CS211 – Programming Practicum Fall 2015

Programming Project 5

Due: Friday, 11/6/14 at 11:59 pm

Maze Solving

For this lab, write a Java program that will find its way through a maze using the depth-first search algorithm. This program takes input from a file specified in the command line argument that contains two integer values per line of input:

- The first valid line gives the size of the 2-D maze (the number of rows given by the first number, then the number of columns given by the second number), valid values are >= 1
- The second valid line gives the coordinates of the starting position in the maze
- The third valid line gives the coordinates of the ending position in the maze
- The remaining valid lines in the file give the coordinates of blocked positions in the maze

The following shows an example of such an input file. The coordinates are given with the row listed first and the column listed second. A maze of NxM has rows numbered from 1 to N and columns number from 1 to M.

```
10 20
1 1
10 20
5 1
4 2
3 3
1 10
2 9
3 8
4 7
5 6
6 5
7 4
8 3
```

This input creates the following maze will 11 blocked positions:

```
size: 10, 20
start: 1, 1
end: 10, 20
**********************
*s........*..........*
*........*...........*
*..*....*............*
*.*....*.............*
**....*..............*
*....*...............*
*...*................*
*..*.................*
*....................*
*...................e*
**********************
```

The blocked positions and the edges of the above maze are filled in with *'s. The start position is filled in with a 's'. The end position in filled in with a 'e'. The other positions are filled in with periods.

You may assume that the input will always have two integer values per line; however, the values may be out of range. If an invalid value is given on an input line, print a descriptive error message to STANDARD ERROR (using something along the lines of System.err.print() or System.err.println()), ignore those two input values and continue processing input using the next line of input. Any value of zero or less is invalid in this program. For a coordinate value in the maze, valid value range from 1 to the maximum row size or column size. Also, the starting and ending positions of the maze must never be blocked. Since errors may exist:

- The size of the maze is on the first valid line of input.
- The starting position of the maze is the second valid line of input.
- The ending position of the maze is the third valid line of input.

Input with invalid values is shown below. The comments are each line are not part of the input.

```
10 0            => Invalid: Maze sizes must be greater than 0
15 7            => Maze becomes size 15 x 7
10 20           => Invalid: column 20 is outside range from 1 to 7
5 1             => Starting position is at position 5, 1
24 2            => Invalid: row 24 is outside of range from 1 to 15
3 3             => Ending position is at position 3, 3
1 10            => Invalid: column 10 is outside range from 1 to 7
2 9             => Invalid: column 9 is outside range from 1 to 7
3 8             => Invalid: column 8 is outside range from 1 to 7
4 7
5 6
5 1             => Invalid: attempting to block starting position
6 5
7 4
8 3
```

The algorithm you are to use to find a path through the maze is a Depth First Search. You MUST use the following form Depth First Search. You are NOT allowed to use a recursive version of the depth first search (since you are required to use your own stack to solve this).

- Mark all unblocked positions in the maze as "UNVISITED"
- push the start position's coordinates on the stack
- mark the start position as visited
- While (stack is not empty and end has not been found)
  - if the coordinate at the Top of the Stack is the end position
    - then end has been found
  - if the coordinate at the Top of the Stack has an unvisited (and unblocked) neighbor
    - push the coordinates of the unvisited neighbor on the stack
    - mark the unvisited neighbor as visited
  - else
    - pop the coordinate at the Top of the Stack
- If the stack is empty
  - The maze has no solution

- else
    - The items on the stack contain the coordinates of the solution from the end of the maze to the start of the maze.

When referring to neighbors, those positions will be the ones above, below, left or right of the current position (not diagonal). So for position x,y its neighbors are at:

- x+1, y
- x-1, y
- x, y+1
- x, y-1

Program Output

You will find a class GridDisplay in the file GD2.java. This class will allow you to display a maze (2-D grid) of any size. You can also place any character at any maze/grid position and color any maze/grid position. The GridDisplay API (Application Program Interface) is as follows:

- public GridDisplay(int rows, int cols)
  This constructor will take two integers which will set up the rows and columns for the grid. Each grid position will initially display the space character and will be WHITE.

- public void setChar (int row, int col, char c)
  This method will display the character given in the third parameter as the grid position specified by the row and column parameter values given.

- public void setChar (int row, int col, Color c)
  This method will display the color given in the third parameter as the grid position specified by the row and column parameter values given.

The file GD2.java also contains the following static method which will cause the code to sleep for the number of milliseconds specified by the parameter. This method will allow the changes made to the GridDisplay grid to appear as animation. This is needed since without it, the screen will get updated too fast so that no one can see which changes occurred when.

- public static void mySleep(int milliseconds)

Your program is to use an instance of the GridDisplay class to provide the output for the program. Once/as the data has been read in, the maze should be created and displayed using this GridDisplay instance. The blocked positions (and the external walls) should be displayed in some predetermined color. The start and end positions should also be displayed in some different color (or colors).

As the path through the maze is explored, the GridDisplay instance should color in the path at each position with one color as the grid positions are pushed onto the stack. Anytime the grid positions are popped from the stack, the position should be colored with a different color showing that those positions were tried but discarded. Perhaps using Color.GREEN when pushing positions onto the stack and Color.LIGHT_GRAY when popping positions from the stack.

At the end of the program the display MUST show the colored path chosen by the depth first algorithm from the start position to the end position and/or the paths tried and discarded. You may add character information to the path if you wish but this is not required.

Note that the GridDisplay class is to only be used the output for your program. Your program MUST check for invalid input.

Once the maze solving algorithm is run, you must then print out a message to standard output stating either:

- the maze has no solution

or

- listing the coordinates of the locations of the path in the maze from the start of the maze to the end of the maze that was found by the algorithm. Note this means printing out the contents of the stack in reverse order.

Internal Storage Structure

Your program MUST use a dynamic 2-D array to create a back-end storage structure for the array. A 2-D array in Java is created by creating an array of arrays. This is described in:
http://www.willamette.edu/~gorr/classes/cs231/lectures/chapter9/arrays2d.htm
There are many other places to find out about 2-D arrays (however, the Head First Java text did decide to only briefly discuss them).

The stack may use a linked list of coordinate values or it may use a dynamic array of coordinate values. You must write the code for the linked list or dynamic array yourself. **YOU MAY NOT USE ONE OF THE JAVA COLLECTIONS FRAMEWORK CLASSES FOR THIS.** The Java Foundation Classes includes ArrayList, Set, Vector, Stack, HashSet, HashMap, LinkedList and many others. A dynamic array may be the easier than a linked list.

The stack variable MUST NOT be global. It must be declared as a local variable in main( ) or some other function. You may create a class to contain the stack if you desire but again the initial instance of the class must be a local variable.

Program Submission

You are to submit the programs for this lab via the Assignments Page in Blackboard.

To help the TA, name your file with your net-id and the assignment name, like:

- ptroy1LabX.java