CS2123 Data Structures
Assignment 3 - Recursion

Description

Jethro and Cletus 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 period (i.e. '.') can be moved through. An octothorpe (i.e. '#') 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
```

Here's some example rooms:

Example 1:

```
Base room:                          Room after algorithm:
#####                               #####
#A..#                               #ABC#
###.#                               ###D#
#...#                               #GFE#
#####                               #####
```

(We can't pass through the '#' symbols)

Example 2:

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

Example 3:

```
Base room:                          Room after algorithm:
#################################   #################################
#...A..........................#   #DCBABCDEFGHIJKLMNOPQRSTUVWXYZZZZ#
#################################   #################################
```

(We don't count past 'Z' since Jethro has a pretty small apartment)

Example 4:

```
Base room:                          Room after algorithm:
#######                             #######
#A....#                             #ABCDE#
####..#                             ####EF#
#..#..#                             #..#FG#
####..#                             ####GH#
#.....#                             #KJIHI#
#######                             #######
```

(Unreachable '.' areas should remain unchanged)

Part 1 - Programming: (15 points)

Write a brute force **recursive** program to solve the problem described above in *PaintRoom.java*.

Hint 1: This program will take fewer lines of code than our other assignments but you'll need to think about and test them carefully. For reference, my *recPaintRoom* only had 9 lines of code in it.

Hint 2: From any given space, you need to try to continue moving up, down, left, and right (i.e. 4 recursive calls).

Hint 3: Your algorithm can move into a '#', but upon recognizing it as an obstacle, it should return from that call.

Hint 4: It may help to track the distance traveled so far from the starting 'A' character.

Hint 5: Only updating locations that contain a '.' won't be enough. Sometimes you'll need to update a location you previously visited and labeled.

Hint 6: chars can be treated like small valued integers. In particular, it may be helpful to use operations like + and <= with your chars. For example, (char)('A'+2) will evaluate to 'C'.

Part 2 - Runtime: (5 points)

What is the runtime complexity of your algorithm? There are multiple reasonable answers and the answer depends on your code so you should also **justify your reasoning** in order to receive credit.

Deliverables:

Your program solution should be submitted as "paintRoom.java" (also submit any additional files you created to solve this problem). Your solutions to part 2 should be submitted as a 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.