Lecture 8: Hashing

Course Overview

This course covers several modules:

1. Review: dictionaries, chaining, simple uniform
2. Universal hashing
3. Perfect hashing

Review

Dictionary Problem

A dictionary is an Abstract Data Type (ADT) that maintains a set of items. Each item has a key. The dictionary supports the following operations:

- **insert(item)**: add item to set
- **delete(item)**: remove item from set
- **search(key)**: return item with key if it exists

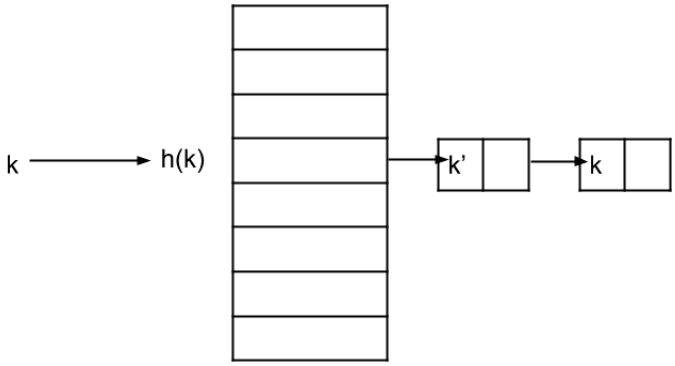
We assume that items have distinct keys (or that inserting new ones clobbers old ones).

This problem is easier than predecessor/successor problems solved in previous lecture (by van Emde Boas trees, or by AVL/2-3 trees/skip lists).

Hashing from 6.006

Goal: $O(1)$ time per operation and $O(n)$ space complexity.
Definitions:

- u = number of keys over all possible items
- n = number of keys/items currently in the table
- m = number of slots in the table



Solution: hashing with chaining

Assuming simple uniform hashing,

$$\Pr_{k_1 \neq k_2} \{h(k_1) = h(k_2)\} = \frac{1}{m}$$

we achieve $\Theta(1 + \alpha)$ time per operation, where $\alpha = \frac{n}{m}$ is called load factor. The downside of the algorithm is that it requires assuming input keys are random, and it only works in average case, like basic quicksort. Today we are going to remove the unreasonable simple uniform hashing assumption.

Etymology

The English 'hash' (1650s) means "cut into small pieces", which comes from the French 'hacher' which means "chop up", which comes from the Old French 'hache' which means "axe" (cf. English 'hatchet'). Alternatively, perhaps they come from Vulcan 'la'ash', which means "axe". (R.I.P. Leonard Nimoy.)

Universal Hashing

The idea of universal hashing is listed as following:

- choose a random hash function h from $\mathcal{H}$
- require $\mathcal{H}$ to be a *universal hashing family* such that

$$\Pr_{h \in \mathcal{H}} \{h(k) = h(k')\} \leq \frac{1}{m} \quad \text{for all} \quad k \neq k'$$

.

- now we just assume h is random, and make no assumption about input keys. (like Randomized Quicksort)

Theorem: For n arbitrary distinct keys and random $h \in \mathcal{H}$, where $\mathcal{H}$ is a universal hashing family,

$$E[\text{ number of keys colliding in a slot }] \leq 1 + \alpha \quad \text{where} \quad \alpha = \frac{n}{m}$$

Proof: Consider keys $k_1, k_2, \ldots, k_n$. Let $I_{i,j} = \begin{cases} 1 & \text{if } h(k_i) = h(k_j) \\ 0 & \text{otherwise} \end{cases}$. Then we have

$$\begin{aligned} E[I_{i,j}] &= \Pr\{I_{i,j} = 1\} \\ &= \Pr\{h(k_i) = h(k_j)\} \\ &\leq \frac{1}{m} \text{ for any } j \neq i \end{aligned} \tag{1}$$

$$\begin{aligned} E[\#\text{ keys hashing to the same slot as } k_i] &= E\left[\sum_{j=1}^{n} I_{i,j}\right] \\ &= \sum_{j=1}^{n} E[I_{i,j}] \text{ (linearity of expectation)} \\ &= \sum_{j \neq i} E[I_{i,j}] + E[I_{i,i}] \\ &\leq \frac{n}{m} + 1 \end{aligned} \tag{2}$$

□

From the above theorem, we know that Insert, Delete, and Search all take $O(1+\alpha)$ expected time. Here we give some examples of universal hash functions.

All hash functions: $\mathcal{H} = \{\text{all hash functions } h : \{0, 1, \ldots, u-1\} \to \{0, 1, \ldots, m-1\}\}$. Apparently, $\mathcal{H}$ is universal, but it is useless. On one hand, storing a single hashing function h takes $\log(m^u) = u \log(m)$ bits $\gg n$ bits. On the other hand, we would need to precompute u values, which takes $\Omega(u)$ time.

Dot-product hash family:
Assumptions

- m is a prime

- $u = m^r$ where r is an integer

In real cases, we can always round up m and u to satisfy the above assumptions. Now let's view keys in base m: $k = \langle k_0, k_1, \ldots, k_{r-1} \rangle$. For key $a = \langle a_0, a_1, a_2, \ldots, a_{r-1} \rangle$, define

$$\begin{aligned} h_a(k) &= a \cdot k \bmod m \text{ (dot product)} \\ &= \sum_{i=0}^{r-1} a_i k_i \bmod m \end{aligned} \tag{3}$$

Then our hash family is $\mathcal{H} = \{h_a \mid a \in \{0, 1, \ldots, u-1\}\}$

Storing $h_a \in \mathcal{H}$ requires just storing one key, which is a. In the **word RAM model**, manipulating $O(1)$ machine words takes $O(1)$ time and "objects of interest" (here, keys) fit into a machine word. Thus computing $h_a(k)$ takes $O(1)$ time.

Theorem: Dot-product hash family $\mathcal{H}$ is universal.

Proof: Take any two keys $k \neq k'$. They must differ in some digits. Say $k_d \neq k_{d'}$. Define *not* $d = \{0, 1, \ldots, r-1\} \setminus \{d\}$. Now we have

$$\begin{aligned} \Pr_a\{h_a(k) = h_a(k')\} &= \Pr_a \left\{ \sum_{i=0}^{r-1} a_i k_i = \sum_{i=0}^{r-1} a_i k'_i \pmod m \right\} \\ &= \Pr_a \left\{ \sum_{i \neq d} a_i k_i + a_d k_d = \sum_{i \neq d} a_i k'_i + a_d k'_d \pmod m \right\} \\ &= \Pr_a \left\{ \sum_{i \neq d} a_i (k_i - k'_i) + a_d (k_d - k'_d) = 0 \pmod m \right\} \\ &= \Pr_a \left\{ a_d = -(k_d - k'_d)^{-1} \sum_{i \neq d} a_i (k_i - k'_i) \pmod m \right\} \end{aligned} \tag{4}$$

$$\begin{aligned} &(m \text{ is prime} \Rightarrow \mathbb{Z}_m \text{ has multiplicative inverses}) \\ &= \underset{a_{not\ d}}{E} [\Pr_{a_d}\{a_d = f(k, k', a_{not\ d})\}] \\ &(= \sum_x \Pr\{a_{not\ d} = x\} \Pr_{a_d}\{a_d = f(k, k', x)\}) \\ &= \underset{a_{not\ d}}{E} \left[\frac{1}{m}\right] \\ &= \frac{1}{m} \end{aligned}$$

□

Another universal hash family from CLRS: Choose prime $p \geq u$ (once). Define $h_{ab}(k) = [(ak + b) \bmod p)] \bmod m$. Let $\mathcal{H} = \{h_{ab} \mid a, b \in \{0, 1, \ldots, u-1\}\}$.

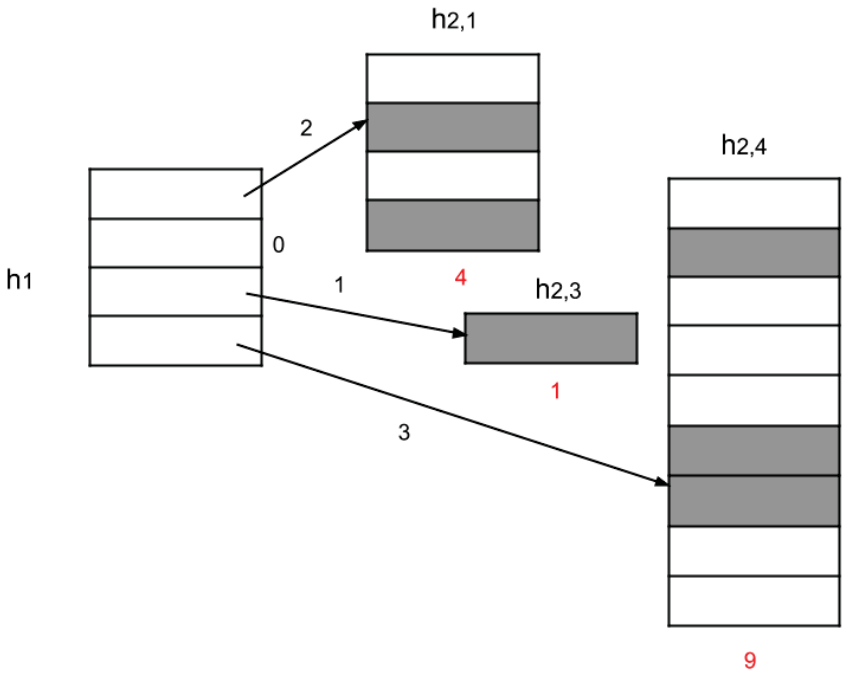
Perfect Hashing

Static dictionary problem: Given n keys to store in table, only need to support search(k). No insertion or deletion will happen.

Perfect hashing: [Fredman, Komlós, Szemerédi 1984]

- polynomial build time with high probability (w.h.p.)
- $O(1)$ time for search in worst case
- $O(n)$ space in worst case

Idea: 2-level hashing



The algorithm contains the following two major steps:

Step 1: Pick $h_1 : \{0, 1, \ldots, u-1\} \to \{0, 1, \ldots, m-1\}$ from a universal hash family for $m = \Theta(n)$ (e.g., nearby prime). Hash all items with chaining using h_1.

Step 2: For each slot $j \in \{0, 1, \ldots, m-1\}$, let l_j be the number of items in slot j. $l_j = |\{i \mid h(k_i) = j\}|$. Pick $h_{2,j} : \{0, 1, \ldots, u-1\} \to \{0, 1, \ldots, m_j\}$ from a universal hash family for $l_j^2 \leq m_j \leq O(l_j^2)$ (e.g., nearby prime). Replace chain in slot j with hashing-with-chaining using $h_{2,j}$.

The space complexity is $O(n + \sum_{j=0}^{m-1} l_j^2)$. In order to reduce it to $O(n)$, we need to add two more steps:

Step 1.5: If $\sum_{j=0}^{m-1} l_j^2 > cn$ where c is a chose constant, then redo Step 1.

Step 2.5: While $h_{2,j}(k_i) = h_{2,j}(k_i')$ for any $i \neq i', j$, repick $h_{2,j}$ and rehash those l_j.

The above two steps guarantee that there are no collisions at second level, and the space complexity is $O(n)$. As a result, search time is $O(1)$. Now let's look at the build time of the algorithm. Both Step 1 and Step 2 are $O(n)$. How about Step 1.5 and Step 2.5?

For Step 2.5,

$$\begin{aligned}
\Pr_{h_{2,j}}\{h_{2,j}(k_i) = h_{2,j}(k_i') \text{ for some } i \neq i'\} &\leq \sum_{i \neq i'} \Pr_{h_{2,j}}\{h_{2,j}(k_i) = h_{2,j}(k_i')\} \text{ (union bound)} \\
&\leq \binom{l_j}{2} \cdot \frac{1}{l_j^2} \\
&< \frac{1}{2}
\end{aligned} \tag{5}$$

As a result, each trial is like a coin flip. If the outcome is "tail", we move to the next step. By Lecture 7, we have $E[\#\text{trials}] \leq 2$ and $\#\text{trials} = O(\log n)$ w.h.p. By a Chernoff bound, $l_j = O(\log n)$ w.h.p., so each trial takes $O(\log n)$ time. Because we have to do this for each j, the total time complexity is $O(\log n) \cdot O(\log n) \cdot O(n) = O(n \log^2 n)$ w.h.p.

For Step 1.5, we define $I_{i,i'} = \begin{cases} 1 & \text{if } h(k_i) = h(k'_i) \\ 0 & \text{otherwise} \end{cases}$. Then we have

$$\begin{aligned} E\left[\sum_{j=0}^{m-1} l_j^2\right] &= E\left[\sum_{i=1}^{n}\sum_{i'=1}^{n} I_{i,i'}\right] \\ &= \sum_{i=1}^{n}\sum_{i'=1}^{n} E[I_{i,i'}] \text{ (linearity of expectation)} \\ &\leq n + 2\binom{n}{2} \cdot \frac{1}{m} \\ &= O(n) \text{ because } m = \Theta(n) \end{aligned} \tag{6}$$

By Markov inequality, we have

$$\Pr_{h_1}\left\{\sum_{j=0}^{m-1} l_j^2 \leq cn\right\} \leq \frac{\sum_{j=0}^{m-1} l_j^2}{cn} \leq \frac{1}{2}$$

for a sufficiently large constant c. By Lecture 7, we have $E[\#\text{trials}] \leq 2$ and $\#\text{trials} = O(\log n)$ w.h.p. As a result, Step 1 and Step 1.5 combined takes $O(n \log n)$ time w.h.p.

6.046J / 18.410J Design and Analysis of Algorithms
Spring 2015