COMS 3134 Homework 6

Submission instructions

All programs must compile and run on CUNIX to receive credit. Submit your electronic files via `http://courseworks.columbia.edu`. We prefer electronic submission of theory, though you will not be penalized for paper submissions. (Do **not** print out your programs.) I recommend learning LaTeX for typesetting math. Include a README file that explains exactly what each file in your submission is. Place all the files you want to submit into a submission directory with the following naming scheme.

`<your_uni>_hw<number>`

So if my UNI is `uni1234` am submitting homework 6, my directory would be `uni1234_hw6`. Archive your submission directory using

```
tar -czvf uni1234_hw6.tar.gz uni1234_hw6
```

and upload `uni1234_hw6.tar.gz` to courseworks. (You will probably need to first download the file to a local directory using an FTP program. See CUNIX tutorial for more info.)

Multiple Submissions: You can submit multiple times, but we will only consider the latest submission based on the timestamp in courseworks. Please give at least 1-2 minutes between two submissions so we can tell which is the newest submission.

Keep a pristine copy of your submission folder in case there is any submission error.

Theory Problems

Make sure your solutions are clear. Diagrams and math are often insufficient to convey exactly what you mean, so supplement with some text. Either pseudocode or Java are acceptable when asked to provide algorithms. Nevertheless, clear, concise English is often preferable.

1. (4 points) **Weiss 7.4** Show the result of running Shellsort on the input 9, 8, 7, 6, 5, 4, 3, 2, 1 using the increments $\{1, 3, 7\}$. [Obviously, you will be graded on intermediary steps, so show your work; i.e., don't just submit 1 2 3 4 5 6 7 8 9]

2. **Weiss 7.17 a and b** Determine the running time of mergesort for

   (a) (2 points) sorted input

   (b) (2 points) reverse-ordered input

3. (4 points) **Weiss 7.19** Sort 3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5 using quicksort with median-of-three partitioning and a cutoff of 3. [Show intermediary steps and use the convention that you choose the first, middle, and last elements for the median-of-three pivot.]

4. (4 points) **Weiss 7.38** Suppose arrays A and B are both sorted and both contain N elements. Give an $O(\log N)$ algorithm to find the median of $A \cup B$.

5. **Weiss 8.1 b and c** Show the result of the following sequence of instructions: `union(1,2)`, `union(3,4)`, `union(3,5)`, `union(1,7)`, `union(3,6)`, `union(8,9)`, `union(1,8)`, `union(3,10)`, `union(3,11)`, `union(3,12)`, `union(3,13)`, `union(14,15)`, `union(16,0)`, `union(14,16)`, `union(1,3)`, `union(1,14)` when the unions are:

   (a) (0 points; **Don't do this**) Performed arbitrarily

   (b) (2 points) Performed by height

   (c) (2 points) Performed by size

   [Draw the trees, not the arrays]

6. (2 points each) **Weiss 8.2** For each of the trees [just b and c] in the previous exercise, perform a `find` with path compression on the deepest node.

7. (4 points) **Weiss 8.6** Prove that for the mazes generated in Section 8.7, the path from the starting point to ending points is unique.

8. (4 points) Describe an algorithm to solve the Hamiltonian Path Problem on a **non-deterministic** computer in polynomial time. (Note that if you are able to do this for a deterministic computer, you will win the Millennium Prize from the Clay Mathematics Institute).

Programming (30 points)

Based on Weiss 8.15 Write a program that generates mazes of arbitrary size. Generate a maze similar to that in Figure 8.19.

- Your program should be called with `java MazeGenerator <rows> <columns>`, which will build and display a maze of size `<rows>` by `<columns>`.
- Use the `DisjSet.java` class provided by the textbook (or write your own if you prefer).
- One way to interpret the maze-building algorithm is to think of an initial grid-graph such that each square in the maze is a node and each node is connected to its north, south, east and west neighbors. Then build a *random* spanning tree, such that there is some path from the start to the end. However, since your program will just be building a maze, it is up to you if you want to write code in terms of graphs or specifically for the maze.
- The provided file `Maze.java` provides an easy interface to create a maze and remove walls, with an ASCII output of the maze (via a `toString()` call). Feel free to modify this file.
- Your code should be able to generate mazes of up to a thousand rows and columns in a few seconds. You can save the output of a program to a file using the '>' character in cunix

  ```
  java MazeGenerator 1000 1000 > giantMaze.txt
  ```

  Or you can write a file output routine in your program for large mazes.

10 points extra credit Maze solving. Add a function that solves the maze. Using depth first search starting from the start square, find the path to the end square and display the path in a GUI output of the maze. You can use the `drawPath` method in `Maze.java` to draw the path.