ACM Template

目录

[ACM Template 1](#_Toc142920639)

[数学 1](#_Toc142920640)

[欧拉函数 1](#_Toc142920641)

# 数学

## \*实用数学结论

逆元存在的条件

在模p的群中 则称x为a的逆(inv).

逆元存在的条件是

鸽笼原理 Pigeonhole Principle：

13个人中至少2个人的生日在同一个月

找规律

通过通向表达式找规律，将数字用a，b等字母替代即可

中位数

中位数的性质：所有数与中位数的绝对差之和最小

## \*整除分块

作用用的时间复杂度计算的问题

先得到处的的值然后计算这个值的右边界

1

…

…

n/5

图中橙色区域的值相同都等于而即得到的下标是整除能得到这个值的最大的位置，解释：直接用除法得到结果的余数最小且不会超过除数

1

…

…

//整除分块 复杂度O 根n

long long fenkuai(long long n) {

long long ans = 0;

for (long long l = 1, r; l <= n; l = r + 1) {

r = n / (n / l);

ans += (r - l + 1) \* (n / l);

}

return ans;

}

计算

if(n>m)swap(n,m);

for(int l=1;l<=n;l++){

int r=min(n/(n/l),m/(m/l);

int v1=n/l;

int v2=m/l;

ans+=(l-r)\*v1\*v2;

l=r;

}

## \*欧拉函数

中与 互质的数的个数，被称为欧拉函数

// 1~N 中与 N 互质的数的个数，被称为欧拉函数

//时间复杂度为根号n

inline int euler\_one(int n) {

int ans = n;

for (int i = 2; i \* i <= n; ++i) {

if (n % i == 0) {

ans = ans / i \* (i - 1);

while (n % i == 0)

n /= i;

}

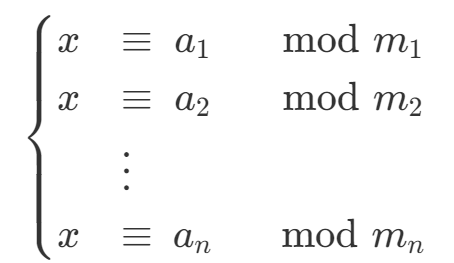
}

if (n > 1) ans = ans / n \* (n - 1); // n至多有一个比根号n大的质因子

return ans;

}

## \*中国剩余定理：韩信点兵



1.若两两互质则有解

2. 在，下有唯一解为

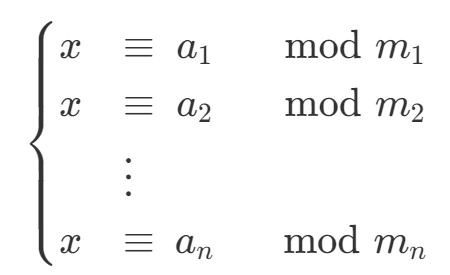
并设 是除了 以外的 个整数的乘积。

设 为 模 意义下的逆元。

理解性解释：

中国剩余定理主要用于解决这样的线性同余方程组，其中

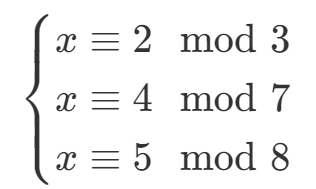
两两互质



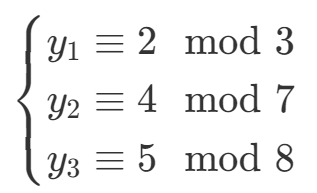
对于这样的方程组我们只需要求出 的一组特解，把这个特解加上 的最小公倍数的任意整数倍就可以得到所有解，所以我们就只需要考虑怎么构造出一组特解

例子

下面举个例子来帮助理解



直接构造不太行，但是对于每一个线性同余方程构造出一个特解是非常轻松的，所以我们先设第 个式子的解是 ，那么有



最终的 一定是 以各种方式组合起来，那么问题就是怎么组合才能得到

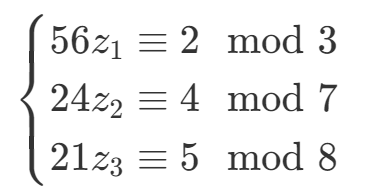
，如果 之间相乘就把问题变得更复杂了，所以我们把它们加起来，想办法凑出 如果 ，那么每一个 需要满足什么条件呢？因为 ，所以 一定是 3 的倍数，当 分别都是 3 的倍数时这一定也是满足条件的一个特解

所以我们不妨设 为3的倍数

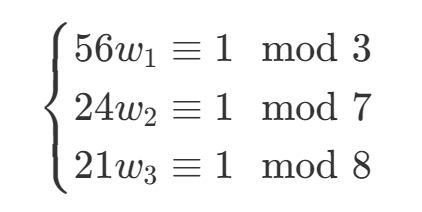
同理 为7的倍数，为8的倍数

那我们就可以把 写成 ，把 写成 ，把 写成

原方程就变成了这样：



然后就非常好求了，只需要求出



也就是 56,24,21 分别对于 3,7,8 的乘法逆元，然后在等式两边同时分别乘以 2,4,5 就可以得到 的一组特解,用扩展欧几里得算法求出 ，所以 于是

最后就得到 由于3,7,8的最小公倍数是168，我们将1229对于168取模就能得到 的最小正整数解

的通解就能表示为

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

#define int long long

const int N = 15;

int n, a[N], mods[N];

int exgcd(int a, int b, int& x, int& y) {

if (!b) {

x = 1, y = 0;

return a;

} else {

int gcd = exgcd(b, a % b, y, x);

y -= x \* (a / b); // 此处一定要先除后乘不然可能爆精度建议加括号

return gcd;

}

}

inline ll inv(ll x, ll p) // 求逆元

{

ll X, Y;

exgcd(x, p, X, Y);

return (X % p + p) % p;

}

inline ll crt() // CRT

{

ll prod = 1, ans = 0;

for (int i = 1; i <= n; ++i)

prod \*= mods[i];

ll tmp;

for (int i = 1; i <= n; ++i) {

tmp = prod / mods[i];

ans += (inv(tmp, mods[i]) \* a[i] \* tmp) % prod;

}

return ans % prod;

}

/\*

a[i]为每个方程式的余数

mods[i]为每个方程的模数

使用条件为所有mods[i]互质

\*/

signed example() {

cin >> n;

for (int i = 1; i <= n; ++i)

cin >> a[i] >> mods[i];

int ans = crt();

cout << ans;

}

## 数论基本知识

数论函数：定义域在整数上的函数

符号说明：(a,b)=c 相当于gcd(a,b)=c

积性函数：时

积性函数的性质： 则的全展开

积性函数f被所有处的值决定的

完全积性函数:不要求(a,b)=1即可f(ab)=f(a)f(b)完全积性函数也是积性函数

典型函数代表：

f(x)=1 I(x)=1

f(x)=x id(x)=x

f(x)=1 (x==1) =0(x!=1) e(x)=1 (x==1) =0(x!=1)

[x=1] 表示x==1时表达式=1，否则=0

中与 互质的数的个数，被称为欧拉函数

d(n) 因子个数

因子和

莫比乌斯函数

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | -1 | -1 | 0 | -1 | 1 | -1 | 0 | 0 | 1 |

// O(n)线性筛

// 其中p[i]表示i的最小素因子是多少，pr[tot]表示1~n范围内的质数

p[1]=1;

for(int i=2;i<=n;i++){

if(!p[i]) p[i]=i,pr[++tot]=I;

for(int j=1;j<=tot&&pr[j]\*i<=n;j++){

p[i\*pr[j]]=pr[j]; // 保证每个数只会被其最小的因子筛出

if(p[i]==pr[j])break; // 如果两者最小素因子相同就退出

}

}

2

3

5

7

9

1

1

+1

1

1

+1

1

1

2

1

break

// O(n)线性筛加求出指数

// 其中p[i]表示i的最小素因子是多少，pr[tot]表示1~n范围内的质数，pe[i]表示最小素因

// 子含指数一共是多少

p[1]=1;

for(int i=2;i<=n;i++){

if(!p[i]) p[i]=I,pe[i]=i,pr[++tot]=I;

for(int j=1;j<=tot&&pr[j]\*i<=n;j++){

p[i\*pr[j]]=pr[j]; // 保证每个数只会被其最小的因子筛出

if(p[i]==pr[j]){

pe[i\*pr[j]]=pe[i]\*pr[j];

break;

}else{

pe[i\*pr[j]]=pr[j];

}

}

}

线性筛筛出

i的情况：

1.pe[i] i为质数好求

2.i!=pe[i] i为合数 f[i]=f[i/pe[i]]\*f[pe[i]]

3.i=pe[i] i为质数的>=2次幂

// 线性筛筛出d sigma varphi mu O(n)

fd[1]=1;

for(int i=2;i<=n;i++){

if(i==pe[i])fd[i]=fd[i/p[i]]+1;

else fd[i]=fd[pe[i]]\*fd[i/pe[i]];

}

fsigma[1]=1;

for(int i=2;i<=n;i++){

if(i==pe[i]) fsigma [i]= fsigma [i/p[i]]+i;

else fsigma [i]= fsigma [pe[i]]\* fsigma [i/pe[i]];

}

fphi[1]=1;

for(int i=2;i<=n;i++){

if(i==pe[i])fphi[i]=i/p[i]\*(p[i]-1);

else fphi[i]=fphi[pe[i]]\*fphi[i/pe[i]];

}

fmu[1]=1;

for(int i=2;i<=n;i++){

if(i==pe[i]){

if(i==p[i])fmu[i]=-1;

else fmu[i]=0;

}else fmu[i]=fmu[pe[i]]\*fmu[i/pe[i]];

}

## 狄利克雷卷积

狄利克雷卷积是用来简化表示的

对于两个数论函数f，g => h=f\*g

狄利克雷卷积满足交换律

狄利克雷卷积满足结合律

重要：f和g 是积性函数 => f\*g 也是积性函数

## 莫比乌斯反演

莫比乌斯反演证明法1：

由证明积性函数只需要证质数的幂次得

证毕

莫比乌斯反演证明法2：

证毕

例题&模板：

给定一个函数 已知 求

在用上面的代码求出之间的的值之后用下面的代码求出g

// 通过求和式子求函数值写法 时间复杂度O(n)

for(int d1=1;d1<=n;d1++)

for(int d2=1;d1\*d2<=n;d2++){

g[d1\*d2]+=f[d1]\*mu[d2]

例题2：

问有多少对i,j满足且i，j互质

分析：

把d提出来

这样i，j就变得独立了

用分块整除优化。

if(n>m)sawp(n,m);

for(int l=1;l<=n;l++){  
 int v1=n/l;

int v2=m/l;

int r=min(n/v1,m/v2);

ans+=(summu[r]-summu[l])\*v1\*v2;

l=r;

}

小结：

形如 其中f是任意数论函数 (i,j)表示gcd(i,j)

例如例题2中f为e例题3中f为id

我们都可以搞出一个数论函数g使

于是

然后就把d提出来

例题3：

求

分析：

借助上面的小结，这题中f就相当于数论函数id

下面计算

首先因为其为积性函数，所以我们只用计算定义域为素数的幂次的位置即可

根据的定义只有d=1和d=p时项不等于0

时

时

于是

借助上面的结论

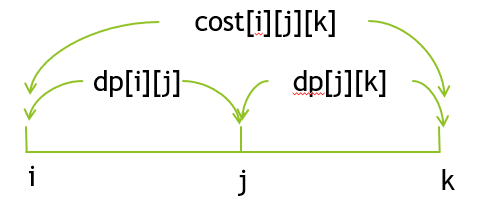
寄语：

刚刚的题目的做法不是偶然，对于上面小结中总结的题型，就是这种固定的做法，属于是学了就会不学就完全不会。前几年比较常考，但由于比较机械化，就没考的那么频繁了。 --杜瑜皓老师

# 动态规划

## 区间DP

定义：区间dp，顾名思义就是在区间上进行dp，主要思路是通过合并小区间来得到大区间的最优解。



区间[i,j][j,k]的最优解分别为dp[i][j],dp[j][k]，将他们合并为区间[i,k]的代价为cost[i][j][k]

那么dp[i][k]=dp[i][j]+dp[j][k]+cost[i][j][k]

## 状压DP

在 n\*n 的棋盘里面放 k 个国王（1 <= n <= 9,k<=n\*n），使他们互不攻击，共有多少种摆放方案。

国王能攻击到它上下左右，以及左上左下右上右下八个方向上附近的各一个格子，共 8 个格子。

我们发现n<=9于是可以用一个int中的9位表示有没有国王

有就将这一位从0变成1然后通过位运算就可以加速比较过程

## 树形DP

定义：树形dp，就是在树上进行dp，主要思路是通过合并子树的解来得到问题的解。

## 树形背包

例4

给定一棵有n 个节点的树，每个节点有一个权值，要求你从中选出m 个节点，使得这些选出的节点的权值和最大，一个节点能被选当且仅当其父亲节点被选中，根节点可以直接选。

状态定义与普通树形背包相似

设dp[i][j]为第i个节点为根的子树已经选中了j个节点的最大权值和

## 数位DP(属于计数DP)

数位dp是一种计数用的dp，一般就是要统计一个区间[le,ri]内满足一些条件数的个数。所谓数位dp，字面意思就是在数位上进行dp咯。数位还算是比较好听的名字，数位的含义：一个数有个位、十位、百位、千位......数的每一位就是数位啦！

数位dp的实质就是换一种暴力枚举的方式，使得新的枚举方式满足dp的性质，然后记忆化就可以了。

# 字符串

## \*哈希算法

用途：将字符串哈希成一个整数，将字符串比较和查找优化成O(1)

枚举该字符串的每一位，与base相乘，转化为base进制，加(ull)是为了防止爆栈搞出一个负数，(ull)是无符号的，但其实加了一个ull是可以不用mod的，加个mod更保险

#include <bits/stdc++.h>

using namespace std;

#define int unsigned long long

typedef unsigned long long ull;

const int hashmod = 1e18 + 2049;

//

int mhash(char s[]) { // 用scanf加快读入

int base = 131;

int val = 0, len = strlen(s);

for (int i = 0; i < len; i++) {

val = (base \* val + (ull)s[i]) % hashmod;

}

return val;

// 枚举该字符串的每一位，与base相乘，转化为base进制，加(ull)是为了防止爆栈搞出一个负数，(ull)是无符号的，但其实加了一个ull是可以不用mod的，加个mod更保险

}

## \*KMP算法

用途：O(n)判断一个串中是否存在另一个串

若令next[j]=k则next[j]表明当模式中第j个字符与主串中相应字符“失配”时在模式中需要重新和主串中该字符进行比较的字符的位置。

next串(j)

主串(i)

匹配成功或者j到-1

则i下一位,j匹配成功才下一位

-1

0

1

2

3

#include <bits/stdc++.h>

using namespace std;

int kmp[1000];

// kmp[j]表示j+1位匹配失败后要跳转的位置

// a[]从0开始有效

void kmp\_match(char a[], char b[]) { // a为主串b为模式串 （简单点说a长，b短）

int la = strlen(a);

int lb = strlen(b);

kmp[0] = -1;

int j = -1;

for (int i = 1; i < lb; i++) {

while (j != -1 && b[i] != b[j + 1])

j = kmp[j];

//此处判断j是否为0的原因在于，如果回跳到第一个字符就不用再回跳了

//通过自己匹配自己来得出每一个点的kmp值

if (b[j + 1] == b[i]) j++;

kmp[i] = j;

// i+1失配后应该如何跳

}

j = -1; // j可以看做表示当前已经匹配完的模式串的最后一位的位置

//如果楼上看不懂，你也可以理解为j表示模式串匹配到第几位了

for (int i = 0; i < la; i++) {

while (j != -1 && b[j + 1] != a[i])

j = kmp[j];

//如果失配 ，那么就不断向回跳，直到可以继续匹配

if (b[j + 1] == a[i]) j++;

//如果匹配成功，那么对应的模式串位置++

if (j == lb - 1) {

j = kmp[j];

//继续匹配

}

}

}

## \*马拉车算法

用途： O(n)查找字符串最长回文子串

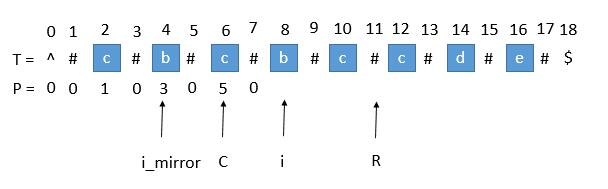
马拉车算法 Manacher‘s Algorithm

首先我们解决下奇数和偶数的问题，在每个字符间插入"#"，并且为了使得扩展的过程中，到边界后自动结束，在两端分别插入 "^" 和 "$"，两个不可能在字符串中出现的字符，这样中心扩展的时候，判断两端字符是否相等的时候，如果到了边界就一定会不相等，从而出了循环。经过处理，字符串的长度永远都是奇数了。

首先我们用一个数组 P 保存从中心扩展的最大个数。

例如下图中下标是 6 的地方。可以看到 P[ 6 ] 等于 5，所以它是从左边扩展 5 个字符，相应的右边也是扩展 5 个字符，也就是 "#c#b#c#b#c#"。而去掉 # 恢复到原来的字符串，变成 "cbcbc"，它的长度刚好也就是 5。

用 P 的下标 i 减去 P [ i ]，再除以 2 ，就是原字符串的开头下标了，开头下标从零开始。





用 i\_mirror 表示当前需要求的第 i 个字符关于 C 对应的下标。

我们其实可以利用回文串 C 的对称性。i 关于 C 的对称点是 i\_mirror ，P [ i\_mirror ] = 3，所以 P [ i ] 也等于 3 。

当然要注意i直接复制过来的回文范围不超出 R

然后再对i用中心扩展算法看能否继续扩展

判断是否需要更新回文右边界 R

// 算法讲解

// <https://zhuanlan.zhihu.com/p/70532099>

// 马拉车算法 Manacher‘s Algorithm 是用来查找一个字符串的最长回文子串的线性方法

#include <bits/stdc++.h>

using namespace std;

// 马拉车算法

const int maxn = 1e6 + 10;

string temp;

int p[maxn];

string longestPalindrome2(string& s) {

temp = "^";

for (auto ch : s) {

temp += "#" + ch;

}

if (s.length()) temp += "#";

temp += "$";

int n = temp.length();

int C = 0, R = 0;

for (int i = 1; i < n - 1; i++) {

int i\_mirror = 2 \* C - i;

p[i] = min(R - i, p[i\_mirror]); // 防止超出 R

// 对i用中心扩展算法看能否继续扩展

while (temp[i + 1 + p[i]] == temp[i - 1 - p[i]]) {

p[i]++;

}

// 判断是否需要更新 R

if (i + p[i] > R) {

C = i;

R = i + p[i];

}

}

// 找出 p 的最大值

int maxLen = 0;

int centerIndex = 0;

for (int i = 1; i < n - 1; i++) {

if (p[i] > maxLen) {

maxLen = p[i];

centerIndex = i;

}

}

int start = (centerIndex - maxLen) / 2; // 最开始讲的求原字符串下标

return s.substr(start, start + maxLen);

}

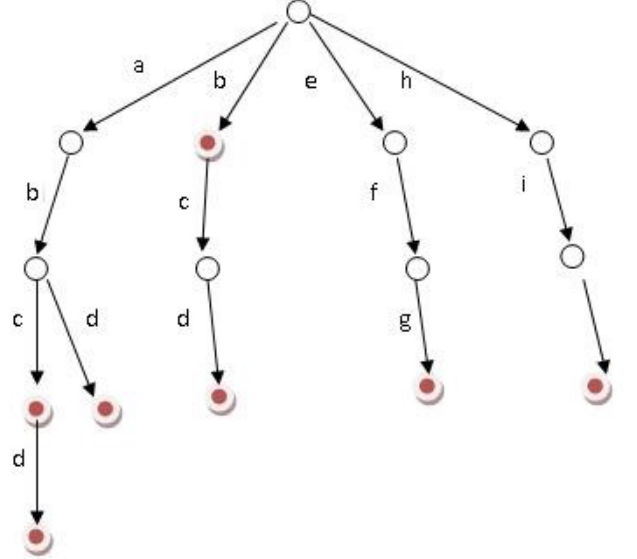
## 字典树(前缀树，Tire树)

用途：用于统计和排序大量的字符串，时间复杂度O（L）L为串长度

Trie树，即字典树，又称单词查找树或键树，是一种树形结构，是一种哈希树的变种。典型应用是用于统计和排序大量的字符串（但不仅限于字符串），所以经常被搜索引擎系统用于文本词频统计。它的优点是：利用字符串的公共前缀来减少查询时间，最大限度地减少无谓的字符串比较。

Trie的核心思想是空间换时间。利用字符串的公共前缀来降低查询时间的开销以达到提高效率的目的。

<https://zhuanlan.zhihu.com/p/28891541>



// trie tree的储存方式：将字母储存在边上，边的节点连接与它相连的字母

// trie[rt][x]=tot:rt是上个节点编号，x是字母，tot是下个节点编号

#include <bits/stdc++.h>

using namespace std;

const int N = 2000010;

int tot = 1;

int trie[N][26]; //trie[rt][x]=tot:rt是上个节点编号x是字母，tot是下个节点编号

int isw[N]; //标志该单词末位字母的尾结点，在查询整个单词时用到

int sum[N]; //前缀出现的次数

int search(char s[], int op = 1) {

int root = 0;

int len = strlen(s);

for (int i = 0; i < len; i++) {

int id = s[i] - 'a';

if (!trie[root][id]) return 0;

root = trie[root][id];

} // root经过此循环后变成前缀最后一个字母所在位置

if (op == 1) return isw[root]; // op=1查询是否为单词或单词出现的次数

if (op == 2) return sum[root]; // op=2查询前缀出现的次数

return 1; // op=0是否为前缀或单词

}

void insert(char s[]) {

int len = strlen(s);

int root = 0;

for (int i = 0; i < len; i++) {

int id = s[i] - 'a';

if (!trie[root][id]) trie[root][id] = ++tot;

root = trie[root][id];

sum[root]++; //前缀保存

}

isw[root]++; //标志该单词末位字母的尾结点，在查询整个单词时用到

}

## AC自动机

用途：在主串中找到多个词汇所在的位置和出现频率

时间复杂度为O(n+m)，其中n是文本串的长度，m是所有模式串的长度之和。空间复杂度为O(m)

AC自动机是最一种多模式匹配算法，

第一步：构建Trie树

第二步：构建fail表,构建过程使用了BFS（宽度优先搜索）算法

第三步：模式匹配

构建过程文字描述

1）将根结点的所有孩子结点的fail指向根结点，然后将根结点的所有孩子结点依次入列。

2）若队列不为空：

2.1）出列，我们将出列的结点记为u,

2.2) 判断u.child[i]是否存在，

存在：u.child[i].fail = fail[u].child[i]，

u.child[i]入列，再次执行步骤2)

不存在：将u的子节点直接等于fail[u]的子节点u.child[i] = fail[u].child[i]

若队列为空:结束

<https://www.cnblogs.com/nullzx/p/7499397.html>

建fail过程中v的fail跳转到u的fail的相同子节点位置

如果u没有此子节点直接将此子节点等于fail的子节点

匹配过程中会复制一份结点位置use，让use往回一直跳到被匹配过的位置或0

而P直接走到树上的下一个位置

，代码实现时是先走P再复制use

fail[u]数组含义：u的下一位匹配失败跳转的位置 fail[u]要跳转的地方最后一位与u相同 与kmp模板一样 与数据结构书上不同

#include <bits/stdc++.h>

using namespace std;

const int N = 10010, S = 55, M = 1000010;

char s[M];

int idx, cnt[N \* S], tr[N \* S][26];

int fail[N \* S];

// fail[u]数组含义：u的下一位匹配失败跳转的位置 fail[u]要跳转的地方最后一位与u相同 与kmp模板一样 与数据结构书上不同

void insert() { // 像普通字典树一样插入

int p = 0;

for (int i = 0; s[i]; i++) {

int t = s[i] - 'a';

if (!tr[p][t]) tr[p][t] = ++idx;

p = tr[p][t];

}

cnt[p]++;

}

void build\_fail() {

queue<int> Q;

for (int i = 0; i < 26; i++) {

if (tr[0][i]) Q.push(tr[0][i]);

}

while (Q.size()) {

int u = Q.front();

Q.pop();

for (int i = 0; i < 26; i++) {

int v = tr[u][i];

// 如果!v说明没有该儿子,那么我们可以通过路径压缩的思想,看看其父亲对于该儿子是指向何处,如果

// 其父亲也不知道该指向何处,那么就会询问其父亲的父亲。

if (!v) {

tr[u][i] = tr[fail[u]][i];

} else {

fail[v] = tr[fail[u]][i];

Q.push(v);

}

}

}

}

void example\_use() {

idx = 0;

int n;

scanf("%d", &n);

for (int i = 1; i <= n; i++) {

scanf("%s", s);

insert();

}

build\_fail(); // 树上建立NEXT数组,用BFS

scanf("%s", s);

int res = 0; // 统计不同子串的个数

for (int i = 0, p = 0; s[i]; i++) {

int t = s[i] - 'a';

p = tr[p][t];

int use = p; // 统计以当前串为后缀的条件下，符合条件的子串个数。

// 为-1的话说明当前串串前面已经统计过了,我们不必重复统计,因为我们只是统计不同串出没出现,而不是出现几次。

while (use && cnt[use] != -1) {

res += cnt[use];

cnt[use] = -1; // 统计过了更新成-1

use = fail[use];

}

}

printf("%d\n", res);

}

# Debug记录和解题注意

## 数学

### 精度考虑先乘最后除，遇到圆周率的问题pi的小数尽量往后取多一点

## 图论

### 和路径相关的问题要注意起点和终点有没有可能相等

例如在起点和终点可能相等的情况下：

就只能这样写

void bfs(int dis[], int s) {}

而不能这样写

void bfs(int s) {

int h = 0;

if (s == T) h = 1;

if (s == N) h = 2;

}

2023年百度之星初赛第一场公园这一题，TFN三者可能相等，于是代码只能这样写void bfs(int dis[], int s) {}

### 无向边要开两倍数组，数组可能开小

## 通用

### 遇到计量单位是考虑是否可能为零为负

如速度，质量，价值

### 跟榜开题