



# Chapter 5

## Hashing

# 散列



- 散列函数(Hash function)
  - $\text{Address} = \text{hash}(\text{key})$

# 散列



- 散列函数 (Hash function)
  - $\text{Address} = \text{hash}(\text{key})$
- 散列表 (Hash Table )的设计目标

## 5.1 General Idea

- **Sequention search :  $O(n)$**
- **Binary search:  $O(\log_2 n)$**
- **hashing method:  $O(C)$**

**Address=hash(key)**

**also called : name-address function**

## 5.1 General Idea

- **Example:**

$$H(x) = x \% 7$$

$$x1 = 8849 \quad H(x1) = 8849 \% 7 = 1$$

$$x2 = 8850 \quad H(x2) = 8850 \% 7 = 2$$

$$x3 = 8851 \quad H(x3) = 8851 \% 7 = 3$$

$$\text{beta} = 66698465 \quad H(\text{beta}) = 66698465 \% 7 = 1$$

$$y2 = 8950 \quad H(y2) = 8950 \% 7 = 4$$

	name	type	address	link
0				
1	x1	float	1000	
2	x2	float	1004	
3	x3	float	1008	
4	y2	int	2000	
5				
6				



beta	int	2002	^
------	-----	------	---

## 5.1 General Idea

- **problems**

**Find a proper hash function**

**How to solve a collision**

**Select a suitable load factor  $\alpha$ .**

$$\underline{\alpha = n/b}$$

**n is number of elements in the hash table**

**b is the number of buckets in the hash table**

$\alpha > 1$  碰撞频率大

$\alpha < 1$  碰撞频率小

## 5.2 Hash Function

### 1. 取余法

$$H(\text{Key}) = \text{Key} \% M$$

其中： $M \leq$  基本区长度的最大质数

基本区长	M
------	---

8	7
---	---

16	13
----	----

2048	2039
------	------

为什么取最大质数？

1) 若取偶数，如 10, 100, ..., 2, 4, ..., 冲突率是比较大的；

2) 若取含有质因子的M，如  $M=21$  ( $3*7$ ) 含质因子3和7，对下面的例子：

key: 28 35 63 77 105

则 7 14 0 14 0 关键码中含质因子7的哈希值均为7的倍数。

### 2. 平方取中法

$H(\text{Key}) = \text{Key}^2$  的中间部分，其长度取决于表的大小。

设表长  $= 2^9 = (512)_{10}$  地址 000~777(八进制)

$(2061)_8$	<u>4310541</u>
------------	----------------

$(2062)_8$	<u>4314704</u>
------------	----------------

$(2161)_8$	<u>4734741</u>
------------	----------------

$(2162)_8$	<u>4741304</u>
------------	----------------

$(1100)_8$	<u>1210000</u>
------------	----------------



## 5.2 Hash Function

### 3. 乘法杂凑函数

$$H(\text{Key}) = \lfloor M * ((\phi * \text{Key}) \% 1) \rfloor$$

例：设表长 =  $2^9 = (512)_{10}$  地址 000~777(八进制)，则

$$H(1) = \lfloor 2^9 * (0.618)_{10} \rfloor = \lfloor 2^9 * (0.4743...)_{8} \rfloor = 474$$

## 5.2 Hash Function

有些书中的

### 1. Hash1:

to add up the ASCII( or Unicode ) value of the characters in the string.

```
public static int hash( String Key, int tableSize )  
{  int hashVal = 0;  
   for( int i = 0; i < Key.length( ); i++ )  
       hashVal += Key.charAt( i );  
   return hashVal % tableSize;  
}
```

**Example:**

Suppose TableSize = 10007,

Suppose all the keys are eight or fewer characters long,  $8 \times 127 = 1016$

hash function typically can only assume value between 0~1016 引起浪费

## 5.2 Hash Function

### 2. Hash2:

$$h_{\text{key}} = k_0 + 37k_1 + 37^2k_2 + \dots$$

```
public static int hash( String key, int tableSize ) // good hash function
{
    int hashVal = 0;
    for( int i = key.length()-1; i >= 0; i-- )
        hashVal = 37 * hashVal + key.charAt( i );

    hashVal %= tableSize;
    if( hashVal < 0 ) // 函数允许溢出, 这可能会引进负数
        hashVal += tableSize;
    return hashVal;
}
```

## 5.3 how to solve a collision

### —— linear Probing

碰撞的两个(或多个)关键码称为同义词, 即 $H(k_1) = H(k_2)$ ,  $k_1$ 不等于 $k_2$

## 1. Open Addressing

### 1) linear Probing

If  $\text{hash}(\text{key})=d$  and the bucket is already occupied then we will examine successive buckets  $d+1, d+2, \dots, m-1, 0, 1, 2, \dots, d-1$ , in the array

## 5.3 how to solve a collision

—— linear Probing

Example 1: a hash table with 11 buckets,

$$H(k) = k \% 11,$$

Then 80, 40, 65, 24, 58, 35

$$H(80) = 3, H(40) = 7, H(65) = 10,$$

$$H(24) = 2, H(58) = 3, H(35) = 2$$

## 5.3 how to solve a collision

—— linear Probing

			80				40			65
--	--	--	----	--	--	--	----	--	--	----

Ht    0    1    2    3    4    5    6    7    8    9    10

		24	80	58			40			65
--	--	----	----	----	--	--	----	--	--	----

Ht    0    1    2    3    4    5    6    7    8    9    10

		24	80	58	35		40			65
--	--	----	----	----	----	--	----	--	--	----

Ht    0    1    2    3    4    5    6    7    8    9    10

## 5.3 how to solve a collision

——linear Probing

Performance analysis

the adding sequence is 80,40,65,24,58,35

$$ASL_{succ} = (1+1+1+1+2+4)/6 = 10/6$$

## 5.3 how to solve a collision

### ———— linear Probing

example 2 :

keys : Burke, Ekers, Broad, Blum, Attlee, Hecht, Alton,  
Ederly

$\text{hash}(\text{key}) = \text{ord}(x) - \text{ord}('A')$

x为取key第一个字母在字母表中的位置。例如：

$\text{hash}(\text{Attlee}) = 0$

$H(\text{Burke}) = 1$  ,  $H(\text{Ekers}) = 4$  ,  $H(\text{Broad}) = 1$  ,  $H(\text{Blum}) = 1$  ,

$H(\text{Attlee}) = 0$  ,  $H(\text{Hecht}) = 7$  ,  $H(\text{Alton}) = 0$  ,  $H(\text{Ederly}) = 4$  ,

设散列表长  $m = 26$  ( 0~25 )

0	1	2	3	4	5	6	7	8		25
Attlee	Burke	Broad	Blum	Ekers	Alton	Ederly	Hecht	.....	.....	
1	1	2	3	1	6	3	1			

分析比较次数：

搜索成功的平均搜索长度

$$(1 + 1 + 2 + 3 + 1 + 1 + 6 + 3) * 1/8 = 18/8$$



## 5.3 how to solve a collision

### linear Probing

example 3 :

$\text{hash}(\text{key}) = \text{key} \% 10 ;$

{ 89, 18, 49, 58, 69 }

0	1	2	3	4	5	6	7	8	9
49	58	69						18	89
2	4	4						1	1

“clustering problem” 堆积----指不同的同义词表合为一张了。从而增加了插入，查找的时间。

## 5.3 how to solve a collision

### linear Probing

$$H(k) = k \% 11,$$

Then 80, 40, 65, 24, 58, 35

$$H(80) = 3, H(40) = 7, H(65) = 10,$$

$$H(24) = 2, H(58) = 3, H(35) = 2$$

		24	80	58	35		40			65
--	--	----	----	----	----	--	----	--	--	----

Ht    0    1    2    3    4    5    6    7    8    9    10

要删除58，如果真的删了，则后面要查找35就找不到了。

## 5.3 how to solve a collision

### ——linear Probing

#### C++ Implementation

- Assume that each element to be stored in the hash table is of type E and has a field key of type k.
- the hash table is implemented using two arrays: ht and empty .
- empty[i] is true iff ht[i] does not have an element in it. It is defined for the deletion operation

## 5.3 how to solve a collision

### ——linear Probing

```
template<class E,class K>class HashTable
{ public:
    HashTable(int divisor =11);
    ~HashTable() {delete[]ht; delete [] empty;}
    bool Search(const K&k ,E& e)const;
    HashTable<E,K>& Insert(const E&e);
private:
    int hSearch(const K& k)const;
    int D; //hash function divisor
    E *ht ; //hash table array
    bool *empty ; //1D array
};
```

## 5.3 how to solve a collision

### ——linear Probing

- **Constructor for hashtable**

```
template<class E,class K>
```

```
HashTable<E,K>::HashTable(int divisor)
```

```
{ D=divisor;
```

```
  ht=new E[D];
```

```
  empty= new bool[D];
```

```
  for(int i=0;i<D;i++)
```

```
    empty[i]=true;
```

```
}
```

## 5.3 how to solve a collision

### ——linear Probing

```
template<class E,class K>
int HashTable<E,K>::hSearch(const K&k)const
{ int i=k%D; //home bucket
  int j=i; //start at home bucket
  do
  { if(empty[j] || ht[j]==k) return j; //fit
    j=(j+1)%D; //next bucket
  } while(j!=i); //returned to home?
  return j; //table full;
}
```

## 5.3 how to solve a collision

### ——linear Probing

- Search function

```
Template<class E,class K>
```

```
bool HashTable<E,K>::Search(const  
    K&k,E&e)const
```

```
{//put element that matches k in e.
```

```
//return false if no match.
```

```
int b=hSearch(k);
```

```
if(empty[b]||Hash(ht[b])!=k)return false;
```

```
e=ht[b];
```

```
return true;
```

```
}
```

## 5.3 how to solve a collision

### ——linear Probing

- Insertion into a hash table

```
template<class E,class K>
HashTable<E,K>& HashTable<E,K>::Insert(const E& e)
{ K k=Hash(e); //extract key
  int b=hSearch(k);
  if(empty[b]){empty[b]=false; ht[b]=e;
               return *this;}
  if(ht[b]==k)throw BadInput();//duplicate
  throw NoMem(); //table full
}
```



## 5.3 how to solve a collision

### —— Quadratic probing

#### 2) Quadratic probing

If  $\text{hash}(k)=d$  and the bucket is already occupied then we will examine successive buckets  $d-1, d-2^2, d-3^2, \dots$ , in the array

example :

( 89, 18, 49, 58, 69 ) [个位数为key]

$\text{hash}(k) = k \% 10;$

0	1	2	3	4	5	6	7	8	9
49		58	69					18	89
2		3	3					1	1

$$\text{ASL}_{\text{succ}} = (1+1+2+3+3)/5$$

## 5.3 how to solve a collision

### —— Quadratic probing

- Java Implementation

element isActive

HashEntry



第一种情况: null

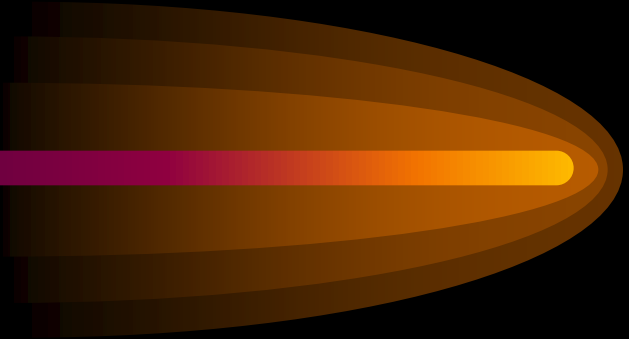
第二种情况: 非null且该项是活动的, isActive为true

第三种情况: 非null 且该项标记为被删除, isActive为false

```
public interface Hashable
{ int hash( int tableSize );
}

class HashEntry
{ Hashable element;
  boolean isActive;

  public HashEntry( Hashable e ) { this( e, true ) ; }
  public HashEntry( Hashable e, boolean i )
  { element = e;
    isActive = i;
  }
}
```



## 5.3 how to solve a collision

### —— Quadratic probing

```
public class QuadraticProbingHashTable
{
    public QuadraticProbingHashable()
    public QuadraticProbingHashable( int size )
    public void makeEmpty( )

    public Hashable find( Hashable x )
    public void insert( Hashable x )
    public void remove( Hashable x )

    public static int hash( String key, int tableSize )
    private static final int DEFAULT_TABLE_SIZE = 11;

    protected HashEntry [ ] array;
    private int currentSize;
```

## 5.3 how to solve a collision

### ——Quadratic probing

```
private void allocateArray(int arraySize )
```

```
private boolean isActive( int currentPos )
```

```
private int findPos( Hashable x )
```

```
private void rehash( )
```

```
private static int nextPrime( int n )
```

```
private static boolean isPrime( int n )
```

```
}
```

## 5.3 how to solve a collision

### —— Quadratic probing

- Constructor

```
public QuadraticProbingHashTable()  
{ this( DEFAULT_TABLE_SIZE );  
}  
  
public QuadraticProbingHashTable( int size )  
{ allocateArray( size );  
  makeEmpty();  
}
```

## 5.3 how to solve a collision

### ——Quadratic probing

- Some other function

```
private void allocateArray( int arraySize )  
{  array = new HashEntry[ arraySize ];  
}
```

```
public void makeEmpty()  
{  currentSize = 0;  
    for( int i = 0; i < array.length; i++ )  
        array[ i ] = null;  
}
```

## 5.3 how to solve a collision

### ——Quadratic probing

- Find function

```
public Hashable find( Hashable x )
{
    int currentPos = findPos( x );
    return isActive( currentPos ) ? array[ currentPos ].element : null ;
}

private int findPos( hashable x )
{
    int collisionNum = 0;
    int currentPos = x.hash( array.length );
    while( array[ currentPos ] != null &&
           !array[ currentPos ].element.equals( x ) )
    {
        currentPos += 2 * ++collisionNum - 1;
        if( currentPos >= array . length )
            currentPos -= array . length;
    }
    return currentPos;
}
```



## 5.3 how to solve a collision

### ——Quadratic probing

```
private boolean isActive( int currentPos )  
{  return array[ currentPos ] != null &&  
    array[ currentPos ].isActive;  
}
```

- Insert function

```
public void insert( Hashable x )  
{  int currentPos = findPos( x );  
    if( isActive( currentPos ) ) return;  
    array[ currentPos ] = new HashEntry( x, true );  
    if( ++currentSize > array . length / 2 )  
        rehash( );  
}
```

## 5.3 how to solve a collision

### ——Quadratic probing

- Remove function

```
public final void remove( Hashable x )  
{  int currentPos = findPos( x );  
    if( isActive( currentPos ) )  
        array[ currentPos ] . isActive = false;  
}
```

## 5.3 how to solve a collision

### ———— Double Hashing

#### 3) Double Hashing

If  $\text{hash}_1(k)=d$  and the bucket is already occupied then we will counting  $\text{hash}_2(k)=c$ , examine successive buckets  $d+c, d+2c, d+3c, \dots$ , in the array

**example:**

$$\text{hash1}(k) = k \% 10;$$

$$\text{hash2}(k) = R - (k \% R);$$

其中  $R < \text{TableSize}$  的质数

( 89, 18, 49, 58, 69 )

$$\text{hash2}(49)=7-49\%7=7 \quad (R=7)$$

$$\text{hash2}(58)=7-58\%7=5$$

$$\text{hash2}(69)=7-69\%7=1$$

0	1	2	3	4	5	6	7	8	9
69			58			49		18	89

# rehashing

example:

{ 13, 15, 24, 6 }

23

$h(x) = x \% 7;$

$23 \% 7 = 2$

$13 \% 7 = 6$

0

1

2

3

4

5

6

$15 \% 7 = 1$

6	15	23	24			13
---	----	----	----	--	--	----

$24 \% 7 = 3$

$6 \% 7 = 6$

当表项数 > 表的70% 时,可再散列.

即, 取比( $2 * \text{原表长} = 14$ )大的质数17再散列.

$6 \% 17 = 6, 15 \% 17 = 15, 23 \% 17 = 6, 24 \% 17 = 7, 13 \% 17 = 13$

# rehashing

0	6
1	15
2	23
3	24
4	
5	
6	13



0	
1	
2	
3	
4	
5	
6	6
7	23
8	24
9	
10	
11	
12	
13	13
14	
15	15

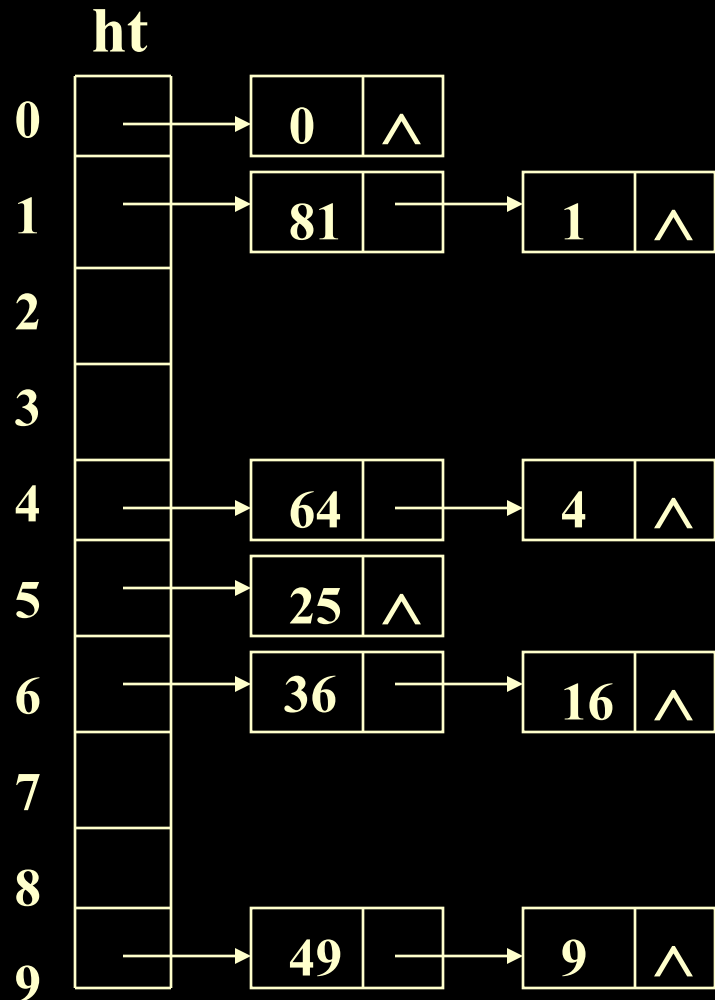
# rehashing

```
private void rehash( )  
{  HashEntry [ ] oldArray = array ;  
  
    allocateArray( nextPrime( 2* oldArray.length ) );  
    currentSize = 0;  
  
    for( int i = 0; i < oldArray.length; i++ )  
        if( oldArray[ i ] != null && oldArray[ i ] . isActive )  
            insert( oldArray[ i ] . Element );  
}
```

## 5.4 how to solve a collision

### Separate Chaining

#### 2. Separate Chaining



0, 1, 4, 9, 16, 25, 36, 49, 64, 81

$$\text{Hash}(x) = x \% 10$$

## 5.4 how to solve a collision

### —————Separate Chaining

```
public class SeparateChainingHashTable
{
    public SeparateChainingHashTable()
    public SeparateChainingHashTable( int size )
    public void insert( Hashable x )
    public void remove( Hashable x )
    public Hashable find( Hashable x )
    public void makeEmpty( )
    public static int hash( String key, int tableSize )

    private static final int DEFAULT_TABLE_SIZE = 101;

    private LinkedList [ ] theLists;

    private static int nextPrime( int n )
    private static boolean isPrime( int n )
}
```



## 5.4 how to solve a collision

### —— Separate Chaining

```
public interface Hashable  
{ int hash( int tableSize );  
}
```

```
public class Employee implements Hashable  
{ public int hash( int tableSize )  
    { return SeparateChainingHashTable.hash( name, tableSize ); }  
  public boolean equals( object rhs )  
    { return name.equals( ( Employee) rhs ).name ); }  
}
```

```
private String name;  
private double salary;  
private int seniority;
```

```
}
```

## 5.4 how to solve a collision

### ——Separate Chaining

```
public SeparateChainingHashTable()  
{ this( DEFAULT_TABLE_SIZE );  
}
```

```
public SeparateChainingHashTable( int size )  
{ theLists = new LinkedList[ nextPrime( size ) ];  
  for( int i = 0; i < theLists.length; i++ )  
    theLists[ i ] = new LinkedList();  
}
```

```
public void makeEmpty()  
{ for( int i = 0; i < theLists.length; i++ )  
  theLists[ i ].makeEmpty();  
}
```

## 5.4 how to solve a collision

### ———— Separate Chaining

```
public void remove( Hashable x )
{ theLists[ x.hash( theLists.length ) ].remove( x );
}

public Hashable find( Hashable x )
{ return ( Hashable ) theLists[ x.hash( theLists.length ) ]. Find( x ).
  Retrieve( );
}

public void insert( Hashable x )
{ LinkedList whichList = theLists[ x.hash( theLists.length ) ];
  LinkedListItr itr = whichList.find( x );

  if( itr.isPastEnd( ) )
    whichList.insert( x, whichList.zeroth( ) );
}
```

# Chapter 5

## exercises:

1. Given input { 4371, 1323, 6173, 4199, 4344, 9679, 1989 } and a hash function  $h(x) = x \pmod{10}$ , show the resulting:

- a. Separate chaining hash table.
- b. Hash table using linear probing.
- c. Hash table using quadratic probing.
- d. Hash table with second hash function  $h_2(x) = 7 - (x \pmod{7})$ .

2. 设散列表为HT[13]，散列函数为

$H(\text{key}) = \text{key} \% 13$ 。用线性开地址法解决冲突，对下列关键码序列 12,23,45,57,20,03,78,31,15,36：

- 1) 画出其散列表。
- 2) 计算等概率下搜索成功的平均搜索长度。
- 3) 如果采用链表散列解决冲突，画出该链表。