湖南大學

数据结构

课程实验报告

题　　目：查 找 应 用

学生姓名　李晓彤

学生学号　201726010128

专业班级　软件 1701 班

完成日 期　2018.12.29

一、需求分析

0 问题描述

自组织线性表根据估算的访问频率排列记录，先放置请求频率最高的记录，接下来是请求频率次高的记录，依此类推。自组织线性表根据实际的记录访问模式在线性表中修改记录顺序。自组织线性表使用启发式规则决定如何重新排列线性表。转置方法的基本原理是，在一次查找过程中，一旦找到一个记录，则将它与前一个位置的记录交换位置。这样，随着时间的推移，经常访问的记录将移动到线性表的前端，而曾经频繁使用但以后不再访问的记录将逐渐退至线性表的后面。

尽管一般情况下自组织线性表的效率可能没有查找数和已排序的线性表那么好，但它也有自身的优势。它可以不必对线性表进行排序，新记录的插入代价很小；同时也比查找树更容易实现，且无需额外的存储空间。

基本要求：

1.从文件中读入一组汉字集合，用自组织线性表保存。自组织线性表在查询时，采用转置法调整自组织线性表的内容。

2.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）

1.问题分析

问题中的需求：

1.使用线性表的ADT实现。

2.从文件中读入一组汉字集合，用自组织线性表保存。自组织线性表在查询时，采用转置法调整自组织线性表的内容。

3.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.从文件中读入一组汉字集合构造自组织线性表。

2.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。

3.并将每次查找后的线性表输出至文件中。

2.输入数据

需要用户输入要打开的文件名称，这里只有五个文件，例如打开text1则输入1，查找结果在result1中显示。

数据输入格式为：输入的数字为1-5的整数。

例如输入：

1

则表示你打开text1，查找结果在result1中显示。

3.输出数据

1.先输出查找结果。

2.接下来输出查找这个字的比较次数。

3.最后输出查找这个字后线性表的状态。

例如输出：

查找成功！查找(我)一共比较了1次

当前顺序表存储的汉字为：

我爱你祖国

查找成功！查找(爱)一共比较了2次

当前顺序表存储的汉字为：

爱我你祖国

查找失败！查找(过)一共比较了5次
当前顺序表存储的汉字为：
爱我你祖国

4.测试样例设计

测试样例一：（普通）
输入：
1
错误输出：
查找成功！查找(我)一共比较了1次
当前顺序表存储的汉字为：
我爱你祖国
查找成功！查找(爱)一共比较了2次
当前顺序表存储的汉字为：
爱我你祖国
查找失败！查找(过)一共比较了5次
当前顺序表存储的汉字为：
爱我你祖国
查找成功！查找(我)一共比较了2次
当前顺序表存储的汉字为：
我爱你祖国
查找成功！查找(爱)一共比较了2次
当前顺序表存储的汉字为：
爱我你祖国
查找失败！查找(过)一共比较了5次
当前顺序表存储的汉字为：
爱我你祖国

正确输出：
---欢迎来到自组织线性表的文件内容查找---
请输入要打开的文件（这里只有5个）：例如打开text1则输入1，结果在result1中显示
1
查找成功！查找(我)一共比较了1次
当前顺序表存储的汉字为：
我爱你祖国
查找成功！查找(爱)一共比较了2次
当前顺序表存储的汉字为：
爱我你祖国
查找失败！查找(过)一共比较了5次
当前顺序表存储的汉字为：
爱我你祖国
样例说明：该样例是最普通的样例，线性表中有5个汉字，要查找的汉字个数小于线性表汉字个数。这里查找“我”，因为我是线性表第一个所以不转置，查找“爱”是线性表第二个所以将它和前面的那个转置。因为查找“过”字不存在，所以比较次数为5次，查找失败！

测试样例二：（特殊样例，查找字数大于线性表中汉字个数）
输入：
2

输出：
---欢迎来到自组织线性表的文件内容查找---
请输入要打开的文件（这里只有5个）：例如打开text1则输入1，结果在result1中显示
2
查找失败！查找(七)一共比较了8次
当前顺序表存储的汉字为：
数据结构课程实验
查找失败！查找(段)一共比较了8次
当前顺序表存储的汉字为：
数据结构课程实验
查找成功！查找(数)一共比较了1次
当前顺序表存储的汉字为：
数据结构课程实验
查找失败！查找(码)一共比较了8次
当前顺序表存储的汉字为：
数据结构课程实验
查找失败！查找(管)一共比较了8次
当前顺序表存储的汉字为：
数据结构课程实验
查找失败！查找(啦)一共比较了8次
当前顺序表存储的汉字为：
数据结构课程实验
查找失败！查找(啦)一共比较了8次
当前顺序表存储的汉字为：
数据结构课程实验
查找失败！查找(啦)一共比较了8次
当前顺序表存储的汉字为：
数据结构课程实验
查找成功！查找(数)一共比较了1次
当前顺序表存储的汉字为：
数据结构课程实验
查找成功！查找(据)一共比较了2次
当前顺序表存储的汉字为：
据数结构课程实验
样例说明：该样例是查找汉字个数大于线性表中存储的个数。查找的汉字有10个，线性表中的汉字有8个。

测试样例三：（普通）
输入：
3

输出：
---欢迎来到自组织线性表的文件内容查找---
请输入要打开的文件（这里只有5个）：例如打开text1则输入1，结果在result1中显示
3
查找成功！查找(数)一共比较了3次
当前顺序表存储的汉字为：
我数爱据结构一一
查找成功！查找(据)一共比较了4次
当前顺序表存储的汉字为：
我数据爱结构一一
查找成功！查找(结)一共比较了5次
当前顺序表存储的汉字为：
我数据结爱构一一
查找成功！查找(构)一共比较了6次
当前顺序表存储的汉字为：
我数据结构爱一一
查找失败！查找(不)一共比较了8次
当前顺序表存储的汉字为：
我数据结构爱一一
查找成功！查找(爱)一共比较了6次
当前顺序表存储的汉字为：
我数据结爱构一一
查找成功！查找(我)一共比较了1次
当前顺序表存储的汉字为：
我数据结爱构一一
样例说明：该样例是普通样例，和样例一一样，这里不做过多赘述。

测试样例四：（特殊，线性表都是同样的内容，查找的文字也是并且都查找失败）
输入：
3

输出：
---欢迎来到自组织线性表的文件内容查找---
请输入要打开的文件（这里只有5个）：例如打开text1则输入1，结果在result1中显示
4
查找失败！查找(二)一共比较了7次
当前顺序表存储的汉字为：
一一一一一一一
查找失败！查找(二)一共比较了7次
当前顺序表存储的汉字为：
一一一一一一一
查找失败！查找(二)一共比较了7次
当前顺序表存储的汉字为：
一一一一一一一
查找失败！查找(二)一共比较了7次

当前顺序表存储的汉字为：
一一一一一一一一
查找失败！查找(二)一共比较了7次
当前顺序表存储的汉字为：
一一一一一一一一
样例说明：这是一个特殊样例。该样例是线性表中汉字都为一，查找的汉字都是二，并且在线性表中查不出来。

测试样例五（普通）
输入：
5

输出：
---欢迎来到自组织线性表的文件内容查找---
请输入要打开的文件（这里只有5个）：例如打开text1则输入1，结果在result1中显示
5
查找成功！查找(我)一共比较了6次
当前顺序表存储的汉字为：
期末了希我望能顺
查找失败！查找(觉)一共比较了8次
当前顺序表存储的汉字为：
期末了希我望能顺
查找失败！查找(得)一共比较了8次
当前顺序表存储的汉字为：
期末了希我望能顺
查找失败！查找(海)一共比较了8次
当前顺序表存储的汉字为：
期末了希我望能顺
查找失败！查找(信)一共比较了8次
当前顺序表存储的汉字为：
期末了希我望能顺

样例说明：这是一个简单普通样例。查找的汉字有5个，线性表中存储的汉字有8个。

二、概要设计

1.抽象数据类型

因为题目要求用自组织线性表的ADT实现，并且存储的数据满足：通过 for 循环依次存入汉字，按照存储的先后次序，满足线性特征，即<ai,ai+1>(0≤a＜n)。

所以这里采用了数组实现。

2.算法的基本思想

本题利用char数组来存储汉字。一个汉字是两字节，所以一个汉字由数组中的两个char空间来存储。我设计的是在text文件中放入要构造的线性表和要查找的汉字，第一行是线性表，第二行是查找的汉字。所以用string来读取每一行的数据，然后再进行相应的操作。

关于查找：因为题目要求在查找时采用转置法调整自组织线性表的内容，所以我将转置直接放在了查找函数中，每进行一次查找，就进行转置。如果查找的汉字是线性表的第一个则不转置，否则则用转置法来调整自组织线性表的内容。

3.程序的流程

程序由两个模块组成：

（1） 构造自组织线性表：

这一模块的功能是：将从text文件中读入的第一行汉字存入数组。这一步完成了从文件中读入一组汉字集合，用自组织线性表保存的要求。

（2） 查找+转置模块：

这一模块的功能是：从text文件中读入第二行汉字，然后在数组中查找进行查找，如果查找到了则输出查找成功的提示信息，并输出比较次数，如果查找失败了则输出查找失败的提示信息，并输出比较次数。这里对于每一次查找后都输出转置的线性表。这里同时向屏幕和文件中都输出查找成功与否的信息和转置后的线性表。这一模块完成了“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）”这一要求，并且从向文件result输出的转置结果可以看出，这一模块也实现了“自组织线性表在查询时，采用转置法调整自组织线性表的内容”这一要求。

三、详细设计

1.物理数据类型

数据是一串汉字，根据这串汉字来构建自组织线性表，因为从文件中读取的是汉字，一个汉字是两个字节，所以两个数组空间存储一个汉字。之后因为要求用转置法调整自组织线性表，所以采用数组我认为比采用链表更方便。

抽象数据类型的具体设计：

```
ADT   IntegerSet {
      数据对象：D = { ai | ai Î 汉字，i = 1,2，…, n, n>=0}
      数据关系：R = {< ai-1，ai> | ai-1, ai ÎD}
      成员：
       int maxSize;//顺序表容量
       int listSize;//目前的大小
       int curr;//当前元素的位置
       char* listArray;//列表元素将存放在该元素中
      成员函数：
       SOList(int size=defaultSize){
       maxSize = size;listSize = curr = 0;listArray = new char[maxSize];}
       ~SOList() ;//析构函数
       void print();//打印线性表
       void append(char,char);//末尾插入线性表值
       int find(char a,char b);//查找+转换
       int length() const;//返回顺序表长度
       char getValue(int) const;//返回值是当前元素
```

2.输入和输出的格式

输入流程及格式：输入的数字为要打开的文件名，例如打开text1则输入1，结果在result1中显示。

输出流程：

先输出系统名称：---欢迎来到自组织线性表的文件内容查找---

然后输出提示信息和输入例子： 请输入要打开的文件（这里只有5个）：例如打开text1则输入1，结果在result1中显示

当用户输入要打开的文件名后，输出查找和转置的结果。
如果查找成功则输出：查找成功！然后输出查找该字比较了几次：查找(我)一共比较了6次
然后输出当前顺序表存储的汉字为：
期末了希我望能顺
如果查找失败则输出：查找失败！然后输出查找该字比较了几次：查找(觉)一共比较了8次
然后输出当前顺序表存储的汉字为：
期末了希我望能顺

2.算法的具体步骤

1.先输入要打开的文件名称，然后根据用户输入的文件名来读取相应文件中的内容。

2.根据文件中的第一行汉字来构造自组织线性表。

3.根据文件中的第一行汉字进行查找操作。如果查找成功则输出查找成功。这里是自己改造后的数组本身的成员函数来完成。

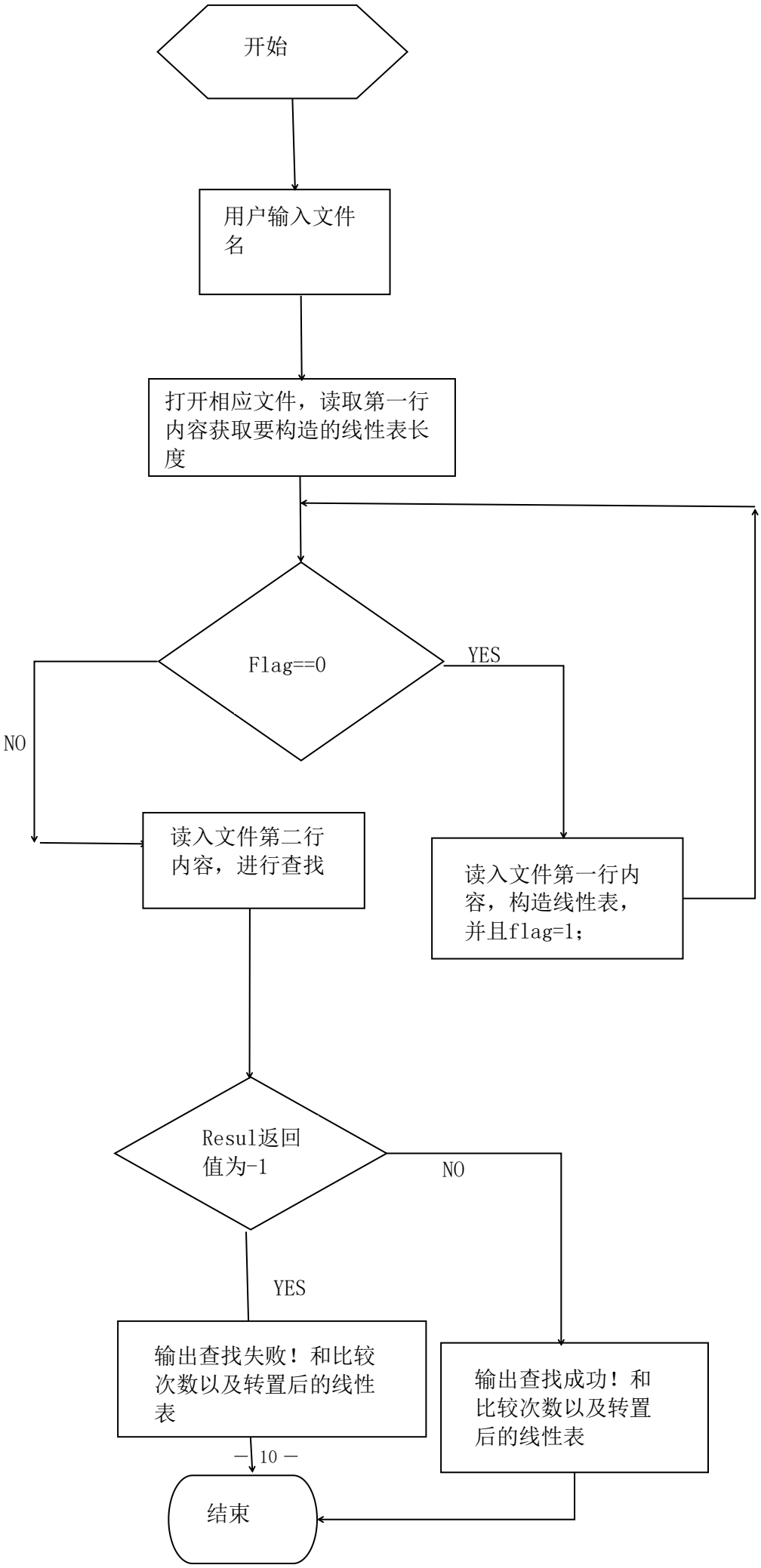
4.输出查找需要的比较次数和转置法调整自组织线性表的内容。

5.伪代码：

```
int main(int argc, char** argv) {
const char* FileIn[5] = {//线性表文档
            "text1.txt",
            "text2.txt",
            "text3.txt",
            "text4.txt",
            "text5.txt"
    };
    const char* Fileresult[5] = {//结果文档
           "result1.txt",
           "result2.txt",
           "result3.txt",
           "result4.txt",
           "result5.txt"
    };

    cout<<"---欢迎来到自组织线性表的文件内容查找---"<<endl;
    cout<<"请输入要打开的文件（这里只有5个）：例如打开text1则输入1，结果在
result1中显示"<<endl;
    int name;
    cin>>name;
    ifstream infile,in;
    ofstream outfile;

    in.open(FileIn[name-1],ios::in);
    outfile.open(Fileresult[name-1],ios::out);//输出查找结果的文件
    string  temp,temp2;

    in>>temp2;//获取查找文件中原来线性表的长度
    in.close();//关闭文件关联
    infile.open(FileIn[name-1],ios::in);//查找内容的文件
    //in.open(Find[name-1],ios::in);
```

```
        int n=0;
        n=temp2.length();
        SOList list(n);
        int flag=0;//插入回合
        while(!infile.eof()) {//到文件末尾则停止
                infile>>temp;//把文件的一行字符串输出到temp里,遇到空格或者换行结
    束
                for (int i = 0; i < temp.length(); i =i+2) {
                        if (flag==0) {//插入回合
                                list.append(temp[i], temp[i + 1]);
                        } else {//查找回合
                                int resul = list.find(temp[i],temp[i+1]);
                                if (resul != -1) {//查找到了
                                        outfile<<"查找成功!查找次数为:"<<resul/2+1;
                                        outfile<<endl;
                                        outfile<<"当前自组织线性表为：";
                                        for(int i=0;i<n;i+=2)
                                        {

        outfile<<list.getValue(i)<<list.getValue(i+1);
                                        }
                                        outfile<<endl;
                                } else {
                                        outfile<<"查找失败!查找次数为:"<<n/2;
                                        outfile<<endl;
                                        outfile<<"当前自组织线性表为：";
                                        for(int i=0;i<n;i+=2)
                                        {
                                outfile<<list.getValue(i)<<list.getValue(i+1);
                                        }
                                        outfile<<endl;
                                        //outfile<<list.print();
                                }
                        }
                }
                flag=1;//进入查找回合
        }
        infile.close(); //关闭与文件的连接
       outfile.close();//关闭与文件的连接
        return 0;
}
```

流程图：


开始
用户输入文件名
打开相应文件，读取第一行内容获取要构造的线性表长度
Flag==0
YES
NO
读入文件第二行内容，进行查找
读入文件第一行内容，构造线性表，并且flag=1；
Resul返回值为-1
NO
YES
输出查找失败！和比较次数以及转置后的线性表
输出查找成功！和比较次数以及转置后的线性表
结束

4.算法的时空分析

（1）构造线性表模块：

这一模块主要是用for循环来构造线性表，for循环次数取决于文件中第一行汉字的个数，所以时间复杂度为θ（n）。由于是用数组实现的线性表，并且每一个汉字占两个数组空间，所以数组的空间代价为θ（2n），化简后为θ（n）。

（2）查找和转置模块：

这一模块主要是利用数组的查找功能，然后每一次查找结束后进行转置。由于需要利用for循环遍历数组，所以所以总时间代价为θ（n）。查找不需要空间开销，但是转置需要。因为转置需要交换两个汉字，如果汉字是数组的第一个则不需要转置，如果汉字不是数组的第一个则需要转置。这里申请两个字符空间来存放查找汉字的前一个汉字，然后将查找的汉字向前移动，再将存放空间中的汉字放在查找汉字原来的位置。这样就实现了转置。所以从要申请新的空间来看，总空间代价为θ（n）。故该模块的空间代价为θ（n）。

四、调试分析

1.调试方案设计

调试目的：为了发现代码是否有语法错误、连接错误、逻辑错误和运算错误，有不严谨之处。

样例：

1

设置断点处：查找模块处。

2. 调试过程和结果，及分析

调试过程：对线性表进行遍历，如果找到了这个汉字则返回查找次数，如果找不到则返回-1，然后输出转置后的线性表。

问题：发现输出的转置后的线性表中汉字个数不对。

解决方法：在数组中添加一个新成员存储数组中汉字的个数，然后在插入汉字的时候listSize++，利用数组的getvalue函数来获取汉字。这里的getvalue函数是用for循环来对线性表遍历，遍历终止条件是线性表的存储长度。

原因：没有准确掌握数组中存储的线性表长度。因为是在查找模块输出查找后的线性表，刚开始的getvalue函数是用文件中查找的汉字个数，并不是用线性表的长度所以导致了输出的汉字个数不对这种情况的发生。

五、测试结果

测试样例一：（普通）

错误输出：

```
---欢迎来到自组织线性表的文件内容查找---
请输入要打开的文件（这里只有5个）：例如打开text1则输入1，结果在result1中显示
1
查找成功！查找(我)一共比较了1次
当前顺序表存储的汉字为：
我爱你祖国
查找成功！查找(爱)一共比较了2次
当前顺序表存储的汉字为：
爱我你祖国
查找失败！查找(过)一共比较了5次
当前顺序表存储的汉字为：
爱我你祖国
查找成功！查找(我)一共比较了2次
当前顺序表存储的汉字为：
我爱你祖国
查找成功！查找(爱)一共比较了2次
当前顺序表存储的汉字为：
爱我你祖国
查找失败！查找(过)一共比较了5次
当前顺序表存储的汉字为：
爱我你祖国
请按任意键继续. . .
```

正确输出：

```
---欢迎来到自组织线性表的文件内容查找---
请输入要打开的文件（这里只有5个）：例如打开text1则输入1，结果在result1中显示
1
查找成功！查找(我)一共比较了1次
当前顺序表存储的汉字为：
我爱你祖国
查找成功！查找(爱)一共比较了2次
当前顺序表存储的汉字为：
爱我你祖国
查找失败！查找(过)一共比较了5次
当前顺序表存储的汉字为：
爱我你祖国
请按任意键继续. . .
```

分析结论：该样例是最普通的样例，线性表中有5个汉字，要查找的汉字个数小于线性表汉字个数。这里查找“我”，因为我是线性表第一个所以不转置，查找“爱”是线性表第二个所以将它和前面的那个转置。因为查找“过”字不存在，所以比较次数为5次，查找失败！这里因为查找的汉字只有3个，只查找三次即可，而不是查找6次。

测试样例二：（特殊样例，查找字数大于线性表中汉字个数）

输入：

2

输出：

```
---欢迎来到自组织线性表的文件内容查找---
请输入要打开的文件（这里只有5个）：例如打开text1则输入1，结果在result1中显示
2
查找失败！查找(七)一共比较了8次
当前顺序表存储的汉字为：
数据结构课程实验
查找失败！查找(段)一共比较了8次
当前顺序表存储的汉字为：
数据结构课程实验
查找成功！查找(数)一共比较了1次
当前顺序表存储的汉字为：
数据结构课程实验
查找失败！查找(码)一共比较了8次
当前顺序表存储的汉字为：
数据结构课程实验
查找失败！查找(管)一共比较了8次
当前顺序表存储的汉字为：
数据结构课程实验
查找失败！查找(啦)一共比较了8次
当前顺序表存储的汉字为：
数据结构课程实验
查找失败！查找(啦)一共比较了8次
当前顺序表存储的汉字为：
数据结构课程实验
查找失败！查找(啦)一共比较了8次
当前顺序表存储的汉字为：
数据结构课程实验
查找成功！查找(数)一共比较了1次
当前顺序表存储的汉字为：
数据结构课程实验
查找成功！查找(据)一共比较了2次
当前顺序表存储的汉字为：
据数结构课程实验
请按任意键继续. . .
```

分析结论：该样例是查找汉字个数大于线性表中存储的个数。查找的汉字有10个，线性表中的汉字有8个。线性表中存储的汉字为：数据结构课程实验，待查找的汉字为：七段数码管啦啦啦数据。

测试样例三：（普通）

输入：

3

输出：

```
---欢迎来到自组织线性表的文件内容查找---
请输入要打开的文件（这里只有5个）：例如打开text1则输入1，结果在result1中显示
3
查找成功！查找(数)一共比较了3次
当前顺序表存储的汉字为：
我数爱据结构——
查找成功！查找(据)一共比较了4次
当前顺序表存储的汉字为：
我数据爱结构——
查找成功！查找(结)一共比较了5次
当前顺序表存储的汉字为：
我数据结爱构——
查找成功！查找(构)一共比较了6次
当前顺序表存储的汉字为：
我数据结构爱——
查找失败！查找(不)一共比较了8次
当前顺序表存储的汉字为：
我数据结构爱——
查找成功！查找(爱)一共比较了6次
当前顺序表存储的汉字为：
我数据结爱构——
查找成功！查找(我)一共比较了1次
当前顺序表存储的汉字为：
我数据结爱构——
请按任意键继续. . .
```

分析结论：该样例是普通样例，和样例一一样，这里线性表存储的汉字为：我爱数据结构一一查找的汉字为：数据结构不爱我。

测试样例四：（特殊，线性表都是同样的内容，查找的文字也是并且都查找失败）

输入：

4

输出：

```
---欢迎来到自组织线性表的文件内容查找---
请输入要打开的文件（这里只有5个）：例如打开text1则输入1，结果在result1中显示
4
查找失败！查找(二)一共比较了7次
当前顺序表存储的汉字为：
一一一一一一一
查找失败！查找(二)一共比较了7次
当前顺序表存储的汉字为：
一一一一一一一
查找失败！查找(二)一共比较了7次
当前顺序表存储的汉字为：
一一一一一一一
查找失败！查找(二)一共比较了7次
当前顺序表存储的汉字为：
一一一一一一一
查找失败！查找(二)一共比较了7次
当前顺序表存储的汉字为：
一一一一一一一
请按任意键继续. . .
```

分析结论：这是一个特殊样例。该样例是线性表中汉字都为一，查找的汉字都是二，并且在线性表中查不出来。

测试样例五（普通）

输入：

5

输出：

```
---欢迎来到自组织线性表的文件内容查找---
请输入要打开的文件（这里只有5个）：例如打开text1则输入1，结果在result1中显示
5
查找成功！查找(我)一共比较了6次
当前顺序表存储的汉字为：
期末了希我望能顺
查找失败！查找(觉)一共比较了8次
当前顺序表存储的汉字为：
期末了希我望能顺
查找失败！查找(得)一共比较了8次
当前顺序表存储的汉字为：
期末了希我望能顺
查找失败！查找(海)一共比较了8次
当前顺序表存储的汉字为：
期末了希我望能顺
查找失败！查找(信)一共比较了8次
当前顺序表存储的汉字为：
期末了希我望能顺
请按任意键继续. . .
```

分析结论：这是一个简单普通样例。存储的汉字有8个，他们是：期末了希望我能顺，查找的汉字有5个，他们是：我觉得海信。

六、实验日志

2018/12/23

内容：实验七上机

得知要对文件的操作，所以回去翻看了c++中关于文件操作的部分

2018/12/27

内容：如何存储汉字

1.Char数组：

初想法：刚开始的想法是用char数组来构成一个结构体，然后在数组中调用这个结构体来存储每一个汉字。

遇到的问题：之后对于汉字的查找操作不知道怎么查找，故放弃。

解决方法：直接用char数组存储，不再写结构体。

2018/12/28

内容：确定查找和转置函数

遇到的问题：

1.不知道怎样输出转置后的线性表。

解决方法：利用getvalue函数来得到线性表的每一个汉字，再将其输出。

2.输出的转置后的线性表中汉字个数不对

解决方法：将getvalue函数的for循环次数由查找汉字个数改为数组中存储的汉字个数。

3.查找的汉字有3个但却输出6次

解决方法：在while循环中将while(infile.peek!=EOF)改为while(!infile.eof())，这也是询问同学后得知。刚开始在网上如何结束文件的读取，网站上给的是利用这种操作：

主要是把eof()改为peek() == EOF来判别，其中peek()是取文件当前指针，EOF是文件尾尾标符，它的值为-1，所以采用这种方法就解决由eof()导致文本中最后一行的数据写了两遍的问题了。但是不知道为什么用到这里却输出了6次查找结果。

4.线性表中存储的文字个数不能超过8个

解决方法：在多次尝试将线性表的最大空间从100改为999之后发现还是不可以，于是将“线性表中存储的文字个数不能超过8个”加入文档说明的限制条件中。

5.存储的线性表不能有空格，查找的汉字也不能包括空格

解决方法：由于采用的是infile>>temp，这种输入方法，所以不能读取空格，但是也不知道如何从文件中可以读取空格，所以将“存储的线性表不能有空格，查找的汉字也不能包括空格”放入文档说明的限制条件中。

经验和体会：这次实验七主要是复习了对文档的操作以及如何存储汉字。在实现过程中也上网查找了许多资料，但是最后由于限制条件太多，也没能找到为什么会出现这样的结果，所以我认为这次实验七的结果做的并不理想。但我还是通过实验七熟悉了文件的读取和写入操作，虽然有遗憾但还是有收获的。