Data Structures and Algorithms [1]

BITS-Pilani K. K. Birla Goa Campus

[1] Material for the presentation taken from Cormen, Leiserson, Rivest and Stein, *Introduction to Algorithms, Third Edition*;

Searching in constant time

- Suppose we use roll number as the key.

Searching in constant time

- Suppose we use roll number as the key.
- Data Structures seen so far :
  - Queues
  - Priority Queues
  - Red-black Trees

Searching in constant time

- Suppose we use roll number as the key.
- Data Structures seen so far :
  - Queues
  - Priority Queues
  - Red-black Trees
- It is not possible to search a key in constant time.

Searching in constant time

- Suppose we use roll number as the key.
- Data Structures seen so far :
  - Queues
  - Priority Queues
  - Red-black Trees
- It is not possible to search a key in constant time.
- Hash tables is a datastructure that allows us to perform SEARCH operation in $O(1)$ time.

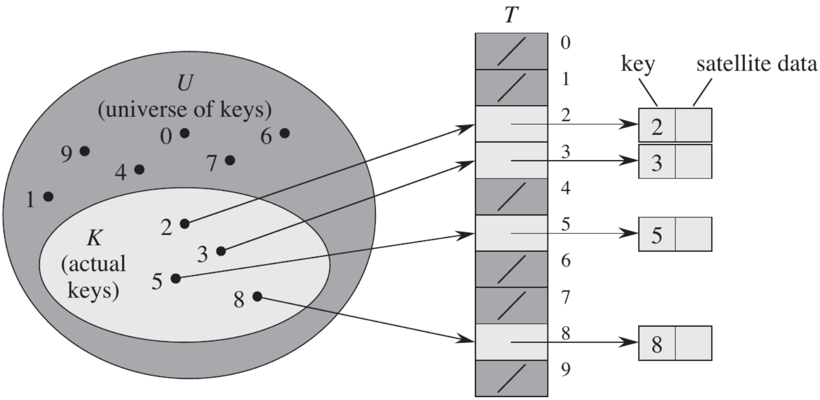
Searching in constant time

- Suppose we use roll number as the key.
- Data Structures seen so far :
  - Queues
  - Priority Queues
  - Red-black Trees
- It is not possible to search a key in constant time.
- Hash tables is a datastructure that allows us to perform SEARCH operation in $O(1)$ time.
- Average time to perform any operation (INSERT, SEARCH and DELETE) should be $O(1)$.

Satellite Data related to keys

- Roll : **1023**
  Name : Ravi
  YOB : 2000
- Roll : **0912**
  Name : Lata
  YOB : 2000
- Roll : **1756**
  Name : Sagar
  YOB : 1999
- Roll : **1504**
  Name : Mohit
  YOB : 1999
- Roll : **1393**
  Name : Maya
  YOB : 1999

Direct-address Table



Direct-address Table

- Direct-address Table is an array $T[0 \dots m-1]$, where each slot corresponds to a key in the universe U.

Direct-address Table

- Direct-address Table is an array $T[0 \ldots m-1]$, where each slot corresponds to a key in the universe U.
- Works well if :

Direct-address Table

- Direct-address Table is an array $T[0 \dots m-1]$, where each slot corresponds to a key in the universe U.
- Works well if :

1. The universe U of keys is small. $U = 0, 1, \dots, m-1$

Direct-address Table

- Direct-address Table is an array $T[0 \dots m-1]$, where each slot corresponds to a key in the universe U.
- Works well if :

1. The universe U of keys is small. $U = 0, 1, \dots, m-1$
2. No two elements in the dynamic set have the same key.

Direct-address Table

DIRECT-ADDRESS-SEARCH(T, k)

return $T[k]$

DIRECT-ADDRESS-INSERT(T, x)

$T[x.key] = x$

DIRECT-ADDRESS-DELETE(T, x)

$T[x.key] = \text{NIL}$

Direct-address Table

```
DIRECT-ADDRESS-SEARCH(T, k)
    return T[k]
```

```
DIRECT-ADDRESS-INSERT(T, x)
    T[x.key] = x
```

```
DIRECT-ADDRESS-DELETE(T, x)
    T[x.key] = NIL
```

- Each of the operations take $O(1)$ time.

Direct-address Table

DIRECT-ADDRESS-SEARCH(T, k)

1 **return** $T[k]$

DIRECT-ADDRESS-INSERT(T, x)

1 $T[x.key] = x$

DIRECT-ADDRESS-DELETE(T, x)

1 $T[x.key] = \text{NIL}$

- Each of the operations take $O(1)$ time.
- What are the problems?

Direct-address Table

DIRECT-ADDRESS-SEARCH(T, k)

1 **return** $T[k]$

DIRECT-ADDRESS-INSERT(T, x)

1 $T[x.key] = x$

DIRECT-ADDRESS-DELETE(T, x)

1 $T[x.key] = \text{NIL}$

- Each of the operations take $O(1)$ time.
- What are the problems?
  1. m cannot be large.

Direct-address Table

DIRECT-ADDRESS-SEARCH(T, k)

1 **return** $T[k]$

DIRECT-ADDRESS-INSERT(T, x)

1 $T[x.key] = x$

DIRECT-ADDRESS-DELETE(T, x)

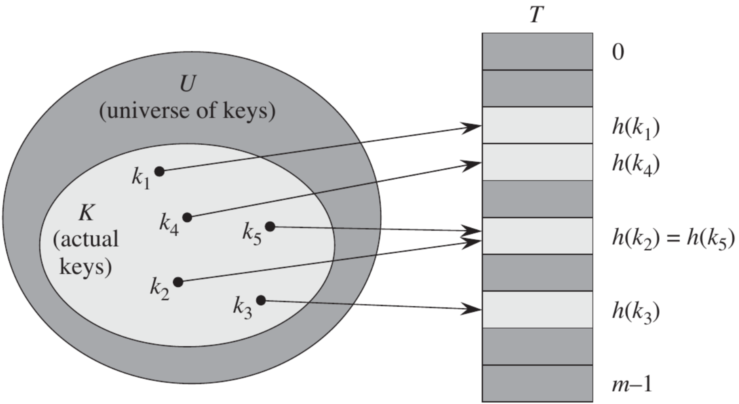
1 $T[x.key] = \text{NIL}$

- Each of the operations take $O(1)$ time.
- What are the problems?
  1. m cannot be large.
  2. Two elements cannot have the same key.

Satellite Data related to keys

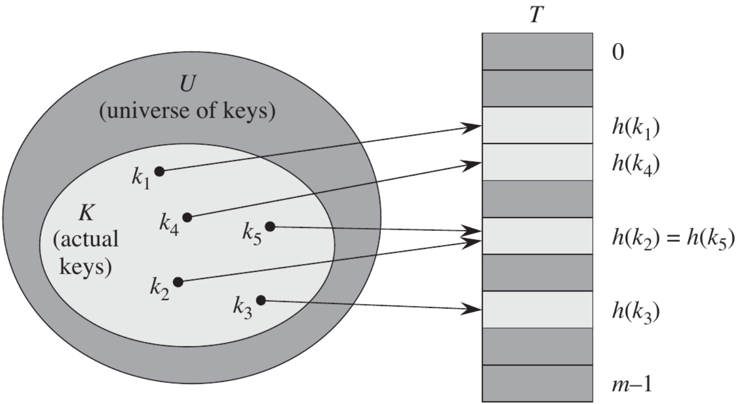
- Roll : **1023**
  Name : Ravi
  YOB : 2000
- Roll : **0912**
  Name : Lata
  YOB : 2000
- Roll : **1756**
  Name : Sagar
  YOB : 1999
- Roll : **1504**
  Name : Mohit
  YOB : 1999
- Roll : **1393**
  Name : Maya
  YOB : 1999

Hash tables



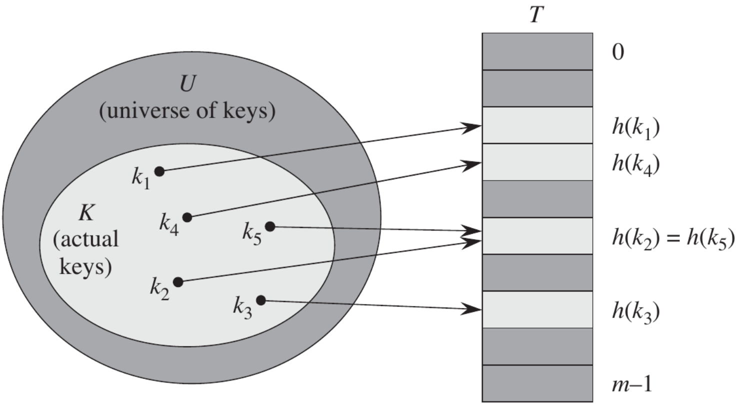
T
U
(universe of keys)
K
(actual keys)
k_1
k_4
k_5
k_2
k_3
0
$h(k_1)$
$h(k_4)$
$h(k_2) = h(k_5)$
$h(k_3)$
$m-1$

Hash tables



- Using the hash value we can search an item in $O(1)$ time.

Hash tables



- Using the hash value we can search an item in $O(1)$ time.
- However, two keys can hash to the same slot (collision)

Hash tables

- A Hash table has m slots $T[0 \ldots m-1]$ similar to Direct-address table.

Hash tables

- A Hash table has m slots $T[0 \ldots m-1]$ similar to Direct-address table.
- However, m is usually much less than $|U|$.

Hash tables

- A Hash table has m slots $T[0 \dots m-1]$ similar to Direct-address table.
- However, m is usually much less than $|U|$.
- An element with key k is stored in slot $h(k)$, where h is the hash function.

Hash tables

- A Hash table has m slots $T[0 \dots m-1]$ similar to Direct-address table.
- However, m is usually much less than $|U|$.
- An element with key k is stored in slot $h(k)$, where h is the hash function.
- h is a mapping from the universe of keys U to the slots of the hash table.

  $h : U \rightarrow 0, 1, \dots, m-1$

Hash tables

- A Hash table has m slots $T[0 \dots m-1]$ similar to Direct-address table.
- However, m is usually much less than $|U|$.
- An element with key k is stored in slot $h(k)$, where h is the hash function.
- h is a mapping from the universe of keys U to the slots of the hash table.

  $h : U \rightarrow 0, 1, \dots, m-1$
- $h(k)$ is called the hash value of key k

Hash table vs. Direct-address table

- Storage requirements of Direct-address table is $\Theta(|U|)$ with no collisions.

Hash table vs. Direct-address table

- Storage requirements of Direct-address table is $\Theta(|U|)$ with no collisions.
- Storage requirements of Hash table is only $\Theta(m)$, but collisions may occur.

Dealing with Collisions

1. Minimizing collision:

Dealing with Collisions

1. Minimizing collision: Choose a suitable hash function h such that h appears to be random. That is, each slot has the same number of preimages in the universe U.

Dealing with Collisions

1. Minimizing collision: Choose a suitable hash function h such that h appears to be random. That is, each slot has the same number of preimages in the universe U.
2. Resolving collision:

Dealing with Collisions

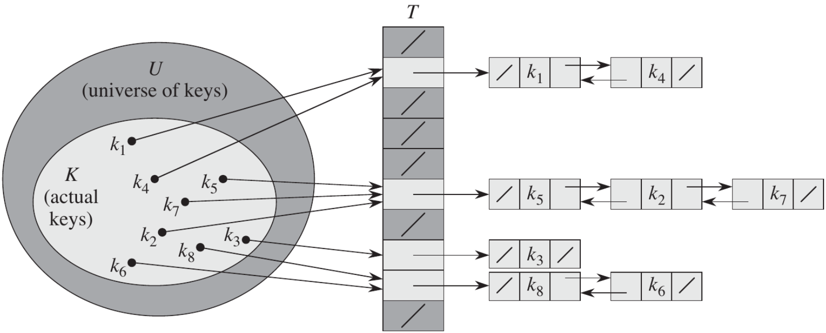
1. Minimizing collision: Choose a suitable hash function h such that h appears to be random. That is, each slot has the same number of preimages in the universe U.
2. Resolving collision: Use chaining

Dealing with Collisions

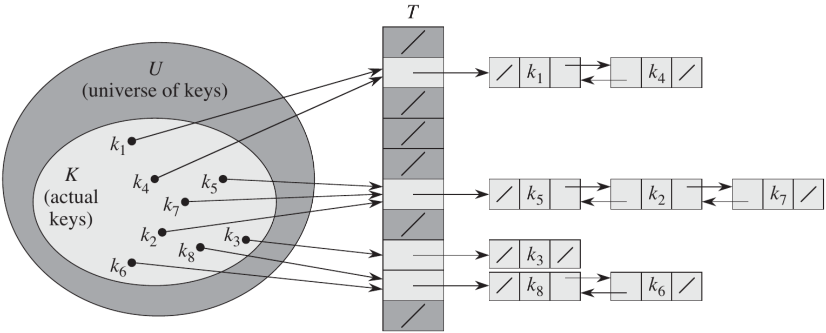
1. Minimizing collision: Choose a suitable hash function h such that h appears to be random. That is, each slot has the same number of preimages in the universe U.
2. Resolving collision: Use chaining

   In chaining, we place all the elements that hash to the same slot into the same linked list.

Collision resolution by chaining



Collision resolution by chaining



- Each slot contains pointer to the head of the linked list.

Operations on Hash table with chaining

CHAINED-HASH-INSERT(T, x)
1 LIST-PREPEND$(T[h(x.key)], x)$

CHAINED-HASH-SEARCH(T, k)
1 **return** LIST-SEARCH$(T[h(k)], k)$

CHAINED-HASH-DELETE(T, x)
1 LIST-DELETE$(T[h(x.key)], x)$

Operations on Hash table with chaining

CHAINED-HASH-INSERT(T, x)
1 LIST-PREPEND$(T[h(x.key)], x)$

CHAINED-HASH-SEARCH(T, k)
1 **return** LIST-SEARCH$(T[h(k)], k)$

CHAINED-HASH-DELETE(T, x)
1 LIST-DELETE$(T[h(x.key)], x)$

- For searching, the worst case running time is proportional to the length of the list.

Operations on Hash table with chaining

CHAINED-HASH-INSERT(T, x)
1 LIST-PREPEND$(T[h(x.key)], x)$

CHAINED-HASH-SEARCH(T, k)
1 **return** LIST-SEARCH$(T[h(k)], k)$

CHAINED-HASH-DELETE(T, x)
1 LIST-DELETE$(T[h(x.key)], x)$

- For searching, the worst case running time is proportional to the length of the list.
- We can insert an element in $O(1)$ time.

Operations on Hash table with chaining

CHAINED-HASH-INSERT(T, x)
1 LIST-PREPEND$(T[h(x.key)], x)$

CHAINED-HASH-SEARCH(T, k)
1 **return** LIST-SEARCH$(T[h(k)], k)$

CHAINED-HASH-DELETE(T, x)
1 LIST-DELETE$(T[h(x.key)], x)$

- For searching, the worst case running time is proportional to the length of the list.
- We can insert an element in $O(1)$ time. We are assuming that the key is not already present in the list.

Operations on Hash table with chaining

```
CHAINED-HASH-INSERT(T, x)
1  LIST-PREPEND(T[h(x.key)], x)

CHAINED-HASH-SEARCH(T, k)
1  return LIST-SEARCH(T[h(k)], k)

CHAINED-HASH-DELETE(T, x)
1  LIST-DELETE(T[h(x.key)], x)
```

- For searching, the worst case running time is proportional to the length of the list.
- We can insert an element in $O(1)$ time. We are assuming that the key is not already present in the list.
- An element can be deleted in $O(1)$ time if we have a pointer to the element.

Operations on Hash table with chaining

```
CHAINED-HASH-INSERT(T, x)
1  LIST-PREPEND(T[h(x.key)], x)

CHAINED-HASH-SEARCH(T, k)
1  return LIST-SEARCH(T[h(k)], k)

CHAINED-HASH-DELETE(T, x)
1  LIST-DELETE(T[h(x.key)], x)
```

- For searching, the worst case running time is proportional to the length of the list.
- We can insert an element in $O(1)$ time. We are assuming that the key is not already present in the list.
- An element can be deleted in $O(1)$ time if we have a pointer to the element. This is because we are using a doubly linked list.

Analysis of hashing with chaining

- How long does it take to search an element with a given key?

Analysis of hashing with chaining

- How long does it take to search an element with a given key?
- Suppose we store n elements in a hash table having m slots.

Analysis of hashing with chaining

- How long does it take to search an element with a given key?
- Suppose we store n elements in a hash table having m slots.
- Load factor $\alpha = n/m$

Analysis of hashing with chaining

- How long does it take to search an element with a given key?
- Suppose we store n elements in a hash table having m slots.
- Load factor $\alpha = n/m$
- Our analysis will be in terms of α.

Analysis of hashing with chaining

- How long does it take to search an element with a given key?
- Suppose we store n elements in a hash table having m slots.
- Load factor $\alpha = n/m$
- Our analysis will be in terms of α.
- Worst-case running time $\Theta(n)$.

Hashing with chaining : average-case performance

- Depends on how well the hash function distributes the set of keys among the m slots.

Hashing with chaining : average-case performance

- Depends on how well the hash function distributes the set of keys among the m slots.
- Independent uniform hashing : any given key k is equally likely to hash into any of the m slots independent of other keys.

Hashing with chaining : average-case performance

- Depends on how well the hash function distributes the set of keys among the m slots.
- Independent uniform hashing : any given key k is equally likely to hash into any of the m slots independent of other keys.
- Let length of the linked list at slot $T[j]$ be denoted by n_j.

Hashing with chaining : average-case performance

- Depends on how well the hash function distributes the set of keys among the m slots.
- Independent uniform hashing : any given key k is equally likely to hash into any of the m slots independent of other keys.
- Let length of the linked list at slot $T[j]$ be denoted by n_j.
- Expected value of n_j, $E[n_j] = \alpha = n/m$

Hashing with chaining : average-case performance

- Depends on how well the hash function distributes the set of keys among the m slots.
- Independent uniform hashing : any given key k is equally likely to hash into any of the m slots independent of other keys.
- Let length of the linked list at slot $T[j]$ be denoted by n_j.
- Expected value of n_j, $E[n_j] = \alpha = n/m$
- Time required to search an element with key k depends linearly on the length $n_{h(k)}$ of the list $T[h(k)]$.

Hashing with chaining : SEARCH average-case time

Theorem: (Unsuccessful search)

In a hash table in which collisions are resolved by chaining, an unsuccessful search takes average-case time $\Theta(1 + \alpha)$, under the assumption of independent uniform hashing.

Hashing with chaining : SEARCH average-case time

Theorem: (Unsuccessful search)

In a hash table in which collisions are resolved by chaining, an unsuccessful search takes average-case time $\Theta(1+\alpha)$, under the assumption of independent uniform hashing.

Theorem: (Successful search)

In a hash table in which collisions are resolved by chaining, a successful search takes average-case time $\Theta(1+\alpha)$, under the assumption of independent uniform hashing.

Hash functions

- What makes a good hash function?

Hash functions

- What makes a good hash function?

  Independent uniform hashing

Hash functions

- What makes a good hash function?

  Independent uniform hashing
- E.g. Suppose we know that the keys k are uniformly distributed in the range $0 \leq k < 1$.

Hash functions

- What makes a good hash function?

  Independent uniform hashing
- E.g. Suppose we know that the keys k are uniformly distributed in the range $0 \leq k < 1$.

  Then $h(k) = \lfloor km \rfloor$ satisfies independent uniform hashing.

Hash functions

- What makes a good hash function?

  Independent uniform hashing
- E.g. Suppose we know that the keys k are uniformly distributed in the range $0 \leq k < 1$.

  Then $h(k) = \lfloor km \rfloor$ satisfies independent uniform hashing.
- Goal : Come up with heuristic methods to achieve independent uniform hashing.

Creating Hash functions

- Compiler's symbol table

Creating Hash functions

- Compiler's symbol table
- Closely related symbols (**pt** and **pts**) often occur in the same program.

Creating Hash functions

- Compiler's symbol table
- Closely related symbols (**pt** and **pts**) often occur in the same program.
- The computed hash value should be independent of any pattern that might exist in the keys.

Interpreting keys as natural numbers

- Most hash functions assume that the universe of keys is the set of natural numbers.

  $\mathbb{N} = \{0, 1, 2, \ldots\}$

Interpreting keys as natural numbers

- Most hash functions assume that the universe of keys is the set of natural numbers.

  $\mathbb{N} = \{0, 1, 2, \ldots\}$
- Approach : Find a way to interpret non-integer keys as natural numbers.

Interpreting keys as natural numbers

- Most hash functions assume that the universe of keys is the set of natural numbers.

  $\mathbb{N} = \{0, 1, 2, \ldots\}$
- Approach : Find a way to interpret non-integer keys as natural numbers.
- For example: variable **pt**

Interpreting keys as natural numbers

- Most hash functions assume that the universe of keys is the set of natural numbers.

  $\mathbb{N} = \{0, 1, 2, \ldots\}$
- Approach : Find a way to interpret non-integer keys as natural numbers.
- For example: variable **pt**

  We can think of this as a radix-128 integer

Interpreting keys as natural numbers

- Most hash functions assume that the universe of keys is the set of natural numbers.

  $\mathbb{N} = \{0, 1, 2, \ldots\}$
- Approach : Find a way to interpret non-integer keys as natural numbers.
- For example: variable **pt**

  We can think of this as a radix-128 integer

  ASCII values of p and t are 112 and 116

Interpreting keys as natural numbers

- Most hash functions assume that the universe of keys is the set of natural numbers.

  $\mathbb{N} = \{0, 1, 2, \ldots\}$
- Approach : Find a way to interpret non-integer keys as natural numbers.
- For example: variable **pt**

  We can think of this as a radix-128 integer

  ASCII values of p and t are 112 and 116

  $112 \times 128^1 + 116 \times 128^0$

Interpreting keys as natural numbers

- Most hash functions assume that the universe of keys is the set of natural numbers.

  $\mathbb{N} = \{0, 1, 2, \ldots\}$
- Approach : Find a way to interpret non-integer keys as natural numbers.
- For example: variable **pt**

  We can think of this as a radix-128 integer

  ASCII values of p and t are 112 and 116

  $112 \times 128^1 + 116 \times 128^0 = 14452$

Interpreting keys as natural numbers

- Most hash functions assume that the universe of keys is the set of natural numbers.

  $\mathbb{N} = \{0, 1, 2, \ldots\}$
- Approach : Find a way to interpret non-integer keys as natural numbers.
- For example: variable **pt**

  We can think of this as a radix-128 integer

  ASCII values of p and t are 112 and 116

  $112 \times 128^1 + 116 \times 128^0 = 14452$
- In our discussion, we will assume that keys are natural numbers.

The division method

- $h(k) = k \bmod m$

The division method

- $h(k) = k \bmod m$
  E.g. If $m = 12$ and $k = 100$, then $h(k) = 4$.

The division method

- $h(k) = k \bmod m$
  E.g. If $m = 12$ and $k = 100$, then $h(k) = 4$.
- m should not be a power of 2

The division method

- $h(k) = k \bmod m$

  E.g. If $m = 12$ and $k = 100$, then $h(k) = 4$.
- m should not be a power of 2

  If $m = 2^p$, then $k \bmod m$ will be just p lower-order bits of k.

The division method

- $h(k) = k \bmod m$

  E.g. If $m = 12$ and $k = 100$, then $h(k) = 4$.

- m should not be a power of 2

  If $m = 2^p$, then $k \bmod m$ will be just p lower-order bits of k.

  If $k = 15$ and $m = 2^3$, then $h(k) = (15 \bmod 2^3) = 7$.

The division method

- $h(k) = k \bmod m$

  E.g. If $m = 12$ and $k = 100$, then $h(k) = 4$.
- m should not be a power of 2

  If $m = 2^p$, then $k \bmod m$ will be just p lower-order bits of k.

  If $k = 15$ and $m = 2^3$, then $h(k) = (15 \bmod 2^3) = 7$.
- A good choice for m : A prime number that is not too close to any power of 2.

The division method

- Suppose $n = 2000$ and $\alpha = \dfrac{n}{m} = 3$.

The division method

- Suppose $n = 2000$ and $\alpha = \dfrac{n}{m} = 3$.
- $m = 701$ could be a good choice because it is a prime number close to $\dfrac{n}{\alpha} = \dfrac{2000}{3}$

The division method

- Suppose $n = 2000$ and $\alpha = \dfrac{n}{m} = 3$.
- $m = 701$ could be a good choice because it is a prime number close to $\dfrac{n}{\alpha} = \dfrac{2000}{3}$
  Also, $m = 701$ is not close to any power of 2.

The multiplication method

- $h(k) = \lfloor m(kA \bmod 1) \rfloor$

The multiplication method

- $h(k) = \lfloor m(kA \bmod 1) \rfloor$

1. We multiply k with a constant A in the range $0 < A < 1$.

The multiplication method

- $h(k) = \lfloor m(kA \bmod 1) \rfloor$

1. We multiply k with a constant A in the range $0 < A < 1$.
2. We extract the fractional part of kA.

The multiplication method

- $h(k) = \lfloor m(kA \bmod 1) \rfloor$

1. We multiply k with a constant A in the range $0 < A < 1$.
2. We extract the fractional part of kA.
3. We multiply the fractional part with m and take the floor of the result.

The multiplication method

- $h(k) = \lfloor m(kA \bmod 1) \rfloor$

1. We multiply k with a constant A in the range $0 < A < 1$.
2. We extract the fractional part of kA.
3. We multiply the fractional part with m and take the floor of the result.

- The value of m is not critical. It is typically chosen to be a power of 2.

The multiplication method

- $h(k) = \lfloor m(kA \bmod 1) \rfloor$

1. We multiply k with a constant A in the range $0 < A < 1$.
2. We extract the fractional part of kA.
3. We multiply the fractional part with m and take the floor of the result.

- The value of m is not critical. It is typically chosen to be a power of 2.
- The fraction A should be an irrational number.

The multiplication method

- $h(k) = \lfloor m(kA \bmod 1) \rfloor$

1. We multiply k with a constant A in the range $0 < A < 1$.
2. We extract the fractional part of kA.
3. We multiply the fractional part with m and take the floor of the result.

- The value of m is not critical. It is typically chosen to be a power of 2.
- The fraction A should be an irrational number.
- Knuth has suggested A be $\dfrac{\sqrt{5}-1}{2}$. (It is inverse of the golden ratio.)

The multiplication method

- $h(k) = \lfloor m(kA \bmod 1) \rfloor$

1. We multiply k with a constant A in the range $0 < A < 1$.
2. We extract the fractional part of kA.
3. We multiply the fractional part with m and take the floor of the result.

- The value of m is not critical. It is typically chosen to be a power of 2.
- The fraction A should be an irrational number.
- Knuth has suggested A be $\dfrac{\sqrt{5}-1}{2}$. (It is inverse of the golden ratio.)

  $$\frac{\sqrt{5}-1}{2} = 0.6180339887\ldots$$

The multiplication method

- Suppose A is a rational number. Then kA will also be a rational number.

The multiplication method

- Suppose A is a rational number. Then kA will also be a rational number.
- E.g. $543 \times \dfrac{322}{555} = 315.0378378378\ldots$

The multiplication method

- Suppose A is a rational number. Then kA will also be a rational number.
- E.g. $543 \times \frac{322}{555} = 315.0378378378\ldots$ (Recurring number)

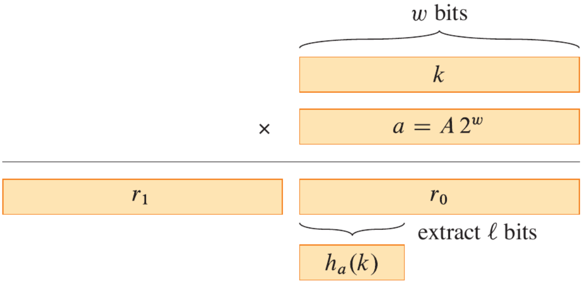
The multiplication method

- Suppose A is a rational number. Then kA will also be a rational number.
- E.g. $543 \times \frac{322}{555} = 315.0378378378\ldots$ (Recurring number)
- So, the fraction part of kA is not uniformly distributed in the interval $(0, 1)$.

The multiplication method

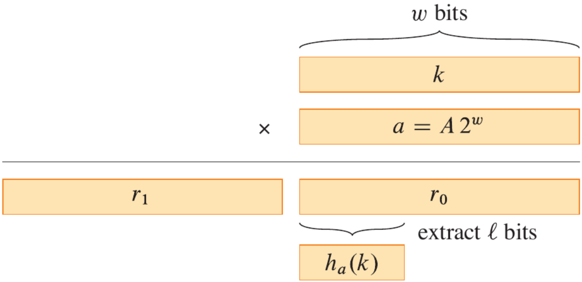
- Suppose A is a rational number. Then kA will also be a rational number.
- E.g. $543 \times \frac{322}{555} = 315.0378378378\ldots$ (Recurring number)
- So, the fraction part of kA is not uniformly distributed in the interval $(0, 1)$.
- When A is an irrational number, the pattern in the fractional part of kA becomes less predictable (there is no recurring pattern).

Multiply-shift Method



- Product (2-w bit value) $= r_1\,2^w + r_0$

Multiply-shift Method

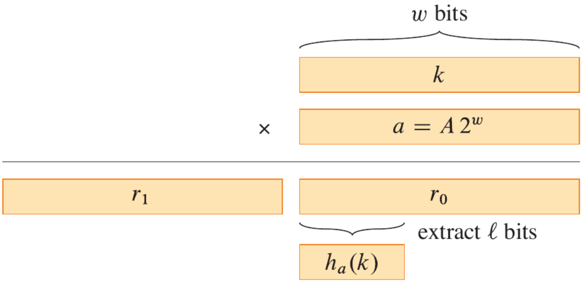


- Product (2-w bit value) $= r_1\,2^w + r_0$
- $h_a(k) = (k\,a \mod 2^w) \ggg (w - l)$

Static Hashing vs. Random Hashing

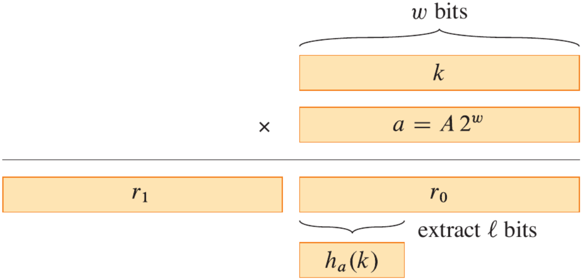
Static Hashing vs. Random Hashing

- Division method : $h(k) = k \mod m$
- Multiplication method : $h(k) = \lfloor m(kA \mod 1) \rfloor$
- Multiply-shift method :



Static Hashing vs. Random Hashing

- Division method : $h(k) = k \mod m$
- Multiplication method : $h(k) = \lfloor m(kA \mod 1) \rfloor$
- Multiply-shift method :



- $h(k) = (k(2^w A) \mod 2^w) \ggg (w - l)$

Take first l bits of remainder(result of mod)

Motivation for Random Hashing

- Denial-of-service attack based on Session IDs.

Motivation for Random Hashing

- Denial-of-service attack based on Session IDs.
- Hash value of IP address.

Universal family of Hash functions

- $h_{ab}(k) = ((ak + b) \mod p) \mod m$
  where $a \in \{1, \ldots, p-1\}$ and $b \in \{0, 1, \ldots, p-1\}$.

Universal family of Hash functions

- $h_{ab}(k) = ((ak + b) \mod p) \mod m$
  where $a \in \{1, \ldots, p-1\}$ and $b \in \{0, 1, \ldots, p-1\}$.
  $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$, $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$

Universal family of Hash functions

- $h_{ab}(k) = ((ak + b) \mod p) \mod m$
  where $a \in \{1, \ldots, p-1\}$ and $b \in \{0, 1, \ldots, p-1\}$.
  $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$, $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$
- Example, $p = 19$, $m = 8$, $a = 2$ and $b = 3$:
  What is $h_{2,3}(10)$?

Universal family of Hash functions

- $h_{ab}(k) = ((ak + b) \mod p) \mod m$
  where $a \in \{1, \ldots, p-1\}$ and $b \in \{0, 1, \ldots, p-1\}$.
  $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$, $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$
- Example, $p = 19$, $m = 8$, $a = 2$ and $b = 3$:
  What is $h_{2,3}(10)$? 4

Universal family of Hash functions

- $h_{ab}(k) = ((ak + b) \mod p) \mod m$
  where $a \in \{1, \dots, p-1\}$ and $b \in \{0, 1, \dots, p-1\}$.
  $\mathbb{Z}_p^* = \{1, \dots, p-1\}$, $\mathbb{Z}_p = \{0, 1, \dots, p-1\}$
- Example, $p = 19$, $m = 8$, $a = 2$ and $b = 3$:
  What is $h_{2,3}(10)$? 4
- p is chosen large enough such that every possible key $k \in \mathbb{Z}_p$.

Universal family of Hash functions

- $h_{ab}(k) = ((ak + b) \mod p) \mod m$
  where $a \in \{1, \ldots, p-1\}$ and $b \in \{0, 1, \ldots, p-1\}$.
  $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$, $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$
- Example, $p = 19$, $m = 8$, $a = 2$ and $b = 3$:
  What is $h_{2,3}(10)$? 4
- p is chosen large enough such that every possible key $k \in \mathbb{Z}_p$.
- Family of hash functions :
  $\mathcal{H} = \{h_{ab} : a \in \mathbb{Z}_p^* \text{ and } b \in \mathbb{Z}_p\}$

Universal family of Hash functions

- $h_{ab}(k) = ((ak + b) \mod p) \mod m$
  where $a \in \{1, \ldots, p-1\}$ and $b \in \{0, 1, \ldots, p-1\}$.
  $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$, $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$
- Example, $p = 19$, $m = 8$, $a = 2$ and $b = 3$:
  What is $h_{2,3}(10)$? 4
- p is chosen large enough such that every possible key $k \in \mathbb{Z}_p$.
- Family of hash functions :
  $\mathcal{H} = \{h_{ab} : a \in \mathbb{Z}_p^* \text{ and } b \in \mathbb{Z}_p\}$
- What is $|\mathcal{H}|$?

Universal family of hash functions

- Let h be a hash function that is picked uniformly randomly from $\mathcal{H}_{pm}$:

  A family $\mathcal{H}_{pm}$ is universal if for any distinct keys k_1 and k_2 in U, the probability that $h(k_1) = h(k_2)$ is at most $1/m$.

Universal family of hash functions

- Let h be a hash function that is picked uniformly randomly from $\mathcal{H}_{pm}$:

  A family $\mathcal{H}_{pm}$ is universal if for any distinct keys k_1 and k_2 in U, the probability that $h(k_1) = h(k_2)$ is at most $1/m$.
- **Theorem :** The family $\mathcal{H}_{pm}$ of hash functions defined below is universal.

  $h_{ab}(k) = ((ak + b) \mod p) \mod m$, where $a \in \mathbb{Z}_p^*$ and $b \in \mathbb{Z}_p$.

  $\mathcal{H}_{pm} = \{h_{ab} : a \in \mathbb{Z}_p^* \text{ and } b \in \mathbb{Z}_p\}$

Universal family of hash functions

- **Theorem :** The family $\mathcal{H}_{pm}$ of hash functions defined below is universal.

  $h_{ab}(k) = ((ak + b) \mod p) \mod m$, where $a \in \mathbb{Z}_p^*$ and $b \in \mathbb{Z}_p$.

  $\mathcal{H}_{pm} = \{h_{ab} : a \in \mathbb{Z}_p^* \text{ and } b \in \mathbb{Z}_p\}$

Universal family of hash functions

- **Theorem :** The family $\mathcal{H}_{pm}$ of hash functions defined below is universal.

  $h_{ab}(k) = ((ak + b) \mod p) \mod m$, where $a \in \mathbb{Z}_p^*$ and $b \in \mathbb{Z}_p$.

  $\mathcal{H}_{pm} = \{h_{ab} : a \in \mathbb{Z}_p^* \text{ and } b \in \mathbb{Z}_p\}$
- Proof:

Cryptographic Hash Functions

- SHA-256 (Secure Hash Algorithm 2)
  - Produces a 256-bit (32 byte) output for any input.

Cryptographic Hash Functions

- SHA-256 (Secure Hash Algorithm 2)
  - Produces a 256-bit (32 byte) output for any input.
  - Same 256-bit hash value for same input (deterministic algorithm).

Cryptographic Hash Functions

- SHA-256 (Secure Hash Algorithm 2)
  - Produces a 256-bit (32 byte) output for any input.
  - Same 256-bit hash value for same input (deterministic algorithm).
  - SHA-256 is designed to be computationally efficient.

Cryptographic Hash Functions

- SHA-256 (Secure Hash Algorithm 2)
  - Produces a 256-bit (32 byte) output for any input.
  - Same 256-bit hash value for same input (deterministic algorithm).
  - SHA-256 is designed to be computationally efficient.
- $h(k) = \text{SHA-256}(k) \mod m$

Cryptographic Hash Functions

- SHA-256 (Secure Hash Algorithm 2)
  - Produces a 256-bit (32 byte) output for any input.
  - Same 256-bit hash value for same input (deterministic algorithm).
  - SHA-256 is designed to be computationally efficient.
- $h(k) = \text{SHA-256}(k) \mod m$
- Family of hash functions:

  $h_a(k) = \text{SHA-256}(a \parallel k) \mod m$

Syllabus

- Only Sections 11.1, 11.2 and 11.3 will be part of CS F211 syllabus.

Syllabus

- Only Sections 11.1, 11.2 and 11.3 will be part of CS F211 syllabus.
- Open addressing (Section 11.4) will not be part of CS F211 syllabus.