Project7 实验报告

22336044 陈圳煌

1. 程序功能简要说明

程序可以根据读入的关键字（string 类型），通过特定的哈希函数确定在存储的线性表中的位置，如果发生冲突，可以使用两种方法（二重哈希函数再散列和随机数序列法）进行处理，存储到哈希表中，并可以根据输入的关键字读取在表中的位置（如有），最后计算平均查找长度。

2. 部分代码以及程序说明

首先定义哈希表的存储类型：

```
16  struct data{
17      bool valid;
18      string name;
19      data():valid(1),name(""){}
20  };
21  struct Haxi{
22      int size;
23      data *peoplename;
24      Haxi():size(0),peoplename(NULL){}
25  };
```

Haxi 结构体中存储当前哈希表已存入的数据量 size 和 peoplename 哈希线性表。data 类型中存储关键字 name 和当前位置是否被访问的标志 valid。

哈希函数：除留余数法

读入当前输入的关键字（一个人的名字，例如张三 ZhangSan，名字的每个字首字母大写，然后算名字每个字首字母的 ASCII 码的和取模得到初始位置，倘若发生冲突再进行再散列或者随机数法处理）

```
//哈希函数：除留余数法
int getHaxi(string str,int N){
    //取姓名中的每个字的首字母在字母表上的位置相加
    int len=str.length();
    int num=0;
    for(int i=0;i<len;i++){
        if(str[i]>='A'&&str[i]<='Z'){
            num+=str[i]-'0';
        }
    }
    return num%N;
}
```

第一种处理冲突方法：再散列，使用双重哈希函数

当第一次哈希函数求初始位置发生冲突时，进行二次哈希函数，取名字每个字母的 ASCII 码相加的和取模作为新的位置，倘若再次冲突，则使用线性探测，往后加一寻址。

```
}
//再散列，第二个哈希函数
int getHaxi2(string str,int N){
    //取每个字母在字母表上的位置相加
    int num=0;
    int len=str.length();
    for(int i=0;i<len;i++){
        num+=str[i]-'0';
    }
    return num%N;
}
```

建立哈希表：

```
//方法1：
//建立哈希表
void setHaxi(Haxi &s,string str,int N){
    if(s.size==N){
        cout << "Store Fail!" << endl;
        return;
    }
    int index=getHaxi(str,N);
```

```
62      int count=0;
63      while(s.peoplename[index].valid==false){
64          int d=0;
65          count++;
66          if(count==1){
67              d=getHaxi2(str,N);
68          }else{
69              d=1;
70          }
71          index=(index+d)%N;
72      }
73      s.peoplename[index].name=str;
74      s.peoplename[index].valid=false;
75      s.size++;
76  }
```

查找：跟关键字存入哈希表的步骤相同。

```
77  //查找
78  void findindex(Haxi s,string op,int N){
79      int count=0;
80      int index=getHaxi(op,N);
81      while(s.peoplename[index].valid==false){
82          count++;
83          if(s.peoplename[index].name==op){
84              cout << op << "的位置在: " << index << endl;
85              cout << "查找次数为: " << count << endl;
86              return;
87          }
88          if(count==N){
89              cout << "未找到" << endl;
90              return;
91          }
92          if(count==1){
93              index=(index+getHaxi2(op,N))%N;
94          }else{
95              index=(index+1)%N;
96          }
97      }
98      cout << "未找到" << endl;
99  }
```

第二种处理方法：随机数处理法

```
11  // 创建一个随机数引擎
12  random_device rd;
13  mt19937 gen(rd());
14  // 创建一个整数分布，范围是 [1, 100]
15  uniform_int_distribution<int> intDist(1,100);
```

初始位置仍由哈希函数生成，发生冲突时，生成一个随机数

```
26  //获得随机数
27  int getrandom(){
28      int randomnum=intDist(gen);
29      //cout << randomnum << endl;
30      return randomnum;
31  }
```

建立哈希表过程除了迭代步骤以外其他与第一种方法相同

```
129     int index=getHaxi(str,N);
130     while(s.peoplename[index].valid==false){
131         int d=getrandom();
132         index=(index+d)%N;
133     }
```

查找过程使用线性探测法。

最后可以计算查找方法的平均查找长度：（仅以第一种方法为例）

```
99  }
100 //计算平均值
101 void calculate(Haxi s,string op,int N,vector<int> &record){
102     int count=0;
103     int index=getHaxi(op,N);
104     while(s.peoplename[index].valid==false){
105         count++;
106         if(s.peoplename[index].name==op){
107             record.push_back(count);
108             return;
109         }
110         if(count==30){
111             record.push_back(0);
112             return;
113         }
114         if(count==1){
115             index=(index+getHaxi2(op,N))%N;
116         }else{
117             index=(index+1)%N;
118         }
119     }
120     record.push_back(0);
121 }
```

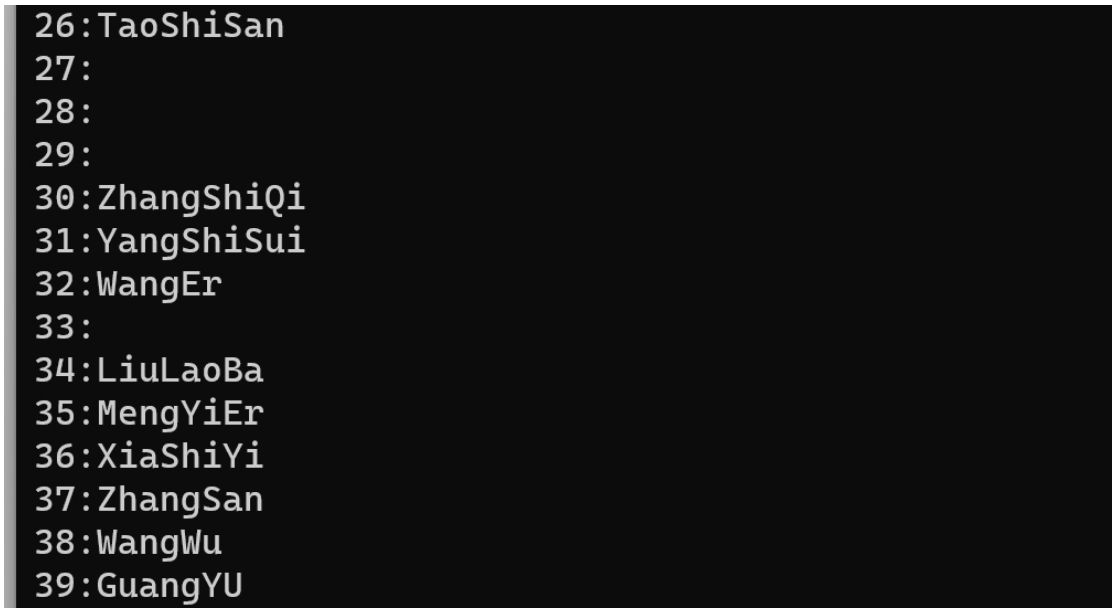
3. 测试

测试数据如图:

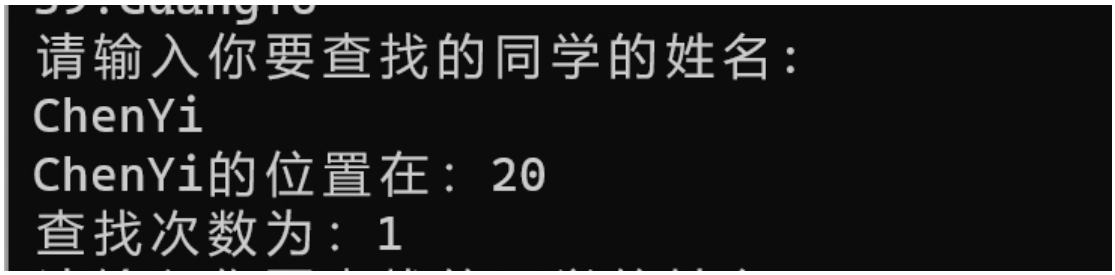
```
string name[30]={"ChenYi","WangEr","ZhangSan","LiSi","WangWu","ZhaoLiuLiu",
                 "LuXiaoQi","LiuLaoBa","QinJiu","YangShiSui","XiaShiYi","HuangShiEr",
                 "TaoShiSan","LinShiSi","ChenShiWu","LiShiLu","ZhangShiQi","WangShiBa",
                 "YangDaTou","ChenShuaiShuai","MaoDaYi","MengYiEr","WuZhongYou","GanNinNiang",
                 "LiangShangPo","LuJunYi","SongJiang","GuangYU","ZhangYiDe","LiuXuanDe"};
```

为 30 个人名。

```
请选择使用的Haxi表处理冲突
1.双函数
2.随机数
1
0:HuangShiEr
1:
2:WuZhongYou
3:GanNinNiang
4:
5:LinShiSi
6:SongJiang
7:LuJunYi
8:LiuXuanDe
9:ChenShuaiShuai
10:MaoDaYi
11:LiShiLu
12:WangShiBa
13:ChenShiWu
14:
15:LiangShangPo
16:
17:YangDaTou
18:ZhaoLiuLiu
19:QinJiu
20:ChenYi
21:LuXiaoQi
22:ZhangYiDe
23:LiSi
24:
25:
```



上图为哈希表。

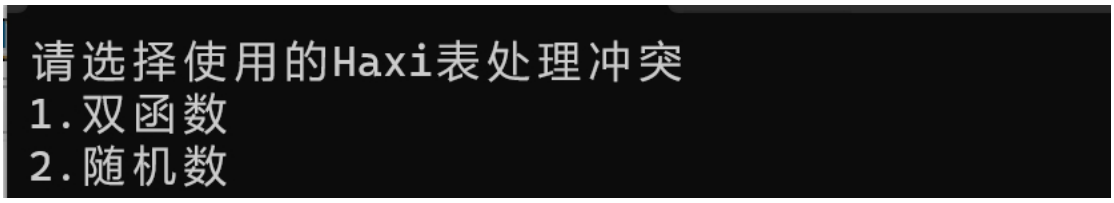


输入查找的关键字，返回出现在哈希表中的位置以及查找次数。

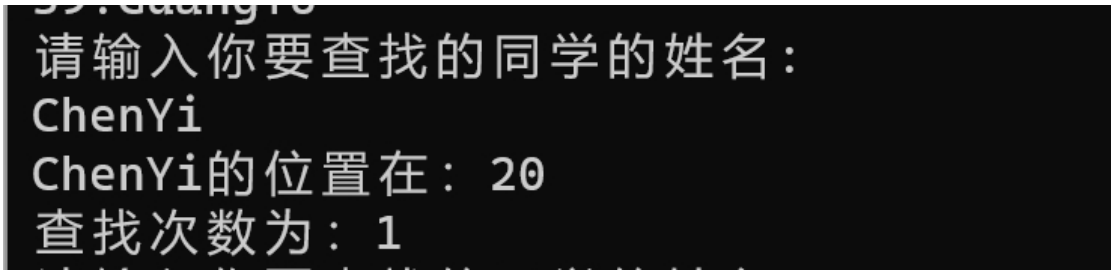
输入 stop 停止查找，并计算出查找长度，第一种方法二重哈希函数再散列的平均查找长度为 1.46667,小于 2，符合要求。(随机数法类似,不作展示,平均查找长度为 9.1)

4. 程序运行方式

打开可执行文件,选择方法处理哈希表冲突问题:



接着可以输入要查找的关键字,即可返回所在位置以及查找次数.



最后输入 stop 结束查找,计算平均查找长度.