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　　：202200400053	姓名： 王宇涵	班级： 22 级 2 班
实验题目：散列表		
实验学时：2	实验日期：　　2023-11-01	

实验目的：

1.掌握散列表结构的定义和实现。

2.掌握散列表结构的应用。

软件开发环境：

VSCODE

1. 实验内容

1、题目描述：

给定散列函数的除数 D 和操作的次数 m，输出每次操作后的状态。

有以下三种操作：

1.插入 x，若散列表已含有 x 则输出“Existed”，否则向散列表中插入 x，

并输出所在的下标。

2.查询 x，若散列表不含有 x 则输出“－1”，否则输出 x 对应的下标。

3.删除 x，若散列表不含有 x 则输出“Not Found”，否则输出删除 x 过程

中移动元素的个数。

输入输出格式：

输入：第一行输入两个整数 D 和 m，分别代表散列函数的除数和操作的

次数。接下来 m 行，每行输入两个整数 opt 和 x，分别代表操作类型和操

作数。opt 为 0 时插入 x，opt 为 1 时查询 x，opt 为 2 时删除 x。

输出：按需输出。

2、题目描述：

给定散列函数的除数 D 和操作的次数 m，输出每次操作后的状态。

有以下三种操作：

1.插入 x，若散列表已含有 x 则输出“Existed”；

2.查询 x，若散列表不含有 x 则输出“Not Found”，否则输出 x 所在的链

表长度；

3.删除 x，若散列表不含有 x 则输出“Delete Failed”，否则输出 x 所在链

表删除 x 后的长度；

输入输出格式：

输入：第一行输入两个整数 D（1≤D≤3000）和 m（1≤m≤3000），分

别代表散列函数的除数和操作的次数。接下来的 m 行，每行输入两个整

数 opt 和 x，分别代表操作类型和操作数。opt 为 0 时插入 x，opt 为 1 时

查询 x，opt 为 2 时删除 x。

输出：按需输出。

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）**

题目一:线性探查

Hash 类：定义了一个用于整数键的哈希函数。

hashTable 类：表示哈希表数据结构。

构造函数初始化了哈希表，其中 table 是一个指向键-值对的指针数组，用于存储数据。

searchIndex 方法根据给定的键找到对应的桶的索引。

findPair 方法用于查找指定键的位置，并输出结果

insert 方法用于插入键-值对，如果位置已被占用，则输出 "Existed"，否则插入并输出位置。

erase 方法用于删除键对应的值，输出删除的位置数量。

main 函数

从输入读取哈希表桶数 d 和操作数 m。

使用 hashTable 类创建一个哈希表对象 m_hash。

在循环中，根据输入的操作类型（0：插入，1：查找，2：删除），执行相应的操作。

Erase 函数解析

首先，它通过调用 searchIndex 方法来查找指定键的索引位置 b，也即该键的理想位置。

然后，它使用 a 和 origin 来记录当前位置和初始位置，以便后续使用。

接下来，它初始化一个计数器 ct 用于记录删除的位置数量。

如果找到的位置为空或者键不匹配，说明没有找到要删除的键，此时输出 "Not Found"。

如果找到了要删除的键，首先将当前位置置为空（table[b] = NULL），表示删除。

然后，使用线性探测法，在哈希表中找到下一个非空位置，并检查是否需要交换位置。如果需要交换位置，将当前位置的值移到目标位置，并将目标位置置为空，然后更新 a 和 b，同时增加 ct 计数。

继续寻找下一个非空位置，直到回到起始位置 origin 或直到所有可能的位置都被检查。

最后，输出 ct，表示删除的位置数量。

题目二：链表散列

Hash 类：定义了一个用于对整数键进行哈希的哈希函数。

PairNode 结构体：表示链表中的节点，用于存储键-值对。每个节点包含一个 pair<K, E> 元素和指向下一个节点的指针。

SortedChain 类：是 Dictionary 接口的实现。

它使用排序链表来存储键-值对。

find 方法搜索键并在找到时打印链大小，如果未找到则打印 "Not Found"。

insert 方法将键-值对插入链中，如果键已存在则打印 "Existed"。

erase 方法从链中删除键-值对并打印新的链大小。

output 方法用于打印链中的元素。

<< 运算符被重载，以允许打印 SortedChain 对象。

HashChains 类：表示使用分离链接的哈希表。

它使用 Hash 类来计算给定键的哈希值。

empty 方法检查哈希表是否为空。

size 方法返回哈希表中键-值对的总数。

findPair 方法查找哈希表中的键并返回指向相应键对的指针（如果找到）。

insert 方法将键-值对插入哈希表的适当桶中，增加字典大小。

erase 方法从哈希表中删除键-值对。

output 方法用于打印哈希表中每个桶中的元素。

<< 运算符被重载，以允许打印 HashChains 对象。

main 函数：

读取输入值 d（哈希表中的桶数）和 m（操作数的数量）。

根据输入执行一系列操作：

0：将键-值对插入哈希表。

1：在哈希表中搜索键。

2：从哈希表中删除键。

3. 测试结果（测试输入，测试输出）

题目一

输入

7 12

1 21

0 1

0 13

0 5

0 23

0 26

0 33

1 33

1 33

1 13

1 5

1 1

输出

-1

1

6

5

2

0

3

3

3

6

5

1

题目二

输入

7 12

1 21

0 1

0 13

0 5

0 23

0 26

0 33

1 33

1 33

1 13

1 5

1 1

输出

Not Found

3

3

1

3

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）

线性探查实现:最大困难时删除时移动元素的操作,需要通过分类讨论和判断,在满足三个条件的时候要删除的地方记为 a,移动的指针记为 b,指向的元素的初始桶记为 m,则

(if ((b!=m)&&(((m<=a)&&(b>a))||((m>b)&&((m<=a)||(b>a))))))实现桶中元素的互换,并记录移动次数.

链表实现:整体没有遇到棘手的难题,实现过程比较顺利.

总结:链表和线性探查相比,链表的用时相对更少,更快,线性探查更节省空间,更能体现散列表的特性.

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）

第一题

```
 #include<iostream>
#include<functional>
#include<string>
using namespace std;
#define MAXSIZE 1e5+10
int d,m;


template <class K>
class Hash {
public:
	size_t operator()(const int theKey) const {
		return size_t(theKey);
	}
};



template<class K,class E>
class hashTable
{
public:
	hashTable(int theDivisor);
	~hashTable(){delete[]table;}
	bool empty()const {return dSize==0;}
```

```
        int size()const {return dSize;}
        int searchIndex(const K&theKey)const;
        void findPair(const K&theKey)const;
        void insert(const pair<K,E>& thePair);
        void erase(const K&theKey);
private:
        Hash<K> hash;
        pair<K,E>** table;
        int dSize;
        int divisor;
};


template <class K, class E>
inline hashTable<K, E>::hashTable(int theDivisor)
{
        divisor=theDivisor;
        dSize=0;


        table =new pair<K,E>* [divisor];
        for(int i=0;i<divisor;i++)
                table[i]=NULL;
}
// 搜索对应的桶的编号
template <class K, class E>
inline int hashTable<K, E>::searchIndex(const K &theKey) const
{
        int i=(int)hash(theKey) %divisor;
        int j=i;
        do
        {
                if(table[j]==NULL||table[j]->first==theKey)
                        return j;
```

```
            j=(j+1)%divisor;
        } while (j!=i);
        //如果表已经满了,回到了原来的位置
        return j;
}

//找到对应的 pair
template <class K, class E>
inline void hashTable<K, E>::findPair(const K &theKey) const
{
        int b=searchIndex(theKey);
      //找不到
        if(table[b]==NULL||table[b]->first!=theKey)
                cout<<"-1"<<endl;
        else
                cout<<b<<endl;
}

template <class K, class E>
inline void hashTable<K, E>::insert(const pair<K, E> &thePair)
{
        int b=searchIndex(thePair.first);
        //有相关位置
        if(table[b]==NULL)
        {
                table[b]=new pair<K,E> (thePair);
                cout<<b<<endl;
                dSize++;
        }
        else
        {
                //检查是有相同的关键字,还是表满了
```

```
            if(table[b]->first==thePair.first)
            {
                cout <<"Existed"<<endl;
            }
            else
            {
                return ;
            }
        }
}

template <class K, class E>
inline void hashTable<K, E>::erase(const K &theKey)
{
    int b=searchIndex(theKey);
    int a,origin;
    a=origin=b;
    int ct=0;
    //没找到
    if(table[b]==NULL||table[b]->first!=theKey)
    {
        cout<<"Not Found"<<endl;
    }
    else
   {
        table[b] = NULL;
        b=(b+1)%divisor ;
        while(table[b] != NULL && b!= origin)
        {
            //应该在的位置
            int m=(table[b]->first)%divisor;
            //需要交换位置的三种情况
```

```
                if ((b!=m)&&(((m<=a)&&(b>a))||((m>b)&&((m<=a)||(b>a)))))
                {
                        table[a]=table[b];
                        table[b] = NULL;
                        a=b;
                        ct++;
                }
               b=(b+1) % divisor;
            }
              cout<<ct<<endl;
    }
}
int main()
{
        int opt,x;
        cin>>d>>m;
        hashTable<int,int>m_hash(d);
        while(m--)
        {
              cin>>opt>>x;
              switch (opt)
              {
              case 0:
              {
                    m_hash.insert(pair<int,int>(x,x));
                    break;
              }
              case 1:
              {
                    m_hash.findPair(x);
                    break;
              }
```

```
            case 2:
            {
                m_hash.erase(x);
            }
            default:
                break;
            }
        }
}
```

第二题

```
#pragma once
#pragma once
#pragma once
#include<iostream>
using namespace std;

template <class K>
class Hash {
public:
    size_t operator()(const int theKey) const {
        return size_t(theKey);
    }
};

template<class K ,class E>
class Dictionary
{
    public:
        virtual ~Dictionary(){}
        virtual bool empty()const=0;
        virtual int size()const=0;
        virtual pair<K,E>* find(const K&)const=0;
```

```
        virtual void erase(const K&)=0;
        virtual void insert(const pair<K,E>&)=0;
};
#pragma once
#include<iostream>
using namespace std;
template<class K,class E>
struct PairNode
{
    pair<K,E> element;
    PairNode<K,E>*next;
    PairNode(const pair<K,E>thePair,PairNode<K,E>*nextNode)
    {
        element=thePair;
        next=nextNode;
    }
    PairNode(){}
};
template<class K,class E>
class SortedChain:public Dictionary<K,E>
{
    public:
        bool empty()const
        {
            return chainSize==0;
        }
        int size()const
        {
            return chainSize;
        }
        pair<K,E>* find(const K& )const;
        void insert(const pair<K,E>&);
```

```
        void erase(const K&);
        void output(ostream& out)const;
    private:
        PairNode<K,E>* firstNode=NULL;
        int chainSize=0;
};

template <class K, class E>
inline pair<K,E>* SortedChain<K, E>::find(const K &theKey) const
{
    PairNode<K,E>* currentNode=firstNode;
    while(currentNode!=NULL&&currentNode->element.first<theKey)
    {
        currentNode=currentNode->next;
    }
    //匹配上了
    if(currentNode!=NULL&& currentNode->element.first==theKey)
    {
        cout<<chainSize<<endl;
    }
    else
    {
        cout<<"Not Found"<<endl;
    }
}

template <class K, class E>
inline void SortedChain<K, E>::insert(const pair<K, E> &thePair)
{
    int theKey=thePair.first;
    PairNode<K,E>*tp=NULL;
    PairNode<K,E>*p=firstNode;
```

```
        while(p!=NULL&&p->element.first<theKey)
        {
            tp=p;
            p=p->next;
        }

        //找到了相同关键字
        if(p!=NULL&&p->element.first==theKey)
        {
            cout<<"Existed"<<endl;
        }
        //没找到
        else
        {
            PairNode<K,E> *newNode=new PairNode<K,E>(thePair,p);
            if(tp==NULL)
                firstNode=newNode;
            else
            {
                tp->next=newNode;
            }
            chainSize++;
        }
        return ;
}

template <class K, class E>
inline void SortedChain<K, E>::erase(const K &theKey)
{
        PairNode<K,E>*tp=NULL;
        PairNode<K,E>*p=firstNode;
        while(p!=NULL&&p->element.first<theKey)
```

```
        {
            tp=p;
            p=p->next;
        }
        //找到了
        if(p!=NULL&&p->element.first==theKey)
        {
            //找到了第一个结点
            if(tp==NULL)
                firstNode=p->next;
            else
            {
                tp->next=p->next;
            }
            delete p;
            chainSize--;
            cout<<chainSize<<endl;
        }
        //没找到
        else
        {
            cout<<"Delete Failed"<<endl;
            return ;
        }
}

template<class K,class E>
void SortedChain<K,E>::output(ostream& out)const
{
        for(PairNode<K,E>* curruntNode=firstNode;curruntNode!=NULL;
        curruntNode=curruntNode->next)
        {
```

```
            auto k=currruntNode->element;
            out<<k.first<<" "<<k.second<<" ";
        }
}

template<class K,class E>
ostream & operator<<(ostream& out,const SortedChain<K,E>& x)
{
    x.output(out);return out;
}



template<class K,class E>
class HashChains
{
public:
    HashChains(int theDivisor)
    {
        divisor=theDivisor;
        dSize=0;
        table=new SortedChain<K,E>[divisor];
    }
    ~HashChains()
    {
        delete[] table;
    }
    bool empty()const
    {
        return dSize==0;
    }
    int size()const
    {
```

```
        return dSize;
    }
    pair<K,E>* findPair(const K&theKey)const
    {
        return table[hash(theKey)%divisor].find(theKey);
    }
    void insert(const pair<K,E>& thePair)
    {
        int homeBucket=(int)hash(thePair.first)%divisor;
        int homeSize=table[homeBucket].size();
        table[homeBucket].insert(thePair);
        if(table[homeBucket].size()>homeSize)
            dSize++;
    }
    void erase(const K& theKey)
    {
        table[hash(theKey)%divisor].erase(theKey);
    }
    void output(ostream& mout)const
    {
        for(int i=0;i<divisor;i++)
            if(table[i].size()==0)
                mout<<"NULL"<<endl;
            else
                mout<<table[i]<<endl;
    }

private:
    int divisor;
    SortedChain<K,E>* table;
    int dSize;
    Hash<K>hash;
```

```
};


template<class K,class E>
ostream& operator<<(ostream &mout,const HashChains<K,E>&x)
{
	x.output(mout);return mout;
}


int d,m;
int main()
{
	cin>>d>>m;
	HashChains<int,int>s(d);
	while(m--)
	{
		int opt,x;
		cin>>opt>>x;
		switch (opt)
		{
			case 0:
			{
				s.insert(pair<int,int>(x,x));
				break;
			}
			case 1:
			{
				s.findPair(x);
				break;
			}
			case 2:
			{
				s.erase(x);
```

```
                    break;
                }
                default:
                    break;
            }
        }
    }
```