Admin:

Quiz 1 Wed 10/16 at 7:30 pm
in 32-124, 32-144, and 34-101 (see website)
May bring one 8½” x 11” ‘cheat sheet’ (o.w. closed book)
Coverage through L10 (Hashing III)

Readings: Chapter 11 (11.1-11.4)

Cuckoo Hashing for Undergraduates

Outline:

☐ Open Addressing for Collision Resolution

☐ Double Hashing

☐ Analysis (assuming ‘uniform hashing’)

☐ Cuckoo Hashing (good worst-case lookups!)
Today's topic: "open addressing" for collision resolution

With "chaining", each table position T[i] points to a head of linked list of all keys hashing to position i:

No keys stored in T itself; "extra" storage needed for keys (and links).

With "open addressing": no linked lists; all keys stored in table T. Each T[i] is either NIL (empty).

So:

\[ n \leq m \text{ necessarily} \]
\[ \alpha \leq 1 \quad (\alpha = n/m). \]

Table-doubling is necessary for open addressing (but not chaining) as table grows.

Typically work to keep \( \alpha \leq \frac{1}{2} \).
With open addressing we replace \( h(k) \) with
\[
h(k, i) \quad \text{for} \quad i = 0, 1, \ldots, m-1
\]
giving probe sequence \( h(k,0), h(k,1), \ldots, h(k,m-1) \)
for key \( k \). Probe sequence is a permutation of \( 0, 1, \ldots, m-1 \)
specifying sequence of slot positions to be probed in a search.

**Search \((T, k)\):**

\[
\begin{align*}
\hat{c} &= 0 \\
\text{repeat} & \quad j = h(k, i) \\
& \quad \text{if } T[j] = k \quad \text{return } j \\
& \quad \hat{c} = \hat{c} + 1 \\
& \quad \text{until } T[j] = \text{NIL} \text{ or } \hat{c} = m \\
& \quad \text{return } \text{NIL}
\end{align*}
\]

Uniform hashing often assumed for analysis;
for a given \( k \), each permutation is "equally likely".

[\text{E.g., } h(k, \hat{c}) \text{ chosen independently as random perm. of } 0, 1, \ldots, m-1]
In practice, double hashing gives good approx to uniform

- $h_1(k)$ maps random to $\{0, 1, \ldots, m-1\}$
- $h_2(k)$ maps random to $\{1, 2, \ldots, m-1\}$
- $m$ is prime (so $\gcd(m, h_2(k)) = 1$.)

Then

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

i.e. probe $h_1(k)$

$$h_1(k) + h_2(k)$$

$$h_1(k) + 2 \cdot h_2(k)$$

Perm of $0, 1, \ldots, m-1$

\[\text{if } \gcd(m, h_2(k)) = 1.\]

(\[\text{Note that linear probing is similar, but}\]

\[\text{has } h_2(k) = 1 \text{ always, so we start at position}\]
\[h_1(k), \text{ then examine following positions as needed:}\]

$$h_1(k), h_1(k)+1, h_1(k)+2, \ldots$$

Linear probing is often a poor choice, as keys tend

to end up in clusters...
Insertion:

- Need to insert it where it can be found later!

- Two approaches:
  1. Use first empty slot in probe sequence.
  2. Move some previously inserted keys around to "make room" for new key. "Cuckoo hashing" is an example of this strategy.

Example of (1):

\[
\begin{array}{c|c|c|c|c|c}
\text{m} & 0 & 1 & 2 & 3 & 4 \\
\text{n} & \_ & \_ & k_2 & \_ & k_a \\
\end{array}
\]

\[
\begin{align*}
\text{probe sequences} & : \begin{cases}
  k_1 \rightarrow 3, 2, 4, 1, 0 \\
  k_2 \rightarrow 3, 4, 2, 0, 1 \\
  k_3 \rightarrow 4, 3, 1, 2, 0 \\
\end{cases}
\end{align*}
\]

(Note: double hashing...)

Deletion: Note problem if we delete \( k_1 \), by changing \( T[S] \) to \( N[S] \)

then \( k_2 \) & \( k_3 \) can no longer be retrieved!

*: typically don't use deletion with open addressing.

(Could have special "deleted" entry, though...)

Analysis:

Unsuccessful search (or insertion):

(average) \# probes \( \approx 1 + \alpha + \alpha^2 + \ldots \)

(since \( \alpha \) chance 1st probe is full, \( \alpha^2 \) that second & first both full, etc...)

\[
\frac{1}{1 - \alpha} = \frac{m}{1 - \frac{n}{m}} = \frac{m}{m-n}
\]

(e.g., 2 probes for \( \alpha = 1/2 \))

Successful search:

Average time to lookup key in table

= average time to insert it originally (assume it isn’t moved!)

= \[
\frac{1}{n} \sum_{i=0}^{\infty} \frac{m}{m-i} \approx \frac{1}{\alpha} \ln \left( \frac{1}{1-\alpha} \right)
\]

\( \approx 1 + \frac{\alpha}{2} + \frac{\alpha^2}{3} + \frac{\alpha^3}{4} + \ldots \)

For \( \alpha = 1/2 \), successful search time \( \leq 1.4 \) probes (on average)
Cuckoo Hashing

- Uses two hash func. \( h_1(k) \) & \( h_2(k) \)
  \[ h_1(k) \neq h_2(k) \]
- Key \( k \) stored in either \( T[h_1(k)] \) or \( T[h_2(k)] \).
- **Lookup**: Just look in at most 2 places!
- **Insertion**:
  - If \( T[h_1(k)] \) empty: store key there
  - If \( T[h_2(k)] \) empty: store key there
  - If both full:
    - Store \( k \) in \( T[h_1(k)] \)
    - Move key that was there to its other location (bumping out key that might be there, etc.)
- If \( \alpha < \frac{1}{2} \), insertion succeed with high probability.
- If insertion "loops": **rehash** entire table (can double table size too if desired.)
- Insertion takes constant time on average.