《数据结构》上机报告

2018 年 12 月 18 日

姓名：刘思源 学号：1651390 班级：电子三班 得分：

<table>
<tr><td>实验题目</td><td colspan="2">哈希表</td></tr>
<tr><td>问题描述</td><td colspan="2">哈希表（hash table,散列表）是一种用于以常数平均时间执行插入、删除和查找的查找表，其基本思想是：找到一个从关键字到查找表的地址的映射 h（称为散列函数），将关键字 key 的元素存到 h(key)所指示的存储单元中。当两个不相等的关键字被散列到同一个值时称为冲突，产生冲突的两个（或多个）关键字称为同义词，冲突处理的方法主要有：开放定址法，再哈希法，链地址法。</td></tr>
<tr><td rowspan="2">基本要求</td><td colspan="2">1. (p1) 本题针对字符串设计哈希函数。假定有一个班级的人名名单，用汉语拼音（英文字母）表示。
要求：首先把人名转换成整数，采用函数 h(key)=key[0]＊37^(n-1)+key[1]＊37^(n-2)+...+key[n-2]＊37+key[n-1]，其中 key[i]表示人名从左往右的第 i 个字母(i 从 0 计数,字符串长度为 n)，例如字母 a 和 A 的序号都是 1。
采取除留余数法（模是 M）将整数映射到长度为 p 的散列表中，h(key)=h(key)%M;
采用线性探测法解决冲突；
注意：计算 h(key)时会发生溢出，需要先取模再计算。</td></tr>
<tr><td>已完成基本内容（序号）：</td><td>1</td></tr>
<tr><td rowspan="2">选做要求</td><td colspan="2"></td></tr>
<tr><td>已完成选做内容（序号）</td><td></td></tr>
<tr><td>数据结构设计</td><td colspan="2">

```
typedef string ElemType;

typedef struct Node {
    ElemType value;
    int choosed;
} *PNode;

typedef struct HashTable {
    int size;
    PNode arr;
} *PHashTable;
```

</td></tr>
</table>

功能(函数)说明	1、 哈希表初始化 使用一个动态分配的数组来模拟哈希表。首先分配需要大小的数组，然后将哈希表的长度初始化，再将哈希表中所有的元素均设为空字符串（因为本题中给出的元素为人名，即字符串）。再将查找次数的整型数设置为 0。 2、 散列计算 散列计算最主要的是要进行溢出判断。首先判断输入的字符是大写字母还是小写字母，将他们统一转换为与之对应的数字。接下来进行散列公示的计算，注意每一次乘以 37 的 n 次方都有可能溢出。所以不应该直接调用 pow 函数，而是应该使用一个循环进行判断。如果当前的数大于 INT_MAX 的 37 分之一，那么接下来的计算一定会溢出。将现在的数 mod37 再进行计算。 注意判断是使用除法，如果判断 value*37 和 INT_MAX 的关系那在判断时就已经溢出了。

```
PHashTable init(int P)
{
    PHashTable table = (PHashTable)malloc(sizeof(struct HashTable));
    table->size = P;
    table->arr = new Node[P];
    int i;
    for (i = 0; i < P; i++) {
        table->arr[i].choosed = 0;
        table->arr[i].value = "";
    }
    return table;
}
```

```
int hashcal(string key, int M)
{


    unsigned int i;
    int value = 0;
    for (i = 0; i<key.length(); i++) {
        if (key[i] >= 97) {
            key[i] -= 96;
        }
        else {
            key[i] -= 64;
        }
        //value += (int)(key[i] * pow(37, key.length() - 1 - i));
        int len = key.length() - 1 - i;
        int temp = key[i];
        while (len) {
            if (temp>= 2147483647 / 37) {
                temp %= M;
            }
            temp *= 37;
```

```
            len--;
        }
        temp %= M;
        value += temp;



        value %= M;
    }
    return value;
}
```

3、 插入与线性探测

Insert 函数实现每一个元素的插入，同时进行冲突的检测。首先调用散列计算函数得到插入的位置，接下来如果这个位置的 choosed 为 1，即已经被选择，那么根据线性检测的原则，递增向后面的元素进行检测。直到找到 choosed 为 0 的状况，将值赋进去，将 choosed 赋为 1。注意在线性递增的时候有可能会超出数组的边界，所有要及时的使用 mod 回到数组的开始。

```
int insert(PHashTable table, string key, int M)
{
    int index = hashcal(key, M);
    while (table->arr[index].choosed
        && table->arr[index].value != key && table->arr[index].value != "") {
        table->arr[index].choosed++;
        index++;
        index = index % table->size;
    }
    if (!table->arr[index].choosed) {
        table->arr[index].value = key;
        table->arr[index].choosed = 1;
    }
    return index;
}
```

4、 哈希查找

哈希查找的思想和插入的思想一致。首先调用散列计算函数得到插入的位置，接下来如果这个位置的元素就是搜求的，返回 1。否则，根据线性检测的原则，递增向后面的元素进行检测，每次检测一个将计数器加一，最后返回计数器。

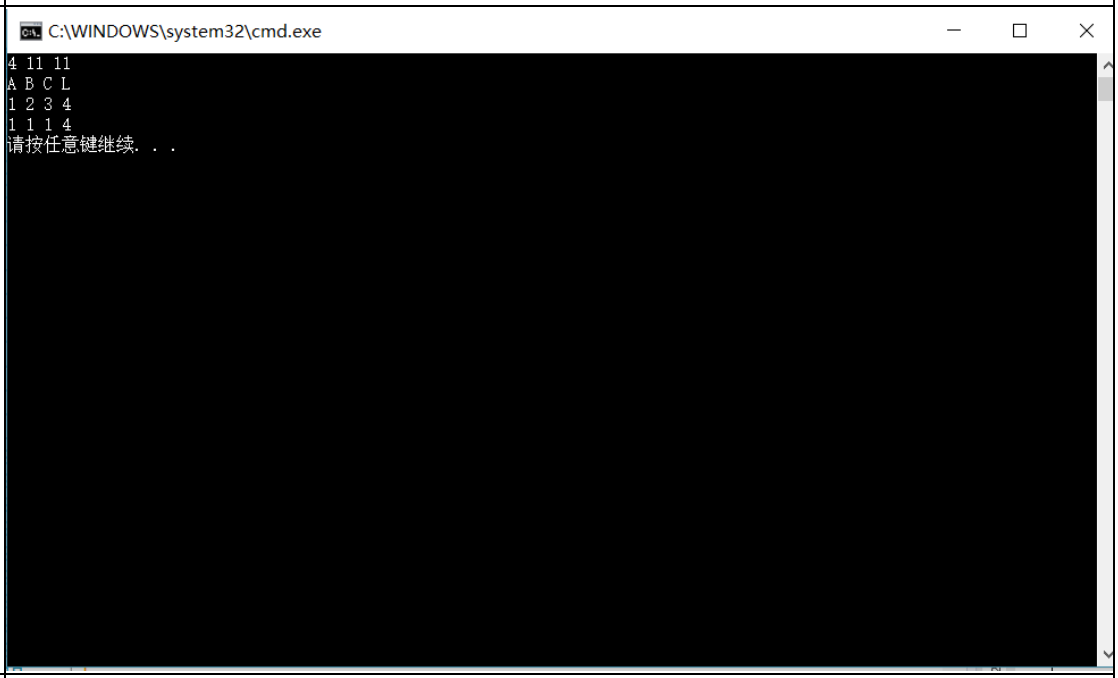
```
int hashFind(PHashTable table, string key, int M, int P)
{
    int index = hashcal(key, M);
    int count = 1;
    while (index<P) {
        if (table->arr[index].value == key && table->arr[index].value != "") {
            return count;
        }
        else {
```

```
            index++;
            index = index % table->size;
            count++;
        }
    }
    return -1;
}
```

开发环境

Visual studio 2017

调试分析



心得体会

1、 判断溢出是注意使用除法，否则在判断条件中就会出现溢出。

2、 在进行溢出的计算时注意要使用一个整型的变量 temp，不能直接使用 char 类型的 key[i]来进行计算，这样会超出 char 的范围，带来不可预知的结果。