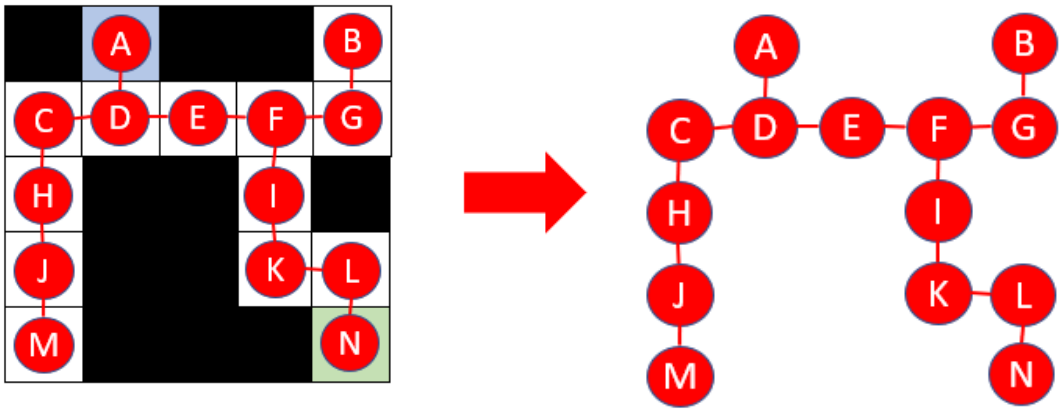
George Webster
CSCI 313
HW2—Question 7: Solve a maze using a stack

A maze can be represented in numerous ways in a computer program. All mazes, however, can be represented as graphs. Accordingly, this program stores the maze as a vector of vectors. The first dimension refers to the integer value for a node of the graph, and the second dimension is the list of nodes to which that node connects. For example, the third node, assigned the number 2, connects to the fourth node, denoted 3, and to the eighth node, denoted 7. Therefore, the value at [2][0] will be 3, and the value at [2][1] will be 7. For the purposes of legibility, it is easier to assign the nodes letters, and use simple function to convert between characters and integers as needed. The following maze is manually entered in the main function:



Node A is the starting point, and Node N is the target. Next, the function creates a stack called solutionStack, and initializes it. Additionally, it creates a Boolean array call visited, which will keep track of which nodes have already been accessed, and assigns all of the values to false. Then it marks the starting node of the maze as visited (by switching visited[0] to true) and calls the function checkNode, giving it the node (in this case A = 0), the maze, the visited array, and the solutionStack.

The function checkNode is a recursive function. It's first element is simply to return if visited[13] is true, indicating the target has been found. If that is not true, it creates and iterator for a vector of integers, and uses a for loop to iterate through the vector stored at the node. For the node A, the vector will iterator through all of the values stored at maze[0], which in this case is a single one, D. The for loop checks if D is the target. If it is the target, it marks visited[13] as true and pushes it to the stack, then returns. Since it isn't, it then goes on to check if D has already been visited. If it has, the loop skips it. Since D hasn't yet been visited, the function marks visited[3] as true, pushes D to the stack, and then calls checkNode on D.

The resulting function creates, effectively, a depth-first search of the maze. It will travel to the first node seen by the first node seen by the first node etc. until it reaches a node that does not see an unvisited node. If the target node still hasn't been found, and the node does not see an unvisited node, the node is popped off the stack, and the function returns to the preceding node. This continues until all nodes have been visited or until the target is found.

Once the function check node has been finished, the stack will either be empty, in which case there is no solution to get to the target, or it will contain the solution to the maze in order, with the target on the top and the start at the bottom. To display it, the main function empties the stack into another temporary stack, which reverses the order and allows the function to display the solution from start to finish. There are many ways to solve this problem, both iteratively and recursively. However, if one wishes to use depth-first search, a stack is the perfect tool to use.