# 后缀自动机SAM

定义：对给定字符串s的后缀自动机是一个最小化确定有限状态自动机(有限自动机的是什么参见ac自动机文章)，它能够接收字符串s的所有后缀。

## 构建：

<https://hihocoder.com/problemset/problem/1441>

hihocoder的这个文章比较容易理解，而且正确。

最好先要阅读这两个题的文章。

给定字符串s，比如aabbadb

在ac自动机文章里，知道了怎么构建dfa，那么构建后缀自动机，最朴素的办法，找到s所有后缀，分别从根开始构建。但这样显然这样时间复杂度，空间复杂度最坏是len^2 ,并且也无法保证有效连接pre指针。

需要一个时间空间on的算法来构造后缀自动机。叫做增量法：

后缀自动机构建的是一个在线过程，已知字符串s 前i个字符的后缀自动机，可以求出前i+1个字符的后缀自动机。如此递推就求出了整个字符串的后缀自动机。

以下是后缀自动机构造方法，读这个之前请仔细阅读hihocoder1441的基本原理。

<https://hihocoder.com/problemset/problem/1441>

后缀自动机每个节点3个必须属性:

**son[26]**：表示当前节点的26个叉dfa最基本的元素，具体请看dfa的指针实现和伪指针实现。

P：表示当前节点上一个节点指针，它的意义参加hihocoder1441的文章，除了根节点的p是NULL，其他节点的p一定存在。

NodeLen：表示到当前节点所代表最大的字符串长度。

首先肯定初始化root节点，len=0其他指针都是NULL，和尾节点last,last初始是root，之后它每次都等于新加入的节点node

接下来把字符串s每个字符按顺序加入sam,加入一个字符执行如下步骤：

1. 新加入了字符c,新建节点node，从last开始沿着p指针找(设当前访问的节点是now)，看now的第c个son指针存不存在。如果不存在就连接son[c]=node;如果一直执行这个操作直到now是NULL，意味着回到根节点的p指针。直接退出就好。若，访问到了某个节点，它的 son[c]存在，那就执行步骤2再退出这次程序。
2. 为了方便书写，设q=now->son[c]; 要分为2种情况：now.len+1==q.len,，说明两个节点紧挨着，就把node的p指针指向q,完事。若now.len+1<q.len执行步骤3
3. 新建节点nq,nq的所有son指针是q的所有son指针的拷贝。把now的son[c]指针由原先指向q改为指向nq.不仅如此，还应该沿着now的p指针迭代所，对于每个son[c]指向q的节点，都应该改为指向nq。

对于p指针的更改，设pp=q的p指针,q的p指针应当由pp改为nq,同时把nq的p指针指向pp; 最后把node的p指针指向nq

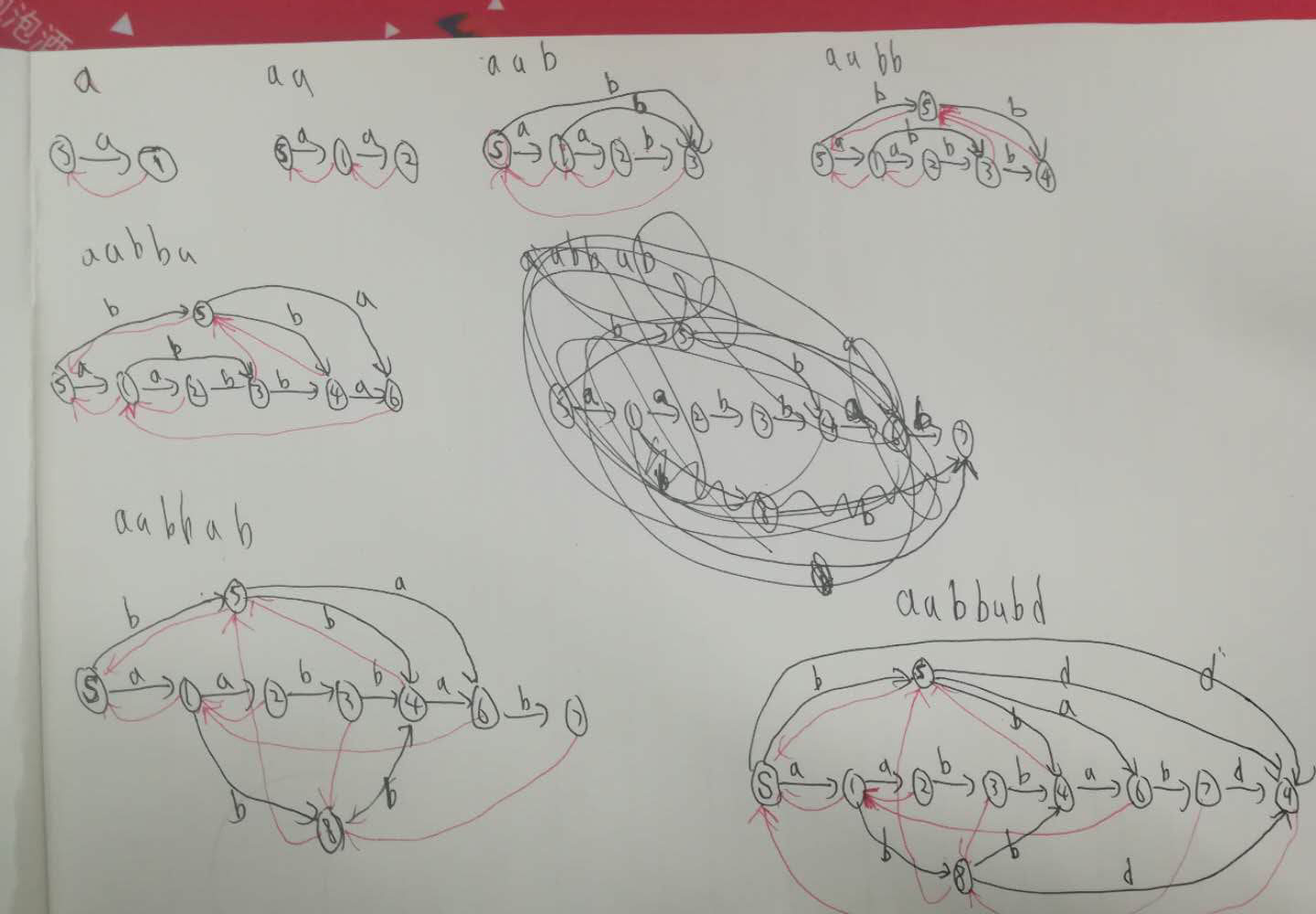
步骤完毕不要忘了更变last节点。

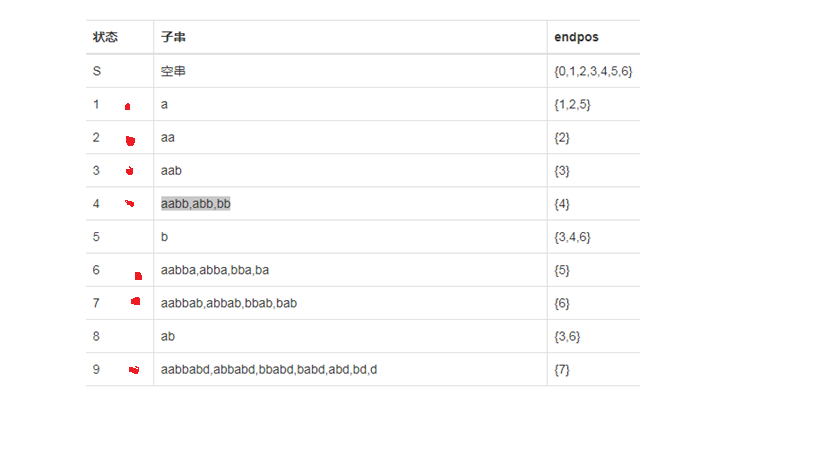
注意到，以上步骤每次最多建立2个节点。空间小于2\*n。的时间上不超过

len(s)次就一定能走到last节点。代码实现也很短（40行不到），指针实现比较理解。也可以用数组实现，开2倍len(s)的空间即可，并且实测数组的比指针快1倍，算法竞赛的后缀自动机,数组比指针灵活，所以建议用数组实现。

此外后缀自动机的时空是on的严谨证明比较麻烦，但只是想理解大致原因读完hihocoder1441就行了。

给出hihocoder样例的构建后缀自动机的步骤





总字符串为aabbabd用表示，红点为主点

## 性质：

**记：** 初始状态root节点叫初始点。由它能够到达其余所有状态。

**只看p指针的所构成的图:记:**

入读为0的节点(示例图里横着的那条链)，叫做主点，除了初始点之外，入度非0的点叫副点。由此可把后缀自动机节点分为3类：初始点(主一个)，主点，副点。主点和初始点构成主链

**记：**某个节点的前驱边是指向它的dfa的叉，root节点没有前驱边

* 后缀自动机是一张有向无环图，其中顶点是状态，而边代表了状态之间的转移。
* 除了root，每个节点p指针不为空，且只有一个。
* 把某个主链的节点看做终止点。从初始点严格沿着主链走到终止点所成的字符串看做s。那么，我们从初始点root经任意路径走到终止点所构成的字符串s1，必然是s的某一后缀,但是走所有路径构成的字符串集合不一定是s的全部后缀。
* 在符合上述诸条件的所有自动机中，后缀自动机有这最少的顶点数。（后缀自动机并不被要求拥有最少的边数）
* 对于某得点node，沿主链到node所构成字符串记为s。node->p的前驱边 和 node前驱边所表示字符相同，并且沿着node的p走直到root为止，也都相同，记:这一条路径上的的点的集合是ns。

从root开始到ns所包含的点的全部路径 所构成的字符串的集合。它们是s全部后缀(主链代表的字符串)。

* Sam每个节点都有一个endpos属性，它代表一个数字集合：【表示根到当前节点所表示的字符串的集合中，[每个字符串在中的全部位置]的末尾编号的集合】。末尾编号指最后一个字符的在总串中的位置。Endpos在构建sam的时候不需要用到，但某些题目中有用。

**副点的endpos是：**所有p指向自己的节点的endpos的并集

**主点的endpos是：**所有p指向自己的节点的endpos，以及自己本身所表示的字符的序号

* 对于sam某个节点p,和node->son[c]节点np，p表示的字符串集加上字母c是np字符串集的子集；经过son指针可以连到np的所有节点所表示的字符串集合并，就相当于是np字符串集每个字符串去掉一个c。

# 模型：

## 字符串子串个数

给定长度最多1e6的字符串，求子串的种类数。(种类数就是不同的字符串个数)

这是sam最基本的模板

我们知道sam的节点len，代表了以这个字母为后缀的最长的字符串的长度。只要再知道：以这个字母为后缀的最短的字符串的长度 就能知道从root到当前节点所表示的字符串集的元素个数了。怎么知道呢？由性质4，可以知道。求子串的种类数的方式是：遍历全部后缀自动机主点节点，主点意思是每次添加字符所建立的node节点，nq不算。

ans+=now->len - now->p->len。实际上构建的过程就可以顺便累加完毕

## 字符串中长度是k的子串出现最多的一个出现次数

设len(endpos)是某个点endpos集合的元素个数，根据性质6,可以知道一件事是副点len(endpos)是所有p指向自己的点之.。主点则是之和加一。

某个点len(endpos)就是：s中长度区间在[minlen,maxlen]的子串的出现次数。maxlen就是每个节点的len,minlen模型1说过了。设ans[i]代表长度是i的子串出现次数最多的次数，遍历每个节点，用len(endpos)所对应长度区间的值取最大即可。但是复杂度是len\*len.

怎么优化，显而易见ans[i]随i增大而递减，所以对于每个节点，只需每次更新ans[maxlen]。完了之后为了保证递增倒着扫一遍ans[i]=max(ans[i],ans[i+1])

这个转化需要思考一下。

还有就是怎么求len(endpos)，构建sam完毕后，在拓扑序上，往前累加就可以求得eqlen,由于需要拓扑排序，需要维护每个点的p指针的出度值。排拓扑序转到图论拓扑排序的文章。

## 求多个数字符串不同种类的子串的加和

首先思考对于一个单独的字符串s,如何求。

通过性质7来思考，对于某个节点node来说，设root到它自身表示的所有数字字符串加和是sum(now),那么对于每一个son能指向now的点x.sum都要加上sum(x) \* 10+c\*|substrings(x)| |substrings(x)|表示了节点x表示的字符串集元素个数

sum(now) = Σ{sum(x) \* 10 + c \* |substrings(x)| }

利用拓扑排序遍历全部节点可以求得字符串s不同种类的子串的加和。

求多个字符串，可以把它们连接成一个，中间用一个字符集里没有的特殊字符隔开，比如对于数字字符串，它们只有0-9，我就可以用’:’隔开字符串。而’:’恰好是’9’的阿斯克码下一个。原本求sum(now)的公式变成了：sum(x) \* 10+c\*|unsubstrings(x)| |unsubstrings(x)|代表了节点x表示的字符串集中，无’:’字符串的个数。求法是找到root不经过’:’边走到x节点的路径数目，有向无环图路径计数是拓扑排序的经典模板，这个虽然有条件但方法也一样只不过遇到’:’的边不走。同时求sum也是拓扑排序来求，拓扑排序需要维护每个点son指针的入度值。总体就是一遍拓扑排序求出unsubstrings 和sum。

## 最长公共子串

给定字符串S和T，求他们最长公共子串，也是经典模型，时间复杂度o(n+m)

方法是先构建S的后缀自动机，再构建关于T的数组U和L，U[i]代表以T[i]为结尾的最长公共子串在SAM哪个节点结束，L[i]表示以T[i]为结尾的最长公共子串长度。这里U和L下标从1开始

让T串在S的后缀自动机里匹配，初始now是根节点，i=0;代表匹配T[0],T的字符按照顺序逐个匹配。对于某个now和i,有如下情况：

son[now][T[i]]存在:就令U[i]= son[now][T[i]] 说明字符串连续L[i]=L[i-1]+1

son[now][T[i]]不存在：表示匹配失败，沿着now节点p指针向走，直到走到某个点son[now][T[i]]存在，U[i]= son[now][T[i]] L[i]=len[n] +1 len[now]就是sam当前节点的len属性。P指针充当了类似kmp的next数组。

走到root后son[now][T[i]]都不存在：把now重新初始化为0，L[i]=0,跳过这个字符T[i]直接匹配下一个

最后遍历L数组可以知道最长的长度和最长公共子串所在s2中的位置j。

想要得到最长的有多少个，就是找状态U[i]的eplen属性

## 后缀排序：

可以用来找出字典序最小的后缀，对于一个字符串s,构建后缀自动机，后缀自动机S到末尾节点的所有路径一定是s的后缀，但不意味着是所有后缀，它们是包含关系，正招很简单，就是从last节点开始，沿着p指针向上走直到走到根为止，标记走过的节点，然后从根dfs,根到标记节点路径一定表示一个后缀，且所有路径一定恰好是s无重复的全部后缀