EC330 Applied Algorithms and Data Structures for Engineers
Spring 2022

Homework 4

Out: March 15, 2022
Due: March 25, 2022

This homework has a written part and a programming part. Both are due at 11:59 pm on March 25. You should submit both parts on Gradescope.

This is an individual assignment. See course syllabus for policy on collaboration.

1. **Heap [10 pt]**
   Given any eight numbers $n_1, n_2, \dots, n_8$, show that you can construct a max heap of these numbers using only eight pairwise comparisons (e.g. comparing n_1 and n_2 would be one comparison).

2. **Hashing [20 pt]**
   Given a sequence of inputs 631, 23, 33, 19, 44, 195, 64 and the hash function $h(x) = x \bmod 10$, show the resulting hash table for each of the following cases of conflict resolution.
   (a) Chaining
   (b) Linear probing with $h_i(x) = [h(x) + i] \bmod 10$, for $i = 0, 1, 2, \dots, 9$
   (c) Quadratic probing with $h_i(x) = [h(x) + i^2 + i] \bmod 10$, for $i = 0, 1, 2, \dots, 9$
   (d) Using a second hash function $h'(x) = 7 - (x \bmod 7)$ as the step function during probing, i.e. $H_i(x) = [h(x) + i*h'(x)] \bmod 10$, for $i = 0, 1, 2, \dots, 9$

3. **Programming [70 pt]**
   a) In this problem, you are tasked with sorting a string in *increasing* order based on the number of occurrences of characters. If there is a tie, output them based on *alphabetical order*, e.g., 'a' before 'e'. You can assume that all the characters are lower-case letters (so a total of 26 possible types of characters). Below are some example inputs and the corresponding expected outputs.

      Input1: "engineers"
      Output2: "girsnneee"

      Input2: "engineering"
      Output2: "rggiieeennn"

      Implement *sortByFreq* in *sorta.cpp*. You are allowed to use *only* the libraries included in *sort.h* (you may not need to use all of them). You are welcome to implement your own data structure or use built-in ones like *int[]*. You *cannot* use any of the built-in sort functions including those provided by the <*algorithm*> library and will *need to implement your own* (you should write this as a separate function and call it from *sortByFreq*). Submit *sorta.cpp* on Gradescope. **[20 pt]**

Your code will be graded not only on correctness but also on efficiency. For efficiency consideration, you should expect long string inputs (i.e. large *n*). **[10 pt]**

b) We will implement a Bloom filter to check active phishing URLs in this problem. The files *phishing-links-ACTIVE.txt* and *phishing-links-INACTIVE.txt* contain two sets of phishing URLs obtained from https://github.com/mitchellkrogza/Phishing.Database. Each line in these two files is a URL. We will design a Bloom filter so that the Bloom filter contains the set of ACTIVE phishing links, and the false positive rate for classifying an INACTIVE phishing link as ACTIVE is minimized. The file *phishing-links-INACTIVE.txt* contains 10% of all the INACTIVE links that we will use to test your implementation. The size of the Bloom filter is set to 330. Your Bloom filter implementation should have *a false positive rate strictly less than 30%*. Submit *bfilter.cpp* on Gradescope. **[40 pt]**

The submission(s) that has the lowest false positive rate will get **10 bonus points**.

Below are some resources for hash functions that you can use. You are free to use other hash functions.
http://www.partow.net/programming/hashfunctions/index.html#AvailableHashFunctions
https://en.wikipedia.org/wiki/Double_hashing
http://hashlib2plus.sourceforge.net