|  |
| --- |
| ZZULI |
| 叹息的模板 |
| 队友：躺好，我带你飞。我：好的。 |

|  |
| --- |
| XCP  2020-10-12 |

数学 6

卡特兰数 6

逆元 7

N以内的素数和 7

扩展欧几里德： 8

扩展中国剩余定理： 9

EXBSGS： 10

欧拉函数： 11

欧拉降幂： 11

线性筛： 12

母函数 12

康托展开： 13

逆康托展开： 13

大数加法 14

大数减法： 15

大数乘法 16

大数乘法(FFT) 16

大数开根号： 19

FFT： 19

Pollard\_rho算法（大数分解素因子） 21

线性基: 24

拉格朗日插值： 26

高斯消元(浮点数) 27

高斯消元(开关问题)： 28

矩阵快速幂： 31

Lucas定理： 33

Polya定理： 33

旋转： 33

圆环翻转： 33

N\*N大小的方阵： 34

图论 35

前向星： 35

最小树形图： 35

SPFA： 36

网络流最大流dinic: 37

最小费用流： 39

Dijsktra优化版： 39

Spfa版： 41

欧拉路径： 43

混合欧拉： 43

二分图最大匹配（匈牙利算法）： 46

二分图带权最大匹配（KM算法）： 47

一般图最大匹配（带花树算法）： 49

Tarjan算法： 52

求割点，桥，vis数组需初始化为0 52

强连通分量，缩点 52

离线求LCA 55

2-SAT： 55

LCA(最近公共祖先)： 57

树上差分： 58

树的重心： 60

Matrix\_tree定理： 61

动态规划 63

区间dp: 63

数位dp： 63

计算几何 66

二维几何： 66

精度控制： 66

点类： 66

点对线 67

点a与直线k1k2垂足的坐标 67

判断点p是否在线段ab上 67

点p到线段ab最短的距离 67

点a关于k1，k2对称的点 68

线对线 68

两直线垂直或平行 68

判断线段ab与线段cd是否有交点 68

判断直线ab与直线cd是否有交点 69

直线ab与线段cd交点坐标 69

求向量p0p1 与 向量p0p2 位置关系 69

求向量ab与向量ac夹角 69

极角排序： 70

利用叉积进行排序 70

判断是否是凸包 70

求任意多边形重心： 71

判断P点是否在任意多边形内 71

最近点对 72

最小球覆盖： 73

Pick公式： 74

字符串 75

KMP 75

AC自动机 75

AC自动机(简易版)： 78

Exkmp： 79

Manachar： 80

后缀数组： 82

后缀自动机 84

回文树： 85

最小表示法： 87

回文自动机 87

后缀自动机： 89

数据结构 91

主席树： 91

ST表： 92

左偏树： 93

树套树(树状数组套线段树)： 94

伸展树(SPLAY): 98

动态树(LCT)： 101

树上哈希： 104

树链剖分： 105

单调栈： 109

单调队列： 112

虚树： 113

支配树(lengauer\_trajan)： 116

博弈论 120

巴什博弈： 120

威佐夫博弈（Wythoff's game）： 120

Nim博弈： 121

阶梯博弈： 121

Anti-nim游戏： 121

Fibonacci博弈: 121

SG组合游戏 122

SG值： 122

Anti-SG: 122

常用技巧、特定问题 123

约瑟夫环问题 123

N较小，复杂度O(n) 123

N较大,M较小，复杂度O(mlogn) 123

分数规划： 124

最优比率生成树： 124

挡板问题： 126

离散化： 126

输入·输出挂 by kuangbin： 126

杂项 128

优先队列自定义结构体： 128

Bitset： 128

特殊值、库函数 128

定理、性质篇： 130

数论： 130

欧拉函数性质： 130

原根： 130

二项式定理： 130

斐波那契数列性质： 130

Bertrand猜想： 130

费马小定理： 130

欧拉定理： 131

错排公式： 131

威尔逊定理： 131

约数个数： 131

约数和： 131

Zeckendorf定理（齐肯多夫定理）： 131

费马平方和定理 131

平方和 131

等差等比数列求和： 131

Farey序列 132

公式： 132

几何： 132

圆方程 132

叉积： 132

正弦定理 132

余弦定理 132

图论： 133

欧拉公式： 133

路径计数 133

Dilworth定理（狄尔沃斯定理） 133

Java大整数： 134

# 数学

## 卡特兰数

**h(n) = h(1) \*h(n-1)+h(2)\*h(n-2)+……+h(n-2)\*h(2)+h(n-1)\*(h1)**

第0项开始为：1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190,

另类递归式：**h(n)=h(n-1) \*(4\*n-2)/(n+1)**;

该递推关系的解为：**h(n)=C(2n ,n)/(n+1) (n=1,2,3,...)**

#include<stdio.h>

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//打表卡特兰数

//第 n个 卡特兰数存在a[n]中，a[n][0]表示长度；

//注意数是倒着存的，个位是 a[n][1] 输出时注意倒过来。

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int a[105][100];

void ktl()

{

    int i,j,yu,len;

    a[2][0]=1;

    a[2][1]=2;

    a[1][0]=1;

    a[1][1]=1;

    len=1;

    for(i=3;i<101;i++)

    {

        yu=0;

        for(j=1;j<=len;j++)

        {

            int t=(a[i-1][j])\*(4\*i-2)+yu;

            yu=t/10;

            a[i][j]=t%10;

        }

        while(yu)

        {

            a[i][++len]=yu%10;

            yu/=10;

        }

        for(j=len;j>=1;j--)

        {

            int t=a[i][j]+yu\*10;

            a[i][j]=t/(i+1);

            yu = t%(i+1);

        }

        while(!a[i][len])

        {

            len--;

        }

        a[i][0]=len;

    }

}

int main()

{

    ktl();

    int n;

    while(scanf("%d",&n)!=EOF)

    {

        for(int i=a[n][0];i>0;i--)

        {

            printf("%d",a[n][i]);

        }

        puts("");

    }

    return 0;

}

## 逆元

( a / b ) % p = a \* ( b ^ ( p – 2 ) ) % p

## N以内的素数和

LL a[maxn];

LL sum[maxn];

LL getindex(LL val, LL n, LL group, LL cnt)

{

if(val>=group)

return n/val-1;

return cnt-val;

}

LL getsum(LL n)

{

LL number = sqrt(n+0.5);

LL group = n/number;

LL cnt = number + group - 1;

for (LL i=0; i<cnt; i++)

{

if(i<number)

a[i] = n/(i+1);

else

a[i] = a[i-1] - 1;

sum[i] = a[i] \* (a[i] + 1) / 2 - 1;

}

for(LL i = 2; i <= number; i++)

{

if(sum[cnt-i]>sum[cnt-i+1])

{

LL tmpsum = sum[cnt-i+1];

LL tmp = i\*i;

for(LL j = 0; j < cnt; j++)

{

if(a[j]<tmp)

break;

LL index = getindex(a[j]/i, n, group, cnt);

sum[j] -= i\*(sum[index]-tmpsum);

}

}

}

return sum[0];

}

## 扩展欧几里德：

//求ax+by=gcd(a,b), a,b已知

//返回d = gcd(a,b)

//对于ax+by = c, 若c%d == 0,则有整数解

//解集为x1=x\*c/d+k\*b/d, y1 = y\*c/d-k\*a/d;

LL exgcd(LL a, LL b, LL &x, LL &y)

{

if(b == 0)

{

x = 1, y = 0;

return a;

}

LL d = exgcd(b, a%b, y, x);

y -= x\*(a/b);

return d;

}

## 扩展中国剩余定理：

求最下的x使满足x%m[i] = a[i]

LL m[maxn], a[maxn];

LL exCRT(int n)

{

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

{

LL c, e, x, y;

//求解k1\*m1-k2\*m2=a2-a1

c = exgcd(m[0], m[i], x, y);

e = a[i]-a[0];

if(e%c == 0)

{

//解正整数k1

x = e/c\*x%m[i];

while(x<0)x += m[i]/c;

//求k1\*m1+a1,同时使其对m2取余为a2

a[0] += m[0]\*x;

//将m1变为lcm(m1,m2);

m[0] = m[0]/c\*m[i];

a[0] = (a[0]%m[0]+m[0])%m[0];

}

else

return -1;

}

return a[0];

}

## EXBSGS：

已知x^y % p = z % p，求y

#define MOD 106711

int hs[MOD], hd[MOD], nex[MOD], id[MOD], top;

LL EXBSGS(LL x, LL z, LL p)

{

x %= p, z %= p;

if(z == 1)return x==0?-1:0;

if(x == 0)return z>1?-1:z==0&&p>1;

memset(hd, -1, sizeof(hd));

top = 1;

LL m, d, tmp1, tmp2, num = 0, xi = 1;

//当gcd(x,p)!=1 时，对公式进行转换

while((d = gcd(x, p))!= 1){

if(z == 1)return num;

if(z % d != 0)return -1;

z /= d, p /= d;

xi = x/d\*xi%p, num++;

}

m = sqrt(p\*1.0);

tmp1 = z, tmp2 = 1;

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

tmp1 = tmp1 \* x % p;

tmp2 = tmp2 \* x % p;

Insert(tmp1, i);

}

tmp1 = xi;

for(LL i=m;i<p;i+=m){

tmp1 = tmp2 \* tmp1 % p;

if((d=Find(tmp1))>-1)

return i+num-d;

}

return -1;

}

//插入键值对[x,y]

void Insert(LL x, LL y)

{

int k = x % MOD;

hs[top] = x, id[top] = y;

nex[top] = hd[k], hd[k] = top++;

}

//查询x对应的键值

int Find(int x){

int k = x%MOD;

for(int i=hd[k];i!=-1;i=nex[i])

{

if(hs[i] == x)return id[i];

}

return -1;

}

## 欧拉函数：

筛选欧拉函数

void euler()

{

int i, j, n = 66000;

memset(phi, 0, sizeof(phi));

phi[1] = 1;

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

if(!phi[i])

for(j=i;j<=n;j+=i){

if(!phi[j])

phi[j] = j;

phi[j] = phi[j]/i\*(i-1);

}

}

## 欧拉降幂：

https://img-blog.csdnimg.cn/20190512212812374.png

求aa[0]^aa[1]^aa[2]^…^aa[n-1]

Phi：i的欧拉函数，aa：幂， n：个数

LL phi[maxn], aa[maxn], n;

LL solve(int b, int n, LL m)

{

if(phi[m] == 1LL)return aa[b];

LL ans;

//如果只剩最后两项，就不用在递归了

if(b == n-2)

ans = aa[n-1];

else

ans = solve(b+1, n, phi[m]);

if(gcd(aa[b], m) == 1LL){

if(ans == 0)ans += phi[m];

ans = Pow(aa[b], ans%phi[m], m);

}

else if(ans < phi[m] && ans!= 0)

ans = Pow(aa[b], ans%phi[m], m);

else

ans = Pow(aa[b], ans%phi[m]+phi[m], m);

return ans;

}

## 线性筛：

筛选1~n内的素数，同时求每个数的欧拉函数。

int top, a[maxn], p[maxn], phi[maxn];

void line\_sieve(int n)

{

memset(a, 0, sizeof(a));

top = 0;

phi[1] = a[1] = 1;

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

{

if(!a[i])

{

p[top++] = i;

}

for(int j=0;j<top && (LL)i\*p[j]<n;j++)

{

a[i\*p[j]] = 1;

if(i%p[j] == 0)

{

phi[i\*p[j]] = phi[i]\*p[j];

break;

}

else phi[i\*p[j]] = phi[i]\*(p[j]-1);

}

}

}

## 母函数

//a为计算结果，b为中间结果。

int a[MAX],b[MAX];

//初始化a

memset(a,0,sizeof(a));

a[0]=1;

for (int i=1;i<=17;i++)//循环每个因子

{

memset(b,0,sizeof(b));

for (int j=n1[i];j<=n2[i]&&j\*v[i]<=P;j++)//循环每个因子的每一项

for (int k=0;k+j\*v[i]<=P;k++)//循环a的每个项

b[k+j\*v[i]]+=a[k];//把结果加到对应位

memcpy(a,b,sizeof(b));//b赋值给a

}

## 康托展开：

求一个排列是全排列的第num个

num = an\*(n-1)! + an-1\*(n-2)! +.....+ a1\* 0!

其中ai表示原排列中，在下标 i后面的，比下标i的数字还小的数字个数。

其中a存储原排列，fac为阶乘，c为标记数组

int contor(int n, int a[])

{

int num = 0, n1;

memset(c, 0, sizeof(c));

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

{

n1 = 0;

for(int j=1;j<a[i];j++)

if(!c[j])n1++;

num += fac[n-i]\*n1;

c[a[i]] = 1;

}

//从1开始数，所以num+1，如果从0开始，则不加1

return num+1;

}

## 逆康托展开：

求总共有n个数的全排列的第num个排列是多少，（从0开始计数）

对于第i位，设a = num / (n-i)!, b = num % (n-i)!

第a+1个未使用过的数即为当前位的数，b为下一位的num

void revcontor(int n, int a[], int num)

{

num--;

int i, j, n1;

memset(c, 0, sizeof(c));

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

{

n1 = num / fac[n-i] + 1;

for(j=1;;j++)

{

if(!c[j])n1--;

if(!n1)break;

}

a[i] = j;

num %= fac[n-i];

c[a[i]] = 1;

}

}

## 大数加法

Str为第一个加数，add为第二个加数，结果保存在str中，注意：str需初始化为0

void ADD(char str[], char add[])

{

int len1 = strlen(str), len2 = strlen(add),i;

revser(str, len1);

revser(add, len2);

for(i=0;i<len1;i++)str[i] -= '0';

for(i=len1;i<=len2;i++)str[i] = 0;

for(i=0;i<len2;i++)add[i] -= '0';

for(i=0;i<len2;i++)

str[i] += add[i];

len1 = max(len1, len2);

for(i=0;i<len1;i++)

if(str[i] > 9)str[i] %= 10, str[i+1]++;

if(str[len1] != 0)len1++;

revser(str, len1);

for(i=0;i<len1;i++)str[i] += '0';

//将add恢复原数组

for(i=0;i<len2;i++)add[i] += '0';

revser(add, len2);

}

void revser(char str[], int len)

{

for(int i=0;i<=len-1-i;i++)

{

char ch = str[len-1-i];

str[len-1-i] = str[i];

str[i] = ch;

}

}

递归写法：

void add(int i,int n){

if(sum[i] + n >= 10){

sum[i] = (sum[i]+n) % 10;

add(i+1,1);

}

else sum[i] += n;

}

while(gets(input)){

if(strcmp(input,"0") == 0) break;

int len = strlen(input);

for(int i=0; i<=len-1; i++)

add(i,input[len-1-i]-'0');

}

## 大数减法：

只适用于正整数减法，str1为较大的数，str2较小，结果保存在str1中。

void SUB(char str1[], char str2[])

{

int len1=strlen(str1), len2=strlen(str2), i;

if(len1 < len2 || (len1 == len2 && strcmp(str1, str2)<0))

swap(str1, str2), swap(len1, len2);

revser(str1, len1), revser(str2, len2);

for(i=0;i<len2;i++)

str1[i] -= str2[i];

for(;i<len1;i++)str1[i]-='0';

for(int i=0;i<len1;i++)

{

str1[i] += '0';

if(str1[i]<'0'){

str1[i] += 10;

str1[i+1]--;

}

}

//如果需要str2保持原数组

revser(str2, len2);

while(len1>1 && str1[len1-1] == '0'){

str1[len1-1] = 0;len1--;

}

revser(str1, len1);

}

## 大数乘法

浮点数乘法可以转化为大数乘法，去除小数位，记录小数点后有几位就行，注意str1和str2数组是否需要减去字符0

void Mulbigint(char str1[], char str2[], char str3[])

{

int i, j, len1 = strlen(str1), len2 = strlen(str2), len3=0;

revser(str1, len1), revser(str2, len2);

for(i=0;i<len1;i++)str1[i] -= '0';

for(i=0;i<len2;i++)str2[i] -= '0';

//str3可以在主函数里初始化，也可以在函数里初始化

//但不要使用sizeof，那样初始化会有问题

//memset(str3, 0, 1000);

for(i=0;i<len2;i++)

{

for(j=0;j<len1;j++)

str3[i+j] += str1[j] \* str2[i];

len3 = max(len3, i+len1);

for(j=0;j<len3;j++)

if(str3[j] > 9)

str3[j+1]+= str3[j]/10,str3[j]%=10;

while(str3[len3] > 0)len3++;

}

i = len3-1;

while(str3[i] == 0 && i!=0)len3--,i--;

for(i=0;i<len3;i++)str3[i] += '0';

revser(str3, len3);

//将原数组还原

revser(str1, len1), revser(str2, len2);

for(i=0;i<len1;i++)str1[i] += '0';

for(i=0;i<len2;i++)str2[i] += '0';

}

## 大数乘法(FFT)

maxn>=2^k>=len1+len2(len1,len2为两数长度)

const int maxn = 150;

const int mod = 1e9;

const double PI = acos(-1.0);

struct Complex{

double x, y;

Complex(double \_x=0.0, double \_y=0.0){

x = \_x;

y = \_y;

}

Complex operator -(const Complex &b)const{

return Complex(x-b.x, y-b.y);

}

Complex operator +(const Complex &b)const{

return Complex(x+b.x, y+b.y);

}

Complex operator \*(const Complex &b)const{

return Complex(x\*b.x-y\*b.y,x\*b.y+y\*b.x);

}

};

void change(Complex y[], int len){

int i, j, k;

for(i=1,j=len/2;i<len-1;i++){

if(i<j)swap(y[i], y[j]);

k = len/2;

while(j>=k){

j -= k;k/=2;

}

if(j<k)j+=k;

}

}

void fft(Complex y[], int len, int on)

{

change(y, len);

for(int h=2;h<=len;h<<=1){

Complex wn(cos(-on\*2\*PI/h), sin(-on\*2\*PI/h));

for(int j=0;j<len;j+=h){

Complex w(1, 0);

for(int k=j;k<j+h/2;k++){

Complex u = y[k];

Complex t = w\*y[k+h/2];

y[k] = u+t;

y[k+h/2] = u-t;

w = w\*wn;

}

}

}

if(on == -1)

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

y[i].x /= len;

}

Complex x1[maxn], x2[maxn];

char str1[maxn], str2[maxn];

int sum[maxn];

int main(){

while(scanf(" %s %s", str1, str2) == 2){

int len1 = strlen(str1);

int len2 = strlen(str2);

int len = 1;

while(len<len1\*2 || len<len2\*2)len<<=1;

for(int i=0;i<len1;i++)

x1[i] = Complex(str1[len1-1-i]-'0', 0);

for(int i=len1;i<len;i++)

x1[i] = Complex(0, 0);

for(int i=0;i<len2;i++)

x2[i] = Complex(str2[len2-1-i]-'0', 0);

for(int i=len2;i<len;i++)

x2[i] = Complex(0, 0);

fft(x1, len, 1);

fft(x2, len, 1);

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

x1[i] = x1[i] \* x2[i];

fft(x1, len, -1);

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

sum[i] = (int)(x1[i].x + 0.5);

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

sum[i+1] += sum[i]/10;

sum[i] %= 10;

}

len = len1+len2-1;

while(sum[len]<=0 && len>0)len--;

for(int i=len;i>=0;i--)

printf("%c", sum[i]+'0');

printf("\n");

}

}

## 大数开根号：

Str：原数n ans：sqrt(n)，num为ans长度，注意s的长度需要是2的倍数，不足前补0

初始o=2,I=0;

char str[maxn], ans[maxn];

int l, num;

int solve(int o, char \*s, int I)

{

char c, \*d = s;

if(o>0)

{

for(l=0;d[l];d[l++]-=10)

{

d[l++] -= 120;

d[l] -= 110;

while(!solve(0, s, l))

d[l]+=20;

ans[num++] = (d[l]+1032)/20;

}

}

else

{

c=o+(d[I]+82)%10-(I>l/2)\*(d[I-l+I]+72)/10-9;

d[I]+=I<0 ? 0 : !(o=solve(c/10,s,I-1))\*((c+999)%10-(d[I]+92)%10);

}

return o;

}

## FFT：

在使用时，一定要注意数据的范围，和Complex数组的范围

Complex x1[8\*maxn], x2[8\*maxn];

int p[maxn], a[maxn];

LL sum[8\*maxn], q[4\*maxn];

int main(){

int t, n, i, j, len1, len;

scanf("%d", &t);

while(t--){

LL au, ad;

scanf("%d", &n);

len = 1;

memset(p, 0, sizeof(p));

for(i=0;i<n;i++){

scanf("%d", &a[i]);

p[a[i]]++;

}

sort(a, a+n);

len1 = a[n-1]+1;

while( len < len1\*2)len<<=1;

for(i=0;i<len1;i++)

x1[i] = Complex(p[i], 0);

for(i=len1;i<len;i++)

x1[i] = Complex(0, 0);

//求DFT

fft(x1, len, 1);

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

x1[i] = x1[i]\*x1[i];

fft(x1, len, -1);

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

q[i] = (LL)(x1[i].x + 0.5);

for(i=0;i<n;i++)

q[a[i]+a[i]]--;

//len之前是2的整次幂可能很大，后面都是0

//实际上最多只到2\*a[n-1]

len = 2\*a[n-1];

sum[0] = 0;

for(i=1;i<=len;i++){

q[i] /= 2;

sum[i] = sum[i-1]+q[i];

}

au = 0;

//假设当前边作为最大边，能得到的数量

for(i=0;i<n;i++){

//可以选择两边之和大于a[i]的，但存在

//其他情况，需减去

au += sum[len]-sum[a[i]];

//减去一大一小的，

au -= (LL)(n-1LL-i)\*i;

//减去其自身

au -= n-1;

//减去两个都大于它的

au -= (LL)(n-1-i)\*(n-i-2)/2;

}

ad = (LL)n\*(n-1)\*(n-2)/6;

printf("%.7lf\n", au\*1.0/ad);

}

return 0;

}

## Pollard\_rho算法（大数分解素因子）

//接口为Find\_fac，参数n为需要分解的数

vector<LL> fac; //fac存储分解得到的素因子

void Find\_fac(LL n)

{

if(n == 1) return;

//确定当前数是否是素数

if(Miller\_Rabin(n))

{

fac.push\_back(n);

return;

}

LL p = n;

int z=0;

while(p>=n)

p = pollard\_rho(p, rand()%(n-1)+1);

Find\_fac(p);

Find\_fac(n/p);

}

//找出n的一个因子

LL pollard\_rho(LL n, LL c)

{

//随机生成x0

LL i=1, x0 = rand()%n;

LL y = x0, k =2;

while(1)

{

i++;

//+c!!+c!!+c！！！！！

x0 = (Mul(x0,x0, n)+c)%n;

LL d = gcd(y-x0, n);

//如果gcd(y-x0,n)不等于1或者n，则d是n的一个因子

if(d!=1 && d!=n)return d;

if(y == x0)return n;

if(i == k)

{

y = x0;

k+=k;

}

}

}

LL Pow(LL a, LL b, LL n)

{

b %= n;

LL m = 1;

while(b>0)

{

if(b&1)m=Mul(m, a, n);

b >>= 1;

a = Mul(a, a, n);

}

return m;

}

LL Mul(LL a, LL b, LL n)

{

a %= n, b %= n;

LL m = 0;

while(b>0)

{

if(b&1)

{

m += a;

if(m >= n)m-=n;

}

b=b>>1;

a <<= 1;

if(a >= n)a-=n;

}

return m;

}

//判断一个数是否为素数

bool Miller\_Rabin(LL n)

{

if(n == 2)

return true;

else if(n<2 || !(n&1))

return false;

//费马小定理，欧拉定理的特殊情况

LL m = n-1, j =0;

while(m%2==0)

{

m /= 2;

j++;

}

if(j>=1 && (m&1) == 1)

{

for(int i=1;i<20;i++)

{

LL a = rand()%(n-1)+1;

//求a^m%n

LL x = Pow(a, m, n);

LL y;

for(int k=0;k<j;k++)

{

//y=a^(2^(k+1)\*m)%n

y = Mul(x, x, n);

//如果y==1，但x!=1且x!=n-1，则

if(y==1 && x!=1 && x!=n-1)

return false;

x = y;

}

if(y!=1)return false;

}

}

return true;

}

LL gcd(LL a, LL b)

{

if(a<0)return gcd(-a, b);

LL k;

do

{

k = a % b;

a = b;

b = k;

}while(k!=0);

return a;

}

## 线性基:

线性基性质：

1. 线性基能相互异或得到原集合的所有相互异或得到的值。
2. 线性基是满足性质1的最小的集合
3. 线性基没有异或和为0的子集。

//求一组数进行异或能得到的最大值或求能得到的第k小的值

#include<cstdio>

#include<cstdlib>

#include<cmath>

#include<algorithm>

#include<ctype.h>

#include<cstring>

#include<queue>

#include<set>

#include<iostream>

#include<iterator>

#define INF 0x3f3f3f3f

using namespace std;

typedef long long int LL;

typedef pair<int, int> P;

const int maxn = 110;

LL maxx, ans, sig, kk, a[10020], b[66];

void insert(LL n);

int main()

{

int t, n, i, j, k, q;

scanf("%d", &t);

for(int z=1;z<=t;z++)

{

sig = 1;

scanf("%d", &n);

memset(b, 0, sizeof(b));

for(i=0;i<n;i++)

{

scanf("%lld", &a[i]);

insert(a[i]);

}

//将每个数尽量变为只有最高位为1

for(i=0;i<=62;i++)

if(b[i])

for(j=i-1;j>=0;j--)

if((b[i]>>j)&1 == 1)b[i]^=b[j];

vector<LL> g;

for(i=0;i<=62;i++)

if(b[i])g.push\_back(b[i]);

scanf("%d", &q);

printf("Case #%d:\n", z);

//能组成不同的数的数量为2^g.size()

maxx = 1LL<<g.size();

while(q--)

{

scanf("%lld", &kk);

if(!sig)kk--;

if(maxx<=kk)

printf("-1\n");

else

{

ans = i = 0;

while(kk>0)

{

if(kk%2)ans ^= g[i];

i++;

kk /= 2;

}

printf("%lld\n", ans);

}

}

}

return 0;

}

//构造线性基，设n的最高为1的位为第s位，如果

//"向量"组中存在第s位为1的数，则将n与此数异或，并

//继续重复此操作；如果不存在第s位为1的数，则放入n

//并退出循环

void insert(LL n)

{

for(int i=62;i>=0;i--)

{

if((n>>i)&1 == 1)

{

if(!b[i]){

b[i] = n;break;

}

else

n ^= b[i];

}

if(n == 0)

{

sig = 0;

break;

}

}

}

## 拉格朗日插值：

//拉格朗日插值

//已知n个点可以唯一确定一个函数

//求该函数在x=k时的值

//可以通过前缀后缀的方式来优化

#include<cstdio>

#include<cstdlib>

#include<cmath>

#include<string>

#include<ctype.h>

#include<cstring>

#include<stack>

#include<queue>

#include<iostream>

#include<iterator>

#define INF 100000000

using namespace std;

typedef long long LL;

typedef pair<int, int> P;

const int maxn = 2020;

const LL mod = 998244353;

LL Pow(LL a, LL b);

int main()

{

int n, i, j;

LL ans = 0, k, x[maxn], y[maxn];

scanf("%d %lld", &n, &k);

//函数上的n个点坐标

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

scanf("%lld %lld", &x[i], &y[i]);

/\*预处理快速求分子，然而没啥用，时间根本没少

LL bf[maxn], ed[maxn];

bf[0] = ed[n+1] = 1;

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

bf[i] = (bf[i-1]\*(k-x[i]+mod))%mod;

for(i=n;i>=1;i--)

ed[i] = (ed[i+1]\*(k-x[i]+mod))%mod;\*/

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

{

LL up, down;

down = 1;

up = (bf[i-1]\*ed[i+1])%mod;

for(j=1;j<=n;j++){

if(i == j)continue;

up = up \* (k-x[j]+mod) % mod;

down = down \* (x[i]-x[j]+mod) % mod;

}

LL lei = (y[i] \* up % mod)\*Pow(down, mod-2)%mod;

ans = (ans + lei) % mod;

}

printf("%lld\n", ans);

return 0;

}

LL Pow(LL a, LL b)

{

LL n=1;

while(b){

if(b%2)n = n\*a % mod;

a = a\*a % mod;

b /= 2;

}

return n;

}

## 高斯消元(浮点数)

#define eps 1e-8

const int maxn = 110;

//a:存储方程组，x:解集

double a[maxn][maxn], x[maxn];

int gauss(int n, int m);

//n:方程数量 m:变量数量

int gauss(int n, int m)

{

int i, j, k;

for(i=0;i<n;i++)

{

j=i;

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

if(fabs(a[k][i])>fabs(a[j][i]))

j = k;

if(j != i){

for(k=i;k<=m;k++){

double t=a[i][k];a[i][k]=a[j][k];

a[j][k] = t;

}

}

if(fabs(a[i][i])<eps)return 0;

for(j=i+1;j<=m;j++)

a[i][j] /= a[i][i];

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

if(fabs(a[j][i])>eps){

for(k=i+1;k<=m;k++)

a[j][k] -= a[j][i]\*a[i][k];

}

}

for(i=m-1;i>=0;i--){

double ans = a[i][m];

for(j=i+1;j<m;j++)

ans -= a[i][j]\*x[j];

x[i] = ans;

}

return 1;

}

## 高斯消元(开关问题)：

const int maxn = 15\*15+10;

int a[maxn][maxn], fre[maxn], x[maxn];

int dx[5]={0,-1,0,1,0}, dy[5]={-1,0,1,0,0};

vector<int> g;

void init(int n);

int gauss(int n, int m);

void pri(int n, int b[][maxn]);

int main()

{

int t, n, i, j, k;

char str[18][18];

scanf("%d", &t);

while(t--)

{

scanf("%d", &n);

int ed = n\*n;

//求方程组

init(n);

for(i=0;i<n;i++){

scanf(" %s", str[i]);

for(j=0;j<n;j++)

if(str[i][j] == 'w')

a[i\*n+j][ed] = 1;

}

int num = gauss(ed, ed);

if(num == -1)

printf("inf\n");

else if(num == 0){

for(i=0;i<ed;i++)num+=x[i];

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

}

else{

//枚举自由变量值

int ans = ed, tot=1<<num;

for(i=0;i<tot;i++)

{

int sum = 0;

for(j=0;j<num;j++)

if(i&(1<<j))

x[g[j]] = 1;

else

x[g[j]] = 0;

for(j=ed-1;j>=0;j--)

{

if(fre[j])continue;

int ts = a[j][ed];

for(k=j+1;k<ed;k++)

ts ^= a[j][k] & x[k];

x[j] = ts;

}

for(j=0;j<ed;j++)

sum += x[j];

ans = min(ans, sum);

}

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

}

}

return 0;

}

void init(int n)

{

int row;

memset(a, 0, sizeof(a));

for(int i=0;i<n;i++)

for(int j=0;j<n;j++)

{

row = i\*n+j;

for(int k=0;k<5;k++)

{

int x = i + dx[k], y = j + dy[k];

if(x>=0 && x<n && y>=0 && y<n)

a[row][x\*n+y] = 1;

}

}

}

int gauss(int n, int m)

{

int i, j, k, col;

memset(fre, 0, sizeof(fre));

memset(x, 0, sizeof(x));

for(i=col=0;i<n&&col<m;i++,col++)

{

j = i;

//01矩阵找到1就可以结束了的

for(k=j+1;k<n;k++)

if(a[j][col]<a[k][col])

j = k;

if(j!=i){

for(k=col;k<=m;k++){

int t=a[i][k];a[i][k]=a[j][k];

a[j][k] = t;

}

}

if(a[i][col] == 0){

fre[col] = 1;

g.push\_back(col);

i--;continue;

}

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

if(a[j][col] != 0)

{

for(k=col;k<=m;k++)

a[j][k] ^= a[i][k];

}

}

}

for(k=i;k<n;k++)

if(a[k][m]!=0)return -1;

if(i < m)return m-i;

for(i=m-1;i>=0;i--)

{

x[i] = a[i][m];

for(j=i+1;j<m;j++)

x[i] ^= a[i][j] & x[j];

}

return 0;

}

## 矩阵快速幂：

struct matrix{

int n;

LL a[2][2];

matrix(){}

matrix(int e, int b, int c, int d){

n = 2;

a[0][0] = e, a[0][1] = b;

a[1][0] = c, a[1][1] = d;

}

};

matrix mul(matrix A, matrix B);

matrix Pow(matrix a, LL b);

int main()

{

int i, x0, x1, a, b, len;

matrix A, B(1,0,0,1);

scanf("%d %d %d %d", &x0, &x1, &a, &b);

A.n = B.n = 2, A.a[0][0] = a, A.a[0][1] = b;

A.a[1][0] = 1, A.a[1][1] = 0;

scanf(" %s %d", str, &mod);

len = strlen(str);

for(i=len-1;i>=0;i--){

matrix C = Pow(A, str[i]-'0');

B = mul(B, C);

A = Pow(A, 10);

}

printf("%lld\n", (B.a[1][0]\*x1+B.a[1][1]\*x0)%mod);

return 0;

}

matrix mul(matrix A, matrix B){

matrix C(1,0,0,1);

memset(C.a, 0, sizeof(C.a));

for(int i=0;i<A.n;i++)

for(int j=0;j<B.n;j++){

//这里如果多此累加不会爆的话，尽量

//少做取余操作

C.a[i][j] = 0;

for(int k=0;k<A.n;k++)

C.a[i][j] += A.a[i][k]\*B.a[k][j];

C.a[i][j] %= mod;

}

return C;

}

matrix Pow(matrix A, LL b)

{

matrix C(1,0,0,1);

while(b > 0){

if(b&1) C = mul(C, A);

A = mul(A, A);

b >>= 1;

}

return C;

}

## Lucas定理：

返回C(n,m)%p,调用lucas函数

LL lucas(LL n, LL m, LL p)

{

if(m == 0)return 1;

return C(n%p, m%p, p)\*lucas(n/p, m/p, p)%p;

}

LL C(LL n, LL m, LL p)

{

LL up = 1, dn = 1;

for(int i=0;i<m;i++)

{

up = up\*(n-i)%p;

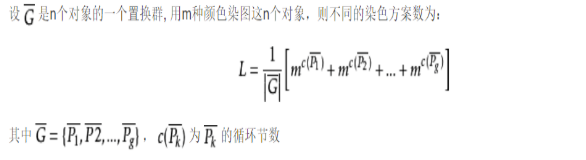
dn = dn\*(i+1)%p;

}

return up\*Pow(dn, p-2, p)%p;

}

## Polya定理：



### 旋转：

循环节数量为gcd(i,n)，1<=i<=n;

### 圆环翻转：

N为偶数

对称轴不过顶点：循环节：n / 2，n/2条对称轴

对称轴过顶点：循环节：(n / 2 +1)，n/2条对称轴

N为奇数：

循环节： (n + 1) / 2 ，n条对称轴

### N\*N大小的方阵：

置换为：{转0°，转90°，转180°，转270°}。

转0°：循环节为n\*n个

转180°：循环节为（n\*n+1）/2个

转90°，转270°:循环节为(n\*n+3)/4个

# 图论

## 前向星：

添加一条i –> j的边, h为存边的数组，nex为下一条边位置，to为实际的j点

注意nex，to数组应该比所有边的总数大

void add(int h[], int f, int t)

{

to[++tot] = t;

nex[tot] = h[f];

h[f] = tot;

}

## 最小树形图：

//求有向图上，以某点为根的最小树形图，是该点可以到达其余的所有点

//eg:边的信息，in:到i点最近的距离，pre:到i点最近的前驱

//vis:标记数组，id:重新编号

struct node{

int u, v, w;

}eg[maxn];

int in[120], pre[120], vis[120], id[120];

//n:点的数量，m:边的数量，root:求以root为根的树

LL DG\_MST(int n, int m,int root)

{

LL res = 0;

int i, j, cnt;

while(1)

{

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

in[i] = INF;

//遍历所有边，求到每个点最近的边(根节点除外)

for(i=1;i<=m;i++)

{

int u=eg[i].u, v=eg[i].v;

if(u!=v && eg[i].w<in[v])

in[v] = eg[i].w, pre[v] = u;

}

//除根节点外，有 点 无法被到达，则无最小树形图

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

if(i!=root && in[i] == INF)return -1;

cnt = 0;

memset(id, 0, sizeof(id));

memset(vis, 0, sizeof(vis));

//将取出边中形成环的部分，缩点

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

{

if(i == root)continue;

res += in[i];

int v = i;

while(vis[v]!=i && !id[v] && v!=root)

{

vis[v] = i;

v = pre[v];

}

if(v != root && !id[v])

{

id[v] = ++cnt;

for(int u=pre[v];u!=v;u=pre[u])

id[u] = cnt;

}

}

if(cnt == 0)break;

//将点重新编号

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

if(!id[i])id[i] = ++cnt;

for(i=1;i<=m;i++)

{

if(id[eg[i].u] != id[eg[i].v])

eg[i].w -= in[eg[i].v];

eg[i].v = id[eg[i].v];

eg[i].u = id[eg[i].u];

}

n = cnt;

root = id[root];

}

return res;

}

## SPFA：

算法时间复杂度不稳定，可以使用栈或队列完成，如果一个超时，不妨试试另一个

//vis：当前元素是否在队列(栈中)

//cnt：当前元素入栈次数,一个元素入队超过n次即存在负环

//st：模拟栈

int vis[maxn], cnt[maxn], st[maxn];

bool spfa(int u){

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

dis[i] = 10000000000000;

int l = 0;

st[++l] = u;

cnt[u]++, vis[u] = 1, dis[u] = 0;

while(l)

{

u = st[l--];

vis[u] = 0;

for(int i=g[u];i;i=nex[i])

if(dis[to[i]]>dis[u]+pi[i]){

dis[to[i]] = dis[u]+pi[i];

if(!vis[to[i]]){

vis[to[i]] = 1, cnt[to[i]]++;

if(cnt[to[i]]>n)return false;

st[++l] = to[i];

}

}

}

return true;

}

## 网络流最大流dinic:

网络流常用建图技巧：建立超级源点、超级汇点，拆点，分层建图

struct node{

int to, nex, cap, flow;

}eg[maxn\*maxn];

int ed, cnt, hd[maxn], dis[maxn], vis[maxn];

void init()

{

cnt = 1;

memset(hd, -1, sizeof(hd));

}

void add(int from, int to, int cap)

{

eg[++cnt].to = to;

eg[cnt].cap = cap;

eg[cnt].flow = 0;

eg[cnt].nex = hd[from];

hd[from] = cnt;

eg[++cnt].to = from;

eg[cnt].cap = 0;

eg[cnt].flow = 0;

eg[cnt].nex = hd[to];

hd[to] = cnt;

}

int dinic(int s, int t)

{

int res = 0, x=max(s,t);

while(bfs(s, t))

{

for(int i=0;i<=x;i++)

vis[i] = hd[i];

int d;

while((d=dfs(s, t, INF))>0)

res += d;

}

return res;

}

bool bfs(int s, int t)

{

int x = max(s, t);

for(int i=0;i<=x;i++)

dis[i] = -1;

queue<int> que;

que.push(s);

dis[s] = 0;

while(!que.empty())

{

int u = que.front();que.pop();

for(int i=hd[u];i!=-1;i=eg[i].nex)

{

node e = eg[i];

if(e.cap>e.flow && dis[e.to] == -1)

{

dis[e.to] = dis[u]+1;

que.push(e.to);

}

}

}

return dis[t] != -1;

}

int dfs(int s, int t, int ans)

{

if(s == t)return ans;

for(int &i=vis[s];i!=-1;i=eg[i].nex)

{

node e = eg[i];

if(e.cap > e.flow && dis[e.to] == dis[s]+1)

{

int d = dfs(e.to, t, min(ans, e.cap-e.flow));

if(d>0)

{

eg[i].flow += d;

eg[i^1].flow -= d;

return d;

}

}

}

return 0;

}

## 最小费用流：

### Dijsktra优化版：

//无向边的话，a-b流量为cost，可以建a->b cost，b->a –cost，b->a cost，a->b –cost;

struct node

{

int to, cap, cost, rev;

node(int a, int b, int c, int d):to(a),cap(b),cost(c),rev(d){

}

};

vector<node> g[maxn];

//pnum网络流构图中点的总个数

int pnum, dis[maxn], prevv[maxn], preve[maxn];

void add(int from, int to, int cost, int flow)

{

g[from].push\_back(node(to, flow, cost, g[to].size()));

g[to].push\_back(node(from, 0, -cost, g[from].size()-1));

}

int min\_cost\_flow(int s, int t, int f)

{

int res = 0;

//memset(h, 0, sizeof(h));

while(f>0)

{

priority\_queue<P, vector<P>, greater<P> > que;

fill(dis, dis+pnum+1, INF);

dis[s] = 0;

que.push(P(0, s));

while(!que.empty())

{

P p = que.top();que.pop();

int v = p.second;

if(dis[v]<p.first)continue;

int ss = g[v].size();

for(int i=0;i<ss;i++)

{

node &e = g[v][i];

if(e.cap>0 && dis[e.to]>dis[v]+e.cost)

{

dis[e.to] = dis[v]+e.cost;

prevv[e.to] = v;

preve[e.to] = i;

que.push(P(dis[e.to], e.to));

}

}

}

if(dis[t] == INF)return -1;

//for(int i=1;i<=n;i++)h[i] += dis[i];

int d = f;

for(int v=t;v!=s;v=prevv[v])

{

d = min(d, g[prevv[v]][preve[v]].cap);

}

f -= d;

res += d \* dis[t];

for(int v=t;v!=s; v=prevv[v])

{

node &e = g[prevv[v]][preve[v]];

e.cap -= d;

g[v][e.rev].cap += d;

}

}

return res;

}

### Spfa版：

求最小费用流，对于求最大费用，将费用取反，然后再将结果取反即可

struct node{

int to, nex, flow, cap, cost;

}eg[10\*maxn];

int tot, ed, pre[maxn], vis[maxn], hd[maxn], dis[maxn];

void init()

{

memset(hd, -1, sizeof(hd));

tot = 1;

}

void add(int f, int t, int cost, int cap)

{

eg[++tot].to = t;

eg[tot].cost = cost;

eg[tot].cap = cap;

eg[tot].nex = hd[f];

eg[tot].flow = 0;

hd[f] = tot;

eg[++tot].to = f;

eg[tot].cost = -cost;

eg[tot].cap = 0;

eg[tot].nex = hd[t];

eg[tot].flow = 0;

hd[t] = tot;

}

bool spfa(int s, int t)

{

for(int i=0;i<=t;i++)

dis[i] = INF, vis[i] = 0, pre[i] = -1;

queue<int> que;

que.push(s);

dis[s] = 0, vis[s] = 1;

while(!que.empty())

{

int u = que.front();que.pop();

vis[u] = 0;

for(int i=hd[u];i!=-1;i=eg[i].nex)

{

int v = eg[i].to;

if(eg[i].cap > eg[i].flow && dis[v]>dis[u]+eg[i].cost)

{

dis[v] = dis[u] + eg[i].cost;

pre[v] = i;

if(!vis[v]){

vis[v] = 1;

que.push(v);

}

}

}

}

if(dis[t] == INF)return false;

else return true;

}

//返回最大流，cost：最小费用

int min\_cost\_flow(int s, int t, int &cost)

{

int flow = 0;

cost = 0;

while(spfa(s, t))

{

int mi = INF;

for(int i=pre[t];i!=-1;i=pre[eg[i^1].to])

{

if(mi > eg[i].cap-eg[i].flow)

mi = eg[i].cap - eg[i].flow;

}

for(int i=pre[t];i!=-1;i=pre[eg[i^1].to])

{

eg[i].flow += mi;

eg[i^1].flow -= mi;

cost += eg[i].cost \* mi;

}

flow += mi;

}

return flow;

}

## 欧拉路径：

非递归求欧拉路径，数组b存储路径，路径从x点开始。

int a[maxn][maxn], b[maxn\*maxn], st[maxn\*maxn], c[maxn];

void getEuler(int x, int n)

{

int j, k, sig, cnt = 0, num = 0;

for(int i=1;i<=n;i++)c[i] = 1;

st[++cnt] = x;

while(cnt)

{

int sig = 0;

x = st[cnt--];

for(int &i=c[x];i<=n;i++)

if(a[x][i]){

a[x][i]--, a[i][x]--;

st[++cnt] =x;

st[++cnt] = i;

sig = 1;

break;

}

if(!sig)b[++num] = x;

}

}

## 混合欧拉：

//dinic代码同上

//首先判断图是否连通。然后将无向边任意定向,

//在网络流图中加入该边，流量为1(原点s,汇点t)。

//求每个点的入度与出度，然后两者相减为奇数，则不是欧拉图

//如果点入度in大于出度out，则加一条i到t，容量为(in-out)/2的边

//如果点入度in小于出度out，则加一条s到i，容量为(out-in)/2的边

//图若满流，则是欧拉图

//欧拉路径(摘自kuangbin博客)：

//先把图中的无向边随便定向，然后求每个点的D值，

//若有且只有两个点的初始D值为奇数，其余的点初始D值都为偶数，

//则有可能存在欧拉路径（否则不可能存在）。

//进一步，检查这两个初始D值为奇数的点，设为点i和点j，

//若有D[i]>0且D[j]<0，则i作起点j作终点（否则若D[i]与D[j]同号则不存在欧拉路径），

//连边<j, i>，求是否存在欧拉环即可（将求出的欧拉环中删去边<j, i>即可）。这样只需求一次最大流。

#include<cstdio>

#include<cstdlib>

#include<cmath>

#include<algorithm>

#include<ctype.h>

#include<cstring>

#include<queue>

#include<vector>

#include<iostream>

#include<iterator>

#define INF 0x3f3f3f3f

using namespace std;

typedef long long int LL;

typedef pair<int, int> P;

struct edg{

int f, t;

}p[1020];

struct node

{

int to, flow, rev;

node(int a, int b, int c):to(a),flow(b),rev(c){

}

};

vector<node> g[2020];

int vis[2020], dis[2020], a[220];

void add(int from, int to, int flow);

int dinic(int s, int t);

void bfs(int s, int t);

int dfs(int s, int t, int ans);

int Find(int x);

bool ishui(int n, int m);

int main()

{

int t, n, m, i, j, ans, num, in[220], out[220];

scanf("%d", &t);

while(t--)

{

memset(in, 0, sizeof(in));

memset(out, 0, sizeof(out));

ans = 1, num = 0;

scanf("%d %d", &n, &m);

for(i=0;i<=n+1;i++)g[i].clear();

for(i=0;i<m;i++)

{

int dir;

scanf("%d %d %d", &p[i].f, &p[i].t, &dir);

if(!dir)add(p[i].f, p[i].t, 1);

in[p[i].t]++, out[p[i].f]++;

}

//判断图是否连通

if(!ishui(n, m))ans = 0;

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

{

if(abs(in[i]-out[i])&1)ans = 0;

//建边

if(in[i]<out[i])add(0, i, (out[i]-in[i])/2);

if(in[i]>out[i])add(i, n+1, (in[i]-out[i])/2);

num += abs(in[i]-out[i])/2;

}

//判断满流

if(dinic(0, n+1)!=num/2)

ans = 0;

if(ans)printf("possible\n");

else printf("impossible\n");

}

return 0;

}

bool ishui(int n, int m)

{

int i, j, num = 0;

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

a[i] = i;

for(i=0;i<m;i++)

{

int x = Find(p[i].f), y = Find(p[i].t);

if(x!=y){

a[x] = y;

num++;

}

if(num == n-1)return true;

}

return false;

}

int Find(int x)

{

return a[x]=(a[x]==x?x:Find(a[x]));

}

void add(int from, int to, int flow)

{

g[from].push\_back(node(to, flow, g[to].size()));

g[to].push\_back(node(from, 0, g[from].size()-1));

}

## 二分图：

### 最小点覆盖：

点集中的点能覆盖所有边，最小点覆盖就是集合的点数最小的那个

(二分图)最小点覆盖 = 最大匹配

### 最小边覆盖：

边集中的边能覆盖所有边，最小边覆盖就是集合的边数最少的那个

(二分图)最小边覆盖 = 顶点数-最小点覆盖(最大匹配)

### 最大独立集：

定义：选出一些顶点使得这些顶点两两不相邻，则这些点构成的集合称为独立集。找出一个包含顶点数最多的独立集称为最大独立集。

(二分图)最大独立集 = 顶点数 - 最小顶点覆盖 = 顶点数 - 最大匹配

### DAG图上最小路径覆盖(路径不相交)：

定义：在DAG图中，找出最少的路径，使这些路径经过了所有点，路径之间不相交。

将每个点拆成两个点ux, uy，若存在边(u,v)，则连边ux-vy，新图是一个二分图，结果就是点的数量-最大匹配数。

### DAG图上最小路径覆盖(路径相交)：

路径可以相交，可以先利用算法(如floyd)求出传递闭包，就可以转化为路径不相交问题。

## 二分图最大匹配（匈牙利算法）：

时间复杂度：邻接矩阵：O(n3),邻接表：O(nm)

mat:二分图右部所匹配到的左边人的编号，

int n1, mat[maxn], vis[maxn];

//匈牙利算法，返回最大匹配数

int hungary(int n)

{

int tot = 0;

memset(mat, -1, sizeof(mat));

for(int i=1;i<=n1;i++)

{

memset(vis, 0, sizeof(vis));

if(match(i))tot++;

}

return tot;

}

bool match(int x)

{

for(int i=hd[x];i!=-1;i=eg[i].nex)

{

int u = eg[i].to;

if(!vis[u]){

vis[u] = 1;

//如果u本身为被匹配，或者匹配u的人可以匹配其他人

if(mat[u] == -1 || match(mat[u])){

mat[u] = x;

return true;

}

}

}

return false;

}

## 二分图带权最大匹配（KM算法）：

时间复杂度O(n3)，有些板子表面n3，但只适用于随机数据，

pop：点的数量，x,y:标杆，w：匹配权值，

LL lx, ly, pop, x[maxn], y[maxn], w[maxn][maxn], par[maxn];

LL sonx[maxn], sony[maxn], sla[maxn], prex[maxn], prey[maxn];

void adjust(int v)

{

sony[v] = prey[v];

if(prex[sony[v]] != -2)

adjust(prex[sony[v]]);

}

bool find(int v)

{

int i;

for(i=0;i<pop;i++)

{

if(prey[i] == -1){

if(sla[i] > x[v]+y[i]-w[v][i]){

sla[i] = x[v]+y[i]-w[v][i];

par[i] = v;

}

if(x[v] + y[i] == w[v][i]){

prey[i] = v;

if(sony[i] == -1){

adjust(i);return true;

}

if(prex[sony[i]] != -1)continue;

prex[sony[i]] = i;

if(find(sony[i]))return true;

}

}

}

return false;

}

LL km()

{

int i, j;

LL m;

for(i=0;i<pop;i++){

sony[i] = -1;

y[i] = 0;

}

for(i=0;i<pop;i++){

x[i] = 0;

for(j=0;j<pop;j++)

x[i] = max(x[i], w[i][j]);

}

bool flag;

for(i=0;i<pop;i++)

{

for(j=0;j<pop;j++){

prex[j] = prey[j] = -1;

sla[j] = INF;

}

prex[i] = -2;

if(find(i))continue;

flag = false;

while(!flag)

{

m = INF;

for(j=0;j<pop;j++)

if(prey[j] == -1)m = min(m, sla[j]);

for(j=0;j<pop;j++)

{

if(prex[j] != -1)x[j] -= m;

if(prey[j] != -1)y[j] += m;

else sla[j] -= m;

}

for(j=0;j<pop;j++)

if(prey[j] == -1 && !sla[j])

{

prey[j] = par[j];

if(sony[j] == -1)

{

adjust(j);

flag = true;

break;

}

prex[sony[j]] = j;

if(find(sony[j])){

flag = true;

break;

}

}

}

}

LL ans = 0;

for(int i=0;i<pop;i++)

ans += w[sony[i]][i];

return ans;

}

## 一般图最大匹配（带花树算法）：

时间复杂度：O(n^3)

有些东西和匈牙利算法差不多，主要就是处理奇环，将其缩为一个点。

struct node{

int to, nex;

}eg[8\*maxn];

int cnt, hd[maxn], dx[4]={-1,0,1,0}, dy[4]={0,-1,0,1};

int n, ml, fs[maxn], nt[18\*maxn], dt[18\*maxn], pre[maxn], match[maxn], f[maxn], bz[maxn], bp[maxn], ti, d[maxn];

bool check[maxn], treec[maxn], pathc[maxn];

char str[52][52];

int main()

{

int t, m, i, j, k, num;

scanf("%d", &t);

while(t--)

{

scanf("%d %d", &n, &m);

num = 0;

init();

for(i=0;i<n;i++)

scanf(" %s", str[i]);

for(i=0;i<n;i++)

for(j=0;j<m;j++)

if(str[i][j] == '\*')

{

num++;

for(k=0;k<4;k++){

int nx = i+dx[k], ny = j+dy[k];

if(nx>=0 && nx<n && ny>=0 &&ny<m && str[nx][ny] == '\*')

add(i\*m+j+1, nx\*m+ny+1);

}

}

n = n\*m;

int ans = 0;

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

if(!match[i])ans += find(i);

printf("%d\n", num-ans);

}

return 0;

}

void init()

{

cnt = ti = 0;

memset(hd, -1, sizeof(hd));

memset(match, 0, sizeof(match));

}

void add(int f, int t)

{

eg[++cnt].to = t;

eg[cnt].nex = hd[f];

hd[f] = cnt;

}

int getf(int k){

return f[k] = k==f[k]?k:getf(f[k]);

}

int lca(int x, int y)

{

ti++; x = getf(x), y = getf(y);

while(bp[x] != ti)

{

bp[x] = ti;

x = getf(pre[match[x]]);

if(y)swap(x, y);

}

return x;

}

void make(int x, int y, int w)

{

while(getf(x) != w)

{

pre[x] = y, y = match[x];

if(bz[y] == 2)bz[y] = 1, d[++d[0]] = y;

if(getf(x) == x)f[x] = w;

if(getf(y) == y)f[y] = w;

x = pre[y];

}

}

int find(int st)

{

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

f[i] = i, pre[i] = bz[i] = 0;

d[d[0] = 1] = st, bz[st] = 1;

int l = 0;

while(l<d[0])

{

int k = d[++l];

for(int i=hd[k];i != -1;i=eg[i].nex)

{

int p = eg[i].to;

if(getf(p) == getf(k) || bz[p] == 2)continue;

if(!bz[p]){

bz[p] = 2, pre[p] = k;

if(!match[p])

{

for(int x = p, y;x;x = y)

y = match[pre[x]], match[x]=pre[x], match[pre[x]] = x;

return 1;

}

bz[match[p]] = 1, d[++d[0]] = match[p];

}

else

{

int w = lca(k, p);

make(k, p, w);

make(p, k, w);

}

}

}

return 0;

}

## Tarjan算法：

### 求割点，桥，vis数组需初始化为0

其实这里的vis数组可以用dfn数组代替，如果是无向图求割点注意当前节点不能访问其父节点，其他的与有向图都一样

//dfn：节点首次出现的时间

//low：节点能访问到的最早出现的时间点

//a：是否为割点

int con, vis[maxn], par[maxn], low[maxn], dfn[maxn], a[maxn];

void tarjan(int u);

void tarjan(int u)

{

int sum = 0;//sum为子树的数量

vis[u] = 1;

dfn[u] = low[u] = ++con;

for(int i=0;i<g[u].size();i++)

{

if(!vis[g[u][i]])

{

sum++;

par[g[u][i]] = u;

tarjan(g[u][i]);

low[u] = min(low[u], low[g[u][i]]);

//如果是最开始访问的那个节点并且有两颗子树，则为割点

if(par[u] == -1 && sum>1)

a[u] = 1;

//如果不是起始点，且u的子节点中没有能返回u及其父节点的边，则为割点

else if(par[u]!=-1 && low[g[u][i]]>=dfn[u])

a[u] = 1;

//dfn[u]<low[v]如果成立则边(u,v)为桥

}

else

low[u] = min(low[u], dfn[g[u][i]]);

}

}

### 强连通分量，缩点

**缩完点后需要将不同颜色的点之间的边加到sg图中**

//g为原图，sg为缩点后的图

vector<int> g[maxn], sg[maxn];

int vis[maxn], low[maxn], dfn[maxn], in[maxn], out[maxn];

//color为缩点后的颜色，值相同代表在一个点

//sum为缩点后的图不同颜色的数量(即点的数量)

int con, top, sum, color[maxn], stack[maxn];

void init();

void tarjan(int u);

int main()

{

int n, m, i, j, k, ans=0, col;

scanf("%d %d", &n, &m);

init();

while(m--)

{

scanf("%d %d", &i, &j);

g[i].push\_back(j);

}

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

if(!dfn[i])tarjan(i);

//重构图

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

for(j=0;j<g[i].size();j++){

if(color[i] != color[g[i][j]]){

out[color[i]]++;

sg[color[i]].push\_back(color[g[i][j]]);

}

}

//当只有一个出度为0的点时，存在都崇拜它的牛

for(i=1;i<=sum;i++)

if(out[i] == 0){

ans++;

col = i;

}

if(ans != 1)

printf("0\n");

else{

ans = 0;

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

if(color[i] == col)ans++;

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

}

return 0;

}

//初始化

void init()

{

top = 0;

sum = 0;

memset(vis, 0, sizeof(vis));

memset(dfn, 0, sizeof(dfn));

memset(color, 0, sizeof(color));

memset(out, 0, sizeof(out));

}

void tarjan(int u)

{

vis[u] = 1;

dfn[u] = low[u] = ++con;

stack[++top] = u;

for(int i=0;i<g[u].size();i++)

{

int v = g[u][i];

if(!dfn[v])//如果未访问过

{

tarjan(v);

low[u] = min(low[u], low[v]);

}

//如果访问过且在栈里

else if(dfn[v] && vis[v])

low[u] = min(low[u], dfn[v]);

}

if(dfn[u] == low[u])

{

color[u] = ++sum;

vis[u] = 0;

while(stack[top] != u)

{

vis[stack[top]] = 0;

color[stack[top--]] = sum;

}

top--;

}

}

### 离线求LCA

时间复杂度为O（n+q），

hd：存储树的边信息，qe：存储询问信息

vis需初始化为0

//查询u的所有子树

void tarjan\_lca(int u)

{

vis[u] = 1;

fa[u] = u;

//访问其所有子节点，

for(int i=hd[u];i;i=nex[i]){

int v = to[i];

if(!vis[v]){

dis[v] = dis[u] + id[i];

tarjan\_lca(v);

fa[v] = u;

}

}

//访问和u有关的所有询问

for(int i=qe[u];i;i=nex[i]){

int v = to[i];

//如果另一个节点被访问过

//则另一个节点所在的并查集根节点即为LCA

if(vis[v]){

int tmp = get\_fa(v);

ans[id[i]] = dis[u]+dis[v]-2\*dis[tmp];

}

}

}

//求x的所在并查集的根节点

int get\_fa(int x)

{

if(x == fa[x])return x;

return fa[x] = get\_fa(fa[x]);

}

## 2-SAT：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

边i->j代表选i必须选j，部分形式构图参考如下：

i或j：非i-> j， 非j->i

i必选：非i -> i

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

int n, m, i, j, a1, a2, b1, b2;

while(~scanf("%d %d", &n, &m))

{

init(n);

for(i=0;i<m;i++){

scanf("%d %d", &p[i].first, &p[i].second);

if(p[i].first>p[i].second)swap(p[i].first, p[i].second);

}

for(i=0;i<m;i++)

for(j=i+1;j<m;j++)

{

int a=p[i].first, b=p[i].second, c=p[j].first, d=p[j].second;

if(isok(a, c, d) && b>d || isok(b, c, d)&& a< c){

add(2\*i, 2\*j+1);

add(2\*j+1, 2\*i);

add(2\*j, 2\*i+1);

add(2\*i+1, 2\*j);

}

}

for(i=0;i<m;i++){

if(!dfn[2\*i])trajan(2\*i);

if(!dfn[2\*i+1])trajan(2\*i+1);

}

for(i=0;i<m;i++)

//如果i和非i在同一scc内，则不成立

if(color[2\*i] == color[2\*i+1])

break;

/\*可以输出一组可行解

对缩点后的DAG图逆序拓扑， 可以得一组解

for(i=0;i<m;i++)

if(color[2\*i] < color[2\*i-1])printf("%d\n", 2\*i);

else printf("%d\n", 2\*i-1);\*/

if(i == m)

printf("panda is telling the truth...\n");

else

printf("the evil panda is lying again\n");

}

return 0;

}

bool isok(int a, int b, int c)

{

return a>b && a<c;

}

//以下包含trajan缩点和前向星建边函数

## LCA(最近公共祖先)：

//白书p329

const int maxn = 920;

const int maxv = 12;//1<<maxv > maxn

//图的邻接表示，

vector<int> g[maxn];

//root:根节点;dep:深度;par[i][j]:j往上走2^i步的节点

//(超过根节点为-1)

int root, dep[maxn], par[maxv+1][maxn];

//初始化深度和往上走2^1步的点

void dfs(int v, int p, int d){

par[0][v] = p;

dep[v] = d;

for(int i=0;i<g[v].size();i++)

if(g[v][i] != p)dfs(g[v][i], v, d+1);

}

void init(int n)

{

dfs(root, -1, 0);

for(int k=0;k<maxv;k++)

{

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

if(par[k][v] < 0)par[k+1][v] = -1;

else par[k+1][v] = par[k][par[k][v]];

}

}

int lca(int u, int v)

{

if(dep[u]>dep[v])

{

int t=u;

u=v;

v=t;

}

//设dep[v]-dep[u] = x, 往上走x步，即可保证

//两点在同一深度，v=par[k][v]即v向上走2^k步

for(int k=0;k<maxv;k++){

if((dep[v]-dep[u]) >> k & 1)

v = par[k][v];

}

if(u == v)return u;

for(int k=maxv-1;k>=0;k--)

if(par[k][u] != par[k][v]){

u = par[k][u];

v = par[k][v];

}

return par[0][u];

}

## 树上差分：

//其实这东西完全可以用树链剖分代替

//将s到t路径上的所有点(边)加固定值

//假设a[]存储边权值

//边：a[s]++,a[t++],a[lca(s,t)]-=2；

//两端点加1，lca减2

//点：a[s]++,a[t++],a[lca(s,t)]--,a[fa[lca(s,t)]]—

//两端点加1，lca和lca父结点减1

const int maxn = 100100;

struct node{

int num;

node\* nex;

};

int m, a[maxn], root, dep[maxn], par[maxn][20];

LL ans;

node\* head[maxn];

void dfs(int v, int p, int d);

void init(int n);

int lca(int u, int v);

void end\_dfs(int u);

void add(int f, int t);

int main()

{

int n, i, f, t;

scanf("%d %d", &n, &m);

for(i=0;i<=n;i++)

head[i] = NULL;

ans = 0;

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

scanf("%d %d", &f, &t);

add(f, t);

add(t, f);

}

root = 1;

init(n);

for(i=1;i<=m;i++)

{

scanf("%d %d", &f, &t);

if(f == t)continue;

int fa = lca(f, t);

//将其两个结点增加1，lca减2

a[f]++, a[t]++, a[fa]-=2;

}

end\_dfs(1);

printf("%lld\n", ans);

return 0;

}

void add(int f, int t)

{

node \*p = new node;

p->nex = head[f];

p->num = t;

head[f] = p;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

此部分为上面LCA算法函数实现，具体参照以上LCA(最近公共祖先)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//递归访问求各结点的值

void end\_dfs(int u)

{

node \*p;

//当前结点的值等于其所有子结点的值和加上自身值

for(p = head[u];p!=NULL;p=p->nex)

{

if(par[u][0] == p->num)continue;

end\_dfs(p->num);

int &e = a[p->num];

a[u] += e;

if(e == 0)ans+=m;

else if(e == 1)ans++;

}

}

## 树的重心：

定义：删除此节点后剩下的最大连通块最小

树的重心的性质：

1.树中所有点到某个点的距离和中，到重心的距离和最小，如果有两个重心，他们的距离和一样。

2.把两棵树通过一条链相连，新的树的重心在原来两颗树重心的连线上

3.一棵树添加或删除一个节点，树的重心最多只移动一条边的位置

4.一棵树最多有两个重心，且相邻

5. 以一棵树的重心为根的子树的节点个数，一定大于等于节点总数的一半

6. 在一棵树的所有子树中，找到一棵树使其节点数恰好大于等于原树节点总数的一半(满足"节点数大于等于原树节点总数一半"这个条件的子树节点数最小的子树)，那么该子树的根一定是一个重心。

//si[i]：以i为根的子树的节点数量

求一颗树以i号节点为根的子树的重心

int fa[maxn], si[maxn], ans[maxn];

vector<int> g[maxn];

void dfs(int u, int step)

{

LL mi;

int a1, a2;

si[u] = 1;

ans[u] = u;

for(int i=0;i<g[u].size();i++)

{

int v = g[u][i];

dfs(v, step+1);

si[u] += si[v];

}

//重心所在的子树的节点数量必定大于等于当前树节点数的一半

for(int i=0;i<g[u].size();i++)

{

if(si[g[u][i]]\*2 > si[u])

ans[u] = ans[g[u][i]];

}

//找子树节点最小的满足节点数大于等于原树节点一半的点

while((si[u]-si[ans[u]])\*2>si[u])

ans[u] = fa[ans[u]];

}

## Matrix\_tree定理：

Kirchhoff矩阵任意n-1阶子矩阵的行列式的绝对值就是无向图的生成树的数量。

Kirchhoff矩阵的定义是度数矩阵-邻接矩阵。

1、G的度数矩阵D[G]：n\*n的矩阵，Dii等于Vi的度数，其余为0。  
2、G的邻接矩阵A[G]：n\*n的矩阵， Vi、Vj之间有边直接相连，则 Aij=ij之间的边数，否则为0。

int id[maxn][maxn];

LL a[maxn][maxn];

int dx[4]={0,-1,0,1}, dy[4]={-1,0,1,0};

LL det(int n);

void getgrape(int n, int m, char str[][12]);

int main()

{

int n, m, i, j, num = 0;

char str[12][12];

scanf("%d %d", &n, &m);

for(i=0;i<n;i++){

scanf("%s", str[i]);

for(j=0;j<m;j++)

if(str[i][j]=='.')

id[i][j] = num++;

}

getgrape(n, m, str);

LL ans = det(num-1);

printf("%d", ans);

return 0;

}

//求Kirchhoff矩阵

void getgrape(int n, int m, char str[][12])

{

int i, j, k, nx, ny;

memset(a, 0, sizeof(a));

for(i=0;i<n;i++)

for(j=0;j<m;j++){

if(str[i][j] != '\*'){

for(k=0;k<4;k++){

nx = i + dx[k];

ny = j + dy[k];

if(nx>=0 && nx<n && ny>=0 && ny<m && str[nx][ny]=='.'){

a[id[i][j]][id[i][j]]++,a[id[i][j]][id[nx][ny]]--;

}

}

}

}

}

//高斯消元辗转消元法

//可避免精度问题，但时间复杂度为n\*n\*n\*log n

LL det(int n){

LL i, j, k, t, ret=1;

for(i=0;i<n;i++)

for(j=0;j<n;j++)

a[i][j] %= mod;

for(i=0;i<n;i++){

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

while(a[j][i]){

t = a[i][i] / a[j][i];

for(k=i;k<n;k++)

a[i][k] = (a[i][k]-a[j][k]\*t)%mod;

swap(a[i], a[j]);

ret = -ret;

}

if(a[i][i] == 0)

return 0LL;

ret = (ret\*a[i][i]) % mod;

}

return (ret+mod)%mod;

}

# 动态规划

## 区间dp:

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

{

dp[i][i]=初始值

}

for(int len=2;len<=n;len++) //区间长度

for(int i=1;i<=n;i++) //枚举起点

{

int j=i+len-1; //区间终点

if(j>n) break; //越界结束

for(int k=i;k<j;k++) //枚举分割点，构造状态转移方程

{

dp[i][j]=max(dp[i][j],dp[i][k]+dp[k+1][j]+w[i][j]);

}

}

## 数位dp：

int a[100];

LL n, dp[100][12], z[20];

int main()

{

z[0] = 1;

for(int i=1;i<20;i++)

z[i] = z[i-1]\*10;

int t;

while(~scanf("%d", &t))

{

while(t--)

{

memset(dp, -1, sizeof(dp));

scanf("%lld", &n);

//LL ans = n-solve(n)+1;

LL ans = solve();

printf("%lld\n", ans);

}

}

return 0;

}

LL solve()

{

LL s = n;

int k = 0;

while(s>0)

{

a[k] = s % 10;

k++;

s /= 10;

}

LL ans = dfs(k-1, 0, 1);

/\*for(int i=0;i<k;i++)

{

for(int j=0;j<=9;j++)

printf("%d ", dp[i][j]);

printf("\n");

}\*/

return ans;

}

//直接求包含49的个数

LL dfs(int pos, int bf, bool limit)

{

if(pos == -1)return 0;

if(!limit && dp[pos][bf] != -1)return dp[pos][bf];

LL ans = 0;

int s = limit?a[pos]:9;

for(int i=0;i<=s;i++)

//如果这两位直接就是49，如果是上限则加上余数，

//不是上限则加上所有后面的数

if(bf == 4 && i == 9)

ans += limit?n%z[pos]+1:z[pos];

else

ans += dfs(pos-1, i, limit && i == a[pos]);

if(!limit)

dp[pos][bf] = ans;

return ans;

}

/\*求不包含49的个数

LL dfs(int pos, int bf, bool limit)

{

if(pos == -1)return 1;

if(!limit && dp[pos][bf] != -1)return dp[pos][bf];

LL ans = 0;

int s = limit?a[pos]:9;

for(int i=0;i<=s;i++)

if(bf == 4 && i == 9);

else

ans += dfs(pos-1, i, limit && i==a[pos]);

if(!limit)

dp[pos][bf] = ans;

return ans;

}

\*/

# 计算几何

## 二维几何：

### 精度控制：

int sgn(double x)

{

if(x < -1e-9)return -1;

else if(x > 1e-9)return 1;

return 0;

}

### 点类：

struct point{

double x, y;

point( ){}

point( double a, double b):x(a),y(b){}

point operator +(point b)const{

return point(x+b.x, y+b.y);

}

point operator -(point b)const{

return point(x-b.x, y-b.y);

}

//点积

double operator \*(point b)const{

return x\*b.x + y\*b.y;

}

//叉积

double operator ^(point b)const{

return x\*b.y-y\*b.x;

}

};

## 点对线

### 点a与直线k1k2垂足的坐标

point proj(point a, point k1, point k2)

{

point res;

double dx = k1.x - k2.x;

double dy = k1.y - k2.y;

double u = (a.x - k1.x)\*(k1.x - k2.x) +

(a.y - k1.y)\*(k1.y - k2.y);

u = u/((dx\*dx)+(dy\*dy));

res.x = k1.x + u\*dx;

res.y = k1.y + u\*dy;

return res;

}

### 判断点p是否在线段ab上

bool OnSeg(point p, point a, point b)

{

return sgn((a-p)^(b-p))==0 &&

sgn((p.x-a.x)\*(p.x-b.x))<=0 &&

sgn((p.y-a.y)\*(p.y-b.y))<=0;

}

### 点p到线段ab最短的距离

//返回最近的的坐标

point PointToLine(point p, point a, point b)

{

point result;

double t = ((p-a)\*(b-a))/((b-a)\*(b-a));

if(t>=0 && t<=1)

{

result.x = a.x+(b.x-a.x)\*t;

result.y = a.y+(b.y-a.y)\*t;

}

else

{

if(dist(p, a)<dist(p, b))

result = a;

else

result = b;

}

return result;

}

### 点a关于k1，k2对称的点

point reflect(point a,point k1,point k2)

{

point p3 = proj(a, k1, k2);

return p3+p3-a;

}

## 线对线

### 两直线垂直或平行

//向量平行，叉积为0，向量垂直，点积为0

int ParOrt(point a, point b, point c, point d)

{

if(sgn((b-a)^(d-c)) == 0)return 2;

if(sgn((b-a)\*(d-c)) == 0)return 1;

return 0;

}

### 判断线段ab与线段cd是否有交点

int intersect(double l1,double r1,double l2,double r2){//快速排斥实验

if (l1>r1) swap(l1,r1); if (l2>r2) swap(l2,r2); return sgn(r1-l2)!=-1&&sgn(r2-l1)!=-1;

}

bool IsCom(point a, point b, point c, point d)

{

if(!(intersect(a.x, b.x, c.x, d.x) && intersect(a.y, b.y, c.y, d.y)))return false;

double d1 = (b-c)^(d-c), d2 = (a-c)^(d-c);

double d3 = (c-a)^(b-a), d4 = (d-a)^(b-a);

if(sgn(d1\*d2)<= 0 && sgn(d3\*d4)<=0)

return true;

else return false;

}

### 判断直线ab与直线cd是否有交点

bool iscom(point a, point b, point c, point d)

{

//如果c、d都在直线的同一边则无交点

double tmp = Cross(a, b, c)\*Cross(a, b, d);

return tmp < 0.0 || fabs(tmp)<1e-6;

}

### 直线ab与线段cd交点坐标

//求直线ab与线段cd交点坐标

double getPoint(point a, point b, point c, point d){

double a1 = a.x, b1 = a.y;

double a2 = b.x, b2 = b.y;

double a3 = c.x, b3 = c.y;

double a4 = d.x, b4 = d.y;

double t=((a2-a1)\*(b3-b1)-(a3-a1)\*(b2-b1))/((a2-a1)\*(b3-b4)-(a3-a4)\*(b2-b1));

//交点坐标x: a3+t\*(a4-a3), y: b3+t\*(b4-b3)

return a3 + t\*(a4-a3);

}

### 求向量p0p1 与 向量p0p2 位置关系

int LineStu(point p1, point p2, point p0)

{

double x = (p1-p0)^(p2-p0);

if(sgn(x) > 0)return 1;//p1位于p2顺时针方向

else if(sgn(x) < 0)return 2;//p1位于p2逆时针方向

if(((p1-p0)\*(p2-p0)) < 0)return 3;//p1,p2反向

else if(sgn((p1.x-p0.x)\*(p1.x-p2.x))<=0 && sgn((p1.y - p0.y)\*(p1.y-p2.y))<=0)return 5;//p1,p2同向，p1较短

else return 4;//p1,p2同向，p2较短

}

### 求向量ab与向量ac夹角

double getAng(point a, point b, point c)

{

point p1 = c-a, p2 = b-a;

return acos(p1\*p2/(p1.dis()\*p2.dis()));

}

## 极角排序：

### 利用叉积进行排序

C为极点，PI角度内，可以得到正确结果

bool cmp(point a, point b)

{

if(((a-c)^(b-c)) == 0){

return c.dis(a)<c.dis(b);

}

else

return ((a-c)^(b-c))>0;

}

## 判断是否是凸包

判断某种顺序点0~n-1连边，是否是凸包

bool IsTu(point poly[], int n)

{

int af = 0;

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

int tmp = sgn(Cross(poly[i], poly[(i+1)%n], poly[(i+2)%n]));

if(!af)af = tmp;

if(af\*tmp < 0){

return false;

}

}

return true;

}

//求向量ab与向量bc方向关系

double Cross(point a, point b, point c)

{

return (b-a)^(c-b);

}

## 求任意多边形重心：

struct point{

double x, y;

}p[maxn];

void getzhong(int n, double &x, double &y, double &sumarea)

{

x = y = sumarea = 0.0;

double area;

point p0=p[0], p1=p[1];

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

area = getArea(p0, p1, p[i]);

sumarea += area;

x += (p0.x+p1.x+p[i].x)\*area;

y += (p0.y+p1.y+p[i].y)\*area;

p1 = p[i];

}

x = x/sumarea/3, y = y/sumarea/3;

}

double getArea(point a, point b, point c)

{

double area = 0;

area = a.x\*b.y+b.x\*c.y+c.x\*a.y-a.y\*b.x-b.y\*c.x-c.y\*a.x;

return area\*0.5;

}

## 判断P点是否在任意多边形内

Poly是各点的坐标，从0~n-1，注意共线边

返回值：

0：P在边上

1：P在多边形内

-1：P在多边形外

int inPoly(point p, point poly[], int n)

{

int cnt = 0;

point ray1, ray2, side1, side2;

ray1 = p, ray2.y = p.y, ray2.x = -1000000000.0;

for(int i=0;i<n;i++)

{

if(sgn((poly[i]-poly[(i+1)%n])^(poly[i]-poly[(i+2)%n])) == 0)continue;

side1 = poly[i], side2 = poly[(i+1)%n];

if(OnSeg(p, side1, side2))return 0;

if(sgn(side1.y - side2.y) == 0)

continue;

if(OnSeg(side1, ray1, ray2))

{

if(sgn(side1.y-side2.y)>0)cnt++;

}

else if(OnSeg(side2, ray1, ray2))

{

if(sgn(side2.y-side1.y)>0)cnt++;

}

else if(IsCom(ray1, ray2, side1, side2))

cnt++;

}

if(cnt%2 == 1)return 1;

return -1;

}

## 最近点对

struct node{

int x, y;

}p[maxn];

int b[maxn];

LL solve(int l, int r)

{

if(l == r)return 1e18;

else if(l+1 == r)return dist(p[l], p[r]);

int mid = (l+r)/2;

LL d = min(solve(l, mid), solve(mid+1, r));

int cnt = 0;

LL D = sqrt(d)+1;

for(int i=mid;i>=l && p[mid+1].x-p[i].x<D;i--)

b[cnt++] = i;

for(int i=mid+1;i<=r && p[i].x-p[mid].x<D;i++)

b[cnt++] = i;

sort(b, b+cnt, cmp);

for(int i=0;i<cnt;i++)

for(int j=i+1;j<cnt&&p[b[j]].y-p[b[i]].y<D;++j)

d = min(d, dist(p[b[i]], p[b[j]]));

return d;

}

LL dist(node a, node b)

{

return 1LL\*(a.x-b.x)\*(a.x-b.x)+1LL\*(a.y-b.y)\*(a.y-b.y);

}

bool cmp(int a, int b)

{

return p[a].y<p[b].y;

}

## 最小球覆盖：

！！！！模拟退火求最小球覆盖，处理的ptnum必须大于等于4，并且并不能保证正确性，

struct point3

{

double x, y, z;

}pt[maxn];

int ptnum;//点的数量

double SA()

{

//step控制循环次数，循环越多次，精度越高

double step=1000000000,ans=1e30,mt;

point3 z;

z.x=z.y=z.z=0;

int s=0;

while(step>eps)

{

for(int i=0;i<ptnum;i++)

if(dist(z,pt[s])<dist(z,pt[i])) s=i;

mt=dist(z,pt[s]);

ans=min(ans,mt);

z.x+=(pt[s].x-z.x)/mt\*step;

z.y+=(pt[s].y-z.y)/mt\*step;

z.z+=(pt[s].z-z.z)/mt\*step;

step\*=0.98;

}

return ans;

}

double dist(point3 p1,point3 p2)

{

return sqrt((p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y)

+(p1.z-p2.z)\*(p1.z-p2.z));

}

## Pick公式：

顶点坐标均是整点的简单多边形：面积=内部格点数目+边上格点数目/2-1

# 字符串

## KMP

//最小循环节长度为len-nex[len]，若len%(len-nex[len]) == 0 ,则字符串完全由循环节组成

void getnex(char str[], int len, int nex[])

{

int i, j;

j = nex[0] = -1;

i = 0;

while(i<len)

{

if(j==-1 || str[i] == str[j])

nex[++i] = ++j;

else j = nex[j];

}

}

## AC自动机

struct node

{

int nex[4], fail, num;

};

node p[1010];

char str[1010];

int l, root, dp[1020][1020];

int creat();

void insert(char str[]);

void getfail();

int isn(char ch);

int main()

{

int n, ans, i, j, k, z = 1;

while(scanf("%d", &n), n)

{

l = 0;

root = creat();

for(i=0;i<n;i++)

{

char str1[50];

scanf(" %s", str1);

insert(str1);

}

getfail();

scanf(" %s", str);

int len = strlen(str);

//初始化dp数组，

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

for(j=0;j<l;j++)

dp[i][j] = INF;

dp[0][0] = 0;

//i表示处于第i个位置，处于第j个节点

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

for(j=0;j<l;j++)

{ //dp[i][j]<INF表示可以转移到此位置

if(dp[i][j] < INF)

{

//printf("\ni:%d\n", i);

//往A、C、G、T转移

for(k=0;k<4;k++)

{

int nows = p[j].nex[k];

//如果为危险节点，跳过

if(p[nows].num)continue;

int x = isn(str[i]), add;

//printf("x:%d %d ", x, k);

if(x != k)add=1;

else add=0;

//printf("add:%d\n", add);

dp[i+1][nows]=min(dp[i+1][nows], add+dp[i][j]);

}

}

}

ans = INF;

for(j=0;j<l;j++)

ans = min(ans, dp[len][j]);

if(ans == INF)

printf("Case %d: -1\n", z);

else

printf("Case %d: %d\n", z, ans);

z++;

}

return 0;

}

int isn(char ch)

{

if(ch == 'A')

return 0;

else if(ch == 'C')

return 1;

else if(ch == 'G')

return 2;

else if(ch == 'T')

return 3;

}

int creat()

{

for(int i=0;i<4;i++)

p[l].nex[i] = -1;

p[l].fail = -1;

p[l].num = 0;

l++;

return l-1;

}

void insert(char str[])

{

int s = 0;

for(int i=0;str[i];i++)

{

int x = isn(str[i]);

if(p[s].nex[x] == -1)

p[s].nex[x] = creat();

s = p[s].nex[x];

}

p[s].num=1;

}

void getfail()

{

queue<int> que;

que.push(0);

p[0].fail = 0;

while(!que.empty())

{

int s = que.front();

que.pop();

for(int i=0;i<4;i++)

{

if(p[s].nex[i] != -1)

{

if(s == 0)

p[p[s].nex[i]].fail = 0;

else

{

int q = p[s].fail;

while(q != -1)

{

if(p[q].nex[i] != -1)

{

p[p[s].nex[i]].fail = p[q].nex[i];

break;

}

q = p[q].fail;

}

if(q == -1)

p[p[s].nex[i]].fail = 0;

if(q != -1 && p[p[q].nex[i]].num == 1)

p[p[s].nex[i]].num = 1;

}

que.push(p[s].nex[i]);

}

else

p[s].nex[i] = s==0?0:p[p[s].fail].nex[i];

}

}

}

## AC自动机(简易版)：

struct AC{

int cnt, nex[maxn][26], len[maxn], fail[maxn], sig[maxn], st[maxn];

int v[maxn];

void insert(int cost, char str[])

{

int pos = 0;

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

int x = str[i]-'a';

if(!nex[pos][x]){

nex[pos][x] = ++cnt;v[cnt] = INF;

}

pos = nex[pos][x];

len[pos] = i+1;

}

v[pos] = min(v[pos], cost);

}

void getfail()

{

int l = 0, r = 0;

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

if(nex[0][i])st[++r] = nex[0][i];

while(l<r){

int p = st[++l];

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

if(nex[p][i])fail[nex[p][i]] = nex[fail[p]][i], st[++r]=nex[p][i];

else nex[p][i] = nex[fail[p]][i];

}

}

}

}AC;

## Exkmp：

求串s的每个后缀与模式串t的最长公共前缀

nex[i]:t[i…tlen-1]与t的最长公共前缀的长度

exnex[i]:s[i…tlen-1]与t的最长公共前缀的长度

char s[maxn], t[maxn];

int tlen, slen, nex[maxn], exnex[maxn];

void getnex()

{

int i, j, po;

nex[0] = tlen;

i=0;

while(i+1<tlen && t[i] == t[i+1])i++;

nex[1] = i;

po = 1;

for(i=2;i<tlen;i++)

{

if(nex[i-po]+i < nex[po] + po)

nex[i] = nex[i-po];

else

{

j = nex[po]+po-i;

if(j<0)j = 0;

while(i+j<tlen && t[i+j] == t[j])j++;

nex[i] = j;

po = i;

}

}

}

//计算exnext数组

void getextend()

{

int i=0, j, po;

slen = strlen(s);

while(s[i] == t[i] && i<slen && i<tlen)i++;

exnex[0] = i;

po = 0;

for(i=1;i<slen;i++)

{

//exnex[po]+po代表匹配到过的最远位置，

//nex[i-po]代表在t中匹配到的位置,

if(nex[i-po]+i < exnex[po]+po)

exnex[i] = nex[i-po];

else

{

j = exnex[po]+po-i;

if(j<0)j=0;

while(i+j<slen &&j<tlen && s[i+j] == t[j])

j++;

exnex[i] = j;

po = i;

}

}

}

## Manachar：

const int maxn = 200030;

char str1[maxn], str2[2\*maxn];

int p[2\*maxn];

int main()

{

int mx, maxx, i, j, res, id, b;

char ch;

while(~scanf(" %c %s", &ch, str1))

{

int len = insert();

mx = maxx = 0;

for(i=1;i<len;i++)

{

p[i] = mx>i?min(mx-i, p[2\*id-i]):1;

while(str2[i-p[i]] == str2[i+p[i]])

p[i]++;

if(i+p[i]>mx)

{

id = i;

mx = i+p[i];

}

if(p[i]-1>2 && p[i]-1>maxx)

{

maxx = p[i] -1;

if(str2[i] == '#')

b = i/2 - maxx/2;

else

b = i/2-1-maxx/2;

}

}

if(!maxx)

printf("No solution!\n");

else

{

printf("%d %d\n", b, b+maxx-1);

for(i=b;i<b+maxx;i++)

printf("%c", (str1[i]-ch+26)%26+'a');

printf("\n");

}

}

return 0;

}

int insert()

{

int i, j=2;

str2[0] = '\*';

str2[1] = '#';

for(i=0;str1[i];i++)

{

str2[j] = str1[i];

str2[j+1] = '#';

j+=2;

}

str2[j] = 0;

return j;

}

## 后缀数组：

复杂度为O(O log n)，如果串长度较大，可以考虑使用快排，实践复杂度理论为O(O\*log n\*log n)

//N是串的长度，M是桶排序桶的个数

//tmp:桶排序辅助数组,

//tp :第二关键字排名为i的下标,桶排序第二关键字，多样例需要初始化

//sa ：排名为i的起始下标

//rank: 以i为后缀的排名

//heg ：sa[i]与sa[i-1]的最长公共前缀

int N, M, rank[maxn], tmp[maxn], tp[maxn], sa[maxn], heg[maxn];

char s1[maxn], s2[maxn];

void QSORT();

void binary\_sa();

void GetHeight();

int main()

{

while(~scanf("%s %s", s1+1, s2+1))

{

int ans = 0;

int len1 = strlen(s1+1), len2 = strlen(s2+1);

strcat(s1+1, "\*");

strcat(s1+1, s2+1);

binary\_sa();

GetHeight();

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

{

if(sa[i]<= len1 && sa[i-1]>len1)

ans = max(ans, heg[i]);

else if(sa[i-1]<=len1 && sa[i]>len1)

ans = max(ans, heg[i]);

}

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

}

return 0;

}

//求sa

void binary\_sa()

{

N = strlen(s1+1);

M = 200;

for(int i=1;i<=N;i++)rank[i] = s1[i], tp[i] = i;

QSORT();

for(int w=1,q=0;q<N;w<<=1, M=q)

{

q = 0;

//利用w的结果来求2\*w

//串的后w个字符没有字符，后接0，所以排名最小

for(int i=1;i<=w;i++)tp[++q] = N-w+i;

//从小到大访问前w个字母最小的，

for(int i=1;i<=N;i++)if(sa[i]>w)tp[++q] = sa[i] - w;

QSORT();

//将原来的rank复制到tp数组里，以用来更新rank

//这里实际上可以换成下面那句的

//strcpy(tp, rank, sizeof(tp));

swap(tp, rank);

rank[sa[1]] = q = 1;

for(int i=2;i<=N;i++)

rank[sa[i]] = (tp[sa[i]] == tp[sa[i-1]] && tp[sa[i]+w] == tp[sa[i-1]+w])?q:++q;

}

}

//桶排序

void QSORT()

{

for(int i=0;i<=M;i++)tmp[i] = 0;

for(int i=1;i<=N;i++)tmp[rank[i]]++;

for(int i=1;i<=M;i++)tmp[i] += tmp[i-1];

for(int i=N;i>=1;i--)sa[tmp[rank[tp[i]]]--] = tp[i];

}

//求排名相邻的两个串的最长公共前缀

//ps:这个我还没弄懂是怎么回事

void GetHeight() {

int j, k = 0;

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

if(k) k--;

int j = sa[rank[i] - 1];

while(s1[i + k] == s1[j + k]) k++;

heg[rank[i]] = k;

}

}

## 后缀自动机

struct node{

    int len, link, nex[26];

}st[maxn\*2];

int tot, last;

//初始化

void init()

{

    st[0].len = 0;

    st[0].link = -1;

    memset(st[0].nex, -1, sizeof(st[0].nex));

    tot = 1;

}

void sam(int ch)

{

/\*

定义广义SAM，每次模式串插入之前，置last为0

if(st[last].nex.count(ch) && st[st[last].nex[ch]].len == st[last].len+1){

last = st[last].nex[ch];

return ;

}

\*/

    int p, q, cur = tot++;

    st[cur].len = st[last].len+1;

    memset(st[cur].nex, -1, sizeof(st[cur].nex));

    for(p=last;p!=-1 && st[p].nex[ch] == -1;p=st[p].link)

        st[p].nex[ch] = cur;

    if(p == -1)

        st[cur].link = 0;

    else{

        q = st[p].nex[ch];

        if(st[p].len+1 == st[q].len)

            st[cur].link = q;

        else{

            int clo = tot++;

            st[clo] = st[q];

            st[clo].len = st[p].len+1;

            for(;p!=-1 && st[p].nex[ch] == q;p=st[p].link)

                st[p].nex[ch] = clo;

            st[cur].link = st[q].link = clo;

        }

    }

    last = cur;

}

//返回不同子串个数。

LL dfs()

{

    LL ans = 0;

    for(int i=1;i<tot;i++)ans += st[i].len - st[st[i].link].len;

    return ans;

}

//桶排，O(n)，将节点按len大小排序，结果放在rk数组中

Int tmp[maxn\*2], rk[maxn\*2];

void QSORT(int tot)

{

for(int i=0;i<=tot;i++)tmp[i] = 0;

for(int i=0;i<tot;i++)tmp[st[i].len]++;

for(int i=1;i<=tot;i++)tmp[i] += tmp[i-1];

for(int i=0;i<tot;i++)

rk[tot-(tmp[st[i].len]--)] = i;

}

## 回文树：

struct node{

    //fail：失配指针，hal：长度小于一半的失配指针

    //len:长度，num：不同回文子串数量，cnt：回文子串出现次数

    int fail, len, hal, num, cnt, nex[26];

}st[maxn];

int cnt, last, len;

char s[maxn];

void init()

{

    cnt = -1;

    creat(0);

    creat(-1);

    st[0].fail = 1;

    len = last = 0;

    s[0] = '\*';

}

int creat(int x)

{

    cnt++;

    memset(st[cnt].nex, 0, sizeof(st[cnt].nex));

    st[cnt].len = x;

    st[cnt].num = st[cnt].cnt = 0;

    return cnt;

}

int getfail(int x)

{

    while(s[len-st[x].len-1] != s[len])x = st[x].fail;

    return x;

}

void Insert(char ch)

{

    int x = ch-'a';

    s[++len] = ch;

    int cur = getfail(last);

    if(st[cur].nex[x] == 0){

        int now = creat(st[cur].len+2);

        st[now].fail = st[getfail(st[cur].fail)].nex[x];

        st[cur].nex[x] = now;

        if(st[now].len<= 2)st[now].hal = st[now].fail;

        else{

            int tmp = st[cur].hal;

            while(s[len-st[tmp].len-1] != s[len] || (st[tmp].len+2)\*2 > st[now].len)

                tmp = st[tmp].fail;

            st[now].hal = st[tmp].nex[x];

        }

        if(st[st[now].hal].len\*2 == st[now].len && st[now].len%4 == 0)mx = max(mx, st[now].len);

    }

    last = st[cur].nex[x];

    st[last].cnt++;

}

//求每个回文串出现次数

void getsum()

{

    for(int i=cnt;i>1;i--)

        st[st[i].fail].cnt += st[i].cnt;

}

## 最小表示法：

//最小表示法

//一个串首尾相连，从第几个开始，得到的串字典序最小

//如果暴力，应该会肯定先找字典序最小的那个字母，如果有多个

//则比较往后比较，如果相同，则一直往后比

//最大表示法思想类似

int getmin()

{

//i,j代表开始的位置，k代表长度

int i=0,j=1,k=0,t;

while(i<len && j<len && k<len)

{

//比较从i，j开始，偏移k个位置的字母大小

t = str[(i+k)%len]-str[(j+k)%len];

//printf("%d %d %d %d\n", i, j, k, t);

//如果相等，则继续向后偏移

if(!t)k++;

else

{

//t>0，则i开始的串比k开始的串大，则i加上k+1

//因为K个字段内已经比较过了，如果较小早就偏移过去了

if(t>0)i += k+1;

else j += k+1;

if(i == j)j++;

k = 0;

}

}

return min(i, j);

}

## 回文自动机

#include <bits/stdc++.h>

using namespace std;

const int maxn = 300050;

typedef pair<int, int> P;

typedef long long ll;

const int inf = 0x3f3f3f3f;

char s[maxn];

int tot,last,len[maxn],fail[maxn],ch[maxn][30],cnt[maxn];

set<int> ss[maxn];

int newnode(int x)

{

len[++tot]=x;

return tot;

}

int getfail(int x,int i)

{

while(s[i-len[x]-1]!=s[i])

x=fail[x];

return x;

}

void pam()

{

scanf("%s",s+1);

s[0]=-1,fail[0]=1,last=0;

len[0]=0,len[1]=-1,tot=1;

for(int i=1; s[i]; i++)

{

s[i]-='a';

int p=getfail(last,i);

if(!ch[p][s[i]])

{

int q=newnode(len[p]+2);

fail[q]=ch[getfail(fail[p],i)][s[i]];

ss[q]=ss[p];

ss[q].insert(s[i]);

ch[p][s[i]]=q;

}

cnt[last=ch[p][s[i]]]++;

}

for(int i=tot; i>0; i--)

cnt[fail[i]]+=cnt[i];

}

int main()

{

pam();

ll ans=0;

for(int i=tot;i>0;i--)

{

ans+=1ll\*cnt[i]\*ss[i].size();

}

printf("%lld\n",ans);

return 0;

}

## 后缀自动机：

struct state

{

int len, link, fr;

bool is\_clone;

vector<int> inlink;

map<char, int> next;

} st[maxn];

int sz, last;

void sam\_init()

{

st[0].len = 0;

st[0].link = -1;

st[0].next.clear();

st[0].inlink.clear();

sz = 1;

last = 0;

}

void sam\_extend(char c)

{

int cur = sz++;

st[cur].len = st[last].len + 1;

st[cur].fr = st[cur].len - 1;

st[cur].is\_clone = false;

st[cur].next.clear();

st[cur].inlink.clear();

int p = last;

while (p != -1 && !st[p].next.count(c))

{

st[p].next[c] = cur;

p = st[p].link;

}

if (p == -1)

st[cur].link = 0;

else

{

int q = st[p].next[c];

if (st[p].len + 1 == st[q].len)

st[cur].link = q;

else

{

int clone = sz++;

st[clone].next.clear();

st[clone].inlink.clear();

st[clone].len = st[p].len + 1;

st[clone].link = st[q].link;

st[clone].next = st[q].next;

while (p != -1 && st[p].next[c] == q)

{

st[p].next[c] = clone;

p = st[p].link;

}

st[q].link = st[cur].link = clone;

st[clone].fr = st[q].fr;

st[clone].is\_clone = true;

}

}

last = cur;

}

# 数据结构

## 主席树：

struct node

{

int l, r, sum;

}p[maxn\*20];

int num, root[maxn], a[maxn], b[maxn];

int main()

{

int n, i, j, m;

num = 0;

scanf("%d %d", &n, &m);

root[0] = creat(1, n);

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

{

scanf("%d", &a[i]);

b[i] = a[i];

}

sort(b+1, b+n+1);

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

{

int x = lower\_bound(b+1,b+n+1, a[i])-b;

root[i] = update(root[i-1], 1, n, x);

}

while(m--)

{

int l, r, x;

scanf("%d %d %d", &l, &r, &x);

printf("%d\n", b[query(root[l-1], root[r], 1, n, x)]);

}

return 0;

}

//建立原始的线段树(空树)

int creat(int l, int r)

{

int ts = ++num;

p[ts].sum = 0;

if(l == r)return ts;

int mid = (l+r)>>1;

p[ts].l = creat(l, mid);

p[ts].r = creat(mid+1, r);

return ts;

}

//建立当把x插入树时，单独更新相关的链

int update(int la, int l, int r, int x)

{

int now = ++num;

p[now] = p[la];

if(l == r){

p[now].sum++;return now;

}

int mid=(l+r)>>1;

if(x<=mid)

p[now].l = update(p[la].l, l, mid, x);

else

p[now].r = update(p[la].r, mid+1, r, x);

p[now].sum++;

return now;

}

//查询，因为区间满足相减性，比如树所保存的同样区间[l,r]，

//在第qr状态时减去ql状态时的数，就是[ql+1,qr]区间的状态

int query(int ql, int qr, int l, int r, int x)

{

if(l == r)return l;

int mid = (l+r)>>1, sum = p[p[qr].l].sum-p[p[ql].l].sum;

if(x<=sum)

return query(p[ql].l, p[qr].l, l, mid, x);

else

return query(p[ql].r, p[qr].r, mid+1, r, x-sum);

}

## ST表：

求区间[l,r]内的最大(小)值

//st表查询

int k = log(1.0\* r- l+1)/log(2.0);

int ans1 = max(sx[ l ][ k ], sx[ r - (1<<k)+1 ][ k ]);

int ans2 = min(si[ l ][ k ], si[ r - (1<<k)+1 ][ k ]);

;//预处理ST表，[i][j]代表从i开始，长度为2^j次幂的区间内最大(小)值是多少

void st(int n)

{

int i, j;

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

si[i][0] = sx[i][0] = a[i];

for(j=1;j<19;j++)

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

if(i+(1<<j)-1<=n)

{

si[i][j] = min(si[i][j-1], si[i+(1<<(j-1))][j-1]);

sx[i][j] = max(sx[i][j-1], sx[i+(1<<(j-1))][j-1]);

}

else break;

}

## 左偏树：

struct node{

int cl, cr, val, dis, fa;

node(){

cl = cr = dis = fa = 0;

}

}S[maxn];

int a[maxn], b[maxn];

int getf(int x);

int merge(int x, int y);

void remove(int x);

int main()

{

int n, m, i, j;

S[0].dis = -1;

scanf("%d %d", &n, &m);

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

scanf("%d", &S[i].val);

S[i].fa = i;

}

while(m--){

int s, x, y;

scanf("%d", &s);

if(s == 1){

scanf("%d %d", &x, &y);

if(S[x].val == -1 || S[y].val == -1)continue;

int nx = getf(x), ny = getf(y);

if(nx!=ny)S[nx].fa = S[ny].fa = merge(nx, ny);

}

else{

scanf("%d", &x);

if(S[x].val == -1)printf("-1\n");

else{

y = getf(x);

printf("%d\n",S[y].val);

remove(y);

}

}

}

return 0;

}

//合并x、y两颗左偏树

int merge(int x, int y)

{

if(!x || !y)return x+y;

if(S[x].val > S[y].val || (S[x].val == S[y].val && x > y))

swap(x, y);

S[x].cr = merge(S[x].cr, y);

S[S[x].cr].fa = x;

if(S[S[x].cl].dis < S[S[x].cr].dis)

swap(S[x].cl, S[x].cr);

S[x].dis = S[S[x].cr].dis+1;

return x;

}

//删除堆顶

void remove(int x)

{

S[x].val = -1;

S[S[x].cl].fa = S[x].cl;

S[S[x].cr].fa = S[x].cr;

S[x].fa = merge(S[x].cl, S[x].cr);

}

//获取堆顶

int getf(int x)

{

return S[x].fa = x == S[x].fa?x:getf(S[x].fa);

}

## 树套树(树状数组套线段树)：

//tot:当前使用结点编号，tl：左子树结点编号，tr：右子树结点编号，p：存储的数值

int tot, tl[100\*maxn], tr[100\*maxn], p[100\*maxn];

//n：数组大小，le：离散化后的所有可能取值的数量，dic：离散数组

int n, le, dic[2\*maxn], a[maxn], op[maxn], ql[maxn], qr[maxn], qk[maxn];

//lsz，rsz：以l，r为前缀和的树状数组需要访问的元素数量，vl，vr：存储需要访问的元素

int lsz, rsz, vl[maxn], vr[maxn];

//初始化数组

void init()

{

le = 0;

memset(tl, 0, sizeof(tl));

memset(tr, 0, sizeof(tr));

memset(p, 0, sizeof(p));

}

//构造初始的树(链)

void built(int n)

{

tot = 1;

while(tot<n)

tot<<=1;

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

{

int pos = lower\_bound(dic+1, dic+1+le, a[i])-dic;

Add(i, pos, 1);

}

}

//更新k有关的外层树状数组

void Add(int k, int pos, int x)

{

for(int i=k;i<=n;i+=lowbit(i))

Insert(1, le, i, pos, x);

}

int lowbit(int x)

{

return x&(-x);

}

//更新k这条链内层线段树

void Insert(int l, int r, int k, int pos, int x)

{

if(l == r){

p[k] += x;

return;

}

int mid = (l+r)/2;

if(pos<=mid){

if(!tl[k])tl[k]=++tot;

Insert(l, mid, tl[k], pos, x);

}

else{

if(!tr[k])tr[k]=++tot;

Insert(mid+1, r, tr[k], pos, x);

}

p[k] = p[tl[k]] + p[tr[k]];

}

//将原本第x个数修改为pk

void Update(int x, int pk)

{

int pos = lower\_bound(dic+1, dic+le+1, a[x])-dic;

Add(x, pos, -1);

pos = lower\_bound(dic+1, dic+le+1, pk)-dic;

Add(x, pos, 1);

a[x] = pk;

}

//查询值qk在区间内[ql,qr]的排名(x为1不包含自身，x为0包含自身)

//查询区间比qk小的数量，可以将分别求出l-1，r总共有多少个，再减一下

int queryRank(int ql, int qr, int qk, int x)

{

int pos = lower\_bound(dic+1, dic+le+1, qk)-dic;

int sum = 0;

if(pos-x<1)return 0;

for(int i=qr;i;i-=lowbit(i))

sum += querymin(1, le, i, pos-x);

for(int i=ql-1;i;i-=lowbit(i))

sum -= querymin(1, le, i, pos-x);

return sum;

}

//查询小于等于x的数的数量

int querymin(int l, int r, int k, int x)

{

if(l == r && l == x)

return p[k];

int mid = (l+r)/2;

if(x <= mid)

return querymin(l, mid, tl[k], x);

else

return p[tl[k]]+querymin(mid+1, r, tr[k], x);

}

//返回区间排名为k的数

int queryVal(int l, int r, int k)

{

lsz = rsz = 0;

for(int i=l-1;i;i-=lowbit(i))

vl[lsz++] = i;

for(int i=r;i;i-=lowbit(i))

vr[rsz++] = i;

int ks = queryK(1, le, k);

return dic[ks];

}

//查询内层，求第k小

//对于每一层先求出当前层[l,r]内的左边有多少个

int queryK(int l, int r, int x)

{

if(l == r)return l;

int i, sum = 0;

for(i=0;i<rsz;i++)

sum += p[tl[vr[i]]];

for(i=0;i<lsz;i++)

sum -= p[tl[vl[i]]];

int mid = (l+r)/2;

if(x <= sum){

for(i=0;i<rsz;i++)

vr[i] = tl[vr[i]];

for(i=0;i<lsz;i++)

vl[i] = tl[vl[i]];

return queryK(l, mid, x);

}

else {

for(i=0;i<rsz;i++)

vr[i] = tr[vr[i]];

for(i=0;i<lsz;i++)

vl[i] = tr[vl[i]];

return queryK(mid+1, r, x-sum);

}

}

//返回[ql,qr]中小于qk的最大的数

int KsBefore(int ql, int qr, int qk)

{

int num = queryRank(ql, qr, qk, 1);

if(num == 0)return -1;

return queryVal(ql, qr, num);

}

//返回[ql,qr]中大于qk的最小的数

int KsAfter(int ql, int qr, int qk)

{

int num = queryRank(ql, qr, qk, 0);

if(num == qr-ql+1)return -1;

return queryVal(ql, qr, num+1);

}

## 伸展树(SPLAY):

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

伸展树较快的完成区间翻转的操作，可以借助lazy标记

伸展树的旋转并不影响中序遍历，所以可以进行区间操作，对于区间[L，R]，可以将L-1旋转至根节点，然后将R+1旋转至根节点的右节点，然后R+1的那个结点的左子树即[L,R]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Fa:父结点编号，childs：子结点编号，val：值，sz：当前子树大小

struct node{

int fa, val, sz, lazy, childs[2];

}sp[maxn];

int tot;

//新建一个值为val，父亲为fa的结点

int newpoint(int val, int fa)

{

sp[++tot].fa = fa;

sp[tot].val = val;

sp[tot].lazy = sp[tot].childs[0] = sp[tot].childs[1] = 0;

sp[tot].sz = 1;

return tot;

}

//判断x是其父亲的左右孩子

bool findd(int x)

{

return sp[sp[x].fa].childs[0]==x?0:1;

}

//连接边，将x作为fa的son儿子

void connect(int x, int fa, int son)

{

sp[x].fa = fa;

sp[fa].childs[son] = x;

}

//将lazy标记向下更新

void pushdown(int x)

{

if(sp[x].lazy){

sp[sp[x].childs[0]].lazy ^= 1;

sp[sp[x].childs[1]].lazy ^= 1;

swap(sp[x].childs[0], sp[x].childs[1]);

sp[x].lazy = 0;

}

}

//更新子树大小

void update(int x)

{

sp[x].sz = sp[sp[x].childs[0]].sz + sp[sp[x].childs[1]].sz + 1;

}

//旋转x到其父结点，

void rotate(int x)

{

int Y = sp[x].fa;

int R = sp[Y].fa;

//lazy标记更新，如果没有可以删除这两句

pushdown(Y), pushdown(x);

int Yson = findd(x), Rson = findd(Y);

int B = sp[x].childs[Yson^1];

connect(B, Y, Yson);

connect(Y, x, Yson^1);

connect(x, R, Rson);

update(Y), update(x);

}

//旋转x到现在to的位置

void splay(int x, int to)

{

to = sp[to].fa;

while(sp[x].fa != to)

{

int y =sp[x].fa;

if(sp[y].fa == to)

rotate(x);

else if(findd(x) == findd(y))

rotate(y), rotate(x);

else

rotate(x), rotate(x);

}

}

//插入值为x的结点

void Insert(int x)

{

int now = sp[0].childs[1];

if(now == 0){

sp[0].childs[1] = newpoint(x, 0);

}

else{

while(1)

{

int nex = x<p[now].id?0:1;

if(!p[now].childs[nex])

{

p[now].childs[nex] = newpoint(x, now);

splay(p[now].childs[nex], p[0].childs[1]);

return;

}

now = p[now].childs[nex];

}

}

}

//删除编号为x的树,如果需要删除值为x的结点，需要先查找其编号

void deleted(int x)

{

//需要看情况将x移动至根节点

int pos = x;

splay(x, sp[0].childs[1]);

if(!sp[pos].childs[0] && !sp[pos].childs[1])

sp[0].childs[1] = 0;

else if(!sp[pos].childs[0])

connect(sp[pos].childs[1], 0, 1);

else{

int now = sp[pos].childs[0];

pushdown(now);

while(sp[now].childs[1]){

now = sp[now].childs[1];

pushdown(now);

}

splay(now, sp[0].childs[1]);

connect(sp[pos].childs[1], now, 1);

connect(now, 0, 1);

update(now);

}

}

//查找序列中第x个元素编号

int findSize(int x)

{

int now = sp[0].childs[1];

pushdown(now);

while(1){

int lsz = sp[sp[now].childs[0]].sz;

if(x <= lsz)

now = sp[now].childs[0];

else if(x == lsz + 1)

return now;

else

now = sp[now].childs[1], x-=lsz+1;

pushdown(now);

}

}

//翻转区间[l,r]

void Revser(int l, int r)

{

int pos1 = findSize(l);

splay(pos1, sp[0].childs[1]);

int pos2 = findSize(r+2);

splay(pos2, sp[pos1].childs[1]);

sp[sp[pos2].childs[0]].lazy ^= 1;

}

//调试，输出树

void debug()

{

printf("id val cnt sz child-left child-right\n");

queue<int> que;

que.push(sp[0].childs[1]);

while(!que.empty())

{

int u = que.front();que.pop();

if(!u)continue;

printf("%-3d %-3d %-3d %-3d %-10d %-10d\n", u, sp[u].lazy, sp[u].lazy, sp[u].sz, sp[u].childs[0], sp[u].childs[1]);

que.push(sp[u].childs[0]), que.push(sp[u].childs[1]);

}

printf("\n\n");

}

## 动态树(LCT)：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

动态树可以实现动态的树上操作，比如建边，删除边，动态树

主要借助伸展树进行操作，原树可以看作是许多个SPLAY，每个SPLAY

的中序遍历，深度小的总是在前面

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct node{

int ch[2], fa, val, lz, sum;

}lct[maxn];

int n, stk[maxn];

void pushdown(int x)

{

if(lct[x].lz)

{

swap(lct[x].ch[0], lct[x].ch[1]);

if(lct[x].ch[0])lct[lct[x].ch[0]].lz ^= lct[x].lz;

if(lct[x].ch[1])lct[lct[x].ch[1]].lz ^= lct[x].lz;

lct[x].lz = 0;

}

}

void Update(int x)

{

lct[x].sum = lct[x].val^lct[lct[x].ch[0]].sum^lct[lct[x].ch[1]].sum;

}

bool noroot(int x)

{

return lct[lct[x].fa].ch[0] == x || lct[lct[x].fa].ch[1] == x;

}

void rotate(int x)

{

int y = lct[x].fa, z = lct[y].fa;

int k=lct[y].ch[1]==x, w=lct[x].ch[!k];

if(noroot(y))lct[z].ch[lct[z].ch[1]==y] = x;

lct[y].ch[k] = w, lct[x].ch[!k] = y;

if(w)lct[w].fa = y;

lct[x].fa = z, lct[y].fa = x;

Update(y),Update(x);

}

void splay(int x)

{

int top = 0, now = x;

stk[++top] = now;

while(noroot(now))stk[++top] = now = lct[now].fa;

while(top)pushdown(stk[top--]);

while(noroot(x))

{

int y = lct[x].fa, z= lct[y].fa;

if(noroot(y))

rotate((lct[y].ch[0]==x) ^ (lct[z].ch[0]==y)?x:y);

rotate(x);

}

Update(x);

}

//连通x到根节点的路径

void access(int x)

{

for(int y=0;x;y=x,x=lct[x].fa)

splay(x), lct[x].ch[1] = y, Update(x);

}

//将x变为原树的根

void makeroot(int x)

{

access(x);splay(x);

lct[x].lz ^= 1;

pushdown(x);

}

//找到x所在的原树的根

int findroot(int x)

{

access(x), splay(x);

pushdown(x);

while(lct[x].ch[0]){

x = lct[x].ch[0];

pushdown(x);

}

splay(x);

return x;

}

//加入边x-y

void link(int x, int y)

{

makeroot(y);

if(findroot(x) != y)

lct[y].fa = x;

}

//删除边x-y

void cut(int x, int y)

{

makeroot(x);

if(findroot(y)==x && lct[y].fa ==x && !lct[y].ch[0]){

lct[y].fa = lct[x].ch[1] = 0;

Update(y), Update(x);

}

}

//打通x-y的路径

void split(int x, int y)

{

makeroot(x);

access(y);splay(y);

}

## 树上哈希：

//此题与判断二叉树同构不同！！！

//有一颗多叉树，如果能将所有点重新编号，然后

//两颗树完全相同即认为两棵树同构

//此题无根树，将每个点作为根，遍历树，然后求以

//该节点作为根的Hash值，然后将所有的点的值按hash

//出的值排序，如果两个树得到的所有值都相等，即同构

//此题其实还可以通过寻找树的重心的方法对每棵树

//只进行最多两次hash。

//另外图的同构也可以通过hash来判定

const int maxn = 64;

vector<int> g[maxn];

int a[maxn][maxn], vis[maxn];

int Hash(int u, int f);

int main()

{

int t, n, i, j, k;

scanf("%d", &t);

for(i=1;i<=t;i++){

scanf("%d", &n);

//构造无根树

for(j=1;j<=n;j++)g[j].clear();

for(j=1;j<=n;j++){

scanf("%d", &k);

if(k == 0)continue;

g[k].push\_back(j);

g[j].push\_back(k);

}

//求以每个结点为根的树的hash值

for(j=1;j<=n;j++)

a[i][j] = Hash(j, -1);

sort(a[i]+1, a[i]+n+1);

for(j=1;j<=i;j++){

for(k=1;k<=n;k++)

if(a[i][k] != a[j][k])

break;

if(k>n){

printf("%d\n", j);break;

}

}

}

return 0;

}

//对当前子树计算hash值

int Hash(int u, int f){

int p[maxn], top=0, ans=7;

for(int i=0;i<g[u].size();i++)

if(g[u][i] != f){

p[top] = Hash(g[u][i], u);

top++;

}

sort(p, p+top);

//此处仍以确定需要乘的值

for(int i=0;i<top;i++)

ans = ans \* 11 + p[i];

return ans \* 11+1;

}

## 树链剖分：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Notice：

1.需要注意题目要求是对点还是对边的值进行维护！！！

2.如果是对点进行建边，则最好建立2~n的线段树，不要统计那个无边的根节点

3.注意区分结点原本的编号与其dfs后的编号

fa：存放结点的父结点 son：存放结点的重儿子

de: 结点的深度 size：以结点为根的子树大小

id：重新编号后其原结点得应的编号

rk：重新编号后其编号对应的原结点

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct node{

LL sum, lazy;

}p[maxn\*4];

int cnt, fa[maxn], son[maxn], de[maxn], size[maxn], top[maxn];

int id[maxn], rk[maxn];

int n, a[maxn], b[maxn];

LL ans, modd;

vector<int> g[maxn];

void dfs1(int f, int u, int d);

void dfs2(int t, int u);

void bulit(int l, int r, int k);

void upd(int l, int r, int al, int ar, int x, int k);

void down(int k, int l);

void query(int l, int r, int al, int ar, int k);

void quiry(int l, int r);

void road\_upd(int l, int r, int x);

int main()

{

int t, i, j, k, m, l, r, root;

scanf("%d %d %d %lld", &n, &m, &root, &modd);

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

scanf("%d", &a[i]);

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

scanf("%d %d", &j, &k);

g[j].push\_back(k);

g[k].push\_back(j);

}

dfs1(-1, root, 1);

dfs2(root, root);

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

b[id[i]] = a[i];

bulit(1, n, 1);

while(m--){

int x;

scanf("%d", &t);

switch(t){

case 1:

scanf("%d %d %d", &l, &r, &x);

road\_upd(l, r, x);

break;

case 2:

scanf("%d %d", &l, &r);

quiry(l, r);

printf("%lld\n", ans);

break;

case 3:

//l所有子树加上r，即id[l]+size[l]-1

scanf("%d %d", &l, &r);

upd(1, n, id[l], id[l]+size[l]-1, r, 1);

break;

case 4:

scanf("%d", &x);

ans = 0;

query(1, n, id[x], id[x]+size[x]-1, 1);

printf("%lld\n", ans);

break;

}

}

return 0;

}

void dfs1(int f, int u, int d){

//记录各点子树结点数量，最多的为重儿子

int mx = -1;

fa[u] = f;

size[u] = 1;

de[u] = d;

for(int i=0;i<g[u].size();i++){

if(g[u][i] != f){

dfs1(u, g[u][i], d+1);

size[u] += size[g[u][i]];

if(mx == -1 || size[g[u][i]]>size[mx])

mx = g[u][i];

}

}

son[u] = mx;

}

//t:u所在重链根节点，u当前结点

void dfs2(int t, int u){

top[u] = t;

id[u] = ++cnt;

rk[cnt] = u;

//先遍历重儿子，以保证重链编号连续

if(son[u]==-1)return;

dfs2(t, son[u]);

for(int i=0;i<g[u].size();i++){

if(g[u][i] != fa[u] && g[u][i] != son[u])

//轻链的根节点为自身

dfs2(g[u][i], g[u][i]);

}

}

void bulit(int l, int r, int k)

{

p[k].lazy = 0;

if(l == r){

p[k].sum = b[l] % modd;

return;

}

int mid = (l+r)/2;

bulit(l, mid, 2\*k);

bulit(mid+1, r, 2\*k+1);

p[k].sum = (p[2\*k].sum + p[2\*k+1].sum) % modd;

}

void upd(int l, int r, int al, int ar, int x, int k){

//将当前的lazy标记计算完成

down(k, r-l+1);

if(l == al && r == ar){

p[k].lazy = x;

down(k, r-l+1);

return;

}

//[al,ar]区间都加上x，则当前区间总和加上(ar-al+1)\*x;

p[k].sum = (p[k].sum + (ar-al+1)\*x%modd) % modd;

int mid = (l+r)/2;

if(ar <= mid)

upd(l, mid, al, ar, x, 2\*k);

else if(al>mid)

upd(mid+1, r, al, ar, x, 2\*k+1);

else{

upd(l, mid, al, mid, x, 2\*k);

upd(mid+1, r, mid+1, ar, x, 2\*k+1);

}

}

void query(int l, int r, int al, int ar, int k){

down(k, r-l+1);

if(l == al && r == ar){

ans = (ans + p[k].sum)%modd;

return;

}

int mid = (l+r)/2;

if(ar <= mid)

query(l, mid, al, ar, 2\*k);

else if(al>mid)

query(mid+1, r, al, ar, 2\*k+1);

else{

query(l, mid, al, mid, 2\*k);

query(mid+1, r, mid+1, ar, 2\*k+1);

}

}

//将当前区间的lazy标记加到sum中，并将lazy标记下移

//以保证当前区间sum值正确

void down(int k, int l){

p[k].sum = (p[k].sum + p[k].lazy \* l % modd)%modd;

if(l != 1){

p[2\*k].lazy = (p[2\*k].lazy+p[k].lazy) % modd;

p[2\*k+1].lazy = (p[2\*k+1].lazy+p[k].lazy) % modd;

}

p[k].lazy = 0;

}

//将l到r的路径上的所有点加上x

//寻找LCA，可通过top数组 (询问操作类似)

void road\_upd(int l, int r, int x){

//如果两点不在一条重链上，则将重链根结点深度较大

//的链先计算(如果先移动深度较小的，可能错过LCA)

while(top[l] != top[r]){

if(de[top[l]] > de[top[r]]){

//更新重链，将其更改为重链根节点的父结点

upd(1, n, id[top[l]], id[l], x, 1);

l = fa[top[l]];

}else{

upd(1, n, id[top[r]], id[r], x, 1);

r = fa[top[r]];

}

}

//在同一条重链上，更新即可

if(de[l] > de[r])

upd(1, n, id[r], id[l], x, 1);

else

upd(1, n, id[l], id[r], x, 1);

}

## 单调栈：

//一串序列，目标值为区间最小值乘区间和，求最大的目标值

//首先使用单调栈求出以i为最小值的，区间的左右端点l和r

//然后问题可以转化为求区间[l,r]内包含i的最大

//(最小(因为a[i]可能为负))子段和，可以通过st表

//求区间的最大子段和,因为包含i,所以等于在[l,i]中

//选一个位置作为最大字段和左端点，在[i,r]选一个右端点

//因为求区间[l,r]的和为sum[r]-sum[l-1]，所以在

//选取值时，左端点时间上应该选[l-1,i-1]

int li[maxn], ri[maxn], sk[maxn];

LL b[maxn], a[maxn], res, sx[21][maxn], si[21][maxn];

void inst(int n);

void instack(int n);

int main()

{

int n, i, j;

scanf("%d", &n);

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

{

scanf("%lld", &a[i]);

b[i] = b[i-1]+a[i];

}

a[n+1] = a[0] = -1e18;

res = -1e18;

instack(n);

inst(n);

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

{

//求以a[i]为最小值，能得到的最大值

//如果a[i]为正，则[l,r]中最大区间和

//即sum中[i,r]的最大值，[l-1,i-1]的最小值

li[i] = max(li[i]-1, 0);

int k = log(1.0\*i-li[i])/log(2.0);

LL mlx = max(sx[k][li[i]], sx[k][i-(1<<k)]);

LL mli = min(si[k][li[i]], si[k][i-(1<<k)]);

k = log(1.0\*ri[i]-i+1)/log(2.0);

LL mrx = max(sx[k][i], sx[k][ri[i]-(1<<k)+1]);

LL mri = min(si[k][i], si[k][ri[i]-(1<<k)+1]);

res = max(res, (mrx-mli)\*a[i]);

res = max(res, (mri-mlx)\*a[i]);

}

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

return 0;

}

//st表预处理出前缀和的最小和最大值

void inst(int n)

{

for(int i=0;i<=n;i++)

sx[0][i] = si[0][i] = b[i];

for(int j=1;j<=19;j++)

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

if(i+(1<<j)-1 <= n){

sx[j][i] = max(sx[j-1][i], sx[j-1][i+(1<<(j-1))]);

si[j][i] = min(si[j-1][i], si[j-1][i+(1<<(j-1))]);

}

else break;

}

}

//单调栈求以第i个元素作为最小值，其最远的左右区间

//原串为1~n,但前缀和可能为0，所以应该求0~n的值

void instack(int n)

{

int i, l;

l = -1;

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

//如果a[i]比栈顶元素大，则其左区间端点为

//栈顶所在的位置的后一个位置

if(l==-1 || a[i]>a[sk[l]]){

if(l == -1)

li[i] = 1;

else

li[i] = sk[l]+1;

sk[++l] = i;

}

//如果a[i]比栈顶元素小，则弹出栈顶元素

//直到a[i]大于栈顶元素或栈为空

else{

while(l>=0 && a[i]<= a[sk[l]]){

l--;

}

if(l == -1)li[i] = 1;

else li[i] = sk[l]+1;

sk[++l] = i;

}

}

l = -1;

for(i=n;i>=1;i--){

if(l==-1 || a[i]>a[sk[l]]){

if(l == -1)

ri[i] = n;

else

ri[i] = sk[l]-1;

sk[++l] = i;

}

else{

while(l>=0 && a[i]<= a[sk[l]]){

l--;

}

if(l == -1)ri[i] = n;

else ri[i] = sk[l]-1;

sk[++l] = i;

}

}

}

## 单调队列：

长度为w的序列a，每个位置i开始取n个数的最大(小)值

struct node{

int id, mx;

node(){}

node(int a, int b){

id = a, mx = b;

}

};

void upque(int a[], int w, int n, int b[])

{

int i, j;

deque<node> que;

que.push\_back(node(0, a[0]));

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

{

//如果求最小值，将<改为>

while(!que.empty() && que.back().mx < a[i])

que.pop\_back();

que.push\_back(node(i, a[i]));

}

b[0] = que.front().mx;

for(;i<w;i++)

{

while(!que.empty() && que.front().id+n-1 < i)que.pop\_front();

while(!que.empty() && que.back().mx < a[i])

que.pop\_back();

que.push\_back(node(i, a[i]));

b[i-n+1] = que.front().mx;

}

}

## 虚树：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

虚树

有一棵n个点的数，m次询问，每次询问有k个点，求最少删除多少个这k个点之外的点，使这k个点不连通。

思路：对于每次询问若有k个点中有两个点为父子关系，无解。否则，构造虚树。

首先对每个点按DFS序排序。

依次将每个点放入栈中，

设lca为栈顶元素和当前点u的LCA，

若lca==栈顶元素，将放入栈中

否则弹出栈中深度大于lca的元素，并连st[top-1]->st[top]。

若lca不在栈中，将其入栈

将u入栈。

所有点都处理后，依次将栈中元素弹出，并连边

建树后，DFS虚树，

当前节点为u，

若u为标记点，依次访问其子节点v，若子节点v为标记点，ans++。

若u不为标记点，统计v为标记点的个数num，若num>=2，ans++，若num==1，则u变为标记点。

在DFS时，对树中节点清除信息，以便下次询问。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<iostream>

#include<cstdlib>

#include<cmath>

#include<algorithm>

#include<cstring>

#include<set>

#include<vector>

#include<queue>

#include<iterator>

#define dbg(x) cout<<#x<<" = "<<x<<endl;

#define INF 0x3f3f3f3f

#define eps 1e-7

using namespace std;

typedef long long LL;

typedef pair<int, int> P;

const int maxn = 100100;

const int mod = 998244353;

int top, ans, dep[maxn], fa[maxn][30], a[maxn], vis[maxn], st[maxn];

int tot, cnt, dfn[maxn], hd1[maxn], hd2[maxn], nex[maxn\*10], to[maxn\*10];

int main()

{

tot = cnt = 1;

memset(hd1, -1, sizeof(hd1));

memset(hd2, -1, sizeof(hd2));

int t, n, i, j, k, q, m;

scanf("%d", &n);

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

scanf("%d %d", &j, &k);

add(j, k, hd1);

add(k, j, hd1);

}

init(n);

scanf("%d", &q);

while(q--)

{

scanf("%d", &m);

for(i=0;i<m;i++){

scanf("%d", &a[i]);

vis[a[i]] = 1;

}

int sig = 1;

for(i=0;i<m;i++)

if(vis[fa[a[i]][0]])

{

sig = 0;break;

}

if(!sig){

printf("-1\n");

for(i=0;i<m;i++)

vis[a[i]] = 0;

continue;

}

sort(a, a+m, cmp);

top = ans = 0;

if(a[0] != 1)st[++top] = 1;

for(i=0;i<m;i++)

Insert(a[i]);

while(top>1){

add(st[top-1], st[top], hd2);

top--;

}

solve(1);

vis[1] = cnt = 0;

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

}

return 0;

}

//将标记点按DFS序排序

bool cmp(int a, int b)

{

return dfn[a]<dfn[b];

}

//构造虚树，

void Insert(int u)

{

if(top == 0){

st[++top] = u;

return;

}

int lca = LCA(u, st[top]);

while(top>1 && dep[lca]<dep[st[top-1]])

{

add(st[top-1], st[top], hd2);

top--;

}

if(dep[lca]<dep[st[top]])add(lca, st[top--], hd2);

if((!top) || st[top]!=lca)st[++top] = lca;

st[++top] = u;

}

void solve(int u)

{

if(vis[u]){

for(int i=hd2[u];i!=-1;i=nex[i]){

solve(to[i]);

if(vis[to[i]]){

ans++;

vis[to[i]] = 0;

}

}

}else{

for(int i=hd2[u];i!=-1;i=nex[i]){

solve(to[i]);

vis[u] += vis[to[i]];

vis[to[i]] = 0;

}

if(vis[u]>=2)

{

ans++;

vis[u] = 0;

}

}

hd2[u] = -1;

}

void add(int f, int t, int hd[])

{

to[++cnt] = t;

nex[cnt] = hd[f];

hd[f] = cnt;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LCA部分代码

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## 支配树(lengauer\_trajan)：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

求有向图支配点，点到根节点链上的点，即为当前点的支配点

对于DAG图，可以利用拓扑排序，找出每个点的最近支配点一个点的最近支配点就是所有可以到达它的点的LCA，其在支配树上的父结点就是其最近支配点，可以同时建树维护LCA

对于有向有环图，首先对图进行dfs序，重新编号；然后按照编号顺序从大到小求各点(不包括dfs树的根点)的半必经点 然后通过各点的半必经点求其支配点

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//fa:dfs序上点的父结点，semi：编号最小半必经点

//idom：最近必经点 dom：支配树

int cnt, fa[maxn], semi[maxn], idom[maxn], dom[maxn];

//a,mi:带权并查集所需数组，dfn：dfs序编号 rk：编号对应原结点

int a[maxn], mi[maxn], dfn[maxn], rk[maxn];

int tot, g[maxn], rg[maxn], nex[maxn\*12], to[maxn\*12];

void init(int n);

int Find(int x);

void dfs(int p, int u);

void lengauer\_trajan(int n);

void add(int h[], int f, int t);

int main()

{

int m, s, i, j, k;

scanf("%d %d %d", &n, &m, &s);

init(n);

while(m--){

scanf("%d %d %d", &i, &j, &k);

add1(i, j, k), add1(j, i, k);

}

dijkstra(s);

memset(g, 0, sizeof(g));

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

//printf("pre:%d\n", pre[i]);

for(j=pre[i];j;j=nex[j]){

add(g, i, to[j]);

add(rg, to[j], i);

//printf("%d %d\n", to[j], i);

}

}

dfs(s, s);

lengauer\_trajan(cnt);

return 0;

}

void init(int n)

{

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

a[i] = mi[i] = semi[i] = idom[i] = i;

g[i] = rg[i] = dfn[i] = hd[i] = dom[i] = 0;

}

cnt = tot = 0;

}

void add(int h[], int f, int t)

{

to[++tot] = t;

nex[tot] = h[f];

h[f] = tot;

}

int Find(int x)

{

if(x == a[x])return x;

int fa=a[x], y = Find(a[x]);

if(dfn[semi[mi[x]]] > dfn[semi[mi[fa]]])mi[x] = mi[fa];

return a[x] = y;

}

void dfs(int p, int u)

{

fa[u] = p;

dfn[u] = ++cnt;

rk[cnt] = u;

for(int i=rg[u];i;i=nex[i]){

if(!dfn[to[i]])

dfs(u, to[i]);

}

}

//n为当前这个连通图的结点数量，即dfs后cnt的值

void lengauer\_trajan(int n)

{

for(int i=n;i>=2;i--)

{

//访问所有能到达y结点的边

int y = rk[i], tmp = n;

for(int j=g[y];j;j=nex[j])

{

int v = to[j];

//这句我也不知道为啥，反正加不加好像都一样

if(!dfn[v])continue;

//如果dfn[v]<dfn[y]则v是y的一个半必经点

if(dfn[v] < dfn[y])tmp = min(tmp, dfn[v]);

//如果dfn[v]>dfn[y],

else Find(v), tmp = min(tmp, dfn[semi[mi[v]]]);

}

semi[y] = rk[tmp];

a[y] = fa[y];

//在原图添加一条semi[y]->y的边

add(rg, semi[y], y);

//我们求出深搜树后，考虑原图中所有非树边（即不在树上的边），我们将这些边删掉，

//加入一些新的边 {(semi[w],w)|w∈V{(semi[w],w)|w∈V andand w≠r}w≠r}，我们会

//发现构建出的新图中每一个点的支配点是不变的，

//通过这样的改造我们使得原图变成了DAG

y = rk[i-1];

for(int j=rg[y];j;j=nex[j]){

int v = to[j];

Find(v);

if(semi[mi[v]] == y)idom[v] = y;

else idom[v] = mi[v];

}

}

//构造支配树

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

int id = rk[i];

if(idom[id] != semi[id])

idom[id] = idom[idom[id]];

add(dom, idom[id], id);

}

}

# 博弈论

## 巴什博弈：

只有一堆n个物品，两个人轮流从这堆物品中取物，规定每次至少取一个，最多取m个。最后取光者得胜。

n%（m+1）==0则后手胜利，否则先手胜利

## 威佐夫博弈（Wythoff's game）：

有两堆各若干个物品，两个人轮流从任一堆取至少一个或同时从两堆中取同样多的物品，规定每次至少取一个，多者不限，最后取光者得胜。

用（a[k]，b[k]）（a[k] ≤ b[k] ,k=0，1，2，...,n)表示两堆物品的数量并称其为局势，如果甲面对（0，0），那么甲已经输了，这种局势我们称为奇异局势。前几个奇异局势是：（0，0）、（1，2）、（3，5）、（4，7）、（6，10）、（8，13）、（9，15）、（11，18）、（12，20）。（注：k表示奇异局势的序号， 第一个奇异局势k=0）。

可以看出,a[0]=b[0]=0,a[k]是未在前面出现过的最小自然数,而 b[k]= a[k] + k。

两堆石子分别有a,b个，是否有必胜策略

bool weizuofu(int a, int b)

{

if( a > b)swap(a, b);

int c = (int )((double)(b - a)\* ( (1 + sqrt(5)) / 2));

if( c == a)return false;

else return true;

}

打表求出必败态：

b[i]：(i,b[i])为必败态，(x[i],y[i])为必败态，且y[i]-x[i]=i;

int a[2\*maxn], b[2\*maxn], x[maxn], y[maxn];

void getwei()

{

int i, j=1, k;

for(i=1;i<maxn;i++)

if(!a[i]){

a[i] = 1, a[i+j] = 1;

b[i] = i+j, b[i+j] = i;

x[j] = i, y[j] = i+j;

j++;

}

}

## Nim博弈：

Nim游戏定义：有若干堆石子，每堆石子的数量都是有限的，合法的移动是“选择一堆石子并拿走若干颗（不能不拿）”，如果轮到某个人时所有的石子堆都已经被拿空了，则判负。

设第i堆的石子数为ai，若a1^a2^…^an==0，则先手必败否则必胜。

### 阶梯博弈：

从左到右有n阶阶梯，编号为1~n，地面为0号阶梯。第i阶阶梯上有ai个石子。每次可以将阶梯上任意数量的石子向其左侧移动(即从i移动至i-1)。无法移动的人数。

等价于编号为奇数的阶梯进行NIM博弈。若对手移动奇数阶梯，则同样移动奇数阶梯进行NIM博弈，若移动偶数阶梯，则与其进行同样操作。

### Anti-nim游戏：

Nim游戏定义：有若干堆石子，每堆石子的数量都是有限的，合法的移动是“选择一堆石子并拿走若干颗（不能不拿）”，取走最后一个石子者败。

先手必胜当且仅当：

1. 所有堆的石子都为1，且游戏的SG值为0；
2. 存在堆的石子大于1，且游戏的SG值不为0；

策略：

当所有石子都为1时，只能顺序取。

当存在堆石子大于1时。若SG值不为0，若只有一堆石子大于1，则可以选择这堆留0个或1个，保证剩下偶数个1，若不止一堆，则先手将SG值变为0即可。若SG值为0，无论先手如何操作，其对手都会面临SG值不为0的状态，使其面对先手必胜态。

## Fibonacci博弈:

有一堆石子，两人进行博弈。首次取可以取任意多个，但不可全取；后面每次最少取1个，最多取对手上次所取的石子数的两倍。取走最后一个的为赢家。

当石子数为Fibonacci数时，先手必败。

## SG组合游戏

### SG值：

任何一个公平组合游戏都可以通过把每个局面看成一个顶点，对每个局面和它的子局面连一条有向边来抽象成这个“有向图游戏”。下面我们就在有向无环图的顶点上定义Sprague-Grundy函数。首先定义mex(minimal excludant)运算，这是施加于一个集合的运算，表示最小的不属于这个集合的非负整数。例如mex{0,1,2,4}=3、mex{1,3,5}=0、mex{}=0。

对于一个给定的有向无环图，定义关于图的每个顶点的Sprague-Grundy函数g如下：g(x)=mex{ g(y) | y是x的后继 }。

若组合游戏的规则与NIM博弈类似，求出每个单一游戏的SG值时，可以将其等价于NIM博弈。

### Anti-SG:

Anti-SG游戏规定，决策集合为空的游戏者赢，其余规则与SG游戏相同。

SJ定理：对于任意一个Anti- SG游戏，如果我们规定当局面中所有的单一游戏的SG值为 0时，游戏结束，则先手必胜当且仅当：（1）游戏的SG函数不为0且游戏中某个单一游戏的SG函数大于1；（2）游戏的 SG函数为0且游戏中没有单一游戏的SG函数大于1 。

# 常用技巧、特定问题

## 约瑟夫环问题

N个人轮流报数，数字为M的人离开，并从下一个人开始重新报数，求最后剩下的一个人。

设初始编号为0,1,2,3,…,n-1,(m<=n)，则现在将会淘汰编号为m-1的人，将剩下的人按顺序重新编号,则m,m+1,m+2…,1,2,..m-2变为0,1,2,3,..n-2，

设人数为i-1时留下的人此轮编号为p,人数为i时留下的人此轮编号为q，

则q = (p+m)%i.

递推式：

f[i] = (f[i-1]+m)%i;(编号从0开始)

返回第k个被淘汰的人的编号(从1开始)：

### N较小，复杂度O(n)

LL yuesefu1(LL n, LL m, LL k)

{

    LL result = (m-1)%(n-k+1);

    for(LL i=n-k+2;i<=n;i++)

        result = (result+m)%i;

    return result+1;

}

### N较大,M较小，复杂度O(mlogn)

LL yuesefu2(LL n, LL m, LL k)

{

LL result = (m-1)%(n-k+1), w = 1, t;

while(w < k){

t = (n-k+w-result+m-2)/(m-1);

if(w + t >= k)

result = (result+m\*(k-w))%n;

else

result = (result+m\*t)%(n-k+w+t);

w = w+t;

}

return result+1;

}

## 分数规划：

### 最优比率生成树：

POJ2728：

三维空间中有n个点，两点间连边长度pij为其平面距离，花费zij为其垂直高度差。求一棵生成树，最小化。

#include<stdio.h>

#include<iostream>

#include<cstdlib>

#include<cmath>

#include<algorithm>

#include<cstring>

#include<map>

#include<vector>

#include<queue>

#include<iterator>

#define dbg(x) cout<<#x<<" = "<<x<<endl;

#define INF 0x3f3f3f3f

#define eps 1e-7

using namespace std;

typedef long long LL;

typedef pair<int, int> P;

const int maxn = 1020;

const int mod = 998244353;

int vis[maxn], x[maxn], y[maxn], z[maxn];

double dis[maxn], d[maxn][maxn], w[maxn][maxn], b[maxn][maxn];

double gettree(int n);

int main()

{

int n, i, j, k;

while(scanf("%d", &n), n)

{

for(i=0;i<n;i++){

scanf("%d %d %d", &x[i], &y[i], &z[i]);

}

for(i=0;i<n;i++)

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

w[i][j] = w[j][i] = sqrt((1.0\*x[i]-x[j])\*(x[i]-x[j])+(1.0\*y[i]-y[j])\*(y[i]-y[j]));

b[i][j] = b[j][i] = abs(z[i]-z[j]);

}

double l = 0, r = 1e5;

while(l+eps<r)

{

double mid = (l+r)/2;

for(i=0;i<n;i++)

{

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

d[i][j] = d[j][i] = b[i][j]-mid\*w[i][j];

d[i][i] = 0;

}

if(gettree(n)<=0)r = mid;

else l = mid;

}

printf("%.3f\n", l);

}

return 0;

}

double gettree(int n)

{

int i, j, k;

double sum = 0;

memset(vis, 0, sizeof(vis));

for(i=0;i<n;i++)dis[i] = d[0][i];

vis[0] = 1;

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

{

int u = -1;

for(j=0;j<n;j++)

if(!vis[j] && (u == -1 || dis[u] > dis[j]))

u = j;

if(u == -1)break;

vis[u] = 1;

sum += dis[u];

for(j=0;j<n;j++)

if(!vis[j])

dis[j] = min(dis[j], d[u][j]);

}

return sum;

}

## 挡板问题：

有n个容器，m个物品，将m个物品放到n个容器里，容器之间相互区分(即[0,2],[2,0]算两种方案)，求方案数：

容器不能为空：可以看成m个物品插入n-1个隔板，将其分为n块，即对应每种方案，方案数为C(m-1,n-1);

容器能为空：则假设每个容器都“借”一个物品，共n+m个物品，则可以转化为容器不为空的状态，方案数为：C(n+m-1, n-1)。

## 离散化：

离散化并去重，返回去重后的长度

int lisan(int dic[], int n)

{

int l=1;

sort(dic+1, dic+1+n);

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

if(dic[i] != dic[l])

dic[++l] = dic[i];

}

return l;

}

## 输入·输出挂 by kuangbin：

输入输出挂并不一定比scanf，printf快

template <class T>

inline bool scan\_d(T &ret) {

char c; int sgn;

if(c=getchar(),c==EOF) return 0; //EOF

while(c!='-'&&(c<'0'||c>'9')) c=getchar();

sgn=(c=='-')?-1:1;

ret=(c=='-')?0:(c-'0');

while(c=getchar(),c>='0'&&c<='9') ret=ret\*10+(c-'0');

ret\*=sgn;

return 1;

}

inline void out(int x) {

if(x>9) out(x/10);

putchar(x%10+'0');

}

# 杂项

## 优先队列自定义结构体：

struct node

{

int num, id;

node(int a, int b):num(a),id(b){}

bool operator < (const node& b) const

{

return num>b.num;

}

};

## Bitset：

C++的 bitset 在 bitset 头文件中，它的每一个元素只能是０或１，每个元素仅用１bit空间。支持位运算。

声明：bitset<N> bit1; //N：长度(位数)

函数：

count():求bitset中1的位数

size():求bitset的大小

any():检查bitset中是否存在位为1

none():检查bitset中是否没有1

all():检查bitset中是否全为1

flip(int x):将下标为x位的数取反

set():无参数时全置1，set(int x),将第x位置1

reset():无参数时全置0，reset(int x)，将第x为置0

to\_string():将bitset转换成string类型

to\_ulong():将bitset转换成unsigned long类型

to\_ullong():将bitset转换成unsigned long long类型

## 特殊值、库函数

lower\_bound(g.begin(),g.end(),x):返回第一个>=x的位置的迭代器

upper\_bound(g.begin(),g.end(),x):返回第一个>x的元素的迭代器

ceil(x):返回大于等于x的最小整数（向上取整）

floor(x):返回小于等于x的最大整数（向下取整）

PI(Π)：acos(-1.0)

角度制x转弧度制：x \* PI / 180

e:exp(double x)求e的x次幂

dev支持c++11：编译环境加上-std=c++11

重定向从文件读取：freopen("data.txt", "r", stdin);

# 定理、性质篇：

## 数论：

### 欧拉函数性质：

1. 对于质数p，φ(p)=p-1;
2. 若p为质数，n=p^k,则φ(n)=p^k-p^(k-1);
3. 欧拉函数是积性函数，但不是完全积性函数。若m,n互质，则φ(m∗n)=φ(m)∗φ(n)。特殊的，当m=2，n为奇数时，φ(2\*n)=φ(n)。
4. 当n>2时，φ(n)是偶数。
5. 小于n的数中，与n互质的数的总和为：φ(n) \* n / 2 (n>1)。
6. n=,即n的因数（包括1和它自己）的欧拉函数之和等于n。

### 原根：

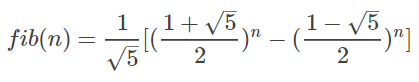
1. 模m有原根的充要条件是m= 1, 2 , 4 , p , 2p , pn，其中p是奇质数，n是任意正整数。
2. 当模m有原根时，它有φ(φ(m))个原根。

### 二项式定理：

2n = +++……+

### 斐波那契数列性质：

通项公式：



gcd( Fn, Fn-1 ) = 1

gcd( Fn, Fm ) = F( gcd(n,m) )

### Bertrand猜想：

对于任意n>3,都存在n<p<2\*n, 其中p为质数

### 费马小定理：

如果p是一个质数，而整数a不是p的倍数，则有a^（p-1）≡1（mod p）。

### 欧拉定理：

若n,a为正整数，且n,a互质，则: https://gss2.bdstatic.com/9fo3dSag_xI4khGkpoWK1HF6hhy/baike/pic/item/0823dd54564e92584b7cba389d82d158cdbf4e9f.jpg

### 错排公式：

递推：f(1) = 0, f(2)=1, f(n) = (n-1)\*[f(n-2)+f(n-1)]。

公式：

### 威尔逊定理：

( p - 1 ) ! % p = -1, p是素数

### 约数个数：

要求一个数约数的个数，设它分解质因数的结果为p1a1 \* p2a2 \* p3a3 \*……\*pm am（pi为质因数，ai为其对应的指数），那么该数的约数的个数为(a1+1)\*(a2+1)\*……\*(am+1)。

### 约数和：

要求一个数约数的和，设它分解质因数的结果为p1a1 \* p2a2 \* p3a3 \*……\*pm am（pi为质因数，ai为其对应的指数），那么该数的约数的和为\*\*……\*。

### Zeckendorf定理（齐肯多夫定理）：

任何正整数可以表示为若干个不连续的Fibonacci数之和。

### 费马平方和定理

奇质数能表示为两个平方数之和的充分必要条件是该质数被4除余1。

### 平方和

设正整数n的质因数分解为n=ai ，则x2+y2 = n 有整数解的充要条件为：当ai为奇数时，。该定理其实就是上面的费马平方和定理。

### 等差等比数列求和：

等差数列{an}的通项公式为：an=a1+(n-1)d。前n项和公式为：Sn=n\*a1+n(n-1)d/2或Sn=n(a1+an)/2 。

等比数列{an}的通项公式为：an=a1\*q^(n-1)。前n项和公式为：Sn = a1\*(1-q^n)/(1-q) = (a1-an\*q)/(1-q)。

### Farey序列

分子分母均不大于N且分数值在[0,1]中的最简真分数，升序排列，叫做Farey序列。设为序列Fn。

性质：

1. 除了N为1，Fn的分数个数都为奇数个，且中间一个为1/2，序列左右对称，两者和为.
2. 对于任意三个连续的分数a/b,e/f,c/d,有e/f=(a+c)/(b+d)，且a\*f-b\*e = e\*d-f\*c = 1。可推出c=⌊(N+b)/f⌋\*e-a, d = ⌊(N+b)/f⌋\*f-b。

### 公式：

1\*2+2\*3+3\*4+4\*5+…..+n\*(n+1) = n\*(n+1)\*(n+2)/6

## 几何：

### 圆方程

### 叉积：

向量P1(x1,y1) P2(x2,y2),其叉积代表以P1, P2构成的平行四边形的有向面积，如果P1 X P2为正，则P1在P2顺时针方向；如果为负，P1在P2逆时针方向如果为0，则共线(方向相同或相反)

P1ⅹP2 = x1 y2 - x2 y1 = - P2ⅹP1

### 正弦定理

a/sinA=b/sinB=c/sinC=2R(R为其外接圆半径)

### 余弦定理

a2 = b2 + c2 - 2bc \* cosA

cosA =

## 图论：

### 欧拉公式：

平面中，V顶点数量，E面数量，F边数量， 有：V-F+E = 2

### 路径计数

一个图的邻接矩阵的L次幂，i，j代表长度为L的从i到j的路径的数量

### Dilworth定理（狄尔沃斯定理）

狄尔沃斯定理亦称偏序集分解定理，是关于偏序集的极大极小的定理，该定理断言：对于任意有限偏序集，其最大反链中元素的数目必等于最小链划分中链的数目。对偶形式亦真，它断言：对于任意有限偏序集，其最长链中元素的数目必等于其最小反链划分中反链的数目

## Java大整数：

韩信点兵

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.InputStreamReader;

import java.io.OutputStreamWriter;

import java.io.PrintWriter;

import java.math.BigInteger;

import java.util.Scanner;

class Solve

{

private static BigInteger x;

private static BigInteger y;

public BigInteger kBigInteger;

int n;

BigInteger []ai=new BigInteger[300];

BigInteger []bi=new BigInteger[300];

public static BigInteger exGcd(BigInteger a, BigInteger b) {

if (b.compareTo(BigInteger.valueOf(0)) == 0) {

x = BigInteger.valueOf(1);

y = BigInteger.valueOf(0);

return a;

} else {

BigInteger d = exGcd(b, a.mod(b));

BigInteger temp = x;

x = y;

y = temp.subtract(a.divide(b).multiply(y));

return d;

}

}

public BigInteger excrt()

{

BigInteger M=bi[1],ans=ai[1];//第一个方程的解特判

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

{

BigInteger a=M,b=bi[i],c=ai[i].subtract(ans.mod(b)).add(b).mod(b);

BigInteger gcd=exGcd(a,b);

BigInteger bg=b.divide(gcd);

if(!c.mod(gcd).equals(BigInteger.valueOf(0))) {

return BigInteger.valueOf(-1);

}

c=c.divide(gcd);

x=x.multiply(c);

x=x.mod(bg);

ans=ans.add(x.multiply(M));

M=M.multiply(bg);

ans=ans.mod(M).add(M).mod(M);

}

return ans.mod(M).add(M).mod(M);

}

}

public class Main {

public static void main(String[] args) {

// TODO 自动生成的方法存根

Scanner in = new Scanner(new BufferedReader(new InputStreamReader(System.in)));

PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(System.out)));

Solve solve = new Solve();

solve.n=in.nextInt();

solve.kBigInteger=in.nextBigInteger();

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

solve.bi[i]=in.nextBigInteger();

solve.ai[i]=in.nextBigInteger();

}

BigInteger ans=solve.excrt();

if(ans.equals(BigInteger.valueOf(-1)))

out.println("he was definitely lying");

else if(ans.compareTo(solve.kBigInteger)>0)

out.println("he was probably lying");

else

out.println(ans);

out.flush();

}

}