LECTURE 7 - HASHING - I

Today: Hashing (1st of 2 lectures)

- Dictionary Problem
- Hashing: Definition + Goals
- Collision Resolution by
  - Chaining
  - Open Addressing

---

Dictionary Problem: Motivation

Maintain large database of "words", so as to be able to

\[
\begin{align*}
\text{INSERT} \ (k, S) & : \text{Insert key } k \text{ into } S. \\
\text{DELETE} \ (k, S) & : \text{Delete element with key } k \text{ from } S. \\
\text{SEARCH} \ (k, S) & : \text{Determine whether } k \text{ in } S.
\end{align*}
\]

---

Common Problem in Computing:

- Maintain Phonebooks
- Database Searching (in Google/Yahoo/Msn)
- 6.046 TA Update
- Compiler / Operating System ---
Naive Solutions

- Unsorted Array - Search/Delete slow
- Sorted Array - Insert/Delete slow
- Lists - Similar
- Trees - Complex to implement (will be later in course); but still takes \( \Omega(\log n) \) time to handle set \( S \) of size \( n \).

Today's Approach: Hashing ....

... leads to expected \( O(1) \) time solution to \( \text{INSERT}, \text{DELETE}, \text{SEARCH} \).

Definition:

Assume key \( k \) is a (large) integer.

\[
h: \mathbb{Z}^+ \to \mathbb{Z}^+ \to \mathbb{Z}^+ \to \mathbb{Z}^+
\]

Hash function maps it down to small set \( m \).
Goals of good hash function:

1. $h(k)$ must be easy to compute
2. "$h(k_1) = h(k_2)" must occur rarely for $k_1, k_2 \in S$.

Collision Resolution:

Two approaches:
- Chaining
- Open Addressing
Dictionary based on Hashing + Chaining

Data structure = Array of size $m$ + $m$ linked lists.

```
SEARCH ($k$, $s$)
\[ i \leftarrow h(k) \]
Search for $k$ in list $(i)$;

INSERT ($k$, $s$)
\[ i \leftarrow h(k) \]
Insert $k$ into list $(i)$;

DELETE ($k$, $s$)
\[ i \leftarrow h(k) \]
Delete $k$ from list $(i)$;
```
Worst-Case Analysis

* Unfortunately, h pruned before S.

* Given h, let i, S be the elements of \( \{0, \ldots, m-1\} \) that minimize the set \( S = \{ k \mid h(k) = i \} \).

* Perform as badly as linked list.

Average Case Analysis

* Assume inserted elements \( S = \{ k_1, k_2, \ldots, k_n \} \) are chosen uniformly, independently from \( \{0, \ldots, U\} \).

SEARCH two sequences

"Unsuccessful": k uniform from 0;

"Successful": k uniform from S;
Take a simple hash function

\[ h(k) = k \pmod{m}. \]

Define \textbf{LOAD FACTOR} \( \alpha = \frac{n}{m} \)

- size of set \( S \)
- \( n \) = # of keys
- \( m \) = size of table
- \( n \) = # slots

Expected length of list \( i \) if \( i \) is chosen uniformly from \( \{0 \ldots m-1\} \)

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

Expected time for unsuccessful \textbf{SEARCH}

\[ = \Theta(1 + \alpha) \]

---

(Successful search is different. Why?)

Prob. of picking \( i \) \( \propto \) length of list \( i \)

So expected time to execute search query
\[= \sum_{i=0}^{m-1} \frac{l_i}{n} \cdot (1 + \Theta(l_i))\]

\[= \frac{m \cdot E[l_i]}{n} + \Theta \left( \frac{m \cdot E[l_i^2]}{n} \right)\]

\[
\frac{m}{n} = \frac{1}{\alpha}; \quad E[l_i] = \alpha; \quad E[l_i^2] = \alpha^2 \quad \text{[did n’t prove this]}
\]

Yields

Expected time to execute successful search

\[= \frac{1}{\alpha} \cdot \alpha + \Theta \left( \frac{1}{\alpha} \cdot \alpha^2 \right)\]

\[= \Theta(1 + \alpha)\]
Collision resolution - 2: Open Addressing.

Use a hash function that takes two arguments $h(k, i)$.

When inserting/searching/deleting:

for $i = 1$ to $\infty$ do

$\hat{j} \leftarrow h(k, i) ;$

if $T[\hat{j}] \cdot \text{key} = k$ then take appropriate action;

if $T[\hat{j}] = \phi$ then stop;

else continue.

Why do this?

"Clean storage," I know exactly how much space we need.

Analysis -- trickier, even if we assume $h_1, \ldots, h_m$ independent.

Why?
(Even if we assume \( k_i \sim k \sim i \sim \ldots \),
\((k_1, 1), (k_2, 2), (k_3, 3) \) are not !)

So we use more complex hashing

\[ h(k, i) = h_1(k) + i \cdot h_2(k) \]

where \( h_1(k) \) & \( h_2(k) \) are chosen to be independent of each other.

Analysis (under more flaky assumptions)

\* Assumption: \( h(k, i) \) is uniform in \( \{0, \ldots, m-1\} \).

So \( m \cdot \alpha \).

If load factor = \( \lambda \)

then Expected time of unsuccessful search = ?

\[ \text{Prob} \left[ h(k, i) = \emptyset \right] = 1 - \lambda \]
Expected trials before hitting $\phi$ key

$$= \frac{1}{1 - \alpha}$$

Conclusions:

- Hashing is promising wrt the goal of maintaining Dictionary ($\text{INSERT}$, $\text{DELETE}$, $\text{SEARCH}$);

- Promise $\Theta(1)$ time complexity with $\Theta(n)$ space complexity.

- But needs careful construction; formal analysis;
  less dependence on unproven assumptions.

- Wait for Revenge Of The Hash Table