Collisions and collision resolution
Random functions have many collisions

Suppose we map $n$ keys randomly to a hash table with $N$ cells. Then the expected number of pairs of keys that collide is

$$\sum_{\text{keys } i,j} \Pr[i \text{ collides with } j] = \binom{n}{2} \frac{1}{N} \approx \frac{n^2}{2N}$$

The sum comes from linearity of expectation. There are $\binom{n}{2}$ key pairs, each colliding with probability $1/N$.

E.g. birthday paradox: a class of 80 students has in expectation $\binom{80}{2} \frac{1}{365} \approx 8.66$ pairs with the same birthday.

Conclusion: Avoiding all collisions with random functions needs hash table size $\Omega(n^2)$, way too big to be efficient.
Resolving collisions: Open addressing

Instead of having a single array cell that each key can go to, use a “probe sequence” of multiple cells.

Look for the key at each position of the probe sequence until either:

- You find a cell containing the key
- You find an empty cell (and deduce that key is not present)

Many variations; we’ll see two later this week.
Resolving collisions: Hash chaining (bucketing)

Instead of storing a single key–value pair in each array cell, store an association list (collection of key–value pairs)

To set or get value for key $k$, search the list in $A[h(k)]$ only

Time will be fast enough if these lists are small

(But in practice the overhead of having a multi-level data structure and keeping a list per array cell means the constant factors in time and space bounds are larger than other methods.)

First hashing method ever published: Peter Luhn, 1953
Expected analysis of hash chaining

Assumption: $N$ cells with load factor $\alpha = n/N = O(1)$
(Use dynamic tables and increase the table size when $N << n$)

Then, for any key $k$, the time to set or get the value for $k$ is
$O(\ell_k)$, the number of key–value pairs in $A[h(k)]$.

By linearity of expectation,

$$E[\ell_k] = 1 + \sum_{\text{key } j \neq k} \Pr[j \text{ collides with } k] = \frac{n - 1}{N} < \alpha = O(1).$$

So with tables of this size, expected time/operation is $O(1)$.  

Expected size of the biggest chain

A random or arbitrary key has expected time per operation $O(1)$

But what if attacker chooses key whose cell has max ≠ keys?

(Again, assuming constant load factor)

Chernoff bound: The probability that any given cell has $\geq x\alpha$ keys is at most

$$\left( \frac{e^x}{x^x} \right)^{\alpha}$$

(and is lower-bounded by a similar expression)

For $x = C \frac{\log n}{\log \log n}$, simplifies to $1/N^c$ for some $c$ (depending on $C$)

$c > 1 \Rightarrow$ high probability of no cells bigger than $x\alpha$

$c < 1 \Rightarrow$ expect many cells bigger than $x\alpha$

Can prove: Expected size of largest cell is $\Theta\left( \frac{\log n}{\log \log n} \right)$
Linear probing
What is linear probing?

The simplest possible probe sequence:
Look for key $k$ in cells $h(k)$, $h(k) + 1$, $h(k) + 2$, ... (mod $N$)

Invented by Gene Amdahl, Elaine M. McGraw, and Arthur Samuel, 1954, and first analyzed by Donald Knuth, 1963

- Fast in practice: simple lookups and insertions, works well with cached memory
- Commonly used
- Requires a high-quality hash function (next time)
- Load factor $\alpha$ must be $< 1$ (else no room for all keys)
Lookups and insertions

With array $A$ of length $N$, holding $n$ keys, and hash function $h$:

To look up value for key $k$:

$$i = h(k)$$

while $A[i]$ has wrong key:

$$i = (i + 1) \mod N$$

if $A[i]$ is empty:

raise exception

return value from $A[i]$

To set value for $k$ to $x$:

$$i = h(k)$$

while $A[i]$ has wrong key:

$$i = (i + 1) \mod N$$

if $A[i]$ is empty:

$$n += 1$$

if $n/N$ too large:

expand $A$ and rebuild

$$A[i] = k,x$$
### Example

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
<th>hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Y</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Z</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

\[
i = 5 \quad 6 \quad 7 \quad 8
\]

A:

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X,2</td>
<td>Y,3</td>
<td></td>
</tr>
<tr>
<td>X,2</td>
<td>Y,3</td>
<td>Z,5</td>
</tr>
</tbody>
</table>
```

Default position for Z was filled ⇒ placed at the next available cell

Have to be careful with deletion: blanking the cell for X would make the lookup algorithm think Z is also missing!
Deletion

First, find key and blank cell:

\[ i = h(k) \]

while A[i] has wrong key:

\[ i = (i + 1) \mod N \]

if A[i] is empty:

raise exception

A[i] = empty

Then, pull other keys forward:

\[ j = (i + 1) \mod N \]

while A[j] nonempty:

k = key in A[j]

if \( h(k) \leq i < j \mod N \):


A[j] = empty

i = j

j = (j + 1) \mod N

Result = As if remaining keys inserted without the deleted key
Blocks of nonempty cells

The analysis is controlled by the lengths of contiguous blocks of nonempty cells (with empty cells at both ends)

Any sequence of $B$ cells is a block only when:

- Nothing hashes to the empty cells at both ends
- Exactly $B$ items hash to somewhere inside
- For each $i$, at least $i$ items hash to the first $i$ cells

Our analysis will only use the middle condition:
Exactly $B$ items hash into the block
Probability of forming a block

Suppose we have a sequence of exactly $B$ cells

Expected number of keys hashing into it: $\alpha B$
To be a block, actual number $= B = \frac{1}{\alpha}$ expected

Chernoff bound:
Probability this happens is at most

$$\left( \frac{e^{1/\alpha - 1}}{(1/\alpha)^{1/\alpha}} \right)^{\alpha B} = c^B$$

for some constant $c < 1$ depending on $\alpha$

Long sequences of cells are exponentially unlikely to form blocks!
Linear probing analysis

Expected time per operation on key \( k \)

\[
= O(\text{expected length of block containing } h(k))
\]

\[
= \sum \Pr[\text{it is a block}] \times \text{length}
\]

\[
= \sum (\text{number of blocks of length } \ell \text{ containing } k) \times c^\ell \times \ell
\]

\[
= \sum \ell \times c^\ell \times \ell
\]

\[
= O(1)
\]

(because the exponentially-small \( c^\ell \) overwhelms the factors of \( \ell \))
Expected length of the longest block

A random or arbitrary key has expected time per operation $O(1)$
But what if attacker chooses key whose block has max # length?

Same analysis $\Rightarrow$ blocks of length $C \log n$ have probability $1/N^\Theta(1)$
(inverse-exponential in length) of existing

Large $C \Rightarrow$ high probability of no blocks bigger than $C \log n$
Small $C \Rightarrow$ expect many blocks bigger than $C \log n$

Can prove: Expected size of largest block is $\Theta(\log n)$