1. I would use a 3D graph rather than use an adjacency list. The 3D graph for each node of it the i, j and k would represent the x, y, z coordinates respectively. The edges represent the connection or lack of wall for the individual node. To get the final path for the maze I would store the path traversal as a string of correct movements that then get printed out. I am also using an unordered map with weights given with the 3d coordinates.
2. I would use a DFS to solve this 3D maze. The dfs will output the necessary moves the spider would need to take to reach the goal. The implementation of the nodes would have the adjacent nodes it can reach together.

DFS:

Call DFS-Recursive(v)

If any vertex undiscovered pick one and restart

DFS-Recursive(v):

Mark v as discovered

For c in N(v):

If c is unexplored

Call DFS-Recursive

Mark v as explored

I am using this DFS to search for the goal so the algorithm branches through the maze until it finds the goal and then it stops. I tried to implement backtracking through the DFS to give the path from the algorithm. I check where the node we are looking at is coming from and use that as the output.

The node itself has the value of all its directions of NESWUD and pointers to those NESWUD nodes and the position of itself in x,y,z coordinates.