后缀自动机

浙江大学 宋逸群

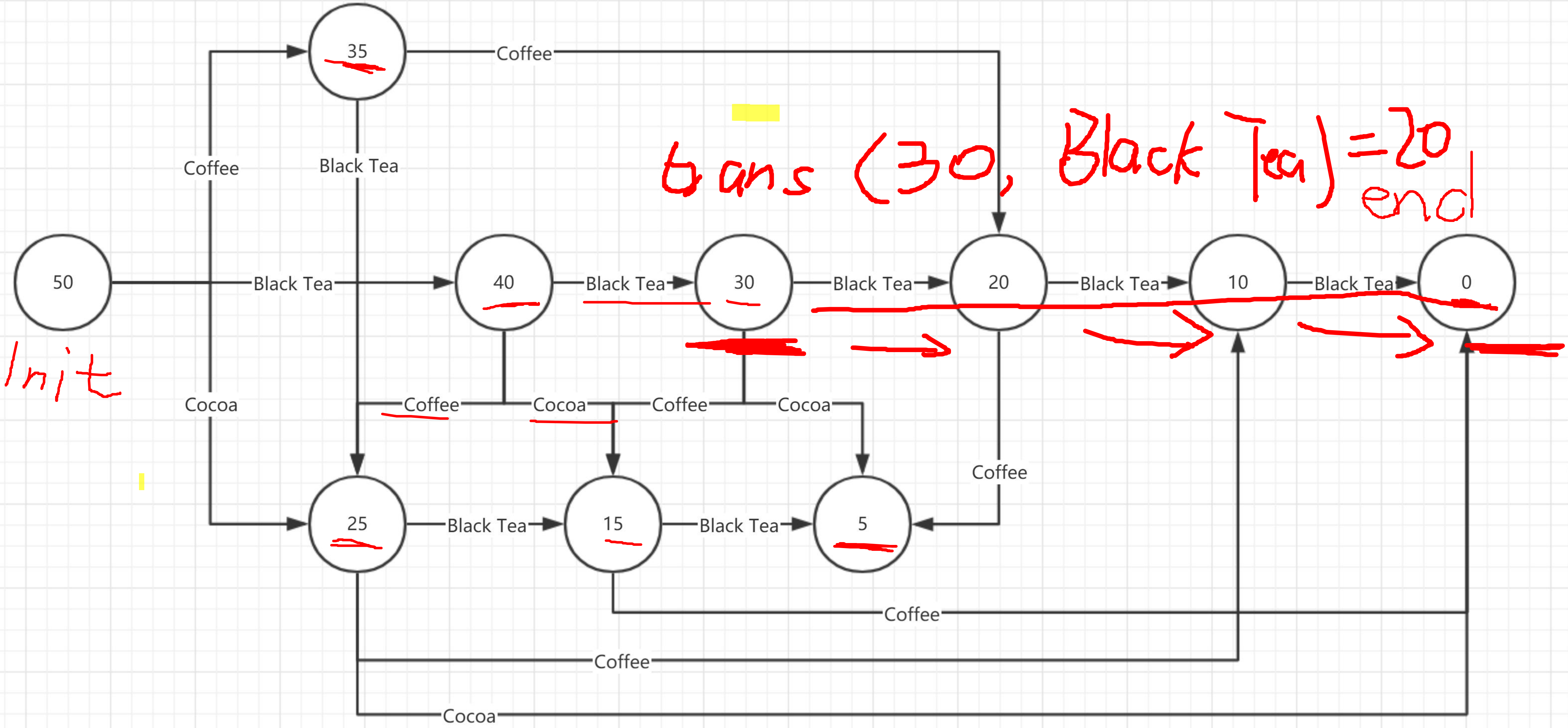
目录

- 有限状态自动机
- 后缀自动机构造原理
- 代码实现
- 经典应用
- 练习题目

致敬陈老师

有限状态机

- 绫小路现在有 50 日元，他想购买一些饮料招待即将到来的同(hou)学(gong)，可供选择的饮料有：Black Tea(10日元)，Coffee(15日元)，Cocoa(25日元)
- 那么我们可以用下面的图表示剩余钱数的状态



trans (30, Black Tea) = 20
end
Init
50
35
40
30
20
10
0
25
15
5
Coffee
Black Tea
Cocoa

有限状态机

- 我们发现这张图上共出现了五个部分：

  $alpha$：转移边的取值集合

  $state$：状态，即图中的点

  $init$：初始状态，由初始状态出发可到达其它所有状态

  end：结束状态集合，可以不止一个

  $trans$：状态转移函数，$trans(G,x)$表示状态G经过转移边x后所达到的状态

- 状态机描述了所有状态之间的转移关系

有限状态自动机

- 有限状态自动机，简称自动机，它具有识别字符串的功能

  $alpha$：字符集，一般为26个字母组成的集合

  $trans$：状态转移函数，$trans(G,ch)$表示状态G读入字符ch后所达到的状态

- 如果从初始状态沿任意路径走到某个结束状态，按顺序记下转移边上的字母得到一个字符串，那么称自动机能识别这个字符串
- 若一个自动机 A 能识别字符串 S，则记 A(S)=True，否则A(S)=False

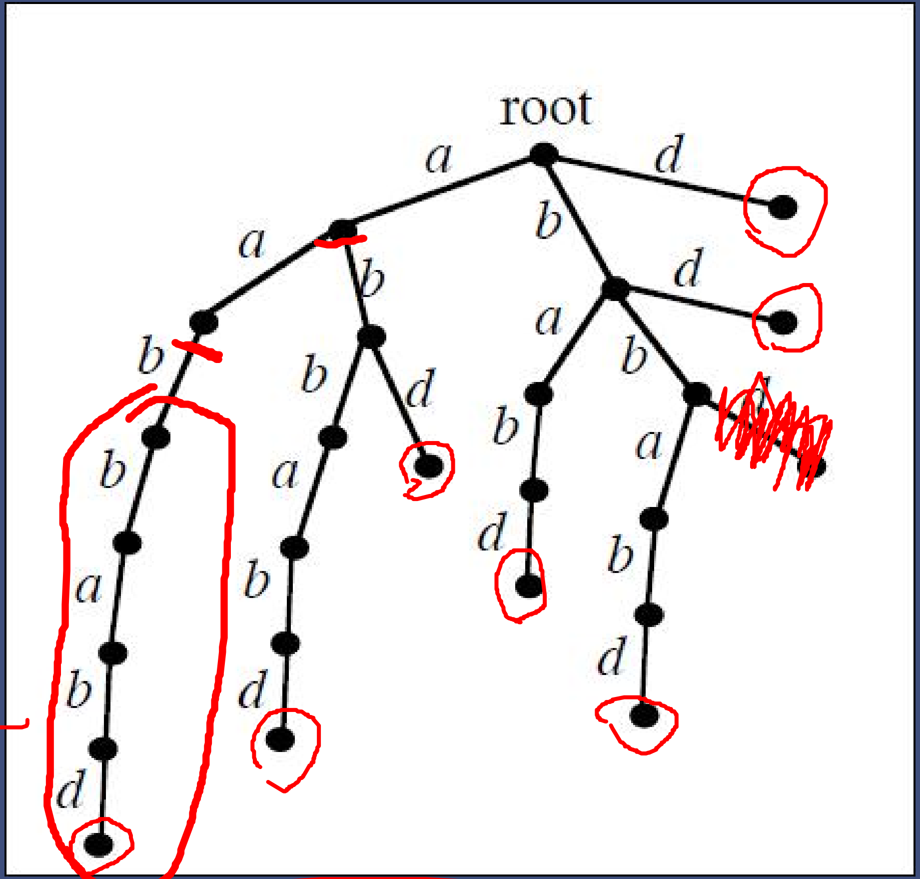
后缀自动机

- 字符串 S 的后缀自动机定义为能识别 S 的所有后缀的自动机，根据定义：
- （1）后缀自动机是一张有向无环图
- （2）后缀自动机的每个状态表示从初始状态沿不同路径到达该状态时，按顺序记下转移边上的字母得到的字符串的集合
- （3）所有结束状态的并集表示 S 的所有后缀的集合
- （4）所有状态表示的字符串都是 S 的子串

- 接下来考虑如何构建满足这些要求的后缀自动机

朴素状态后缀自动机

- 考虑字符串 S=aabbabd
- 如果将该字符串的所有后缀建成 Trie 树
- 我们发现这颗树满足条件，其中
- （1）初始状态就是根
- （2）状态转移函数就是这颗树的边
- （3）结束状态集合就是所有的叶子
- 这样我们就得到了一个 S 的朴素状态后缀自动机，但是它的节点数和边数都是$O(n^2)$的，时间和空间上都不够优秀，考虑优化



Right集合

end

- 我们引入 Right 集合的概念
- 设子串 x 在母串 S 的 $[l_1,r_1),[l_2,r_2),\ldots,[l_n,r_n)$ 位置出现
- 那么 $Right(x)=\{r_1,r_2,\ldots,r_n\}$

[3,5], [6,8], [12,14]

- 例如 S=ABBBABBABBBBBA , x=BBA

1 2 3 4 5 6 7 8 9 10 11 12 13 14

- 则 $Right(x)=\{6,9,15\}$
- 在后缀自动机上，一个状态的 Right 集就是该状态所代表字符串在母串中出现位置右端点的集合，根据定义，结束状态的 Right 集中必然包含 |S|+1

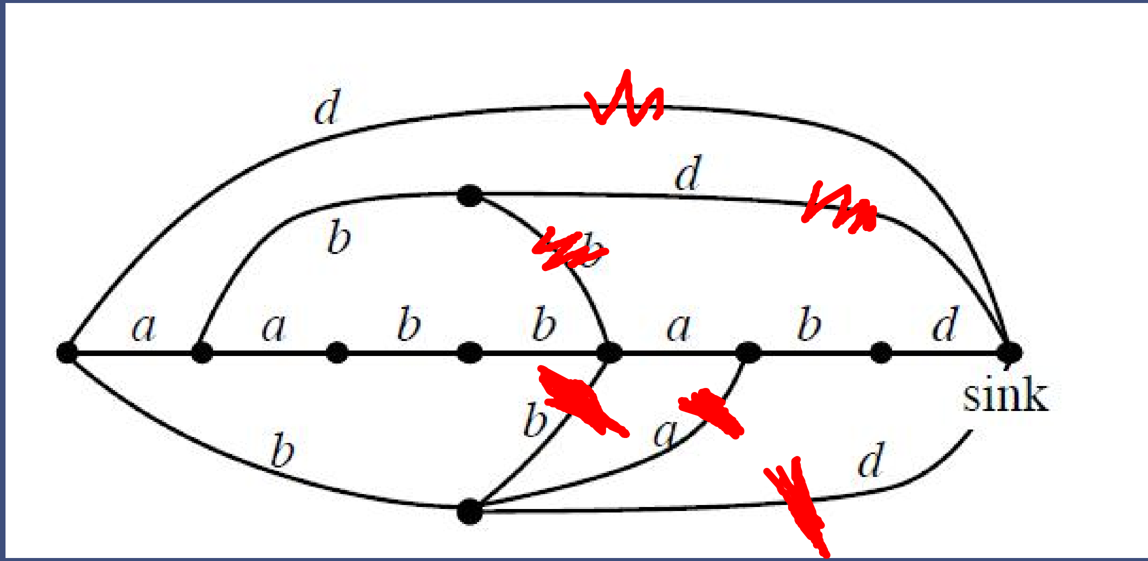
优化的核心思想

- 我们发现之前的朴素状态后缀自动机上某些状态是等价的
- 考虑两个等价的条件是什么？
- 分别从两个状态出发，走到结束状态，如果得到字符串集合相等，那么这两个状态是等价的，我们发现如果这两个状态的 Right 集相等，那么可以满足这个条件，即两个状态等价，也就是说，我们可以把 Right 集合相同的状态压缩为一个状态

- 上面这句话是后缀自动机的核心思想，必须理解

后缀自动机

- 之前的那个图，优化完成后得到的后缀自动机长这个样子



- 接下来，我们尝试挖机后缀自动机的性质，然后由这些性质出发得到后缀自动机的构造方法

子串的确定

- 假设我们已经得到了所有状态的 Right 集合，那么只需要给定一个长度 len 就可以确定子串了
- 例如：S=ABBBABBABBBBBA，以及 $Right(BBA)=\{6,9,15\}$
- 那么符合条件的子串就是BBA、BA，对应的 len 等于 3 和 2
- 可以看出所有的 len 组成了一个连续区间
- 记区间左端点为 Min，右端点为 Max
- $Max(x)$：这个串如果更长的话，Right(x) 中的某些位置就不合法了
- $Min(x)$：这个串如果更短的话，某些不属于 Right(x) 的位置也合法了

线性复杂度的证明

- 后缀自动机的状态数是线性的：

考虑两个状态a, b,若$Right(a)$和$Right(b)$有交集

因为一个子串至多属于一个状态，所以状态a，b中不会有相同的子串

所以$[Min(a), Max(a)]$和$[Min(b), Max(b)]$没有交集

不妨设$Max(a) < Min(b)$，那么状态a中所有串都比b中的短，所以a中的串都是b中串的后缀

因此，a中某串的出现位置，b中某串也必然出现了

所以$Right(a)$是$Right(b)$的真子集

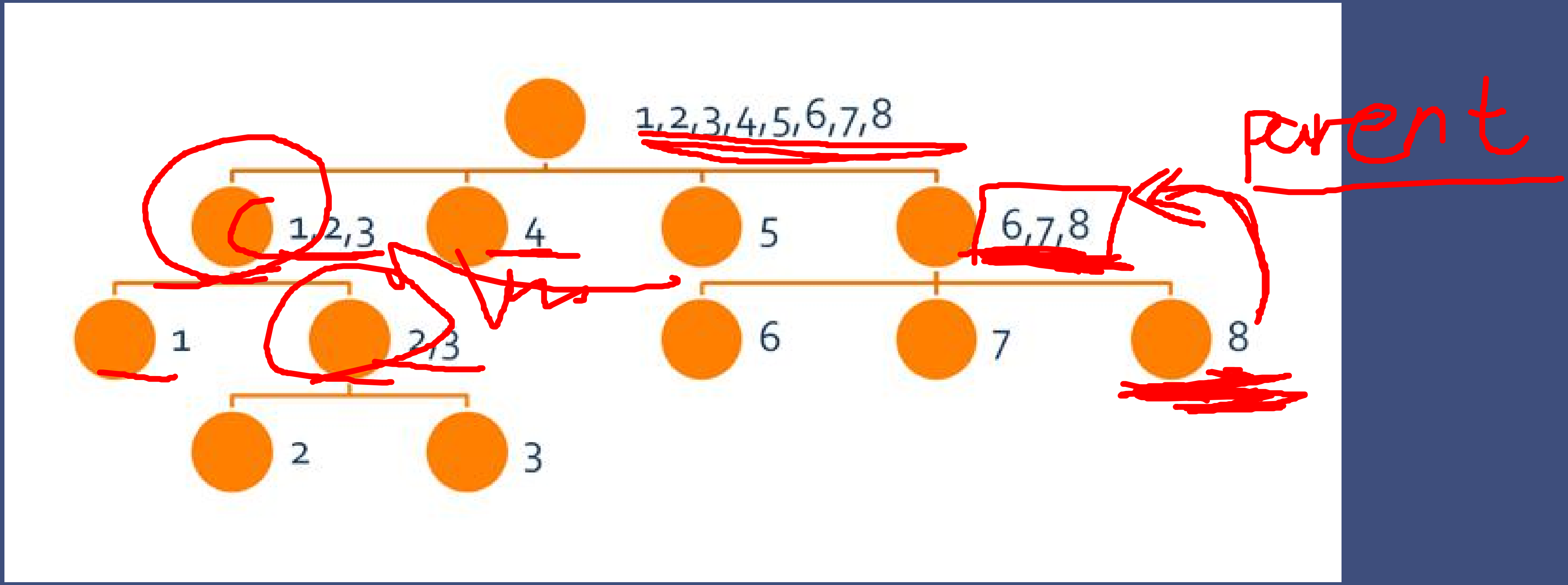
相

所以就有如下结论：任意两个状态的$Right$集合要么不想交，要么一个是另一个的真子集

- 蛤？你告我截图侵权？我写的博客我为什么不能截图

Parent树

- 事实上，所有状态的 Right 集合构成了一个树形结构，如下图



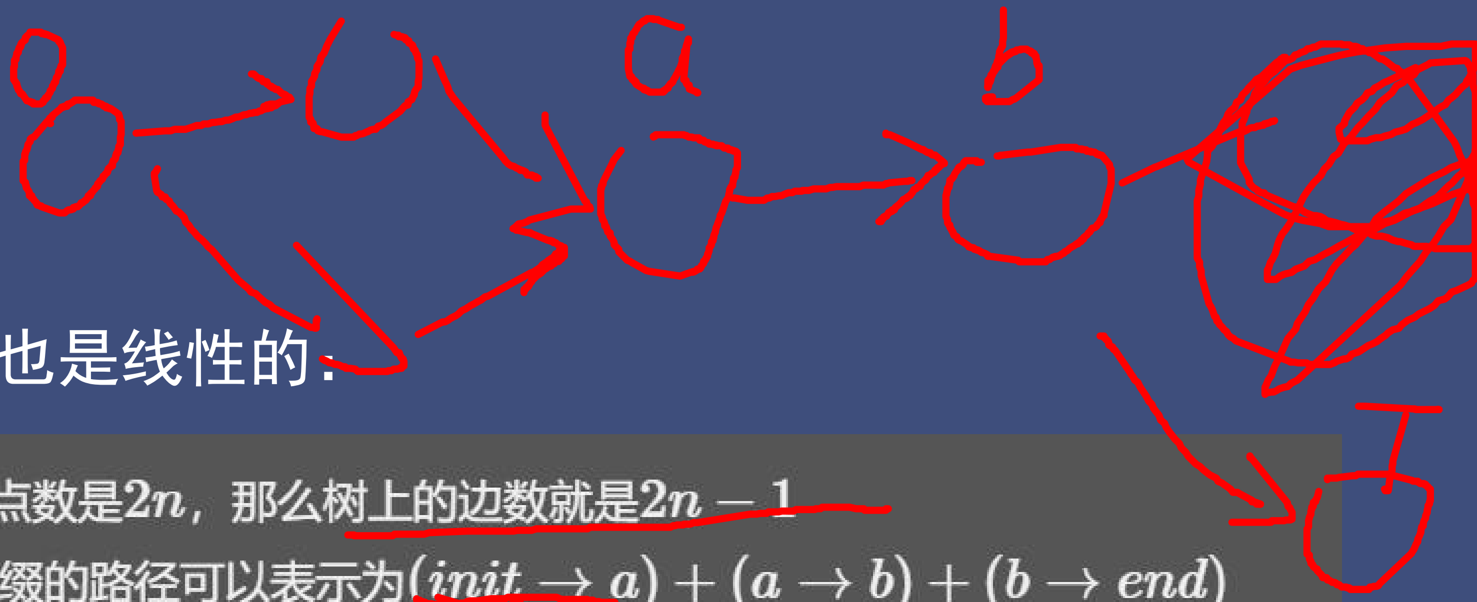
Parent树

suffix Automation

- 我们把这棵树称为 Parent 树
- 叶子结点有 n 个，内部结点至少有 2 个儿子，所以结点个数最多为 2n 个
- 这就是 SAM 为什么要开 2 倍数组的原因
- 写SAM一定要开两倍数组！
- 写SAM一定要开两倍数组！
- 写SAM一定要开两倍数组！
- 这样我们就证明了 SAM 的状态数是线性的

olxdxh

线性复杂度的证明



- 下面我们来证明转移边的数量也是线性的:

> 我们首先来求一棵SAM的生成树，因为点数是$2n$，那么树上的边数就是$2n-1$
> 对于一条非树边$a\to b$，SAM上的一个后缀的路径可以表示为$(init\to a)+(a\to b)+(b\to end)$
> 所以一个后缀对应一条非树边，一条非树边可以对应到若干个后缀
> 所以非树边最多只有n条
> 所以边数也是线性的

$\circ \to a + a \to b +$
$b \to end$

性质与结论

- 首先我们引出 $Parent$ 的定义：（这里的 $Parent$ 和树上的是一样的）
- 设两个状态 G、H，若 $Right(G) \subseteq Right(H)$ ，且 $|Right(H)|$ 是满足条件的 $|Right(H_i)|$ 中的最小的，则称 H 为 G 的 $Parent$
- 引理：$Max(Parent(G)) = \boxed{Min(G) - 1}$
- 证明：设 $H = Parent(G)$

  因为 $|Right(H)| > |Right(G)|$，所以对于 $Min(G)-1$ 这个子串

  它的出现位置的右端点必定超出了 $Right(G)$

  它属于 $Right(H)$ 的范围

性质与结论

- Right 集越大，状态中的子串越少
- 母串 S 的每个子串必然在 SAM 的某个状态里
- 一个状态的 Right 集合等于该状态在 Parent 树中所有儿子 Right 集合的并集

aaab $G \xrightarrow{c} H$ $\{r_i+1\}$

$\{r_1, r_2, \dots r_n\}$ r_i

性质与结论

S=aaabc r_i

- 考虑 S 的后缀自动机上状态的转移的过程：
- 考虑一个状态 G，设 $Right(G)=\{r_1,r_2,\dots,r_n\}$，如果有一条 $G\to H$ 且标号为 c 的转移边，考虑 H 的 Right 集合
- 由于多了一个字符，G 的 Right 集中只有满足 $S[r_i]=c$ 的 r_i 可以匹配，因此

$$Right(H)=\{r_i+1 \mid S[r_i]==c\}$$

- 因此有如下结论：
- （1）$|Right(H)|<|Right(G)|$，因此 $Max(H)>Max(G)$
- （2）如果 G 出发有标号为 c 的边，那么 Parent(G) 出发必然也有

$Right(G) \subseteq Right[Parent(G)]$

构造过程

- 后缀自动机的构造是在线的，每插入一个字符完成一次状态转移
- 我们假设已经建好了 S 的前 len 个字符构成的串 T 的 SAM
- 考虑新插入字符 x 后的情况，此时多了 len+1 个子串，且这些子串都是 Tx 的后缀设 T 的 SAM 上所有的结束状态结点 $v_1, v_2, \ldots, v_n$（Right 集中都包含 len+1）

T+x

len+1

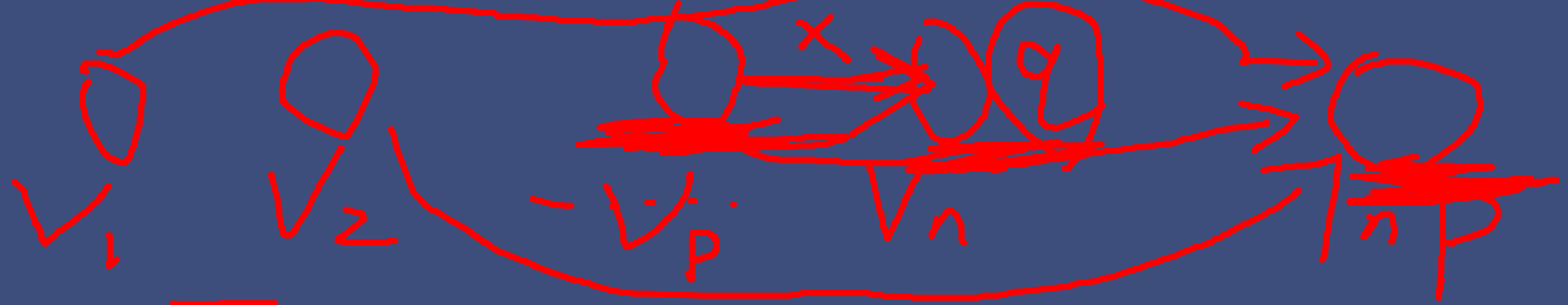
构造过程

p

- 由于必然存在一个状态，它的 Right 集中仅包含 len +1
- 那么其他的结束状态结点在 Parent 树中都是 p 的祖先
- 可以用 parent 指针得到它们

- 不妨把这些结点按照从后代到祖先的顺序排序

构造过程



- 我们新添加一个字符 x 后，新建结点 np，我们知道 $Right(np)=\{len+1\}$
- （1）对于出发没有标号为 x 的边的结点 v_i，它的 Right 集中只有 r_n 是可以匹配的，直接连一条 $v_i \to np$ 的边即可，同时更新 v_i 的 Right 集
- （2）对于第一个出发有标号为 x 的边的结点 v_p，设 $Right(v_p)=\{r_1,r_2,\ldots,r_n\}$ 令 $trans(v_p,x)=q$，则 $Right(q)=\{r_i+1 \mid S[r_i]=x\}$，由于 $v_p \to q$ 有标号为 x 的边，所以 q 是可接受态结点，需要在 Right(q) 中插入 len+1，然而直接插入可能会使 Max(q) 变小而引发一系列问题

结束状态结点

$v_p \xrightarrow{c} q$

问题出在何处

- 我们举例来看为什么会出现问题：
- 设 S = AAAAABCAAAAABAAAAAAABCAAADAAAABAAAAB
- 状态v_p代表的最长子串是AAAAB，且$Right(v_p)=\{7,14,22,32,37\}$
- 我们在 S 末尾插入字符 C，则新的 T 就是
- T = AAAAABCAAAAABAAAAAAABCAAADAAAABAAAABC
- 原本$Right(q)=\{8,23\}$，$Max(q)=7$，但是如果在$Right(q)$中插入$len+1$
- T = AAAAABCAAAAABAAAAAAABCAAADAAAABAAAABC
- $Max(q)$就会变成6，这是不符合要求的
- 如果$Max(q)=Max(v_p)+1$就不会出现这样的问题

构造过程

q

nq

- 如果$Max(q)=Max(v_p+1)$，直接令$Parent(np)=q$ 即可
- 因为$Right(np)=\{L+1\}$ 必定是$Right(q)$ 的真子集
- 否则，我们新建结点 nq，结点 q 不变，继续维护原来的转移
- 结点 nq 复制一份 q 的信息，然后在其$Right$ 集中插入$len+1$，负责维护插入新字符后的状态转移

构造过程

q nq np

- 接下来就是要考虑自动机上信息的维护
- 显然 $Right(nq) = Right(q) \cup \{L+1\}$，$Max(nq) = Max(v_p) + 1$
- 由于 $Right(q)$和$Right(np)$都是$Right(nq)$的真子集，所以 $Parent(q) = Parent(np) = nq$
- 对于v_p 的满足 $trans(v, x) = q$ 祖先 v ，令$trans(v, x) = nq$ 就行了

代码实现

- 代码实现：

```
int cnt=1,last=1;
void extend(int x){
        int np=++cnt,p=last; Right[np]=1;
        mx[np]=mx[p]+1;  last=np;
        while(p&&!tr[p][x])  tr[p][x]=np,p=par[p];
        if(!p)  par[np]=1;
        else{
                int q=tr[p][x];
                if(mx[q]==mx[p]+1)  par[np]=q;
                else{
                        int nq=++cnt; mx[nq]=mx[p]+1;
                        memcpy(tr[nq],tr[q],sizeof(tr[q]));
                        par[nq]=par[q];  par[q]=par[np]=nq;
                        while(p&&tr[p][x]==q)  tr[p][x]=nq,p=par[p];
                }
        }
}
```

求Right集

- 在很多时候，我们需要用到 Right 集的大小，如何求 Right 集的大小呢？
- 我们发现构建好的 SAM 是个有向无环图
- 我们对它进行拓扑排序，parent 树上叶子节点 |Right| 设为1，然后按照拓扑序更新 parent 即可

代码实现

- 代码实现：

```
void topsort(){
        for(int i=1;i<=cnt;++i)  c[mx[i]]++;
        for(int i=1;i<=n;++i)    c[i]+=c[i-1];
        for(int i=1;i<=cnt;++i)  id[c[mx[i]]--]=i;
        for(int i=cnt;i;--i)  Right[par[id[i]]]+=Right[id[i]];
}
```

最长公共子串

- 给定两个串A、B，求它们的最长公共子串
- 数据范围：$|A|,|B| \leq 10^5$

最长公共子串

- 算法一：二分答案+Hash验证 时间复杂度：$O(n\log n)$
- 算法二：后缀数组 时间复杂度：$O(n\log n)$
- 算法三：后缀自动机 时间复杂度：$O(n)$

- 虽然DC3后缀数组也能做到线性复杂度，但代码十分复杂
- 相比之下，SAM无疑是最优秀的算法

最长公共子串

- 后缀自动机做法：
- 将 A 串建成后缀自动机，在 SAM 上跑 B 串，应用贪心的思想
- 走转移边相当于移动 B 串匹配的右端点
- 如果无法继续走下去，就返回 parent，为什么要返回 parent ?
- 因为 parent 对应的 Right 集包含当前结点的 Right 集，而其子串长度小，这相当于移动 B 串匹配的左端点

最长公共子串 II

- 给定 n 个串，求它们的最长公共子串
- 数据范围：$|S| \le 10^5, n \le 10$

最长公共子串 II

- 我们选出一个串建 SAM，然后其他串在 SAM 上跑
- 对于每个结点记录下和每个串匹配的长度，然后取最小值

- 注意：如果一个结点匹配上了，其 parent 指针一定能匹配到最长的子串，需要沿着 parent 更新，还是拓扑序实现

最小循环串

- 给出一个字符串 S， 每次可以将它的第一个字符移到最后面，求这样能得到的字典序最小的字符串

- 比如 BBAAB， 最小的就是 AABBB

最小循环串

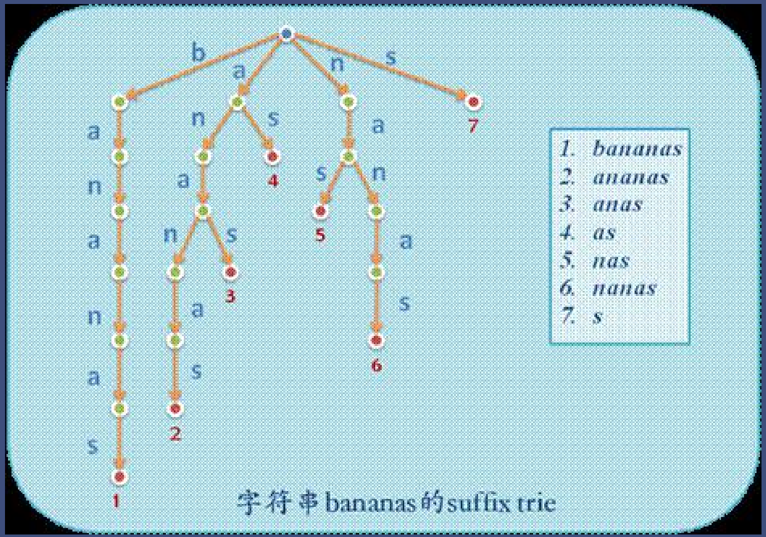
- 我们把 S 延长一倍为 SS，把它建成 SAM
- 在 SAM 上每次沿标号最小的边走，走 |S| 步得到的串就是答案

感悟

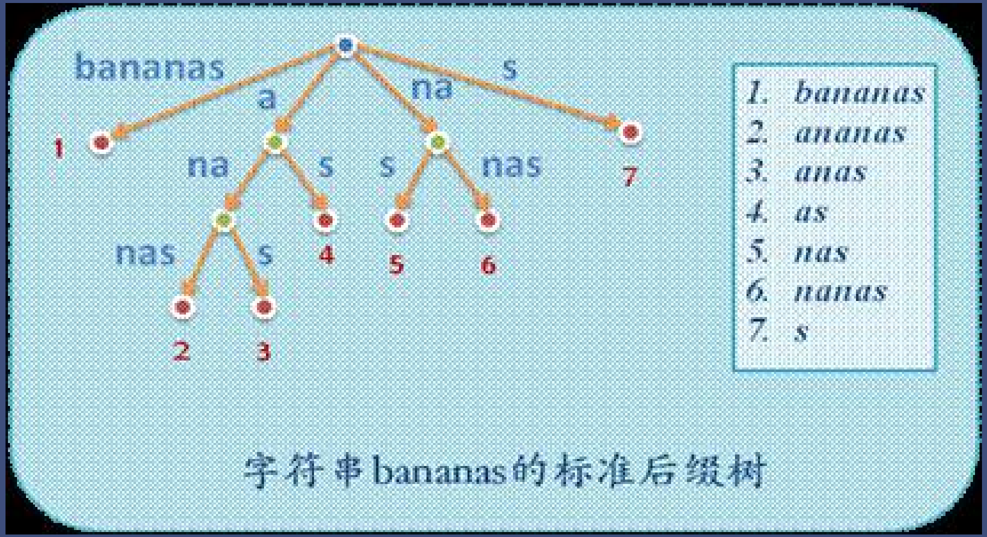
- 出现次数向父亲传递，接收串数从儿子获取
- 刷过了一些题之后，你就能理解这句话的含义

后缀树

- 把一个串所有的后缀建成一棵 Trie 树，就是这个串的后缀树
- 然而这棵 Trie 树的时空复杂度都是$O(n^2)$
- 我们考虑把边进行压缩，比如说 S=bananas



$\Rightarrow$



后缀树

- 如何建立后缀树呢？
- 其实我们得到的 Parent 树就是其反串的后缀树
- 在 Parent 树上，父亲一定是儿子的后缀
- 而在后缀树上，父亲是儿子的前缀
- 我们记录每个字符插入时在 SAM 中的位置就能求出每条边的字符

后缀数组

- 我们建好了后缀树
- 只需要在后缀树上按照字符顺序 DFS 一遍就能求出后缀数组

BZOJ4516 生成魔咒

- 现有一个空串 S，共 n 次操作
- 每次操作在 S 末尾添加一个数字 x
- 求每次操作后 S 的本质不同的非空子串个数

- 数据范围：$n \le 10^5$，$x \le 10^9$

BZOJ4516 生成魔咒

- 字符集大小为1e9， map 大法好
- 统计本质不同的子串个数是 SAM 的经典应用之一
- 本质不同的子串个数其实就是 $\sum Max(x) - Min(x) + 1$
- 所以我们新建结点 np 时统计它的答案即可

- 思考一个问题：q 结点的变化与 nq 结点的建立会影响答案吗？

BZOJ4516 生成魔咒

新建节点前：

> (1) $ans(q) = max(q) - max(parent(q))$

新建nq节点后：parent(q)变成了nq，$parent(nq)$变成了parent(q)

> (2) $ans(q) = max(q) - max(nq)$
> (3) $ans(nq) = max(nq) - max(parent(q))$

$(2)(3)$两式相加，我们发现新建节点后的$ans(q) + ans(nq)$的答案与新建节点前$ans(q)$是相等的

这就完美的解释了为什么不统计nq节点的答案

练习题目

- 【BZOJ4327】玄武密码（经典题）
- 【BZOJ3998】弦论（经典题）
- 【BZOJ4566】找相同字符
- 【BZOJ3238】差异
- 【51nod1469】淋漓尽致子串
- 【BZOJ4199】品酒大会（配合后缀树）
- 【BZOJ2555】SubString（配合LCT）

言射言射