Lecture #25

CS 170

Spring 2021

Hashing

Goal: Build a "Dictionary": a data structure D
with following properties

Contains key-value pairs $(k_1, v_1), \dots, (k_n, v_n)$
assume all keys distinct

Implements $\text{Search}(D, k) = \begin{cases} v_i & \text{if } k = k_i \\ \text{nil} & \text{if } k \text{ not in } D \end{cases}$

Search as fast as possible

Size $|D|$ as small as possible

Also want insert$(D, (k, v))$ and delete(D, k)
but won't discuss today

Dictionaries: 2 simple approaches

#1: D = list of n (key, value) pairs sorted by key

search(D, k) does binary search on keys

Time(search) $= O(\log n)$ ... OK

$|D| = O(n)$ ... as small as possible

#2: U = set of all possible keys, treated as integer $\in [1, |U|]$

D = array of length $|U|$, $D(k_i) = v_i$, other $D(\cdot) = \text{nil}$

search(D, k) looks up $D(k)$

Time(search) $= O(1)$ ... as small as possible

$|D| = O(|U|)$ ... enormous

Goal: Time(search) $= O(1)$ and $|D| = O(n)$

Use hashing

Hashing

- Need a "hash function" $h: U \to [1:m]$, $m = O(n)$
  where $h(k) \to$ linked list containing all (k, v) pairs with same $h(k)$
- Want h with as few "collisions" as possible, i.e. shortest linked lists, since
  Time(search) $= O$(length of longest linked list)
  - Best case: all linked lists same length $= \frac{n}{m}$
  - Some linked list $\geq \frac{n}{m}$
- Problem with choosing any one fixed h
  - $\exists$ subset $U' \subseteq U$ of size $\geq |U|/m$ s.t. $h(U') =$ one value
  - Ex: $h(k) =$ first $\log_2 m$ bits of k
- Solution: pick h randomly

Picking a random hash function

- Idea: If we pick h randomly from a set $\mathcal{H}$, it should assign roughly equally many keys to each linked list, independent of which keys appear
- Def: $\mathcal{H}$ is universal if for all $k \neq k'$, both in U

$$P(h(k) = h(k')) \leq \frac{1}{m}, \quad m = \# \text{ linked lists}$$

- Thm: For all $1 \leq i \leq n$,

$$\mathbb{E}(\#\text{keys in same linked list as } k_i) < \frac{n}{m}$$

Proof: let $C(j,l) = 1$ if $h(k_j) = h(k_l)$, 0 otherwise

$$\mathbb{E}(\#\text{keys in same linked list as } k_i)$$
$$= \mathbb{E}\Big(\sum_{j \neq i} C(i,j)\Big) = \sum_{j \neq i} \mathbb{E}(C(i,j)) \leq \sum_{j \neq i} \frac{1}{m} \leq \frac{n-1}{m}$$

- If we use a random hash function h from universal $\mathcal{H}$ and $m = \Theta(n)$, $\mathbb{E}(\text{search time}) = \mathbb{E}(\#\text{keys in linked list}) = \Theta(1)$

Constructing a Universal $\mathcal{H}$ (1/2)

- Def: $\mathcal{H}$ is universal if for all $k \neq k'$, both in U

  $P(h(k) = h(k')) \leq \frac{1}{m}$, $m = \#$ linked lists
- First try: if $h : U \rightarrow [1:m]$ completely random, costs $|U|$ to store, defeats goal of $O(n)$ memory
- Second try: inner product with random vector
  - Assume m prime (round up if needed)
  - Assume $|U| = m^r$ for some r
  - View each $k \in U$ in base m: $k = \sum_{i=0}^{r-1} k^{(i)} m^i$, $0 \leq k^{(i)} < m$

    or $k \equiv (k^{(0)}, k^{(1)}, \ldots, k^{(r-1)})$
  - Def: $\mathcal{H} = \{h_a, a \in U\} = \{(a^{(0)}, a^{(1)}, \ldots, a^{(r-1)}), 0 \leq a^{(i)} < m\}$
  - $|h_a| = |a| = \log |U| = r \log m$, much smaller than before
  - Def: $h_a(k) = \sum_{i=0}^{r-1} a^{(i)} \cdot k^{(i)} \mod m$

Constructing a Universal $\mathcal{H}$ (2/2)

- Def: $\mathcal{H}$ is universal if for all $k \neq k'$, both in U
  $P(h(k)=h(k')) \le \frac{1}{m}$, $m = \#$ linked lists
- Second try: inner product with random vector
  - Assume m prime, $|U| = m^r$ for some r
  - Each $k \in U$: $k = \sum_{i=0}^{r-1} k^{(i)} m^i$, $0 \le k^{(i)} < m$ or $k \equiv (k^{(0)}, k^{(1)}, \ldots, k^{(r-1)})$
  - Def: $\mathcal{H} = \{h_a, a \in U\} = \{(a^{(0)}, a^{(1)}, \ldots, a^{(r-1)}), 0 \le a^{(i)} < m\}$
  - $|h_a| = |a| = \log|U| = r \log m$, much smaller than before
  - Def: $h_a(k) = \sum_{i=0}^{r-1} a^{(i)} \cdot k^{(i)} \mod m$
  - Claim: Inner product hash family $\mathcal{H}$ is universal

  Proof: $P(h(k)=h(k')) = P(\sum_{i=0}^{r-1} a^{(i)} k^{(i)} = \sum_{i=0}^{r-1} a^{(i)} k'^{(i)} \mod m)$
  $= P(\sum_{i=0}^{r-1} a^{(i)} (k^{(i)} - k'^{(i)}) = 0 \mod m)$. $k \neq k' \Rightarrow$ some $k^{(j)} \neq k'^{(j)} \Rightarrow$
  $= P(a^{(j)}(k^{(j)} - k'^{(j)}) = \sum_{i \neq j} a^{(i)} (k^{(i)} - k'^{(i)}) \mod m)$
  $= P(a^{(j)} = (k^{(j)} - k'^{(j)})^{-1} \cdot \sum_{i \neq j} a^{(i)} (k^{(i)} - k'^{(i)}) \mod m)$
  $= 1/m$ as desired, since each $a^{(j)}$ random in $[0, m-1]$

Improving $\mathbb{E}$(search time) $= O(1)$ to max(search time) $= O(1)$

(1/4)

- Def: Perfect hashing uses 2 layers of hashing
  - Layer 1: $h_0 : U \to [1:m]$, maps each $u \in U$ to another hash function $h_1, \dots, h_m$
  - Layer 2: $h_i : U \to [1:l_i]$, l_i chosen to have no collisions
- Size goal: $|D| = |h_0| + |h_1| + \cdots + |h_m| = O(n)$
- Search time goal: $\text{time}(h_0) + \text{time}(h_i) = O(1)$ for all i
- Repeatly choose random h_0, h_i until goals met
  - Show $P(\text{meeting goal}) \geq \frac{1}{2} \Rightarrow$
    $\mathbb{E}(\#\text{random choices of each } h_0, h_i \text{ needed}) \leq 2$

Improving $\mathbb{E}$(search time) $= O(1)$ to max(search time) $= O(1)$

- Def: Perfect hashing uses 2 layers of hashing
  - L1: $h_0 : U \to [1:m]$, maps each $u \in U$ to $h_1, \ldots, h_m$
  - L2: $h_i : U \to [1:\ell_i]$, ℓ_i chosen to have no collisions
  - Size goal: $|D| = |h_0| + |h_1| + \cdots + |h_m| = O(n)$
  - Search time goal: $\text{time}(h_0) + \text{time}(h_i) = O(1)$ for all i
  - Repeatly choose random h_0, h_i until goals met
- How to sample:

  L1: Repeat: sample h_0 until $\sum_{i=1}^{m} c_i^2 \le \nu n$, $\nu = O(1)$
  where $c_i = \#$keys mapped to i

  L2: for $i = 1:m$, Repeat: sample $h_i : U \to [1:c_i^2]$ until no collisions
- Size goal: $|h_0| + |h_1| + \cdots + |h_m| = O(n) + \sum_{i=1}^{m} c_i^2 = O(n)$
- Time goal: No collisions $\Rightarrow \text{time}(h_i) = O(1) \Rightarrow$
  $\text{time}(h_0) + \text{time}(h_i) = O(1)$

Improving $\mathbb{E}(\text{search time}) = O(1)$ to $\max(\text{search time}) = O(1)$

- Def: Perfect hashing uses 2 layers of hashing
  - L1: $h_0 : U \to [1:m]$, maps each $u \in U$ to $h_1, \ldots, h_m$
  - L2: $h_i : U \to [1:\ell_i]$, ℓ_i chosen to have no collisions
- How to sample:

  L1: Repeat: sample h_0 until $\sum_{i=1}^{m} c_i^2 \le \nu n$, $\nu = O(1)$
  where c_i = #keys mapped to i

  L2: for $i = 1:m$, Repeat: sample $h_i : U \to [1 : c_i^2]$ until no collisions
- Analysis of sampling h_i for L2: P(collision) =

  P(h_i maps 2 of the c_i keys to same index out of c_i^2)

  $= \sum_{k \ne k' \text{ mapped to } i = h_0(k) = h_0(k')} P(h_i(k) = h_i(k'))$

  $= \binom{c_i}{2} \frac{1}{c_i^2} \le \frac{1}{2} \Rightarrow P(\text{successful sampling}) \ge \frac{1}{2}$

Improving $\mathbb{E}$(search time) $= O(1)$ to max(search time) $= O(1)$ (4/4)

- Def: Perfect hashing uses 2 layers of hashing
  - L1: $h_0 : U \to [1:m]$, maps each $u \in U$ to $h_1, \dots, h_m$
  - L2: $h_i : U \to [1:\ell_i]$, ℓ_i chosen to have no collisions
- How to sample:

  L1: Repeat: sample h_0 until $\sum_{i=1}^{m} c_i^2 \le \nu n$, $\nu = O(1)$
  where $c_i = \#$keys mapped to i

  L2: for $i = 1:m$, Repeat: sample $h_i : U \to [1 : c_i^2]$ until no collisions
- Analysis of sampling h_0 for L1:

$$\mathbb{E}\left(\sum_{i=1}^{m} c_i^2\right) = \mathbb{E}\left(\sum_{i=1}^{m}\left(\sum_{j=1}^{n} \mathrm{In}(h_0(k_j)=i)\right)^2\right)$$

$$= \mathbb{E}\left(\sum_{i=1}^{m} \sum_{j,j'=1}^{n} \mathrm{In}(h_0(k_j)=h_0(k_{j'})=i)\right) \quad \dots \text{reverse sums}$$

$$= \mathbb{E}\left(\sum_{j,j'=1}^{n} \sum_{i=1}^{m} \mathrm{In}(h_0(k_j)=h_0(k_{j'})=i)\right) = \mathbb{E}\left(\sum_{j,j'=1}^{n} \mathrm{In}(h_0(k_j)=h_0(k_{j'}))\right)$$

$$= n + \binom{n}{2} \cdot \frac{1}{m} = \Theta(n) \text{ because } m = \Theta(n)$$

10 Markov $\Rightarrow P\left(\sum_{i=1}^{m} c_i^2 > \nu n\right) \le \mathbb{E}\left(\sum_{i=1}^{m} c_i^2\right) / (\nu n) = \dfrac{\Theta(n)}{\nu n} \le \dfrac{1}{2}$ if ν big enough