CSCI 0500: Data Structures, Algorithms, and Intractability (Fall 2025)

Assignment 3

1. (1 point) In this question, we consider another universal family of hash functions.

   Definition 1. Fix key space U and table size m. Let $\mathcal{H}$ be a family of hash functions that map U to $\{0, \ldots, m-1\}$. We say $\mathcal{H}$ is universal if, for every pair of keys $x, y \in U$ and $x \neq y$, we have $|\{h \in \mathcal{H} : h(x) = h(y)\}| \leq \frac{|\mathcal{H}|}{m}$.

   For simplicity, we assume that m is prime and $|U| = m^r$ for some positive integer r. We can view each key $k \in U$ as an r-digit number in base m: $k = (k_{r-1}, \ldots, k_0)$ where $0 \leq k_i \leq m-1$.

   Consider the following family $\mathcal{H}$ of hash functions, where $|\mathcal{H}| = m^r$:

   $$\mathcal{H} = \{h_a : a = (a_{r-1}, \ldots, a_0) \in \{0, \ldots, m-1\}^r\} \quad \text{where} \quad h_a(k) = \left(\sum_{i=0}^{r-1} a_i k_i\right) \bmod m.$$

   We will prove that $\mathcal{H}$ is universal.

   Fix any pair of keys $x, y \in U$ with $x \neq y$. We can view x and y as numbers in base m: $x = (x_{r-1}, \ldots, x_0)$ and $y = (y_{r-1}, \ldots, y_0)$. Because $x \neq y$, there is an index d with $x_d \neq y_d$.

   Your task is to prove that for every choice of $(a_i)_{i \neq d} \in \{0, \ldots, m-1\}^{r-1}$, there is exactly one choice of $a_d \in \{0, \ldots, m-1\}$ such that $h_a(x) = h_a(y)$. Therefore, $|\{h_a \in \mathcal{H} : h_a(x) = h_a(y)\}| = m^{r-1} = \frac{|\mathcal{H}|}{m}$, so $\mathcal{H}$ is universal.

   (Hint: You can use the following fact without proving it: Let m and $0 < b < m$ be integers. If $\gcd(b, m) = 1$, then there is a unique integer $0 < z < m$ such that $bz \equiv 1 \pmod{m}$.)

2. (1 point) In class, we discussed using chaining to handle hash collisions, and we analyzed the expected length of a chain when storing n keys in a hash table of size m. In this question, we study the length of the longest chain.

   For simplicity, assume that the hash function is truly random: it maps each key uniformly at random to $\{0, \ldots, m-1\}$. If we view each key as a ball and each entry in the table as a bin, this is equivalent to throwing m balls independently and uniformly at random into n bins. We focus on the case with $m = n$.

   Prove that with probability at least $1 - \frac{1}{n}$, the maximum-loaded bin has $O\left(\frac{\log n}{\log \log n}\right)$ balls.

   (Hint: First upper bound the probability that the first bin has at least k balls. You may find the inequality $\binom{n}{k} = \frac{n!}{k!(n-k)!} \leq \frac{n^k}{k!}$ useful. Then, by symmetry and a union bound, we have $\Pr[\text{at least one bin has} \geq k \text{ balls}] \leq n \cdot \Pr[\text{first bin has} \geq k \text{ balls}]$.)

3. (1 point) In Q2, we saw that querying a key in a hash table with n stored keys can take $\Theta\left(\frac{\log n}{\log\log n}\right)$ time, even with ideal hash functions. In this question, we study perfect hashing, which achieves $O(1)$ worst-case query time and $O(n)$ space.

   For simplicity, we consider the following static hashing problem: A set S of n keys is given in advance. The goal is to build a hash table for S that only supports querying whether a key is in S, but not inserting or deleting keys.

   Perfect hashing uses two levels of hashing. First, we choose a hash function h_1 that maps the n keys into a table of size $m = n$. For each index $0 \le j < m$, let $\ell_j = |\{k \in S : h_1(k) = j\}|$ denote the number of keys mapped to slot j. The key idea is that, instead of chaining, we create a second-level hash table of size ℓ_j^2 to store these ℓ_j keys using a hash function $h_{2,j}$.

   (a) Prove that if h_2 is chosen from a universal family of hash functions that map keys to $\{0, \dots, \ell^2 - 1\}$, then the probability that h_2 has a collision on ℓ given keys is at most $\frac{1}{2}$.

   (b) Prove that if h_1 is chosen uniformly from a universal family of hash functions that map keys to $\{0, \dots, m-1\}$ where $m = n$, then $\mathbb{E}\left[\sum_{j=0}^{m-1} \ell_j^2\right] \le 2n$.

   (Consequently, by Markov's inequality, $\Pr\left[\sum_j \ell_j^2 \ge 4n\right] \le \frac{\mathbb{E}[\sum_j \ell_j^2]}{4n} \le \frac{1}{2}$.)

   To build the data structure, we repeatedly draw h_1 until $\sum_j \ell_j^2 \le 4n$. Each draw succeeds with probability at least $\frac{1}{2}$, so the expected number of trials is at most 2. Then, similarly for each j, we repeatedly draw $h_{2,j}$ until no collisions happen on those ℓ_j keys. After a successful (randomized) construction, the hash functions are fixed.

   The total space is $m + \sum_j \ell_j^2 \le 5n$, which is $O(n)$. Querying a key k takes $O(1)$ worst-case time because there are no collisions in the second-level tables: first compute $j = h_1(k)$ and then check the $h_{2,j}(k)$-th slot in the j-th second-level table.