

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： | 2018.12.15 |

1. **需求分析**
2. **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

1. **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

1. **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：**自组织线性表中随即查询

**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 = {** **<keyi>| keyi∈UTF-8, i = 1, 2, 3, ……, n, 1≤n≤1000 }**

**数据关系：R = {** **<keyi, keyi+1> | <keyi, keyi+1>∈SOArrayList, keyi ≠ keyj }**

**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。

1. 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。

1. 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。

1. 对结果进行格式化输出

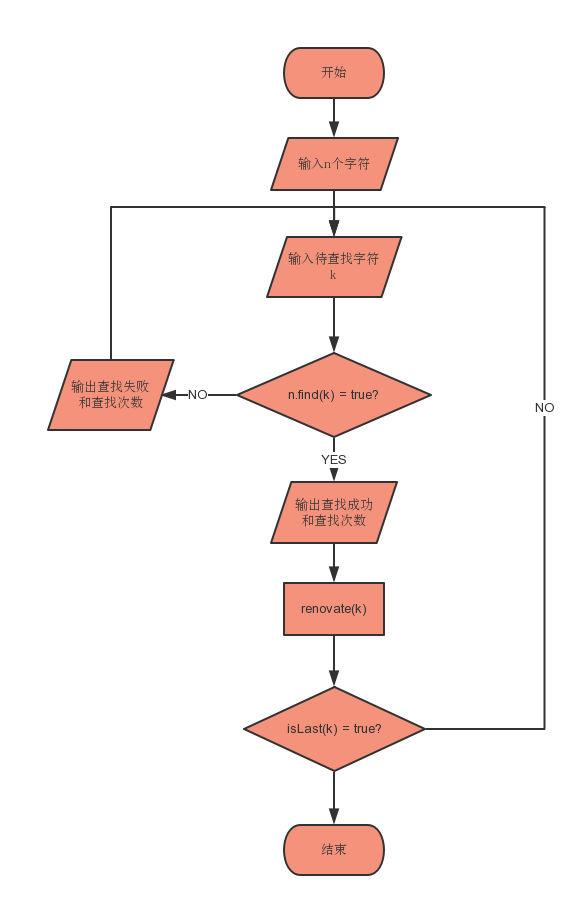
从文件中依次读入需查询的汉字，把查询结果保存在文件中（如找到，返回比较的次数，如果没有找到，返回比较的次数）。

**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**

输入的数据为有向图的信息与有向边的信息，都是整数类型，由于有向图是基于邻接矩阵的，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**

输入时有提示语句，说明输入的方法与结束条件，将每个元素存储在同一个有向图，将查找信息格式化输出。

**3. 算法的具体步骤**

1. **基于邻接矩阵建立有向图**

邻接矩阵是表示一个图的常用存储表示。它用两个数组分别存储数据元素（顶点）的信息和数据元素之间的关系（边或弧）的信息。

阶为n的图G的邻接矩阵A是n×n的。将G的顶点标签为v1, v2, …, vn。若 (vi, vj) ∈ E(G)，Aij = 1，否则Aij = 0。

**void MatrixGraph::setEdge(int vt1, int vt2, int wt) {**

**try {**

**if (wt <= 0) {**

//如果边权不是正整数，则抛出非法输入的警告

**throw "Illegal weight value";**

**}**

**if (matrix[vt1][vt2] == 0) {**

//如果该边没有被设置过，则边数+1

**numEdge++;**

**}**

//设置边权，表示从vt1指向vt2的边权为wt

**matrix[vt1][vt2] = wt;**

**}**

**catch (const char\* str) {**

//输出错误信息

**std::cerr << str << std::endl;**

**}**

**}**

1. **对每个顶点执行一次深度优先搜索**

从一个顶点v0开始，沿着一条路一直走到底，如果到达底部，那就返回到上一个节点，然后从另一条路开始走到底，这种尽量往深处走的概念即是深度优先的概念。

**void DFSTraverse(MatrixGraph\* m, int vt) {**

**m->setMark(vt, VISITED);**

//将访问过的顶点数组置为VISITED

**for (int w = m->getFirst(vt); w <= m->n(); w = m->next(vt, w)) {**

//如果没有到达路径的最底部，则继续往深处遍历

**if (m->getMark(w) == UNVISITED) {**

//如果改顶点没有被访问过，则对该顶点继续执行深度优先搜索

**DFSTraverse(m, w);**

**}**

**}**

**}**

1. **对深度优先搜索的结果即标记数组进行处理**

标记数组的每一行表示从该顶点出发可以访问到的所有顶点，若该行的值全为1，则表明这个部门知道其他所有部门的存在，则count值加一。

**int count = 0;**

**for (int i = 0; i < N; i++) {**

//外层循环遍历每个部门

**for (int j = 0; j < N; j++) {**

//内层循环遍历该部门与其他部门的访问结果

**if (visit[i][j] == 0) {**

//若部门i不知道部门j的存在，则结束该次循环

**break;**

**}**

**if (j == N - 1) {**

//若该部门知道其他所有部门的存在，则count++

**count++;**

**}**

**}**

**}**

1. **格式化输出结果**

输出count值即为所求答案

**std::cout << count << std::endl;**

**4. 算法的时空分析**

1. 设置有向图中边的信息，时间复杂度O（1）。
2. 对每个顶点的深度优先搜索，时间复杂度O（n²+e）。
3. 对标记数组进行处理，时间复杂度O（n²）。

**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。

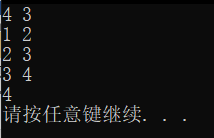
**五、测试结果**

1. **样例一**



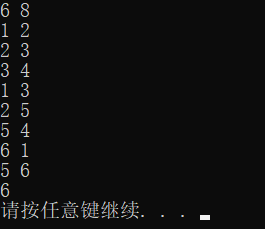
一般情况，输出结果正确

1. **样例二**



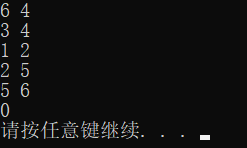
有向图为链式结构，所有部门都知道其他的所有部门，输出结果正确

1. **样例三**



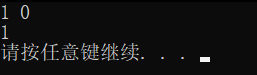
有向图中存在回路，对深度优先搜索没有影响，输出结果正确

1. **样例四**



有向图不是连通图，则所有部门都不知道其他所有部门，输出结果正确

1. **样例五**



有向图中只有一个点，显然输出为1，输出结果正确

**六、实验日志**

**12/08**

在历年的三个题目中选择了通信网络，对题目分析后确定用图以及图的算法可以解决，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oj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。

**12/10**

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图，然后用深度优先搜索遍历所有的部门，再次遍历计算哪些部门知 道所有部门的存在。

用图 ADT 去解决问题时，尤其注意ADT里基本操作的使用。

**12/12**

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指针指向崩溃，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。