**什么是后缀树？**

**Suffix Trie ：**又称后缀Trie或后缀树。它与Trie树的最大不同在于，后缀Trie的字符串集合是由指定字符串的后缀子串构成的。比如、完整字符串"minimize"的后缀子串组成的集合S分别如下：

         s1=minimize

         s2=inimize

         s3=nimize

         s4=imize

         s5=mize

         s6=ize

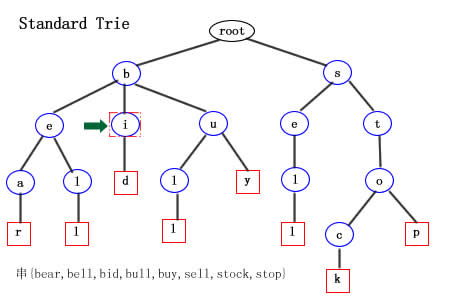
         s7=ze

         s8=e

      然后把这些子串的公共前缀作为内部结点构成一棵"minimize"的后缀树，如图所示，其中上图是Trie树的字符表示，下图是压缩表示(详细见《[Trie树](http://hxraid.iteye.com/blog/618962) 》)。可见Suffic Trie是一种很适合操作字符串子串的数据结构。 它和PAT tree在这一点上类似。

**Tire数：**

**标准 Trie树的结构**： 所有含有公共前缀的字符串将挂在树中同一个结点下。实际上trie简明的存储了存在于串集合中的所有公共前缀。 假如有这样一个字符串集合X{bear,bell,bid,bull,buy,sell,stock,stop}。它的标准Trie树如下图：



      上图（蓝色圆形结点为内部结点，红色方形结点为外部结点），我们可以很清楚的看到字符串集合X构造的Trie树结构。其中从根结点到红色方框叶子节点所经历的所有字符组成的串就是字符串集合X中的一个串。

      注意这里有一个问题： 如果X集合中有一个串是另一个串的前缀呢？ 比如，X集合中加入串bi。那么上图的Trie树在绿色箭头所指的内部结点i 就应该也标记成红色方形结点。这样话，一棵树的枝干上将出现两个连续的叶子结点(这是不合常理的)。

      也就是说字符串集合X中不存在一个串是另外一个串的前缀 。如何满足这个要求呢？我们可以在X中的每个串后面加入一个特殊字符$(这个字符将不会出现在字母表中)。这样，集合X{bear$、bell$、.... bi$、bid$}一定会满足这个要求。

      总结：一个存储长度为n，来自大小为d的字母表中s个串的集合X的标准trie具有性质如下：

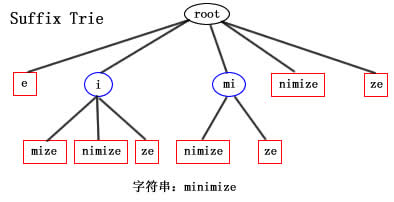
      (1) 树中每个内部结点至多有d个子结点。

      (2) 树有s个外部结点。

      (3) 树的高度等于X中最长串的长度。

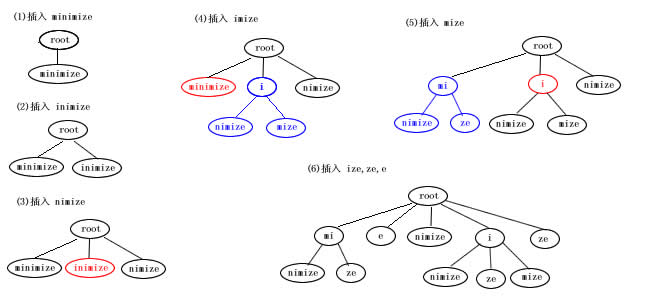
      (4) 树中的结点数为O(n)。

**后缀树如下：**



**Suffix Trie的创建**

      标准Tire树的每一个内部结点只有一个字符，也就是说公共前缀每一次只找一个。而Suffix Trie的公共前缀可以是多个字符，因此在创建Suffix Trie的时候，每插入一个后缀子串，就可能对内部结点造成一次分类。下面我们我们看一种后缀树构造算法。以"minimize"为例：



      当插入子串时，发现叶子结点中的关键字与子串有公共前缀，则需要将该叶子结点分裂。如上图第3到4步。否则，重新创建一个叶子结点来存放后缀，如上图第1到2步

**Suffix Trie的子串查询**

     如果在后缀树T中查找子串P，我们需要这样的过程：

     (1) 从根结点root出发，遍历所有的根的孩子结点：N1,N2,N3....

     (2) 如果所有孩子结点中的关键字的第一个字符都和P的第一个字符不匹配，则没有这个子串，查找结束。

     (3) 假如N3结点的关键字K3第一个字符与P的相同，则匹配K3和P。

          若 K3.length>=P.length  并且K3.subString(0,P.length-1)=P，则匹配成功，否则匹配失败。

          若 K3.length<=P.length  并且K3=P.subString(0, K3.length-1)，则将子串P1=P.subString(K3.length, P.length); 即取出P中排除K3之后的子串。然后P1以N3为根结点继续重复(1)~(3)的步骤。直到匹配完P1的所有字符，则匹配成功。否则匹配失败。

      查询效率：很显然，在上面的算法中。匹配成功正好比较了P.length次字符。而定位结点的孩子指针，和Trie情况类似，假如字母表数量为d。则查询效率为O(d\*m)，实际上，d是固定常数，如果使用Hash表直接定位，则d=1.

      因此，后缀树查询子串P的时间复杂度为O(m)，其中m为P的长度。

**Suffix Trie的应用**

      标准Trie树只适合前缀匹配和全字匹配，并不适合后缀和子串匹配。而后缀树在这方面则非常合适。

      另外后缀树也可以进行前缀匹配。 如果模式串P是字符串S的前缀的话，那么从根结点出发遍历后缀树，一定能够寻找到一条路径完全匹配完P。比如上图： 模式串P=“mini”，主串S="minimize"。P从根节点出发，首先匹配到结点mi，然后再匹配孩子结点nimize。直到P中所有的字符都找到为止。所以P是S的前缀。