Set ADT Implementations

Set holds \{\text{elements}\}

Mapping \{ \text{tag} \rightarrow \text{element} \}
(Tagged Set, Map)

BucketMapping \{ \text{tag} \rightarrow \{\text{elements}\} \}
Direct Addressing

insert, locate, remove

000
001

314 ➔ Object for area code 314
(bucket/collection of all countries with this area code)

999 array ➔ called a table
Key here

every element you might possibly
insert into set has a dedicated

index slot in the table

\textit{worst-case}

\begin{align*}
\text{insert} & : \text{table}[\text{slot}] = \text{element} \quad \Theta(1) \\
\text{locate} & : \text{access table}[\text{slot}] \quad \Theta(1) \\
\text{remove} & : \text{table}[\text{slot}] = \text{null} \quad \Theta(1)
\end{align*}
What is the big limitation?

Size of table is as big as Universe of all possible elements \( U \) that might be inserted.

Consider a univ of 5000 students where SS # is the id #.

\( 10^9 \)
Direct addressing is only a reasonable choice (in terms of space usage) when roughly $n > \frac{|U|}{4}$

# elements held in set
We want high efficiency of direct addressing but we can't waste so much space.

- $10^a$
- $S_s$
- $S$
- $F$

Table: 5000 students

Must have lots of elements in universe map to same slot.

Collision
Hash Function

function that maps from hashcode

to \{0, \ldots, m-1\}
Pick a hash function to be a mathematical function that maps a random integer to $0, \ldots, m-1$.

One thought: hashcode mod $m$ \text{division method} 

Problem: real data can have patterns in low order digits

Multiplication method - multiply by irrational \#
\[ \text{hash code} \times \text{irrational number} \]
The remaining question is how to handle collisions.

Create a chain for each slot

Bucket Mapping
Other option:

Find a different empty slot

cause clustering

Second hash function to give step size
Open Addressing

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

Let \(x = \text{element, hash code}()\)

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

\[ i = 1, \ldots, m-1 \quad S_i = (S_{i-1} + \text{stepHash}(x)) \mod m \]
m = 8

hash table size

hash(x) = 3
step Hash(x) = 5

IF m + step size are relatively prime then probe sequence will be a permutation of slots
Provided code makes m a power of 2 make step size odd
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 - element is not in the set

Go through hash table in order given by probe sequence until the element is found or an empty slot $s$ is reached.

element was not there & we will place it at slot $s$: $table[s] = e$
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>$\text{hash}(e.\text{hashCode}())$</td>
<td>5</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>$\text{stepHash}(e.\text{hashCode}())$</td>
<td>7</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Table $m=8$

- Insert in order $A, B, E, F, H, J$

- Elements stored: $B, E, H, E, A$

- Empty slot at index 7
Care must be selected in how $m$ and $\text{stepHash}(x)$ relate. Why?
How can you remove an element?

Let's delete A.

Now search for F.

<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>

**table**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>F</td>
<td>H</td>
<td>E</td>
<td>A</td>
<td></td>
<td>J</td>
<td></td>
</tr>
</tbody>
</table>

EMPTY ← DELETED
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 an internal locate method (to check if element is in set), remember the 1st DELETED slot found & return that (or first empty slot if there were no deleted slots).
\[ \alpha = \text{load factor} = \frac{\text{fraction of slots in use}}{M} \]

\[ N \quad \text{# tags/element} \]

\[ \frac{\text{mapping}}{\text{set}} \]

\[ \text{bucket collection} \]

\[ n + d \]

\[ M \quad \text{hash table size} \]

\[ d \quad \text{# slots “marked as deleted”} \]
Actual load versus target load

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

$\#$ elements in set

$\#$ slots marked as deleted

$\text{table size}$

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

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

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

Limit frequency of resizing ← expensive

Double table size (m) when

$\lambda$ reaches $\left\lfloor \frac{1 + \lambda^*}{2} \right\rfloor$ halfway between $\lambda^* + 1$

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

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

Go through slots, move to next if empty or deleted, re-insert elements in new table
The hash table could be oversized (+ cluttered with deleted slots)

\[ \frac{n}{m} \text{ drops down to } \frac{1}{2} \]

half table size when reduce hash table size by a factor of 2.
Analysis

\[ E\left[ \text{# probes in an unsuccessful search} \right] = 1 \cdot \frac{\text{prob. probe 0 occurs}}{1} + 1 \cdot \frac{\text{prob. probe occurs}}{2} + \ldots \]

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

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

\[ \alpha = \frac{1}{2}, \quad 2 \]
\[ \alpha = \frac{1}{4}, \quad 4/3 \]
\[ \alpha = \frac{3}{4}, \quad 4 \]
$E\left[ \text{# probes in a successful search} \right] = \frac{1}{2} \ln \frac{1}{1-x}$
Lab 2

acaacgcggtagaggagacc → sequences of k in common

gaca ...

bucket mapping

string

k-mer → { offsets }
Special hardware parallel processors
Filter to find most unsuccessful searches

off-chip hash table