数据结构

课程实验报告

题　　目：　自组织查找表

学生姓名：　魏子铖

学生学号：　201726010308

专业班级：　软件 1703

完成时间：　2018.12.16

一、需求分析

0) 问题分析

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。

现在，我们有如下的需求：

- 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。
- 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）。

对于上述问题，需要实现的功能有：

- 将待查找序列存储在自组织线性表中。
- 将所有需查询的汉字依次读入，并对自组织线性表采用转置法。
- 把每个汉字的查询结果格式化输出到文件中。

1) 输入数据

【输入格式】

- 输入的第一行包含一组汉字集合，它们的值各不相同，表示自组织线性表中的查找元素关键字。
- 接下来若干行，是待查询的汉字，它们中的部分值不存在于自组织线性表中。

【输入样例】

一二三四五六

三三四

2) 输出数据

【输出格式】

- 输出若干行，每一行表示一个汉字的查询结果。
- 最后一行输出进行若干次转置后的自组织线性表中的值，按照顺序输出，以空格间隔。

【输出样例】

查询成功，查找次数为 3

查询成功，查找次数为 2

查询成功，查找次数为 3

三 一 四 二 五 六

3) 测试样例设计

样例一

样例输入

一二三四五六

三三四

样例输出

查找成功，查找次数为 3
查找成功，查找次数为 2
设计理由：一般情况

样例二

样例输入

一二三四五六
六五六五

样例输出

查找成功，查找次数为 6
查找成功，查找次数为 6
查找成功，查找次数为 6
查找成功，查找次数为 6
设计理由：每次都查询自组织线性表中的最后一个值

样例三

样例输入

一二三四五六
一一一

样例输出

查找成功，查找次数为 1
查找成功，查找次数为 1
查找成功，查找次数为 1
设计理由：每次都查询自组织线性表中的第一个值

样例四

样例输入

一二三四五六
七八九

样例输出

查找失败，查找次数为 6
查找失败，查找次数为 6
查找失败，查找次数为 6
设计理由：所有待查询的值都不在自组织线性表中

样例五

样例输入

一二三四五六
二一二三

样例输出

查找成功，查找次数为 2
查找成功，查找次数为 2
查找成功，查找次数为 2
查找成功，查找次数为 3
设计理由：自组织线性表中随机查询

二、概要设计

1. 抽象数据类型

为实现上述功能，由于题目规定输入数据均为汉字，所以使用两个 char 类型变

量来存储一个汉字，并将用户的输出储存在自组织线性表中。

抽象数据类型设计：

- 数据对象：一组互不相同的汉字，表示自组织线性表中的值。
- 数据关系：查找表中的第一个元素只有一个后继，最后一个元素只有一个前驱，其他的元素既有一个前驱，也有一个后继。
- 基本操作：向自组织线性表中插入一个元素；查找该元素是否在表中；对查找到的元素进行转置法操作。
- **ADT：**

  SOArrayList {

  数据对象：D = { <key_i>| key_i∈UTF-8, i = 1, 2, 3, ……, n, 1≤n≤1000 }

  数据关系：R = { <key_i, key_{i+1}> | <key_i, key_{i+1}>∈SOArrayList, key_i ≠ key_j}

  基本操作：

  ```
  void append(const char, const char);
  //向自组织线性表中插入元素，时间复杂度 O（1）
  int find(const char, const char);
  //返回被查询元素在自组织线性表中的下标，时间复杂度 O（n）
  void renovate(const int);
  //将自组织线性表中该位置的元素与前一个元素交换位置，时间复杂度 O（1）
  ```

  }

2. 算法的基本思想

对于各种功能：

1) 建立自组织线性表

   基于顺序表来建立自组织线性表，线性表中的每一个元素都有唯一的位置。

2) 实现自组织查找表的查找功能

   对自组织查找表进行 for 循环的遍历即可。

3) 实现自组织查找表的转置功能

   对上一查找模块得到的下标位置进行操作，将自组织线性表中该位置的元素与前一个元素交换位置，若下表为 0，则不进行转置。

4) 对结果进行格式化输出

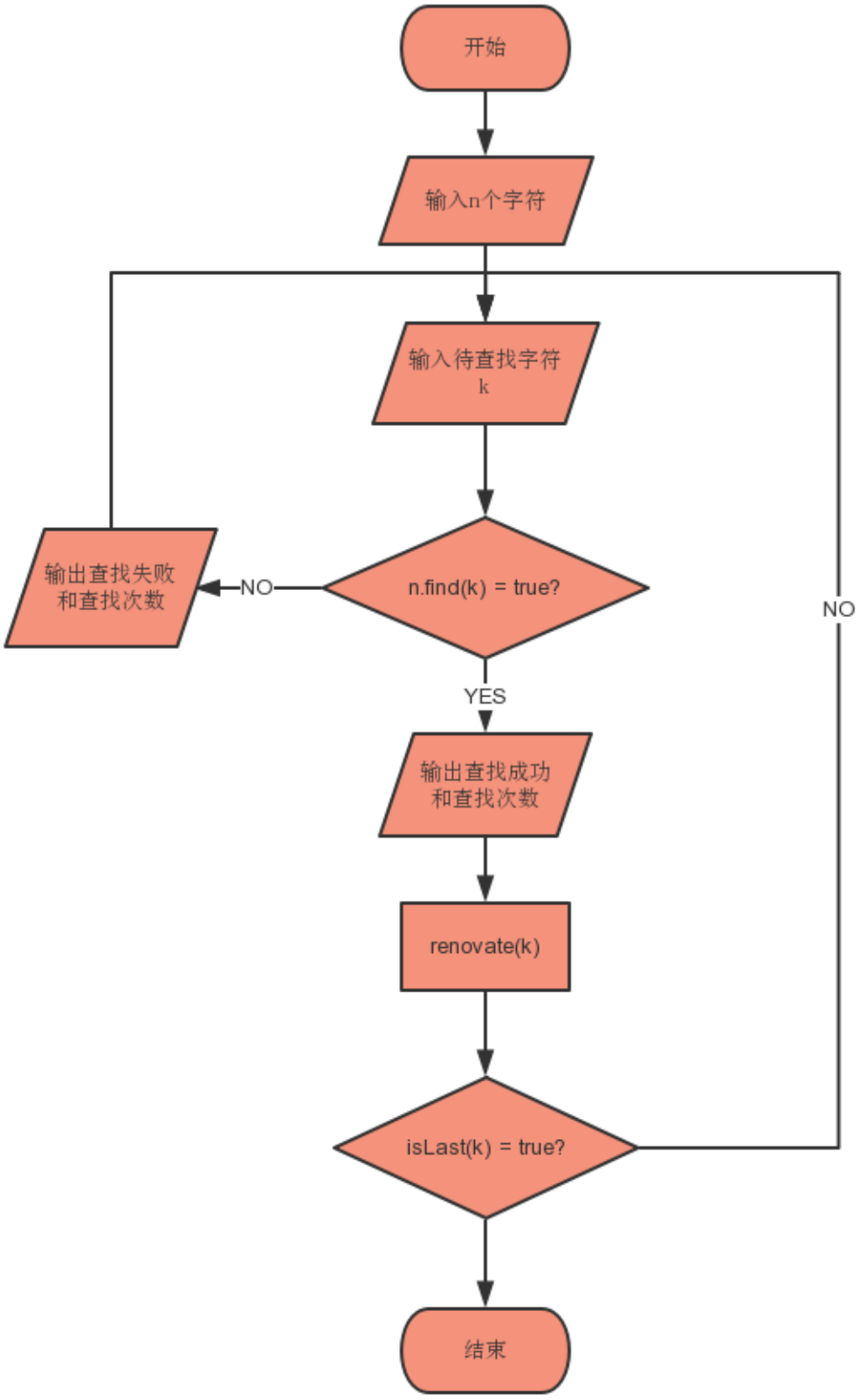
   从文件中依次读入需查询的汉字，把查询结果保存在文件中（如找到，返回比较的次数，如果没有找到，返回比较的次数）。

3. 程序的流程

程序由四个模块组成：

1) 输入模块：根据文件路径打开文件，对其中的内容进行读取。
2) 构建自组织查找表：将从文件中读取的内容插入到自组织查找表中。
3) 查找模块：顺序遍历查找表，若找到该元素，则对表中的该元素执行转置操作。
4) 输出模块：格式化输出查找信息到文件中。

程序流程图如下：



三、详细设计

1. 物理数据类型

输入的数据为自组织线性表的内容与待查找元素的值，均为汉字，由于自组织线性表是基于顺序表实现的，满足顺序特征，所以逻辑实现上可以采用数组的形式。

2. 输入和输出的格式

从文件按行读取输入，将第一行的元素存储到自组织线性表中，之后的每一行待查找元素对应的查找结果输出到相应的文件中。

3. 算法的具体步骤

a) 基于顺序表建立自组织线性表

按照输入顺序将输入值依次插入线性表的尾部，线性表中的每一个元素至此都有一个唯一的位置。

```
void SOArrayList::append(const char k1, const char k2) {
    //一个汉字占两个 char 类型的存储空间
    listArray[currSize++] = k1;
    listArray[currSize++] = k2;
}
```

b) 对输入元素进行查找

从头至尾遍历线性表，若找到，则返回对应数组的下标，若找不到，则返回线性表的长度。

```
int SOArrayList::find(const char k1, const char k2) {
    int index = 0;
    bool flag = false;
    //遍历自组织线性表中的每个元素
    for (int i = 0; i < maxSize; i += 2) {
        //若找到，则结束循环
        if (listArray[i] == k1 && listArray[i + 1] == k2) {
            index = i;
            flag = true;
            break;
        }
    }
    //若没找到，则返回线性表的长度
    if (index == 0 && !flag) {
        return maxSize;
    }
    //若找到，则返回对应数组的下标
    return index;
}
```

c) 若待查找元素存在于自组织线性表中，则执行转置操作

在找到需要的元素之后，只要该元素不在线性表的开头，就将该元素与其前驱元素调换位置。

```
void SOArrayList::renovate(const int index) {
    //若该元素在现线性表的开头，则不进行操作
    if (index == 0) {
        return;
    }
```

```
            //否则将该元素与其前驱元素调换位置
            char temp1 = listArray[index];
            listArray[index] = listArray[index - 2];
            listArray[index - 2] = temp1;
            char temp2 = listArray[index + 1];
            listArray[index + 1] = listArray[index - 1];
            listArray[index - 1] = temp2;
        }
```

d) 将查询结果输出到相应文件中

```
    void SOArrayList::print() {
        for (int i = 0; i < maxSize; i += 2) {
            std::cout << listArray[i] << listArray[i + 1] << ' ';
        }
    }
```

4. 算法的时空分析

a) 建立自组织线性表，时间复杂度 O（n）

b) 自组织线性表的查找功能，时间复杂度 O（n）

c) 对线性表中元素执行转置操作，时间复杂度 O（1）

四、调试分析

1.调试方案设计

调试目的：发现思维逻辑与代码实现上的区别，改进代码结构，排除语法逻辑上的错误。

样例：

一二三四五六

四五四六

调试计划：设置好断点，注意观察每一步时各个变量的变化情况，找出错误的地方，然后改正；单步调试，更能准确定位出现错误的代码区域。

2.调试过程和结果，及分析

调试过程中由于出现数组越界情况，导致代码多次崩溃，发现是next()方法出现了问题，排除错误后调试成功，输出了正确的结果。

五、测试结果

1. 样例一

```
查找成功，查找次数为3
查找成功，查找次数为2
```

```
test1.txt经过查询后线性表中的值为：
三 一 二 四 五 六
```

一般情况，输出结果正确

2. 样例二

```
查找成功，查找次数为6
查找成功，查找次数为6
查找成功，查找次数为6
查找成功，查找次数为6
```

```
test2.txt经过查询后线性表中的值为：
一 二 三 四 五 六
```

每次都查询自组织线性表中的最后一个值，输出结果正确

3. **样例三**

```
查找成功，查找次数为1
查找成功，查找次数为1
查找成功，查找次数为1
```

```
test3.txt经过查询后线性表中的值为：
一 二 三 四 五 六
```

每次都查询自组织线性表中的第一个值，输出结果正确

4. **样例四**

```
查找失败，查找次数为6
查找失败，查找次数为6
查找失败，查找次数为6
```

```
test4.txt经过查询后线性表中的值为：
一 二 三 四 五 六
```

所有待查询的值都不在自组织线性表中，输出结果正确

5. **样例五**

```
查找成功，查找次数为2
查找成功，查找次数为2
查找成功，查找次数为2
查找成功，查找次数为3
```

```
test5.txt经过查询后线性表中的值为：
二 三 一 四 五 六
```

自组织线性表中随机查询，输出结果正确

六、实验日志

12/19

对题目分析后确定用自组织线性表以及相应的算法可以解决，再设计输入输出格式与抽象数据类型还有算法思想。

阅读题目后，以前碰到题都直接看有没有思路，这次脑子里首先想到的是用什么数据结构存储，然后才是算法思想。

12/20

完成了主函数的实现，在主函数中分为输入、计算和输出三个模块。调用类中的函数先构建自组织线性表，然后寻找每一个待查询元素，再对找到的元素进行转置操作。

用自组织线性表 ADT 去解决问题时，尤其注意 ADT 里基本操作的使用。

12/22

在大概完成整个程序，编译运行后，对设计的测试样例进行调试测试，在这个过程中发现了错误，经程序调试后发现是逻辑错误导致数组越界崩溃，在调试后修改正确。

调试对于完善和找到程序的 bug 十分重要。