Department of Electrical and Computer Engineering
ECE 3326: Optimization Methods
Fall 2012

Project #5

In this project, you will develop algorithms that find paths through a maze.

The input is a text file containing a maze. Each maze begins with the number of rows and columns in the maze and a character for every cell in the maze. A cell contains a `O` if the solver is allowed to occupy the cell. A cell contains `X` if the solver is not allowed to occupy the cell.

The solver starts at cell `(0,0)` in the upper left, and the goal is to get to cell `(rows-1, cols-1)` in the lower right. A legal move from a cell is to move left, right, up, or down to an immediately adjacent cell that contains a space. Moving off any edge of the board is not allowed.

Part a

Functions to handle file I/O, and a complete graph class, are included as part of the assignment. In the printout of the graph, the current cell is represented by + and the goal cell is represented by *. Add functions that:

1. Create a graph that represents the legal moves between cells. Each node should represent a cell, and each edge should represent a legal move between adjacent cells.
2. Write functions that map a cell's coordinates into a node number, and map a node number into a cell's coordinates. Use these functions to create the graph.
3. Write a recursive function `findPathRecursive` that looks for a path from the start cell to the goal cell using depth-first search. If a path from the start to the goal exists, your program should print a sequence of correct moves (`Go left, go right`, etc.). If no path from the start to the goal exists, the program should print, `No path exists`.
4. Write a function `findPathNonRecursive` that does the same thing as in 3 using depth-first search, but without using recursion.

The code you submit should apply both findPath functions to each maze, one after the other. If a solution exists, the solver should simulate the solution to each maze by calling the `maze::print()` function after each move.