Today
- Hashing
- Amortization

Motivation

Arrays are great!

\[ \begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & \cdots \\n0 & 0 & 1 & 1 & 1 & 0 & 1 & \end{array} \]

represents \( \{3, 4, 5, 7, \ldots, n\} \)

Insert in \( O(1) \) time
Delete in \( O(1) \) time
Search in \( O(1) \) time.

But what if we want to represent sets of up to \( n \) elements from a domain different than \( \{1, 2, 3, \ldots, n\} \), possibly a much larger domain!

E.g., a set of ssn.

a set of user names.
Idea

Map the huge domain to \( \{1, \ldots, n\} \).

**Problem** For any \( h: U \rightarrow \{1, \ldots, n\} \) there is an element \( i \in \{1, \ldots, n\} \) that at least \( 101 \frac{1}{n} \) of the elements in \( U \) are mapped to \( i \).

If our set contains only elements mapped to \( i \), we have a problem...

Pick \( h \) at random.

Very unlikely that the set \( S \) all maps to the same element
\[
\text{prob} = \binom{n}{2} \cdot \left( \frac{1}{n} \right)^{\text{size of subset}} = \frac{1}{n^{1.1}}
\]
\((\varepsilon)\)-Universal Hash \(H = \{h: U \to \{1, \ldots, n\}\}\)

For all \(x_1 \neq x_2 \in U\),
\[
P(h(x_1) = h(x_2)) \leq \frac{1}{n}.
\]

\(\varepsilon\)-universal replaces this with \(\varepsilon\).

**Thm** Fix \(S \subseteq U\), \(x \in U\). Pick \(h \in H\) at random.

The expected number of elements from \(S\) mapped to \(h(x)\) is
\[
1 + \frac{|S|}{n}.
\]

**Pf** For every \(x' \in S\), \(x' \neq x\),
\[
P(h(x') = h(x)) \leq \frac{1}{n}.
\]

\[
\Rightarrow \quad \mathbb{E} \sum_{x' \in S} \sum_{x' \neq x} I_{h(x') = h(x)} \leq \frac{|S|}{n}.
\]

If \(x \in S\), the \(\mathbb{E} \sum_{x' \in S} I_{x = h(x')} \leq 1 + \frac{|S|}{n}.
\]

\(\square\)

**Cor** Insert, Delete, Search have expected running time \(O(1 + \frac{|S|}{n})\) when \(h\) is uniform in \(H\), universal.

For \(\varepsilon\)-universal \(H\), it's \(O(1 + \varepsilon |S|)\).

**Cor** The prob the run-time is \(> C\) times the expectation is \(\leq \frac{1}{n}\).
The expected number of elements mapped to is \( O(n) \) as long as \( |S| = O(n) \).

Does this mean that a typical hash table looks like this?

\[
\begin{array}{c}
0 \rightarrow [ ] \\
1 \rightarrow [ ] \\
2 \rightarrow [ ] \\
\vdots \\
|S| - 1 \rightarrow [ ] \\
|S| \rightarrow [ ]
\end{array}
\]

all bins have \( O(1) \) elements mapped to them.

We'll see that the (counter-intuitive) answer is no, even when the mapping is a completely random mapping.
Examples of universal hash families

1. \( H = \text{all functions } h: \{0\} \rightarrow \{1, \ldots, n\} \). \( x_1 \neq x_2 \Rightarrow P(h(x_1) = h(x_2)) = \frac{1}{n} \)

   (can’t expect to be any more balanced)

   How many elements are mapped to \( i \in \{1, \ldots, n\} \)?

   \( L_i \sim B(\lfloor s / n \rfloor) \) (the “load” on \( i \))

   Fact For low probabilities, the binomial distribution \( B(np) \)
   is approximately the Poisson distribution \( P(\frac{\lfloor s / n \rfloor}{n}) \)

   \[ P(L_i = k) = e^{-\frac{\lfloor s / n \rfloor}{n}} \frac{\left(\frac{\lfloor s / n \rfloor}{n}\right)^k}{k!} \]

   Hence, for instance, when \( \lfloor s / n \rfloor = n \):

   \[ P(L_i = 0) = e^{-1} \frac{1}{0!} = \frac{1}{e} \approx 0.3679 \]

   \[ P(L_i = 1) = e^{-1} \frac{1}{1!} = \frac{1}{e} \approx 0.3679 \]

   \[ P(L_i = 2) = e^{-1} \frac{1}{2!} = \frac{1}{2e} \approx 0.1839 \]

   \[ \vdots \]

   Cor When \( \lfloor s / n \rfloor = n \)

   \[ E[\max L_i] = O\left(\frac{\log n}{\log \log n}\right) \]

   Moreover, with high prob.

   For all \( i \) \( L_i \leq O\left(\frac{\log n}{\log \log n}\right) \)
So, a typical hash table when the hash is completely random looks more like this: \( |S| = n \)

- \( 1 \rightarrow 1 \) \{ about 37\% of table empty \}
- \( 2 \rightarrow 1 \) \{ about 37\% of table has one element \}
- \( \vdots \)
- \( n \rightarrow 1 \) \{ maximal pile is of size \( O(\frac{\log n}{\log \log n}) \) \}

\[ \frac{\log n}{\log \log n} \]
(2) In pset 4, \( h_p : \{0, 1\}^m \to \{0, \ldots, K-1\} \)

\[ h_p(x) = x \mod p \] for prime \( p \leq K \)

\[ 4.a : x \neq y \Rightarrow P(h_p(x) = h_p(y)) \leq \frac{M \cdot \text{len}}{K} . \]

(3) In pset 5, \( h_a : \{0, 1\}^m \to \mathbb{Z}_p \quad a \in \mathbb{Z}_p \)

\[ h_a(x) = \langle x, a \rangle = \sum x_i a_i \mod p \]

(4) In pset 6, \( h_A : \mathbb{Z}_p^m \to \mathbb{Z}_p^k \quad A \in \mathbb{Z}_p^{k \times m} \)

\[ h_A(x) = Ax \]

(5) If \( H_1 \) is \( \varepsilon_1 \)-universal, \( \{0, 1\}^m \to \{0, 1\}^k \)

\( H_2 \) is \( \varepsilon_2 \)-universal, \( \{0, 1\}^k \to \{0, 1\}^8 \)

then \( \exists \lambda = \{ h_2 \circ h_1 \mid h_1 \in H_1 \} \) is \( (\varepsilon_1 + \varepsilon_2) \)-universal

\[ h_2 \in H_2 \]

\( x \neq x' \Rightarrow \) except w.p. \( \varepsilon_1 \) \( h_1(x) \neq h_1(x') \Rightarrow \)

except w.p. \( \varepsilon_2 \) \( h_2(h_1(x)) \neq h_2(h_1(x')) \).
What if we want that the number of elements mapped to every $i \in \{1, \ldots, n\}$ will be $\leq 1$?

Can make $n$ much larger than $5$!

Thus $H$ - universal hash family, $U \rightarrow \{1, \ldots, n\}$.

$\sum_{x \neq y \in U} \mathbb{P}[f_h(x) = f_h(y)] = \binom{|U|}{2} \cdot \frac{1}{n}$

the expected number of collisions

$\mathbb{P}[f_h(x) = f_h(y)] \leq \frac{1}{n} \quad \forall x \neq y \in U$

The thin follows from linearity of expectation.

Cor: If $n \geq 100|S|^2$, then the expected number of collisions $\leq \frac{1}{2}n$, and the probability that there's a collision is $\leq \frac{1}{200}$. 

Proof: Markov.
That is, if \( n \) is sufficiently large with respect to \( S \), a typical table looks like this:

![A mostly empty table; some of the bins have 1 element; in a tiny fraction there's more than 1 element.]

In recitation, how to construct a table with \( n = O(1/\varepsilon) \) [so we don't waste] \( 1/\varepsilon^2 \) space and no collisions, for a static \( S \subseteq U \).

("perfect hashing")
What if $|S|$ is not known in advance?

**Table doubling**

- Start with some table size $n_0 = O(1)$.  
- When table is full, double it.  
  
  $A[1..n] \rightarrow A'[1..2n]$  
  
  Copy all the elements from $A$ to $A'$.  

* Simplifying assumption: no deletions, only insertions.

What's the running time?

- **Search** - $O(1)$  
- **Insert** - $O(n)$ - might need to double $|S|$

Think about an algorithm that uses the data structure:

For $i \leftarrow i \leftarrow k$,

Insert($x_i$)
What's the running time of this algorithm?

\[ O(k \cdot k) \]

Is this tight? No! The actual running time in the \( k \) insertion

\[ i \quad 1 \quad \ldots \quad n \quad n+1 \quad n+2 \quad \ldots \quad 2n \]

\[ O(1) \quad \ldots \quad O(1) \quad n \quad O(1) \quad \ldots \quad O(1) \quad 2n \quad \ldots \quad \]

\[ \underbrace{n_0} \quad \underbrace{2n_0} \]

time = \[ n_0 \cdot O(1) + n_0 + 2n_0 \cdot O(1) + 2n_0 + \ldots \]

\[ = \sum_{i=0}^{\log k} 2^i \cdot n_0 \cdot O(1) = O(k). \]

Amortized running time. The total running time for \( k \) operations divided by \( k \).

Next lecture methods of analysis of amortized running time.