CSCI 235 - Spring 2015
Homework 2

Due: Wednesday, April 22nd, 2015, 11:59 pm

1 Navigating through mazes

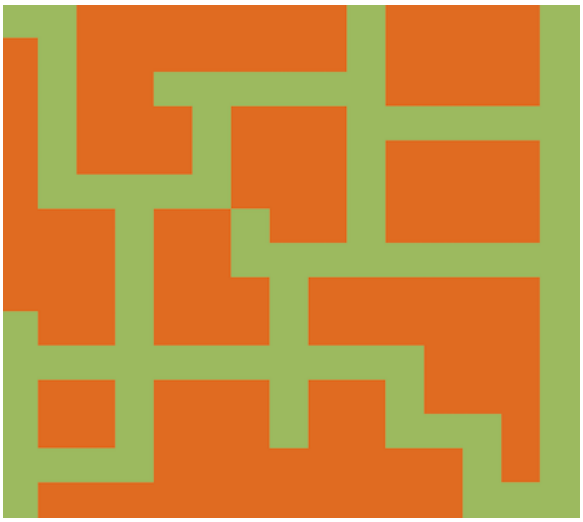
In class, we have studied a couple of algorithms to find if there is a path from one city to another. The same concept can be extended to derive algorithms which solve mazes. In this homework, you will write a program that reads in a file that represents a maze, and then outputs paths that it could find from the origin (top left corner) to the destination (bottom left corner).

1.1 Input

The input will be a text file. The first line contains two numbers. Separated by a space, the first number is the number of rows (N) and the second the number of columns (M). In the next N lines are the N rows of the maze, each with M cells, to make up M columns. Below are the contents of a file that represents a maze with 15 rows and 15 columns. Note that cells are separated by spaces.

```
15 15
. . - - - - - - - . - - - - .
- . - - - - - - - . - - - - .
- . - - . . . . . - - - - - .
- . - - - . - - - . . . . . .
- . - - - . - - - . - - - - .
- . . . . . - - - . - - - - .
- - - . - - . - - . - - - - .
- - - . - - . . . . . . . . .
- - - . - - - . - - - - - - .
. - - . - - - . - - - - - - .
. . . . . - - - .
. - - . - - - . - - . - - - .
. - - . - - - . - - . . . - .
. . . . - - - - - - - - . - .
. - - - - - - - - - - - . . .
```

A '-' represents a wall, whereas a '.' represents a cell that is part of a path. The above text represents the following maze:



You may assume that a path's width is exactly one cell.

1.1.1 Representation when programming

It is better not to think of a maze as a graph, since that representation is a bit complicated (for e.g., what does a node represent? An intersection, or some other point in the path?) Instead, use a 2-D array to store the types of the cells, and use it when deciding whether to take the next cell.

1.1.2 General approach

This can be solved using either stacks or recursion. Recursion may be a bit cleaner, and will give you a better sense of what is going on. Start with the (0,0) cell, and in each call, decide how to proceed to a different cell. Note that the only directions you can move our Up, Down, Left and Right, and you may move only one cell at a time. When writing your recursive function, you can fix a strategy: as in the cities example, you can fix an order on the directions that you want to try each time (in the cities we chose lexicographically); here you can try U, D, L, R.

Of course, you cannot climb walls, pass through walls, or go outside the grid. Make sure to take care of cycles (i.e., avoid them). You have found a path when you reach the bottom-right cell.

Also note that your program must report if a path was not found.

1.1.3 Output

The output should be a grid that shows the path by numbering cells in it, from start to end. Below is an output of this program (on the above example) where the strategy of

choosing the next cell was Down, Up, Right, Left. You should get the same path for this example if you choose this strategy.

```
 0  1  -  -  -  -  -  -  -  -  -  -  -  -  .
 -  2  -  -  -  -  -  -  -  -  -  -  -  -  .
 -  3  -  -  -  -  -  -  -  -  -  -  -  -  .
 -  4  -  -  -  -  -  -  - 26 27 28 29 30 31
 -  5  -  -  -  -  -  -  - 25  -  -  -  - 32
 -  6  7  8  -  -  -  -  - 24  -  -  -  - 33
 -  -  -  9  -  -  .  -  - 23  -  -  -  - 34
 -  -  - 10  -  -  . 20 21 22  .  .  .  . 35
 -  -  - 11  -  -  - 19  -  -  -  -  -  - 36
 -  -  - 12  -  -  - 18  -  -  -  -  -  - 37
 -  -  - 13 14 15 16 17  .  .  .  -  -  - 38
 -  -  -  -  -  -  -  -  -  -  .  -  -  - 39
 -  -  -  -  -  -  -  -  -  -  .  .  .  - 40
 -  -  -  -  -  -  -  -  -  -  -  -  .  - 41
 -  -  -  -  -  -  -  -  -  -  -  -  .  . 42
```

1.2 Tasks

There are two tasks, the second one building on the first one. In each case, your program will read the maze map and try to solve it.

1.2.1 Find one path

Your task is to write a program that finds one path from start to end. It doesn't have to be an efficient path. Just pick cells according to a fixed order.

Note that as before, you have to mark cells as you visit them. However, to produce an output of the sort we want, you will also have to keep track of the cells you have visited in the previous calls leading up to each recursive call. To do this, you may want to pass the entire path discovered up till now to each recursive call. Determining how to solve this is part of the task.

This is important because simply globally numbering cells as you visit them is misleading, since you may backtrack after a visiting a cell.

1.2.2 Find all paths

In this task, you have to find all paths, and display a grid for each path found. This is rather easy, if you have figured out how each recursive call can know the path in its history. For this task, instead of marking cells globally as visited, you need to check if the next cell is not already in this path. This will help you avoid cycles.

2 Execution and Submission

Your program must run and take arguments from the command line. The program will be run in the following way:

```
$ ./MazeSolver <TASK_NUM> <INPUT_FILE>
```

That is, your program should take two extra arguments, the first being the number of the task (1 or 2) that the user wishes to run, and the second being the name of the input file.

This means that you have to write and submit one program that accomplishes all of this. You can create as many files as functions as you want, but as always, executing `make` should build your program.

Submit all your work in a zip or tar.gz file with the name `<YOUR_FULL_NAME>_hw3.zip` or `<YOUR_FULL_NAME>_hw3.zip`, on Blackboard. The same rules as before apply for grading.

On Blackboard, in the comments section, please write down how many hours it took you complete this homework. Failure to write this will result in a deduction of 5 points.