**山东大学计算机科学与技术学院  
《数据结构与算法》课程设计报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201700301147 | 姓名： 杜瀛川 | | 班级： 17.4 |
| 上机学时：16 | | 日期： 2019.4.15 | |
| 课程设计题目：N-ary Trie | | | |
| 软件开发环境：Qt Creater | | | |
| 项目最终效果图          报告内容：  1.需求描述  1.1 问题描述  哈夫曼算法输出的结果就是一个二元Trie，在二元Trie中，每个左子树分支用0表示，右子树用1表示。可以将二元Trie扩展到N-ary Trie。在N-ary Trie中，每个结点都有0~N之间任何个儿子结点，其中每个分支都用一个相应的符号表示（在N-ary Trie中有N个不同的符号）。  字典树又称单词查找树，[Trie树](https://baike.baidu.com/item/Trie%E6%A0%91" \t "https://baike.baidu.com/item/%E5%AD%97%E5%85%B8%E6%A0%91/_blank)，是一种[树形结构](https://baike.baidu.com/item/%E6%A0%91%E5%BD%A2%E7%BB%93%E6%9E%84/9663807" \t "https://baike.baidu.com/item/%E5%AD%97%E5%85%B8%E6%A0%91/_blank)，是一种哈希树的变种。典型应用是用于统计，排序和保存大量的[字符](https://baike.baidu.com/item/%E5%AD%97%E7%AC%A6" \t "https://baike.baidu.com/item/%E5%AD%97%E5%85%B8%E6%A0%91/_blank)串（但不仅限于字符串），所以经常被搜索引擎系统用于文本词频统计。它的优点是：利用字符串的公共前缀来减少查询时间，最大限度地减少无谓的字符串比较，查询效率比哈希树高。 字典树基本性质： 1、根节点不包含字符，除根节点外的每一个子节点都包含一个字符。  2、从根节点到某一节点。路径上经过的字符连接起来，就是该节点对应的字符串。  3、每个节点的所有子节点包含的字符都不相同。  d62a6059252dd42a745cc2c2033b5bb5c9eab806  1.2 基本要求  1、设计并实现N-ary Trie的ADT（N =26,建立在英语上的Trie）,该ADT包括Trie的组织存储以及其上的基本操作：包括初始化，查找，插入，删除等。  2、应用Trie结构实现文本文档的索引化。首先扫描文本文档（存放在txt文件中），然后在此基础上用Trie登记单词出现的位置（行号），最后在Trie上实现查询。  3、用户输入的查询可以针对一个单词的，也可以是针对一个字符序列的（以某几个字母开头的单词）。  1.3扩展功能  1、空间优化，双数组Trie（Double-ArrayTrie）。  2、字典树的图形化显示。  3、实现文档单词字典排序。  4、实现可设置前缀的文档词频排序，两种实现方法：内排序，外排序（竞赛树）。  1.4 输入说明  主程序与用户交互实验图形化界面，为了使功能展示简洁直观，切换自如，故使用多个子窗口设计，分别是文件子窗口（记事本功能），排序子窗口（字典排序、词频排序），图形子窗口（Trie图形化）。同时一级菜单中还有重启程序，字典树初始化，插入节点、删除节点功能。如图1所示。    图1  1.4.1 文件界面  文件界面实现了类似记事本的相关功能，如图2所示。    图2  1.4.1.1 打开文件  选择要在窗口中进行操作的文件，文档内容显示在文本框中，如图3所示。    图3  1.4.1.2 生成文件  随机生成文件，可设置单词数量和单词最大长度。随机生成满字典树的文件，可设置单词最大长度，如图4所示。      图4  1.4.1.3 保存文件  可直接原位置保存文件，也可选择任何位置保存文件，如图5所示。    图5  1.4.1.4 删除文件  提供了删除所选择文件的功能。  1.4.1.5 文本操作  提供了字体选择、复制、粘贴、剪切、清除等功能，如图6所示。      图6  1.4.2 图形界面  图形子界面提供了字典树的图形化显示功能，如图7所示。    图7  1.4.2.1 聚焦  输入字符串，聚焦到所选顶点，如图8所示。    图8  1.4.2.2 查找  输入字符串，场景中只显示所选字符串的子树，如图9所示。    图9  1.4.3 排序界面  排序界面提供了对初试字典树的文件进行单词排序和查找的功能，如图10所示。    图10  1.4.3.1词频排序  输入字符串或前缀，程序输出词频排序的结果，如图11所示。  1.4.3.2字典排序  输入字符串或前缀，程序输出字典排序的结果，如图11所示。  1.4.3.3查询  输入字符串，查询单词在文本中出现的位置，如图11所示。    图11  1.4.4 初始化  提供了两种初始化字典树的方法，分别是从所选文件中初始化字典树，还有在弹出的输入窗口中输入要进行初始化的文本，如图12所示。      图12  1.4.5 插入  输入要插入的字符串和行号，即可将字符串插入到字典树中，如图13所示。    图13  1.4.6 删除  输入要删除的字符串，即可将该节点删除，如图14所示。    图14  1.5 输出说明  1.5.1 文件界面  1.5.1.1 打开文件  打开文件的结果如图15所示。    图15  1.5.1.2 生成文件  生成的文件存储在硬盘中，内容显示在文件界面上，文件地址显示在状态栏中。如图16。    图16  1.5.1.3 保存文件  将此时文件编辑器中的内容保存回原文件或者保存到所选文件中。  1.5.2 图形界面  图形化的字典树显示在文件中，如图17所示。    图17  1.5.2.1 聚焦  聚焦到所选节点的图形化，如图18所示。    图18  1.5.2.2 查找  显示所选子树的图形化，如图19所示。    图19  1.5.3 排序界面  1.5.3.1词频排序  词频内排序或者词频外排序的结果显示在文本框中，如图20所示。    图20  1.5.3.2字典排序  输入字符串或前缀，程序输出字典排序的结果，如图21所示。    图21  1.5.3.3查询  输入字符串，查询单词在文本中出现的行号，如图22所示。    图22  1.5.4 初始化 插入 删除  初始化、插入、删除的方法可以用图形界面表示，排序结果在排序界面中显示。  2.设计  2.1 系统结构设计    2.2 设计思路  本问题是N-ary Trie结构的实现和分析，我使用了两种实现方法，分别是链表实现和double array trie双数组字典树（DAT）实现。  链表实现定义节点类，顶点类中有指向子节点的指针数组，以链表形式表示树的层次关系。  DAT使用两个数组表示字典树的层次关系，分别是base(基数组)和check(父数组)。通过两个数组值的数值计算表示字典树的结构。  2.3 链表实现  2.3.1 结点类TrieNode  TrieNode\*类型的指针数组指向所有子节点，节点类中的vector用来存储该节点单词出现的行号，char\* 类型的变量word是该节点表示的单词。height变量表示节点层数，为图形化提供参照。    2.3.2 字典树类Trie  提供了字典树的插入、删除、查找、先序遍历、析构等函数。    2.3.3 插入  定义当前节点变量pRoot，对输入字符数组逐个字符进行遍历操作，根据当前字符的序号确定当前节点转移到指针数组的那一节点。若在转移过程中发现空节点，则对节点进行初始化，直到字符数组遍历完成。    2.3.4 查找  定义当前节点变量pRoot，对输入字符数组逐个字符进行遍历操作，根据当前字符的序号确定当前节点转移到指针数组的那一节点。若在转移过程中发现空节点，则查找失败，没有要查找的节点。    2.3.5 删除  定义当前节点变量pRoot，对输入字符数组逐个字符进行遍历操作，根据当前字符的序号确定当前节点转移到指针数组的那一节点。若在转移过程中发现空节点，则查找失败，没有要查找的节点。若查找到该节点，则对该节点进行析构。    2.3.6 先序遍历  递归得对每一个节点进行先序遍历操作，将结果存入vector中。    2.3.7 析构  递归得对每一个节点进行析构操作。    2.4 double array trie双数组字典树（DAT）实现  STATE（i）：状态，实际为在数组中的下标  BASE ：表示后继节点的基地址的数组，叶子节点没有后继，标识为字符序列的结尾标志  CHECK ：标识前驱节点的地址    2.4.1 DAT类DoubleArrayTrie  关键成员变量为base数组和check数组，用来表示树节点之间的层次关系。使用stl的vector存储。      2.4.算法设计及分析  2.4.1 构造DAT  1 建立根节点root,令base[root] =1  2 找出root的子节点 集{root.childreni }(i = 1...n) , 使得 check[root.childreni ] = base[root] = 1  3 对 each element in root.children :  　　1）找到{elemenet.childreni }(i = 1...n) ，注意若一个字符位于字符序列的结尾，则其孩子节点包 括一个空节点，其code值设置为0找到一个值begin使得每一个check[ begini + element.childreni .code] = 0  　　2）设置base[element.childreni] = begini  3）对element.childreni 递归执行步骤3，若遍历到某个element，其没有children，即叶节点，则设置base[element]为负值（一般为在字典中的index取负）  4 每个节点表示一个状态，子节点的check值即为父节点的base值  5 叶子节点的code值为0，叶子节点代表一个字符序列的结尾 且base[leaf] = -index。  6 对叶节点tleaf , check[tleaf] = tleaf， 因为到叶节点的转移字符leaf.code = 0,寻找begin值时，begin + leaf.code =tleaf ,check[begin+ leaf.code] = begin ， 由于leaf.code = 0 , 则有begin = tleaf，即check[tleaf] = tleaf。  伪代码表示：  初始化 ： code = ASCII+1 base[0] = 1  递归构造: base[s] +c1 = t1 base[s] +cn = tn (!check[t1] && !check[tn])  check[t] = base[s]  根节点： check[s] = i base[s] = -idx  实例构造过程  Dic = { AC,ACE,ACFF,AD,CD,CF,ZQ }            构造过程图示        2.4.2 查找  如执行搜索操作，如搜索AC，  check[base[root] + code(A)] == base[root] &&  check[base[base[root] + code(A)] + code(C)] == base[base[root] + code(A)]  && base[base[base[root] + code(A)] + code(C)] < 0  程序实现如下：    3. 测试结果  3.1图形化显示  一层满字典树    二层满字典树    聚焦结点    查找结点    3.2 文件操作  随机生成的最大长度为3的文件    修改字体操作    3.3 排序  对以下英文文件进行词频外排序    词频外排序结果    查询more单词位置结果    词频内排序结果    字典排序结果    以a为前缀的词频外排序结果    以b为前缀的字典排序结果    以c为前缀的词频外排序结果    3.4 插入与删除  原始字典树    删除m节点后    删除u节点后    插入m节点后     1. 分析与探讨   4.1 Trie操作复杂度分析  时间复杂度  insert search remove的时间复杂度只与操作节点的深度有关，而操作节点的深度就是 字符串的长度。假设所有字符串长度之和为n，构建字典树的时间复杂度为O(n)。假设要查找的字符串长度为k，查找的时间复杂度为O(k)  空间复杂度  Trie本身是一种空间换时间的数据结构，字典树每个节点都需要用一个数组来存储子节点的指针，即便实际只有两三个子节点，但依然需要一个完整大小的数组。所以，字典树比较耗内存，空间复杂度较高。  4.2 Double-ArrayTrie操作复杂度分析  查找时间复杂度  如执行搜索操作，如搜索AC，时间复杂度只与操作节点的深度有关。  check[base[root] + code(A)] == base[root] &&  check[base[base[root] + code(A)] + code(C)] == base[base[root] + code(A)]  && base[base[base[root] + code(A)] + code(C)] < 0  执行插入或删除操作时时间复杂度较高(相比于Trie)  如插入ACH，则需要将以AC为前缀的单词删除后，重新插入（包括ACH）  如删除ACH，则需要将以AC为前缀的单词删除后，重新插入（不包括ACH）  4.3 排序  字典排序  先序遍历Trie即可。  词频内排序  适用于字典树较小的情况，先序遍历Trie，结果存入vector，快速排序。  词频外排序  适用于字典树很大的情况，先序遍历Trie，结果存入vector，同时count计数， count即将超过规定内存容量时，vector数据输出到txt文件，清空vector，直 到遍历完成。对txt文件通过竞赛树结构实现外排序。    4.4 竞赛树类    4.5 Trie图形化  使用了QT提供的图元类QGraphicsView（InteractiveView）QGraphicsScene QGraphicsItem，使用深度优先递归遍历。  4.6 程序实现  文件io使用了QFile QTextStream类。  耗时操作（文件生成、Trie初始化、排序）全部在子线程中进行，主线程更新UI显示进度与结果，使用了QThread类。  程序采用多窗口结构，展示直观，方便功能切换。   1. 附录：实现源代码   因项目文件和源码过多，故将源码和可运行程序打包附在报告之后。项目图如图所示。 | | | |