Maze Report

- Used Data Structures:-
  1. Stack (Generic) for DFS
  2. Queue (Generic) for BFS
  3. 2D array to store the map
  4. 2D array to check if visited or not
  5. Stack (Generic) for storing the path

- Algorithm:
  1. Coordinate System:

     The coordinates of each point is stored in a 2D array of (1*2) storing the i and j coordinate of the point. At first the start and end locations are both stored as attributes in the class.

  2. Searching for a path:

     The search for the path begins by checking the children (neighboring points) in a specific direction (up, right, down and left). As we check each point it is marked as visited and placed in a suitable structure whether stack for DFS or Queue for BFS.

  3. Storing the path:

     The path is determined as follows, push the start point at first. As the search process goes, if the point being checked has at least one neighboring point and the difference in the i coordinates or the j coordinates but not both between it and the Top of the stack of the path, the point is pushed to the stack.

4. Returning the right type

At the end the Stack is popped into a 2D array of (stack_size * 2) and returned.

- Details to be clarified:

1. Map and visitedMap:

The map and visitedMap both have 2 extra rows and 2 extra columns as padding. This padding at first is set to be visited. The padding reduces the condition to test whether the next coordinate is out of bounds of the array.

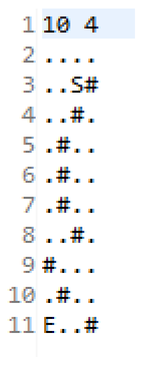
2. Generic Data Type:

The Generic data structures all use int[][] which stores the coordinates of the point.

- Sample Runs:

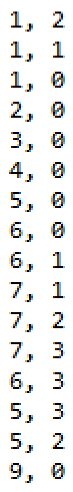
Case 1:

Test



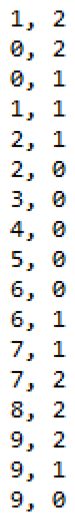
```
10 4
....
..S#
..#.
.#..
.#..
.#..
..#.
#...
.#..
E..#
```

BFS



```
1, 2
1, 1
1, 0
2, 0
3, 0
4, 0
5, 0
6, 0
6, 1
7, 1
7, 2
7, 3
6, 3
5, 3
5, 2
9, 0
```

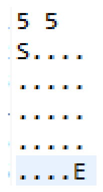
DFS



```
1, 2
0, 2
0, 1
1, 1
2, 1
2, 0
3, 0
4, 0
5, 0
6, 0
6, 1
7, 1
7, 2
8, 2
9, 2
9, 1
9, 0
```

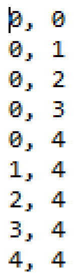
Case 2:

Test



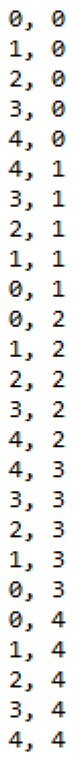
```
5 5
S....
.....
.....
.....
....E
```

BFS



```
0, 0
0, 1
0, 2
0, 3
0, 4
1, 4
2, 4
3, 4
4, 4
```

DFS



```
0, 0
1, 0
2, 0
3, 0
4, 0
4, 1
3, 1
2, 1
1, 1
0, 1
0, 2
1, 2
2, 2
3, 2
4, 2
4, 3
3, 3
2, 3
1, 3
0, 3
0, 4
1, 4
2, 4
3, 4
4, 4
```

- Comparing the two algorithms:

  As seen from test case 2, DFS tends to search all the possible routes and it takes the longest path, while BFS tends to have the shortest path.