Data Structures

Dictionaries

DataLab

November 12, 2016

Outline

DataLab
Data Science Community

Outline

Dictionaries

Definition

The **ADT dictionary**—also called a map, table, or associative array—contains entries that each have two parts:

- A keyword—usually called a search key—such as an English word or a person's name
- A value—such as a definition, an address, or a telephone number—associated with that key

Dictionaries

Definition

The **ADT dictionary**—also called a map, table, or associative array—contains entries that each have two parts:

- A keyword—usually called a search key—such as an English word or a person's name
- A value—such as a definition, an address, or a telephone number—associated with that key

Dictionaries

Definition

The **ADT dictionary**—also called a map, table, or associative array—contains entries that each have two parts:

- A keyword—usually called a search key—such as an English word or a person's name
- A value—such as a definition, an address, or a telephone number—associated with that key

DataLab
Data Science Community

Example

Dictionary with Duplicates

Pairs are of the form (word, meaning).

Example

Dictionary with Duplicates

Pairs are of the form (word, meaning).

More than a single meaning

- (bolt, a threaded pin)
- (bolt, a crash of thunder)
- (bolt, to shoot forth suddenly)
- (bolt, a gulp)
- (bolt, a standard roll of cloth)
- etc.

DataLab
Data Science Community

Example

Dictionary with Duplicates

Pairs are of the form (word, meaning).

More than a single meaning

- (bolt, a threaded pin)
- (bolt, a crash of thunder)
- (bolt, to shoot forth suddenly)
- (bolt, a gulp)
- (bolt, a standard roll of cloth)
- etc.

DataLab
Data Science Community

Example

Dictionary with Duplicates

Pairs are of the form (word, meaning).

More than a single meaning

- (bolt, a threaded pin)
- (bolt, a crash of thunder)
- (bolt, to shoot forth suddenly)
- (bolt, a gulp)
- (bolt, a standard roll of cloth)
- etc.

Example

Dictionary with Duplicates

Pairs are of the form (word, meaning).

More than a single meaning

- (bolt, a threaded pin)
- (bolt, a crash of thunder)
- (bolt, to shoot forth suddenly)
- (bolt, a gulp)
- (bolt, a standard roll of cloth)
- etc.

Example

Dictionary with Duplicates

Pairs are of the form (word, meaning).

More than a single meaning

- (bolt, a threaded pin)
- (bolt, a crash of thunder)
- (bolt, to shoot forth suddenly)
- (bolt, a gulp)
- (bolt, a standard roll of cloth)
- etc.

Example

Dictionary with Duplicates

Pairs are of the form (word, meaning).

More than a single meaning

- (bolt, a threaded pin)
- (bolt, a crash of thunder)
- (bolt, to shoot forth suddenly)
- (bolt, a gulp)
- (bolt, a standard roll of cloth)
- etc.

Thus

We have possibly in a dictionary

- Sorted keys
- Duplicate keys

Outline

DataLab
Data Science Community

Operation in the ADT dictionary

Common operations with most databases

- **insert** adds a new entry to the dictionary, given a search key and associated value.
- **delete** removes an entry, given its associated search key
- **retrieve** retrieves the value associated with a given search key
- **search** sees whether the dictionary contains a given search key
- **traverse**
  - It traverse all the search keys in the dictionary
  - It traverse all the values in the dictionary

Operation in the ADT dictionary

Common operations with most databases

- **insert** adds a new entry to the dictionary, given a search key and associated value.
- **delete** removes an entry, given its associated search key
- **retrieve** retrieves the value associated with a given search key
- **search** sees whether the dictionary contains a given search key
- **traverse**
  - It traverse all the search keys in the dictionary
  - It traverse all the values in the dictionary

Operation in the ADT dictionary

Common operations with most databases

- **insert** adds a new entry to the dictionary, given a search key and associated value.
- **delete** removes an entry, given its associated search key
- **retrieve** retrieves the value associated with a given search key
- **search** sees whether the dictionary contains a given search key
- **traverse**
  - It traverse all the search keys in the dictionary
  - It traverse all the values in the dictionary

Operation in the ADT dictionary

Common operations with most databases

- **insert** adds a new entry to the dictionary, given a search key and associated value.
- **delete** removes an entry, given its associated search key
- **retrieve** retrieves the value associated with a given search key
- **search** sees whether the dictionary contains a given search key
- **traverse**
  - It traverse all the search keys in the dictionary
  - It traverse all the values in the dictionary

Operation in the ADT dictionary

Common operations with most databases

- **insert** adds a new entry to the dictionary, given a search key and associated value.
- **delete** removes an entry, given its associated search key
- **retrieve** retrieves the value associated with a given search key
- **search** sees whether the dictionary contains a given search key
- **traverse**
  - It traverse all the search keys in the dictionary
  - It traverse all the values in the dictionary

Operation in the ADT dictionary

Common operations with most databases

- **insert** adds a new entry to the dictionary, given a search key and associated value.
- **delete** removes an entry, given its associated search key
- **retrieve** retrieves the value associated with a given search key
- **search** sees whether the dictionary contains a given search key
- **traverse**
  - It traverse all the search keys in the dictionary
  - It traverse all the values in the dictionary

Operation in the ADT dictionary

Common operations with most databases

- **insert** adds a new entry to the dictionary, given a search key and associated value.
- **delete** removes an entry, given its associated search key
- **retrieve** retrieves the value associated with a given search key
- **search** sees whether the dictionary contains a given search key
- **traverse**
  - It traverse all the search keys in the dictionary
  - It traverse all the values in the dictionary

In addition

We have the following extra operations

- Detect whether a dictionary is empty
- Get the number of entries in the dictionary
- Remove all entries from the dictionary

In addition

We have the following extra operations

- Detect whether a dictionary is empty
- Get the number of entries in the dictionary
- Remove all entries from the dictionary

In addition

We have the following extra operations

- Detect whether a dictionary is empty
- Get the number of entries in the dictionary
- Remove all entries from the dictionary

Outline

Specifications: Add

Pseudocode

add(key, value)

Task

It adds the pair (key , value) to the dictionary.

Output : None

Specifications: Add

Pseudocode

add(key, value)

Task

It adds the pair (key , value) to the dictionary.

Input and Output

Input: key is an object search key, value is an associated object.

Output: None

Specifications: Add

Pseudocode

add(key, value)

Task

It adds the pair (key , value) to the dictionary.

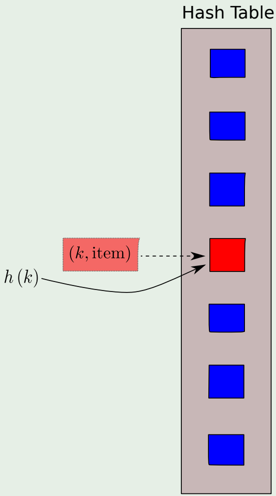
Input and Output

Input: key is an object search key, value is an associated object.

Output: None.

Example

Adding something



Outline

Specifications: Remove

Pseudocode

remove(key)

Specifications: Remove

Pseudocode

remove(key)

Task

It removes from the dictionary the entry that corresponds to a given search key.

Output: Returns either the value that was associated with the search key or null if no such object exists.

Specifications: Remove

Pseudocode

remove(key)

Task

It removes from the dictionary the entry that corresponds to a given search key.

Input and Output

Input: key is an object search key.

Output: Returns either the value that was associated with the search key or null if no such object exists.

Specifications: Remove

Pseudocode

remove(key)

Task

It removes from the dictionary the entry that corresponds to a given search key.

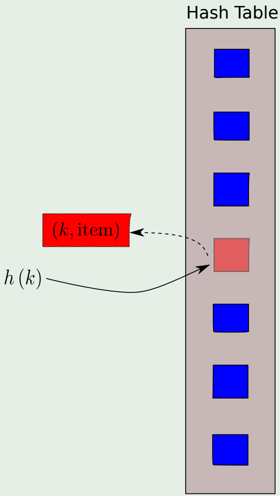
Input and Output

Input: key is an object search key.

Output: Returns either the value that was associated with the search key or null if no such object exists.

Example

Removing something



Outline

Specifications: GetValue

Pseudocode

getValue(key)

Specifications: GetValue

Pseudocode

getValue(key)

Task

It retrieves from the dictionary the value that corresponds to a given search key.

Output: Returns either the value associated with the search key or null if no such object exists.

Specifications: GetValue

Pseudocode

getValue(key)

Task

It retrieves from the dictionary the value that corresponds to a given search key.

Input and Output

Input: key is an object search key.

Output: Returns either the value associated with the search key or null if no such object exists.

Specifications: GetValue

Pseudocode

getValue(key)

Task

It retrieves from the dictionary the value that corresponds to a given search key.

Input and Output

Input: key is an object search key.

Output: Returns either the value associated with the search key or null if no such object exists.

Outline

DataLab
Data Science Community

Specifications: Contains

Pseudocode

contains(key)

Specifications: Contains

Pseudocode

contains(key)

Task

It sees whether any entry in the dictionary has a given search key.

Output: Returns true if an entry in the dictionary has key as its search key.

Specifications: Contains

Pseudocode

contains(key)

Task

It sees whether any entry in the dictionary has a given search key.

Input and Output

Input: key is an object search key.

Output: Returns true if an entry in the dictionary has key as its search key.

Specifications: Contains

Pseudocode

contains(key)

Task

It sees whether any entry in the dictionary has a given search key.

Input and Output

Input: key is an object search key.

Output: Returns true if an entry in the dictionary has key as its search key.

Outline

DataLab
Data Science Community

Specifications: GetKeyIterator

Pseudocode

getKeyIterator()

Specifications: GetKeyIterator

Pseudocode

getKeyIterator()

Task

It creates an iterator that traverses all search keys in the dictionary.

Output: Returns an iterator that provides sequential access to the search keys in the dictionary.

DataLab
Data Science Community

Specifications: GetKeyIterator

Pseudocode

getKeyIterator()

Task

It creates an iterator that traverses all search keys in the dictionary.

Input and Output

Input: None.

Output: Returns an iterator that provides sequential access to the search keys in the dictionary.

Specifications: GetKeyIterator

Pseudocode

getKeyIterator()

Task

It creates an iterator that traverses all search keys in the dictionary.

Input and Output

Input: None.

Output: Returns an iterator that provides sequential access to the search keys in the dictionary.

DataLab
Data Science Community

Specifications: GetValueIterator

Pseudocode

getValueIterator()

Task

It creates an iterator that traverses all values in the dictionary.

Output: Returns an iterator that provides sequential access to the values in the dictionary.

Specifications: GetValueIterator

Pseudocode

getValueIterator()

Task

It creates an iterator that traverses all values in the dictionary.

Input and Output

Input: None.

Output: Returns an iterator that provides [illegible] values in the dictionary.

DataLab
Data Science Community

Specifications: GetValueIterator

Pseudocode

getValueIterator()

Task

It creates an iterator that traverses all values in the dictionary.

Input and Output

Input: None.

Output: Returns an iterator that provides sequential access to the values in the dictionary.

Outline

Other Operations

isEmpty()

It sees whether the dictionary is empty.

getSize()

It gets the size of the dictionary.

clear()

It removes all entries from the dictionary.

Other Operations

isEmpty()

It sees whether the dictionary is empty.

getSize()

It gets the size of the dictionary.

clear()

It removes all entries from the dictionary.

Other Operations

isEmpty()

It sees whether the dictionary is empty.

getSize()

It gets the size of the dictionary.

clear()

It removes all entries from the dictionary.

Outline

DataLab
Data Science Community

However, we have two scenarios

Distinct search keys

Case 1 You can refuse to add another key-value.

Case 2 You can change the existing value associated with key to the new value. Then, you return the old value

However, we have two scenarios

Distinct search keys

Case 1 You can refuse to add another key-value.

Case 2 You can change the existing value associated with key to the new value. Then, you return the old value

Duplicate search keys

if the method add adds every given key-value entry to a dictionary

- The methods **remove** and **getValue** must deal with multiple entries that have the same search key.
- What do you remove or return!!!

However, we have two scenarios

Distinct search keys

Case 1 You can refuse to add another key-value.

Case 2 You can change the existing value associated with key to the new value. Then, you return the old value

Duplicate search keys

if the method add adds every given key-value entry to a dictionary

- The methods **remove** and **getValue** must deal with multiple entries that have the same search key.
- What do you remove or return!!!

However, we have two scenarios

Distinct search keys

Case 1 You can refuse to add another key-value.

Case 2 You can change the existing value associated with key to the new value. Then, you return the old value

Duplicate search keys

if the method add adds every given key-value entry to a dictionary

- The methods **remove** and **getValue** must deal with multiple entries that have the same search key.
- What do you remove or return!!!

However, we have two scenarios

Distinct search keys

Case 1 You can refuse to add another key-value.

Case 2 You can change the existing value associated with key to the new value. Then, you return the old value

Duplicate search keys

if the method add adds every given key-value entry to a dictionary

- The methods **remove** and **getValue** must deal with multiple entries that have the same search key.
- What do you remove or return!!!

Interface

We have the following interface

```
interface DictionaryInterface
{
  add(k, Item);
  remove(k);
  getValue(k);
  contains(k);
  getKeyIterator();
  getValueIterator();
  isEmpty();
  getSize();
  clear();
}
```

Outline

DataLab
Data Science Community

Where we can use this ADT?

In the phone directory problem

It is a directory that uses a name as the key and adds and returns a phone number

For example

Name	Number
Suzanne Nouveaux	401-555-1234
Andres Mendez-Vazquez	301-123-2345
...	...

Where we can use this ADT?

In the phone directory problem

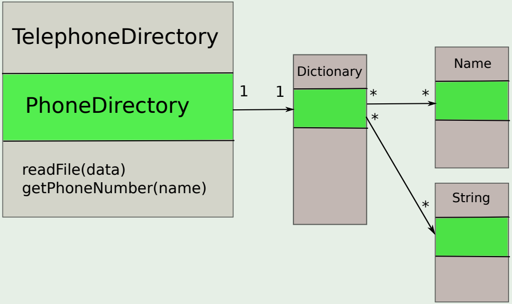
It is a directory that uses a name as the key and adds and returns a phone number

For example

Name	Number
Suzanne Nouveaux	401-555-1234
Andres Mendez-Vazquez	301-123-2345
...	...

Thus, we have the following diagram

A class Diagram



Outline

Now, the Big Question

It is a big one

How do we implement this data structure?

Now, the Big Question

It is a big one

How do we implement this data structure?

Possible ways

- Linear List
- Skip List
- Hash Tables
- ...

Now, the Big Question

It is a big one

How do we implement this data structure?

Possible ways

- Linear List
- Skip List
- Hash Tables
- ...

DataLab
Data Science Community

Now, the Big Question

It is a big one

How do we implement this data structure?

Possible ways

- Linear List
- Skip List
- Hash Tables
- ...

Now, the Big Question

It is a big one

How do we implement this data structure?

Possible ways

- Linear List
- Skip List
- Hash Tables
- ...

Outline

First: Represent It As A Linear List

You have

$L = (e_0, e_1, ..., e_{n-1})$

Where

Each e_i is a pair (key, element).

We can use the following representations

Array or linked representation.

First: Represent It As A Linear List

You have

$L = (e_0, e_1, ..., e_{n-1})$

Where

Each e_i is a pair (key, element).

We can use the following representations

Array or linked representation.

First: Represent It As A Linear List

You have

$L = (e_0, e_1, ..., e_{n-1})$

Where

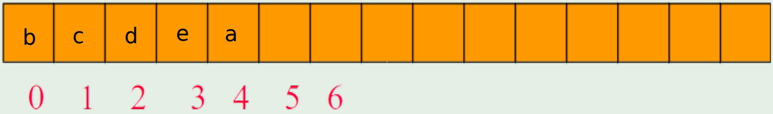
Each e_i is a pair (key, element).

We can use the following representations

Array or linked representation.

Array Representation

Our classic array

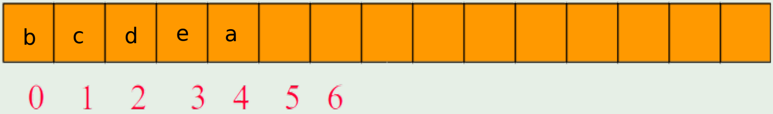


We have then

Operation in Array Representation	Complexity
getValue(theKey)	O(size)
add(theKey, theItem)	O(size) to find duplicate
	O(1) to add at right end
remove(theKey)	O(size)

Array Representation

Our classic array



We have then

Operation in Array Representation	Complexity
getValue(theKey)	O(size)
add(theKey, theItem)	O(size) to find duplicate O(1) to add at right end
remove(theKey)	O(size)

What if we sort the array?

Sorted Keys

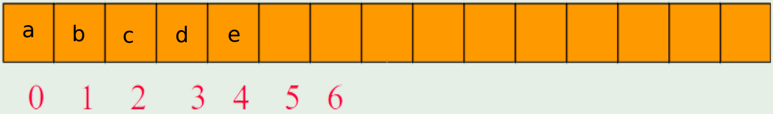
We have then

Operation in Array Representation	Complexity
getValue(theKey)	O(logsize) Using Binary Search
add(theKey, theItem)	O(logsize) to find duplicate
	O(size) to add
remove(theKey)	O(size)

DataLab
Data Science Community

What if we sort the array?

Sorted Keys

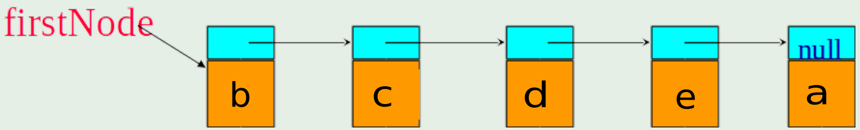


We have then

Operation in Array Representation	Complexity
getValue(theKey)	O(log size) Using Binary Search
add(theKey, theItem)	O(log size) to find duplicate O(size) to add
remove(theKey)	O(size)

Unsorted Chain

Our Structure

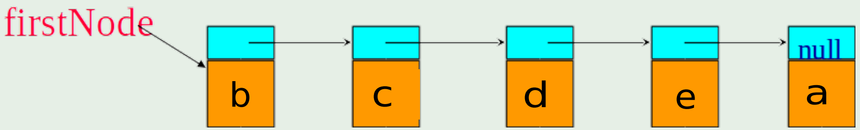


Complexity

Operation in Chain Representation	Complexity
getValue(theKey)	O(size)
add(theKey, theItem)	O(size) to find duplicate
	O(1) to add
remove(theKey)	O(size)

Unsorted Chain

Our Structure

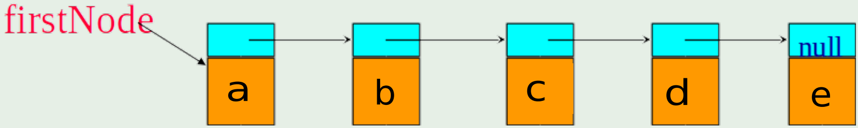


Complexity

Operation in Chain Representation	Complexity
getValue(theKey)	O(size)
add(theKey, theItem)	O(size) to find duplicate
	O(1) to add
remove(theKey)	O(size)

Sorted Chain

Our Structure



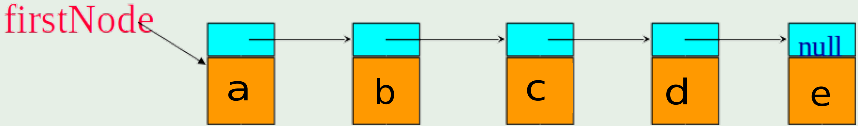
Complexity

Operation in Chain Representation	Complexity
getValue(theKey)	O(size)
add(theKey, theItem)	O(size) to find duplicate
	O(1) to add
remove(theKey)	O(size)

DataLab
Data Science Community

Sorted Chain

Our Structure

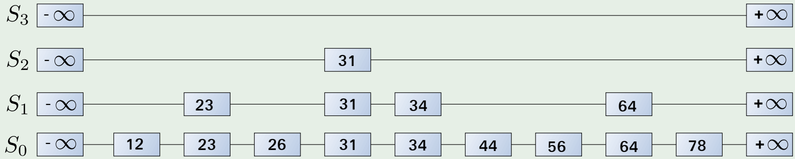


Complexity

Operation in Chain Representation	Complexity
getValue(theKey)	O(size)
add(theKey, theItem)	O(size) to find duplicate O(1) to add
remove(theKey)	O(size)

Skip Lists: we will skip it - It is for an advance class of analysis of algorithms

We have something like this

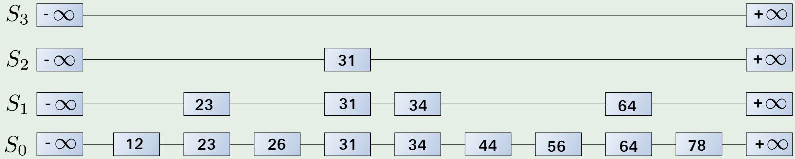


Complexity

Operation	Complexity - Worst Case	Complexity - Expected
getValue(theKey)	O(size)	O(log size)
add(theKey, theItem)	O(size)	O(log size)
remove(theKey)	O(size)	O(log size)

Skip Lists: we will skip it - It is for an advance class of analysis of algorithms

We have something like this



Complexity

Operation	Complexity - Worst Case	Complexity - Expected
getValue(theKey)	O(size)	O($\log$ size)
add(theKey, theItem)	O(size)	O($\log$ size)
remove(theKey)	O(size)	O($\log$ size)

Outline

DataLab
Data Science Community

We will concentrate our efforts in the Hash Tables

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.

Why?

Operation in Array Representation	Complexity - Worst Case	Complexity - Expected
getValue(theKey)	O(size)	$O(1+C)$
add(theKey, theItem)	O(size)	$O(1+C)$
remove(theKey)	O(size)	$O(1+C)$

We will concentrate our efforts in the Hash Tables

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.

Why?

Operation in Array Representation	Complexity - Worst Case	Complexity - Expected
getValue(theKey)	O(size)	$O(1+C)$
add(theKey, theItem)	O(size)	$O(1+C)$
remove(theKey)	O(size)	$O(1+C)$

Then

Advantages

- They have the advantage of having a expected complexity of operations of $O(1 + C)$
  - Still, be aware of C because this will change depending on which overflow policy you use...

Then

Advantages

- They have the advantage of having a expected complexity of operations of $O(1 + C)$
  - Still, be aware of C because this will change depending on which overflow policy you use...

Outline

DataLab
Data Science Community

You have two cases for this data structure

First

Small universe of keys.

Second

Large number of keys

You have two cases for this data structure

First

Small universe of keys.

Second

Large number of keys

When you have a small universe of keys, U

We can do the following

- Key values are direct addresses in the array.
- Direct implementation or Direct-address tables.

When you have a small universe of keys, U

We can do the following

- Key values are direct addresses in the array.
- Direct implementation or Direct-address tables.

Operations

1. Direct-Address-Search(Table, key)
   - return Table[key]
2. Direct-Address-Search(Table,key,value)
   - Table[key]=value
3. Direct-Address-Delete(T, x)
   - Table[key]=null

When you have a small universe of keys, U

We can do the following

- Key values are direct addresses in the array.
- Direct implementation or Direct-address tables.

Operations

1. Direct-Address-Search(Table, key)
   - return Table[key]
2. Direct-Address-Search(Table,key,value)
   - Table[key]=value
3. Direct-Address-Delete(T, x)
   - Table[key]=null

When you have a small universe of keys, U

We can do the following

- Key values are direct addresses in the array.
- Direct implementation or Direct-address tables.

Operations

1. Direct-Address-Search(Table, key)
   - return Table[key]
2. Direct-Address-Search(Table,key,value)
   - Table[key]=value
3. Direct-Address-Delete(T, x)
   - Table[key]=null

When you have a small universe of keys, U

We can do the following

- Key values are direct addresses in the array.
- Direct implementation or Direct-address tables.

Operations

1. Direct-Address-Search(Table, key)
   - return Table[key]
2. Direct-Address-Search(Table,key,value)
   - Table[key]=value
3. Direct-Address-Delete(T, x)
   - Table[key]=null

When you have a small universe of keys, U

We can do the following

- Key values are direct addresses in the array.
- Direct implementation or Direct-address tables.

Operations

1. Direct-Address-Search(Table, key)
   - return Table[key]
2. Direct-Address-Search(Table,key,value)
   - Table[key]=value
3. Direct-Address-Delete(T, x)
   - Table[key]=null

DataLab
Data Science Community

When you have a small universe of keys, U

We can do the following

- Key values are direct addresses in the array.
- Direct implementation or Direct-address tables.

Operations

1. Direct-Address-Search(Table, key)
   - return Table[key]
2. Direct-Address-Search(Table,key,value)
   - Table[key]=value
3. Direct-Address-Delete(T, x)
   - Table[key]=null

When you have a small universe of keys, U

We can do the following

- Key values are direct addresses in the array.
- Direct implementation or Direct-address tables.

Operations

1. Direct-Address-Search(Table, key)
   - return Table[key]
2. Direct-Address-Search(Table,key,value)
   - Table[key]=value
3. Direct-Address-Delete(T, x)
   - Table[key]=null

DataLab
Data Science Community

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

$$h : U \rightarrow \{0, 1, ..., m-1\} \quad (1)$$

DataLab
Data Science Community

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

$$h : U \rightarrow \{0, 1, ..., m-1\} \quad (1)$$

DataLab
Data Science Community

Example

Imagine that you have

A 1D array (or table) table$[0 : m-1]$.

Thus

$h(k)$is the home bucket for key k.

Then

Every dictionary pair (key, Item) is stored in its home bucket table$[h[key]]$.

Example

Imagine that you have

A 1D array (or table) table$[0 : m-1]$.

Thus

$h(k)$is the home bucket for key k.

Then

Every dictionary pair (key, Item) is stored in its home bucket table$[h[key]]$.

Example

Imagine that you have

A 1D array (or table) table$[0 : m - 1]$.

Thus

$h(k)$is the home bucket for key k.

Then

Every dictionary pair (key, Item) is stored in its home bucket table$[h[key]]$.

Example

Push the following pairs in a hash table of size $m = 8$

(22,a), (33,c), (3,d), (73,e), (85,f).

Hash function is

$key/11$

Then, we have that

Example

Push the following pairs in a hash table of size $m = 8$

(22,a), (33,c), (3,d), (73,e), (85,f).

Hash function is

$key/11$

Then, we have that

Example

Push the following pairs in a hash table of size $m = 8$

(22,a), (33,c), (3,d), (73,e), (85,f).

Hash function is

$key/11$

Then, we have that

(3,d)		(22,a)	(33,c)			(73,e)	(85,f)
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

What Can Go Wrong?

What if we add?

Where does (26,g) go?

What Can Go Wrong?

What if we add?

Where does (26,g) go?

Then

(3,d)		(22,a)	(33,c)			(73,e)	(85,f)
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

- 22 and 26 are synonyms with respect to the hash function that is in use
- This is known as collision or overflow.

What Can Go Wrong?

What if we add?

Where does (26,g) go?

Then

(3,d)		(22,a)	(33,c)			(73,e)	(85,f)
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

PROBLEM!!!

- Keys that have the same home bucket are synonyms.
- 22 and 26 are synonyms with respect to the hash function that is in use.
- This is known as **collision or overflow**.

What Can Go Wrong?

What if we add?

Where does (26,g) go?

Then

PROBLEM!!!

- Keys that have the same home bucket are synonyms.
- 22 and 26 are synonyms with respect to the hash function that is in use.
- This is known as collision or overflow.

What Can Go Wrong?

What if we add?

Where does (26,g) go?

Then

PROBLEM!!!

- Keys that have the same home bucket are synonyms.
- 22 and 26 are synonyms with respect to the hash function that is in use.
- This is known as **collision or overflow**.

DataLab
Data Science Community

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

DataLab
Data Science Community

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

Other Issues

First Issue

The choice of the possible hash function.

Second

The collision handling method

Third

The size (number of buckets) at the hash table

Other Issues

First Issue

The choice of the possible hash function.

Second

The collision handling method

Third

The size (number of buckets) at the hash table

Other Issues

First Issue

The choice of the possible hash function.

Second

The collision handling method

Third

The size (number of buckets) at the hash table

DataLab
Data Science Community

Outline

Hash Functions

They have two parts

1. The conversion of the key into an integer in the case the key is not an integer.
2. The mapping to the home bucket.

Hash Functions

They have two parts

1. The conversion of the key into an integer in the case the key is not an integer.
2. The mapping to the home bucket.

Hash Functions

They have two parts

1. The conversion of the key into an integer in the case the key is not an integer.
2. The mapping to the home bucket.

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.

DataLab
Data Science Community

Analysis of hashing: Which hash function?

Consider that:

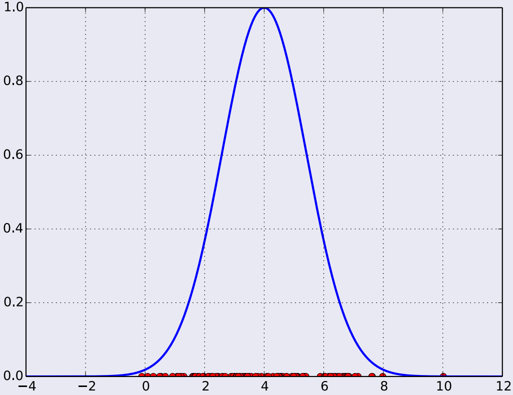
Good hash functions should maintain the property of simple uniform hashing!

- 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.

What if...?

Question:

What about something with keys in a normal distribution?



Hashing By Division

Universe of keys

keySpace = all integers.

Thus, we have that

For every m, the number of integers that get mapped (hashed) into bucket i is approximately $2^{32}/m$.

Properties

The division method results in a uniform hash function when keySpace = all integers.

Hashing By Division

Universe of keys

keySpace $=$ all integers.

Thus, we have that

For every m, the number of integers that get mapped (hashed) into bucket i is approximately $2^{32}/m$.

Properties

The division method results in a uniform hash function when keySpace $=$ all integers.

Hashing By Division

Universe of keys

keySpace = all integers.

Thus, we have that

For every m, the number of integers that get mapped (hashed) into bucket i is approximately $2^{32}/m$.

Properties

The division method results in a uniform hash function when keySpace = all integers.

However

Problem

In practice, keys tend to be correlated.

Thus

The choice of the divisor b affects the distribution of home buckets.

Then

Because of this correlation, applications tend to have a bias towards keys that map into odd integers (or into even ones).

However

Problem

In practice, keys tend to be correlated.

Thus

The choice of the divisor b affects the distribution of home buckets.

Then

Because of this correlation, applications tend to have a bias towards keys that map into odd integers (or into even ones).

However

Problem

In practice, keys tend to be correlated.

Thus

The choice of the divisor b affects the distribution of home buckets.

Then

Because of this correlation, applications tend to have a bias towards keys that map into odd integers (or into even ones).

Examples

Odd number and m an even number

Odd integers hash into odd home buckets

- $15\%14 = 1$, $3\%14 = 3$, $23\%14 = 9$

and m an even number

Even integers into even home buckets.

- $20\%14 = 6$, $30\%14 = 2$, $8\%14 = 8$

Properties

The bias in the keys results in a bias toward either the odd or even home buckets.

Examples

Odd number and m an even number

Odd integers hash into odd home buckets

- $15\%14 = 1$, $3\%14 = 3$, $23\%14 = 9$

Even number and m an even number

Even integers into even home buckets.

- $20\%14 = 6$, $30\%14 = 2$, $8\%14 = 8$

Properties

The bias in the keys results in a bias toward either the odd or even home buckets.

Examples

Odd number and m an even number

Odd integers hash into odd home buckets

- $15\%14 = 1$, $3\%14 = 3$, $23\%14 = 9$

Even number and m an even number

Even integers into even home buckets.

- $20\%14 = 6$, $30\%14 = 2$, $8\%14 = 8$

Properties

The bias in the keys results in a bias toward either the odd or even home buckets.

What if we use an Odd number?

Odd number and m an odd number

odd integers may hash into any home.

- $15\%15 = 0$, $3\%15 = 3$, $23\%15 = 8$

and m an odd number

Even integers may hash into any home.

- $20\%15 = 5$, $30\%15 = 0$, $8\%15 = 8$

Thus

The bias in the keys does not result in a bias toward either the odd or even home buckets.

What if we use an Odd number?

Odd number and m an odd number

odd integers may hash into any home.

- $15\%15 = 0$, $3\%15 = 3$, $23\%15 = 8$

Even number and m an odd number

Even integers may hash into any home.

- $20\%15 = 5$, $30\%15 = 0$, $8\%15 = 8$

Thus

The bias in the keys does not result in a bias toward either the odd or even home buckets.

What if we use an Odd number?

Odd number and m an odd number

odd integers may hash into any home.

- $15\%15 = 0$, $3\%15 = 3$, $23\%15 = 8$

Even number and m an odd number

Even integers may hash into any home.

- $20\%15 = 5$, $30\%15 = 0$, $8\%15 = 8$

Thus

The bias in the keys does not result in a bias toward either the odd or even home buckets.

Bias

Something Notable

The bias in the keys does not result in a bias toward either the odd or even home buckets.

Then, we have

We have a better chance of uniformly distributed home buckets.

Thus

So do not use an even divisor.

Bias

Something Notable

The bias in the keys does not result in a bias toward either the odd or even home buckets.

Then, we have

We have a better chance of uniformly distributed home buckets.

Thus

So do not use an even divisor.

Bias

Something Notable

The bias in the keys does not result in a bias toward either the odd or even home buckets.

Then, we have

We have a better chance of uniformly distributed home buckets.

Thus

So do not use an even divisor.

Selecting The Divisor

Another Problem

Similar biased distribution of home buckets is seen, in practice, when the divisor is a multiple of prime numbers such as 3, 5, 7, . . .

However

The effect of each prime divisor p of m decreases as p gets larger.

Rules of Choosing m

- Ideally, choose m so that it is a prime number.
- Not to close to a power of 2.

Selecting The Divisor

Another Problem

Similar biased distribution of home buckets is seen, in practice, when the divisor is a multiple of prime numbers such as 3, 5, 7, . . .

However

The effect of each prime divisor p of m decreases as p gets larger.

Rules of Choosing m

- Ideally, choose m so that it is a prime number.
- Not to close to a power of 2.

Selecting The Divisor

Another Problem

Similar biased distribution of home buckets is seen, in practice, when the divisor is a multiple of prime numbers such as 3, 5, 7, . . .

However

The effect of each prime divisor p of m decreases as p gets larger.

Rules of Choosing m

- Ideally, choose m so that it is a prime number.
- Not to close to a power of 2.

However

Something Notable

Even with this hash function, we can have problems

Remember

The Gaussian Keys...

However

Something Notable

Even with this hash function, we can have problems

Remember

The Gaussian Keys...

Universal Hashing

Issues

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

Universal Hashing

Issues

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

Goal

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

Universal Hashing

Issues

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

Issues

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

Goal

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

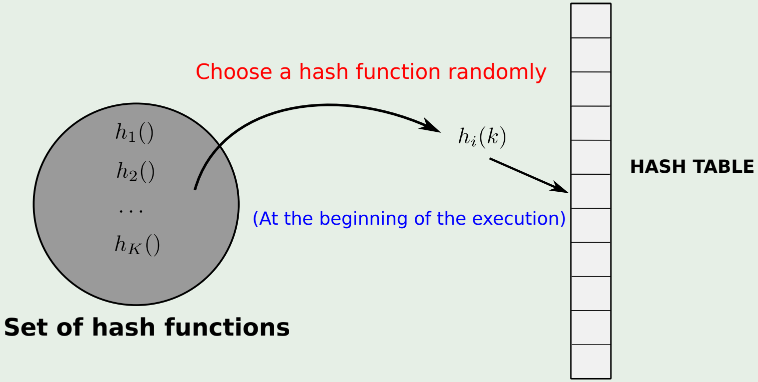
Idea

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

DataLab
Data Science Community

Universal Hashing

Example



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.

Example: Universal hash functions

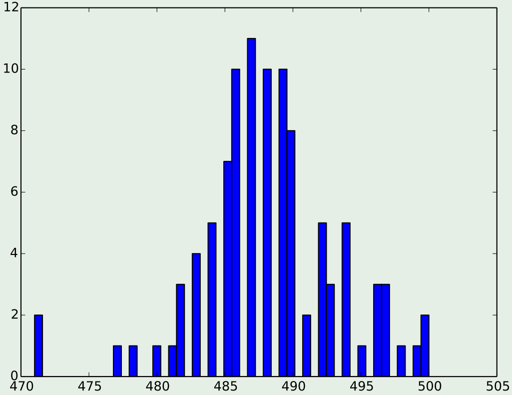
Example

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

DataLab
Data Science Community

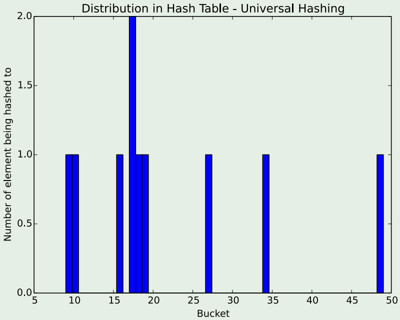
Example of key distribution

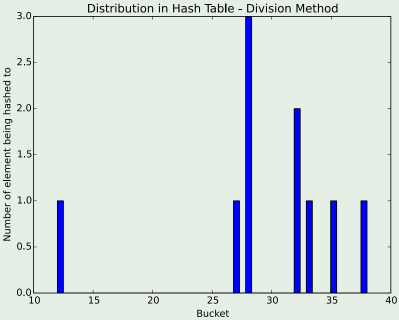
Example, mean $=$ 488.5 and dispersion $=$ 5



Example with 10 keys

Universal Hashing Vs Division Method

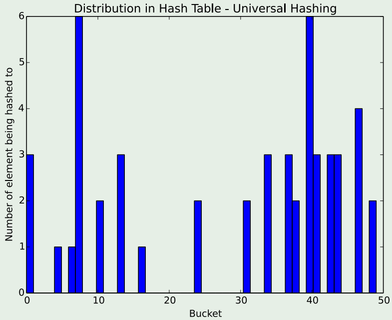


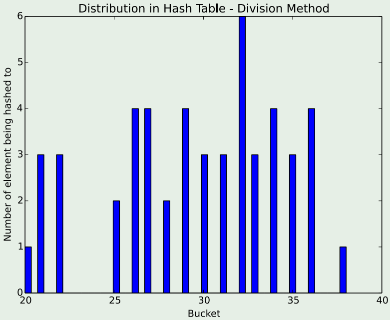


DataLab
Data Science Community

Example with 50 keys

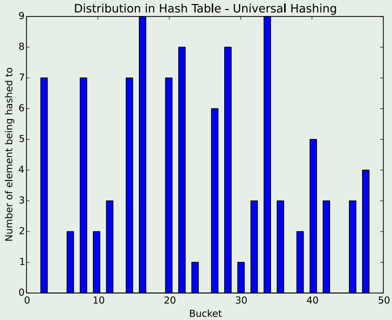
Universal Hashing Vs Division Method

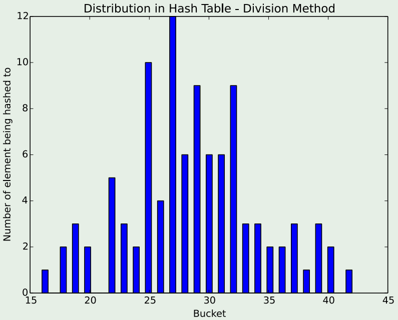




Example with 100 keys

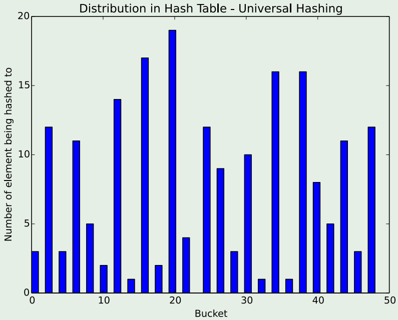
Universal Hashing Vs Division Method

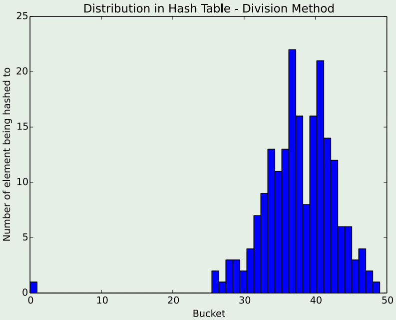




Example with 200 keys

Universal Hashing Vs Division Method





DataLab
Data Science Community

Outline

DataLab
Data Science Community

Overflow Handling

Overflow

An overflow occurs when the home bucket for a new pair (key, element) is full.

Overflow Handling

Overflow

An overflow occurs when the home bucket for a new pair (key, element) is full.

One strategy to handle overflow, small universe of keys

Search the hash table in some systematic fashion for a bucket that is not full.

- Linear probing (linear open addressing).
- Quadratic probing.
- Random probing.

- Array linear list.
- Chain.

Overflow Handling

Overflow

An overflow occurs when the home bucket for a new pair (key, element) is full.

One strategy to handle overflow, small universe of keys

Search the hash table in some systematic fashion for a bucket that is not full.

- Linear probing (linear open addressing).
- Quadratic probing.
- Random probing.

The other strategy, a large universe of keys

Eliminate overflows by permitting each bucket to keep a list of all pairs for which it is the home bucket.

- Array linear list.
- Chain.

Overflow Handling

Overflow

An overflow occurs when the home bucket for a new pair (key, element) is full.

One strategy to handle overflow, small universe of keys

Search the hash table in some systematic fashion for a bucket that is not full.

- Linear probing (linear open addressing).
- Quadratic probing.
- Random probing.

The other strategy, a large universe of keys

Eliminate overflows by permitting each bucket to keep a list of all pairs for which it is the home bucket.

- Array linear list.
- Chain.

Overflow Handling

Overflow

An overflow occurs when the home bucket for a new pair (key, element) is full.

One strategy to handle overflow, small universe of keys

Search the hash table in some systematic fashion for a bucket that is not full.

- Linear probing (linear open addressing).
- Quadratic probing.
- Random probing.

The other strategy, a large universe of keys

Eliminate overflows by permitting each bucket to keep a list of all pairs for which it is the home bucket.

- Array linear list.
- Chain.

Overflow Handling

Overflow

An overflow occurs when the home bucket for a new pair (key, element) is full.

One strategy to handle overflow, small universe of keys

Search the hash table in some systematic fashion for a bucket that is not full.

- Linear probing (linear open addressing).
- Quadratic probing.
- Random probing.

The other strategy, a large universe of keys

Eliminate overflows by permitting each bucket to keep a list of all pairs for which it is the home bucket.

- Array linear list.
- Chain.

Overflow Handling

Overflow

An overflow occurs when the home bucket for a new pair (key, element) is full.

One strategy to handle overflow, small universe of keys

Search the hash table in some systematic fashion for a bucket that is not full.

- Linear probing (linear open addressing).
- Quadratic probing.
- Random probing.

The other strategy, a large universe of keys

Eliminate overflows by permitting each bucket to keep a list of all pairs for which it is the home bucket.

- Array linear list.
- Chain.

Outline

DataLab
Data Science Community

Linear List Of Synonyms

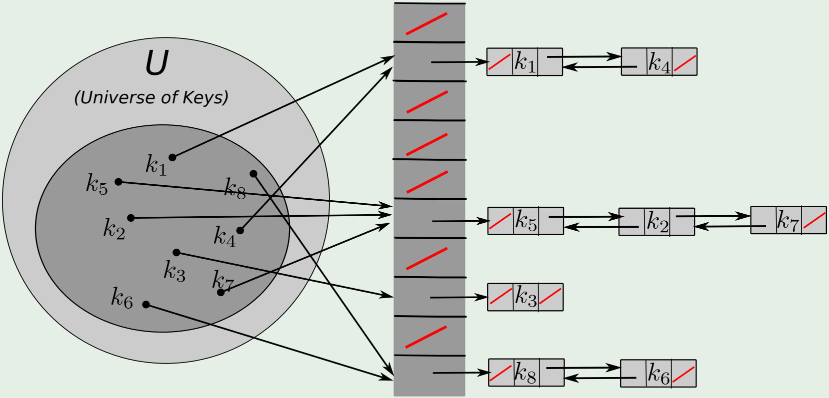
Thus

- Each bucket keeps a linear list of all pairs for which it is the home bucket.
- The linear list may or may not be sorted by key.
- The linear list may be an array linear list or a chain.

Collision Handling: Chaining

A Possible Solution

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



Example Sorted Chains

Add to a hash table with $m = 11$

Put in pairs whose keys are 6, 17, 12, 23, 28, 5, 16, 3, 8

So, we have

Home bucket = key % 11.

Example Sorted Chains

Add to a hash table with $m = 11$

Put in pairs whose keys are 6, 17, 12, 23, 28, 5, 16, 3, 8

So, we have

Home bucket = key % 11.

Example

The Table

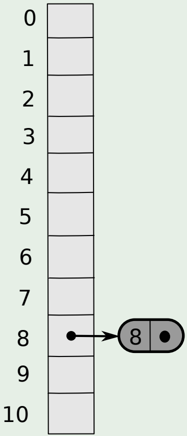
6, 17, 12, 23, 28, 5, 16, 3, 8

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Example

The Table

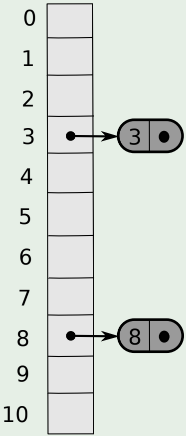
6, 17, 12, 23, 28, 5, 16, 3



Example

The Table

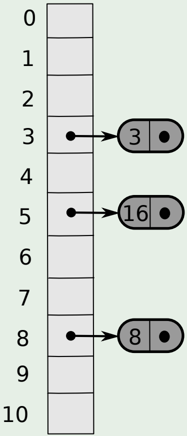
6, 17, 12, 23, 28, 5, 16



Example

The Table

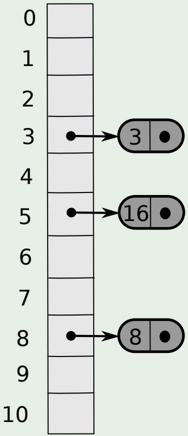
6, 17, 12, 23, 28, 5



Example

The Table

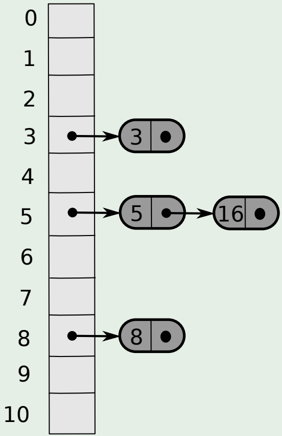
6, 17, 12, 23, 28, 5



Example

The Table

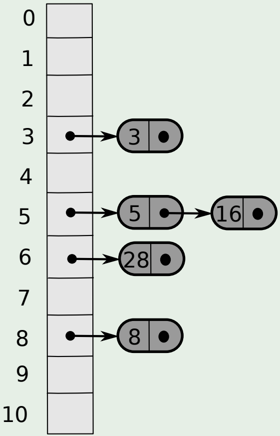
6, 17, 12, 23, 28



Example

The Table

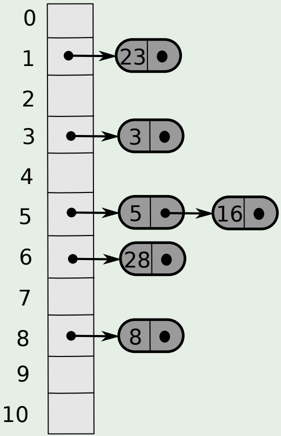
6, 17, 12, 23



Example

The Table

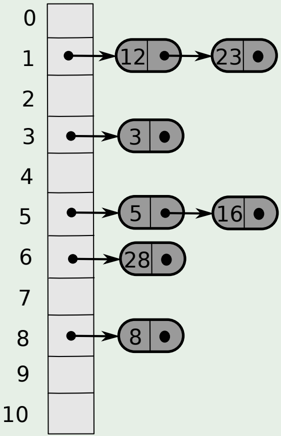
6, 17, 12



Example

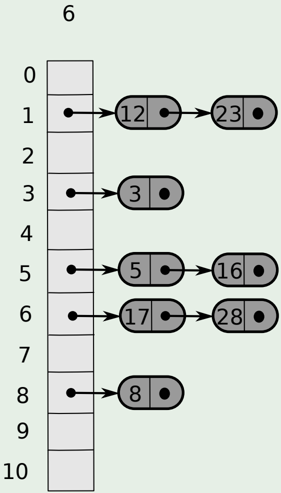
The Table

6, 17



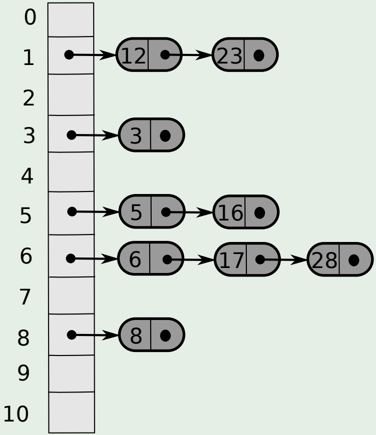
Example

The Table



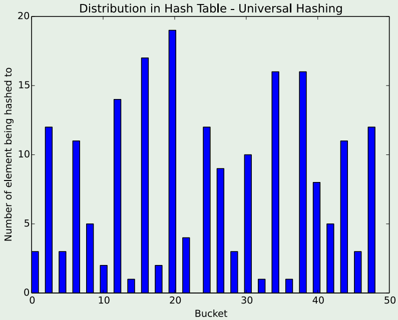
Example

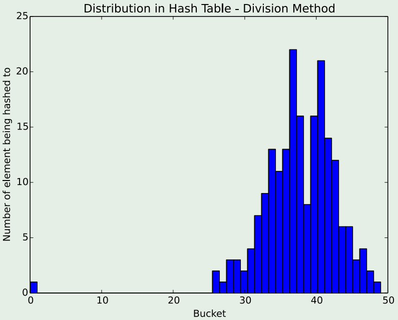
The Table



Do You Remember This?

Universal Hashing Vs Division Method





DataLab
Data Science Community

Expected Complexity of Hash Table under Chaining

We have for unsuccessful search

$$U_n = O\left(1 + \alpha\right) \tag{2}$$

We have for successful search

$$S_n = O\left(1 + \alpha\right) \tag{3}$$

Expected Complexity of Hash Table under Chaining

We have for unsuccessful search

$$U_n = O\left(1 + \alpha\right) \tag{2}$$

We have for successful search

$$S_n = O\left(1 + \alpha\right) \tag{3}$$

Now, Back to the real world

java.util.Hashtable

- It uses unsorted chains.
- It uses a default initial m = divisor = 101
- It uses a default $\alpha \leq 0.75$
- When loading density exceeds a max permissible threshold, It rehash with new $m = 2m+1$.

Now, Back to the real world

java.util.Hashtable

- It uses unsorted chains.
- It uses a default initial m = divisor = 101
- It uses a default $\alpha \leq 0.75$
- When loading density exceeds a max permissible threshold, It rehash with new $m = 2m+1$.

DataLab
Data Science Community

Now, Back to the real world

java.util.Hashtable

- It uses unsorted chains.
- It uses a default initial m = divisor = 101
- It uses a default $\alpha \leq 0.75$
- When loading density exceeds a max permissible threshold, It rehash with new $m = 2m+1$.

Now, Back to the real world

java.util.Hashtable

- It uses unsorted chains.
- It uses a default initial m = divisor = 101
- It uses a default $\alpha \leq 0.75$
- When loading density exceeds a max permissible threshold, It rehash with new $m = 2m{+}1$.

DataLab
Data Science Community