## 回文自动机

### 简介

回文自动机PAM又是一种dfa(有限状态机),如果说后缀自动机解决后缀有关的问题(字串也可以看成后缀)，且对应了后缀树，那么回文自动机顾名思义解决了回文串的问题，且对应了回文树，相当于回文树加上fail指针

**LPS:**不包括自己本身的最长回文后缀的简称, 对于不回文的字符串，末尾的单个字符显然是LPS

### 基本属性和意义

回文自动机有2个根节点记为r0和r1,分别代表 [偶数长度的回文串] 和 [奇数长度回文串的字符串] 集合的初始状态。.

Struct Node{//这是节点属性

Son[26]//代表：字符集的26个分叉，是dfa基本属性，son[i]在PAM里代表在当前字符串左右两边各加一个字符，而非ac自动机那样只加一个字符。

Fail //代表：走到当前节点所表示的回文串的LPS对应的链子的末尾节点

len//代表：每个节点所表示的最长回文串的长度，特殊的，对于r1->len=-1

除了基本属性，不同题目需要额外维护其他属性。

Sum：回文串出现次数

}

Last //代表上一次加入字符所新增节点的编号

r0代表偶数树的根,r1代表奇数树的根:

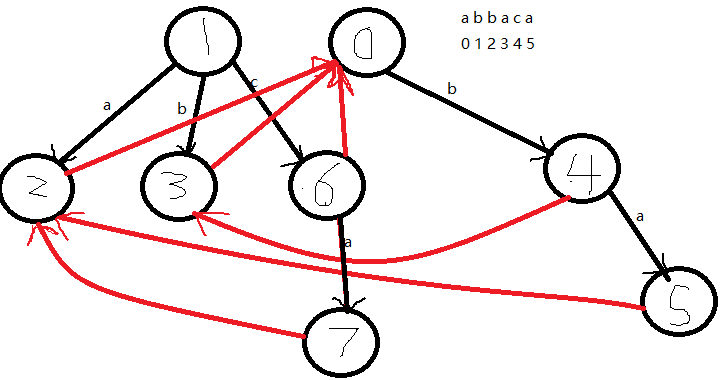
边的意义：和其他自动机一样代表字符

从r0到任意节点代表：路径上的字符的两倍所构成的回文串，如：路径上字符是”ab”,就代表回文串”abba”。从r0到自己代表空字符串。

它也代表空节点，所有节点的儿子如果指向空，则也连接到r0

从r1到任意节点代表：路径上的第一个字符 加上 其余字符的两倍所构成的回文串，如：路径上字符是”abc”,就代表回文串”cbabc”。 从r1到自己代表”负”字符串,因为定义了r1->len=-1，就相当于把从r1走的边代表了单字符串，而其余节点出发走到的字符串都是两倍来处理的。

为了解PAM怎么表示回文串，看一张图

**图1**

其中0-4-5分别为ba, 那他们就代表回文串” abba”

其中1-3分别为b, 那他们就代表回文串” b”

其中1-6-7分别为ca, 那他们就代表回文串” aca”

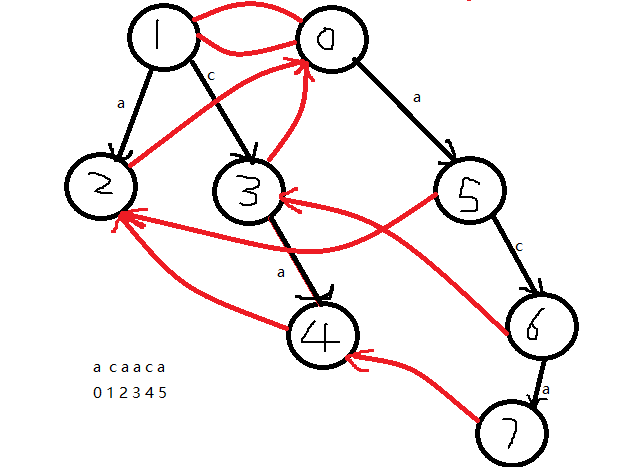
* 为了简便，我们可以直接说一个节点就能代表一个回文串

特殊的，0号节点代表空字符串，1号节点代表”负字符串”，长度为-1.

* 构建PAM的字符串不一定回文，但PAM可以找到它的回文子串

**-----------------------------------------------------------------------------**

为了理解fail指针，看下面的图

**图2**

以7号节点为例，它代表字符串”acaaca”,s[5]=’a’,除了”acaaca”以外最长回文后缀为”aca”,4号节点代表字符串”aca”,则7号节点fail指针指向4

以4号节点为例，它代表字符串”aca”,s[2]=’a’,除了”aca”以外最长回文后缀为”a”,2号节点代表字符串”a”,则4号节点fail指针指向2

以2号节点为例，它代表字符串”a”,s[0]=’a’,除了” a”以外最长回文后缀为””,0号节点代表字符串””,则2号节点fail指针指向0

可见，Fail代表：走到当前节点所表示的回文串的LPS对应的链子的末尾节点

可见，对于任意节点沿着它的fail走，则就相当于遍历了以它为弧头的边代表的字符的所以回文后缀，且长度递减。比如沿着节点7->4->2->0

则分别访问了：”acaaca”， ”aca”，”a”，”“

但如何快速构造PAM的每个节点的fail指针呢？这是下面要说的

### 构建

*//一下说的第i个字符，i都是从0开始*

首先r0和r1是初始节点，在自动机为空时就有，且他们的fail互相指对方

且r0->len=0，r1->len=-1，其他点的len值，是自己父亲的len值加2

给定一个字符串S，构建回文自动机的方法和后缀自动机类似，也是用增量法构造，即：把字符串的字符从左到右一个个加入到回文自动机(默认插入到r0上)。设last为上一次加入字符所新增节点的编号，初始last=0代表没加入过字符。

**定义getFail(now,i)操作:**

代表对带插入字符串的第i个字符操作，沿着节点now的fail指针走，找到一个满足条件的节点(什么样的节点？在这个节点代表的回文串的右侧加上一个字符s[i]在加上左侧的字符s[i-len[now]-1]可以构成回文串,这里还要求s[i-len[now]-1]必须存在)，

**getFail(now,i)返回结果是一个节点地址**

以图2来举例：对于4号节点如果我调用getFail(4,3)

1. 从4号节点开始看，它代表字符串”aca”,右边加上s[3]，左边加上s[-1]，由于s[-1]不存在则说明不满足条件，沿着4号节点的fail走到2号节点
2. 2号节点代表字符串”a”,右边加上s[3]=’a’，左边加上s[1]=’c’，显然”caa”不是回文串说明不满足条件，沿着2号节点的fail走到0号节点
3. 0号节点代表字符串””,右边加上s[3]=’a’，左边加上s[2]=’a’，满足条件，结束，找到0节点最为返回结果

以图2来举例：对于3号节点如果我调用getFail(2,1)

1. 从2号节点开始看，它代表字符串”a”,右边加上s[1]=’c’，左边加上s[-1]，由于s[-1]不存在则说明不满足条件，沿着2号节点的fail走到0号节点
2. 0号节点代表字符串””,右边加上s[1]=’c’，左边加上s[0]=’a’，显然”ac”不是回文串说明不满足条件, 沿着0号节点的fail走到1号节点
3. 1号节点代表字符串””,1号节点代表负字符串，右边加上s[1]=’c’，左边加上s[1]=’c’，显然”c”本身是回文串，满足条件，结束，找到1节点最为返回结果

* 1号节点很特殊，对于任意getFail函数，s[i]==s[i-len[1]-1]必定成立，因为len[1]=-1，所以1号的逻辑判断和其他节点一样，且getFail函数走到1号节点必然会结束。

构建回文自动机关键在于以下几个问题

1. 怎么加如第i个字符？

由于记录了last，只要从last开始沿着fail指针找到第一个满足条件的节点(什么样的节点？在这个节点代表的回文串的右侧加上一个字符s[i]在加上左侧的字符s[i-len[now]-1]可以构成回文串,这里还要求s[i-len[now]-1]必须存在)

实际就是调用pos=**getFail(last,i),** pos是getFail的返回结果，在第pos号节点下用字点树的方式建立新节点作为儿子，同时要给新节点的len赋值为posNode.len+2

1. 新节点怎么连接fail？

考虑新节点newNode的意义是：走到当前节点所表示的回文串的LPS对应的链子的末尾节点。

实际上就是找到比它自己代表的回文串短一些的回文串，这个回文串要是它的后缀。这还是getFail操作，要从posNode开始找，因为要保证找到节点代表的回文串是newNode代表的回文串的后缀。但是要去掉newNode代表的回文串本身，所以从posNode.fail开始找，所以操作为：f=getFail(posNode.fail,i)

f为返回结果，且f号节点代表的字符串一定可以在右边加上字符s[i],左边加上s[i-len[f]-1]还是回文串,只要f不是r1,f必然有儿子son[s[i]],如果f是r1，则f.son[s[i]]有可能为r0也就是空，那么直接顺势把newNode.fail连接到r0即可

无论如何，令newNode.fail=f.son[s[i]]可以使得f. son[s[i]]号节点代表的字符串是newNode代表字符串的**LPS**， newNode.fail连接正确

### 扩展属性和操作

节点扩展属性：

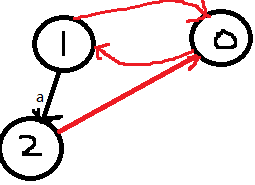
Num: 回文串后缀的本质不同的回文串个数。

r0和r1的num都是0，其他节点的num是自己fail指针节点的num+1

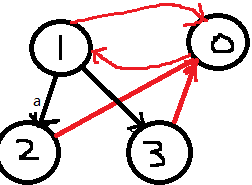
转移代码为：newNode.num=tree[newNode.fail].num+1

### 构造演示

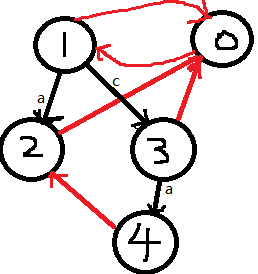
字符串为acaacaa, 初试PAM为,last=0

加入s[0]=a:此时last=0

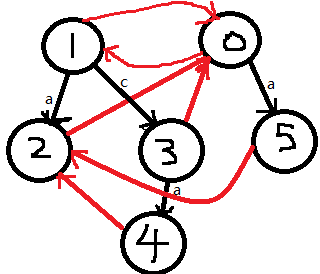
1. 调用getFail(0,0):发现s[0-len[0]-1]不存在，跳fail指针到1，s[0-len[1]-1]==s[0],故在1号节点下新建2,, last=2
2. 1号节点的fail是0, 调用getFail(0,0):发现s[0-len[0]-1]不存在，跳fail指针到1，s[0-len[1]-1]==s[0],故2号节点的fail是1的son[s[0]]所指位置，由于1的son[s[0]]不存在，则认为是0号节点，所以2号节点的fail是0

加入s[1]=c:此时last=2

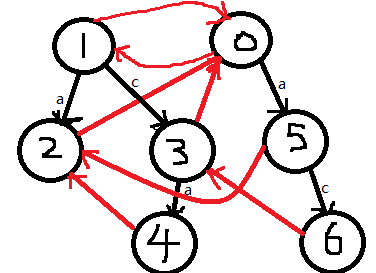
1. 调用getFail(2,1):发现s[1-len[2]-1]不存在，跳fail指针到0;发现s[1-len[0]-1]!=s[1], 跳fail指针到1，s[1-len[0]-1]==s[1],故在1号节点下新建3, last=3
2. 1号节点的fail是0, 调用getFail(0,1):发现s[1-len[0]-1]!=s[1]，跳fail指针到1，s[1-len[0]-1]==s[1], 故3号节点的fail是1的son[s[1]]所指位置，由于1的son[s[1]]不存在，则认为是0号节点，所以3号节点的fail是0

加入s[2]=a:此时last=3

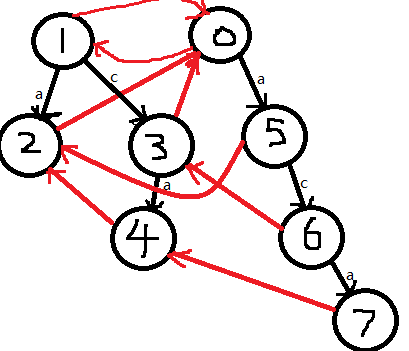
1. 调用getFail(3,2):发现s[2-len[3]-1]==s[2],故在4号节点下新建4, last=4
2. 3号节点的fail是0, 调用getFail(0,2):发现s[2-len[0]-1]!=s[2]，跳fail指针到1，s[2-len[1]-1]==s[2], 故4号节点的fail是1的son[s[2]]所指位置，所以4号节点的fail是2

加入s[3]=a:此时last=4

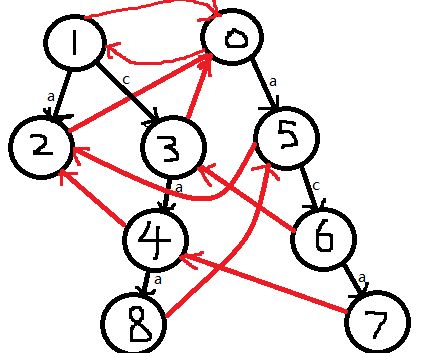
1. 调用getFail(4,3):发现s[3-len[4]-1]不存在,跳fail指针到2；发现s[3-len[2]-1]!=s[3]跳fail指针到0; 发现s[3-len[0]-1]==s[3],故在0号节点下新建5, last=5
2. 0号节点的fail是1, 调用getFail(0,3):发现s[3-len[1]-1]==s[3]，故5号节点的fail是1的son[s[3]]所指位置，所以5号节点的fail是2

加入s[4]=c:此时last=5

1. 调用getFail(5,4):发现s[4-len[5]-1]==s[4],故在5号节点下新建6, last=6
2. 5号节点的fail是2, 调用getFail(2,4):发现s[4-len[2]-1]!=s[4].跳fail到0；发现s[4-len[0]-1]!=s[4].跳fail到1; 发现s[4-len[1]-1]==s[4].故6号节点的fail是1的son[s[4]]所指位置，所以5号节点的fail是3

加入s[5]=a:此时last=6

1. 调用getFail(6,5):发现s[5-len[6]-1]==s[5],故在6号节点下新建7, last=7
2. 6号节点的fail是3, 调用getFail(3,5):发现s[5-len[3]-1]==s[5].故7号节点的fail是3的son[s[5]]所指位置，所以7号节点的fail是4

加入s[6]=a:此时last=7

1. 调用getFail(7,6):发现s[6-len[7]-1]不存在,跳fail指针到4;发现s[6-len[4]-1]==s[6]故在4号节点下新建8, last=8
2. 4号节点的fail是2, 调用getFail(2,6):发现s[6-len[2]-1]!=s[6],跳fail指针到0,；发现s[6-len[0]-1]==s[6],故8号节点的fail是0的son[s[6]]所指位置，所以8号节点的fail是5