字符串算法 part 3[1]

赖金霖[2]
清华大学　计算机科学与技术系

Aug 11, 2019

[1]本文件可在这里找到: https://github.com/lll6924/public_slides
[2]邮箱: laijl16@mails.tsinghua.edu.cn

单词统计问题

给定一个英语句子，统计若干给定单词的出现次数

样例输入

He will go with her, but he will not stay for long.
he
wi

样例输出

3
3

规定

- 不区分大小写
- 单词可以出现在另一个单词里
- 单词可以相交（如 aa 在 aaa 里出现了两次）

和上一部分相比，我们只增加了寻找的单词的数量

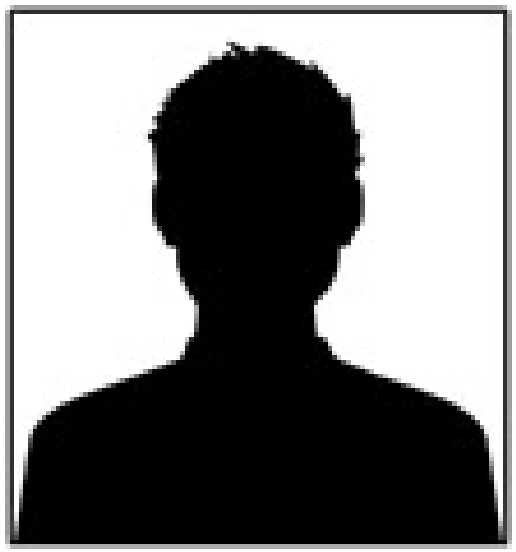
单词统计问题

- 我们已经熟练掌握了 KMP 算法，这种问题当然可以直接做
- 对每个单词，首先计算 next 数组，再用 KMP 算法统计句子里的出现次数
- 时间复杂度 O(NM+L)，其中 N 为句子长度，M 为单词个数，L 为单词长度之和
- 复杂度瓶颈在于，对不同的单词，都需要重新遍历一遍句子
- 如果有一种算法能通过一次遍历算出所有单词的出现次数，复杂度就是 O(N+L)

AC 自动机

- 解决这个问题的通用算法叫做~~自动 AC 机~~ AC 自动机

Alfred V. Aho



Margaret J.Corasick

据经验，AC 自动机会比 KMP 算法好理解

AC 自动机

- 我们以一个例子出发，介绍 AC 自动机的思路
- 假设我们要在 ababacabaa 里找 abab、abaca 和 acab
- 首先，我们要分别为 abab、abaca 和 acab 建立 next 数组
- 你发现了什么问题吗?

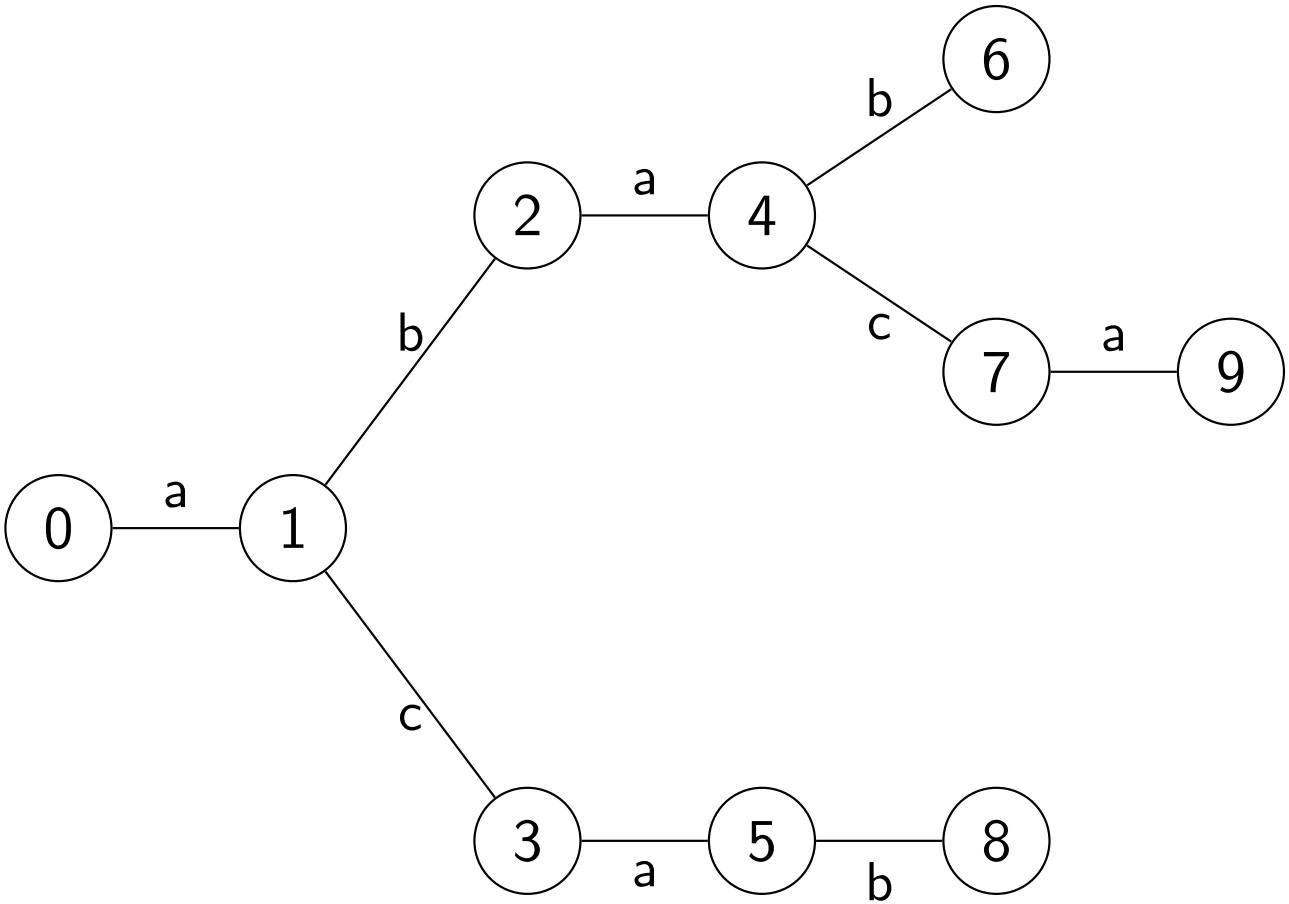
AC 自动机

- 我们以一个例子出发，介绍 AC 自动机的思路
- 假设我们要在 ababacabaa 里找 abab、abaca 和 acab
- 首先，我们要分别为 abab、abaca 和 acab 建立 next 数组
- 你发现了什么问题吗?
- 1. abab 和 abaca 的前三个字符相同，所以 next 数组的数值也相同
- 2. 更进一步，如果我们匹配 abaca 的最后一个字符失败了，当前匹配的字串是 abac，是否可以转到匹配 acab?
- 我们能利用 Trie 树的结构来加速计算吗?

AC 自动机

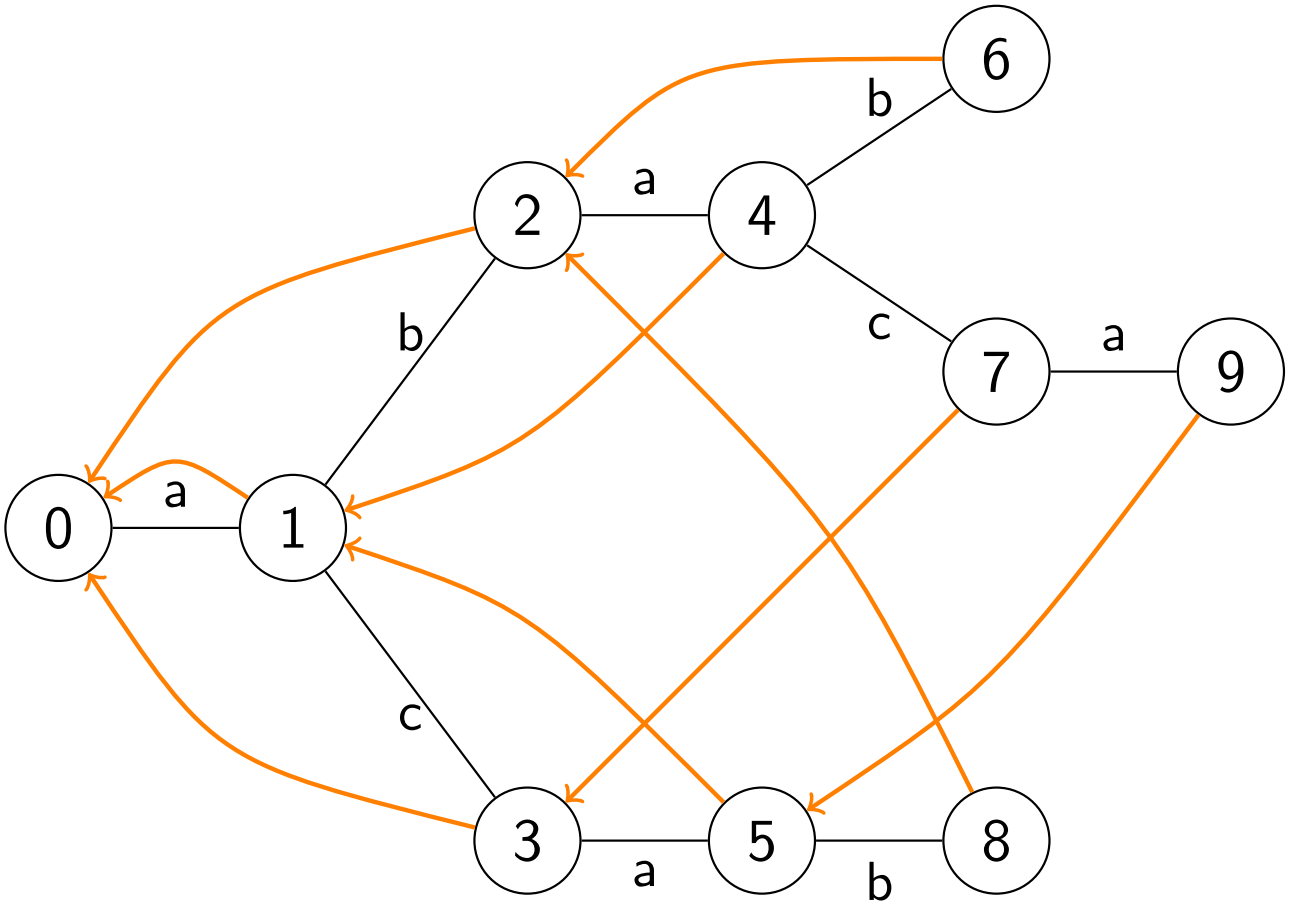
- 我们以一个例子出发，介绍 AC 自动机的思路
- 假设我们要在 ababacabaa 里找 abab、abaca 和 acab
- 首先，我们要分别为 abab、abaca 和 acab 建立 next 数组
- 你发现了什么问题吗?
- 1. abab 和 abaca 的前三个字符相同，所以 next 数组的数值也相同
- 2. 更进一步，如果我们匹配 abaca 的最后一个字符失败了，当前匹配的字串是 abac，是否可以转到匹配 acab?
- 我们能利用 Trie 树的结构来加速计算吗?
- AC 自动机正是这样的结构

AC 自动机



将三个字符串转成 Trie 树如上图所示。和 KMP 算法类似，我们可以计算 Trie 树的 “next 数组”！

AC 自动机



我们需要计算如上的关系，其中橙色边被称为失配边

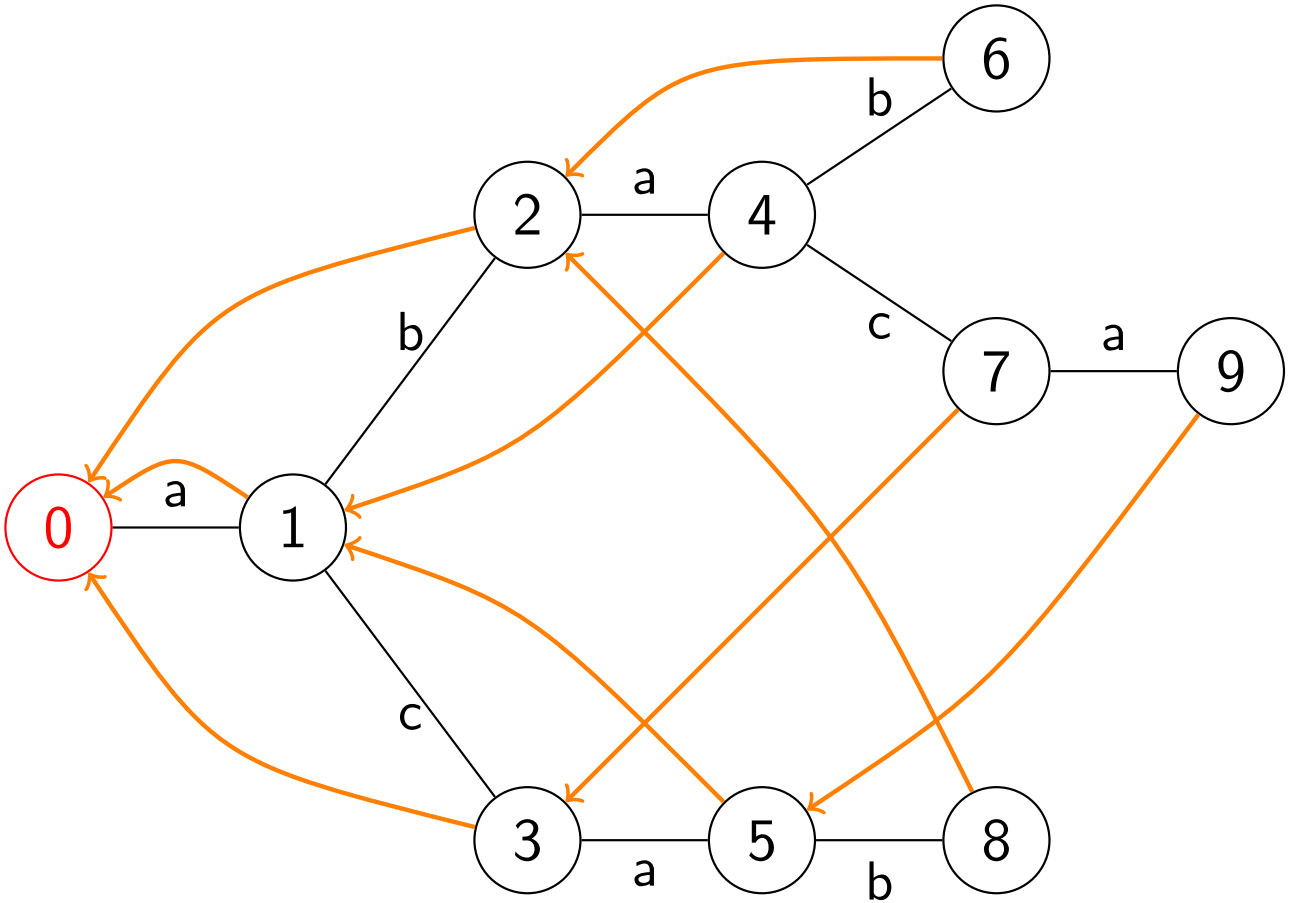
AC 自动机

- 与 KMP 算法类似，我们可以设计失配边的计算算法
- 我们设 next[i] 表示 i 的失配边指向的结点，pre[i] 表示 i 在 Trie 上的父结点
- next[i] 可以通过 next[pre[i]] 推出，怎么做?

AC 自动机

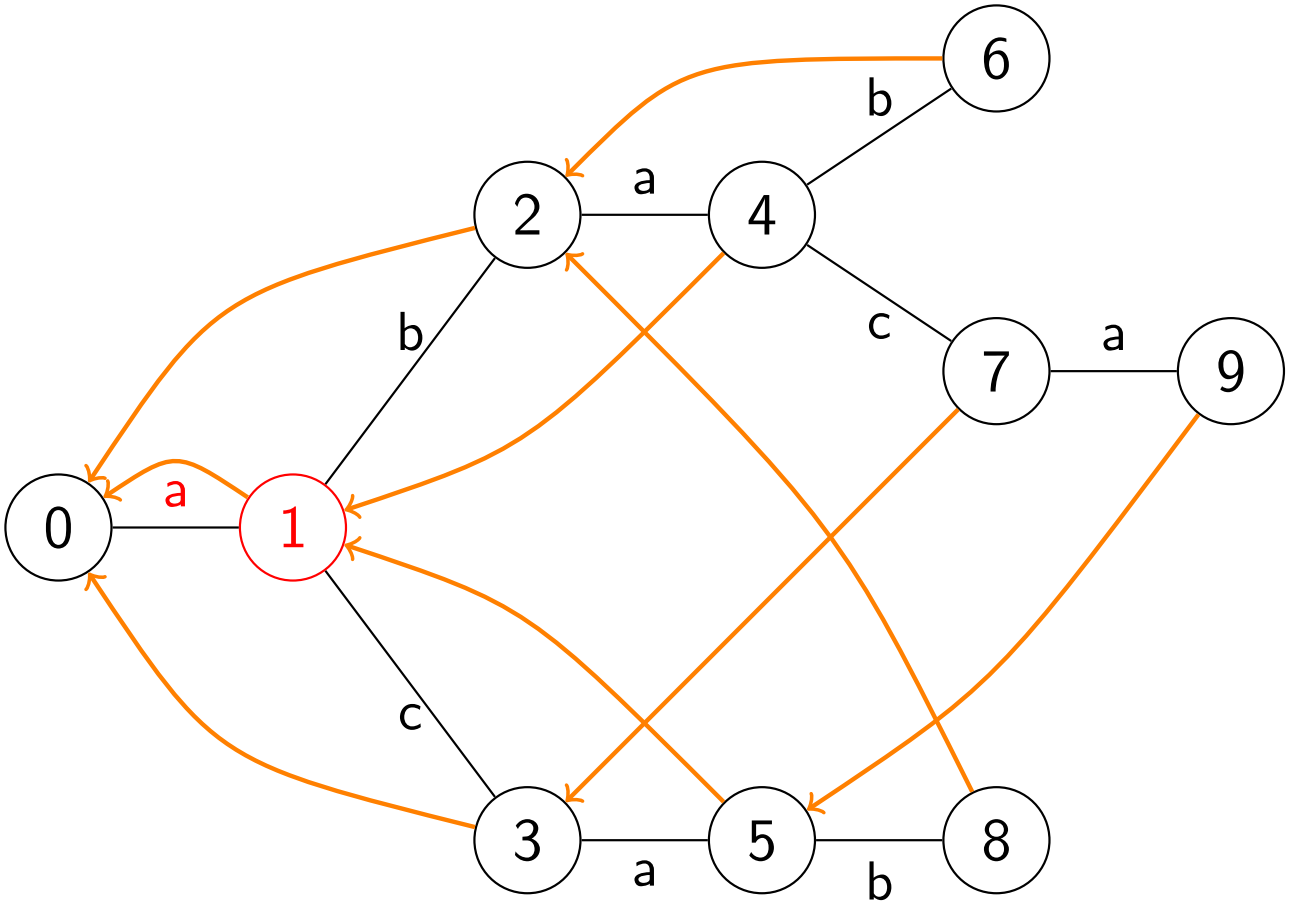
- 与 KMP 算法类似，我们可以设计失配边的计算算法
- 我们设 next[i] 表示 i 的失配边指向的结点，pre[i] 表示 i 在 Trie 上的父结点
- next[i] 可以通过 next[pre[i]] 推出，怎么做?
- 设从 pre[i] 到 i 的字符为 c，那么如果 c 是 next[pre[i]] 的后代边，next[i] 就是 next[pre[i]].son[c]
- 如果 c 不是 next[pre[i]] 的后代边，我们接着考虑 next[next[pre[i]]]，看 c 是否是 next[next[pre[i]]] 的后代边，此后的分析都是类似的
- 计算出失配边后，我们可以看看这个系统是如何匹配 ababacabaa 的

AC 自动机



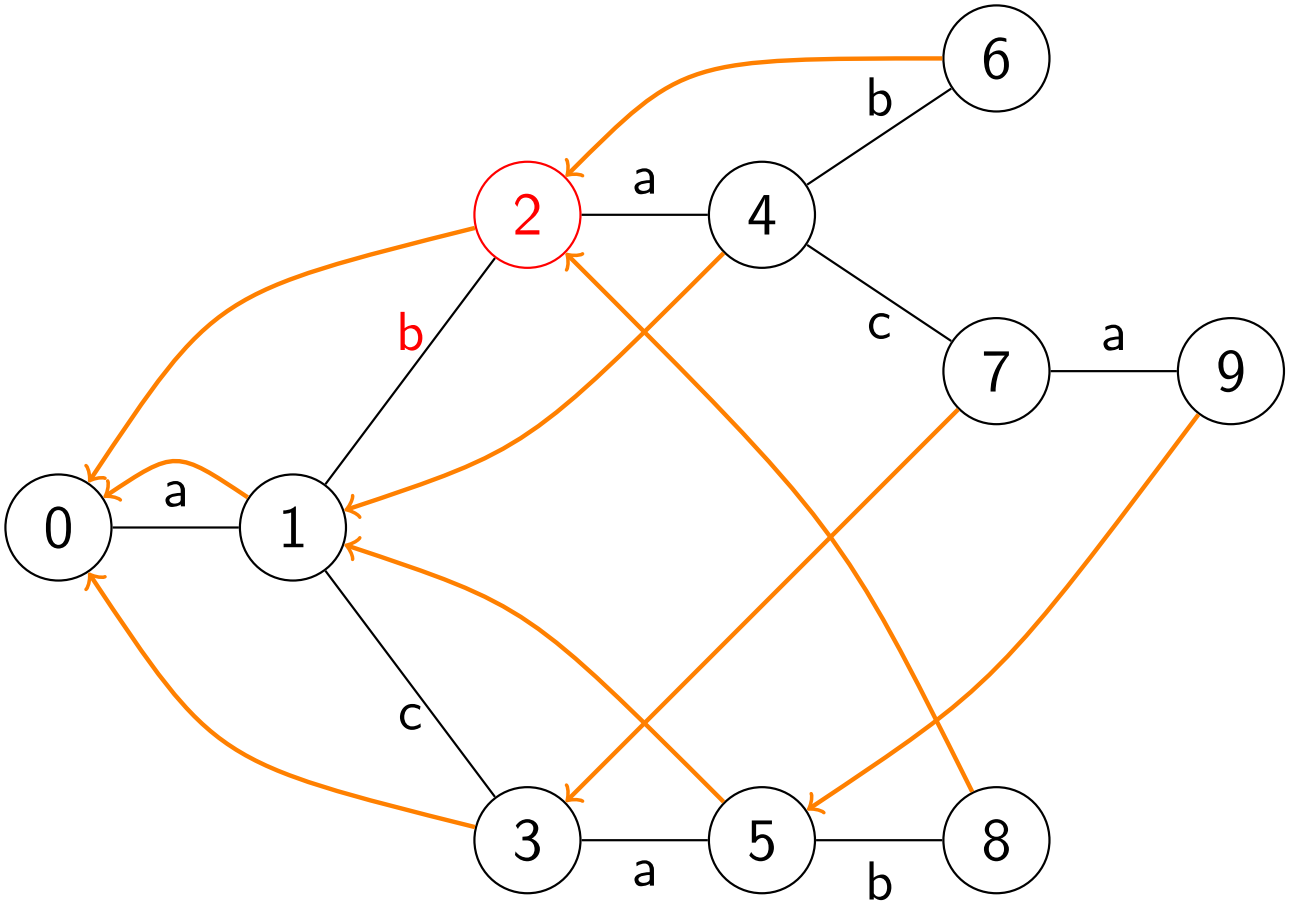
ababacabaa

AC 自动机



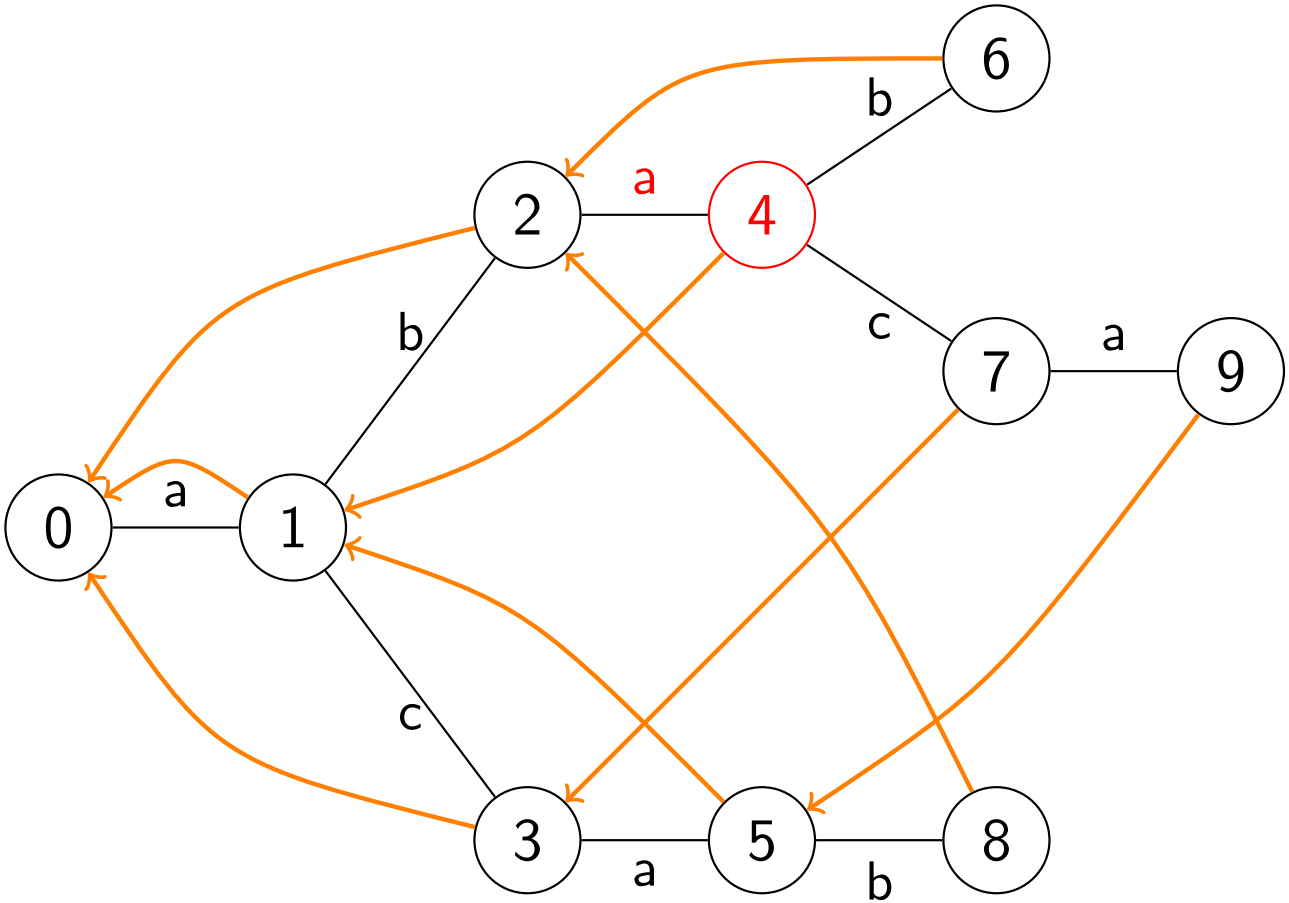
ababacabaa

AC 自动机



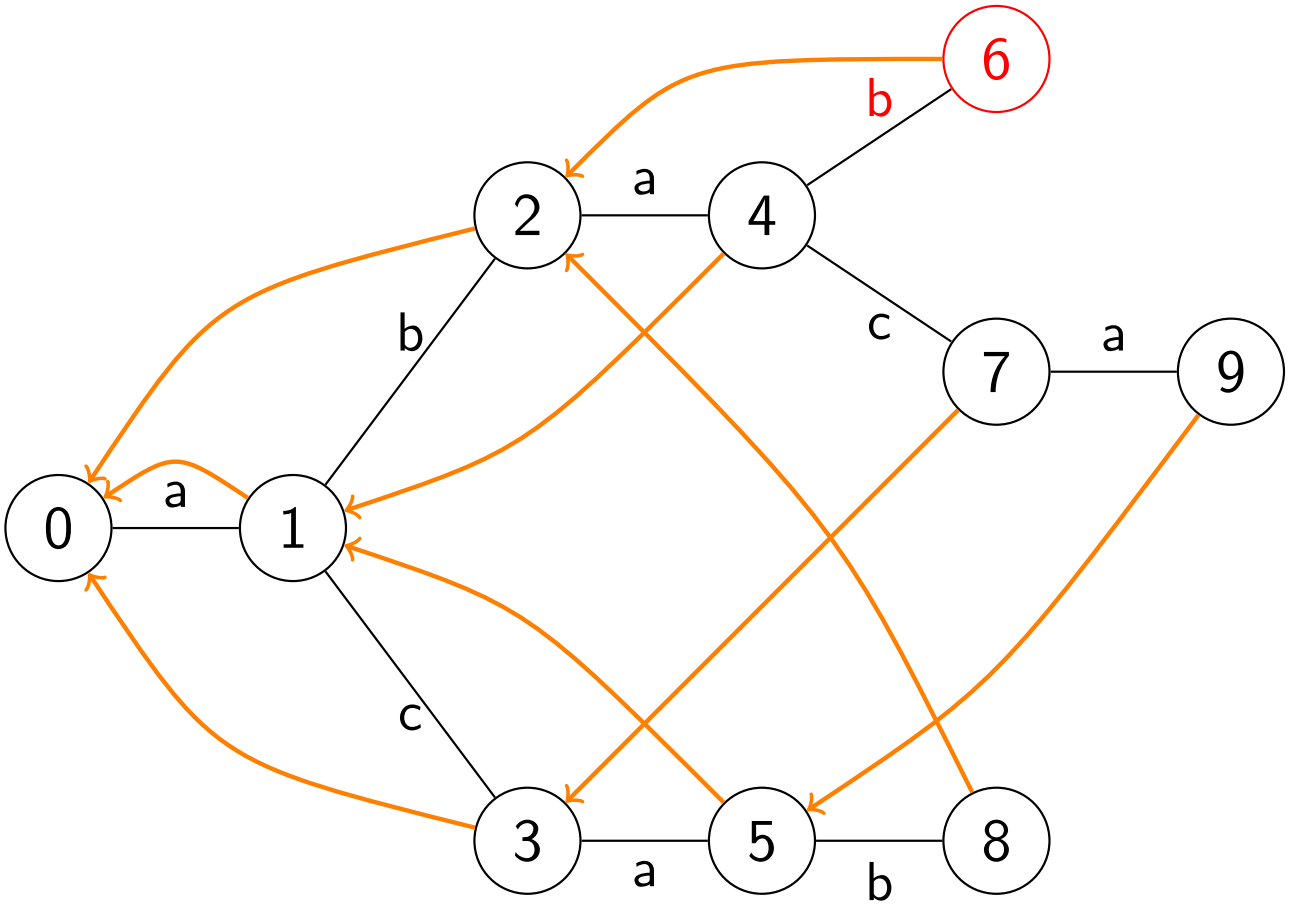
ababacabaa

AC 自动机



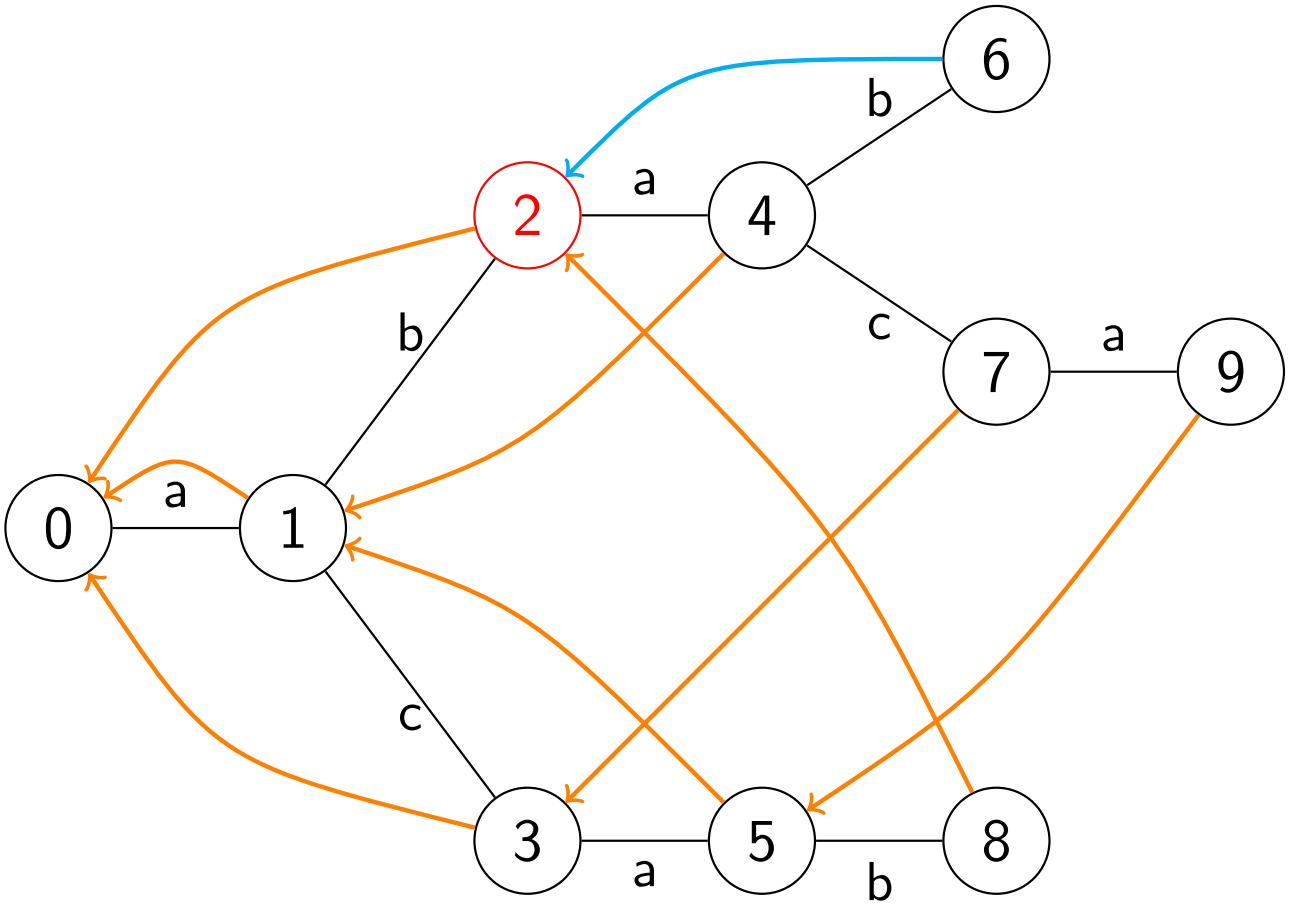
ababacabaa

AC 自动机



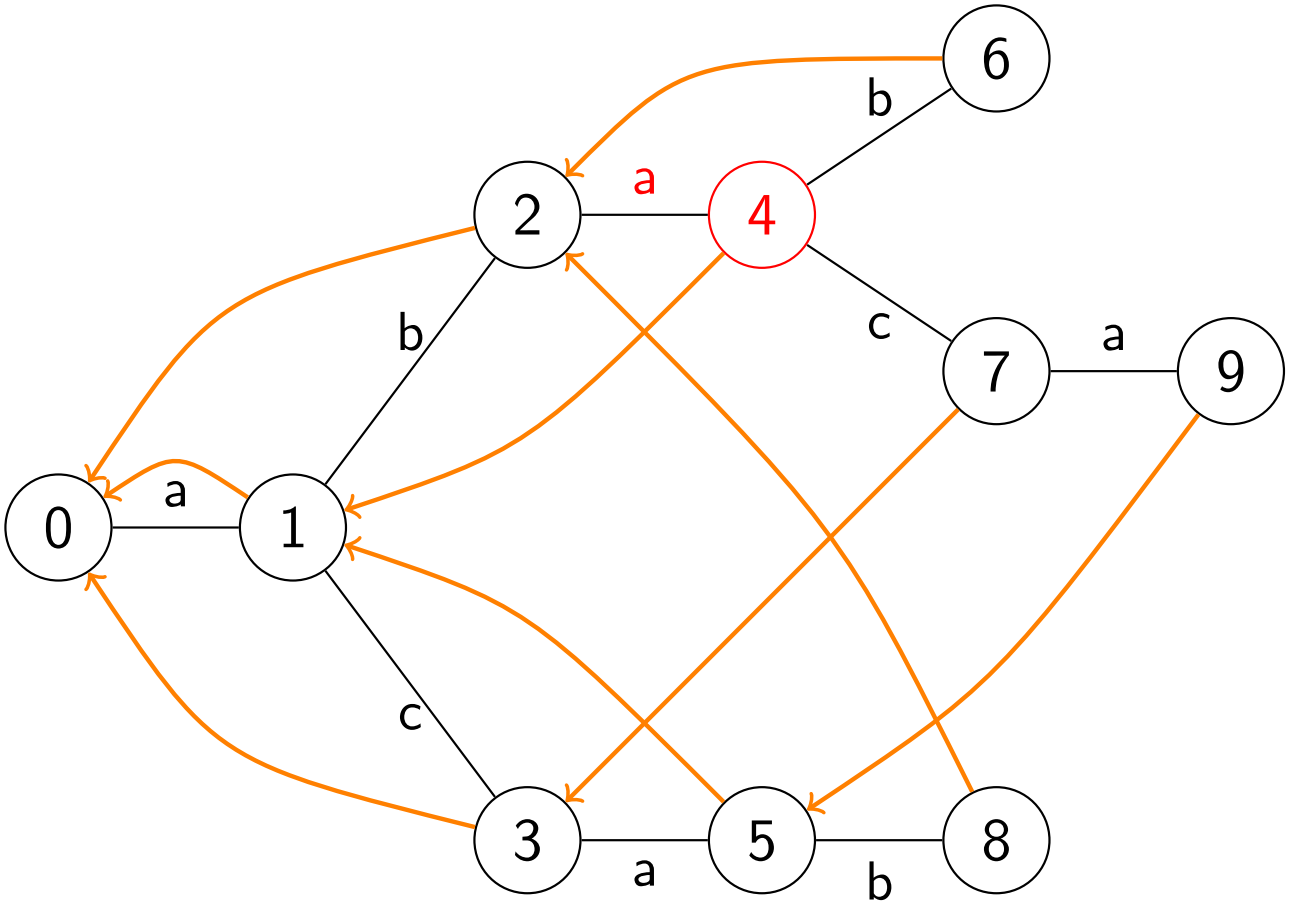
ababacabaa，匹配到一个单词，走一步失配边

AC 自动机



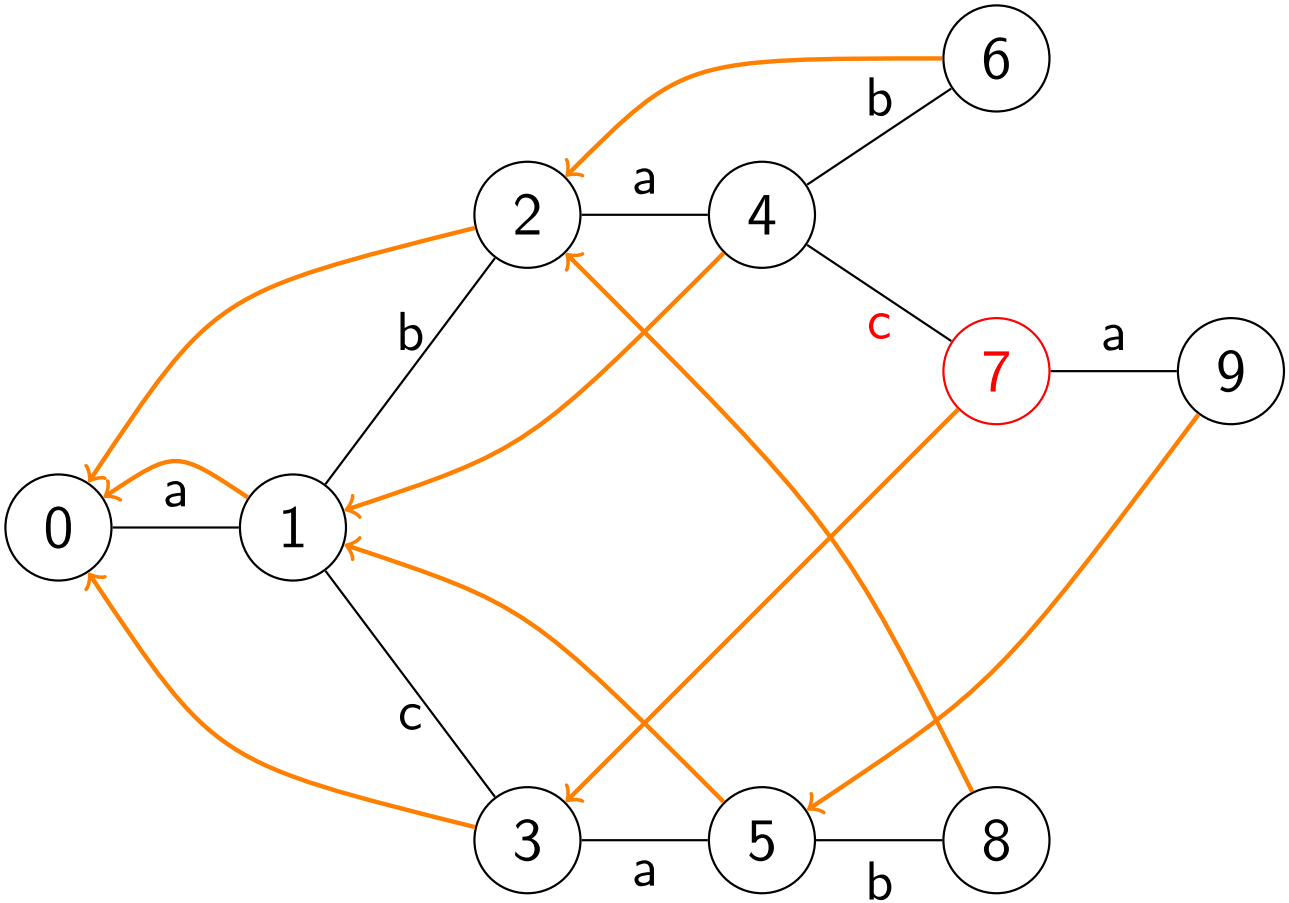
ababacabaa，继续匹配下一个字母

AC 自动机



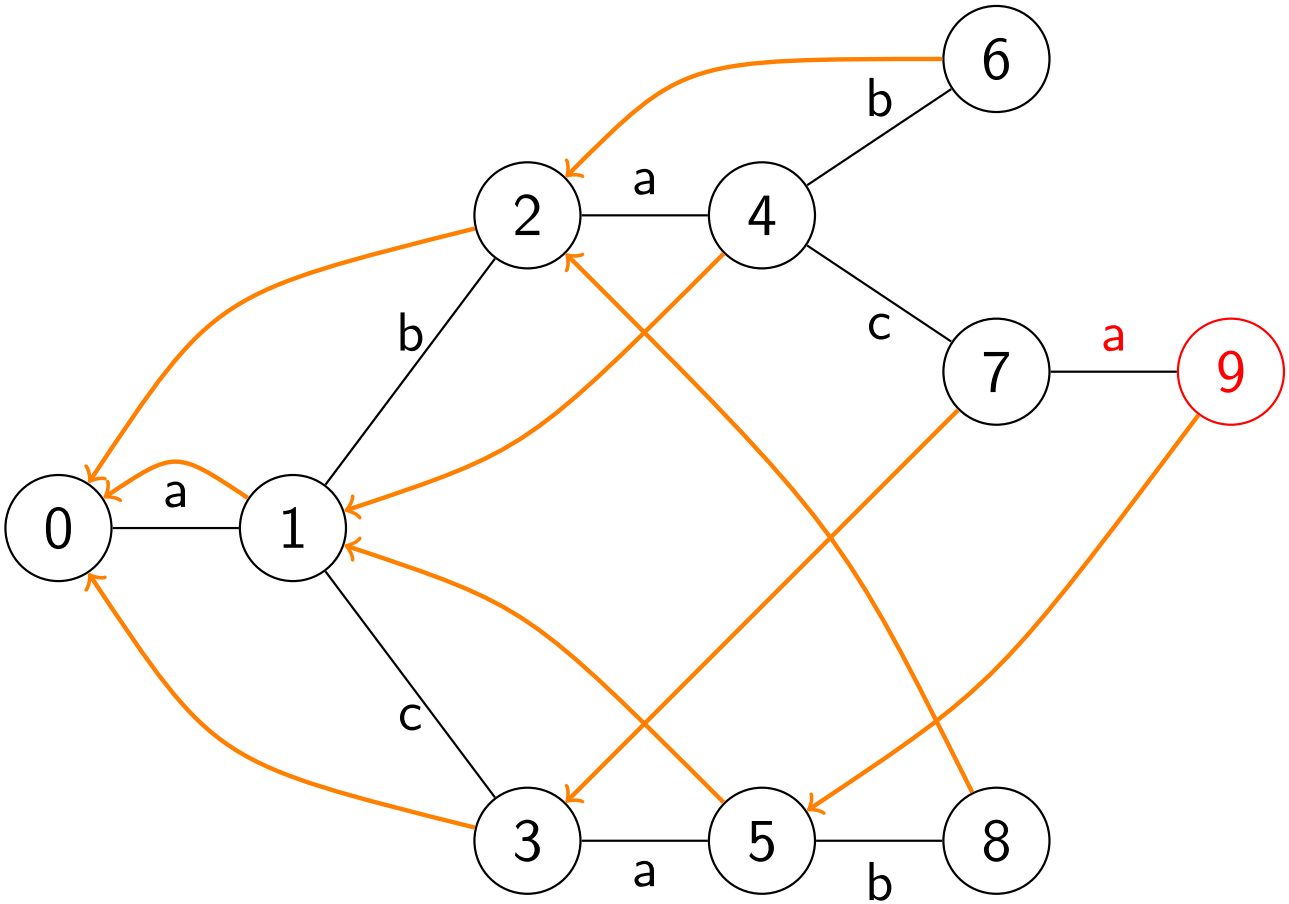
ababacabaa

AC 自动机



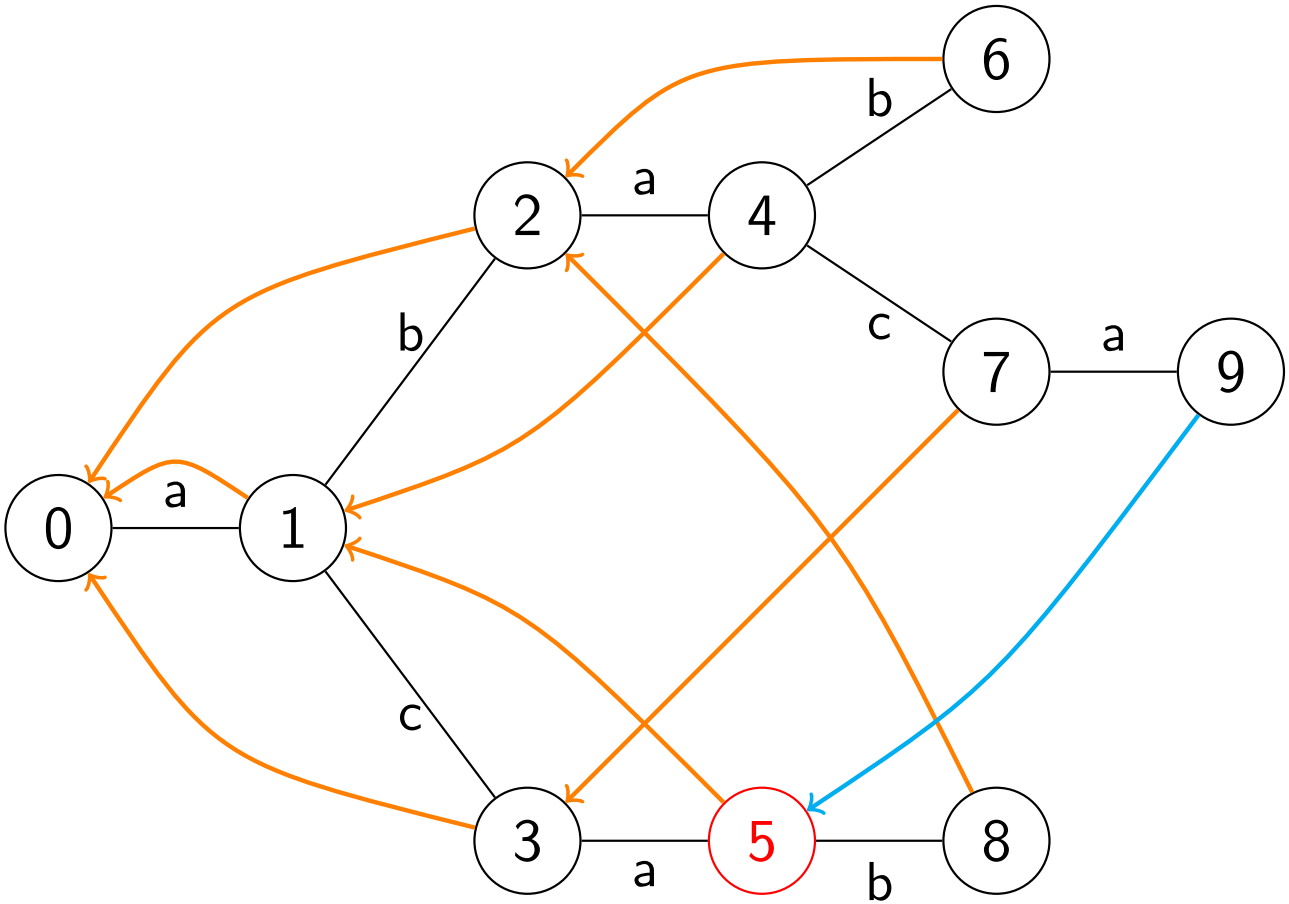
ababacabaa

AC 自动机



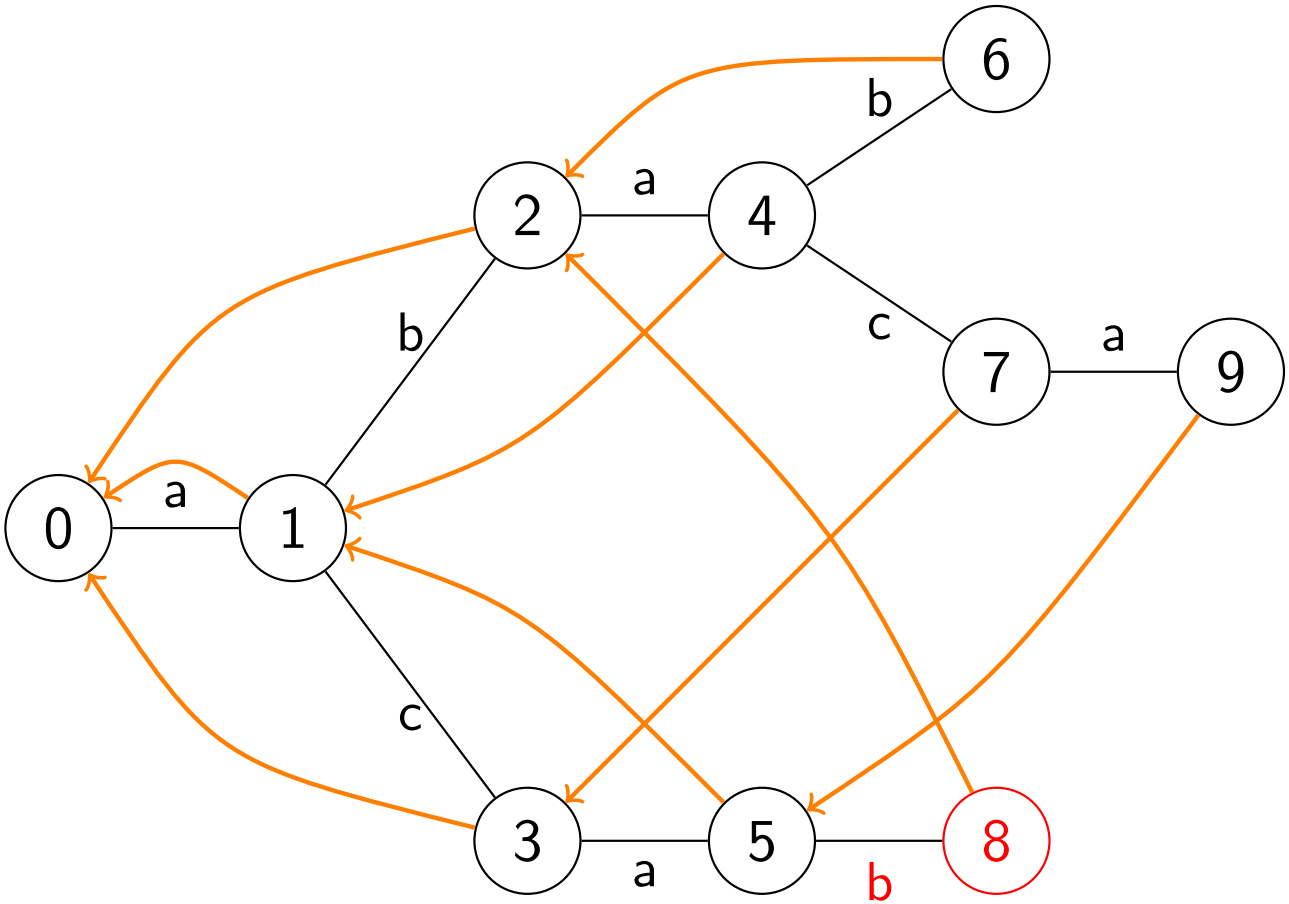
ababacabaa，匹配到一个单词，走一步失配边

AC 自动机



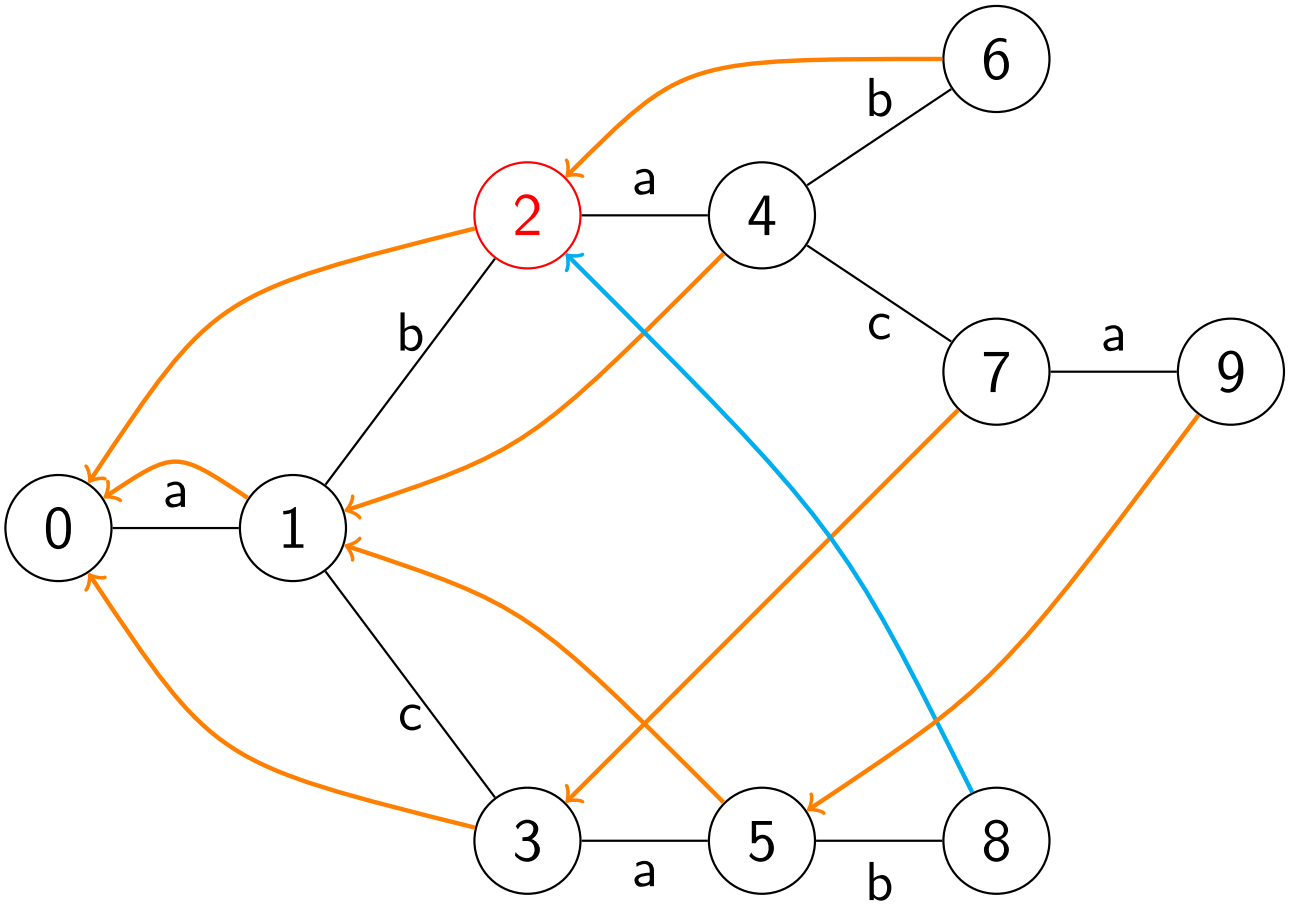
ababacabaa，匹配下一个字母

AC 自动机



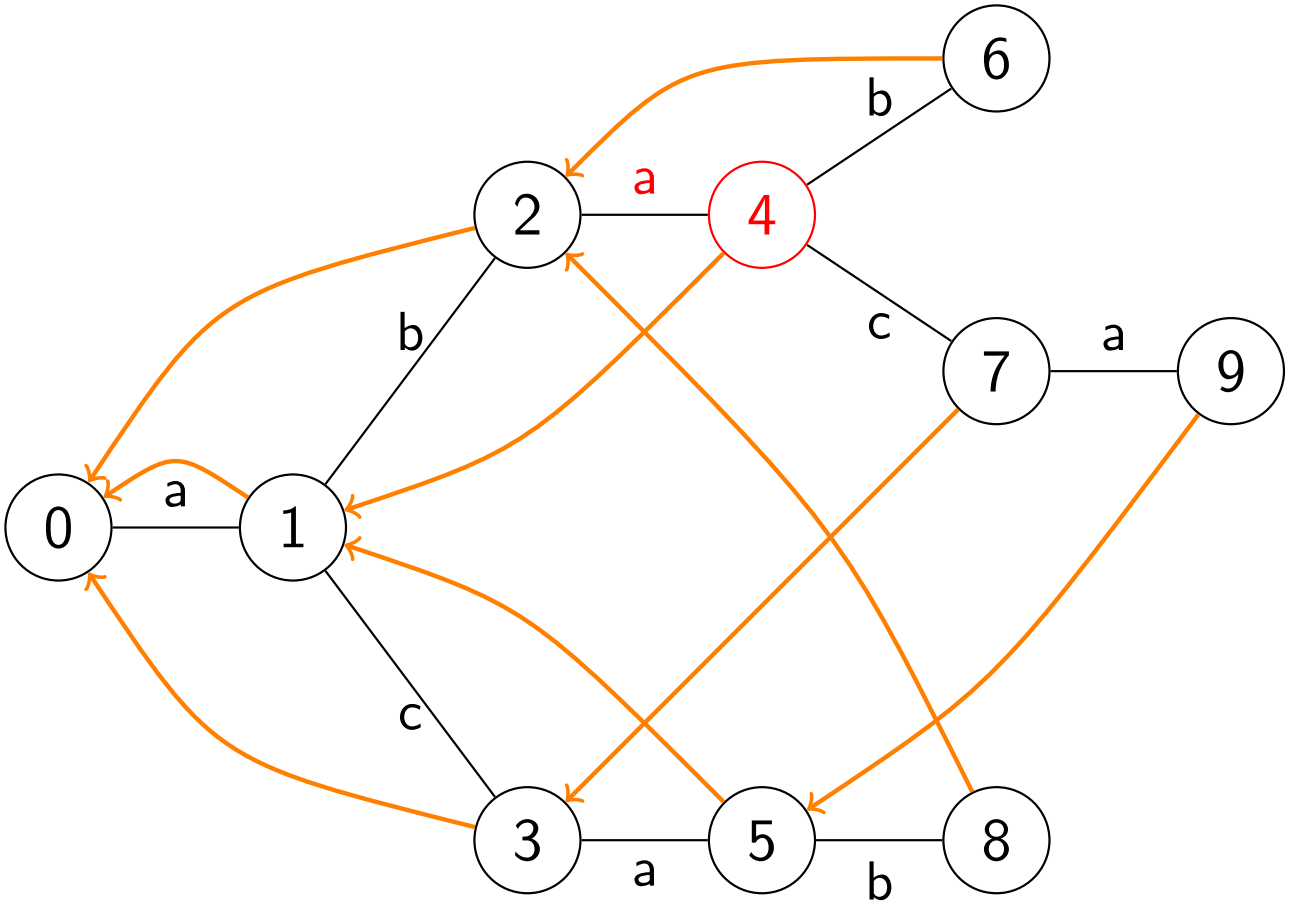
ababacabaa，匹配到单词，走一步失配边

AC 自动机



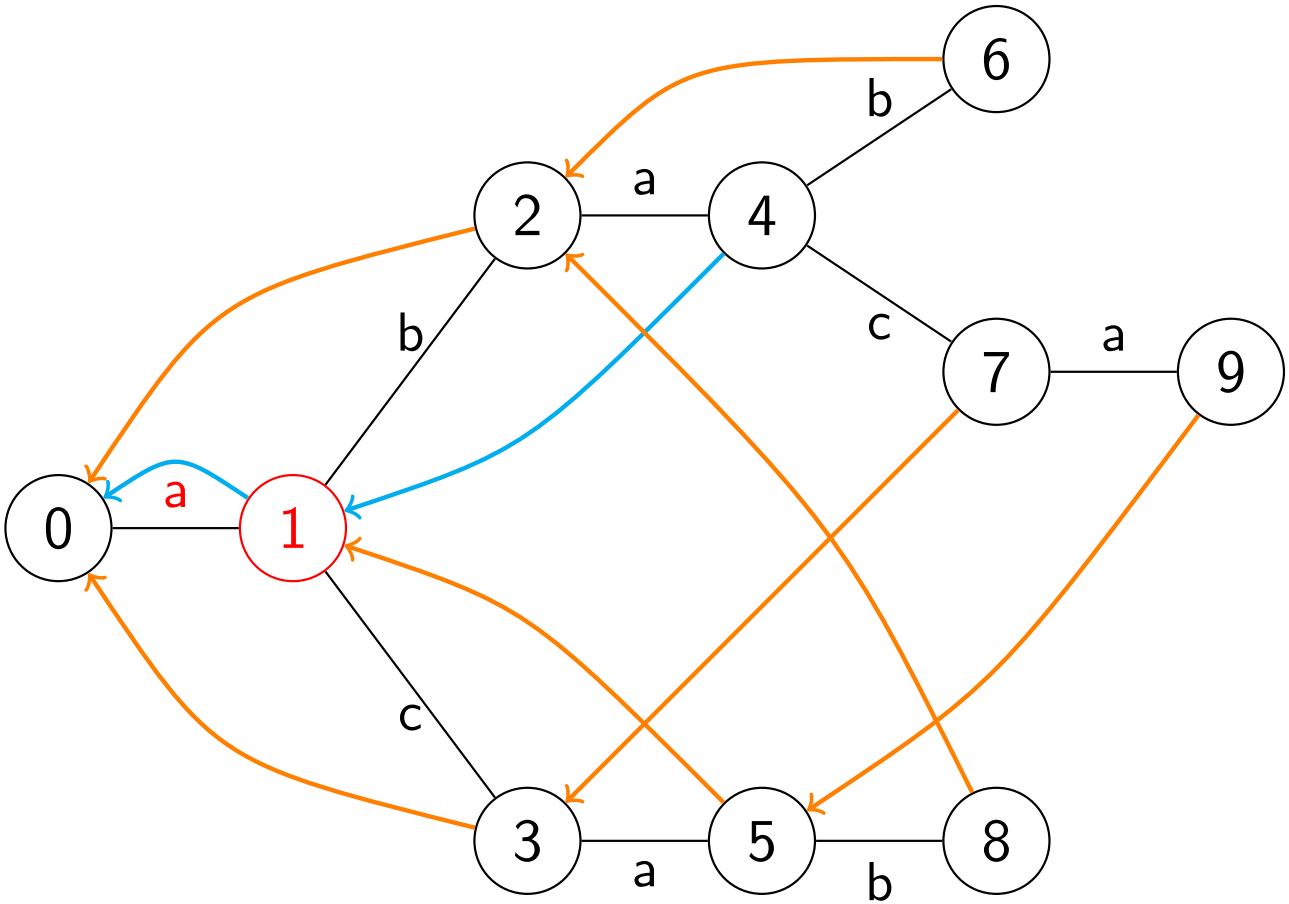
ababacabaa，继续匹配下一个字母

AC 自动机



ababacabaa

AC 自动机



ababacabaa，4 号结点之后没有匹配 a 字母的边，沿失配边往回找，直到 0 号节点，才有匹配 a 字母的边

AC 自动机

- 在 Trie 树上计算失配边后，我们可以以 O(N) 的复杂度找到每个匹配的字符串
- 你发现什么问题了吗?

AC 自动机

- 在 Trie 树上计算失配边后，我们可以以 O(N) 的复杂度找到每个匹配的字符串
- 你发现什么问题了吗?
- 这个算法仍然有瑕疵!
- 在上页 AC 自动机的基础上，假如我们还要找 ab 出现的次数，算法会怎么算?

AC 自动机

- 在 Trie 树上计算失配边后，我们可以以 O(N) 的复杂度找到每个匹配的字符串
- 你发现什么问题了吗?
- 这个算法仍然有瑕疵!
- 在上页 AC 自动机的基础上，假如我们还要找 ab 出现的次数，算法会怎么算?
- 假如我们的输入就是 abab，算法会先走到 2 号结点，找到一个 ab，再走到 6 号节点，找到一个 abab
- 第二个 ab 没有被直接找到!
- 更糟糕的是，如果输入是 acab，最后会走到 8 号结点，一个 ab 都没找到

AC 自动机

- 在 Trie 树上计算失配边后，我们可以以 O(N) 的复杂度找到每个匹配的字符串
- 你发现什么问题了吗?
- 这个算法仍然有瑕疵!
- 在上页 AC 自动机的基础上，假如我们还要找 ab 出现的次数，算法会怎么算?
- 假如我们的输入就是 abab，算法会先走到 2 号结点，找到一个 ab，再走到 6 号节点，找到一个 abab
- 第二个 ab 没有被直接找到!
- 更糟糕的是，如果输入是 acab，最后会走到 8 号结点，一个 ab 都没找到
- 从失配边的角度看，失配边也形成了一棵树
- ab 的出现次数，其实是表示 ab 的结点在失配边树上的后代的访问次数之和!

更多内容

- 有许多 OI 选手反映，学完 AC 自动机后才理解 KMP 算法，希望这部分内容能用来加深大家对 KMP 算法的理解
- 自动机有一套完整的理论，AC 自动机是其中最经典的模型之一
- 如果对相关理论在 OI 界的应用感兴趣，可以自学后缀自动机、回文自动机
- 在计算机科学领域，自动机理论是编译器的基础
- 因为这部分内容超出 NOIP 范围，所以不安排例题讲解，大家可以自行刷模板题

回文串问题

给定一个字符串，求以每个字符为中心的最长回文串长度

样例输入

abacabadaba

样例输出

1 3 1 7 1 3 1 7 1 3 1

回文串问题

给定一个字符串，求以每个字符为中心的最长回文串长度

样例输入

abacabadaba

样例输出

1 3 1 7 1 3 1 7 1 3 1

- 以字符为中心，只考虑了长度为奇数的回文串
- 长度为偶数的回文串，中心是两个相同的字符
- 如果我们遇到需要考虑偶数回文串的情况，要怎么考虑?

回文串问题

给定一个字符串，求以每个字符为中心的最长回文串长度

样例输入

abacabadaba

样例输出

1 3 1 7 1 3 1 7 1 3 1

- 以字符为中心，只考虑了长度为奇数的回文串
- 长度为偶数的回文串，中心是两个相同的字符
- 如果我们遇到需要考虑偶数回文串的情况，要怎么考虑?
- 把偶数回文串，转化为奇数回文串
- 做法是在相邻字符之间增加其他字符 #
- 以 # 为中心的回文串代表了偶数回文串
- 所以我们只需要考虑以字符为中心的奇数回文串问题

暴力算法

- 枚举每个字符，不断比较左右两边的字符，求得答案
- 最坏时间复杂度 $O(N^2)$
- 平均时间复杂度是多少?

暴力算法

- 枚举每个字符，不断比较左右两边的字符，求得答案
- 最坏时间复杂度 $O(N^2)$
- 平均时间复杂度是多少?
- 平均而言，以每个字符为中心，只会进行 $\frac{26}{25}$ 次比较
- 平均时间复杂度 O(N)

暴力算法

- 枚举每个字符，不断比较左右两边的字符，求得答案
- 最坏时间复杂度 O(N^2)
- 平均时间复杂度是多少?
- 平均而言，以每个字符为中心，只会进行 $\frac{26}{25}$ 次比较
- 平均时间复杂度 O(N)
- 很显然，我们可以用二分 + 字符串 Hash 优化这一算法，把最坏复杂度降低到 O(NlogN)，把平均复杂度上升到 O(NlogN)
- 这些暴力算法没有充分利用不同中心的回文串之间的关系

Manacher 算法

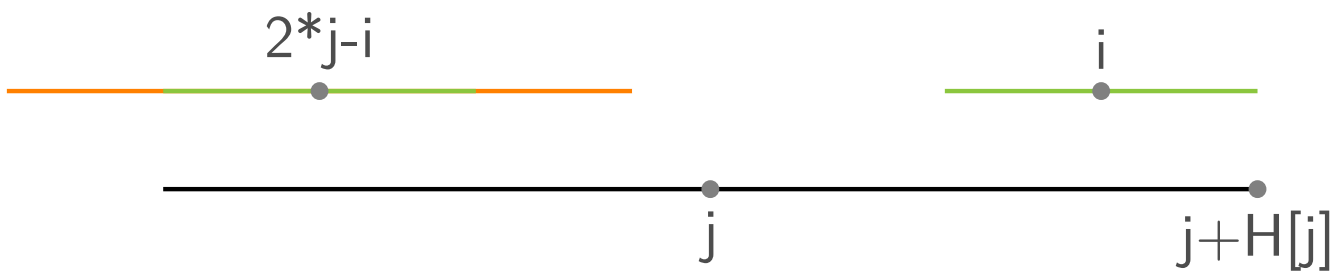
- Manacher 算法的作者叫做 Manacher，关于他的资料很少
- Manacher 算法是一个以 O(N) 的时空复杂度计算以每个字符为中心的最长回文串的算法，俗称马拉车
- 需要注意的是，Manacher 算法只是处理回文串问题的思路之一，有些问题可以使用字符串 Hash，而在更难的问题里还有回文自动机等工具

Manacher 算法

- Manacher 算法的作者叫做 Manacher，关于他的资料很少
- Manacher 算法是一个以 O(N) 的时空复杂度计算以每个字符为中心的最长回文串的算法，俗称马拉车
- 需要注意的是，Manacher 算法只是处理回文串问题的思路之一，有些问题可以使用字符串 Hash，而在更难的问题里还有回文自动机等工具
- 设以 i 字符为中心的最长回文串长度为 2*H[i]+1，H[i] 的意义是 i 字符往两边匹配的字符数量
- Manacher 算法的思路是，在计算 H[i] 时，用 $0 \leq j < i$ 中 j+H[j] 最大的回文串，即当前最右的回文串来节省计算

Manacher 算法

- 共有两种情况
- 1、如果 i>j+H[j]，直接以 i 为中心暴力计算，计算完后 i+H[i] 成为最右的回文串
- 2、如果 i<=j+H[j]，我们可以用下图来理解



Manacher 算法

- 我们再分两种情况
- 如果 i+H[2*j-i]<j+H[j]，即不存在橙色部分，且绿色部分严格在 j 的回文串内，那么 H[i]=H[2*j-i]
- 如果 i+H[2*j-i]≥j+H[j]，则 H[i] 至少为 j+H[j]-i，可以在这个基础上，暴力求解
- 算出的 i+H[i] 可以更新最右的回文串
- 这个算法的复杂度怎么分析?

Manacher 算法

- 我们再分两种情况
- 如果 i+H[2*j-i]<j+H[j]，即不存在橙色部分，且绿色部分严格在 j 的回文串内，那么 H[i]=H[2*j-i]
- 如果 i+H[2*j-i]≥j+H[j]，则 H[i] 至少为 j+H[j]-i，可以在这个基础上，暴力求解
- 算出的 i+H[i] 可以更新最右的回文串
- 这个算法的复杂度怎么分析?
- 只考虑最右的字符串，我们发现，每当我们暴力求解 H[i] 时，最右的字符串都相应地往右移动
- 因为最右的字符串最多移动 N 次，所以暴力的执行次数最多为 N 次
- 总时间复杂度 O(N)

Manacher 算法

```
int j=-1,right=-1;
for(int i=0;i<n;i++){
    if(i>right)H[i]=0;
    else if(i+H[2*j-i]<right)H[i]=H[2*j-i];
    else H[i]=right-i;
    while(i-H[i]>0&&i+H[i]+1<n
            &&x[i+H[i]+1]==x[i-H[i]-1])
        H[i]++;
    if(i+H[i]>right){
        j=i;
        right=i+H[i];
    }
}
```

计算代码如上所示

例题讲解-BZOJ2160 拉拉队排练

题目大意：给一个长度为 n 的字符串和 k，求子串中前 k 长的奇数回文串的长度之积。答案模 19930726，不够 k 个答案为-1

$n \leq 1,000,000 \quad k \leq 1,000,000,000,000$

样例输入

5 3
ababa

样例输出

45

例题讲解-BZOJ2160 拉拉队排练

题目大意：给一个长度为 n 的字符串和 k，求子串中前 k 长的奇数回文串的长度之积。答案模 19930726，不够 k 个答案为-1

$n \leq 1,000,000 \quad k \leq 1,000,000,000,000$

样例输入

5 3
ababa

样例输出

45

- 机智的同学可能已经发现了，这题的重点不是 Manacher 算法本身

- 很显然，我们能先使用 Manacher 算法算出以每个字符为中心的最长回文串
- 以 i 为中心的回文串有 $[i - H[i], i + H[i]]$，$[i - H[i] + 1, i + H[i] - 1]$,$[i - H[i] + 2, i + H[i] - 2]$,...,$[i, i]$，长度分别为 $2H[i] + 1, 2H[i] - 1, 2H[i] - 3, ..., 1$
- 如果我们能有一种高效的方法，算出所有 i 的所有区间中前 k 长区间长度之积，就能解决这个问题了

- 很显然，我们能先使用 Manacher 算法算出以每个字符为中心的最长回文串
- 以 i 为中心的回文串有 $[i-H[i], i+H[i]]$，$[i-H[i]+1, i+H[i]-1]$,$[i-H[i]+2, i+H[i]-2]$,...,$[i, i]$，长度分别为 $2H[i]+1, 2H[i]-1, 2H[i]-3, ..., 1$
- 如果我们能有一种高效的方法，算出所有 i 的所有区间中前 k 长区间长度之积，就能解决这个问题了
- 解法一：二分第 k 长区间长度，每次遍历计算数量，最后遍历 H 数组计算答案，以每个字符为中心的答案是一个双阶乘除以双阶乘的形式，需要计算奇数的前缀积与奇数逆元的前缀积，时间复杂度 O(NlogK)

- 很显然，我们能先使用 Manacher 算法算出以每个字符为中心的最长回文串
- 以 i 为中心的回文串有 $[i-H[i], i+H[i]]$，$[i-H[i]+1, i+H[i]-1]$,$[i-H[i]+2, i+H[i]-2]$,...,$[i, i]$，长度分别为 $2H[i]+1, 2H[i]-1, 2H[i]-3, ..., 1$
- 如果我们能有一种高效的方法，算出所有 i 的所有区间中前 k 长区间长度之积，就能解决这个问题了
- 解法一：二分第 k 长区间长度，每次遍历计算数量，最后遍历 H 数组计算答案，以每个字符为中心的答案是一个双阶乘除以双阶乘的形式，需要计算奇数的前缀积与奇数逆元的前缀积，时间复杂度 O(NlogK)
- 解法二：统计各长度最长回文串的数量，求一个后缀和，变成各长度回文串数量，再贪心计算即可，使用快速幂计算，时间复杂度 $O(Nlog\frac{K}{N})$

- 很显然，我们能先使用 Manacher 算法算出以每个字符为中心的最长回文串
- 以 i 为中心的回文串有 $[i-H[i], i+H[i]]$，$[i-H[i]+1, i+H[i]-1]$,$[i-H[i]+2, i+H[i]-2]$,...,$[i, i]$，长度分别为 $2H[i]+1, 2H[i]-1, 2H[i]-3, ..., 1$
- 如果我们能有一种高效的方法，算出所有 i 的所有区间中前 k 长区间长度之积，就能解决这个问题了
- 解法一：二分第 k 长区间长度，每次遍历计算数量，最后遍历 H 数组计算答案，以每个字符为中心的答案是一个双阶乘除以双阶乘的形式，需要计算奇数的前缀积与奇数逆元的前缀积，时间复杂度 O(NlogK)
- 解法二：统计各长度最长回文串的数量，求一个后缀和，变成各长度回文串数量，再贪心计算即可，使用快速幂计算，时间复杂度 $O(Nlog\frac{K}{N})$
- 解法三：结合解法一和解法二，使用解法二的思路求出第 k 长区间长度，然后使用解法一的方法遍历 H 数组计算答案，时间复杂度 O(N)

例题讲解-SHOI2011 双倍回文

- 题目大意：设字符串 w 的反转为 w^R，定义双倍回文串为形如 ww^Rww^R 的字符串，对给定字符串，求最长双倍回文子串的长度

样例输入

16
ggabaabaabaaball

样例输出

12

对样例输入，最长的双倍回文子串是 abaabaabaaba

例题讲解-SHOI2011 双倍回文

- 注意到，本题只用考虑偶数回文串
- 设 H[i] 表示以第 i 个字符和第 i+1 个字符为中心的最长回文串半径（长度/2），问题能转化成什么样?

例题讲解-SHOI2011 双倍回文

- 注意到，本题只用考虑偶数回文串
- 设 H[i] 表示以第 i 个字符和第 i+1 个字符为中心的最长回文串半径（长度/2），问题能转化成什么样?
- 双倍回文串可以用两个字符 i，j 确定，其中 i 是双倍回文串中心，j 是左半回文串中心；对任给的 i 和 j，需要满足什么条件，才是双倍回文串?

例题讲解-SHOI2011 双倍回文

- 注意到，本题只用考虑偶数回文串
- 设 H[i] 表示以第 i 个字符和第 i+1 个字符为中心的最长回文串半径（长度/2），问题能转化成什么样?
- 双倍回文串可以用两个字符 i，j 确定，其中 i 是双倍回文串中心，j 是左半回文串中心；对任给的 i 和 j，需要满足什么条件，才是双倍回文串?
- 条件是 j+H[j]+1>=i 且 j>=i-H[i]/2，固定 i 时，j 越小，双倍回文串越长

例题讲解-SHOI2011 双倍回文

- 注意到，本题只用考虑偶数回文串
- 设 H[i] 表示以第 i 个字符和第 i+1 个字符为中心的最长回文串半径（长度/2），问题能转化成什么样?
- 双倍回文串可以用两个字符 i，j 确定，其中 i 是双倍回文串中心，j 是左半回文串中心；对任给的 i 和 j，需要满足什么条件，才是双倍回文串?
- 条件是 j+H[j]+1>=i 且 j>=i-H[i]/2，固定 i 时，j 越小，双倍回文串越长
- 算法一：将 j+H[j]+1 排序，循环 i，维护存储 j 的 set，确保当遍历到 i 时，所有满足 j+H[j]+1>=i 的 j 都在 set 中，我们只需要找最小的满足 j>=i-H[i]/2 的 j 即可，时间复杂度 O(NlogN)

例题讲解-SHOI2011 双倍回文

- 注意到，本题只用考虑偶数回文串
- 设 H[i] 表示以第 i 个字符和第 i+1 个字符为中心的最长回文串半径（长度/2），问题能转化成什么样?
- 双倍回文串可以用两个字符 i，j 确定，其中 i 是双倍回文串中心，j 是左半回文串中心；对任给的 i 和 j，需要满足什么条件，才是双倍回文串?
- 条件是 j+H[j]+1>=i 且 j>=i-H[i]/2，固定 i 时，j 越小，双倍回文串越长
- 算法一：将 j+H[j]+1 排序，循环 i，维护存储 j 的 set，确保当遍历到 i 时，所有满足 j+H[j]+1>=i 的 j 都在 set 中，我们只需要找最小的满足 j>=i-H[i]/2 的 j 即可，时间复杂度 O(NlogN)
- 算法二：固定 j 时，i 越大，双倍回文串越长，我们也可以反过来设计类似算法一的做法

回文树

问题：给出一个长度为 n 的字符串 S，有 q 组询问，每组询问给出一个回文串 a，求 a 是否是 S 的子串

- 一个直接的想法是，枚举 a 在 S 中的位置，采用字符串 Hash 的方法比较，时间复杂度为 O(nq)
- 如何利用回文串的性质?

回文树

问题：给出一个长度为 n 的字符串 S，有 q 组询问，每组询问给出一个回文串 a，求 a 是否是 S 的子串

- 一个直接的想法是，枚举 a 在 S 中的位置，采用字符串 Hash 的方法比较，时间复杂度为 O(nq)
- 如何利用回文串的性质?
- 一个直接的想法是，用 Manacher 算法预处理以每个字符（包括 #）为中心的最长回文串长度，然后把这个最长回文串的后半部分插入 Trie 树中
- 在查询时，只要把询问的后半拿出，在 Trie 树上查询即可
- 时间复杂度 $O(n^2 + q|a|)$，还能优化吗

回文树

- 观察到，这样的 Trie 树分为两块——奇数回文串部分和偶数回文串部分，整个树被称为回文树
- 统计回文树中的结点数 m，发现总是 $m \leq n$，这是有什么性质吗?
- 回文树中的结点数等于互不相同的回文子串数量

回文树

- 观察到，这样的 Trie 树分为两块——奇数回文串部分和偶数回文串部分，整个树被称为回文树
- 统计回文树中的结点数 m，发现总是 $m \leq n$，这是有什么性质吗?
- 回文树中的结点数等于互不相同的回文子串数量
- 对于一个字符串 S，在 S 之后增加一个字符 c，它的回文树最多增加一个字符，即最多增加一个不同的回文子串
- 增加的回文子串一定以 c 结尾，考虑所有以 c 为结尾的回文子串，设其中最长的为 S_0，其他为 $S_1, S_2, \ldots$
- S_1 是 S_0 的后缀；而因为 S_0 是回文串，所以 S_1 还是 S_0 的前缀。所以除 S_0 外，其他回文串都一定在回文树里出现了

回文树

- 在 Manacher 算法的计算过程中，回文树就能构建出来
- 和倍增类似，维护回文树每个结点的第 2^i 个祖先，那么可以 O(logn) 地找到每个结点的第 k 个祖先
- 同时，维护以每个字符为中点的最长回文串在回文树中的位置
- Manacher 计算 H[i] 时，可能从左方对称过来，之后可能会往两边扩展

回文树

- 在 Manacher 算法的计算过程中，回文树就能构建出来
- 和倍增类似，维护回文树每个结点的第 2^i 个祖先，那么可以 O(logn) 地找到每个结点的第 k 个祖先
- 同时，维护以每个字符为中点的最长回文串在回文树中的位置
- Manacher 计算 H[i] 时，可能从左方对称过来，之后可能会往两边扩展
- 从左方对称过来时，一定是左边的最长回文串的“同心子串”，“同心子串”在回文树中的位置可以用倍增算出，设为以 i 为中心的最长回文串所在位置
- 在往两边扩展时，需要从当前最长回文串所在位置出发，往回文树的下方走，过程中可能要新建结点
- 时间和空间复杂度都是 O(nlogn)
- 回文树还有基于回文自动机的构建做法，复杂度为 O(n)，感兴趣的同学可以自行学习

字符串算法总结

- 几乎所有字符串算法，都有字符串 Hash 算法的替代品
- 几乎所有字符串算法，都需要考虑最坏复杂度
- AC 自动机是 Trie 树和 KMP 算法的结合体
- Manacher 算法、回文树是处理回文串问题的基础算法
- 虽然字符串算法在 NOIP 涉及较少，但是字符串算法中的技巧、思想是通用的

Thanks for listening

Any Questions?