字符串

黎伟诺

7.18.2024

哈希（Hash）

简介

将一个把字符串（树/图/集合/...）映射到整数的函数 f，这个 f 称为是 Hash 函数。
我们希望这个函数 f 可以方便地帮我们判断两个字符串是否相等。
Hash 的核心思想在于，将输入映射到一个值域较小、可以方便比较的范围。

字符串哈希

简介

具体来说，哈希函数最重要的性质可以概括为下面两条：
在 Hash 函数值不一样的时候，两个字符串一定不一样；
在 Hash 函数值一样的时候，两个字符串不一定一样（但有大概率一样，且我们当然希望它们总是一样的）。
我们将 Hash 函数值一样但原字符串不一样的现象称为哈希碰撞。

通常来说，我们会采用这样的正向多项式 hash ：
$f(s) = \sum_{i=1}^{l} s[i] \times B^{l-i} \pmod{M}$
或者这样的反向多项式 hash ：
$f(s) = \sum_{i=1}^{l} s[i] \times B^{i-1} \pmod{M}$

字符串哈希

简介

如何求出子串（区间）的哈希值？

先计算前缀和，$f(s[1\ldots i]) = f(s[1\ldots i-1]) * B + s[i] \pmod M$

那么 $f(s[l\ldots r]) = f(s[1\ldots r]) - f(s[1\ldots l-1]) \times B^{r-l+1} \pmod M$

B 的次幂可以预处理

字符串哈希

简介

如何选取模数 M 和基底 B

一般 M 需要为质数，两个哈希值冲突的概率约为 $O(\frac{1}{M})$

常见的质数 $10^9+7, 10^9+9, 998244353, 993244853, 19260817$ 尽量不要选用（容易被出题人直接针对卡）

准备一些自己的质数（比如自己的生日之类的）

基底 B 只要比字符集 $size$ 大就行，可以选 $97, 101$ 等

字符串哈希

简介

假如我们希望子串的哈希值两两不同，有什么办法？
子串数量是 $O(n^2)$ 的，冲突概率是 $\sum_{i=1}^{n^2} \frac{M-i+1}{M}$
由生日悖论，我们希望 $n^2 \leq \sqrt{M}$
也就是 $n = 10^5$ 时，需要 $M > 10^{20}$，这时候需要双模数甚至三模数
在 codeforces 这种在线可 hack 比赛中，使用哈希是一种相当危险的行为，通常你需要使用不固定双/三模数、不固定基底才能避免被 hack/fail system test。

字符串匹配

求出模式串的哈希值后，求出文本串每个长度为模式串长度的子串的哈希值，分别与模式串的哈希值比较即可。

字符串排序

难点在快速字符串比较
我们可以二分出最后一个前缀哈希值相等的位置，看下一个字符的大小

允许 k 次失配的字符串匹配

问题：给定长为 n 的源串 s，以及长度为 m 的模式串 p，要求查找源串中有多少子串与模式串匹配。s' 与 s 匹配，当且仅当 s' 与 s 长度相同，且最多有 k 个位置字符不同。其中 $1 \leq n, m \leq 10^6$，$0 \leq k \leq 5$。

允许 k 次失配的字符串匹配

枚举所有可能匹配的子串，假设现在枚举的子串为 s'
通过哈希加二分可以快速找到 s' 与 p 第一个不同的位置。
之后将 s' 与 p 在这个失配位置及之前的部分删除掉，继续查找下一个失配位置。这样的过程最多发生 k 次。
总的时间复杂度为 $O(m + kn \log m)$。

最长回文子串

枚举中心位置，二分 + 哈希查出两边最长延伸长度

哈希表

把 key 映射到 value 的数据结构，可以把哈希表理解为一种高级的数组，这种数组的下标可以是很大的整数，浮点数，字符串甚至结构体。
通常是利用哈希函数映射到 $[0, M-1]$ 的范围中
这需要 M 在 10^8 以内

哈希表

如何解决哈希冲突？
开散列法（open hashing）：每个存放数据的地方开一个链表，如果有多个键值索引到同一个地方，只用把他们都放到那个位置的链表里就行了。查询的时候需要把对应位置的链表整个扫一遍，对其中的每个数据比较其键值与查询的键值是否一致。
闭散列法：闭散列方法把所有记录直接存储在散列表中，如果发生冲突则根据某种方式继续进行探查。
比如线性探查法：如果在 d 处发生冲突，就依次检查 $d+1$，$d+2$
C++ 有对应的封装数据结构：unordered_map<type1,type2>，速度会稍微慢一点

Fixing a Binary String

CF 1979D

D. Fixing a Binary String

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a binary string s of length n, consisting of zeros and ones. You can perform the following operation **exactly once**:

1. Choose an integer p ($1 \le p \le n$).
2. Reverse the substring $s_1 s_2 \ldots s_p$. After this step, the string $s_1 s_2 \ldots s_n$ will become $s_p s_{p-1} \ldots s_1 s_{p+1} s_{p+2} \ldots s_n$.
3. Then, perform a cyclic shift of the string s to the left p times. After this step, the initial string $s_1 s_2 \ldots s_n$ will become $s_{p+1} s_{p+2} \ldots s_n s_p s_{p-1} \ldots s_1$.

For example, if you apply the operation to the string `110001100110` with $p = 3$, after the second step, the string will become **`011`**`001100110`, and after the third step, it will become `001100110`**`011`**.

A string s is called *k-proper* if two conditions are met:

- $s_1 = s_2 = \ldots = s_k$;
- $s_{i+k} \neq s_i$ for any i ($1 \le i \le n - k$).

For example, with $k = 3$, the strings `000`, `111000111`, and `111000` are k-proper, while the strings `000000`, `001100`, and `1110000` are not.

You are given an integer k, which **is a divisor** of n. Find an integer p ($1 \le p \le n$) such that after performing the operation, the string s becomes k-proper, or determine that it is impossible. Note that if the string is initially k-proper, you still need to apply exactly one operation to it.

Fixing a Binary String

CF 1979D

只需要维护正向哈希和反向哈希
目标串就两种。操作之后的串可以由正向哈希和反向哈希的结果拼起来
不过这里是 Codeforces，各位也可以想想非哈希的做法

Division + LCP (easy version)

CF 1968G1

This is the easy version of the problem. In this version $l = r$.

You are given a string s. For a fixed k, consider a division of s into exactly k continuous substrings $w_1, \ldots, w_k$. Let f_k be the maximal possible $LCP(w_1, \ldots, w_k)$ among all divisions.

$LCP(w_1, \ldots, w_m)$ is the length of the Longest Common Prefix of the strings $w_1, \ldots, w_m$.

For example, if $s = abababcab$ and $k = 4$, a possible division is $ab|ab|abc|ab$. The $LCP(ab, ab, abc, ab)$ is 2, since ab is the Longest Common Prefix of those four strings. Note that each substring consists of a continuous segment of characters and each character belongs to **exactly** one substring.

Your task is to find $f_l, f_{l+1}, \ldots, f_r$. **In this version $l = r$.**

$n \leq 2 \times 10^5$，时限 2 s

Division + LCP (easy version)

CF 1968G1

二分答案 $\geq k$，相当于判断长度为 k 的前缀是否能在整个字符串里不重叠的出现 l 次

时间复杂度是 $O(n \log n)$。

Division + LCP (hard version)

CF 1968G2

This is the hard version of the problem. In this version $l \leq r$.

You are given a string s. For a fixed k, consider a division of s into exactly k continuous substrings $w_1, \ldots, w_k$. Let f_k be the maximal possible $LCP(w_1, \ldots, w_k)$ among all divisions.

$LCP(w_1, \ldots, w_m)$ is the length of the Longest Common Prefix of the strings $w_1, \ldots, w_m$.

For example, if $s = abababcab$ and $k = 4$, a possible division is $ababa|bcab$. The $LCP(ab, ab, abc, ab)$ is 2, since ab is the Longest Common Prefix of those four strings. Note that each substring consists of a continuous segment of characters and each character belongs to **exactly** one substring.

Your task is to find $f_l, f_{l+1}, \ldots, f_r$.

$n \leq 2 \times 10^5$，时限 3 s

Division + LCP (hard version)

CF 1968G1

最极端的情况 $l = 1, r = n$，也就是整个 f 数组都要求出来

我们发现在重叠次数 t 比较大的时候前缀长度不能过长，所以可以想到 *Bigsmall*

当 $t \geq \sqrt{n}$ 的时候，直接枚举前缀长度 $L \leq \sqrt{n}$ 去算出对应的重叠次数贡献到 *ans* 上，时间复杂度为 $O(n\sqrt{n})$

当 $t \leq \sqrt{n}$ 的时候，枚举 t 然后采用 easy version 的做法，时间复杂度为 $O(n\sqrt{n}\log n)$

这样分块不是最优秀的，还可以优化到 $O(n\sqrt{n\log n})$

KMP 算法

简介

假设我们要做字符串匹配，需要在文本串 text 中找到模式串 pattern 是否作为子串出现了。
暴力的做法（匹配的开头一直是 (i-j-1,-1)，某一位对不上就从 (i-j,-1) 重新尝试）：

```
getline(cin,T);
getline(cin,P);
int n=T.length(),m=P.length();
int i=-1,j=-1;
while (i<n-1 && j<m-1){
    if (T[i+1]==P[j+1]) i++,j++;
    else i=i-j,j=-1;
}
if (j==m-1) cout << "Yes" << "\n";
else cout << "No" << "\n";
```

KMP 算法

简介

如果 T="aaaaaab"，P="aaaaac"，那么每次都要到最后一位才发现失败
最坏时间复杂度是 $O(nm)$

```
getline(cin,T);
getline(cin,P);
int n=T.length(),m=P.length();
int i=-1,j=-1;
while (i<n-1 && j<m-1){
    if (T[i+1]==P[j+1]) i++,j++;
    else i=i-j,j=-1;
}
if (j==m-1) cout << "Yes" << "\n";
else cout << "No" << "\n";
```

KMP 算法

我们从 $(i-j-1,-1)$ 开始，顺利的匹配到了 (i,j) 然后下一位失败了，
那么 $(i-j,-1)$ 还能至少顺利匹配到 $(i,j-1)$ 需要什么条件？
需要 $P[0,j-1] == P[1,j]$
那 $(i-j+k,-1)$ 至少顺利匹配到 $(i,j-k-1)$ 呢？
需要 $P[0,j-k-1] == P[k+1,j]$
都需要一个前缀等于后缀的结构。

KMP 算法

这意味着假如我们提前知道对于每个 j，有一个最大的 k 满足 $P[0,k] = P[j-k,j]$，
也就是对于串 $P[0,j]$ 来说，找到了最长相等的真前缀与真后缀
这样子的 k 我们记为 $nxt[j]$（假如不存在这样相等的真前缀或者真后缀那么 $nxt[j] = -1$）
在 (i,j) 的下一位匹配失败后，我们可以直接跳转到 (i,k)
因为从 $(i-j-1,-1),\cdots,(i-k-2,-1)$ 这些地方开始，都不可能完整的匹配完整个串
并且从 $(i-k-1,-1)$ 开始一定至少能走到 (i,k)

KMP 算法

我们能写出这样的代码：

```
int n=T.length(),m=P.length();
int i=-1,j=-1;
while (i<n-1 && j<m-1){
    if (T[i+1]==P[j+1]) i++,j++;
    else if (j!=-1) j=nxt[j];
}
if (j==m-1) cout << "Yes" << "\n";
else cout << "No" << "\n";
```

如何去分析这部分的复杂度？

$j = nxt[j]$ 这部分是使 j 减小的，并且不会小于 -1，我们只需要分析 j 增加的次数，这个最多为 $O(n)$

KMP 算法

那么 nxt 数组如何求？
首先 nxt[0]=-1
假设我已经求出了 $nxt[0], nxt[1], \ldots, nxt[i-1]$，接下来要如何求 $nxt[i]$
其实和前面相似，是个自己匹配自己的过程。

KMP 算法

```
int m=P.length();
int i,j;
for (nxt[0]=j=-1,i=1;i<m;nxt[i++]=j){
    while (j!=-1 && P[j+1]!=P[i]) j=nxt[j];
    if (P[j+1]==P[i]) j++;
}
```

KMP 算法

综合一下:

```
void kmp(int n,char*a,int m,char*b){
    int i,j;  //不需要初始化
    for(nxt[0]=j=-1,i=1;i<n;nxt[i++]=j){
        while(~j&&a[j+1]!=a[i])j=nxt[j];
        if(a[j+1]==a[i])j++;
    }
    for(j=-1,i=0;i<m;i++){
        while(~j&&a[j+1]!=b[i])j=nxt[j];
        if(a[j+1]==b[i])j++;
        if(j==n-1)printf(" %d " ,i),j=nxt[j]; //不要漏了
    }
}
```

OKR-Periods of Words

P3435

对于一个仅含小写字母的字符串 a，p 为 a 的前缀且 $p \neq a$，那么我们称 p 为 a 的 proper 前缀。

规定字符串 Q 表示 a 的周期，当且仅当 Q 是 a 的 proper 前缀且 a 是 $Q + Q$ 的前缀。若这样的字符串不存在，则 a 的周期为空串。

例如 `ab` 是 `abab` 的一个周期，因为 `ab` 是 `abab` 的 proper 前缀，且 `abab` 是 `ab+ab` 的前缀。

求给定字符串所有前缀的最大周期长度之和。

我们希望前后匹配的长度尽可能的短，这样子对应的 proper 前缀就会尽量长
KMP 只能求前后匹配的最大长度，这需要我们不断地跳 nxt 数组
答案应该是 $\sum_{i=0}^{n} i - nxt[nxt[...[i]]]$
假如字符串全是'a'，这样子求时间复杂度是 $O(n^2)$ 的，这需要我们去记忆化。
$f[i] = nxt[i] == -1 ? -1 : (f[nxt[i]] == -1 ? nxt[i] : f[nxt[i]])$

PRE-Prefixuffix

P3546

对于两个串 S_1, S_2，如果能够将 S_1 的一个后缀移动到开头后变成 S_2，就称 S_1 和 S_2 循环相同。例如串 ababba 和串 abbaab 是循环相同的。

给出一个长度为 n 的串 S，求满足下面条件的最大的 $L(L \leq \frac{n}{2})$：S 的 L 前缀和 S 的 L 后缀是循环相同的。

PRE-Prefixuffix

P3546

补充定义：
我们称 B 为 A 的一个 border 当且仅当 B 为 A 的真前缀，也为 A 的真后缀
那么 KMP 求出的 $nxt[i]$ 就是 $A[0,i]$ 的最长 border，A 的 border 全集为 $nxt[i], nxt[nxt[i]], \ldots$
这题要求我们：
用 KMP 求出 S 的所有 Border
对于每个 Border，设当前 Border 为 a，S 的前后去掉 a 的字符串为 T
需要求出 T 的最长 Border，设为 b，则 $ans = max(ans, \|a\| + \|b\|)$

PRE-Prefixuffix

P3546

朴素地求 T 的 border 是 $O(n^2)$ 的
但是我们观察到 T 的中心位置保持不变
设 S 去掉前 i 个字符、后 i 个字符的 T 的最长 border 为 f_i
可以证明 $f_i \leq f_{i+1}+2$，所以 f_i 倒着求，f_i 从 $f_{i+1}+2$ 开始倒着枚举并用哈希验证是否为真的答案
每次 f_i 加 2 然后减去若干，时间复杂度为 $O(n)$

Manacher 算法

回文字符串：正着读和反着读一样
manacher 算法：$O(n)$ 的时间内对每个 i 求出以它为中心的最长回文子串长度
首先我们需要做一个预处理：
将字符串"abca" 转化为这种形式："$a#b#c#a#"，长度从 len 变为 $2 \times len + 2$
这样子可以将长度为奇数的回文串和偶数的回文串统一起来

Manacher 算法

假设 $p[1], p[2], \ldots, p[i-1]$ 都已经求出来了
我们维护 mx 和 id 代表前面 $j + p[j]$ 的最大值以及对应的 j
这代表前面的回文串往右最多能延伸长到哪里
case1：$(id - p[id], id + p[id])$ 包含点 i
case1.1：对称位置 $2 * id - i$ 的最长回文串被 $(id - p[id], id + p[id])$ 完全包住
那只有 $p[i] = p[2 * id - i]$
case1.2：对称位置 $2 * id - i$ 的最长回文串有露出的部分
先 $p[i] = mx - i - 1$，再试图向两侧探索延伸
case2：$(id - p[id], id + p[id])$ 不包含点 i
先 $p[i] = 1$，再试图向两侧探索延伸

Manacher 算法

注意到探索延伸的时候都会使得 mx 往右移动一格，总次数不超过 $O(n)$
所以时间复杂度为 $O(n)$

Manacher 算法

```
char st[N*2], s[N];
int len, p[N*2]; //p[i]表示以i为对称轴的最长回文串长度
int init(char *s){
    len=strlen(s);
    st[0]='$',st[1]='#'; //'$'已经避免了越界
    for(int i=1;i<=len;i++)st[i*2]=s[i-1],st[i*2+1]='#';
    len=len*2+2;
    int mx=0,id=0;
    for(int i=1;i<=len;i++) {
        p[i]=(mx>i)?min(p[id*2-i]+1,mx-i):1;
        for(;st[i+p[i]]==st[i-p[i]];++p[i]);
        if (p[i]+i>mx)mx=p[i]+i,id=i;
        p[i]--;
    }
}
```

拉拉队排练

P1659

艾利斯顿商学院篮球队要参加一年一度的市篮球比赛了。拉拉队是篮球比赛的一个看点，好的拉拉队往往能帮助球队增加士气，赢得最终的比赛。所以作为拉拉队队长的楚雨荨同学知道，帮助篮球队训练好拉拉队有多么的重要。

拉拉队的选拔工作已经结束，在雨荨和校长的挑选下，n 位集优秀的身材、舞技于一体的美女从众多报名的女生中脱颖而出。这些女生将随着篮球队的小伙子们一起，和对手抗衡，为艾利斯顿篮球队加油助威。

一个阳光明媚的早晨，雨荨带领拉拉队的队员们开始了排练。n 个女生从左到右排成一行，每个人手中都举了一个写有 26 个小写字母中的某一个的牌子，在比赛的时候挥舞，为小伙子们呐喊、加油。

雨荨发现，如果连续的一段女生，有奇数个，并且他们手中的牌子所写的字母，从左到右和从右到左读起来一样，那么这一段女生就被称作和谐小群体。

现在雨荨想找出所有和谐小群体，并且按照女生的个数降序排序之后，前 K 个和谐小群体的女生个数的乘积是多少。由于答案可能很大，雨荨只要你告诉她，答案除以 19930726 的余数是多少就行了。

拉拉队排练

P1659

先跑一次类似 manacher 的东西。因为只需要求奇数个的回文串，所以不需要再字符串中插入其他字符。
求完 p 数组以后用一个桶存下来，从小于 n 的最大奇数开始往下统计，统计完就好了。
需要快速幂优化

最长双回文串

P4555

顺序和逆序读起来完全一样的串叫做回文串。比如 `acbca` 是回文串，而 `abc` 不是：`abc` 的顺序为 `abc`，逆序为 `cba`，不相同。

输入长度为 n 的串 S，求 S 的最长双回文子串 T，即可将 T 分为两部分 X, Y（$|X|, |Y| \geq 1$）且 X 和 Y 都是回文串。

最长双回文串

P4555

我们维护最长回文半径 $p[i]$ 的同时，再分别维护两个东西，以 i 为结尾的最长回文子串的长度 $ll[i]$，和以 i 为开头的最长回文子串的长度 $rr[i]$。

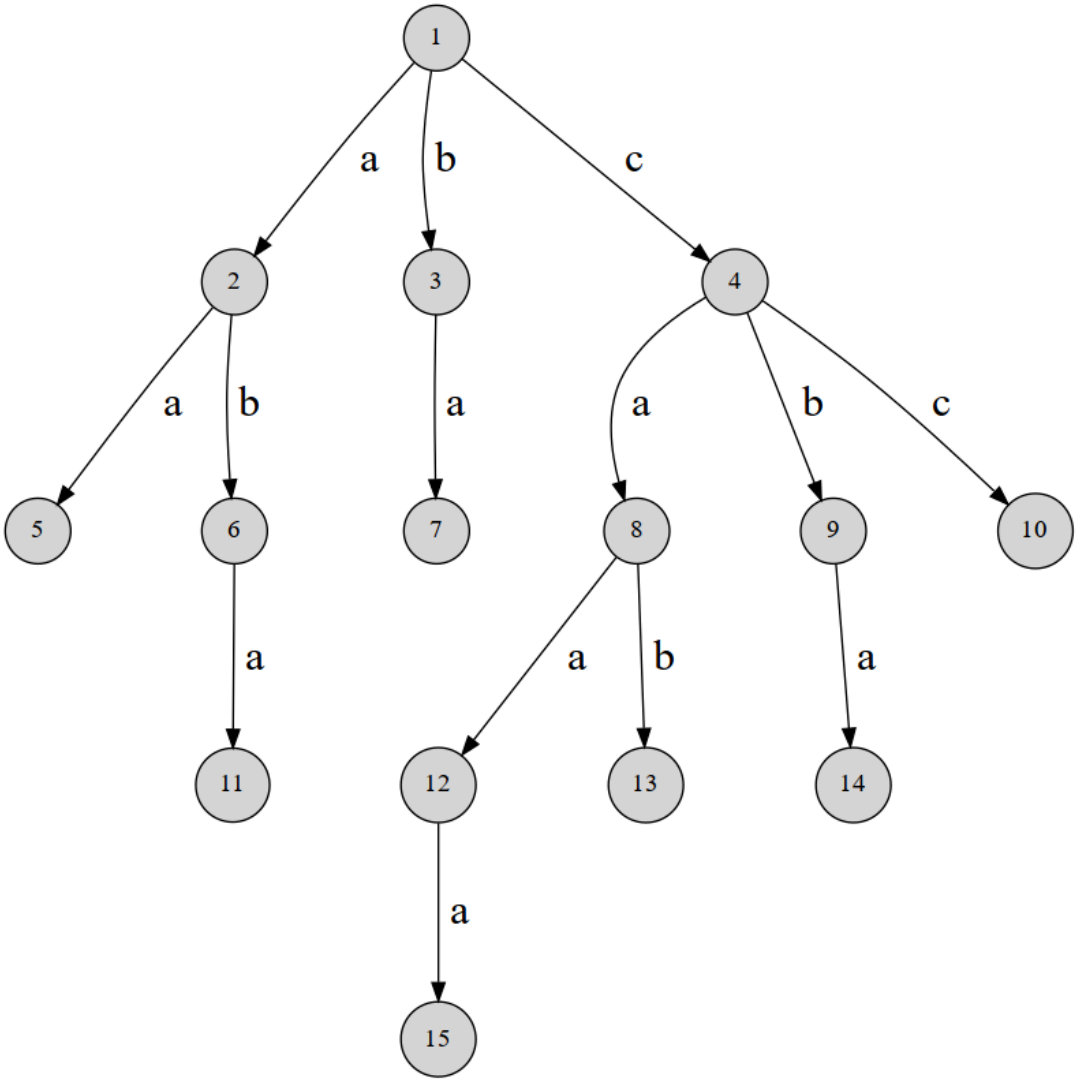
先更新以 $i+p[i]-1$ 为结尾的最长回文长度，

$ll[i+p[i]-1] = max(ll[i+p[i]-1], p[i-1])$

然后再反向推回来 $ll[i] = max(ll[i], ll[i+2]-2)$

$rr[i]$ 同理

字典树（Trie）



Trie

字典树（Trie）

字典树用边来代表字母，而从根结点到树上某一结点的路径就代表了一个字符串。

字典树（Trie）

建立

插入字符串：

```
int nxt[maxn+50][26],cnt;
int exist[maxn+50];
void insert(string &s){
    int pos=0;
    for (auto ch:s){
        int c=ch-'a';
        if (nxt[pos][c]==0) nxt[pos][c]=++cnt;
        pos=nxt[pos][c];
    }
    exist[pos]=1;
}
```

字典树（Trie）

查找

查找字符串：

```
int nxt[maxn+50][26],cnt;
int exist[maxn+50];
int find(string &s){
    int pos=0;
    for (auto ch:s){
        int c=ch-'a';
        if (nxt[pos][c]==0) return 0;
        pos=nxt[pos][c];
    }
    return exist[pos];
}
```

01 Trie

01-trie 是指字符集为 $\{0, 1\}$ 的 trie。
01-trie 可以用来维护一些数字的异或和，支持修改（删除 + 重新插入）。

最大异或对

P10471

给定 N 个整数 $A_1.A_2,\cdots,A_N$ 中选出两个进行异或计算，得到的结果最大是多少？

最大异或对

P10471

将每个数转换成一个 32 位二进制数，并建立一颗 01 trie，
同时枚举每一个 a_i，从高位往低位扫，尽量走不同的节点并跳到该节点上并在答案末尾添加一个 1，如果实在没有就走另一边并添加一个 0，没有节点了就更新 ans 即可。

最长异或路径

P4551

给定一颗 n 个点的带权树，结点下标从 1 开始到 n。寻找树中两个结点间的最长异或路径（异或路径指的是两个结点之间唯一路径上的所有边权的异或）

重要的结论：$x \oplus x = 0$。那么 i 到 j 路径上的异或和，就可以表示成根到 i 的异或和根到 j 的异或和。那么思路就是：将每个点到根的异或和 X_i 预处理出来，将 X_i 插入到 trie 树上。然后对于每一个 X_i，在 trie 树上询问异或最大值。

Master of both

2022 ICPC 杭州 K

给定一个仅由小写英文字母组成的 n 字符串序列，当按字典顺序比较字符串时，该序列中有多少个逆序对？
Putata 和 Budada 他们轻松地一起解决了这个问题。然而，存在 q 个不同的平行宇宙，其中字母表中的字符并不按原始顺序出现。
形式上，每个宇宙中的字母都是一个字符串，它是 26 小写英文字母的排列，表示每个字符出现的顺序。当且仅当以下条件之一成立时，字符串 a 按字典顺序小于字符串 b:
1、a 是 b 的前缀，但 $a \neq b$;
2、在 a 和 b 不同的第一个位置，字符串 a 有一个字母在字母表中出现的时间早于 b 中对应的字母.
$1 \leq n \leq 5 \times 10^5, 1 \leq q \leq 5 \times 10^4$
$\sum \|s_i\| \leq 10^6$

任意两个字符串的比较其实只取决于他们第一个不相同的字母，因此可以用一个数组 $cnt_{a,b}$，来表示有 $cnt_{a,b}$，对字符串的第一个不同的字母是 a, b，且 A 所在的串在 B 串的前面。在每次询问的时候只需看两个字母 a, b 的顺序，若 a 排在 b 后面，则会产生 $cnt_{a,b}$ 的逆序对贡献。

那么怎么求 cnt 数组呢？这里使用字典树来求，按照插入顺序更新数组的值。
注意如果某个串为另一个串的前缀，那么字典序会更小，这个判断可以通过在每个串末尾加一个比 a 小的字符来实现
时间复杂度 $O(\sum \|s\| \times 26 + q \times 26^2)$

Type Printer

P4683

你需要利用一台可移动的打印机打印出 n 个单词。这种可移动式打印机是一种老式打印机，它需要你将一些小的金属块（每个包含一个字母）放到打印机上以组成单词。然后将这些小金属块压在一张纸上以打印出这个词。这种打印机允许你进行下列操作：

- 在打印机当前词的末端（尾部）添加一个字母；
- 在打印机当前词的尾部删去一个字母（将打印机当前词的最后一个字母删去）。仅当打印机当前至少有一个字母时才允许进行该操作；
- 将打印机上的当前词打印出来。

初始时打印机为空，或者说它不含任何带字母的金属块。打印结束时，允许有部分字母留在打印机内。同时也允许你按照任意的次序打印单词。

由于每一个操作都需要一定时间，所以需要你尽可能减少所需操作的总数目（将操作的总数最小化）。

你需要编写一个程序，给定所要打印的 n 个单词，找出以任意次序打印所有单词所需操作的最小数目，并输出一种这样的操作序列。

Type Printer

P4683

我们只需要在 Trie 树上遍历就可以了。但是题目要求的是最少的操作，所以我们要思考一下怎么去减少操作。
在打印完最后一个单词时可以直接输出结果结束程序，不用再去将打印机清空。
这样就可以想出一种策略：我们只需要将最长的单词放在最后去打印，就可以尽量减少删除次数
把最长单词打上标记，遍历的时候优先去跑没有标记的节点。

AC 自动机

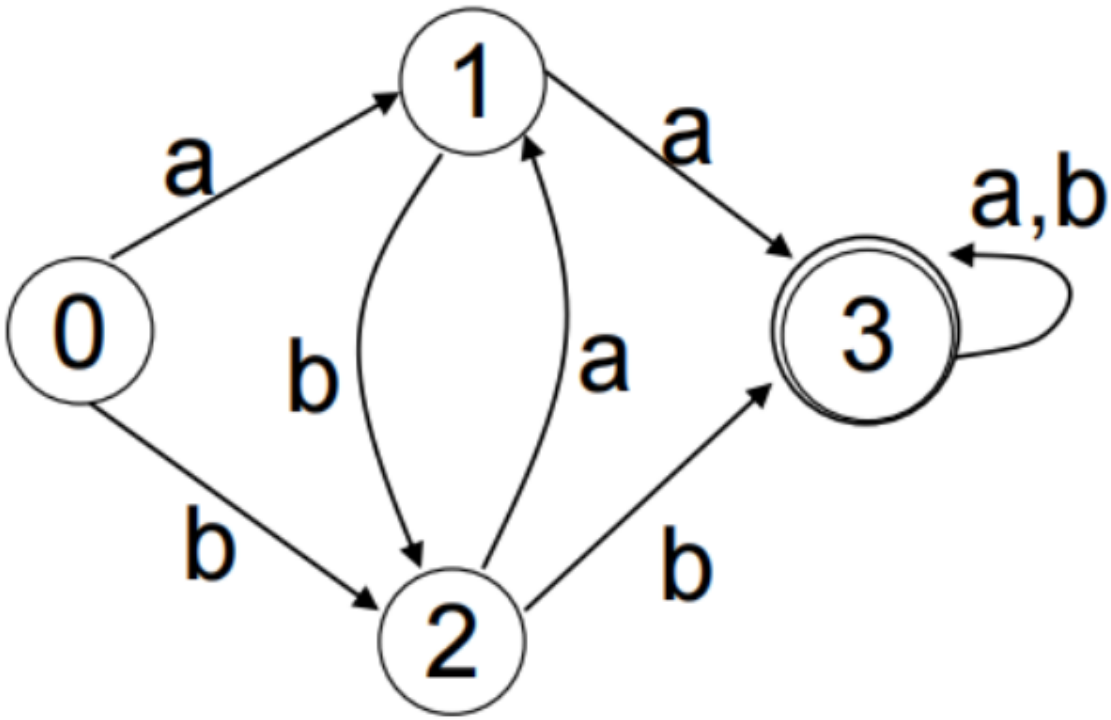
简介

AC 自动机，全称 Aho-Corasick automaton。用于解决一个文本串 T 和多个模式串 $P_1, P_2, \ldots, P_m$ 匹配的问题
然而 AC 自动机并不能自动 Accepted。。。

AC 自动机

简介

介绍一下前置知识 DFA（确定性有穷状态自动机）。
有 n 个状态，每个状态接收到下一个字符之后会跳转到对应的下一个状态
有一个起始状态，所有状态分为终结状态和非终结状态，如果串被消耗完，并且停留在终结状态，那么称 DFA 接受该串，否则不接受该串

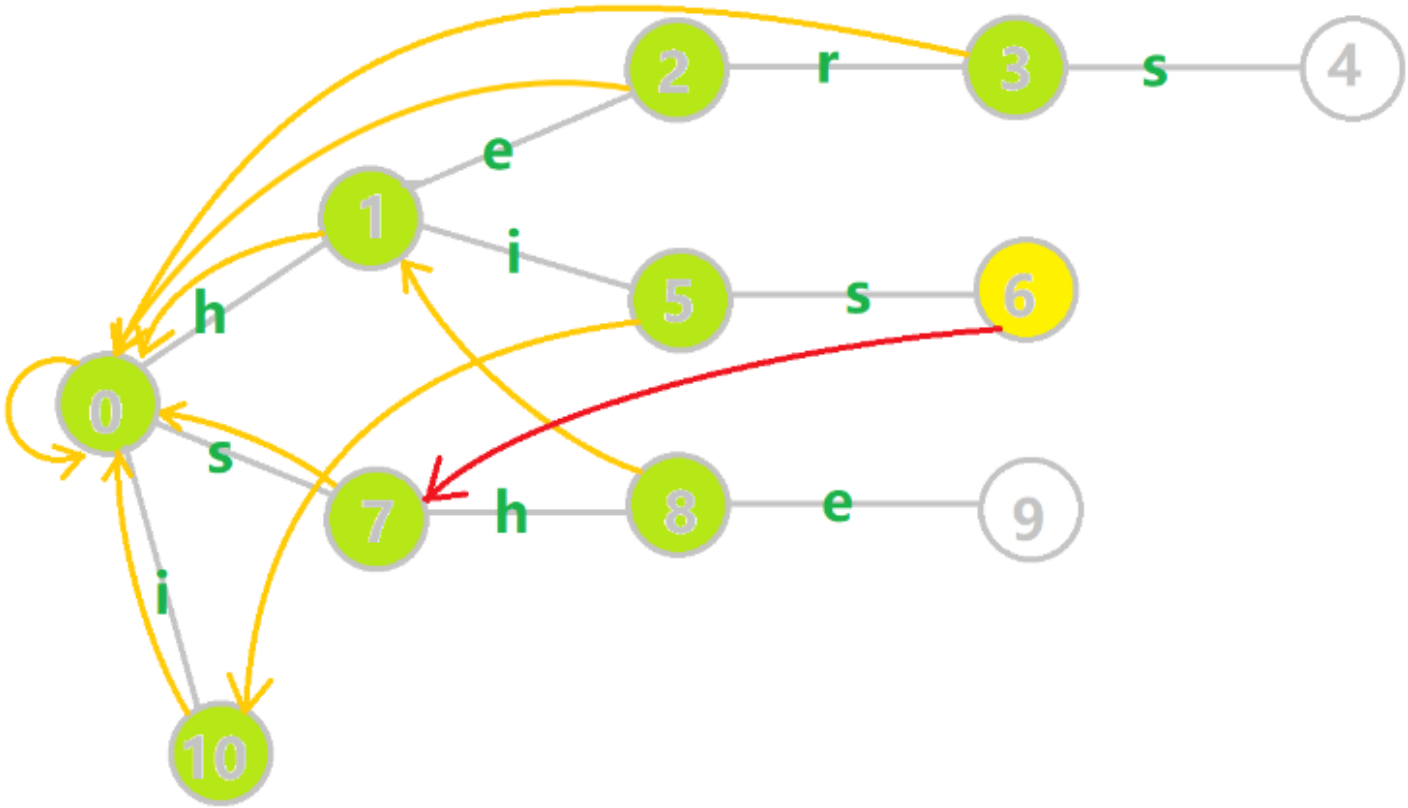


AC 自动机

简介

AC 自动机的目标就是构造一个这样的自动机（并尽量的小），去接受包含若干个模式串的文本串。

fail 指针：指向最长的真后缀等于真前缀的地方。



AC 自动机

简介

匹配的时候像 KMP 一样不停的跳 $p = fail[p]$ 直到 $trie[p][c]! = 0$
如果只是做是否存在某个模式串，这样复杂度还是对的
但是假如我们需要统计每个模式串的出现次数呢？

我们可以扩展一下 $trie[p][c]$，假如 $trie[p][c]! = 0$ 维持原样
否则跳到“该跳的地方”，$trie[u][c] = trie[fail[u]][c]$
这样匹配的时候相当于 $i = i + 1, j = trie[p][ch]$

统计模式串出现次数的时候需要把经过状态的 cnt 加一，最后沿着 fail 树拓扑排序把 cnt 贡献到更小的模式串中。

AC 自动机（二次加强版）

P5357

模板题，需要你正确的实现 AC 自动机

AC 自动机（二次加强版）

P5357

```
void insert(char* s,int num){
    int u=1,len=strlen(s);
    for(int i=0;i<len;i++){
        int v=s[i]-'a';
        if(!trie[u].son[v])trie[u].son[v]=++cnt;
        u=trie[u].son[v];
    }
    if(!trie[u].flag)trie[u].flag=num;
    Map[num]=trie[u].flag;
}
```

AC 自动机（二次加强版）

P5357

```
void getFail(){
    for(int i=0;i<26;i++)trie[0].son[i]=1;
    q.push(1);
    while(!q.empty()){
        int u=q.front();q.pop();
        int Fail=trie[u].fail;
        for(int i=0;i<26;i++){
            int v=trie[u].son[i];
            if(!v){trie[u].son[i]=trie[Fail].son[i];continue;}
            trie[v].fail=trie[Fail].son[i]; in[trie[v].fail]++;
            q.push(v);
        }
    }
}
```

AC 自动机（二次加强版）

P5357

```
void query(char* s){
    int u=1,len=strlen(s);
    for(int i=0;i<len;i++)
    u=trie[u].son[s[i]-'a'],trie[u].ans++;
}
```

AC 自动机（二次加强版）

P5357

```
void topu(){
    for(int i=1;i<=cnt;i++)
    if(in[i]==0)q.push(i);
    while(!q.empty()){
        int u=q.front();q.pop();vis[trie[u].flag]=trie[u].ans;
        int v=trie[u].fail;in[v]--;
        trie[v].ans+=trie[u].ans;
        if(in[v]==0)q.push(v);
    }
}
```

谢谢