Open Addressing

Use first empty slot defined by probe sequence \( \langle S_0, \ldots, S_{m-1} \rangle \)

Let \( x = \) element, \( \text{hashCode()} \)

\[ S_0 = \text{hash}(x) \]

\[ S_i = (S_{i-1} + \text{stepHash}(x)) \mod m \]

\( \text{def of probe seq} \)
<table>
<thead>
<tr>
<th>Slot</th>
<th>Probe</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>S₁</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>S₆</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>S₃</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>S₀</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>S₅</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>S₂</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>S₇</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>S₄</td>
</tr>
</tbody>
</table>

$m = 8$

hash table size

$\text{hash}(x) = 3$

Step: Hash($x$) = 5
Locating an element - Go through hash table in order of the probe sequence until you either reach the desired element or an empty (unused) slot is reached.

Adding an element

Go through hash table in order given by probe sequence until the element is found or an empty slot $s$ is reached. If the element was not there, we will place it at slot $s$: $T[s] = e$.
Let's look at an example.

<table>
<thead>
<tr>
<th>element e</th>
<th>A</th>
<th>B</th>
<th>E</th>
<th>F</th>
<th>H</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>hash(e.hashCode())</td>
<td>5</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>stepHash(e.hashCode())</td>
<td>7</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Insert in order: \{A, B, E, F, H, J\}

Table:

```
 0 1 2 3 4 5 6 7
 B F H E A J
```

`m = 8`
Care must be selected in how \( m \) and \( \text{stepHash}(x) \) relate. Why?

\[
m = 8 \\
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}
\]

\[
\text{hash} = i \\
\text{stepHash} = 4
\]

Avoid this bad behavior and guarantee probe sequence is a permutation of \( \langle 0, \ldots, m-1 \rangle \) by making \( m + \text{stepHash} \) relatively prime.

Pick \( m \) to be a power of 2, make \( \text{stepHash} \) odd.
How can you remove an element?

Let's delete A.
Now search for F.

Replace reference to A, by a ref. to DELETED sentinel.

Search must continue until EMPTY is reached (or element is found)
Deleting an element

Problem: hash table can fill up with slots "marked as deleted" (ref to DELETED)

Partly address this problem by re-using a deleted slot along probe sequence when inserting a new element.

In a internal locate method (to check if element is in set), remember the first DELETED slot found & return that (or first empty slot if there were no deleted slots).
Actual load versus target load

Load factor $\alpha = \frac{n + d}{m}$

# elements in Set

# slots marked as deleted

$m$ = table size

During an unsuccessful search, $\alpha$ is fraction of slots that will cause search to continue.

desired load factor $\alpha^*$ (e.g., $1/2$)

actual load is current value of $\frac{n + d}{m}$
Goal: keep $\lambda$ close to $\lambda^*$

Limit frequency of resizing $\leftarrow$ expensive

Double table size (m) when

$\lambda$ reaches $\frac{1 + \lambda^*}{2}$ halfway between $\lambda^* + 1$

$\lambda^* = \frac{1}{2}$, resize when $\lambda = \frac{3}{4}$

Hash functions change $\Rightarrow$ you must rebuild by re-inserting all elements (in order)

Go through slots, move to next if Empty or DELETED, reinsert elements in new table
The hash table could be oversized (+ cluttered with deleted slots)

half table size when \( \frac{n}{m} \) drops down to \( \frac{2^k}{2} \)

reduce hash table size by a factor of 2.
Analysis

\[ E\left[ \text{# probes in an unsuccessful search} \right] = 1 \cdot \text{prob. probe 0 occurs} + 1 \cdot \text{prob. probe 1 occurs} + \ldots \]

\[ \leq 1 + \lambda + \lambda^2 + \lambda^3 + \ldots \]

Probe 0 always occurs, \( n \leq m \) prob. that a collision occurs on probe 2, very, very close to prob. probe 2 occurs.

\[ \frac{n}{m} \cdot \frac{(n-1)}{(m-1)} \leq \lambda^2 \]

\[ \sum_{i=0}^{\infty} \lambda^i = \frac{1}{1-\lambda} \]

\[ E\left[ \text{# probes in a successful search} \right] = \frac{1}{2} \ln \frac{1}{1-\lambda} \]
Tradeoff between space and search time

As we increase $\lambda_a$, use less space but higher expected cost per search.

This is a pretty large value $\frac{1}{\lambda_a} = \frac{7}{8}$

As we decrease $\lambda_a$, use more space but have lower expected cost per search.

$\lambda_a = \frac{1}{4}$

$\frac{1}{1 - \frac{1}{4}} = \frac{4}{3}$ space usage
Separate Chaining

Have a list referenced by each slot of hash table that holds all elements that hash to that slot (one hash function)

\[ e \rightarrow \text{hash}() \rightarrow \text{list} \]

insert \( e \) add it to list \( \text{table}[\text{hash}(x)] \)

locate \( e \) search within list \( \text{table}[\text{hash}(x)] \)

remove \( e \) remove \( e \) from list \( \text{table}[\text{hash}(x)] \)
Resizing hash table

No absolute limit on $n/m$ (could go arbitrarily high) but cost is too high

Open addressing $d < 1$

Resize upward when $d$ reaches $2d^*$

Resize downward when $d$ reaches $d^*/2$
Analysis

Expected cost for unsuccessful search

\[ \frac{n}{m} = \lambda \]

**Open addressing**

\[ \frac{1}{1-\lambda} = 1+\lambda+\lambda^2+\ldots \]

\[ E\left[\text{probes in an unsuccessful search}\right] = 1+\lambda \]

\[ E\left[\text{probes in a successful search}\right] = 1+\frac{\lambda}{2} - \frac{\lambda}{2n} \]
Summary of Set ADT Implementations

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Unsuccessful Search</th>
<th>Successful Search</th>
<th>Approximate Space Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Addressing</td>
<td>1</td>
<td>1</td>
<td>( m )</td>
</tr>
<tr>
<td>Separate Chaining</td>
<td>( 1 + \alpha )</td>
<td>( 1 + \frac{\alpha}{2} - \frac{\alpha}{2n} )</td>
<td>( 2n + m = n \left( 3 + \frac{1}{\alpha} \right) )</td>
</tr>
<tr>
<td>( \alpha = \frac{n}{m} )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \alpha = \frac{n}{m} )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Separate Chaining</td>
<td>1.5</td>
<td>( \approx 1.25 )</td>
<td>( 4n )</td>
</tr>
<tr>
<td>( \alpha = \frac{3}{4} )</td>
<td>1.75</td>
<td>( \approx 1.375 )</td>
<td>( 3 \frac{1}{3}n )</td>
</tr>
<tr>
<td>Separate Chaining</td>
<td>2</td>
<td>( \approx 1.5 )</td>
<td>( 3n )</td>
</tr>
<tr>
<td>( \alpha = 1 )</td>
<td>4</td>
<td>( \approx 2.5 )</td>
<td>( 2 \frac{1}{3}n )</td>
</tr>
<tr>
<td>Separate Chaining</td>
<td>( \alpha = 3 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open Addressing</td>
<td>( \frac{1}{1 - \alpha} )</td>
<td>( \frac{1}{\alpha} \ln \frac{1}{1 - \alpha} )</td>
<td>( m = \frac{(n + d)}{\alpha} )</td>
</tr>
<tr>
<td>( \alpha = \frac{1}{4} )</td>
<td>4/3</td>
<td>( \approx 1.15 )</td>
<td>( 4(n + d) )</td>
</tr>
<tr>
<td>Open Addressing</td>
<td>2</td>
<td>( \approx 1.39 )</td>
<td>( 2(n + d) )</td>
</tr>
<tr>
<td>( \alpha = \frac{1}{2} )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open Addressing</td>
<td>4</td>
<td>( \approx 1.85 )</td>
<td>( \frac{4}{3}(n + d) )</td>
</tr>
<tr>
<td>( \alpha = \frac{3}{4} )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( m = \| U \| \)
Unsuccessful Search Cost as a Function of Load

\[
\frac{1}{1- \lambda} = 1 + \lambda + \lambda^2 + \lambda^3 + \ldots
\]
Comparison of Search Cost for Space usage of $\sim 3n$

$d = 0$