Resolving collisions w/ open addressing assuming $n \leq m$
Resolving collisions w/ open addressing assuming $n \leq m$

The point is to avoid auxiliary linked lists. Use that space for table, (and pointers)
Resolving Collisions w/ Open Addressing assuming $n \leq m$

The point is to avoid auxiliary linked lists. Use that space for table. Instead, create a probe sequence as a function of key value. → permutation of slots to try.
Resolving collisions w/ open addressing assuming \( n \leq m \)

The point is to avoid auxiliary linked lists. Use that space for table. Instead, create a probe sequence as a function of key value. \( \implies \) permutation of slots to try.

\[
\begin{align*}
1 & \quad 36 \\
2 & \quad 43 \\
3 & \quad 78 \\
4 & \quad 5 \\
5 & \quad 103 \\
6 & \quad 2014 \\
7 & \\
8 & \\
9 & \\
10 & \\
11 & \\
\end{align*}
\]

ex: \( h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6. \)
Resolving Collisions w/ Open Addressing assuming \( n \leq m \)

The point is to avoid auxiliary linked lists. Use that space for table. Instead, create a probe sequence as a function of key value.

\[ \text{permutation of slots to try.} \]

ex: \( h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6. \)

Insert(64): Try \( T[9] \): full
Resolving Collisions w/ Open Addressing assuming \( n \leq m \)

The point is to avoid auxiliary linked lists. Use that space for table.
Instead, create a probe sequence as a function of key value.
\( \rightarrow \) permutation of slots to try.

ex: \( h(64) \rightarrow 9, 2, 4, 5, 1, 3, 11, 7, 10, 5, 6 \).

Insert(64): Try \( T[9] \): full
\( \& \) Try \( T[2] \): full
Resolving collisions w/ open addressing assuming $n \leq m$

The point is to avoid auxiliary linked lists. Use that space for table. Instead, create a probe sequence as a function of key value.  

$\rightarrow$ permutation of slots to try.

**Example:** $h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6$.

`Insert(64)`:
- Try $T[9]$: full
Resolving collisions w/ open addressing assuming \( n \leq m \)

The point is to avoid auxiliary linked lists. Use that space for table. Instead, create a probe sequence as a function of key value.

\( \text{Insert}(64) : \) Try \( T[9] \) : full
  \( \text{Try } T[2] \) : full
  \( \text{Try } T[4] \) : full
  \( \text{Try } T[8] \) : ok

\( \text{ex: } \text{h}(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6. \)
Resolving collisions w/ open addressing assuming \( n \leq m \)

The point is to avoid auxiliary linked lists. Use that space for table. Instead, create a probe sequence as a function of key value. → permutation of slots to try.

\[
\text{ex: } h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6.
\]

\[
\text{Insert}(64): \quad \begin{array}{l} 
\text{Try } T[9]: \text{ full} \\
\text{Try } T[2]: \text{ full} \\
\text{Try } T[4]: \text{ full} \\
\text{Try } T[8]: \text{ ok}
\end{array}
\]

Search(64) follows same sequence. Would return "not found" after 4 attempts.
ex: \( h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6. \)

Really this is \( h(k, i) \)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>43</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>78</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>103</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2014</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ex: $h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6$.

Really this is $h(k, i)$

$h(64, 1) = 9$
ex: $h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6$.

Really this is $h(k, i)$

$h(64, 1) = 9$ / $h(64, 2) = 2$
ex: $h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6$. 
Really this is $h(k, i)$. $h(64, 1) = 9 \quad h(64, 2) = 2 \quad h(64, 3) = 4 \quad \text{etc}$
ex: \( h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6. \)

Really this is \( h(k, i) \) 

\[ h(64, 1) = 9 \quad h(64, 2) = 2 \quad h(64, 3) = 4 \quad \text{etc.} \]

**Delete(64):** ?
ex: \( h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6. \)

Really this is \( h(k, i) \)

\[
\begin{align*}
\text{Delete}(64): & \quad h(64, 1) = 9, \text{ occupied by 2014} \\
1 & \quad 36 \\
2 & \quad 43 \\
3 & \quad 78 \\
4 & \quad 5 \\
5 & \quad 103 \\
6 & \quad 64 \\
7 & \quad 2014 \\
8 & \quad \text{cell empty} \\
9 & \quad \text{cell empty} \\
10 & \quad \text{cell empty} \\
11 & \quad \text{cell empty}
\end{align*}
\]
Example: \( h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6. \)

Really this is \( h(k, i) \)

\[
\begin{align*}
h(64, 1) &= 9 \\ h(64, 2) &= 2 \\ h(64, 3) &= 4 \end{align*}
\]
etc.

Delete(64): \( h(64, 1) = 9 \), occupied by 2014

\[
\begin{align*}
h(64, 2) &= 2 \), occupied by 43
\end{align*}
\]
ex: \( h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6. \)

Really this is \( h(k, i) \)

\[
\begin{align*}
h(64, 1) &= 9 & h(64, 2) &= 2 & h(64, 3) &= 4 & \text{etc.}
\end{align*}
\]

**Delete(64):**

\[
\begin{align*}
h(64, 1) &= 9, \text{ occupied by 2014} \\
h(64, 2) &= 2, \text{ occupied by 43} \\
h(64, 3) &= 4, \text{ occupied by 78}
\end{align*}
\]
ex: $h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6$.

Really this is $h(k, i)$

$h(64, 1) = 9$ / $h(64, 2) = 2$ / $h(64, 3) = 4$ / etc.

Delete(64):

$h(64, 1) = 9$, occupied by 2014
$h(64, 2) = 2$, occupied by 43
$h(64, 3) = 4$, occupied by 78
$h(64, 4) = 8$, found 64, DELETE IT.

OK?
ex: \( h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6. \)

Really this is \( h(k, i) \)

\( h(64, 1) = 9 \)/\( h(64, 2) = 2 \)/\( h(64, 3) = 4 \)/etc.

Delete(64):

\( h(64, 1) = 9 \), occupied by 2014
\( h(64, 2) = 2 \), occupied by 43
\( h(64, 3) = 4 \), occupied by 78
\( h(64, 4) = 8 \), found 64, DELETE IT.

what if \( h(103) \rightarrow 4, 8, 2, 7, 1, 10, 11, 3, 5, 9, 6 \)?
ex: \( h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6. \)

Really this is \( h(k, i) \)

\[
\begin{align*}
\ h(64,1) &= 9 / \ h(64,2) &= 2 / \ h(64,3) &= 4 / \ etc. \\
\end{align*}
\]

Delete(64): \( h(64,1) = 9 \), occupied by 2014
\( h(64,2) = 2 \), occupied by 43
\( h(64,3) = 4 \), occupied by 78
\( h(64,4) = 8 \), found 64, DELETE IT.

what if \( h(103) \rightarrow 4, 8, 2, 7, 1, 10, 11, 3, 5, 9, 6 \) ?

Search(103): \( h(103,1) = 4 \), occupied by 78
Example: \( h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6 \).

Really this is \( h(k, i) \) where:
- \( h(64, 1) = 9 \)
- \( h(64, 2) = 2 \)
- \( h(64, 3) = 4 \)

Delete(64):

- \( h(64, 1) = 9 \), occupied by 2014
- \( h(64, 2) = 2 \), occupied by 43
- \( h(64, 3) = 4 \), occupied by 78
- \( h(64, 4) = 8 \), found 64, DELETE IT.

What if \( h(103) \rightarrow 4, 8, 2, 7, 1, 10, 11, 3, 5, 9, 6 \)?

Search(103):
- \( h(103, 1) = 4 \), occupied by 78
- \( h(103, 2) = 8 \), empty: declare 103 not in \( T \).
Example: $h(64) \to 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6$.

Really this is $h(k, i)$

- $h(64, 1) = 9$
- $h(64, 2) = 2$
- $h(64, 3) = 4$

Delete(64):
- $h(64, 1) = 9$, occupied by 2014
- $h(64, 2) = 2$, occupied by 43
- $h(64, 3) = 4$, occupied by 78
- $h(64, 4) = 8$, found 64, DELETE IT.

What if $h(103) \to 4, 8, 2, 7, 1, 10, 11, 3, 5, 9, 6$?

Search(103):
- $h(103, 1) = 4$, occupied by 78
- $h(103, 2) = 8$, empty: declare 103 not in $T$.

Could use special "deleted" markers, but search time increases.

(Consider deleting all but one element, and then searching for it)
Typical probing sequences
Typical probing sequences

Linear probing : \[ h(k, i) = (h(k, 0) + i) \mod m \]
Typical probing sequences

Linear probing: \( h(k,i) = (h(k,0) + i) \mod m \sim h(k) \) and wrap around.
Typical probing sequences

Linear probing: \( h(k,i) = (h(k,0) + i) \mod m \sim h(k) \) and wrap around.

...tends to generate clusters.

- Probability of extending a cluster
  \[ = \frac{|\text{cluster}|}{m} \]
  slows down search
Typical probing sequences

Linear probing : \( h(k,i) = (h(k,0) + i) \mod m \sim h(k) \) and wrap around.

...tends to generate clusters.
Typical probing sequences

Linear probing: \( h(k, i) = (h(k, 0) + i) \mod m \) ~ \( h(k) \) and wrap around.

... tends to generate clusters.

Quadratic probing: \( h(k, i) = (h(k, 0) + c \cdot i + d \cdot i^2) \mod m \)
Typical probing sequences

Linear probing: $h(k, i) = (h(k, 0) + i) \mod m \sim h(k)$ and wrap around.

...tends to generate clusters.

Quadratic probing: $h(k, i) = (h(k, 0) + c \cdot i + d \cdot i^2) \mod m$

linear make it look more random
Typical probing sequences

Linear probing: \( h(k, i) = (h(k, 0) + i) \mod m \) \( \sim h(k) \) and wrap around.

...tends to generate clusters.

Quadratic probing: \( h(k, i) = (h(k, 0) + c \cdot i + d \cdot i^2) \mod m \)

Less clustering, need to make sure sequence hits all slots
Typical probing sequences

Linear probing: \( h(k, i) = (h(k, 0) + i) \mod m \) \( \sim h(k) \) and wrap around.

...tends to generate clusters.

Quadratic probing: \( h(k, i) = (h(k, 0) + c \cdot i + d \cdot i^2) \mod m \)

linear \( \overbrace{\text{make it look more random}} \)

Less clustering, need to make sure sequence hits all slots

\[ \rightarrow \] Both generate \( m \) probe sequences in total
Typical probing sequences

- **Linear probing**:
  \[ h(k,i) = (h(k,0) + i) \mod m \]
  \( \sim h(k) \) and wrap around. 
  ... tends to generate clusters.

- **Quadratic probing**:
  \[ h(k,i) = (h(k,0) + c \cdot i + d \cdot i^2) \mod m \]
  Linear make it look more random
  Less clustering, need to make sure sequence hits all slots

\[ \rightarrow \text{Both generate } m \text{ probe sequences in total} \]

- **Double hashing**:
  \[ h(k,i) = (h_1(k) + i \cdot h_2(k)) \mod m \]
Typical probing sequences

Linear probing: $h(k,i) = (h(k,0) + i) \mod m \sim h(k)$ and wrap around.

... tends to generate clusters.

Quadratic probing: $h(k,i) = (h(k,0) + c \cdot i + d \cdot i^2) \mod m$

Less clustering, need to make sure sequence hits all slots

Both generate $m$ probe sequences in total

Double hashing: $h(k,i) = (h_1(k) + i \cdot h_2(k)) \mod m$

each $k$ has "random" offset
Typical probing sequences

Linear probing: \( h(k,i) = (h(k,0) + i) \mod m \) \( \sim h(k) \) and wrap around.

... tends to generate clusters.

Quadratic probing: \( h(k,i) = (h(k,0) + c \cdot i + d \cdot i^2) \mod m \)

Less clustering, need to make sure sequence hits all slots

Both generate \( m \) probe sequences in total

Double hashing: \( h(k,i) = (h_1(k) + i \cdot h_2(k)) \mod m \)

Each \( k \) has "random" offset

Generates \( O(m^2) \) probe sequences: better
Typical probing sequences

Linear probing: \( h(k,i) = (h(k,0) + i) \mod m \) \( \sim h(k) \) and wrap around.

... tends to generate clusters.

Quadratic probing: \( h(k,i) = (h(k,0) + c \cdot i + d \cdot i^2) \mod m \)

Less clustering, need to make sure sequence hits all slots

Both generate \( m \) probe sequences in total

Double hashing: \( h(k,i) = (h_1(k) + i \cdot h_2(k)) \mod m \)

Each \( k \) has "random" offset

Generates \( O(m^2) \) probe sequences: better

Heuristic: choose \( m = 2^r \) & \( h_2(k) \) odd.
Analysis of open addressing
Analysis of open addressing

Assuming uniform hashing:

Each key is equally likely to have any of the $m!$ permutations as probe sequence (independent of other keys).

For a random $h$, every slot is equally likely.
Analysis of Open Addressing

Assuming Uniform Hashing: each key is equally likely to have any of the m! permutations as probe sequence (independent of other keys)

Even though all we have so far is $O(m^2)$

Simple Uniform Hashing

For a random $h$, every slot is equally likely
Analysis of open addressing

Assuming uniform hashing: each key is equally likely to have any of the $m!$ permutations as probe sequence (independent of other keys).

Recall $n < m$, so $\alpha < 1$. Claim: $E[\#\text{probes}] \leq \frac{1}{1-\alpha} \left( \frac{m}{m-n} \right)$ (search)
Analysis of open addressing

Assuming uniform hashing: each key is equally likely to have any of the m! permutations as probe sequence (independent of other keys)

Recall n < m, so $\alpha < 1$. Claim: $E[\#\text{probes}] \leq \frac{1}{1-\alpha} \left( \frac{m}{m-n} \right)$ (search)

If true, then for $n \ll m$ $E[\#\text{probes}] = O(1)$
Analysis of Open Addressing

Assuming uniform hashing: each key is equally likely to have any of the $m!$ permutations as probe sequence (independent of other keys)

Recall $n < m$, so $\alpha < 1$. Claim: $E[\# \text{probes}] \leq \frac{1}{1-\alpha} \left( \frac{m}{m-n} \right)$ (search)

If true, then for $n \ll m$ $E[\# \text{probes}] = O(1)$

$\Rightarrow n = \frac{1}{2}m \rightarrow 2$ probes
$\Rightarrow 90\%$ full table $\rightarrow$ 10 probes
Analysis of Open Addressing

Assuming Uniform Hashing: each key is equally likely to have any of the \( m! \) permutations as probe sequence (independent of other keys)

Recall \( n < m \), so \( \alpha < 1 \). Claim: \( E[\#\text{probes}] \leq \frac{1}{1-\alpha} \left( \frac{m}{m-n} \right) \)

(search)

If true, then for \( n \ll m \) \( E[\#\text{probes}] = O(1) \)

\( \xrightarrow{\Rightarrow} n = \frac{1}{2} m \rightarrow 2 \) probes

\( \xrightarrow{\Rightarrow} 90\% \text{ full table} \rightarrow 10 \) probes

Works well if you can afford a table \( \sim \text{data} \times 2 \)

but keep in mind: we’re using a very strong assumption
Claim: $E[\#\text{probes}] \leq \frac{1}{1-\alpha}$

Look at unsuccessful search
Claim: $E[\#\text{probes}] \leq \frac{1}{1-\alpha}$

Look at unsuccessful search

Remember, probe sequence is a permutation.
Never check one slot twice.
Claim: $\mathbb{E}[\#\text{probes}] \leq \frac{1}{1-\alpha}$

$P[\text{1st probe collides}] = \frac{n}{m}$

Look at unsuccessful search

Remember, probe sequence is a permutation.
Never check one slot twice.
Claim: $E[\#\text{probes}] \leq \frac{1}{1-\alpha}$

$P[1\text{st probe collides}] = \frac{n}{m}$ → need 2nd probe

Look at unsuccessful search

Remember, probe sequence is a permutation. Never check one slot twice.
Claim: $\mathbb{E}[\#\text{probes}] \leq \frac{1}{1-\alpha}$

Look at unsuccessful search

$P[1\text{st probe collides}] = \frac{n}{m} \rightarrow \text{need 2nd probe}$

$P[2\text{nd probe collides}] = \frac{n-1}{m-1} \rightarrow \text{need 3rd probe}$

Remember, probe sequence is a permutation.
Never check one slot twice.
Claim: $E[\#\text{probes}] \leq \frac{1}{1-\alpha}$

Look at unsuccessful search

$$P[\text{1st probe collides}] = \frac{n}{m} \quad \rightarrow \text{need 2nd probe}$$

$$P[\text{2nd probe collides}] = \frac{n-1}{m-1} \quad \rightarrow \text{need 3rd probe}$$

\[ \vdots \]

$$\frac{n-i}{m-i}$$

Remember, probe sequence is a permutation.
Never check one slot twice.
Claim: $E[\#\text{probes}] \leq \frac{1}{1-\alpha}$

Look at unsuccessful search

$P[\text{1st probe collides}] = \frac{n}{m} \rightarrow \text{need 2nd probe}$

$P[\text{2nd probe collides}] = \frac{n-1}{m-1} \rightarrow \text{need 3rd probe}$

\[ \vdots \]

$\frac{n-i}{m-i} < \frac{n}{m} = \alpha$

Remember, probe sequence is a permutation.
Never check one slot twice.
Claim: \( E[\#\text{probes}] \leq \frac{1}{1-\alpha} \)

Look at unsuccessful search

\[
P[1\text{st probe collides}] = \frac{n}{m} \quad \rightarrow \text{need 2nd probe}
\]

\[
P[2\text{nd probe collides}] = \frac{n-1}{m-1} \quad \rightarrow \text{need 3rd probe}
\]

\[
\vdots
\]

\[
\frac{n-i}{m-i} < \frac{n}{m} = \alpha
\]

\[
E[\#\text{probes}] = 1 + \frac{n}{m} \left( \text{need at least a 2nd probe} \right)
\]
Claim: $E[\#\text{probes}] \leq \frac{1}{1-\alpha}$

Look at unsuccessful search

$P[1\text{st probe collides}] = \frac{n}{m} \rightarrow \text{need 2nd probe}$

$\star P[2\text{nd probe collides}] = \frac{n-1}{m-1} \rightarrow \text{need 3rd probe}$

\[ \vdots \]
\[ \frac{n-i}{m-i} < \frac{n}{m} = \alpha \]

$E[\#\text{probes}] = 1 + \frac{n}{m} \left( 1 + \frac{n-1}{m-1} \left( \begin{array}{c}
\text{need a 3rd probe}
\end{array} \right) \right)$

Remember, probe sequence is a permutation.

Never check one slot twice.
Claim: \[ E[\#\text{probes}] \leq \frac{1}{1-\alpha} \]

Look at unsuccessful search

\[ P[\text{1st probe collides}] = \frac{n}{m} \rightarrow \text{need 2nd probe} \]

\[ P[\text{2nd probe collides}] = \frac{n-1}{m-1} \rightarrow \text{need 3rd probe} \]

\[ \vdots \]

\[ \frac{n-i}{m-i} \leq \frac{n}{m} = \alpha \]

\[ E[\#\text{probes}] = 1 + \frac{n}{m} \left( 1 + \frac{n-1}{m-1} \right) \left( 1 + \frac{n-2}{m-2} \right) (\text{\textit{\rule{0pt}{1.25ex}}}) (\text{\textit{\rule{0pt}{1.25ex}}}) \]

Remember, probe sequence is a permutation.
Never check one slot twice.
Claim: $E[\# \text{probes}] \leq \frac{1}{1-\alpha}$

Look at unsuccessful search

$P[\text{1st probe collides}] = \frac{n}{m} \rightarrow \text{need 2nd probe}$

$P[\text{2nd probe collides}] = \frac{n-1}{m-1} \rightarrow \text{need 3rd probe}$

$\vdots$

$\frac{n-i}{m-i} < \frac{n}{m} = \alpha$

$E[\# \text{probes}] = 1 + \frac{n}{m} \left( 1 + \frac{n-1}{m-1} \left( 1 + \frac{n-2}{m-2} \left( \cdots \left( 1 + \frac{0}{m-n} \right) \right) \right) \right)$
Claim: \( E[\#\text{probes}] \leq \frac{1}{1-\alpha} \)

Look at unsuccessful search

\[
P[\text{1st probe collides}] = \frac{n}{m} \quad \rightarrow \text{need 2nd probe}
\]

\[
P[\text{2nd probe collides}] = \frac{n-1}{m-1} \quad \rightarrow \text{need 3rd probe}
\]

\[ \vdots \]

\[
\frac{n-i}{m-i} < \frac{n}{m} = \alpha
\]

\[
E[\#\text{probes}] = 1 + \frac{n}{m} \left( 1 + \frac{n-1}{m-1} \left( 1 + \frac{n-2}{m-2} \left( \cdots \left( 1 + \frac{0}{m-n} \right) \right) \right) \right)
\]

\[
\leq 1 + \alpha \left( 1 + \alpha \left( 1 + \alpha \left( \cdots \left( 1 + \alpha \right) \right) \right) \right) \quad \cdots \text{n terms}
\]

Remember, probe sequence is a permutation.

Never check one slot twice.
Claim:  \( E[\#\text{probes}] \leq \frac{1}{1-\alpha} \)  

Look at unsuccessful search

\[
P[\text{1st probe collides}] = \frac{n}{m} \quad \rightarrow \text{need 2nd probe}
\]

\[
P[\text{2nd probe collides}] = \frac{n-1}{m-1} \quad \rightarrow \text{need 3rd probe}
\]

\[
\vdots
\]

\[
\frac{n-i}{m-i} < \frac{n}{m} = \alpha
\]

\[
E[\#\text{probes}] = 1 + \frac{n}{m}\left(1 + \frac{n-1}{m-1}\left(1 + \frac{n-2}{m-2}\left(\cdots\left(1 + \frac{0}{m-n}\right)\right)\right)\right)
\]

\[
\leq 1 + \alpha\left(1 + \alpha\left(1 + \alpha\left(\cdots\left(1 + \alpha\right)\right)\right)\right) \quad \cdots \text{n terms}
\]

\[
\leq 1 + \alpha + \alpha^2 + \alpha^3 + \cdots \quad \cdots \infty \text{ terms}
\]

Remember, probe sequence is a permutation.

Never check one slot twice.
Claim: $E[\#\text{probes}] \leq \frac{1}{1-\alpha}$

Look at unsuccessful search

$P[1\text{st probe collides}] = \frac{n}{m}$ $\longrightarrow$ need 2nd probe

$P[2\text{nd probe collides}] = \frac{n-1}{m-1}$ $\longrightarrow$ need 3rd probe

$\vdots$

$\frac{n-i}{m-i} < \frac{n}{m} = \alpha$

$E[\#\text{probes}] = 1 + \frac{n}{m} \left( 1 + \frac{n-1}{m-1} \left( 1 + \frac{n-2}{m-2} \left( \cdots \left( 1 + \frac{0}{m-n} \right) \right) \right) \right) \leq 1 + \alpha \left( 1 + \alpha \left( 1 + \alpha \left( \cdots \left( 1 + \alpha \right) \right) \right) \right) \cdots$ $n$ terms

$\leq 1 + \alpha + \alpha^2 + \alpha^3 + \cdots \cdots \infty$ terms

$= \sum_{i=0}^{\infty} \alpha^i$
Claim: \( E[\# \text{probes}] \leq \frac{1}{1-\alpha} \)

Look at unsuccessful search

\[
P[1\text{st probe collides}] = \frac{n}{m} \quad \rightarrow \text{need 2nd probe}
\]

\[
P[2\text{nd probe collides}] = \frac{n-1}{m-1} \quad \rightarrow \text{need 3rd probe}
\]

\[
\vdots
\]

\[
\frac{n-i}{m-i} \leq \frac{n}{m} = \alpha
\]

\[
E[\# \text{probes}] = 1 + \frac{n}{m} \left( 1 + \frac{n-1}{m-1} \left( 1 + \frac{n-2}{m-2} \left( \cdots \left( 1 + \frac{0}{m-n} \right) \right) \right) \right)
\]

\[
\leq 1 + \alpha \left( 1 + \alpha \left( 1 + \alpha \left( \cdots \left( 1 + \alpha \right) \right) \right) \right) \quad \vdots \text{n terms}
\]

\[
\leq 1 + \alpha + \alpha^2 + \alpha^3 + \cdots \quad \vdots \text{\inf \ terms}
\]

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

see CLRS for alternate analysis incl. successful search

Remember, probe sequence is a permutation.

Never check one slot twice.
Suggested reading:

perfect hashing

Family of hash functions, pick one randomly. Beats adversaries.
Suggested reading:

**perfect hashing**
Family of hash functions, pick one randomly. Beats adversaries.

**universal hashing**
Fixed input. Create \( h(\cdot) \) based on this.
Suggested reading:

**perfect hashing**

Family of hash functions, pick one randomly. Beats adversaries.

**universal hashing**

Fixed input. Create $h(\cdot)$ based on this.

**cuckoo hashing**

*Cuckoos Use Mafia Tactics, And They Work*

*April 18, 2014 | by Stephen Luntz*