山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

学号：202200130048	姓名： 陈静雯	班级： 6
实验题目：散列表		
实验学时：2	实验日期： 11.16	

实验目的：
用线性表和链表描述散列表

软件开发工具：
Vscode

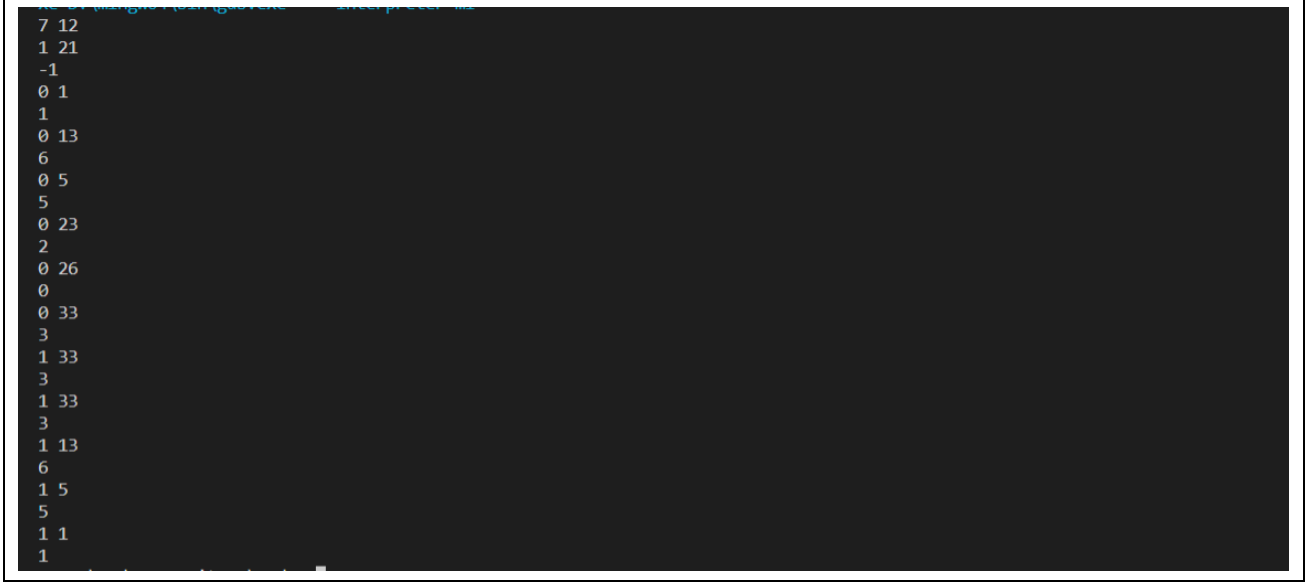
1. 实验内容
（1）给定散列函数的除数 D 和操作数 m，输出每次操作后的状态。
有以下三种操作：
插入 x，若散列表已存在 x，输出“Existed”，否则插入 x 到散列表中，输出所在的下标。
查询 x，若散列表不含有 x，输出“-1”，否则输出 x 对应下标。
删除 x，若散列表不含有 x，输出“Not Found”，否则输出删除 x 过程中移动元素的个数。
（2）给定散列函数的除数 D 和操作数 m，输出每次操作后的状态。
有以下三种操作：
插入 x，若散列表已存在 x，输出"Existed"
查询 x，若散列表不含有 x，输出"Not Found"，否则输出 x 所在的链表长度
删除 x，若散列表不含有 x，输出"Delete Failed"，否则输出 x 所在链表删除 x 后的长度

2. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）
（1）线性开型寻址
插入：查找元素所对应的位置，找到该位置及其后面位置中第一个空位，将其插入
查找：从元素对应的位置起，按顺序找到该元素，若找到返回下标，若找不到返回第一个空位下标，若队满返回-1
删除：从元素对应的位置起，找到要删除的位置，记为 pre，i 指向 pre 下一个元素，m 为 i 对应的元素本来应该在的位置，若三个下标大小顺序为 m pre i 或者为其循环顺序，因为散列表为循环表，满足条件，则将 i 指向的元素放入 pre，知道遍历整个表或 i 为 NULL 为止

（2）链表散列
插入：找到元素对应的链表位置（% 散列长度），遍历链表，找到比它大的第一个元素，插入该元素前面
查找：找到该元素对应的位置，遍历该位置指向的链表，找到该元素，若无输出“not found”
删除：找到链表中该元素的位置，将元素的前一个元素指向元素的后一个元素即可

3. 测试结果（测试输入，测试输出）


20 30
0 84
4
0 15
15
0 54
14
2 15
0
2 84
0
1 54
14
2 54
0
0 89
9
1 89
9
0 13
13
0 48
8
2 89
0
0 60
0
0 24
4
1 13
13
0 6
6
1 24
4
0 31
11
2 60
0
2 48
0
0 49
9
0 9
10
1 6


7 12
1 21
-1
0 1
1
0 13
6
0 5
5
0 23
2
0 26
0
0 33
3
1 33
3
1 33
3
1 13
6
1 5
5
1 1
1

```
    7 12
    1 21
Not Found
    0 1
    0 13
    0 5
    0 23
    0 26
    0 33
    1 33
    1 33
    1 13
1
    1 5
3
    1 1
1
```

```
    7 15
    2 10
Delete Failed
    0 10
    0 10
Existed
    2 10
0
    1 10
Not Found
    0 10
    1 10
1
    0 17
    0 2
    0 16
    0 11
    2 2
1
    2 10
1
    1 11
1
    1 17
1
```

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

删除的时候不只移动特征值相同的元素，按顺序遍历散列表，符合条件的元素都要移动，来填补空缺

5. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）

（1）

```
#include <iostream>
#include <utility>
using namespace std;
```

```
template<class K,class E>
class hashtable{
public:
    hashtable(int thedivisor);
    int search(const K& thekey);
    bool checknull(int b){
        return table[b]==NULL;
    }
    void insert(const K& thekey);
    void erase( K& thekey);
private:
    pair<const K,E>** table;
    int size;
    int divisor;
};

template<class K,class E>
hashtable<K,E>::hashtable(int thedivisor){     //初始化
    divisor=thedivisor;
    size=0;
    table = new pair<const K,E>* [divisor];
    for(int i=0;i<divisor;i++){
        table[i]=NULL;
    }
}

template<class K,class E>
int hashtable<K,E>::search(const K& thekey){    //查找
    int temp = thekey % divisor;
    int i=temp;
    do{
        if(table[i]==NULL||table[i]->first==thekey){       //table[i]为空即找不
到，返回空位置的下标
            return i;
        }
        i=(i+1)%divisor;
    }while(i!=temp);
    return -1;                          //-1 为表满
}

template<class K,class E>
void hashtable<K,E>::insert(const K& thekey){
    int b = search(thekey);
    if(b==-1){
        return ;
```

```
    }
    if(table[b]==NULL){                                    //table[b]空,即插入这个位
置
        table[b]=new pair<const K,E> (thekey,0);
        cout<<b<<endl;
        size++;
    }
    else if(table[b]->first==thekey){
        cout<<"Existed"<<endl;
    }
}

template<class K,class E>
void hashtable<K,E>::erase( K& thekey){
    int b = search(thekey);
    if(b==-1||table[b]==NULL){              //b 为-1，即表满且没找到该元素；b 为空
位置下标，也是找不到
        cout<<"Not Found"<<endl;
        return ;
    }
    else{
        size--;
        int num=0;
        int pre=b;
        int i=(b+1)%divisor;
        int m=0;
        while(i!=b){
            if(table[i]!=NULL) m = (int)table[i]->first % divisor;  //m 为该元素
本应在的位置
            if(table[i]==NULL){                                     //为空 退出
循环
                table[pre]=NULL;
                break;
            }
            else if(table[i]!=NULL && ((pre < i && i < m) || (i < m && m <= pre)
|| (m <= pre && pre < i) )){
                //有三种情况，都可以把 table[i]移到 table[pre]，因为表是循环插入
的， pre i m, i m pre , m pre i，即 m pre i 的循环顺序
                table[pre]=table[i];
                pre=i;
                num++;
            }
            i=(i+1)%divisor;
        }
        cout<<num<<endl;
    }
```

```
}

int main(){
    int d,m;
    cin>>d>>m;
    hashtable<int,int> T(d);
    for(int i=0;i<m;i++){
        int p,x;
        cin>>p>>x;
        if(p==0) T.insert(x);
        if(p==1){
            int b = T.search(x);
            if(b==-1||T.checknull(b)){    //b 为-1 或者 table[b]为空都是找不到的
情况
                cout<<-1<<endl;
            }
            else{
                cout<<b<<endl;
            }
        }
        if(p==2) T.erase(x);
    }
}
 (2)
#include <iostream>
#include <utility>
using namespace std;

template<class K,class E>
struct pairnode{       //节点类
    pair<K,E> element;
    pairnode<K,E> *next;
    pairnode(){ }
    pairnode(const pair<K,E>& thelement){  //构造函数
        this->element=thelement;
    }
    pairnode(const pair<K,E>& element,pairnode<K,E>* next){
        this->element=element;
        this->next=next;
    }
};

template<class K,class E>
class sortlist{                          //链表类
public:
    sortlist(){
```

```
        firstnode=NULL;
        size=0;
    }
    void insert(const pair<K,E>& thelement);
    void erase(const K& thekey);
    void find(const K& thekey);
    ~sortlist(){
        while(firstnode!=NULL){
            pairnode<K,E>* temp=firstnode->next;
            delete firstnode;
            firstnode=temp;
        }
    }
private:
    pairnode<K,E>* firstnode;
    int size;
};

template<class K,class E>
void sortlist<K,E>::insert(const pair<K,E>& thelement){
    pairnode<K,E> *p = firstnode,                                    //tp 为 p 的前一
节点
                  *tp = NULL;
    while(p!=NULL && p->element.first < thelement.first){
        tp=p;
        p=p->next;
    }
    if(p!=NULL && p->element.first==thelement.first){
        cout<<"Existed"<<endl;
        return ;
    }
    pairnode<K,E>* newnode = new pairnode<K,E>(make_pair(thelement.first,0),p);
    if(tp==NULL) firstnode = newnode;
    else tp->next=newnode;
    size++;
}

template<class K,class E>
void sortlist<K,E>::erase(const K& thekey){
    pairnode<K,E> *p = firstnode,
                  *tp = NULL;
    while(p!=NULL && p->element.first < thekey){
        tp=p;
        p=p->next;
    }
    if(p!=NULL && p->element.first==thekey){
```

```
        if(tp==NULL) firstnode = p->next;
        else tp->next = p->next;
        delete p;
        size--;
        cout<<size<<endl;
    }
    else{
        cout<<"Delete Failed"<<endl;
    }
}

template<class K,class E>
void sortlist<K,E>::find(const K& thekey){
    pairnode<K,E> *p = firstnode,
                  *tp = NULL;
    while(p!=NULL && p->element.first < thekey){
        tp=p;
        p=p->next;
    }
    if(p!=NULL && p->element.first==thekey){
        cout<<size<<endl;
    }
    else{
        cout<<"Not Found"<<endl;
    }
}

template<class K,class E>
class listhash{
public:
    listhash(int n){
        divisor = n;
        table = new sortlist<K,E> [n];
    }
    void find(const K& thekey){
        table[thekey%divisor].find(thekey);
    }
    void insert(const pair<K,E>& thelement){
        table[thelement.first%divisor].insert(thelement);
    }
    void erase(const K& thekey){
        table[thekey%divisor].erase(thekey);
    }
private:
    sortlist<K,E>* table;
    int divisor;
```

```
};

int main(){
    int d,m;
    cin>>d>>m;
    listhash<int,int> T(d);
    for(int i=0;i<m;i++){
        int p,x;
        cin>>p>>x;
        if(p==0){
            pair<int,int> temp = make_pair(x,0);
            T.insert(temp);
        }
        if(p==1) T.find(x);
        if(p==2) T.erase(x);
    }
}
```