**数据结构实验报告**

**院系：信息工程学院**

专业：软件工程二班

学号：20122480212

姓名：霍静静

**哈希表的实验报告**

**一、实验目的**：熟练掌握哈希表的构造方法及处理冲突的方法；

**二、实验要求**：

1）设计一个合理的哈希函数，构造一个哈希表；

2）熟练掌握哈希表处理冲突的几种方法；

3）熟练掌握哈希表的查找与插入的算法；

**三、实验内容：**

哈希表的构造，本实验我是采用开放定址法处理法的线性探测再散列来处理冲突的；

**四、算法思想：**

构造哈希函数的方法有：直接定址法，数字分析法，平方取中法，折叠法，除留余数法，本实验采用除六余数法；

哈希表处理冲突的方法有：开放定址法（线性探测再散列，二次探测再散列，随机探测再散列），链地址法，再哈希法，建立一个公共溢出区法，本实验采用线性探测再散列法；

**五、主要函数的实现：**

**#include <stdio.h>**

**#include <malloc.h>**

**#define NULLKEY 0 // 0为无记录标志**

**#define N 10 // 数据元素个数**

**typedef int KeyType;// 设关键字域为整型**

**typedef struct**

**{**

**KeyType key;**

**int ord;**

**}ElemType; // 数据元素类型**

**int hashsize[]={11,19,29,37}; // 哈希表容量递增表，一个合适的素数序列**

**int m=0; // 哈希表表长，全局变量**

**typedef struct**

**{**

**ElemType \*elem; // 数据元素存储基址，动态分配数组**

**int count; // 当前数据元素个数**

**int sizeindex; // hashsize[sizeindex]为当前容量**

**}HashTable;**

**#define SUCCESS 1**

**#define UNSUCCESS 0**

**#define DUPLICATE -1**

**// 构造一个空的哈希表**

**int InitHashTable(HashTable \*H)**

**{**

**int i;**

**(\*H).count=0; // 当前元素个数为0**

**(\*H).sizeindex=0; // 初始存储容量为hashsize[0]**

**m=hashsize[0];**

**(\*H).elem=(ElemType\*)malloc(m\*sizeof(ElemType));**

**if(!(\*H).elem)**

**exit(0); // 存储分配失败**

**for(i=0;i<m;i++)**

**(\*H).elem[i].key=NULLKEY; // 未填记录的标志**

**return 1;**

**}**

**// 销毁哈希表H**

**void DestroyHashTable(HashTable \*H)**

**{**

**free((\*H).elem);**

**(\*H).elem=NULL;**

**(\*H).count=0;**

**(\*H).sizeindex=0;**

**}**

**// 一个简单的哈希函数(m为表长，全局变量)**

**unsigned Hash(KeyType K)**

**{**

**return K%m;**

**}**

**// 开放定址法处理冲突**

**void collision(int \*p,int d) // 线性探测再散列**

**{**

**\*p=(\*p+d)%m;**

**}**

**// 算法9.17**

**// 在开放定址哈希表H中查找关键码为K的元素,若查找成功,以p指示待查数据**

**// 元素在表中位置,并返回SUCCESS;否则,以p指示插入位置,并返回UNSUCCESS**

**// c用以计冲突次数，其初值置零，供建表插入时参考。**

**int SearchHash(HashTable H,KeyType K,int \*p,int \*c)**

**{**

**\*p=Hash(K); // 求得哈希地址**

**while(H.elem[\*p].key!=NULLKEY&&!(K == H.elem[\*p].key))**

**{**

**// 该位置中填有记录．并且关键字不相等**

**(\*c)++;**

**if(\*c<m)**

**collision(p,\*c); // 求得下一探查地址p**

**else**

**break;**

**}**

**if (K == H.elem[\*p].key)**

**return SUCCESS; // 查找成功，p返回待查数据元素位置**

**else**

**return UNSUCCESS; // 查找不成功(H.elem[p].key==NULLKEY)，p返回的是插入位置**

**}**

**int InsertHash(HashTable \*,ElemType); // 对函数的声明**

**// 重建哈希表**

**void RecreateHashTable(HashTable \*H) // 重建哈希表**

**{**

**int i,count=(\*H).count;**

**ElemType \*p,\*elem=(ElemType\*)malloc(count\*sizeof(ElemType));**

**p=elem;**

**printf("重建哈希表\n");**

**for(i=0;i<m;i++) // 保存原有的数据到elem中**

**if(((\*H).elem+i)->key!=NULLKEY) // 该单元有数据**

**\*p++=\*((\*H).elem+i);**

**(\*H).count=0;**

**(\*H).sizeindex++; // 增大存储容量**

**m=hashsize[(\*H).sizeindex];**

**p=(ElemType\*)realloc((\*H).elem,m\*sizeof(ElemType));**

**if(!p)**

**exit(0); // 存储分配失败**

**(\*H).elem=p;**

**for(i=0;i<m;i++)**

**(\*H).elem[i].key=NULLKEY; // 未填记录的标志(初始化)**

**for(p=elem;p<elem+count;p++) // 将原有的数据按照新的表长插入到重建的哈希表中**

**InsertHash(H,\*p);**

**}**

**// 算法9.18**

**// 查找不成功时插入数据元素e到开放定址哈希表H中，并返回1；**

**// 若冲突次数过大，则重建哈希表。**

**int InsertHash(HashTable \*H,ElemType e)**

**{**

**int c,p;**

**c=0;**

**if(SearchHash(\*H,e.key,&p,&c)) // 表中已有与e有相同关键字的元素**

**return DUPLICATE;**

**else if(c<hashsize[(\*H).sizeindex]/2) // 冲突次数c未达到上限,(c的阀值可调)**

**{**

**// 插入e**

**(\*H).elem[p]=e;**

**++(\*H).count;**

**return 1;**

**}**

**else**

**RecreateHashTable(H); // 重建哈希表**

**return 0;**

**}**

**// 按哈希地址的顺序遍历哈希表**

**void TraverseHash(HashTable H,void(\*Vi)(int,ElemType))**

**{**

**int i;**

**printf("哈希地址0～%d\n",m-1);**

**for(i=0;i<m;i++)**

**if(H.elem[i].key!=NULLKEY) // 有数据**

**Vi(i,H.elem[i]);**

**}**

**// 在开放定址哈希表H中查找关键码为K的元素,若查找成功,以p指示待查数据**

**// 元素在表中位置,并返回SUCCESS;否则,返回UNSUCCESS**

**int Find(HashTable H,KeyType K,int \*p)**

**{**

**int c=0;**

**\*p=Hash(K); // 求得哈希地址**

**while(H.elem[\*p].key!=NULLKEY&&!(K == H.elem[\*p].key))**

**{ // 该位置中填有记录．并且关键字不相等**

**c++;**

**if(c<m)**

**collision(p,c); // 求得下一探查地址p**

**else**

**return UNSUCCESS; // 查找不成功(H.elem[p].key==NULLKEY)**

**}**

**if (K == H.elem[\*p].key)**

**return SUCCESS; // 查找成功，p返回待查数据元素位置**

**else**

**return UNSUCCESS; // 查找不成功(H.elem[p].key==NULLKEY)**

**}**

**void print(int p,ElemType r)**

**{**

**printf("address=%d (%d,%d)\n",p,r.key,r.ord);**

**}**

**int main()**

**{**

**ElemType r[N] = {**

**{17,1},{60,2},{29,3},{38,4},{1,5},**

**{2,6},{3,7},{4,8},{60,9},{13,10}**

**};**

**HashTable h;**

**int i, j, p;**

**KeyType k;**

**InitHashTable(&h);**

**for(i=0;i<N-1;i++)**

**{**

**// 插入前N-1个记录**

**j=InsertHash(&h,r[i]);**

**if(j==DUPLICATE)**

**printf("表中已有关键字为%d的记录，无法再插入记录(%d,%d)\n",**

**r[i].key,r[i].key,r[i].ord);**

**}**

**printf("按哈希地址的顺序遍历哈希表:\n");**

**TraverseHash(h,print);**

**printf("请输入待查找记录的关键字: ");**

**scanf("%d",&k);**

**j=Find(h,k,&p);**

**if(j==SUCCESS)**

**print(p,h.elem[p]);**

**else**

**printf("没找到\n");**

**j=InsertHash(&h,r[i]); // 插入第N个记录**

**if(j==0) // 重建哈希表**

**j=InsertHash(&h,r[i]); // 重建哈希表后重新插入第N个记录**

**printf("按哈希地址的顺序遍历重建后的哈希表:\n");**

**TraverseHash(h,print);**

**printf("请输入待查找记录的关键字: ");**

**scanf("%d",&k);**

**j=Find(h,k,&p);**

**if(j==SUCCESS)**

**print(p,h.elem[p]);**

**else**

**printf("没找到\n");**

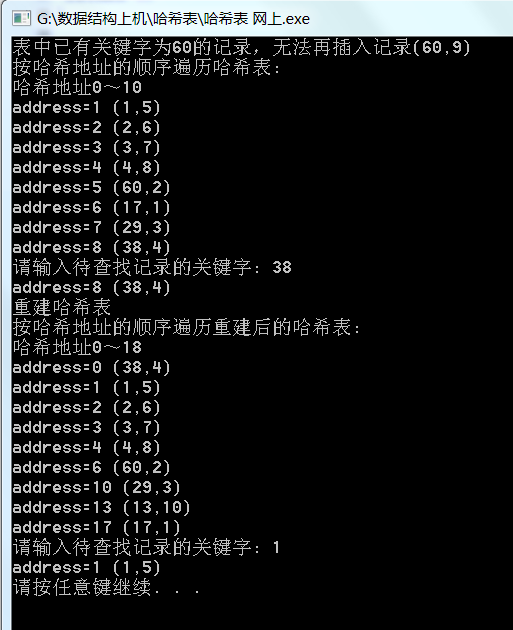
**DestroyHashTable(&h);**

**system("pause");**

**return 0;**

**}**

**六、测试结果：**



**七、实验总结：**

**对哈希表的构造，插入，查找算法不是很熟悉，重构哈希表那个算法也有一些问题，有待改善；**