Analysis of Algorithms

Hash Tables

Andres Mendez-Vazquez

September 24, 2020

Outline

Outline

About Basic Data Structures

Remark

It is quite interesting to notice that many data structures actually share similar operations!!!

Yes

If you think them as ADT

About Basic Data Structures

Remark

It is quite interesting to notice that many data structures actually share similar operations!!!

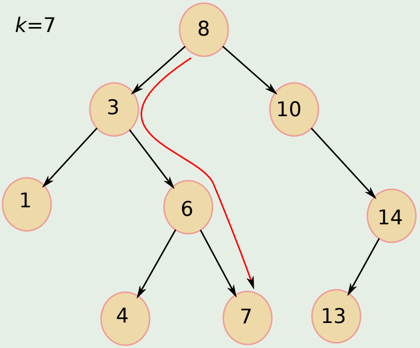
Yes

If you think them as ADT

Examples

Search(S,k)

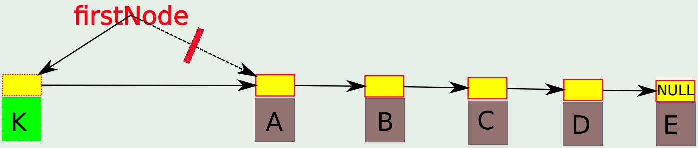
Example: Search in a BST



Examples

Insert(S,x)

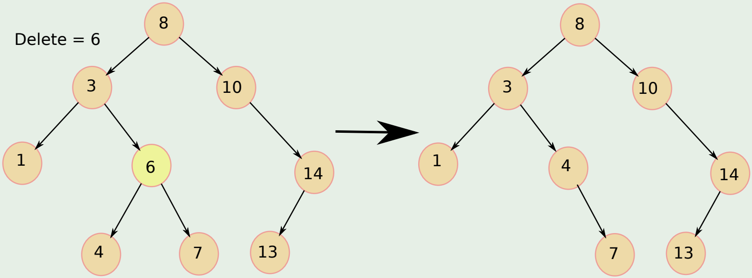
Example: Insert in a linked list



And Again

Delete(S,x)

Example: Delete in a BST



Basic data structures and operations.

Therefore

This are basic structures, it is up to you to read about them.

- Chapter 10 Cormen's book

Outline

Hash tables: Concepts

Definition

- A hash table or hash map T is a data structure, most commonly an array, that uses a hash function to efficiently map certain identifiers of keys (e.g. person names) to associated values.

Hash tables: Concepts

Definition

- A hash table or hash map T is a data structure, most commonly an array, that uses a hash function to efficiently map certain identifiers of keys (e.g. person names) to associated values.

Advantages

- They have the advantage of having a expected complexity of operations of $O(1+\alpha)$
  - Still, be aware of α

Hash tables: Concepts

Definition

- A hash table or hash map T is a data structure, most commonly an array, that uses a hash function to efficiently map certain identifiers of keys (e.g. person names) to associated values.

Advantages

- They have the advantage of having a expected complexity of operations of $O(1+\alpha)$
  - Still, be aware of α

However, If you have a large number of keys, U

- Then, it is impractical to store a table of the size of $|U|$.
- Thus, you can use a hash function $h : U \rightarrow \{0, 1, ..., m-1\}$

Hash tables: Concepts

Definition

- A hash table or hash map T is a data structure, most commonly an array, that uses a hash function to efficiently map certain identifiers of keys (e.g. person names) to associated values.

Advantages

- They have the advantage of having a expected complexity of operations of $O(1+\alpha)$
  - Still, be aware of α

However, If you have a large number of keys, U

- Then, it is impractical to store a table of the size of $|U|$.
- Thus, you can use a hash function $h : U \rightarrow \{0, 1, ..., m-1\}$

Hash tables: Concepts

Definition

- A hash table or hash map T is a data structure, most commonly an array, that uses a hash function to efficiently map certain identifiers of keys (e.g. person names) to associated values.

Advantages

- They have the advantage of having a expected complexity of operations of $O(1+\alpha)$
  - Still, be aware of α

However, If you have a large number of keys, U

- Then, it is impractical to store a table of the size of $|U|$.
- Thus, you can use a hash function $h : U \rightarrow \{0, 1, ..., m-1\}$

Outline

When you have a small universe of keys, U

Remarks

- It is not necessary to map the key values.
- Key values are direct addresses in the array.
- Direct implementation or Direct-address tables.

When you have a small universe of keys, U

Remarks

- It is not necessary to map the key values.
- Key values are direct addresses in the array.
- Direct implementation or Direct-address tables.

Operations

1. Direct-Address-Search(T, k)
   - return $T[k]$
2. Direct-Address-Search(T, x)
   - $T[x.key] = x$
3. Direct-Address-Delete(T, x)
   - $T[x.key] = NIL$

When you have a small universe of keys, U

Remarks

- It is not necessary to map the key values.
- Key values are direct addresses in the array.
- Direct implementation or Direct-address tables.

Operations

1. Direct-Address-Search(T, k)
   - return $T[k]$
2. Direct-Address-Search(T, x)
   - $T[x.key] = x$
3. Direct-Address-Delete(T, x)
   - $T[x.key] = NIL$

When you have a small universe of keys, U

Remarks

- It is not necessary to map the key values.
- Key values are direct addresses in the array.
- Direct implementation or Direct-address tables.

Operations

1. Direct-Address-Search(T, k)
   - return $T[k]$
2. Direct-Address-Search(T, x)
   - $T\,[x.key] = x$
3. Direct-Address-Delete(T, x)
   - $T\,[x.key] = NIL$

When you have a small universe of keys, U

Remarks

- It is not necessary to map the key values.
- Key values are direct addresses in the array.
- Direct implementation or Direct-address tables.

Operations

1. Direct-Address-Search(T, k)
   - return $T[k]$
2. Direct-Address-Search(T, x)
   - $T[x.key] = x$
3. Direct-Address-Delete(T, x)
   - $T[x.key] = NIL$

When you have a small universe of keys, U

Remarks

- It is not necessary to map the key values.
- Key values are direct addresses in the array.
- Direct implementation or Direct-address tables.

Operations

1. Direct-Address-Search(T, k)
   - return $T[k]$
2. Direct-Address-Search(T, x)
   - $T[x.key] = x$
3. Direct-Address-Delete(T, x)
   - $T[x.key] = NIL$

When you have a small universe of keys, U

Remarks

- It is not necessary to map the key values.
- Key values are direct addresses in the array.
- Direct implementation or Direct-address tables.

Operations

1. Direct-Address-Search(T, k)
   - return $T[k]$
2. Direct-Address-Search(T, x)
   - $T[x.key] = x$
3. Direct-Address-Delete(T, x)
   - $T[x.key] = NIL$

When you have a small universe of keys, U

Remarks

- It is not necessary to map the key values.
- Key values are direct addresses in the array.
- Direct implementation or Direct-address tables.

Operations

1. Direct-Address-Search(T, k)
   - return $T[k]$
2. Direct-Address-Search(T, x)
   - $T[x.key] = x$
3. Direct-Address-Delete(T, x)
   - $T[x.key] = NIL$

When you have a small universe of keys, U

Remarks

- It is not necessary to map the key values.
- Key values are direct addresses in the array.
- Direct implementation or Direct-address tables.

Operations

1. Direct-Address-Search(T, k)
   - return $T[k]$
2. Direct-Address-Search(T, x)
   - $T[x.key] = x$
3. Direct-Address-Delete(T, x)
   - $T[x.key] = NIL$

When you have a large universe of keys, U

Then

Then, it is impractical to store a table of the size of $|U|$.

You can use a especial function for mapping

$$h : U \rightarrow \{0, 1, ..., m - 1\} \tag{1}$$

Problem

With a large enough universe U, two keys can hash to the same value

- This is called a collision.

When you have a large universe of keys, U

Then

Then, it is impractical to store a table of the size of $|U|$.

You can use a especial function for mapping

$$h : U \rightarrow \{0, 1, ..., m-1\} \tag{1}$$

Problem

With a large enough universe U, two keys can hash to the same value

- This is called a collision.

When you have a large universe of keys, U

Then

Then, it is impractical to store a table of the size of $|U|$.

You can use a especial function for mapping

$$h : U \rightarrow \{0, 1, ..., m-1\} \tag{1}$$

Problem

With a large enough universe U, two keys can hash to the same value

- This is called a collision.

Outline

Collisions

This is a problem

We might try to avoid this by using a suitable hash function h.

Collisions

This is a problem

We might try to avoid this by using a suitable hash function h.

Idea

Make appear to be "random" enough to avoid collisions altogether (**Highly Improbable**) or to minimize the probability of them.

1. Chaining
2. Open Addressing

Collisions

This is a problem

We might try to avoid this by using a suitable hash function h.

Idea

Make appear to be "random" enough to avoid collisions altogether (**Highly Improbable**) or to minimize the probability of them.

You still have the problem of collisions

Possible Solutions to the problem:

1. Chaining
2. Open Addressing

Collisions

This is a problem

We might try to avoid this by using a suitable hash function h.

Idea

Make appear to be "random" enough to avoid collisions altogether (**Highly Improbable**) or to minimize the probability of them.

You still have the problem of collisions

Possible Solutions to the problem:

1. Chaining
2. Open Addressing

Collisions

This is a problem

We might try to avoid this by using a suitable hash function h.

Idea

Make appear to be "random" enough to avoid collisions altogether (**Highly Improbable**) or to minimize the probability of them.

You still have the problem of collisions

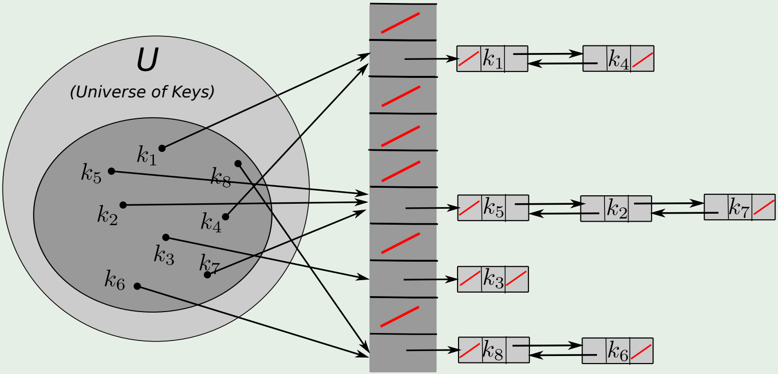
Possible Solutions to the problem:

1. Chaining
2. Open Addressing

Hash tables: Chaining

A Possible Solution

Insert the elements that hash to the same slot into a linked list.



Outline

Analysis of hashing with Chaining: Assumptions

Assumptions

- We have a load factor $\alpha = \frac{n}{m}$, where m is the size of the hash table T, and n is the number of elements to store.
- **Simple uniform hashing** property:
  - This means that any of the m slots can be selected.
  - This means that if $n = n_0 + n_1 + ... + n_{m-1}$, we have that $E(n_j) = \alpha$.

Analysis of hashing with Chaining: Assumptions

Assumptions

- We have a load factor $\alpha = \frac{n}{m}$, where m is the size of the hash table T, and n is the number of elements to store.
- **Simple uniform hashing** property:
  - This means that any of the m slots can be selected.
  - This means that if $n = n_0 + n_1 + ... + n_{m-1}$, we have that $E(n_j) = \alpha$.

To simplify the analysis, you need to consider two cases

- Unsuccessful search.
- Successful search.

Analysis of hashing with Chaining: Assumptions

Assumptions

- We have a load factor $\alpha = \frac{n}{m}$, where m is the size of the hash table T, and n is the number of elements to store.
- **Simple uniform hashing** property:
  - This means that any of the m slots can be selected.
  - This means that if $n = n_0 + n_1 + ... + n_{m-1}$, we have that $E(n_j) = \alpha$.

To simplify the analysis, you need to consider two cases

- Unsuccessful search.
- Successful search.

Analysis of hashing with Chaining: Assumptions

Assumptions

- We have a load factor $\alpha = \frac{n}{m}$, where m is the size of the hash table T, and n is the number of elements to store.
- **Simple uniform hashing** property:
  - This means that any of the m slots can be selected.
  - This means that if $n = n_0 + n_1 + ... + n_{m-1}$, we have that $E(n_j) = \alpha$.

To simplify the analysis, you need to consider two cases

- Unsuccessful search.
- Successful search.

cinvestav

Unidad Guadalajara

Analysis of hashing with Chaining: Assumptions

Assumptions

- We have a load factor $\alpha = \frac{n}{m}$, where m is the size of the hash table T, and n is the number of elements to store.
- **Simple uniform hashing** property:
  - This means that any of the m slots can be selected.
  - This means that if $n = n_0 + n_1 + ... + n_{m-1}$, we have that $E(n_j) = \alpha$.

To simplify the analysis, you need to consider two cases

- Unsuccessful search.
- Successful search.

Analysis of hashing with Chaining: Assumptions

Assumptions

- We have a load factor $\alpha = \frac{n}{m}$, where m is the size of the hash table T, and n is the number of elements to store.
- **Simple uniform hashing** property:
  - This means that any of the m slots can be selected.
  - This means that if $n = n_0 + n_1 + ... + n_{m-1}$, we have that $E(n_j) = \alpha$.

To simplify the analysis, you need to consider two cases

- Unsuccessful search.
- Successful search.

Why?

After all

You are always looking for keys when

- Searching
- Inserting
- Deleting

Why?

After all

You are always looking for keys when

- Searching
- Inserting
- Deleting

It is clear that we have two possibilities

Finding the key or not finding the key

Why?

After all

You are always looking for keys when

- Searching
- Inserting
- Deleting

It is clear that we have two possibilities

Finding the key or not finding the key

Why?

After all

You are always looking for keys when

- Searching
- Inserting
- Deleting

It is clear that we have two possibilities

Finding the key or not finding the key

Why?

After all

You are always looking for keys when

- Searching
- Inserting
- Deleting

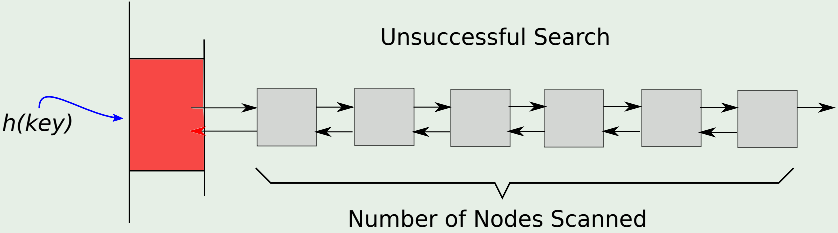
It is clear that we have two possibilities

Finding the key or not finding the key

Outline

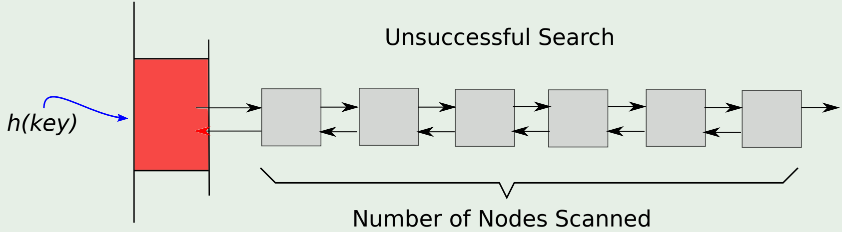
Therefore

We have two phenomena's

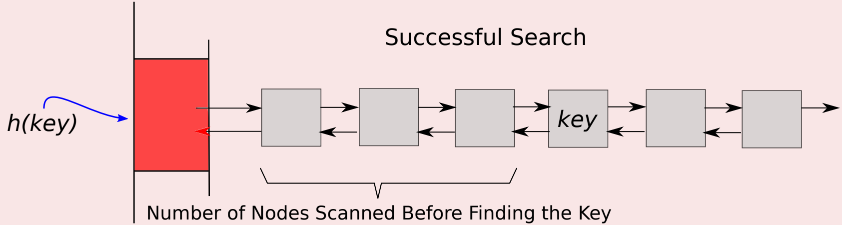


Therefore

We have two phenomena's



Second one



For this, we have the following theorems

Theorem 11.1

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 simple uniform hashing.

Theorem 11.2

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 simple uniform hashing.

For this, we have the following theorems

Theorem 11.1

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 simple uniform hashing.

Theorem 11.2

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 simple uniform hashing.

Analysis of hashing: Constant time.

Finally

These two theorems tell us that if $n = O(m)$

$$\alpha = \frac{n}{m} = \frac{O(m)}{m} = O(1)$$

Or search time is constant.

Analysis of hashing: Constant time.

Finally

These two theorems tell us that if $n = O(m)$

$$\alpha = \frac{n}{m} = \frac{O(m)}{m} = O(1)$$

Or search time is constant.

Analysis of hashing: Constant time.

Finally

These two theorems tell us that if $n = O(m)$

$$\alpha = \frac{n}{m} = \frac{O(m)}{m} = O(1)$$

Or search time is constant.

Outline

Analysis of hashing: Which hash function?

Consider that:

Good hash functions should maintain the property of simple uniform hashing!

- The keys have the same probability $1/m$ to be hashed to any bucket!!!
- A uniform hash function minimizes the likelihood of an overflow when keys are selected at random.

Analysis of hashing: Which hash function?

Consider that:

Good hash functions should maintain the property of simple uniform hashing!

- The keys have the same probability $1/m$ to be hashed to any bucket!!!
- A uniform hash function minimizes the likelihood of an overflow when keys are selected at random.

Then:

What should we use?

- If we know how the keys are distributed uniformly at the following interval $0 \leq k < 1$ then $h(k) = \lfloor km \rfloor$.

Analysis of hashing: Which hash function?

Consider that:

Good hash functions should maintain the property of simple uniform hashing!

- The keys have the same probability $1/m$ to be hashed to any bucket!!!
- A uniform hash function minimizes the likelihood of an overflow when keys are selected at random.

Then:

What should we use?

- If we know how the keys are distributed uniformly at the following interval $0 \leq k < 1$ then $h(k) = \lfloor km \rfloor$.

Analysis of hashing: Which hash function?

Consider that:

Good hash functions should maintain the property of simple uniform hashing!

- The keys have the same probability $1/m$ to be hashed to any bucket!!!
- A uniform hash function minimizes the likelihood of an overflow when keys are selected at random.

Then:

What should we use?

- If we know how the keys are distributed uniformly at the following interval $0 \leq k < 1$ then $h(k) = \lfloor km \rfloor$.

Analysis of hashing: Which hash function?

Consider that:

Good hash functions should maintain the property of simple uniform hashing!

- The keys have the same probability $1/m$ to be hashed to any bucket!!!
- A uniform hash function minimizes the likelihood of an overflow when keys are selected at random.

Then:

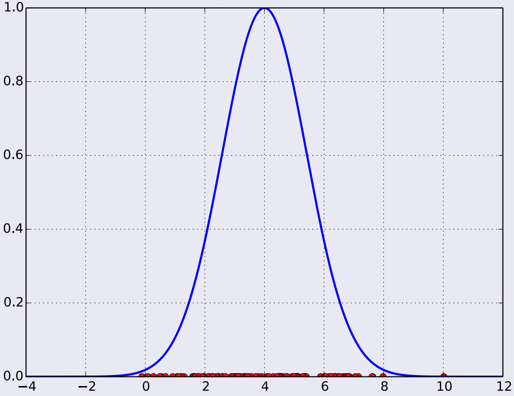
What should we use?

- If we know how the keys are distributed uniformly at the following interval $0 \leq k < 1$ then $h(k) = \lfloor km \rfloor$.

What if...

Question:

What about something with keys in a normal distribution?



Possible hash functions when the keys are natural numbers

The division method

- $h(k) = k \mod m$.
- Good choices for m are primes not too close to a power of 2.

Possible hash functions when the keys are natural numbers

The division method

- $h(k) = k \mod m$.
- Good choices for m are primes not too close to a power of 2.

The multiplication method

- $h(k) = \lfloor m(kA \mod 1) \rfloor$ with $0 < A < 1$.
- The value of m is not critical.
- Easy to implement in a computer.

Possible hash functions when the keys are natural numbers

The division method

- $h(k) = k \mod m$.
- Good choices for m are primes not too close to a power of 2.

The multiplication method

- $h(k) = \lfloor m(kA \mod 1) \rfloor$ with $0 < A < 1$.
- The value of m is not critical.
- Easy to implement in a computer.

Possible hash functions when the keys are natural numbers

The division method

- $h(k) = k \mod m$.
- Good choices for m are primes not too close to a power of 2.

The multiplication method

- $h(k) = \lfloor m(kA \mod 1) \rfloor$ with $0 < A < 1$.
- The value of m is not critical.
- Easy to implement in a computer.

Possible hash functions when the keys are natural numbers

The division method

- $h(k) = k \mod m$.
- Good choices for m are primes not too close to a power of 2.

The multiplication method

- $h(k) = \lfloor m(kA \mod 1) \rfloor$ with $0 < A < 1$.
- The value of m is not critical.
- Easy to implement in a computer.

When they are not, we need to interpreting the keys as natural numbers

Keys interpreted as natural numbers

Given a string "pt", we can say p = 112 and t=116 (ASCII numbers)

- ASCII has 128 possible symbols.
  - Then $(128 \times 112) + 128^0 \times 116 = 14452$

Nevertheless

This is highly dependent on the origins of the keys!!!

When they are not, we need to interpreting the keys as natural numbers

Keys interpreted as natural numbers

Given a string “pt”, we can say p = 112 and t=116 (ASCII numbers)

- ASCII has 128 possible symbols.
  - Then $(128 \times 112) + 128^0 \times 116 = 14452$

Nevertheless

This is highly dependent on the origins of the keys!!!

When they are not, we need to interpreting the keys as natural numbers

Keys interpreted as natural numbers

Given a string "pt", we can say p = 112 and t=116 (ASCII numbers)

- ASCII has 128 possible symbols.
  - Then $(128 \times 112) + 128^0 \times 116 = 14452$

Nevertheless

This is highly dependent on the origins of the keys!!!

When they are not, we need to interpreting the keys as natural numbers

Keys interpreted as natural numbers

Given a string "pt", we can say p = 112 and t=116 (ASCII numbers)

- ASCII has 128 possible symbols.
  - Then $(128 \times 112) + 128^0 \times 116 = 14452$

Nevertheless

This is highly dependent on the origins of the keys!!!

Outline

Hashing methods: The division method

Hash function

$h(k) = k \mod m$

Hashing methods: The division method

Hash function

$h(k) = k \mod m$

Problems with some selections

- $m = 2^p$, $h(k)$ is only the p lowest-order bits.
- $m = 2^p - 1$, when k is interpreted as a character string interpreted in radix 2^p, permuting characters in k does not change the value.

- For example, given $n = 2000$ elements
  - We can use $m = 701$ because it is near to $2000/3$ but not near a power of two.

Hashing methods: The division method

Hash function

$h(k) = k \mod m$

Problems with some selections

- $m = 2^p$, $h(k)$ is only the p lowest-order bits.
- $m = 2^p - 1$, when k is interpreted as a character string interpreted in radix 2^p, permuting characters in k does not change the value.

It is better to select

- Prime numbers not too close to an exact power of two.
  - For example, given $n = 2000$ elements.
    - We can use $m = 701$ because it is near to $2000/3$ but not near a power of two.

Hashing methods: The division method

Hash function

$h(k) = k \mod m$

Problems with some selections

- $m = 2^p$, $h(k)$ is only the p lowest-order bits.
- $m = 2^p - 1$, when k is interpreted as a character string interpreted in radix 2^p, permuting characters in k does not change the value.

It is better to select

- Prime numbers not too close to an exact power of two.
  - For example, given $n = 2000$ elements.
    - We can use $m = 701$ because it is near to $2000/3$ but not near a power of two.

Hashing methods: The division method

Hash function

$h(k) = k \mod m$

Problems with some selections

- $m = 2^p$, $h(k)$ is only the p lowest-order bits.
- $m = 2^p - 1$, when k is interpreted as a character string interpreted in radix 2^p, permuting characters in k does not change the value.

It is better to select

- Prime numbers not too close to an exact power of two.
  - For example, given $n = 2000$ elements.
    - We can use $m = 701$ because it is near to $2000/3$ but not near a power of two.

Hashing methods: The division method

Hash function

$h(k) = k \mod m$

Problems with some selections

- $m = 2^p$, $h(k)$ is only the p lowest-order bits.
- $m = 2^p - 1$, when k is interpreted as a character string interpreted in radix 2^p, permuting characters in k does not change the value.

It is better to select

- Prime numbers not too close to an exact power of two.
  - For example, given $n = 2000$ elements.
    - We can use $m = 701$ because it is near to $^{2000}/_3$ but not near a power of two.

Outline

Hashing methods: The multiplication method

The multiplication method for creating hash functions has two steps

1. Multiply the key k by a constant A in the range $0 < A < 1$ and extract the fractional part of kA.
2. Then, you multiply the value by m an take the floor, $h(k) = \lfloor m\,(kA \mod 1) \rfloor$.

Hashing methods: The multiplication method

The multiplication method for creating hash functions has two steps

1. Multiply the key k by a constant A in the range $0 < A < 1$ and extract the fractional part of kA.
2. Then, you multiply the value by m an take the floor, $h(k) = \lfloor m \,(kA \mod 1) \rfloor$.

The mod allows to extract that fractional part!!!

$kA \mod 1 = kA - \lfloor kA \rfloor$, $0 < A < 1$.

Hashing methods: The multiplication method

The multiplication method for creating hash functions has two steps

1. Multiply the key k by a constant A in the range $0 < A < 1$ and extract the fractional part of kA.
2. Then, you multiply the value by m an take the floor, $h(k) = \lfloor m\,(kA \mod 1) \rfloor$.

The mod allows to extract that fractional part!!!

$kA \mod 1 = kA - \lfloor kA \rfloor$, $0 < A < 1$.

Advantages:

m is not critical, normally $m = 2^p$.

Hashing methods: The multiplication method

The multiplication method for creating hash functions has two steps

1. Multiply the key k by a constant A in the range $0 < A < 1$ and extract the fractional part of kA.
2. Then, you multiply the value by m an take the floor, $h(k) = \lfloor m \left(kA \mod 1 \right) \rfloor$.

The mod allows to extract that fractional part!!!

$kA \mod 1 = kA - \lfloor kA \rfloor$, $0 < A < 1$.

Advantages:

m is not critical, normally $m = 2^p$.

Hashing methods: The multiplication method

The multiplication method for creating hash functions has two steps

1. Multiply the key k by a constant A in the range $0 < A < 1$ and extract the fractional part of kA.
2. Then, you multiply the value by m an take the floor, $h(k) = \lfloor m\,(kA \mod 1) \rfloor$.

The mod allows to extract that fractional part!!!

$kA \mod 1 = kA - \lfloor kA \rfloor$, $0 < A < 1$.

Advantages:

m is not critical, normally $m = 2^p$.

Implementing in a computer

First

First, imagine that the word in a machine has w **bits size** and k fits on those bits.

Second

Then, select an s in the range $0 < s < 2^w$ and assume $A = \frac{s}{2^w}$.

Third

Now, we multiply k by the number $s = A2^w$.

Implementing in a computer

First

First, imagine that the word in a machine has w **bits size** and k fits on those bits.

Second

Then, select an s in the range $0 < s < 2^w$ and assume $A = \frac{s}{2^w}$.

Third

Now, we multiply k by the number $s = A2^w$.

Implementing in a computer

First

First, imagine that the word in a machine has w **bits size** and k fits on those bits.

Second

Then, select an s in the range $0 < s < 2^w$ and assume $A = \frac{s}{2^w}$.

Third

Now, we multiply k by the number $s = A2^w$.

Example

Fourth

The result of that is $r_1 2^w + r_0$, a $2w$-bit value word, where the first p-most significative bits of r_0 are the desired hash value.

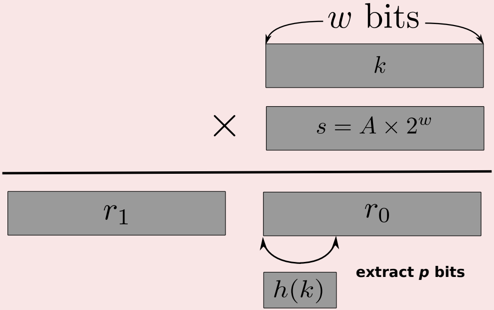
Graphically

Example

Fourth

The result of that is $r_1 2^w + r_0$, a $2w$-bit value word, where the first p-most significative bits of r_0 are the desired hash value.

Graphically



Outline

However

Sooner or Latter

We can pick up a hash function that does not give us the desired uniform randomized property

Thus

We are required to analyze the possible clustering of the data by the hash function

However

Sooner or Latter

We can pick up a hash function that does not give us the desired uniform randomized property

Thus

We are required to analyze the possible clustering of the data by the hash function

However

Unfortunately

Hash table do not give a way to measure clustering

Thus, table designers

They should provide some clustering estimation as part of the interface.

Thus

The clustering measure needs an estimate of the variance of the distribution of bucket sizes.

However

Unfortunately

Hash table do not give a way to measure clustering

Thus, table designers

They should provide some clustering estimation as part of the interface.

Thus

The clustering measure needs an estimate of the variance of the distribution of bucket sizes.

However

Unfortunately

Hash table do not give a way to measure clustering

Thus, table designers

They should provide some clustering estimation as part of the interface.

Thus

The clustering measure needs an estimate of the variance of the distribution of bucket sizes.

Measuring Clustering through a metric C

Definition

If bucket i contains n_i elements, then

$$C = \frac{m}{n-1}\left[\frac{\sum_{i=1}^{m} n_i^2}{n} - 1\right] \tag{2}$$

Measuring Clustering through a metric C

Definition

If bucket i contains n_i elements, then

$$C = \frac{m}{n-1}\left[\frac{\sum_{i=1}^{m} n_i^2}{n} - 1\right] \quad (2)$$

Properties

1. If $C = 1$, then you have uniform hashing.
2. If $C > 1$, it means that the performance of the hash table is slowed down by clustering by approximately a factor of C.
3. If $C < 1$, the spread of the elements is more even than uniform!!! Not going to happen!!!

Measuring Clustering through a metric C

Definition

If bucket i contains n_i elements, then

$$C = \frac{m}{n-1}\left[\frac{\sum_{i=1}^{m} n_i^2}{n} - 1\right] \tag{2}$$

Properties

1. If $C = 1$, then you have uniform hashing.
2. If $C > 1$, it means that the performance of the hash table is slowed down by clustering by approximately a factor of C.
3. If $C < 1$, the spread of the elements is more even than uniform!!! Not going to happen!!!

Measuring Clustering through a metric C

Definition

If bucket i contains n_i elements, then

$$C = \frac{m}{n-1}\left[\frac{\sum_{i=1}^{m} n_i^2}{n} - 1\right] \tag{2}$$

Properties

1. If $C = 1$, then you have uniform hashing.
2. If $C > 1$, it means that the performance of the hash table is slowed down by clustering by approximately a factor of C.
3. If $C < 1$, the spread of the elements is more even than uniform!!! Not going to happen!!!

Measuring Clustering through a metric C

Definition

If bucket i contains n_i elements, then

$$C = \frac{m}{n-1}\left[\frac{\sum_{i=1}^{m} n_i^2}{n} - 1\right] \quad (2)$$

Properties

1. If $C = 1$, then you have uniform hashing.
2. If $C > 1$, it means that the performance of the hash table is slowed down by clustering by approximately a factor of C.
3. If $C < 1$, the spread of the elements is more even than uniform!!! Not going to happen!!!

Thus

First

If clustering is occurring, some buckets will have more elements than they should, and some will have fewer.

Second

There will be a **wider range of bucket sizes** than one would expect from a random hash function.

Thus

First

If clustering is occurring, some buckets will have more elements than they should, and some will have fewer.

Second

There will be a **wider range of bucket sizes** than one would expect from a random hash function.

Outline

Analysis of C

Consider the following random variable

Consider bucket i containing n_i elements, with $X_{ij}= I\{\text{element } j \text{ lands in bucket } i\}$

Then, given

$$n_i = \sum_{j=1}^{n} X_{ij} \tag{3}$$

We have, given the unifrom hash property that

$$E\left[X_{ij}\right] = \frac{1}{m},\ E\left[X_{ij}^2\right] = \frac{1}{m} \tag{4}$$

Analysis of C

Consider the following random variable

Consider bucket i containing n_i elements, with $X_{ij} = I\{\text{element } j \text{ lands in bucket } i\}$

Then, given

$$n_i = \sum_{j=1}^{n} X_{ij} \tag{3}$$

We have, given the unifrom hash property that

$$E[X_{ij}] = \frac{1}{m},\ E\left[X_{ij}^2\right] = \frac{1}{m} \tag{4}$$

Analysis of C

Consider the following random variable

Consider bucket i containing n_i elements, with $X_{ij}= I\{$element j lands in bucket $i\}$

Then, given

$$n_i = \sum_{j=1}^{n} X_{ij} \tag{3}$$

We have, given the unifrom hash property that

$$E\left[X_{ij}\right] = \frac{1}{m},\ E\left[X_{ij}^2\right] = \frac{1}{m} \tag{4}$$

We look at the Variance of X_{ij}

We look at the dispersion of X_{ij}

$$Var\left[X_{ij}\right]=E\left[X_{ij}^{2}\right]-\left(E\left[X_{ij}\right]\right)^{2}=\frac{1}{m}-\frac{1}{m^{2}} \tag{5}$$

What about the expected number of elements at each bucket?

$$E\left[n_i\right]=E\left[\sum_{j=1}^{n}X_{ij}\right]=\frac{n}{m}=\alpha \tag{6}$$

We look at the Variance of X_{ij}

We look at the dispersion of X_{ij}

$$Var\left[X_{ij}\right] = E\left[X_{ij}^2\right] - \left(E\left[X_{ij}\right]\right)^2 = \frac{1}{m} - \frac{1}{m^2} \tag{5}$$

What about the expected number of elements at each bucket?

$$E\left[n_i\right] = E\left[\sum_{j=1}^{n} X_{ij}\right] = \frac{n}{m} = \alpha \tag{6}$$

Then, we have given independence of $\{X_{ij}\}$

Because independence of $\{X_{ij}\}$, the scattering of n_i

$$Var\left[n_i\right] = Var\left[\sum_{j=1}^{n} X_{ij}\right]$$

$$= \sum_{j=1}^{n} Var\left[X_{ij}\right]$$

$$= nVar\left[X_{ij}\right]$$

Then, we have given independence of $\{X_{ij}\}$

Because independence of $\{X_{ij}\}$, the scattering of n_i

$$
\begin{aligned}
Var[n_i] &= Var\left[\sum_{j=1}^{n} X_{ij}\right] \\
&= \sum_{j=1}^{n} Var[X_{ij}] \\
&= nVar[X_{ij}]
\end{aligned}
$$

Then, we have given independence of $\{X_{ij}\}$

Because independence of $\{X_{ij}\}$, the scattering of n_i

$$
\begin{aligned}
Var\left[n_i\right] &= Var\left[\sum_{j=1}^{n} X_{ij}\right] \\
&= \sum_{j=1}^{n} Var\left[X_{ij}\right] \\
&= nVar\left[X_{ij}\right]
\end{aligned}
$$

Then

What about the dispersion of possible number of elements at each bucket?

$$Var\left[n_i\right] = E\left[n_i^2\right] - \left(E\left[n_i\right]\right)^2$$

But, we have that

Then

What about the dispersion of possible number of elements at each bucket?

$$Var\left[n_i\right] = E\left[n_i^2\right] - \left(E\left[n_i\right]\right)^2$$

But, we have that

$$E\left[n_i^2\right] = E\left[\sum_{j=1}^{n} X_{ij}^2 + \sum_{j=1}^{n}\sum_{k=1,k\neq j}^{n} X_{ij}X_{ik}\right] \quad (7)$$

Or

Then

What about the dispersion of possible number of elements at each bucket?

$$Var\left[n_i\right] = E\left[n_i^2\right] - \left(E\left[n_i\right]\right)^2$$

But, we have that

$$E\left[n_i^2\right] = E\left[\sum_{j=1}^{n} X_{ij}^2 + \sum_{j=1}^{n}\sum_{k=1,k\neq j}^{n} X_{ij}X_{ik}\right] \tag{7}$$

Or

$$E\left[n_i^2\right] = \frac{n}{m} + \sum_{j=1}^{n}\sum_{k=1,k\neq j}^{n} \frac{1}{m^2} \tag{8}$$

Thus

We re-express the range on term of expected values of n_i

$$E\left[n_i^2\right] = \frac{n}{m} + \frac{n\left(n-1\right)}{m^2} \tag{9}$$

Thus

We re-express the range on term of expected values of n_i

$$E\left[n_i^2\right] = \frac{n}{m} + \frac{n\left(n-1\right)}{m^2} \tag{9}$$

Then

$$\begin{aligned} Var\left(n_i\right) = E\left[n_i^2\right] - E\left[n_i\right]^2 &= \frac{n}{m} + \frac{n\left(n-1\right)}{m^2} - \frac{n^2}{m^2} \\ &= \frac{n}{m} - \frac{n}{m^2} \\ &= \alpha - \frac{\alpha}{m} \end{aligned}$$

Thus

We re-express the range on term of expected values of n_i

$$E\left[n_i^2\right] = \frac{n}{m} + \frac{n\left(n-1\right)}{m^2} \tag{9}$$

Then

$$\begin{aligned} Var\left(n_i\right) = E\left[n_i^2\right] - E\left[n_i\right]^2 &= \frac{n}{m} + \frac{n\left(n-1\right)}{m^2} - \frac{n^2}{m^2} \\ &= \frac{n}{m} - \frac{n}{m^2} \\ &= \alpha - \frac{\alpha}{m} \end{aligned}$$

Thus

We re-express the range on term of expected values of n_i

$$E\left[n_i^2\right] = \frac{n}{m} + \frac{n\left(n-1\right)}{m^2} \quad (9)$$

Then

$$\begin{aligned} Var\left(n_i\right) = E\left[n_i^2\right] - E\left[n_i\right]^2 &= \frac{n}{m} + \frac{n\left(n-1\right)}{m^2} - \frac{n^2}{m^2} \\ &= \frac{n}{m} - \frac{n}{m^2} \\ &= \alpha - \frac{\alpha}{m} \end{aligned}$$

Then, we have that

Now we build an estimator of the mean of n_i^2 which is part of $C = \frac{m}{n-1}\left[\frac{\sum_{i=1}^{m} n_i^2}{n} - 1\right]$

$$\frac{1}{n}\sum_{i=1}^{m} n_i^2 \tag{10}$$

Thus, we ask what is the expected value of the mean of the variances

Then, we have that

Now we build an estimator of the mean of n_i^2 which is part of $C = \frac{m}{n-1}\left[\frac{\sum_{i=1}^{m} n_i^2}{n} - 1\right]$

$$\frac{1}{n}\sum_{i=1}^{m} n_i^2 \tag{10}$$

Thus, we ask what is the expected value of the mean of the variances

$$\begin{aligned} E\left[\frac{1}{n}\sum_{i=1}^{m} n_i^2\right] &= \frac{1}{n}\sum_{i=1}^{m} E\left[n_i^2\right] \\ &= \frac{m}{n}\left[\alpha\left(1-\frac{1}{m}\right)+\alpha^2\right] \\ &= \frac{1}{\alpha}\left[\alpha\left(1-\frac{1}{m}\right)+\alpha^2\right] \\ &= 1-\frac{1}{m}+\alpha \end{aligned}$$

Then, we have that

Now we build an estimator of the mean of n_i^2 which is part of $C = \frac{m}{n-1}\left[\frac{\sum_{i=1}^{m} n_i^2}{n} - 1\right]$

$$\frac{1}{n}\sum_{i=1}^{m} n_i^2 \tag{10}$$

Thus, we ask what is the expected value of the mean of the variances

$$\begin{aligned} E\left[\frac{1}{n}\sum_{i=1}^{m} n_i^2\right] &= \frac{1}{n}\sum_{i=1}^{m} E\left[n_i^2\right] \\ &= \frac{m}{n}\left[\alpha\left(1-\frac{1}{m}\right)+\alpha^2\right] \\ &= \frac{1}{\alpha}\left[\alpha\left(1-\frac{1}{m}\right)+\alpha^2\right] \\ &= 1-\frac{1}{m}+\alpha \end{aligned}$$

Then, we have that

Now we build an estimator of the mean of n_i^2 which is part of $C = \frac{m}{n-1}\left[\frac{\sum_{i=1}^{m} n_i^2}{n} - 1\right]$

$$\frac{1}{n}\sum_{i=1}^{m} n_i^2 \tag{10}$$

Thus, we ask what is the expected value of the mean of the variances

$$\begin{aligned}
E\left[\frac{1}{n}\sum_{i=1}^{m} n_i^2\right] &= \frac{1}{n}\sum_{i=1}^{m} E\left[n_i^2\right] \\
&= \frac{m}{n}\left[\alpha\left(1-\frac{1}{m}\right)+\alpha^2\right] \\
&= \frac{1}{\alpha}\left[\alpha\left(1-\frac{1}{m}\right)+\alpha^2\right] \\
&= 1-\frac{1}{m}+\alpha
\end{aligned}$$

Finally, we analyze the Expected Value of C under uniform hashing

We can plug back on C using the expected value

$$
\begin{aligned}
E\left[C\right] &= \frac{m}{n-1}\left[E\left[\frac{\sum_{i=1}^{m} n_i^2}{n}\right]-1\right]\\
&= \frac{m}{n-1}\left[1-\frac{1}{m}+\alpha-1\right]\\
&= \frac{m}{n-1}\left[\frac{n}{m}-\frac{1}{m}\right]\\
&= \frac{m}{n-1}\left[\frac{n-1}{m}\right]\\
&= 1
\end{aligned}
$$

Finally, we analyze the Expected Value of C under uniform hashing

We can plug back on C using the expected value

$$
\begin{aligned}
E[C] &= \frac{m}{n-1}\left[E\left[\frac{\sum_{i=1}^{m} n_i^2}{n}\right] - 1\right] \\
&= \frac{m}{n-1}\left[1 - \frac{1}{m} + \alpha - 1\right] \\
&= \frac{m}{n-1}\left[\frac{n}{m} - \frac{1}{m}\right] \\
&= \frac{m}{n-1}\left[\frac{n-1}{m}\right] \\
&= 1
\end{aligned}
$$

Finally, we analyze the Expected Value of C under uniform hashing

We can plug back on C using the expected value

$$
\begin{aligned}
E[C] &= \frac{m}{n-1}\left[E\left[\frac{\sum_{i=1}^{m} n_i^2}{n}\right] - 1\right] \\
&= \frac{m}{n-1}\left[1 - \frac{1}{m} + \alpha - 1\right] \\
&= \frac{m}{n-1}\left[\frac{n}{m} - \frac{1}{m}\right] \\
&= \frac{m}{n-1}\left[\frac{n-1}{m}\right] \\
&= 1
\end{aligned}
$$

Finally, we analyze the Expected Value of C under uniform hashing

We can plug back on C using the expected value

$$
\begin{aligned}
E\left[C\right] &= \frac{m}{n-1}\left[E\left[\frac{\sum_{i=1}^{m} n_i^2}{n}\right]-1\right] \\
&= \frac{m}{n-1}\left[1-\frac{1}{m}+\alpha-1\right] \\
&= \frac{m}{n-1}\left[\frac{n}{m}-\frac{1}{m}\right] \\
&= \frac{m}{n-1}\left[\frac{n-1}{m}\right] \\
&= 1
\end{aligned}
$$

cinvestav

Finally, we analyze the Expected Value of C under uniform hashing

We can plug back on C using the expected value

$$
\begin{aligned}
E\left[C\right] &= \frac{m}{n-1}\left[E\left[\frac{\sum_{i=1}^{m} n_i^2}{n}\right]-1\right]\\
&= \frac{m}{n-1}\left[1-\frac{1}{m}+\alpha-1\right]\\
&= \frac{m}{n-1}\left[\frac{n}{m}-\frac{1}{m}\right]\\
&= \frac{m}{n-1}\left[\frac{n-1}{m}\right]\\
&= 1
\end{aligned}
$$

Explanation

Using a hash table that enforce a uniform distribution in the buckets

- We get that $C = 1$ or the best distribution of keys

Outline

Now, we have a really horrible hash function $\equiv$ It hits only one of every b buckets

Thus

$$E\left[X_{ij}\right] = E\left[X_{ij}^2\right] = \frac{b}{m} \tag{11}$$

Thus, we have

$$E\left[n_i\right] = \alpha b \tag{12}$$

Then, we have

$$\begin{aligned} E\left[\frac{1}{n}\sum_{i=1}^{m} n_i^2\right] &= \frac{1}{n}\sum_{i=1}^{m} E\left[n_i^2\right] \\ &= \alpha b - \frac{b}{m} + 1 \end{aligned}$$

Now, we have a really horrible hash function $\equiv$ It hits only one of every b buckets

Thus

$$E\left[X_{ij}\right] = E\left[X_{ij}^2\right] = \frac{b}{m} \tag{11}$$

Thus, we have

$$E\left[n_i\right] = \alpha b \tag{12}$$

Then, we have

$$\begin{aligned} E\left[\frac{1}{n}\sum_{i=1}^{m} n_i^2\right] &= \frac{1}{n}\sum_{i=1}^{m} E\left[n_i^2\right] \\ &= \alpha b - \frac{b}{m} + 1 \end{aligned}$$

Now, we have a really horrible hash function $\equiv$ It hits only one of every b buckets

Thus

$$E\left[X_{ij}\right] = E\left[X_{ij}^2\right] = \frac{b}{m} \tag{11}$$

Thus, we have

$$E\left[n_i\right] = \alpha b \tag{12}$$

Then, we have

$$\begin{aligned} E\left[\frac{1}{n}\sum_{i=1}^{m} n_i^2\right] &= \frac{1}{n}\sum_{i=1}^{m} E\left[n_i^2\right] \\ &= \alpha b - \frac{b}{m} + 1 \end{aligned}$$

Finally

We can plug back on C using the expected value

$$\begin{aligned} E\left[C\right] &= \frac{m}{n-1}\left[E\left[\frac{\sum_{i=1}^{m} n_i^2}{n}\right] - 1\right] \\ &= \frac{m}{n-1}\left[\alpha b - \frac{b}{m} + 1 - 1\right] \\ &= \frac{m}{n-1}\left[\frac{nb}{m} - \frac{b}{m}\right] \\ &= \frac{m}{n-1}\left[\frac{b\left(n-1\right)}{m}\right] \\ &= b \end{aligned}$$

Finally

We can plug back on C using the expected value

$$
\begin{aligned}
E\left[C\right] &= \frac{m}{n-1}\left[E\left[\frac{\sum_{i=1}^{m} n_i^2}{n}\right]-1\right]\\
&= \frac{m}{n-1}\left[\alpha b-\frac{b}{m}+1-1\right]\\
&= \frac{m}{n-1}\left[\frac{nb}{m}-\frac{b}{m}\right]\\
&= \frac{m}{n-1}\left[\frac{b\left(n-1\right)}{m}\right]\\
&= b
\end{aligned}
$$

Finally

We can plug back on C using the expected value

$$
\begin{aligned}
E\left[C\right] &= \frac{m}{n-1}\left[E\left[\frac{\sum_{i=1}^{m} n_i^2}{n}\right]-1\right]\\
&= \frac{m}{n-1}\left[\alpha b-\frac{b}{m}+1-1\right]\\
&= \frac{m}{n-1}\left[\frac{nb}{m}-\frac{b}{m}\right]\\
&= \frac{m}{n-1}\left[\frac{b\left(n-1\right)}{m}\right]\\
&= b
\end{aligned}
$$

Finally

We can plug back on C using the expected value

$$
\begin{aligned}
E\left[C\right] &= \frac{m}{n-1}\left[E\left[\frac{\sum_{i=1}^{m} n_i^2}{n}\right]-1\right]\\
&= \frac{m}{n-1}\left[\alpha b-\frac{b}{m}+1-1\right]\\
&= \frac{m}{n-1}\left[\frac{nb}{m}-\frac{b}{m}\right]\\
&= \frac{m}{n-1}\left[\frac{b\left(n-1\right)}{m}\right]\\
&= b
\end{aligned}
$$

Finally

We can plug back on C using the expected value

$$
\begin{aligned}
E\left[C\right] &= \frac{m}{n-1}\left[E\left[\frac{\sum_{i=1}^{m} n_i^2}{n}\right] - 1\right] \\
&= \frac{m}{n-1}\left[\alpha b - \frac{b}{m} + 1 - 1\right] \\
&= \frac{m}{n-1}\left[\frac{nb}{m} - \frac{b}{m}\right] \\
&= \frac{m}{n-1}\left[\frac{b\left(n-1\right)}{m}\right] \\
&= b
\end{aligned}
$$

Explanation

Using a hash table that enforce a uniform distribution in the buckets

- We get that $C = b > 1$ or a really bad distribution of the keys!!!

Thus, you only need the following to evaluate a hash function

$$\frac{1}{n}\sum_{i=1}^{m} n_i^2 \tag{13}$$

Explanation

Using a hash table that enforce a uniform distribution in the buckets

- We get that $C = b > 1$ or a really bad distribution of the keys!!!

Thus, you only need the following to evaluate a hash function

$$\frac{1}{n}\sum_{i=1}^{m} n_i^2 \tag{13}$$

Outline

A Possible Solution, Universal Hashing

Issues

- In practice, keys are not randomly distributed.
- Any fixed hash function might yield retrieval $\Theta(n)$ time.

A Possible Solution, Universal Hashing

Issues

- In practice, keys are not randomly distributed.
- Any fixed hash function might yield retrieval $\Theta(n)$ time.

Goal

To find hash functions that produce uniform random table indexes irrespective of the keys.

A Possible Solution, Universal Hashing

Issues

- In practice, keys are not randomly distributed.
- Any fixed hash function might yield retrieval $\Theta(n)$ time.

Goal

To find hash functions that produce uniform random table indexes irrespective of the keys.

Idea

To select a hash function at random from a designed class of functions at the beginning of the execution.

A Possible Solution, Universal Hashing

Issues

- In practice, keys are not randomly distributed.
- Any fixed hash function might yield retrieval $\Theta(n)$ time.

Goal

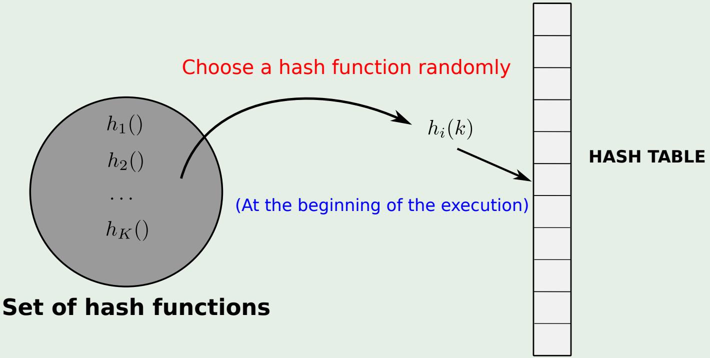
To find hash functions that produce uniform random table indexes irrespective of the keys.

Idea

To select a hash function at random from a designed class of functions at the beginning of the execution.

Universal hashing

Example



Outline

Definition of Universal Hash Functions

Definition

Let $H = \{h : U \to \{0, 1, ..., m-1\}\}$ be a family of hash functions. H is called a universal family if

$$\forall x, y \in U, x \neq y : \Pr_{h \in H}(h(x) = h(y)) \leq \frac{1}{m} \tag{14}$$

Main result

With universal hashing the chance of collision between distinct keys k and l is no more than the $\frac{1}{m}$ chance of collision if locations $h(k)$ and $h(l)$ were randomly and independently chosen from the set $\{0, 1, ..., m-1\}$.

Definition of Universal Hash Functions

Definition

Let $H = \{h : U \to \{0, 1, ..., m-1\}\}$ be a family of hash functions. H is called a universal family if

$$\forall x, y \in U, x \neq y : \underset{h \in H}{Pr}(h(x) = h(y)) \leq \frac{1}{m} \tag{14}$$

Main result

With universal hashing the chance of collision between distinct keys k and l is no more than the $\frac{1}{m}$ chance of collision if locations $h(k)$ and $h(l)$ were randomly and independently chosen from the set $\{0, 1, ..., m-1\}$.

Universal Hashing

Theorem 11.3

- Suppose that a hash function h is chosen randomly from a universal collection of hash functions and has been used to hash n keys into a table T of size m, using chaining to resolve collisions.
- If key k is not in the table, then the expected length $E[n_{h(k)}]$ of the list that key k hashes to is at most the load factor $\alpha = \frac{n}{m}$. If key k is in the table, then the expected length $E[n_{h(k)}]$ of the list containing key k is at most $1 + \alpha$.

Universal Hashing

Theorem 11.3

- Suppose that a hash function h is chosen randomly from a universal collection of hash functions and has been used to hash n keys into a table T of size m, using chaining to resolve collisions.
- If key k is not in the table, then the expected length $E[n_{h(k)}]$ of the list that key k hashes to is at most the load factor $\alpha = \frac{n}{m}$. If key k is in the table, then the expected length $E[n_{h(k)}]$ of the list containing key k is at most $1 + \alpha$.

Corollary 11.4

Using universal hashing and collision resolution by chaining in an initially empty table with m slots, it takes expected time $\Theta(n)$ to handle any sequence of n INSERT, SEARCH, and DELETE operations $O(m)$ INSERT operations.

Universal Hashing

Theorem 11.3

- Suppose that a hash function h is chosen randomly from a universal collection of hash functions and has been used to hash n keys into a table T of size m, using chaining to resolve collisions.
- If key k is not in the table, then the expected length $E[n_{h(k)}]$ of the list that key k hashes to is at most the load factor $\alpha = \frac{n}{m}$. If key k is in the table, then the expected length $E[n_{h(k)}]$ of the list containing key k is at most $1 + \alpha$.

Corollary 11.4

Using universal hashing and collision resolution by chaining in an initially empty table with m slots, it takes expected time $\Theta(n)$ to handle any sequence of n INSERT, SEARCH, and DELETE operations $O(m)$ INSERT operations.

Example of Universal Hash

Proceed as follows:

- Choose a primer number p large enough so that every possible key k is in the range $[0, ..., p-1]$

$$\mathbb{Z}_p = \{0, 1, ..., p-1\} \text{and } \mathbb{Z}_p^* = \{1, ..., p-1\}$$

- Define the following hash function:

$$h_{a,b}(k) = ((ak+b) \mod p) \mod m, \forall a \in Z_p^* \text{and } b \in Z_p$$

- The family of all such hash functions is:

$$H_{p,m} = \{h_{a,b} : a \in Z_p^* \text{and } b \in Z_p\}$$

Example of Universal Hash

Proceed as follows:

- Choose a primer number p large enough so that every possible key k is in the range $[0, ..., p-1]$

$$\mathbb{Z}_p = \{0, 1, ..., p-1\} \text{and } \mathbb{Z}_p^* = \{1, ..., p-1\}$$

- Define the following hash function:

$$h_{a,b}(k) = ((ak+b) \mod p) \mod m, \forall a \in Z_p^* \text{and } b \in Z_p$$

- The family of all such hash functions is:

$$H_{p,m} = \{h_{a,b} : a \in Z_p^* \text{and } b \in Z_p\}$$

Important

- a and b are chosen randomly at the beginning of execution.
- The class $H_{p,m}$ of hash functions is universal.

Example of Universal Hash

Proceed as follows:

- Choose a primer number p large enough so that every possible key k is in the range $[0, ..., p-1]$

$$\mathbb{Z}_p = \{0, 1, ..., p-1\} \text{and } \mathbb{Z}_p^* = \{1, ..., p-1\}$$

- Define the following hash function:

$$h_{a,b}(k) = ((ak+b) \mod p) \mod m, \forall a \in Z_p^* \text{and } b \in Z_p$$

- The family of all such hash functions is:

$$H_{p,m} = \{h_{a,b} : a \in Z_p^* \text{and } b \in Z_p\}$$

Important

- a and b are chosen randomly at the beginning of execution.
- The class $H_{p,m}$ of hash functions is universal.

Example of Universal Hash

Proceed as follows:

- Choose a primer number p large enough so that every possible key k is in the range $[0, ..., p-1]$

$$\mathbb{Z}_p = \{0, 1, ..., p-1\} \text{and } \mathbb{Z}_p^* = \{1, ..., p-1\}$$

- Define the following hash function:

$$h_{a,b}(k) = ((ak + b) \mod p) \mod m, \forall a \in Z_p^* \text{and } b \in Z_p$$

- The family of all such hash functions is:

$$H_{p,m} = \{h_{a,b} : a \in Z_p^* \text{and } b \in Z_p\}$$

Important

- a and b are chosen randomly at the beginning of execution.
- The class $H_{p,m}$ of hash functions is universal.

Example of Universal Hash

Proceed as follows:

- Choose a primer number p large enough so that every possible key k is in the range $[0, ..., p-1]$

$$\mathbb{Z}_p = \{0, 1, ..., p-1\} \text{and } \mathbb{Z}_p^* = \{1, ..., p-1\}$$

- Define the following hash function:

$$h_{a,b}(k) = ((ak + b) \mod p) \mod m, \forall a \in Z_p^* \text{and } b \in Z_p$$

- The family of all such hash functions is:

$$H_{p,m} = \{h_{a,b} : a \in Z_p^* \text{and } b \in Z_p\}$$

Important

- a and b are chosen randomly at the beginning of execution.
- The class $H_{p,m}$ of hash functions is universal.

Example of Universal Hash

Proceed as follows:

- Choose a primer number p large enough so that every possible key k is in the range $[0, ..., p-1]$

$$\mathbb{Z}_p = \{0, 1, ..., p-1\} \text{and } \mathbb{Z}_p^* = \{1, ..., p-1\}$$

- Define the following hash function:

$$h_{a,b}(k) = ((ak+b) \mod p) \mod m, \forall a \in Z_p^* \text{and } b \in Z_p$$

- The family of all such hash functions is:

$$H_{p,m} = \{h_{a,b} : a \in Z_p^* \text{and } b \in Z_p\}$$

Important

- a and b are chosen randomly at the beginning of execution.
- The class $H_{p,m}$ of hash functions is universal.

Example of Universal Hash

Proceed as follows:

- Choose a primer number p large enough so that every possible key k is in the range $[0, ..., p-1]$

$$\mathbb{Z}_p = \{0, 1, ..., p-1\} \text{and } \mathbb{Z}_p^* = \{1, ..., p-1\}$$

- Define the following hash function:

$$h_{a,b}(k) = ((ak+b) \mod p) \mod m, \forall a \in Z_p^* \text{and } b \in Z_p$$

- The family of all such hash functions is:

$$H_{p,m} = \{h_{a,b} : a \in Z_p^* \text{and } b \in Z_p\}$$

Important

- a and b are chosen randomly at the beginning of execution.
- The class $H_{p,m}$ of hash functions is universal.

Example of Universal Hash

Proceed as follows:

- Choose a primer number p large enough so that every possible key k is in the range $[0, ..., p-1]$

$$\mathbb{Z}_p = \{0, 1, ..., p-1\} \text{and } \mathbb{Z}_p^* = \{1, ..., p-1\}$$

- Define the following hash function:

$$h_{a,b}(k) = ((ak+b) \mod p) \mod m, \forall a \in Z_p^* \text{and } b \in Z_p$$

- The family of all such hash functions is:

$$H_{p,m} = \{h_{a,b} : a \in Z_p^* \text{and } b \in Z_p\}$$

Important

- a and b are chosen randomly at the beginning of execution.
- The class $H_{p,m}$ of hash functions is universal.

Outline

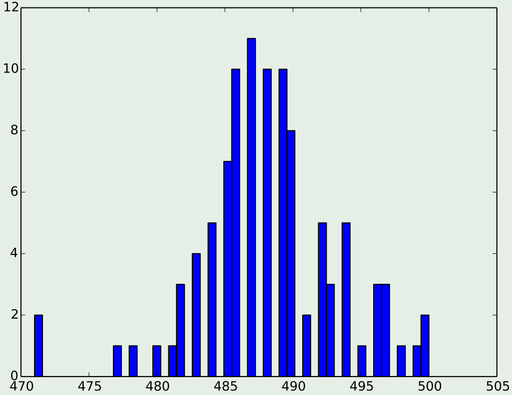
Example

Example

- $p = 977$, $m = 50$, a and b random numbers
  - $h_{a,b}(k) = ((ak + b) \mod p) \mod m$

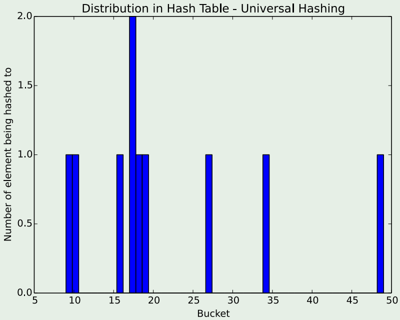
Example of key distribution

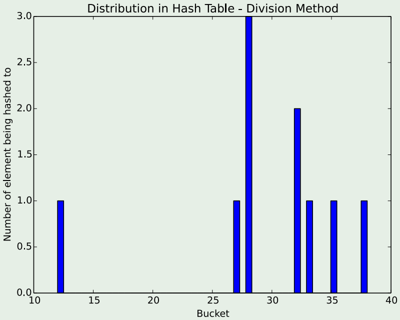
Example, mean = 488.5 and dispersion = 5



Example with 10 keys

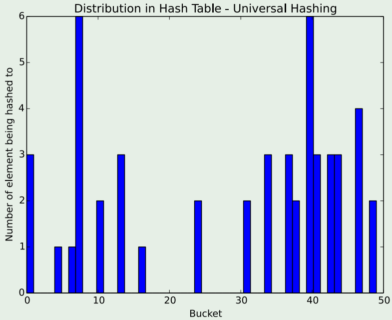
Universal Hashing Vs Division Method

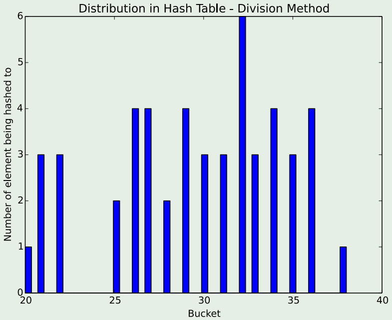




Example with 50 keys

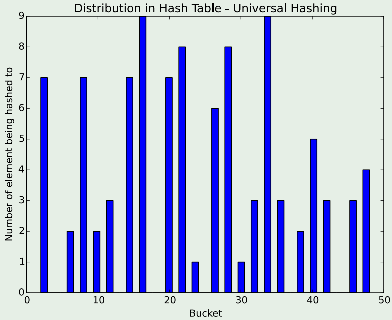
Universal Hashing Vs Division Method

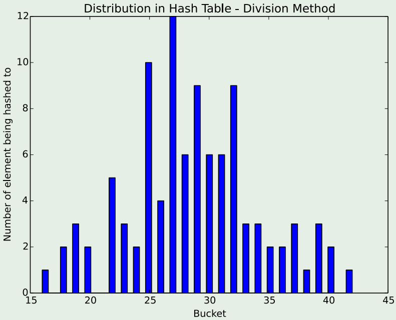




Example with 100 keys

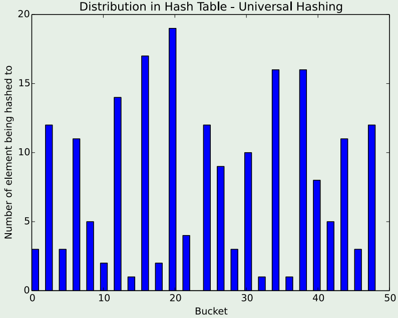
Universal Hashing Vs Division Method

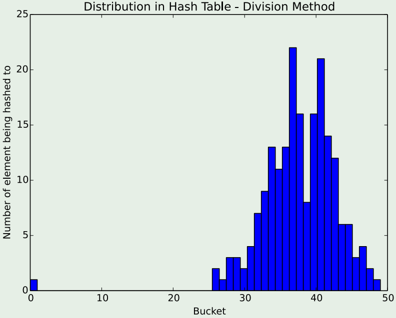




Example with 200 keys

An example of $P(\Theta|X) = P(X|\Theta) P(\Theta)$





Another Example: Matrix Method

Then

- Let us say keys are u-bits long.
- Say the table size M is power of 2.
- an index is b-bits long with $M = 2^b$.

Another Example: Matrix Method

Then

- Let us say keys are u-bits long.
- Say the table size M is power of 2.
- an index is b-bits long with $M = 2^b$.

The h function

- Pick h to be a random b-by-u 0/1 matrix, and define $h(x) = hx$ where after the inner product we apply mod 2

Another Example: Matrix Method

Then

- Let us say keys are u-bits long.
- Say the table size M is power of 2.
- an index is b-bits long with $M = 2^b$.

The h function

- Pick h to be a random b-by-u 0/1 matrix, and define $h(x) = hx$ where after the inner product we apply mod 2

Example

$$
\begin{array}{cc}
 & h \\
b & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \\
 & u
\end{array}
\begin{array}{c}
x \\
\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}
\end{array}
=
\begin{array}{c}
h(x) \\
\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}
\end{array}
$$

Another Example: Matrix Method

Then

- Let us say keys are u-bits long.
- Say the table size M is power of 2.
- an index is b-bits long with $M = 2^b$.

The h function

- Pick h to be a random b-by-u 0/1 matrix, and define $h(x) = hx$ where after the inner product we apply mod 2

Example

$$\begin{array}{cc} & h \\ b & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \\ & u \end{array} \begin{array}{c} x \\ \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \end{array} = \begin{array}{c} h(x) \\ \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \end{array}$$

Another Example: Matrix Method

Then

- Let us say keys are u-bits long.
- Say the table size M is power of 2.
- an index is b-bits long with $M = 2^b$.

The h function

- Pick h to be a random b-by-u 0/1 matrix, and define $h(x) = hx$ where after the inner product we apply mod 2

Example

$$
\begin{array}{cc} & h \\ b & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \\ & u \end{array}
\begin{array}{c} x \\ \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \end{array}
=
\begin{array}{c} h(x) \\ \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \end{array}
$$

First than anything

What is the meaning of multiply h by x

- We can think of it as adding some of the columns of h where the 1 bits in indicate which to add

First than anything

What is the meaning of multiply h by x

- We can think of it as adding some of the columns of h where the 1 bits in indicate which to add

- Without loosing generality assume the following
  1. $l_i \neq m_i \Rightarrow$ for example $l_i = 0$ and $m_i = 1$
  2. $l_j = m_j \ \forall j \neq i$

First than anything

What is the meaning of multiply h by x

- We can think of it as adding some of the columns of h where the 1 bits in indicate which to add

- Without loosing generality assume the following
  1. $l_i \neq m_i \Rightarrow$ for example $l_i = 0$ and $m_i = 1$
  2. $l_j = m_j \; \forall j \neq i$

Now Proof of being a Universal Family

Thus

- The column i does not contribute to the final answer of $h(l)$ because of the zero!!!

$$
\begin{array}{cc}
 & \begin{array}{cccc} & h & & \end{array} \\
b & \begin{bmatrix} 1 & \mathbf{0} & 0 & 0 \\ 0 & \mathbf{1} & 1 & 1 \\ 1 & \mathbf{1} & 1 & 0 \end{bmatrix} \\
 & \begin{array}{cccc} & u & & \end{array}
\end{array}
\begin{array}{c} x \\ \begin{bmatrix} 1 \\ \mathbf{0} \\ 1 \\ 0 \end{bmatrix} \end{array}
=
\begin{array}{c} h(x) \\ \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \end{array}
$$

Now

- Imagine that we fix all the other columns in h, and we allow the i^{th} column you have free choices

Now Proof of being a Universal Family

Thus

- The column i does not contribute to the final answer of $h(l)$ because of the zero!!!

$$\begin{array}{cc} & h \\ b & \begin{bmatrix} 1 & \mathbf{0} & 0 & 0 \\ 0 & \mathbf{1} & 1 & 1 \\ 1 & \mathbf{1} & 1 & 0 \end{bmatrix} \\ & u \end{array} \begin{array}{c} x \\ \begin{bmatrix} 1 \\ \mathbf{0} \\ 1 \\ 0 \end{bmatrix} \end{array} = \begin{array}{c} h(x) \\ \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \end{array}$$

Now

- Imagine that we fix all the other columns in h, and we allow the i^{th} column you have free choices

Now, we do something strange

But having $x_i = 0$ make $h(x)$ to have a fix value, for example

$$\begin{bmatrix} 1 & 0 & \mathbf{0} & 0 \\ 0 & 1 & \mathbf{1} & 1 \\ 1 & 1 & \mathbf{1} & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \mathbf{0} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

In the contrary, we have y and with respect to a specific flipping column of h

$$\begin{bmatrix} 1 & 0 & \mathbf{0} & 0 \\ 0 & 1 & \mathbf{1} & 1 \\ 1 & 1 & \mathbf{1} & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \mathbf{1} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

Now, we do something strange

But having $x_i = 0$ make $h(x)$ to have a fix value, for example

$$\begin{bmatrix} 1 & 0 & \mathbf{0} & 0 \\ 0 & 1 & \mathbf{1} & 1 \\ 1 & 1 & \mathbf{1} & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \mathbf{0} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

In the contrary, we have y and with respect to a specific flipping column of h

$$\begin{bmatrix} 1 & 0 & \mathbf{0} & 0 \\ 0 & 1 & \mathbf{1} & 1 \\ 1 & 1 & \mathbf{1} & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \mathbf{1} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

We have others

For example

$$\begin{bmatrix} 1 & 0 & \mathbf{1} & 0 \\ 0 & 1 & \mathbf{0} & 1 \\ 1 & 1 & \mathbf{1} & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \mathbf{1} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

How many of them, when flipping on the i^{th} column

$$2^b$$

We have others

For example

$$\begin{bmatrix} 1 & 0 & \mathbf{1} & 0 \\ 0 & 1 & \mathbf{0} & 1 \\ 1 & 1 & \mathbf{1} & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \mathbf{1} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

How many of them, when flipping on the i^{th} column

$$2^b$$

Even the one that looks like

We have

$$\begin{bmatrix} 1 & 0 & \mathbf{0} & 0 \\ 0 & 1 & \mathbf{0} & 1 \\ 1 & 1 & \mathbf{1} & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \mathbf{1} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

What is the probability of getting the same values i.e.

$$h(l) = h(m)$$

Even the one that looks like

We have

$$\begin{bmatrix} 1 & 0 & \mathbf{0} & 0 \\ 0 & 1 & \mathbf{0} & 1 \\ 1 & 1 & \mathbf{1} & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \mathbf{1} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

What is the probability of getting the same values i.e.

$$h(l) = h(m)$$

Quite easy

Thus, given the randomness of the zeros and ones

- The probability that we get equality is

$$P\left(h\left(l\right) = h\left(m\right)\right) = \frac{1}{2^b}$$

Or more formally

$$P\left(h\left(l\right) = h\left(m\right)\right) \leq \frac{1}{2^b}$$

Quite easy

Thus, given the randomness of the zeros and ones

- The probability that we get equality is

$$P\left(h\left(l\right)=h\left(m\right)\right)=\frac{1}{2^{b}}$$

Or more formally

$$P\left(h\left(l\right)=h\left(m\right)\right)\leq\frac{1}{2^{b}}$$

Implementation of the column*vector mod 2

Code

```
int product(int row, int vector){

  int i = row & vector;

  i = i - ((i >> 1) & 0x55555555);
  i = (i & 0x33333333) + ((i >> 2) & 0x33333333);
  i = (((i + (i >> 4)) & 0x0F0F0F0F) * 0x01010101) >> 24;

  return i & i & 0x00000001;

}
```

Advantages of universal hashing

Advantages

- Universal hashing provides good results on average, independently of the keys to be stored.
- Guarantees that no input will always elicit the worst-case behavior.
- Poor performance occurs only when the random choice returns an inefficient hash function; this has a small probability.

Advantages of universal hashing

Advantages

- Universal hashing provides good results on average, independently of the keys to be stored.
- Guarantees that no input will always elicit the worst-case behavior.
- Poor performance occurs only when the random choice returns an inefficient hash function; this has a small probability.

Advantages of universal hashing

Advantages

- Universal hashing provides good results on average, independently of the keys to be stored.
- Guarantees that no input will always elicit the worst-case behavior.
- Poor performance occurs only when the random choice returns an inefficient hash function; this has a small probability.

Outline

Open addressing

Definition

All the elements occupy the hash table itself.

What is it?

We systematically examine table slots until either we find the desired element or we have ascertained that the element is not in the table.

Advantages

The advantage of open addressing is that it avoids pointers altogether.

Open addressing

Definition

All the elements occupy the hash table itself.

What is it?

We systematically examine table slots until either we find the desired element or we have ascertained that the element is not in the table.

Advantages

The advantage of open addressing is that it avoids pointers altogether.

Open addressing

Definition

All the elements occupy the hash table itself.

What is it?

We systematically examine table slots until either we find the desired element or we have ascertained that the element is not in the table.

Advantages

The advantage of open addressing is that it avoids pointers altogether.

Insert in Open addressing

Extended hash function to **probe**

- Instead of being fixed in the order $0, 1, 2, ..., m-1$ with $\Theta(n)$ search time.
- Extend the hash function to $h : U \times \{0, 1, ..., m-1\} \to \{0, 1, ..., m-1\}$.
- This gives the probe sequence $\langle h(k, 0), h(k, 1), ..., h(k, m-1) \rangle$.
  - A permutation of $\langle 0, 1, 2, ..., m-1 \rangle$

Insert in Open addressing

Extended hash function to **probe**

- Instead of being fixed in the order $0, 1, 2, ..., m-1$ with $\Theta(n)$ search time.
- Extend the hash function to
  $h : U \times \{0, 1, ..., m-1\} \rightarrow \{0, 1, ..., m-1\}$.
- This gives the probe sequence $\langle h(k,0), h(k,1), ..., h(k, m-1) \rangle$.
  - A permutation of $\langle 0, 1, 2, ..., m-1 \rangle$

Insert in Open addressing

Extended hash function to **probe**

- Instead of being fixed in the order $0, 1, 2, ..., m-1$ with $\Theta(n)$ search time.
- Extend the hash function to $h : U \times \{0, 1, ..., m-1\} \rightarrow \{0, 1, ..., m-1\}$.
- This gives the probe sequence $\langle h(k,0), h(k,1), ..., h(k,m-1) \rangle$.
  - A permutation of $\langle 0, 1, 2, ..., m-1 \rangle$

Insert in Open addressing

Extended hash function to **probe**

- Instead of being fixed in the order $0, 1, 2, ..., m-1$ with $\Theta(n)$ search time.
- Extend the hash function to $h : U \times \{0, 1, ..., m-1\} \to \{0, 1, ..., m-1\}$.
- This gives the probe sequence $\langle h(k, 0), h(k, 1), ..., h(k, m-1)\rangle$.
  - A permutation of $\langle 0, 1, 2, ..., m-1\rangle$

Outline

Hashing methods in Open Addressing

HASH-INSERT(T, k)

1. $i = 0$
2. **repeat**
3. $\quad j = h(k, i)$
4. $\quad$ **if** $T[j] == NIL$
5. $\quad\quad T[j] = k$
6. $\quad\quad$ **return** j
7. $\quad$ **else** $i = i + 1$
8. **until** $i == m$
9. **error** "Hash Table Overflow"

Hashing methods in Open Addressing

HASH-INSERT(T, k)

1. $i = 0$
2. **repeat**
3. $\quad j = h(k, i)$
4. $\quad$ **if** $T[j] == NIL$
5. $\quad\quad T[j] = k$
6. $\quad\quad$ **return** j
7. $\quad$ **else** $i = i + 1$
8. **until** $i == m$
9. **error** "Hash Table Overflow"

Hashing methods in Open Addressing

HASH-INSERT(T, k)

1. $i = 0$
2. **repeat**
3. $\quad j = h(k, i)$
4. $\quad$ **if** $T[j] == NIL$
5. $\quad\quad T[j] = k$
6. $\quad\quad$ **return** j
7. $\quad$ **else** $i = i + 1$
8. **until** $i == m$
9. **error** "Hash Table Overflow"

Hashing methods in Open Addressing

HASH-INSERT(T, k)

$i = 0$

repeat

  $j = h(k, i)$

  if $T[j] == NIL$

    $T[j] = k$

    return j

  else $i = i + 1$

until $i == m$

error "Hash Table Overflow"

Hashing methods in Open Addressing

HASH-INSERT(T, k)

$i = 0$

repeat

    $j = h(k, i)$

    if $T[j] == NIL$

        $T[j] = k$

        return j

    else $i = i + 1$

until $i == m$

error "Hash Table Overflow"

Hashing methods in Open Addressing

HASH-INSERT(T, k)

$i = 0$
repeat
    $j = h(k, i)$
    if $T[j] == NIL$
        $T[j] = k$
        return j
    else $i = i + 1$
until $i == m$

error "Hash Table Overflow"

Hashing methods in Open Addressing

HASH-INSERT(T, k)

```
i = 0
repeat
        j = h(k, i)
        if T[j] == NIL
                T[j] = k
                return j
        else i = i + 1
until i == m
error "Hash Table Overflow"
```

Hashing methods in Open Addressing

HASH-INSERT(T, k)

```
i = 0
repeat
        j = h(k, i)
        if T[j] == NIL
                T[j] = k
                return j
        else i = i + 1
until i == m
error "Hash Table Overflow"
```

Hashing methods in Open Addressing

HASH-INSERT(T, k)

$i = 0$

repeat

    $j = h(k, i)$

    if $T[j] == NIL$

        $T[j] = k$

        return j

    else $i = i + 1$

until $i == m$

error "Hash Table Overflow"

Hashing methods in Open Addressing

HASH-SEARCH(T,k)

1. $i = 0$
2. **repeat**
3. $\quad j = h(k, i)$
4. $\quad$ **if** $T[j] == k$
5. $\quad\quad$ **return** j
6. $\quad i = i + 1$
7. **until** $T[j] == NIL$ or $i == m$
8. **return** NIL

Hashing methods in Open Addressing

HASH-SEARCH(T,k)

1. $i = 0$
2. **repeat**
3. $\quad j = h(k, i)$
4. $\quad$ **if** $T[j] == k$
5. $\qquad$ **return** j
6. $\quad i = i + 1$
7. **until** $T[j] == NIL$ or $i == m$
8. **return** NIL

Hashing methods in Open Addressing

HASH-SEARCH(T,k)

1. $i = 0$
2. **repeat**
3. $\qquad j = h(k, i)$
4. $\qquad$ **if** $T[j] == k$
5. $\qquad\qquad$ **return** j
6. $\qquad i = i + 1$
7. **until** $T[j] == NIL$ or $i == m$
8. **return** NIL

Hashing methods in Open Addressing

HASH-SEARCH(T,k)

1. $i = 0$
2. **repeat**
3. $\quad j = h(k, i)$
4. $\quad$ **if** $T[j] == k$
5. $\qquad$ **return** j
6. $\quad i = i + 1$
7. **until** $T[j] == NIL$ or $i == m$
8. **return** NIL

Hashing methods in Open Addressing

HASH-SEARCH(T,k)

$i = 0$
repeat
  $j = h(k, i)$
  if $T[j] == k$
    return j
  $i = i + 1$
until $T[j] == NIL$ or $i == m$
return NIL

Hashing methods in Open Addressing

HASH-SEARCH(T,k)

$i = 0$

repeat

  $j = h(k, i)$

  if $T[j] == k$

    return j

  $i = i + 1$

until $T[j] == NIL$ or $i == m$

return NIL

Hashing methods in Open Addressing

HASH-SEARCH(T,k)

```
i = 0
repeat
    j = h(k, i)
    if T[j] == k
        return j
    i = i + 1
until T[j] == NIL or i == m
return NIL
```

Hashing methods in Open Addressing

HASH-SEARCH(T,k)

```
i = 0
repeat
    j = h(k, i)
    if T[j] == k
        return j
    i = i + 1
until T[j] == NIL or i == m
return NIL
```

Outline

Linear probing: Definition and properties

Hash function

- Given an ordinary hash function $h' : U \rightarrow \{0, 1, ..., m-1\}$ for $i = 0, 1, ..., m-1$, we get the extended hash function

$$h(k, i) = (h'(k) + i) \mod m, \tag{15}$$

Sequence of probes

Given key k, we first probe $T[h'(k)]$, then $T[h'(k)+1]$ and so on until $T[m-1]$. Then, we wrap around $T[0]$ to $T[h'(k)-1]$.

Distinct probes

Because the initial probe determines the entire probe sequence, there are m distinct probe sequences.

Linear probing: Definition and properties

Hash function

- Given an ordinary hash function $h' : U \rightarrow \{0, 1, ..., m-1\}$ for $i = 0, 1, ..., m-1$, we get the extended hash function

$$h(k, i) = (h'(k) + i) \mod m, \tag{15}$$

Sequence of probes

Given key k, we first probe $T[h'(k)]$, then $T[h'(k)+1]$ and so on until $T[m-1]$. Then, we wrap around $T[0]$ to $T[h'(k)-1]$.

Distinct probes

Because the initial probe determines the entire probe sequence, there are m distinct probe sequences.

Linear probing: Definition and properties

Hash function

- Given an ordinary hash function $h' : U \rightarrow \{0, 1, ..., m-1\}$ for $i = 0, 1, ..., m-1$, we get the extended hash function

$$h(k,i) = (h'(k) + i) \mod m, \tag{15}$$

Sequence of probes

Given key k, we first probe $T[h'(k)]$, then $T[h'(k)+1]$ and so on until $T[m-1]$. Then, we wrap around $T[0]$ to $T[h'(k)-1]$.

Distinct probes

Because the initial probe determines the entire probe sequence, there are m distinct probe sequences.

Linear probing: Definition and properties

Disadvantages

- Linear probing suffers of primary clustering.
- Long runs of occupied slots build up increasing the average search time.
- Long runs of occupied slots tend to get longer, and the average search time increases.

Linear probing: Definition and properties

Disadvantages

- Linear probing suffers of primary clustering.
- Long runs of occupied slots build up increasing the average search time.
- Long runs of occupied slots tend to get longer, and the average search time increases.

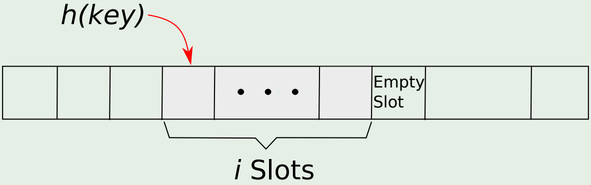
Linear probing: Definition and properties

Disadvantages

- Linear probing suffers of primary clustering.
- Long runs of occupied slots build up increasing the average search time.
- Long runs of occupied slots tend to get longer, and the average search time increases.

Why?

Clusters arise because an empty slot preceded by i full slots gets filled next with probability $\frac{i+1}{m}$.

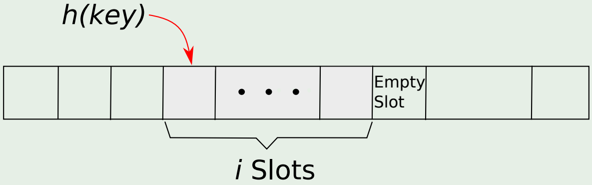


Thus

The probability of getting a collision increases dramatically after each insertion.

Why?

Clusters arise because an empty slot preceded by i full slots gets filled next with probability $\frac{i+1}{m}$.



Thus

The probability of getting a collision increases dramatically after each insertion.

Example

Example using keys uniformly distributed

It was generated using the division method

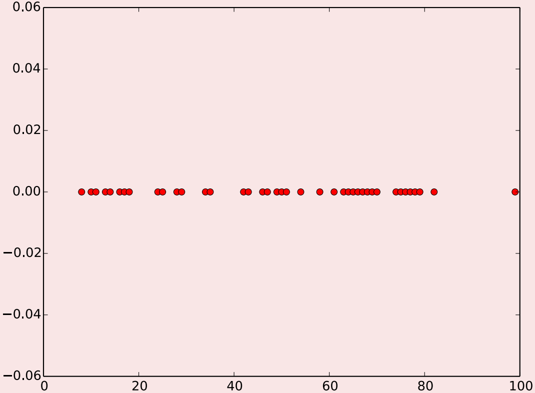
Then

Example

Example using keys uniformly distributed

It was generated using the division method

Then



Example

Example using Gaussian keys

It was generated using the division method

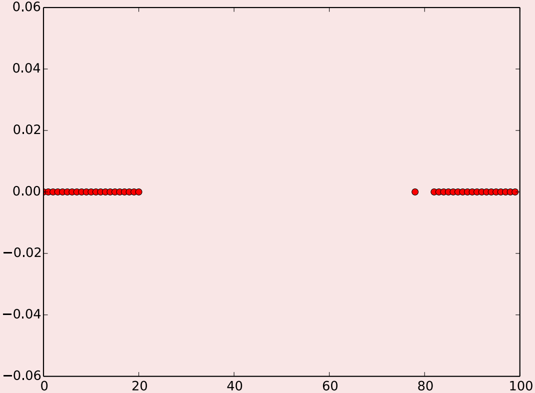
Then

Example

Example using Gaussian keys

It was generated using the division method

Then



Outline

Linear Probing, Insertion and Deletion

Constraints

- Divisor $= m$ (number of buckets) $= 17$.
- Home bucket $=$ key % 17.

Then

Put in pairs whose keys are 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45

We have

Linear Probing, Insertion and Deletion

Constraints

- Divisor $= m$ (number of buckets) $= 17$.
- Home bucket $=$ key % 17.

Then

Put in pairs whose keys are 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45

We have

Linear Probing, Insertion and Deletion

Constraints

- Divisor $= m$ (number of buckets) $= 17$.
- Home bucket $=$ key % 17.

Then

Put in pairs whose keys are 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45

We have

0				4				8				12				16
34	0	45				6	23	7			28	12	29	11	30	33

Outline

Linear Probing – Remove

Example

0				4				8				12				16
34	0	45				6	23	7			28	12	29	11	30	33

remove(0)

Compact Cluster

Search cluster for pair (if any) to fill vacated bucket.

Linear Probing – Remove

Example

0				4				8				12				16
34	0	45				6	23	7			28	12	29	11	30	33

remove(0)

0				4				8				12				16
34		45				6	23	7			28	12	29	11	30	33

Compact Cluster

Search cluster for pair (if any) to fill vacated bucket.

Linear Probing – Remove

Example

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
34	0	45				6	23	7			28	12	29	11	30	33

remove(0)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
34		45				6	23	7			28	12	29	11	30	33

Compact Cluster

Search cluster for pair (if any) to fill vacated bucket.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
34	45					6	23	7			28	12	29	11	30	33

Linear Probing – remove(34)

Example

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
34	0	45				6	23	7			28	12	29	11	30	33

remove(34)

Compact Cluster

Search cluster for pair (if any) to fill vacated bucket.

Linear Probing – remove(34)

Example

0				4				8				12				16
34	0	45				6	23	7			28	12	29	11	30	33

remove(34)

0				4				8				12				16
	0	45				6	23	7			28	12	29	11	30	33

Compact Cluster

Search cluster for pair (if any) to fill vacated bucket.

Linear Probing – remove(34)

Example

0				4				8				12				16
34	0	45				6	23	7			28	12	29	11	30	33

remove(34)

0				4				8				12				16
	0	45				6	23	7			28	12	29	11	30	33

Compact Cluster

Search cluster for pair (if any) to fill vacated bucket.

0				4				8				12				16
0		45				6	23	7			28	12	29	11	30	33

0				4				8				12				16
0	45					6	23	7			28	12	29	11	30	33

Linear Probing – remove(29)

Example

0				4				8				12				16
34	0	45				6	23	7			28	12	29	11	30	33

0				4				8				12				16
34	0	45				6	23	7			28	12		11	30	33

Compact Cluster

Search cluster for pair (if any) to fill vacated bucket.

Linear Probing – remove(29)

Example

0				4				8				12				16
34	0	45				6	23	7			28	12	29	11	30	33

0				4				8				12				16
34	0	45				6	23	7			28	12		11	30	33

Compact Cluster

Search cluster for pair (if any) to fill vacated bucket.

0				4				8				12				16
34	0	45				6	23	7			28	12	11		30	33

0				4				8				12				16
34	0	45				6	23	7			28	12	11	30		33

0				4				8				12				16
34	0					6	23	7			28	12	11	30	45	33

Code for Removing

We have the following

```
public void remove(key){
    int positionsChecked = 1;
    int i = FindSlot(Key);
    if (Table[i] == null)
        return;   // key is not in the table
    j = i;
    while(positionsChecked <= Table.length){
        j = (j+1) % Table.length;
        if (Table[j] == null) break;
        k = Hashing(Table[j].key);
        if (i < j && (k <= i || k > j)) ||
           (j < i && (k <= i && k > j)) {
                Table[i] = Table[j];
                i = j;
        }
        positionChecked++;
    }
    Table[i] = null;
}
```

Explanation

First

For all records in a cluster, there must be no vacant slots between their natural hash position and their current position (else lookups will terminate before finding the record).

Second

- k is the raw hash where the record at j would naturally land in the hash table if there were no collisions.

Thus

This test is asking if the record at j is invalidly positioned with respect to the required properties of a cluster now that i is vacant.

Explanation

First

For all records in a cluster, there must be no vacant slots between their natural hash position and their current position (else lookups will terminate before finding the record).

Second

- k is the raw hash where the record at j would naturally land in the hash table if there were no collisions.

Thus

This test is asking if the record at j is invalidly positioned with respect to the required properties of a cluster now that i is vacant.

Explanation

First

For all records in a cluster, there must be no vacant slots between their natural hash position and their current position (else lookups will terminate before finding the record).

Second

- k is the raw hash where the record at j would naturally land in the hash table if there were no collisions.

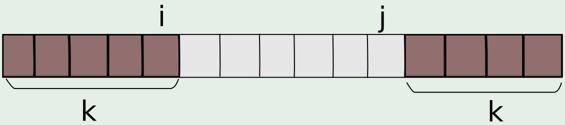
Thus

This test is asking if the record at j is invalidly positioned with respect to the required properties of a cluster now that i is vacant.

Case 1

We have the following

Case 1



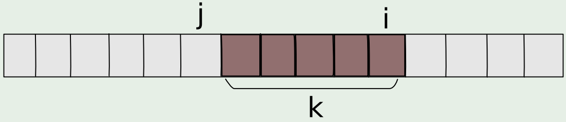
we have $i < j$

- If $i < k \leq j$ then moving j to the i position will be incorrect... Why?

Case 2

We have the following

Case 2



We have $j < i$

- If $k \leq j <$ or $i < k$ then moving j to the i position will be incorrect... Why?

Outline

Quadratic probing: Definition and properties

Hash function

- Given an auxiliary hash function $h' : U \rightarrow \{0, 1, ..., m-1\}$ for $i = 0, 1, ..., m-1$, we get the extended hash function

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \mod m, \tag{16}$$

where c_1, c_2 are auxiliary constants

Quadratic probing: Definition and properties

Hash function

- Given an auxiliary hash function $h' : U \to \{0, 1, ..., m-1\}$ for $i = 0, 1, ..., m-1$, we get the extended hash function

$$h(k, i) = \left(h'(k) + c_1 i + c_2 i^2\right) \mod m, \tag{16}$$

where c_1, c_2 are auxiliary constants

Sequence of probes

- Given key k, we first probe $T[h'(k)]$, later positions probed are offset by amounts that depend in a quadratic manner on the probe number i.
- The initial probe determines the entire sequence, and so only m distinct probe sequences are used.

Quadratic probing: Definition and properties

Hash function

- Given an auxiliary hash function $h' : U \to \{0, 1, ..., m-1\}$ for $i = 0, 1, ..., m-1$, we get the extended hash function

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \mod m, \tag{16}$$

where c_1, c_2 are auxiliary constants

Sequence of probes

- Given key k, we first probe $T[h'(k)]$, later positions probed are offset by amounts that depend in a quadratic manner on the probe number i.

- The initial probe determines the entire sequence, and so only m distinct probe sequences are used.

Quadratic probing: Definition and properties

Hash function

- Given an auxiliary hash function $h' : U \to \{0, 1, ..., m-1\}$ for $i = 0, 1, ..., m-1$, we get the extended hash function

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \mod m, \tag{16}$$

where c_1, c_2 are auxiliary constants

Sequence of probes

- Given key k, we first probe $T[h'(k)]$, later positions probed are offset by amounts that depend in a quadratic manner on the probe number i.
- The initial probe determines the entire sequence, and so only m distinct probe sequences are used.

Quadratic probing: Definition and properties

Hash function

- Given an auxiliary hash function $h' : U \rightarrow \{0, 1, ..., m-1\}$ for $i = 0, 1, ..., m-1$, we get the extended hash function

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \mod m, \tag{16}$$

where c_1, c_2 are auxiliary constants

Sequence of probes

- Given key k, we first probe $T[h'(k)]$, later positions probed are offset by amounts that depend in a quadratic manner on the probe number i.
- The initial probe determines the entire sequence, and so only m distinct probe sequences are used.

Quadratic probing: Definition and properties

Advantages

This method works much better than linear probing, but to make full use of the hash table, the values of c_1,c_2, and m are constrained.

Disadvantages

If two keys have the same initial probe position, then their probe sequences are the same, since $h(k_1, 0) = h(k_2, 0)$ implies $h(k_1, i) = h(k_2, i)$. This property leads to a milder form of clustering, called secondary clustering.

Quadratic probing: Definition and properties

Advantages

This method works much better than linear probing, but to make full use of the hash table, the values of c_1,c_2, and m are constrained.

Disadvantages

If two keys have the same initial probe position, then their probe sequences are the same, since $h(k_1, 0) = h(k_2, 0)$ implies $h(k_1, i) = h(k_2, i)$. This property leads to a milder form of clustering, called secondary clustering.

Outline

Definition and properties

Hash function

Double hashing uses a hash function of the form

$$h(k,i) = (h_1(k) + ih_2(k)) \mod m, \qquad (17)$$

where $i = 0, 1, ..., m-1$ and h_1, h_2 are auxiliary hash functions (Normally for a Universal family)

Definition and properties

Hash function

Double hashing uses a hash function of the form

$$h(k, i) = (h_1(k) + ih_2(k)) \mod m, \tag{17}$$

where $i = 0, 1, ..., m-1$ and h_1, h_2 are auxiliary hash functions (Normally for a Universal family)

Sequence of probes

- Given key k, we first probe $T[h_1(k)]$, successive probe positions are offset from previous positions by the amount $h_2(k) \mod m$.
- Thus, unlike the case of linear or quadratic probing, the probe sequence here depends in two ways upon the key k, since the initial probe position, the offset, or both, may vary.

Definition and properties

Hash function

Double hashing uses a hash function of the form

$$h(k,i) = (h_1(k) + ih_2(k)) \mod m, \tag{17}$$

where $i = 0, 1, ..., m-1$ and h_1, h_2 are auxiliary hash functions (Normally for a Universal family)

Sequence of probes

- Given key k, we first probe $T[h_1(k)]$, successive probe positions are offset from previous positions by the amount $h_2(k) \mod m$.
- Thus, unlike the case of linear or quadratic probing, the probe sequence here depends in two ways upon the key k, since the initial probe position, the offset, or both, may vary.

Definition and properties

Hash function

Double hashing uses a hash function of the form

$$h(k,i) = (h_1(k) + ih_2(k)) \mod m, \tag{17}$$

where $i = 0, 1, ..., m-1$ and h_1, h_2 are auxiliary hash functions (Normally for a Universal family)

Sequence of probes

- Given key k, we first probe $T[h_1(k)]$, successive probe positions are offset from previous positions by the amount $h_2(k) \mod m$.
- Thus, unlike the case of linear or quadratic probing, the probe sequence here depends in two ways upon the key k, since the initial probe position, the offset, or both, may vary.

Definition and properties

Hash function

Double hashing uses a hash function of the form

$$h(k, i) = (h_1(k) + ih_2(k)) \mod m, \tag{17}$$

where $i = 0, 1, ..., m-1$ and h_1, h_2 are auxiliary hash functions (Normally for a Universal family)

Sequence of probes

- Given key k, we first probe $T[h_1(k)]$, successive probe positions are offset from previous positions by the amount $h_2(k) \mod m$.
- Thus, unlike the case of linear or quadratic probing, the probe sequence here depends in two ways upon the key k, since the initial probe position, the offset, or both, may vary.

Definition and properties

Advantages

- When m is prime or a power of 2, double hashing improves over linear or quadratic probing in that $\Theta(m^2)$ probe sequences are used, rather than $\Theta(m)$ since each possible $(h_1(k), h_2(k))$ pair yields a distinct probe sequence.
- The performance of double hashing appears to be very close to the performance of the "ideal" scheme of uniform hashing.

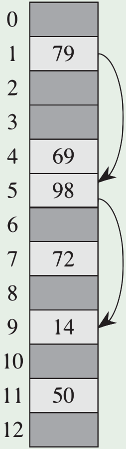
Definition and properties

Advantages

- When m is prime or a power of 2, double hashing improves over linear or quadratic probing in that $\Theta(m^2)$ probe sequences are used, rather than $\Theta(m)$ since each possible $(h_1(k), h_2(k))$ pair yields a distinct probe sequence.
- The performance of double hashing appears to be very close to the performance of the "ideal" scheme of uniform hashing.

Example

Jumping around to insert 14 with $h_1(k) = k \mod 13$ and $h_2(k) = 1 + (k \mod 11)$



Outline

Analysis of Open Addressing

Theorem 11.6

Given an open-address hash table with load factor $\alpha = \frac{n}{m} < 1$, the expected number of probes in an unsuccessful search is at most $\frac{1}{1-\alpha}$ assuming uniform hashing.

Corollary

Inserting an element into an open-address hash table with load factor requires at most $\frac{1}{1-\alpha}$ probes on average, assuming uniform hashing.

Theorem 11.8

Given an open-address hash table with load factor $\alpha < 1$, the expected number of probes in a successful search is at most $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$ assuming uniform hashing and assuming that each key in the table is equally likely to be searched for.

Analysis of Open Addressing

Theorem 11.6

Given an open-address hash table with load factor $\alpha = \frac{n}{m} < 1$, the expected number of probes in an unsuccessful search is at most $\frac{1}{1-\alpha}$ assuming uniform hashing.

Corollary

Inserting an element into an open-address hash table with load factor¿ requires at most $\frac{1}{1-\alpha}$ probes on average, assuming uniform hashing.

Theorem 11.8

Given an open-address hash table with load factor $\alpha < 1$, the expected number of probes in a successful search is at most $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$ assuming uniform hashing and assuming that each key in the table is equally likely to be searched for.

Analysis of Open Addressing

Theorem 11.6

Given an open-address hash table with load factor $\alpha = \frac{n}{m} < 1$, the expected number of probes in an unsuccessful search is at most $\frac{1}{1-\alpha}$ assuming uniform hashing.

Corollary

Inserting an element into an open-address hash table with load factor¿ requires at most $\frac{1}{1-\alpha}$ probes on average, assuming uniform hashing.

Theorem 11.8

Given an open-address hash table with load factor $\alpha < 1$, the expected number of probes in a successful search is at most $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$ assuming uniform hashing and assuming that each key in the table is equally likely to be searched for.

Exercise's

From Cormen's book, chapters 11

- 11.1-2
- 11.2-1
- 11.2-2
- 11.2-3
- 11.3-1
- 11.3-3