LECTURE 6
Hashing

• Resolving collisions by chaining
• Choosing hash functions
• Universal hashing
• Constructing a set of universal hash functions
• Perfect hashing
Symbol-table problem

Symbol table $S$ holding $n$ records:

- key $[x]$
- Other fields containing satellite data

Operations on $S$:
- $\text{INSERT}(S, x)$
- $\text{DELETE}(S, x)$
- $\text{SEARCH}(S, k)$

How should the data structure $S$ be organized?
Direct-access table

**Idea:** Suppose that the keys are drawn from the set $U \subseteq \{0, 1, \ldots, m-1\}$, and keys are distinct. Set up an array $T[0 \ldots m-1]$: $T[k] = \begin{cases} x & \text{if } x \in K \text{ and } \text{key}[x] = k, \\ \text{NIL} & \text{otherwise.} \end{cases}$ Then, operations take $\Theta(1)$ time.

**Problem:** The range of keys can be large:
- 64-bit numbers (which represent 18,446,744,073,709,551,616 different keys),
- character strings (even larger!).
Hash functions

Solution: Use a hash function $h$ to map the universe $U$ of all keys into 
\{0, 1, \ldots, m-1\}:

When a record to be inserted maps to an already occupied slot in $T$, a collision occurs.
Resolving collisions by chaining

- Link records in the same slot into a list.

\[ h(49) = h(86) = h(52) = i \]

**Worst case:**
- Every key hashes to the same slot.
- Access time = \( \Theta(n) \) if \( |S| = n \)
Average-case analysis of chaining

We make the assumption of *simple uniform hashing*:

- Each key \( k \in S \) is equally likely to be hashed to any slot of table \( T \), independent of where other keys are hashed.

Let \( n \) be the number of keys in the table, and let \( m \) be the number of slots.

Define the **load factor** of \( T \) to be

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

= average number of keys per slot.
Search cost

The expected time for an unsuccessful search for a record with a given key is $= \Theta(1 + \alpha)$. 
Search cost

The expected time for an unsuccessful search for a record with a given key is

$$\Theta(1 + \alpha).$$

1. apply hash function
2. search the list
3. and access slot
Search cost

The expected time for an unsuccessful search for a record with a given key is

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

Expected search time = \( \Theta(1) \) if \( \alpha = O(1) \), or equivalently, if \( n = O(m) \).
Search cost

The expected time for an unsuccessful search for a record with a given key is

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

apply hash function
and access slot

Expected search time = \( \Theta(1) \) if \( \alpha = O(1) \),
or equivalently, if \( n = O(m) \).

A successful search has same asymptotic bound, but a rigorous argument is a little more complicated. (See textbook.)
Choosing a hash function

The assumption of simple uniform hashing is hard to guarantee, but several common techniques tend to work well in practice as long as their deficiencies can be avoided.

Desirata:

• A good hash function should distribute the keys uniformly into the slots of the table.
• Regularity in the key distribution should not affect this uniformity.
Division method

Assume all keys are integers, and define

\[ h(k) = k \mod m. \]

**Deficiency:** Don’t pick an \( m \) that has a small divisor \( d \). A preponderance of keys that are congruent modulo \( d \) can adversely affect uniformity.

**Extreme deficiency:** If \( m = 2^r \), then the hash doesn’t even depend on all the bits of \( k \):

- If \( k = 1011000111011010_2 \) and \( r = 6 \), then

\[ h(k) = 011010_2. \]

\( h(k) \)
Division method (continued)

\[ h(k) = k \mod m. \]

Pick \( m \) to be a prime not too close to a power of 2 or 10 and not otherwise used prominently in the computing environment.

**Annoyance:**
- Sometimes, making the table size a prime is inconvenient.

But, this method is popular, although the next method we’ll see is usually superior.
Multiplication method

Assume that all keys are integers, \( m = 2^r \), and our computer has \( w \)-bit words. Define

\[
h(k) = (A \cdot k \mod 2^w) \text{ rsh } (w - r),
\]

where rsh is the “bitwise right-shift” operator and \( A \) is an odd integer in the range \( 2^{w-1} < A < 2^w \).

- Don’t pick \( A \) too close to \( 2^{w-1} \) or \( 2^w \).
- Multiplication modulo \( 2^w \) is fast compared to division.
- The rsh operator is fast.
Multiplication method example

\[ h(k) = (A \cdot k \mod 2^w) \text{ rsh } (w - r) \]

Suppose that \( m = 8 = 2^3 \) and that our computer has \( w = 7 \)-bit words:

\[
\begin{array}{c}
1 & 0 & 1 & 1 & 0 & 0 & 1 \\
\times & & & & & & \\
1 & 1 & 0 & 1 & 0 & 1 & 1 \\
\hline
1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\
\end{array}
\]

\( h(k) = A \) and \( k = 1001010011 \)
A weakness of hashing

**Problem:** For any hash function $h$, a set of keys exists that can cause the average access time of a hash table to skyrocket.

- An adversary can pick all keys from \( \{ k \in U : h(k) = i \} \) for some slot $i$.

**Idea:** Choose the hash function at random, independently of the keys.

- Even if an adversary can see your code, he or she cannot find a bad set of keys, since he or she doesn’t know exactly which hash function will be chosen.
Universal hashing

**Definition.** Let $U$ be a universe of keys, and let $H$ be a finite collection of hash functions, each mapping $U$ to $\{0, 1, \ldots, m-1\}$. We say $H$ is *universal* if for all $x, y \in U$, where $x \neq y$, we have $|\{h \in H : h(x) = h(y)\}| \leq |H|/m$.

That is, the chance of a collision between $x$ and $y$ is $\leq 1/m$ if we choose $h$ randomly from $H$. 

\[ \{h : h(x) = h(y)\} \]

\[ \frac{|H|}{m} \]
Universality is good

**Theorem.** Let $h$ be a hash function chosen (uniformly) at random from a universal set $\mathcal{H}$ of hash functions. Suppose $h$ is used to hash $n$ arbitrary keys into the $m$ slots of a table $T$. Then, for a given key $x$, we have

$$E[\#\text{collisions with } x] < \frac{n}{m}.$$
Proof of theorem

Proof. Let $C_x$ be the random variable denoting the total number of collisions of keys in $T$ with $x$, and let

$$c_{xy} = \begin{cases} 1 & \text{if } h(x) = h(y), \\ 0 & \text{otherwise}. \end{cases}$$

Note: $E[c_{xy}] = 1/m$ and $C_x = \sum_{y \in T - \{x\}} c_{xy}$. 
Proof (continued)

\[ E[C_x] = E \left[ \sum_{y \in T \setminus \{x\}} c_{xy} \right] \]

- Take expectation of both sides.
Proof (continued)

\[ E[C_x] = E \left[ \sum_{y \in T - \{x\}} c_{xy} \right] \]

\[ = \sum_{y \in T - \{x\}} E[c_{xy}] \]

- Take expectation of both sides.
- Linearity of expectation.
Proof (continued)

\[ E[C_x] = E \left[ \sum_{y \in T \setminus \{x\}} c_{xy} \right] \]

\[ = \sum_{y \in T \setminus \{x\}} E[c_{xy}] \]

\[ = \sum_{y \in T \setminus \{x\}} 1/m \]

• Take expectation of both sides.

• Linearity of expectation.

• \( E[c_{xy}] = 1/m. \)
Proof (continued)

\[ E[C_x] = E \left[ \sum_{y \in T - \{x\}} c_{xy} \right] \]

\[ = \sum_{y \in T - \{x\}} E[c_{xy}] \]

\[ = \sum_{y \in T - \{x\}} 1/m \]

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

- **Take expectation of both sides.**
- **Linearity of expectation.**
- **\( E[c_{xy}] = 1/m. \)**
- **Algebra.**
Constructing a set of universal hash functions

Let \( m \) be prime. Decompose key \( k \) into \( r + 1 \) digits, each with value in the set \( \{0, 1, \ldots, m-1\} \). That is, let \( k = \langle k_0, k_1, \ldots, k_r \rangle \), where \( 0 \leq k_i < m \).

Randomized strategy:
Pick \( a = \langle a_0, a_1, \ldots, a_r \rangle \) where each \( a_i \) is chosen randomly from \( \{0, 1, \ldots, m-1\} \).

Define \( h_a(k) = \sum_{i=0}^{r} a_i k_i \mod m \).

Dot product, modulo \( m \)

How big is \( H = \{ h_a \} \)? \( |H| = m^{r+1} \). ← REMEMBER THIS!
Universality of dot-product hash functions

**Theorem.** The set $H = \{ h_a \}$ is universal.

**Proof.** Suppose that $x = \langle x_0, x_1, \ldots, x_r \rangle$ and $y = \langle y_0, y_1, \ldots, y_r \rangle$ be distinct keys. Thus, they differ in at least one digit position, wlog position 0. For how many $h_a \in H$ do $x$ and $y$ collide?

We must have $h_a(x) = h_a(y)$, which implies that

$$\sum_{i=0}^{r} a_i x_i \equiv \sum_{i=0}^{r} a_i y_i \pmod{m}.$$
Proof (continued)

Equivalently, we have

\[ \sum_{i=0}^{r} a_i (x_i - y_i) \equiv 0 \pmod{m} \]

or

\[ a_0 (x_0 - y_0) + \sum_{i=1}^{r} a_i (x_i - y_i) \equiv 0 \pmod{m}, \]

which implies that

\[ a_0 (x_0 - y_0) \equiv -\sum_{i=1}^{r} a_i (x_i - y_i) \pmod{m}. \]
Fact from number theory

**Theorem.** Let $m$ be prime. For any $z \in \mathbb{Z}_m$ such that $z \neq 0$, there exists a unique $z^{-1} \in \mathbb{Z}_m$ such that

$$z \cdot z^{-1} \equiv 1 \pmod{m}.$$

**Example:** $m = 7$.

<table>
<thead>
<tr>
<th>$z$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z^{-1}$</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>
Back to the proof

We have

\[ a_0(x_0 - y_0) \equiv -\sum_{i=1}^{r} a_i(x_i - y_i) \pmod{m}, \]

and since \( x_0 \neq y_0 \), an inverse \((x_0 - y_0)^{-1}\) must exist, which implies that

\[ a_0 \equiv \left( -\sum_{i=1}^{r} a_i(x_i - y_i) \right) \cdot (x_0 - y_0)^{-1} \pmod{m}. \]

Thus, for any choices of \( a_1, a_2, \ldots, a_r \), exactly one choice of \( a_0 \) causes \( x \) and \( y \) to collide.
**Q.** How many $h_a$’s cause $x$ and $y$ to collide?

**A.** There are $m$ choices for each of $a_1, a_2, \ldots, a_r$, but once these are chosen, exactly one choice for $a_0$ causes $x$ and $y$ to collide, namely

$$a_0 = \left( - \sum_{i=1}^{r} a_i (x_i - y_i) \right) \cdot (x_0 - y_0)^{-1} \mod m.$$ 

Thus, the number of $h_a$’s that cause $x$ and $y$ to collide is $m^r \cdot 1 = m^r = |H|/m$. 


Perfect hashing

Can we achieve \textit{worst case} \textsc{search} takes $\Theta(1)$ time?

Yes, if the set of $n$ keys is known in advance. For example, set of keywords in a document, files on a CD-ROM
Collisions

**Theorem.** Let $\mathcal{H}$ be a class of universal hash functions for a table of size $m = n^2$. Then, if we use a random $h \in \mathcal{H}$ to hash $n$ keys into the table, the expected number of collisions is at most $1/2$.

**Proof.** By the definition of universality, the probability that 2 given keys in the table collide under $h$ is $1/m = 1/n^2$. Since there are $\binom{n}{2}$ pairs of keys that can possibly collide, the expected number of collisions is

$$\binom{n}{2} \cdot \frac{1}{n^2} = \frac{n(n-1)}{2} \cdot \frac{1}{n^2} < \frac{1}{2}.$$
No collisions

**Corollary.** The probability of no collisions is at least $\frac{1}{2}$.

**Proof.** *Markov’s inequality* says that for any nonnegative random variable $X$, we have

$$\Pr\{X \geq t\} \leq \frac{E[X]}{t}.$$

Applying this inequality with $t = 1$, we find that the probability of $\geq 1$ collisions is $\leq 1/2$.

Thus, just by testing random hash functions in $H$, we’ll quickly find one that works.

**Problem:** $n^2$ is too big!
**Two-Level Scheme**

**IDEA:** Two-level scheme with universal hashing at both levels.

<table>
<thead>
<tr>
<th>m</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>986</td>
</tr>
<tr>
<td>1</td>
<td>431</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>40</td>
</tr>
</tbody>
</table>

![Diagram](image)

**Outer level:** Hash function $h$ is used to hash $n$ keys to $m = n$ slots.

$h_{31}(14) = h_{31}(27) = 1$
**Perfect hashing**

**IDEA: Two-level scheme with universal hashing at both levels.**

**Inner level:** If $n_j$ keys hash to slot $j$, use hash table $S_j$ with $m_j = n_j^2$ slots and choose each hash function $h_j$ so it produces zero collisions.
Analysis of storage

For the level-1 hash table $T$, choose $m = n$, and let $n_i$ be random variable for the number of keys that hash to slot $i$ in $T$. By using $n_i^2$ slots for the level-2 hash table $S_i$, the expected total storage required for the two-level scheme is

$$E \left[ \sum_{i=0}^{m-1} \Theta(n_i^2) \right] = \Theta(n).$$

See CLRS Theorem 11.10 for proof. It can be shown that

$$E \left[ \sum_{i=0}^{m-1} \Theta(n_i^2) \right] < 2n$$