

字符串相关问题

1

西安交通大学少年班 王袁广

- ▶ 模式串匹配算法 KMP
- ▶ 待匹配串 T (长度为 m), 模式串 S (长度为 n)
- ▶ 普通算法复杂度 $O(mn)$
- ▶ 在随机数据情况下表现好过KMP算法
- ▶ KMP算法通过 $next$ 数组将复杂度降至读入复杂度 $O(m + n)$
- ▶ $next[i]$ 表示前 i 个字符组成的子串中最长的满足后缀与前缀相等的后缀

► NOI2014 D2T1

- 对于串 T , $num[i]$ 表示 $T[0 \sim i]$ 有多少个后缀与前缀相等且后缀与前缀不覆盖.
- 即对于 $T[0 \sim i]$, 有多少个 j 使得 $T[0 \sim j]$ 与 $T[i - j + 1 \sim i]$ 相等且 $j \leq \frac{i}{2}$
- $n \leq 10^6$

- ▶ 先忽略 $j \leq \frac{i}{2}$ 这个条件
- ▶ 显然，对于长度为 i 的前缀， $next[i]$ 是最长的满足条件的子串， $next[next[i]]$ 是次长的， $next[next[next[i]]]$ 是第三长的，以此类推
- ▶ $num[i]$ 即为 $next[next \dots next[i] \dots] = 0$ （经过 $num[i] + 1$ ） 次迭代
- ▶ 直接递推即可 $O(n)$

- ▶ 处理 $j \leq \frac{i}{2}$
- ▶ 直接倍增即可 $O(n \log n)$ 需注意常数
- ▶ $next[i] \rightarrow i$ 建树, $num[i]$ 即为 i 到根节点路径上编号 $\leq \frac{i}{2}$ 的结点个数
- ▶ DFS 用树状数组维护即可 $O(n \log n)$ 比起倍增常数小
- ▶ $cnt[i]$ 长度为 i 的前缀最多经过的迭代次数, 再次匹配一次利用 cnt 数组求解, 时间复杂度 $O(n)$

- ✖ 对于多个待匹配串，需在线性时间内求出模式串是否在待匹配串中出现
- ✖ 引入*Trie*树（字母树）
- ✖ 线性时间复杂度，线性空间复杂度

- ✗ 线性时间多模式串匹配：AC自动机
- ✗ 1.建立Trie树，所有模式串建立Trie树
- ✗ 2.构造Fail指针，类似KMP的方法构造失配指针
- ✗ 3.匹配主串，类似KMP的方法匹配
- ✗ 整体与KMP算法很像，相当于KMP是链结构，AC自动机是树结构

► HDU 2222

► 给定多个模式串，求有多少个模式串在主串中出现

► $N \leq 10000, |S| \leq 50, |T| \leq 1000000$

- ▶ 裸AC自动机
- ▶ 时间复杂度 $O(N|S| + |T|)$

► HDU 3065

- 对于多个模式串，求出每个模式串在主串中出现的次数
- $N \leq 1000, |S| \leq 50, |T| \leq 2000000$

11

- ▶ 与上一题区别不大，加个数组统计即可
- ▶ 时间复杂度 $O(N|S| + |T|)$

► POJ 2778

- 对于 m 个模式串，求长度为 L 的主串有多少个不包含任何模式串
- 模式串和主串均只包含A C G T四个字母，答案对 10^5 取模
- $m \leq 10, |S| \leq 10, L \leq 2 * 10^9$

- ▶ 将所有模式串建立AC自动机
- ▶ 题目即使求有多少个长度为 n 的主串在AC自动机上无法匹配成功
- ▶ 即，从根节点走 n 步，不经过加标记的点(经过代表匹配成功，注意如果一个节点的Fail指针指向的点加标记则这个点加标记)的路径条数

- ▶ 子问题
- ▶ 对于n个点的有向图，求i到j恰好走m步的路径条数
- ▶ $n \leq 100, m \leq 2 * 10^9$

- ▶ 建立邻接矩阵 F
- ▶ 则 $F^m[i][j]$ 即为答案
- ▶ 时间复杂度 $O(n^3 \log m)$

- ▶ 回到原问题
- ▶ AC自动机相当于有向图，将所有加标记的点的入边出边删掉
- ▶ 建立邻接矩阵，原问题变为子问题
- ▶ $result = \sum F^L[root]$
- ▶ 时间复杂度 $O((m|S|)^3 \log L)$

► HDU 2243

- 求有多少个长度小于等于L的主串，满足至少包含一个模式串
- 主串和模式串均只包含小写字母，答案对 2^{64} 取模
- $0 < N < 6, |S| \leq 5, 0 < L < 2^{31}$

- ▶ 设转移矩阵为 F (即AC自动机)
- ▶ 则答案为 $\sum 26^k - \sum \sum F^k[\text{root}]$
- ▶ $\sum 26^k$ 可用二分的方法求出
- ▶ $\sum \sum F^k[\text{root}]$ 即为 $\sum (\sum F^k)[\text{root}]$ 可用二分的方法求出
- ▶ 时间复杂度 $O((N|S|)^3 \log^2 L)$

► 根据矩阵的性质

► $\begin{vmatrix} F & 1 \\ 0 & 1 \end{vmatrix}^L = \begin{vmatrix} F^L & \sum_{i=1}^{L-1} F^i + 1 \\ 0 & 1 \end{vmatrix}$

► 其中1为和 F 同阶的单位矩阵，0为和 F 同阶的零矩阵

► 时间复杂度 $O((N|S|)^3 \log L)$

► POJ 1625

- 求有多少种长度为M的主串满足不包含任何模式串，模式串和主串中仅可能出现N种字符
- $N \leq 50, M \leq 50, P \leq 10, P$ 为模式串个数

- ▶ 需要高精度
- ▶ DP转移
- ▶ $F[i][j]$ 表示长度为*i*的串在AC自动机上的*j*节点时的答案
- ▶ $F[i][j] = \sum F[i - 1][k], flag[k] \& \& next[k][son] = i$
- ▶ 即*k*未标记且*i*为*k*的儿子节点

► HDU 2825

- 求长度为n的串至少包含k个模式串的方案数
- 模式串和主串均只包含小写字母，答案对20090717取模
- $n \leq 25, k \leq m \leq 10, m$ 为模式串个数

- ▶ 和上题类似，至少包含k个模式串这个条件我们可以将状态多加一维来解决
- ▶ 状态压缩动态规划
- ▶ $F[i][j][state]$ 表示长度为i的串在AC自动机上的j节点且包含模式串状态为state时的答案
- ▶ $F[i][j][state] = \sum F[i - 1][k][state'], next[k][son] = i$
- ▶ 即i为k的儿子节点
- ▶ $O(nm^2 2^m)$

► ZOJ 3494

- 求在A和B之间有多少个数满足BCD编码不包含非法01串
- BCD编码即为十进制中每一位分别转换为二进制
- 答案对 $10^9 + 9$ 取模
- $0 \leq N \leq 100, 0 < A \leq B < 10^{200}, |S| \leq 20, N$ 为非法01串个数

- ▶ $\text{Suffix}[i]$ 代表从第 i 个字符开始的后缀
- ▶ 后缀数组 SA ：排名为 i 的后缀是 $\text{Suffix}[SA[i]]$
- ▶ 名字数组 $Rank$ ： $\text{Suffix}[i]$ 在所有后缀中排名为 $Rank[i]$
- ▶ 后缀数组与名字数组是互逆运算

- ▶ 倍增算法 $O(n \log n)$
- ▶ DC3算法 $O(n)$

a	a	b	a	a	a	a	b
---	---	---	---	---	---	---	---

b

a	b
---	---

a	a	b
---	---	---

a	a	a	b
---	---	---	---

a	a	a	a	b
---	---	---	---	---

b	a	a	a	a	b
---	---	---	---	---	---

a	b	a	a	a	a	b
---	---	---	---	---	---	---

a	a	b	a	a	a	a	b
---	---	---	---	---	---	---	---

Rank=

4 6 8 1 2 3 5 7

a	a	b	a	a	a	a	b
---	---	---	---	---	---	---	---

sa[1]=4

sa[2]=5

sa[3]=6

sa[4]=1

sa[5]=7

sa[6]=2

sa[7]=8

sa[8]=3

a	a	a	a	b
---	---	---	---	---

a	a	a	b
---	---	---	---

a	a	b
---	---	---

a	a	b	a	a	a	a	b
---	---	---	---	---	---	---	---

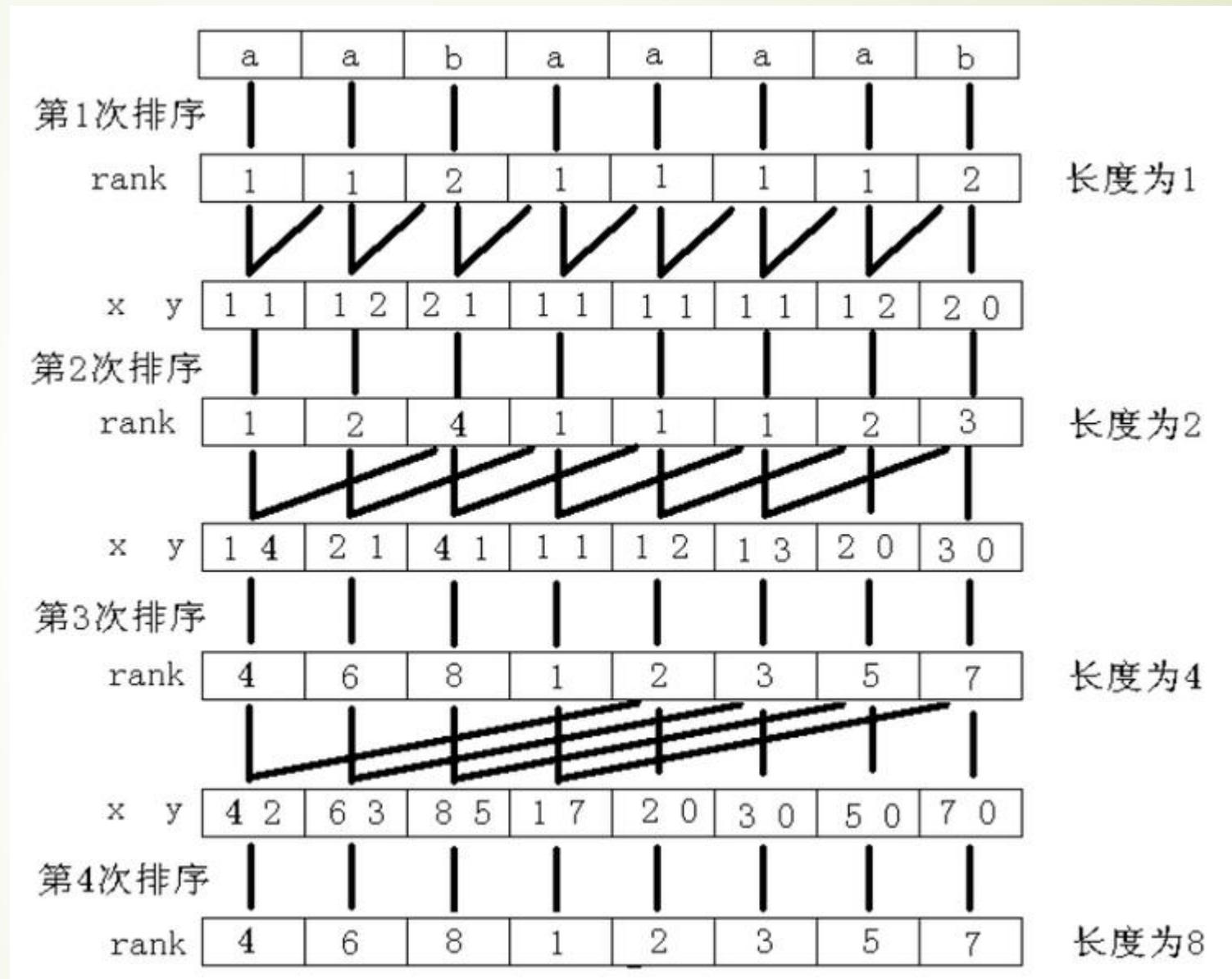
a	b
---	---

a	b	a	a	a	a	b
---	---	---	---	---	---	---

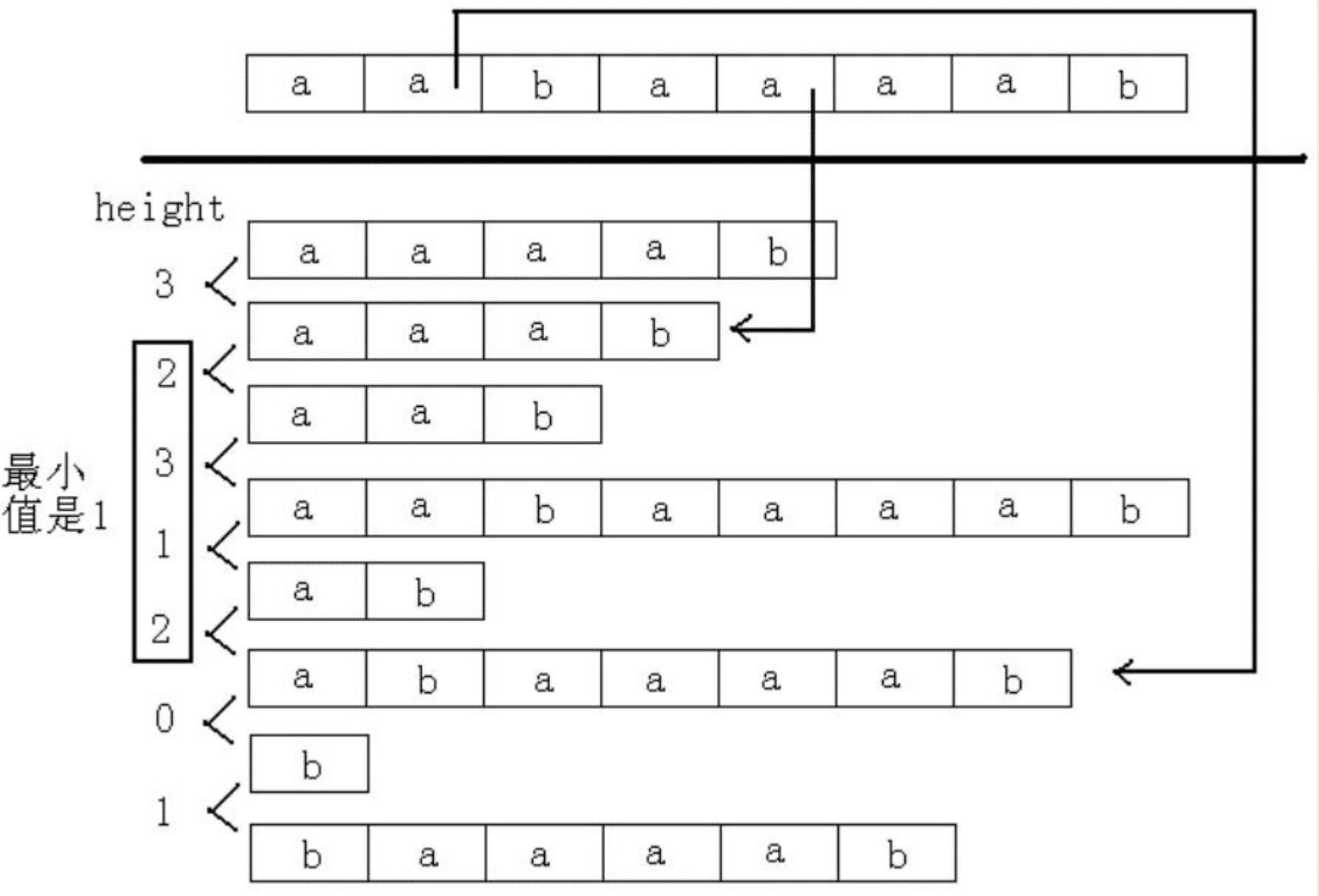
b

b	a	a	a	a	b
---	---	---	---	---	---

- 倍增算法
- 基数排序
- $O(n \log n)$



- ▶ 解决*LCP*问题
- ▶ 最长公共前缀 (Longest Common Prefix, *LCP*)
- ▶ 定义 $height[i] = LCP(Suffix[SA[i - 1]], Suffix[SA[i]])$, 即为排名相邻的两个后缀的最长公共前缀
- ▶ 对于 i 和 j , 不妨设 $rank[i] < rank[j]$, 则有以下性质:
- ▶ $LCP[Suffix[i], Suffix[j]] = \min\{height[rank[k]], i < k \leq j\}$



- ▶ 定义 $h[i] = \text{height}[\text{rank}[i]]$, 即 $\text{Suffix}[i]$ 和在它前一名的后缀的最长公共前缀
- ▶ 性质: $h[i] \geq h[i - 1] - 1$
- ▶ 证明:
 - ▶ 不妨设 $h[i - 1] > 1, h[i - 1] \leq 1$ 原式显然成立
 - ▶ 设 $\text{Suffix}[k]$ 是排在 $\text{Suffix}[i - 1]$ 前一名的后缀, 则它们的最长公共后缀为 $h[i - 1]$
 - ▶ 那么 $\text{Suffix}[k + 1]$ 排在 $\text{Suffix}[i]$ 的前面并且最长公共前缀为 $h[i - 1] - 1$
 - ▶ 故 $\text{Suffix}[i]$ 与它前一名的后缀的最长公共前缀至少是 $h[i - 1] - 1$
 - ▶ 证毕.

- ▶ 按照 h 数组的顺序并利用 h 数组的性质，计算 $height$ 数组的时间复杂度为 $O(n)$
- ▶ $for (i = 0; i < n; height[rank[i + +]] = k)$
 - ▶ $for (k ? k --: 0, j = sa[rank[i] - 1]; r[i + k] == r[j + k]; k + +);$

- ▶ LCP 问题→ RMQ 问题
- ▶ 计算后缀数组时间复杂度 $O(n \log n)$
- ▶ 计算*height*数组复杂度 $O(n)$
- ▶ 预处理*ST*表时间复杂度 $O(n \log n)$
- ▶ 单次询问*LCP*时间复杂度 $O(1)$

✗ 可重叠最长重复子串

- ▶ 建立后缀数组，求出*height*数组
- ▶ 答案即为*height*数组最大值
- ▶ 根据*LCP*的性质易证
- ▶ 时间复杂度 $O(n \log n)$

✗ POJ 1743

✗ 不可重叠最长重复子串

- ▶ 二分答案转换为判定性问题
- ▶ 即问题转换为是否存在长度为 k 的不重叠重复子串
- ▶ 将 $height$ 分组，每组内 $height$ 值均大于 k
- ▶ 若存在一组内最大的 SA 值减最小的 SA 值大于等于 k 则存在长度为 k 的不重叠重复子串，否则不存在
- ▶ 时间复杂度 $O(n \log n)$

- ▶ POJ 3261
- ▶ 可重叠的*K*次最长重复子串

- ▶ 二分答案转换为判定性问题
- ▶ 即问题转换为是否存在长度为 k 的 K 次重复子串
- ▶ 将 $height$ 分组，每组内 $height$ 值均大于 k
- ▶ 若存在一组的元素个数不小于 K 则存在长度为 k 的 K 次重复子串，否则不存在
- ▶ 时间复杂度 $O(n \log n)$

- ✗ SPOJ 694 SPOJ 705
- ✗ 不相同的子串的个数

- ▶ 子串一定是某个后缀的前缀
- ▶ 问题转化为求所有后缀之间的不相同的前缀的个数
- ▶ 按照 $\text{Suffix}[SA[1]], \text{Suffix}[SA[2]], \dots, \text{Suffix}[SA[n]]$ 的顺序计算，即按后缀的排名从小到大计算
- ▶ 假设 $\text{Suffix}[SA[1]], \text{Suffix}[SA[2]], \dots, \text{Suffix}[SA[i - 1]]$ 已计算完毕，现在需计算 $\text{Suffix}[SA[i]]$
- ▶ $\text{Suffix}[SA[i]]$ 有 $n - SA[i] + 1$ 个不同前缀，其中与前面 $i - 1$ 个后缀重复的前缀个数为 $height[i]$
- ▶ 则贡献出 $n - SA[i] + 1 - height[i]$ 个不同子串
- ▶ 时间复杂度 $O(n \log n)$

✗ URAL 1297

✗ 最长回文子串

- ▶ 将原字符串反转后复制到原字符串后变为新字符串
- ▶ “abc”→“abccba”
- ▶ 枚举中间点，则最长回文串长度转换为LCP
- ▶ 时间复杂度 $O(n \log n)$

- ✗ HDU 3068
- ✗ 最长回文子串长度
- ✗ 数据规模大需线性时间复杂度算法

- ✗ 线性时间复杂度回文子串算法
- ✗ *Manacher*算法
- ✗ 时间复杂度低，常数小，代码难度低

- ▶ 1.加入'#'将奇数回文串和偶数回文串统一为奇数回文串考虑
- ▶ $ababa \rightarrow \#a\#b\#a\#b\#a\#$
- ▶ 2.定义 $P[i]$ 为以第*i*个字符为中心的回文串向右扩展的长度(如 $P[6] = 6, P[3] = 1, P[i] - 1$ 即为其在原串中的长度)
- ▶ 问题关键在于如何在线性时间内求出P数组

- ▶ 算法大致流程为线性从前向后扫
- ▶ 求 $P[i]$ 时 $P[1] \sim P[i - 1]$ 均已求出
- ▶ 记 mx 为 $\max\{P[j] + j, j < i\}$ 即之前的回文串向右延伸的最远位
置
- ▶ 记 id 为取得 mx 时的 j 值即 $P[id] + id = mx$

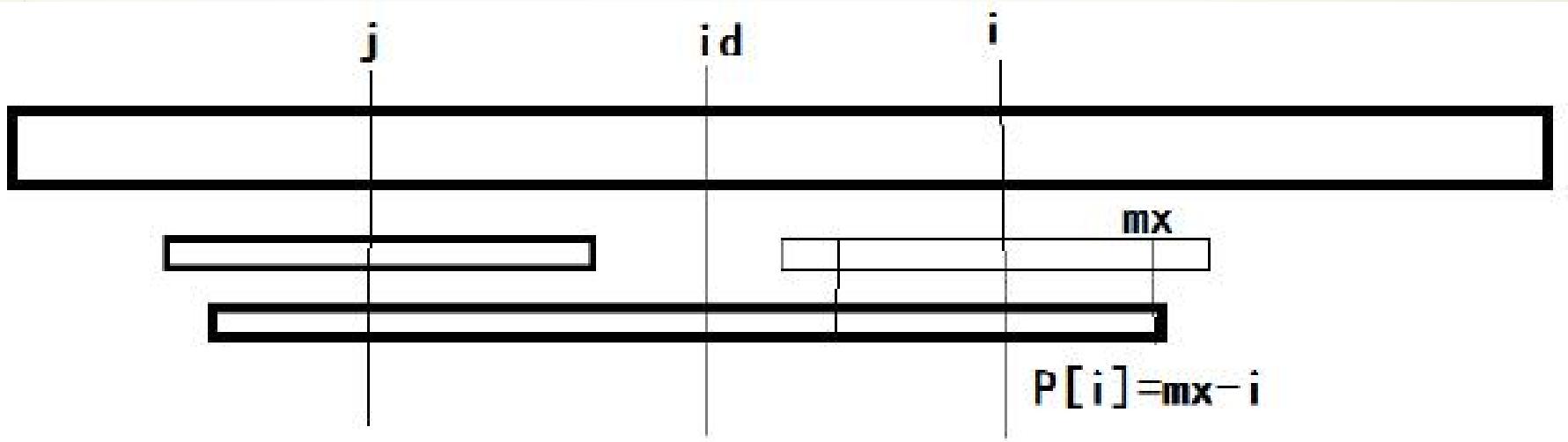
```
29 void Manacher(void) {  
30     int mx = 0, id = 0;  
31     for (int i = 1; i < n; i++) {  
32         if (mx > i) P[i] = min(P[2 * id - i], mx - i);  
33         else P[i] = 1;  
34         while (S[i + P[i]] == S[i - P[i]]) P[i]++;  
35         if (P[i] + i > mx) {  
36             mx = P[i] + i;  
37             id = i;  
38         }  
39     }  
40 }
```

► 代码非常短，重点在于 $P[i] = \min(P[2 * id - i], mx - i)$ 这一句

- 当 $mx > i$ 时 $P[i]$ 有一个最小值，这是算法核心思想
- min函数的前半部分



► min函数的后半部分



- ▶ 思考为什么算法复杂度是线性的？
- ▶ 若 $mx > i \ \&\& P[i] = P[2 * id - i]$ 则不会进入 while 和 if
- ▶ 若 $P[i] = mx - i \ || \ mx < i$ 则必然进入 while 和 if，并且 while 每循环一次相当于 mx 变大一次
- ▶ 因为每次循环要么 $O(1)$ 计算 $P[i]$ 要么 mx 至少加 1 且 mx 有最大值 n ，故 Manacher 算法时间复杂度为 $O(n)$

► POJ 2406

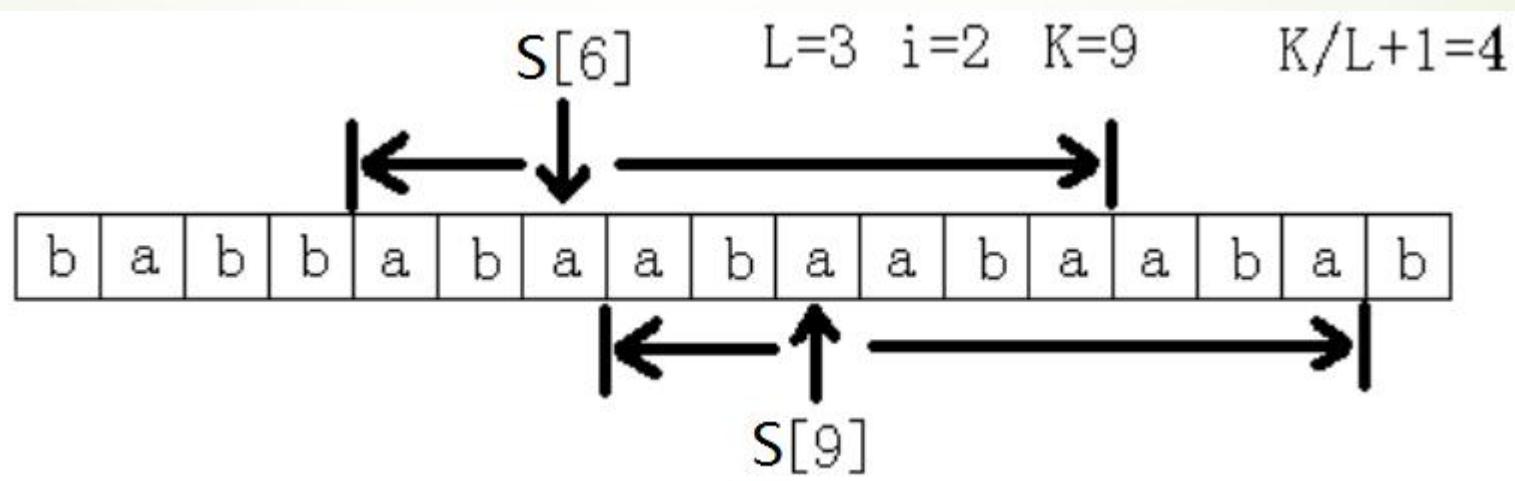
► 求最长连续重复子串

► 已知给定长度为 L 的字符串是由某个字符串 S 重复 R 次得到的，求 R 的最大值

- 枚举S的长度k
- 若 $k|L$ 并且 $LCP(Suffix[1], Suffix[k + 1]) = n - k$ 则满足条件
- 时间复杂度 $O(n \log n)$

- ✗ POJ 3693
- ✗ 重复次数最多的连续重复子串

- 枚举长度 L , 求长度为 L 的子串最多能连续出现几次
- 至少连续出现一次
- 假设子串 r 在原字符串中连续出现两次
- 则 r 必然包括了字符 $S[0], S[L], S[L * 2] \dots$ 中的相邻两个
- 记 $S[L * i]$ 和 $S[L * (i + 1)]$ 往前往后各能匹配多远, 记总长度为 K
- 则子串 r 在这里连续出现了 $\left\lfloor \frac{K}{L} \right\rfloor + 1$



- ▶ 时间复杂度还是 $O(n^2)$?
- ▶ 穷举长度L的时间复杂度 $O(n)$, 每次计算时间复杂度 $O(\frac{n}{L})$
- ▶ 总时间复杂度 $O\left(n \log n + \sum_{i=1}^n \frac{n}{i}\right) = O(n \log n)$

- ▶ POJ 2774
- ▶ 最长公共子串
- ▶ 给定两个字符串 A, B , 求最长公共子串

- ▶ 将 B 放在 A 后面并在两个字符串之间加入分隔符#构成新字符串 S
- ▶ 求出 $height$ 数组
- ▶ 答案即为，最大的 $height[i]$ 并且 $Prefix[SA[i - 1]]$ 与 $Prefix[SA[i]]$ 不是同一个字符串中的两个后缀
- ▶ 设 $n = |A| + |B|$
- ▶ 时间复杂度 $O(n \log n)$

► POJ 3415

► 长度不小于 K 的公共子串个数

A substring of a string T is defined as:

$$T(i, k) = T_i T_{i+1} \dots T_{i+k-1}, \quad 1 \leq i \leq i+k-1 \leq |T|.$$

Given two strings A, B and one integer K , we define S , a set of triples (i, j, k) :

$$S = \{(i, j, k) \mid k \geq K, A(i, k) = B(j, k)\}.$$

You are to give the value of $|S|$ for specific A, B and K .

► $A = "xx", B = "xx", ans = 5$

- ▶ 将 B 字符串复制到 A 后组成新字符串
- ▶ 求出 $height$ 数组后，分组统计
- ▶ 每组统计可用单调队列维护
- ▶ 时间复杂度 $O(n \log n)$

- ▶ POJ 3294
- ▶ 出现在不小于 K 个字符串中的最长子串

- ▶ 二分答案
- ▶ 按 $height$ 值分组
- ▶ $O(n)$ 判定可行性
- ▶ 时间复杂度 $O(n \log n)$

► HDU 4416

► 求出A串中不在所有B串中出现的不同子串个数

- ▶ 将A串与所有B串连接起来，中间用不同的分隔符
- ▶ 求出不包含分隔符的不同子串个数 $SUM1$
- ▶ 将所有B串连接起来，中间用不同的分隔符
- ▶ 求出不包含分隔符的不同子串个数 $SUM2$
- ▶ $SUM1 - SUM2$ 即为答案
- ▶ 时间复杂度 $O(n \log n)$

- ▶ 后缀自动机 Suffix Automation (SAM)
- ▶ Trie树存储所有后缀时间空间复杂度均为 $O(n^2)$
- ▶ 后缀自动机存储所有后缀时间空间复杂度均为 $O(n)$
- ▶ 并且后缀自动机仅用 $O(n)$ 时间和空间就存下了字符串的所有连续子串
- ▶ 后缀数组不支持在线操作，后缀自动机支持在线操作（不支持删除操作），实际工程中优势非常明显
- ▶ 功能强大代码简单但难理解

- ✗ 大致思路
- ✗ 线性从左到右的依次将字符加入SAM
- ✗ SAM由边+点+*parent*边组成
- ✗ 在SAM中,为了节省空间, 某个点有可能成为多个结点的儿子, 可以保证在后缀自动机中遍历出来的所有字符串不会重复, 而且刚好是原串的所有子串

- ▶ 每个点有三个信息
- ▶ ① $son[]$: 值为 $NULL$ 或者指向某个点, 指向某个点表示从 $root$ 遍历到此点的任意路径均代表原串的一个子串
- ▶ ② len : 根节点到此点的最长路径长
- ▶ ③ fa : 【不是父节点】是上一个能“接收”后缀的点

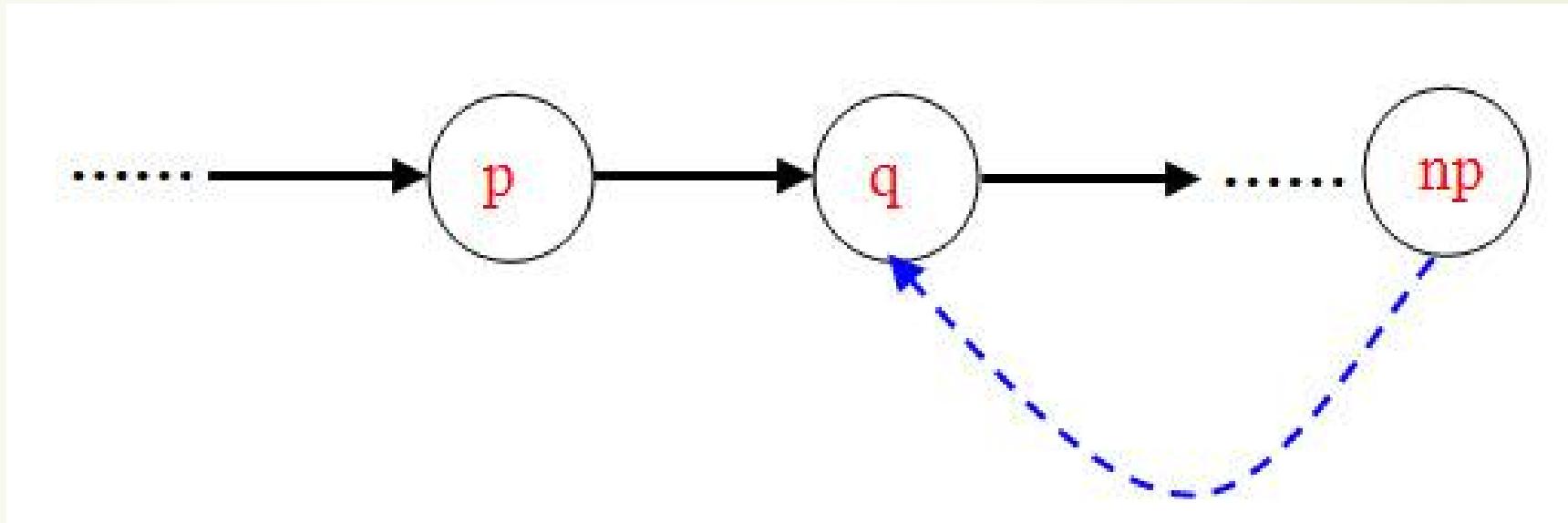
► SAM的三个性质

- ①从 $root$ 到任意结点 p 的每条路径上的字符组成的字符串都是前缀 t 的子串
- ②如果当前结点是可以接收新后缀的结点，那么从 $root$ 到任意结点 p 的每条路径上的字符组成的字符串都必定是前缀 t 的后缀
- ③如果结点 p 可以接收新的后缀，那么 p 的 pre 指向的结点也可以接收后缀

- ▶ 当前建立了前缀 t 对应的SAM，现在要将新的字符 x 加入SAM
- ▶ 首先建立存储当前字符的结点 np
- ▶ 找到之前最后一个建立的结点（因为它一定满足性质②），然后不断按 pre 指针跳（直到跳到有 x 儿子的结点为止）
- ▶ 假设当前跳到 p 结点，如果 p 没有 x 儿子，那么将 p 结点的 x 儿子赋值为 np 结点
- ▶ 现在处理 p 结点有 x 儿子的情况，设 p 结点的 x 儿子是 q
- ▶ ① $len[q] = len[p] + 1$
- ▶ ② $len[q] > len[p] + 1$

- ▶ ①将 np 结点的 pre 指针指向 q 结点
- ▶ 为什么？
- ▶ 只要证明 q 结点满足性质②即可（因为这样连相当于将 q 结点和 np 结点看成“一样”的结点， q 结点和 np 结点均可接受后缀）
- ▶ 只需证明从 $root$ 到 q 结点的所有路径都经过 p 结点
- ▶ 反证法：
 - ▶ 1.若除了 q 结点不存在 x 字符的点，则说明所有包含 x 字符的子串均为后缀，矛盾
 - ▶ 2.若除了 q 结点存在 x 字符的点，则有 x 字符但不是后缀的子串必然与之前的某个代表 x 字符的结点连接，而不是现在的 q 结点

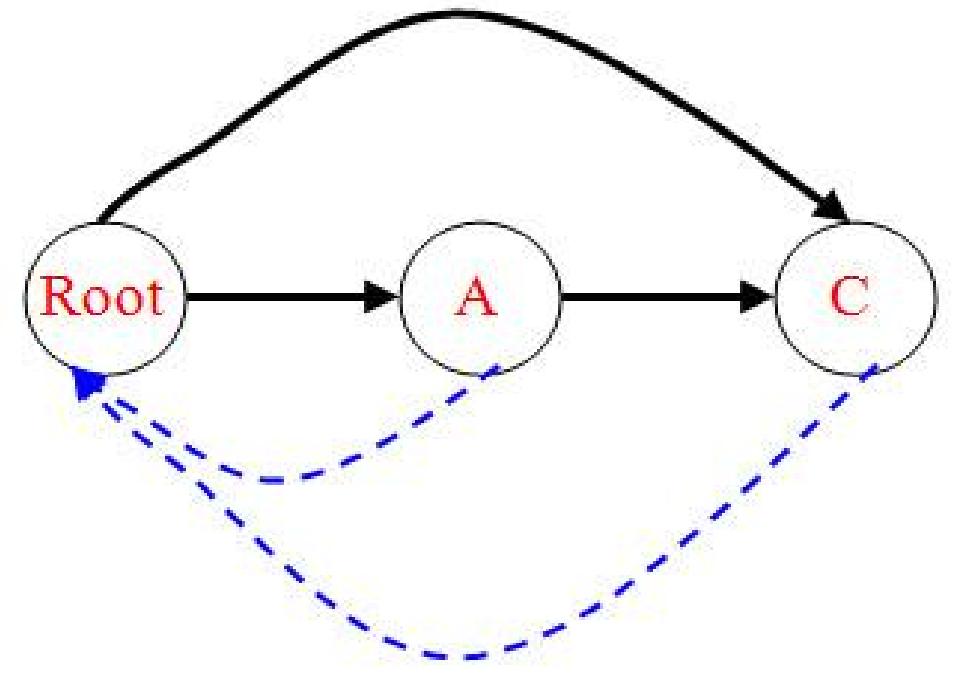
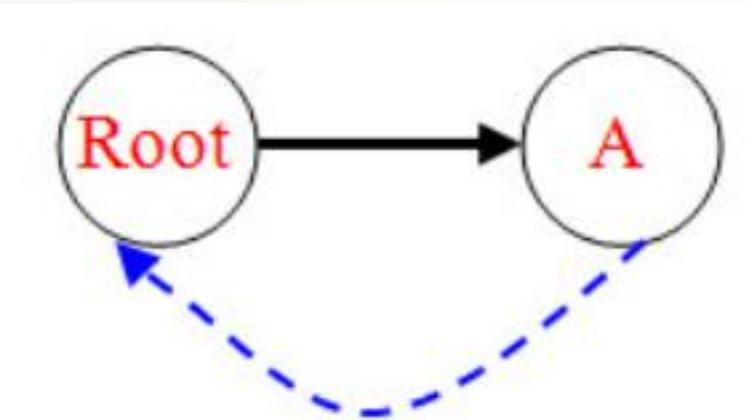
- 综上, $\text{len}[q] = \text{len}[p] + 1$ 保证了到达 q 结点的都是后缀, 故①只需要将 np 结点的 pre 指针指向 q 结点 (找最后的有 x 儿子的结点比较显然不再赘述)

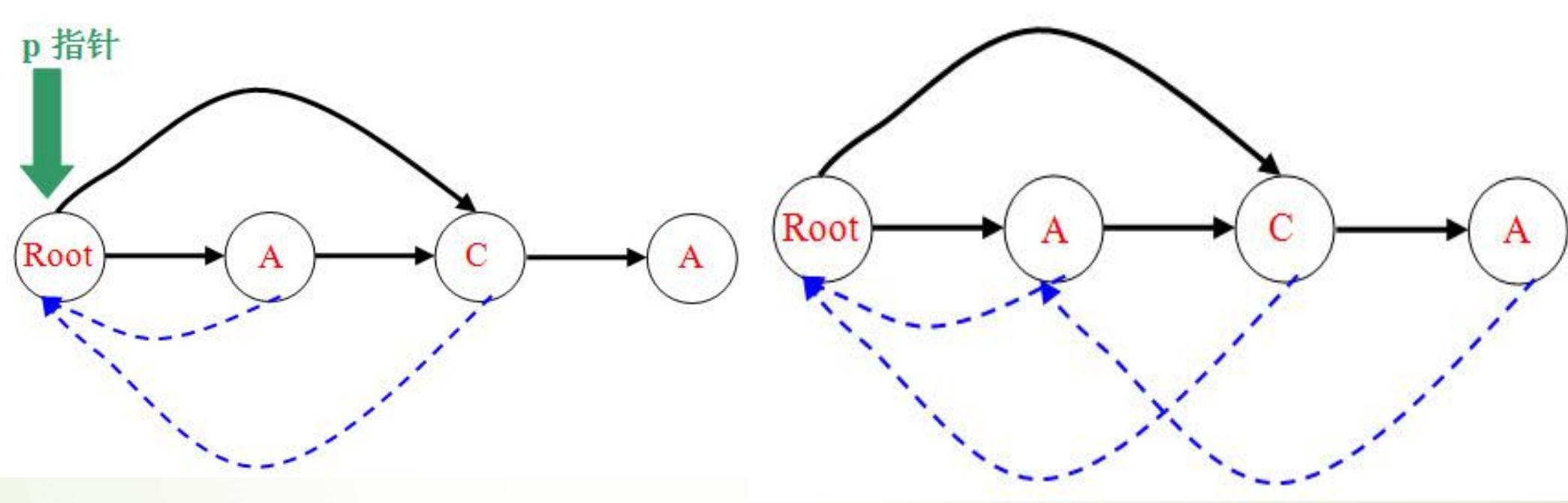


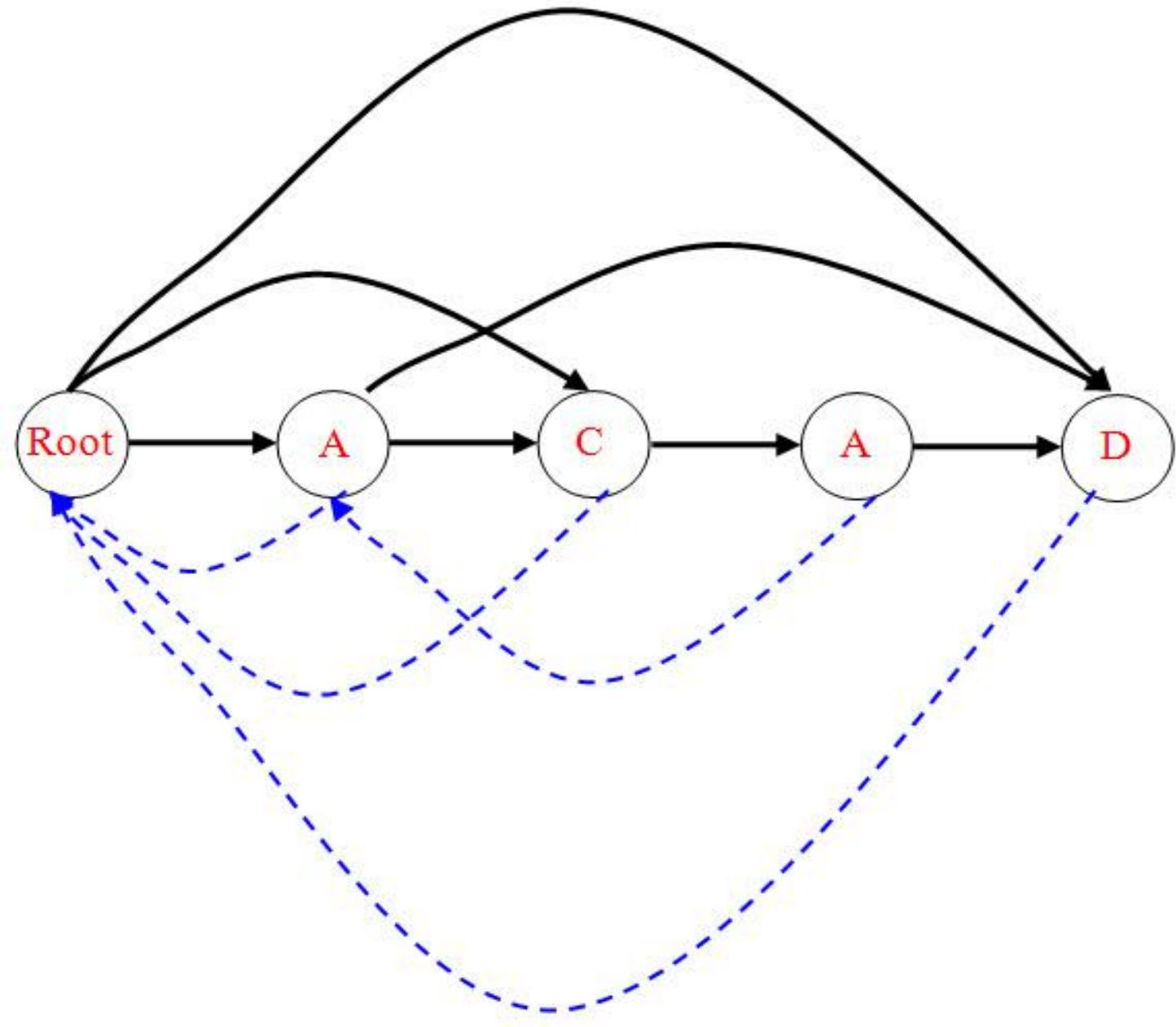
- ▶ ②思路为建立新的“代替” q 结点的 nq 结点转换为①
- ▶ 1.将 q 结点的所有边（不包括连向 fa 的边）复制给 nq 结点
- ▶ 2.将 q 结点的 pre 指针指向 p 结点
- ▶ 3.将 q 结点和 np 结点的 pre 指针指向 nq 结点
- ▶ 现在 p 结点和 nq 结点满足①即 $len[nq] = len[p] + 1$
- ▶ 均满足性质①②③

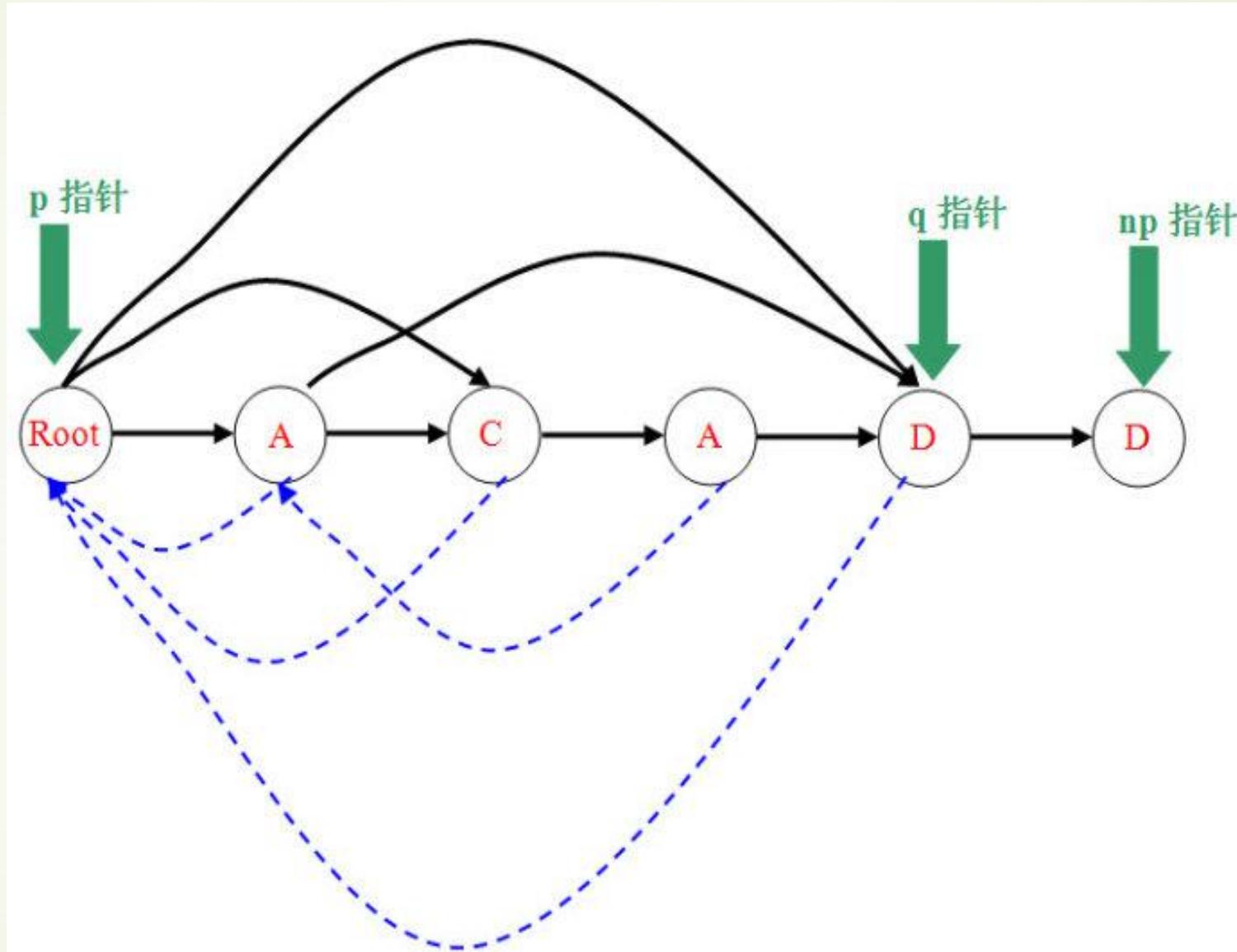
```
28 struct Suffix_Automation {
29     struct Node {
30         int len, f, ch[26];
31         Node() {
32             len = 0, f = -1;
33             memset(ch, -1, sizeof(ch));
34         }
35         }e[maxn << 1];
36         int idx, last;
37         Suffix_Automation() {
38             idx = last = 0;
39         }
40         void add(int c) {
41             int end = ++idx;
42             int tmp = last;
43             e[end].len = e[last].len + 1;
44             while (tmp != -1 && e[tmp].ch[c] == -1) {
45                 e[tmp].ch[c] = end;
46                 tmp = e[tmp].f;
47             }
48             if (tmp == -1) e[end].f = 0;
49             else {
50                 int next = e[tmp].ch[c];
51                 if (e[tmp].len + 1 == e[next].len) e[end].f = next;
52                 else {
53                     int np = ++idx;
54                     e[np] = e[next];
55                     e[np].len = e[tmp].len + 1;
56                     e[next].f = e[end].f = np;
57                     while (tmp != -1 && e[tmp].ch[c] == next) {
58                         e[tmp].ch[c] = np;
59                         tmp = e[tmp].f;
60                     }
61                 }
62             }
63             last = end;
64         }
65 }SAM;
```

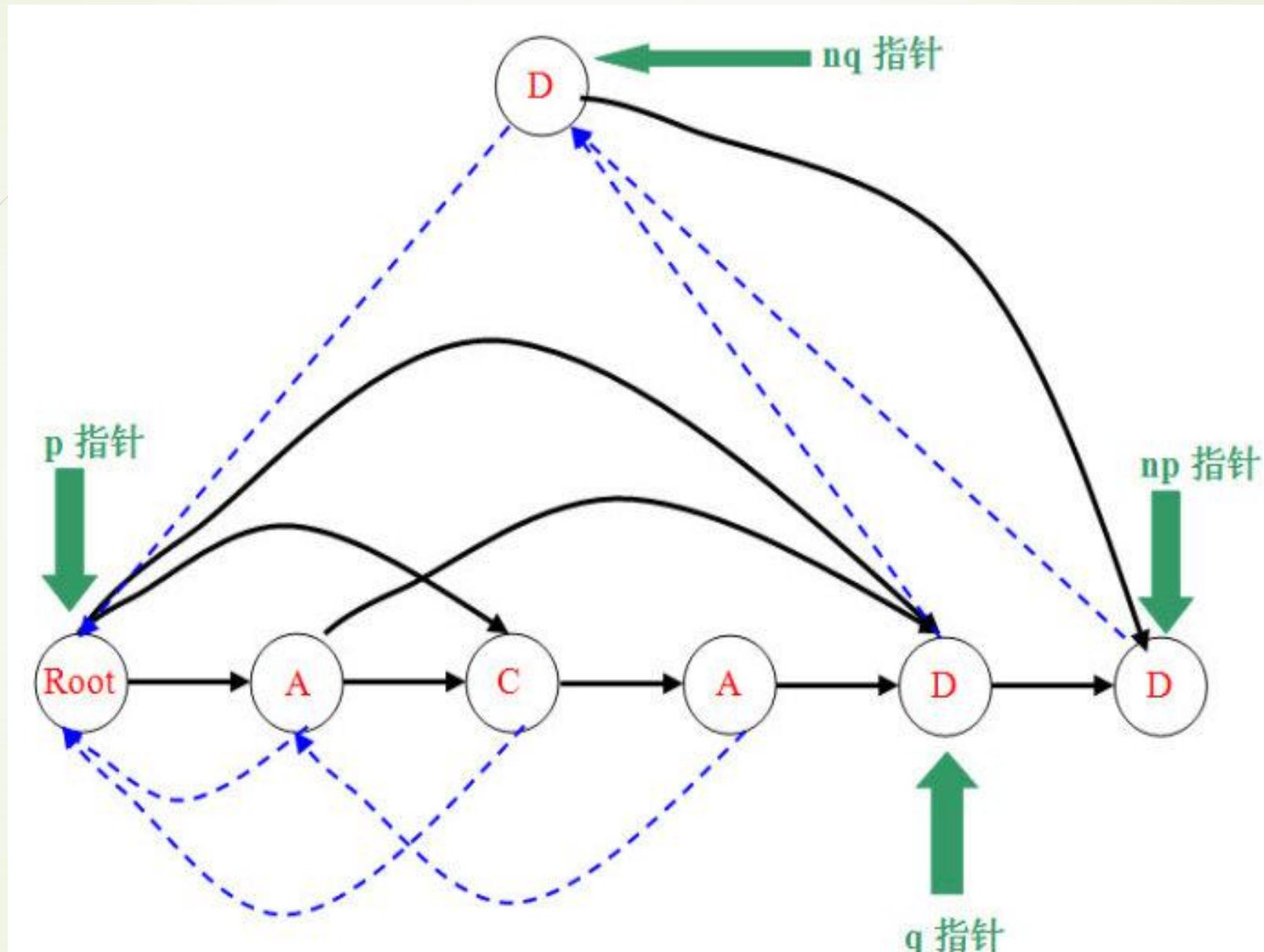
× SAM样例：构造ACADD

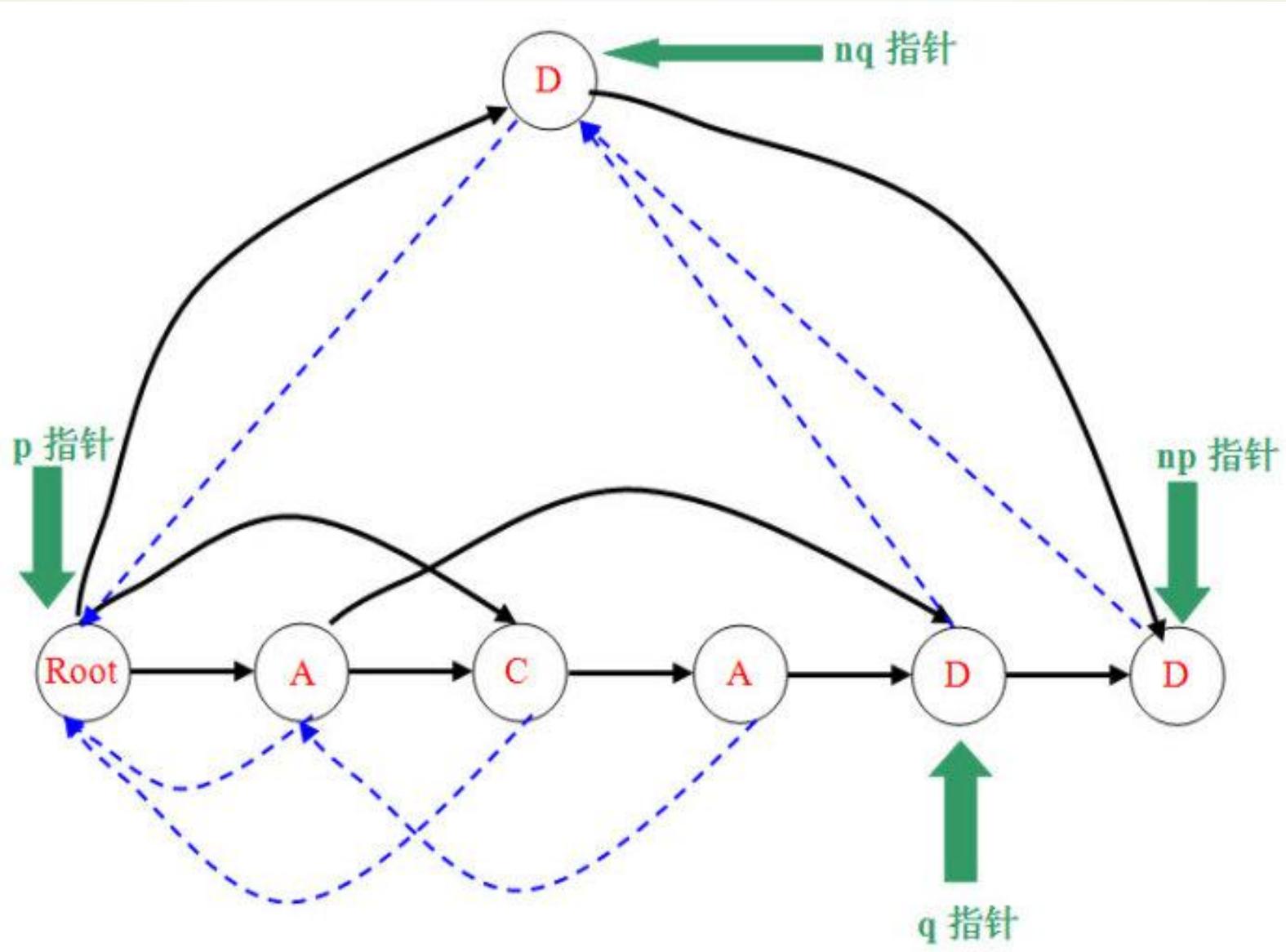












- ✗ ①如何按字典序输出所有子串？
- ✗ ②如何计算不相同的子串个数？

- ✗ SPOJ 1811
- ✗ 求最长公共子串

- ▶ 将 A 串建立SAM
- ▶ 用 B 串在SAM上进行匹配
- ▶ 时间复杂度 $O(n)$

- ▶ SPOJ 1812
- ▶ 求T个串的最长公共子串
- ▶ $L \leq 10^5, T \leq 10$

- ▶ 将第一个串建立SAM
- ▶ 根据SAM的性质，每一个结点的匹配结果可以传递给父结点
- ▶ 由于SAM是DAG，所以对于每一个新串可以自底向上的更新结果
- ▶ $O(TL)$

► HDU 4622

- 对于一个字符串，每次询问一个连续子串中有多少个不同的子串
- $L \leq 2000, Q \leq 10000$
- $O(LQ) \rightarrow \text{TLE}$

- ▶ 将询问排序
- ▶ 利用SAM支持在线的性质处理询问
- ▶ 时间复杂度 $O(L^2 + Q \log Q)$