**第三部分 第3题：哈希表设计**

**实验报告**

题目：设计一个可以迅速查找人名的哈希表。

班级：F1703407 姓名：赵伟基 学号：517021910883 完成日期：2018-12-22

1. **需求分析**
2. 假设人名为中国人姓名的汉语拼音形式。填入哈希表的人名共有 30 个。哈希函数用除留余数法构造，用线性探测法或开散列（链地址法）处理冲突。
3. 对于人名哈希表需实现：构表，查找，增加数据的功能。

注：本实验人名取自东19栋517宿舍成员和小说《倚天屠龙记》出场人物，name为其人名拼写，key值为人名中每个字符对应的ASCⅡ值+1的总和。预估在的区间范围内。

1. 挑选部分哈希函数并计算其冲突率和平均查找长度。对各种哈希函数进行比较。

本文采取的哈希函数是：

* 1. 除留余数法：
  2. 平方留中法：

1. 在哈希函数确定的前提下尝试各种不同处理冲突的方法，考察平均查找长度的变化。关于冲突处理，本实验对**线性探测法**和**二次探测法**做出尝试并比较其冲突率和平均查找长度。

以下给出冲突率和平均查找长度的定义：

记一次建表总冲突次数为 *chongtucnt* 总查找次数为*chazhaocnt* ，哈希表长为N；

地址冲突率 平均查找长度

1. **概要设计**

由于需要将人名数据转化为数字，设置结构体Data存储数据。同时定义哈希表类，分别定义线性探测法和二次探测法的建表、插入及查找函数。以及更新哈希表状态便于重新存值的状态清零函数，同时在类外建立对接的查找函数和建表函数。

1. 数据结点类

struct Data

数据对象：关键字 key; 人名name

// key值为人名中每个字符对应的ASCⅡ值+1的总和

1. 哈希表类（class namehash）
   1. 数据对象：

哈希表的结点类 node：

Data ：数据成员

state：保存data和当前表的状态state=1非空 state=0空

* 1. 成员函数

（1） namehash():

构造函数 ：对哈希表初始化，构造空哈希表；

（2） ~namehash():

析构函数 ：释放哈希表空间。

（3） createhash():

初始条件：存在已有的人名数据表

操作结果：将数据表依条按线性探测法插入哈希表中，同时统计平均查

找长度和地址冲突率，并打印所建的哈希表。

（4） createhash1():

初始条件： 存在已有的人名数据表

操作结果： 将数据表依条按二次探测法插入哈希表中，同时统计平均查找长度和地址冲突率，并打印所建的哈希表。

（5）insert ():

初始条件：输入单条Data类型数据

操作结果：根据除留余数法构建哈希函数，将数据按线性探测法插入哈

希表中，对插入次数和冲突次数进行统计。

1. ）insert1():

初始条件：输入单条Data类型数据

操作结果：根据除留余数法构建哈希函数，将数据按线性探测法插入哈希表中，对插入次数和冲突次数进行统计。

（7）insert2():

初始条件：输入单条Data类型数据

操作结果：根据平方留中法构建哈希函数，将数据按线性探测法插入 哈希表中，对插入次数和冲突次数进行统计。

1. find():

初始条件：输入需要查找的人名

操作结果：根据线性探测搜索返回该人名是否在表中，如是则输出查

找的长度。

1. find1():

初始条件：输入需要查找的人名

操作结果：根据二次探测搜索返回该人名是否在表中，如是则输出查

找的长度。

1. clear():

操作结果：将表中数据状态全部置零，便于下次操作。

1. 功能函数模块：
2. createnewlist():

初始条件：输入一个空的人名数据表；

返回结果：返回一个填满人名的数据表；

1. findname()

初始条件：输入一个哈希表；

返回结果：返回按线性探测的查找结果。

1. findname1()

初始条件：输入一个哈希表；

返回结果：返回按二次探测的查找结果。

1. 主函数模块

int main{

初始化哈希表；

构建数据表；

构建线性探测哈希表；

线性探测寻找模块；

数据清零；

构建二次探测哈希表；

二次探测寻找模块；

}

1. **详细设计**

（1）main.cpp

#include <iostream>

#include "hashname.h"

using namespace std;

int main()

{

namehash list1;

Data namelist[30];

createnewlist(namelist);

list1.createhash(namelist);

findname(list1);

list1.clear();

list1.createhash1(namelist);

findname1(list1);

return 0;

}

（2） hashname.h

#ifndef HEAD\_H\_INCLUDED

#define HEAD\_H\_INCLUDED

#include <iostream>

#include <cstring>

using namespace std;

struct Data

{

int key;

char\* name;

};

class namehash

{

private :

struct node

{

Data data;

int state=0; //1非空 0为空

};

node\* names;

int size;

public:

namehash(){size=47;names=new node[size];}

~namehash(){delete names;}

void createhash(Data namelist[]);//线性探测法构造

void createhash1(Data namelist[]);//二次探测法构造

void insert(const Data &x,int& chazhaocnt,int &chongtucnt);//线性探测法+除留余数法散列函数

void insert1(const Data &x,int& chazhaocnt,int &chongtucnt);//二次探测法+除留余数法散列函数

void insert2(const Data &x,int& chazhaocnt,int &chongtucnt);//线性探测法+平方取中法

void insert3(const Data &x,int& chazhaocnt,int &chongtucnt);//二次探测法+平方取中法

void clear();

bool find(const Data &x);

bool find1(const Data &x);

};

void createnewlist(Data namelist[]);

void findname(namehash &list1);

void findname1(namehash &list1);

#endif // HEAD\_H\_INCLUDED

(3) hashname.cpp

#include <cmath>

#include <iostream>

#include "hashname.h"

using namespace std;

void namehash::insert(const Data &x,int& chazhaocnt,int &chongtucnt)

{

int init,pos;

init=pos=x.key%size;

do{

if(names[pos].state!=1)

{

names[pos].data=x;

names[pos].state=1;

++chazhaocnt;

return;

}

pos=(pos+1)%size;

++chazhaocnt;

++chongtucnt;

}while(pos!=init);

}

void namehash::insert1(const Data &x,int& chazhaocnt,int &chongtucnt)

{

int init,pos,n=1;

init=pos=x.key%size;

do{

if(names[pos].state!=1)

{

names[pos].data=x;

names[pos].state=1;

++chazhaocnt;

return;

}

pos=init+n\*n;

while(pos>size-1)

{

pos-=size;

}

++chazhaocnt;

++chongtucnt;

++n;

}while(pos!=init);

}

void namehash::createhash(Data namelist[])

{

int chazhaocnt=0;

int chongtucnt=0;

for(int i=0;i<30;i++)

{

insert(namelist[i],chazhaocnt,chongtucnt);

}

cout<<"线性探测法平均查找长度为:"<<(double)chazhaocnt/30<<endl;

cout<<"地址冲突率为:"<<(double)chongtucnt/chazhaocnt<<endl;

cout<<"该方法所建哈希表为:"<<endl;

for(int i=0;i<size;i++)

if(names[i].state==1)

cout<<names[i].data.name<<" "<<names[i].data.key<<endl;

else continue;

}

void namehash::createhash1(Data namelist[])

{

int chazhaocnt=0;

int chongtucnt=0;

for(int i=0;i<30;i++)

{

insert1(namelist[i],chazhaocnt,chongtucnt);

}

cout<<"二次探测法平均查找长度为:"<<(double)chazhaocnt/size<<endl;

cout<<"地址冲突率为:"<<(double)chongtucnt/chazhaocnt<<endl;

cout<<"该方法所建哈希表为:"<<endl;

for(int i=0;i<size;i++)

if(names[i].state==1)

cout<<names[i].data.name<<" "<<names[i].data.key<<endl;

else continue;

}

void createnewlist(Data namelist[])

{

int len=0,j=0;

namelist[0].name="zhaoweiji";

namelist[1].name="bunailong";

namelist[2].name="yanglei";

namelist[3].name="zengjiaqi";

namelist[4].name="zhangwuji"; namelist[5].name="zhaomin"; namelist[6].name="zhouzhiruo";

namelist[7].name="xiaozhao"; namelist[8].name="yingli"; namelist[9].name="zhangcuishan";

namelist[10].name="xiexun"; namelist[11].name="yingtianzheng"; namelist[12].name="weiyixiao";

namelist[13].name="zhuangzhengrui"; namelist[14].name="pengying"; namelist[15].name="lengqian";

namelist[16].name="zhoudian"; namelist[17].name="yanhuan"; namelist[18].name="wencangsong";

namelist[19].name="tangyang"; namelist[20].name="xinran"; namelist[21].name="zhuyuanzhang";

namelist[22].name="xuda"; namelist[23].name="huayun"; namelist[24].name="changyuchun";

namelist[25].name="tanghe"; namelist[26].name="dengyu"; namelist[27].name="chenyouliang";

namelist[28].name="songyuanqiao"; namelist[29].name="yulianzhou";

for(int i=0;i<30;i++)

{

len=strlen(namelist[i].name);

namelist[i].key=0;

while(j<len)

{

namelist[i].key+=namelist[i].name[j]+1;//key值设定为

j++;

}

j=0;

}

}

void namehash::clear()

{

for(int i=0;i<size;i++)

{

names[i].state=0;

}

}

bool namehash::find(const Data &x)

{

int init,pos,cnt=0;

init=pos=x.key%size;

do{

if(names[pos].state==0) {cout<<"没有找到"<<endl;return NULL;}

if(names[pos].state==1&&names[pos].data.key==x.key)

{

cnt++;

cout<<"找到了,查找长度为:"<<cnt<<endl;

return 1;

}

pos=(pos+1)%size;

cnt++;

}while(pos!=init);

}

bool namehash::find1(const Data &x)

{

int init,pos,cnt=0;

int n=1;

init=pos=x.key%size;

do{

if(names[pos].state==0) {cout<<"没有找到"<<endl;return NULL;}

if(names[pos].state==1&&names[pos].data.key==x.key)

{

cnt++;

cout<<"找到了,查找长度为: "<<cnt<<endl;

return 1;

}

pos=init+n\*n;

while(pos>size-1)

{

pos-=size;

}

cnt++;

n++;

}while(pos!=init);

}

void findname(namehash &list1)

{

char ch;

char nam[20];

int len=0,j=0;

Data fin;

cout<<"--------------------------------------"<<endl;

cout<<"---------线性探测法查找模块-----------"<<endl;

cout<<"--------------------------------------"<<endl;

cout<<"--------------------------------------"<<endl;

ch=cin.get();

while(ch!='\n')

{

nam[j]=ch;

ch=cin.get();

j++;

}

if(ch=='\n')

nam[j]='\0';

fin.name=nam;

len=strlen(fin.name);

fin.key=0;

for(int i=0;i<len;i++)

fin.key+=fin.name[i]+1;

list1.find(fin);

}

void findname1(namehash &list1)

{

char ch;

char nam[20];

int len=0,j=0;

Data fin;

cout<<"--------------------------------------"<<endl;

cout<<"---------二次探测法查找模块-----------"<<endl;

cout<<"--------------------------------------"<<endl;

cout<<"--------------------------------------"<<endl;

ch=cin.get();

while(ch!='\n')

{

nam[j]=ch;

ch=cin.get();

j++;

}

if(ch=='\n')

nam[j]='\0';

fin.name=nam;

len=strlen(fin.name);

fin.key=0;

for(int i=0;i<len;i++)

fin.key+=fin.name[i]+1;

list1.find1(fin);

}

//平方留中法

void namehash::insert2(const Data &x,int& chazhaocnt,int &chongtucnt)

{

int init,pos;

init=pos=(((int)pow(x.key,2)/100))%100%size;

do{

if(names[pos].state!=1)

{

names[pos].data=x;

names[pos].state=1;

++chazhaocnt;

return;

}

pos=(pos+1)%size;

++chazhaocnt;

++chongtucnt;

}while(pos!=init);

}

void namehash::insert3(const Data &x,int& chazhaocnt,int &chongtucnt)

{

int init,pos,n=1;

init=pos=(((int)pow(x.key,2)/100))%100%size;

do{

if(names[pos].state!=1)

{

names[pos].data=x;

names[pos].state=1;

++chazhaocnt;

return;

}

pos=init+n\*n;

while(pos>size-1)

{

pos-=size;

}

++chazhaocnt;

++chongtucnt;

++n;

}while(pos!=init);

}

1. **调试分析**
2. 该程序将两种冲突方法构造的哈希表统一到一个类里面，虽保障了数据安全性，但仍有冗余，可以考虑构建两个哈希表继承类
3. 算法复杂度分析

（1）：时间复杂度

①O（n^2）: createnewlist()

②O(n): insert() ;insert1() ;insert2();createhash();createhash1()

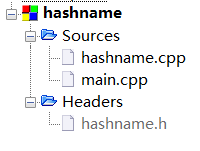
findname();findname1();createhash();createhash1()

（2）：空间复杂度

O(n^2)

1. **用户手册**

1．本程序使用的 Code::Blocks 16.01 IDE，程序以项目（project）方式组织，如图 1 所示



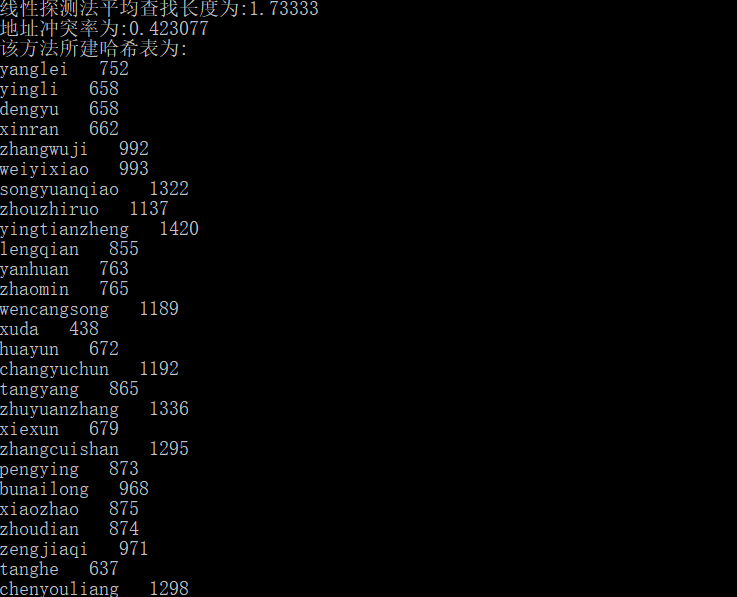
2． 用户点击main.cpp->build and run->依照交互提示输入数据。

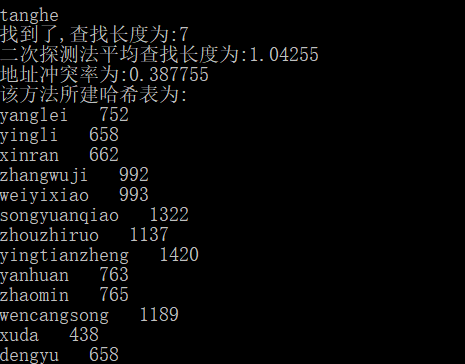
经多次实验推荐在寻找模块中输入人名“**tanghe**”，便于体现算法的可靠性；

如需更换人名在createnewlist函数中更换即可。

程序执行后，算法自动默认为线性探测法+除留余数法及二次探测法+除留余数法，包含查找函数。对于平方留中法对应的哈希表未设计查找函数，如需要平方留中法的数据请至createhash函数及createhash1函数中更改insert为insert1，更改insert2为insert3，更改后查找函数不可用。

1. **测试结果**





|  |  |  |
| --- | --- | --- |
| 方案 | 平均查找长度 | 地址冲突率 |
| 线性探测法+除留余数法 | 1.7333 | 0.4231 |
| 二次探测法+除留余数法 | 1.0426 | 0.3878 |
| 线性探测法+平方留中法 | 1.9000 | 0.4737 |
| 二次探测法+平方留中法 | 1.0213 | 0.375 |

结果分析：由于哈希表的元素数值较少，故各方案地址冲突和平均查找函数的差别并不大。key值与哈希表长相对接近，故选择除留余数和平方留中法来构造哈希函数，从冲突率可以看出效果良好。多次测验并没有得到一个能使冲突降为0的哈希函数。最终可以得出结论，对于该key之下二次探测和平方留中法的方案最适合该数据。

1. **附录**

main.cpp //主函数

hashname.h //头文件

hashname.cpp//功能函数实现模块