昵称：罗松超
园龄：6年10个月
粉丝：28
关注：5
+加关注

< 2019年7月 >

日	一	二	三	四	五	六
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

搜索

找找看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

随笔分类

Android(1)
C++(28)
Cocos2dx(16)
Database(8)
Github(9)
Java(8)
JQuery(3)
JSP(12)
Linux编程(15)
Linux工具&配置(27)
Log4j(5)
Lua(4)
Python(11)
UC/OS学习(9)
VIM(11)
Windows(2)
XML(2)
操作系统(1)
设计模式(7)
算法&数据结构(26)

随笔档案

2015年3月 (13)
2014年11月 (5)
2014年9月 (1)
2014年8月 (1)
2014年7月 (1)
2014年6月 (10)
2014年5月 (13)
2014年4月 (20)
2014年3月 (2)
2014年2月 (8)
2013年12月 (7)
2013年11月 (14)
2013年10月 (19)
2013年9月 (25)
2013年8月 (17)
2013年7月 (31)
2013年6月 (8)
2013年5月 (11)
2013年4月 (16)

最新评论

1. Re:vim选中字符复制/剪切/粘贴
@ctqlxNERDTREE...
--greatofdream

后缀树（Suffix Tree）

问题描述：

后缀树（Suffix Tree）

参考资料：

http://www.cppblog.com/yuyang7/archive/2009/03/29/78252.html

http://blog.csdn.net/v_july_v/article/details/6897097

简介

后缀树是一种PAT树，它描述了给定字符串的所有后缀，许多重要的字符串操作都能够在后缀树上快速地实现。

定义

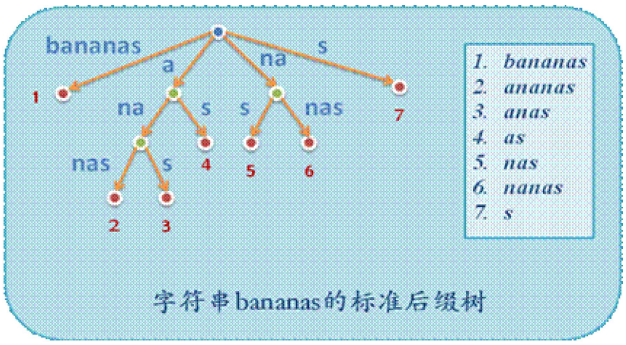
一个长度为n的字符串S，它的后缀树定义为一棵满足如下条件的树：

1. 从根到树叶的路径与**S的后缀一一对应。即每条路径惟一代表了S的一个后缀；**

2. 每条边都代表一个非空的字符串；

3. 所有内部节点（根节点除外）都有至少两个子节点。

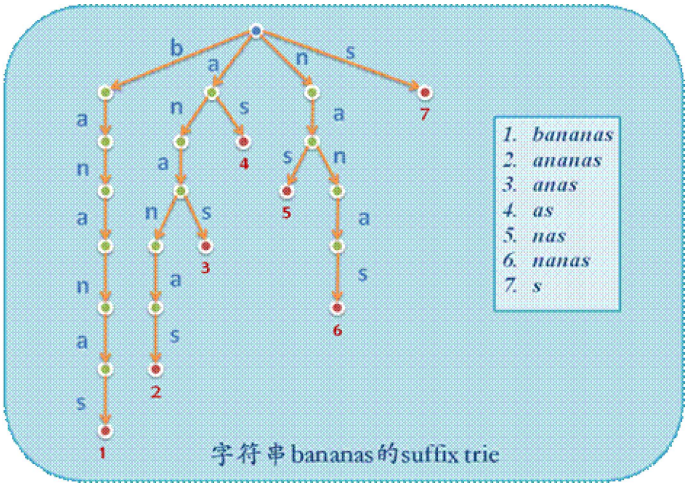
由于并非所有的字符串都存在这样的树，因此S通常使用一个终止符号进行填充（通常使用$）。



优点

1. 匹配快。对于长度为*m*的模式串，只需花费至多*O(m)*的时间进行匹配。

2. 空间省。Suffix tree的空间耗费要低于Suffix trie，因为Suffix tree除根节点外不允许其内部节点只含单个子节点，因此它是Suffix trie的压缩表示。



出来的，谢谢。

--ctqlx

3. Re:git删除远程分支和本地分支

@wlzxdm不好意思，搞错了，没合并的分支，-d删不掉...

--1129778586

4. Re:git删除远程分支和本地分支

@1129778586-d没错。-D是强制删除，你为什么要强制删除？...

--wlzxdm

5. Re:Linux进程栈和线程栈

查线程栈默认大小（8KB），是 8MB

--享受工作，享受生活

阅读排行榜

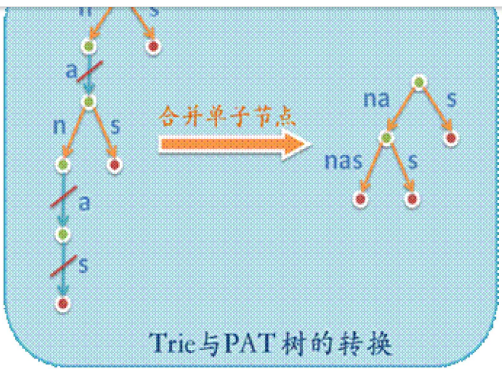
1. git删除远程分支和本地分支(195813)
2. vim选中字符复制/剪切/粘贴(125770)
3. java对象数组(26759)
4. 【Cocos2dx 3.3 Lua】定时器事件(15758)
5. C# Socket连接超时设置(11237)

评论排行榜

1. Linux进程栈和线程栈(4)
2. vim选中字符复制/剪切/粘贴(3)
3. ZedGrap控件绘制图表曲线(2)
4. git删除远程分支和本地分支(2)
5. 队列实现二叉树层序遍历(1)

推荐排行榜

1. git删除远程分支和本地分支(4)
2. trie树（前缀树）(3)
3. vim选中字符复制/剪切/粘贴(3)
4. C++同步串口通信(2)
5. java Mail发送邮件(2)



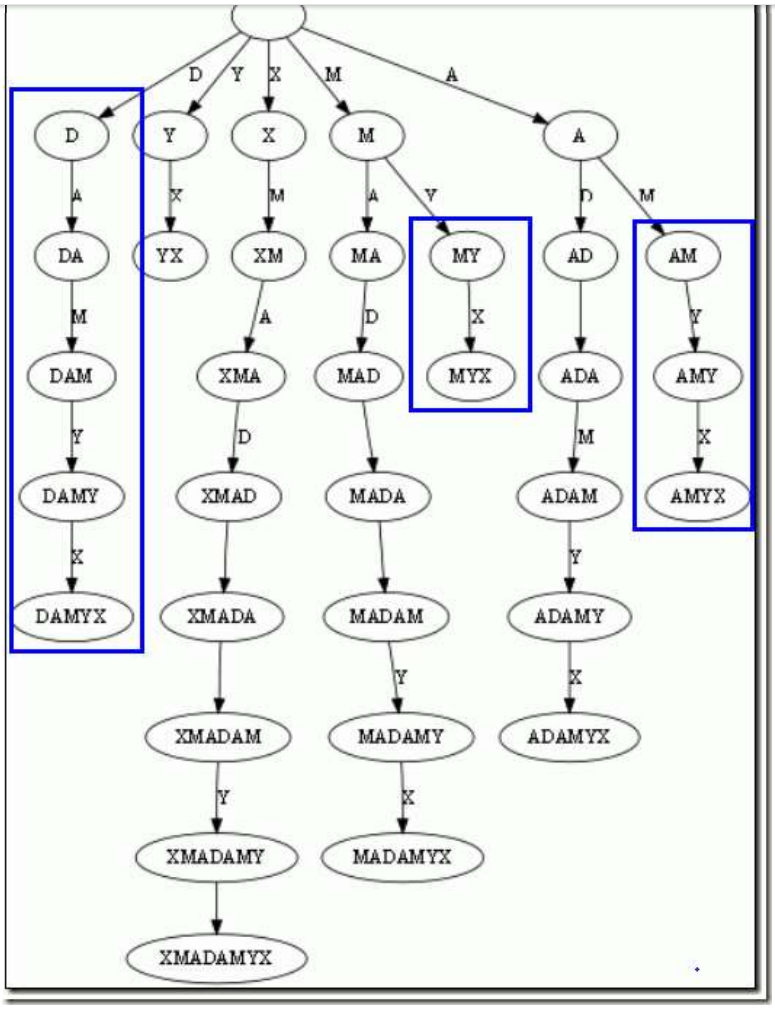
后缀树的生成，Suffix trie ------>> Suffix tree

后缀，顾名思义，甚至通俗点来说，就是所谓后缀就是后面尾巴的意思。比如说给定一长度为n的字符串S=S1S2..Si..Sn，和整数i，1 <= i <= n，子串SiSi+1...Sn便都是字符串S的后缀。

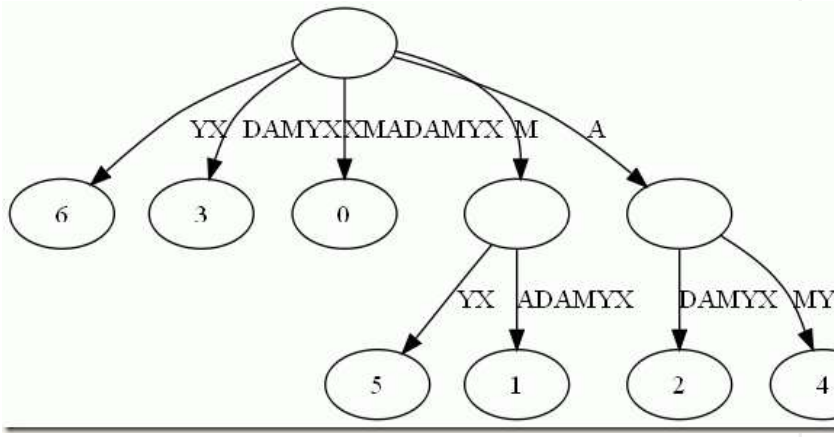
以字符串S=XMADAMYX为例，它的长度为8，所以S[1..8], S[2..8], ... , S[8..8]都算S的后缀，我们一般还把空字串也算成后缀。这样，我们一共有如下后缀。对于后缀S[i..n]，我们说这项后缀起始于i。

S[1..8], XMADAMYX， 也就是字符串本身，起始位置为1
S[2..8], MADAMYX，起始位置为2
S[3..8], ADAMYX，起始位置为3
S[4..8], DAMYX，起始位置为4
S[5..8], AMYX，起始位置为5
S[6..8], MYX，起始位置为6
S[7..8], YX，起始位置为7
S[8..8], X，起始位置为8
空字串，记为$。

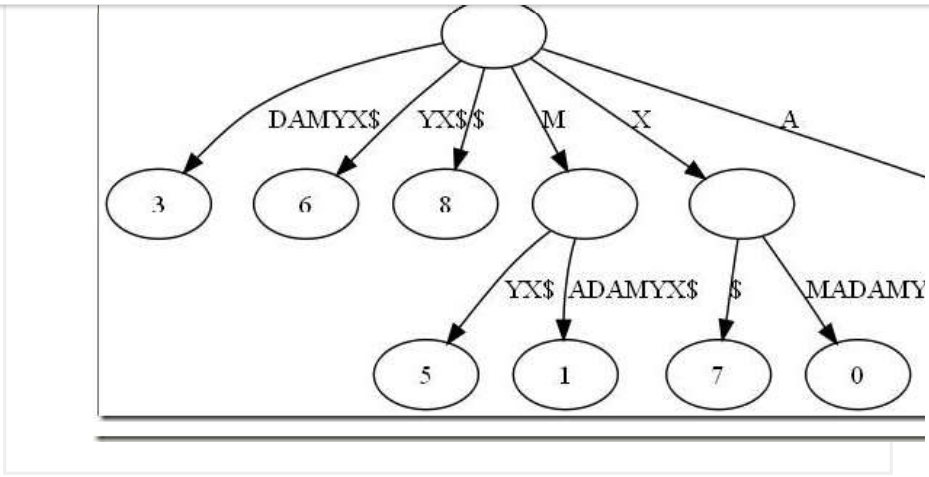
而后缀树，就是包含一则字符串所有后缀的压缩Trie。把上面的后缀加入Trie后，我们得到下面的结构：



仔细观察上图，我们可以看到不少值得压缩的地方。比如蓝框标注的分支都是独苗，没有必要用单独的节点同边表示。如果我们允许任意一条边里包含多个字 母，就可以把这种没有分叉的路径压缩到一条边。另外每条边已经包含了足够的后缀信息，我们就不用再给节点标注字符串信息了。我们只需要在叶节点上标注上每项后缀的起始位置。于是我们得到下图：



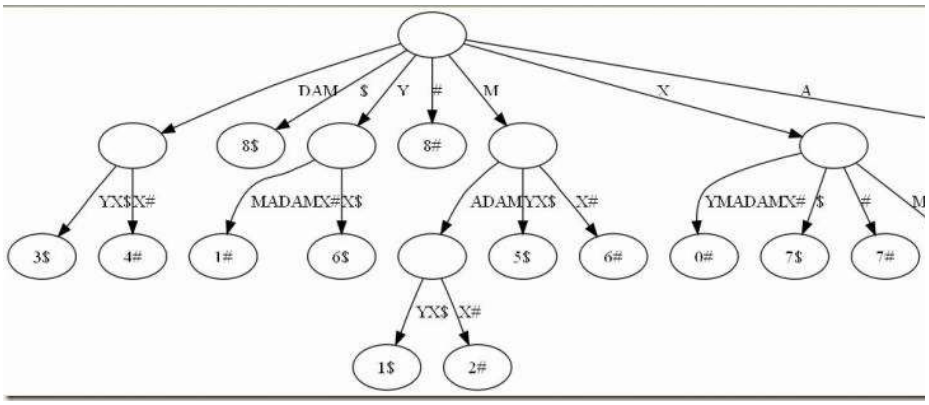
这样的结构丢失了某些后缀。比如后缀X在上图中消失了，因为它正好是字符串XMADAMYX的前缀。为了避免这种情况，我们也规定每项后缀不能是其它后缀的前缀。要解决这个问题其实挺简单，在待处理的子串后加一个空字串就行了。例如我们处理XMADAMYX前，先把XMADAMYX变为 XMADAMYX$，于是就得到suffix tree-



2.2、后缀树与回文问题的关联

那后缀树同最长回文有什么关系呢？我们得先知道两个简单概念：

1. 最低共有祖先，LCA（Lowest Common Ancestor），也就是任意两节点（多个也行）最长的共有前缀。比如下图中，节点7同节点10的共同祖先是节点1与节点，但最低共同祖先是5。 查找LCA的算法是O(1)的复杂度，这年头少见。代价是需要对后缀树做复杂度为O(n)的预处理。
2. 广义后缀树(Generalized Suffix Tree)。传统的后缀树处理一坨单词的所有后缀。广义后缀树存储任意多个单词的所有后缀。例如下图是单词XMADAMYX与XYMADAMX的广义后缀 树。注意我们需要区分不同单词的后缀，所以叶节点用不同的特殊符号与后缀位置配对。



2.3、最长回文问题的解决

有了上面的概念，本文引言中提出的查找最长回文问题就相对简单了。咱们来回顾下引言中提出的回文问题的具体描述：找出给定字符串里的最长回文。例如输入XMADAMYX，则输出MADAM。

思维的突破点在于考察回文的半径，而不是回文本身。所谓半径，就是回文对折后的字串。比如回文MADAM 的半径为MAD，半径长度为3，半径的中心是字母D。显然，最长回文必有最长半径，且两条半径相等。还是以MADAM为例，以D为中心往左，我们得到半径 DAM；以D为中心向右，我们得到半径DAM。二者肯定相等。因为MADAM已经是单词XMADAMYX里的最长回文，我们可以肯定从D往左数的字串DAMX与从D往右数的子串DAMYX共享最长前缀DAM。而这，正是解决回文问题的关键。现在我们有后缀树，怎么把从D向左数的字串DAMX变成后缀 呢？

到这个地步，答案应该明显：把单词XMADAMYX翻转（XMADAMYX=>XYMADAMX，DAMX就变成后缀了）就行了。于是我们把寻找回文的问题转换成了寻找两坨后缀的LCA的问题。当然，我们还需要知道到底查询那些后缀间的LCA。很简单，给定字符串S，如果最长回文的中心在i，那从位置i向右数的后缀刚好是S(i)，而向左数的字符串刚好是翻转S后得到的字符串S'的后缀S'(n-i+1)。这里的n是字符串S的长度。

可能上面的阐述还不够直观，我再细细说明下：

1、首先，还记得本第二部分开头关于后缀树的定义么：“先说说后缀的定义，顾名思义，甚至通俗点来说，就是所谓后缀就是后面尾巴的意思。比如说给定一长度为n的字符串S=S1S2..Si..Sn，和整数i，1 <= i <= n，子串SiSi+1...Sn便都是字符串S的后缀。”

以字符串S=XMADAMYX为例，它的长度为8，所以S[1..8], S[2..8], ... , S[8..8]都算S的后缀，我们一般还把空字串也算成后缀。这样，我们一共有如下后缀。对于后缀S[i..n]，我们说这项后缀起始于i。

S[1..8], XMADAMYX， 也就是字符串本身，起始位置为1
S[2..8], MADAMYX，起始位置为2
S[3..8], ADAMYX，起始位置为3

S[6..8], MYX，起始位置为6
S[7..8], YX，起始位置为7
S[8..8], X，起始位置为8
空字串，记为$。

2、对单词XMADAMYX而言，回文中心为D，那么D向右的后缀DAMYX假设是S(i)（当N=8，i从1开始计数，i=4时，便是S(4..8)）;而对于翻转后的单词XYMADAMX而言，回文中心D向右对应的后缀为DAMX，也就是S'(N-i+1)(（N=8，i=4，便是S'（5..8）） 。此刻已经可以得出，它们共享最长前缀，即LCA（DAMYX，DAMX）=DAM。有了这套直观解释，算法自然呼之欲出：

1. **预处理后缀树，使得查询LCA的复杂度为O(1)。这步的开销是O(N)，N是单词S的长度；**
2. **对单词的每一位置i(也就是从0到N-1)，获取LCA(S(i), S'(N-i+1)) 以及LCA(S(i+1), S'(n-i+1))。查找两次的原因是我们需要考虑奇数回文和偶数回文的情况。这步要考察每坨i，所以复杂度是O(N)；**
3. **找到最大的LCA，我们也就得到了回文的中心i以及回文的半径长度，自然也就得到了最长回文。总的复杂度O(n)。**

用上图做例子，i为4时，LCA(4$, 5#)为DAM，正好是最长半径。当然，这只是直观的叙述。
上面大致描述了后缀树的基本思路。要想写出实用代码，至少还得知道下面的知识：

- **创建后缀树的O(n)算法。此算法有很多种，无论Peter Weiner的73年年度最佳算法，还是Edward Mc Creight1976的改进算法，还是1995年E. Ukkonen大幅简化的算法（本文第4部分将重点阐述这种方法），还是Juha Kärkkäinen 和 Peter Sanders2003年进一步简化的线性算法，都是O（n）的时间复杂度。至于实际中具体选择哪一种算法，可依实际情况而定。**
- **实现后缀树用的数据结构。比如常用的子结点加兄弟节点列表，Directed 优化后缀树空间的办法。比如不存储子串，而存储读取子串必需的位置。以及Directed Acyclic Word Graph，常缩写为黑哥哥们挂在嘴边的DAWG。**

2.4、后缀树的应用

后缀树的用途，总结起来大概有如下几种

1. **查找字符串o是否在字符串S中。**
   方案：用S构造后缀树，按在trie中搜索字串的方法搜索o即可。
   原理：若o在S中，则o必然是S的某个后缀的前缀。
   例如S: leconte，查找o: con是否在S中,则o(con)必然是S(leconte)的后缀之一conte的前缀.有了这个前提，采用trie搜索的方法就不难理解了。
2. **指定字符串T在字符串S中的重复次数。**
   方案：用S+'$'构造后缀树，搜索T节点下的叶节点数目即为重复次数
   原理：如果T在S中重复了两次，则S应有两个后缀以T为前缀，重复次数就自然统计出来了。
3. **字符串S中的最长重复子串**
   方案：原理同2，具体做法就是找到最深的非叶节点。
   这个深是指从root所经历过的字符个数，最深非叶节点所经历的字符串起来就是最长重复子串。
   为什么要非叶节点呢?因为既然是要重复，当然叶节点个数要>=2。
4. **两个字符串S1，S2的最长公共部分**
   方案：将S1#S2$作为字符串压入后缀树，找到最深的非叶节点，且该节点的叶节点既有#也有$(无#)。

后缀树实现：

http://www.pcw8510.com/?p=1296

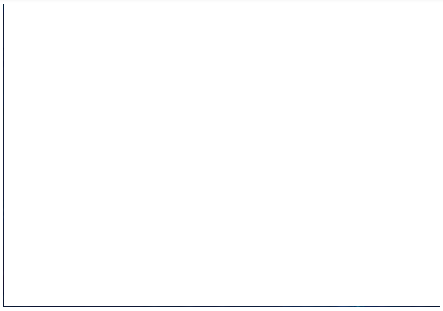
分类: 算法&数据结构

罗松超
关注 - 5
粉丝 - 28
+加关注

0 0

« 上一篇：字符串旋转子串
» 下一篇：数据结构之并查集

posted @ 2013-08-09 10:07 罗松超 阅读(5103) 评论(0) 编辑 收藏



相关博文：
· 从Trie树（字典树）谈到后缀树
· 从Trie树（字典树）谈到后缀树（10.28修订）
· 从Trie树（字典树）谈到后缀树（10.28修订） - Rollen Holt - 博客园
· trie树和后缀树的应用
· [算法]后缀树suffix tree