CS2124 Data Structures - Spring 2021

Assignment 3 - Recursion Program

Due 3/19/21 by 11:59pm

1 Description

Jethro and Cleatus are quarantined at home and bored. They spend most of their day sitting at their computer monitor working or browsing the internet in their small apartment. Out of boredom Jethro begins counting the number of steps needed to reach each location in their small apartment. After seeing that taking different paths from their computer to their coffee maker yields different numbers of steps a question dawns on them. They want to know what is the fewest number of steps needed to reach **all** of the locations in their small apartment starting from their computer.

Fortunately, Jethro is quite skilled at ASCII art. So they model their room with ASCII characters. A space ‘ ’ can be moved through. An asterisk ‘*’ is a wall or furniture that cannot be traversed (no climbing over furniture!). Going up, down, left, right one space takes exactly one step (we’ll assume no diagonal movements for that sake of simplicity). For example, here is a possible model of their room:

```
**************
* A          *
*    *       *
*    *       *
*  ***       *
*            *
*     *****  *
*      ***   *
**************
```

Assume that (0, 0) is the upper lefthand corner. For the sake of simplicity you can assume the apartment is enclosed in ‘*’ characters and that the location of Jethro’s computer has been marked with an ‘A’.

Jethro is still new to programming and wants to hire you to write the program to label all of the ‘ ’ locations in their apartment with the minimum number of steps needed to reach them. To keep with the ASCII art theme you’ll use the letters A-Z Such that:

```
‘A’ is 0 steps
‘B’ is 1 step
‘C’ is 2 steps
```

```
...
'Y' is 24 steps
'Z' is 25 (or more) steps
```

For example, after running your algorithm on it, the apartment above should look as follows:

```
**************
*BABCDEFGHIJK*
*CBCD*FGHIJKL*
*DCDE*GHIJKLM*
*ED***HIJKLMN*
*FEFGHIJKLMNO*
*GFGHI*****OP*
*HGHIJ***RQPQ*
**************
```

Example 2:

```
**********************************
*   A                            *
**********************************
```

After applying your algorithm (we don't count past 'Z' since Jethro has a pretty small apartment):

```
**********************************
*DCBABCDEFGHIJKLMNOPQRSTUVWXYZZZZ*
**********************************
```

Example 3:

```
*******
*A    *
****  *
*  *  *
*******
```

After applying your algorithm (unreachable ' ' areas should remain unchanged):

```
*******
*ABCDE*
****EF*
*  *FG*
*******
```

2 Problems

Part 1 - Planning: (4 points)

Write a short brute force recursive idea for solving this problem. Specifically write a recursive algorithm attempts to update the symbol located at (x, y) where
$0 \leq x < MAXWIDTH, 0 \leq y < MAXHEIGHT$ and then is recursively called on all adjacent locations.

Hint 1: From any given space, you need to try to continue moving up, down, left, and right. DO NOT try to travel diagonally.

Hint 2: Your algorithm CAN move into a wall or furniture spot, but upon recognizing it as uncountable, it should return from that call.

Hint 3: It may help to track the distance traveled so far from the start.

Part 2 - Programming: (12 points)

Write a RECURSIVE program in C that implements your recursive algorithm. The program must compile and run on the fox servers. Use valgrind on your program to try to find all possible defects before submitting.

Hint 1: chars can be treated like small valued integers. In particular, it may be helpful to use '+' and '<=' with your chars.

Your program must output the base room first, and then the updated room when done, like so:

```
Base room:                         Room after algorithm:
**************                     **************
* A          *                     *BABCDEFGHIJK*
*    *       *                     *CBCD*FGHIJKL*
*    *       *                     *DCDE*GHIJKLM*
*  ***       *                     *ED***HIJKLMN*
*            *                     *FEFGHIJKLMNO*
*     *****  *                     *GFGHI*****OP*
*     ***    *                     *HGHIJ***RQPQ*
**************                     **************
```

Part 3 - Runtime: (4 points)

What is the runtime complexity of your algorithm? The are multiple reasonable answers and it depends on your code so be sure to justify your reasoning.

Deliverables:

Your program solution should be submitted as “answer3.c” (also submit any additional files you modified/created to solve this problem). Your solutions to the planning and runtime parts should be submitted as a single text file (.txt, .pdf, or .doc).

Upload these files to Blackboard under Assignment 3. **Do not zip your files**.

To receive full credit, your code must compile and execute. You should use valgrind to ensure that you do not have any memory leaks.

Remember:

The program you submit should be the work of only you and one other partner. Cheating will be reported to SCCS. Both the copier and copiee will be held responsible.