Homework 5

CSC 140 – Advanced Algorithm Design and Analysis

Follow the "homework procedures" found on Piazza to submit files to DBInbox. If anything in this assignment does not make sense, please ask for help.

Quiz: We will have a closed-note 20-30 minute quiz on this homework in class Fri Mar 29.

Non-submitted work:

Read: CLRS 17 through Section 17.3, Chapter 19 through Section 19.2, and Chapter 21 through Section 21.3.

Written Problems:

There are three steps to follow for the written problems. Step 1: Do them as if they were homework assigned to be graded (ie, take them seriously and try to do a good job). Step 2: Self-assess your work by comparing your solutions to the solutions provided by me. Step 3: Revise your original attempts as little as possible to make them correct. Submit both your original attempt and your revision as separate files.

Submit two files to DBInbox following the procedure documented on Piazza. The first file should be named exactly **hw5.pdf** and should be your homework solutions before looking at my solutions. The second file should be named exactly **hw5revised.pdf** and should be your homework solutions after looking at my solutions and revising your answers.

NOTE: If you wish to handwrite your solutions you may, but only if your handwriting is very *easy to read and the file you submit is less than 1MB in size. Also, part of your homework grade may be based on my subjective opinion of how seriously you take this process.*

1) Do CLRS Exercises 17.1-3, 17.2-2, 17.3-2, 17.3-6 (use potential method), 21.3-1.

2) If Fib-Heap-Extract-Min were called on the heap at Figure 19.5a, what would the resulting data structure be? Do this carefully, following the algorithms and figures of the chapter. You can ignore all marking, that's only used by algorithms we did not discuss in class.

Programming:

Due: The program(s) may be first graded sometime – maybe hours, maybe days – after the quiz.

A) One way to create a random undirected simple graph of k vertices is following this pseudocode.

```
V = {0,...,k-1}
E = { }
for i = 0..k-2
   for j = i+1..k-1
      if (random double in range [0,1) < p)
          E = E union {(i,j)}
```

In other words, each possible edge has a p probability of being in the graph.

Through simulation, determine what value of p makes Same-Component return true on a random graph of 1,000 vertices for about 50% of (i, j) pairs where $i \neq j$. Note that you do not need to actually create a graph to answer this question, you simply need to enumerate all possible edges in an imaginary graph, randomly decide whether each edge exists in the imaginary graph, and use the connected-components algorithm from Section 21.1 of CLRS to keep track of which vertices are in which components.

To find your result, implement a disjoint set data structure (with union-by-rank and path compression), and implement the connected-components algorithm from Section 21.1 of CLRS. Then use the pseudocode above as a guide for building your connected components. Once you've done that, enumerate all (i, j) pairs to find precisely what fraction of vertices are in the same component. Repeat the experiment for various p values until you find your answer. Running your program should do the following: output the p value being used, (ii) run the simulation using the p value, and (iii) output the fraction of (i, j) pairs that your simulation found to be in the same component. Your program should output nothing else but these two values.

Submit to DBInbox one file named **hw5.c** or **Hw5.java**. It should be possible to compile C programs with `gcc -Wextra -Wall -std=c99 hw5.c` or Java programs with `javac Hw5.java` and the command should generate an executable without displaying any warnings. Running the executable should produce the output described above. C programs should not include any headers other than standard C headers (`https://en.cppreference.com/w/c/header`). Java programs should not use package directives.

All submitted code should include a comment at the beginning with your name, 4-digit code, and brief explanation of the what's in the file. Furthermore, any hard-to-understand code should have local comments to help the reader understand.