Hashing (Chapter 12)

- Heaps support:
  - Insert
  - Delete
  - Max/min

- How about Search?
  (How would you implement "find" in a heap?)

- Possible solutions:
  - Ordered array: slow "insert", efficient "delete".
  - Ordered list: both find and insert are slow.

Direct address table

- Maintain in table:
  - $T[i][j] = x$ if $x$ is in slot $i$.

- Disadvantage: too much memory!

- Idea: maintain small table:

Collisions

- Table $|\ll| \text{Universe of keys} \mid = |\text{collisions}|

  - How to resolve collisions:
    - Chaining:
    - Open addressing: if $A[h(x)]$ full, try next slot.

Analysis of Chaining

- Assume each key equally likely hashed to any slot.
- $n$ keys, $m$ slots; $\alpha = \frac{n}{m} = \text{load factor}$
- Expected length of a chain:
  $\sum_{i=0}^{\alpha} \frac{1}{i+1}$

  $= \frac{\alpha}{\alpha+1} \sum_{i=0}^{\alpha} \frac{1}{i}$

  $= \frac{\alpha}{\alpha+1} \ln(\alpha+1) + O(1)$

  Access time $= O(\ln(\alpha))$

- Unsuccessful search:
  - Expected length of a randomly chosen list + 1: $O(1 + \alpha)$

Successful Search

- Expected time to find $i$-th element = time to insert $i$-th element

- Assume that the key being searched for is equally likely to be any one of the keys stored.

- Conditioned on "key was the $i$-th element inserted",
  - Expected time $= \frac{1}{i+1}$

  Overall:
  $\sum_{i=0}^{n} \frac{1}{i+1}$

  $= \frac{1}{2} \ln(n+1) + \frac{1}{2} + O(1)$

  $\Rightarrow \text{Intuition: need to search 1/2 of a list on the average.}$

Open Addressing

- If $A[h(x)]$ full, try "next" slot.

- Linear probing:
  - Pick some integer $b$, relatively prime to size of table $m$.
  - For $i = 0, 1, 2, \ldots$, try to place $x$ in position:
    $h(x) + ib \mod m$

  - Bad idea: results in large clusters.
  - Increased search time and insert time as $\alpha \rightarrow 1$.

- Double hashing:
  - Works well in practice.
  - Pick two hash functions $h_1, h_2$
  - For $i = 0, 1, 2, \ldots$, try to place $x$ in position:
    $h_1(x) + ih_2(x) \mod m$
Analysis of Open Addressing

- Simplifying assumption: h(key, probe #), random and uniform.
- Probability that at least i probes lead to already occupied slots?

\[ q = \frac{1}{2^i} \]

- Expected # probes in unsuccessful search:

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

Why?

More open addressing

- What about successful search? Depends on the element: element inserted earlier will be easier to find!
- Assume uniform distribution on the element we search for.
  If element was inserted at (i-1)-th step, expected number of probes was

\[ \frac{1}{2^i} \]

- Condition on i, take expectation:

\[ \sum_{i=1}^{\infty} \frac{1}{2^i} = 1 \sum_{i=1}^{\infty} \frac{1}{2^i} = \sum_{i=1}^{\infty} \frac{1}{2^i} \]

But: \( \sum_{i=1}^{\infty} \frac{1}{2^i} = \frac{1}{2} \)

Thus expected probes:

\[ \sum_{i=1}^{\infty} \frac{1}{2^i} = \frac{1}{2} \sum_{i=1}^{\infty} \frac{1}{2^i} \]