ECE36800 Programming Assignment 4

Due Sunday, July 23, 2023, 11:59pm

You are required to implement a program to find the shortest paths. You are allowed to implement whatever algorithms you wish. However, it is suggested you use the algorithms presented in class. There are 3 modes your program can run in: Time, Path, and Remove. They are described in detail below.

It is fine for you to use the code provided in the lecture notes for this assignment. However, you should re-organize the code. For example, you may want to break functionality up into separate functions to help with code readability.

This assignment is to be completed individually. It is encouraged to discuss solutions with your classmates but you should refrain from looking at each other's code.

Input File

The input file you are given for this assignment is a text file. This means the file consists of ASCII characters and you should use calls to `fscanf` in order to read its contents. The input file holds 3 pieces of information: the grid size, the start location, and the grid itself.

Grid Size Section

The grid size section is a single line containing an ordered pair, `(m,n)`, where m is the number of rows and n is the number of columns in the grid. The line will be printed using the format string `"(%d,%d)\n"`. Note, there are no spaces.

Start Location Section

The start location section also consists of a single line. The start location will be printed out as a coordinate `(r,c)` where r is the start row and c is the start column. The line will be printed using the format string `"(%d,%d)\n"`. Note, there are no spaces. Assume the grid is zero-indexed so `(0, 0)` is the top left cell of the grid and `(m-1, n-1)` is the bottom right cell of the grid.

Grid Section

The grid section consists of `m` lines. Each line has `n` `chars` (excluding the new line character, `\n`). Each line represents a row in the grid and each `char` represents a cell in that row. Because it is a grid, every row will have the same number of `chars`. The grid consists of two types of cells: walls and rooms. A wall is represented by `1` and a room is represented by `0`.

Example File

The following file contents would correspond to the grid in Figure 1. The light cells represent rooms and the dark cells represent walls. The green cell represents the start location.

```
(5,6)
(2,3)
000000
111110
000000
011111
000000
```


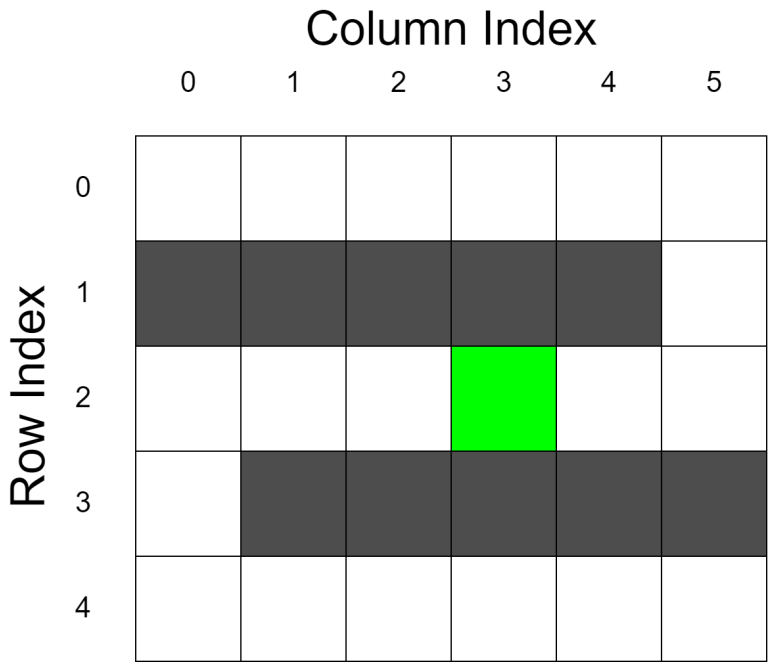

Figure 1: Example Grid

Restrictions

The start location will always be a room. You should return `EXIT_FAILURE` if it is not. All four corners of the grid will also be rooms. If all four corners are not rooms, you should return `EXIT_FAILURE`. `m` and `n` will always be larger than 1 and their product will be smaller than `INT_MAX`. If `m` or `n` fail to meet this criterion, return `EXIT_FAILURE`. The grid will always be the correct size, matching `m` and `n`.

Grid Traversal

The goal of this assignment is to find the shortest path from the starting cell to the 4 corners of the grid. The grid consists of two types of cells, rooms, and walls. You can move from one room to another room. However, you cannot move through a wall. You are only allowed to move to rooms that you share a border with. This means you can move up, down, left, and right but you cannot move diagonally. For example, say we have the grid as in Figure 1. You can move from the cell (2,3) to (2,4), but you cannot move from (2,3) to (1,3) because (1,3) is a wall. You are also unable to move from the cell (2,1) to (3,0) because they do not share a border (this is a diagonal move). It always takes 1 time unit to move between adjacent rooms.

Program Mode: Time

In this mode, your program will be called with the following syntax. `<input_fname>` will be populated with the input file containing the grid.

```
./pa4 -t <input_fname>
```

In this mode, you are required to find the travel time from the start node to all 4 corners. It is not guaranteed that you can reach all 4 corners from the start location. Your program should still be able to handle this. When your program is running in this mode, it should print 4 `int`s out to the `stdout`. Each

`int` represents the travel time to a different corner. Your program should print out each `int` on the same line separated by commas (no spaces) starting with the top left, then the top right, then the bottom right, then the bottom left (clockwise order). You should print a new line character after the 4 `int`s. You should print out using a print statement similar to the following.

```
fprintf(stdout, "%d,%d,%d,%d\n", top_left, top_right, bottom_right, bottom_left);
```

If you cannot reach a corner, you should print out `-1` for that corner. You should count the time it takes (i.e number of transitions), not the number of cells visited. If the start cell is one of the corner cells, the time to reach that corner cell would be `0`.

Program Mode: Path

In this mode, your program will be called with the following syntax. `<input_fname>` will be populated with the input file containing the grid.

```
./pa4 -p <input_fname>
```

In this mode, you are required to find the path from the start node to all 4 corners. It is not guaranteed that you can reach all 4 corners from the start location. Your program should still be able to handle this. The start and end cells should be included in your path. When your program is running in this mode, it should print 4 different lines, 1 line for the path to each corner. The path is printed out as a list of ordered pairs. The ordered pairs are separated by commas and elements of the ordered pairs are also separated by commas. For example, if your path contained 3 cells, it would use a format string similar to the following.

```
"(%d,%d),(%d,%d),(%d,%d)\n"
```

In reality, you would just use some sort of loop and iterate through the path instead of having longer format strings for longer path lengths. Note, there were no spaces in the former example. If you cannot reach a corner, simply print `-1` for that path. **There may be multiple correct paths**. You just need to print out a single correct path to receive full credit. More examples are listed at the end of the document.

Program Mode: Remove

In this mode, your program will be called with the following syntax. `<input_fname>` will be populated with the input file containing the grid.

```
./pa4 -r <input_fname>
```

In this mode, you are only required to find the time it takes to get from the start node to the top left corner. You will print the time it takes to get from the start node to the top left corner to the `stdout` followed by the new line character. You would use something like the following statement.

```
printf("%d\n", time);
```

However, this time you are able to remove a single wall and replace it with a room. If there is a shorter path possible by removing a different wall than the one you selected, you will fail the test case. Note, there may be cases where the top left corner was unreachable before but converting a wall to a room makes it accessible.

CLI Option	Mode Name	Weight
`-t`	Time	30%
`-p`	Path	70%
`-r`	Remove	25%
	Total	125%

Table 1: Grade weighting

Submission and Compiling

We will use the following command to compile your submission. `gcc -std=c99 -pedantic -Wvla -Wall -Wshadow -O3 *.c -o pa4`

The project requires the submission (electronically) of the C-code (source and include files) through Brightspace. You should create and submit a zip file called `pa4.zip`, which contains the `.h` and `.c` files. Your zip file should not contain a folder. **Incorrectly named submissions will not be graded correctly.**

Grading

For `-t` (Time) mode and `-p` (Path) mode, there are 4 points available for each test case. A point is earned for having the correct time or path to a corner. There are 4 corners, so 1 point for each path. For `-r` (Remove) mode, you either get full credit for a test case or no credit, your answer is either right or wrong.

The weights for each mode are listed in Table 1. Note, the total adds up to 125% instead of 100%. The remove mode is extra credit for this assignment. All test cases within a category are weighted equally.

If any memory issues are detected, you will be deducted 50% for that test case. If your program does not complete before the timeout, you will earn zero points for that test case. Because running your program with a memory checker like Valgrind is very slow, we will run your program by itself to ensure you have the correct output. Once we verify your program has produced at least partially correct output, we will test it with Valgrind for memory errors.

Final Notes

It is important your program prints nothing else to the `stdout` in your submission. It must match the format the autograder is expecting or else your program will not be graded correctly. However, we will not read any output printed to the `stderr`. It is recommended you generate some of your own test cases to test your program. While it will be difficult to test your program for correctness with the test cases you generate, you can still test to make sure your program does not have any memory issues or runtime errors with larger test cases.

Below we list some examples to help get you started. Please post on Piazza or attend office hours if you have any questions. Good luck and happy coding!

Example 1

Input file:

```
(5,6)
(2,3)
000000
111110
000000
011111
000000
```

Output in `-t` (Time) mode.

```
9,4,10,5
```

Output in `-p` (Path) mode.

```
(2,3),(2,4),(2,5),(1,5),(0,5),(0,4),(0,3),(0,2),(0,1),(0,0)
(2,3),(2,4),(2,5),(1,5),(0,5)
(2,3),(2,2),(2,1),(2,0),(3,0),(4,0),(4,1),(4,2),(4,3),(4,4),(4,5)
(2,3),(2,2),(2,1),(2,0),(3,0),(4,0)
```

Output in `-r` (Remove) mode.

```
5
```

Explanation (do not print): You can obtain a path with length 5 from (2,3) to (0,0) by removing any of the following walls: (1,0), (1,1), (1,2), (1,3)

Example 2

Input file:

```
(4,5)
(1,2)
01010
10011
00000
01110
```

Output in `-t` (Time) mode.

```
-1,-1,4,4
```

Output in `-p` (Path) mode.

```
-1
-1
(1,2),(2,2),(2,3),(2,4),(3,4)
(1,2),(1,1),(2,1),(2,0),(3,0)
```

Output in `-r` (Remove) mode.

```
3
```

Explanation (do not print): You can obtain a path with length 3 from (1,2) to (0,0) by removing any of the following walls: (1,0), (0, 1)

