Introduction to Randomised Algorithms:
Randomised Hashing
Scenario:

- You are given an assignment to implement hashing.
- You will self-grade in pairs, testing and grading your partner’s implementation;
- Your partner plays dirty:
  - he analyses your hash function;
  - picks a sequence of the worst-case keys, causing your implementation to take $O(n)$ time to search.

- What would you do?
Scenario:

- You are given an assignment to implement hashing.
- You will self-grade in pairs, testing and grading your partner’s implementation;
- Your partner plays dirty:
  - he analyses your hash function;
  - picks a sequence of the worst-case keys, causing your implementation to take $O(n)$ time to search.

- What would you do?
Scenario:

- You are given an assignment to implement hashing.
- You will self-grade in pairs, testing and grading your partner’s implementation;
- Your partner plays dirty:
  - he analyses your hash function;
  - picks a sequence of the worst-case keys, causing your implementation to take $O(n)$ time to search.

- What would you do?
Scenario:

- You are given an assignment to implement hashing.
- You will self-grade in pairs, testing and grading your partner’s implementation;
- Your partner plays dirty:
  - he analyses your hash function;
  - picks a sequence of the worst-case keys, causing your implementation to take $O(n)$ time to search.

What would you do?
Scenario:

- You are given an assignment to implement hashing.
- You will self-grade in pairs, testing and grading your partner’s implementation;
- Your partner plays dirty:
  - he analyses your hash function;
  - picks a sequence of the worst-case keys, causing your implementation to take $O(n)$ time to search.

What would you do?
Scenario:

- You are given an assignment to implement hashing.
- You will self-grade in pairs, testing and grading your partner’s implementation;
- Your partner plays dirty:
  - he analyses your hash function;
  - picks a sequence of the worst-case keys, causing your implementation to take $O(n)$ time to search.

- What would you do?
Solution:

- Randomise your hashing!

- Pick a hash function randomly in a way that is independent of the keys that are actually going to be stored.

- In this way no single input always evokes worst case performance!

- Guarantees good performance on average over many runs of your program, no matter what keys adversary chooses.
Solution:

- Randomise your hashing!

- Pick a hash function randomly in a way that is independent of the keys that are actually going to be stored.

- In this way no single input always evokes worst case performance!

- Guarantees good performance on average over many runs of your program, no matter what keys adversary chooses.
Hash functions: randomised hashing

Solution:

- Randomise your hashing!

- Pick a hash function randomly in a way that is independent of the keys that are actually going to be stored.

- In this way no single input always evokes worst case performance!

- Guarantees good performance \textit{on average} over many runs of your program, no matter what keys adversary chooses.
Solution:

- Randomise your hashing!
- Pick a hash function randomly in a way that is independent of the keys that are actually going to be stored.
- In this way no single input always evokes worst case performance!
- Guarantees good performance on average over many runs of your program, no matter what keys adversary chooses.
Hash functions: randomised hashing

**Solution:**

- Randomise your hashing!

- Pick a hash function randomly in a way that is independent of the keys that are actually going to be stored.

- In this way no single input always evokes worst case performance!

- Guarantees good performance **on average** over many runs of your program, no matter what keys adversary chooses.
Towards randomised hashing: universal families of hash functions

- Let $H$ be a (finite) collection of hash functions that map a given universe $U$ of keys into the (much smaller) range $\{0, 1, m-1\}$;

- $H$ is said to be universal if:
  - for each pair of distinct keys $x, y \in U$, the number of hash functions $h \in H$ for which $h(x) = h(y)$ is $|H|/m$.

- In other words: if you take any two keys $x$ and $y$ and if you randomly pick a hash function from $H$, the chance of a collision between $x$ and $y$ is the same for all $x$ and $y$ and it is equal to $1/m$. 
Towards randomised hashing: universal families of hash functions

- Let $H$ be a (finite) collection of hash functions that map a given universe $U$ of keys into the (much smaller) range $\{0, 1, m-1\}$;

- $H$ is said to be **universal** if:
  - for each pair of distinct keys $x, y \in U$, the number of hash functions $h \in H$ for which $h(x) = h(y)$ is $|H|/m$.
  - In other words: if you take any two keys $x$ and $y$ and if you randomly pick a hash function from $H$, the chance of a collision between $x$ and $y$ is the same for all $x$ and $y$ and it is equal to $1/m$. 

Towards randomised hashing: universal families of hash functions

Let $H$ be a (finite) collection of hash functions that map a given universe $U$ of keys into the (much smaller) range $\{0, 1, m-1\}$;

$H$ is said to be **universal** if:

- for each pair of distinct keys $x, y \in U$, the number of hash functions $h \in H$ for which $h(x) = h(y)$ is $|H|/m$.

In other words: if you take any two keys $x$ and $y$ and if you randomly pick a hash function from $H$, the chance of a collision between $x$ and $y$ is the same for all $x$ and $y$ and it is equal to $1/m$. 

Shuiqiao Yang

COMP4121 Introduction to Randomised Algorithms
Towards randomised hashing: universal families of hash functions

Let $H$ be a (finite) collection of hash functions that map a given universe $U$ of keys into the (much smaller) range $\{0, 1, m - 1\}$;

$H$ is said to be **universal** if:
- for each pair of distinct keys $x, y \in U$, the number of hash functions $h \in H$ for which $h(x) = h(y)$ is $|H|/m$.

In other words: if you take any two keys $x$ and $y$ and if you randomly pick a hash function from $H$, the chance of a collision between $x$ and $y$ is the same for all $x$ and $y$ and it is equal to $1/m$. 
Assume a family of hash functions $H$ is universal. Let $x, y \in U$ be arbitrary keys. For a randomly chosen $h \in H$ let the random variable $c_{xy}$ be defined by $c_{xy} = 1$ if the keys $x$ and $y$ collide under $h$, i.e., $h(x) = h(y)$, and $c_{xy} = 0$ otherwise.

Fix $x$; then, by definition of a universal family, the expected value $E[c_{xy}]$ satisfies

$$E[c_{xy}] = P(h(y) = h(x)) \cdot 1 + P(h(y) \neq h(x)) \cdot 0$$

$$= \frac{|H|/m}{|H|} \cdot 1 + \left(1 - \frac{|H|/m}{|H|}\right) \cdot 0$$

$$= \frac{1}{m} \cdot 1 + \left(1 - \frac{1}{m}\right) \cdot 0$$

$$= \frac{1}{m}$$
Universal hashing

- Assume a family of hash functions $H$ is universal.
  - Let $x, y \in U$ be arbitrary keys. For a randomly chosen $h \in H$ let the random variable $c_{xy}$ be defined by $c_{xy} = 1$ if the keys $x$ and $y$ collide under $h$, i.e., $h(x) = h(y)$, and $c_{xy} = 0$ otherwise.

- Fix $x$; then, by definition of a universal family, the expected value $E[c_{xy}]$ satisfies

$$E[c_{xy}] = P(h(y) = h(x)) \cdot 1 + P(h(y) \neq h(x)) \cdot 0$$

$$= \frac{|H|/m}{|H|} \cdot 1 + \left(1 - \frac{|H|/m}{|H|}\right) \cdot 0$$

$$= \frac{1}{m} \cdot 1 + \left(1 - \frac{1}{m}\right) \cdot 0$$

$$= \frac{1}{m}$$
Universal hashing

- Assume a family of hash functions $H$ is universal.
  - Let $x, y \in U$ be arbitrary keys. For a randomly chosen $h \in H$ let the random variable $c_{xy}$ be defined by $c_{xy} = 1$ if the keys $x$ and $y$ collide under $h$, i.e., $h(x) = h(y)$, and $c_{xy} = 0$ otherwise.

- Fix $x$; then, by definition of a universal family, the expected value $E[c_{xy}]$ satisfies

  $$E[c_{xy}] = P(h(y) = h(x)) \cdot 1 + P(h(y) \neq h(x)) \cdot 0$$

  $$= \frac{|H|/m}{|H|} \cdot 1 + \left(1 - \frac{|H|/m}{|H|}\right) \cdot 0$$

  $$= \frac{1}{m} \cdot 1 + \left(1 - \frac{1}{m}\right) \cdot 0$$

  $$= \frac{1}{m}$$
Universal hashing

- Assume a family of hash functions $H$ is universal.
  - Let $x, y \in U$ be arbitrary keys. For a randomly chosen $h \in H$ let the random variable $c_{xy}$ be defined by $c_{xy} = 1$ if the keys $x$ and $y$ collide under $h$, i.e., $h(x) = h(y)$, and $c_{xy} = 0$ otherwise.

- Fix $x$; then, by definition of a universal family, the expected value $E[c_{xy}]$ satisfies

  \[
  E[c_{xy}] = P(h(y) = h(x)) \cdot 1 + P(h(y) \neq h(x)) \cdot 0
  \]
  \[
  = \frac{|H|/m}{|H|} \cdot 1 + \left(1 - \frac{|H|/m}{|H|}\right) \cdot 0
  \]
  \[
  = \frac{1}{m} \cdot 1 + \left(1 - \frac{1}{m}\right) \cdot 0
  \]
  \[
  = \frac{1}{m}
  \]
Universal hashing

- Assume that a family of hash functions $H$ is universal, and assume that we are hashing $n$ keys into a hash table of size $m$.

- Let $C_x$ be total number of collisions involving key $x$, i.e., let

  $$C_x = \sum_{y \neq x} c_{yx}$$

- Then the expected value $E[C_x]$ satisfies

  $$E[C_x] = \sum_{y \neq x} E[c_{yx}] = \frac{n - 1}{m} \quad (1)$$

- Consequently, if $n \leq m$ then the expected total number of collisions involving any particular key $x$ is less than 1.

- By the Markov inequality the probability that any particular slot of the hash table has lots of elements in it is small!
Universal hashing

- Assume that a family of hash functions $H$ is universal, and assume that we are hashing $n$ keys into a hash table of size $m$.

- Let $C_x$ be total number of collisions involving key $x$, i.e., let

$$C_x = \sum_{y \neq x} c_{yx}$$

- Then the expected value $E[C_x]$ satisfies

$$E[C_x] = \sum_{y \neq x} E[c_{yx}] = \frac{n - 1}{m} \quad (1)$$

- Consequently, if $n \leq m$ then the expected total number of collisions involving any particular key $x$ is less than 1.

- By the Markov inequality the probability that any particular slot of the hash table has lots of elements in it is small.
Universal hashing

- Assume that a family of hash functions $H$ is universal, and assume that we are hashing $n$ keys into a hash table of size $m$.

- Let $C_x$ be total number of collisions involving key $x$, i.e., let

$$C_x = \sum_{y \neq x} c_{yx}$$

- Then the expected value $E[C_x]$ satisfies

$$E[C_x] = \sum_{y \neq x} E[c_{yx}] = \frac{n - 1}{m} \quad (1)$$

- Consequently, if $n \leq m$ then the expected total number of collisions involving any particular key $x$ is less than 1.

- By the Markov inequality the probability that any particular slot of the hash table has lots of elements in it is small!
Assume that a family of hash functions $H$ is universal, and assume that we are hashing $n$ keys into a hash table of size $m$.

Let $C_x$ be total number of collisions involving key $x$, i.e., let

$$C_x = \sum_{y \neq x} c_{yx}$$

Then the expected value $E[C_x]$ satisfies

$$E[C_x] = \sum_{y \neq x} E[c_{yx}] = \frac{n - 1}{m}$$

Consequently, if $n \leq m$ then the expected total number of collisions involving any particular key $x$ is less than $1$.

By the Markov inequality the probability that any particular slot of the hash table has lots of elements in it is small!
Universal hashing

- Assume that a family of hash functions $H$ is universal, and assume that we are hashing $n$ keys into a hash table of size $m$.

- Let $C_x$ be total number of collisions involving key $x$, i.e., let

  $$C_x = \sum_{y \neq x} c_{yx}$$

- Then the expected value $E[C_x]$ satisfies

  $$E[C_x] = \sum_{y \neq x} E[c_{yx}] = \frac{n - 1}{m} \quad (1)$$

- Consequently, if $n \leq m$ then the expected total number of collisions involving any particular key $x$ is less than 1.

- By the Markov inequality the probability that any particular slot of the hash table has lots of elements in it is small!
Designing a universal family of hash Functions

- Choose the size $m$ of the hash table to be a prime number.

- Let $r$ be such that the size $|U|$ of the universe $U$ of all keys satisfies $m^r \leq |U| < m^{r+1}$ (i.e. $r = \lfloor \log_m |U| \rfloor$).

- Represent each key $x$ in base $m$, i.e., let $\vec{x} = (x_0, x_1, \ldots, x_r)$ be such that $0 \leq x_i < m$ for all $i$ such that $0 \leq i \leq r$ and such that

$$x = \sum_{i=0}^{r} x_i m^i$$

- Let $\vec{a} = (a_0, a_1, \ldots, a_r)$ be a vector of $r + 1$ randomly chosen elements from the set $\{0, 1, \ldots, m - 1\}$

- Define a corresponding hash function $h_{\vec{a}}(x) = \left( \sum_{i=0}^{r} x_i a_i \right) \mod m$

- Note that $h_{\vec{a}}(x) = \langle \vec{x}, \vec{a} \rangle \mod m$, where $\langle \vec{x}, \vec{a} \rangle$ denotes the dot product of vectors $\vec{x}$ and $\vec{a}$ (also called the scalar product).
Designing a universal family of hash Functions

- Choose the size $m$ of the hash table to be a prime number.

- Let $r$ be such that the size $|U|$ of the universe $U$ of all keys satisfies $m^r \leq |U| < m^{r+1}$ (i.e. $r = \lfloor \log_m |U| \rfloor$).

- Represent each key $x$ in base $m$, i.e., let $\vec{x} = (x_0, x_1, \ldots, x_r)$ be such that $0 \leq x_i < m$ for all $i$ such that $0 \leq i \leq r$ and such that

$$x = \sum_{i=0}^{r} x_i m^i$$

- Let $\vec{a} = (a_0, a_1, \ldots, a_r)$ be a vector of $r + 1$ randomly chosen elements from the set $\{0, 1, \ldots, m - 1\}$

- Define a corresponding hash function $h_{\vec{a}}(x) = \left( \sum_{i=0}^{r} x_i a_i \right) \mod m$

- Note that $h_{\vec{a}}(x) = \langle \vec{x}, \vec{a} \rangle \mod m$, where $\langle \vec{x}, \vec{a} \rangle$ denotes the dot product of vectors $\vec{x}$ and $\vec{a}$ (also called the scalar product).
Designing a universal family of hash Functions

- Choose the size $m$ of the hash table to be a prime number.

- Let $r$ be such that the size $|U|$ of the universe $U$ of all keys satisfies $m^r \leq |U| < m^{r+1}$ (i.e. $r = \lfloor \log_m |U| \rfloor$).

- Represent each key $x$ in base $m$, i.e., let $\vec{x} = (x_0, x_1, \ldots, x_r)$ be such that $0 \leq x_i < m$ for all $i$ such that $0 \leq i \leq r$ and such that

$$x = \sum_{i=0}^{r} x_i m^i$$

- Let $\vec{a} = (a_0, a_1, \ldots, a_r)$ be a vector of $r + 1$ randomly chosen elements from the set $\{0, 1, \ldots, m - 1\}$

- Define a corresponding hash function $h_{\vec{a}}(x) = \left(\sum_{i=0}^{r} x_i a_i \right) \mod m$

- Note that $h_{\vec{a}}(x) = \langle \vec{x}, \vec{a} \rangle \mod m$, where $\langle \vec{x}, \vec{a} \rangle$ denotes the dot product of vectors $\vec{x}$ and $\vec{a}$ (also called the scalar product).
Designing a universal family of hash Functions

- Choose the size $m$ of the hash table to be a prime number.

- Let $r$ be such that the size $|U|$ of the universe $U$ of all keys satisfies $m^r \leq |U| < m^{r+1}$ (i.e. $r = \lceil \log_m |U| \rceil$).

- Represent each key $x$ in base $m$, i.e., let $\vec{x} = (x_0, x_1, \ldots, x_r)$ be such that $0 \leq x_i < m$ for all $i$ such that $0 \leq i \leq r$ and such that

$$x = \sum_{i=0}^{r} x_i m^i$$

- Let $\vec{a} = (a_0, a_1, \ldots, a_r)$ be a vector of $r + 1$ randomly chosen elements from the set $\{0, 1, \ldots, m - 1\}$

- Define a corresponding hash function $h_{\vec{a}}(x) = \left( \sum_{i=0}^{r} x_i a_i \right) \mod m$

- Note that $h_{\vec{a}}(x) = \langle \vec{x}, \vec{a} \rangle \mod m$, where $\langle \vec{x}, \vec{a} \rangle$ denotes the dot product of vectors $\vec{x}$ and $\vec{a}$ (also called the scalar product).
Choose the size $m$ of the hash table to be a prime number.

Let $r$ be such that the size $|U|$ of the universe $U$ of all keys satisfies $m^r \leq |U| < m^{r+1}$ (i.e. $r = \lfloor \log_m |U| \rfloor$).

Represent each key $x$ in base $m$, i.e., let $\vec{x} = (x_0, x_1, \ldots, x_r)$ be such that $0 \leq x_i < m$ for all $i$ such that $0 \leq i \leq r$ and such that

$$x = \sum_{i=0}^{r} x_i m^i$$

Let $\vec{a} = (a_0, a_1, \ldots, a_r)$ be a vector of $r + 1$ randomly chosen elements from the set $\{0, 1, \ldots, m - 1\}$.

Define a corresponding hash function $h_{\vec{a}}(x) = \left( \sum_{i=0}^{r} x_i a_i \right) \mod m$

Note that $h_{\vec{a}}(x) = \langle \vec{x}, \vec{a} \rangle \mod m$, where $\langle \vec{x}, \vec{a} \rangle$ denotes the dot product of vectors $\vec{x}$ and $\vec{a}$ (also called the scalar product).
Choose the size $m$ of the hash table to be a prime number.

Let $r$ be such that the size $|U|$ of the universe $U$ of all keys satisfies $m^r \leq |U| < m^{r+1}$ (i.e. $r = \lceil \log_m |U| \rceil$).

Represent each key $x$ in base $m$, i.e., let $\vec{x} = (x_0, x_1, \ldots, x_r)$ be such that $0 \leq x_i < m$ for all $i$ such that $0 \leq i \leq r$ and such that

$$x = \sum_{i=0}^{r} x_i m^i$$

Let $\vec{a} = (a_0, a_1, \ldots, a_r)$ be a vector of $r + 1$ randomly chosen elements from the set $\{0, 1, \ldots, m - 1\}$.

Define a corresponding hash function $h_{\vec{a}}(x) = \left( \sum_{i=0}^{r} x_i a_i \right) \mod m$

Note that $h_{\vec{a}}(x) = \langle \vec{x}, \vec{a} \rangle \mod m$, where $\langle \vec{x}, \vec{a} \rangle$ denotes the dot product of vectors $\vec{x}$ and $\vec{a}$ (also called the scalar product).
Designing a universal family of hash Functions

- Choose the size $m$ of the hash table to be a prime number.

- Let $r$ be such that the size $|U|$ of the universe $U$ of all keys satisfies $m^r \leq |U| < m^{r+1}$ (i.e. $r = \lfloor \log_m |U| \rfloor$).

- Represent each key $x$ in base $m$, i.e., let $\mathbf{x} = (x_0, x_1, \ldots, x_r)$ be such that $0 \leq x_i < m$ for all $i$ such that $0 \leq i \leq r$ and such that

$$x = \sum_{i=0}^{r} x_i m^i$$

- Let $\mathbf{a} = (a_0, a_1, \ldots, a_r)$ be a vector of $r + 1$ randomly chosen elements from the set $\{0, 1, \ldots, m - 1\}$

- Define a corresponding hash function $h_{\mathbf{a}}(x) = \left( \sum_{i=0}^{r} x_i a_i \right) \mod m$

- Note that $h_{\mathbf{a}}(x) = \langle \mathbf{x}, \mathbf{a} \rangle \mod m$, where $\langle \mathbf{x}, \mathbf{a} \rangle$ denotes the dot product of vectors $\mathbf{x}$ and $\mathbf{a}$ (also called the scalar product).
Proving universality of family of hash functions $h_{\vec{a}}$

- Assume $x, y$ are two distinct keys.
- Let the corresponding sequences be $(x_0, x_1, \ldots, x_r)$ and $(y_0, y_1, \ldots, y_r)$;
- then

$$h_{\vec{a}}(x) = h_{\vec{a}}(y) \iff \sum_{i=0}^{r} x_i a_i = \sum_{i=0}^{r} y_i a_i \mod m$$

$$\iff \sum_{i=0}^{r} (x_i - y_i) a_i = 0 \mod m$$

- since $x \neq y$ there exists $k \leq r$ such that $x_k \neq y_k$;
- let us assume that $x_0 \neq y_0$;

- then $(x_0 - y_0)a_0 = -\sum_{i=1}^{r} (x_i - y_i) a_i \mod m$
Assume \( x, y \) are two distinct keys.

Let the corresponding sequences be \((x_0, x_1, \ldots, x_r)\) and \((y_0, y_1, \ldots, y_r)\);

then

\[
h_{\vec{a}}(x) = h_{\vec{a}}(y) \iff \sum_{i=0}^{r} x_i a_i = \sum_{i=0}^{r} y_i a_i \mod m
\]
\[
\iff \sum_{i=0}^{r} (x_i - y_i) a_i = 0 \mod m
\]

since \( x \neq y \) there exists \( k \leq r \) such that \( x_k \neq y_k \);

let us assume that \( x_0 \neq y_0 \);

then \( (x_0 - y_0)a_0 = -\sum_{i=1}^{r} (x_i - y_i) a_i \mod m \)
Assume \( x, y \) are two distinct keys.

Let the corresponding sequences be \((x_0, x_1, \ldots, x_r)\) and \((y_0, y_1, \ldots, y_r)\);

then

\[
\mathbf{h}_{\bar{a}}(x) = \mathbf{h}_{\bar{a}}(y) \iff \sum_{i=0}^{r} x_i a_i = \sum_{i=0}^{r} y_i a_i \mod m
\]

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

since \( x \neq y \) there exists \( k \leq r \) such that \( x_k \neq y_k \);

let us assume that \( x_0 \neq y_0 \);

then \( (x_0 - y_0)a_0 = -\sum_{i=1}^{r} (x_i - y_i)a_i \mod m \)
Assume $x, y$ are two distinct keys.

Let the corresponding sequences be $(x_0, x_1, \ldots, x_r)$ and $(y_0, y_1, \ldots, y_r)$;

then

$$h_{\vec{a}}(x) = h_{\vec{a}}(y) \iff \sum_{i=0}^{r} x_i a_i = \sum_{i=0}^{r} y_i a_i \mod m$$

$$\iff \sum_{i=0}^{r} (x_i - y_i) a_i = 0 \mod m$$

since $x \neq y$ there exists $k \leq r$ such that $x_k \neq y_k$;

let us assume that $x_0 \neq y_0$;

then $(x_0 - y_0)a_0 = -\sum_{i=1}^{r} (x_i - y_i)a_i \mod m$
Assume $x$, $y$ are two distinct keys.

Let the corresponding sequences be $(x_0, x_1, \ldots, x_r)$ and $(y_0, y_1, \ldots, y_r)$;

then

\[ h_{\vec{a}}(x) = h_{\vec{a}}(y) \iff \sum_{i=0}^{r} x_i a_i = \sum_{i=0}^{r} y_i a_i \mod m \]

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

since $x \neq y$ there exists $k \leq r$ such that $x_k \neq y_k$;

let us assume that $x_0 \neq y_0$;

then $(x_0 - y_0)a_0 = -\sum_{i=1}^{r} (x_i - y_i)a_i \mod m$
Assume $x, y$ are two distinct keys.

Let the corresponding sequences be $(x_0, x_1, \ldots, x_r)$ and $(y_0, y_1, \ldots, y_r)$;

then

$$h\vec{a}(x) = h\vec{a}(y) \iff \sum_{i=0}^{r} x_i a_i = \sum_{i=0}^{r} y_i a_i \mod m$$

$$\iff \sum_{i=0}^{r} (x_i - y_i) a_i = 0 \mod m$$

since $x \neq y$ there exists $k \leq r$ such that $x_k \neq y_k$;

let us assume that $x_0 \neq y_0$;

then $(x_0 - y_0) a_0 = -\sum_{i=1}^{r} (x_i - y_i) a_i \mod m$
Universality of family of hash functions $h_{\vec{a}}$ (continued)

- Since $m$ is a prime, every non-zero element $z \in \{0, 1, \ldots, m - 1\}$ has a multiplicative inverse $z^{-1}$, such that $z \cdot z^{-1} = 1 \pmod{m}$;

- since $x_0 - y_0 \neq 0$ we have that

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

implies

\[
a_0 = \left( -\sum_{i=1}^{r} (x_i - y_i)a_i \right) (x_0 - y_0)^{-1} \mod m
\]
Since \( m \) is a prime, every non-zero element \( z \in \{0, 1, \ldots, m - 1\} \) has a multiplicative inverse \( z^{-1} \), such that \( z \cdot z^{-1} = 1 \pmod{m} \);

since \( x_0 - y_0 \neq 0 \) we have that

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

implies

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

implies that

- for any two keys \( x, y \) such that \( x_0 \neq y_0 \)
- and
- for any randomly chosen \( r \) numbers \( a_1, a_2, \ldots, a_r \)

there exists exactly one \( a_0 \) (the one given by the above equation) such that for \( \vec{a} = \langle a_0, a_1, \ldots, a_r \rangle \) we have

\[ h_{\vec{a}}(x) = h_{\vec{a}}(y) \]
Since there are:

- $m^r$ sequences of the form $\langle a_1, \ldots, a_r \rangle$, each of which can uniquely be extended to a sequence $\vec{a} = \langle a_0, a_1, \ldots, a_r \rangle$ such that $h_{\vec{a}}(x) = h_{\vec{a}}(y)$

- and $m^{r+1}$ sequences of the form $\vec{a} = \langle a_0, a_1, \ldots, a_r \rangle$ in total,

we conclude that the probability to randomly chose a sequence $\vec{a} = \langle a_0, a_1, \ldots, a_r \rangle$ such that $h_{\vec{a}}(x) = h_{\vec{a}}(y)$, i.e., such that $x$ and $y$ collide, is equal to

$$\frac{m^r}{m^{r+1}} = \frac{1}{m}$$

Thus, the family $H$ is a universal collection of hash functions.
Universality of family of hash functions $h_{\vec{a}}$ (continued)

- Since there are:
  - $m^r$ sequences of the form $\langle a_1, \ldots, a_r \rangle$, each of which can uniquely be extended to a sequence $\vec{a} = \langle a_0, a_1, \ldots, a_r \rangle$ such that $h_{\vec{a}}(x) = h_{\vec{a}}(y)$
  - and $m^{r+1}$ sequences of the form $\vec{a} = \langle a_0, a_1, \ldots, a_r \rangle$ in total,

we conclude that the probability to randomly chose a sequence $\vec{a} = \langle a_0, a_1, \ldots, a_r \rangle$ such that $h_{\vec{a}}(x) = h_{\vec{a}}(y)$, i.e., such that $x$ and $y$ collide, is equal to

$$\frac{m^r}{m^{r+1}} = \frac{1}{m}$$

- Thus, the family $H$ is a universal collection of hash functions.
Since there are:

- $m^r$ sequences of the form $\langle a_1, \ldots, a_r \rangle$, each of which can uniquely be extended to a sequence $\vec{a} = \langle a_0, a_1, \ldots, a_r \rangle$ such that $h_{\vec{a}}(x) = h_{\vec{a}}(y)$

- and $m^{r+1}$ sequences of the form $\vec{a} = \langle a_0, a_1, \ldots, a_r \rangle$ in total,

we conclude that the probability to randomly chose a sequence $\vec{a} = \langle a_0, a_1, \ldots, a_r \rangle$ such that $h_{\vec{a}}(x) = h_{\vec{a}}(y)$, i.e., such that $x$ and $y$ collide, is equal to

$$\frac{m^r}{m^{r+1}} = \frac{1}{m}$$

Thus, the family $H$ is a universal collection of hash functions.
Since there are:

- \( m^r \) sequences of the form \( \langle a_1, \ldots, a_r \rangle \), each of which can uniquely be extended to a sequence \( \vec{a} = \langle a_0, a_1, \ldots, a_r \rangle \) such that \( h_{\vec{a}}(x) = h_{\vec{a}}(y) \)

- and \( m^{r+1} \) sequences of the form \( \vec{a} = \langle a_0, a_1, \ldots, a_r \rangle \) in total,

we conclude that the probability to randomly chose a sequence \( \vec{a} = \langle a_0, a_1, \ldots, a_r \rangle \) such that \( h_{\vec{a}}(x) = h_{\vec{a}}(y) \), i.e., such that \( x \) and \( y \) collide, is equal to

\[
\frac{m^r}{m^{r+1}} = \frac{1}{m}
\]

Thus, the family \( H \) is a universal collection of hash functions.
Universality of family of hash functions $h_{\vec{a}}$ (continued)

- Since there are:
  - $m^r$ sequences of the form $\langle a_1, \ldots, a_r \rangle$, each of which can uniquely be extended to a sequence $\vec{a} = \langle a_0, a_1, \ldots, a_r \rangle$ such that $h_{\vec{a}}(x) = h_{\vec{a}}(y)$
  - and $m^{r+1}$ sequences of the form $\vec{a} = \langle a_0, a_1, \ldots, a_r \rangle$ in total,

we conclude that the probability to randomly chose a sequence $\vec{a} = \langle a_0, a_1, \ldots, a_r \rangle$ such that $h_{\vec{a}}(x) = h_{\vec{a}}(y)$, i.e., such that $x$ and $y$ collide, is equal to

$$\frac{m^r}{m^{r+1}} = \frac{1}{m}$$

- Thus, the family $H$ is a universal collection of hash functions.
Using universal family of hash functions $h_{\vec{a}}$:

- Pick $r = \lfloor \log_m |U| \rfloor$, so that $m^r \leq |U| < m^{r+1}$;

- For each run, pick a hash function by randomly picking a vector $\vec{a} = (a_0, a_1, \ldots, a_r)$ such that $0 \leq a_i < m$ for all $i, 0 \leq i \leq r$.

- During each run use that function on all keys.

- Note that the dot (scalar) product $\langle x, y \rangle$ in

$$h_{\vec{a}}(x) = \left( \sum_{i=0}^{r} x_i a_i \right) \mod m = \langle x, a \rangle \mod m;$$

can be computed very fast on modern hardware.
Using universal family of hash functions $h_{\vec{a}}$:

- Pick $r = \lfloor \log_m |U| \rfloor$, so that $m^r \leq |U| < m^{r+1}$;

- For each run, pick a hash function by randomly picking a vector $\vec{a} = (a_0, a_1, \ldots, a_r)$ such that $0 \leq a_i < m$ for all $i$, $0 \leq i \leq r$.

- During each run use that function on all keys.

- Note that the dot (scalar) product $\langle x, y \rangle$ in

$$h_{\vec{a}}(x) = \left( \sum_{i=0}^{r} x_i a_i \right) \mod m = \langle x, a \rangle \mod m;$$

can be computed very fast on modern hardware.
Using universal family of hash functions $h_{\vec{a}}$:

- Pick $r = \lfloor \log_m |U| \rfloor$, so that $m^r \leq |U| < m^{r+1}$;

- For each run, pick a hash function by randomly picking a vector $\vec{a} = (a_0, a_1, \ldots, a_r)$ such that $0 \leq a_i < m$ for all $i$, $0 \leq i \leq r$.

- During each run use that function on all keys.

- Note that the dot (scalar) product $\langle x, y \rangle$ in

$$h_{\vec{a}}(x) = \left( \sum_{i=0}^{r} x_i a_i \right) \mod m = \langle x, a \rangle \mod m;$$

can be computed very fast on modern hardware.
Using universal family of hash functions $h_{\vec{a}}$:

- Pick $r = \lfloor \log_m |U| \rfloor$, so that $m^r \leq |U| < m^{r+1}$;

- For each run, pick a hash function by randomly picking a vector $\vec{a} = (a_0, a_1, \ldots, a_r)$ such that $0 \leq a_i < m$ for all $i$, $0 \leq i \leq r$.

- During each run use that function on all keys.

- Note that the dot (scalar) product $\langle x, y \rangle$ in

$$h_{\vec{a}}(x) = \left( \sum_{i=0}^{r} x_i a_i \right) \mod m = \langle x, a \rangle \mod m;$$

  can be computed very fast on modern hardware.
Applications of Universal Hash Families: Perfect Hashing

Problem:

- Assume that you need a **static hash table** to store $n$ keys (i.e., a look-up table; no insertions or deletions, just search).

- The size of the table should be linear in $n$.

- The table should be completely collision free.

- The corresponding hash function should be very efficient to compute.

To get such a table we will employ a randomised **design method**; however, the resulting hash function will be **completely deterministic**.
Problem:

- Assume that you need a static hash table to store \( n \) keys (i.e., a look-up table; no insertions or deletions, just search).

- The size of the table should be linear in \( n \).

- The table should be completely collision free.

- The corresponding hash function should be very efficient to compute.

To get such a table we will employ a randomised design method; however, the resulting hash function will be completely deterministic.
Problem:

- Assume that you need a **static hash table** to store $n$ keys (i.e., a look-up table; no insertions or deletions, just search).

- The size of the table should be linear in $n$.

- The table should be completely collision free.

- The corresponding hash function should be very efficient to compute.

To get such a table we will employ a randomised *design method*; however, the resulting hash function will be completely deterministic.
Problem:

- Assume that you need a static hash table to store \( n \) keys (i.e., a look-up table; no insertions or deletions, just search).

- The size of the table should be linear in \( n \).

- The table should be completely collision free.

- The corresponding hash function should be very efficient to compute.

To get such a table we will employ a randomised design method; however, the resulting hash function will be completely deterministic.
Applications of Universal Hash Families: Perfect Hashing

Problem:

- Assume that you need a static hash table to store $n$ keys (i.e., a look-up table; no insertions or deletions, just search).
- The size of the table should be linear in $n$.
- The table should be completely collision free.
- The corresponding hash function should be very efficient to compute.

To get such a table we will employ a randomised design method; however, the resulting hash function will be completely deterministic.
Method: trial and error procedure with very low probability of many consecutive failures.

First step:
- given $n$ keys we will be constructing hash tables of size $m < 2n^2$ using universal hashing;

probability that such a table is collision free will be $> 1/2$.

How do we accomplish this? We use a randomised design procedure:

- We pick the least prime $m$ such that $m > n^2$; then $m < 2n^2$ (Note: for every $x > 1$ there exists a prime $m$ such that $x \leq m < 2x$; here $m > n^2$ because $n^2$ cannot be prime)

- We pick a random vector $\vec{a}$ and hash all keys using the corresponding hash function $h_{\vec{a}}$ from the universal family.
Designing a Perfect Hash table

- **Method:** trial and error procedure with very low probability of many consecutive failures.

- **First step:**
  - given \( n \) keys we will be constructing hash tables of size \( m < 2n^2 \) using universal hashing;
  - probability that such a table is collision free will be \( > 1/2 \).

- How do we accomplish this? We use a randomised design procedure:
  - We pick the least prime \( m \) such that \( m > n^2 \); then \( m < 2n^2 \)
    (Note: for every \( x > 1 \) there exists a prime \( m \) such that \( x \leq m < 2x \); here \( m > n^2 \) because \( n^2 \) cannot be prime)
  - We pick a random vector \( \vec{a} \) and hash all keys using the corresponding hash function \( h_{\vec{a}} \) from the universal family.
Designing a Perfect Hash table

- **Method:** trial and error procedure with very low probability of many consecutive failures.

- **First step:**
  - given $n$ keys we will be constructing hash tables of size $m < 2n^2$ using universal hashing;
  - probability that such a table is collision free will be $> 1/2$.

- How do we accomplish this? We use a randomised design procedure:
  - We pick the least prime $m$ such that $m > n^2$; then $m < 2n^2$ (Note: for every $x > 1$ there exists a prime $m$ such that $x \leq m < 2x$; here $m > n^2$ because $n^2$ cannot be prime)
  - We pick a random vector $\vec{a}$ and hash all keys using the corresponding hash function $h_{\vec{a}}$ from the universal family.
Designing a Perfect Hash table

- **Method:** trial and error procedure with very low probability of many consecutive failures.

- **First step:**
  - given $n$ keys we will be constructing hash tables of size $m < 2n^2$ using universal hashing;
  - probability that such a table is collision free will be $> 1/2$.

- How do we accomplish this? We use a randomised design procedure:
  - We pick the least prime $m$ such that $m > n^2$; then $m < 2n^2$ (Note: for every $x > 1$ there exists a prime $m$ such that $x \leq m < 2x$; here $m > n^2$ because $n^2$ cannot be prime)
  - We pick a random vector $\vec{a}$ and hash all keys using the corresponding hash function $h_{\vec{a}}$ from the universal family.
Designing a Perfect Hash table

- **Method:** trial and error procedure with very low probability of many consecutive failures.

- **First step:**
  - given $n$ keys we will be constructing hash tables of size $m < 2n^2$ using universal hashing;
  - probability that such a table is collision free will be $> 1/2$.

- How do we accomplish this? We use a randomised design procedure:
  - We pick the least prime $m$ such that $m > n^2$; then $m < 2n^2$ (Note: for every $x > 1$ there exists a prime $m$ such that $x \leq m < 2x$; here $m > n^2$ because $n^2$ cannot be prime)
  - We pick a random vector $\vec{a}$ and hash all keys using the corresponding hash function $h_{\vec{a}}$ from the universal family.
Designing a Perfect Hash table

- **Method:** trial and error procedure with very low probability of many consecutive failures.

- **First step:**
  - given \( n \) keys we will be constructing hash tables of size \( m < 2n^2 \) using universal hashing;
  - probability that such a table is collision free will be \( > 1/2 \).

- How do we accomplish this? We use a randomised design procedure:
  - We pick the least prime \( m \) such that \( m > n^2 \); then \( m < 2n^2 \).
    (Note: for every \( x > 1 \) there exists a prime \( m \) such that \( x \leq m < 2x \); here \( m > n^2 \) because \( n^2 \) cannot be prime)
  - We pick a random vector \( \vec{a} \) and hash all keys using the corresponding hash function \( h_{\vec{a}} \) from the universal family.
Method: trial and error procedure with very low probability of many consecutive failures.

First step:
- given $n$ keys we will be constructing hash tables of size $m < 2n^2$ using universal hashing;
- probability that such a table is collision free will be $> 1/2$.

How do we accomplish this? We use a randomised design procedure:
- We pick the least prime $m$ such that $m > n^2$; then $m < 2n^2$
  (Note: for every $x > 1$ there exists a prime $m$ such that $x \leq m < 2x$; here $m > n^2$ because $n^2$ cannot be prime)
- We pick a random vector $\vec{a}$ and hash all keys using the corresponding hash function $h_{\vec{a}}$ from the universal family.
Designing a Perfect Hash table

- **Method:** trial and error procedure with very low probability of many consecutive failures.

- **First step:**
  - given $n$ keys we will be constructing hash tables of size $m < 2n^2$ using universal hashing;
  - probability that such a table is collision free will be $> 1/2$.

- How do we accomplish this? We use a randomised design procedure:
  - We pick the least prime $m$ such that $m > n^2$; then $m < 2n^2$ (Note: for every $x > 1$ there exists a prime $m$ such that $x \leq m < 2x$; here $m > n^2$ because $n^2$ cannot be prime)
  - We pick a random vector $\vec{a}$ and hash all keys using the corresponding hash function $h_{\vec{a}}$ from the universal family.
Given \( n \) keys, there will be \( \binom{n}{2} \) pairs of keys.

By universality of the family of hash functions used, for each pair of keys probability of a collision is \( \frac{1}{m} \).

Since \( m \geq n^2 \) we have \( \frac{1}{m} \leq \frac{1}{n^2} \).

Thus, the expected total number of collisions in the table is at most

\[
\binom{n}{2} \frac{1}{m} \leq \frac{n(n-1)}{2} \frac{1}{n^2} < \frac{1}{2}
\]

Note that if \( Y = \sum_{i=1}^{j} X_i \) then \( E(Y) = \sum_{i=1}^{j} E(X_i) \) regardless of whether variables \( X_i \) are independent or not.
Given $n$ keys, there will be $\binom{n}{2}$ pairs of keys.

By universality of the family of hash functions used, for each pair of keys probability of a collision is $\frac{1}{m}$.

Since $m \geq n^2$ we have $\frac{1}{m} \leq \frac{1}{n^2}$.

Thus, the expected total number of collisions in the table is at most

$$\binom{n}{2} \cdot \frac{1}{m} \leq \frac{n(n-1)}{2} \cdot \frac{1}{n^2} < \frac{1}{2}$$

Note that if $Y = \sum_{i=1}^{j} X_i$ then $E(Y) = \sum_{i=1}^{j} E(X_i)$ regardless of whether variables $X_i$ are independent or not.
Designing a Perfect Hash table (continued)

- Given $n$ keys, there will be $\binom{n}{2}$ pairs of keys.

- By universality of the family of hash functions used, for each pair of keys probability of a collision is $\frac{1}{m}$.

- Since $m \geq n^2$ we have $\frac{1}{m} \leq \frac{1}{n^2}$.

- Thus, the expected total number of collisions in the table is at most

  \[
  \binom{n}{2} \frac{1}{m} \leq \frac{n(n-1)}{2} \frac{1}{n^2} < \frac{1}{2}
  \]

- Note that if $Y = \sum_{i=1}^{j} X_i$ then $E(Y) = \sum_{i=1}^{j} E(X_i)$ regardless of whether variables $X_i$ are independent or not.
Given \( n \) keys, there will be \( \binom{n}{2} \) pairs of keys.

By universality of the family of hash functions used, for each pair of keys probability of a collision is \( \frac{1}{m} \).

Since \( m \geq n^2 \) we have \( \frac{1}{m} \leq \frac{1}{n^2} \).

Thus, the expected total number of collisions in the table is at most

\[
\left( \binom{n}{2} \right) \frac{1}{m} \leq \frac{n(n-1)}{2} \frac{1}{n^2} < \frac{1}{2}
\]

Note that if \( Y = \sum_{i=1}^{j} X_i \) then \( E(Y) = \sum_{i=1}^{j} E(X_i) \) regardless of whether variables \( X_i \) are independent or not.
Given $n$ keys, there will be $\binom{n}{2}$ pairs of keys.

By universality of the family of hash functions used, for each pair of keys probability of a collision is $\frac{1}{m}$.

Since $m \geq n^2$ we have $\frac{1}{m} \leq \frac{1}{n^2}$.

Thus, the expected total number of collisions in the table is at most

$$\binom{n}{2} \frac{1}{m} \leq \frac{n(n-1)}{2} \frac{1}{n^2} < \frac{1}{2}$$

Note that if $Y = \sum_{i=1}^{j} X_i$ then $E(Y) = \sum_{i=1}^{j} E(X_i)$ regardless of whether variables $X_i$ are independent or not.
For the given $n$ keys, we have constructed a table of size $m < 2n^2$, such that the expected total number of collisions is $< 1/2$.

Let $X$ be the random variable equal to the number of collisions in thus constructed table.

By the Markov Inequality with $t = 1$ we now get that

$$P\{X \geq 1\} \leq \frac{E[X]}{1} < \frac{1}{2}$$

Thus, if we keep picking hash functions at random from a universal family, the probability that there will be at least one collision in each of $k$ consecutive attempts (i.e., that $X \geq 1$ in each attempt) is smaller than $(1/2)^k$, which rapidly tends to 0.

Consequently, after a few random trial-and-error attempts we will obtain a collision free hash table of size $< 2n^2$. 

Shuiqiao Yang

COMP4121

Introduction to Randomised Algorithms
Designing a Perfect Hash table (continued)

- For the given $n$ keys, we have constructed a table of size $m < 2n^2$, such that the expected total number of collisions is $< 1/2$.

- Let $X$ be the random variable equal to the number of collisions in thus constructed table.

- By the Markov Inequality with $t = 1$ we now get that

  $$P\{X \geq 1\} \leq \frac{E[X]}{1} < \frac{1}{2}$$

- Thus, if we keep picking hash functions at random from a universal family, the probability that there will be at least one collision in each of $k$ consecutive attempts (i.e., that $X \geq 1$ in each attempt) is smaller than $(1/2)^k$, which rapidly tends to 0.

- Consequently, after a few random trial-and-error attempts we will obtain a collision free hash table of size $< 2n^2$. 

For the given $n$ keys, we have constructed a table of size $m < 2n^2$, such that the expected total number of collisions is $< 1/2$.

Let $X$ be the random variable equal to the number of collisions in thus constructed table.

By the Markov Inequality with $t = 1$ we now get that

$$P\{X \geq 1\} \leq \frac{E[X]}{1} < \frac{1}{2}$$

Thus, if we keep picking hash functions at random from a universal family, the probability that there will be at least one collision in each of $k$ consecutive attempts (i.e., that $X \geq 1$ in each attempt) is smaller than $(1/2)^k$, which rapidly tends to 0.

Consequently, after a few random trial-and-error attempts we will obtain a collision free hash table of size $< 2n^2$. 
For the given \( n \) keys, we have constructed a table of size \( m < 2n^2 \), such that the expected total number of collisions is \(< 1/2\).

Let \( X \) be the random variable equal to the number of collisions in thus constructed table.

By the Markov Inequality with \( t = 1 \) we now get that

\[
P\{X \geq 1\} \leq \frac{E[X]}{1} < \frac{1}{2}
\]

Thus, if we keep picking hash functions at random from a universal family, the the probability that there will be at least one collision in each of \( k \) consecutive attempts (i.e., that \( X \geq 1 \) in each attempt) is smaller than \((1/2)^k\), which rapidly tends to 0.

Consequently, after a few random trial-and-error attempts we will obtain a collision free hash table of size \(< 2n^2\).
Designing a Perfect Hash table (continued)

- For the given \( n \) keys, we have constructed a table of size \( m < 2n^2 \), such that the expected total number of collisions is \( < 1/2 \).

- Let \( X \) be the random variable equal to the number of collisions in thus constructed table.

- By the Markov Inequality with \( t = 1 \) we now get that

\[
P\{X \geq 1\} \leq \frac{E[X]}{1} < \frac{1}{2}
\]

- Thus, if we keep picking hash functions at random from a universal family, the probability that there will be at least one collision in each of \( k \) consecutive attempts (i.e., that \( X \geq 1 \) in each attempt) is smaller than \((1/2)^k\), which rapidly tends to 0.

- Consequently, after a few random trial-and-error attempts we will obtain a collision free hash table of size \( < 2n^2 \).
How many trials $N$ do we expect to have to make before we hit a collision free hash table of size $< 2n^2$?

Let the probability of failure be denoted by $p$; then $p < 1/2$, and the probability of a success is $1 - p$.

Thus,

$$E[N] = 1 \cdot (1 - p) + 2 \cdot p(1 - p) + 3 \cdot p^2(1 - p) + 4 \cdot p^3(1 - p) + \ldots$$

$$= (1 - p)(1 + 2p + 3p^2 + 4p^3 + \ldots)$$

We have already shown that $1 + 2p + 3p^2 + 4p^3 + \ldots = \frac{1}{(1-p)^2}$.

Thus, $E[N] = \frac{1}{1-p}$; since $p < 1/2$ we get $E[N] < 2$. 

How many trials \( N \) do we expect to have to make before we hit a collision free hash table of size \( < 2n^2 \)?

Let the probability of failure be denoted by \( p \); then \( p < 1/2 \), and the probability of a success is \( 1 - p \).

Thus,

\[
E[N] = 1 \cdot (1 - p) + 2 \cdot p(1 - p) + 3 \cdot p^2(1 - p) + 4 \cdot p^3(1 - p) + \ldots \\
= (1 - p)(1 + 2p + 3p^2 + 4p^3 + \ldots)
\]

We have already shown that \( 1 + 2p + 3p^2 + 4p^3 + \ldots = \frac{1}{(1-p)^2} \).

Thus, \( E[N] = \frac{1}{1-p} \); since \( p < 1/2 \) we get \( E[N] < 2 \).
How many trials $N$ do we expect to have to make before we hit a collision free hash table of size $< 2n^2$?

Let the probability of failure be denoted by $p$; then $p < 1/2$, and the probability of a success is $1 - p$.

Thus,

$$E[N] = 1 \cdot (1 - p) + 2 \cdot p(1 - p) + 3 \cdot p^2(1 - p) + 4 \cdot p^3(1 - p) + \ldots$$

$$= (1 - p)(1 + 2p + 3p^2 + 4p^3 + \ldots)$$

We have already shown that $1 + 2p + 3p^2 + 4p^3 + \ldots = \frac{1}{(1-p)^2}$.

Thus, $E[N] = \frac{1}{1-p}$; since $p < 1/2$ we get $E[N] < 2$. 
Designing a Perfect Hash table (continued)

- How many trials $N$ do we expect to have to make before we hit a collision free hash table of size $< 2n^2$?

- Let the probability of failure be denoted by $p$; then $p < 1/2$, and the probability of a success is $1 - p$.

- Thus,

$$E[N] = (1 - p)(1 + 2p + 3p^2 + 4p^3 + \ldots)$$

- We have already shown that $1 + 2p + 3p^2 + 4p^3 + \ldots = \frac{1}{(1-p)^2}$.

- Thus, $E[N] = \frac{1}{1-p}$; since $p < 1/2$ we get $E[N] < 2$. 
How many trials \( N \) do we expect to have to make before we hit a collision free hash table of size \( < 2n^2 \)?

Let the probability of failure be denoted by \( p \); then \( p < 1/2 \), and the probability of a success is \( 1 - p \).

Thus,

\[
E[N] = 1 \cdot (1 - p) + 2 \cdot p(1 - p) + 3 \cdot p^2(1 - p) + 4 \cdot p^3(1 - p) + \ldots \\
= (1 - p)(1 + 2p + 3p^2 + 4p^3 + \ldots)
\]

We have already shown that \( 1 + 2p + 3p^2 + 4p^3 + \ldots = \frac{1}{(1-p)^2} \).

Thus, \( E[N] = \frac{1}{1-p} \); since \( p < 1/2 \) we get \( E[N] < 2 \).
Thus, on average, less than two trials will be enough to obtain a collision free table of size $< 2n^2$.

Recall that we aim to produce a collision free hash table of size linear in $n$ for storing $n$ keys.

Second step: Choose $M$ to be the smallest prime larger than $n$.

Thus $n \leq M < 2n$; we now produce a hash table of size $M$ again by choosing randomly from a universal family of hash functions.
Thus, on average, less than two trials will be enough to obtain a collision free table of size $< 2n^2$.

Recall that we aim to produce a collision free hash table of size linear in $n$ for storing $n$ keys.

**Second step:** Choose $M$ to be the smallest prime larger than $n$.

Thus $n \leq M < 2n$; we now produce a hash table of size $M$ again by choosing randomly from a universal family of hash functions.
Thus, on average, less than two trials will be enough to obtain a collision free table of size $< 2n^2$.

Recall that we aim to produce a collision free hash table of size linear in $n$ for storing $n$ keys.

**Second step:** Choose $M$ to be the smallest prime larger than $n$.

Thus $n \leq M < 2n$; we now produce a hash table of size $M$ again by choosing randomly from a universal family of hash functions.
Assume that a slot $i$ of this table has $n_i$ many elements.

We will hash these $n_i$ many elements into a secondary hash table of size $m_i < 2n_i^2$.

We have to guarantee that the sum total of sizes of all secondary hash tables, i.e., $\sum_{i=1}^{M} m_i$ is linear in $n$.

Note that

$$\binom{n_i}{2} = \frac{n_i(n_i - 1)}{2} = \frac{n_i^2}{2} - \frac{n_i}{2}$$

Thus, since $n_i$ is the number of elements in the $i^{th}$ slot, we get

$$\sum_{i=1}^{M} n_i^2 = 2 \sum_{i=1}^{M} \binom{n_i}{2} + \sum_{i=1}^{M} n_i = 2 \sum_{i=1}^{M} \binom{n_i}{2} + n$$

(2)
Designing a Perfect Hash table (continued)

- Assume that a slot $i$ of this table has $n_i$ many elements.

- We will hash these $n_i$ many elements into a secondary hash table of size $m_i < 2n_i^2$.

- We have to guarantee that the sum total of sizes of all secondary hash tables, i.e., $\sum_{i=1}^{M} m_i$ is linear in $n$.

- Note that

$$\binom{n_i}{2} = \frac{n_i(n_i - 1)}{2} = \frac{n_i^2}{2} - \frac{n_i}{2}$$

- Thus, since $n_i$ is the number of elements in the $i^{th}$ slot, we get

$$\sum_{i=1}^{M} n_i^2 = 2 \sum_{i=1}^{M} \binom{n_i}{2} + \sum_{i=1}^{M} n_i = 2 \sum_{i=1}^{M} \binom{n_i}{2} + n$$ (2)
Designing a Perfect Hash table (continued)

- Assume that a slot $i$ of this table has $n_i$ many elements.

- We will hash these $n_i$ many elements into a secondary hash table of size $m_i < 2n_i^2$.

- We have to guarantee that the sum total of sizes of all secondary hash tables, i.e., $\sum_{i=1}^{M} m_i$ is linear in $n$.

- Note that

$$\binom{n_i}{2} = \frac{n_i(n_i - 1)}{2} = \frac{n_i^2}{2} - \frac{n_i}{2}$$

- Thus, since $n_i$ is the number of elements in the $i^{th}$ slot, we get

$$\sum_{i=1}^{M} n_i^2 = 2 \sum_{i=1}^{M} \binom{n_i}{2} + \sum_{i=1}^{M} n_i = 2 \sum_{i=1}^{M} \binom{n_i}{2} + n$$

(2)
Assume that a slot $i$ of this table has $n_i$ many elements.

We will hash these $n_i$ many elements into a secondary hash table of size $m_i < 2n_i^2$.

We have to guarantee that the sum total of sizes of all secondary hash tables, i.e., $\sum_{i=1}^{M} m_i$ is linear in $n$.

Note that

$$\binom{n_i}{2} = \frac{n_i(n_i - 1)}{2} = \frac{n_i^2}{2} - \frac{n_i}{2}$$

Thus, since $n_i$ is the number of elements in the $i^{th}$ slot, we get

$$\sum_{i=1}^{M} n_i^2 = 2 \sum_{i=1}^{M} \binom{n_i}{2} + \sum_{i=1}^{M} n_i = 2 \sum_{i=1}^{M} \binom{n_i}{2} + n$$

(2)
Assume that a slot $i$ of this table has $n_i$ many elements.

We will hash these $n_i$ many elements into a secondary hash table of size $m_i < 2n_i^2$.

We have to guarantee that the sum total of sizes of all secondary hash tables, i.e., $\sum_{i=1}^{M} m_i$ is linear in $n$.

Note that

\[
\binom{n_i}{2} = \frac{n_i(n_i - 1)}{2} = \frac{n_i^2}{2} - \frac{n_i}{2}
\]

Thus, since $n_i$ is the number of elements in the $i^{th}$ slot, we get

\[
\sum_{i=1}^{M} n_i^2 = 2 \sum_{i=1}^{M} \binom{n_i}{2} + \sum_{i=1}^{M} n_i = 2 \sum_{i=1}^{M} \binom{n_i}{2} + n \tag{2}
\]
Total size of tables: \( \sum_{i=1}^{M} n_i^2 = 2 \sum_{i=1}^{M} \binom{n_i}{2} + n \)

- However, \( \binom{n_i}{2} \) is the total number of collisions in slot \( i \);

- thus, \( \sum_{i=1}^{M} \binom{n_i}{2} \) is the total number of collisions in the hash table.

- Since there are \( \binom{n}{2} \) pairs of keys and for each pair of keys the probability of a collision with universal hashing is \( 1/M \), we obtain that the expected total number of collisions is \( \binom{n}{2} \frac{1}{M} \).

Thus,

\[
E \left[ \sum_{i=1}^{M} \binom{n_i}{2} \right] = \binom{n}{2} \frac{1}{M} = \frac{n(n-1)}{2M}
\]
Total size of tables: \[ \sum_{i=1}^{M} n_i^2 = 2 \sum_{i=1}^{M} \binom{n_i}{2} + n \]

- However, \( \binom{n_i}{2} \) is the total number of collisions in slot \( i \);

- thus, \( \sum_{i=1}^{M} \binom{n_i}{2} \) is the total number of collisions in the hash table.

- Since there are \( \binom{n}{2} \) pairs of keys and for each pair of keys the probability of a collision with universal hashing is \( 1/M \), we obtain that the expected total number of collisions is \( \binom{n}{2} \frac{1}{M} \).

Thus,

\[
E \left[ \sum_{i=1}^{M} \binom{n_i}{2} \right] = \binom{n}{2} \frac{1}{M} = \frac{n(n - 1)}{2M} \tag{3}
\]
Total size of tables: \[ \sum_{i=1}^{M} n_i^2 = 2 \sum_{i=1}^{M} \binom{n_i}{2} + n \]

- However, \( \binom{n_i}{2} \) is the total number of collisions in slot \( i \);

- thus, \( \sum_{i=1}^{M} \binom{n_i}{2} \) is the total number of collisions in the hash table.

- Since there are \( \binom{n}{2} \) pairs of keys and for each pair of keys the probability of a collision with universal hashing is \( 1/M \), we obtain that the expected total number of collisions is \( \binom{n}{2} \frac{1}{M} \).

Thus,

\[
E \left[ \sum_{i=1}^{M} \binom{n_i}{2} \right] = \binom{n}{2} \frac{1}{M} = \frac{n(n-1)}{2M} \tag{3}
\]
Total size of tables: \[ \sum_{i=1}^{M} n_i^2 = 2 \sum_{i=1}^{M} \binom{n_i}{2} + n \]

- However, \( \binom{n_i}{2} \) is the total number of collisions in slot \( i \);

- thus, \( \sum_{i=1}^{M} \binom{n_i}{2} \) is the total number of collisions in the hash table.

- Since there are \( \binom{n}{2} \) pairs of keys and for each pair of keys the probability of a collision with universal hashing is \( 1/M \), we obtain that the expected total number of collisions is \( \binom{n}{2} \frac{1}{M} \).

Thus,

\[
E \left[ \sum_{i=1}^{M} \binom{n_i}{2} \right] = \binom{n}{2} \frac{1}{M} = \frac{n(n-1)}{2M}
\]  

(3)
Total size of tables: \[ \sum_{i=1}^{M} n_i^2 = 2 \sum_{i=1}^{M} \binom{n_i}{2} + n \]

So we have:

\[ E \left[ \sum_{i=1}^{M} n_i^2 \right] = 2E \left[ \sum_{i=1}^{M} \binom{n_i}{2} \right] + n; \]

\[ E \left[ \sum_{i=1}^{M} \binom{n_i}{2} \right] = \frac{n(n-1)}{2M}; \]

Thus,

\[ E \left[ \sum_{i=1}^{M} n_i^2 \right] = \frac{n(n-1)}{M} + n \]

But \( M \) was chosen so that \( M \geq n \); thus

\[ E \left[ \sum_{i=1}^{M} n_i^2 \right] \leq \frac{n(n-1)}{n} + n = 2n - 1 < 2n \]
Total size of tables: \[ \sum_{i=1}^{M} n_i^2 = 2 \sum_{i=1}^{M} \left( \frac{n_i}{2} \right) + n \]

So we have:

\[
E \left[ \sum_{i=1}^{M} n_i^2 \right] = 2E \left[ \sum_{i=1}^{M} \left( \frac{n_i}{2} \right) \right] + n;
\]

\[
E \left[ \sum_{i=1}^{M} \left( \frac{n_i}{2} \right) \right] = \frac{n(n-1)}{2M};
\]

Thus,

\[
E \left[ \sum_{i=1}^{M} n_i^2 \right] = \frac{n(n-1)}{M} + n
\]

But \( M \) was chosen so that \( M \geq n \); thus

\[
E \left[ \sum_{i=1}^{M} n_i^2 \right] \leq \frac{n(n-1)}{n} + n = 2n - 1 < 2n
\]
Applying the Markov Inequality once again we obtain

\[ P\left\{ \sum_{i=1}^{M} n_i^2 > 4n \right\} \leq \frac{E \left[ \sum_{i=1}^{M} n_i^2 \right]}{4n} < \frac{2n}{4n} = \frac{1}{2} \quad (4) \]

Thus, after a few attempts we will produce a hash table of size \( M < 2n \) for which \( \sum_{i=1}^{M} n_i^2 < 4n \).

If we choose primes \( m_i < 2n_i^2 \) then \( \sum_{i=1}^{M} m_i < 8n \).

In this way the size of the primary hash table plus the sum of sizes of all secondary hash tables satisfies

\[ M + \sum_{i=1}^{M} m_i < 2n + 8n = 10n. \]
Designing a Perfect Hash table (continued)

Applying the Markov Inequality once again we obtain

$$P\left\{ \sum_{i=1}^{M} n_i^2 > 4n \right\} \leq \frac{E \left[ \sum_{i=1}^{M} n_i^2 \right]}{4n} < \frac{2n}{4n} = \frac{1}{2} \quad (4)$$

Thus, after a few attempts we will produce a hash table of size $M < 2n$ for which $\sum_{i=1}^{M} n_i^2 < 4n$.

If we choose primes $m_i < 2n^2$ then $\sum_{i=1}^{M} m_i < 8n$.

In this way the size of the primary hash table plus the sum of sizes of all secondary hash tables satisfies

$$M + \sum_{i=1}^{M} m_i < 2n + 8n = 10n.$$
Applying the Markov Inequality once again we obtain

\[
P\left\{ \sum_{i=1}^{M} n_i^2 > 4n \right\} \leq \frac{E\left[ \sum_{i=1}^{M} n_i^2 \right]}{4n} < \frac{2n}{4n} = \frac{1}{2}
\]  

Thus, after a few attempts we will produce a hash table of size \( M < 2n \) for which \( \sum_{i=1}^{M} n_i^2 < 4n \).

If we choose primes \( m_i < 2n_i^2 \) then \( \sum_{i=1}^{M} m_i < 8n \).

In this way the size of the primary hash table plus the sum of sizes of all secondary hash tables satisfies

\[
M + \sum_{i=1}^{M} m_i < 2n + 8n = 10n.
\]
Applying the Markov Inequality once again we obtain

\[
P \left\{ \sum_{i=1}^{M} n_i^2 > 4n \right\} \leq \frac{E \left[ \sum_{i=1}^{M} n_i^2 \right]}{4n} < \frac{2n}{4n} = \frac{1}{2} \quad (4)
\]

Thus, after a few attempts we will produce a hash table of size \( M < 2n \) for which \( \sum_{i=1}^{M} n_i^2 < 4n \).

If we choose primes \( m_i < 2n_i^2 \) then \( \sum_{i=1}^{M} m_i < 8n \).

In this way the size of the primary hash table plus the sum of sizes of all secondary hash tables satisfies

\[
M + \sum_{i=1}^{M} m_i < 2n + 8n = 10n.
\]
Designing a Perfect Hash table (continued)

We now describe the entire randomised construction.

1. Choose a prime number $M$ such that $n \leq M < 2n$.
2. Pick randomly a hash function with hash table size $M$ from a universal hash functions family.
3. Use it to hash all $n$ elements into such a table.
4. Check if the numbers of elements $n_i$ in slots $i = 1, 2 \ldots M$ satisfy $\sum_{i=1}^{M} n_i^2 < 4n$.
5. If not, pick randomly another hash function and try again, repeating until the above condition is satisfied.
6. As we have shown (see equation 4), you will succeed fast (on average after only two trials).
7. For each slot $i$ of the table containing $n_i$ elements use the first described randomised construction to obtain hash functions $h_i$ which produce no collisions at all and such that the size of the corresponding hash tables are prime numbers $m_i$ satisfying $m_i < 2n_i^2$. 
Designing a Perfect Hash table (continued)

We now describe the entire randomised construction.

1. Choose a prime number $M$ such that $n \leq M < 2n$.
2. Pick randomly a hash function with hash table size $M$ from a universal hash functions family.
3. Use it to hash all $n$ elements into such a table.
4. Check if the numbers of elements $n_i$ in slots $i = 1, 2 \ldots M$ satisfy $\sum_{i=1}^{M} n_i^2 < 4n$.
5. If not, pick randomly another hash function and try again, repeating until the above condition is satisfied.
6. As we have shown (see equation 4), you will succeed fast (on average after only two trials).
7. For each slot $i$ of the table containing $n_i$ elements use the first described randomised construction to obtain hash functions $h_i$ which produce no collisions at all and such that the size of the corresponding hash tables are prime numbers $m_i$ satisfying $m_i < 2n_i^2$. 
Designing a Perfect Hash table (continued)

We now describe the entire randomised construction.

1. Choose a prime number $M$ such that $n \leq M < 2n$.
2. Pick randomly a hash function with hash table size $M$ from a universal hash functions family.
3. Use it to hash all $n$ elements into such a table.
4. Check if the numbers of elements $n_i$ in slots $i = 1, 2 \ldots M$ satisfy $\sum_{i=1}^{M} n_i^2 < 4n$.
5. If not, pick randomly another hash function and try again, repeating until the above condition is satisfied.
6. As we have shown (see equation 4), you will succeed fast (on average after only two trials).
7. For each slot $i$ of the table containing $n_i$ elements use the first described randomised construction to obtain hash functions $h_i$ which produce no collisions at all and such that the size of the corresponding hash tables are prime numbers $m_i$ satisfying $m_i < 2n_i^2$. 

Shuiqiao Yang
COMP4121
Introduction to Randomised Algorithms
We now describe the entire randomised construction.

1. Choose a prime number $M$ such that $n \leq M < 2n$.
2. Pick randomly a hash function with hash table size $M$ from a universal hash functions family.
3. Use it to hash all $n$ elements into such a table.
4. Check if the numbers of elements $n_i$ in slots $i = 1, 2 \ldots M$ satisfy $\sum_{i=1}^{M} n_i^2 < 4n$.
5. If not, pick randomly another hash function and try again, repeating until the above condition is satisfied.
6. As we have shown (see equation 4), you will succeed fast (on average after only two trials).
7. For each slot $i$ of the table containing $n_i$ elements use the first described randomised construction to obtain hash functions $h_i$ which produce no collisions at all and such that the size of the corresponding hash tables are prime numbers $m_i$ satisfying $m_i < 2n_i^2$. 
We now describe the entire randomised construction.

1. Choose a prime number $M$ such that $n \leq M < 2n$.
2. Pick randomly a hash function with hash table size $M$ from a universal hash functions family.
3. Use it to hash all $n$ elements into such a table.
4. Check if the numbers of elements $n_i$ in slots $i = 1, 2 \ldots M$ satisfy $\sum_{i=1}^{M} n_i^2 < 4n$.
5. If not, pick randomly another hash function and try again, repeating until the above condition is satisfied.
6. As we have shown (see equation 4), you will succeed fast (on average after only two trials).
7. For each slot $i$ of the table containing $n_i$ elements use the first described randomised construction to obtain hash functions $h_i$ which produce no collisions at all and such that the size of the corresponding hash tables are prime numbers $m_i$ satisfying $m_i < 2n_i^2$. 
We now describe the entire randomised construction.

1. Choose a prime number $M$ such that $n \leq M < 2n$.
2. Pick randomly a hash function with hash table size $M$ from a universal hash functions family.
3. Use it to hash all $n$ elements into such a table.
4. Check if the numbers of elements $n_i$ in slots $i = 1, 2 \ldots M$ satisfy
   $\sum_{i=1}^{M} n_i^2 < 4n$.
5. If not, pick randomly another hash function and try again, repeating until the above condition is satisfied.
6. As we have shown (see equation 4), you will succeed fast (on average after only two trials).
7. For each slot $i$ of the table containing $n_i$ elements use the first described randomised construction to obtain hash functions $h_i$ which produce no collisions at all and such that the size of the corresponding hash tables are prime numbers $m_i$ satisfying
   $m_i < 2n_i^2$. 

Shuiqiao Yang  
COMP4121  
Introduction to Randomised Algorithms
We now describe the entire randomised construction.

1. Choose a prime number $M$ such that $n \leq M < 2n$.
2. Pick randomly a hash function with hash table size $M$ from a universal hash functions family.
3. Use it to hash all $n$ elements into such a table.
4. Check if the numbers of elements $n_i$ in slots $i = 1, 2 \ldots M$ satisfy $\sum_{i=1}^{M} n_i^2 < 4n$.
5. If not, pick randomly another hash function and try again, repeating until the above condition is satisfied.
6. As we have shown (see equation 4), you will succeed fast (on average after only two trials).
7. For each slot $i$ of the table containing $n_i$ elements use the first described randomised construction to obtain hash functions $h_i$ which produce no collisions at all and such that the size of the corresponding hash tables are prime numbers $m_i$ satisfying $m_i < 2n_i^2$. 
We now describe the entire randomised construction.

1. Choose a prime number $M$ such that $n \leq M < 2n$.
2. Pick randomly a hash function with hash table size $M$ from a universal hash functions family.
3. Use it to hash all $n$ elements into such a table.
4. Check if the numbers of elements $n_i$ in slots $i = 1, 2 \ldots M$ satisfy $\sum_{i=1}^{M} n_i^2 < 4n$.
5. If not, pick randomly another hash function and try again, repeating until the above condition is satisfied.
6. As we have shown (see equation 4), you will succeed fast (on average after only two trials).
7. For each slot $i$ of the table containing $n_i$ elements use the first described randomised construction to obtain hash functions $h_i$ which produce no collisions at all and such that the size of the corresponding hash tables are prime numbers $m_i$ satisfying $m_i < 2n_i^2$. 

Shuiqiao Yang

COMP4121 Introduction to Randomised Algorithms
Note that our procedure eventually produces a **fully deterministic** hash function.

It is only that our search for such a function was a randomised procedure by "trial and error".

How does the resulting hash function operate?

1. For a given key \( x \) compute \( h(x) = i_x \) which is an index \( 1 \leq i_x \leq M \) for the primary hash table \( T \) of size \( M < 2^n \);
2. in the slot \( i_x \) of \( T \) find the secondary hash function \( h_{i_x} \) and compute \( h_{i_x}(x) = j_x \) which is an index in the secondary table \( t_{i_x} \);
3. \( x \) (with the associated record \( R_x \) for \( x \)) is stored in slot \( j_x \) of the secondary table \( t_{i_x} \).
Note that our procedure eventually produces a fully deterministic hash function.

It is only that our search for such a function was a randomised procedure by “trial and error”.

How does the resulting hash function operate?

1. For a given key \( x \) compute \( h(x) = i_x \) which is an index \( 1 \leq i_x \leq M \) for the primary hash table \( T \) of size \( M < 2n \);
2. in the slot \( i_x \) of \( T \) find the secondary hash function \( h_{i_x} \) and compute \( h_{i_x}(x) = j_x \) which is an index in the secondary table \( t_{i_x} \);
3. \( x \) (with the associated record \( R_x \) for \( x \)) is stored in slot \( j_x \) of the secondary table \( t_{i_x} \).
Note that our procedure eventually produces a **fully deterministic** hash function.

It is only that our search for such a function was a randomised procedure by “trial and error”.

How does the resulting hash function operate?

1. For a given key $x$ compute $h(x) = i_x$ which is an index $1 \leq i_x \leq M$ for the primary hash table $T$ of size $M < 2^n$;
2. in the slot $i_x$ of $T$ find the secondary hash function $h_{i_x}$ and compute $h_{i_x}(x) = j_x$ which is an index in the secondary table $t_{i_x}$;
3. $x$ (with the associated record $R_x$ for $x$) is stored in slot $j_x$ of the secondary table $t_{i_x}$. 
Note that our procedure eventually produces a **fully deterministic** hash function.

It is only that our search for such a function was a randomised procedure by “trial and error”.

How does the resulting hash function operate?

1. For a given key $x$ compute $h(x) = i_x$ which is an index $1 \leq i_x \leq M$ for the primary hash table $T$ of size $M < 2^n$;
2. in the slot $i_x$ of $T$ find the secondary hash function $h_{i_x}$ and compute $h_{i_x}(x) = j_x$ which is an index in the secondary table $t_{i_x}$;
3. $x$ (with the associated record $R_x$ for $x$) is stored in slot $j_x$ of the secondary table $t_{i_x}$. 
Note that our procedure eventually produces a **fully deterministic** hash function.

It is only that our search for such a function was a randomised procedure by “trial and error”.

How does the resulting hash function operate?

1. For a given key $x$ compute $h(x) = i_x$ which is an index $1 \leq i_x \leq M$ for the primary hash table $T$ of size $M < 2^n$;

2. in the slot $i_x$ of $T$ find the secondary hash function $h_{i_x}$ and compute $h_{i_x}(x) = j_x$ which is an index in the secondary table $t_{i_x}$;

3. $x$ (with the associated record $R_x$ for $x$) is stored in slot $j_x$ of the secondary table $t_{i_x}$.
Designing a Perfect Hash table (continued)

Note that our procedure eventually produces a **fully deterministic** hash function.

It is only that our search for such a function was a randomised procedure by “trial and error”.

How does the resulting hash function operate?

1. For a given key \( x \) compute \( h(x) = i_x \) which is an index \( 1 \leq i_x \leq M \) for the primary hash table \( T \) of size \( M < 2n \);
2. in the slot \( i_x \) of \( T \) find the secondary hash function \( h_{i_x} \) and compute \( h_{i_x}(x) = j_x \) which is an index in the secondary table \( t_{i_x} \);
3. \( x \) (with the associated record \( R_x \) for \( x \)) is stored in slot \( j_x \) of the secondary table \( t_{i_x} \).
Designing a Perfect Hash table (continued)

$h(x) = i_x$

$x$

$h(x) = i_x$

$i_x \quad h_{ix}$

$j_x = h_{ix}(x)$

Secondary hash tables

$1 \quad 2 \quad \vdots \quad i_x \quad h_{ix} \quad j_x \quad (x,R_x)$

$\vdots$

$t_1 \quad t_2 \quad \ldots \quad t_{ix} \quad \ldots \quad t_M$

primary hash table $T$

$1 \quad 2 \quad \vdots$