CS 270 Combinatorial Algorithms & Data Structures — Spring 2023

Problem Set 4

Due: 11:59pm, Wednesday, March 8th
Solution maximum page limit: 3 pages

See homework policy at `https://cs270.org/spring23/syllabus/#homework-policies`

Problem 1: (20 points) In *simple tabulation hashing*, we write a key $x \in [u]$ in base $u^{1/c}$ (assume $u^{1/c}$ is an integer and power of 2) so that $x = \sum_{i=0}^{c-1} x_i \cdot u^{i/c}$. We call the x_i "characters". We then allocate c completely random lookup tables $H_0, \ldots, H_{c-1}$ each of size $[u^{1/c}]$. Then for each $y \in [u^{1/c}]$ we set $H_i(y)$ uniformly at random from $[m]$ (assume also m is a power of 2). Then to hash x, we define (where $\oplus$ denotes bitwise XOR)

$$h(x) = H_0(x_0) \oplus H_1(x_1) \oplus \cdots \oplus H_{c-1}(x_{c-1}).$$

(a) (5 points) Fix $c, m \geq 1$. Show that the family $\mathcal{H}$ of all such hash functions is 3-wise independent.

(b) (5 points) Show that $\mathcal{H}$ is not 4-wise independent.

(c) (10 points) Imagine using such an $\mathcal{H}$ to implement hashing with chaining. Show that if $m > n^{1.01}$ then with probability at least 2/3, every linked list in the hash table has size $O(1)$. **Hint:** show that if a subset T of the n items is "large", then $\mathcal{H}$ behaves completely randomly on some "somewhat large" subset T' of T.

Problem 2: (20 points) In the *static* dictionary problem, $(x_1, v_1), \ldots, (x_n, v_n)$ are n (key, value) pairs given up front, and we would like to create a data structure with low memory and fast preprocessing that supports querying keys in $O(1)$ worst-case time, guaranteed (not just in expectation!). A hash function $h : [U] \to [m]$ is *perfect* if $h(x_i) \neq h(x_j)$ for any $1 \leq i < j \leq n$. Note that if we had a perfect hash function, we could simply have an array in which we store (x_i, v_i) directly in the $h(x_i)$'th position of the array, and queries would be constant time (assuming h supports evaluation in constant time).

(a) (5 points) Suppose we pick a hash function $h : [U] \to [m]$ at random from a 2-wise independent hash family for $m = n^2$, such that $h(x)$ can be computed in $O(1)$ for any x. Show that h is a perfect hash function with probability $\Omega(1)$.

(b) (2 points) Deduce from (a) that there is a solution to static dictionary using $O(n^2)$ space, with $O(1)$ worst case query time, and $O(n^2)$ expected preprocessing time.

(c) (8 points) Give a solution to static dictionary with $O(n)$ space, $O(1)$ worst case query time, and $O(n)$ expected preprocessing time. **Hint:** do 2-level hashing. First pick $h : [U] \to [n]$ from a 2-wise independent family. Then for each $i = 1, \ldots, n$ pick a new 2-wise hash function g_i to again hash the keys x with $h(x) = i$.

Problem 3: (20 points) In the *static approximate dictionary* problem, we are given n (key, value) pairs $\{(x_i, v_i)\}_{i=1}^n$ (with distinct keys x_i) and would like to create a data structure supporting `query`(x). The answer to this query should be v_i if $x = x_i$ for some i; otherwise, if x does not equal x_i for any i, the data structure can silently fail; that is, any (garbage) output is allowed. The x_i are in $\{0, 1 \ldots, U-1\}$, and the v_i are in $\{0,1\}^r$. Show that a randomized solution exists using $O(nr)$ *bits* of space and $O(n)$ expected preprocessing time, and $O(1)$ worst case query time (assuming r is at most the word size w). For the sake of this problem, you can assume access to as many perfectly random hash functions as you'd like without paying for the memory to store them, and which are (1) fully random, and (2) can be evaluated in constant time. **Hint:** use cuckoo hashing, where you break up the value associated with key x into "pieces", and store these pieces separately in `A[`$h(x)$`]` and `A[`$g(x)$`]` for your hash table `A`.

Problem 4: (1 point) How much time did you spend on this problem set? If you can remember the breakdown, please report this per problem. (sum of time spent solving problem and typing up your solution)