

# 河南工业大学 数据结构 实验报告

课程名称	数据结构	实验项目	实验五 哈希表的算法实现
院 系	信息学院计科系	专业班级	计科 xxxxx 班
姓 名	cos	学 号	XXXXXXXXXXXXXX
指导老师	XXXXXX	日 期	2020.11.23
批改日期		成 绩	

## 一 实验目的

掌握哈希表的构造，以及解决冲突的方法——开放冲突法、链地址法等。

## 二 实验内容及要求

### 实验内容：

采用除留余数法实现哈希表的创建，任意采用一种处理冲突的方法解决冲突，计算哈希表的平均查找长度。编程实现以下功能：

已知一组关键字 (19, 14, 23, 1, 68, 20, 84, 27, 55, 11, 10, 79)，哈希函数定义为： $H(\text{key}) = \text{key} \text{ MOD } 13$ ，哈希表长为  $m=16$ 。实现该哈希表的散列，并计算平均查找长度（设每个记录的查找概率相等）。

- (1) 哈希表定义为定长的数组结构；
- (2) 使用线性探测再散列或链地址法解决冲突；
- (3) 散列完成后在屏幕上输出数组内容或链表；
- (4) 输出等概率查找下的平均查找长度；
- (5) 完成散列后，输入关键字完成查找操作，要分别测试查找成功与查找不成功两种情况。

### 实验要求：

1. 键盘输入数据；
2. 屏幕输出运行结果。
3. 要求记录实验源代码及运行结果。
4. 运行环境：CodeBlocks/Dev c++/VC6.0 等 C 编译环境

## 三 实验过程及运行结果

### (1) 算法设计思路

核心思想：散列表定义为一个结构体 HashTable，其中包括了表长与存放散列单元的数组，散列单元定义为 HashNode，其中包含了数据元素 data 和单元状态 flag（合法元素、空单元、已删除元素），这是由于散列表删除时元素并不能直接置为空，否则查找时会出问题。操作函数定义如下几个：创建一个长度为 TableSize 的空表（CreateTable）、哈希函数返回经散列函数计算后的下标（Hash）、查找 Key 元素（Find）、返回 Key 元素查找时的冲突次数（FindCnum）、插入 Key 到表中（Insert）、计算 ASL (CountASL)，均采用线性探测再散列法处理冲突。

```
#include <cmath>
#include <iostream>
#include <utility>
using namespace std;
const int HashSize = 13; //哈希函数取模用
const int MaxSize = 16; //表长
typedef int Index; //散列后的下标
//散列单元状态类型，分别对应：有合法元素、空单元、有已删除元素
typedef enum { Legitimate, Empty, Deleted } EntryType;
typedef int DataType; //数据的类型
struct HashNode { //散列表单元类型
    DataType data; //存放元素
    EntryType flag; //单元状态
};
struct HashTable { //散列表类型
    int TableSize; //表长
    HashNode* Units; //存放散列单元的数组
};
typedef HashTable* ptrHash;
```

## (2) 源程序代码

### 1. 创建一个长度为 TableSize 的空表 (CreateTable)

核心思想：根据 TableSize 给散列表单元数组分配合适空间，并将其初始类型全都置为空，返回指向该散列表的指针。

核心代码：//创建一个长度为TableSize的空表

```
ptrHash CreateTable(int TableSize) {  
    ptrHash H;  
    int i;  
    H = new HashTable;  
    H->TableSize = TableSize;  
    H->Units = new HashNode[H->TableSize];  
    for (int i = 0; i < H->TableSize; ++i) {  
        H->Units[i].flag = Empty;  
    }  
    return H;  
}
```

### 2. 返回经散列函数计算后的下标 (Hash)

核心思想：哈希函数定义为： $H(\text{key}) = \text{key} \text{ MOD } 13$ 。

核心代码：//返回经散列函数计算后的下标

```
inline Index Hash(DataType Key) {  
    return Key % HashSize;  
}
```

### 3. 查找 Key 元素，返回下标 (Hash)

核心思想：哈希表长为  $m=16$ ，故再散列时是模表长 16 而不是 13。

核心代码：//查找Key元素, 这里采用线性探测再散列处理冲突

```
Index Find(ptrHash H, DataType Key) {  
    Index nowp, newp;  
    int Cnum = 0; //记录冲突次数  
    newp = nowp = Hash(Key);  
    //若该位置的单元非空且不是要找的元素时发生冲突  
    while (H->Units[newp].flag != Empty && H->Units[newp].data != Key) {  
        ++Cnum; //冲突增加一次  
        newp = (nowp + Cnum) % H->TableSize;  
    }  
    return newp; //返回位置，该位置若为一个空单元的位置则表示找不到  
}
```

### 4. 查找 Key 元素，返回下标 (Find)

核心思想：哈希表长为  $m=16$ ，故再散列时是模表长 16 而不是 13。设两个指针，nowp 记录 Key 元素散列

后的下标值，newp 记录冲突处理后的下标，当单元非空单元并且不是要找的元素是发生冲突，记录冲突次数并将 newp 值向下一个试探。

核心代码：//查找Key元素,这里采用线性探测再散列处理冲突

```
Index Find(ptrHash H, DataType Key) {
    Index nowp, newp;
    int Cnum = 0; //记录冲突次数
    newp = nowp = Hash(Key);
    //若该位置的单元非空且不是要找的元素时发生冲突
    while (H->Units[newp].flag != Empty && H->Units[newp].data != Key) {
        ++Cnum; //冲突增加一次
        newp = (nowp + Cnum) % H->TableSize;
    }
    return newp; //返回位置，该位置若为一个空单元的位置则表示找不到
}
```

## 5. 返回冲突次数 (FindCnum)

核心思想：与 4 大体相同，不过返回冲突次数

核心代码：

```
Index FindCnum(ptrHash H, DataType Key) {
    Index nowp, newp;
    int Cnum = 0; //记录冲突次数
    newp = nowp = Hash(Key);
    //若该位置的单元非空且不是要找的元素时发生冲突
    while (H->Units[newp].flag != Empty && H->Units[newp].data != Key) {
        ++Cnum; //冲突增加一次
        newp = (nowp + Cnum) % H->TableSize;
    }
    return Cnum; //返回冲突次数
}
```

## 6. 插入 Key 到表中 (Insert)

核心思想：返回插入成功与否。先由 Find 函数找到该放的位置，然后只要这个单元能放入新的合法元素就将其放入，否则该键值已存在，插入失败。

核心代码：//插入Key到表中 返回插入成功与否

```
bool Insert(ptrHash H, DataType Key) {
    Index p = Find(H, Key);
    if (H->Units[p].flag !=
        Legitimate) { //只要这个单位能放入新元素（有已删除元素或空单元）
        H->Units[p].flag = Legitimate;
        H->Units[p].data = Key;
        return true;
    }
    else { //该键值已存在
```

```

        return false;
    }
}

```

## 7. 计算 ASL(CountASL)

核心思想：计算冲突次数，并根据冲突次数得出查找成功比较次数（冲突次数+1），加起来除以总数就是平均查找成功长度（因为各元素出现概率相同）

核心代码：

```

int a[12] = {19, 14, 23, 1, 68, 20, 84, 27, 55, 11, 10, 79};
int cnum[12]; //冲突次数, +1即为查找成功比较次数
int cnt, sum; //比较次数、比较总数（算ASL）
int t;        //待查找元素

void CountASL(ptrHash H) {
    //冲突次数
    for (int i = 0; i < MaxSize; ++i) {
        if (H->Units[i].flag == Empty)
            printf("Null\t");
        else {
            cnum[cnt] = FindCnum(H, H->Units[i].data);
            printf("%d\t", cnum[cnt]);
            ++cnt;
        }
    }

    cout << endl << "ASL = ";
    for (int i = 0; i < 12; ++i) {
        if (i == 11)
            cout << cnum[i] + 1 << " ";
        else {
            cout << cnum[i] + 1 << " + ";
        }
        sum += cnum[i] + 1;
    }
    cout << " / 12 = " << sum / 12.0 << endl;
}

```

## 8. 主函数

核心思想：按题目要求测试各功能，计算 ASL。

核心代码：

```

int main() {
    ptrHash H = CreateTable(MaxSize);
    for (int i = 0; i < 12; ++i) {
        Insert(H, a[i]);
    }
}

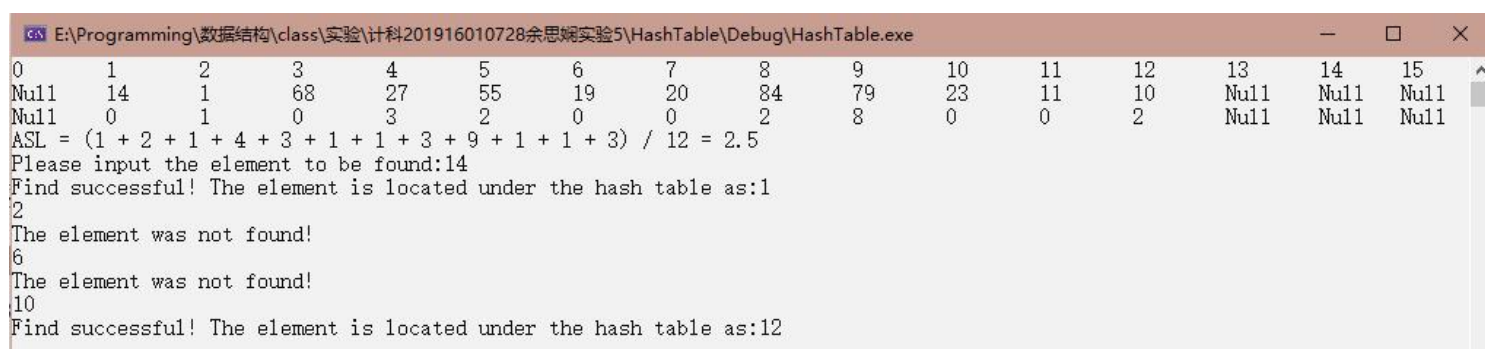
```

```

//下标
for (int i = 0; i < MaxSize; ++i) printf("%d\t", i);
cout << endl;
//数据
for (int i = 0; i < MaxSize; ++i) {
    if (H->Units[i].flag == Empty) printf("Null\t");
    else printf("%d\t", H->Units[i].data);
}
cout << endl;
CountASL(H);
cout << "Please input the element to be found:";
while (cin >> t) {
    int i = Find(H, t);
    if (H->Units[i].flag == Empty)
        cout << "The element was not found!" << endl;
    else cout << "Find successful! The element is located under the hash table as:" << i << endl;
}
return 0;
}

```

### (3) 运行结果截图



```

E:\Programming\数据结构\class\实验\计科201916010728余思妮实验5\HashTable\Debug\HashTable.exe
0      1      2      3      4      5      6      7      8      9      10     11     12     13     14     15
Null   14     1      68     27     55     19     20     84     79     23     11     10     Null   Null   Null
Null   0      1      0      3      2      0      0      2      8      0      0      2      Null   Null   Null
ASL = (1 + 2 + 1 + 1 + 4 + 3 + 1 + 1 + 3 + 9 + 1 + 1 + 3) / 12 = 2.5
Please input the element to be found:14
Find successful! The element is located under the hash table as:1
2
The element was not found!
6
The element was not found!
10
Find successful! The element is located under the hash table as:12

```

## 四 调试情况、设计技巧及体会

通过本次实验，我掌握了哈希表的构造，以及如何用线性探测再散列法解决冲突、如何计算 ASL 等，过程中并未遇到逻辑上的 bug。散列查找是在记录的存储位置和它的关键字之间建立一个确定的对应关系即哈希函数  $f$ ，使得每个关键字  $key$  对应一个存储位置  $f(key)$ 。而所谓的开放定址法就是一旦发生了冲突，就去寻找下一个空的散列地址，只要散列表足够大，空的散列地址总能找到，并将记录存入。

这次实验我使用的是线性探测再散列法，来解决对于给定的一组整数和散列函数，处理冲突构造散列表问题。再建了一个数组用来存放比较次数，同时也方便了计算 ASL。（比较次数是散列后下标对应元素的比较次数，表中第三行）