Problem Set 4

This problem set is due **at 10:00pm** on **Wednesday, March 4, 2020**.

Please make note of the following instructions:

- This assignment, like later assignments, consists of *exercises* and *problems*. **Hand in solutions to the problems only.** However, we strongly advise that you work out the exercises for yourself, since they will help you learn the course material. You are responsible for the material they cover.
- Remember that the problem set must be submitted on Gradescope. If you haven't done so already, please signup for 6.046 Spring 2020 on Gradescope, with the entry code MNEBKP, to submit this assignment.
- We require that the solution to the problems is submitted as a PDF file, **typeset on LaTeX**, using the template available in the course materials. Each submitted solution should start with your name, the course number, the problem number, your recitation section, the date, and the names of any students with whom you collaborated.
- You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of your essay should provide the following:
  1. A description of the algorithm in English and, if helpful, pseudocode.
  2. A proof (or indication) of the correctness of the algorithm.
  3. An analysis of the asymptotic running time behavior of the algorithm.
  4. Optionally, you may find it useful to include a worked example or diagram to show more precisely how your algorithm works.

EXERCISES (NOT TO BE TURNED IN)

Hashing

- Do Exercise 11.4-1 in CLRS (pg. 277)
- Do Exercise 11.4-2 in CLRS (pg. 277)
- Do Exercise 11-3 in CLRS (pg. 283)
- Do Exercise 11-4 in CLRS (pg. 284)

Problem 4-1. Ben's 2-Hash Method [60 points]
Ben Bitdiddle hears about perfect hashing and thinks that 2-level hashing is too complex. Instead, he has an idea to use one level of hashing but with two hash functions. For this problem, assume that Ben can generate hash functions which are completely random. That is, over $h \in \mathcal{H}$, for all n, and for any distinct $x_1, x_2, \ldots, x_n$, the probability distribution of $\langle h(x_1), h(x_2), \ldots, h(x_n)\rangle$ is uniform over $\{0, \ldots, m-1\}^n$.

Ben uses the two hash functions h_1 and h_2 in order to place items into m slots. With two hash functions, each item maps to two slots (or one if the two functions map a key to the same slot). Ben wants to know when it is possible to place each item into one of its two possible slots without collisions between items. If he can do this, Ben knows that if an item x_i is in the hash table, it must be in one of slots $h_1(x_i)$ or $h_2(x_i)$.

Ben can represent this as a random graph G, where there are m vertices numbered 0 to $m-1$ corresponding to the slots in the hash tables and n edges corresponding to the pair of hashed values for an item (with possible self-loops). In this representation, edge e_i has endpoints at vertices $h_1(x_i)$ and $h_2(x_i)$. Note that a given edge (u, v) is defined by key k if either $h_1(k) = u$ and $h_2(k) = v$ or $h_1(k) = v$ and $h_2(k) = u$.

(a) [15 points] Show that, if there are no cycles in G, then Ben can find a *collision-free* assignment that assign each of the n items to the slot of one of its hashed values without collisions. Design and analyze an algorithm A such that i) if G does not have a cycle, then A finds a collision-free assignment, or ii) if G has a cycle, the A outputs "CYCLE". For full credit, A should run in $O(n)$ time.

In the next few parts, you are going to show that, for $m = 10n$, the probability of a collision is less than 1/2.

(b) [15 points] Show that the probability of a given k-cycle (a cycle of length k) appearing in G is at most $(2n/m^2)^k$. Make sure to consider the case where $k = 1$. *Hint:* For a given edge e, show that the probability that $e \in G$ is at most $\frac{2n}{m^2}$.

(c) [10 points] Show that the probability of any k-cycle appearing in G is at most $(2n/m)^k$. *Hint:* Argue that the number of k-cycles is $\frac{1}{k}\frac{m!}{(m-k)!} \leq m^k$.

(d) [10 points] Show that the probability of any cycle (of length 1 or greater) appearing in G when $m = 10n$ is no greater than 1/4.

By (a) and (d), we can conclude that, for a pair of random hash functions, the probability that Ben can assign n items to $m = 10n$ slots using the hashes is at least 3/4.

(e) [10 points] Suppose Ben can generate as many random hash functions as he wants. Using your algorithm from (a), design and analyze an algorithm which will put n items into a hash table with $m = 10n$ slots in expected $O(n)$.

Problem 4-2. Usually-Correct [30 points] Ben Bitdiddle is building a dictionary for a spell-checker, but storing all n words explicitly can take a lot of space. Thus, he has created a data structure that he thinks will work most of the time. Ben has a table A with m slots each containing a single bit. Ben picks k independent hash functions $h_1, h_2, \ldots, h_k \in \mathcal{H}, h_i : w \to \{1, \ldots, m\}$ for $1 \le i \le k$. Each word w is hashed by each of the chosen hash functions to k (not necessarily distinct) slots of A, where $k << m$.

You may assume the probability distribution of $h_i(w)$ for all $1 \le i \le k$ is uniform and independent of all hashes for all other words.

INITIALIZE(): Create an array A of m bits each initialized to 0.

INSERT(w): Set bits $A[h_1(w)], A[h_2(w)], \ldots, A[h_k(w)]$ to 1.

LOOKUP(w): Return true iff $A[h_1(w)], A[h_2(w)], \ldots, A[h_k(w)]$ are all set to 1.

(a) [4 points] Can his data structure have false negatives or false positives? Yes or no.

False Negative: If w has been inserted, then LOOKUP(w) can return false. **Yes / No**

False Positive: If w has not been inserted, then LOOKUP(w) can return true. **Yes / No**

(b) [6 points] Prove that the probability the first bit in the table is 1 after n words have been inserted is $1 - (1 - 1/m)^{kn}$.

(c) [6 points] The hash table is most informative when the probability that any given bit is 1 is 1/2. Find a simple lower bound on the number of hash functions Ben should use to get a probability of at least 1/2. You may use the fact that $1 - x \le e^{-x}$.

(d) [8 points] Prove that the probability LOOKUP(w) returns an answer inconsistent with whether w has been inserted is at most $(\frac{1}{2})^k$ when w is mapped to k **distinct** slots. *Hint: The values of the bits of A are* ***not*** *independent.*

(e) [6 points] Assume k is set as the lower bound in (c) and that $m = 20n$. When using k hash functions to hash w, the probability of having 1 or more collisions is at most $\frac{k^2}{2m}$. Given this, argue that $\left(\frac{1}{2}\right)^k + O\left(\frac{1}{n}\right)$ is an upper bound on the probability that LOOKUP fails.

Problem 4-3. Feedback Form [10 points] Please fill out a feedback form about this problem set at

`https://forms.gle/kK53C9y7kUH4R9MVA`.

It should not take more than a few minutes and will greatly help us improve teaching and material for future semesters!