Problem Set 4 Solutions

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

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.

Solution: Because items correspond to edges and slots correspond to vertices, assigning each item to a distinct slot is equivalent to assigning each edge to one of its endpoints such that no two edges are assigned to the same vertex.

If there are no cycles in G, then G is a forest. Find a root for each tree in the forest so that each edge has a parent and child vertex. Then since we have a collection of trees, each edge has a distinct child vertex. We assign each edge to its child vertex, thereby assigning each item to a different slot.

This can be implemented as follows.

1. Start off with all edges unmarked.
2. Pick an unmarked edge and make one of the vertices the root of a connected component (which is a tree if there are no cycles).
3. Run DFS from the root. If a vertex is ever revisited in the DFS, return that there is a cycle. Otherwise, for each edge traversed, mark the edge and assign the corresponding item to the slot corresponding to the child vertex of the edge.
4. Repeat steps 2-3 until all edges are marked and return the resulting assignment.

DFS runs with time linear in the number of edges in the connected component. Since we run DFS once for each connected component, the total runtime is linear in the total number of edges, or $O(n)$.

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 $\frac{2n}{m^2}$.

Solution: First we consider the case of self loops ($k = 1$). Fix a given node i with a loop. For a given key k, the probability that $h_1(k) = hj_2(k) = i$ is $\frac{1}{m}$. As there are n keys, by the union bound, the probability of the loop around node i is at most $\frac{n}{m^2}$.

Now consider a non trivial cycle of length k. For any edge (u, v) in the cycle, the edge will be in G *iff.* for some key k, either $h_1(k) = u$ and $h_2(k) = v$ or $h_1(k) = v$ and $h_2(k) = u$. This happens with probability $\frac{2}{m^2}$. As there are n keys, by the union bound, the probability that the given edge is in G is at most $\frac{2n}{m^2}$.
Now we look at all the edges in the given k-cycle. The probability that the first edge in the cycle is in G is $\frac{2n}{m^2}$. Conditioned on this, the probability that the second edge in the cycle is also in G is $\frac{2(n-1)}{m^2}$. This is because we already know that one other key was mapped to a different edge. So there are only $n - 1$ possible keys left that could be mapped to the second edge. The probability that every edge in the cycle is in G is $\prod_{i=0}^{k-1} \frac{2(n-i)}{m^2} \leq \prod_{i=0}^{k-1} \frac{2n}{m^2} = \left(\frac{2n}{m^2}\right)^k$.

(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 $\frac{1}{k}\frac{m!}{(m-k)!} \leq m^k$.

Solution: How many k-cycles are there? We have m choices for the starting node, $m - 1$ for the next one and so on and so forth. So we have $m(m - 1) \ldots (m-k+1)$ many possibilities. However, every k-cycle is counted k times (depending on which of its nodes we use as the starting node) so we need to divide by k. Therefore, we have $\frac{1}{k}m(m - 1) \ldots (m - k + 1) \leq m^k$ possible k-cycles. By (b), each occurs with probability at most $\left(\frac{2n}{m^2}\right)^k$. Then by union bound, the probability that any of these occur is at most $m^k \left(\frac{2n}{m^2}\right)^k = \left(\frac{2n}{m}\right)^k$.

(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.

Solution: By union bound, the probability a cycle exists is bounded by the sum of the probabilities that any k-cycle exists for all $1 \leq k \leq n$. Thus we sum the answer from (c) for each k from 1 to n.

$$\begin{aligned}
\sum_{k=1}^{n} \left(\frac{2n}{m}\right)^k &\leq \sum_{k=1}^{\infty} \left(\frac{2n}{m}\right)^k \\
&= \frac{2n/m}{1 - 2n/m} \\
&= \frac{2n}{m - 2n} \\
&= \frac{2n}{8n} && \text{for } m = 10n \\
&= \frac{1}{4}
\end{aligned}$$

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 $O(n)$.

Solution: We can repeatedly generate a pair of hash functions and run (a) until it is successful. Each iteration of (a) takes $O(n)$. By (d), (a) is successful with probability at least 3/4. Then it will take an expected 4/3 (or rounded to 2) iterations of (a) until we can successfully place each item without collisions. The expected runtime is twice the runtime of (a), which is still $O(n)$.

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 $1 \leq i \leq k$. 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 $1 \leq i \leq k$ 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**

Solution:
False Negative: No.
Note that, once set to 1, a given entry of A will always remain at 1. Therefore, on INSERT(w), we set all the bits $A[h_1(w)], A[h_2(w)], \ldots, A[h_k(w)]$ to 1, and they will remain 1. Therefore, LOOKUP(w) will always return true.

False Positive: Yes.
One possible situation is that we set the bits $A[h_1(w)], A[h_2(w)], \ldots, A[h_k(w)]$ to 1 when inserting other keys. So LOOKUP(w) may return true even when word w has not been inserted.

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

Solution:
For each w, the probability that none of the values $h_1(w), \ldots, h_k(w)$ equal 1 is $(1 - 1/m)^k$, independent of the hashes of the other words. By independence, the probability that none of the hashes associated with the n words map to 1 is $(1 - 1/m)^{kn}$. Therefore, the probability that some hash of some word maps to 1 is the complement of this, or $1 - (1 - 1/m)^{kn}$.

(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 $1 - x \leq e^{-x}$.

Solution: Answer: $k \geq \frac{m}{n} \ln 2$

From part (b), we know that the probability of any particular bit of A being set is $1 - (1 - 1/m)^{kn} \geq 1 - e^{-nk/m}$. We want this to be equal to $1/2$.

$$1 - e^{-nk/m} = \frac{1}{2}$$

Solving for k, we get

$$k \geq \frac{m}{n} \ln 2.$$

(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.*

Solution: Answer: $\left(\frac{1}{2}\right)^k = \left(\frac{1}{2}\right)^{\frac{m}{n} \ln 2}$

If w has been been inserted, LOOKUP(w) always returns true. There is only a false positive if w has not been inserted.

Consider the k bits of $h_i(w)$ for $1 \leq i \leq k$ when w has not been inserted. The first bit is set with probability 1/2. Conditioned on the first bit being set, the second bit is set with probability less than 1/2 because one of the bits in the n sets of k bits being set must have been used to set the first bit. Likewise, the rest of the k bits are set with probability less than 1/2 conditioned on the previous bits being set. Thus the probability that all k bits are set (corresponding with the false positive) is at most $\left(\frac{1}{2}\right)^k$ (which is also $\left(\frac{1}{2}\right)^{\frac{m}{n} \ln 2}$).

(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.

Solution: The probability that the LOOKUP fails is

$$\begin{aligned}
&\leq \Pr\left[\text{all } k \text{ slots are distinct}\right] \cdot \Pr\left[\text{error}|\text{ slots are distinct}\right] + \\
&\Pr\left[\text{there is at least one collision}\right] \cdot \Pr\left[\text{error}|\text{ there is at least one collision}\right] \\
&\leq \Pr\left[\text{all } k \text{ slots are distinct}\right] \cdot \left(\frac{1}{2}\right)^k + \Pr\left[\text{there is at least one collision}\right] \cdot 1 \\
&\leq \left(\frac{1}{2}\right)^k + \frac{k^2}{2m}
\end{aligned}$$

We know k is set to be $\frac{m}{n}\ln(2)$ and, with $m = 20n$ we find that $k = 20\ln(2)$. Since k is constant, the probability of 1 or more collisions is $\frac{k^2}{2m} = \frac{k^2}{40n} = O(\frac{1}{n})$, and we are done.

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!