CS302 -- Lab 7 -- The Maze Solver

- CS302 -- Data Structures and Algorithms II
- Fall, 2021
- James S. Plank
- This file: **http://web.eecs.utk.edu/~jplank/plank/classes/cs302/Labs/Lab7**
- Lab Directory: **/home/plank/cs302/Labs/Lab7**

What you Submit

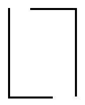
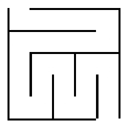
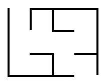
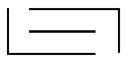
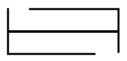
maze_solve.cpp

Maze Solving

You are to write the program **maze_solve**, where you will solve a maze by employing depth first search. The program takes no command line arguments. It receives a maze on standard input. Mazes are in the following format:

- Maze files are composed of words with no line dependencies.
- The first word should be "ROWS" followed by the number of rows in the maze.
- The next word should be "COLS" followed by the number of columns in the maze.
- After that should be any number of "WALL" specifications, which are of the form "WALL", then two indices. Each index is of a cell in the maze. The cell in row *i* column *j* is in index *ic*+*j*, where *c* is the number of columns in the maze.

Here are some example mazes with their graphical representations:

m1.txt	**m2.txt**	**m3.txt**	**m4.txt**	**m5.txt**
				
`ROWS 4 COLS 3`	`ROWS 5 COLS 5` `WALL 0 5` `WALL 1 6` `WALL 2 7` `WALL 3 8` `WALL 9 14` `WALL 8 13` `WALL 7 12` `WALL 6 11` `WALL 10 11` `WALL 15 16` `WALL 21 22` `WALL 16 17` `WALL 12 13` `WALL 17 18` `WALL 23 24` `WALL 18 19`	`ROWS 3 COLS 4` `WALL 7 11` `WALL 10 9` `WALL 1 0` `WALL 1 2` `WALL 5 9` `WALL 6 2`	`ROWS 2 COLS 5` `WALL 1 6` `WALL 2 7` `WALL 3 8`	`ROWS 2 COLS 5 WALL 0 5 WALL 1 6 WALL 2 7 WALL 3 8 WALL 4 9`

Observe a few things about mazes. First, they don't have to have any walls. They do not have to have unique paths through the maze (like **m4.txt**), and they don't have to have any paths at all (like **m5.txt**). As you can see from **m2.txt**, the walls may be specified in any order. And there is no line formatting -- just words in order.

You are to read in a maze file, turn it into a graph, and use depth-first search to find a path through the graph from cell 0 to cell *rc-1*. As output, you should output the maze in the same format as the input, and you should include "PATH" and an index of each cell that belongs in the path. The "PATH" statements should specify the path in the proper order. For example:

```
UNIX> maze_solve < m1.txt
ROWS 4 COLS 3
PATH 0
PATH 3
PATH 6
PATH 9
PATH 10
PATH 11
UNIX> maze_solve < m2.txt
ROWS 5 COLS 5
WALL 0 5
WALL 1 6
WALL 2 7
WALL 3 8
WALL 6 11
WALL 7 12
WALL 8 13
WALL 9 14
WALL 10 11
WALL 12 13
WALL 15 16
WALL 16 17
WALL 17 18
WALL 18 19
WALL 21 22
WALL 23 24
PATH 0
PATH 1
PATH 2
PATH 3
PATH 4
PATH 9
PATH 8
PATH 7
PATH 6
PATH 5
PATH 10
PATH 15
PATH 20
PATH 21
PATH 16
PATH 11
PATH 12
PATH 17
PATH 22
PATH 23
PATH 18
PATH 13
PATH 14
PATH 19
PATH 24
UNIX> maze_solve < m3.txt
ROWS 3 COLS 4
WALL 0 1
WALL 1 2
WALL 2 6
WALL 5 9
WALL 7 11
WALL 9 10
PATH 0
PATH 4
PATH 5
PATH 6
PATH 10
PATH 11
UNIX> maze_solve < m5.txt
ROWS 2 COLS 5
WALL 0 5
WALL 1 6
WALL 2 7
WALL 3 8
WALL 4 9
UNIX>
```

The output specification of the maze does not have to be the exact same as the input specification, nor does it have to be in any order. It must represent the same maze though.

If there are multiple paths through the maze, your program only has to find one of them. It doesn't have to be the shortest path. If there is no path, then no path should be specified.

Don't worry about error-checking your input files.

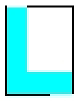
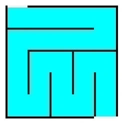
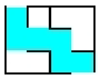
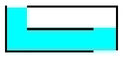
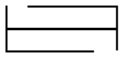
Resources

I have two useful programs for testing and displaying mazes:

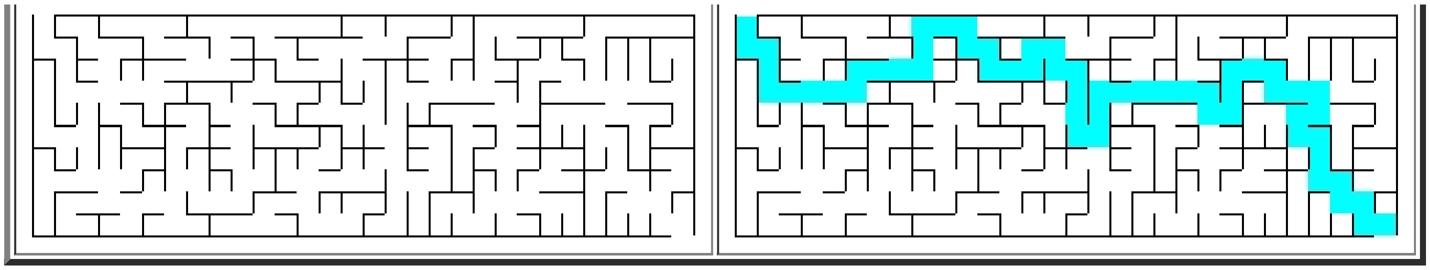
- **maze_gen rows cols** generates a random maze with the given rows and columns. It spits out walls in random order.
- **maze_ppm cell-width** takes a maze on standard input (with or without paths) and turns it into a PPM file. **Cell-width** must be an odd number.

Since PPM files are big, you can use **convert** to skip making a PPM file. For example:

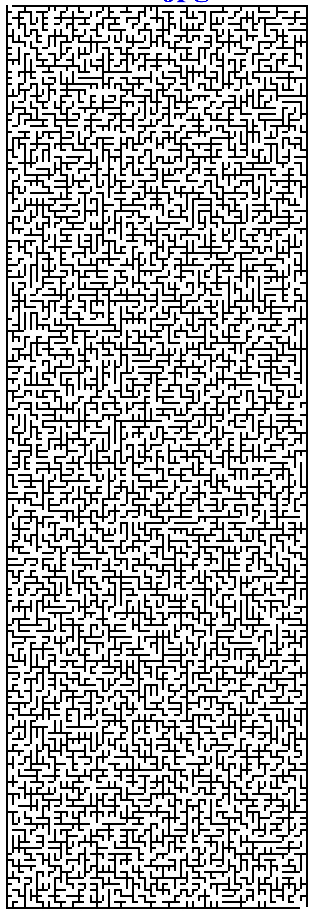
```
UNIX> maze_solve < m1.txt | maze_ppm 11 | convert - m1s.jpg
UNIX> maze_solve < m2.txt | maze_ppm 11 | convert - m2s.jpg
UNIX> maze_solve < m3.txt | maze_ppm 11 | convert - m3s.jpg
UNIX> maze_solve < m4.txt | maze_ppm 11 | convert - m4s.jpg
UNIX> maze_solve < m5.txt | maze_ppm 11 | convert - m5s.jpg
UNIX> maze_gen 10 30 > ex1.txt
UNIX> maze_gen 150 50 > ex2.txt
UNIX> maze_ppm 11 < ex1.txt | convert - ex1.jpg
UNIX> maze_solve < ex1.txt | maze_ppm 11 | convert - ex1s.jpg
UNIX> maze_ppm 3 < ex2.txt | convert - ex2.jpg
UNIX> maze_solve < ex2.txt | maze_ppm 3 | convert - ex2s.jpg
UNIX> maze_gen 400 400 | maze_solve | maze_ppm 3 | convert - big-ol-maze.jpg
```

m1s.jpg	m2s.jpg	m3s.jpg	m4s.jpg	m5s.jpg
				

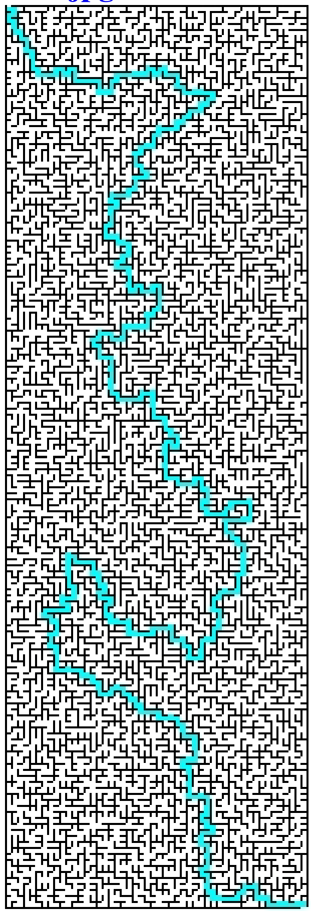
ex1.txt, ex1.jpg	ex1s.jpg



ex2.txt, ex2.jpg



ex2s.jpg



big-ol-maze.jpg

Finally, the program **ms_checker** checks the output of **maze_solve**. It takes a non-solved maze file as a command line argument, and the output of **maze_solve** on that maze on standard input. If the solution is correct, then it simply returns. Otherwise, it specifies the error:

```
UNIX> maze_solve < m1.txt | ms_checker m1.txt
UNIX> maze_solve < m1.txt | ms_checker m2.txt
ROWS of the solution does not match ROWS of the input file
UNIX> maze_solve < m2.txt | ms_checker m2.txt
UNIX> cat m2.txt | ms_checker m2.txt
Bad Solution: Empty path, yet there is a solution to the maze
UNIX> maze_solve < m4.txt | ms_checker m4.txt
UNIX> maze_solve < m4.txt | ms_checker m5.txt
Bad Solution: Bad maze -- missing some of the original walls
UNIX>
```