COMP251: Hashing

Jérôme Waldispühl
School of Computer Science
McGill University

Based on (Cormen et al., 2002)
Problem Definition

Table $S$ with $n$ records $x$:

We want a data structure to store and retrieve these data.

Operations:

- $\text{insert}(S,x) : S \leftarrow S \cup \{x\}$
- $\text{delete}(S,x) : S \leftarrow S \setminus \{x\}$
- $\text{search}(S,k)$
Direct Address Table

- Each slot, or position, corresponds to a key in $U$.
- If there is an element $x$ with key $k$, then $T[k]$ contains a pointer to $x$.
- If $T[k]$ is empty, represented by NIL.

All operations in $O(1)$, but if $n$ (#keys) < $m$ (#slots), lot of wasted space.
Hash Tables

- Reduce storage to $O(n)$ keys.
- Resolve conflicts by chaining.
- Search time in $O(1)$ time in average, but not the worst case.

Hash function: $h : U \rightarrow \{0, 1, \ldots, m - 1\}$
Analysis of Hashing with Chaining

**Insertion:** $O(1)$ time (Insert at the beginning of the list).

**Deletion:** Search time + $O(1)$ if we use a double linked list.

**Search:**

- **Worst case:** Worst search time to is $O(n)$.
  
  Search time = time to compute hash function +
  
  time to search the list.

  Assuming the time to compute hash function is $O(1)$.

  Worst time happens when all keys go the same slot (list of size $n$),
  and we need to scan the full list => $O(n)$.

- **Average case:** It depends how keys are distributed among slots.
Average case Analysis

Assume a simple uniform hashing: $n$ keys are distributed uniformly among $m$ slots.

Let $n$ be the number of keys, and $m$ the number of slots.

Average number of element per linked list?

Load factor: $\alpha = \frac{n}{m}$

**Theorem:**

The expected time of a search is $\Theta(1 + \alpha)$.

Note: $O(1)$ if $\alpha < 1$, but $O(n)$ if $\alpha$ is $O(n)$. 
Average case Analysis

**Theorem:**
The expected time of a search is $\Theta(1 + \alpha)$.

**Proof?**

Distinguish two cases:

- search is unsuccessful
- search is successful
Unsuccessful search

• Assume that we can compute the hash function in $O(1)$ time.
• An unsuccessful search requires to scan all the keys in the list.

Search time = $O(1 + \text{average length of lists})$

Let $n_i$ be the length of the list attached to slot $i$.

Average value of $n_i$?

$$E(n_i) = \alpha = \frac{n}{m} \quad \text{(Load factor)}$$

$$\Rightarrow O(1) + O(\alpha) = O(1 + \alpha)$$
Successful search

- Assume the position of the searched key \( x \) is equally likely to be any of the elements stored in the list.
- New keys inserted at the head of the list \( \Rightarrow \) Keys scanned \textit{after} finding \( x \) have been inserted in the hash table before \( x \).
- Use indicator to count the number of collisions:

\[
X_{ij} = I \left\{ h(k_i) = h(k_j) \right\}; \quad E(X_{ij}) = \frac{1}{m} \quad \text{(probability of a collision)}
\]
Successful search

number of keys inserted after $x = 1 + \sum_{j=i+1}^{n} X_{ij}$

expected value over all slots $= E \left[ \frac{1}{n} \sum_{i=1}^{n} \left( 1 + \sum_{j=i+1}^{n} X_{ij} \right) \right]$

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

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

Search time:

$\Theta \left( 1 + 1 + \frac{\alpha}{2} - \frac{\alpha}{2n} \right) = \Theta(1 + \alpha)$
Choosing a hash function

Properties:
1. Uniform distribution of keys into slots
2. Regularity in key disturb should not affect uniformity.

List of functions:
• Division method
• Multiplication methods
• Open addressing:
  • Linear probing
  • Quadratic probing
  • Double hashing
Each integer $x$ accepts an unique decomposition $x = \sum_{i} a_i \cdot 2^i$ where $0 \leq a_i < 2$

Example: $x = 11 = 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3$

The binary number representation of an integer $x$ is its (reversed) sequence of $a$’s.

Example: $x = 11 \rightarrow \begin{array}{cccc}
2^3 & 2^2 & 2^1 & 2^0 \\
1 & 0 & 1 & 1 \\
\end{array} \rightarrow 1011$

**Binary number operations:**

$101101 >> 1 = 10110$ (right shift) : quotient of division by $2^k$

$101101 << 2 = 10110100$ (left shift) : multiplication by $2^k$

$101101 \mod 2^2 = 01$ (modulo $2^k$) : remainder of division by $2^k$
Division Method

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

*d must be chosen carefully!*

Example 1: \( d = 2 \) and all keys are even?
   
   Odd slots are never used...

Example 2: \( d = 2^r \)

\[
\begin{align*}
  k &= 100010110101101011 \\
  r &= 2 \rightarrow 11 \\
  r &= 3 \rightarrow 011 \\
  r &= 4 \rightarrow 1011
\end{align*}
\]

keeps only \( r \) last bits...

Good heuristic: Choose \( d \) prime not too close from a power of 2 or 10.

Note: Easy to implement, but division is slow...
Multiplication method

\[ h(k) = \left( A \cdot k \mod 2^w \right) \gg (w - r) \]

\[ 2^{w-1} < A < 2^w \]
Open addressing

No storage for multiple keys on single slot (i.e. no chaining).

**Idea:** Probe the table.

- Insert if the slot if empty,
- Try another hash function otherwise.

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

Universe of keys      probe number      slot

Constraints:
- $n < m$ (i.e. more slots than keys to store)
- Deletion is difficult

Challenge: How to build the hash function?
Open addressing

Illustration: Where to store key 282?

```
<table>
<thead>
<tr>
<th>index</th>
<th>key</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>355</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>567</td>
</tr>
<tr>
<td>4</td>
<td>233</td>
</tr>
<tr>
<td>5</td>
<td>282</td>
</tr>
<tr>
<td>6</td>
<td>799</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
```

Note: Search must use the same probe sequence.
Linear & Quadratic probing

Linear probing:

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

Note: tendency to create clusters.

Quadratic probing:

\[ h(k,i) = \left( h'(k) + c_1 \cdot i + c_2 \cdot i^2 \right) \mod m \]

Remarks:

- We must ensure that we have a full permutation of \( \langle 0, ..., m-1 \rangle \).
- **Secondary clustering**: 2 distinct keys have the same \( h' \) value, if they have the same probe sequence.
Double hashing

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

Must have \( h_2(k) \) be “relatively” prime to \( m \) to guarantee that the probe sequence is a full permutation of \( \langle 0, 1, \ldots, m - 1 \rangle \).

Examples:
- \( m \) power of 2 and \( h_2 \) returns odd numbers
- \( m \) prime number and \( 1 < h_2(k) < m \)
Analysis of open-addressing

We assume uniform hashing: Each key equally likely to have anyone of the m’s permutations as its probe sequence, independently of other keys.

**Theorem 1:** The expected number of probes in an unsuccessful search is at most $\frac{1}{1-\alpha}$.

**Theorem 2:** The expected number of probes in a successful search is at most $\frac{1}{\alpha} \cdot \log\left(\frac{1}{1-\alpha}\right)$

Reminder: $\alpha = \frac{n}{m}$ is the load factor
Proof for unsuccessful searches

Initial state: \( n \) keys are already stored in \( m \) slots.

Probability that the 1\(^{st} \) slot is occupied: \( \frac{n}{m} \).
Probability that the 2\(^{nd} \) slot is occupied: \( \frac{n-1}{m-1} \).
Probability that the 3\(^{rd} \) slot is occupied: \( \frac{n-2}{m-2} \).

Let \( X \) be the number of unsuccessful probes.

\[
E(X) = 1 + P_{occupied}^1 \cdot \langle \# \text{ probes after } 1 \text{ failed probe} \rangle
\]
\[
E(X) = 1 + \frac{n}{m} \cdot (1 + P_{occupied}^2 \cdot \langle \# \text{ probes after } 2 \text{ failed probes} \rangle)
\]
\[
E(X) = 1 + \frac{n}{m} \cdot \left(1 + \frac{n-1}{m-1} \cdot (1 + P_{occupied}^3 \cdot \langle \# \text{ probes after } 3 \text{ failed probes} \rangle)\right)
\]
\[\vdots\]
\[
E(X) = 1 + \frac{n}{m} \left(1 + \frac{n-1}{m-1} \left(1 + \frac{n-2}{m-2} \left(\ldots \left(1 + \frac{1}{m-n}\right)\right)\right)\right)
\]
Proof for unsuccessful searches

\[ E(X) = 1 + \frac{n}{m} \left( 1 + \frac{n-1}{m-1} \left( 1 + \frac{n-2}{m-2} \left( \ldots \left( 1 + \frac{1}{m-n} \right) \right) \right) \right) \]

Since \( \frac{n-k}{m-k} \leq \frac{n}{m} = \alpha \), we have the upper bound:

\[ \leq 1 + \alpha \left( 1 + \alpha \left( 1 + \alpha \left( \ldots \left( 1 + \alpha \right) \right) \right) \right) \leq 1 + \alpha + \alpha^2 + \ldots = \sum_{i=1}^{\infty} \alpha^i = \frac{1}{1-\alpha} \]
Consequences

Corollary
The expected number of probes to insert is at most $1/(1 - \alpha)$.

Interpretation:
• If $\alpha$ is constant, an unsuccessful search takes $O(1)$ time.
• If $\alpha = 0.5$, then an unsuccessful search takes an average of $1/(1 - 0.5) = 2$ probes.
• If $\alpha = 0.9$, takes an average of $1/(1 - 0.9) = 10$ probes.

Proof of Theorem on successful searches: See [CLRS, 2009].
Universal Hashing

• Set-up: We solve collision by chaining.

• A malicious adversary who has learned the hash function chooses keys that all map to the same slot, giving worst-case behavior.

• Defeat the adversary using **Universal Hashing**
  – Use a different random hash function each time.
  – Ensure that the random hash function is independent of the keys that are actually going to be stored.
  – Ensure that the random hash function is “good” by carefully designing a class of functions to choose from:
    • Design a universal class of functions.
Universal Set of Hash Functions

• A finite collection of hash functions $H$ that maps a universe $U$ of keys into the range $\{0, 1, \ldots, m-1\}$ is **universal** if:

  for each pair of distinct keys $x, y \in U$, the number of hash functions $h \in H$ for which $h(x) = h(y)$ is $\leq |H|/m$.

• For a hash function $h$ chosen randomly from $H$, the chance of a collision between two keys is $\leq 1/m$.

Universal hash functions give good hashing behavior.
Example of Universal Hashing

• The size of the table $m$ is a prime,
• We write a key $x$ in bytes s.t. $x = <x_0,\ldots, x_r>$,
• $a = <a_0,\ldots, a_r>$ denotes a sequence of $r+1$ elements randomly chosen from \{0, 1, \ldots, m - 1\}.

The class $H$ defined by:

$$H = \bigcup_a \{h_a\} \text{ with } h_a(x) = \sum_{i=0}^{r} a_i x_i \mod m$$

is an universal function.
Cost of Universal Hashing

**Theorem:**
Using chaining and universal hashing on key k:

- If k is not in the table T, the expected length of the list that k hashes to is \( \leq \alpha \).
- If k is in the table T, the expected length of the list that k hashes to is \( \leq 1+\alpha \).

**Proof:**

\( X_k = \) # of keys (\( \neq k \)) that hash to the same slot as k.

\( C_{kl} = I\{h(k)=h(l)\}; \ E[C_{kl}] = Pr\{h(k)=h(l)\} \leq 1/m. \)

\[
X_k = \sum_{l \in T \setminus \{k\}} C_{kl}, \text{ and } E[X_k] = E\left[\sum_{l \in T \setminus \{k\}} C_{kl}\right] = \sum_{l \in T \setminus \{k\}} E[C_{kl}] \leq \sum_{l \in T \setminus \{k\}} \frac{1}{m}
\]

If \( k \not\in T \), \( E[X_k] \leq n/m = \alpha. \)

If \( k \in T \), \( E[X_k] + 1 \leq (n-1)/m + 1 = 1 + \alpha - 1/m < 1 + \alpha. \)
Proof (universal hashing function)

Let \( X = \langle x_0, x_1, \ldots, x_r \rangle \) and \( Y = \langle y_0, y_1, \ldots, y_r \rangle \) be 2 distinct keys. They differ at (at least) one position. WLOG let 0 be this position.

For how many \( h \) do \( X \) and \( Y \) collide?

\[
\sum_{i=0}^{r} a_i x_i \equiv \sum_{i=0}^{r} a_i y_i \pmod{m}
\]

\[
\sum_{i=0}^{r} a_i (x_i - y_i) \equiv 0 \pmod{m}
\]

For any choice of \( < a_1, a_2, \ldots, a_r > \) there is only one choice of \( a_0 \) s.t. \( X \) and \( Y \) collide.

\#\{h that collide\} = m \times m \times \ldots \times m \times 1
= m^r = |H|/m

\[
a_0 (x_0 - y_0) \equiv - \sum_{i=1}^{r} a_i (x_i - y_i) \pmod{m}
\]

\[
a_0 \equiv \left( - \sum_{i=1}^{r} a_i (x_i - y_i) \right) \cdot (x_0 - y_0)^{-1} \pmod{m}
\]