A 1-universal System of Functions

Recall the definition of a *c-universal system of functions* from $\mathcal{U} \to [m]$ for $c \geq 1$: a system $\mathcal{H}$ is c-universal if for every two distinct elements $x, y \in \mathcal{U}$, it holds that $\Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq c/m$.
Let the number of buckets m be some prime number p. (This is fine because of Bertrand's postulate: for every desired m, there is a prime number $p \leq 2m$, so we are asymptotically not losing any space.) Notice that $\mathbb{Z}_p$ is a field. Let $\mathcal{U} = \mathbb{Z}_p^d$. We will have one hashing function for each d-tuple $\mathbf{t} \in \mathbb{Z}_p^d$ defined by the scalar product $h_\mathbf{t}(\mathbf{x}) = \mathbf{t} \cdot \mathbf{x}$.

Theorem. The system of function $\mathcal{H} = \{h_\mathbf{t} \mid \mathbf{t} \in \mathbb{Z}_p^d\}$, where $h_\mathbf{t}(\mathbf{x}) = \mathbf{t} \cdot \mathbf{x}$, is 1-universal.

Proof. Let $\mathbf{x}, \mathbf{y} \in \mathbb{Z}_d^p$ be two distinct vectors. Let k be a coordinate where $x_k \neq y_k$. Since the scalar product is invariant under permuting the coordinates, we can reorder the coordinates so that $\mathbf{x}$ and $\mathbf{y}$ differ in the last coordinate, that is, $k = d$.
Let us now choose $\mathbf{t}$ coordinate by coordinate, and compute the probability of a collision. (Equalities $\mod p$ will be denoted $\equiv$.)

$$\Pr_{\mathbf{t} \in \mathbb{Z}_p^d}[h_\mathbf{t}(\mathbf{x}) = h_\mathbf{t}(\mathbf{y})] = \Pr[\mathbf{x} \cdot \mathbf{t} \equiv \mathbf{y} \cdot \mathbf{x}] = \Pr[(\mathbf{x} - \mathbf{y}) \cdot t \equiv 0] =$$

$$= \Pr\left[\sum_{i=1}^{d} (x_i - y_i) t_i \equiv 0\right] = \Pr\left[(x_d - y_d) t_d \equiv -\sum_{i=1}^{d-1} (x_i - y_i) t_i\right] .$$

If we have already chosen $t_1, \dots, t_{d-1}$, and now we are choosing t_d randomly, a collision will occur for exactly one choice: the last expression is a linear equality of the form $az = b$ for non-zero a, and this has a unique solution z in any field. Thus, the probability of a collision is at most $1/p = 1/m$, as required by 1-universality. □

---

1. Data Structure 1. Construct a (composite) data structure which can handle the following operations in the required time:

- `Init()` – initializes the data structure– $\mathcal{O}(1)$.
- `INSERT(`X`)` – inserts element X, if it is not yet in the structure – $\mathcal{O}(\log n)$.
- `DELETE(`X`)` – deletes X, if it is in the structure – $\mathcal{O}(\log n)$.
- `DELETE_IN_PLACE(`I`)` – deletes element which was the I-th added – $\mathcal{O}(\log n)$.
- `GET_PLACE(`X`)` – returns a number I such that X was the I-th added element – $\mathcal{O}(\log n)$.

2. Data Structure 2. An electrician wants to maintain a list of clients indexed by their IDs together with a record of whether they are male or female. Design a data structure which handles the following operations in the time $\mathcal{O}(\log n)$:

- `INSERT(`K`, `C`)` – inserts a new client C with `ID`=K, designates them female.
- `UPDATE(`K`)` – designates client with `ID`=K as male.
- `FINDDIFF(`K`)` – finds the difference between the numbers of male and female clients among those with `ID` $\leq K$.

3. Window. Numbers are arriving on input. Whenever a new number arrives, report the median and average of the last k numbers. Try to attain $\mathcal{O}(\log k)$ complexity per report.

4. (a, b) **in one direction.** Modify the `INSERT` and `DELETE` operations in (a, b)-trees so that they only make modifications on the way down.

5. Sum. Say we have a set of natural numbers and a number x. We want to find out as quickly as possible whether our set contains a pair of elements which sum up to x.
What if I had a fixed x, but wanted my set to be dynamic, that is, I can `INSERT` and `DELETE` elements, and I want to be able to quickly check whether there is or isn't a pair of elements summing up to x. In what time is this possible?

6. Convenience Store. Frank's convenience store has customers come in and add orders into a queue; an order is a triple (item, quantity, name of customer). Frank would like to have a good overview of whether he has enough goods of each kind in stock.
Design a data structure for his store, which will be able to execute the following operations in $\mathcal{O}(1)$ time:

(1) `ENQUEUE(`R`)` — enqueues the order R
(2) `DEQUEUE()` — prints the next order and removes it from the queue.
(3) `QUERY(`P`)` — for item P reports the total quantity of orders of this product.

(I'm assuming you know the FIFO queue data structure.) You are guaranteed that the queue will never contain more than m orders, and you know that there are n types of items in the store. Can you find a solution in space $\mathcal{O}(n)$? What about space $\mathcal{O}(m)$, in case that $m \ll n$?

7. Collision. You were given a hash function $h : [U] \to [m]$. Unless you know anything else about the function, how many function evaluations do you need to find a set of k elements of $[U]$ which all collide, that is, $\{a_1, \ldots, a_k\} \subseteq [U]$ and $h(a_1) = \cdots = h(a_k)$?

8. List. Design a data structure for storing a list such that we can quickly find the k-the element and move it to the beginning of the list.