Programming Assignment #6: Maze Creation

COP 3503, Fall 2013

Due: Sunday, November 17, 11:59 PM via Webcourses@UCF

Abstract

In this assignment, you will generate an ASCII maze. In the process, you will solidify your understanding of disjoint sets. It will be fun.

Perhaps the most difficult aspect of this assignment will be developing a a numerical indexing of walls and cells in your array, and coming up with mathematical transformations that tell you which cells are separated by which walls.

If you use any code that I have given you so far in class, you should probably include a comment to give me credit. The intellectually curious student will, of course, try to write the whole program from scratch.

Deliverables

Maze.java

1. Problem Statement

Generate an $m \times n$ maze (where m is *width* and n is *height*) using the following algorithm:

1. initialize the maze such that all $m \times n$ cells are surrounded by walls on all four sides
2. while (not all cells belong to the same set)
3. randomly select an inner wall, w
4. if (w divides two cells, x and y, that are in disjoint sets)
5. remove w
6. union(x, y)
7. add start ('s') and exit ('e') markers

In step (3), you should never consider the same wall more than once. (How can you avoid that?)

2. Maze Representation

Your maze will be represented using a 2D char array. The '#' character will represent a wall. The top-left cell will be the start position, represented by a lower-case 's' character. The bottom-right cell will be the end position (or exit), and will be represented with the 'e' character. All other non-wall spaces will be represented using a single space character.

Initially, the O($m \times n$) array contains $m \times n$ individual cells, each of which is surrounded by walls on all sides. Non-cell gaps created by this representation should also be filled in with walls ('#'). For example, here is a freshly initialized 6 × 3 grid. Notice there are initially 6 × 3 = 18 cells:

```
#############
# # # # # # #
#############
# # # # # # #
#############
# # # # # # #
#############
```

After running the maze generation algorithm describe above, one potential output is:

```
#############
#s    # # # #
### ### # # #
# #         #
# ### # # # #
#     # # #e#
#############
```

Note: None of the red cells in the diagram below can *ever* be removed from the graph, because they do not directly separate any of the *original* 18 cells:

```
#############
# # # # # # #
#############
# # # # # # #
#############
# # # # # # #
#############
```

3. Method and Class Requirements

Implement the following methods in a class named `Maze`. Please note that they are all **`static`**. You may implement helper methods as you see fit.

`public static char [][]` `create(int width, int height)`

Given positive integers *width* and *height*, create a *width* × *height* maze using the disjoint sets approach described above. Use the path compression and union-by-rank optimizations. Return a 2D char array that contains the new maze. Do not print anything to the screen.

`public static double` `difficultyRating()`

Return a double on the range 1.0 (ridiculously easy) to 5.0 (insanely difficult).

`public static double` `hoursSpent()`

Return an estimate (greater than zero) of the number of hours you spent on this assignment.

4. Grading Criteria and Miscellaneous Requirements

The *tentative* scoring breakdown (not set in stone) for this programming assignment is:

80%	program passes test cases and uses disjoint sets to do it
10%	`difficultyRating()` and `hoursSpent()` return doubles in the specified ranges
10%	adequate comments and whitespace

Programs that do not compile will receive zero credit. Please be sure to submit your `.java` file, not a `.class` file (and certainly not a `.doc` or `.pdf` file). Your best bet is to submit your program in advance of the deadline, then download the source code from Webcourses, re-compile, and re-test your code in order to ensure that you uploaded the correct version of your source code.

NEW! Please remove `main()` before submitting. A lot of `main()` methods are causing compilation issues because they include references to home-brewed classes that are not submitted with the assignment. Please remove.

Your program should not print anything to the screen. Extraneous output is disruptive to the TAs' grading process and will result in severe point deductions. Please do not print to the screen.

Please do not create a java `package`. Articulating a `package` in your source code could be disruptive to the grading process and will result in severe point deductions.

Name your source file, class(es), and method(s) correctly. Minor errors in spelling and/or capitalization could be hugely disruptive to the grading process and may result in severe point deductions. Please double check your work!

Test your code thoroughly. Please be sure to create your own test cases and thoroughly test your code.

Start early! Work hard! Ask questions! Good luck!