LECTURE 16: HASHING

[Typo error on page 6+7 corrected]

Today:

- Dynamic Dictionary
  - Insert, Delete, Search

- Hashing
  - Everything in $O(1)$ expected time.

- Random Hashing

- Universal Hashing

Main Problem: Can we maintain set $S$ with INSERT, DELETE, SEARCH efficiently?
**Known Answers**

1. If \( S \subseteq \{1, 2, \ldots, N\} \) then can do it with \( O(N) \) space.
2. \( O(N) \) initialization (graduate problem: do this in \( O(1) \) time)
3. \( O(1) \) INSERT, DELETE, SEARCH.
4. If \( |S| = n \ll N \) then can also do
   - \( O(n) \) space / initialization
   - \( O(\log n) \) INSERT / DELETE / SEARCH.

Today: HASHING: best of both worlds. Randomized.
Aside: Dictionary: very common problem in computing
- Maintaining Phonebooks
- Search (a la Google/Yahoo/mns)
- 6.046 TA update
- Compilers/OS

Hashing: Commonly used ...

Today: Theory behind Hashing.
**Hashing**

**Goal:** Efficiently handle $n$ requests

1. **Initialize**
2. **Insert** $(k_i)$
3. **Insert** $(k_2)$
4. **Delete** $(k_i)$
5. **Search** $(k_{in})$
6. **Insert**
7. **Delete**
8. **Search**
9. **Search** $(k_n)$

**Idea:** 0. Pick hash function

$h : \{1, \ldots, N\} \rightarrow \{1, \ldots, m\}$

Somehow.
2) Store keys into array of size $m$.

$$\text{key } k \rightarrow \text{array loc. } h(k)$$

Hope: if hash function $h$ chosen independently of $k_1, \ldots, k_n$

then no "collisions"

Collision: $k_i \neq k_j \text{ s.t. } h(k_i) = h(k_j)$. 

**Bad News**

1. If $n > m$ & key $R_1, \ldots, R_n$ distinct then $\exists i, j$ s.t.

$$h(R_i) = h(R_j)$$

**[Pigeonhole Principle]**

**Fix:** Pick $n \leq m$; say $n \neq 2m$.

Pick $m > n$; say $m = 2n$.  

**Corrected Fix**
Bad News (contd.)

\[ m \ll n^2 \]

2. if \( n \gg m^2 \) \( \in \{k_1, \ldots, k_n\} \) "random"
   
   (and also \( h(k_1), \ldots, h(k_n) \))

Then, with high probability,

\[ \exists i \neq j, k_i \neq k_j \text{ s.t. } h(k_i) = h(k_j) \]

[BIRTHDAY PARADOX]

Fix: "CHAINING"

- At every array location, maintain a list of keys that hash to that location.
- Search entire list to search.
**Implementation Details**

**Initialize** \((N, n)\);

0. Set \(m = 2n\);

1. Pick \(h : \{1, \ldots, N\} \rightarrow \{1, \ldots, m\}\)

2. For \(i = 1 \) to \(m\) do
   
   initialize \(\text{LIST}[i] \leftarrow \phi\)

**List**

```
1 → NIL
2 → NIL
   ↓
   ...
   ↓
m → NIL
```
**INSERT** \((k)\):
- let \(i \leftarrow h(k)\);
- Insert \(k\) into \(\text{LIST}(i)\); \(\in O(1)\) time

**DELETE** \((k)\):
- let \(i \leftarrow h(k)\);
- Delete \(k\) from \(\text{LIST}(i)\); \(\in O(\text{length}\ \text{LIST}(i))\)

**SEARCH** \((k)\):
- let \(i \leftarrow h(k)\);
- Search for \(k\) in \(\text{LIST}(i)\);
Good News:
Algorithm as "simple" as maintaining lists.

Question: How to pick $h$?

Bad Choices:
1. $h(k) = 1$ for every $k$
   "Not well distributed"
2. $h(k_i) = l$ if $k_i$ new $\exists k_i: |k_i| = l$
   $= h(k_i)$ if $k_i = k_j$.
   "Not easy to compute"; "depends on input"
Fixing Problem 1: Random h

- Definition of h:
  
  for every \( k \in \{1, \ldots, n\} \)
  
  \( h(k) \) chosen randomly from \( \{1, \ldots, m\} \).

- Well distributed? YES

- Easy to compute? NO!

- Still... let's analyze this.
- Assume we’re inserted $k_1, \ldots, k_n$ distinct
- Say list sizes = $l_1, \ldots, l_m$
- Time to SEARCH($k$) = ?

**Case 1:** $k \notin \{k_1, \ldots, k_n\}$

Then $h(k) = i$ is random in $[1, \ldots, m]$

**Expected [Search Time]**

$$E \left[ \sum_i l_i \right] + \Theta(1)$$

$$= \frac{n}{m} + \Theta(1)$$
Case 2: \( k \in \{ k_1, \ldots, k_n \} \)

\[ \Pr[h(k) = i] = ? \]

Example

\[
\begin{array}{c|c}
\text{h(n)} & 1 \\
\text{h(2)} & 1 \\
\vdots & 2 \\
\vdots & 4 \\
\text{h(10)} & 4 \\
\end{array}
\]

\( h(1) = ? \)

\[
\begin{array}{c|cc}
\text{1} & \text{wp.} & \frac{3}{10} \\
\text{4} & \text{wp.} & \frac{3}{10} \\
\end{array}
\]

\[
= 2 \quad \text{wp.} \quad \frac{1}{10} \quad \text{and} \quad 8 \quad \text{wp.} \quad \frac{3}{10}
\]
In general

$$\Pr[h(k) = li] = \frac{li}{n}$$

$$E[\text{Search Time}] = \Theta(1) + \sum_i E[li]$$

$$= \Theta(1) + \sum_{i=1}^m \frac{li^2}{n}$$

$$= \Theta(1) + \sum_{i=1}^m \frac{li^2}{n}$$

**Fact:**

$$E[li^2] = \sum_{t=1}^n \sum_{t'=1}^n \Pr[h(t) = h(t')]$$

$$= n + n(n-1) \frac{1}{m} = n + \frac{n-1}{2} = \Theta(n)$$
But: What use is this if Random $h$

is not easy to compute?

**Universal Hashing**

- Has well-distributed property, just like Random

- But has short description; can be computed in $O(1)$ time.
**Example Universal Hash Family**

- Let $m = \text{prime number}$
- Let $N = m^d$, integer $d$
- View key $x = x_1 \ldots x_d$ as $d$-digit number in base $m$.
- To pick hash function $h : \{0 \ldots N-1\} \rightarrow \{0 \ldots m-1\}$, pick $a = a_1 \ldots a_d$ at random and let
  \[ h_{a}(x) = \sum a_i \cdot x_i \pmod{m} \]
- If $d = O(1)$ (our assumption), then $h$ takes $O(1)$ time to compute
**Universality Property**

**Theorem:** If $x \neq y$ then $\Pr_a \left[ h_a(x) = h_a(y) \right] = \frac{1}{m}$

**Proof:**

$h_a(x) = h_a(y)$

$\iff \sum_{i=1}^{d} a_i x_i = \sum_{i=1}^{d} a_i y_i \pmod{m}$

$\iff \sum_{i=1}^{d} a_i (x_i - y_i) = 0 \pmod{m}$

Let $d'$ be largest integer s.t. $x_{d'} - y_{d'} \neq 0$

$\Rightarrow a_{d'} = \sum_{i=1}^{d'} a_i (x_i - y_i) \pmod{m}$

$\Rightarrow x_{d'} - y_{d'}$
But $a_0$ takes on this value w.p. $\frac{1}{m}$.

**Analysing Search Cost For Universal Hashing**

- Say inserted $k_1, \ldots, k_n$ distinct.
- $\text{SEARCH}(k)$: Cost = ?
- Let $X_j = 1$ if $h(k_j) = h(k)$

\[
\exp\text{-Search\text{-Cost} = } E\left[ \sum_{j=1}^{n} X_j \right] \\
\leq E\left[ \sum_{j=1}^{n} X_j \right] \\
= \sum_{j=1}^{n} \Pr\left[ h(k_j) = h(k) \right]
\]
if some $k_j = k$ 
for the rest

\[
\leq 1 + \sum_{j=1}^{n} \Pr [ h(k_j) = h(k) \mid k_j \neq k ]
\]

\[= 1 + \frac{n}{m}\]

\[
\uparrow
\]

As good as random!

**Conclusions:**

- There exist many **universal** hashing schemes; can pick to suit other constraints if we have them.

- Can **insert/delete/search** in expected $O(1)$ time.

- Only in **expectation**; must keep in mind.