Hashing

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong
This lecture will revisit the **dictionary search** problem, where we want to locate an integer \( q \) in a set of size \( n \) or declare the absence of \( q \). Binary search solves the problem in \( O(\log n) \) time (assuming a sorted array on the \( n \) integers). We will reduce the cost to \( O(1) \) in expectation with a structure called the *hash table*. 
The Dictionary Search Problem (Redefined)

$S$ is a set of $n$ integers. We want to preprocess $S$ into a data structure to answer the following queries efficiently:

- **(Dictionary search) query**: given an integer $q$, decide whether $q \in S$.

We will measure a data structure’s performance by:

- **Space consumption**: the number of memory cells occupied;
- **Query cost**: query time;
- **Preprocessing cost**: time of building the structure.
We can solve the problem by storing $S$ in a sorted array of length $n$ and answering a query with binary search. This ensures:

- Space consumption: $O(n)$;
- Query cost: $O(\log n)$;
- Preprocessing cost: $O(n \log n)$. 

Dictionary Search — Solution Based on Binary Search
We will improve the previous solution in expectation:

- Space consumption: $O(n)$
- Query cost: $O(\log n) \Rightarrow O(1)$ in expectation;
- Preprocessing cost: $O(n \log n) \Rightarrow O(n)$. 
Main idea: divide $S$ into small disjoint subsets such that a query only needs to search one subset.

We assume that every integer is in $[1, U]$. Denote by $[m]$ the set of integers from 1 to $m$.

A hash function $h$ is a function from $[U]$ to $[m]$. Namely, given any integer $k$, the function's output $h(k)$ is an integer in $[m]$.

The value $h(k)$ is called the hash value of $k$. 
Hash Table — Preprocessing

First, choose an integer \( m > 0 \), and a hash function \( h \) from \([U]\) to \([m]\).

Then, preprocess \( S \) as follows:

1. Create an array \( H \) of length \( m \).
2. For each \( i \in [1, m] \), create an empty linked list \( L_i \). Keep the head and tail pointers of \( L_i \) in \( H[i] \).
3. For each integer \( x \in S \):
   - Calculate the hash value \( h(x) \).
   - Insert \( x \) into \( L_{h(x)} \).

Space consumption: \( O(n + m) \).
Preprocessing time: \( O(n + m) \).
Hash Table — Querying

We answer a query with value $q$ as follows:

1. Calculate the hash value $h(q)$.
2. Scan the whole $L_{h(q)}$. If $q$ is not found, answer “no”; otherwise, answer “yes”.

Query time: $O(|L_{h(v)}|)$, where $|L_{h(v)}|$ is the number of elements in $L_{h(v)}$. 
Example

Let $S = \{34, 19, 67, 2, 81, 75, 92, 56\}$. Suppose that we choose $m = 5$ and $h(k) = 1 + (k \mod m)$.

To answer a query with $q = 57$, we scan all the elements in $L_3$ and answer “no”. For this hash function, the maximum query time is the cost of scanning a linked list of 3 elements.
Example

Let \( S = \{34, 19, 67, 2, 81, 75, 92, 56\} \). Suppose that we choose \( m = 5 \), and \( h(k) = 2 \).

For this hash function, the maximum query time is the cost of scanning a linked list of 8 elements (i.e., the worst possible).
A good hash function should create linked lists of roughly the same size.

Next we will introduce a technique that can choose a good hash function to guarantee $O(1)$ expected query time.
Let $\mathcal{H}$ be a family of hash functions from $[U]$ to $[m]$. $\mathcal{H}$ is \textbf{universal} if the following holds:

Let $k_1, k_2$ be two distinct integers in $[U]$. By picking a function $h \in \mathcal{H}$ uniformly at random, we guarantee that

$$\Pr[h(k_1) = h(k_2)] \leq \frac{1}{m}.$$ 

We will prove that universality ensures $O(1)$ expected query time. Then, we will describe a way to obtain such a good hash function.
Analysis of Query Time under Universality

We focus on the case where $q$ does not exist in $S$ (the case where it does is similar). Recall that our algorithm probes all the elements in the linked list $L_{h(q)}$. The query cost is therefore $O(|L_{h(q)}|)$.

Define random variable $X_i$ ($i \in [1, n]$) to be 1 if the $i$-th element $e$ of $S$ has the same hash value as $q$ (i.e., $h(e) = h(q)$), and 0 otherwise. Thus:

$$|L_{h(q)}| = \sum_{i=1}^{n} X_i$$
Analysis of Query Time under Universality

By universality, $\mathbb{P}r[X_i = 1] \leq 1/m$, meaning that

$$E[X_i] = 1 \cdot \mathbb{P}r[X_i = 1] + 0 \cdot \mathbb{P}r[X_i = 0] \leq 1/m.$$  

Hence:

$$E[|L_{h(q)}|] = \sum_{i=1}^{n} E[X_i] \leq n/m.$$  

By choosing $m = \Theta(n)$, we have $n/m = O(1)$.  

Yufei Tao
Designing a Universal Function

We now construct a universal family $\mathcal{H}$ of hash functions from $[U]$ to $[m]$.

- Pick a prime number $p$ such that $p \geq m$ and $p \geq U$.
- For every $\alpha \in \{1, 2, \ldots, p-1\}$ and every $\beta \in \{0, 1, \ldots, p-1\}$, define:
  \[ h_{\alpha,\beta}(k) = 1 + (((\alpha k + \beta) \mod p) \mod m). \]
- This defines $p(p-1)$ hash functions, which constitute our $\mathcal{H}$.

The proof of universality can be found in the appendix (not required for CSCI2100).
Existence of the Prime Number

Is it always possible to choose a desired prime number $p$?

Recall that the RAM model is defined with a word length $w$, namely, the number of bits in a word. Hence, $U \leq 2^w - 1$.

Number theory shows that there is at least one prime number between $x$ and $2x$. Hence, one can prepare in advance such a prime number $p$ in the range $[2^w, 2^{w+1}]$ and use this $p$ to construct a universal hash family.
We have shown that, for any set $S$ of $n$ integers, it is always possible to construct a hash table with the following guarantees on the dictionary search problem:

- Space $O(n)$.
- Preprocessing time $O(n)$.
- Query time $O(1)$ in expectation.
Appendix: Proof of Universality
(not required for CSCI2100)
The Prime Ring

Denote by $\mathbb{Z}_p$ the set of integers $\{0, 1, ..., p - 1\}$. $\mathbb{Z}_p$ forms a **commutative ring** under “$+$” and “$\cdot$” (**both defined using modulo** $p$). This means:

- $\mathbb{Z}_p$ is closed under $+$ and $\cdot$.
- $+$ satisfies commutativity and associativity.
  - $a + b = b + a$ (mod $p$) and $a + b + c = a + (b + c)$ (mod $p$)
- $+$ has a zero element, that is, $0 + a = a$ (mod $p$).
- Every element $a$ has an **additive inverse** $-a$, that is, $a + (-a) = 0$ (mod $p$).
- $\cdot$ satisfies commutativity and associativity.
  - $a \cdot b = b \cdot a$ (mod $p$) and $a \cdot b \cdot c = a \cdot (b \cdot c)$ (mod $p$)
- $\cdot$ modulo $p$ has a **one element**, that is, $1 \cdot a = a$ (mod $a$).
- $+$ and $\cdot$ satisfy distributivity.
  - $a \cdot (b + c) = a \cdot b + a \cdot c$ (mod $p$)
  - $(b + c) \cdot a = b \cdot a + c \cdot a$ (mod $p$)
The ring $\mathbb{Z}_p$ has several crucial properties. Let us start with:

**Lemma:** Let $a$ be a non-zero element in $\mathbb{Z}_p$. Then, $a \cdot j \neq a \cdot k \pmod{p}$ for any $j, k \in \mathbb{Z}_p$ with $j \neq k$.

**Proof:** Suppose without loss of generality $j > k$. Assume $a \cdot j = a \cdot k \pmod{p}$, then $a \cdot (j - k) = 0 \pmod{p}$. This means that $a \cdot (j - k)$ must be a multiple of $p$. Since $p$ is prime, either $a$ or $j - k$ must be a multiple of $p$. This is impossible because $a$ and $j - k$ are non-zero elements in $\mathbb{Z}_p$.

The lemma implies that $a \cdot 0, a \cdot 1, ..., a \cdot (p - 1)$ must take unique values in $\{0, 1, ..., p - 1\}$. 
The previous lemma implies:

**Corollary:** Every non-zero element $a$ has a unique multiplicative inverse $a^{-1}$, namely, $a \cdot a^{-1} = 1 \pmod{p}$.

In other words, $\mathbb{Z}_p$ is a division ring.
The next property then follows:

**Lemma:** Every equation \( a \cdot x + b = c \pmod{p} \) where \( a, b, c \) are in \( \mathbb{Z}_p \) and \( a \neq 0 \) has a unique solution in \( \mathbb{Z}_p \).

**Proof:**

\[
\begin{align*}
a \cdot x &= c - b \pmod{p} \\
x &= a^{-1} \cdot (c - b) \pmod{p}
\end{align*}
\]
Next, we will prove that the hash family $H$ we constructed in Slide 15 is universal. As before, let $k_1$ and $k_2$ be distinct integers in $[U]$.

**Fact 1:** Let

$$
g_{\alpha,\beta}(k_1) = (\alpha \cdot k_1 + \beta) \mod p$$
$$
g_{\alpha,\beta}(k_2) = (\alpha \cdot k_2 + \beta) \mod p$$

We must have: $g_{\alpha,\beta}(k_1) \neq g_{\alpha,\beta}(k_2)$.

**Proof:** Otherwise, it must hold that

$$\alpha \cdot k_1 + \beta = \alpha \cdot k_2 + \beta \pmod{p}$$
$$\Rightarrow \alpha \cdot (k_1 - k_2) = 0 \pmod{p}$$

which is not possible. \[\square.\]
Proof of Universality

How many different choices are there for the pair \((g(k_1), g(k_2))\)? The answer is at most \(p(p - 1)\) according to Fact 1: there are \(p^2\) possible pairs in \(\mathbb{Z}_p \times \mathbb{Z}_p\) but we need to exclude the \(p\) pairs where the two values are the same.

Recall that \(\mathcal{H}\) has \(p(p - 1)\) functions.

Next, we will prove a one-to-one mapping between the possible choices of \((g(k_1), g(k_2))\) and the hash functions in \(\mathcal{H}\).
Fact 2: Fix any two \( x, y \in \mathbb{Z}_p \) such that \( x \neq y \). There is a unique pair \((\alpha, \beta)\) — with \( \alpha \in \{1, 2, \ldots, p-1\} \) and \( \beta \in \{0, 1, \ldots, p-1\} \) — that makes \( g_{\alpha,\beta}(k_1) = x \) and \( g_{\alpha,\beta}(k_2) = y \).

Proof: Suppose that \( h \) is determined by \( \alpha, \beta \) selected as explained in Slide 15. Thus:

\[
\begin{align*}
\alpha \cdot k_1 + \beta &= x \pmod{p} \\
\alpha \cdot k_2 + \beta &= y \pmod{p}
\end{align*}
\]

Hence:

\[
\begin{align*}
\alpha \cdot (k_1 - k_2) &= x - y \pmod{p} \\
\Rightarrow \quad \alpha &= (k_1 - k_2)^{-1} \cdot (x - y) \pmod{p} \\
\Rightarrow \quad \beta &= x - (k_1 - k_2)^{-1} \cdot (x - y) \cdot k_1 \pmod{p}
\end{align*}
\]
Proof of Universality

Let $P$ be the set of pairs $(x, y)$ such that $x, y \in \mathbb{Z}_p$ and $x \neq y$.

By choosing $\alpha, \beta$ randomly in their respective ranges, we set $(g_{\alpha, \beta}(k_1), g_{\alpha, \beta}(k_2))$ to a pair $(x, y) \in P$ chosen uniformly at random.

Notice that $h(k_1) = h(k_2)$ if and only if $g_{\alpha, \beta}(k_1) = g_{\alpha, \beta}(k_2) \pmod{m}$. So now the question boils down to: how many pairs $(x, y)$ in $P$ satisfy $x = y \pmod{m}$?
Proof of Universality

How many pairs \((x, y)\) in \(P\) satisfy \(x = y \pmod{m}\)?

- For \(x = 0\), \(y\) can take \(m, 2m, 3m, \ldots\). The number of such \(y\)'s is no more than \([p/m] - 1 \leq (p - 1)/m\).

- For \(x = 1\), \(y\) can take \(m + 1, 2m + 1, 3m + 1, \ldots\). The number of such \(y\)'s is no more than \([p/m] - 1 \leq (p - 1)/m\).

- ...

Hence, the number of such pairs is no more than \(p(p - 1)/m = |P|/m\).

Now we conclude that the probability of \(h(k_1) = h(k_2)\) is at most \(1/m\).