# Learning Goals

- Understand the desiderata of hashing
- State the guarantee of universal hashing
- Understand the construction of universal hashing using finite fields

# The Fundamental Data-Structuring Problem

- A *data structure* is a way to store and organize data so that it can be used efficiently.

# The Fundamental Data-Structuring Problem

- A *data structure* is a way to store and organize data so that it can be used efficiently.
- We have a set $S$ of entries to store. Denote $|S|$ by $n$.

# The Fundamental Data-Structuring Problem

- A *data structure* is a way to store and organize data so that it can be used efficiently.
- We have a set $S$ of entries to store. Denote $|S|$ by $n$.
- Each entry $i$ has a *key* $k(i)$ belonging to a universe $U$.

# The Fundamental Data-Structuring Problem

- A *data structure* is a way to store and organize data so that it can be used efficiently.
- We have a set $S$ of entries to store. Denote $|S|$ by $n$.
- Each entry $i$ has a *key* $k(i)$ belonging to a universe $U$.
  - For student records, the key can be the student numbers.

# The Fundamental Data-Structuring Problem

- A *data structure* is a way to store and organize data so that it can be used efficiently.
- We have a set $S$ of entries to store. Denote $|S|$ by $n$.
- Each entry $i$ has a *key* $k(i)$ belonging to a universe $U$.
  - For student records, the key can be the student numbers.
  - For webpages, the key can be the URLs.

# The Fundamental Data-Structuring Problem

- A *data structure* is a way to store and organize data so that it can be used efficiently.
- We have a set $S$ of entries to store. Denote $|S|$ by $n$.
- Each entry $i$ has a *key* $k(i)$ belonging to a universe $U$.
  - For student records, the key can be the student numbers.
  - For webpages, the key can be the URLs.
- Basic operations we'd like to support:
  - $\textsc{Find}(k)$: given key value $k$, decide whether any entry in $S$ has that key, and if so, return a handle/pointer to the entry.

# The Fundamental Data-Structuring Problem

- A *data structure* is a way to store and organize data so that it can be used efficiently.
- We have a set $S$ of entries to store. Denote $|S|$ by $n$.
- Each entry $i$ has a *key* $k(i)$ belonging to a universe $U$.
  - For student records, the key can be the student numbers.
  - For webpages, the key can be the URLs.
- Basic operations we'd like to support:
  - $\textsc{Find}(k)$: given key value $k$, decide whether any entry in $S$ has that key, and if so, return a handle/pointer to the entry.
  - $\textsc{Insert}((k(i), i)$: insert entry $i$ with key value $k(i)$ to the current data set.

# The Fundamental Data-Structuring Problem

- A *data structure* is a way to store and organize data so that it can be used efficiently.
- We have a set $S$ of entries to store. Denote $|S|$ by $n$.
- Each entry $i$ has a *key* $k(i)$ belonging to a universe $U$.
  - For student records, the key can be the student numbers.
  - For webpages, the key can be the URLs.
- Basic operations we'd like to support:
  - $\text{FIND}(k)$: given key value $k$, decide whether any entry in $S$ has that key, and if so, return a handle/pointer to the entry.
  - $\text{INSERT}((k(i), i)$: insert entry $i$ with key value $k(i)$ to the current data set.
- There may be other operations such as , DELETE, MERGE, TRAVERSE, etc..

# Scenario for hashing

- The choice of a data structure depends on the scenario.

# Scenario for hashing

- The choice of a data structure depends on the scenario.
- When we talk about data structures, we are almost exclusively interested in the keys (and not the contents) of the entries, so we will make no distinction between an entry and its key, and refer to entry $i$ as if it is also its key.

# Scenario for hashing

- The choice of a data structure depends on the scenario.
- When we talk about data structures, we are almost exclusively interested in the keys (and not the contents) of the entries, so we will make no distinction between an entry and its key, and refer to entry $i$ as if it is also its key.
- A typical scenario is when $U$ is very large, but the number of entries is much smaller, i.e., $|S| \ll |U|$.

# Scenario for hashing

- The choice of a data structure depends on the scenario.
- When we talk about data structures, we are almost exclusively interested in the keys (and not the contents) of the entries, so we will make no distinction between an entry and its key, and refer to entry $i$ as if it is also its key.
- A typical scenario is when $U$ is very large, but the number of entries is much smaller, i.e., $|S| \ll |U|$.
  - Example: when $U$ is the set of all URLs.
- Opening an array for all possible key values is wasteful and often impractical.

# Scenario for hashing

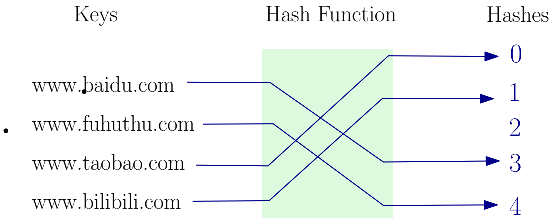
- The choice of a data structure depends on the scenario.
- When we talk about data structures, we are almost exclusively interested in the keys (and not the contents) of the entries, so we will make no distinction between an entry and its key, and refer to entry $i$ as if it is also its key.
- A typical scenario is when $U$ is very large, but the number of entries is much smaller, i.e., $|S| \ll |U|$.
  - Example: when $U$ is the set of all URLs.
- Opening an array for all possible key values is wasteful and often impractical.
- Using a linked list means very slow (linear time) $\textsc{Find}$ operation.

# Idea of Hashing

Open an array (*hash table*) with size $m$, with $m \ll |U|$. Have a function $h : U \to \{0, 1, \ldots, m-1\}$, and store entry $i$ at position $h(i)$.

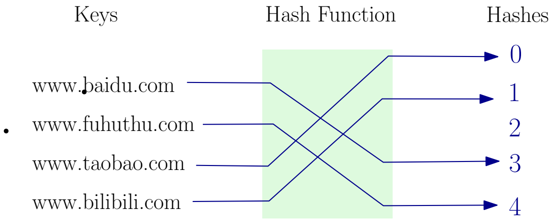
# Idea of Hashing

Open an array (*hash table*) with size $m$, with $m \ll |U|$. Have a function $h: U \to \{0, 1, \ldots, m-1\}$, and store entry $i$ at position $h(i)$.



# Idea of Hashing

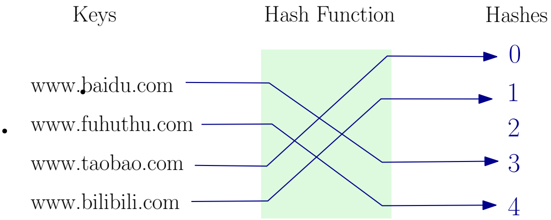
Open an array (*hash table*) with size $m$, with $m \ll |U|$. Have a function $h : U \to \{0, 1, \ldots, m-1\}$, and store entry $i$ at position $h(i)$.




- $h$ should be computed very fast, say, in $O(1)$ time.

# Idea of Hashing

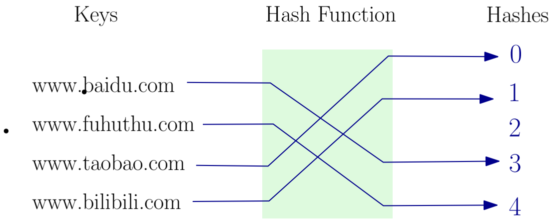
Open an array (*hash table*) with size $m$, with $m \ll |U|$. Have a function $h : U \to \{0, 1, \ldots, m-1\}$, and store entry $i$ at position $h(i)$.




- $h$ should be computed very fast, say, in $O(1)$ time.
- Ideally $m = O(n)$.

# Idea of Hashing

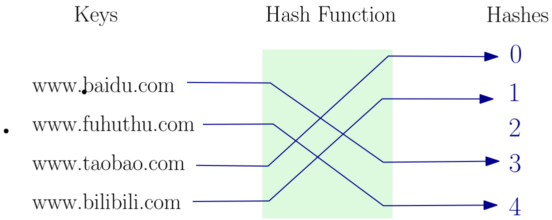
Open an array (*hash table*) with size $m$, with $m \ll |U|$. Have a function $h : U \to \{0, 1, \ldots, m-1\}$, and store entry $i$ at position $h(i)$.




- $h$ should be computed very fast, say, in $O(1)$ time.
- Ideally $m = O(n)$.
- What if two keys are mapped to the same position? I.e., if for $x, y \in U, x \neq y, h(x) = h(y)$.

# Idea of Hashing

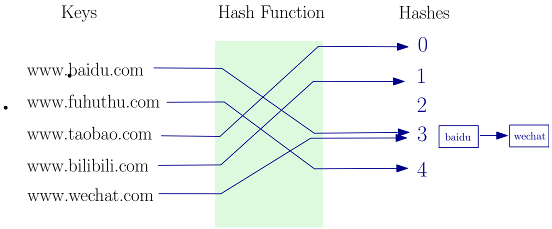
Open an array (*hash table*) with size $m$, with $m \ll |U|$. Have a function $h : U \to \{0, 1, \ldots, m-1\}$, and store entry $i$ at position $h(i)$.




- $h$ should be computed very fast, say, in $O(1)$ time.
- Ideally $m = O(n)$.
- What if two keys are mapped to the same position? I.e., if for $x, y \in U, x \neq y, h(x) = h(y)$.
  - This is called a *collision*.

# Dealing with collisions

Separate chaining: build a linked list at each entry of the hash table.




- $h$ should be computed very fast, say, in $O(1)$ time.
- Ideally $m = O(n)$.
- There should be few collisions.

# Failed Attempts

- Does there exist a particularly good function $h$?

# Failed Attempts

- Does there exist a particularly good function $h$?

**Proposition**

For any given function $h : U \to \{0, 1, \ldots, m-1\}$, if $|U| \geq (n-1)m+1$, then there exists $S \subseteq U$, with $|S| = n$, whose elements are mapped to the same position by $h$.

# Failed Attempts

- Does there exist a particularly good function $h$?

**Proposition**

For any given function $h : U \to \{0, 1, \ldots, m-1\}$, if $|U| \geq (n-1)m + 1$, then there exists $S \subseteq U$, with $|S| = n$, whose elements are mapped to the same position by $h$.

- Let $h$ be a function that maps any key to a position uniformly at random?

# Failed Attempts

- Does there exist a particularly good function $h$?

**Proposition**

For any given function $h : U \to \{0, 1, \ldots, m-1\}$, if $|U| \geq (n-1)m + 1$, then there exists $S \subseteq U$, with $|S| = n$, whose elements are mapped to the same position by $h$.

- Let $h$ be a function that maps any key to a position uniformly at random?
  - How do you keep the record of $h$ and access that record? We are back to where we started!

# Failed Attempts

- Does there exist a particularly good function $h$?

**Proposition**

For any given function $h : U \to \{0, 1, \ldots, m-1\}$, if $|U| \geq (n-1)m + 1$, then there exists $S \subseteq U$, with $|S| = n$, whose elements are mapped to the same position by $h$.

- Let $h$ be a function that maps any key to a position uniformly at random?
  - How do you keep the record of $h$ and access that record? We are back to where we started!
  - This amounts to choosing at random from a family of $m^{|U|}$ hash function.

# Universal Hashing

- It may suffice to keep a much smaller family $H$ of hash functions and choose one at random.

# Universal Hashing

- It may suffice to keep a much smaller family $H$ of hash functions and choose one at random.
- Each function in $H$ should be easy to remember and compute.

# Universal Hashing

- It may suffice to keep a much smaller family $H$ of hash functions and choose one at random.
- Each function in $H$ should be easy to remember and compute.
- A sample from $H$ should look "random enough".

# Universal Hashing

- It may suffice to keep a much smaller family $H$ of hash functions and choose one at random.
- Each function in $H$ should be easy to remember and compute.
- A sample from $H$ should look "random enough".

**Definition**

A family $H$ of hash functions is *universal* if for any $x \neq y$ in $U$,

$$\mathbf{Pr}_{h \leftarrow H}\left[h(x) = h(y)\right] \leq \frac{1}{m},$$

where $h \leftarrow H$ means $h$ is sampled from $H$ uniformly at random.

# Universal Hashing

- It may suffice to keep a much smaller family $H$ of hash functions and choose one at random.
- Each function in $H$ should be easy to remember and compute.
- A sample from $H$ should look "random enough".

**Definition**

A family $H$ of hash functions is *universal* if for any $x \neq y$ in $U$,

$$\mathbf{Pr}_{h \leftarrow H}\left[h(x) = h(y)\right] \leq \frac{1}{m},$$

where $h \leftarrow H$ means $h$ is sampled from $H$ uniformly at random.

**Remark**

The property is not assuming the keys are themselves randomly chosen from $U$! This is still worst case guarantee.

# Guarantees of Universal Hashgin

**Proposition**

Let $H$ be a universal hash functions family from $U$ to $\{0, \ldots, m-1\}$ and let $h$ be a hash function uniformly sampled from $H$. For *any* set $S \subseteq U$ with $|S| = n$ and any $x \in S$, the expected number of collisions of $x$ under $h$ is at most $n/m$.

# Guarantees of Universal Hashgin

**Proposition**

Let $H$ be a universal hash functions family from $U$ to $\{0, \ldots, m-1\}$ and let $h$ be a hash function uniformly sampled from $H$. For *any* set $S \subseteq U$ with $|S| = n$ and any $x \in S$, the expected number of collisions of $x$ under $h$ is at most $n/m$.

**Proof.**

For $x \neq y$ in $S$, let $I_{xy}$ be the indicator variable for the event that $h(x) = h(y)$, then $\mathbf{E}[I_{xy}] = \mathbf{Pr}[h(x) = h(y)] \leq \frac{1}{m}$.

# Guarantees of Universal Hashgin

**Proposition**

Let $H$ be a universal hash functions family from $U$ to $\{0, \ldots, m-1\}$ and let $h$ be a hash function uniformly sampled from $H$. For *any* set $S \subseteq U$ with $|S| = n$ and any $x \in S$, the expected number of collisions of $x$ under $h$ is at most $n/m$.

**Proof.**

For $x \neq y$ in $S$, let $I_{xy}$ be the indicator variable for the event that $h(x) = h(y)$, then $\mathbf{E}[I_{xy}] = \mathbf{Pr}[h(x) = h(y)] \leq \frac{1}{m}$.
The expected number of collisions is $\mathbf{E}[\sum_{y \in S \setminus \{x\}} I_{xy}] \leq \frac{n}{m}$. $\square$

# Guarantees of Universal Hashgin

**Proposition**

Let $H$ be a universal hash functions family from $U$ to $\{0, \ldots, m-1\}$ and let $h$ be a hash function uniformly sampled from $H$. For *any* set $S \subseteq U$ with $|S| = n$ and any $x \in S$, the expected number of collisions of $x$ under $h$ is at most $n/m$.

**Proof.**

For $x \neq y$ in $S$, let $I_{xy}$ be the indicator variable for the event that $h(x) = h(y)$, then $\mathbf{E}[I_{xy}] = \mathbf{Pr}[h(x) = h(y)] \leq \frac{1}{m}$.
The expected number of collisions is $\mathbf{E}[\sum_{y \in S \setminus \{x\}} I_{xy}] \leq \frac{n}{m}$. $\square$

- As long as $m = \Omega(n)$, the expected number of collisions is $O(1)$. Total time for FIND is $O(1)$.

# Construction of Universal Hashing Families

We introduce a popular construction for universal hashing function families.

# Construction of Universal Hashing Families

We introduce a popular construction for universal hashing function families.

Let $m$ be a prime number.

# Construction of Universal Hashing Families

We introduce a popular construction for universal hashing function families.
Let $m$ be a prime number.

**Fact**

*For any integer $N \geq 2$, there is a prime number in $\{N, N+1, \cdots, 2N-1\}$.*

# Construction of Universal Hashing Families

We introduce a popular construction for universal hashing function families.
Let $m$ be a prime number.

**Fact**

*For any integer $N \geq 2$, there is a prime number in $\{N, N+1, \cdots, 2N-1\}$.*

Suppose each key $x$ is a vector of $k$ integers $(x_1, \ldots, x_k)$, for $x_i \in \{0, \ldots, m-1\}$, i.e., $U = \{0, \ldots, m-1\}^k$.

# Construction of Universal Hashing Families

We introduce a popular construction for universal hashing function families.
Let $m$ be a prime number.

**Fact**

*For any integer $N \geq 2$, there is a prime number in $\{N, N+1, \cdots, 2N-1\}$.*

Suppose each key $x$ is a vector of $k$ integers $(x_1, \ldots, x_k)$, for $x_i \in \{0, \ldots, m-1\}$, i.e., $U = \{0, \ldots, m-1\}^k$.
Each hash function $H$ is indexed by a string of $k$ random numbers $\mathbf{r} = (r_1, \ldots, r_k) \in \{0, \ldots, m-1\}^k$, denoted as $h_{\mathbf{r}}$.

# Construction of Universal Hashing Families

We introduce a popular construction for universal hashing function families.
Let $m$ be a prime number.

**Fact**

*For any integer $N \geq 2$, there is a prime number in $\{N, N+1, \cdots, 2N-1\}$.*

Suppose each key $x$ is a vector of $k$ integers $(x_1, \ldots, x_k)$, for $x_i \in \{0, \ldots, m-1\}$, i.e., $U = \{0, \ldots, m-1\}^k$.
Each hash function $H$ is indexed by a string of $k$ random numbers $\mathbf{r} = (r_1, \ldots, r_k) \in \{0, \ldots, m-1\}^k$, denoted as $h_{\mathbf{r}}$.

$$\forall x \in U, h_{\mathbf{r}}(x) := r_1 x_1 + \ldots + r_k x_k \mod m.$$

# Proof of Universality

**Theorem**

*The family of hash functions thus constructed is universal.*

# Proof of Universality

## Theorem

*The family of hash functions thus constructed is universal.*

## Proof.

By definition, we need to show, for any $x \neq y$ in $U$,
$\mathbf{Pr}_{h_{\mathbf{r}}}[h_{\mathbf{r}}(x) = h_{\mathbf{r}}(y)] \leq \frac{1}{m}$.

# Proof of Universality

**Theorem**

*The family of hash functions thus constructed is universal.*

**Proof.**

By definition, we need to show, for any $x \neq y$ in $U$,
$\mathbf{Pr}_{h_{\mathbf{r}}}[h_{\mathbf{r}}(x) = h_{\mathbf{r}}(y)] \leq \frac{1}{m}$.
Since $x \neq y$, there exists $i \in \{1, \cdots, k\}$ such that $x_i \neq y_i$.

# Proof of Universality

**Theorem**

*The family of hash functions thus constructed is universal.*

**Proof.**

By definition, we need to show, for any $x \neq y$ in $U$,
$\mathbf{Pr}_{h_{\mathbf{r}}}[h_{\mathbf{r}}(x) = h_{\mathbf{r}}(y)] \leq \frac{1}{m}$.
Since $x \neq y$, there exists $i \in \{1, \cdots, k\}$ such that $x_i \neq y_i$.
In order for $h_{\mathbf{r}}(x) = h_{\mathbf{r}}(y)$, we must have

$$(x_i - y_i)r_i \equiv \sum_{j \neq i} (y_j - x_j) r_j \mod m. \tag{1}$$

# Proof of Universality

## Theorem

*The family of hash functions thus constructed is universal.*

## Proof.

By definition, we need to show, for any $x \neq y$ in $U$,
$\mathbf{Pr}_{h_\mathbf{r}}[h_\mathbf{r}(x) = h_\mathbf{r}(y)] \leq \frac{1}{m}$.
Since $x \neq y$, there exists $i \in \{1, \cdots, k\}$ such that $x_i \neq y_i$.
In order for $h_\mathbf{r}(x) = h_\mathbf{r}(y)$, we must have

$$(x_i - y_i)r_i \equiv \sum_{j \neq i}(y_j - x_j)r_j \mod m. \tag{1}$$

For any choice of $r_1, \ldots, r_{i-1}, r_{i+1}, \ldots, r_k$, there is a unique $r_i$ that solves (1). With probability $1/m$, $r_i$ is chosen to be that. $\square$