CS 630, Fall 2024, Homework 6 Solutions
Due Wednesday, November 13, 2024, 11:59 pm EST, via Gradescope

Homework Guidelines

Collaboration policy Collaboration on homework problems, with the exception of programming assignments and reading quizzes, is permitted, but not encouraged. If you choose to collaborate on some problems, you are allowed to discuss each problem with at most 5 other students currently enrolled in the class. Before working with others on a problem, you should think about it yourself for at least 45 minutes. Finding answers to problems on the Web or from other outside sources (including generative AI tools or anyone not enrolled in the class) is strictly forbidden.

You must write up each problem solution by yourself without assistance, even if you collaborate with others to solve the problem. You must also identify your collaborators. If you did not work with anyone, you should write "Collaborators: none." It is a violation of this policy to submit a problem solution that you cannot orally explain to an instructor or TA.

Typesetting Solutions should be typed and submitted as a PDF file on Gradescope. You may use any program you like to type your solutions. LaTeX, or "Latex", is commonly used for technical writing (`overleaf.com` is a free web-based platform for writing in Latex) since it handles math very well. Word, Google Docs, Markdown or other software are also fine.

Solution guidelines For problems that require you to provide an algorithm, you must provide:

1. pseudocode and, if helpful, a precise description of the algorithm in English. As always, pseudocode should include
   - A clear description of the inputs and outputs
   - Any assumptions you are making about the input (format, for example)
   - Instructions that are clear enough that a classmate who hasn't thought about the problem yet would understand how to turn them into working code. Inputs and outputs of any subroutines should be clear, data structures should be explained, etc.

   If the algorithm is not clear enough for graders to understand easily, it may not be graded.
2. a proof of correctness,
3. an analysis of running time and space.

You may use algorithms from class as subroutines. You may also use facts that we proved in class.

You should be as clear and concise as possible in your write-up of solutions. A simple, direct analysis is worth more points than a convoluted one, both because it is simpler and less prone to error and because it is easier to read and understand.

Problem 1 *Bloom Filter Transformations (5 points)*

The Bloom filter B is of size m (number of bits), where m is a power of 2. So far n items have been inserted into B.

In this problem we create a new Bloom filter B' that is half the size of B, thus has $m/2$ bits. We "copy" the current content of B to B' in the following way: we denote the first $m/2$ bits of B by B_1, the second $m/2$ bits is denoted by B_2. Then we take the bit-wise Boolean *or* (inclusive-or) of B_1 and B_2, thus $B' = B_1 \vee B_2$.

Suppose $h_1(x), \ldots h_r(x)$ are the hash functions originally used by B. We will use the same functions for B' with the modification that $h_i(x)$ *mod* $(m/2)$ for each i.

1. Show that for every element $x \in B$ the look up in B' will return positive.
2. Compute and explain the false negative and false positive rates in B' for items that
   (a) were in B prior to creating B'
   (b) were not in B
   (c) $O(n)$ more items are inserted into B' what are the false negative and false positive rates now?

Problem 2 *Hash Evaluations for Bloom Filters. (5 points)*

In this problem, you will be asked to analyze the number of invocations of the hash function of a Bloom filter for different queries. The Bloom filter has m bits and n items were inserted using $k = \lfloor \ln 2(m/n) \rfloor$ hash functions.

1. How many times are the hash functions invoked when inserting an item into the Bloom filter?
2. What is the average number of invocations of the hash functions when checking an item that was inserted into the Bloom filter?
3. What is the average number of invocations of the hash functions when checking an item that was not inserted into the Bloom filter? *Hint: do not forget that the check can stop at the first zero bit found.*

Problem 3 *Exploiting Weak Hash Functions (10 points)*

In this problem, you will be given "blackbox" access to a hash function used by a service. With this blackbox, you may invoke the hash function h with input x and the output bucket index $h(x)$ of the input will be returned. All you know about the hash function $h(x)$ is that it takes in integer inputs and has the form $h(x) = ax + b \pmod{2^k}$ where a, b, and k are unknown non-negative integers. To be clear, you know that the result you get is really just the hash function, and is not affected by collisions in the table.

1. Write and justify an algorithm to determine a, b assuming $k = 64$ with as few invocations of $h(x)$ as possible. To avoid ambiguity, return the smallest non-negative values for a and b that work.

2. Later, you are hired as a consultant by this company after they notice weird performance of their hash table. Their architecture uses hashing with separate chaining, and the same linear hash function that you previously analyzed. Upon inspection, you note that only ~~half~~ the buckets with odd indexes have any data, so those odd buckets have twice the expected load factor. And after further testing with many random keys, you are unable to find any items that are into even buckets. What can you infer about the current hash function $h(x) = ax + b \pmod{2^k}$? How should we modify a and b to avoid this problem?

3. The company has noticed that 1/1024 of the buckets have significantly more load than average. They suspect that an adversary – who knows the structure of the hash function – is deliberately sending data to cause bucket collisions.

   Identify a set of keys that could be chosen to cluster in 1/1024 of the keys only knowing that the hash function the form $h(x) = ax + b \pmod{2^k}$ where a, b, and k are unknown non-negative integers.

   After identifying the problem, suggest a minimal change to the choice of the hash function design to prevent this problem recurring.