CS 161: Homework 4

Due by October 28, 2016 at noon

Instructions: Please answer the following questions to the best of your ability. Provide full and rigorous proofs and include all relevant calculations. When writing proofs, please strive for clarity and brevity (in that order). Please see the course website for submission instructions and collaboration policy. Cite any sources that you use.

Remember that when you design an algorithm, in addition to an appropriate description of your algorithm, you must provide sufficient explanation for its correctness and analyze its running time. See the Homework Advice document on the course website for details.

In your submission, please start each question on a new page.

Question 1 (25 points)

In this problem, we prove that the average depth of a node in a randomly built binary search tree with n nodes is $O(\log n)$. A *randomly built binary search tree* with n nodes is one that arises from inserting the n keys in random order into an initially empty tree, where each of the $n!$ permutations of the input keys is equally likely.

Let $d(x, T)$ be the depth of node x in a binary tree T (the depth of the root is 0). Then, the average depth of a node in a binary tree T with n nodes is

$$\frac{1}{n} \sum_{x \in T} d(x, T) \ .$$

(a) (4 points) Let the *total path length* $P(T)$ of a binary tree T be defined as the sum, over all nodes x in T, of the depth of node x. We note that the average depth of a node in T with n nodes is equal to $\frac{1}{n}P(T)$. Show that $P(T) = P(T_L) + P(T_R) + n - 1$, where T_L and T_R are the left and right subtrees of T, respectively.

(b) (8 points) Let $P(n)$ be the expected total path length of a randomly built binary search tree with n nodes. Show that $P(n) = \frac{1}{n} \sum_{i=0}^{n-1} (P(i) + P(n - i - 1) + n - 1)$.

(c) (5 points) Show that $P(n) = O(n \log n)$. You may cite a result previously proven in the context of other topics covered in class.

(d) (8 points) Design a sorting algorithm based on randomly building a binary search tree. Show that its (expected) running time is $O(n \log n)$. Assume that a random permutation of n keys can be generated in time $O(n)$.

Question 2 (40 points)

As the newest recruit to the Google Trends team, you are given the task of designing a data structure to produce estimates of how many times a query (i.e. search term) appears in the query stream (this is called the *frequency* of the query). Your goal is to process a sequence of queries in one pass, storing relevant information in a data structure, so that you can later produce an estimate of the frequency of any query of interest.

This is a great time to flex your CS 161 muscles! Your first instinct is to use a big hash table, with counters for every distinct query to compute exact query frequencies. You quickly realize that the size of this hash table will have to be larger than the number of distinct queries which is prohibitively large.

How could you possibly get away with using less space? Your key insight is that you don't have to compute query frequencies exactly! All the time slogging away at CS 161 homeworks finally pays off when you design a cool new data structure to estimate frequencies of queries with low error, using very little space. In this problem, you will analyze this new method.

We use an array A of size w whose entries are initially zero, and a hash function h (chosen at random from a universal family) that maps query x to index $h(x)$ in A. We process queries in one pass. For each query x_i, we increment $A[h(x_i)]$ by 1.

(a) (5 points) Suppose query x_i has a true frequency f_i (for $1 \leq i \leq n$). For any pair of distinct x_i, x_j, define C_{ij} to be the indicator random variable $C_{ij} = 1\{h(x_i) = h(x_j)\}$. Define V_i to be a random variable where $V_i = A[h(x_i)]$ for any x_i. Express V_i in terms of the x, f, and C. All may not be used.

(b) (11 points) Let N be the total number of queries processed, $N = \sum_{i=1}^{n} f_i$. Show that $f_i \leq E[V_i] \leq f_i + N/w$.

(c) (12 points) From part (c), we see that as w gets larger, the expected value gets closer to f_i, the true frequency. Show that for any constant $c > 1$, $P(V_i \geq f_i + cN/w) \leq 1/c$, which indicates that the probability of getting a very large overestimate decreases with c. *Hint: You can use Markov's inequality.*

(d) (12 points) Now we use multiple independent copies of the simple scheme analyzed in the previous parts to get accurate frequency estimates with high probability. We use d arrays $A_1, A_2, ..., A_d$ each of length w, each with its own hash function h_k. Each h_k is chosen randomly from a universal family, independently of the other hash functions. For each query x_i, we go to each array A_k, compute $h_k(x)$ and increment the counter of its bucket in A_k for $1 \leq k \leq d$. Thus, we have d frequency estimates for each query x_i. We define V_{ik} to be the estimate of the frequency of x_i from array A_k. The final frequency estimate for query x_i is

$$U_i = \min\{V_{i1}, V_{i2}, \ldots, V_{id}\}$$

Show that $P(U_i \geq f_i + cN/w) \leq (1/c)^d$.

This can be rewritten as

$$P(U_i \geq f_i + \epsilon N) \leq \delta$$

where $\epsilon = c/w$ and $\delta = (1/c)^d$. We can think of ϵ as the "accuracy" parameter and δ as the "confidence" parameter. By lowering ϵ, we increase precision. By lowering δ, we increase the chance our answer is within the desired accuracy bound. Note that c is a constant, and w and d can be chosen to match the desired parameters ϵ, δ. We see that the space complexity is $O(wd) = O\left(\frac{log_c(1/\delta)}{\epsilon}\right)$, a function of the desired accuracy and confidence, that does not depend on the number of distinct queries!

Question 3 (35 points)

You are given a hash table of n buckets, where collisions are resolved by chaining; that is, each bucket contains a linked list of elements that were hashed to it. After inserting n elements into the hash table, let X be the maximum size of a linked list in the hash table. In this problem, we will show that $E[X] = O\left(\frac{\log n}{\log \log n}\right)$.

Assume that each element has a distinct key and the simple uniform hashing property: each element has equal probability of hashing into any of the n buckets independently of other elements. Also, you may assume that n is sufficiently large where needed (i.e., that there is some $n_0 > 0$ such that the analysis holds for $n \geq n_0$).

(a) (5 points) Given a bucket in the hash table, find Q_r, the probability that exactly r elements (out of the n elements inserted) map to that bucket.

(b) (5 points) Let $P_r = P(X = r)$. Show that $P_r \leq nQ_r$.

(c) (5 points) Use Stirling's approximation to show that

$$Q_r < \frac{e^r}{r^r}.$$

(d) (10 points) Show that there exists a constant $c > 1$ such that $Q_r < \frac{1}{n^3}$ for all $r \geq r_0 = \frac{c \log n}{\log \log n}$. Conclude that $P_r < \frac{1}{n^2}$ for $r \geq r_0 = \frac{c \log n}{\log \log n}$.

(e) (10 points) Show that $E[X] \leq P\left(X > \frac{c \log n}{\log \log n}\right) \cdot n + P\left(X \leq \frac{c \log n}{\log \log n}\right) \cdot \frac{c \log n}{\log \log n}$. Conclude that $E[X] = O\left(\frac{\log n}{\log \log n}\right)$.