EC330 Applied Algorithms and Data Structures for Engineers
Spring 2020

Homework 5

Out: March 23, 2020
Due: April 1, 2020

This homework has a written part and a programming part. Both are due at 11:59 pm on April 1. You should submit both parts on Gradescope.
For the written part, you should submit a single PDF file containing either typeset answers or scanned copies of hand-written answers. Make sure you write your answers clearly.
For the programming part, you should make sure your code compile and run on Gradescope. You should also try to create your own test cases to test and debug your code. Make sure you write your name and BU ID in a comment at the top of the program, and use clear comments to explain the key steps in your code.

1. **Heap [8 pt]**
   Given eight numbers $\{n_1, n_2, \ldots, n_8\}$, show that you can construct a heap (either min-heap or max-heap) using eight comparisons between these numbers (e.g., comparing n_1 and n_2 would be one comparison).

2. **Hashing [32 pt]**
   Given a sequence of inputs $17, 423, 64, 79, 411, 89, 99, 15$ and the hash function $h'(x) = x \bmod 13$, show the resulting hash table for each of the following cases of conflict resolution.
   (a) Chaining
   (b) Linear probing ($h_i(x) = [h'(x) + i] \bmod 13$)
   (c) Quadratic probing ($h_i(x) = [h'(x) + i^2 + i] \bmod 13$)
   (d) Using a second hash function $h_2(x) = 7 - (x \bmod 7)$ for the step function

3. **Programming [60 pt]**
   NOTE: You should implement your solution in the provided hw5.cpp file. You only need to submit this file on Gradescope.

   - Given a binary min-heap, implement the *findMinK* method in *hw5.cpp* to find the *k* smallest elements in the heap. The elements returned should be in increasing order (smallest element first). You are not allowed to modify the heap or copy the heap to another data structure for this problem (e.g. copy the content of the heap to an array and sort the array). **[20 pt]**
     Hint: You can use a mini min-heap to keep track of the current candidates for the next smallest element.

- 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. **[40 pt]**
  For full credit, your Bloom filter implementation should have a false positive rate strictly less than 30%.
  The submission 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