计算机编程实验-Lab 1

实验报告

12组：王韬、王丹阳

* 实验目标

进行高性能串的匹配任务，即在给定的主串中查找模式串的一个或全部出现。

已知给定的大规模中文文本串，数据量约为800M，和容量约为130万个词的模式串，现需以GB2312的编码格式从大文本串中查找出所有模式串出现的位置和次数。

由于大文本串规模较大，一次性内存无法容下全部主串，因此会涉及到分批读取。另外，模式串的规模也比较庞大，所以需要构造一个高效的数据结构进行处理。

* 系统架构设计

本组采用了Trie树与Ac自动机多模匹配算法相结合的形式进行大文本字符串的匹配任务。下面，报告将逐一介绍架构的各个部分。

1、Trie树（字典树）

本组采用了简单树的树形数据结构，该树有一个固定的关键码集合，并且会对结点进行分层标记，是一种基于关键码分解的数据结构，它的查找效率只与关键词长度有关，而与词表长度无关，因此可以最大限度地减少无谓的字符串比较，查询效率要高于哈希表。并且，Trie树对于插入索引的合并速度也比较快。（这一段，我不太理解……）

相应地，本组采用了Trie查找，即：在Trie树上从根结点开始进行检索，当取得要查找关键词的第一个字符后，根据该字符选择对应的子树并转到该子树继续进行检索。在某个结点处相应的子树上，将取得要查找关键词的第二个字符，并进一步选择下一个对应的子树进行检索。当关键词的所有字符都已被取出后，则将读取附在该结点上的信息，即完成查找。

本组也利用了Trie树的插入，即：首先根据插入纪录的关键码找到需要插入的结点位置（这个是对的），如果该结点是叶结点，那么就将为其分裂出两个子结点，分别存储这个纪录和以前的那个纪录；如果是内部结点，则在那个分支上应该是空的，所以直接为该分支建立一个新的叶结点。（这段是不对的，插入的逻辑是这样的，每一个节点都有一个hash表，是pattern的一个字符：此字符结点的指针结构的，所以插入的时候，就是查找这个字符，找不到的时候，就新建一个结点，同时在找不到的地方的hash表中加一个pattern的一个字符：此字符结点的指针的对，这样子插入的，叶结点的区别就是这个结点的那个hash表是空的，因为叶节点没有子节点了）

总之，本组所采用的Trie树结构的核心思想是：以空间换时间，利用字符串的公共前缀来降低查询时间的开销以达到提高效率的目的。不过，Trie树对于内存空间的消耗是比较大的。

2、Ac自动机多模匹配算法

本组采用了Ac自动机多模匹配算法，将字符比较转化成为了状态转移。借助该算法，可以在扫描文本时避免回溯，并且时间复杂度与关键字的数目和长度无关。

在预处理阶段，该算法建立了三个函数：转移函数、failure失效函数、output输出函数，由此构造了一个树型有限自动机。转移函数把一个由状态和输入字符组成的二元组映射成另一个状态或者一条失效消息。失效函数把一个状态映射成另一个状态。当转移函数报告失效时，失效函数就会被询问。输出状态表示已经有一组关键字被发现，其通过把一组关键字集 (可能是空集)和每个状态相联系的方法，使得这种输出状态的概念形式化。

在搜索查找阶段，当文本扫描开始时，初始状态置为状态机的当前状态，而输入文本的首字符作为当前输入字符。然后，树型有限自动机通过对每个文本串的字符都做一次操作循环的方式来处理文本。

该算法通过这三个函数的交叉使用扫描文本，定位出关键字在文本中的所有出现位置。

本组的树型有限自动机包含一组状态，每个状态用一个数字代表，状态机读入文本串y中的字符，然后通过产生状态转移或者偶尔发送输出的方式来处理文本。树型有限自动机的行为通过三个函数来指示:转向函数g、失效函数f、输出函数output。

* 系统实现过程

本组的实现过程主要可以分为5个部分：

1、建树

赋予模式树形的数据结构，建立Trie树。

本组建立一个Trie树类，将一些insert插入、遍历生成fail指针、查找主串的方法、以及遍历查找均定义在这个类中；并且，建立了一个结点类，由于本组并没有对叶子节点和分支节点进行细分，所以每个节点的结构都是一样的，即每个节点均包含一个output、一个fail指针（失效函数的指针）、一个形式为map<字符串，节点指针>的map（该map中储存了转移函数）。（这个map是hashmap）

2、读取模式串文件并构造AC自动机

这个过程是这样的，每读一个pattern，就执行一次insert，这个insert的逻辑是这样的，读取此pattern的每个字符，在树上查找这个字符，如果找得到，就接着往下找，直到这个pattern查完，在最后一个节点上加output；如果没找到，那么就从这个没找到pattern字符的结点开始，将pattern后面的字符都插进去，并且在最后一个节点上加output。（例子：比如her，在树上找到了h->e->r,那就直接把r上的output变成her；如果只找到了h，那就从h这个结点开始，插入->e->r，把r的output变成her）

所有的pattern执行insert完毕后，执行ergodicFail，来构建失效函数。失效函数是进行一个层序遍历，逐层找每个结点的失效函数。找的过程是这样的，对当前结点C，找这个结点父节点A的失效函数指向的结点B，如果B的子结点D和C一样，那么D就是C的失效函数指向的结点；如果B中找不到这样的D，那么就接着找B的失效函数指向的结点E，执行同样的操作，直到找到C的失效函数指向的结点。如果没找到，那么C的失效函数指向的结点，就是根结点。

每找到一个失效函数的时候，就把此结点和其失效函数指向的结点的output合并，经过这些步骤，失效函数和output就弄完了

3、读取字符串文件进行匹配

程序的主函数包含了读写文件、调用方法等语句。将输出函数的结果打印，并写入到.txt文档中。还运用偏移量获取了所查找字符的位置信息。

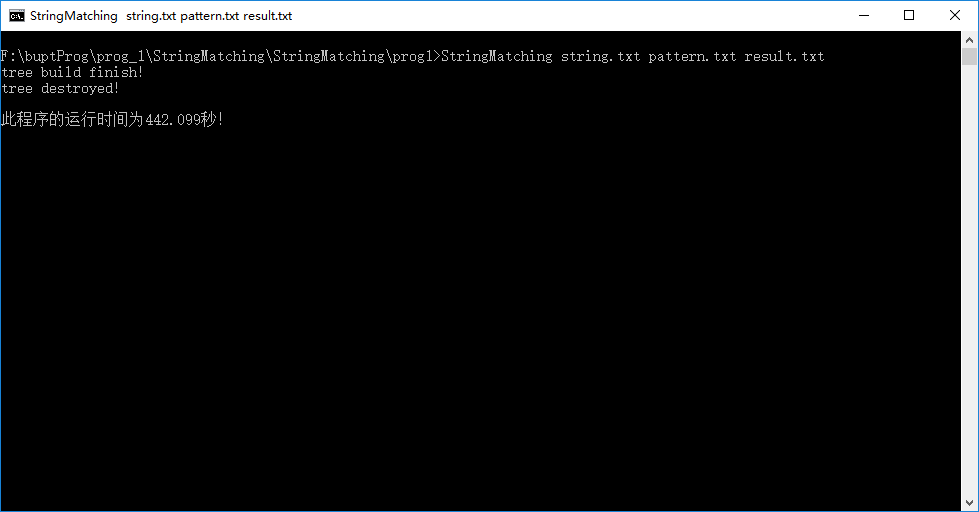
另外，由于编码格式为GB2312，所以本组在查找的过程中，是根据检测高八位的方法来判断中文字段，例如：高八位大于0x80的就是汉字或汉字字符。

4、释放结点

建树的时候每个结点都分配了内存，这个过程就是把每个结点的内存都释放掉，把这个树销毁。

* 实验结果

截止至目前，本组已经通过转移函数实现了建树，并添加了自定义的模式字符串。经测试，可正常运行，但是还并未进行进一步的优化。下图是打印出的模式串、位置行、位置列的结果。



本组已完成查询整个大规模的文本字符串，经过抽查，结果正确，根据程序的运行结果显示，运行时间较长，生成的结果被设置为写入到.txt文件中，本组将其命名为“result”，该文件所占内存较大，大概有3G左右。（结果上，和正确结果数量级相同，但是正确的214393004，我的220310472……不知道这是不是差不多）

本组所占用用的内存大概为600M，总查询时间为7分钟左右。

* 结论

实验的过程带给本组的总体感受是：痛并快乐着。在找不到问题根源时非常痛苦，可是当发现了改进的方法后，却会欣喜万分，也同时积累了很多小经验。

例如：本组的第一版本的代码的输出部分是主要采用C++编写的，耗时非常久，当改为用C的指针来编写输出后，本组发现程序的运行效率瞬间提升了很多，使得所用时间大幅度缩小。（之前的问题，乱用C++的stl，导致效率低下，在合适的地方换成指针，能够大幅提高效率不）

另外，本组发现，栈的数据结构在速度上是优于队列的。

（这条是事实，但是这次试验没用过，所以删了吧……）

不过，由于目前采用逐行读取的读写文件形式，导致大部分程序的运行时间是由此部分占用的，今后可适当尝试进行时空性能的优化。（这样子的，最开始用的C++读写文件的方式，后来发现读文件改成C模式的方法差别不大，但是写文件用C的模式会速度大幅提升，C++模式是cout cin等等，C是用File指针用fprintf，fscanf等等。这部分还可以优化，就是用缓冲区，读写设置缓冲区，数量到了一定程度再写到磁盘里，这样能更快）

* 数据流程图