Name: Sohayla Mohammed Abu El-Fotouh Omar

S.N: 32

Assignment: Maze Solve

Additional Classes :

Point {

X coordinate.

Y coordinate

}

Relation {

Point location -> of the current cell

Point Parent -> the parent of the cell / == null if it's the start cell

}/BFS

Data structure :

DoublyLinkedList :

Visited -> contains all cells that already have been visited

neighbours()->n -> all the children of a cell

Stack :

path ->contains the parents cells that we visit during the solving

temp->contains all the children that will be visited during the solving/DFS

findRoute()->all-> store the shortest path from 'S' to 'E'.

QueueLinked:

temp->contains all the children that will be visited during the solving /BFS

Algorithms :

DFS: <clear path>

Point tmp = 'S'

temp.push(tmp);

While temp is not empty

tmp=temp.pop();

If path is not empty

While !checkparent(tmp , path.peek)

path.pop

End while

End if

```
visited.add(tmp)
if(tmp = 'E')
        path.push('E')
        Break;
End if
DLL n = neighbours(tmp,maze)
Count =0
Loop : i=0; i<n.size;
        Point t= n.get(i)
        if(!contains(visited,t) and not wall)
                temp.push(t);
                Count++
        End if
Endloop
if(count>0)
        path,push(tmp);
        End if
        End while
        ->check if temp is empty and the last cell in path is not 'E'
                If so return null as there's no path
        ->return path
```

BFS:

```
<clear path>
Relation tmp = set(location->'S', parent ->null)
temp.enqueue(tmp);
While temp is not empty
        tmp=temp.dequeue();
        visited.add(tmp.getLoc())
        if(tmp.location = 'E')
                path.push('E')
                Break;
        End if
        DLL n = neighbours(tmp,maze)
        Count =0
        Loop : i=0; i<n.size;
```

Point t= n.get(i)

if(!contains(visited,t) and not wall)

Relation te.set(t,tmp)

temp.push(te);

Count++

End if

End loop

if(count>0)

path,push(tmp);

End if

End while

->check if temp is empty and the last cell in path is not 'E'

If so return null as there's no path

->path=findRoute(path)->find the shortest route

->return path

Assumbtions:

1.The maze is sent as a file ; {

N M

0 1 2 3 ...M

0

1

. .

N }

2.if the file is null throw null

3.if the dimensions in the file does not match the grid's dimensions throw null

4.In BFS , i implemented a class relation that stores each visited cell and its parent

---

Functions:

solveDFS -> It take a parameter File Maze

Solve the maze by using depth first algorithm

Calls functions:

readFile(File Maze)->gets the dimensions and the maze from

The file and stores it in a grid of char[][]

startAndEndTiles->finds the ‘S’ and ‘E’ cells

checkParent(point a,point b)-> check if b is a parent of a or not

neighbours(point)-> gets all the adjacent cells of the point

contains(DDL,point)->check if the point has already been visited or not

Returns the path in an [][] int array where col. 0 is x and col. 1 is Y an and each row is a point

solveBFS-> It take a parameter File Maze

Solve the maze by using depth first algorithm

Calls functions :

readFile(File Maze)->gets the dimensions and the maze from The file and stores it in a grid of char[][]

startAndEndTiles->finds the ‘S’ and ‘E’ cells

neighbours(point)-> gets all the adjacent cells of the point

contains(DDL,point)->check if the point has already been visited or not

findRoute(Stack)->find the shortest path

Returns the path in an [][] int array where col. 0 is x and col. 1 is Y an and each row is a point

---

Comparison:

They both have the same algorithms the only difference is gonna be the data structure used in the algorithms.

In dfs, we visit a cell A then one of its adj. B And then one of the adj. Cells of B ,C and so on until we reach the ‘E’ if we reached a dead end we take another child of the last parent if there’s no other childrens we remove the parent from the path and go for its parent and so on until we reach ‘E’, to do that i choosed a stack to store the to do list and the last pushed cell is gonna be my next visited and so on.

In bfs we visit cell A then all the adj cells of A (B,C,D,E) and then all of the adj. Cell of each of B,C,D&E and so on until we reach ‘E’ when we do that we trace back the parents to get the shortest path, to do that i used a queue to store the to do list and the first enqueued cell is gonna be my next visited.

Sample runs:

-> input .txt file { 5 3

.E.

...

.S.

…

#..}

->output dfs ->

```
.E.
...
.S.
...
#..
 2 1
 3 1
 4 1
 4 2
 3 2
 2 2
 1 2
 0 2
 0 1
```

->output bfs->

```
.E.
...
.S.
...
#..
 2 1
 1 1
 0 1
```

->input .txt file{ 5 5

##..S

..#..

.##..

E....

..###}

->output,dfs->

```
##..S
..#..
.##..
E....
..###
 0 4
 1 4
 2 4
 3 4
 3 3
 3 2
 3 1
 4 1
 4 0
 3 0
```

->output->bfs->

```
##..S
..#..
.##..
E....
..###
 0 4
 1 4
 2 4
 3 4
 3 3
 3 2
 3 1
 3 0
```

->input .txt file{

7 7

.....S

..#..#.

.##.##.

..#E...

..###..

.....#.

.##.##.

}

->output,dfs->

```
.....S
..#..#.
.##.##.
..#E...
..###..
.....#.
.##.##.
 0 6
 1 6
 2 6
 3 6
 4 6
 4 5
 3 5
 3 4
 3 3
```

->output,bfs->

```
<terminated> I
.....S
..#..#.
.##.##.
..#E...
..###..
.....#.
.##.##.
 0 6
 0 5
 0 4
 1 4
 1 3
 2 3
 3 3
```