Hashing

- Dictionaries
- Hashing with chaining
- Hash functions
- Linear Probing
Hashing

- Dictionaries
- Hashing with chaining
- Hash functions
- Linear Probing
**Dictionaries**

**Dictionary:** Maintain a dynamic set $S$. Every element $x$ has a key $x.key$ from a universe $U$, along with satellite data $x.data$.

**Operations:**

- **search($k$)** determine whether an element $x$ with $x.key = k$ exists, and return it.
- **insert($x$)** add $x$ to the set $S$.
- **delete($x$)** remove $x$ from the set $S$.

---

[Diagram of a set $S$ with elements $U$, $\text{Anchor}$, $\text{Tree}$, $\text{Hamster}$, $\text{Knife}$, $\text{Lever}$, and $\text{Mallet}$.]

- insert($\text{Hamster}$)
- search($\text{Tree}$)
- search($\text{Mallet}$)
Dictionaries

Applications
- Basic data structure for representing a set
- Used in many algorithms and data structures

Challenge How can we solve the dictionary problem using current techniques?
Dictionaries - solution with a chained list - too slow!

Time:
- \(\text{search}(k) - O(|S|)\) time (search through all elements)
- \(\text{insert}(x) - O(1)\) time (insert at head of list)
- \(\text{delete}(x) - O(1)\) time to change pointers.

Space: \(O(|S|)\) space.
Dictionaries - solution with an array - too large!

- $A$ is an array of size $U$
- Save $x$ on the position $A[x.key]$ in $A$.

- $\text{search}(k)$ - $O(1)$ time to return $A[k]$
- $\text{insert}(x)$ - $O(1)$ time to set $A[x.key] = x$
- $\text{delete}(x)$ - $O(1)$ time to set $A[x.key] = \text{null}$. 

**Space:** $O(|U|)$

**Exercise:** When is this a problem?
Dictionaries - two dissatisfactory solutions

<table>
<thead>
<tr>
<th>Data structure</th>
<th>Search</th>
<th>Insert</th>
<th>Delete</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chained list</td>
<td>$O(</td>
<td>S</td>
<td>)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Array</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(</td>
</tr>
</tbody>
</table>

**Challenge:** Can we do better?
Hashing

- Dictionaries
- Hashing with chaining
- Hash functions
- Linear Probing
Hashing with chaining

**Idea:** Use a hash function $h : U \rightarrow \{0, \ldots, m\}$ where $m = O(|S|)$.

- Maintain an array $A$ of size $m$,
- Each entry of the array points to a chained list,
- The element $x$ is stored somewhere in the chained list at $A[h(x.key)]$.

**Collision:** When $m < |U|$, then even when $x.key \neq y.key$, we risk $h(x.key) = h(y.key)$. We call this a collision.

We want $h$ such that there are few collisions.

**Hash** (vb tr) “to confuse, muddle, or mess up”.
Hashing with chaining

How it works.

- **insert( учитывающий)**
  \( h( учитывающий) = 0 \)

- **insert( лопатка)**
  \( h( лопатка) = 7 \)

- **search( 🐿)**
  \( h( 🐿) = 15 \)

- **search( 🦊)**
  \( h( 🦊) = 2 \)
Hashing with chaining

How it works.

search(k) - search through $A[k]$ ’s list for $k$.
insert(x) - insert $x$ in $A[h(x.key)]$ ’s list.
delete(x) - delete $x$ from list.

Time:

search(k) - $O(|\text{list’s length}|)$ time
insert(x) - $O(1)$ time (at head of list)
delete(x) - $O(1)$ time to change pointers.

Plus the time it takes to calculate $h(x.key)$

Space:

$O(m + |S|) = O(|S|)$
Hashing with chaining - Exercise

Insert the following keys $K$ in a hash table of size 9 using hashing with chaining using the hash function

$$h(k) = k \mod 9$$

$K = 5, 28, 19, 15, 20, 33, 12, 17, 10$

How long is the longest list?
Imagine there’s a uniform hash function $h : U \rightarrow \{0, \ldots, m - 1\}$. 
Uniform hashing

Definition (Load factor)

\[ \alpha = \frac{|S|}{m}. \] The average length of lists.

\[ m = \Theta(|S|) \Rightarrow \alpha = \Theta(1). \]

Dream world: Imagine there’s a hash function \( h \) that is

- computable in \( O(1) \) time, and
- For any \( x \in U: h(x) \) is independent uniformly random in \( \{0, \ldots, m-1\} \).

Then:

- Expected length of list = \( \alpha \).
- \( \Rightarrow \) search(\( k \)) in \( O(\alpha) = O(1) \) time.
- Search, Insert, Delete: \( O(1) \) time.
- \( O(|S|) \) space.
Dictionaries - two dissatisfactory and one imaginary

<table>
<thead>
<tr>
<th>Data structure</th>
<th>Search</th>
<th>Insert</th>
<th>Delete</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chained list</td>
<td>$O(</td>
<td>S</td>
<td>)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Array</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(</td>
</tr>
<tr>
<td>Hashing with chaining</td>
<td>$O(1)^\dagger$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(</td>
</tr>
</tbody>
</table>

$^\dagger$: Expected running time. Assuming uniform hashing.

**Challenge**: Find a real-life hash function that works.
Hashing

- Dictionaries
- Hashing with chaining
- Hash functions
- Linear Probing
Universal hashing

**Goal:** Avoid collisions \( h(y) = h(x) \) for \( x \neq y \).
If \( h(x) \) and \( h(y) \) are independent uniform random, then

\[
\Pr [ h(x) = h(y) ] = 1/m
\]

**Definition (Universal hashfunction)**

\( h \) is universal if for any \( x, y \in U \) with \( x \neq y \),
\[
\Pr [ h(x) = h(y) ] \leq 1/m
\]

If \( h \) is universal, what is the expected size of the list at \( A[h(x)] \)?

\[
\sum_{y \in S} \Pr [ h(y) = h(x) ] \leq 1 + \sum_{y \in S\setminus\{x\}} \frac{1}{m} \leq 1 + \frac{|S|}{m} = O(1)
\]

All operations in (expected) \( O(1) \) time!
Hash function: multiply-mod-prime

\[ p \text{ is a prime } > |U|. \]

\[ h_{a,b}(x) = (((ax + b) \mod p) \mod m) \]

- Select \( a \in \{1, \ldots, m-1\} \) and \( b \in \{0, \ldots, m-1\} \) independently uniformly at random.
- Use \( h(x) = h_{a,b}(x) = \pi(\tilde{h}_{a,b}(x)) \) as hash function.
- \( \tilde{h}_{a,b} \) is collision free because \( a \neq 0 \)
- \( \pi \) introduces collisions when \( m < p \)
- Given \( x \neq y \), then \( \Pr[h(x) = h(y)] < \frac{1}{m} \)
Hash function: multiply-shift

Assume $|U|$ and $m$ are powers of 2.
E.g $|U| = 2^w = 2^{64}$ and $m = 2^L$.

\[
\begin{array}{c|c|c|c}
ax & w-1 & w-L & 0 \\
\hline
(a*x)>>w-L
\end{array}
\]

- Select odd $a \in \{1, 3, 5, \ldots, |U| - 1\}$
- $h_a(x) = \lfloor (ax \mod 2^w) / 2^{w-L} \rfloor$
- Implementation: return $(a*x)>>64(64-L)$;
- $\Pr[h_a(x) = h_a(y)] \leq \frac{2}{m}$
Hashing

- Dictionaries
- Hashing with chaining
- Hash functions
- Linear Probing
**Analogies**

**Chaining:** Like a desk of drawers. Must linear-search through drawer no. 8 to find 🐭

**Linear Probing:** Like a shelf. No space for 🔨 at $h( sacrificer ) = 7$, So insert 🔨 at the nearest vacant spot to the right.
Linear probing

- Maintain an array of size $m$
- **Idea**: Save $x$ in $A[x.key]$
- **Challenge**: Collisions.
Linear probing

- Maintain an array of size $m$
- A *cluster* is a sequence of consecutive non-empty positions.
- Store $x$ in $A[x.key]$, or somewhere in the cluster containing $x.key$, to the right of $x.key$.

**Example:**
- Insert(โย). $h(โย) = 8$.
- Delete(仓). $h(仓) = 7$, $h(🔨) = 7$.
- Search(🔮). $h(🔮) = 2$.

**Space:** $m = O(|S|)$. **Time:** $O(|\text{cluster}|)$. ← $O(1)$ for some hash functions $h$. 
Linear Probing

Huge advantage: Linear Probing is cache efficient.

![Graph showing comparison between Chaining and Linear Probing]

http://en.wikipedia.org/wiki/Hash_table
Hashing

- Dictionaries
- Hashing with chaining
- Hash functions
- Linear Probing