湖南大學

数据结构

课程实验报告

题　　目：自组织线性表

学生姓名　吴多智

学生学号　201626010520

专业班级　软件 1605

完成日期　2017/12/29

一、 需求分析

1.问题描述

问题描述

用转置法实现一个自组织线性表，保存一组汉字用于查询。

基本要求

从文件中读入一组汉字集合，用自组织线性表保存。

在查询时，采用转置法调整自组织线性表的内容。

从文件中依次读入需查询的汉字，把查询结果保存在文件中（如找到，返回比较的次数，如果没有找到，返回比较的次数）

功能：

1.实现对文件的读写

2.用线性表存储一组汉字

3.实现两个数据元素的位置交换

4.遍历线性表

2.输入数据

（1）输入数据

在 character.txt 输入一组汉字，每个汉字之间以空格隔开。

在 search 输入一组需要查询的汉字，每个汉字之间以空格隔开。

（2）输出范围：

在 character.txt 汉字的个数不超过 50，且没有相同的汉字

3.输出数据

在 result.txt 中，如果需要查询的汉字有 n 个，则写入 n 行。

每行写入该汉字、查找结果和比较次数。

4.测试样例设计

1.

character.txt

你 好 我 是 三 体 人 地 球 已 经 被 占 领 了 虫 子 受 死 吧

search.txt

三 吧 这

result.txt

三 查找成功 比较次数：5

吧 查找成功 比较次数：20

这 查找失败 比较次数：20

2.

character.txt

唯 一 不 可 阻 挡 的 是 时 间

search.txt

唯 一 一

result.txt

唯 查找成功 比较次数：1

一 查找成功 比较次数：2

一 查找成功 比较次数：1

3.

character.txt

你 站 在 这 里 不 要 走 动 我 去 买 几 个 橘 子

search.txt

这 这 这 这

result.txt

这 查找成功 比较次数：4
这 查找成功 比较次数：3
这 查找成功 比较次数：2
这 查找成功 比较次数：1

4.

character.txt

从 文 件 中 读 入 一 组 汉 字 集 合

searchr.txt

你

result.txt

你 查找失败 比较次数：12

5.

character.txt

分 析 问 题 的 数 据 和 结 构 特 征

search.txt

和 据

result.txt

和 查找成功 比较次数：8
据 查找成功 比较次数：8

二、概要设计

1.抽象数据类型

（1）数据：character.txt 中一组汉字集合

（2）结构特征：线性结构

（3）ADT{

数据对象：一组汉字集合

数据关系：线性关系

基本操作：

```
void init() //初始化一个线性表
void append(elemtype a)//在线性表表尾添加数据 a
void moveToStart()//设置当前位置为表头
void next()//使当前位置向右移动一位
elemtype getvalue()//返回当前位置的值
bool search(elemtype a,int* count)
```

}

2.算法的基本思想

新建一个带有头结点的线性表，从 character.txt 中逐次读取一个汉字，然后将每个

汉字添加到线性表的表尾，直到文件内容结束。再从 search.txt 文件中逐次读取需要查询的汉字，每次读取，都将遍历线性表，查找是否有该汉字，查找的同时计算比较的次数，如果找到该汉字且汉字不是第一个，就将其与前一个汉字交换位置。然后在 result.txt 写入查询结果和比较次数。

3.程序的流程

（1）初始化模块：从文件中读取汉字，并将汉字依次存入线性表

（2）查询模块：从文件中读取需要查询的汉字，然后遍历线性表进行查找，并计算比较次数。如果找到该汉字，就将其与前一个汉字交换位置。返回查找结果

（3）写入模块：将查询结果写入文件中

三、详细设计

1.物理数据类型

（1）物理数据类型：因为每个汉字占两个字符，所以数据类型是 char*

（2）物理数据结构：因为经常将两个汉字进行交换，所以基于单向链表实现线性表

（3）ADT:

```
    class node{
public:
    char* element;//数据域
    node *next;//指向下一个结点
    node(char* elem){//构造函数
        element=new char[2];
        strcpy(element,elem);
        next=NULL;
    }
    node(){//构造函数
        next=NULL;
    }
};
class list{
    private:
        node* head;
        node* tail;
        node* curr;
        int cnt;
        void init(){
            curr=tail=head=new node();
            cnt=0;
        }
    public:
        list(){
            init();
        }
        char* getvalue(){
            return curr->element;
        }
```

```
void moveToStart(){
    curr=head;
}
void next(){
    if(curr->next){
        curr=curr->next;
    }
}
void append(char* a){
    node* temp=new node(a);
    tail=curr->next=temp;
    curr=tail;
    cnt++;

}
bool search(char* c,int*count){
    moveToStart();
    while(curr->next!=NULL){
        next();
        *count+=1;
        if(strcmp(getvalue(),c)==0){
            if(strcmp(c,head->next->element)!=0){
                moveToStart();
                node*temp;
                while(strcmp(c,curr->next->next->element)!=0){
                    next();
                }
                temp=curr->next->next;
                curr->next->next=temp->next;
                temp->next=curr->next;
                curr->next=temp;
            }
            return true;
        }
    }
    return false;
}
```

2.输入和输出的格式

（1）输入数据

在 character.txt 输入一组汉字，每个汉字之间以空格隔开。

在 search 输入一组需要查询的汉字，每个汉字之间以空格隔开。

（2）输出范围：

在 character.txt 汉字的个数不超过 50，且没有相同的汉字

（3）输出数据：

在 result.txt 中，如果需要查询的汉字有 n 个，则写入 n 行。

每行写入该汉字、查找结果和比较次数。

3.算法的具体步骤

针对每一个模块，设计并阐述算法的具体步骤，要用文字的形式描述步骤，也可以用流程图的形式描述步骤，要给出伪代码。如果一个模块算法复杂，可以采用分为更小功能模块的方式来分别阐述。

（1）初始化模块：新建一个链表,打开文件 character.txt 读取汉字。每读取一个汉字，就新建一个结点存储该汉字，然后链接到表尾。

```
node(char* elem){//构造函数
    element=new char[2];
    strcpy(element,elem);
    next=NULL;
}
void init(){
        curr=tail=head=new node();
        cnt=0;
}
list l=list();
ifstream in("character.txt");
char c[2];
if(in.is_open()){
    while(!in.eof()&&in>>c){
        l.append(c);
    }
}
else{
    cout<<"未成功打开 character.txt"<<endl;
}
```

（2）查询模块和写入模块：打开 search.txt 和 result.txt.逐次从 search.txt 中读取要查找的汉字，然后遍历链表，查找该汉字。然后将查询结果写入 result.txt.

```
ifstream find("search.txt");
    ofstream result("result.txt");
    if(find.is_open()){
        while(!find.eof()&&find>>c){
            int a=0;
            int* count=&a;
            bool flag;
            flag=l.search(c,count);
            if(flag==true){
                if(result.is_open()){
                    result<<c<<" 查找成功 比较次数："<<*count<<"\n";
                }
                else{
                    cout<<"未成功打开 result.txt"<<endl;
```

```
                }
            }
            else{
                if(result.is_open()){
                    result<<c<<"  查找失败  比较次数：  "<<*count<<"\n";
                }
                else{
                    cout<<"未成功打开 result.txt"<<endl;
                }
            }
        }
    }
    else{
            cout<<"未成功打开 search.txt"<<endl;
    }
```

4.算法的时空分析

（1）初始化模块：将 n 个汉字存储到链表， $\Theta(n)$.

（2）查询模块：遍历链表 $\Theta(n)$

（3）写入模块：在文件中写入查询结果， $\Theta(1)$

四、调试分析

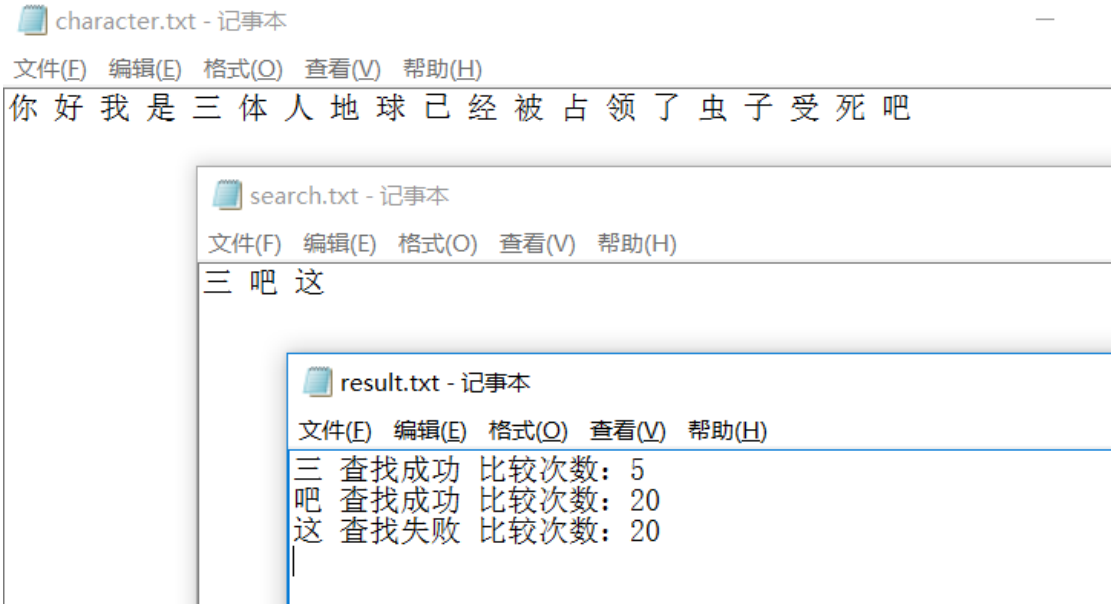
1.调试方案设计

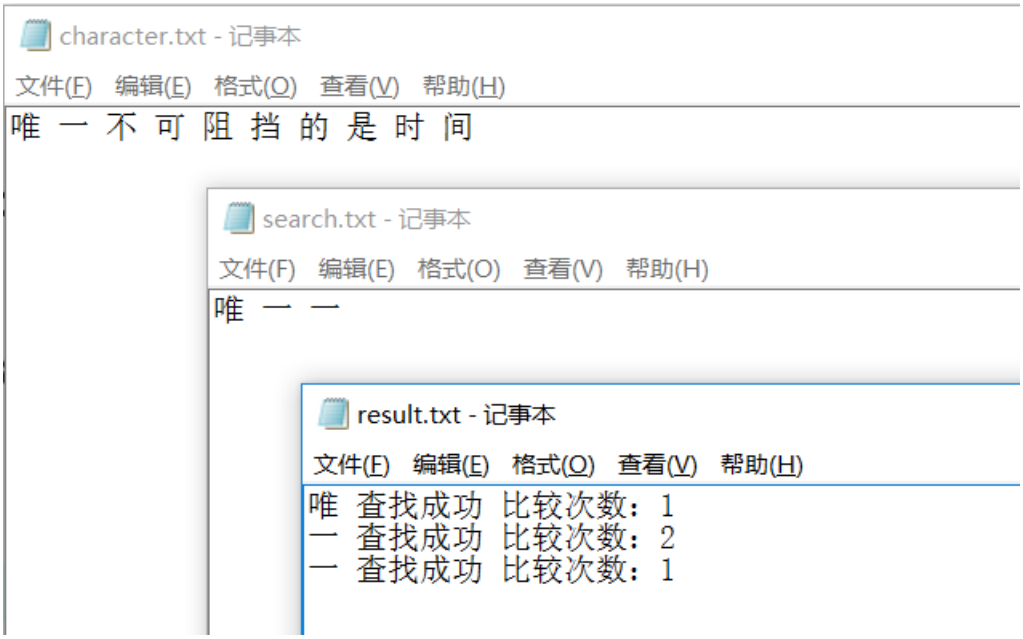
（1）语法错误：从列出的第一个错误着手，修改这个错误，然后重新编译程序。有时，在修改了一个错误之后，很多错误会随之消失。

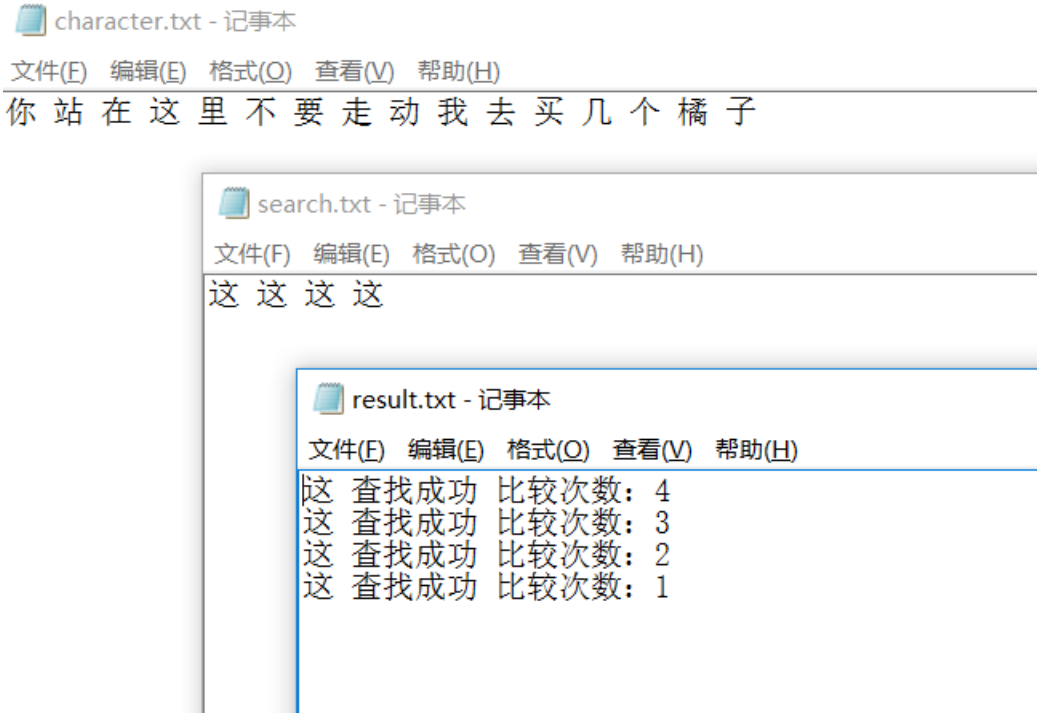
（2）运行错误：输入多组已设计好的测试样例，如果程序在运行中出现异常终止运行，一般有可能是出现了野指针。可设置断点，进行单步调试，找出错误。

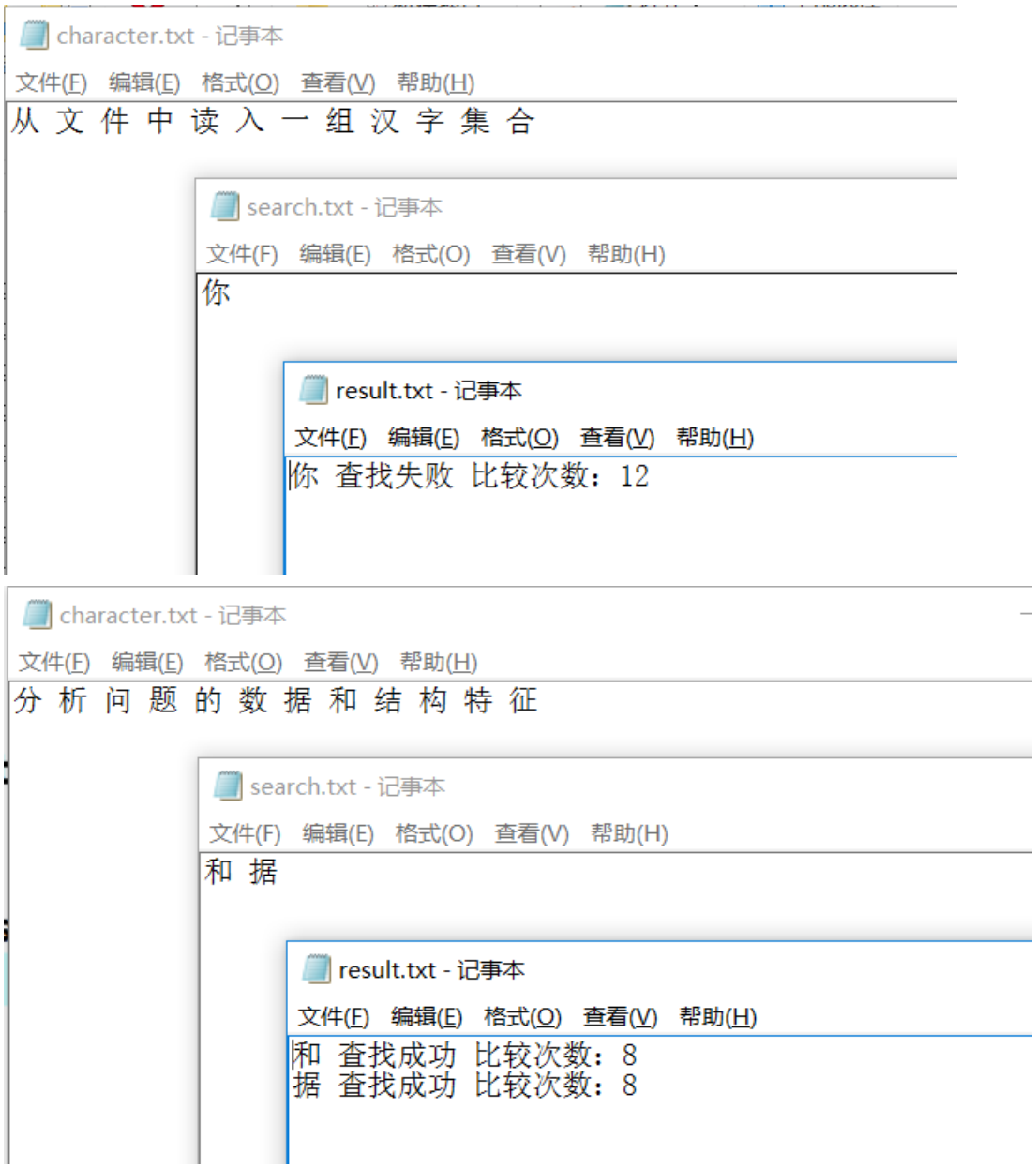
五、测试结果

用截屏的方式，贴出每个测试样例的运行结果。




character.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
唯 一 不 可 阻 挡 的 是 时 间
search.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
唯 一 一
result.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
唯 查找成功 比较次数：1
一 查找成功 比较次数：2
一 查找成功 比较次数：1


character.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
你 站 在 这 里 不 要 走 动 我 去 买 几 个 橘 子
search.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
这 这 这 这
result.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
这 查找成功 比较次数：4
这 查找成功 比较次数：3
这 查找成功 比较次数：2
这 查找成功 比较次数：1



六、实验心得

自组织查找表，本质上来说可以算是线性表的高级玩法，因为其在查找成功时会改变表中元素的位置，将其前置，这是因为我们觉得一个元素被查找后，在次被查找的概率就会比其他元素大。在自组织查找表中，前置的策略也要根据情况的不同而不同，我们可以根据查找次数来排列，或者直接置首，不同的策略有不同的优缺点，我们要根据应用场景的不同选择合适的策略。通过实验，也知道了线性表还能有这样的玩法，只要我们多想，还是能发现很多不一样的玩法。