[ACM 模板]

[by mickeyandkaka]

数论

快速幂

```
LL product_mod(LL a,LL p,LL mod){
	LL r = 0;
	while(p){
		if(p&1) r = (r+a)%mod;
		a = (a+a)%mod;
		p>>=1;
	}
	return r;
}

LL power_mod(LL a,LL p,LL mod){
	LL r = 1;
	while(p){
		if(p&1) r = product_mod(r,a,mod);
		a = product_mod(a,a,mod);
		p>>=1;
	}
	return r;
}
```

扩展欧几里德与同余方程

```
LL exgcd(LL a, LL b, LL &x, LL &y)
{
	if(!b) {x = 1,y = 0;return a;}
	LL d = exgcd(b, a % b, x, y);
	LL t = x;x = y;y = t - (a / b) * y;
	return d;
}
/*
对 ax+by=c
1.若 c==0  特判;
2.c%gcd(a,b)!=0 无解;
3.保证 a,b 为正，负号放到 x,y 中;
4.计算 ax'+by' = gcd(a,b)  得到一组  x0 = x'*c/gcd(a,b) y0 = y'*c/gcd(a,b);
5.整数解为：
x = x0 + b/gcd(a,b) * k
y = y0 - a/gcd(a,b) * k    (k 为整数)
6.由 a,b 还原 x,y 正负号;

X 的最小非负整数解
t = b/gcd(a,b);
x = (x%t+t)%t;
if(原来 a<0)    x-=t,x=-x;
```

```
*/

//逆元 a/b%p = a*(b^-1)%p
//要保证 a%b == 0
//b, p 不互质时逆元不存在
LL inv(LL b,LL p)
{
	LL x,y,d,t;
	d = exgcd(b,p,x,y);
	t = p/d;
	x = (x%t+t)%t;
	return x;
}

//线性同余方程 (ax = b) % p
//-1 无解 否则返回最小非负整数
//共有 d 个解 x0+k(n/d) k~[0,n-1]均为解
LL MLES(LL a, LL b, LL p)
{
	LL x, y, d = exgcd(a, p, x, y);
	if(b % d) return -1;
	x = x*b/d;
	return (x%p+p)%p;
}

// len 是个数, b[]是余数,w[]模数
LL china(int len, LL b[], LL w[])
{
	LL d, x, y, m, n = 1, res = 0;
	for(int i=1; i<=len; i++) n *= w[i];
	for(int i=1; i<=len; i++)
	{
		m = n / w[i];
		d = exgcd(w[i], m, x, y);
		res = (res + y * m % n * b[i] % n) % n;
	}
	return (n + res % n) % n;
}

// 扩展中国剩余
LL exchina(int len,LL b[],LL w[])
{
	LL tp = 1, tlcm = 1, res = 0;
	for(int i=1; i <=len; i++)
```

```
    {
        LL k = MLES(tp, b[i] - res, w[i]);
        if(k == -1) return -1;
        tlcm = lcm(tlcm, w[i]);
        res = (res + k * tp) % tlcm;
        tp = tlcm;
    }
    return res;
}
```

素数筛选

```
const int MAX = 4E7; //1s 数量级
bool isprm[MAX+10];
int prm[2500000];

int select(int n)
{
    clr(isprm, true);
    int cnt = 0;
    for(int i=2; i<=n; i++)
    {
        if(isprm[i])    prm[++cnt] = i;
        for(int j=1; j<=cnt && LL(i)*prm[j]<=n; j++)
        {
            isprm[ i*prm[j] ] = false;
            if(i % prm[j] == 0)   break;
        }
    }
    return cnt;
}

//最小素因子
const int MAX = 2E7;
bool isprm[MAX+10];
int prm[MAX+10],fac[MAX+10];

int minfac(int n)
{
    fac[1] = 1;
    clr(isprm, true);
    int cnt = 0;
    for(int i=2; i<=n; i++)
    {
```

```
        if(isprm[i])        prm[++cnt] = fac[i] = i;

        for(int j=1; j<=cnt && (LL)i*prm[j]<=n; j++)
        {
            isprm[ i*prm[j] ] = false;
            fac[ i*prm[j] ] = prm[j];
            if(i % prm[j] == 0)    break;
        }
    }
    return cnt;
}

//区间筛[L,R]素数  R-L 不能太大
if(L == 1)    L++;
clr(ans,true);
for(int i=1; i<=cnt && (LL)prime[i]*(LL)prime[i]<=R; i++)
{
    LL st = L/prime[i]+(L%prime[i]>0);
    if(st==1)      st++;
    for(LL j=st*prime[i]; j<=R; j+=prime[i])
        ans[j-L] = false;
}
c = 0;
for(LL i=L; i<=R; i++)
    if(ans[i-L])      t[++c] = i;
```

素数测试

```
LL product_mod(LL a,LL p,LL mod);
LL power_mod(LL a,LL p,LL mod);

bool isprime(LL n)
{
    if(n==2)return true;
    if(n<2 || !(n&1))return false;
    int i,j,k = 0;
    LL pri[] = {2,3,5,7,11,13,17,23,37,51,61};
    LL a,m = n-1;
    while(m % 2 == 0) m>>=1,k++;
    for(i=0; i<11; i++)
    {
        if(pri[i]>=n)      return true;
        a = power_mod(pri[i], m, n);
        if(a == 1) continue;
```

```
        for(j=0; j<k; j++)
        {
            if(a == n-1)     break;
            a = product_mod(a,a,n);
        }
        if(j==k)     return false ;
    }
    return true;
}
```

欧拉函数

```
//x 的欧拉函数 事先要 sqrt(x)的质数表
int euler(int x)
{
    int res = x;
    for(int i=1; prime[i]<=sqrt(x*1.0)+1 && i<=tot; i++)
        if(x%prime[i] == 0)
        {
            res = res/prime[i]*(prime[i]-1);
            while(x%prime[i] == 0) x/=prime[i];
        }
    if(x>1) res = res/x*(x-1);
    return res;
}


//1~n 的欧拉函数
const int MAX = 1E6+10;//8E6 为极限
int phi[MAX];

void euler(int n)
{
    for(int i=1; i<=n; i++) phi[i] = i;
    for(int i=2; i<=n; i+=2)     phi[i]/=2;
    for(int i=3; i<=n; i+=2)
        if(phi[i] == i)
            for(int j=i; j<=n; j+=i)
                phi[j] = phi[j]/i*(i-1);
}
```

整数因子分解

```
LL gcd(LL a,LL b);
LL product_mod(LL a,LL p,LL mod);
LL power_mod(LL a,LL p,LL mod);
bool isprime(LL n);

//1
LL pollard_rho(LL c,LL n)//某个因子,返回 n 失败
{
	int i=1,k=2;
	LL x=rand()%n, y=x;
	do{
		i++;
		LL d = gcd(n+y-x,n);
		if(d>1 && d<n)	return d;
		if(i==k)	y=x,k*=2;
		x = (product_mod(x,x,n)+n-c)%n;
	}
	while(y!=x);
	return n;
}

//2
LL rho(LL n)//最小素因子,返回 n 失败
{
	if(isprime(n))	return n;
	while(1){
		LL t = pollard_rho(rand()%(n-1)+1,n);
		if(t<n){
			LL a=rho(t),b=rho(n/t);
			return a<b?a:b;
		}
	}
}

LL p[100],c[100];//c 先存个数,随后存因数
int cnt;//记得初始为 0

//3
void add(LL n)//将素因子添加到素因子表中
{
	int i;
	for(i=1;i<=cnt;i++)
```

```
        if(p[i]==n) break;
    if(i<=cnt) c[i]++;//第 i 个素因子个数加 1
    else
        p[++cnt]=n,c[cnt] = 1;
}

//4 因数分解
void factor(LL n)
{
    if(n < 2)    return;
    if(isprime(n))    add(n);
    else{
        LL p = pollard_rho(rand()%(n-1)+1,n);
        factor(p),factor(n/p);
    }
}

//5 合并相同的素因子 c[]数组存放因子
for(i=1;i<=cnt;i++)
{
    for(j=1,k=p[i]; j<c[i]; j++)    k*=p[i];
    c[i]=k;
}

//具体用法 调用 4;5;
```

矩阵操作

```
const int mod;
const int MAX = 300;//上限
//注意矩阵下标以 0 开始
struct MAT
{
    int r, c;
    LL d[MAX][MAX];

    MAT(int size) //初始为单位矩阵
    {
        r=c=size;
        clr(d,0);
        for(int i=0; i<size; i++)    d[i][i]=1;
    }
    MAT(int _r,int _c) //初始为全零矩阵
    {
```

```
        r=_r,c=_c;
        clr(d,0);
    }

    friend MAT operator+(const MAT& m1,const MAT &m2);
    friend MAT operator*(const MAT& m1,const MAT &m2);
};

MAT operator+(const MAT& m1,const MAT &m2)
{
    MAT ret(m1.r,m2.c);

    for(int i=0; i<m1.r; i++)
        for(int j=0; j<m1.c; j++)
            ret.d[i][j] = (m1.d[i][j] + m2.d[i][j])%mod;
    return ret;
}

MAT operator*(const MAT& m1,const MAT &m2)
{
    MAT ret(m1.r,m2.c);
    for(int i=0; i<m1.r; i++)
        for(int j=0; j<m1.c; j++)
            if(m1.d[i][j])
                for(int k=0; k!=m2.c; k++)
                    if(m2.d[j][k])
                        ret.d[i][k] = (ret.d[i][k] + (m1.d[i][j] * m2.d[j][k]))%mod;
    return ret;
}
MAT pow(MAT &a, int p)
{
    MAT t = a, ans(a.r);
    for(; p; p>>=1)
    {
        if(p & 1)     ans = ans * t;
        t = t * t;
    }
    return ans;
}

MAT sum(MAT &a, int p)
{
    int r = a.r;
    MAT t(2*r), ret(r);
```

```
    for(int i=0; i<r; i++)
        for(int j=0; j<r; j++)
            t.d[i][j] = t.d[i][j + r] = a.d[i][j];
    t = pow(t,p);

    for(int i=0; i<r; i++)
        for(int j=0; j<r; j++)
            ret.d[i][j] = t.d[i][j + r];
    return ret;
}
```

反素数

```
LL ans,sum,n;
LL prime[16]={2,3,5,7,11,13,17,19,23,29,31,37,41,43,47};
void dfs(LL cur,LL cnt,int k,int limit)
{
    if(cur > n) return ;
    if(cnt > sum || (cnt == sum && cur < ans))
        ans = cur,sum = cnt;
    LL p = prime[k];
    for(int i=1;i<=limit;i++,p*=prime[k])
    {
        if(cur*p > n)     break;
        dfs(cur*p,cnt*(i+1),k+1,i);
    }
}
```

求出<=n 的反素数

dfs(13,1,1,1);

//约数的个数记做 g(x) 如果某个正整数 x 满足:对于任意 i(0<i<x),都有 g(i)<g(x),则称 x 为反素数.

//性质一:一个反素数的质因子必然是从 2 开始连续的质数.

//性质二:p=2^t1*3^t2*5^t3*7^t4.....必然 t1>=t2>=t3>=....

//对于 N<=1e100 质数到 350 以内 int lim[] = {0,13, 9, 5, 4, 2, 2, 2, 2, 2, 1}; 枚举上界质数

附表：

1,2,4,6,12,24,36,48,60,120,180,240,360,720,840,1260,1680,2520,5040,7560,10080,15120,20160,25200,27720,45360,50400,55440,83160,110880,166320,221760,277200,332640,498960,554400,665280,720720,1081080,1441440,2162160,2882880,3603600,4324320,6486480,7207200

Mobius 反演

Mobius 和 容斥原理 有点关系。

$F(n)=\sum_{d|n} f(d)$ 则: $f(n)=\sum_{d|n}\mu(d)F(n/d)$

$$\mu(m)=\begin{cases}1, m=1;\\(-1)^k, m\text{是}k\text{个不同素数乘积}, k\geq 1;\\0,\text{其他情形}.\end{cases}$$

举例：令 $R(M,N)= 1\le x\le M,\ 1\le y\le N$ 中 gcd(x,y)=1 的个数

我们说 G(z)表示 gcd(x,y)是 z 的倍数的个数（以后都省略 $1\le x\le M,1\le y\le N$ 的前提），即 z|gcd(x,y)，

那么 $G(z)=\lfloor M/z\rfloor*\lfloor N/z\rfloor$。令 F(z)表示 gcd(x,y)=z 的个数，所以 $G(z)=\sum(F(z)+F(2*z)+F(3*z)...)$，

于是我们得到 $F(z)=\sum(G(z)*\mu(z)+G(2*z)*\mu(2*z)...)$，从而得到了我们最终的答案

$ans=\sum z\ge1\lfloor M/z\rfloor*\lfloor N/z\rfloor*\mu(z)$

对于其中连续的 k 项，有可能有 a/x=a/(x+k);b/x=b/(x+k),可以优化。

例:POI 求 gcd(i,j)=d,1<=i<=a,1<=j<=b

```
const int MAX = 1E5;
bool isprm[MAX+10];
int prm[MAX+10],mu[MAX+10];

int calmu(int n){
    mu[1] = 1;
    clr(isprm, true);
    int cnt = 0;
    for(int i=2; i<=n; i++){
        if(isprm[i])     prm[++cnt] = i,mu[i] = -1;
        for(int j=1; j<=cnt && (LL)i*prm[j]<=n; j++)
        {
            isprm[ i*prm[j] ] = false;
            if(i % prm[j] == 0)
            {
                mu[ i*prm[j] ] = 0;
                break;
            }
            else     mu[ i*prm[j] ] = -mu[i];
        }
```

```
        }
        return cnt;
}

int sum_mu[MAX+10];
int T,a,b,d;

int main(){
        sum_mu[0] = 0;
        calmu(50000);
        for(int i=1; i<=50000; i++) sum_mu[i] = sum_mu[i-1] + mu[i];

        scanf("%d",&T);
        while(T--){
                scanf("%d%d%d",&a,&b,&d);
                int ans = 0;

                if(a > b)      swap(a,b);
                a/=d,b/=d;

                for(int i=1; i<=a; ){
                        int now = min(a/(a/i),b/(b/i));
                        ans += ((a/i)*(b/i))*(sum_mu[now]-sum_mu[i-1]);
                        i = now+1;
                }
                printf("%d\n",ans);
        }
        return 0;
}
```

Gcd(i,j) = gcd(j,k) = gcd(i,k) = 1 1<=i<=a,1<=j<=b,1<=k<=c

```
if(i <= a)   now = min(now, a/(a/i));

if(i <= b)   now = min(now, b/(b/i));

if(i <= c)   now = min(now, c/(c/i));

ans += ((a/i+1)*(b/i+1)*(c/i+1)-1)*(sum_mu[now]-sum_mu[i-1]);
```

Gcd(i,j,k) = 1

a(n)=sum(k=1, n, mu(k)*floor(n/k)^3)

Romberg 积分

```
double romberg(double aa, double bb)
{
    int m, n;//m 控制迭代次数, 而 n 控制复化梯形积分的分点数. n=2^m
    double h, x;
    double s, q;
    double ep; //精度要求
    double *y = new double[MAXREPT];//为节省空间,只需一维数组
    //每次循环依次存储 Romberg 计算表的每行元素,以供计算下一行,算完后更新
    double p;//p 总是指示待计算元素的前一个元素(同一行)

    //迭代初值
    h = bb - aa;
    y[0] = h*(fun(aa) + fun(bb))/2.0;
    m = 1;
    n = 1;
    ep = eps + 1.0;

    //迭代计算
    while ((ep >= eps) && (m    < MAXREPT))
    {
        //复化积分公式求 T2n(Romberg 计算表中的第一列),n 初始为 1,以后倍增
        p = 0.0;
        for (int i=0; i <n; i++)//求 Hn
        {
            x = aa + (i+0.5)*h;
            p = p + fun(x);
        }
        p = (y[0] + h*p)/2.0;//求 T2n = 1/2(Tn+Hn),用 p 指示

        //求第 m 行元素,根据 Romberg 计算表本行的前一个元素(p 指示),
        //和上一行左上角元素(y[k-1]指示)求得.
        s = 1.0;
        for (int k=1; k <=m; k++)
        {
            s = 4.0*s;
            q = (s*p - y[k-1])/(s - 1.0);
            y[k-1] = p;
            p = q;
        }

        p = fabs(q - y[m-1]);
        m = m + 1;
```

```
		y[m-1] = q;
		n = n + n; h = h/2.0;
	}
	return (q);
}
```

pell 方程（java）

```
//求 x^2-ny^2=1 的最小解  n 不为完全平方数
class pell{
	BigInteger[] solve(int n){
		BigInteger ans[] = new BigInteger[2];
		BigInteger N, p0, p1, q0, q1, a0, a1, a2, g1, g2, h1, h2, p, q;

		a0 = a1 = BigInteger.valueOf((long)Math.sqrt(n*1.0));
		h1 = p1 = q0 = BigInteger.ONE;
		g1 = p0 = q1 = BigInteger.ZERO;
		N = BigInteger.valueOf(n);
		for(int i=2; ;i++){
			g2 = a1.multiply(h1).subtract(g1);		//g2=a1*h1-g1
			h2 = N.subtract(g2.pow(2)).divide(h1);	//h2=(n-g2^2)/h1
			a2 = g2.add(a0).divide(h2);			//a2=(g2+a0)/h2
			p = a1.multiply(p1).add(p0);
			q = a1.multiply(q1).add(q0);
			if(p.pow(2).subtract(N.multiply(q.pow(2))).equals(BigInteger.ONE)){
				ans[0] = p;
				ans[1] = q;
				break;
			}
			g1 = g2;h1 = h2;a1 = a2;
			p0 = p1;p1 = p;
			q0 = q1;q1 = q;
		}
		return ans;
	}
}
```

高斯消元

```
//普通整数 gauss a[n][n]*x[n]=b[n] (mod p);
//true 返回唯一整数解  a[i][i]*xi = b[i], false 非唯一
bool gauss(int n){
	int i,j,row=1,col=1;
```

```
    LL t1,t2,maxi;
    while(row <=n && col <=n){
        for(i=row,j=-1,maxi = 0; i<=n; i++)
            if(labs(a[i][col]) > maxi){
                j = i;
                maxi = labs(a[i][col]);
            }
        if(j == -1) return false;

        for(i=col; i<=n; i++)
            swap(a[row][i],a[j][i]);
        swap(b[row],b[j]);

        t1 = a[row][col];
        for(i=row+1; i<=n; i++){
            t2 = a[i][col];
            for(j = col; j<=n; j++)
                a[i][j] = (a[i][j]*t1 - a[row][j]*t2+p)%p;
            b[i] = (b[i]*t1-b[row]*t2+p)%p;
        }
        row++,col++;
    }

    for(i = n; i>=1; i--){
        LL sum = 0;
        for(j = i+1; j <= n; j++)
            sum = (sum + a[i][j]*ans[j])%p;
        ans[i] = find(a[i][i],b[i]-sum);
    }
    return true;
}

//整数高斯
//-2 表示有浮点数解，但无整数解，-1 表示无解，0 表示唯一解，大于 0 表示无穷解，并返回自由变元的个数
int gauss(int equ,int var,int p)
{
    int i,j,row,col,maxr;

    for(row=col=1; row<=equ && col<=var; row++,col++)
    {
        for(maxr=i=row; i<=equ; i++)
            if(labs(a[i][col]) > labs(a[row][col]))
                maxr = i;
```

```
        if (maxr != row)
            for (j=col; j<=var+1; j++)
                swap(a[row][j], a[maxr][j]);

        if (!a[row][col])    {row--;continue;}

        for(i=row+1; i<=equ; i++)
        {
            if(a[i][col])
            {
                LL LCM = lcm(labs(a[row][col]),labs(a[i][col]));
                LL ta = LCM / labs(a[i][col]), tb = LCM / labs(a[row][col]);

                if (a[i][col]*a[row][col] < 0) tb = -tb;

                for (j = col; j <= var+1; j++)
                    a[i][j] = (a[i][j]*ta - a[row][j]*tb+p)%p;
                    //模 2 用位运算
                    //a[i][j]^=a[row][j];

            }
        }
    }

    for (i=row; i <= equ; i++)
        if (a[i][col])
            return -1;

    if (row-1 < var)
        return var-row+1;

    for(i=row-1; i>=1; i--)
    {
        LL sum = 0;
        for(j = i+1; j<=var; j++)
            sum = (sum + a[i][j]*ans[j])%p;
        a[i][var+1] = ((a[i][var+1]-sum)%p+p)%p;
        ans[i] = MLES(a[i][i],a[i][var+1],p);
    }
    return 0;
}
```

连分数

```
//num 为连分数序列  从 0 开始
//fun1 化分数为连分数,fun2 化序列为分数
int num[1000],pos;
void fun1(int a,int b,int& pos)
{
	num[pos] = a/b;
	a-=num[pos]*b;
	if(a==1)	{num[++pos] = b;return ;}
	fun1(b,a,++pos);
}
int a=0,b=1;
void fun2(int& a,int &b,int pos)
{
	a += b*num[pos];
	if(pos==0)	return;
	int t;
	t=a,a=b,b=t;
	fun2(a,b,pos-1);
}
```

康托展开

```
//1-n 排列  返回比他小的排列的个数
int fac[10] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880};
int zip(int *a, int n)
{
	int ans = 0, j, r;
	bool p[10] = {0};
	for (int i = 0; i < n; i++)
	{
		for (j = 1, r = 0; j <= a[i]; j++)
			if (p[j] == 0) r++;
		ans += (r - 1) * fac[n - 1 - i];
		p[ a[i] ] = 1;
	}
	return ans;
}

void unzip(int s, int *a,int n)
{
	int j, r;
```

```
    bool p[10] = {0};
    for (int i=0; i<n; i++)
    {
        int t = s / fac[n-1-i]+1;
        s %= fac[n-1-i];
        r = 0, j = 1;
        while (1)
        {
            if (p[j] == 0) r++;
            if (r == t) break;
            j++;
        }
        p[ a[i] = j ] = 1;
    }
}
```

杂

1）A^x = A^(x % Phi(C) + Phi(C)) (mod C)
2）n/d 设 d=2^A * 5^B * m 其中 gcd(m,10)=1 则小数中不循环部分的长度是 max(A,B) 循环节的长度是最小的某个 k 使得 10^k mod m = 1
3）inv[1]=1;
for(i=2; i<mod; i++)   inv[i] = inv[mod%i]*(mod-mod/i)%mod;

动态规划

背包

```
//W=V dp 判可达 cnt 表示当前的面值需要几个当前的物品
for(int i=1; i<=n; i++)
{
    num += c[i]*v[i];
    int bound = min(num,lim);
    clr(cnt,0);
    for(int j=v[i]; j<=bound; j++)
    {
        if(!dp[j] && dp[j-v[i]] && cnt[j-v[i]]<c[i])
        {
            cnt[j] = cnt[j-v[i]]+1;
            res++;
            dp[j] = true;
        }
```

```
        }
}
//单调队列优化
int w[60],v[60],a[60];
int n;

int complete(int n,int lim)    {
    for(int i=1; i<=n; i++)
    {
        for(int d=0; d<v[i]; d++)
        {
            st = 1,ed = 0;
            for(int j=0; j<=(lim-d)/v[i]; j++)
            {
                int val = dp[j*v[i] + d]-j*w[i];
                while(st<=ed && K[ed] <= val)    ed--;
                K[++ed] = val;
                L[ed] = j;
                while(st<=ed && L[st] < j-a[i])    st++;
                if(st <= ed)
                    dp[j*v[i]+d] = K[st]+j*w[i];
            }
        }
    }
    return dp[lim];
}

//分组背包
for(int i=1; i<=n; i++) //组数
    for(int j=m; j>=0; j--) //容量
        for(int k=1; k<=m; k++)//费用
            if(j>=k)//这里的费用就是 K  注意费用
                dp[j] = max(dp[j],dp[j-k]+w[i][k]);
```

斜率优化

```
LL f1(int p1,int p2) {    //分子
}

LL f2(int p1,int p2) {   //分母
}

LL g(int i,int j) {   //状态转移
}
```

```
//f1/f2 > k 且 k 单调不降
for(int i=k; i<=n; i++)
{
	//维护队尾后满足 f(a,b)<f(b,c)<f(c,d)<.....（严格单调递增）
	while(st < ed && f1(q[ed-1],q[ed])*f2(q[ed],i-k) >= f1(q[ed],i-k)*f2(q[ed-1],q[ed])) ed--;
	q[++ed] = i-k;

	//维护队头后满足 k<f(a,b)<f(b,c)<f(c,d)<
	while(st < ed && f2(q[st],q[st+1])*i <= f1(q[st],q[st+1])) st++;
	dp[i] = g(i,q[st]);
}
```

第一类四边形不等式

d[i,j] = min{d[i, k - 1] + d[k, j]} + w[i, j] | (i < k <= j)

s[i][j - 1] <= s[i][j] <= s[i+1][j]

```
int dp[1005][1005],s[1005][1005];
int i,n;

void solve(){
	int i,j,k,l;
	for(i=1;i<=n;i++){
		dp[i][i]=0;
		s[i][i]=i;
	}
	for(l=2;l<=n;l++){
		for(i=1;i<=n-l+1;i++){
			j=i+l-1;
			int minn=inf;
			for(k=s[i][j-1];k<=s[i+1][j];k++){
				int tmp=dp[i][k-1]+dp[k][j]+w(i,j);	//转移方程
				if(tmp<minn){
					minn=tmp;
					s[i][j]=k;
				}
			}
			dp[i][j]=minn;
		}
	}
}
```

第二类四边形不等式

条件: dp[i][j] = min(dp[i - 1][k] + w(k + 1, j))
dp[i][i] == 0; j >= i;
s[i - 1][j] <= s[i][j] <= s[i][j + 1];

```
int dp[2][10005],s[2][10005],a[10005];
int i,n,m;

int w(int k,int j){
	if(k>j) return inf;
	return (a[j]-a[k])*(a[j]-a[k]);
}
void solve(){
	int i,j,k;
	for(i=1;i<=n;i++){
		dp[1][i]=w(1,i);
		s[1][i]=1;
	}
	for(int i1=2;i1<=m;i1++){
		i=(i1&1);
		for(j=n;j>=1;j--){
			dp[i][j]=inf;
			int bg=s[i^1][j],ed=s[i][j+1];
			if(j==n) ed=n;
			for(k=bg;k<=ed;k++){
				int tmp=dp[i^1][k-1]+w(k,j);
				if(tmp<dp[i][j]){
					dp[i][j]=tmp;
					s[i][j]=k;
				}
			}
		}
	}
	return dp[m&1][n];
}
```

数位 DP

```
int dig[20],dp[位][];

LL dfs(int pos, ... , bool bound)
{
```

```
    if(!pos)        return ;
    if(!bound && ~dp[pos][])    return dp[pos][];

    LL ret = 0;
    int end = bound ? dig[pos] : 9;
    for(int i=0; i<=end; i++)
    {
        ....
        ans += dfs(pos-1, ... , bound && i==end);
    }

    if(!bound)    dp[pos][o][pre] = ret;
    return ret;
}

LL cal(LL x)
{
    int pos = 0;
    while(x)
    {
        dig[++pos] = x % 10;
        x /= 10;
    }
    LL ans = dfs(pos, ..., 1);
    return ans;
}
```

字符串

最小表示

```
int MinPresent(char *s) //返回最小表示的起始位置  0 开始计算
{
    int l = strlen(s);
    int i = 0, j = 1, k = 0, t;
    while(i<l && j<l && k<l)
    {
        t = s[(i+k) >= l ? i+k-l : i+k] - s[(j+k) >= l ? j+k-l : j+k];
        if(!t) k++;
        else
        {
            if(t>0) i = i+k+1;//<为最大表示
```

```
            else j = j+k+1;
            if(i==j) ++j;
            k = 0;
        }
    }
    return (i < j ? i : j);
}
```

KMP

```
/*p 多开一位
s = "a   b   a   b   a   c   a   b   a"
p = -1   0   0   1   2   3   0   1   2   3*/

int fail[100010];
void makefail(char *p){
    int len = strlen(p),j=0;
    fail[0]=-1,fail[1] = 0;
    for(int i=2; i<=len; i++){
        while(j>0 && p[j]!=p[i-1])    j = fail[j];
        if(p[j] == p[i-1])    ++j;
        fail[i] = j;
    }
}

int kmp(char *s,char *p){
    int len = strlen(s),lenp=strlen(p);
    int i,j = 0,cnt = 0;

    for(i=j=0; i<len; i++){
        while(j>0 && s[i] != p[j])      j = fail[j];
        if(s[i] == p[j]) ++j;
        if(j == lenp)      {cnt++; j=fail[j];}
    }
    return cnt;
}
```

Manacher

```
const int MAX = 120000;
char str[MAX],r[2*MAX];
int p[2*MAX];
```

```
int manacher()
{
    int len = strlen(str);
    int n = 0;
    r[n++] = '$',r[n++] = '#';
    for(int i=0; i<len; i++){
        r[n++] = str[i];
        r[n++] = '#';
    }
    r[n] = 0;

    int mx = 0,id;
    for(int i=1; i<n; i++){
        if(mx > i)    p[i] = min(p[2*id-i],mx-i);
        else        p[i] = 1;

        for(;r[i-p[i]] == r[i+p[i]]; p[i]++);
        if(p[i]+i > mx) mx=p[i]+i,id=i;
    }
    int ans = 0;
    for(int i=1; i<n; i++)
        ans = max(ans,p[i]-1);
    return ans;
}
```

后缀数组

一般解题用到 height 数组+二分答案
字串计数：height 扫一便，算贡献
最长重叠/不重叠字串：　二分答案 判断组的大小和最大最小位置的差值

```
/*
rank[0...7]: 4 6 8 1 2 3 5 7
string:          a a b a a a a b
-------------------------------------------
  sa[1] = 3 : a a a a b             height[1] = 0
  sa[2] = 4 : a a a b               height[2] = 3
  sa[3] = 5 : a a b                 height[3] = 2
  sa[4] = 0 : a a b a a a a b       height[4] = 3
  sa[5] = 6 : a b                   height[5] = 1
  sa[6] = 1 : a b a a a a b         height[6] = 2
  sa[7] = 7 : b                     height[7] = 0
  sa[8] = 2 : b a a a a b           height[8] = 1
*/
```

数据结构

线段树框架

```
//某层有 LAZY 标记  当前层一定要计算好  记住是计算好了的
//标记下传的时候  左右儿子的 LAZY 和  数据  要计算好  （根据上面）
//当前层的 LAZY  清空
//叶子的 LAZY 没用
//该线段数为左闭右开[L,R)   但是接口为闭区间[L,R]  小心
#define L(x)      x<<1
#define R(x)      x<<1|1
struct st
{
	int l,r;
	LL lazy,maxi;
}seg[100010<<2];

void pushup(int rt)
{}

void pushdown(int rt)
{
	if(seg[rt].lazy)
	{
		seg[rt].lazy = 0;
	}
}

void build(int l,int r,int rt = 1)
{
	if(rt == 1)      r++;
	seg[rt].l = l, seg[rt].r = r;
	seg[rt].lazy = 0;

	if(l + 1 == r)
	{
		seg[rt].maxi= 0;
		return ;
	}

	int mid = (l + r)>>1;
	build(l, mid, L(rt));
```

```
    build(mid, r, R(rt));

    pushup(rt);
}

void modify(int l,int r,int val,int rt = 1)
{
    if(rt == 1) r++;
    if(l<=seg[rt].l && seg[rt].r<=r)
    {
        //操作
        return ;
    }
    pushdown(rt);

    int mid = (seg[rt].l + seg[rt].r)>>1;
    if(l < mid) modify(l,r,L(rt));
    if(r > mid) modify(l,r,R(rt));
    pushup(rt);
}

LL query(int l,int r,int rt = 1)
{
    if(rt == 1) r++;
    if(l <=seg[rt].l && seg[rt].r<=r)
        return seg[rt].maxi;
    pushdown(rt);

    int mid = (seg[rt].l + seg[rt].r)>>1;
    LL maxi = 0;
    if(l < mid) maxi = max(maxi, query(l,r,L(rt)) );
    if(r > mid) maxi = max(maxi, query(l,r,R(rt)) );

    return maxi;
}
```

线段树操作

```
//段修改  段求和
void pushup(int rt)
{
    seg[rt].sum = seg[L(rt)].sum + seg[R(rt)].sum;
}
```

```
void pushdown(int rt)
{
	if(seg[rt].lazy)
	{
		seg[L(rt)].lazy += seg[rt].lazy;
		seg[R(rt)].lazy += seg[rt].lazy;
		seg[L(rt)].sum += seg[rt].lazy * (seg[L(rt)].r - seg[L(rt)].l);
		seg[R(rt)].sum += seg[rt].lazy * (seg[R(rt)].r - seg[R(rt)].l);
		seg[rt].lazy = 0;
	}
}

//段增加  段最值
void pushup(int rt)
	seg[rt].maxi = max(seg[L(rt)].maxi , seg[R(rt)].maxi);

void pushdown(int rt)
	if(seg[rt].lazy)
	{
		seg[L(rt)].lazy += seg[rt].lazy;
		seg[R(rt)].lazy += seg[rt].lazy;
		seg[L(rt)].maxi += seg[rt].lazy;
		seg[R(rt)].maxi += seg[rt].lazy;
		seg[rt].lazy = 0;
	}
```

树状数组第 K 个位置

```
int getk(int k)
{
	int ans = 0;
	for(int i=maxlog; i>=0; i--)
	{
		ans += 1<<i;
		if(ans > n || a[ans]>=k)    ans -= 1<<i;
		else  k -= a[ans];
	}
	return ans+1;
}
```

并查集

```
int p[N],k[N];
```

```
int find(int x)   {while (x != p[x]) p[x] = p[p[x]], x = p[x];return x;}

//种类并查集
int find(int x)
{
	if(x == p[x])	return x;
	int px = p[x];
	p[x] = find(p[x]);
	k[x] = (k[x] + k[px ]) % K;
	return p[x];
}

bool merge(int x,int y,int d) //x<-y d=a 的种类-b 的种类
{
	int fx = find(x), fy = find(y);
	if(fx == fy)	return false;
	p[fy] = fx;
	k[fy] = ( (k[x] - k[y] - d) % K + K) % K ;
	return true;
}
```

Treap

```
typedef int type;
struct node
{
	type key;
	int fix,sz,cnt;
	node*c[2];
	node(type v,node*n):key(v)
	{
		c[0]=c[1]=n,sz = cnt = 1;
		fix = rand();
	}
	void rz()	{sz=c[0]->sz+c[1]->sz+cnt;}
}*root,*null;

struct Treap
{
	Treap()
	{
		null = new node(0,0);
		null->sz=0,null->fix=INF;
		root=null;
```

```
}

void rotate(node* &t,bool d)
{
	node*c = t->c[d];
	t->c[d] = c->c[!d];
	c->c[!d]=t;
	t->rz(),c->rz();
	t=c;
}

void insert(node*&t,type x)
{
	if(t==null)
	{
		t = new node(x,null);
		return;
	}

	if(x==t->key)
	{
		t->cnt++,t->sz++;
		return;
	}
	bool d = x > t->key;
	insert(t->c[d],x);
	if(t->c[d]->fix < t->fix)	rotate(t,d);
	else	t->rz();
}

void Delete(node*&t,type x)
{
	if(t==null) return;
	if(t->key == x)
	{
		if(t->cnt > 1)
		{
			t->cnt--,t->sz--;
			return;
		}

		bool d=t->c[1]->fix < t->c[0]->fix;
		if(t->c[d]==null)
		{
```

```
                delete t;
                t=null;
                return;
            }
            rotate(t,d);
            Delete(t->c[!d],x);
        }
        else
        {
            bool d=x > t->key;
            Delete(t->c[d],x);
        }
        t->rz();
}

int find(node *t,type x)
{
        if (t==null)        return 0;
        else if (x < t->key)        return find(t->c[0],x);
        else if (x > t->key)        return find(t->c[1],x);
        else        return t->cnt;
}

type select(node* t,int k)
{
        int r=t->c[0]->sz;
        if(k <= r) return select(t->c[0],k);
        if(k > r+t->cnt) return select(t->c[1],k-r-t->cnt);
        return t->key;
}

int rank(node*t,type x)
{
        if(t==null) return 0;
        int r=t->c[0]->sz;
        if(x < t->key) return rank(t->c[0],x);
        if(x > t->key) return r+t->cnt+rank(t->c[1],x);
        return r+1;
}

type pred(node* t,type x)
{
        if(t == null)      return x;
        type ret;
```

```
        if(t->key < x)
        {
            ret = pred(t->c[1],x);
            if(ret == x)    return t->key;
            return ret;
        }
        else
            return pred(t->c[0],x);
    }

    type succ(node*t,type x)
    {
        if(t == null)    return x;
        type ret;
        if(t->key > x)
        {
            ret = succ(t->c[0],x);
            if(ret == x)    return t->key;
            return ret;
        }
        else
            return succ(t->c[1],x);
    }
};
```

2D RMQ

```
//2D
const int MAX = 301;
int dp[MAX][MAX][9][9];
int val[MAX][MAX],ln2[MAX];

void initRMQ(int n,int m)
{
    int i,j,u,v;
    //放到外面
    ln2[0] = ln2[1] = 0;
    for(int i=2; i<=300; i++) ln2[i] = ln2[i>>1]+1;

    for(i=1; i<=n; i++)
        for(j=1; j<=m; j++)
            dp[i][j][0][0] = val[i][j];
```

```
	for(u=0; u<=ln2[n]; u++)
		for(v=0; v<=ln2[m]; v++)	if(u|v)
			for(i=1; i+(1<<u)-1<=n; i++)
				for(j=1; j+(1<<v)-1<=m; j++)
				{
					if(u==0)
						dp[i][j][u][v]=max(dp[i][j][u][v-1],dp[i][j+(1<<(v-1))][u][v-1]);
					else
						dp[i][j][u][v]=max(dp[i][j][u-1][v],dp[i+(1<<(u-1))][j][u-1][v]);
				}
}

int query(int l1,int l2,int r1,int r2)
{
	if(l1>l2)swap(l1,l2);
	if(r1>r2)swap(r1,r2);
	int kn=ln2[l2-l1+1];
	int km=ln2[r2-r1+1];
	int v1=dp[l1][r1][kn][km];
	int v2=dp[l1][r2-(1<<km)+1][kn][km];
	int v3=dp[l2-(1<<kn)+1][r1][kn][km];
	int v4=dp[l2-(1<<kn)+1][r2-(1<<km)+1][kn][km];
	return max(max(v1,v2),max(v3,v4));
}
```

树链剖分(Qtree 1)

```
struct EDGE
{
	int v,w,next;
}edge[MAX<<1];
int head[MAX],e;
int son[MAX],fa[MAX],dep[MAX],sz[MAX],p[MAX];
int top[MAX],w[MAX],id[MAX],cnt;

struct st
{
	int l,r,w,maxi;
}seg[MAX<<2];

void dfs1(int u,int f,int depth)
{
	sz[u] = 1, dep[u] = depth, son[u] = -1, fa[u] = f;
	for(int i = head[u]; ~i; i = edge[i].next)
```

```
    {
        int v = edge[i].v;
        if(v == f)      continue;

        dfs1(v, u, depth+1);
        p[v] = i;
        if(son[u] == -1 || sz[v] > sz[son[u] ])
        {
            son[u] = v;
            sz[u] = sz[v] + 1;
        }
    }
}

void dfs2(int u)
{
    id[u] = ++cnt;
    if(~p[u])     w[cnt] = edge[p[u]].w;
    if(!top[u]) top[u] = u;
    if(~son[u]) top[son[u] ] = top[u], dfs2(son[u]);

    for(int i = head[u]; ~i; i=edge[i].next)
        if(edge[i].v != fa[u] && edge[i].v != son[u])
            dfs2(edge[i].v);
}

#define L(x)      x<<1
#define R(x)      x<<1|1
void build(int l, int r, int k)
{
    seg[k].l = l, seg[k].r = r;
    if(l + 1 == r)    {seg[k].maxi = seg[k].w = w[l];return ;}

    int mid = (l + r)>>1;
    build(l, mid, L(k));
    build(mid, r, R(k));
    seg[k].maxi = max(seg[L(k)].maxi, seg[R(k)].maxi);
}

void modify(int l,int r,int val,int k)
{
    if(l <= seg[k].l && seg[k].r <= r)
    {
        seg[k].w = seg[k].maxi = val;
```

```
        return ;
    }

    int mid = (seg[k].l + seg[k].r) >> 1;
    if(l < mid) modify(l,r,val,L(k));
    if(r > mid) modify(l,r,val,R(k));
    seg[k].maxi = max(seg[L(k)].maxi, seg[R(k)].maxi);
}

int query(int l,int r,int k)
{
    if(l <= seg[k].l && seg[k].r <= r)
        return seg[k].maxi;

    int mid = (seg[k].l + seg[k].r)>>1;
    int ret = 0;

    if(l < mid) ret = max(ret, query(l, r, L(k)));
    if(r > mid) ret = max(ret, query(l, r, R(k)));
    return ret;
}

int find(int u,int v)
{
    int f1 = top[u], f2 = top[v], ret = -INF ,l ,r;
    while (f1 != f2)
    {
        if (dep[f1] < dep[f2])    {swap(f1, f2); swap(u, v); }
        l = id[f1] + 1, r = id[u] + 1;
        if(f1 == u)    l--;
        ret = max(ret, query(l, r, 1));
        ret = max(ret,query(id[f1],id[f1]+1,1));
        u = fa[f1]; f1 = top[u];
    }
    if(u == v)    return ret;
    if(dep[u] > dep[v]) swap(u, v);
    l = id[u] + 1, r = id[v] + 1;
    return max(ret, query(l, r , 1));
}

void addedge(int u,int v,int w)
{
    edge[e].v = v, edge[e].w = w;
    edge[e].next = head[u];
```

```
        head[u] = e++;
}

int T,n,a,b,c;
char cmd[10];

int main()
{
        //freopen("D:\\in.txt","r",stdin);
        for(scanf("%d",&T); T; T--)
        {
                scanf("%d",&n);
                clr(head,-1), e = 0;
                for(int i=1; i<n; i++)
                {
                        scanf("%d%d%d",&a,&b,&c);
                        addedge(a,b,c);
                        addedge(b,a,c);
                }

                p[1] = -1, dfs1(1, 1, 1);
                clr(top, 0), cnt = -1, dfs2(1);
                if(cnt > 0) build(1, cnt+1, 1);

                while(1)
                {
                        scanf("%s",cmd);
                        if(cmd[0] == 'C')
                        {
                                scanf("%d%d",&a,&b);
                                a = (a-1)<<1;
                                int u = edge[a^1].v, v = edge[a].v, t;

                                if(dep[u] > dep[v]) t = u;
                                else        t = v;

                                int pos = id[t];
                                modify(pos, pos+1, b, 1);
                        }
                        else if(cmd[0] == 'Q')
                        {
                                scanf("%d%d",&a,&b);
                                if(a == b)    printf("0\n");
                                else
```

```
                printf("%d\n",find(a,b));
            }
            else    break;
        }
    }
    return 0;
}
```

LCT

```
#define type int
#define keytree root->ch[1]->ch[0]
const int N = 300005,inf=0x3f3f3f3f;
struct Node{
    int val,maxv,minv,lazy,id,size,sum,rev;
    Node *ch[2], *pre ;
    int isroot,isnull;
    void Add(int v){
        if(id<0)return ;
        lazy+=v;
        val+=v;
        minv+=v;
        maxv+=v;
    }
    void Update(){
        if(id<0)return ;
        size = ch[0]->size + ch[1]->size + 1;
        minv = min(val, min(ch[0]->minv, ch[1]->minv));
        maxv = max(val, max(ch[0]->maxv, ch[1]->maxv));
        sum = val + ch[0]->sum + ch[1]->sum;
    }
    void Reverse(){
        if(id<0)return ;
        swap(ch[0],ch[1]);
        rev^=1;
    }
    void PushDown(){
        if(id<0)return ;
        if (lazy){
            ch[0]->Add(lazy);
            ch[1]->Add(lazy);
            lazy=0;
        }
```

```
        if(rev){
            ch[0]->Reverse();
            ch[1]->Reverse();
            rev=0;
        }

    }
};
type arr[N];
Node* Hash[N]; // Hash[i]指向 id = i 的节点，方便查找其位置(id 值唯一)

class LinkCut{
    int eid,n;
    int head[N],ed[N<<1],nxt[N<<1];
    Node *stk[N], data[N];
    int cnt, top;
    Node *null;
    bool vis[N];
public:
    int val[N];
    /*
    * 获得一个新的节点，之前删除的节点会放到 stk 中以便再利用
    * id 表示这个节点的编号，会和 Hash 数组对应起来，编号从 1 到 n
    */
    Node *NewNode(int id,type var){
        Node *p;
        if (top) p = stk[top--];
        else p = &data[cnt++];
        p->val = p->minv = p->maxv = var;
        p->id = id;
        p->size = p->isroot = 1;
        p->isnull = 0;
        p->lazy = 0; p->sum = var;
        p->ch[0] = p->ch[1] = p->pre = null;
        if(id>0)Hash[id]=p;
        return p;
    }
    void AddEdge(int s,int e){
        ed[eid]=e;nxt[eid]=head[s];head[s]=eid++;
        ed[eid]=s;nxt[eid]=head[e];head[e]=eid++;
    }

    void Init(int n){
        this->n=n;
```

```
	top=cnt=eid=0;
	clr(head,-1);
	null=NewNode(-1,inf);
	null->size=0;
	null->maxv=-inf;
	null->sum=0;
	null->isnull=1;
}
/*
* 旋转操作, c=0 表示左旋, c=1 表示右旋
*/
void Rotate(Node *x, int c){
	Node *y = x->pre;
	y->PushDown();
	x->PushDown();
	y->ch[!c] = x->ch[c];
	if (x->ch[c] != null)
		x->ch[c]->pre = y;
	x->pre = y->pre;
	if(y->isroot)y->isroot=0,x->isroot=1;
	else y->pre->ch[ y == y->pre->ch[1] ] = x;
	x->ch[c] = y;
	y->pre = x;
	y->Update();
}
/*
 * 旋转使 x 成为根节点
 * x 会执行 pushdown 和 update 的操作
 */
void Splay(Node *x){
	x->PushDown();
	while (!x->isroot){
		if (x->pre->isroot){
			Rotate(x, x->pre->ch[0] == x);
			break;
		}
		Node *y = x->pre;
		Node *z = y->pre;
		int c = (y == z->ch[0]);
		if (x == y->ch[c]){// 之字形旋转
			Rotate(x, !c);Rotate(x, c);
		}
		else{// 一字形旋转
			Rotate(y, c);Rotate(x, c);
```

```
        }
    }
    x->Update();
}
void Dfs(int s,Node * f){
    Node *p=NewNode(s,val[s]);
    p->pre=f;
    vis[s]=1;
    for(int i=head[s];~i;i=nxt[i])
        if(!vis[ed[i]])Dfs(ed[i],p);
}
/*
 * 构建根据给定的森林用 dfs 建立 link-cut tree。
 */
void BuildTree(){
    clr(vis,0);
    for(int i=1;i<=n;i++)
        if(!vis[i])Dfs(i,null);
    for(int i=1;i<=n;i++)
        Access(Hash[i]);
}
/*
 * 访问节点 x,从根节点一直访问到 x 并更新 Auxiliary Tree。
 * 返回值是包含节点 x 的 Auxiliary Tree 的根
 */
Node* Access(Node * x){
    Node* y;
    for(y=null;x!=null;y=x,x=x->pre){
        Splay(x);
        x->ch[1]->isroot=1;
        x->ch[1]=y;y->isroot=0;
        x->Update();
    }
    return y;
}
/*
 * 返回节点 x 所在的树的根节点
 */
 Node* FindRoot(Node * x){
     x=Access(x);
     while(x->ch[0]!=null)x=x->ch[0];
     Splay(x);
     return x;
 }
```

```
void Evert(Node *x){
    Access(x)->Reverse();
}

void AddVal(Node* a,Node* b,int v){
      Evert(a);
      b=Access(b);
      b->Add(v);
}
/*
 * 取链上的节点最小值，关于链的策略与链上加值一样
 */
int GetMin(Node* a,Node* b){
      Evert(a);b=Access(b);
      return b->minv;
}
/*
 * 取链上的节点最大值，关于链的策略与链上加值一样
 */
int GetMax(Node* a,Node* b){
      Evert(a);b=Access(b);
      return b->maxv;
}
int GetSum(Node* a,Node* b){
      Evert(a);b=Access(b);
      return b->sum;
}
void Cut(Node* x){
      Access(x);Splay(x);
      x->ch[0]->isroot=1;
      x->ch[0]->pre=null;
      x->ch[0]=null;
}
void Link(Node* x,Node* y){
      Evert(x);Splay(x);
      x->pre=y;
      Access(x);
}
Node* FindFa(Node* x){
    Access(x);
    Node* y=x->ch[0];
    if(y==null)return y;
    while(y->ch[1]!=null)y=y->ch[1];
```

```
			Splay(y);
			return y;
		}
}lct;

int n;

int main(){
//	freopen("/home/axorb/in","r",stdin);
//	freopen("/home/axorb/out","w",stdout);
	while(~scanf("%d",&n)){
		lct.Init(n);
		for(int i=1;i<n;i++){
			int a,b;scanf("%d%d",&a,&b);
			lct.AddEdge(a,b);
		}
		for(int i=1;i<=n;i++){
			int a;scanf("%d",&a);
			lct.val[i]=a;
		}
		lct.BuildTree();
		int m;scanf("%d",&m);
		while(m--){
			int a,b,c,d;
			scanf("%d",&a);
			if(a==3)scanf("%d",&d);
			scanf("%d%d",&b,&c);
			int rb=lct.FindRoot(Hash[b])->id;
			int rc=lct.FindRoot(Hash[c])->id;
			if(a==1){
				if(rb==rc)
					puts("-1");
				else
					lct.Link(Hash[b],Hash[c]);
			}else if(a==2){
				if(b==c||rb!=rc)puts("-1");
				else{
					lct.Evert(Hash[b]);
					lct.Cut(Hash[c]);
				}
			}else if(a==3){
				if(rb!=rc)puts("-1");
				else{
					lct.AddVal(Hash[b],Hash[c],d);
```

```
                    }
                }else{
                    if(rb!=rc)puts("-1");
                    else printf("%d\n",lct.GetMax(Hash[b],Hash[c]));
                }
            }
//          lct.debug();
            puts("");
        }
}
```

归并排求逆序对

```
void merge_sort(LL *A, int x, int y, LL *T)  //[x,y)
{
    if(y-x > 1)
    {
        int m = x +(y-x)/2;
        int p = x,q = m;

        merge_sort(A, x, m ,T);
        merge_sort(A, m, y ,T);

        int i = x;

        while(p < m || q < y)
        {
            if(q>=y || (p<m && A[p] <= A[q]))
                T[i++] = A[p++];
            else
            {
                T[i++] = A[q++];
                cnt += m-p;
            }
        }
        for(int i=x; i<y; i++)
            A[i] = T[i];
    }
}
```

树上倍增

```
dfs 算 dep[] p[][0] dis[][0]
for (int i = 1; i < 20; ++i)
            for(int u = 1; u <= n; u++)
                if (p[u][i - 1] != -1)
                {
                    p[u][i] = p[ p[u][i - 1] ][i - 1];
                    if(p[u][i] != -1)
                        dis[u][i] = dis[u][i-1] + dis[ p[u][i - 1] ][i - 1];
                }

int moveDep(int x,int dep)
{
    while (dep > 0) {
        x = p[x][ lg2[dep] ];
        dep -= 1 << lg2[dep];
    } return x;
}

int lca(int x,int y)
{
    if (dep[x] > dep[y]) x = moveDep(x, dep[x] - dep[y]);
    else  y = moveDep(y, dep[y] - dep[x]);
    while(x != y)
    {
        int now = 0;
        while(p[x][now] != p[y][now] )      now++;
        if(now) now--;
        x = p[x][now], y = p[y][now];
    }
    return x;
}
```

图论

TARJAN + 缩点

```
const int NV = 10010;
const int NE = 50010;

int n,m;
```

```
bool instack[NV];
int dfn[NV],low[NV],s[NV],belong[NV],color[NV];
int scc,dindex,stop;
int head[NV],e;
struct E
{
	int u,v,next;
} edge[NE];

struct Tarjan
{
	Tarjan()
	{
		clr(head,-1),e = scc = dindex = stop = 0;
		clr(dfn,0),clr(low,0),clr(instack,false);
	}

	void addedge(int u,int v)
	{edge[e].u = u,edge[e].v = v,edge[e].next = head[u],head[u] = e++;}

	void tarjan(int x)
	{
		int y;
		dfn[x] = low[x] = ++dindex;
		instack[x] = true,s[++stop] = x;
		for(int i=head[x]; ~i; i=edge[i].next)
		{
			y = edge[i].v;
			if(!dfn[y])
			{
				tarjan(y);
				low[x] = min(low[x],low[y]);
			}
			else if(instack[y])
				low[x] = min(low[x],dfn[y]);
		}

		if(dfn[x] == low[x])
		{
			scc++;
			do{
				y = s[stop--];
				instack[y] = false;
				belong[y] = scc;
```

```
                color[scc]++;
            }while(x != y);
        }
    }

    void solve()
    {for(int i=1; i<=n; i++) if(!dfn[i]) tarjan(i);}
};
//缩点
int out_degree[NODE_MAX];//每个强联通分量的出度
int color[NODE_MAX];//每个强联通分量的点数

void solve()
{
    clr(out_degree,0);
    for(int i=1; i<=m; i++)
        if(belong[edge1[i].st] != belong[ edge1[i].ed ])
            out_degree[ belong[edge1[i].st] ]++;

    int tmp = 0,pos;
    for(int i=1; i<=scc; i++)
        if(out_degree[i] == 0){
            ++tmp;
            pos = i;
        }

    if(tmp != 1)     printf("0\n");
    else     printf("%d\n",color[pos]);
}
```

匈牙利

```
int vis[XMAX],mat[YMAX];
int head[XMAX],e;

bool dfs(int x)
{
    vis[x] = 1;
    for(int i=head[x]; ~i; i=edge[i].nxt)
    {
        int y = edge[i].v;
        if(vis[mat[y]])    continue;
        if(!mat[y] || dfs(mat[y]))
        {
```

```
                mat[y] = x;
                return true;
            }
        }
        return false;
}

int hungary(int n)
{
        int ans = 0;
        clr(mat,0);
        for(int i=1; i<=n; i++)
        {
            clr(vis,0);
            if(dfs(i))    ans++;
        }
        return ans;
}
```

HK

```
int head[XMAX],e;
bool vis[YMAX];
int dx[XMAX],dy[YMAX];
int mx[XMAX],my[YMAX];

bool bfs(int n)
{
        queue<int> q;
        clr(dx,-1),clr(dy,-1);
        for(int i=1; i<=n; i++)
            if(mx[i] == -1)
                q.push(i),dx[i]= 0;

        bool flag = false;
        while(!q.empty())
        {
            int u = q.front();
            q.pop();
            for(int i=head[u]; ~i; i=edge[i].next)
            {
                int v = edge[i].v;
                if(dy[v] == -1)
                {
```

```
                dy[v] = dx[u]+1;
                if(my[v] == -1)
                    flag = true;
                else
                {
                    dx[ my[v] ] = dy[v]+1;
                    q.push( my[v] );
                }
            }
        }
    }
    return flag;
}

bool dfs(int u)
{
    for(int i=head[u]; ~i; i=edge[i].next)
    {
        int v = edge[i].v;
        if(!vis[v] && dy[v] == dx[u]+1)
        {
            vis[v] = 1;
            if(my[v]==-1 || dfs(my[v]))
            {
                my[v] = u, mx[u] = v;
                return true;
            }
        }
    }
    return false;
}

int HK(int n)
{
    int ans = 0;
    clr(mx,-1),clr(my,-1);
    while(bfs(n))
    {
        clr(vis,false);
        for(int i=1; i<=n; i++)
            if(mx[i] == -1 && dfs(i))
                ans++;
    }
    return ans;
```

```
}
```

KM

```
int mx[MAX],my[MAX],lx[MAX],ly[MAX];
bool vx[MAX], vy[MAX];
int nx, ny, g[MAX][MAX]; //g 需要初始化

struct KM
{
	KM(int x,int y){
		nx = x,ny = y;
		clr(ly,0),clr(mx,-1),clr(my,-1);
	}

	bool path(int u){
		vx[u] = 1;
		for(int v=1; v<=ny; v++) if (g[u][v] == lx[u] + ly[v] && !vy[v])
			{
				vy[v] = 1;
				if (my[v] == -1 || path(my[v])){
					mx[u] = v,my[v] = u;
					return true;
				}
			}
		return false;
	}

	int solve(){
		int ret = 0,j;
		for(int i=1; i<=nx; i++)
			for(lx[i]=-INF, j=1; j<=ny; j++)
				lx[i] = max(lx[i], g[i][j]);
		for(int u=1; u<=nx; u++)
			if (mx[u] == -1){
				clr(vx,0),clr(vy,0);
				while(!path(u)){
					int ex=INF;
					for(int i=1; i<=nx; i++)	if (vx[i])
							for(int j=1; j<=ny; j++)	if(!vy[j])
									ex = min(ex, lx[i] + ly[j] - g[i][j]);
				for(int i=1; i<=nx; i++)	if (vx[i])  lx[i] -= ex,vx[i] = 0;
				for(int j=1; j<=ny; j++)	if (vy[j])  ly[j] += ex,vy[j] = 0;
				}
```

```
        }
        for(int i=1; i<=nx; i++)        ret += g[i][mx[i]];
        return ret;
    }
};
```

差分约束

```
//注意建图
//a-b<=c  加(b,a,c)的一条有向边
//spfa 判负环(约束是否有解)    应该能加 SLF 优化
//加额外点 v0  和每个点连一条 0 权边    具体加不加看题意
//求最大值用最短路(<=)，求最小值用最长路(>=)  注意设置起点。

int head[NV],e,d[NV];
int ed[NE],w[NE],nxt[NE];
bool inq[NV];
int cnt[NV];

void addedge(int u,int v,int val)
{
    ed[e] = v,w[e] = val;
    nxt[e] = head[u],head[u] = e++;
}

deque<int> q;
int n,m;

int spfa(int st)
{
    for(int i=1; i<=n+1; i++) d[i] = (i==st ? 0 : INF);
    for(int i=1; i<=n+1; i++) inq[i] = cnt[i] = (i==st);

    while(!q.empty())      q.pop_back();
    q.push_back(st);

    while(!q.empty())
    {
        int u = q.front();    q.pop_front();
        inq[u] = false;

        for(int i = head[u]; ~i; i=nxt[i])
        {
            int v = ed[i];
```

```
            if(d[v] > d[u] + w[i])
            {
                d[v] = d[u] + w[i];
                cnt[v]++;
                if(cnt[v] >= n+1)     return 0;

                if(!inq[v])
                {
                    if(q.empty()|| d[v]<d[q.front()])    q.push_front(v);
                    else q.push_back(v);
                    inq[v]    = true;
                }
            }
        }
    }
    return true;
}
```

网络流

```
//必要时 edge.u 删除
const int NV = 110,NE = 20500;
int e,head[NV],d[NV],vd[NV],pre[NV],cur[NV];
struct E
{
    int u,v,w,next;
}edge[NE];

struct FlowNetwork{
    FlowNetwork()     {e=0;clr(head,-1);}
    inline void addedge(int u,int v,int w){
        edge[e].u = u;edge[e].v = v;edge[e].w = w;edge[e].next = head[u];head[u]=e++;
        edge[e].u = v;edge[e].v = u;edge[e].w = 0;edge[e].next = head[v];head[v]=e++;
    }

    int sap(int s,int t,int n){
        int i,u,mini,ans = 0;
        clr(d,0);clr(vd,0);
        vd[0] = n;
        cur[u = s] = head[s];
        pre[s] = -1;

        if(s == t)  return 0;
        while(d[s] < n){
```

```
            if(u == t){
                for(mini = INF,i = pre[u]; ~i; i = pre[edge[i].u])
                    mini = min(mini,edge[i].w);
                for(i = pre[u]; ~i; i = pre[edge[i].u])
                    edge[i].w -= mini,edge[i^1].w += mini;
                ans += mini; u = s;
            }

            for(i = cur[u]; ~i; i = edge[i].next)
                if(edge[i].w >0 && d[u] == d[edge[i].v]+1){
                    cur[u] = i;
                    pre[u = edge[i].v] = i;
                    break;
                }

            if(i == -1){
                cur[u] = head[u];
                if(--vd[d[u]] == 0) break;
                vd[++d[u]]++;
                if(u != s)    u = edge[pre[u]].u;
            }
        }
        return ans;
    }
};
```

费用流

```
typedef int typef;
typedef int typec;
const int NV = 5100, NE = 40010;
const typef INFF = 0x3f3f3f3f;
const typec INFC = 0x3f3f3f3f;

bool vis[NV];
int e,head[NV],dist[NV],pre[NV],road[NV];
struct E{
    int v,next;
    typec cost;
    typef cap;
}edge[NE];
queue<int>    q;

struct MCMF{
```

```
MCMF() {e = 0,clr(head, -1);}

void  addedge(int  u,  int  v, typef f, typec c){
    edge[e].v=v, edge[e].cap=f, edge[e].cost = c, edge[e].next=head[u], head[u]=e++;
    edge[e].v=u, edge[e].cap=0, edge[e].cost =-c, edge[e].next=head[v], head[v]=e++;
}

bool spfa(int s, int t, int n){
    for(int i=1; i<=n; i++)  dist[i] = INFC,vis[i] = 0;//最大改-INFC
    dist[s] = 0,pre[s] = s;
    q.push(s);
    while (!q.empty()){
        int u = q.front();
        q.pop(),vis[u] = false;

        for (int i = head[u]; ~i; i = edge[i].next){
            if (edge[i].cap <= 0) continue;
            int v = edge[i].v;
            if (dist[v] > dist[u] + edge[i].cost)//最大改<
            {
                dist[v] = dist[u] + edge[i].cost;
                pre[v] = u,road[v] = i;
                if (!vis[v])
                    q.push(v),vis[v] = 1;
            }
        }
    }
    return dist[t] != INFC;//最大改-INFC
}

void mincost(int s, int t, int n, typef &f, typec &c){
    c = f = 0;
    if(s == t)  return ;
    while(spfa(s, t, n)){
        typef minf = INFF;
        for(int u = t; u != s; u = pre[u])
            minf = min(minf, edge[road[u]].cap);
        for(int u = t; u != s; u = pre[u]){
            edge[road[u]].cap -= minf;
            edge[road[u]^1].cap += minf;
        }
        f += minf;
        c += minf * dist[t];
    }
```

```
	}
};
```

Havel 定理

Havel 定理:给定度序列 判断无向图能否简单图化
1: (pre) if di>=n || sum(di)%2==1 fail
2: if di<0 fail
3: if all di == 0 success
4: reorder the di to non-increasing order
5: k = d1 and remove d1
6: subtract 1 from the first k term in the remaining
7: goto 2.

K 短路

```
const int NV = 5010;
const int NE = 200010;

struct node
{
	int pos,w,h;
	bool operator < (const node& b)const{
		return h > b.h;
	}
	node(){}
	node(int a,int b,int c)
	{
		pos = a,w=b,h=c;
	}
};

priority_queue<node> q;
int vis[NV];
int head[NV],e,d[NV];
int ed[NE],w[NE],nxt[NE];

void addedge(int u,int v,int val)
{
	ed[e] = v,w[e] = val;
	nxt[e] = head[u],head[u] = e++;
}
```

```
void dijkstra(int st,int n)
{
    for(int i=1; i<=n; i++) d[i] = (i==st ? 0 : INF);
    clr(vis,0);
    while(!q.empty())    q.pop();
    q.push(node(st,0,d[st]));

    while(!q.empty())
    {
        node t = q.top(); q.pop();
        int u = t.pos;

        if(vis[u])    continue;
        vis[u] = 1;
        for(int i=head[u]; ~i; i=nxt[i])
        {
            if(i%2 == 0)        continue;//看图的有向无向
            int v = ed[i], val = w[i];
            if(d[v] > d[u]+val)
            {
                d[v] = d[u]+val;
                q.push(node(v,0,d[v]));
            }
        }
    }
}

int astar(int st,int end,int k)
{
    clr(vis,0);
    while(!q.empty())    q.pop();
    q.push(node(st,0,d[st]));

    while(!q.empty())
    {
        int u = q.top().pos;
        int curw = q.top().w;
        q.pop();

        vis[u]++;
        if(u == end && vis[u] == k)    return curw;
        if(vis[u] > k) continue;

        for(int i=head[u]; ~i; i=nxt[i])
```

```
        {
            if(i&1) continue;//看图的有向无向
            int v = ed[i], val = w[i];
            q.push(node(v,curw+val,curw+val+d[v]));
        }
    }
    return -1;
}
```

Steiner 树

```
int dp[NV][1<<6];
bool vis[NV][1<<6];

struct node
{
    int u,s;
    node() {}
    node(int a,int b):u(a),s(b){}
};
queue<node> q;

int steiner(int n,int m)
{
    clr(dp,0x3f),clr(vis,false);
    for(int i=0; i<=m; i++) dp[i][0] = 0;

    while(!q.empty())    q.pop();

    for(int i=0; i<n; i++)
    {
        int s = (1<<i);
        dp[i+1][s] = 0;
        vis[i+1][s] = true;
        q.push(node(i+1,s));
    }

    while(!q.empty())
    {
        int u=q.front().u, s=q.front().s;
        q.pop();

        vis[u][s]=false;
```

```
            for(int t=0; t<(1<<n); t++)//不同层
            {
                if(s&t) continue;
                int st=(s|t);
                if(dp[u][st] > dp[u][s] + dp[u][t])
                {
                    dp[u][st] = dp[u][s] + dp[u][t];
                    if(!vis[u][st])
                    {
                        vis[u][st] = true;
                        q.push(node(u,st));
                    }
                }
            }

            for(int i=head[u]; ~i; i=nxt[i])//同层
            {
                int v=ed[i];
                if(dp[v][s] > dp[u][s] + w[i])
                {
                    dp[v][s] = dp[u][s] + w[i];
                    if(!vis[v][s])
                    {
                        vis[v][s]=true;
                        q.push(node(v,s));
                    }
                }
            }
    }
    return dp[0][(1<<n)-1];
}
```

离线 LCA

```
int n,m;
const int NV = 40010, NE = 40010*2, NQ = 210;
int p[NV];
int find(int x) {while (x != p[x]) p[x] = p[p[x]], x = p[x];return x;}
void merge(int x,int y) {int px = find(x),py = find(y);p[py] = px;}

struct E{
    int v,w,next;
} edge[NE];
```

```
struct Q{
    int v,id,next;
} query[NQ];

int head2[NV],q,head[NV],e,ans[NQ],lv[NV];
bool check[NV];

struct LCA{
    LCA(int n){
        clr(head,-1),e=0,clr(head2,-1),q=0;
        clr(lv,-1),clr(check,false);
        for(int i=1; i<=n; i++) p[i] = i;
    }

    void addedge(int u,int v,int w){
        edge[++e].v=v,edge[e].w = w,edge[e].next = head[u],head[u] = e;
        edge[++e].v=u,edge[e].w = w,edge[e].next = head[v],head[v] = e;
    }

    void addquery(int u,int v,int id){
        query[++q].v = v,query[q].id=id,query[q].next = head2[u],head2[u] = q;
        query[++q].v = u,query[q].id=id,query[q].next = head2[v],head2[v] = q;
    }

    void lca(int u,int d){
        lv[u] = d, p[u] = u;
        for(int i = head[u]; ~i; i=edge[i].next){
            int v=edge[i].v , w=edge[i].w;
            if(~lv[v])    continue;
            lca(v,d+w);
            merge(u,v);
        }
        check[u] = true;
        for(int i=head2[u]; ~i; i=query[i].next){
            int v=query[i].v , id = query[i].id;
            if(check[v])        ans[id] = lv[v]+lv[u]-2*lv[find(v)];
        }
    }
};
```

生成树计数

```
//C = D-G D 度数矩阵  G  邻接矩阵
int gauss(int n)
```

```
{
    int ret = 1;
    for(int i=1; i<=n; i++)
    {
        for(int j=i+1; j<=n; j++)
            while(C[j][i])
            {
                int t = C[i][i]/C[j][i];
                for(int k=i; k<=n; k++)
                    C[i][k] = (C[i][k]-C[j][k]*t + mod)%mod;//mod 偏小的话当心
                for(int k=i; k<=n; k++)
                    swap(C[i][k], C[j][k]);
                ret = -ret;
            }
        if(C[i][i] == 0) return 0;
        ret = ret*C[i][i]%mod;
    }
    return (ret+mod)%mod;//mod 偏小的话当心
}
```

严格次小生成树

```
//bzoj 1977
const int N = 100010;
const int M = 300010;

struct E
{
    int u,v;
    int w;
    bool operator<(const E&a)const{
        return w<a.w;
    }
}edge[M];

bool chose[M];
int p[N];
int find(int x){
    return p[x] == x?x:p[x] = find(p[x]);
}

int head[N],e;
int ed[2*M],next[2*M];
int val[2*M];
```

```
void addedge(int u,int v,int w)
{
    ed[e] = v,val[e] = w;
    next[e] = head[u],head[u]= e++;
}

int max1[N],max2[N],min1,min2;
int num[N];
VI g[N];

void dfs(int u,int p)
{
    int len = g[u].size();
    for(int i=0; i<len; i++)    dfs(g[u][i],p);
    for(int i=head[u]; ~i; i=next[i])
    {
        if(chose[i>>1]) continue;
        int v= ed[i];
        LL w = val[i];
        if(find(v) == p)
        {
            chose[i>>1] = true;
            if(w < min1){min2 = min1;min1 = w;}
            else if(w < min2 && w>min1){min2 = w;}
        }
    }
}

int main()
{
    int n,m;
    while(~scanf("%d%d",&n,&m))
    {
        for(int i=0; i<m; i++)
            scanf("%d%d%d",&edge[i].u,&edge[i].v,&edge[i].w);
        sort(edge,edge+m);

        for(int i=1; i<=n; i++)
        {
            head[i] = -1;
            max1[i] = max2[i] = -1;
            g[i].clear();
            num[i] = 1;
            p[i] = i;
```

```
}
e = 0;
for(int i=0; i<m; i++)
{
    chose[i] = false;
    addedge(edge[i].u,edge[i].v,edge[i].w);
    addedge(edge[i].v,edge[i].u,edge[i].w);
}

LL mst = 0,ans = INF;

for(int i=0; i<m; i++)
{
    if(chose[i])    continue;
    int fx = find(edge[i].u);
    int fy = find(edge[i].v);
    int v = edge[i].w;
    mst+=v;
    chose[i] = true;

    if(num[fx] > num[fy])
        swap(fx,fy);

    LL v4[4] ={max1[fx],max2[fx],max1[fy],max2[fy]};
    max1[fx] = v;
    sort(v4,v4+4);
    for(int i=3; i>=0; i--)
        if(v4[i]!= v)
        {
            max2[fx] = v4[i];
            break;
        }

    min1 = min2 = INF;
    dfs(fx,fy);

    if(min1 != INF)
    {
        if(min1 == max1[fx])
        {
            if(~max2[fx])
                ans = min(ans,(LL)min(min1-max2[fx],min2 - max1[fx]));
        }
        else
```

```
                    ans = min(ans,(LL)min1-max1[fx]);

                }

                num[fx] += num[fy];
                g[fx].push_back(fy);
                p[fy] = fx;
            }
            printf("%lld\n",mst+ans);
        }

        return 0;
}
```

最小割的关键割边

判断边是否为某一最小割集的边：在残余网络中求强连通分量，顶点不在同一强连通分量且满流的边。
判断边是否为所有最小割集的边：在残余网络中求强连通分量，顶点不在同一强连通分量且满流且流出点与源点 S 同一连通分量、流入点与汇点 S 同一连通分量的边。

二分图的一点东西

最小顶点覆盖 ：在二分图中求最少的点，让每条边都至少和其中的一个点关联。
结论：二分图的最少顶点覆盖数 = 二分图的最大匹配数

二分图的最大独立集 ：在 N 个点的图 G 中选出 m 个点，使这 m 个点两两之间没有边
结论:最大独立集点数 = （左右总）节点数（N） - 最大匹配数
DAG 的最小路径覆盖数=DAG 图中的节点数-相应二分图中的最大匹配数
给定 n-n 二分图的一个完全匹配 求完全匹配的可能边：(poj 1904)
每个王子向喜欢的美女连接一条有向边，再根据匹配好的方案，每个美女向其匹配的王子连接一条有向边。。。构图后简化一下描述就是：如果从 A 点出发，最终能回到 A 点（成环）则能够维持完全匹配的，因此求一次强连通分量，判断每个王子与其喜欢的美女是否在同一强连通分量即可。

二分图的必须边：先根据已知信息建立二分图，通过坐标计算把可能存在的边赋值为 1，然后用匈牙利算法进行最大匹配，当前则得到了一个最大匹配，然后每次删除一条边，对该边的一个顶点进行再次匹配，如果匹配成功则不是必须边，只有匹配不成功才是必须边。

一点东西

最少区间覆盖：给定 n 个区间，用最少的区间覆盖整个区间。
解法:按左端点排序,每次对于还未覆盖的大区间 [L,R] ,在 n 个区间里面找 [st,ed] 的 st<=L 且 ed 最大的那个，不断更新即可。除排序外可以 O(n)实现。
最少点覆盖区间: 给定 n 个区间，选最少的点使得每个区间都有一个点。
解法：按左端点排序，前若干个区间 不断找区间[st,ed]中 ed 最小的 并且比开始的 起始点大，这些若干个区间能用 1 个点覆盖，以此类推。除排序外可以 O(n)实现。

计算几何

三角形内点数目

```
const double PI = acos(-1.);

struct cpoint{
	LL x,y;
	int id;
	double ang;
	cpoint(LL a=0,LL b=0)	{x=a,y=b;}

	cpoint operator- (const cpoint &u) const{
		return cpoint(x-u.x, y-u.y);
	}

	LL operator* (const cpoint &u) const{
		return x*u.y - y*u.x;
	}

	void read() {scanf("%I64d%I64d",&x,&y);}
}p[210],q[510],tmp[810];
int tot[210][210], r[210][210];
int T,n,m;

LL cross(cpoint o, cpoint p, cpoint q) { // 叉积
	return (p-o) * (q-o);
}

bool cmp(const cpoint& a,const cpoint& b){
	return a.ang < b.ang;
}
```

```
int cal(int a,int b,int c)
{
	if(tot[a][c] > tot[a][b])	return tot[a][c] - tot[a][b];
	else if(tot[a][c] < tot[a][b])	return m - (	tot[a][b] - tot[a][c] );
	else
	{
		double ang1 = atan2( (double)(p[b].y-p[a].y), (double)(p[b].x-p[a].x) );
		double ang2 = atan2( (double)(p[c].y-p[a].y), (double)(p[c].x-p[a].x) );
		if(ang2 > ang1) return tot[a][c] - tot[a][b];
		else	return m - (	tot[a][b] - tot[a][c] );
	}
}

int main()
{
	scanf("%d",&T);
	for(int cas = 1; cas <= T; cas++)
	{
		scanf("%d%d",&n,&m);

		for(int i=1; i<=n; i++) p[i].read();
		for(int i=1; i<=m; i++) q[i].read();
		clr(r,0), clr(tot,0);

		for(int i=1; i<=n; i++)
		{
			int cnt = 0;
			//算所有点相对于这个点的极角序
			for(int j=1; j<=n; j++)
			{
				if(i == j)	continue;
				tmp[++cnt].ang = atan2( (double)(p[j].y - p[i].y) , (double)(p[j].x - p[i].x) );
				tmp[cnt].id = j;
			}
			for(int j=1; j<=m; j++)
			{
				tmp[++cnt].ang = atan2( (double)(q[j].y - p[i].y) , (double)(q[j].x - p[i].x) );
				tmp[cnt].id = -1;
			}
			sort(tmp+1, tmp+1+cnt, cmp);

			//计算 tot 和 r
			int cc = 0;
			for(int j=1; j<=cnt; j++)
```

```
            {
                if(tmp[j].id == -1) cc++;
                else      tot[i][ tmp[j].id ] = cc;
            }

            int cur = 1, t = 0;
            for(int j=1; j<=cnt; j++)
            {
                if(tmp[j].ang > eps)      break;
                if(tmp[j].id == -1) continue;

                double lim = tmp[j].ang + PI;
                while(cur <= cnt && tmp[cur].ang < lim)
                {
                    if(tmp[cur].id == -1)     t++;
                    cur++;
                }

                r[i][tmp[j].id ] = m - (t - tot[i][tmp[j].id] );
                r[tmp[j].id ][i] = t - tot[i][tmp[j].id];
            }
        }

        double ans = 1E100;
        bool ok = false;

        //枚举三角形
        for(int i=1; i<=n; i++)
            for(int j=i+1; j<=n; j++)
                for(int k=j+1; k<=n; k++)
                {
                    int a = i, b = j, c = k;
                    if(cross(p[a], p[b], p[c]) < 0)
                        swap(b, c);

                    int v = r[a][b] + r[b][c] + r[c][a];
                    v += cal(a,b,c) + cal(b,c,a) + cal(c,a,b);
                    v -= 2*m;

                    if(v != 0)
                    {
                        ok = true;
                        double area = (fabs(((p[b].x - p[a].x)*(p[c].y-p[a].y) - (p[c].x-p[a].x)*(p[b].y-p[a].y))*0.5);
```

```
                        ans = min(ans, area/v );
                    }
                }
        printf("Case #%d: ",cas);
        if(!ok) printf("-1\n");
        else    printf("%.6f\n",ans);
    }
    return 0;
}
```

四面体外接圆心

```
double cal(double a1,double a2,double a3,double b1,double b2,double b3,double c1,double c2,double c3) {
    return a1*(b2*c3-b3*c2) - b1*(a2*c3-a3*c2) + c1*(a2*b3-a3*b2);
}

void center3(cpoint p0, cpoint p1, cpoint p2, cpoint p3,cpoint &cp) { //四面体外心
    double a1 = p1.x-p0.x, b1 = p1.y-p0.y, c1 = p1.z-p0.z, d1 = (a1*(p1.x+p0.x) + b1*(p1.y+p0.y) + c1*(p1.z+p0.z))/2;
    double a2 = p2.x-p0.x, b2 = p2.y-p0.y, c2 = p2.z-p0.z, d2 = (a2*(p2.x+p0.x) + b2*(p2.y+p0.y) + c2*(p2.z+p0.z))/2;
    double a3 = p3.x-p0.x, b3 = p3.y-p0.y, c3 = p3.z-p0.z, d3 = (a3*(p3.x+p0.x) + b3*(p3.y+p0.y) + c3*(p3.z+p0.z))/2;

    double q,q1,q2,q3;
    q = cal(a1,a2,a3,b1,b2,b3,c1,c2,c3);
    q1 = cal(d1,d2,d3,b1,b2,b3,c1,c2,c3);
    q2 = cal(a1,a2,a3,d1,d2,d3,c1,c2,c3);
    q3 = cal(a1,a2,a3,b1,b2,b3,d1,d2,d3);
    cp.x = q1/q, cp.y = q2/q, cp.z = q3/q;
}
```

两凸包最远距离

```
void anticlockwise(cpoint cp[], int n)
{
    for (int i=0; i <n-2; ++i)
    {
        double t = cross(cp[i], cp[i + 1], cp[i + 2]);
        if (dcmp(t) > 0) return ;
```

```
        if (dcmp(t) < 0)
        {
            reverse(cp, cp + n);
            return;
        }
    }
}

// 旋转卡壳，两凸包必须逆时针，并且需要把两凸包交换再做一遍
double rotating(cpoint ch1[], int n, cpoint ch2[], int m)
{
    int p = 0, q = 0; //p,q 分别找 yminP,ymaxQ  注意凸包给出的顺序
    for (int i = 0; i < n; ++i)
        if (dcmp(ch1[i].y - ch1[p].y) < 0)
            p = i;
    for (int i = 0; i < m; ++i)
        if (dcmp(ch2[i].y - ch2[q].y) > 0)
            q = i;

    ch1[n] = ch1[0],ch2[m] = ch2[0];
    double tmp, res = 1e99;
    for (int i = 0; i < n; ++i)
    {
        while ((tmp = cross(ch1[p], ch1[p + 1], ch2[q + 1]) -
                    cross(ch1[p], ch1[p + 1], ch2[q])) > eps)
            q = (q + 1) % m;
        if (dcmp(tmp) < 0)
            res = min(res, PointToSeg(ch2[q], ch1[p], ch1[p + 1]));
        else
            res = min(res,DisPallSeg(ch1[p],ch1[p+1],ch2[q],ch2[q+1]));
        p = (p + 1) % n;
    }
    return res;
}

double solve()
{
    //使凸包逆时针化  做过凸包后不必要
    anticlockwise(ch1, n);
    anticlockwise(ch2, m);
    return min(rotating(ch1, n, ch2, m), rotating(ch2, m, ch1, n));
}
```

最小矩形覆盖

```
double rotating(cpoint res[],int n){
    int j,l,r;  //j 为 i 对锺点 l r 为左右  点积求他们的距离  注意 dis2
    double ans=1E99;
    res[n] = res[0],j=1,l=1;
    for(int i=0; i<n; i++)
    {
        //默认凸包逆时针给出  否则+fabs
        while( cross(res[i],res[i+1],res[j+1]) - cross(res[i],res[i+1],res[j]) > eps )
            j=(j+1)%n;
        while( dot(res[i],res[i+1],res[l+1]) - dot(res[i],res[i+1],res[l]) > eps )
            l=(l+1)%n;
        if(i == 0)   r = j;
        while( dcmp(dot(res[i],res[i+1],res[r+1]) - dot(res[i],res[i+1],res[r]) ) <= 0 )
            r=(r+1)%n;

        double d = dis2(res[i],res[i+1]);
        ans = min(ans, cross(res[i], res[i + 1], res[j])
                    * ( dot(res[i], res[i + 1], res[l]) - dot(res[i], res[i + 1], res[r]) ) / d);
    }
    return ans;
}
```

最近点对

```
double solve(int l,int r)
{
    if(l == r)   return 1e10;
    int mid = (l + r)>>1;
    double d = min(solve(l,mid), solve(mid+1, r)), ret = d;
    double x = (p[mid].x + p[mid+1].x)/2.0;
    int st, ed, lo, hi;
    for(st = mid; st>=l && p[st].x + d + eps >= x; st--);st++;
    for(ed = mid+1; ed<=r && p[ed].x - d - eps <= x; ed++);ed--;

    sort(p+st, p+mid+1, cmpy);
    sort(p+mid+1, p+ed+1 , cmpy);
    lo = hi = mid+1;

    for(int i=st; i<=mid; i++)
    {
        while(lo < ed && p[lo].y + d + eps <= p[i].y)      lo++;
```

```
        while(hi < min(ed, lo+5) &&   p[hi].y - d - eps <= p[i].y)       hi++;
        for(int j=lo; j<=hi; j++) ret = min(ret, dis(p[i], p[j]));
    }
    return ret;
}
```

位运算

```
#define setbit(x,y) x|=(1<<(y))
#define clrbit(x,y) x&=~(1<<(y))

//u 的二进制 1 的个数
__builtin_popcount (unsigned u)

//生成下个相同 1 的数
int nextN(int n)
{
    int x = n&(-n);
    int t = n+x;
    return ((n^t)/x)>>2|t;
}

//1...1(m 位)0..0(n-m 位)
(1<<n)-(1<<(n-m))

//换位
int revbit(int x){
    x = ((x >> 1) & 0x55555555) | ((x << 1) & 0xaaaaaaaa);
    x = ((x >> 2) & 0x33333333) | ((x << 2) & 0xcccccccc);
    x = ((x >> 4) & 0x0f0f0f0f) | ((x << 4) & 0xf0f0f0f0);
    x = ((x >> 8) & 0x00ff00ff) | ((x << 8) & 0xff00ff00);
    x = ((x >>16) & 0x0000ffff) | ((x <<16) & 0xffff0000);
    return x;
}
```

外挂

多想重边 重点 自环 二分 三分 对数

```
#pragma comment(linker,"/STACK:65536000")
int readint()     //用于整数
{
```

```
        char c;
        while (c = getchar(), '-' != c && !isdigit(c))
                if(c == EOF)   return EOF;
        int f = 1;
        if ('-' == c)
                f = -1,c = getchar();
        int x = c - '0';
        while (isdigit(c = getchar()))
                x = x * 10 + c - '0';
        return x * f;
}
void write(int a) {    //用于正整数
    if(a>9) write(a/10);
    putchar(a%10+'0');
}
#include <cstdio>#include <ext/rope>
__gnu_cxx::crope s;
int t,cur,k,i;
char op[10],buf[1<<21],c;
int main()
{
        scanf("%d",&t);
        for(;t>0;t--){
                scanf("%s",op);
                switch(op[0]){
                        case 'M':
                                scanf("%d",&cur)；break;
                        case 'I':
                                scanf("%d",&k);i=0;
                                while(k){
                                        c=getc(stdin);
                                        if(c>=32&&c<=126){
                                                buf[i++]=c;
                                                k--;
                                        }
                                }
                                buf[i]=0;
                                s.insert(cur,buf);
                                break;
                        case 'D':scanf("%d",&k);s.erase(cur,k);break;
                        case 'G':scanf("%d",&k);puts(s.substr(cur,k).c_str());break;
                        case 'P':cur--;break;
                        case 'N':cur++;break;}}}..
```