Advanced Algorithms: Homework 5

Ivani Patel

November 21, 2022

1. Consider a family $\mathcal{H}$ of hash functions:

$$\mathcal{H} = \{h_i : 1 \leq i \leq 8\}.$$

Each h_i is to map an array of eight bits into its i-th component: $h_i(a_1...a_8) = a_i$. Is $\mathcal{H}$ universal? why or why not?

(a) $\mathcal{H}$ is universal, if $\forall x \neq y \in U$, Prob[h(x) = h(y)] $\leq \frac{1}{M}$, where h is a random variable on $\mathcal{H}$.

Here the family of hash function $\mathcal{H} = \{h_i : 1 \leq i \leq 8\}$. where h_i maps to an array to eight bits. In this case M is 2 as bits consists of only 0 and 1. $\mathcal{H}$ is universal if $\forall x \neq y \in U, Prob[h(x) = h(y)] \leq \frac{1}{2}$.

Now we consider two keys x = 00000000, and y = 00000001, they differ in one digit. So we have 7 hash functions mapping them into the same slot.

$$Prob[h(x) = h(y)] \leq \frac{7}{8}$$

taking another example we take two keys x = 00000000, and y = 00000011, they differ in two digit. So we have 6 hash functions mapping into the same slot.

$$Prob[h(x) = h(y)] \leq \frac{6}{8}$$

Therefore $\mathcal{H}$ is not universal.

2. Here is a classic example of universal family of hash functions. Let M be a prime number and, as usual, $[M] = \{0, 1, ..., M - 1\}$. Consider the following family of hash functions:

$$h_r(x) = (r \cdot xmodM),$$

where r, $x \in [M]^k$ (where k is a given constant like 10), and $r \cdot x = \sum_i r_i x_i$. Show that the family of hash functions (for the given k) is universal.

(a) It is given that
$h_r(x) = (r \cdot xmodM)$, where r, $x \in [M]^k$, and $r \cdot x = \sum_i r_i x_i$.
We decompose keys into k + 1 digits, x = $< x_0, x_1, x_2, ..., x_k >$, where $0 \leq x_k \leq M - 1$, and picking r = $< r_0, r_1, r_2, ..., r_k >$, where each r_k is chosen randomly from [M][1].

All $h_r(x) = (r \cdot xmodM)$ constitute a family H of hash functions. The number of hash function in H, called $\mid H \mid= M^{k+1}$

Let key $x =< x_0, x_1, ..., x_k >$, key $y =< y_0, y_1, ..., y_k >$, be distinct keys. They differ in at least one digit, without loss of generality, position 0.

If x and y collide: $h_r(x) = h_r(y)$, then we have

$$\sum_{i=0}^{k} r_i \cdot x_i modM = \sum_{i=0}^{k} r_i \cdot y_i modM$$

$$\sum_{i=0}^{k} (r_i \cdot x_i - r_i \cdot y_i) \equiv 0(modM)$$

$$r_0(x_0 - y_0) + \sum_{i=1}^{k} (r_i \cdot x_i - r_i \cdot y_i) \equiv 0(modM)$$

$$r_0(x_0 - y_0) \equiv -\sum_{i=0}^{k} (r_i \cdot x_i - r_i \cdot y_i)(modM)$$

Number theory: Let m be a prime. For any Z $\in$ integers mod M, such that Z $\neq$ 0, $\exists unique Z^{-1} \in$ integers mod M, such that $Z \cdot Z^{-1} \equiv 1(modM)$. Since $x_0 \neq y_0, \exists (x_0 - y_0)^{-1}$. Thus, we have:

$$r_0 \equiv [-\sum_{i=0}^{k} (r_i \cdot x_i - r_i \cdot y_i)](x_0 - y_0)^{-1}(modM)$$

Thus, for any choice of $r_1, r_2, ..., r_i$ exactly 1 of M choices for r_o causes x and y to collide, and no collision for other m - 1 choices for r_0. The number of $h_r(x)'s$ that cause x, y to collide = $M \cdot M \cdot M ... \cdot 1 = M^k = \frac{M^{k+1}}{M} = \frac{|H|}{M}$.

So, the family of hash function is universal.

3. So far, what we have learned about hasing is to hash an array of numbers into one number (e.g., locality sensitive hashing). Can you suggest a way to hash a graph into a number (which could be a real number)?

   (a) Lets take graph G of n nodes as input.

   Step 1: We translate the graph into Laplacian matrix L = D - A, where D is a the Degree matrix and A is the Adjacency matrix of the graph.

   Step 2: Now we calculate the eigenvalues and spectrum of the Laplacian matrix.

   Step 3: The spectrum of Laplacian matrix is always real.So we decompose the spectrum of the matrix into m + 1 digit and hash it into a number.

4. Randomized quicksort is a Las Vegas algorithm where the first step is to create a random permutation of the input arrayof numbers before the second step of running quicksort. Now, we assume that we have a high quality psuedo random generator r(n) that will generate a random number in 1..n. Please show how to generate a “random” graph with 5 nodes.

(a) **METHOD 1**

The pseudo random generator r(n) will generate a random number between 1 and n.
Total number of node pairs for n nodes = $\frac{n(n-1)}{2}$
So, total number of node pairs for 5 nodes = 10.

Let n = 10, the pseudo random generator r(10) will generate a random number between 1 and 10. Let number returned by r(n) be x.

Step 1: We construct 5 nodes.

Step 2: For each potential edge between a pair of nodes, we find value of x.

Step 3: If $x > 5$ then we add an edge between two nodes, else we do not add edge between two nodes.

Step 4: Repeat Step 2 and Step 3 for all possible pairs of nodes.

Step 5: Random graph of 5 nodes is generated.

(b) **METHOD 2**

We use the Erdos-Renyi model G(n,M), in which n represents the number of nodes and M represents the total number of edges in G. We assume that there is only one edge between two nodes.
Step 1: In this case n = 5 and $M_{max} = \frac{n*(n-1)}{2} = 10$.

Step 2: Use random generator r(n) to generate random numbers in { 0, 1, 2, ..., 10 }

Step 3: Randomly choose a pair of nodes without edge, add an edge between them and then n = n-1.

Step 4: Repeat Step 3 until n = 0. Now we generate a random graph with 5 nodes.

5. Mr. X drives on I-90 all the way from Pullman to New York (Let's assume that Pullman is Spoakne). On his car, there is a device that can suggest all the interesting places nearby that Mr. X might visit (and spend some money at these places of course). These places are stored in a set S and will be updated automatically while Mr. X is driving. Please suggest a way to implement the S so that Mr. X can query (e.g., "Is there a restrant nearby?", etc.). You shall use Bloom filter to store S. Feel free to look up papers on the Internet.

(a) Bloom filters is a probabilistic data structure that is used to test whether an element is a member of a set or not. False positive means that the query will return either possibly in set or definitely not in set. Elements can be added to the set but can not be removed.

In our problem, the interesting places to visit our stored in set S and is updated automatically when Mr X is driving through I-90. If we keep adding all the places that he passes to set S, the size of set S will become very large. To overcome this we need to remove places from set S after particular time period. Since removing elements is not possible from Bloom filter we use Forgetful Bloom filters(FBF). FBF automatically expires older items with the time period being adjustable.

FBF uses 3 bloom filters:

(1) a future Bloom filter,

(2) a present Bloom filter,

(3) a past Bloom filter.

All this filters are equal in size and identical in their use if hash function.

When an element needs to be inserted, it is first checked for membership in the FBF. If it is not present, it is inserted only into the future and present Bloom filters, but not in the past Bloom filters.

A membership check can be performed by checking if the tested element is present in at least one of the three constituent Bloom filters- id so, the check returns true. If the element is absent in all the three constituent filters, it is considered to be not present in the FBF. In this case, WHen Mr X queries a place, check this place whether present in one of the three constituent filters.

Algorithm:

Step 1: Insert the elements into the FBF. Check every place nearby Mr, X if in the FBF, if not, store this place in present Bloom filter and future Bloom filter.

Step 2: In order to forget older elements, periodically (every t times units), an FBF undergoes a refresh operation. In a basic FBF, at a refresh point, the following operations are performed automatically:

The past Bloom filter is dropped,

The current present Bloom filter is turned into the new past Bloom filter,

The current future Bloom filter is turned into the new present Bloom filter,

A new empty future Bloom filter is added to the FBF.

Step 3: A moving window FBF is created to store the places Mr. X passes while driving.

[Reference: R. Subramanyam, I. Gupta, L. M. Leslie and W. Wang, "Idempotent Distributed Counters Using a Forgetful Bloom Filter," 2015 International Conference on Cloud and Autonomic Computing, Boston, MA, 2015, pp. 113-124, doi:10.1109/ICCAC.2015.30.]

6. We know many ways to hash an array of integrs into a number. However, hash itself is loosy — that is, the function many not be one-to-one. Can you suggest a way to hash an array of 10 bits into a number such that a. the hash is one-to-one, and, b. the hash is locality sensitive (i.e., when the Hamming distance between two such arrays of 10 bits is small, then so is the distance between their hash values). (I have a terrific way to do this — but I wont tell you. You shall figure out your own ways to do this. This problem concerns a lot of fundamental applicational problems in computer science.)

   (a) Hamming distance is used to measure the similarity between two arrays. Here each array is 10 bits.

   The general family for LSH is H:h:U →S is (r_1, r_2, p_1, p_2) such that for all p, p' ∈ U. If the distance between p,p' is at most r_1, then the probability of h(p) and h(p') being equal is greater than or equal to p_1. If the distance between p,p' is greater than r_2, then the probability of h(p) and h(p') being equal is less than or equal to p_2.

   S is the set of 10-bit arrays $\{a_1, a_2...a_n\}$. Hamming distance be between s_1 and s_k then H = $[h_1 \mid h_i(a_1, a_2...a_n) = a_i forc = 1, 2, ..n]$where h_i is a function in the hash family and a_i is a bit array.

   Step 1: Find all the pairs with a small distance between p,p' based on the hash function.

   Step 2: If they have a small distance, then the collision is allowed by insertion of each bit array as an element of a linked list. This allows similar bit arrays with same hash value to be kept together. This is called chaining. This generates a hash table M.

   Step 3: Create a large hash table M' of slot size n x m and making the load factor 1 - $\frac{1}{m}$.

Step 4: Scan the current table M for each slot with and without linked list values.

Step 5: If the chained items in a slot have a consecutive distance of 1, then items are inserted into M’ using the new function.

Step 6: After placing each item into M’, we get one to one mapping of all bit arrays with mapping to hash values done based on how small their hamming distance is.