Hashing

N possible items (say $[N]$).
Given $O(m)$ memory.
Want to build a data structure (dictionary) D that supports
① insert(x) $\forall x \in [N]$. : add x to D if it's not there.
② lookup(x) $\forall x \in [N]$: check $x \in D$.
③ delete(x) $\forall x \in [N]$: delete x from D if it's there.
(Assume that storing a number in $[N]$ takes $O(1)$ space and doing basic arithmetic with a number in $[N]$ takes $O(1)$ time.)
(There are many balanced binary search trees that support each operation in $O(\log |D|)$ time (red-black, AVL, splay, ...))

Hashing: simpler and faster implementation that works "well" with high probability?

"Informal statement": $\forall$ fixed sequence of n operations
(insert(1), lookup(1), insert(2), ...,)
with high probability, algorithm "works well".
(with small prob., algo can "fail")

Common Framework. Let $\mathcal{H} \subseteq \{h : [N] \to [t]\}$ be a set of hash functions.

- Pick $h \in \mathcal{H}$ randomly.
- Given $i \in [N]$, consider $h(i) \in [t]$ and "see i is there or not"

Questions.
- How to sample h?
- How to store h?
- How to compute $h(i)$?

$\Big\}$ time, memory.

Example] $\mathcal{H} = \{h : [N] \to [t]\}$ (t^N functions)

- Sampling h takes $\Theta(N)$ time. (choose $h(i)$ $\forall i \in [N]$)
- Storing h takes $\Theta(N)$ space
- Computing $h(i)$ takes $O(1)$ time

Better $\mathcal{H}$, while retaining (some) power of randomness?

Definition. $\mathcal{H}$ is "k-wise independent / k-universal" for some $k \geq 1$ if $\forall i_1 < \cdots < i_k \in [N]$, random variables $h(i_1), \ldots, h(i_k)$ are independent.

($\forall b_1 \ldots b_k \in [t]$, $\Pr[h(i_1) = b_1, \ldots, h(i_k) = b_k] = 1/t^k$)

Example. $N = 3$, $t = 2$ (say $1 = +$, $2 = -$), then $\mathcal{H} = \{+++, +--, -+-, --+\}$ is 2-universal.

Theorem. $\forall k \geq 1$, $\exists \mathcal{H}$ s.t. sampling h, computing $h(i)$ take $O(k)$ time.
storing h takes $O(k)$ space.

Hashing with Chaining

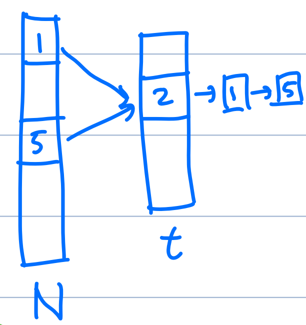
Common Framework, Let $\mathcal{H} \subseteq \{h : [N] \to [t]\}$ be a set of hash functions.

2-universal

- Pick $h \in \mathcal{H}$ randomly.
- Given $i \in [N]$, consider $h(i) \in [t]$ and "see i is there or not"

$\forall j \in [t]$, maintain a linked list $l(j)$ containing items $i \in D$ with $h(i) = j$.

- Space : $O(|D|)$.
- Running time: consider fixed sequence of n operations, and k^{th} operation among them. ($k \in [n]$) (insert / lookup / delete $i \in [N]$)



runtime of this operation $= O(|l_{h(i)}| + 1)$ when this operation happens.

$\mathbb{E}$[runtime of this operation] $= O(1 + \mathbb{E}[|l_{h(i)}|])$.

$$\mathbb{E}[|l_{h(i)}|] \le \mathbb{E}\Big[\sum_{\substack{j \in [N] \\ \text{inserted before } k}} \mathbb{1}[h(j) = h(i)]\Big]$$

$$= \sum_{\substack{j \in [N] \\ \text{inserted before } k}} \underbrace{\mathbb{E}[\mathbb{1}[h(j) = h(i)]]}_{\begin{cases} 1 & \text{if } i = j \\ 1/t & \text{if } i \ne j \end{cases}} \le 1 + k/t \le 1 + n/t\,!$$

So, $\mathbb{E}$[total running time] $= O(n + n^2/t)$. If $t = \Theta(n)$, it's $O(n)$.

Perfect Hashing

Common Framework, Let $\mathcal{H} \subseteq \{h : [N] \to [t]\}$ be a set of hash functions.

- Pick $h \in \mathcal{H}$ randomly. 2-universal.
- Given $i \in [N]$, consider $h(i) \in [t]$ and "see i is there or not"

might be better if no two $i, j \in D$ have $h(i) = h(j)$!

Definition, Say h is "perfect" for $D \subseteq [N]$ if $\forall i \neq j \in D$, $h(i) \neq h(j)$.

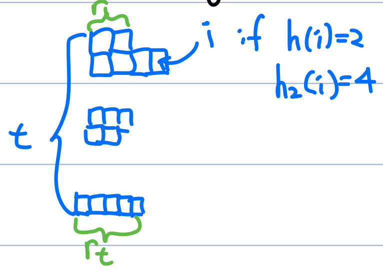
But even when $\mathcal{H} = \{h : [N] \to [t]\}$, we need to have $|D| \leq O(\sqrt{t})$ to have perfect hashing with prob. $\geq 90\%$ (Birthday paradox).

2-level-hashing, (only works for "static dictionary"; n items $i_1 \dots i_n \in [N]$ inserted first and other operations are lookups.)

Let $t = O(n)$ and sample 2-universal $h : [N] \to [t]$.

For each $j \in [t]$, let $k_j = \{\ell \in [n] : h(i_\ell) = j\}$. Allocate a "Block" of $r_j = O(k_j^2)$ array cells to j, sample 2-universal $h_j : [N] \to [r_j]$.

$i \in [N]$ is assigned to $h_{h(i)}(i)^{\text{th}}$ cell of $h(i)^{\text{th}}$ block.



- $\mathbb{E}[\text{Total memory used}] = \mathbb{E}\left[\sum_{j=1}^{t} O(k_j^2)\right] \leq \mathbb{E}\left[\left(\sum_{\substack{i,i' \in [N] \\ \text{inserted}}} \mathbb{1}[h(i)=h(i')]\right)\right]$

$$\leq O(n^2/t + n) = O(n).$$

- Perfectness: Given h, $\forall j \in [t]$, we want to find $h_j : [N] \to [r_j]$ s.t. $\forall i, i'$ inserted with $h(i)=h(i')=j$, $h_j(i) \neq h_j(i')$.

$$\sum_{\substack{i \neq i' \text{ inserted} \\ h(i)=h(i')=j}} \Pr[h_j(i)=h_j(i')] = \binom{k_j}{2} / r_j \leq 0.1 \quad (\text{by choosing } r_j = 10k_j^2).$$

So h_j will be good w.p. $\geq 90\%$. if fail, retry.

$\mathbb{E}[\#\text{ tries}] \leq 1 \cdot 0.9 + 2 \cdot (1-0.9)\, 0.9 + 3 \cdot (1-0.9)^2\, 0.9 + \cdots + \leq O(1)$.