Lecture #25

CS 170
Spring 2021
Hashing

Goal: Build a “Dictionary”: a data structure \( D \) with following properties:

- Contains key-value pairs \((k_1,v_1), \ldots, (k_n,v_n)\)
- Assume all keys distinct

Implements \( \text{Search}(D,k) = \begin{cases} v_i & \text{if } k = k_i \\ \text{nil} & \text{if } k \text{ not in } D \end{cases} \)
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 ∈ [1, |U|]
  D = array of length |U|, D(k_i) = v_i, other D(·) = nil
  search(D, k) looks up D(k)
  Time(search) = O(1) ... as small as possible
  |D| = O(1|U|) ... enormous
Hashing

- Need a "hash function" \( h: U \rightarrow [1:m] \), \( m = o(n) \)
  where \( h(k) \rightarrow \) 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

\[
\text{Time(search)} =
\]

- Problem with choosing any one fixed \( h \)
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 \( \mathcal{U} \)
  
  \[ P(h(k) = h(k')) \leq \frac{1}{m}, \quad m = \# \text{linked lists} \]

- Thm:
Constructing a Universal $H$ (1/2)

- **Def:** $H$ is universal if for all $k \neq k'$, both in $U$
  \[ P(h(k) = h(k')) \leq \frac{1}{m}, \quad m = \#\text{ linked lists} \]

- **First try:** if $h: U \rightarrow [1:m]$ completely random, costs $|U|$ to store, defeats goal of $O(n)$ memory

- **Second try:**
Constructing a Universal \( \mathcal{H} \) (2/2)

- **Def:** \( \mathcal{H} \) is universal if for all \( k \neq k' \), both in \( U \)
  \( P(h(k) = h(k')) \leq \frac{1}{m}, \ m = \# \text{ linked lists} \)

- **Second try:** inner product with random vector
- Assume \( m \) prime, \( |U| = m^r \) for some \( r \)
- Each \( k \in U: k = \sum_{i=0}^{\infty} k^i \cdot m^i, 0 \leq k^i < m \) or \( k = (k^{(0)}, k^{(1)}, \ldots, k^{(r-1)}) \)
- **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:** \( h_a(k) = \sum_{i=0}^{\infty} a^{(i)} \cdot k^{(i)} \mod m \)
Improving $E(\text{search time}) = O(1)$ to $\max(\text{search time}) = O(1)$

- **Def:** Perfect hashing uses 2 layers of hashing
  - **Layer 1:**
  - **Layer 2:**
  - **Size goal:**
  - **Search time goal:**
Improving $E(\text{search time})=O(1)$ to $\max(\text{search time})=O(1)$

- **Def:** Perfect hashing uses 2 layers of hashing
  - $L1$: $h_0 : U \rightarrow [1:m]$, maps each $u \in U$ to $h_1, \ldots, h_m$
  - $L2$: $h_i : U \rightarrow [1:l_i]$, $l_i$ 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 $i$
  - Repeatedly choose random $h_0, h_i$ until goals met
Improving $E(\text{search time})=O(1)$ to $\max(\text{search time})=O(1)$

- Def: Perfect hashing uses 2 layers of hashing
  - $L1$: $h_o: U \rightarrow [1: m]$, maps each $u \in U$ to $h_1, ..., h_m$
  - $L2$: $h_i: U \rightarrow [1: l_i]$, $l_i$ chosen to have no collisions
- How to sample:
  - $L1$: Repeat: sample $h_o$ until $\sum_{i=1}^{m} c_i^2 \leq \mu n$, $\mu = O(1)$
    where $c_i = \# \text{keys mapped to } i$
  - $L2$: for $i = 1: m$, Repeat: sample $h_i: U \rightarrow [1: c_i^2]$ until no collisions
Improving $E(\text{search time}) = O(1)$ to $\max(\text{search time}) = O(1)$

- Def: Perfect hashing uses 2 layers of hashing
  - $L1: h_0: U \rightarrow [1:m]$, maps each $u \in U$ to $h_1, \ldots, h_m$
  - $L2: h_i: U \rightarrow [1:l_i]$, $l_i$ chosen to have no collisions
- How to sample:
  - $L1$: Repeat: sample $h_0$ until $\sum_{i=1}^{m} c_i^2 \leq \mu n$, $\mu = O(1)$ where $c_i = \# \text{keys mapped to } i$
  - $L2$: for $i = 1:m$, Repeat: sample $h_i: U \rightarrow [1:c_i^2]$ until no collisions