Hash Table

Ja-Hee Kim

Agenda

01 Introduction

ADT dictionary, accessing pattern, built-in library HashMap

02 Advanced Java Technique

Iterator and lambda expression

03 Collision

Linear probing, quadratic probing, double hashing

04 Performance

Load factor

Introduction of Hash Table

ADT: Dictionary

- A dictionary is a collection of ordered items.
- Aliases: map, table, associative array
  - Keyword
    - Search key
    - Example: English word, person's name
  - Value
    - Data associated with that key.
    - Example: definition, address, telephone number
- ADT Dictionary should enable you to locate the desired entry efficiently.

Performance Comparison

- Is there any data structure whose expectation time complexities for looking up, adding, and removing are constant?

<table>
<tr><th></th><th>Binary search tree</th><th>Balance BST</th><th>Sorted array list</th><th>Sorted linked list</th></tr>
<tr><td>Look up</td><td rowspan="3">Expected O(log n)
Worst case O(n)</td><td rowspan="3">O(log n)</td><td>O(log n)</td><td rowspan="3">O(n)</td></tr>
<tr><td>add</td><td rowspan="2">O(n)</td></tr>
<tr><td>delete</td></tr>
</table>

Two general access patterns for data structures

- Sequential access

- Direct access (also called random access)

Sequential access

- Provided by a linked list
- You find an element by starting at one end of the structure and looking at each element, one after the other, until you find your target or you reach the other end.
- That search algorithm runs in linear time.
- Example: Audiotapes and videotapes

Direct access

- Provided by arrays
- You find an element by using a given index i to go directly to the element a[i]. We think of i as the address of the element a[i].
- Direct access runs in constant time.
- Unfortunately, it requires prior knowledge of the element's index.
- Example: Audio CD and DVD

What we want

- A data structure that attempts to **provide direct access** without requiring prior knowledge of an element's **index**.

Hash Table

- a data structure that efficiently stores and retrieves data from memory
- a **hash function** that computes the element's index from its contents.
- Example of hash function

```
private int computeHash(String s)  {
        int hash = 0;
        for (int i = 0; i < s.length(); i++)  hash += s.charAt(i);
        return hash % SIZE;    // SIZE = 10 in example
}
```

H(dog) = (100+103+111)%10=4

0	1	2	3	4	5	6	7	8	9

100	0x64	d
101	0x65	e
102	0x66	f
103	0x67	g
104	0x68	h
105	0x69	i
106	0x6A	j
107	0x6B	k
108	0x6C	l
109	0x6D	m
110	0x6E	n
111	0x6F	o

Java built-in

- java.util.HashMap

java.util

Class HashMap<K,V>

java.lang.Object
    java.util.AbstractMap<K,V>
        java.util.HashMap<K,V>

Type Parameters:

K - the type of keys maintained by this map

V - the type of mapped values

```java
import java.util.*;
public class DictionarayTest {
    public static void main(String[] args) {
        Map<String, Integer> address = new HashMap<>();
        if(address.isEmpty()) System.out.println("No number in my emergency phone address book");
        else System.out.println("I have "+address.size()+" numbers");
        address.put("Korea", 119);
        address.put("SC", 116);
        address.put("EU", 112);
        address.put("USA", 911);
        address.put("Australia", 000);
        address.put("London", 999);
        address.put("France", 17);
        if(address.isEmpty()) System.out.println("No number in my emergency phone address book");
        else System.out.println("I have "+address.size()+" numbers");
        if(address.containsKey("USA")) System.out.println("The emergency phone number in USA is "+ address.get("USA"));
        else System.out.println("We cannot find emergency phone number in U.S.");
        if(address.containsKey("Japan")) System.out.println(address.get("Japan"));
        else System.out.println("We cannot find emergency phone number in Japan");
        System.out.println("Emergency phone number book: "+ address);
        System.out.println("Removing USA: "+ address.remove("USA"));
        System.out.println("Removing SC: "+ address.remove("SC"));
        System.out.println("Removing Korea: "+ address.remove("Korea"));
        System.out.println("Emergency phone number book: "+ address);
        System.out.println("===Using iterator");
        Iterator<String> keys = address.keySet().iterator();
        while(keys.hasNext()) {
            String key = keys.next();
            System.out.print(key+": "+address.get(key)+" ");
        }
        System.out.println("\n===Using lambda expression");
        address.forEach((key, value)->{System.out.print(key+": "+value+" ");});
    }
}
```

Problems Javadoc Declaration Console

```
<terminated> DictionarayTest (1) [Java Application] C:₩Program Files₩Java₩jdk-14.0.2₩bin₩javaw.exe  (2020. 11. 13. 오후 10:58:35 – 오후 10:58:35)
No number in my emergency phone address book
I have 7 numbers
The emergency phone number in USA is 911
We cannot find emergency phone number in Japan
Emergency phone number book: {SC=116, EU=112, USA=911, London=999, Australia=0, France=17, Korea=119}
Removing USA: 911
Removing SC: 116
Removing Korea: 119
Emergency phone number book: {EU=112, London=999, Australia=0, France=17}
===Using iterator
EU: 112 London: 999 Australia: 0 France: 17
===Using lambda expression
EU: 112 London: 999 Australia: 0 France: 17
```

Constructors

Constructor Summary

Constructors

Constructor and Description
`HashMap()` Constructs an empty HashMap with the default initial capacity (16) and the default load factor (0.75).
`HashMap(int initialCapacity)` Constructs an empty HashMap with the specified initial capacity and the default load factor (0.75).
`HashMap(int initialCapacity, float loadFactor)` Constructs an empty HashMap with the specified initial capacity and load factor.
`HashMap(Map<? extends K,? extends V> m)` Constructs a new HashMap with the same mappings as the specified Map.

Operations

- put(key: K, value: V): V
  - maps the specified key to the value in this hash table.
  - Input
    - key: key with which the specified value is to be associated
    - value: value to be associated with the specified key
  - Return: the previous value associated with key

- remove(key: K): V
  - the previous value associated with key
  - Input
    - key:key whose mapping is to be removed from the map
  - Return
    - key whose mapping is to be removed from the map

Operations

- ContainsKey(key K): Boolean
  - Tests if the specified value/key is in this hash table.
  - Input
    - key: the key whose associated value is to be returned
  - Return: true if this map contains a mapping for the specified key

- get(key K): V
  - returns the value to which the specified key is mapped in this hash table
  - Input:
    - key: the key whose associated value is to be returned
  - Return: the key whose associated value is to be returned

Visit all items

- Iterator vs lambda expression
  - public Set<K> keySet()
    - returns a Set view of the keys contained in this map.
  - public void forEach(BiConsumer<? super K,? super V> action)
    - Input: The action to be performed for each entry

```
        System.out.println("===Using iterator");
        Iterator<String> keys = address.keySet().iterator();
        while(keys.hasNext()) {
            String key = keys.next();
            System.out.print(key+": "+address.get(key)+" ");
        }
        System.out.println("\n===Using lambda expression");
        address.forEach((key, value)->{System.out.print(key+": "+value+" ");});
```

```
===Using iterator
EU: 112 London: 999 Australia: 0 France: 17
===Using lambda expression
EU: 112 London: 999 Australia: 0 France: 17
```

Iterator

Interface Iterator

java.util

Interface Iterator<E>

```
Iterator<String> keys = address.keySet().iterator();
```

Type Parameters:

E - the type of elements returned by this iterator

---

public interface **Iterator<E>**

An iterator over a collection. Iterator takes the place of Enumeration in the Java Collections Framework. Iterators differ from enumerations in two ways:

- Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
- Method names have been improved.

This interface is a member of the Java Collections Framework.

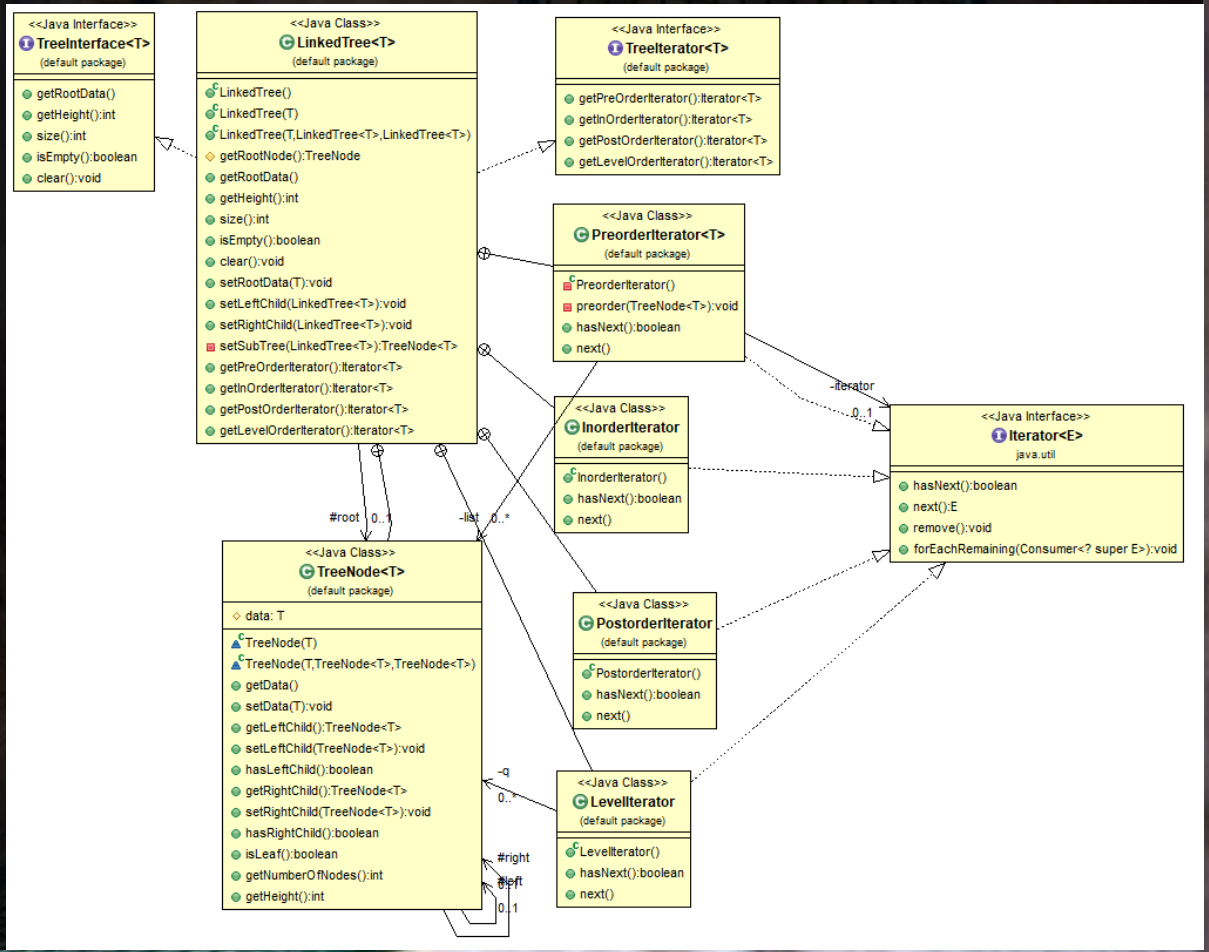
Method Summary

All Methods | Instance Methods | Abstract Methods | Default Methods

Modifier and Type	Method and Description
default void	**forEachRemaining(Consumer<? super E> action)** Performs the given action for each remaining element until all elements have been processed or the action throws an exception.
boolean	**hasNext()** Returns true if the iteration has more elements.
E	**next()** Returns the next element in the iteration.
default void	**remove()** Removes from the underlying collection the last element returned by this iterator (optional operation).

Example using Iterator

- Instead of the method returning String
  - Method for getting iterator
  - Inner class implementing Iterator
  - Interface containing iterators



TreeIterator

- We can defines all traversal methods for a tree.

```
import java.util.Iterator;
public interface TreeIterator<T> {
    public Iterator<T> getPreOrderIterator();
    public Iterator<T> getInOrderIterator();
    public Iterator<T> getPostOrderIterator();
    public Iterator<T> getLevelOrderIterator();
}
```

```
public class LinkedTree<T>
    implements TreeInterface<T>, TreeIterator<T>{
```

Returning the iterators

- They returns some class instances that implements interface Iterator.

```
public Iterator<T> getPreOrderIterator() {
    return new PreorderIterator();
}
public Iterator<T> getInOrderIterator() {
    return new InorderIterator();
}
public Iterator<T> getPostOrderIterator() {
    return new PostorderIterator();
}
public Iterator<T> getLevelOrderIterator() {
    return (Iterator<T>) new LevelIterator();
}
```

Example of implementing Iterator

- They should implement methods hasNext and next

```
private class LevelIterator implements Iterator<T> {
    private Queue<TreeNode<T>> q;
    public LevelIterator() {
        q = new ArrayDeque();
        q.offer(root);
    }
    public boolean hasNext() {
        return !q.isEmpty();
    }
    public T next() {
        TreeNode<T> result = null;
        if(!q.isEmpty()) {
            result = q.poll();
            if(result.hasLeftChild()) q.offer(result.left);
            if(result.hasRightChild()) q.offer(result.right);
        } else new NoSuchElementException();
        return result.data;
    }
}
```

```
public String bfs() {
    Queue<TreeNode<T>> q = new ArrayDeque();
    String s = "";
    if(!isEmpty()) {
        q.offer(root);
        for(TreeNode<T> node = q.poll() ; node!=null ; node = q.poll()) {
            s+=node.data + " ";
            if(node.hasLeftChild()) q.offer(node.left);
            if(node.hasRightChild()) q.offer(node.right);
        }
    }
    return s;
}
```

Example of implementing Iterator

- We can use List and Iterator for the list
- What happens if we change the tree after creating Iterator

```
private class PreorderIterator<T> implements Iterator<T> {
    private List<TreeNode<T>> list;
    private Iterator<TreeNode<T>> iterator;
    private PreorderIterator() {
        list = new LinkedList<>();
        preorder((TreeNode<T>) root);
        iterator = list.iterator();
    }
    private void preorder(TreeNode<T> node) {
        if(node != null ) list.add(node);
        if(node.hasLeftChild()) preorder(node.getLeftChild());
        if(node.hasRightChild()) preorder(node.getRightChild());
    }
    public boolean hasNext() {
        return iterator.hasNext();
    }
    public T next() {
        return iterator.next().data;
    }
}
```

In TreeNode

```
public String preorder() {
    String s = "";
    s+=data + " ";
    if(hasLeftChild())s+=left.preorder();
    if(hasRightChild())s+=right.preorder();
    return s;
}
```

In LinkedTree

```
public String preorder() {
    return root.preorder();
}
```

λ Expression

forEach method

```
default void forEach(Consumer<? super T> action)
```

Performs the given action for each element of the `Iterable` until all elements have been processed or the action throws an exception. Unless otherwise specified by the implementing class, actions are performed in the order of iteration (if an iteration order is specified). Exceptions thrown by the action are relayed to the caller.

Implementation Requirements:

The default implementation behaves as if:

```
for (T t : this)
    action.accept(t);
```

Parameters:

action - The action to be performed for each element

```java
import java.util.*;
import java.util.function.Consumer;

public class Lambda {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1,2, 3, 4, 5, 6, 7, 8, 9, 10);
        numbers.forEach(new MyConsumer<Integer>());
    }
}
class MyConsumer<T> implements Consumer<T>{
    public void accept(T n) {
        System.out.print(n+" ");
    }
}
```

Problems @ Javadoc Declaration Console

<terminated> Lambda [Java Application] C:₩Program Files₩Java₩jdk-14.0.2₩bin₩javaw.exe (2020. 11. 16. 오후 9:33:43 – 오후 9:33:43)

```
1 2 3 4 5 6 7 8 9 10
```

Anonymous class

- An anonymous class is a local class that does not have a name.
- An anonymous class allows an object to be created using an expression that combines object creation with the declaration of the class.
- This avoid naming a class, at the cost of only ever being able to create one instance of that a nonymous class.
- This is handy in the AWT.
- An anonymous class defined as part of `new` expression and must be a subclass or implement an interface.

```
new className(argumentList){
      //class body
}
new interfacName() ){
      //class body
}
```

Example of anonymous class

```java
import java.util.*;
import java.util.function.Consumer;

public class Lambda {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1,2, 3, 4, 5, 6, 7, 8, 9, 10);
        numbers.forEach(new Consumer<Integer>(){
            public void accept(Integer n) {
                System.out.print(n+" ");
            }
        });
    }
}
```

Problems @ Javadoc Declaration Console

```
<terminated> Lambda [Java Application] C:₩Program Files₩Java₩jdk-14.0.2₩bin₩javaw.exe (2020. 11. 16. 오후 9:48:55 – 오후 9:48:55)
1 2 3 4 5 6 7 8 9 10
```

Lambda expression

- A short block of code which takes in parameters and returns a value.
- Do not need a name and they can be implemented right in the body of a method.

- Syntax
  - Parameter ->expression
  - (parameter1, parameter2)->expression
  - (parameter1, parameter2)->{code block}

```java
import java.util.*;
public class Lambda {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1,2, 3, 4, 5, 6, 7, 8, 9, 10);
        numbers.forEach(value->System.out.print(value+" "));
    }
}
```

Problems @ Javadoc Declaration Console

```
<terminated> Lambda [Java Application] C:₩Program Files₩Java₩jdk-14.0.2₩bin₩javaw.exe (2020. 11. 16. 오후 9:58:21 – 오후 9:58:23)
1 2 3 4 5 6 7 8 9 10
```

Advanced lambda expressions

```java
import java.util.*;
public class Lambda {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1,2, 3, 4, 5, 6, 7, 8, 9, 10);
        numbers.forEach(System.out::print);
    }
}
```

Problems @ Javadoc Declaration Console

<terminated> Lambda [Java Application] C:₩Program Files₩Java₩jdk-14.0.2₩bin₩javaw.exe (2020. 11. 16. 오후 9:59:28 – 오후 9:59:28)

```
12345678910
```

```java
import java.util.*;
public class Lambda {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1,2, 3, 4, 5, 6, 7, 8, 9, 10);
        numbers.stream().map(value->value+" ").forEach(System.out::print);
    }
}
```

Problems @ Javadoc Declaration Console

<terminated> Lambda [Java Application] C:₩Program Files₩Java₩jdk-14.0.2₩bin₩javaw.exe (2020. 11. 16. 오후 10:04:09 – 오후 10:04:09)

```
1 2 3 4 5 6 7 8 9 10
```

Lambda expression for HashMap

- Iteration

```
while(keys.hasNext()) {
    String key = keys.next();
    System.out.print(key+": "+address.get(key)+" ");
}
```

- Lambda expression

```
address.forEach((key,v)->System.out.print(key+": "+v+" "));
```

Collision of Hash Table

A situation in which the problem occurs

- h(key) = key % size

Key	hashCode(key)	Location (size=11)	Location (size=9)
AT	2099		
FR	2252		
DE	2177		
GR	2283		
IT	2347		
PT	2564		
SE	2642		

Resolving Collison

- Closed addressing
- Open addressing
  - Linear probing
  - Quadratic probing
  - Double hashing

Closed addressing

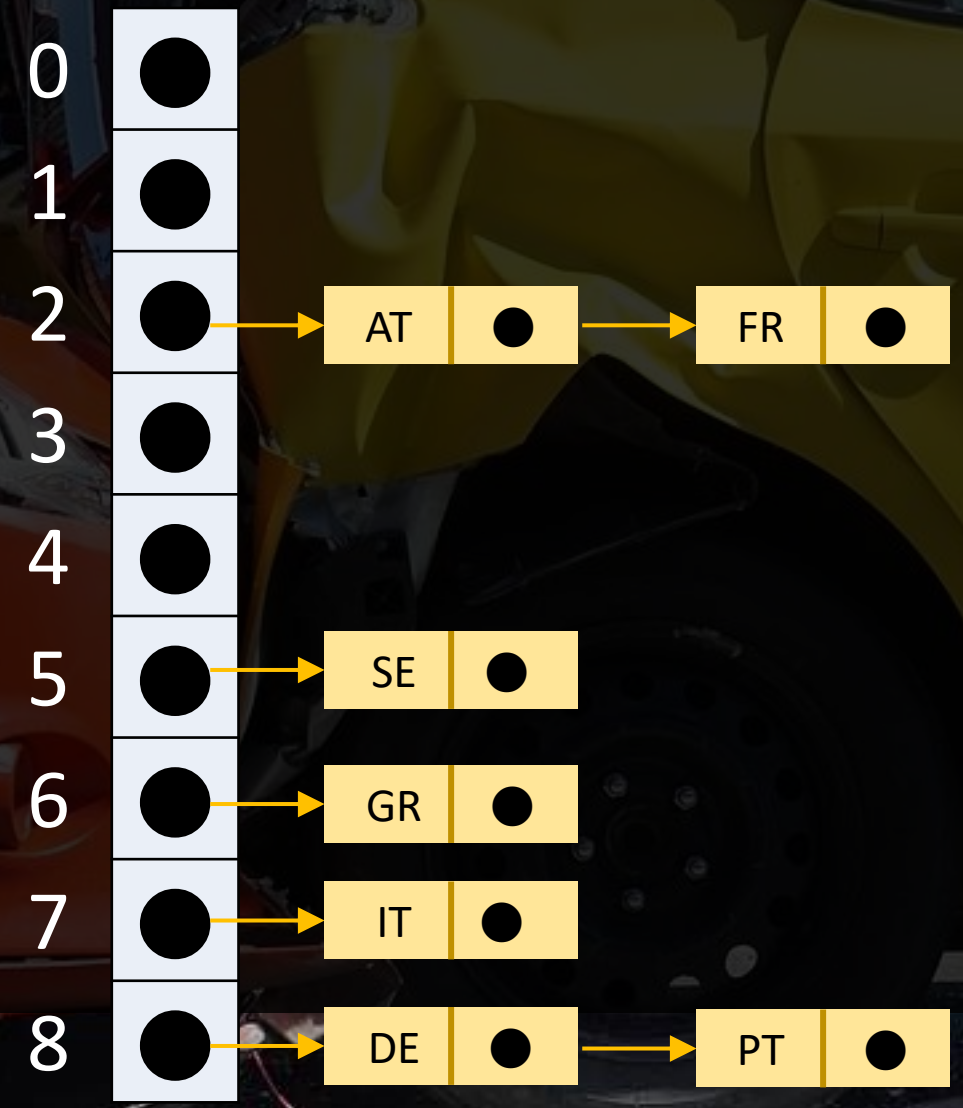
- It avoids collisions altogether by allowing more than one record to be stored at a hash location
- Separate chaining
  - Instead of an array of records, we use an array of buckets, where a bucket is a collection of records.
  - The simplest data structure to use for a bucket is a linked list.

Closed addressing

https://www.cs.usfca.edu/~galles/visualization/OpenHash.html

- Size = 9
- *h(key) = key % size*

Key	key	Location
AT	2099	2
FR	2252	2
DE	2177	8
GR	2283	6
IT	2347	7
PT	2564	8
SE	2642	5



Open addressing

https://www.cs.usfca.edu/~galles/visualization/ClosedHash.html

- It is also called open addressing, because an element is not always placed in the slot indexed by its value.
  - Linear probing
    - The simplest way to resolve collisions in a hash table is to put the colliding record in the next available cell in the array.
  - Quadratic probing
    - That algorithm resolves collisions by incrementing in increasingly greater steps, instead of by 1 each time. Instead of adding i to h after each collision, we add i*i. (Quadratic means square the variable.
  - Double hashing
    - Use a second, independent hash function to determine the probe sequence.

Linear probing

- Size = 9
- $h_i(k) = (h(k)+i) \% size$

Key	key	Location
AT	2099	2
FR	2252	2
DE	2177	8
GR	2283	6
IT	2347	7
PT	2564	8
SE	2642	5

0	
1	
2	AT FR
3	
4	
5	SE
6	GR
7	IT
8	DE PT

Problem: primary clustering

- Insertion: 76→93→40→47→10→55
- Size = 7
- $h_i(k) = (h(k)+i) \% size$

key	Location	probe
76		
93		
40		
47		
10	3	
54	5	

0	
3	
4	
5	
6	

Primary clustering:

If the hash function fails to distribute the records uniformly throughout the table, then linear probing can lead to long chains of records bunched together.

Quadratic probing

- Size = 9
- $h_i(k) = (h(k)+i^2)$ % size

Key	key	Location
AT	2099	2
FR	2252	2
DE	2	
GR	22	
IT	234	
PT	2564	3
SE	2642	5
KR	2609	8

0	
1	
2	AT FR
3	
6	GR
7	IT
8	DE KR

1*1=1➔(8+1)%9=0

2*2=4➔(8+4)%9=3

3*3 8+9)%9=8

8+16)%9=6

Secondary clustering:

ITwo different keys that hash to the same value will have the same probe sequence.

Double hashing

- Unlike linear probing and quadratic probing, the interval depends on the data, so that even values mapping to the same location have different bucket sequences; this minimizes repeated collisions and the effects of clustering.

$$h(i,k) = (h_1(k) + i \cdot h_2(k)) \mod |T|.$$

Example of double hashing

	insert(76)	insert(93)	insert(40)	insert(47)	insert(10)	insert(55)
	76%7 = 6	93%7 = 2	40%7 = 5	47%7 = 5	10%7 = 3	55%7 = 6
				5 - (47%5) = 3		5 - (55%5) = 5
0						
1				47	47	47
2		93	93	93	93	93
3					10	10
4						55
5			40	40	40	40
6	76	76	76	76	76	76
probes:	1	1	1	2	1	2

Performance of Hash Table

Load factor

- Number of items / physical size of the hash table
- Upper limit is typically set at around 75%~80%
- If the load factor is too big, probing occurs too often.
- If the size of a hash table become over the threshold, the size and the table is rehashed.

Key	key	Location
AT	2099	2
FR	2252	2
DE	2177	8
GR	2283	6
IT	2347	7
PT	2564	8
SE	2642	5
KR	2609	8

8*100/9 = 88.89

Perfect hash functions

- Perfect hash functions
  - Lookup table
  - the hash function is one-to-one on the set of all possible keys.
  - no collision.
  - If a perfect hash function is found, then open addressing is best.
  - This data structure is often called a lookup table.
  - wastes space
- Minimum perfect hash function:
  - One-to one
  - Onto function (100% load factor)


Thank you