Joshua Furman
Analysis of Algorithms
Project 3 Itsy-Bitsy-Spider-Maze Report

<u>REPORT QUESTIONS:</u>

1)

To model this problem, I used a directed and unweighted graph, represented by an adjacency list. Each vertex/node in my graph represents a cell in the maze. The data held in each cell is the binary string used in the input, a list holding pointers to the vertices/nodes that it is connected to which is stored as a pair<Node* cell, char direction> where direction is the direction you need to move from the cell you are at to the cell in the pair, the level, row, and column where the cell is in the maze and 3 flags to determine if the vertex/node is discovered, is the start and is the end. Each edge in the graph represents a move that can be made from one cell to another.

2)

The algorithm I used to solve the problem is a variation of a recursive depth-first search that stops and returns once it reaches the vertex/node representing the end of the maze. To save the path, I stored a pair<Node* cell, char direction> in each node/vertex's adjacency list. I then used a flag variable to determine if I've reached the end and when it is set to true, I start returning and adding the direction to a vector that stores the whole path. I then iterate through the vector backwards to get the full path from the start cell to the end cell in the maze.

Pseudocode for my DFS algorithm is on the next page:

```
DFS(starting_cell)
{
    mark cell as discovered

    if cell.end == true
    {
        end_reached flag is set to true
    }

    for v in cells.neighbors
    {
        if v.discovered != true
        {
            DFS(v)
        }

        if end_reached flag == true
        {
            add v.direction to vector holding the path
            return
        }
    }
}
```