Universal Hash Functions and Perfect Hashing

Figure 11.6 Using perfect hashing to store the set \(K = \{10, 22, 37, 40, 52, 60, 70, 72, 75\}\). The outer hash function is \(h(k) = (a \cdot k + b) \mod p \mod m\), where \(a = 3\), \(b = 42\), \(p = 101\), and \(m = 9\). For example, \(h(75) = 2\), and key 75 hashes to slot 2 of table \(T\). A secondary hash table \(S_j\) stores all keys hashing to slot \(j\). The size of hash table \(S_j\) is \(m_j \geq n_j\), and the associated hash function is \(h_j(k) = (a_j \cdot k + b_j) \mod p \mod m_j\). Since \(h_2(75) = 7\), key 75 is stored in slot 7 of secondary hash table \(S_2\). No collisions occur in any of the secondary hash tables, and so searching takes constant time in the worst case.

To create a perfect hashing scheme, we use two levels of hashing, with universal hashing at each level. The first level is essentially the same as for hashing with chaining: we hash the \(n\) keys into \(m\) slots using a hash function carefully selected from a family of universal hash functions. Instead of making a linked list of the keys hashing to slot \(j\), however, we use a small secondary hash table \(S_j\) with an associated hash function \(h_j\). By choosing the hash functions \(h_j\) carefully, we can guarantee that there are no collisions at the secondary level. In order to guarantee that there are no collisions at the secondary level, however, we will need to let the size \(m_j\) of hash table \(S_j\) be the square of the number \(n_j\) of keys hashing to slot \(j\). Although you might think that the quadratic dependence of \(m_j\) on \(n_j\) may seem likely to cause the overall storage requirement to be excessive, we shall show that by choosing the first-level hash function well, we can limit the expected total amount of space used to \(O(n)\).

We use hash functions chosen from the universal classes of hash functions of Section 11.3.3. The first-level hash function comes from the class \(H_{pm}\), whereas in Section 11.3.3, \(p\) is a prime number greater than any key value. Those keys...
If a malicious adversary chooses the keys to be hashed by some fixed hash function, he can choose $n$ keys $x_i$ such that they all hash to the same value

$$H(x_i) = h \text{ for } i = 1, 2, \ldots, n$$

This implies that the hash table will have $\Theta(n)$ retrieval time.

Any fixed hash function would have this worst-case behavior.

The only effective way to improve the situation is to choose the hash function *randomly* in a way that is independent of the keys.

This approach is called *universal hashing*.

It can yield provably good performance in the average, not matter which keys the adversary chooses.
Universal Hashing

- In universal hashing, we select the hash function at random from a carefully designed class of hash functions.
- Randomization guarantees that no single input will evoke worst-case behavior.
- This selection is done at the beginning of each execution.
- Therefore, the algorithm can behave differently on each execution, even for the same input.
- This will guarantee good average-case performance.
- Of course, poor performance will occur when the selected hash function hashes the keys poorly.
- However, the probability of this situation is small, and is the same for any set of keys of the same size.
Let $\mathcal{H}$ be a finite collection of hash functions that map a given universe $U$ of keys into the range $\{0, 1, 2, \ldots, m - 1\}$.

The set $\mathcal{H}$ is said to be universal if for each pair of keys $x, y \in U$, the number of hash functions $h \in \mathcal{H}$ for which $h(x) = h(y)$ is at most $|\mathcal{H}|/m$.

In other words, with a hash function randomly chosen from $\mathcal{H}$, the chance of collision between $x$ and $y$ is $1/m$.

This is the same chance of collision if $h(x)$ and $h(y)$ were randomly and independently chosen from the set $\{0, 1, 2, \ldots, m - 1\}$.
Average Case Behavior

- Suppose a hash function \( h \) is randomly chosen from a universal collection of hash functions.
- It is used to hash \( n \) keys into a table \( T \) of size \( m \), with chaining as the collision resolution method.
- Let \( \alpha \) be the load factor, defined as \( \alpha = \frac{n}{m} \).
- If \( x \) is not in the table, the expected length of the list that the key \( x \) hashes into is at most \( \alpha \).
- If \( x \) is in the table, the expected length of the list that contains the key \( x \) is at most \( 1 + \alpha \).
Average Case Behavior

Consider a pair of keys $x$ and $y$; due to the definition of the universal hashing, the probability that they collide is

$$P[h(x) = h(y)] \leq \frac{1}{m}$$

Let the random variable $R_{xy}$ take the value of 1 when $h(x) = h(y)$ and 0 otherwise; the expected value of $R_{xy}$ is

$$E[R_{xy}] = \frac{1}{m}$$

Let the random variable $S_x$ be the number of keys other than $x$ that hash to the same slot as $x$, given as

$$S_x = \sum_{y \in T, y \neq x} R_{xy}$$
Average Case Behavior

Therefore, we have

\[ E[S_x] = E\left[ \sum_{y \in T \atop y \neq x} R_{xy} \right] = \sum_{y \in T \atop y \neq x} E[R_{xy}] \leq \sum_{y \in T \atop y \neq x} \frac{1}{m} \]

If \( x \notin T \), then the list length is equal to \( S_x \) and

\[ |\{ y : y \in T \text{ and } y \neq x \}| = n \]

and thus the expected list length \( E[S_x] \leq n/m = \alpha \)

If \( x \in T \), then because \( x \) appears in the list \( T[h(x)] \) and the count does not include \( x \), we have the list length as \( S_x + 1 \) and

\[ |\{ y : y \in T \text{ and } y \neq x \}| = n - 1 \]

and thus the expected list length

\[ E[S_x] + 1 \leq (n - 1)/m + 1 = 1 + \alpha - 1/m < 1 + \alpha \]
Average Case Behavior

**Theorem**

*Using universal hashing and collision resolution by chaining in an initially empty table with \( m \) slots, it takes \( \Theta(n) \) time to handle any sequence of \( n \) Insert, Find, and Delete operations containing \( O(m) \) Insert operations.*

- The number of Insert operations is \( O(m) \), thus we have \( n = O(m) \) which implies \( \alpha = O(1) \)
- The Insert and Delete operations take constant time, and the expected time for Find operation is \( O(1) \) since the expected length of the list is at most \( \alpha \)
- Therefore, the expected time for the entire sequence of \( n \) operations is \( O(n) \) since each operation takes \( \Omega(1) \), the bound \( \Theta(n) \) is obtained
We will give 3 constructions and show them that they are universal.

- The first construction is based on linear congruential arithmetic with two distinct moduli: $p$ and $m$, where $p$ is a prime.
- The second construction uses a random 0-1 matrix and mod 2 arithmetic.
- The third method is based the dot-product modulo $m$. 
Construction of $\mathcal{H}_{p,m}$

- Select a prime $p$ that is large enough so that every possible key is in the range 0 to $p - 1$
- Let $\mathbb{Z}_p = \{0, 1, 2, \ldots, p - 1\}$ and $\mathbb{Z}_p^* = \{1, 2, \ldots, p - 1\}$
- The size of the universe of the keys is $p$ which is larger than the hash table size $m$, i.e., $p > m$
- Consider the integer $a \in \mathbb{Z}_p^*$ and $b \in \mathbb{Z}_p$
- Define the hash function family as

$$h_{a,b}(x) = (a \cdot x + b \mod p) \mod m$$

- The class of hash functions is defined as

$$\mathcal{H}_{p,m} = \{ h_{a,b} \mid a \in \mathbb{Z}_p^* \text{ and } b \in \mathbb{Z}_p \}$$
Properties of $\mathcal{H}_{p,m}$

- An Example: $p = 17$ and $m = 6$, we have $h_{3,4}(8) = 5$ since

  $h_{3,4}(8) = ((3 \cdot 8 + 4) \mod 17) \mod 6$
  $= (28 \mod 17) \mod 6$
  $= 11 \mod 6$
  $= 5$

- Each hash function $h_{a,b}$ maps $\mathbb{Z}_p$ to $\mathbb{Z}_m$: the keys are in the range 0 to $p - 1$, while the hash values are from 0 to $m - 1$

- This family has the nice property that the table size $m$ is arbitrary, not necessarily a prime

- There are $p - 1$ choices of $a$ and $p$ choices of $b$, and thus, there are $p(p - 1)$ hash functions
The class $\mathcal{H}_{p,m}$ of hash functions is universal.

- Consider two distinct keys $x$ and $y$ from $\mathbb{Z}_p$, so that $x \neq y$
- For a given hash function $h_{a,b}$, first compute
  
  $$
  r = (a \cdot x + b) \mod p \\
  s = (a \cdot y + b) \mod p
  $$

- $r - s = a(x - y)$ is nonzero since $x \neq y$ and $a \neq 0$, and $p$ is prime
- Therefore, if $x \neq y$, we will always have $r \neq s$
- There will not be collision on the “mod $p$ level”

http://koclab.org

Çetin Kaya Koç
Moreover, each possible \( p(p - 1) \) pair of \((a, b)\) with \( a \neq 0\) yields a different pair \((r, s)\) since

\[
a = (r - s)(x - y)^{-1} \mod p
\]
\[
b = (r - ax) \mod p
\]

There are \( p(p - 1) \) possible pairs \((r, s)\) with \( r \neq s\), and thus, there is a one-to-one correspondence between pairs \((a, b)\) with \( a \neq 0 \) and pairs \((r, s)\) with \( r \neq s\).
Thus, for any given pairs of inputs $x$ and $y$, if we pick $(a, b)$ uniformly at random from $\mathbb{Z}_p^* \times \mathbb{Z}_p$, the resulting pair is equally likely to be any pair of distinct values modulo $p$.

The probability that distinct keys $x$ and $y$ collide is equal to the probability $r = s \pmod{m}$ when $r$ and $s$ are randomly chosen as distinct values modulo $p$.

Furthermore, the probability that $s$ collides with $r$ when reduced modulo $m$ is at most $1/m$, and therefore

$$P[h_{a,b}(x) = h_{a,b}(y)] \leq 1/m$$

so that $\mathcal{H}_{p,m}$ is universal.
Construction of the Matrix Method

- Assume that the keys are $u$ bits long: $x = (x_{u-1} \cdots x_1 x_0)$
- The hash table size as a power of two, as $m = 2^b$, and the hash values $z = h(x)$ are $b$-bit integers: $z = (z_{b-1} \cdots z_1 z_0)$
- The hash function $h$ is computed using a 0-1 random matrix of dimension $b \times u$, denoted as $A$
- The hash operation $h(x)$ takes the key $x$ expressed as a $u$-bit binary number and multiplies with the matrix $A$ to obtain the $b$-bit hash
- All computations are done in mod 2: the Galois field GF(2)
Properties of the Matrix Method

- An Example: Let $u = 4$ and $b = 3$, therefore, the keys are 4-bit long $x = (x_3x_2x_1x_0)$ and the hash values are 3-bit long $z = (z_2z_1z_0)$
- The random 0-1 matrix is of size $b \times u = 3 \times 4$
- Taking $A$ as below, the computation of $z = h(x)$ is performed using

$$
\begin{bmatrix}
    z_0 \\
    z_1 \\
    z_2
\end{bmatrix} =
\begin{bmatrix}
    1 & 0 & 0 & 0 \\
    0 & 1 & 1 & 1 \\
    1 & 1 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
    x_0 \\
    x_1 \\
    x_2 \\
    x_3
\end{bmatrix}
$$

- Let $x = (x_3x_2x_1x_0) = (0101)$, we obtain $(z_2z_1z_0) = (011)$ as

$$
\begin{bmatrix}
    z_0 \\
    z_1 \\
    z_2
\end{bmatrix} =
\begin{bmatrix}
    1 & 0 & 0 & 0 \\
    0 & 1 & 1 & 1 \\
    1 & 1 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
    1 \\
    0 \\
    1
\end{bmatrix} =
\begin{bmatrix}
    1 + 0 + 0 + 0 \\
    0 + 0 + 1 + 0 \\
    1 + 0 + 1 + 0
\end{bmatrix} =
\begin{bmatrix}
    1 \\
    1 \\
    0
\end{bmatrix}
$$
Proving Universality

Theorem

For $x \neq y$, $P[h(x) = h(y)] = 1/m = 2^{-b}$, therefore the class of matrix hash functions with a randomly selected 0-1 matrices is universal.

- Take an arbitrary $x$ and $y$
- They must differ in at least one bit position
- Assume that $x$ and $y$ differ in the $i$th bit, i.e., they are given as $(x_{u-1} \cdots x_i \cdots x_1 x_0)$ and $(y_{u-1} \cdots y_i \cdots y_1 y_0)$ such that $x_i \neq y_i$
- WLOG, assume $x_i = 0$ and $y_i = 1$
- Now choose the entire $A$ matrix except its $i$th column
Proving Universality

- Since this is the column that multiplies the $i$th bit $x$ or $y$, the hash values $h(x)$ and $h(y)$ are the same, except the contribution of the $i$th column of $A$ is not included yet.
- The length of $i$th column is $b$, and there are $2^b$ different choices for this column.
- Every time we change a bit in this column, we flip the corresponding bit in $h(y)$ since $y_i = 1$.
- There are exactly one in $2^b$ chance that $h(x) = h(y)$.
- Therefore, the hash function is universal.
Consider \( x = (x_3x_2x_1x_0) = (0101) \) and \( y = (y_3y_2y_1y_0) = (1101) \) so that \( x \) and \( y \) differ only in the 3rd bit \( x_3 \neq y_3 \)

\[
\begin{bmatrix}
z_0 \\
z_1 \\
z_2
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0
\end{bmatrix} \begin{bmatrix}
1 \\
0 \\
1 \\
0
\end{bmatrix} \quad \text{and} \quad \begin{bmatrix}
z'_0 \\
z'_1 \\
z'_2
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0
\end{bmatrix} \begin{bmatrix}
1 \\
0 \\
1 \\
1
\end{bmatrix}
\]

\[
\begin{bmatrix}
z_0 \\
z_1 \\
z_2
\end{bmatrix} = \begin{bmatrix}
(1 \cdot 1 + 0 \cdot 0 + 0 \cdot 1) + 0 \cdot 0 \\
(0 \cdot 1 + 1 \cdot 0 + 1 \cdot 1) + 1 \cdot 0 \\
(1 \cdot 1 + 1 \cdot 0 + 0 \cdot 1) + 0 \cdot 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
z'_0 \\
z'_1 \\
z'_2
\end{bmatrix} = \begin{bmatrix}
(1 \cdot 1 + 0 \cdot 0 + 0 \cdot 1) + 0 \cdot 1 \\
(0 \cdot 1 + 1 \cdot 0 + 1 \cdot 1) + 1 \cdot 1 \\
(1 \cdot 1 + 1 \cdot 0 + 0 \cdot 1) + 0 \cdot 1
\end{bmatrix}
\]
The contribution of the first three columns of the $A$ matrix to the hash value is the same, and the difference occurs in the contribution of the last column

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} (1) + 0 \cdot 0 \\ (1) + 1 \cdot 0 \\ (1) + 0 \cdot 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} z'_0 \\ z'_1 \\ z'_2 \end{bmatrix} = \begin{bmatrix} (1) + 0 \cdot 1 \\ (1) + 1 \cdot 1 \\ (1) + 0 \cdot 1 \end{bmatrix}$$

As we use $A$ matrices each of which is different in the last column (there are 8 such columns), we obtain different $[z'_0, z'_1, z'_2]^T$ vectors

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} z'_0 \\ z'_1 \\ z'_2 \end{bmatrix} = \begin{bmatrix} (1) + 0 \cdot 1 \\ (1) + 1 \cdot 1 \\ (1) + 0 \cdot 1 \end{bmatrix}$$
Proving Universality

- Only in 0 case in which the last column is \([0, 0, 0]^T\), we will obtain \([z'_0, z'_1, z'_2]^T = [z_0, z_1, z_2]^T\), which is the case when the last column of \(A\) is selected as \([0, 0, 0]^T\)

\[
\begin{bmatrix}
(1) + 0 \cdot 1 \\
(1) + 0 \cdot 1 \\
(1) + 0 \cdot 1
\end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix};
\begin{bmatrix}
(1) + 0 \cdot 1 \\
(1) + 0 \cdot 1 \\
(1) + 1 \cdot 1
\end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix};
\begin{bmatrix}
(1) + 0 \cdot 1 \\
(1) + 1 \cdot 1 \\
(1) + 0 \cdot 1
\end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix};
\begin{bmatrix}
(1) + 0 \cdot 1 \\
(1) + 1 \cdot 1 \\
(1) + 1 \cdot 1
\end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\]

\[
\begin{bmatrix}
(1) + 1 \cdot 1 \\
(1) + 0 \cdot 1 \\
(1) + 0 \cdot 1
\end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix};
\begin{bmatrix}
(1) + 1 \cdot 1 \\
(1) + 0 \cdot 1 \\
(1) + 1 \cdot 1
\end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix};
\begin{bmatrix}
(1) + 1 \cdot 1 \\
(1) + 1 \cdot 1 \\
(1) + 0 \cdot 1
\end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix};
\begin{bmatrix}
(1) + 1 \cdot 1 \\
(1) + 1 \cdot 1 \\
(1) + 1 \cdot 1
\end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}
\]

- Therefore \(h(x) = h(y)\) only in 1 out of 8 cases
- There are exactly one in \(2^b\) chance that \(h(x) = h(y)\)
Construction of the Dot-Product Mod $m$ Method

- Let $m$ be prime
- Decompose the key $x$ into $r + 1$ digits each with the value in the set $\mathbb{Z}_m = \{0, 1, 2, \ldots, m - 1\}$
- We have $x = (x_rx_{r-1} \cdots x_1x_0)$ with $x_i \in \mathbb{Z}_m$
- Let $a = (a_r a_{r-1} \cdots a_1 a_0)$ be a random vector such that $a_i \in \mathbb{Z}_m$
- Define the hash function family as
  \[ h_a(x) = \sum_{i=0}^{r} a_i x_i \pmod{m} \]

- The size of $\mathcal{H}$ is $m^{r+1}$
Theorem

The set \( \mathcal{H} = \{ h_a \} \) is universal.

- Let \( x = (x_r \cdots x_1 x_0) \) and \( y = (y_r \cdots y_1 y_0) \) be two distinct keys.
- Thus, they differ in at least one digit position, WLOG position 0.
- For how many \( h_a \in \mathcal{H} \) do \( x \) and \( y \) collide?
- The equality \( h(x) = h(y) \) implies

\[
\sum_{i=0}^{r} a_i x_i = \sum_{i=0}^{r} a_i y_i \pmod{m}
\]
Equivalently we have

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

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

\[ a_0(x_0 - y_0) = -\sum_{i=1}^{r} a_i(x_i - y_i) \pmod{m} \]
Since $x_0 \neq y_0$ and $m$ is prime, the inverse $(x_0 - y_0)^{-1} \pmod{m}$ exists, which implies

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

Thus, for any choices of $a_1, a_2, \ldots, a_r$, exactly one choice of $a_0$ causes $x$ and $y$ collide.

How many $h_a$ functions cause $x$ and $y$ collide?
There are \( m \) choices for each of \( a_1, a_2, \ldots, a_r \) but once they are chosen, there is only one choice of \( a_0 \) that causes \( x \) and \( y \) collide.

Therefore, the number of hash functions that causes \( x \) and \( y \) collide is

\[
m^r \cdot 1 = m^r = \frac{m^{r+1}}{m} = \frac{|\mathcal{H}|}{m}
\]

that makes \( \mathcal{H} \) a universal hash function family.
A hashing technique is called **perfect hashing** if $O(1)$ memory accesses are required to perform a search in the **worst case**.

To create a perfect hashing, we use two levels of hashing, with universal hashing at each level.

![Diagram of perfect hashing](image-url)

### Figure 11.6
Using perfect hashing to store the set $K = \{10; 22; 37; 40; 52; 60; 70; 72; 75\}$.

The outer hash function is $h(x) = (ax + b) \mod p \mod m$, where $a = 3$, $b = 42$, $p = 101$, and $m = 9$. For example, $h(75) = 2$, and the key $75$ hashes to slot 2 of table $T$.

A secondary hash table $S_j$ stores all keys hashing to slot $j$. The size of hash table $S_j$ is $m_j = n_j^2$, and the associated hash function is $h_j(x) = (ax_j + b_j) \mod p \mod m_j$. Since $h_2(75) = 7$, key $75$ is stored in slot 7 of secondary hash table $S_2$.

No collisions occur in any of the secondary hash tables, and so searching takes constant time in the worst case.
Perfect Hashing

- The first level is the same as hashing with chaining: we hash $n$ keys into $m$ slots using a hash function $h$ from a family of universal hash functions.
- However, instead of making a linked list of keys hashing to slot $j$, we use a secondary hash table $S_j$ with an associate hash function $h_j$.
- By choosing the hash functions $h_j$ carefully, we can guarantee that there are no collisions at the secondary level.
- In order to guarantee that there are no collisions on the secondary level, we need to let the size $m_j$ of the hash table $S_j$ be the square of the number $n_j$ of keys hashing to slot $j$. 
Perfect Hashing

- Consider the key set $K = \{10, 22, 37, 40, 52, 60, 70, 72, 74\}$
- The first level hash function is
  \[ h(k) = (ak + b \mod p) \mod m \]
  with parameters $(m, a, b, p) = (9, 3, 42, 101)$, where $m$ is the table size
- For example, $h(75)$ is computed as
  \[
  h(75) = (3 \cdot 75 + 42 \mod 101) \mod 9 \\
  = (267 \mod 101) \mod 9 \\
  = 65 \mod 9 \\
  = 2
  \]
Perfect Hashing

- A secondary hash table $S_j$ stores all keys hashing to slot $j$
- The size of hash table $S_j$ is $m_j = n_j^2$, where $n_j$ is the number of keys hashing to slot $j$
- The associated hash function of $S_j$ is

$$h_j(k) = (a_j k + b_j \mod p) \mod m_j$$
On the second level, we use the hash function belonging to Slot 2, which has the parameters \((m_2, a_2, b_2) = (9, 10, 18)\) and the same prime \(p = 101\), therefore, we compute \(h_2(75)\) as

\[
h_2(75) = (10 \cdot 75 + 18 \mod 101) \mod 9 = 7
\]

and place the key 75 in the 7th cell of the Slot 2 table.
If we store \( n \) keys in a hash table of size \( m = n^2 \) using a universal hash function, then the probability of collision is \( 1/2 \).

There are \( C(n, 2) \) pairs of different pairs of keys.

The probability that a pair collides is \( 1/m \), if \( h \) is chosen from \( \mathcal{H} \).

Let \( X \) be the number of collisions, since \( m = n^2 \), the expected value of \( X \) is

\[
E[X] = C(n, 2) \cdot \frac{1}{n^2} = \frac{n(n-1)}{2} \cdot \frac{1}{n^2} < \frac{1}{2}
\]
Perfect Hashing Properties

- Since we choose $m = n^2$, a hash function $h$ chosen at random from $\mathcal{H}$ is more likely not to have collisions.
- Given a static set of $n$ keys, it is easy to find a collision-free hash function $h$.
- When $n$ is large, a hash table of size $m = n^2$ is excessive.
- However, in the two-level approach we only hash the entries in each slot.
- On the first level the hash function $h$ hashes $n$ keys into $m = n$ slots.
- Then, if $n_j$ keys hash to slot $j$ we use the secondary hash table of size $m_j = n_j^2$ to provide a collision-free constant-time lookup.
In the first level table size is $m = n$, and therefore, the amount of the memory used is $O(n)$ for the primary hash table.

In the secondary hash tables, each hash table $S_j$ is of size $n_j^2$.

To compute the total memory used in the secondary tables, we need to know the expected sum of the squares of the number of keys $n_j$ that hash to slot $j$, which turns out to be

$$E \left[ \sum_{j=0}^{m-1} m_j \right] = E \left[ \sum_{j=0}^{m-1} n_j^2 \right] < 2n$$

Therefore, the total secondary storage is also $O(n)$.