计算机应用编程实验报告

实验1：高性能字符串匹配

组员：赵孔阳，王浪

赵孔阳：（i）完成处理GB2312编码及构造三叉Trie树；（ii）完成文件读写操作；（iii）完成报告书写。

王浪：（i）完成失效函数；（ii）完成AC搜索函数；（iii）完成代码注释。

针对代码中的某些问题共同调试

**1、实验背景**

在计算机科学领域， 串的模式匹配算法一直都是研究焦点之一。在拼写检查、 语言翻译、 数据压缩、 搜索引擎、 网络入侵检测、 计算机病毒特征码匹配以及DNA序列匹配等应用中， 都需要进行串匹配。串匹配就是在主串中查找模式串的一个或所有出现。

Trie树，又称单词查找树、 字典树， 是一种树形结构， 是一种用于快速检索的多叉树结构，应用有统计和排序大量的字符串、文本词频统计和文本检索。其优点是最大限度地减少无谓的字符串比较， 查询效率比哈希表高。但是内存空间消耗大，如果是完全m叉树， 节点数指数级增长。

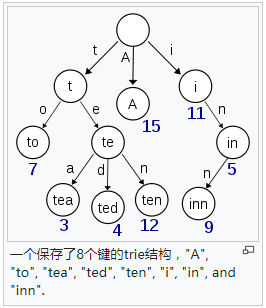


图1-1 trie树结构

Aho-Corasick 自动机，该算法在1975年产生于贝尔实验室，是著名的多模匹配算法之一。多模匹配一个常见的例子就是，给出n个单词，再给出一段包含m个字符的文章，从中找出有多少个单词在文章里出现过。构造AC自动机分为3步：构造一棵Trie树，构造失败指针和模式匹配过程，此算法有两个特点，一个是扫描文本时完全不需要回溯，另一个是时间复杂度为O(n)，时间复杂度与关键字的数目和长度无关。

**2、系统设计实现**

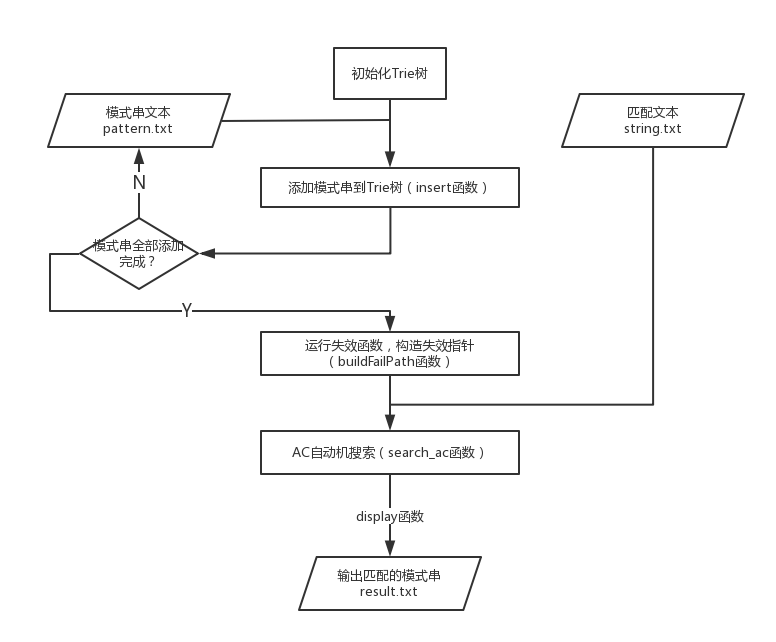
针对普通Trie树有很多优化，例如Radix Tree、二叉Trie树，我们使用三叉Trie树这种数据结构并构造成AC自动机完成实验。高性能字符串匹配系统的数据流程如图2-1所示:

图2-1 高性能字符串匹配系统

**2.1 三叉Trie树结构**

三叉trie树中每个节点包括三个指针，左兄弟指针、右兄弟指针、子节点指针，其中左兄弟、右兄弟与当前节点属同一层，左兄弟的左兄弟、右兄弟也属同一层，所以一个节点的左兄弟、右兄弟可以有很多，在匹配过程中需要在同一层中查找：

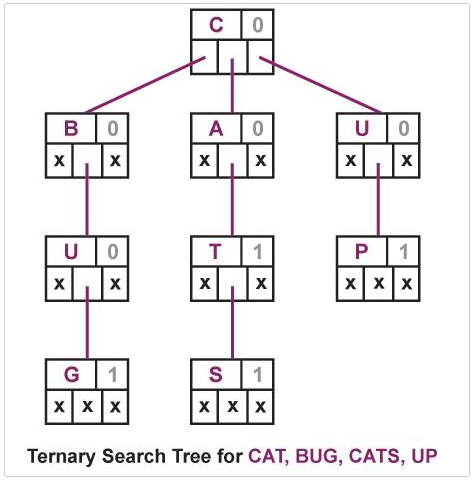


图2-2 三叉trie树结构

三叉Trie树数据结构定义：

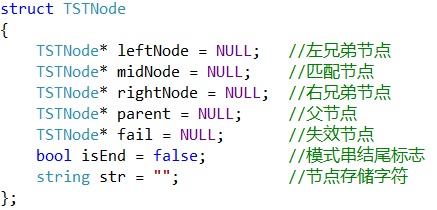


图2-3 三叉trie树数据结构定义

在匹配过程中，

（i）若当前节点的字符值等于待匹配字符，读取下一个带匹配字符，并当前节点的子节点中继续匹配；

（ii）若当前节点的字符值小于待匹配字符，在当前节点的左兄弟中继续寻找；

（iii）若当前节点的字符值大于待匹配字符，在当前节点的右兄弟中继续寻找。

**2.2 构造三叉Trie树（insert函数）**

因为用于构造三叉Trie树的pattern文本是gb2312编码，且其中包含单字节字符（半角符号，数字，英文）和双字节字符（全角字符，汉字），所以我们需要根据每个字节判断存储的字符长度，代码如下：

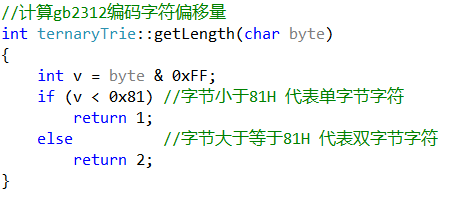


图2-4 计算GB2312编码字符偏移量

获取到字符之后，就可以构建三叉Trie树，为了方便后面构造失效函数，我们在树的根部插入root节点，从root的子节点开始，插入模式串文本pattern.txt中的字符，插入过程为：

（i）当前节点为空，建立新节点，保存模式串字符，记录父节点位置，读取下一个模式串字符，若到模式串末尾，标记当前节点为结束节点；

（ii）模式串字符值小于当前节点字符值，在当前节点的左兄弟节点中寻找插入位置；

（iii）模式串字符值大于当前节点字符值，在当前节点的右兄弟节点中寻找插入位置；

（iv）模式串字符值等于当前节点字符值，在当前节点的子节点中寻找插入位置。

**2.3 失效函数（buildFailPath函数）**

类似于单模匹配的KMP算法，失效函数给出了当匹配失败发生的时候，该继续去哪里匹配的位置。失效函数算法及实现的流程如下：

（i）令第一层的节点的失效节点都为root根节点；

（ii）从第二层开始，对于每一个节点，先找父节点的失效节点，若该失效节点的子节点与当前节点字符值相等，则当前节点的失效节点设为这个子节点；

（iii）若该失效节点的子节点与当前节点字符值不相等，则在该失效节点的失效节点的子节点中寻找，直到回到root根节点为止，则失效指针指向root

如图2-5中虚线所示即为简单Trie树构造的AC自动机的失效指针：

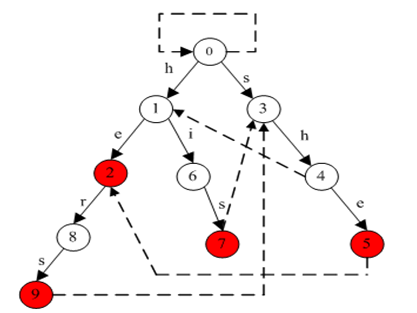


图2-5 普通trie构建的AC自动机

由于三叉Trie树的特殊结构，我们在构造失效函数时使用了两个队列，一个队列q1保存同一层的节点，另一个队列q2保存子节点，每次从q1中取出节点，计算失效指针位置，若q1为空，即这一层计算结束，将q2中的节点取出存到q1中继续计算。

**2.4 AC搜索函数（search\_ac函数）**

通过失效函数，每个节点的失效节点就找到了，接下来可以在AC自动机中开始搜索，对于待处理文本string.txt中的每一行的每个字符，在AC自动机中处理的流程如下：

（i）在当前节点及其左右兄弟节点中寻找匹配的节点，若找到匹配节点且为结束节点，执行输出函数，若不为结束节点，分配指针从该节点沿失效指针方向走到root节点，在途经结束节点时执行输出函数，两种情况结束后，读取行中下一个字符继续匹配；

（ii）找不到匹配节点，若当前节点为root根节点，读取行中下一个字符继续匹配，若不为root节点，从失效节点处开始继续匹配

（iii）若string.txt中一行处理完后，当前节点不为root根节点，还需沿该节点失效指针方向走到root节点，途经结束节点时执行输出函数。

对于图2-5的自动机而言，如主串为：ushers，则在状态5、2、9等状态时发生模式匹配，匹配的模式串有she、he、hers：

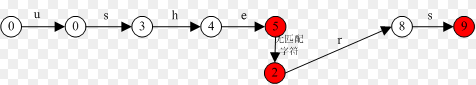


图2-6 AC自动机状态转移

**2.5 输出函数（display函数）**

当匹配到结束节点时，需要通过输出函数将匹配的模式串输出，在函数中，我们利用堆栈保存从当前节点开始回溯到root根节点所经过的父节点，堆栈中各父节点的字符即构成了模式串。得到模式串后，通过文件指针将其输出到result.txt文件中。

**3、实验结果及结论**

（i）程序运行时间：7.6min（Windows下测得）；

（ii）占用内存：130MB（VS 2017测得）；

（iii）CPU占用：50%（平均），67%（峰值）（VS 2017测得）；

（iv）建树时间：2.4s；

（v）树节点个数：2021721个（每个节点存储一个单字节或双字节字符）；

（vi）树高：122；

（vii）result文本匹配模式串条数：213088063条（匹配即输出到文本中）；

（viii）result.txt文本大小3.31GB

群里有同学用python调用pyahocorasick库，测得条目总数是214393004条，但我们在对小文本测试时发现，调用pyahocorasick库产生的结果中，有些模式串并未在文本中出现，这应该是匹配条目存在差异的原因所在。

实验完成后，我们可以看出多模匹配算法或AC自动机算法在处理大文本字符串匹配时的高效性，也更加深刻的理解了这种算法在大数据处理时的优势所在。

**4、优化、不足及遇到的问题**

**优化：**

（i）IO优化，C++流处理改为C语言文件流并设立缓存区，每次读入一行处理，程序运行时间明显缩短，前者20min，后者10min以内；

（ii）函数封装，对一些经常调用的函数进行封装，让代码可读性更强

**不足：**

截止提交代码和报告的前一天，我们才完成了第二版程序即用三叉Trie树实现AC自动机（第一版程序我们用普通Trie树实现AC自动机），所以代码中仍存在一些可以继续优化的地方，例如：（i）代码中使用了C++和C代码，所以同时存在string和char，代码可读性降低；（ii）在类型转换、字符比较、参数传递时设计可以更加简单有效。

**遇到的问题：**

（i）调试阶段我们发现构造的三叉树并不平衡，是由于用string存储中文字符在做比较时仍大于字母或数字，所以我们将待比较的两个字符都转成short，由于中文字符第一个字节大于80H，所以中文字符会转成负数，这样在构造的三叉Trie树中，双字节字符在树的左边，单字节字符在树的右边；

（ii）我们通过与python脚本产生的result.txt文本比对，发现有些模式串没有输出，通过询问同学得知，在匹配到节点时无论其是否为结束节点都需要向root根节点走一圈，这样才不会漏查。

**5、收获**

虽然是课程的第一个实验，但是我们体会到了不小的难度，也意识到自己在很多方面的欠缺，特别是编程能力较弱以及没有良好的编程习惯，让我们在整个过程中走了不少弯路，希望接下来的实验我们可以继续坚持下来，在问题中成长。