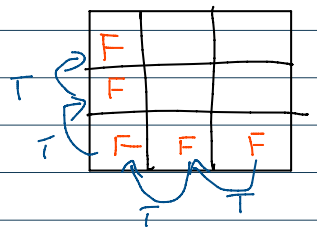
coordinates false, we will be not able to use those sections in future recursive calls, as we have to update main matrix

⇒ Now, How to Solve this problem

⇓

we can make coordinates true again, as we return from, By this these sections will be available for future recursive calls

⇓



This is called

Back tracking

File - E:\Java\7_Recursion\src\Backtracking.java

```java
public class Backtracking {
    public static void main(String[] args) {
        boolean maze[][] = {
                {true, true, true},
                {true, true, true},
                {true, true, true}
        };
        AllPaths("", maze, 0, 0);
    }

    static void AllPaths(String s, boolean[][] maze, int r, int c){
        // Base Condition if we reached the Target
        if (r == maze.length-1 && c == maze[0].length-1){
            System.out.println(s);
            return;
        }
        // If This section is already visited don't go forward
        if (!maze[r][c]){
            return;
        }
        maze[r][c] = false; // make this false as we are on this section

        if (r < maze.length-1){  // go Down
            AllPaths(s+'D', maze, r+1, c);
        }
        if (c < maze[0].length-1){  // go right
            AllPaths(s+'R', maze, r, c+1);
        }
        if (r > 0){ // go up
            AllPaths(s+'U', maze, r-1, c);
```

File - E:\Java\7_Recursion\src\Backtracking.java

```
31          }
32          if (c > 0){  // go left
33              AllPaths(s+'L', maze, r, c-1);
34          }
35          // As from this line we will start going back
36          // we will make this section true again
37          maze[r][c] = true;
38      }
39  }
40
```


File - Backtracking

```
 1 "C:\Program Files\Java\jdk-17.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\
   IntelliJ IDEA Community Edition 2021.3.2\lib\idea_rt.jar=49829:C:\Program Files\JetBrains\
   IntelliJ IDEA Community Edition 2021.3.2\bin" -Dfile.encoding=UTF-8 -classpath E:\out\
   production\7_Recursion Backtracking
 2 DDRR
 3 DDRURD
 4 DDRUURDD
 5 DRDR
 6 DRRD
 7 DRURDD
 8 RDDR
 9 RDRD
10 RDLDRR
11 RRDD
12 RRDLDR
13 RRDLLDRR
14
15 Process finished with exit code 0
16
```


⇒ Backtracking with printing the matrix of Paths

⇒ Backtracking with printing the matrix of Paths

→ Create a step variable, step = level of Recursion + 1

→ Update step in particular section.

File - E:\Java\7_Recursion\src\BacktrackingPrint.java

```java
import java.util.Arrays;

public class BacktrackingPrint {
    public static void main(String[] args) {
        boolean maze[][] = {
                {true, true, true},
                {true, true, true},
                {true, true, true}
        };
        int[][] path = new int[maze.length][maze[0].length];
        AllPaths("", maze, 0, 0, path, 0);
    }

    static void AllPaths(String s, boolean[][] maze, int r, int c, int[][] path, int step){
        // Base Condition if we reached the Target
        if (r == maze.length-1 && c == maze[0].length-1){
            path[r][c] = step +1;
            for(int[] itm : path){
                System.out.println(Arrays.toString(itm));
            }
            System.out.println(s);
            System.out.println();
            return;
        }
        // If This section is already visited don't go forward
        if (!maze[r][c]){
            return;
        }
        maze[r][c] = false; // make this false as we are on this section
        // step = level of recursion + 1
```

File - E:\Java\7_Recursion\src\BacktrackingPrint.java

```
31          path[r][c] = step +1;
32          if (r < maze.length-1){  // go Down
33              AllPaths(s+'D', maze, r+1, c, path, step+1);
34          }
35          if (c < maze[0].length-1){  // go right
36              AllPaths(s+'R', maze, r, c+1, path, step+1);
37          }
38          if (r > 0){ // go up
39              AllPaths(s+'U', maze, r-1, c, path, step+1);
40          }
41          if (c > 0){  // go left
42              AllPaths(s+'L', maze, r, c-1, path, step+1);
43          }
44          // As from this line we will start going back
45          // we will make this section true again
46          maze[r][c] = true;
47          // make step 0 as we Backtrack
48          path[r][c] = 0;
49      }
50  }
51
```

File - BacktrackingPrint

```
 1 "C:\Program Files\Java\jdk-17.0.2\bin\java.exe" "-javaagent:C:\
   Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.3.2
   \lib\idea_rt.jar=49978:C:\Program Files\JetBrains\IntelliJ IDEA
   Community Edition 2021.3.2\bin" -Dfile.encoding=UTF-8 -classpath
    E:\out\production\7_Recursion BacktrackingPrint
 2 [1, 0, 0]
 3 [2, 0, 0]
 4 [3, 4, 5]
 5 DDRR
 6
 7 [1, 0, 0]
 8 [2, 5, 6]
 9 [3, 4, 7]
10 DDRURD
11
12 [1, 6, 7]
13 [2, 5, 8]
14 [3, 4, 9]
15 DDRUURDD
16
17 [1, 0, 0]
18 [2, 3, 0]
19 [0, 4, 5]
20 DRDR
21
22 [1, 0, 0]
23 [2, 3, 4]
24 [0, 0, 5]
25 DRRD
26
27 [1, 4, 5]
28 [2, 3, 6]
29 [0, 0, 7]
30 DRURDD
31
32 [1, 2, 0]
33 [0, 3, 0]
34 [0, 4, 5]
35 RDDR
36
37 [1, 2, 0]
38 [0, 3, 4]
39 [0, 0, 5]
```

File - BacktrackingPrint

```
RDRD

[1, 2, 0]
[4, 3, 0]
[5, 6, 7]
RDLDRR

[1, 2, 3]
[0, 0, 4]
[0, 0, 5]
RRDD

[1, 2, 3]
[0, 5, 4]
[0, 6, 7]
RRDLDR

[1, 2, 3]
[6, 5, 4]
[7, 8, 9]
RRDLLDRR


Process finished with exit code 0

```