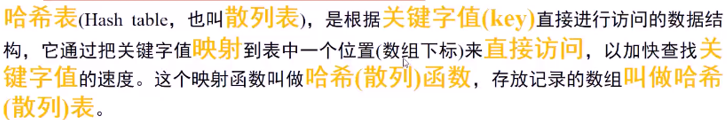
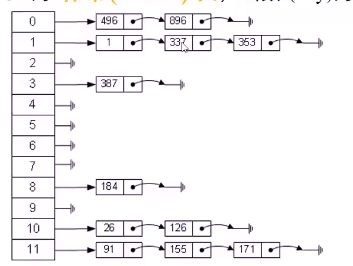
哈希表的基础知识

# 哈希表定义：HashTable



**哈希函数**：





# 最简单的哈希—字符哈希

/\*\*\*\*\*\*\*\*\*\*\*\*1.最简单的哈希：字符哈希\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\* 统计每个字符出现的次数 根据ASCII2 字符数128,1-127

\* 定义一个长度为128的int[]数组。

\* 若限定只是大写字母或小写字母，只需要定义长度为26的int[]数组；

\* **索引就是c-'A'或c-'a'**

\* 若同时含有大小写字母，创建一个长度为52的int[]数组，构建一个hash函数。

\*/

//求解字母的hash值

public int hash(char c){

if('A'<= c && c <='Z') return c-'A';

if('a'<= c && c <='z') return c-'a'+ 26;

return 53;//以防出现非字母字符

}

/\*\*

\* 0-127任意字符

\*/

public void countEveryChar(String str){

if(str == null || str.length() == 0) return;

int[] hashTable = new int[128];

char[] chs = str.toCharArray();

for(char c:chs){

hashTable[c]++;

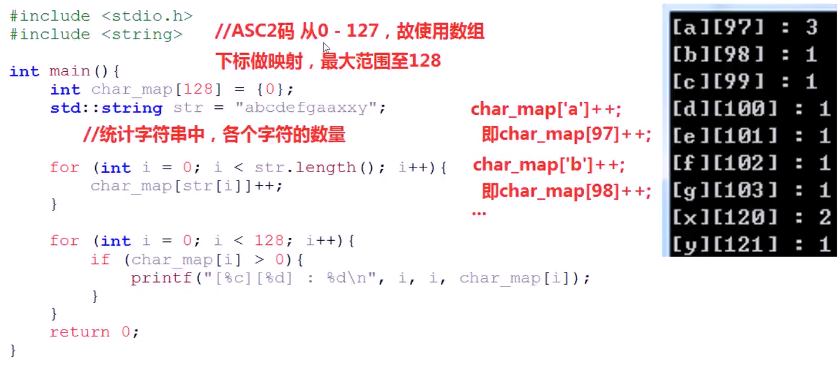
}

for(int i = 0;i < 128;i++){

System.out.println((char)i+" = " + hashTable[i]);

}

}



# 哈希表排序整数

/\*\*\*\*\*\*\*\*\*2. 利用哈希表对整数排序\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* **利用哈希表对整数排序的适用条件：**

\* 1. 不能对负数进行排序；

\* 2. 需要确定一定范围，即哈希表的长度要大于其最大数字；

\* 3. 当数字限定在小范围内，且有可能多次重复出现，适合用哈希表排序，如3,2,3,4,5,3,1,2,3,4,6,8,9,2,2

\* 而对于过于疏散的数据，利用哈希表排序，需要浪费较大的空间，如1,9999,2,3,89,21需要长度10000的哈希数组

\*/

public void sortByHashTable(int[] nums){

if(nums == null||nums.length <= 1) return;

int[] hashTable = new int[100];

for(int num:nums) hashTable[num]++;

int index = 0;

for(int i = 0;i < hashTable.length;i ++){//遍历哈希表

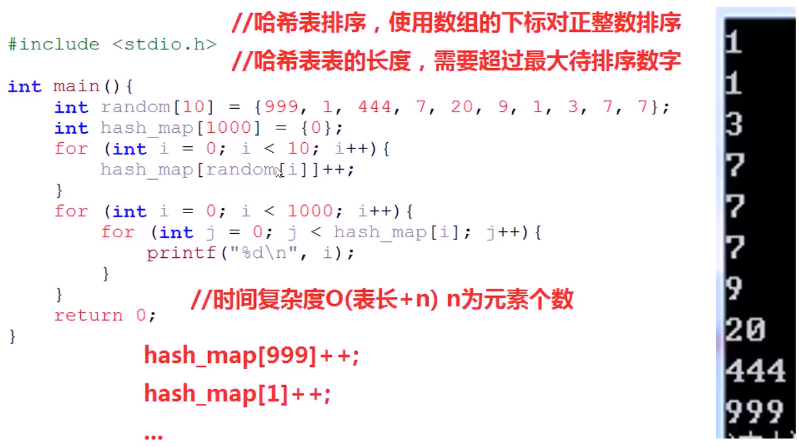
for(int j = 0;j < hashTable[i];j++){//重复的个数

nums[index++] = i;

}

}

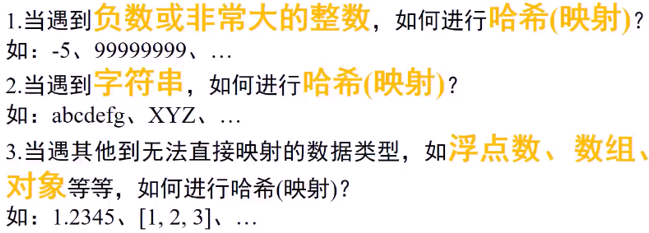
}

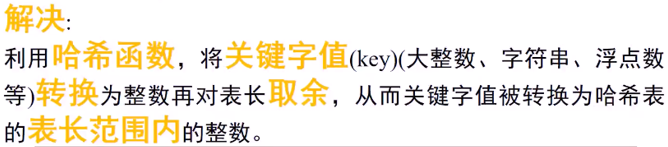


# 哈希函数算法介绍

## 哈希函数

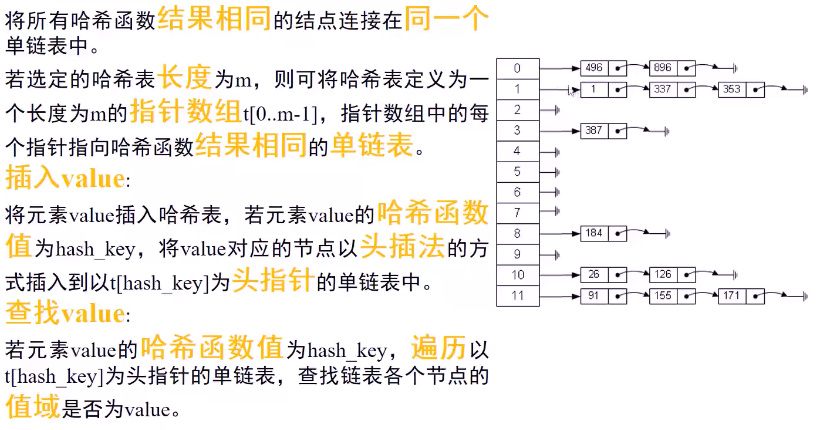
哈希函数：将关键字key转换为整数，再对表长取余。





## 哈希冲突及解决办法：拉链法解决冲突

头插法，插入链表的头部。



## 哈希表+链表实现：不存在映射关系

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3. 哈希表+链表实现\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* put和containsNum方法

\*/

class ListNode{

int val;

ListNode next;

public ListNode(int val){

this.val = val;

}

}

ListNode[] hashTable = new ListNode[8];//默认数组为8

/\*\*

\* put方法：若num已经存在，直接返回；

\* 否则添加该num

\*/

public void put(int num){

if(containsNum(num)) return;

int hashCode = hash(num);

//不存在，则头插法

ListNode newHead = new ListNode(num);

newHead.next = hashTable[hashCode];

hashTable[hashCode] = newHead;

}

/\*\*

\* containsNum判断是否存在num

\*/

public boolean containsNum(int num){

int hashCode = hash(num);

if(hashTable[hashCode] == null){

return false;

}else{

ListNode preHead = hashTable[hashCode];

while(preHead != null){

if(preHead.val == num) return true;

preHead = preHead.next;

}

return false;

}

}

//获取哈希值

public int hash(int num){

return num&7;//哈希数组长度8

}

public void printLinkedList(ListNode head){

StringBuilder sb = new StringBuilder();

while(head != null) {

sb.append(head.val).append("->");

head = head.next;

}

if(sb.length() >= 2) sb.delete(sb.length()-2,sb.length());

System.out.println("nodes : " + sb.toString());

}

//测试

@Test

public void test3(){

for(int i = 0;i < 16;i++)

put(i);

for(ListNode node : hashTable)

printLinkedList(node);

}

结果：

nodes : 8->0

nodes : 9->1

nodes : 10->2

nodes : 11->3

nodes : 12->4

nodes : 13->5

nodes : 14->6

nodes : 15->7

## HashMap的简单模拟实现

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3. 哈希映射的简单实现\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* put和containsNum方法

\*/

class MapListNode{

int key;

String val;

MapListNode next;

public MapListNode(int key,String val){

this.key = key;

this.val = val;

}

}

MapListNode[] hashMap = new MapListNode[4];//默认数组为8

/\*\*

\* put方法：若num已经存在，直接返回；

\* 否则添加该num

\*/

public void put(int key ,String val){

MapListNode node = new MapListNode(key,val);

int hashCode = hash2(key);

if(hashMap[hashCode]==null){

hashMap[hashCode] = node;

return;

}

//若存在

MapListNode preHead = hashMap[hashCode];

MapListNode tempNode = preHead;

while(tempNode != null) {

if (tempNode.key == key){

tempNode.val = node.val;//更新

return;

}

tempNode = tempNode.next;

}

//插入

node.next = preHead;

hashMap[hashCode] = node;

}

/\*\*

\* get

\*/

public String get(int key){

if(!containsKey(key)) return null;//不存在

int hashCode = hash2(key);

MapListNode head = hashMap[hashCode];

while(head != null){

if(head.key == key)

return head.val;

head = head.next;

}

return null;

}

/\*\*

\* containsNum判断是否存在num

\*/

public boolean containsKey(int key){

int hashCode = hash2(key);

if(hashTable[hashCode] == null){

return false;

}else{

MapListNode preHead = hashMap[hashCode];

while(preHead != null){

if(preHead.key == key) return true;

preHead = preHead.next;

}

return false;

}

}

//获取哈希值

public int hash2(int num){

return num&3;//哈希数组长度8

}

public void printLinkedList(MapListNode head){

StringBuilder sb = new StringBuilder();

while(head != null) {

sb.append(head.val).append("->");

head = head.next;

}

if(sb.length() >= 2) sb.delete(sb.length()-2,sb.length());

System.out.println("nodes : " + sb.toString());

}

//测试

@Test

public void test4(){

String[] strs = {"aa","bdc","tea","wetew","aa2","bdcdf","teasad",

"aabds","dsabdc","tsdfea","wetdsafew","aadsa2"};

for(int i = 0;i < strs.length;i++){

put(i,strs[i]);

}

for(MapListNode node : hashMap)

printLinkedList(node);

put(0,"AAAAAAA");

put(6,"DDDDDD");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

for(MapListNode node : hashMap)

printLinkedList(node);

}

结果：

nodes : dsabdc->aa2->aa

nodes : tsdfea->bdcdf->bdc

nodes : wetdsafew->teasad->tea

nodes : aadsa2->aabds->wetew

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

nodes : dsabdc->aa2->AAAAAAA

nodes : tsdfea->bdcdf->bdc

nodes : wetdsafew->DDDDDD->tea

nodes : aadsa2->aabds->wetew

# LeetCode算法题目

