Lecture 7: Hashing III: Open Addressing

Lecture Overview

- Open Addressing, Probing Strategies
- Uniform Hashing, Analysis
- Advanced Hashing

Readings

CLRS Chapter 11.4 (and 11.3.3 and 11.5 if interested)

Open Addressing

Another approach to collisions:

- no chaining; instead all items stored in table (see Fig. 1)

![Open Addressing Table](image)

Figure 1: Open Addressing Table

- one item per slot $\Rightarrow m \geq n$
- hash function specifies order of slots to probe (try) for a key (for insert/search/delete), not just one slot; in math. notation:

We want to design a function $h$, with the property that for all $k \in U$:

$$h : U \times \{0, 1, \ldots, m - 1\} \rightarrow \{0, 1, \ldots, m - 1\}$$

is a permutation of $0, 1, \ldots, m - 1$. i.e. if I keep trying $h(k, i)$ for increasing $i$, I will eventually hit all slots of the table.
**Insert** \((k,v)\) : Keep probing until an empty slot is found. Insert item into that slot.

```python
def insert(k, v):
    for i in xrange(m):
        if T[h(k, i)] is None:
            T[h(k, i)] = (k, v)  # empty slot
            return
    raise 'full'
```

**Example:** Insert \(k = 496\)

```
586  # collision
133  # collision
204  # collision
496  # collision
481  # collision
```

![Figure 3: Insert Example](image)
Search(k): As long as the slots you encounter by probing are occupied by keys \( \neq k \), keep probing until you either encounter \( k \) or find an empty slot—return success or failure respectively.

```python
def Search(k):
    for i in xrange(m):
        if T[h(k, i)] is None:
            return None  # empty slot?
        elif T[h(k, i)][∅] == k:
            return T[h(k, i)]  # matching key
    return None  # exhausted table
```

Deleting Items?
- can’t just find item and remove it from its slot (i.e. set \( T[h(k, i)] = \text{None} \))
- example: delete(586) \( \implies \) search(496) fails
- replace item with special flag: “DeleteMe”, which Insert treats as None but Search doesn’t

Probing Strategies

**Linear Probing**

\[ h(k, i) = (h'(k) + i) \mod m \]

- like street parking
- problem? clustering—cluster: consecutive group of occupied slots as clusters become longer, it gets more likely to grow further (see Fig. 4)
- can be shown that for \( 0.01 < \alpha < 0.99 \) say, clusters of size \( \Theta(\log n) \).

**Double Hashing**

\[ h(k, i) = (h_1(k) + i \cdot h_2(k)) \mod m \]

- actually hit all slots (permutation) if \( h_2(k) \) is relatively prime to \( m \) for all \( k \)
- why?
  \[ h_1(k) + i \cdot h_2(k) \mod m = h_1(k) + j \cdot h_2(k) \mod m \Rightarrow d/(i - j) \]
- e.g. \( m = 2^r \), make \( h_2(k) \) always odd
Uniform Hashing Assumption (cf. Simple Uniform Hashing Assumption)

Each key is equally likely to have any one of the $m!$ permutations as its probe sequence

- not really true
- but double hashing can come close

Analysis

Suppose we have used open addressing to insert $n$ items into table of size $m$. Under the uniform hashing assumption the next operation has expected cost of $\leq \frac{1}{1 - \alpha}$, where $\alpha = \frac{n}{m} (< 1)$.

Example: $\alpha = 90\% \implies 10$ expected probes

Proof:

Suppose we want to insert an item with key $k$. Suppose that the item is not in the table.

- probability first probe successful: $\frac{m-n}{m} = p$  
  ($n$ bad slots, $m$ total slots, and first probe is uniformly random)

- if first probe fails, probability second probe successful: $\frac{m-n}{m-1} \geq \frac{m-n}{m} = p$  
  (one bad slot already found, $m-n$ good slots remain and the second probe is uniformly random over the $m-1$ total slots left)

- if 1st & 2nd probe fail, probability 3rd probe successful: $\frac{m-n}{m-2} \geq \frac{m-n}{m} = p$  
  (since two bad slots already found, $m-n$ good slots remain and the third probe is uniformly random over the $m-2$ total slots left)
• ... ⇒ Every trial, success with probability at least $p$. 

**Expected Number of trials for success?** 

$$\frac{1}{p} = \frac{1}{1 - \alpha}.$$ 

With a little though it follows that search, delete take time $O(1/(1 - \alpha))$. Ditto if we attempt to insert an item that is already there.

**Open Addressing vs. Chaining**

Open Addressing: better cache performance (*better memory usage, no pointers needed*)

Chaining: less sensitive to hash functions (OA requires extra care to avoid clustering) and the load factor $\alpha$ (OA degrades past 70% or so and in any event cannot support values larger than 1)

---

**Advanced Hashing**—This is advanced material for the interested readers only. More about this in 6.046.

**Universal Hashing**

*Goal:* Get rid of the simple uniform hashing assumption, while keeping operations at expected cost $O(1)$.

**Idea:** Instead of defining one hash function, define a family of hash functions

$$\mathcal{H} = \{h_1, h_2, \ldots, h_p \mid h_i : \mathcal{U} \rightarrow \{0, 1, \ldots, m - 1\}\},$$

and select a random $h \in \mathcal{H}$ before starting our insert/delete/search op’s; e.g. multiplication method with *random* multiplier $a$.

**Def:** $\mathcal{H}$ is called a **universal family of hash functions** iff for all pairs of keys $k_1, k_2 \in \mathcal{U}$:

$$\Pr_{\text{over random } h} \{h(k_1) = h(k_2)\} = \frac{1}{m}.$$ 

Such families $\mathcal{H}$ exist. (see CLRS 11.3.3)

$$\implies O(1) \text{ expected time per operation without assuming simple uniform hashing!}$$
Why? Suppose we use chaining, and have inserted keys $k_1, k_2, \ldots, k_n$ into the hash table using a random $h$ from $\mathcal{H}$. Suppose we search for key $k$. The cost to search is bounded by the number of keys stored at slot $h(k)$ of the hash table ($+O(1)$ to compute $h(k)$ etc.). Hence,

$$\text{cost(to search } k) = O(1) + O \left( \sum_{k_i, k_i \neq k} \mathbb{1}_{h(k_i) = h(k)} \right),$$

where $\mathbb{1}_{h(k_i) = h(k)}$ is 1 if $h(k_i) = h(k)$ and 0 otherwise (indicator function). By linearity of expectation, we have:

$$\mathbb{E}_{\text{(over random } h)} \left[ \text{cost(to search } k) \right] = O(1) + O \left( \sum_{k_i, k_i \neq k} \mathbb{E}_{\text{(over random } h)} \left[ \mathbb{1}_{h(k_i) = h(k)} \right] \right)$$

$$= O(1) + O \left( \sum_{k_i, k_i \neq k} \mathbb{P}_{\text{(over random } h)} \{ h(k_1) = h(k_2) \} \right)$$

$$= O(1) + O \left( \sum_{k_i, k_i \neq k} \frac{1}{m} \right) \quad \text{(since $\mathcal{H}$ is a universal family)}$$

$$\leq O(1 + n/m).$$

**Perfect Hashing**

Guarantee $O(1)$ worst-case search, if keys known in advance (see CLRS 11.5 if interested).