Hashing
Searching in a Dynamic Set

S.V.N. (vishy) Vishwanathan

University of California, Santa Cruz
vishy@ucsc.edu

February 26, 2016
Outline

1 Setting

2 Direct-Address Table

3 Hash Table

4 Open Addressing

5 Designing Hash Functions
Problem Setting

- We have a set of dynamic set of keys consisting of a subset of keys drawn from an universe \( U = \{0, 1, \ldots, m − 1\} \). Each key is associated with a value.
- The set of keys (or their values) is changing
- We want to answer queries of the form:
  - Does this key exist in this dynamic set?
  - What is the value associated with a given key?
Outline

1. Setting

2. Direct-Address Table

3. Hash Table

4. Open Addressing

5. Designing Hash Functions
Key Assumption

$m$ the number of keys is not too large
Direct-Address Table

Idea

- Use an array $T[0..m-1]$
- Each slot in the array corresponds to a key in universe $U$
- For a given $k$ either $T[k] = \text{NIL}$ or $T[k]$ contains the value
Pseudocode

DIRECT-ADDRESS-SEARCH(\(T, k\))
1 \textbf{return} \(T[k]\)

DIRECT-ADDRESS-INSERT(\(T, k\))
1 \textbf{return} \(T[x.key] = x\)

DIRECT-ADDRESS-DELETE(\(T, k\))
1 \textbf{return} \(T[x.key] = \text{NIL}\)

Time complexity of all operations \(O(1)\)
Outline

1 Setting

2 Direct-Address Table

3 Hash Table

4 Open Addressing

5 Designing Hash Functions
Issues with Direct-Address Tables

- If $U$ is large and hence storing $T$ of size $|U|$ may be impractical.
- If the set of keys $K$ is small relative to the size of $U$, then we waste lots of memory.
A hash function $h(\cdot)$ maps from the universe $U$ of keys $k$ into slots of the hash table $T[0..m-1]$.

Typically $m \ll |U|$. 

Hash Function
One Issue: Collision
Hash Table

Chaining

**CHAINED-HASH-INSERT**(*T, k*)

1. insert x at the head of the list *T [h (x.key)]*

**CHAINED-HASH-SEARCH**(*T, k*)

1. search for an element with key k in list *T [h (k)]*

**CHAINED-HASH-DELETE**(*T, k*)

1. delete x from the list *T [h (x.key)]*
Worst Case Time Complexity

- Insertion: $O(1)$
- Deletion: $O(n)$
- Searching: $O(n)$
Notation:
- $T$ hash table
- $m$ slots
- $n$ elements
- $\alpha$ is the load factor = $n/m$
- $n_j$ for $j \in \{0, 2, \ldots, m - 1\}$ be length of list $T[j]$

Key question: How does the hash function $h$ distribute the keys?
Uniform Hashing

- Suppose \( h \) maps the keys uniformly at random
- \( \mathbb{E}[n_j] = \sum_j n_j/m = n/m = \alpha \)
Unsuccessful Search

**Theorem**

*In a hash table in which collisions are resolved by chaining, an unsuccessful search takes average-case time \( \Theta(1 + \alpha) \) under the assumption of simple uniform hashing.*

**Proof**

- Time to compute the hash function is \( O(1) \)
- Each list has expected length \( \alpha \)
- You need to search through the entire list before you conclude that the element is not present in the hash table
Successful Search

**Theorem**

In a hash table in which collisions are resolved by chaining, an successful search takes average-case time $Θ(1 + α)$ under the assumption of simple uniform hashing.

**Proof**

- Time to compute the hash function is $O(1)$
- Each list has expected length $α$
- You need to traverse one more than the number of elements that appear before $x$ in the list $T[h(x)]$
- New elements are placed at the front of the list
- All the elements you need to traverse were inserted after $x$ was inserted
Proof Continued

- $x_i$ with key $k_i$ inserted at time $i$
- $X_{ij} = I\{h(k_i) = h(k_j)\}$
- $\Pr[X_{ij} = 1] = \frac{1}{m}$. Why?

Number of elements in a successful search:

$$\mathbb{E} \left[ \frac{1}{n} \sum_{i=1}^{n} \left( 1 + \sum_{j=i+1}^{n} X_{ij} \right) \right] = \frac{1}{n} \sum_{i=1}^{n} \left( 1 + \sum_{j=i+1}^{n} \frac{1}{m} \right)$$

$$= 1 + \frac{1}{nm} \sum_{i=1}^{n} (n - i)$$

$$= 1 + \frac{1}{nm} \left( n^2 - \frac{n}{n+1} \right)$$

$$= 1 + \frac{n - 1}{2m} = 1 + \frac{\alpha}{2} - \frac{\alpha}{2n}.$$
Outline

1. Setting
2. Direct-Address Table
3. Hash Table
4. Open Addressing
5. Designing Hash Functions
Open Addressing

Basic Idea

- Every key is contained within the table itself
- To insert
  - Keep *probing* until you find an empty slot
  - Sequence of positions probed depends on the key
- The hash function

\[ h : U \times \{0, 1, \ldots, m - 1\} \rightarrow \{0, 1, \ldots, m - 1\} \]
**Open Addressing**

Insert

```plaintext
HASH-INSERT(T, k)
1   i = 0
2   repeat
3     j = h(k, i)
4     if T[j] == NIL
5       T[j] = k
6       return j
7   else i = i + 1
8   until i == m
9   error "hash table overflow"
```
Search

**HASH-SEARCH**\( (T, k) \)

1. \( i = 0 \)
2. repeat
3. \( j = h(k, i) \)
4. if \( T[j] == k \)
5. return \( j \)
6. \( i = i + 1 \)
7. until \( T[j] == NIL \) or \( i == m \)
8. return NIL
Handling Deletions

- Mark a slot as **DELETED**
- Modify search and insert to take this into account
**Theorem**

Given an open-address hash table with load factor $\alpha = n/m < 1$, the expected number of probes in an unsuccessful search is at most $1/(1 - \alpha)$, assuming uniform hashing.
Proof

- Let $A_i$ denote the event that an $i$-th probe occurs and it hits an occupied slot
- $\Pr \{A_1 \cap A_2 \cap \ldots A_{i-1}\} = \Pr \{A_1\} \cdot \Pr \{A_2|A_1\} \cdot \ldots \Pr \{A_{i-1}|A_1 \cap A_2 \ldots A_{i-2}\}$
- $\Pr \{A_1\} = \frac{n}{m}$
- $\Pr \{A_j|A_1 \cap A_2 \cap A_3 \ldots \cap A_{j-1}\} = \frac{n-j+1}{m-j+1} \leq \frac{n}{m}$ (since $n < m$)
- Therefore

$$\Pr \{X \geq i\} = \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \ldots \frac{n-i+2}{m-i+2} \leq \left(\frac{n}{m}\right)^{i-1} = \alpha^{i-1}$$
Proof

\[ E[X] \leq \sum_{i=1}^{\infty} \Pr\{X \geq i\} \leq \sum_{i=1}^{\infty} \alpha^{i-1} = \sum_{i=0}^{\infty} \alpha^i = \frac{1}{1 - \alpha}. \]
Corollary

Inserting an element into an open-address hash table with load factor $\alpha$ requires at most $\frac{1}{1-\alpha}$ probes on the average, assuming uniform hashing.
Corollary

Given an open address hash table with load factor $\alpha < 1$, the expected number of probes in a successful search is at most

$$\frac{1}{\alpha} \ln \frac{1}{1 - \alpha}$$
Proof

- Number of probes = Number of probes made during insertion
- Number of probes made during the insertion of $i$-th element is
  \[
  \frac{1}{1 - i/m} = \frac{m}{m-i}
  \]
- Average over all the $n$ keys in the hash table
  \[
  \frac{1}{n} \sum_{i=0}^{n-1} \frac{m}{m-i} = \frac{m}{n} \sum_{i=0}^{n-1} \frac{1}{m-i}
  \]
  \[
  = \frac{1}{\alpha} \sum_{k=m-n+1}^{m} \frac{1}{k}
  \]
  \[
  \leq \frac{1}{\alpha} \ln \frac{m}{m-n} = \frac{1}{\alpha} \ln \frac{1}{1 - \alpha}.
  \]
Outline

1 Setting
2 Direct-Address Table
3 Hash Table
4 Open Addressing
5 Designing Hash Functions
Hashing by Division

\[ h(k) = k \mod m \]

- Need to pay some attention to what value of \( m \) is chosen
Hashing by Multiplication

\[ h(k) = \lfloor m(kA \mod 1) \rfloor \]
Universal Hashing

\[ h(k) = ((ak + b) \mod p) \mod m \]
Hash Functions for Open Addressing

- Linear Probing:

\[ h(k, i) = (h'(k) + i) \mod m \]
Hash Functions for Open Addressing

- Quadratic Probing:

\[ h(k, i) = (h'(k) + c_1i + c_2i^2) \mod m \]
Double Hashing for Open Addressing

- Double Hashing:
  \[ h(k, i) = (h_1(k) + ih_2(k)) \mod m \]

- For instance
  \[
  h_1(k) = k \mod m \\
  h_2(k) = 1 + (k \mod m')
  \]
Questions?