CSCI 313 Prof. M Fried HW2

Problem 7 – Solve a maze using Stacks

Muhammad Abbas Ali Sajjad

The key to solving a maze using stacks requires several parameters to be set accordingly and a few algorithms can be utilized and implemented in different ways. The first step was to create a pseudo-maze that already has a predetermined path in order to verify the accuracy at which the algorithm can solve the maze. I used a 5x5 grid as a 2D array of integers to represent my maze. The maze array is comprised solely of 0s and 1s. A “0” representing a path or block that is impassable or obstructed and a “1” representing a clear path.

The stack I created was a generic stack implemented using linked-lists and nodes. Where each MoveNode object houses a 1D array of 2 elements representing the corresponding grid elements. The second data member being a pointer to another node, as next. We also had to create our own comparator in the MoveNode class in order to compare two nodes and ascertain whether or not they had the same coordinates corresponding to the same position in the maze.

The stack represents all the paths necessary to go from the starting point of (0,0) to the end point (4,4) in our case because our maze is a 5x5 grid but it could be of any size. Each node on the stack represents what coordinates in the path the pointer must traverse in a specific order to reach its destination. The stack method of push() takes in grid coordinates and creates a new MoveNode adding it to the top of the stack.

The key to solving our maze are two important functions, namely isLegal() and solveMaze(). The main algorithm that inspires our solution is the backtracking algorithm. In our case there are two legal moves the pointer can take, either downwards or across. The pointer represented by current_pos will continuously find a path until the final END coordinate is reached. The pointer will take every legal move it can until it can no longer. The key to a backtracking algorithm is to retrace your steps and make alternative decisions to different paths in order to effectively and eventually solve the maze. This is where a stack is especially useful because we have already incorporated a pop() function that removes the top most node. So. If we see that the current path yields a dead end we pop() off the current node or position and immediately take the alternative path.

One of the challenges I face was to make sure that I take an alternative path, as opposed to the same path after popping off a bad path. The way I solved it was to immediately increment either the row or col positioning taking the alternative chose as compared to the past decision. I used a Boolean variable that flips between 2 choices to the program knows which was taken last and not to take it again.

isLegal() is an important function in order to determine whether the position of interest is legal, i.e. not impassable terrain. The isLegal function ensures the coordinate inputs are first not out of bounds and second not a “0” and only passable if it returns a “1.”