Project 3: Graph modeling and graph algorithms

Alexander Ashmore

Modeling the Problem

Question 1

What type of graph would you use to model the problem input (detailed in the Section 3.1), and how would you construct this graph? (I.e., what do the vertices, edges, etc., correspond to?) Be specific here; we discussed a number of different types of graphs in class.

The graph was implemented as an adjacency list for this problem. It will be unweighted and non-directed. The choices were between an adjacency list and an adjacency matrix, but since there are only 6 possible edges for each node, the tree is sparse, so it is better to implement an adjacency list (less space). A simple implementation can be made by using vectors in C++.

The vertices will correspond to the positions on in the maze. The number of vertices can be calculated by multiplying the given dimensions of the maze: level x row x columns. When stored on the adjacency list, it will go from position 0 to (number of vertices - 1).

The starting vertex can be calculated by this equation:

startVertex = startLevel * (mazeRow * mazeColumn) + (startRow * mazeRow) + startColumn;

The goal vertex can be calculated by a similar equation:

goalVertex = goalLevel * (mazeRow * mazeColumn) + (goalRow * mazeRow) + goalColumn;

The index of the vertices can be thought of left to right for each row and then go back to the leftmost column once you hit the end of that row. For example, a 3 x 3 x 3 maze can be represented as:

0	1	2
3	4	5
6	7	8
9	10	11
12	13	14
15	16	17
18	19	20
21	22	23
24	25	26

The edges for the graph represent the directions you can go at each vertex or position in the maze. There is a maximum of 6 possible edges because of the 6-digit binary value representing the directions (N, E, S, W, U, D) the spider can travel. To connect the edge to each vertex on the graph, you would read the 6 digit binary representation and create an edge to the other vertex if the digit is 1 for that corresponding direction. To calculate which new vertex the current vertex will create an edge with, apply some equations for the corresponding direction:

N: index - # of totalcolumns
E: index + 1
S: index + # of totalcolumns
W: index - 1
U: index + (# of totalcolumns x # of totalrows)
D: index - (# of totalcolumns x # of totalrows)

Question 2

What algorithm will you use to solve the problem? Be sure to describe not just the general algorithm you will use, but how you will identify the sequence of moves the spider must take in order to reach the goal.

I used a BFS to traverse the graph created by using an adjacency list. Dijkstra's algorithm seemed appealing, but I could not figure out how to assign a good weight. It ended up having similar weights for each direction, making a BFS more appealing to implement to find the shortest path. BFS works by visiting all the positions or vertexes one step away from the starting position, then visiting all vertexes two steps away and so on until the goal node is found. BFS is then able to return the shortest path back, along with all other possible paths depending on implementation.

You can store the path on a queue and start by pushing the starting vertex. You would check all nodes connected to the current vertex and if it is not yet visited, you would push on the queue and repeat. This is repeated until the goal vertex is reached. You then return the path found.

For each path stored as a queue of vertexes, check the next vertex in the queue corresponds with a directional equation. If the value is the same, that means that is the direction the spider traveled. The sequence of moves the spider takes can be calculated by using the equations used earlier. I'll post them again below:

N: index - # of totalcolumns
E: index + 1
S: index + # of totalcolumns
W: index - 1
U: index + (# of totalcolumns x # of totalrows)
D: index - (# of totalcolumns x # of totalrows)

Once we check the direction for each value, we can return this string as the path taken.