Algorithms Project 3 - Robert Muehler

1. To implement the maze as a Graph, I will use an adjacency list. An adjacency list has the benefit of allowing me to set properties for the edges, like the labels for the direction being traversed. It will also use less memory in general than an adjacency matrix, since a lot of the “cells” in the maze can’t possibly be connected to begin with in a large maze. It also is better than an edge list it is easier to traverse than having to iterate through a Graph to find an edge you are looking for. The adjacency list will be directed, as the “mouse” can always return along the same path it has taken and the directional values of the edges will have to indicate the mouse is going the opposite direction. The list will be unweighted since the distance travelled does not matter.

The vertices correspond to the given “cell” or current location in the cell. I will map out the vertices to be numbered from 0 to TotalNumberOfVertices-1. That means a cell at (0,1,1) will have a label of 4.

The edges represent a possible path from the current vertex. These will be labeled with the direction being traversed.

2. I will use a DFS approach to solve the problem. The spider will recursively travel down a path and mark each cell as visited until it reaches the goal or can no longer travel to another unvisited cell. Each time it visits a cell the direction the spider travelled will be printed to output. When the spider needs to backtrack it will keep travelling a recording the direction travelled until it reaches a well where it can visit an unvisited cell. Eventually the spider will have visited every cell that possibly connects to the starting cell. If the goal hasn’t been reached, this means there is no solution to the problem.