Shortest Path in the Maze

Problem Statement: To find the shortest path from the user-chosen cell to the nearest exit.

Input format: Program takes input in a file format in which there will be walls (“#”), and space (“ ”). These walls and spaces are created in such a way that there should be a way from start point (“S”) to exits (“E”) through spaces.

Example:

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

Output format: Program should produce the output in such a way that a path (“.”) of shortest distance should be created from the start point (“S”) and the end point (“E”).

Example:

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

Notations:

Symbol	Representation
# - Hash	Wall.
( ) - Space	Empty routes.
S	Starting position.
E	Exits.
. - Dot	Path from start to exit.

Algorithm Used: Breadth First Search.

In BFS, the algorithm explores to all the corresponding children coordinate before going to the grandchildren coordinates. So that this algorithm will make sure that all nodes at a particular distance from the parent node, are traversed at the same time.

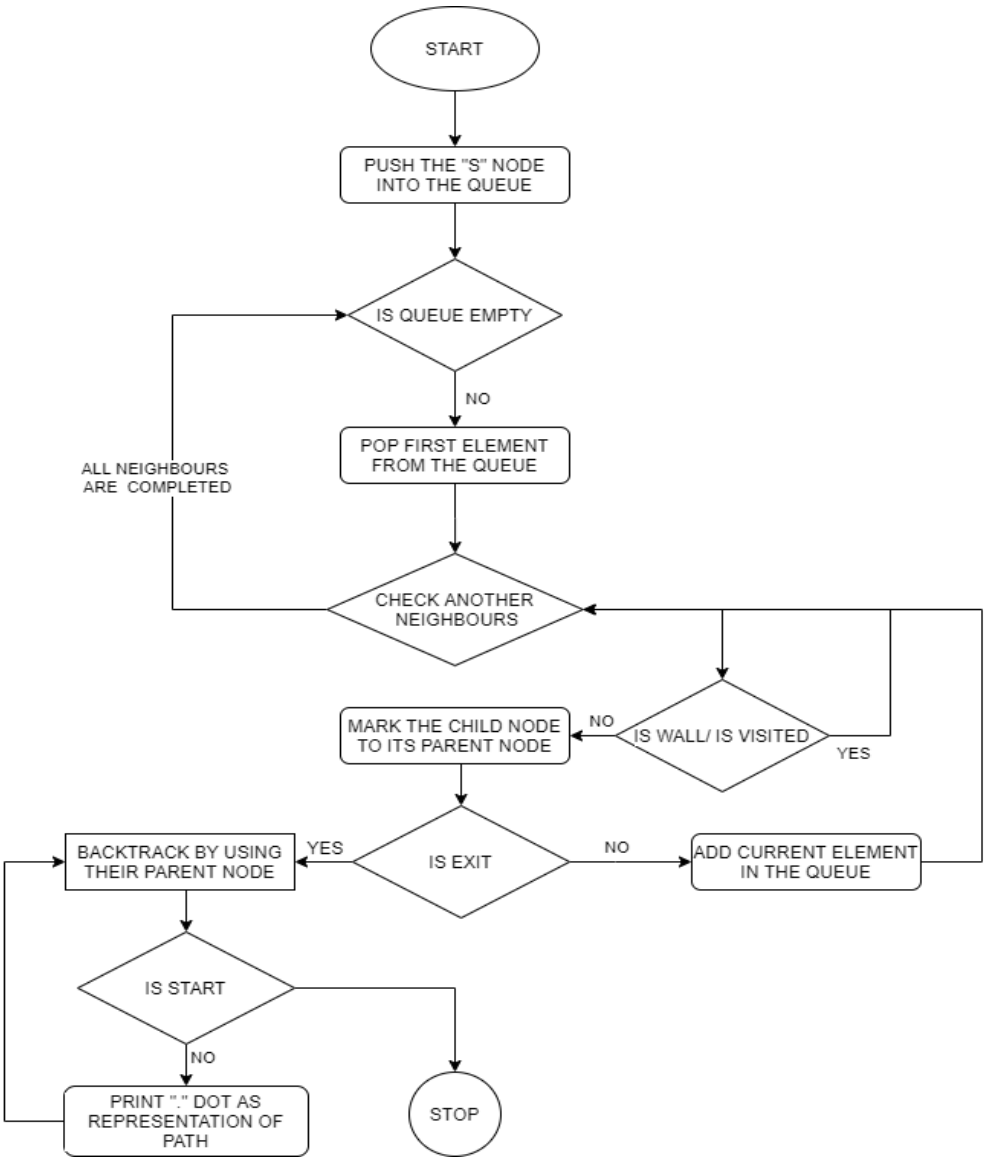
The algorithm can be outlined as follows:

1) Add the starting node in queue
2) Check whether queue is empty or not. While the queue is not empty, pop first node from the queue and do following:

   2.1) If we reach the wall or the node is already visited, skip to next iteration

   2.2) If we reach the exit node, backtrack from current node till start node to find the shortest path

   2.3) Else, add all remaining nodes into the queue and mark those nodes with their respective parent nodes to keep track of their parent. This is important to find the path once exit node is encountered.



Implementation:

Language: JAVA.

Tool / IDE: Eclipse.

As we know at a particular point we have to traverse from that point to all the four directions. For that we have to use a variable called ***DIRECTIONS.***

```
int[][] DIRECTIONS = { { 0, 1 }, { 1, 0 }, { 0, -1 }, { -1, 0 } };
```

For every point by using ***DIRECTIONS*** we have to check isVisited or isWall by using the following code.

```
    public boolean isWall(int row, int col) {
        return maze[row][col] == WALL;
    }

    public boolean isValidLocation(int row, int col) {
        if (row < 0 || row >= getHeight() || col < 0 || col >= getWidth()) {
            return false;
        }
        return true;
    }
```

After checking every node by using isWall and isVisited all those nodes should be added to the queue. Check whether queue is empty or not. While the queue is not empty, pop first node from the queue. For that first node we again check whether there is a wall or it is already visited or not. It is an iteration process. In this process we keeps track on parents node for each and every node.

```
        while (!nextToVisit.isEmpty())
        {
               Point cur = nextToVisit.remove();

            if (!maze.isValidLocation(cur.getX(), cur.getY()) ||
maze.isExplored(cur.getX(), cur.getY()))
            {
                continue;
            }

            if (maze.isWall(cur.getX(), cur.getY()))
            {
                maze.setVisited(cur.getX(), cur.getY(), true);
                continue;
            }

            if (maze.isExit(cur.getX(), cur.getY()))
            {
                return backtrackPath(cur);
            }

            for (int[] direction : DIRECTIONS)
            {
               Point coordinate = new Point(cur.getX() + direction[0], cur.getY() +
direction[1], cur);
                nextToVisit.add(coordinate);
                maze.setVisited(cur.getX(), cur.getY(), true);
            }
        }
        return Collections.emptyList();
    }
```

Whenever we get an exit node then with the help of parent node we will reach from end to start which resulting to shortest exit from the start point.

```
public List<Point> backtrackPath(Point cur)
{
    List<Point> path = new ArrayList<>();
    Point iter = cur;

    while (iter != null)
    {
        path.add(iter);
        iter = iter.parent;
    }

    return path;
}
```

Along with this document I am attaching the code file along with .txt of maze files.