

数学

基础

斐波那契

快速幂

n的阶乘

数论

素数筛法

素数判定

质因子分解

欧拉函数

欧拉降幂公式

莫比乌斯反演

欧几里得算法

类欧算法

乘法逆元

线性同余方程组

中国剩余定理

通解

矩阵

高斯消元

组合数学

Lucas定理

博弈

二分

0-1分数规划

三分

爬山

模拟退火

三分{三分{三分}}

Simpson积分

FFT & NTT

NTT中找原根

拉格朗日插值

杜教BM算法

附录

完美数

欧拉函数

莫比乌斯

杨辉三角

卡特兰数

原根

图论

前向星

最短路

Dijkstra

Dijkstra 堆优化

Bellman_Ford

Bellman_Ford队列优化

Floyd_Warshall

次短路

k短路

差分约束

生成树

最小生成树

Prim $O(V^2)$

Kruskal $O(E \log E)$

次小生成树 $O(V^2)$

最优比例生成树

最近公共祖先

树分治

二分图

二分图染色

二分图最大匹配

二分图多重匹配

二分图最佳匹配

二分图匹配方案

网络流

最大流

最小费用最大流

强联通分量

拓扑排序

欧拉路

字符串

字符串-数值转换

KMP

扩展KMP

Manacher

AC自动机

后缀数组(SA)

应用1: 最长公共前缀 (LCP)

应用2: 最长可重叠重复子串

应用3: 最长不可重叠重复子串

应用4: 不相同的子串的个数

应用5: 连续重复子串

应用6: 最长连续重复子串★

后缀自动机(SAM)

应用1: 不相同的子串的个数

应用2: endpos集合

应用3: 最长公共子串

应用4: 多个串的最长公共子串

应用5: 不同长度下出现次数最多的子串

应用6: 原串S中出现次数在[A,B]之间的子串的个数

应用7: 区间内不同子串的个数

动态规划

背包

最长公共子序列 LCS

最长上升子序列 LIS

最长递增公共子序列 LICS

整数拆分

区间

状压

数位

树形

重心

基环树

多线程

趣题

数据结构

排序

离散化

并查集

RMQ-ST表

分块

莫队

字典树

笛卡尔树

树状数组

区间修改 区间求和

单点修改 块求和

树状数组二分

线段树

1. push

2. build

3. update

4. query

5. 常用操作

扫描线

周长并

面积并

面积交

不相交图形个数

替罪羊树

树堆

1. push

2. rotate

bzoj3224

伸展树

1. push

2. build

3. splay

4. 单点操作

5. 区间操作

6. debug

bzoj3224

bzoj1500

主席树

持久化并查集

树套数

主席树套字典树

线段树套伸展树

珂朵莉树

DFS序

树链剖分

LG3384

POJ3237

动态树

数学

基础

$$\sum_i^n i^2 = \frac{n*(n+1)*(2*n+1)}{6}$$
$$\sum_i^n i^3 = \frac{n^2*(n+1)^2}{4}$$
$$\sum_i^n i^4 = \frac{n*(n+1)*(2n+1)*(3n^2+3n-1)}{30}$$
$$\sum_i^n i^5 = \frac{n^2*(n+1)^2*(2n^2+2n-1)}{12}$$
$$\sum_i^n (2i-1)^2 = \frac{n*(4n^2-1)}{3}$$
$$\sum_i^n (2i-1)^3 = n^2*(2n^2-1)$$
$$\sum_{i=1}^n i(i+1) = \frac{n(n+1)(n+2)}{3}$$
$$\sum_{i=1}^n i(i+1)(i+2) = \frac{n(n+1)(n+2)(n+3)}{4}$$
$$\sum_{i=1}^n i(i+1)(i+2)(i+3) = \frac{n(n+1)(n+2)(n+3)(n+4)}{5}$$
$$\tan(A+B) = \frac{\tan A + \tan B}{1 - \tan A * \tan B}$$
$$\cot(A+B) = \frac{\cot A * \cot B - 1}{\cot A + \cot B}$$

斐波那契

可用矩阵快速幂优化

$f_0 = 0, f_1 = f_2 = 1$	$f_n = f_{n-1} + f_{n-2} (n \geq 2)$
$f_1 + f_2 + \cdots + f_n = f_{n+2} - 1$	加个 f_2 推导, 最后减去
$f_1^2 + f_2^2 + \cdots + f_n^2 = f_n * f_{n+1}$	矩形面积
$f_1 + f_3 + \cdots + f_{2*n-1} = f_{2*n}$	把 f_1 当 f_2 推导
$f_2 + f_4 + \cdots + f_{2*n} = f_{2*n+1} - 1$	加个 f_1 推导
$f_{n-1} + f_{n+1} = \frac{f_{2*n}}{f_n}$	
$f_{n-1} * f_{n+1} = f_n^2 - 1^n$	

快速幂

```
inline ll mul_mod(ll a,ll b,ll m) { //a*b%m
    ll res=0;
    while (b) {
        if (b&1) res=(res+a)%m;
        a=(a+a)%m;
        b>>=1;
    }
    return res;
}
```

```
inline ll pow_mod(ll a,ll b,ll m) { //a^b%m
    ll res=1;
    while (b) {
        if (b&1) res=res*a%m;
        a=a*a%m;
        b>>=1;
    }
    return res;
}
```

n的阶乘

stirling公式

1. $n! \approx \sqrt{2 * n * \pi} * (\frac{n}{e})^n$
2. n!的位数 $\log_{10} n + 1$

```
inline int stirling(int n) {
    return (log(2*PI*n)/2 + n*log(n) - n)/log(10)+1;
}
```

```
tgamma(n) = (n-1)!
lgamma(n) = ln((n-1)!)
exp(n) = e^n
tgamma(5) = exp(lgamma(5)) = 24
```

阶乘打表求组合数

```
int fac[maxn],inv[maxn];
void init(int n) {
    fac[0]=1;
    for (int i=1;i<=n;++i) fac[i]=1ll*fac[i-1]*i%mod;
    inv[n]=pow_mod(fac[n],mod-2,mod);
    for (int i=n-1;i>=0;--i) inv[i]=1ll*inv[i+1]*(i+1)%mod;
}
```

数论

素数筛法

```
int nump, prime[maxn]; bool nprime[maxn];
void get_prime(int n) {
    nprime[0]=nprime[1]=1;
    for (int i=2; i<=n; ++i) {
        if (!nprime[i]) prime[nump++]=i;
        for (int j=0; j<nump&& i*prime[j]<=n; ++j) {
            nprime[i*prime[j]]=1;
            if (i%prime[j]==0) break;
        }
    }
}
```

素数判定

1. $p = m - 1 = 2^r * s$
2. 若 $a^s \% m = 1$ 通过检测, 否则必有 $a^{2^r * s} \% m = n - 1$ 才能通过检测

```
bool miller_rabbin(ll a, ll m) {
    ll r=0, s=m-1;
    while (~s&1) { r++; s>>=1; }
    ll tmp=pow_mod(a, s, m);
    if (tmp==1) return true; //a^s%n==1通过素数检测
    for (int i=0; i<r; ++i, tmp=mul_mod(tmp, tmp, m))
        if (tmp==m-1) //a^s^2^r%n==n-1通过检测
            return true;
    return false;
}

bool isprime(ll n) {
    if (n<2) return false;
    int prime[]={2,3,5,7,11}, nump=5;
    for (int i=0; i<nump; ++i) {
        if (prime[i]==n) return true;
        if (n%prime[i]==0) return false;
    }
    for (int i=0; i<nump; ++i)
        if (!Miller_Rabbin(prime[i], n))
            return false;
    return true;
}
```

质因子分解

$$p = p_1^{a_1} * p_2^{a_2} * p_3^{a_3} * \dots * p_k^{a_k}$$

$$\text{因子个数: } \tau(n) = (a_1 + 1)(a_2 + 1)(a_3 + 1) \dots (a_k + 1) = \prod_{i=1}^k (a_i + 1)$$

$$\text{因子和: } \sigma(n) = \frac{p_1^{a_1+1} - 1}{p_1 - 1} \frac{p_2^{a_2+1} - 1}{p_2 - 1} \frac{p_3^{a_3+1} - 1}{p_3 - 1} \dots \frac{p_k^{a_k+1} - 1}{p_k - 1} = \prod_{i=1}^k \frac{p_i^{a_i+1} - 1}{p_i - 1}$$

```
int fac[66], numf;
void get_fac(ll x) {
    numf=0;
    for (int i=1; prime[i]*prime[i]<=x; i++)
        if (x%prime[i]==0) {
            fac[numf++] = prime[i];
            while (x%prime[i]==0) x/=prime[i];
        }
    if (x>1) fac[numf++] = x;
}
```

欧拉函数

欧拉函数是小于n的正整数中与n互质的数的数量, $\phi(1) = 1$

$$\phi(n) = n * (1 - \frac{1}{p_1}) * (1 - \frac{1}{p_2}) * \dots * (1 - \frac{1}{p_r})$$

$$\text{欧拉定理: } a^b = \begin{cases} a^{b \bmod \phi(p)} & \gcd(a, p) = 1 \\ a^b & \gcd(a, p) \neq 1, b < \phi(p) \\ a^{b \bmod \phi(p) + \phi(p)} & \gcd(a, p) \neq 1, b \geq \phi(p) \end{cases}$$

```
ll phi(ll n) {
    ll res=n;
    for (int i=2; i*i<=n; i++)
        if (n%i==0) {
            res-=res/i;
            while (n%i==0) n/=i;
        }
    if (n>1) res-=res/n;
    return res;
}
```

```
int phi[maxn];
void get_phi(int n) {
    phi[1]=1;
    for (int i=2; i<=n; i++) if(!phi[i])
        for (int j=i; j<=n; j+=i) {
            if (!phi[j]) phi[j]=j;
            phi[j]-=phi[j]/i;
        }
}
```

欧拉降幂公式

```
#define Mod(a,b)    a>=b ?(a%b+b):a
map<ll,ll> mp;

ll a[maxn];
inline ll phi(ll x) {
    ll ans=x,tmp=x;
    if(mp.count(x)) return mp[x];
    for(int i=2;i*i<=x;i++) {
        if(x%i==0) {
            ans=ans/i*(i-1);
            while(x%i==0) x/=i;
        }
    }
    if(x>1) ans=ans/x*(x-1);
    mp[tmp]=ans;
    return ans;
}
inline ll pow_mod(ll a,ll b,ll m) {
    ll res=1;
    while (b) {
        if(b&1) res = Mod(res*a,m); /// !!!
        a = Mod(a*a,m); /// !!!
        b>>=1;
    }
    return res;
}
ll solve(ll l,ll r,ll m) {
    if(l==r||m==1) return Mod(a[l],m);
    return qpow(a[l],solve(l+1,r,phi(m)),m);
}
int main() {
    ll n,m,t;
    scanf("%lld %lld",&n,&m);
    for (int i=1;i<=n;i++) scanf("%lld",&a[i]);
    scanf("%lld",&t);
    while (t--) {
        int l,r;
        scanf("%d %d",&l,&r);
        printf("%lld\n",solve(l,r,m)%m);
    }
    return 0;
}
```

莫比乌斯反演

莫比乌斯函数:
$$\mu(d) = \begin{cases} 1 & d = 1 \\ (-1)^k & p_1 p_2 \cdots p_r, \text{其中 } p_i \text{ 为不同的素数} \\ 0 & \text{其它} \end{cases}$$

性质:

1. $\sum_{d|n} \mu(d) = \begin{cases} 1 & n = 1 \\ 0 & n > 1 \end{cases}$
2. $F(n) = \sum_{d|n} f(d) \Rightarrow f(n) = \sum_{d|n} \mu(d) F(\frac{n}{d})$
3. $F(n) = \sum_{n|d} f(d) \Rightarrow f(n) = \sum_{n|d} \mu(\frac{d}{n}) F(d)$
4. $f(k) = \sum_{i=1}^n \sum_{j=1}^m [\gcd(i, j) = k] = \sum_{i=1}^{\frac{n}{k}} \sum_{j=1}^{\frac{m}{k}} \sum_{d|\gcd(i, j)} \mu(d) = \sum_{d=1}^n \mu(d) \frac{n}{d*k} \frac{m}{d*k}$
5. $\sum_{d|n} \frac{\mu(d)}{d} = \frac{\phi(n)}{n}$
6. $\sum_{d|n} \mu(\frac{n}{d}) \sigma(d) = n$
7. $\sum_{d|n} \mu(\frac{n}{d}) \tau(d) = 1$

```

int n,m;
int nump,prime[maxn],mu[maxn],sum[maxn];
void get_mu(int n) {
    mu[1]=1;
    for (int i=2;i<=n;++i) {
        if (!prime[i]) prime[++nump]=i,mu[i]=-1;
        for (int j=1;j<=nump&& i*prime[j]<=n;++j) {
            prime[i*prime[j]]=1;
            if (i%prime[j]==0) break;
            else mu[i*prime[j]]=-mu[i];
        }
    }
    for (int i=1;i<=n;++i)
        sum[i]=sum[i-1]+mu[i];
}
ll calc(int n,int m,int d) { //求gcd(i,j)=d的个数
    ll res=0; n/=d; m/=d;
    for (int l=1,r;l<=n;l=r+1) {
        r=min(n/(n/l),m/(m/l));
        res+=1ll*(n/l)*(m/l)*(sum[r]-sum[l-1]);
    }
    return res;
}

```

洛谷P2257

给定N,M,求 $1 \leq x \leq N, 1 \leq y \leq M$ 且 $\gcd(x,y)$ 为质数的(x,y)有多少对

设 $f(d)$ 为 $\gcd(i, j) = d$ 的个数, $F(n)$ 为 $\gcd(i, j) = k * d$ 的个数

$$f(d) = \sum_{i=1}^N \sum_{j=1}^M [\gcd(i, j) = d]$$

$$F(n) = \sum_{n|d}^{d \leq \min(N, M)} f(d) = \left\lfloor \frac{N}{n} \right\rfloor \left\lfloor \frac{M}{n} \right\rfloor$$

$$\text{反演公式: } f(n) = \sum_{n|d} \mu\left(\frac{d}{n}\right) F(d)$$

$$Ans = \sum_{p \in \text{prime}} f(p) = \sum_p \sum_{p|d} \mu\left(\frac{d}{p}\right) F(d)$$

每个素数都会对其倍数有贡献, 所以可以求一个数的素因子的贡献

$$\text{让 } T = d: Ans = \sum_{T=1}^{\min(n, m)} F(T) \sum_{p|T} \mu\left(\frac{T}{p}\right)$$

```
get_mu(maxn-1);
for (int i=1; i<=num; ++i)
    for (int j=1; prime[i]*j<maxn; j++)
        g[prime[i]*j] += mu[j];
for (int i=1; i<maxn; i++) sum[i] = sum[i-1] + g[i];

int T;
scanf("%d", &T);
while (T--) {
    scanf("%d%d", &n, &m);
    if (n>m) swap(n, m); ans=0;
    for (int l=1, r; l<=n; l=r+1) { //分块优化
        r = min(n/(n/l), m/(m/l));
        ans += 1ll * (n/l) * (m/l) * (sum[r] - sum[l-1]);
    }
    printf("%lld", ans);
}
```

欧几里得算法

1. 欧几里得算法 $O(\log \max(a, b))$
2. 扩展欧几里得算法 $O(\log \max(a, b))$

已知 a, b, n , 求解方程 $ax + by = n$

令 $d = \gcd(a, b)$, 若 $d \mid n$ 原方程有解

$$ax + by = d \Rightarrow ax_0 + by_0 = d$$

$$ax_0 * \frac{n}{d} + by_0 * \frac{n}{d} = d * \frac{n}{d} \Rightarrow ax_1 + by_1 = n \text{ (一组解)}$$

$$x = x_1 + k * \frac{b}{d} \text{ 等同于增加 } \frac{k*a*b}{d}$$

$$y = y_1 - k * \frac{a}{d} \text{ 等同于减少 } \frac{k*a*b}{d}$$

```

11 gcd(11 a,11 b) {
    return b?gcd(b,a%b):a;
}
void exgcd(11 a,11 b,11 &d,11 &x,11 &y) {
    if (!b) d=a,x=1,y=0;
    else exgcd(b,a%b,d,y,x), y-=a/b*x;
}

```

```

static BigInteger d,x,y;
static void exgcd(BigInteger a,BigInteger b) {
    if (b.compareTo(ZERO)==0) { d = a; x = ONE; y = ZERO; }
    else {
        exgcd(b,a.mod(b));
        BigInteger tmp=x; x=y; y=tmp.subtract(a.divide(b).multiply(x));
    }
}

```

类欧算法

$$f(a,b,c,n) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor$$

- 当 $a \geq c$ 或 $b \geq c$

$$f(a,b,c,n) = f(a \% c, b \% c, c, n) + \frac{n(n+1)}{2} \lfloor \frac{a}{c} \rfloor + (n+1) \lfloor \frac{b}{c} \rfloor$$

- 当 $a < c$ 且 $b < c$

$$\text{令 } m = \lfloor \frac{an+b}{c} \rfloor$$

$$f(a,b,c,n) = \sum_{i=0}^n \sum_{j=1}^m [\lfloor \frac{ai+b}{c} \rfloor \geq j] \quad (\text{可以理解为数 } x \text{ 在 } [1,x] \text{ 每个位置统计一次})$$

$$f(a,b,c,n) = \sum_{i=0}^n \sum_{j=0}^{m-1} [\lfloor \frac{ai+b}{c} \rfloor \geq j+1]$$

$$f(a,b,c,n) = \sum_{i=0}^n \sum_{j=0}^{m-1} [ai \geq jc + c - b]$$

$$f(a,b,c,n) = \sum_{i=0}^n \sum_{j=0}^{m-1} [ai > jc + c - b - 1]$$

$$f(a,b,c,n) = \sum_{i=0}^n \sum_{j=0}^{m-1} [i > \frac{jc+c-b-1}{a}]$$

$$f(a,b,c,n) = \sum_{j=0}^{m-1} [n - \lfloor \frac{jc+c-b-1}{a} \rfloor]$$

$$f(a,b,c,n) = nm - f(c, c-b-1, a, m-1)$$

```

11 f(11 a,11 b,11 c,11 n) {
    if (!a) return (n+1)*(b/c);
    if (a>=c || b>=c) return f(a%c,b%c,c,n)+n*(n+1)/2*(a/c)+(n+1)*(b/c);
    11 m=(a*n+b)/c;
    return n*m-f(c,c-b-1,a,m-1);
}

```

$$g(a,b,c,n) = \sum_{i=0}^n i \lfloor \frac{ai+b}{c} \rfloor$$

- $a \geq c$ 或 $b \geq c$

$$g(a,b,c,n) = g(a \% c, b \% c, c, n) + \frac{n(n+1)(2n+1)}{6} \lfloor \frac{a}{c} \rfloor + \frac{n(n+1)}{2} \lfloor \frac{b}{c} \rfloor$$

- $a < c$ 且 $b < c$

$$\text{令 } m = \lfloor \frac{an+b}{c} \rfloor$$

$$g(a, b, c, n) = \sum_{i=0}^n i \sum_{j=1}^m [\lfloor \frac{ai+b}{c} \rfloor \geq j] \text{ (可以理解为数 } x \text{ 在 } [1, x] \text{ 每个位置统计一次)}$$

$$g(a, b, c, n) = \sum_{i=0}^n i \sum_{j=0}^{m-1} [i > \frac{jc+c-b-1}{a}]$$

$$g(a, b, c, n) = \sum_{j=0}^{m-1} \sum_{i=0}^n i [i > \frac{jc+c-b-1}{a}] \text{ (表达式为真 } i \text{ 至少为等式 } \lfloor \frac{jc+c-b-1}{a} \rfloor + 1)$$

$$g(a, b, c, n) = \sum_{j=0}^{m-1} \frac{1}{2} (\lfloor \frac{jc+c-b-1}{a} \rfloor + 1 + n) (n - \lfloor \frac{jc+c-b-1}{a} \rfloor)$$

$$g(a, b, c, n) = \frac{1}{2} \sum_{j=0}^{m-1} [n * (n+1) - \lfloor \frac{jc+c-b-1}{a} \rfloor - \lfloor \frac{jc+c-b-1}{a} \rfloor^2]$$

$$g(a, b, c, n) = \frac{1}{2} [nm(n+1) - f(c, c-b-1, a, m-1) - h(c, c-b-1, a, m-1)]$$

```

11 g(11 a, 11 b, 11 c, 11 n) {
    if (!a) return n*(n+1)/2*(b/c);
    if (a>=c || b>=c)
        return g(a%c, b%c, c, n) + n*(n+1)*(2*n+1)/6*(a/c) + n*(n+1)/2*(b/c);
    11 m=(a*n+b)/c;
    return (n*m*(n+1)-f(c, c-b-1, a, m-1)-h(c, c-b-1, a, m-1))/2;
}

```

$$h(a, b, c, n) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor^2$$

- $a \geq c$ 或 $b \geq c$

$$h(a, b, c, n) = h(a\%c, b\%c, c, n) + \frac{n(n+1)(2n+1)}{6} \lfloor \frac{a}{c} \rfloor^2 + (n+1) \lfloor \frac{b}{c} \rfloor^2 + 2 \lfloor \frac{b}{c} \rfloor f(a\%c, b\%c, c, n) + 2 \lfloor \frac{a}{c} \rfloor g(a\%c, b\%c, c, n) + 2 \lfloor \frac{a}{c} \rfloor \lfloor \frac{a}{c} \rfloor n(n+1)$$

- $a < c$ 且 $b < c$

$$n^2 = 2 * \frac{n(n+1)}{2} - n = 2 \sum_{i=0}^n i - n$$

$$h(a, b, c, n) = \sum_{i=0}^n 2 \sum_{j=1}^{\frac{ai+b}{c}} j - \frac{ai+b}{c}$$

$$\text{令 } m = \lfloor \frac{an+b}{c} \rfloor$$

$$h(a, b, c, n) = 2 \sum_{i=0}^n \sum_{j=1}^m j - \sum_{i=0}^n \frac{ai+b}{c}$$

$$h(a, b, c, n) = 2 \sum_{j=1}^m j \sum_{i=0}^n [\frac{ai+b}{c} \geq j] - f(a, b, c, n)$$

$$h(a, b, c, n) = 2 \sum_{j=0}^{m-1} (j+1) \sum_{i=0}^n [\frac{ai+b}{c} \geq j+1] - f(a, b, c, n)$$

$$h(a, b, c, n) = 2 \sum_{j=0}^{m-1} (j+1) \sum_{i=0}^n [i > \frac{jc+c-b-1}{a}] - f(a, b, c, n)$$

$$h(a, b, c, n) = 2 \sum_{j=0}^{m-1} (j+1) [n - \lfloor \frac{jc+c-b-1}{a} \rfloor] - f(a, b, c, n)$$

$$h(a, b, c, n) = 2 \sum_{j=0}^{m-1} [(j+1)n - j \lfloor \frac{jc+c-b-1}{a} \rfloor - \lfloor \frac{jc+c-b-1}{a} \rfloor] - f(a, b, c, n)$$

$$h(a, b, c, n) = nm(m+1) - 2g(c, c-b-1, a, m-1)$$

$$- 2f(c, c-b-1, a, m-1) - f(a, b, c, n)$$

```

11 h(11 a,11 b,11 c,11 d) {
    if (!a) return (n+1)*(b/c)*(b/c);
    if (a>=c || b>=c) return h(a%c,b%c,c,n)+n*(n+1)*(2*n+1)/6*(a/c)*(a/c)
        +(n+1)*(b/c)*(b/c)+2*(b/c)*f(a%c,b%c,c,n)
        +2*(a/c)*g(a%c,b%c,c,n)+(a/c)*(b/c)*n*(n+1);
    11 m=(a*n+b)/c;
    return n*m*(m+1)-2*g(c,c-b-1,a,m-1)-2*f(c,c-b-1,a,m-1)-f(a,b,c,n);
}

```

乘法逆元

对于正整数 a, m , 若 a, m 互质, $ax \equiv 1 \pmod{m}$

x 的最小正整数解叫做 a 模 m 的逆元

$$ax \equiv 1 \pmod{m} \Rightarrow x \equiv \frac{1}{a} \pmod{m}$$

如果 a, m 互质, $a^{\phi(m)-1} \equiv 1 \pmod{m}$

```

11 inv(11 a,11 m) { //不存在逆, 返回-1
    11 d,x,y;
    exgcd(a,m,d,x,y);
    return d==1?(x+m)%m:-1;
}

```

线性逆元表

求 i 关于 p 的逆元

令 $a * i + b = p$ (正序递推, $b < i$, $\text{inv}[b]$ 已知)

$$b * \text{inv}[b] \equiv 1 \pmod{p}$$

$$(p - a * i) * \text{inv}[b] \equiv 1 \pmod{p}$$

$$-a * i * \text{inv}[b] \equiv 1 \pmod{p}$$

$$-a * \text{inv}[b] \equiv \text{inv}[i] \pmod{p}$$

$$-p/i * \text{inv}[p\%i] \equiv \text{inv}[i] \pmod{p}$$

$$(p - p/i) * \text{inv}[p\%i] \equiv \text{inv}[i] \pmod{p} \text{ 加 } p \text{ 防止出现复数}$$

```

11 inv[maxn];
inline void get_inv() { //M 模数
    inv[1]=1;
    for (int i=2;i<maxn;++i)
        inv[i]=(mod-mod/i)*inv[mod%i]%mod;
}

```

线性同余方程组

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \vdots \\ x \equiv a_n \pmod{m_n} \end{cases}$$

中国剩余定理

若 m_1, m_2, \dots, m_n 两两互质

$M = m_1 * m_2 * \dots * m_n$, $M_i = \frac{M}{m_i}$, M_i 没有 m_i 这个因子

$M_i * P_i \equiv 1 \pmod{m_i}$

$a_1 M_1 P_1 + a_2 M_2 P_2 + \dots + a_n M_n P_n$ 是一个可行解

```
11 china(int n) {
    11 res=0,M=1,Mi,d,x,y;
    for (int i=1;i<=n;++i) M*=m[i];
    for (int i=1;i<=n;++i) {
        Mi=M/m[i];
        exgcd(Mi,m[i],d,x,y);
        res+=a[i]*Mi*x;
    }
    return (res%M+M)%M;
}
```

通解

令 $m = lcm[m_1, m_2]$, $d = gcd(m_1, m_2)$, X 为终解

$$\begin{cases} x \equiv b_1 \pmod{m_1} \\ x \equiv b_2 \pmod{m_2} \end{cases} \Rightarrow \begin{cases} X \equiv b_1 \pmod{m_1} \\ X \equiv b_2 \pmod{m_2} \end{cases} \Rightarrow \begin{cases} m_1 \mid (X - b_1) \\ m_2 \mid (X - b_2) \end{cases} \Rightarrow m \mid (X - b_1) \text{ (等同于满足前面两个)}$$

$$\begin{cases} x - m_1 * y_1 = b_1 \\ x - m_2 * y_2 = b_2 \end{cases} \Rightarrow m_1 * y_1 \equiv (b_2 - b_1) \pmod{m_2} \text{ 用扩展欧几里得算法求 } y_1$$

最小非负整数解 $(y_1 \% \frac{m_2}{d} + \frac{m_2}{d}) \% \frac{m_2}{d}$, 方程 $x = m_1 * y_1 + b_1$

$X = x \pmod{m}$

```
inline 11 min_ans(11 x,11 m) { return (x%m+m)%m; }
11 solve(int n,11 m[],11 b[]) {
    11 mm=m[1],bb=b[1],d,x,y;
    bool flag=true;
    for (int i=2;i<=n;++i) {
        exgcd(mm,m[i],d,x,y);
        if ((b[i]-bb)%d) flag=false;
        x*=(b[i]-bb)/d;
        x=min_ans(x,m[i]/d);
        bb+=mm*x;
        mm*=m[i]/d;
    }
    return flag?bb:-1;
}
```

矩阵

```
const int sz=2;
struct mat {
    ll a[sz+5][sz+5];
    mat() { memset(a,0,sizeof a); }
    mat operator*(const mat &b) const {
        mat res;
        for (int i=1;i<=sz;++i)
            for (int j=1;j<=sz;++j)
                for (int k=1;k<=sz;++k)
                    res.a[i][j]=(res.a[i][j]+a[i][k]*b.a[k][j])%mod;
        return res;
    }
};

mat mat_pow(mat &a,ll p) {
    mat res;
    for (int i=1;i<=sz;++i) res.a[i][i]=1;
    while (p) {
        if (p&1) res=res*a;
        a=a*a;
        p>>=1;
    }
    return res;
}
```

```
import java.util.Scanner;
import java.math.BigInteger;
import static java.math.BigInteger.*;

public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        BigInteger res[] = new BigInteger[30];
        BigInteger a[][] = { {ONE,ONE},{ONE,ZERO} };
    }

    static BigInteger[][] matMul(BigInteger a[][], BigInteger b[][], int n) {
        BigInteger res[][] = new BigInteger[n][n];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) {
                res[i][j] = ZERO;
                for (int k = 0; k < n; k++)
                    res[i][j] = res[i][j].add(a[i][k].multiply(b[k][j]));
            }
        return res;
    }

    static BigInteger[][] matPow(BigInteger a[][], BigInteger p, int n) {
        BigInteger res[][] = new BigInteger[n][n];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (i == j) res[i][j] = ONE;
    }
}
```

```

        else res[i][j] = ZERO;
    while (p.compareTo(ZERO) != 0) {
        if (p.and(ONE).compareTo(ONE) == 0) res = matMul(res, a, n);
        a = matMul(a, a, n);
        p = p.divide(BigInteger.valueOf(2));
    }
    return res;
}
}

```

矩阵和

$$\begin{bmatrix} A & 0 \\ I & I \end{bmatrix} * \begin{bmatrix} A^{k-1} \\ S_{k-2} \end{bmatrix} = \begin{bmatrix} A^k \\ S_{k-1} \end{bmatrix}$$

$$S_k = I + A^1 + A^2 + \dots + A^k$$

高斯消元

```

void gauss(double a[][maxn],int n) { //n行n+1列
    for (int i=1;i<=n;++i) {
        //交换两行, 保证a[i][i]非零和高精度
        int r=i;
        for (int j=i+1;j<=n;++j)
            if (fabs(a[j][i])>fabs(a[r][i]))
                r=j;
        if (r!=i) for (int k=1;k<=n+1;++k) swap(a[r][k],a[i][k]);
        //消元
        for (int j=i+1;j<=n;++j) {
            double f=a[j][i]/a[i][i];
            for (int k=i;k<=n+1;++k)
                a[j][k]-=f*a[i][k];
        }
    }
    //回代
    for (int i=n;i;i--) {
        for (int j=i+1;j<=n;j++)
            a[i][n+1] -= a[i][j]*a[j][n+1];
        a[i][n+1] /= a[i][i];
    }
}

```

组合数学

n选m的两种表达方式 $\binom{n}{m}$ 和 C_n^m

$$1. A_n^r = \frac{n!}{(n-r)!}, C_n^r = \frac{n!}{(n-r)!*r!}$$

$$2. C_n^r = C_{n-1}^r + C_{n-1}^{r-1}$$

$$3. C_{n+r+1}^r = C_{n+r}^r + C_{n+r-1}^{r-1} + \dots + C_n^0 = C_{n+r}^n + C_{n+r-1}^n + \dots + C_n^n$$

$$4. C_n^k * C_k^r = C_n^r * C_{n-r}^{k-r} (k \geq r)$$

5. Catalan数 $h_0 = h_1 = 1$

$$h_n = h_0 * h_{n-1} + h_1 * h_{n-2} + \cdots + h_{n-1} * h_0 = \frac{4*n-2}{n+1} h_{n-1} = \frac{C_{2*n}^n}{n+1} = C_{2*n}^n - C_{2*n}^{n-1}$$

```
11 com(int n,int r) {
    if (r>n||r<0) return 0;
    11 res=1;
    if (r>n-r) r=n-r;
    for (int i=n,j=1;j<=r;--i,++j)
        res=res*i/j;
    return res;
}
```

Catalan数打表 pyhon

```
n = 105
h = [1,1]
for i in range(2,n+1):
    h.append((4*i-2)*h[i-1]/(i+1))
print ("string str[%d]={"%n)
for i in h:
    print("\ "" + '%d'%i + "\",")
print ("}")
```

Lucas定理

$C_n^m \% p$, p 是质数

博弈

1. 巴什博弈

一堆 n 个物品,两个人轮流从这堆物品中取物,每次至少取一个,最多取 m 个.

```
if (n % (m+1) != 0) //拿走最后一个先手胜
if (n % (m+1) != 1) //拿走最后一个先手输
```

2. 威佐夫博弈

有两堆物品，两个人轮流从某一堆或同时从两堆中取同样多的物品，每次至少取一个

$bn=an+n$ 是奇异局势 (自己取时必输)

0	1	3	4	6	8	9	11	12
0	2	5	7	10	13	15	18	20

根据Betty定理

设 $an=[\alpha n]$, $bn=[\beta n]$, 有 $an+n = [(\alpha+1)n] = [\beta n]$

解方程 $1/\alpha + 1/(\alpha+1) = 1$, 解得 $\alpha = (\sqrt{5}+1)/2$

```

if (a > b) a^=b^=a^=b;
int c = (sqrt(5)+1) * (b-a) / 2;
if (a != c) //bn - an = n会输

```

3. 尼姆博弈

有三堆物品，两个人轮流从某一堆取任意多的物品，规定每次至少取一个。

(a,b,c)表示某种局势,则 $a \oplus b \oplus c = 0$ 是奇异局势

可扩展为n堆

4. 斐波那契博弈

有一堆物品,两人轮流取物品,先手最少取一个,至多无上限，但不能把物品取完

之后每次取的物品数不能超过上一次取的物品数的二倍且至少为一件

取走最后一件物品的人获胜

n不是斐波那契数先手胜

5. SG函数

```

//f[N]:可改变当前状态的方式, N为方式的种类, f[N]要在getSG之前先预处理
//SG[]: 0~n的SG函数值
//S[]: 为x后继状态的集合
int f[N], SG[MAXN], S[MAXN];
void getSG(int n){
    int i, j;
    memset(SG, 0, sizeof(SG));
    //因为SG[0]始终等于0, 所以i从1开始
    for(i = 1; i <= n; i++){
        //每一次都要将上一状态 的 后继集合 重置
        memset(S, 0, sizeof(S));
        for(j = 0; f[j] <= i && j <= N; j++){
            S[SG[i-f[j]]] = 1; //将后继状态的SG函数值进行标记
        }
        for(j = 0; j <= SG[i]; j++){ //查询当前后继状态SG值中最小的非零值
            if(!S[j]) break;
        }
        SG[i] = j;
    }
}

```

二分

符合某条件让某个值更大,初始让L符合条件, R不符合条件。

```

bool judge(int x) {}
int l=0,r=1e5;
while (r-l>1) {
    int m=(l+r)>>1;
    if (judge(m)) l=m;
    else r=m;
}
printf("%d\n",l);

```

0-1分数规划

$$R = \frac{a_1 * x_1 + a_2 * x_2 + \cdots + a_n * x_n}{b_1 * x_1 + b_2 * x_2 + \cdots + b_n * x_n}$$

a_i 表示选取 i 的收益, b_i 表示选取 i 的代价, x_i 代表是否选取(0 or 1)

从 n 个里面选 k 个使 R 最大

先设 $F_R = \sum (a_i - R * b_i) * x_i$, $F_R \geq 0$ 说明有比 R 大的解

```

int n,k;
int a[maxn],b[maxn];
double d[maxn];
bool judge(double R) {
    for (int i=1;i<=n;++i)
        d[i]=a[i]-R*b[i];
    sort(d+1,d+1+n); //从小到大排序,再反向选取
    double res=0;
    for (int i=n-k+1;i<=n;++i) res+=d[i];
    return res>=0;
}
double slove() {
    double l=0,r=1;
    while(r-l>eps) {
        double m=(l+r)/2;
        if (judge(m)) l=m;
        else r=m;
    }
    return r;
}

```

三分

单峰函数求极值 $f(x) = \max \{a_i * x^2 + b_i * x + c\} (a_i \geq 0)$

```

double f(double x) {
    double res=a[1]*x*x+b[1]*x+c[1];
    for (int i=2;i<=n;++i)
        res=max(res,a[i]*x*x+b[i]*x+c[i]);
    return res;
}

```

```
double l=0,r=1000;
while (r-l>eps) {
    double m1=(l+l+r)/3;
    double m2=(l+r+r)/3;
    if (f(m1)<f(m2)) r=m2;
    else l=m1;
}
printf("%.6f\n",f(l));
```

爬山

缺点：局部最优，凸函数不影响

解决方法：随机重启

```
//求到n个点的最大距离最小化的点
int n;
struct point{ double x,y,z; } p[maxn],pp;
double dis(point a,point b){
    return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y)+(a.z-b.z)*(a.z-b.z);
}
int get(point x){
    int k=0; double m=-1;
    for (int i=1;i<=n;++i) {
        double tmp=dis(x,p[i]);
        if (tmp>m) { m=tmp; k=i; }
    }
    return k;
}
void hc() {
    double T=1,eps=1e-11;
    while (T>eps) {
        int k=get(pp);
        pp.x+=(p[k].x-pp.x)*T;
        pp.y+=(p[k].y-pp.y)*T;
        pp.z+=(p[k].z-pp.z)*T;
        T*=0.999;
    }
}
```

模拟退火

多峰函数求极值

1. 跳跃总范围100T,设置合适的T
2. 可以多起点开始

```
//T初始温度,delta变温速率,eps终止条件,注意时间复杂度
//求矩形内到n个点的最短距离最大化的点
int n,x,y;
double ans;
```

```

struct point { double x,y; } p[maxn],now,nex,ansp;
double dis(point a,point b) { return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y)); }
double f(point x) { //评估函数
    double res = 1e11;
    for (int i=1;i<=n;++i) res=min(res,dis(x,p[i]));
    return res;
}
void sa() {
    double T=1e5,d=0.99,TT=T/100000,dd=0.996,eps=1e-8;
    double res=f(now);
    if (res>ans) ans=res,ansp=now;
    while (T>eps) {
        for (int i=-1;i<=1;++i)
            for (int j=-1;j<=1;++j) if (i||j) {
                nex.x=now.x+T*i;
                nex.y=now.y+T*j;
                if (nex.x<0||nex.y<0||nex.x>x||nex.y>y) continue;
                double tmp=f(nex);
                if (tmp>ans) ans=tmp,ansp=nex;
                if (tmp>res) res=tmp,now=nex;
                else if (TT>rand()%10000/10000.0) res=tmp,now=nex;
            }
        T*=d; TT*=dd;
    }
}
int main() {
    srand(time(0));
    scanf("%d%d%d",&x,&y,&n);
    for (int i=1;i<=n;++i) scanf("%lf%lf",&p[i].x,&p[i].y);
    now.x=0,now.y=0; sa();
    now.x=x/2,now.y=y/2; sa();
    printf("The safest point is (%.11f, %.11f).\n",ansp.x,ansp.y);
    return 0;
}

```

三分{三分{三分}}

三维凸函数另一解法

```

int n;
struct point{ double x,y,z; } p[maxn],pp;
double dis(point a,point b){
    return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y)+(a.z-b.z)*(a.z-b.z);
}
double f(double x,double y,double z) {
    double res=0;
    for (int i=1;i<=n;i++) res=max(res,dis(point{x,y,z},p[i]));
    return res;
}
double findz (double x,double y) {
    double l=-1e4,r=1e4;
    while (r-l>eps) {

```

```

        double m1=(l+l+r)/3;
        double m2=(l+r+r)/3;
        if (f(x,y,m1)<f(x,y,m2)) r=m2;
        else l=m1;
    }
    pp.z=(l+r)/2;
    return get(x,y,(l+r)/2);
}
double findy(double x) {
    double l=-1e4,r=1e4;
    while (r-l>eps) {
        double m1=(l+l+r)/3;
        double m2=(l+r+r)/3;
        if (findz(x,m1)<findz(x,m2)) r=m2;
        else l=m1;
    }
    pp.y=(l+r)/2;
    return findz(x,(l+r)/2);
}
double findx() {
    double l=-1e4,r=1e4;
    while (r-l>eps) {
        double m1=(l+l+r)/3;
        double m2=(l+r+r)/3;
        if (findy(m1)<findy(m2)) r=m2;
        else l=m1;
    }
    pp.x=(l+r)/2;
    return findy((l+r)/2);
}
int main() {
    scanf("%d",&n);
    for(int i=1;i<=n;++i) scanf("%lf%lf%lf",&p[i].x,&p[i].y,&p[i].z);
    findx();
    printf("%.9lf %.9lf %.9lf\n",pp.x,pp.y,pp.z);
    return 0;
}

```

Simpson积分

$$\int_a^b f(x)dx \approx \frac{\delta x*(f_0+4*f_1+f_2)}{6} + \frac{\delta x*(f_2+4*f_3+f_4)}{6} + \dots + \frac{\delta x*(f_{n-2}+4*f_{n-1}+f_n)}{6}$$

自适应: $|S(a,c) + S(c,b) - S(a,b)| < 15 * eps$ 就返回结果不进行递归

```

inline double f(double x) { }
inline double simpson(double a,double b) {
    double c=a+(b-a)/2;
    return (f(a)+4*f(c)+f(b))*(b-a)/6;
}
inline double asr(double a,double b,double eps,double A) {
    double c=a+(b-a)/2;
    double L=simpson(a,c),R=simpson(c,b);

```

```

    if (fabs(L+R-A)<=15*eps) return L+R+(L+R-A)/15;
    return asr(a,c,eps/2,L)+asr(c,b,eps/2,R);
}
inline double asr(double a,double b,double eps) {
    return asr(a,b,eps,simpson(a,b));
}

```

FFT & NTT

```

const double pi = acos(-1.0);
const int maxn = 262150; //2的幂!!
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);
    }
} a[maxn],b[maxn];
int n,rev[maxn],ans[maxn];
char s[maxn],t[maxn];
void get_rev(int bit){
    for(int i=0;i<(1<<bit);++i)
        rev[i]=(rev[i>>1]>>1)|((i&1)<<(bit-1));
}
void fft(Complex *a,int len,int dft) {
    for(int i=0;i<len;++i) if(i<rev[i])
        swap(a[i],a[rev[i]]);
    for(int i=1;i<len;i<=1) {
        Complex wn=Complex(cos(-dft*pi/i),sin(-dft*pi/i));
        for(int j=0;j<len;j+=i<<1) {
            Complex wnk(1,0);
            for(int k=j;k<j+i;++k,wnk=wnk*wn) {
                Complex x=a[k],y=wnk*a[k+i];
                a[k]=x+y;a[k+i]=x-y;
            }
        }
    }
    if(dft==1) for(int i=0;i<len;++i) a[i].x/=len;
}
int main() {
    while (~scanf("%s%s",s,t)) {
        memset(a,0,sizeof a);
        memset(b,0,sizeof b);
        memset(ans,0,sizeof ans);
        int l1=strlen(s),l2=strlen(t),bit=1,len=2;
        for (;(1<<bit)<l1+l2-1;++bit,len<=1);
    }
}

```

```

    for (int i=0;i<l1;++i) a[l1-i-1].x=s[i]-'0';
    for (int i=0;i<l2;++i) b[l2-i-1].x=t[i]-'0';
    get_rev(bit); fft(a,len,1); fft(b,len,1);
    for (int i=0;i<len;i++) a[i]=a[i]*b[i];
    fft(a,len,-1);
    for (int i=0;i<len;++i) {
        ans[i]+=int(a[i].x+0.5);
        ans[i+1]+=ans[i]/10;
        ans[i]%=10;
    }
    int i;
    for (i=len;i>=0&&!ans[i];--i);
    if (i==-1) printf("0");
    for (;i>=0;--i) printf("%d",ans[i]);
    printf("\n");
}
return 0;
}

```

```

ll a[maxn],b[maxn];
const int mod=479*(1<<21)+1,G=3;
void ntt(ll* a,int len,int dft) {
    for(int i=0;i<len;++i) if(i<rev[i])
        swap(a[i],a[rev[i]]);
    for(int i=1;i<len;i<=1){
        ll wn=pow_mod(G,dft*(mod-1)/(i*2),mod);
        for(int j=0;j<len;j+=i<=1) {
            ll wnk=1;
            for(int k=j;k<j+i;++k,wnk=(wnk*wn)%mod){
                ll x=a[k]%mod,y=wnk*a[k+i]%mod;
                a[k]=(x+y)%mod;
                a[k+i]=((x-y)%mod+mod)%mod;
            }
        }
    }
    if(dft==-1) {
        ll tmp=inv(len,mod);
        for(int i=0;i<len;++i)
            a[i]=a[i]*tmp%mod;
    }
}
int main() {
    fin;
    while(~scanf("%s%s",s,t)) {
        memset(a,0,sizeof a);
        memset(b,0,sizeof b);
        int l1=strlen(s),l2=strlen(t),bit=1,len=2;;
        for(;(1<<bit)<l1+l2-1;++bit,len<=1);
        for(int i=0;i<l1;++i) a[i]=s[l1-i-1]-'0';
        for(int i=0;i<l2;++i) b[i]=t[l2-i-1]-'0';
        get_rev(bit); ntt(a,len,1); ntt(b,len,1);
        for(int i=0;i<len;++i) a[i]=a[i]*b[i]%mod;
        ntt(a,len,-1);
    }
}

```



```

    for(int i=0;i<len;++i) {
        a[i+1]+=a[i]/10;
        a[i]%=10;
    }
    int i;
    for (i=len;i>=0&&!a[i];--i);
    if (i == -1) printf("0");
    for (;i>=0;--i) printf("%d",a[i]);
    printf("\n");
}
return 0;
}

```

NTT中找原根

```

typedef long long ll;
const int MAX = 105;
ll qpow(ll a,ll b,ll mod) {
    ll ans = 1;
    while(b) {
        if(b&1) ans = ans * a % mod;
        b>>=1;
        a = a * a % mod;
    }
    return ans;
}
int pri[MAX],tot;
ll getRoot(ll p) //求质数p的最小原根
{
    tot=0;
    ll n = p-1, sq = sqrt(p+0.5);
    for(int i=2;i<=sq;i++) if(n%i == 0) {
        pri[tot++]=i;
        while(n%i==0) n/=i;
    }
    if(n != 1) pri[tot++] = n;
    for(int g=2;g<=p-1;g++) {
        int flag=1;
        for(int i=0;i<tot;i++)
            if(qpow(g,(p-1)/pri[i],p) == 1) {
                flag=0;
                break;
            }
        if(flag) return g;
    }
    return -1;
}
int main() {
    fin;
    cout<<getRoot(2013265921);
    return 0;
}

```

HDU6270 分治NTT 生成函数 容斥

HDU 6265 迪利克雷卷积

拉格朗日插值

$$1^k + 2^k + 3^k + \cdots + n^k$$

```
#include<cstdio>
#define N 1000006
#define M 1000000007
int n,k,t;
long long x,y,f=1,fac[N],inv[N],ans,sum;
int main(){
    scanf("%d%d",&n,&k); k++;
    inv[0]=inv[1]=1; fac[0]=n-k;
    for(int i=2;i<=k;i++)
        inv[i]=-M/i*inv[M%i]%M;
    for(int i=1;i<=k;i++)
        fac[i]=fac[i-1]*(n-k+i)%M,
        inv[i]=inv[i]*inv[i-1]%M;
    for(int i=0;i<=k;i++) {
        for(x=i,t=k-1,y=!!i;t>=1)
            t&1?y=y*x%M:0,x=x*x%M;
        (sum+=y)%=M;
        ans+=sum*f%M*(i==k?1:fac[k-i-1])%M*inv[i]%M*inv[k-i]%M*(k-i&1?-1:1);
        f=f*(n-i)%M;
    }
    printf("%lld", (ans%M+M)%M);
    return 0;
}
```

杜教BM算法

解决线性递推式

```
#include <cassert>
#include<bits/stdc++.h>
using namespace std;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define fi first
#define se second
#define SZ(x) ((int)(x).size())
typedef vector<int> VI;
typedef long long ll;
typedef pair<int,int> PII;
const ll mod=1000000007;
```

```

ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0); for(;b>=1;
{if(b&1)res=res*a%mod;a=a*a%mod;}return res;}
// head
int _,n;
namespace linear_seq {
    const int N=10010;
    ll res[N],base[N],_c[N],_md[N];

    vector<int> Md;
    void mul(ll *a,ll *b,int k) {
        rep(i,0,k+k) _c[i]=0;
        rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
        for (int i=k+k-1;i>=k;i--) if (_c[i])
            rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[j]]-_c[i]*_md[Md[j]])%mod;
        rep(i,0,k) a[i]=_c[i];
    }
    int solve(ll n,VI a,VI b) { // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
//      printf("%d\n",SZ(b));
        ll ans=0,pnt=0;
        int k=SZ(a);
        assert(SZ(a)==SZ(b));
        rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
        Md.clear();
        rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
        rep(i,0,k) res[i]=base[i]=0;
        res[0]=1;
        while ((1ll<<pnt)<=n) pnt++;
        for (int p=pnt;p>=0;p--) {
            mul(res,res,k);
            if ((n>>p)&1) {
                for (int i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;
                rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]*_md[Md[j]])%mod;
            }
        }
        rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
        if (ans<0) ans+=mod;
        return ans;
    }
}
VI BM(VI s) {
    VI C(1,1),B(1,1);
    int L=0,m=1,b=1;
    rep(n,0,SZ(s)) {
        ll d=0;
        rep(i,0,L+1) d=(d+(1ll)C[i]*s[n-i])%mod;
        if (d==0) ++m;
        else if (2*L<=n) {
            VI T=C;
            ll c=mod-d*powmod(b,mod-2)%mod;
            while (SZ(C)<SZ(B)+m) C.pb(0);
            rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
            L=n+1-L; B=T; b=d; m=1;
        } else {
            ll c=mod-d*powmod(b,mod-2)%mod;

```

```

        while (SZ(C)<SZ(B)+m) C.pb(0);
        rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
        ++m;
    }
}
return C;
}
int gao(VI a,ll n) {
    VI c=BM(a);
    c.erase(c.begin());
    rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
    return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
}
};

int main() {
    while (~scanf("%d",&n)) {
        vector<int>v;
        v.push_back(1);
        v.push_back(2);
        v.push_back(4);
        v.push_back(7);
        v.push_back(13);
        v.push_back(24);
        printf("%d\n",linear_seq::gao(v,n-1));
    }
}

```

附录

完美数

6
 28
 496
 8128
 33550336
 8589869056
 137438691328
 2305843008139952128
 2658455991569831744654692615953842176
 191561942608236107294793378084303638130997321548169216

欧拉函数

1	1	2	2	4	2	6	4	6	4	10	4	12	6	8	8	16	6	18	8
12	10	22	8	20	12	18	12	28	8	30	16	20	16	24	12	36	18	24	16
40	12	42	20	24	22	46	16	42	20	32	24	52	18	40	24	36	28	58	16
60	30	36	32	48	20	66	32	44	24	70	24	72	36	40	36	60	24	78	32
54	40	82	24	64	42	56	40	88	24	72	44	60	46	72	32	96	42	60	40

莫比乌斯

1	-1	-1	0	-1	1	-1	0	0	1	-1	0	-1	1	1	0	-1	0	-1	0
1	1	-1	0	0	1	0	0	-1	-1	-1	0	1	1	1	0	-1	1	1	0
-1	-1	-1	0	0	1	-1	0	0	0	1	0	-1	0	1	0	1	1	-1	0
-1	1	0	0	1	-1	-1	0	1	-1	-1	0	-1	1	0	0	1	-1	-1	0
0	1	-1	0	1	1	1	0	-1	0	1	0	1	1	1	0	-1	0	0	0

杨辉三角

C_{n+m-2}^{m-1}

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	3	4	5	6	7	8	9	10	11	12	13
3	1	3	6	10	15	21	28	36	45	55	66	78	91
4	1	4	10	20	35	56	84	120	165	220	286	364	455
5	1	5	15	35	70	126	210	330	495	715	1001	1365	1820
6	1	6	21	56	126	252	462	792	1287	2002	3003	4368	6188
7	1	7	28	84	210	462	924	1716	3003	5005	8008	12376	18564
8	1	8	36	120	330	792	1716	3432	6435	11440	19448	31824	50388
9	1	9	45	165	495	1287	3003	6435	12870	24310	43758	75582	125970
10	1	10	55	220	715	2002	5005	11440	24310	48620	92378	167960	293930
11	1	11	66	286	1001	3003	8008	19448	43758	92378	184756	352716	646646
12	1	12	78	364	1365	4368	12376	31824	75582	167960	352716	705432	1352078
13	1	13	91	455	1820	6188	18564	50388	125970	293930	646646	1352078	2704156

卡特兰数

1	1	2	5	14	42	132	429	1430	4862	16796	58786	208012	742900	2674440	9694845
---	---	---	---	----	----	-----	-----	------	------	-------	-------	--------	--------	---------	---------

原根

素数	r	k	g
3	1	1	2
5	1	2	2
17	1	4	3
97	3	5	5
193	3	6	5
257	1	8	3
7681	15	9	17
12289	3	12	11
40961	5	13	3
65537	1	16	3
786433	3	18	10
5767169	11	19	3
7340033	7	20	3
23068673	11	21	3
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
1004535809	479	21	3
2013265921	15	27	31
2281701377	17	27	3
3221225473	3	30	5
75161927681	35	31	3
77309411329	9	33	7
206158430209	3	36	22
2061584302081	15	37	7
2748779069441	5	39	3
6597069766657	3	41	5
39582418599937	9	42	5

素数	r	k	g
79164837199873	9	43	5
263882790666241	15	44	7
1231453023109121	35	45	3
1337006139375617	19	46	3
3799912185593857	27	47	5
4222124650659841	15	48	19
7881299347898369	7	50	6
31525197391593473	7	52	3
180143985094819841	5	55	6
1945555039024054273	27	56	5
4179340454199820289	29	57	3

图论

一般变量含义：源点s，汇点t，边u->v，权值w，花费c，父节点f

前向星

```
int head[maxn], nume;
struct edge { int v, w, c, next; } e[maxm]; // v顶点, w权值, c花费
inline void init_edge() { memset(head, -1, sizeof head); nume=0; }
inline void add_edge(int u, int v, int w=0, int c=0) {
    e[nume].v=v; e[nume].w=w; e[nume].c=c;
    e[nume].next=head[u]; head[u]=nume++;
}

void dfs(int u) { // 遍历方法
    vis[u]=1;
    for (int i=head[u]; ~i; i=e[i].next) {
        int v=e[i].v, w=e[i].w;
        if (vis[v]) continue;
        dfs(v);
    }
    // vis[u]=0;
}
```

最短路

Dijkstra

不能出现负边权，dis数组返回s到各点距离

```
int dis[maxn]; bool vis[maxn];
inline bool dijkstra(int s,int n) {
    memset(vis,0,sizeof vis);
    memset(dis,0x3f,sizeof dis); dis[s]=0;
    for (int j=1;j<=n;++j) { // 每次找一个距离最近的点, 需要n次
        int u,d=inf;
        for (int i=1;i<=n;++i)
            if (!vis[i]&&dis[i]<d)
                u=i,d=dis[i];
        if (d==inf) return false; // 有起点无法到达的顶点
        vis[u]=1;
        for (int i=head[u];~i;i=e[i].next) {
            int v=e[i].v,w=e[i].w;
            if (!vis[v]&&dis[v]>dis[u]+w)
                dis[v]=dis[u]+w;
        }
    }
    return true;
}
```

Dijkstra 堆优化

手写堆 $O(E\log V)$, STL优先队列优化 $O(E\log E)$

```
int dis[maxn];
inline void dijkstra(int s) {
    memset(dis,0x3f,sizeof dis); dis[s]=0;
    priority_queue<PII,vector<PII>,greater<PII> > que;
    que.push(make_pair(0,s));
    while (!que.empty()) {
        int u=que.top().second,d=que.top().first;
        que.pop();
        if (dis[u]<d) continue;
        for (int i=head[u];~i;i=e[i].next) {
            int v=e[i].v,w=e[i].w;
            if (dis[v]>d+w) {
                dis[v]=d+w;
                que.push(make_pair(dis[v],v));
            }
        }
    }
}
```

Bellman_Ford

可以有负权边，可以判断是否存在负环回路

```
struct edge { int u,v,w; } e[maxm];
int dis[maxn];
```



```

inline bool Bellman_Ford(int s,int n,int m) {
    memset(dis,0x3f,sizeof dis); dis[s]=0;
    for (int i=1;i<=n;++i) {
        bool update=0;
        for (int j=1;j<=m;++j)
            if (dis[e[j].v]>dis[e[j].u]+e[j].w) {
                dis[e[j].v]=dis[e[j].u]+e[j].w;
                update=1;
            }
        if (!update) return true; // 没有负环
    }
    return false;
}

```

Bellman_Ford队列优化

```

int dis[maxn],cnt[maxn]; bool vis[maxn];
inline bool bellman_Ford(int s,int n) {
    memset(vis,0,sizeof vis); vis[s]=1;
    memset(cnt,0,sizeof cnt); cnt[s]=1;
    memset(dis,0x3f,sizeof dis); dis[s]=0;
    queue<int> que;
    que.push(s);
    while (!que.empty()) {
        int u=que.front(); que.pop();
        vis[u]=0;
        for (int i=head[u];~i;i=e[i].next) {
            int v=e[i].v,w=e[i].w;
            if (dis[v]>dis[u]+w) {
                dis[v]=dis[u]+w;
                if (!vis[v]) {
                    vis[v]=1;
                    que.push(v);
                    if (++cnt[v]>n) return false;
                }
            }
        }
    }
    return true;
}

```

Floyd_Warshall

任意两点之间的最短路

```

int dis[maxn][maxn];
int cost[maxn][maxn];
inline void floyd_warshall(int s) {
    memcpy(dis, cost, sizeof cost);
    for (int k=1; k<=n; ++k)
        for (int i=1; i<=n; ++i)
            for (int j=1; j<=n; ++j)
                if (dis[i][j]>dis[i][k]+dis[k][j])
                    dis[i][j]=dis[i][k]+dis[k][j];
}

```

次短路

```

int dis1[maxn], dis2[maxn];
inline int dijkstra(int s) {
    memset(dis1, 0x3f, sizeof dis1); dis1[s]=0;
    memset(dis2, 0x3f, sizeof dis2);
    priority_queue<PII, vector<PII>, greater<PII> > que;
    que.push(make_pair(0, s));
    while (!que.empty()) {
        PII tmp=que.top(); que.pop();
        int d=tmp.first, u=tmp.second;
        if (dis2[u]<d) continue;
        for (int i=head[u]; ~i; i=e[i].next) {
            int v=e[i].v, w=e[i].w;
            if (dis1[v]>d+w) {
                dis2[v]=dis1[v];
                dis1[v]=d+w;
                que.push(make_pair(dis1[v], v));
            }
            else if (dis2[v]>d+w) {
                dis2[v]=d+w;
                que.push(make_pair(dis2[v], v));
            }
        }
    }
    return dis2[n];
}

```

k短路

1. 建两个图e, e2 (e2为e的反图)
2. 反图上跑dijkstra堆优化得出汇点到各点的最短距离，当作评估函数
3. 原图跑Astar算法

```

#include<bits/stdc++.h>
using namespace std;
typedef pair<int, int> PII;
const int inf=0x3f3f3f3f;
const int maxn=1e3+5;
const int maxm=1e4+5;

```

```

int head[maxn], nume;
struct edge { int v, w, next; } e[maxm], e2[maxm];
inline void init_edge() { memset(head, -1, sizeof head); nume=0; }
inline void add_edge(int u, int v, int w) {
    e[nume].v=v; e[nume].w=w; e[nume].next=head[u]; head[u]=nume++;
}
int head2[maxn], nume2;
inline void init_edge2() { memset(head2, -1, sizeof head2); nume2=0; }
inline void add_edge2(int u, int v, int w) {
    e2[nume2].v=v; e2[nume2].w=w; e2[nume2].next=head2[u]; head2[u]=nume2++;
}

int dis[maxn];
inline void dijkstra(int s) {
    memset(dis, 0x3f, sizeof dis); dis[s]=0;
    priority_queue<PII, vector<PII>, greater<PII> > que;
    que.push(make_pair(0, s));
    while (!que.empty()) {
        int u=que.top().second, d=que.top().first; que.pop();
        if (dis[u]<d) continue;
        for (int i=head2[u]; ~i; i=e2[i].next) {
            int v=e2[i].v, w=e2[i].w;
            if (dis[v]>d+w) {
                dis[v]=d+w;
                que.push(make_pair(dis[v], v));
            }
        }
    }
}

struct AstarNode {
    int u, g, f;
    AstarNode(int u=0, int g=0, int f=0):u(u), g(g), f(f) {}
    bool operator < (const AstarNode &b) const {
        return f==b.f?g>b.g:f>b.f;
    }
} now;

inline int Astar(int s, int t, int k) {
    int cnt=0;
    priority_queue<AstarNode> que;
    if (s==t) k++; // 第一次判u==t不算
    if (dis[s]==inf) return -1;
    que.push(AstarNode(s, 0, dis[s]));
    while (!que.empty()) {
        now=que.top(); que.pop();
        int u=now.u, g=now.g, f=now.f;
        if (u==t) cnt++;
        if (cnt==k) return now.g;
        for (int i=head[u]; ~i; i=e[i].next) {
            int v=e[i].v, w=e[i].w;
            que.push(AstarNode(v, now.g+w, now.g+w+dis[v]));
        }
    }
}

```

```

    return -1;
}

int n,m,s,t,k,sum;
int main() {
    while (~scanf("%d%d",&n,&m)) {
        init_edge();
        init_edge2();
        scanf("%d%d%d%d",&s,&t,&k,&sum);
        for (int i=1,u,v,w;i<=m;i++) {
            scanf("%d%d%d",&u,&v,&w);
            add_edge(u,v,w);
            add_edge2(v,u,w);
        }
        dijkstra(t);
        int res=Astar(s,t,k);
        if(res==-1) printf("Whitesnake!\n");
        else if(sum >= res) printf("YareYaredawa!\n");
        else printf("Whitesnake!\n");
    }
    return 0;
}

```

差分约束

1. 求最短路(符合条件的最大解), 初始化距离inf

$$d_u - d_v \leq w \Rightarrow d_u \leq d_v + w, \text{ add_edge}(v, u, w);$$

2. 求最长路(符合条件的最小解)

$$d_u - d_v \geq w \Rightarrow d_v \leq d_u - w, \text{ add_edge}(u, v, -w);$$

3. 特殊边

$$d_u - d_v = w, \text{ add_edge}(u, v, -w); \text{ add_edge}(v, u, w);$$

生成树

最小生成树

Prim $O(V^2)$

```

int dis[maxn]; bool vis[maxn];
inline int Prim(int s, int n) {
    memset(vis, 0, sizeof vis);
    memset(dis, 0x3f, sizeof dis); dis[s] = 0;
    int res = 0;
    for (int j = 1; j <= n; ++j) { // 每次找一个距离最近的点, 需要n次
        int u, min_w = inf;
        for (int i = 1; i <= n; ++i)
            if (!vis[i] && dis[i] < min_w)
                u = i, min_w = dis[i];
        if (min_w == inf) return -1; // 有起点无法到达的顶点
    }
}

```

```

        vis[u]=1;
        res+=min_w;
        for (int i=head[u];~i;i=e[i].next) {
            int v=e[i].v,w=e[i].w;
            if (!vis[v]&&dis[v]>w) dis[v]=w;
        }
    }
    return res;
}

```

Kruskal $O(E \log E)$

用并查集判断是否两个点属于一个集合

```

struct edge { int u,v,w; } e[maxm];
bool cmp(edge a,edge b) { return a.w<b.w; }
inline int kruskal(int n) {
    memset(fa,-1,sizeof fa);
    sort(e+1,e+1+n,cmp);
    int res=0,num=0;
    for (int i=1;i<=n&&num<n;++i) {
        int u=e[i].u,v=e[i].v,w=e[i].w;
        if (Find(u)!=Find(v)) {
            res+=w;
            ++num;
            Unite(u,v);
        }
    }
    return res;
}

```

次小生成树 $O(V^2)$

1. 先求最小生成树
2. 枚举所有不在MST中的边，加入到MST中会形成回路
3. 替换掉回路中最大边权的边，更新到答案

```

int pre[maxn];
int dis[maxn];
bool vis[maxn];
bool use[maxn][maxn]; // 最小生成树使用的边
int maxv[maxn][maxn]; // u->v中边权最大值
int cost[maxn][maxn]; // u->v的花费

inline int prim(int s,int n) {
    memset(vis,0,sizeof vis);
    memset(use,0,sizeof use);
    memset(maxv,0,sizeof maxv);
    memset(dis,0x3f,sizeof dis); dis[s]=0;
    for (int i=1;i<=n;i++) pre[i]=s;

    int res=0;

```

```

for (int j=1;j<=n;++j) {
    int u,minw=inf;
    for (int i=1;i<=n;++i)
        if (!vis[i]&&dis[i]<min_w)
            u=i,minw=dis[i];
    if (minw==inf) return -1; // 不连通
    res+=dis[u]; vis[u]=1
    use[u][pre[u]]=use[pre[u]][u]=1;
    for (int v=1;v<=n;++v) {
        if (vis[v])
            maxv[u][v]=maxv[v][u]=max(maxv[v][pre[u]],dis[u]);
        if (!vis[v]&&dis[v]>cost[u][v])
            dis[v]=cost[u][v],pre[v]=u;
    }
}
return res;
}
inline int smst(int s,int n) { // s:起点 ans:最小生成树的值
    int res=inf,ans=prim(s,n);
    if (ans==-1) return -1;
    for (int i=1;i<=n;++i)
        for (int j=i+1;j<=n;++j)
            if (cost[i][j]!=inf&&!use[i][j]) // i->j有边且不在最小生成数里
                res=min(res,ans+cost[i][j]-maxv[i][j]);
    return res==inf?-1:res;
}

```

最优比例生成树

每条边有花费和长度两个非负参数，求一个生成树，使花费 c 和与长度 l 和之比最小

$$R = \frac{c_1 * x_1 + c_2 * x_2 + \dots + c_n * x_n}{l_1 * x_1 + l_2 * x_2 + \dots + l_n * x_n}$$

$$F(R) = \sum (c_i - R * l_i) * x_i$$

$F(R)$ 是单调递减的函数，所以和01分数规划求最大的 R 差不多

$F(R) = 0$ 时 R 为正确答案， $F(R) > 0$ ，说明 R 取小了

POJ2728 两点花费为欧几里得距离，长度为高度差

方法一：01分数规划(详见数学二分)

```

int n;
double x[maxn],y[maxn],h[maxn],g[maxn][maxn];
double cost[maxn][maxn],len[maxn][maxn];
inline bool judge(double R) {
    for (int i=1;i<=n;++i)
        for (int j=i+1;j<=n;++j)
            g[i][j]=g[j][i]=cost[i][j]-R*len[i][j];
    return prime(1)>=0;
}

```

```

}
inline void solve() {
    double l=0,r=1e8;
    while (r-l>eps) {
        double m=(l+r)/2;
        if (judge(m)) l=m;
        else r=m;
    }
    printf("%.31f\n",l);
}

```

方法二：迭代

1. 先让 $R_0 = \text{inf}$ (比正常答案大一点就行)，用 R 取构造新的图， $g[i][j] = \text{cost}[i][j] - R * \text{len}[i][j]$
2. 按新图求最小生成树，记录下总花费和总长度，新答案 $R = \text{总花费} / \text{总长度}$
3. $R < R_0$ 更新答案，否则退出，这个过程 R 是单调递减的

```

double x[maxn],y[maxn],h[maxn],g[maxn][maxn];
double cost[maxn][maxn],len[maxn][maxn];

int pre[maxn]; double dis[maxn]; bool vis[maxn];
inline double prime(int s) {
    memset(pre,0,sizeof pre);
    memset(vis,0,sizeof vis);
    memset(dis,0x66,sizeof dis); dis[s]=0;
    double sumcost=0,sumlen=0;
    for (int j=1,u;j<=n;++j) {
        double min_w=1e11;
        for (int i=1;i<=n;++i)
            if (!vis[i]&&min_w>dis[i])
                u=i,min_w=dis[i];
        vis[u]=1;
        sumcost+=cost[pre[u]][u];
        sumlen+=len[pre[u]][u];
        for (int v=1;v<=n;++v)
            if (!vis[v]&&dis[v]>g[u][v])
                dis[v]=g[u][v], pre[v]=u;
    }
    return sumcost/sumlen;
}

void solve() {
    double R=1e8;
    while (1) {
        double R0=R;
        for (int i=1;i<=n;++i)
            for (int j=i+1;j<=n;++j)
                g[i][j]=g[j][i]=cost[i][j]-R*len[i][j];
        R=prime(1);
        if (fabs(R0-R)<eps) break;
    }
    printf("%.31f\n",R);
}

```

最近公共祖先

倍增法实现

```
int dep[maxn], dp[maxn][20], numpow=19;
void dfs(int u, int f) { // 调用f为0, 点下标从1开始
    dep[u]=dep[f]+1; dp[u][0]=f;
    for (int i=1; i<=numpow; ++i) dp[u][i]=dp[ dp[u][i-1] ][i-1];
    for (int i=head[u]; ~i; i=e[i].next) {
        int v=e[i].v;
        if (v==f) continue;
        dfs(v, u);
    }
}
int lca(int u, int v) {
    if (dep[u]>dep[v]) swap(u, v);
    if (dep[u]<dep[v]) {
        int tmp=dep[v]-dep[u];
        for (int i=0; i<=numpow; ++i) if (tmp&(1<<i)) v=dp[v][i];
    }
    if (u!=v) {
        for (int i=numpow; ~i; --i)
            if (dp[u][i]!=dp[v][i])
                u=dp[u][i], v=dp[v][i];
        u=v=dp[u][0];
    }
    return u;
}
```

树分治

1. `getroot(int u, int f)` 找重心
2. `getval(int u, int f)` 获得以u为根的子树值
3. `solve(int u)` 获得以u为根的子树的答案

```
int dp[maxn], sz[maxn], rt; bool used[maxn];
inline void getroot(int u, int f) {
    dp[u]=0; sz[u]=1;
    for (int i=head[u]; ~i; i=e[i].next) {
        int v=e[i].v;
        if (used[v] || v==f) continue;
        getroot(v, u);
        dp[u]=max(dp[u], sz[v]);
        sz[u]+=sz[v];
    }
    dp[u]=max(dp[u], dp[0]-sz[u]);
    if (dp[u]<dp[rt]) rt=u;
}
inline int getval(int u, int f) {
    sz[u]=1; // 统计子节点个数, 方便下次找重心
    for (int i=head[u]; ~i; i=e[i].next) {
```



```

        int v=e[i].v;
        if (used[v]||v==f) continue;
        getval(v,u);
        sz[u]+=sz[v];
    }
}
inline void solve(int u) {
    getval(u,0);
    used[u]=1;
    for (int i=head[u];~i;i=e[i].next) {
        int v=e[i].v,w=e[i].w;
        if (used[v]) continue;
        dp[0]=sz[v]; getroot(v,rt=0);
        solve(rt);
    }
}
int main () {
    dp[0]=n; get_g(1,rt=0);
    solve(rt);
    return 0;
}

```

二分图

二分图：对于无向图 $G=(V,E)$ ，把顶点集 V 分成不相交的两部分，每条边的顶点属于两个不同的子集

匹配：二分图的某个子图中，任意两边没有公共顶点

最大匹配：边数最大的匹配

完全匹配：匹配数 $|M|=\min\{|A|,|B|\}$

完美匹配： $|M|=|A|=|B|$ 的完全匹配

最佳匹配：边权值和最大的完美匹配

概念

1. 点覆盖集：从无向图中选一个顶点集，图中任意边有顶点属于顶点集
2. 点独立集：从无向图中选一个顶点集，任意两顶点不相邻
3. 边覆盖集：从无向图中选一个边集，图中任意点为边集中边的顶点
4. 边独立集：从无向图中选一个边集，任意两边不相邻
5. 路径覆盖：在有向图中，找若干条路径，使之覆盖图中所有顶点，并且任何一个顶点只出现在一个路径中

定理

1. 二分图最小点覆盖=最大匹配数 (Konig定理)
2. 二分图最大点独立集=顶点数-最大匹配数
3. 有向无环图最小路径覆盖数=顶点数-最大匹配数

二分图染色

```

int color[maxn];
inline bool dfs(int u,int c) { // 把顶点u染色成c
    color[u]=c;
    for (int i=0;~i;i=e[i].next) {
        int v=e[i].v;
        if (color[v]==c) // 相邻顶点颜色相同
            return false;
        if (color[v]==0&&!dfs(v,-c)) // 相邻顶点没被染色, 染成-c
            return false;
    }
    return true;
}
inline bool solve() {
    for (int i=0;i<n;++i)
        if (color[i]==0&&!dfs(i,1)) // 没被染色, 染成1
            return false;
    return true;
}

```

二分图最大匹配

匈牙利算法, 邻接矩阵形式, 适用于稠密图 $O(VE)$

1. n,m为匹配的两点集的数量
2. g是图, 只需要表示u->v有边
3. 从1到n去匹配, 不能增加匹配数就不匹配它

```

int n,m,match[maxn],g[maxn][maxn]; bool vis[maxn];
inline bool dfs(int u) {
    for (int v=1;v<=m;++v) {
        if (vis[v]||!g[u][v]) continue; vis[v]=1;
        if (match[v]==-1||dfs(match[v])) {
            match[v]=u;
            return true;
        }
    }
    return false;
}
inline int Hungary() {
    int res=0;
    memset(match,-1,sizeof match);
    for (int u=1;u<=n;++u) {
        memset(vis,0,sizeof vis);
        if (dfs(u)) ++res;
    }
    return res;
}

```

匈牙利算法 邻接表形式

```

int n,m,match[maxn]; bool vis[maxn];
inline bool dfs(int u) {

```

```

    for (int i=head[u];~i;i=e[i].next) {
        int v=e[i].v;
        if (vis[v]) continue; vis[v]=1;
        if (match[v]==-1||dfs(match[v])) {
            match[v]=u;
            return true;
        }
    }
    return false;
}

inline int Hungary() {
    int res=0;
    memset(match,-1,sizeof match);
    for (int i=1;i<=n;++i) {
        memset(vis,0,sizeof vis);
        if (dfs(i)) ++res;
    }
    return res;
}

```

二分图多重匹配

```

const int maxn = 100005; // 左边的点个数
const int maxm = 15;     // 右边的点个数
int num[maxn],n,m;       // 右边的限制
int vis[maxn],match[maxn][maxn],g[maxn][maxn];
inline bool dfs(int u) {
    for(int v=1;v<=m;++v) {
        if(g[u][v]&&!vis[v]) {
            vis[v] = 1;
            if(match[v][0]<num[v]) {
                match[v][++match[v][0]] = u;
                return true;
            }
            for(int i=1;i<=num[v];++i) {
                if(dfs(match[v][i])) {
                    match[v][i] = u;
                    return true;
                }
            }
        }
    }
    return false;
}

inline int Hungary() {
    int res = 0;
    for(int i=1;i<=m;++i) match[i][0] = 0;
    for(int i=1;i<=n;++i) {
        memset(vis,0,sizeof vis);
        if(dfs(i)) ++res;
        else return 0;
    }
}

```

```

    return res;
}

```

二分图最佳匹配

Kuhn_Munkres算法 $O(n^3)$

```

int n,g[maxn][maxn],slack[maxn];    // 匹配图 松弛期望
int match[maxn],exu[maxn],exv[maxn]; // match[v]=u 期望
bool visu[maxn],visv[maxn];
inline bool dfs(int u) {
    visu[u]=1;
    for (int v=1;v<=n;++v) {
        if (visv[v]) continue;
        int ex=exu[u]+exv[v]-g[u][v];
        if (ex==0) {
            visv[v]=true;
            if (match[v]==-1||dfs(match[v])) {
                match[v]=u;
                return true;
            }
        } else // 左侧期望过高 需要减少(尽量小)
            slack[v]=min(slack[v],ex);
    }
    return false;
}

inline int km() {
    memset(match,-1,sizeof match); // 还没匹配
    memset(exv,0,sizeof exv);      // 右侧期望
    for (int u=1;u<=n;++u) {       // 左侧期望
        exu[u]=-inf;
        for (int v=1;v<=n;++v)
            if (g[u][v]>exu[u])
                exu[u]=g[u][v];
    }

    for (int u=1;u<=n;++u) {
        memset(slack,0x3f,sizeof slack);
        while (1) {
            memset(visu,0,sizeof visu);
            memset(visv,0,sizeof visv);
            if (dfs(u)) break; // 匹配退出
            // 找不到 降低左侧期望 增加右侧期望
            int d=inf;
            for (int v=1;v<=n;++v) if (!visv[v])
                d=min(d,slack[v]);
            for (int i=1;i<=n;++i) {
                if (visu[i]) exu[i]-=d;
                if (visv[i]) exv[i]+=d;
                else slack[i]-=d; // 期望降低
            }
        }
    }
}

```

```

    }
}
int res=0;
for (int v=1;v<=n;++v) if (match[v]!=-1)
    res+=g[match[v]][v];
return res;
}

```

二分图匹配方案

二分图S-左顶点-右顶点-T，流量为1。寻找每个点的满流弧，即为匹配。

```

inline void print(int s,int t) {
    for (int u=1;u<=n;++u)
        for (int i=head[u];~i;i=e[i].next) {
            int v=e[i].v,w=e[i].w;
            if (w==0&&e[j].v!=s)
                printf("%d %d\n",u,v);
        }
}

```

网络流

最大流

加边的同时添加反向边 `add_edge(u,v,w); add_edge(v,u,0)`

如果为无向边 `add_edge(u,v,w); add_edge(v,u,w)`

关于最小割

1. 最大流的流量=最小割的容量
2. 二选一模型可以转化为最小割，然后求最大流
3. 代价最小的前提下，破坏道路最少

假设有10000条边， $w \rightarrow 10001 * w + 1$ ，最终代价 $ans/10001$ ，道路数 $ans \% 10001$

最大闭合图的权=原图权值为正的点的和-最小割

Dinic算法 $O(V^2 E)$

1. bfs分层
2. 每次按最短路增广

```

int dep[maxn], que[maxn];
inline bool bfs(int s,int t) {
    memset(dep,-1,sizeof dep); dep[s]=0;
    int ss=0, ee=0; que[ee++]=s;
    while (ss<ee) {
        int u=que[ss++];
        for (int i=head[u];~i;i=e[i].next) {
            int v=e[i].v,w=e[i].w;

```

```

        if (w>0&&dep[v]==-1) {
            dep[v]=dep[u]+1;
            que[ee++]=v;
        }
    }
}
return dep[t]!=-1;
}
inline int dfs(int u,int t,int flow) {
    if (u==t) return flow;
    int res=0;
    for (int i=head[u];~i;i=e[i].next) {
        int v=e[i].v,w=e[i].w;
        if (w>0&&dep[u]+1==dep[v]) {
            int tmp=dfs(v,t,min(w,flow));
            res+=tmp;
            flow-=tmp;
            e[i].w-=tmp;
            e[i^1].w+=tmp;
            if (flow==0) break;
        }
    }
    if (res==0) dep[u]=-1;
    return res;
}
inline int Dinic(int s,int t) {
    int res=0;
    while (bfs(s,t)) res+=dfs(s,t,inf);
    return res;
}

```

最小费用最大流

加边的同时添加反向边 `add_edge(u,v,w,c); add_edge(v,u,0,-c)`

先找到费用最少的一条路，然后逆向修改流量

最大费用只需要花费取反，求出答案取反

```

int dis[maxn],pre[maxn]; bool vis[maxn];
inline bool Bellman_Ford(int s,int t) {
    memset(pre,-1,sizeof pre);
    memset(vis,0,sizeof vis); vis[s]=1;
    memset(dis,0x3f,sizeof dis); dis[s]=0;
    queue<int> que;
    que.push(s);
    while (!que.empty()) {
        int u=que.front(); que.pop();
        vis[u]=0;
        for (int i=head[u];~i;i=e[i].next) {
            int v=e[i].v,w=e[i].w,c=e[i].c;
            if (w>0&&dis[v]>dis[u]+c) {

```

```

        dis[v]=dis[u]+c;
        pre[v]=i; // 前驱边
        if (!vis[v]) {
            vis[v]=1;
            que.push(v);
        }
    }
}
return pre[t]!=-1;
}
inline ll cost_flow(int s,int t) {
    ll res=0;
    while (Bellman_Ford(s,t)) {
        int flow=inf;
        for (int i=t;i!=s;i=e[pre[i]^1].v)
            flow=min(e[pre[i]].w,flow);
        for (int i=t;i!=s;i=e[pre[i]^1].v) {
            e[pre[i]].w-=flow;
            e[pre[i]^1].w+=flow;
            res+=1ll*e[pre[i]].c*flow;
        }
    }
    return res;
}

```

强联通分量

1. tarjan算法 $O(V + E)$
2. 构成一个强连通分量需要添加max(入度0,出度0)

```

int dfn[maxn],low[maxn],tot; // 时间戳
int belong[maxn],scc; // 强连通分量(缩点)
int sta[maxn],top; bool insta[maxn];

inline void Tarjan(int u) {
    dfn[u]=low[u]=++tot;
    sta[top++]=u; insta[u]=1;
    for (int i=head[u];~i;i=e[i].next) {
        int v=e[i].v;
        if (!dfn[v])
            Tarjan(v),
            low[u]=min(low[u],low[v]);
        else if (insta[v])
            low[u]=min(low[u],dfn[v]);
    }
    if (low[u]==dfn[u]) {
        scc++;
        while (top) {
            int v=sta[--top];
            belong[v]=scc;
            insta[v]=0;
        }
    }
}

```

```

        if(u==v) break;
    }
}

//调用方法
int n,m,in[maxn],out[maxn];
for (int i=1;i<=n;i++) if(!dfn[i]) Tarjan(i);
for (int u=1;u<=n;u++) {
    for (int i=head[u];~i;i=e[i].next) {
        int v=e[i].v;
        if (belong[u]!=belong[v])
            out[belong[u]]++,
            in[belong[v]]++;
    }
}

```

拓扑排序

```

int in[maxn],que[maxn]; // add_edge(u,v,w); ++in[v];
inline bool topo(int n) {
    int ss=0,ee=0;
    for (int i=1;i<=n;++i)
        if(in[i]==0)
            que[ee++]=i;
    while (ss<ee) {
        int u=que[ss++];
        for (int i=head[u];~i;i=e[i].next) {
            int v=e[i].v;
            if (--in[v]==0)
                que[ee++]=v;
        }
    }
    return ee==n;
}

```

欧拉路

欧拉路径：恰好通过图G中每条边一次的路径

欧拉回路：起点和终点相同的欧拉路径

性质:

无向图

1. 图G存在欧拉回路，当且仅当是连通图且所有顶点度数都为偶数
2. 图G存在欧拉路径，可以存在零个或两个奇度顶点(分别为起点和终点)

有向图

1. 图G存在欧拉回路，当且仅当是连通图且所有顶点入度等于出度

2. 图G存在欧拉路径，当且仅当是连通图且满足一下两个条件之一

- 所有顶点的入度等于出度
- 起点出度-入度=1，终点出度-入度=-1，其余点入度等于出度

混合图

无向图路径

```
int g[maxn][maxn];
int indegree[maxn];
inline void dfs(int u) {
    if (indegree[u]>0) {
        for (int v=1;v<=n;++v) if (g[u][v]) {
            g[u][v]=g[v][u]=0;
            dfs(v);
        }
    }
    printf("%d ",u);
}
```

字符串

```
string s="abcde",r="123"; char t[]="+-*/%";
```

查找不到返回特殊值npos= -1或LONG_MAX

string库函数	功能	用法	变化
c_str	返回字符数组指针	strcpy(t,s.c_str())	t="abcde"
substr(pos=0,count=size)	返回子串[pos,pos+count)	r=s.substr(1,2)	r="bc"
erase(pos=0,count=size)	删除子串[pos,pos+count)	s.erase(1,2)	s="ade"
insert(pos,count,ch)	pos处插入count个字符ch	s.insert(1,2,'1')	s="a11bcde"
insert(pos,string s)	pos处插入字符串s	s.insert(1,r)	s="a123bcde"
append(count,ch)	串尾插入count个字符ch	s.append(2,'1')	s="abcde11"
find(ch,pos=0)	返回字符ch的开始位置	s.find('c')	2
find(string s,pos=0)	返回字符串s的开始位置	s.find(r)	npos -1
find_first_of(char *s,pos=0)	返回属于s的首字符	s.find_first("cd")	2
find_first_not_of(char *s,pos)	返回不属于s的首字符	s.find_first_not_of("ab")	2

字符串-数值转换

```
<stdlib.h>的ato_   strt_系列无异常
<string>
char s[]=" \t\r\n 123456789123456789.123456789";
```

```

string s=" \t\r\n 123456789123456789.123456789";
sto_(s,size_t* pos=0,int base=10)
// 忽略串首空白符, 取尽可能多的字符组成合法数值, 不能转换跳异常
// 参数: 字符串,不能转换的位置,进制(0~36) 进制0自动识别前缀0|0x
stoi(s)           // 异常
stoll(s)          // 123456789123456789
stof(s)           // 范围f 1e38 d 1e308 ld 1e4932
stoi("xyz",0,36)   // 44027 = 33*36*36 + 34*36 + 35

char s[]="123 456 789"; size_t pos;
for (char *p=s;*p;p+=pos) // 123\n456\n789
    cout<<stoi(p,&pos,10)<<endl;
to_string(value)

```

KMP

1. 字符串匹配 $O(n + m)$
2. next[i] 真前缀子串和真后缀子串的最大匹配值
3. len-next[len] 周期长度(不考虑最后一组是否成周期)
4. $\text{len} \% (\text{len} - \text{next}[\text{len}]) == 0 ? \text{len} - \text{next}[\text{len}] : \text{len}$ 实际周期长度
5. $\frac{\text{len}}{\text{len} - \text{next}[\text{len}]}$ 完整周期数
6. nextv[i] next[i]=next[next[...next[i]]]

```

char s[maxn],t[maxn]; int next[maxn];
void get_next(char t[]) {
    int i=0,j=next[0]=-1;
    while (t[i]) {
        while (j^=-1&& t[i]^t[j]) j=next[j];
        next[++i]=++j;
    }
}
void get_nextv(char t[]) {
    int i=0,j=next[0]=-1;
    while (t[i]) {
        while (j^=-1&& t[j]^t[i]) j=next[j];
        if (t[++i]==t[++j]) next[i]=next[j];
        else next[i]=j;
    }
}
bool kmp(char s[],char t[]) {
    int i=0,j=0;
    get_next(t,next); // get_nextv(t,next);
    while (s[i]) {
        while (j^=-1 && s[i]^t[j]) j=next[j];
        ++i,++j;
        if (!t[j]) return true;
    }
    return false;
}

```

扩展KMP

1. 最长公共前缀 $O(n + m)$
2. $\text{next}[i]$: $t[i, m-1]$ 与 $t[0, m-1]$ 的最长公共前缀
3. $\text{extend}[i]$: $s[i, n-1]$ 与 $t[0, m-1]$ 的最长公共前缀
4. 在位置 ps 取得最长公共前缀位置 pe , 求位置 i 的最长公共前缀

```
char s[maxn], t[maxn]; int next[maxn], extend[maxn];
void get_exnext(char t[]) {
    int ps, pe; next[0] = strlen(t);
    for (int i = 1, j = -1; t[i]; ++i, --j) {
        if (j < 0 || i + next[i - ps] >= pe) {
            if (j < 0) pe = i, j = 0;
            while (t[pe] && t[pe] == t[j]) ++pe, ++j;
            ps = i, next[i] = j;
        }
        else next[i] = next[i - ps];
    }
}
// s主串, t模式串
void exkmp(char s[], char t[]) {
    int ps, pe;
    get_exnext(t, next);
    for (int i = 0, j = -1; s[i]; ++i, --j) {
        if (j < 0 || i + next[i - ps] >= pe) {
            if (j < 0) pe = i, j = 0;
            while (s[pe] && t[j] && s[pe] == t[j]) ++pe, ++j;
            ps = i, extend[i] = j;
        }
        else extend[i] = next[i - ps];
    }
}
```

Manacher

1. 最长回文子串 $O(n)$
2. ababa => \$#a#b#a#b#a#
3. $\text{len}[i]$: 以 i 为中心最长回文半径, 因为有'#'号, 所以 $\text{len}[i]-1$ 为最长回文串
4. 在位置 ps 取得最长回文串位置 pe , 求位置 i 的回文串长度

```
char s[maxn], t[maxn << 1]; int len[maxn << 1];
int manacher(char s[], char t[]) {
    int l = 0;
    t[l++] = '$'; t[l++] = '#';
    for (int i = 0; s[i]; ++i)
        t[l++] = s[i], t[l++] = '#';
    t[l] = '\0';

    int ps = 1, pe = 1, ans = 1;
    for (int i = 1; t[i]; ++i) {
```

```

        len[i]=i<pe?min(len[2*ps-i],pe-i):1;
        while (t[i+len[i]]==t[i-len[i]]) ++le[i];
        if (pe<i+len[i]) ps=i,pe=i+len[i];
        if (ans<len[i]-1) ans=len[i]-1; // 回文串长度
        //ans+=len[i]/2; // 回文串数量
    }
    return ans;
}

```

AC自动机

1. 求目标串中出现了几个模式串

```

int t[maxn][26], num[maxn], fail[maxn], sz;
void add(char *s, int rt=0) {
    for (; *s; ++s) {
        int ch=*s-'a';
        if (!t[rt][ch]) t[rt][ch]=++sz;
        rt=t[rt][ch];
    }
    ++num[rt];
}
void get_fail(int rt=0) {
    queue<int> que;
    for (int i=0; i<26; ++i) if (t[rt][i])
        que.push(t[rt][i]);
    while (!que.empty()) {
        int u=que.front(); que.pop();
        for (int i=0, v; i<26; ++i)
            if (v=t[u][i]) {
                fail[v]=t[ fail[u] ][i];
                que.push(v);
            }
        else
            t[u][i]=t[ fail[u] ][i];
    }
}
int aho(char *s, int rt=0) {
    get_fail(); // 多次查询写在函数外
    int res=0;
    for (; *s; ++s) {
        int ch=*s-'a';
        rt=t[rt][ch];
        for (int p=rt; p&&num[p]>=0; p=fail[p]) {
            res+=num[p];
            num[p]=-1;
        }
    }
    return res;
}

```

后缀数组(SA)

```
//复杂度O(n*log(n))
const int m = 128; //ASCII
char s[maxn]; //待排序的字符串放在s数组中,从s[0~n-1],长度为n,且最大值小于m,最后一位是0(无效值)
int sa[maxn], t1[maxn], t2[maxn], c[maxn], n, k; //sa[1~n]为有效值,sa[0]必定为n是无效值
int rk[maxn]; //rank[0~n-1]为有效值,rank[n]必定为0无效值
int height[maxn]; //height[1~n]
//lcp(x,y): 字符串x与字符串y的最长公共前缀, 在这里指x号后缀与y号后缀的最长公共前缀
//height[i]: lcp(sa[i], sa[i-1]), 即排名为i的后缀与排名为i-1的后缀的最长公共前缀
//H[i]: height[rk[i]], 即i号后缀与它前一名后缀的最长公共前缀
void build_sa(int m)
{
    n++;
    int *x = t1, *y = t2;
    for(int i=0; i<m; i++) c[i] = 0;
    for(int i=0; i<n; i++) c[x[i] = s[i]]++;
    for(int i=1; i<m; i++) c[i] += c[i-1];
    for(int i=n-1; i>=0; i--) sa[--c[x[i]]] = i;
    for(int j=1; j<=n; j<=1)
    {
        int p = 0;
        for(int i = n-j; i<n; i++) y[p++] = i;
        for(int i=0; i<n; i++) if(sa[i]>=j) y[p++] = sa[i]-j;
        for(int i=0; i<m; i++) c[i] = 0;
        for(int i=0; i<n; i++) c[x[y[i]]]++;
        for(int i=1; i<m; i++) c[i] += c[i-1];
        for(int i=n-1; i>=0; i--) sa[--c[x[y[i]]]] = y[i];
        swap(x, y);
        p = 1; x[sa[0]] = 0;
        for(int i=1; i<n; i++)
            x[sa[i]] = (y[sa[i-1]] == y[sa[i]] && y[sa[i-1]+j] == y[sa[i]+j])?p-1:p++;
        if(p>=n) break;
        m = p;
    }
    n--;
    int k = 0;
    for(int i=0; i<=n; i++) rk[sa[i]] = i;
    for(int i=0; i<n; i++){
        if(k) k--;
        int j = sa[rk[i]-1];
        while(s[i+k] == s[j+k]) k++;
        height[rk[i]] = k;
    }
}

int main()
{
    // for(int i=1; i<=n; i++) cout<<sa[i]<<" ";
    // cout<<endl;
    // for(int i=0; i<n; i++) cout<<rk[i]<<" ";
    // cout<<endl;
    // for(int i=0; i<n; i++) cout<<height[rk[i]]<<" ";
}
```

```
//    cout<<endl;
    cin>>s;
    build_sa(m);
    return 0;
}
```

应用1：最长公共前缀（LCP）

给定一个字符串，询问某两个后缀的最长公共前缀。

求出height数组后，转化为某个区间上的最小值。使用RMQ ST算法 $O(n \cdot \log(n))$ 预处理 $O(1)$ 查询


应用2：最长可重叠重复子串

首先，我们知道的是height[i]表示排名为i的后缀与排名为i-1的后缀的LCP，那么相邻两后缀的LCP就是重叠重复子串，我们直接二分长度答案长度k，判断是否有height[i] ≥ k即可

```
bool check(int res) //以下二分的是 出现次数至少为k次的可重复重叠子串最长是多少
{
    //即可重叠的k次最长重复子串    res  <==> mid
    int cnt = 1;
    for(int i=1;i<=n;i++)
    {
        if(height[i]>=res){
            cnt++;
        }
        else{
            if(cnt>=k) return true;
            cnt = 1;
        }
    }
    return false;
}
```

应用3：最长不可重叠重复子串

相比于上述的"最长可重叠重复子串"稍微复杂一下，二分答案，假设存在长度为k的最长不可重叠重复子串。思想是把排好序的后缀分成若干组，其中每组后缀之间的height值都 ≥ k。

 1541074356993

可以看出，当k=2时，"231"和"23231"的公共前缀大于等于k，"23231"和"2323231"的公共前缀也大于等于k，所以这3个排名连续的后缀会被分到一组。同理"3231"和"323231"也会被分到一组。

对于k=3，"23231"和"2323231"分到一组，"3231"和"323231"分到一组。

对于每一组，我们检查这些后缀对应的sa值(也就是后缀起点在原串中的位置i)。如果 $\max\{sa\} - \min\{sa\} \geq k$ ，那么就说明我们能找出一组不重叠的重复子串。

例如对于k=3，"23231"和"2323231"的sa值是4和2，"3131"和"323231"这一组的sa值是5和3，差值都不满足大于等于3，所以找不出不重叠的。

对于k=2，第一组 $\max\{sa\} - \min\{sa\} = 6 - 2 = 4$ 满足大于等于2，所以能找出不重叠的。

```

bool check(int res)    //res <==> mid
{
    int minsa = INF;
    int maxsa = 0;
    for(int i=1;i<=n;i++)
    {
        if(height[i]<res){
            minsa = sa[i];
            maxsa = sa[i];
        }
        else{
            maxsa = max(maxsa,sa[i]);
            minsa = min(minsa,sa[i]);
            if(maxsa-minsa>=res) return true;
        }
    }
    return false;
}

```

应用4：不相同的子串的个数

给定一个字符串，求不相同的子串的个数。

- 一个字符串的某一子串一定是某一后缀的前缀
- 而且有一个很容易发现的性质
- 排完序后的后缀数组
- $\text{suffix}(sa[i]), \text{suffix}(sa[i+1]), \text{suffix}(sa[i+2]), \dots, \text{suffix}(sa[n])$
- 相邻两后缀之间增加的子串数量为 $(n-1)-sa[i]+1$ ，但是相邻两后缀之间又有公共前缀，所以 $\text{ans} = (n-1)-sa[i]+1-\text{height}[i]$ 依次累加即可

应用5：连续重复子串

倍增常数会被卡TLE，用KMP求最小循环节长度即可

```

void solve()
{
    int n = strlen(s);
    get_nex();
    int ans = -1;
    int mlen = n-nex[n];
    if(nex[n] == 0) ans = 1;
    else if(n%mlen == 0) ans = n/mlen;
    else if(n%mlen != 0) ans = 1;
    printf("%d\n",ans);
}

```

应用6：最长连续重复子串★

我们把一段字符串称为 (k,l) -重复的，如果它满足由一个长度为 l 的字符串重复了 k 次组成。如字符串abaabaabaaba是 $(4,3)$ -重复的，因为它由aba重复4次组成。

求最大的 k 。

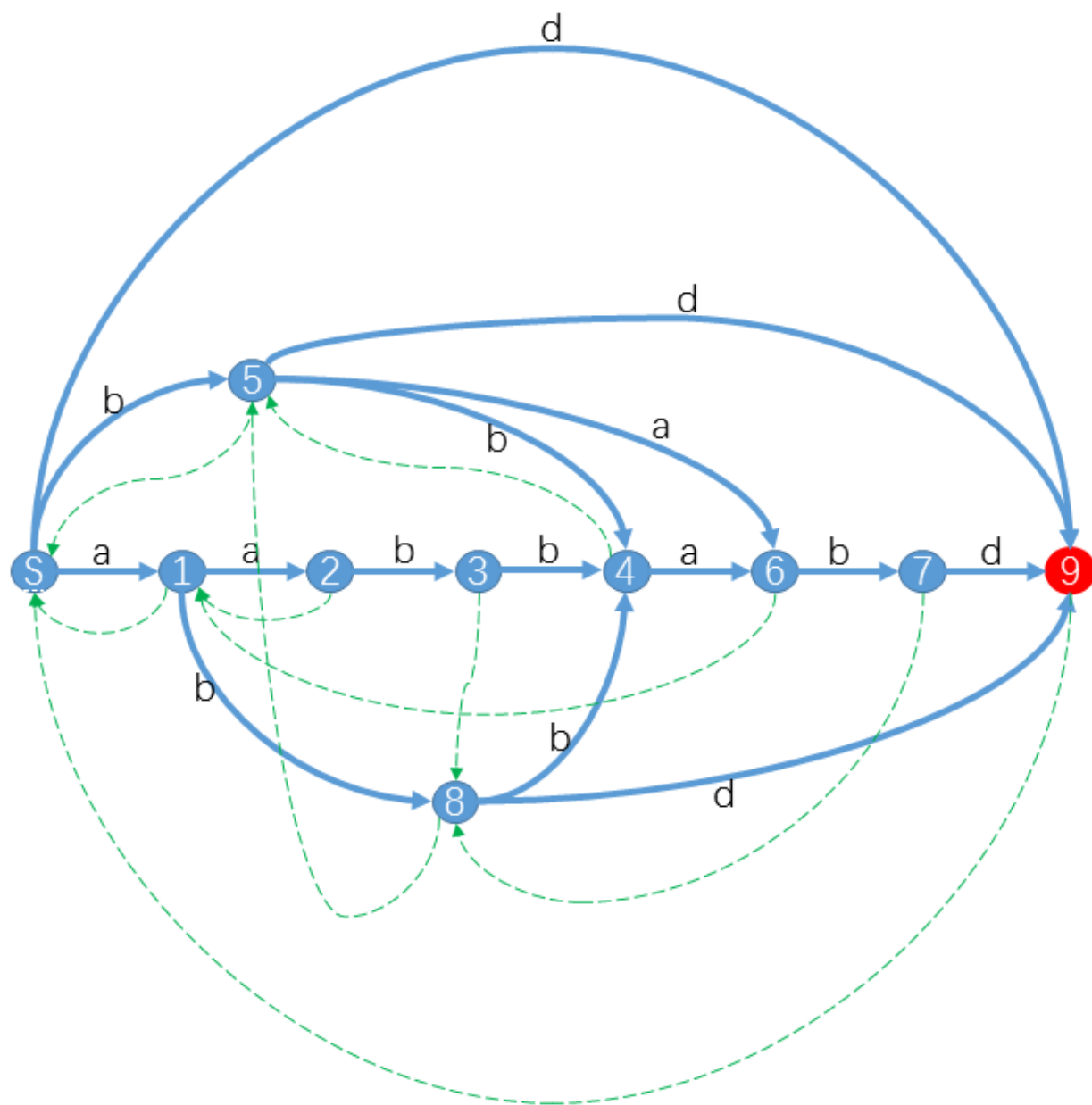
```

int dp[maxn][30];
void init_rmq()
{
    for(int i=1;i<=n;i++) dp[i][0] = height[i];
    for(int j=1;(1<<j)<=n;j++)
    {
        for(int i=1;i+(1<<j)-1<=n;i++)
        {
            dp[i][j] = min(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);
        }
    }
}
int query(int i,int j)
{
    int k = 0;
    int l = rk[i], r = rk[j];
    if(l>r) swap(l,r);
    l++;
    while(1<<(k+1)<=r-l+1) k++;
    return min(dp[l][k],dp[r-(1<<k)+1][k]);
}
int main()
{
    // fin;
    IO;
    cin>>s;
    n = strlen(s);
    build_sa(m);
    init_rmq();
    int ans = -1;
    for(int l = 1;l<=n;l++)
    {
        for(int i=0;i+l<n;i+=l)
        {
            int lcp = query(i,i+l); //后缀i与后缀i+l的最大公共前缀
            ans = max(ans,lcp/l+1); //lcp/l+1就是连续重复子串的重复次数,容易想
            if(i >= l-lcp%l) ans = max(query(i-(l-lcp%l),i+lcp%l)/l+1,ans);
        }
    }
    cout<<ans<<endl;
    return 0;
}

```

后缀自动机(SAM)

后缀自动机 (SuffixAutoMation) 属于后缀家族的一员。后缀自动机其实是一个DAG (有向无环图), 其中各个结点是状态, 边代表状态之间的转移。它是一个DFA (有穷自动机), 学过编译原理的童鞋应该会熟悉 我们把一个串建好它的自动机, 那么该自动机可以识别该串的所有后缀。自动机有一个S (初态), 一个E (终态)



状态	子串	endpos
S	空串	{0,1,2,3,4,5,6}
1	a	{1,2,5}
2	aa	{2}
3	aab	{3}
4	aabb,abb,bb	{4}
5	b	{3,4,6}
6	aabba,abba,bba,ba	{5}
7	aabbab,abbab,bbab,bab	{6}
8	ab	{3,6}
9	aabbabd,abbabd,bbabd,babd,abd,bd,d	{7}

https://blog.csdn.net/m0_37624640

首先知道两点：

1. 从S出发沿着蓝色实验路径转移，到达最终态，得到的是S的后缀。即S的后缀都可以从以S为起点，状态9（终态）为终点的路径中表示出来。
2. S的任一子串，都是以S为起点，终点为某一合法状态所表示出来的，因为串后缀的前缀是该串的子串。

对构造的一些理解：

1. **trans[st][] 数组**表示st的转移函数 比如last是上一状态，np是当前要加入的状态，x是新加入的字符 那么 trans[last][x] = np 表示状态last到np状态的转移
2. **slink[st] 数组**表示st的Suffix Link，即转移序列，图中的绿色虚线。比如 图中：7 ----> 8 ----> 5 ----> s。它的意义是状态7的最长子串aabbab的**后缀依次在状态7, 8, 5, s中**。用Suffix Link表示这一串状态连接起来。**slink数组相当于描述了状态（节点）之间的父子（祖孙）关系。即状态7中的所有串，都可以通过状态8转移过来。即祖先串一定是孩子串的后缀。**比如：上图状态8中的ab ab在前面增加a，----> aab 到达了状态7，aab中的后缀ab就是其父亲串。
3. SAM的构造离不开这两个关键数组，构造复杂度为O(length(s))。采用增量法进行构造，即一个个字符的加入自动机。

自己对构造代码的注解：

```

struct SuffixAutoMation
{
    int last,cnt;
    int trans[maxn<<1][26],slink[maxn<<1],l[maxn<<1];
    void add(int x)
    {
        //当前状态为np,上一状态为last, np的最长子串长度为last的最长子串长度+1,因为np是从last加了字符x
        转移过来的
        int p = last,np = ++cnt; last=np; l[np]=l[p]+1;
        //利用slink数组更新trans数组,即last所在的Suffix link序列中的后缀,都可转移至状态np
        for(;p&&!trans[p][x];p=slink[p]) trans[p][x] = np; //注意p的意义在此循环中已经变成了
        Suffix link序列中,last的上一后缀所在的状态
    }
}

```

```

//如果p为0,说明已走过了空串状态(1表示初态),初态的endpos集合肯定包含np的endpos集合
if(!p) slink[np] = 1;
//否则
else
{
    int q = trans[p][x];
    //如果q所包含的最长子串就是p的最长子串加上字符x,即maxlen[p]+1等于maxlen[q]
    //只要增加slink[np] = q即可
    if(l[p]+1 == l[q]) slink[np] = q;
    //否则
    else
    {
        //从上一状态中拆分出来新状态,把字符x及其后缀分给新状态, 其他子串留给上一状态
        int nq = ++cnt; l[nq]=l[p]+1;
        //同时更新trans和slink数组
        memcpy(trans[nq],trans[q],sizeof(trans[q]));
        slink[nq] = slink[q];
        slink[q] = slink[np] = nq;
        for(;trans[p][x] == q;p = slink[p]) trans[p][x] = nq;
    }
}
}
void build()
{
    scanf("%s",s+1);
    int len = strlen(s+1);
    last = cnt = 1;
    for(int i=1;i<=len;i++) add(s[i]-'a');
}
}sam;

```

//2018.10.23

slink以及l数组不需要每次初始化,因为再用时会覆盖赋值, slink[1] = l[1] = 0即可

初态s用1表示, slink[1] = 0, l[1] = 0

$l[np] - l[slink[np]]$ 表示从父亲态到孩子态所增加的子串数量

//2018.10.24

//观察parent数 父子结点的联系

//注: 拓扑排序后输出

```
for(int i=sam.cnt;i>=1;i--) cout<<sam.rk[i]-1<<" "<<sam.slink[sam.rk[i]]-1<<endl;
```

应用1: 不相同的子串的个数

```

return l[np] - l[slink[np]];
void build(){
    ll ans = 0;
    scanf("%s",s+1);
    int len = strlen(s+1);
    last = cnt = 1;
    for(int i=1;i<=len;i++){
        ans+=add(s[i]-'a');
    }
    printf("%lld\n",ans);
}

```

应用2: endpos集合

```

char s[maxn];
int num[maxn<<1]; //即每个状态的endpos
struct SuffixAutoMation
{
    int last,cnt; //cnt表示状态
    int trans[maxn<<1][26],slink[maxn<<1],l[maxn<<1];
    int t[maxn<<1],a[maxn<<1];
    inline void add(int x)
    {
        int p = last,np = ++cnt; last = np; l[np] = l[p]+1;
        for(;p && !trans[p][x];p=slink[p]) trans[p][x] = np;
        if(!p) slink[np] = 1;
        else{
            int q = trans[p][x];
            if(l[p]+1 == l[q]) slink[np] = q;
            else{
                int nq = ++cnt; l[nq] = l[p]+1;
                memcpy(trans[nq],trans[q],sizeof(trans[q]));
                slink[nq] = slink[q];
                slink[q] = slink[np] = nq;
                for(;trans[p][x] == q; p = slink[p]) trans[p][x] = nq;
            }
        }
        num[np] = 1;
    }
}
void build()
{
    scanf("%s",s+1);
    int len = strlen(s+1);
    last = cnt = 1;
    for(int i=1;i<=len;i++) add(s[i]-'a');
}
void topsort()
{
    int ans = 0;
    for(int i=1;i<=cnt;i++) t[l[i]]++;
    for(int i=1;i<=cnt;i++) t[i]+=t[i-1];
    for(int i=1;i<=cnt;i++) a[t[l[i]]--] = i;
}

```

```

        for(int i=cnt;i>=1;i--){ //在parent树上,自底向上跑
            int x = a[i]; //x指状态
            num[slink[x]]+=num[x];
        }
    }
}
}sam;

```

应用3：最长公共子串

- 后缀家族的基本应用，之前学习了后缀数组的解法，后缀自动机的思想如下：
- 给第一个串建立自动机，第二个串在自动机上匹配，通过tran数组，如果匹配就len+1, 否则通过slink数组 (fa) 向前跳，直到能继续匹配。

```

inline void init()
{
    root = cnt = last = 1;
    memset(trans,0,sizeof(trans));
    slink[1] = l[1] = 0;
}
inline int Find() //第二个串与第一个串匹配
{
    int m = strlen(b+1);
    int res = 0,ans = 0,p = root;
    for(int i=1;i<=m;i++)
    {
        int x = b[i]-'a';
        if(trans[p][x]) res++,p = trans[p][x];
        else{
            for(;p && !trans[p][x];p = slink[p]);
            if(!p) res = 0,p = root;
            else res = l[p]+1, p = trans[p][x];
        }
        ans = max(ans,res);
    }
    return ans;
}

```

应用4：多个串的最长公共子串

- 类似求两个串的最长公共子串。
- 我们对第一个串建立自动机，然后把剩余的n-1个串放进自动机上匹配。
- 每个串都保存它们在每个状态上的匹配的最大长度ml，然后对于每个状态，维护一个数组mn[p]，表示当前串在此状态的LCS。对于每个状态我们要取mn[p]中的最小值。然后答案就是所有状态中最小值的最大值。
- 注意，如果当前状态存在祖先，那么它的祖先的ml要更新其length，为什么呢？
- 因为孩子结点匹配过了，那么长度一定大于祖先的匹配长度。我们在匹配的过长中，如果只是找到了一个子串，可能遗漏了祖先没有匹配到，这样导致祖先的ml为0，在更新状态去min的时候，会去到0，这样就错了。所以我们可以把祖先的ml赋值为祖先的length。因为当前结点的length一定大于其祖先。

```

int ml[maxn<<1];
int mn[maxn<<1];
char s[maxn],b[maxn];

```

```

struct SuffixAutoMation{
    int root,cnt,last,len;
    int trans[maxn<<1][26],slink[maxn<<1],l[maxn<<1],num[maxn<<1];
    int c[maxn<<1],rk[maxn<<1];
    inline void init()
    {
        root = cnt = last = 1;
        memset(trans,0,sizeof(trans));
        slink[1] = l[1] = 0;
    }
    inline void add(int x)
    {
        int p = last,np = ++cnt;last = np;l[np] = l[p]+1;
        for(;p && !trans[p][x];p = slink[p]) trans[p][x] = np;
        if(!p) slink[np] = 1;
        else{
            int q = trans[p][x];
            if(l[p]+1 == l[q]) slink[np] = q;
            else{
                int nq = ++cnt;l[nq] = l[p]+1;
                memcpy(trans[nq],trans[q],sizeof(trans[q]));
                slink[nq] = slink[q];
                slink[q] = slink[np] = nq;
                for(;trans[p][x] == q;p = slink[p]) trans[p][x] = nq;
            }
        }
    }
    inline void build()
    {
        init();
        scanf("%s",s+1);
        len = strlen(s+1);
        for(int i=1;i<=len;i++) add(s[i]-'a');
    }
    inline void toposort()//拓扑排序是对每个状态的最长
    {
        for(int i=1;i<=cnt;i++) c[l[i]]++;
        for(int i=1;i<=len;i++) c[i]+=c[i-1];
        for(int i=1;i<=cnt;i++) rk[c[l[i]]--] = i;
    }
    inline void Find()
    {
        int m = strlen(b+1);
        int res = 0,ans = 0,p = root;
        for(int i=1;i<=m;i++)
        {
            int x = b[i]-'a';
            if(trans[p][x]) res++,p = trans[p][x];
            else{
                for(;p && !trans[p][x];p = slink[p]);
                if(!p) res = 0,p = root;
                else res = l[p]+1, p = trans[p][x];
            }
        }
    }
}

```

```

        ml[p] = max(ml[p],res);
    }
    for(int i=cnt;i>=1;i--)
    {
        int x = rk[i];
        mn[x] = min(mn[x],ml[x]);
        if(ml[x] && slink[x]) ml[slink[x]] = l[slink[x]];
        ml[x] = 0;
    }
}
}sam;
int main()
{
    // fin;
    // IO;
    memset(mn,INF,sizeof(mn)); //初始化为无限大
    sam.build();
    sam.toposort();
    while(~scanf("%s",b+1))
    {
        sam.Find();
    }
    int res = 0;
    for(int i=1;i<=sam.cnt;i++) res = max(res,mn[i]);
    printf("%d\n",res);
    return 0;
}

```

应用5：不同长度下出现次数最多的子串

```

inline void topsort()
{
    for(int i=1;i<=cnt;i++) c[l[i]]++;
    for(int i=1;i<=cnt;i++) c[i]+=c[i-1];
    for(int i=1;i<=cnt;i++) rk[c[l[i]]--] = i;
    for(int i=cnt;i>=1;i--) //自底向上更新
    {
        int x = rk[i];
        num[slink[x]]+=num[x];
        ans[l[x]] = max(ans[l[x]],num[x]);
    }
    for(int i=1;i<=len;i++) printf("%d\n",ans[i]);
}

```

应用6：原串S中出现次数在[A,B]之间的子串的个数

```

inline void topsort()
{
    for(int i=1;i<=cnt;i++) c[l[i]]++;
    for(int i=1;i<=cnt;i++) c[i]+=c[i-1];
    for(int i=1;i<=cnt;i++) rk[c[l[i]]--] = i;
}

```

```

for(int i=cnt;i>=1;i--) //自底向上更新
{
    int x = rk[i];
    num[slink[x]]+=num[x];
}
ll res = 0;
for(int i=1;i<=cnt;i++)
{
    if(num[i]>=L && num[i]<=R) res+=l[i]-l[slink[i]];
}
printf("%lld\n",res);
}

```

应用7：区间内不同子串的个数

```

char s[maxn];
struct SuffixAutoMation{
    int last,cnt,c[maxn<<1],rk[maxn<<1],tot;
    int trans[maxn<<1][26],slink[maxn<<1],l[maxn<<1];
    inline void init()
    {
        tot = 0;
        last = cnt = 1;
        memset(trans,0,sizeof(trans));
        slink[last] = l[last] = 0;
    }
    inline int add(int x)
    {
        int p = last,np = ++cnt;last = np;l[np] = l[p]+1;
        for(;p&&!trans[p][x];p=slink[p]) trans[p][x] = np;
        if(!p) slink[np] = 1;
        else{
            int q = trans[p][x];
            if(l[p]+1 == l[q]) slink[np] = q;
            else{
                int nq = ++cnt;l[nq] = l[p]+1;
                memcpy(trans[nq],trans[q],sizeof(trans[q]));
                slink[nq] = slink[q];
                slink[q] = slink[np] = nq;
                for(;trans[p][x] == q;p = slink[p]) trans[p][x] = nq;
            }
        }
        tot+=l[np]-l[slink[np]];
        return tot;
    }
}sam;
int main()
{
    int t;
    t = read();
    while(t--){

```



```

scanf("%s",s+1);
int len = strlen(s+1);
for(int i=1;i<=len;i++)//所有子区间都跑自动机
{
    sam.init();
    for(int j=i;j<=len;j++)
        ans[i][j] = sam.add(s[j]-'a');
}
int q; q = read();
while(q--)
{
    int l,r;
    l = read(); r = read();
    printf("%d\n",ans[l][r]);
}
}
return 0;
}

```

动态规划

背包

c费用 w价值 m数量 V最大体积

```

void ZeroOnePack(int c,int w) {
    for (int v=V;v>=c;--v)
        dp[v]=max(dp[v],dp[v-c]+w);
}
void CompletePack(int c,int w) {
    for (int v=c;v<=V;++v)
        dp[v]=max(dp[v],dp[v-c]+w);
}
void MultiplePack(int c,int w,int m) {
    if (c*m>=V) {
        CompletePack(c,w);
        return;
    }
    for (int k=1;k<m;m-=k,k<=1)
        ZeroOnePack(k*c,k*w);
    ZeroOnePack(m*c,m*w);
}

```

单调队列

```

void MultiplePack() { // 不等于-1的是可以组成的
    memset(dp,-1,sizeof dp); dp[0]=0;
    for (int i=0;i<n;++i)
        for (int v=0;v<=V;++v) {
            if (dp[v]>=0) dp[v]=m[i];
            else if (v>=c[i]) dp[v]=max(dp[v-c[i]]-1,-1);
        }
}

```

最长公共子序列 LCS

```

int dp[maxn][maxn];
int LCS(char s[],char t[]) { // 下标从1开始
    for (int i=1;s[i];++i)
        for (int j=1;t[j];++j) {
            if(s[i]==t[j])
                dp[i][j]=dp[i-1][j-1]+1;
            else
                dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
        }
    return dp[strlen(s+1)][strlen(t+1)];
}

```

最长上升子序列 LIS

```

int LIS() { // 最长上升子序列
    memset(dp,0x3f,sizeof dp);
    for (int i=1;i<=n;++i)
        *lower_bound(dp,dp+n,a[i])=a[i];
    return lower_bound(dp,dp+n,inf)-dp;
}
int LNDS() { // 最长不下降子序列
    memset(dp,0x3f,sizeof dp);
    for (int i=1;i<=n;++i)
        *upper_bound(dp,dp+n,a[i])=a[i];
    return lower_bound(dp,dp+n,inf)-dp;
}

```

最长递增公共子序列 LICS

[ZOJ2432](#)

```

#include<cmath>
#include<cstdio>
#include<cstring>
const int qq=505;
int a[qq],b[qq];
int dp[qq][qq];
int prex[qq][qq];

```

```

int prey[qq][qq];
int count,ans;
void out(int x,int y) // 递归输出路径、
{
    if(dp[x][y]==0) return;
    int xx=prex[x][y];
    int yy=prey[x][y];
    out(xx,yy);
    if(dp[x][y]!=dp[xx][yy] && y!=0){
        printf("%d",b[y]);
        count++;
        if(count<ans) printf(" ");
        else printf("\n");
    }
}
int main()
{
    int t;scanf("%d",&t);
    while(t--){
        int n;
        scanf("%d",&n);
        for(int i=1;i<=n;++i)
            scanf("%d",&a[i]);
        int m;
        scanf("%d",&m);
        for(int i=1;i<=m;++i)
            scanf("%d",&b[i]);
        memset(prex,-1,sizeof(prex));
        memset(prex,-1,sizeof(prex));
        memset(dp,0,sizeof(dp));
        for(int j,i=1;i<=n;++i){
            int maxn=0;
            int x,y;x=y=0;
            for(j=1;j<=m;++j){
                dp[i][j]=dp[i-1][j]; // 先更新状态, 如果后面会更新的话再去更新
                prex[i][j]=i-1;
                prey[i][j]=j;
                if(a[i]>b[j] && maxn<dp[i-1][j]){
                    // 我开始一直没想通为什么要在a[i]>b[j]的时候才更新值
                    maxn=dp[i-1][j]; // 其实你想想dp[i][j]中i,j都有什么含义、
                    x=i-1; // 这个if里面更新出来的maxn只有在满足a[i]==b[j]的时候才有用
                    y=j; // 这里更新出来的值就是为了保证当a[i]==b[j]的时候
                } // 所更新出来的最大值是一个递增的子序列
            }
            else if(a[i]==b[j]){
                dp[i][j]=maxn+1;
                prex[i][j]=x;
                prey[i][j]=y;
            }
        }
    }
    for(int i=1;i<=n;++i){
        for(int j=1;j<=m;++j)
            printf("%d ",dp[i][j]);
    }
}

```

```

        printf("\n");
    }
    ans=0;
    int flag=-1;
    for(int i=1;i<=m;++i){
        if(ans<dp[n][i]){
            flag=i;
            ans=dp[n][i];
        }
    }
    printf("%d\n",ans);
    int x=n,y=flag;
    count=0;
    if(ans>0)
        out(x,y);
    if(t)    printf("\n");
}
return 0;
}

```

整数拆分

拆分：把正整数分解成若干正整数的和。

拆分数：不同拆分法的总数。

定理：重复次数不超过k次的拆分数 等于 分解数中无k+1倍数的拆分数。

Ferrers图像

定理：拆分成k个数的拆分数 等于 拆分成最大数为k的拆分数。

定理：重复次数不超过k次的拆分数 等于 拆分成最大数不超过k的拆分数。

定理：拆分互不相同若干奇数的拆分数 等于 拆分成自共轭的Ferrers图像的拆分数。

定理：N拆分成不超过k个数的拆分数 等于 N+k拆分成k个数的拆分数。

```

void solve(int n,int m) { // n的m划分数
    dp[0][0]=1;
    for (int i=1;i<=m;++i)
        for (int j=0;j<=n;++j) {
            if (j<i) // 1的(2.3.4.....)划分数 = 1的1划分数
                dp[i][j]=dp[i-1][j];
            else // 最后的+1和 每个+1
                dp[i][j]=dp[i-1][j]+dp[i][j-i];
        }
}

```

1.整数N拆分成k个数的拆分数

区间

```
memset(dp,0x3f,sizeof(dp));
for (int i=1;i<=n;++i) // 区间长度为1的初始化
    dp[i][i]=0;
for (int len=2;len<=n;++len) // 枚举区间长度
    for (int i=1,j=len;j<=n;++i,++j) {}
```

状压

```
for (int i=0;i<(1<<n);++i) {
    for (int j=0;j<n;++j)
        dp[i^(1<<j)]=max(dp[i^(1<<j)],/*calc*/);
}
for (in s0=s;s0;s0=s0-1&s); // 枚举子集
```

数位

hdu4507

定义和7有关的数

1. 某一位是7
2. 每一位加起来的和是7的倍数
3. 这个数是7的倍数

统计区间内和7无关的数字的平方和

$$\sum_i (a + b_i)^2 = \sum_i (a^2 + 2ab_i + b_i^2) = a^2 x + 2ax \sum_i b + \sum_i b_i^2$$

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mod = 1e9+7;
#define mk make_pair
struct node {
    ll num,sum,ans;
    node() {}
    node(int num,ll sum,ll ans):num(num),sum(sum),ans(ans) {}
} dp[20][10][10];
ll a[20],p[25];
node dfs(int pos,int sta1,int sta2,bool lim) {
    if(!pos) return sta1!=0&&sta2!=0?node(1,0,0):node(0,0,0);
    if(!lim&&dp[pos][sta1][sta2].num!=-1) return dp[pos][sta1][sta2];
    int up=lim?a[pos]:9;
    node res=node(0,0,0);
    for(int i=0;i<=up;i++) {
        if(i==7) continue;
        node tmp=dfs(pos-1,(sta1+i)%7,(sta2*10+i)%7,lim&&i==up);
        res.num=(res.num+tmp.num)%mod;
        res.sum=(res.sum+tmp.sum+p[pos]*i%mod*tmp.num%mod)%mod;
        res.ans=(res.ans+tmp.ans+2*p[pos]*i%mod*tmp.sum%mod)%mod;
        res.ans=(res.ans+tmp.num*p[pos]%mod*p[pos]%mod*i*i%mod)%mod;
    }
    return res;
}
```

```

    }
    if(!lim) dp[pos][sta1][sta2]=res;
    return res;
}
ll f(ll x) {
    int pos=0;
    while(x) {
        a[++pos]=x%10;
        x/=10;
    }
    return dfs(pos,0,0,true).ans;
}
int main() {
    int T;
    scanf("%d",&T);
    memset(dp,-1,sizeof dp);
    p[1]=1; for(int i=2;i<=20;++i) p[i]=p[i-1]*10%mod;
    while (T--) {
        ll l,r,ans;
        scanf("%lld%lld",&l,&r);
        ans=(f(r)-f(l-1))%mod+mod;
        printf("%lld\n",ans%mod);
    }
    return 0;
}

```

树形

重心

树: A[B[C],D]

1 A=>2 AB=>3 ABC=>2 AB ABC=>1 A AB ABC=>4 AD ABD ABCD=>1 A AB AD ABC ABD ABCD

```

int n,m,a[3005];
bitset<100005> b[3005],ans;
int rt,dp[3005],sz[3005]; bool used[3005];
void get_g(int u,int f) { // 找重心
    dp[u]=0; sz[u]=1;
    for (int i=head[u];~i;i=e[i].next) {
        int v=e[i].v;
        if (used[v]||v==f) continue;
        get_g(v,u);
        dp[u]=max(dp[u],sz[v]);
        sz[u]+=sz[v];
    }
    dp[u]=max(dp[u],dp[0]-sz[u]);
    if (dp[u]<dp[rt]) rt=u;
}
void get_v(int u,int f) { // 计算子树值
    b[u]<=a[u];
    sz[u]=1;
}

```

```

    for (int i=head[u];~i;i=e[i].next) {
        int v=e[i].v;
        if (used[v]||v==f) continue;
        b[v]=b[u];
        get_v(v,u);
        b[u]|=b[v];
        sz[u]+=sz[v];
    }
}

void solve(int u) {
    used[u]=1;
    b[u]=1;
    get_v(u,0);
    ans|=b[u];
    for (int i=head[u];~i;i=e[i].next) {
        int v=e[i].v;
        if (used[v]) continue;
        dp[0]=sz[v]; rt=0; get_g(v,u);
        solve(rt);
    }
}

int main() {
    int T;
    scanf("%d",&T);
    while (T--) {
        init_edge();
        ans.reset();
        memset(used,0,sizeof used);
        scanf("%d%d",&n,&m);
        for (int i=1,u,v;i<n;++i) {
            scanf("%d%d",&u,&v);
            add_edge(u,v);
            add_edge(v,u);
        }
        for (int i=1;i<=n;++i) scanf("%d",&a[i]);
        dp[0]=n; rt=0; get_g(1,0);
        solve(rt);
        for (int i=1;i<=m;++i) printf("%d",(int)ans[i]);
        printf("\n");
    }
    return 0;
}

```

基环树

```

const int MAX = 1e6 + 5;
const int mod = 1e9 + 7;
const int inf = 0x3f3f3f3f;

int size, head[MAX];
struct Node{ int to, next, w; } E[MAX<<1];

```

```

inline void init(){ size = 0; memset(head, -1, sizeof(head)); }
inline void add(int u, int v,int w) {
    E[size].to = v;          E[size].w = w;
    E[size].next = head[u];  head[u] = size++;
}
int n,m,vis[MAX];
ll dp[MAX][3],a[MAX],ans;
int p1,p2,fla,cnt;
void dfs(int u,int f)
{
    vis[u] = 1;
    for(int e=head[u];~e;e=E[e].next)
    {
        int v = E[e].to;
        if(v == f) continue;
        if(!vis[v]) dfs(v,u);
        else p1 = v,p2 = u,fla = e;
    }
}
void solve(int u,int f)
{
    dp[u][1] = a[u];dp[u][0] = 0;
    for(int e=head[u];~e;e=E[e].next)
    {
        int v = E[e].to;
        if(v == f) continue;
        if(fla == e || (fla ^ 1) == e) continue;
        solve(v,u);
        dp[u][1] += dp[v][0];
        dp[u][0] += max(dp[v][1],dp[v][0]);
    }
}

int main()
{
    scanf("%d",&n);
    init();
    for(int u,v,i=1;i<=n;++i)
    {
        scanf("%lld%d",&a[i],&u);
        add(i,u,0); add(u,i,0);
    }
    for(int i=1;i<=n;++i)
    {
        if(vis[i]) continue;
        dfs(i,-1);
        solve(p1,-1); ll t1 = dp[p1][0];
        solve(p2,-1); ll t2 = dp[p2][0];
        ans += max(t1,t2);
    }
    printf("%lld\n",ans);
    return 0;
}

```


多线程

从矩阵左上角走到右下角(每次只能走右、下), 再走回来(每次只能走左、上)的最大收益

k表示行列坐标和, i代表第一个人走到第几行, j代表第二个人走到第几行

```
int n, m;
int a[maxn][maxn], dp[2][maxn][maxn];
int solve() {
    int cnt=0;
    for (int k=2; k<=n+m; ++k) {
        cnt^=1;
        for (int i=1; i<=n&& i<k; ++i)
            for (int j=i; j<=n&& j<k; ++j) {
                dp[cnt][i][j]=max(max(dp[cnt^1][i][j], dp[cnt^1][i-1][j-1]),
                                   max(dp[cnt^1][i-1][j], dp[cnt^1][i][j-1]));
                if (i==j) dp[cnt][i][j]+=a[i][k-i];
                else dp[cnt][i][j]+=a[i][k-i]+a[j][k-j];
            }
    }
    return dp[cnt][n][n];
}
```

趣题

[NAIPC2018 Flashing Fluorescents](#)

题意:

灯: $n \leq 16$, 亮 | 暗状态, 有按钮, 按下按钮以步长1切换状态

例如: t秒按下灯i的按钮 \Rightarrow t+1灯i变, t+2灯i+1变, t+3灯i+2变.....

问: 灯全亮的最早时间

题解:

注意到答案不超过n, 枚举答案ans, 则可行方案可以由若干个长度互不相等且不超过ans的区间异或得到。

设f[ans][S]表示长度不超过ans能否异或出S, 枚举当前长度的区间位置转移即可。

```
#include<bits/stdc++.h>
using namespace std;
int n, now, S;
char s[20];
bool f[20][1<<16];
int main() {
    scanf("%s", s); n=strlen(s);
    for (int i=0; i<n; ++i) if (s[i]=='0') S^=1<<i; // 01互换, 现在推是否全灭
    f[0][S]=1;
    while (!f[now][0]) {
        for (S=0; S<1<<n; ++S) f[now+1][S]=f[now][S];
        for (int i=0; i<n; ++i) {
            int mask=0;
```

```

        for (int j=0;j<=now&& i+j<n;++j) mask|=1<<(i+j);
        for (s=0;s<1<<n;++s) f[now+1][s^mask]=f[now][s];
    }
    now++;
}
printf("%d\n",now);
return 0;
}

```

数据结构

排序

归并排序

```

int a[maxn],tmp[maxn];
inline void mergearr(int a[],int l,int m,int r) {
    int i=l,j=m+1,k=0;
    while (i<=m&&j<=r) tmp[k++]=a[i]<a[j]?a[i++]:a[j++];
    while (i<=m) tmp[k++]=a[i++];
    while (j<=r) tmp[k++]=a[j++];
    for (i=0;i<k;i++) a[l+i]=tmp[i];
}
void mergesort(int a[],int l,int r) {
    if (l>=r) return;
    int m=l+r>>1;
    mergesort(a,l,m);
    mergesort(a,m+1,r);
    mergearr(a,l,m,r,tmp); // 可以全局创建个tmp数组
}

```

快速排序

```

void quicksort(int a[],int l,int r) {
    int tmp=a[l],i=l,j=r;
    while (i<j) {
        while (i<j&& a[j]>=tmp) j--; a[i]=a[j];
        while (i<j&& a[i]<=tmp) i++; a[j]=a[i];
    }
    a[i]=tmp;
    quicksort(a,l,i-1);
    quicksort(a,i+1,r);
}

```

堆排序

离散化

```
int id[maxn], numid; // [1, numq] 不要零, 防止树无限递归
inline int getid(int x) { return lower_bound(id+1, id+1+numid, x) - id; }

for (int i=1; i<=n; ++i) id[++numid]=a[i];
sort(id+1, id+1+numid);
numid=unique(id+1, id+1+numid)-id-1;
```

并查集

```
int fa[maxn]; // 初始化为-1
inline int Find(int x) { return fa[x]==-1?x:fa[x]=Find(fa[x]); }
inline void Unite(int x, int y) { x=Find(x); y=Find(y); if(x!=y) fa[y]=x; }
```

把n个点分成两个集合，m个两点间关系，让关系中两点处于同一集合的最大权值最小

```
int n, m, fa[maxn];
struct node { int u, v, w; } e[100005];
bool cmp(node a, node b) { return a.w > b.w; }
inline int Find(int x) { return fa[x]==-1?x:fa[x]=Find(fa[x]); }
inline void Union(int x, int y) { x=Find(x); y=Find(y); if(x!=y) fa[y]=x; }
int main() {
    memset(fa, -1, sizeof fa);
    scanf("%d%d", &n, &m);
    for (int i=1; i<=m; i++) scanf("%d%d%d", &e[i].u, &e[i].v, &e[i].w);
    sort(e+1, e+1+m, cmp);
    for (int i=1; i<=m; i++) {
        int fu=Find(e[i].u);
        int fv=Find(e[i].v);
        if (fu==fv) {
            printf("%d\n", e[i].w);
            return 0;
        }
        Union(fu, e[i].v+n);
        Union(fv, e[i].u+n);
    }
    printf("0\n");
    return 0;
}
```

RMQ-ST表

```
int a[maxn], len[maxn], maxv[maxn][20], maxp[maxn][20], minv[maxn][20], minp[maxn][20];
inline void init_rmq(int n) {
    for (int i=2; i<=n; ++i) len[i]=len[i-1]+!(i&i-1); // 从2开始2^k时加1
    for (int i=1; i<=n; ++i) maxv[i][0]=a[i], maxp[i][0]=i;
    for (int i=1; i<=n; ++i) minv[i][0]=a[i], minp[i][0]=i;
    for (int j=1; j<=len[n]; ++j)
        for (int i=1; i+(1<<j)-1<=n; ++i) {
            maxv[i][j]=max(maxv[i][j-1], maxv[i+(1<<(j-1))][j-1]);
            minv[i][j]=min(minv[i][j-1], minv[i+(1<<(j-1))][j-1]);
        }
}
```

```

        maxp[i][j]=maxv[i][j]==maxv[i][j-1]?maxp[i][j-1]:maxp[i+(1<<(j-1))][j-1];
        minv[i][j]=min(minv[i][j-1],minv[i+(1<<(j-1))][j-1]);
        minp[i][j]=minv[i][j]==minv[i][j-1]?minp[i][j-1]:minp[i+(1<<(j-1))][j-1];
    }
}
inline int rmq_maxv(int x,int y) {
    int k=len[y-x+1]; y--=(1<<k)-1;
    return max(maxv[x][k],maxv[y][k]);
}
inline int rmq_maxp(int x,int y) {
    int k=len[y-x+1]; y--=(1<<k)-1;
    return maxv[x][k]>=maxv[y][k]?maxp[x][k]:maxp[y][k];
}
inline int rmq_minv(int x,int y) {
    int k=len[y-x+1]; y--=(1<<k)-1;
    return min(minv[x][k],minv[y][k]);
}
inline int rmq_minp(int x,int y) {
    int k=len[y-x+1]; y--=(1<<k)-1;
    return minv[x][k]<=minv[y][k]?minp[x][k]:minp[y][k];
}

```

二维ST表

```

int a[maxn][maxn],len[maxn];
int dp[maxn][maxn][10][10];
inline void init_rmq(int n,int m) {
    for (int i=2;i<=max(n,m);++i) len[i]=len[i-1]+!(i&i-1);
    for (int i=1;i<=n;++i)
        for (int j=1;j<=m;++j)
            dp[i][j][0][0]=a[i][j];
    for (int ii=0;ii<=len[n];++ii)
        for (int jj=0;jj<=len[m];++jj) if (ii+jj)
            for (int i=1;i+(1<<ii)-1<=n;++i)
                for (int j=1;j+(1<<jj)-1<=m;++j)
                    if (ii) dp[i][j][ii][jj]=max(dp[i][j][ii-1][jj],
                                                    dp[i+(1<<(ii-1))][j][ii-1][jj]);
                    else dp[i][j][ii][jj]=max(dp[i][j][ii][jj-1],
                                                    dp[i][j+(1<<(jj-1))][ii][jj-1]);
}
inline int rmp(int x1,int x2,int y1,int y2) {
    int k1=len[x2-x1+1]; x2--=(1<<k1)-1;
    int k2=len[y2-y1+1]; y2--=(1<<k2)-1;
    return max(max(dp[x1][y1][k1][k2],dp[x1][y2][k1][k2]),
                max(dp[x2][y1][k1][k2],dp[x2][y2][k1][k2]));
}

```

分块

```

int n,b1,sz,be[maxn],l[maxn],r[maxn]; // 属于哪块及该块的左右端点
inline void build(int n) {

```

```

    bl=sqrt(n);
    sz=n/bl+(n%bl!=0);
    for (int i=1;i<=sz;++i) l[i]=(i-1)*bl+1,r[i]=i*bl; r[sz]=n;
    for (int i=1;i<=n;++i) be[i]=(i-1)/bl+1;
}
inline void update(int x,int y) {
    for (int i=x;i<=min(r[be[x]],y);++i) /*---*/;
    if (be[x]!=be[y]) {
        for (int i=l[be[y]];i<=y;++i) /*---*/;
    }
    for (int i=be[x]+1;i<=be[y]-1;++i) /*---*/;
}
inline int query(int x,int y) {
    int ans=0;
    for (int i=x;i<=min(r[be[x]],y);++i) /*---*/;
    if (be[x]!=be[y]) {
        for (int i=l[be[y]];i<=y;++i) /*---*/;
    }
    for (int i=be[x]+1;i<=be[y]-1;++i) /*---*/;
    return ans;
}

```

莫队

```

int n,m,be[maxn],bl;
struct node { int l,r,idx; } q[maxn];
inline bool cmp(node a,node b) { return be[a.l]==be[b.l]?a.r<b.r:a.l<b.l; }
void init() {
    scanf("%d%d",&n,&m);
    bl=sqrt(n);
    for (int i=1;i<=n;++i) be[i]=(i-1)/bl+1;
    for (int i=1;i<=m;++i)
        scanf("%d%d",&q[i].l,&q[i].r),q[i].idx=i;
    sort(q+1,q+1+m,cmp);
}

```

字典树

```

char s[15];
int t[maxn][26],num[maxn],sz;
inline void add(char*s,int rt=0) {
    for (;*s;++s) {
        int ch=*s-'a';
        if (!t[rt][ch]) t[rt][ch]=++sz;
        rt=t[rt][ch];
    }
    ++num[rt];
}
inline int query(char *s,int rt=0) {
    for (;*s;++s) {
        int ch=*s-'a';

```

```

        if (!t[rt][ch]) return 0;
        rt=t[rt][ch];
    }
    return num[rt];
}

```

笛卡尔树

1. 关键字符合二叉搜索树性质(一般关键字为数组下标)
2. 权值满足堆的性质

```

int sta[maxn],top;
int lson[maxn],rson[maxn],vis[maxn];
inline int build(int a[],int n) {
    top=0;
    for (int i=1;i<=n;++i) lson[i]=rson[i]=vis[i]=0;
    for (int i=1;i<=n;++i) {
        int k=top;
        while (k&& a[i]<a[sta[k-1]]) --k; //<小根堆
        if (k) rson[sta[k-1]]=i;
        if (k<top) lson[i]=sta[k];
        sta[k++]=i;
        top=k;
    }
    for (int i=1;i<=n;++i) vis[lson[i]]=vis[rson[i]]=1;
    int rt=0;
    for (int i=1;i<=n;++i) if (!vis[i]) rt=i;
    return rt;
}

```

BZOJ2616 笛卡尔树+树形DP

直线上有 n 个高为 $h[i]$ 的小棋盘组成的畸形棋盘，若两个车相互攻击当且仅当它们在同一列或在同一行且这一行之间的棋盘格子都存在，问在棋盘防止 K 个互不攻击的车的方案数

$n \leq 500, k \leq 500, h[i] \leq 1000000$

```

#include<bits/stdc++.h>
typedef long long ll;
using namespace std;
const int mod=1e9+7;
const int maxn=5e2+5;
const int maxm=1e6+5;
/* 处理逆元 */
ll fac[maxn],inv[maxn];
inline ll pow_mod(ll a,ll b) {
    ll res=1;
    while (b) {
        if (b&1) res=res*a%mod;
        a=a*a%mod;
        b>>=1;
    }
}

```

```

    return res;
}
inline void init_fac_inv(int n) {
    fac[0]=1;
    for (int i=1;i<=n;++i) fac[i]=fac[i-1]*i%mod;
    inv[n]=pow_mod(fac[n],mod-2);
    for (int i=n-1;i>=0;--i) inv[i]=inv[i+1]*(i+1)%mod;
}
/* 建笛卡尔树 */
int sta[maxn],top;
int lson[maxn],rson[maxn],vis[maxn];
inline int build(int a[],int n) {
    top=0;
    for (int i=1;i<=n;i++) lson[i]=rson[i]=vis[i]=0;
    for (int i=1;i<=n;i++) {
        int k=top;
        while (k&& a[i]<a[sta[k-1]]) --k; //小根堆
        if (k) rson[sta[k-1]]=i;
        if (k<top) lson[i]=sta[k];
        sta[k++]=i;
        top=k;
    }
    for (int i=1;i<=n;i++) vis[lson[i]]=vis[rson[i]]=1;
    int rt=0;
    for (int i=1;i<=n;i++) if (!vis[i]) rt=i;
    return rt;
}

int n,k,a[maxn],sz[maxn];
ll g[maxn],f[maxn][maxn];
inline ll getc(int n,int m) {
    if (m>n||m<0) return 0;
    return fac[n]*inv[m]%mod*inv[n-m] % mod;
}
void dfs(int u,int val) {
    int h=a[u]-val; sz[u]=1;
    if (lson[u]) dfs(lson[u],a[u]); sz[u]+=sz[lson[u]];
    if (rson[u]) dfs(rson[u],a[u]); sz[u]+=sz[rson[u]];
    memset(g,0,sizeof g);
    for (int i=0;i<=sz[u];++i)
        for (int j=0;j<=i;++j)
            g[i]=(g[i]+f[lson[u]][j]*f[rson[u]][i-j]%mod)%mod;
    for (int i=0;i<=sz[u];++i)
        for (int j=0;j<=i;++j)
            f[u][i]=(f[u][i]+g[i-j]*fac[j]%mod*getc(h,j)%mod*getc(sz[u]-i+j,j)%mod)%mod;
}
int main() {
    scanf("%d%d",&n,&k);
    init_fac_inv(1000000);
    for (int i=1;i<=n;++i) scanf("%d",&a[i]);
    int rt=build(a,n);
    f[0][0]=1; dfs(rt,0);
    printf("%lld\n",f[rt][k]);
}

```

```

    return 0;
}

```

树状数组

区间修改 区间求和

1. 原数组 $a[i]$, 差分数组 $c[i]=a[i]-a[i-1]$, 辅助数组 $d[i]=(i-1)*c[i]$
2. $a_1 + a_2 + \dots + a_n$
 $= c_1 + (c_1 + c_2) + \dots + (c_1 + c_2 + c_3 + \dots + c_n)$
 $= n * (c_1 + c_2 + \dots + c_n) - (0 * c_1 + 1 * c_2 + \dots + (n-1) * c_n)$
 $= n * query(c, n) - query(d, n)$

```

int n;
ll a[maxn]; //原数组
ll c[maxn]; //差分数组
ll d[maxn]; //辅助数组
inline void add(ll t[],int p,ll v) { for (;p<=n;p+=p&-p) t[p]+=v; }
inline ll query(ll t[],int p) { ll res=0; for (;p;p-=p&-p) res+=t[p]; return res; }
inline void add_xy(int x,int y,ll v) { //a[x]...a[y] += v
    add(c,x,v); add(c,y+1,-v);
    add(d,x,(x-1)*v); add(d,y+1,y*-v);
}
inline ll query_xy(int x,int y) {
    ll sum1=(x-1)*query(c,x-1)-query(d,x-1);
    ll sum2=y*query(c,y)-query(d,y);
    return sum2-sum1;
}
int main() {
    scanf("%d",&n);
    for (int i=1;i<=n;++i) scanf("%lld",&a[i]);
    for (int i=1;i<=n;++i) { //初始化数组
        add(c,i,a[i]-a[i-1]);
        add(d,i,(i-1)*(a[i]-a[i-1]));
    }
    query(c,p); //单点查询
    return 0;
}

```

单点修改 块求和

```

int n;
ll a[1005][1005];
inline void add(int x,int y,int v) {
    for(;x<=n;x+=x&-x)
        for(;y<=n;y+=y&-y)
            a[x][y]+=v;
}
inline ll query(int x,int y) {

```



```

11 res=0;
for(;x;x-=x&-x)
    for(;y;y-=y&-y)
        res=(res+a[x][y])%mod;
return res;
}
add(x,y,v)
query(x2,y2)-query(x2,y1-1)-query(x1-1,y2)+query(x1-1,y1-1)

```

树状数组二分

$$\sum_{i=x-(x\&-x)+1}^x a[i] = sum[x]$$

普通平衡树 bzoj3224

1插入x 2删除x 3查询x的排名 4查询排名为x的数 5求x的前驱 6求x的后继

```

#include<cstdio>
const int maxn=1<<25; // 大于数据范围
const int off=1e7+10; // 偏移量 数据范围[-10^7,10^7]
int a[maxn];
inline void add(int p,int v) { for(;p<maxn;p+=p&-p) a[p]+=v; }
inline int query(int p) { int res=0; for(;p;p-=p&-p) res+=a[p]; return res; }
inline int kth(int k,int rt=maxn){
    for(int i=rt;i>=1)
        if(a[rt-i]>=k) rt-=i;
        else k-=a[rt-i];
    return rt-off;
}
int main(){
    int n,op,x;
    scanf("%d",&n);
    for (int i=1;i<=n;++i) {
        scanf("%d%d",&op,&x);
        if (op==1) add(x+off,1);
        if (op==2) add(x+off,-1);
        if (op==3) printf("%d\n",query(x+off-1)+1);
        if (op==4) printf("%d\n",kth(x));
        if (op==5) printf("%d\n",kth(query(x+off-1)));
        if (op==6) printf("%d\n",kth(query(x+off)+1));
    }
}

```

线段树

```

#define lson rt<<1,l,m
#define rson rt<<1|1,m+1,r

```

1. push

push_down把懒惰标记更新到子节点的值数组和懒惰数组上.

```
inline void push_up(int rt) { }
inline void push_down(int rt,int l,int r) { if (/*降常数*/) }
```

2.build

```
void build(int rt,int l,int r) {
    if (l==r) { /* 与a[l]相关? */ return; }
    int m=(l+r)>>1;
    build(lson);
    build(rson);
    push_up(rt);
}
```

3.update

```
//单点更新
void update(int rt,int l,int r,int p,int v) {
    if (l==r) { /*calc*/ return; }
    push_down(rt,l,r);
    int m=(l+r)>>1;
    if (p<=m) update(lson,p,v);
    else      update(rson,p,v);
    push_up(rt);
}

//区间更新
void update(int rt,int l,int r,int L,int R,int v) {
    if (L<=l&&R<=r) { /*calc*/ return; }
    push_down(rt,l,r);
    int m=(l+r)>>1;
    if (L<=m) update(lson,L,R,v);
    if (m<R)   update(rson,L,R,v);
    push_up(rt);
}
```

4.query

```
//单点查询
ll query(int rt,int l,int r,int p) {
    if (l==r) return /*data*/;
    push_down(rt,l,r);
    int m=(l+r)>>1;
    if (p<=m) return query(lson,p);
    return query(rson,p);
}

//区间查询
ll query(int rt,int l,int r,int L,int R) {
    if (L<=l&&R<=r) return /*data*/;
```

```

push_down(rt,l,r);
int m=(l+r)>>1;
if (L<=m) /*calc*/;
if (m<R)  /*calc*/;
return /*data*/;
}

```

5.常用操作

- 区间修改, 区间求和, 区间最值

```

inline void push_up(int rt) {
    sum[rt]=sum[rt<<1]+sum[rt<<1|1];
    maxv[rt]=max(maxv[rt<<1],maxv[rt<<1|1]);
}
inline void push_down(int rt,int l,int r) {
    if (!lz[rt]) {
        int m=(l+r)>>1;
        sum[rt<<1]+=(m-l+1)*lz[rt];
        sum[rt<<1|1]+=(r-m)*lz[rt];
        maxv[rt<<1]+=lz[rt];
        maxv[rt<<1|1]+=lz[rt];
        lz[rt<<1]+=lz[rt];
        lz[rt<<1|1]+=lz[rt];
        lz[rt]=0;
    }
}
void updata(int rt,int l,int r,int L,int R,int v) {
    if (L<=l&&r<=R) { sum[rt]+=(r-l+1)*v; maxv[rt]+=v; lz[rt]+=v; return; }
}

```

- 区间合并POJ3667

开始n个空房间,m个操作,操作分两种

1. 查询:查询是否存在长度为x的连续空房,输出左端点,找不到为0.
2. 退房:将编号为[x,x+y-1]的房间退房.

```

#include<cstdio>
#include<algorithm>
#define lson rt<<1,l,m
#define rson rt<<1|1,m+1,r
using namespace std;
//0代表有连续空间, 1代表已经占用, -1代表子区间不全相同
int n,m,op,x,y;
int lsum[1<<17],rsum[1<<17],msum[1<<17],lz[1<<17];

inline void push_up(int rt,int l,int r) {
    int m=(l+r)>>1;
    lsum[rt]=lsum[rt<<1];    rsum[rt]=rsum[rt<<1|1];
    if (lsum[rt]==m-l+1)    lsum[rt]+=lsum[rt<<1|1];
    if (rsum[rt]==r-m)      rsum[rt]+=rsum[rt<<1];
    msum[rt]=max(msum[rt<<1],msum[rt<<1|1]);
}

```

```

    msum[rt]=max(msum[rt],rsum[rt<<1]+lsum[rt<<1|1]);
}
inline void push_down(int rt,int l,int r) {
    if (lz[rt]==-1) return;
    int m=(l+r)>>1;
    lsum[rt<<1]=rsum[rt<<1]=msum[rt<<1]= lz[rt]?0:m-l+1;
    lsum[rt<<1|1]=rsum[rt<<1|1]=msum[rt<<1|1]= lz[rt]?0:r-m;
    lz[rt<<1]=lz[rt<<1|1]=lz[rt]; lz[rt]=-1;
}
void update(int rt,int l,int r,int L,int R,int v) {
    if (L<=l&&r<=R) {
        lsum[rt]=rsum[rt]=msum[rt]= v?0:r-l+1;
        lz[rt]=v;
        return;
    }
    push_down(rt,l,r);
    int m=(l+r)>>1;
    if (L<=m) update(lson,L,R,v);
    if (m<R) update(rson,L,R,v);
    push_up(rt,l,r);
}
//返回有连续v个空间的最小下标
int query(int rt,int l,int r,int v) {
    if(l==r) return l;
    push_down(rt,l,r);
    int m=(l+r)>>1;
    if (msum[rt<<1] >= v) return query(lson,v);
    if (rsum[rt<<1]+lsum[rt<<1|1] >= v) return m-rsum[rt<<1]+1;
    return query(rson,v);
}
int main() {
    scanf("%d",&n,&m);
    lsum[1]=rsum[1]=msum[1]=n; lz[1]=0;
    for (int i=1;i<=m;++i) {
        scanf("%d",&op);
        if (op==1) {
            scanf("%d",&x);
            if (msum[1]<x) { printf("0\n"); continue; }
            int res=query(1,1,n,x);
            printf("%d\n",res);
            update(1,1,n,res,res+x-1,1);
        } else {
            scanf("%d",&x,&y);
            update(1,1,n,x,x+y-1,0);
        }
    }
    return 0;
}

```

- 处理区间

1. 区间压点：贴海报Hiho1079，对于[1,4][1,2][3,4]这组数据答案为3
2. 数据加倍：区间交并补POJ3225，加倍后多出来的奇数节点表示区间开闭

扫描线

周长并

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#define lson    rt<<1,l,m
#define rson    rt<<1|1,m+1,r
using namespace std;
const int maxn=5e3+5;

int id[maxn<<1],numid,nume;
int getid(int x) { return lower_bound(id+1,id+1+numid,x)-id; }
struct edge {
    int x,ya,yb,lr;
    edge() {}
    edge(int x,int ya,int yb,int lr):x(x),ya(ya),yb(yb),lr(lr) {}
} e[maxn<<1];
bool cmp(edge a,edge b) { if (a.x==b.x) return a.lr>b.lr; return a.x<b.x; }
//sum线段长度 num线段数量 c是否覆盖 cl,cr左右端点是否覆盖
int n,sum[maxn<<3],num[maxn<<3],c[maxn<<3],cl[maxn<<3],cr[maxn<<3];
inline void push_up(int rt,int l,int r) {
    if (c[rt]) {
        sum[rt]=id[r+1]-id[l];
        cl[rt]=cr[rt]=num[rt]=1;
    }
    else if (l==r)
        sum[rt]=cl[rt]=cr[rt]=num[rt]=0;
    else {
        sum[rt]=sum[rt<<1]+sum[rt<<1|1];
        num[rt]=num[rt<<1]+num[rt<<1|1];
        cl[rt]=cl[rt<<1]; cr[rt]=cr[rt<<1|1];
        if (cr[rt<<1]&&cl[rt<<1|1]) --num[rt];
    }
}
void update(int rt,int l,int r,int L,int R,int v) {
    if (L<=l&&r<=R) {
        c[rt]+=v;
        push_up(rt,l,r);
        return;
    }
    int m=l+r>>1;
    if (L<=m) update(lson,L,R,v);
    if (m<R) update(rson,L,R,v);
    push_up(rt,l,r);
}
int main() {
    while (~scanf("%d",&n)) {
        nume=numid=0;
        for (int i=1;i<maxn;i++) sum[i]=num[i]=c[i]=cl[i]=cr[i]=0;
        for (int i=1;i<=n;++i) {
            int xa,xb,ya,yb;
```

```

scanf("%d%d%d%d",&xa,&ya,&xb,&yb);
e[++nume]=edge(xa,ya,yb,1);
e[++nume]=edge(xb,ya,yb,-1);
id[++numid]=ya;
id[++numid]=yb;
}
sort(e+1,e+1+nume,cmp);
sort(id+1,id+1+numid);
numid=unique(id+1,id+1+numid)-id-1;
int ans=0,len=0;
for (int i=1;i<nume;++i) {
    update(1,1,numid,getId(e[i].ya),getId(e[i].yb)-1,e[i].lr);
    ans+=abs(len-sum[1]); len=sum[1];
    ans+=num[1]*2*(e[i+1].x-e[i].x);
}
printf("%d\n",ans+e[nume].yb-e[nume].ya);
}
return 0;
}

```

面积并

```

#include<cstdio>
#include<cstring>
#include<algorithm>
#define lson rt<<1,l,m
#define rson rt<<1|1,m+1,r
using namespace std;
const int maxn=1e2+5;
int numid,nume;
double id[maxn<<1];
inline int getId(double x) { return lower_bound(id+1,id+1+numid,x)-id; }
struct edge {
    double x,ya,yb; int lr;
    edge() {}
    edge(double x,double ya,double yb,int lr):x(x),ya(ya),yb(yb),lr(lr) {}
} e[maxn<<1];
inline bool cmp(edge a,edge b) { return a.x<b.x; }

int lz[maxn<<3]; double sum[maxn<<3];
inline void push_up(int rt,int l,int r) {
    if (lz[rt]) sum[rt]=id[r+1]-id[l];
    else sum[rt]=sum[rt<<1]+sum[rt<<1|1];
}
void update(int rt,int l,int r,int L,int R,int v) {
    if (L<=l&&r<=R) {
        lz[rt]+=v;
        push_up(rt,l,r);
        return;
    }
    int m=l+r>>1;
    if (L<=m) update(lson,L,R,v);

```

```

    if (m<R) update(rson,L,R,v);
    push_up(rt,l,r);
}
int main() {
    int n,cnt=1;
    while (~scanf("%d",&n)&&n) {
        nume=numid=0;
        memset(lz,0,sizeof lz);
        memset(sum,0,sizeof sum);
        for (int i=1;i<=n;++i) {
            double xa,xb,ya,yb;
            scanf("%lf%lf%lf%lf",&xa,&ya,&xb,&yb);
            id[++numid]=ya;
            id[++numid]=yb;
            e[++nume]=edge(xa,ya,yb,1);
            e[++nume]=edge(xb,ya,yb,-1);
        }
        sort(e+1,e+1+nume,cmp);
        sort(id+1,id+1+numid);
        numid=unique(id+1,id+1+numid)-id-1;
        double ans=0;
        for (int i=1;i<nume;++i) {
            update(1,1,numid-1,getId(e[i].ya),getId(e[i].yb)-1,e[i].lr);
            ans+=sum[1]*(e[i+1].x-e[i].x);
        }
        printf("Test case #%d\n",cnt++);
        printf("Total explored area: %.2f\n\n",ans);
    }
    return 0;
}

```

面积交

```

#include<cstdio>
#include<cstring>
#include<algorithm>
#define lson rt<<1,l,m
#define rson rt<<1|1,m+1,r
using namespace std;
const int maxn=1e3+5;
int numid,nume;
double id[maxn<<1];
inline int getId(double x) { return lower_bound(id+1,id+1+numid,x)-id; }
struct edge {
    double x,ya,yb; int lr;
    edge() {}
    edge(double x,double ya,double yb,int lr):x(x),ya(ya),yb(yb),lr(lr) {}
} e[maxn<<1];
inline bool cmp(edge a,edge b) { return a.x<b.x; }

int c[maxn<<3]; double one[maxn<<3],two[maxn<<3];
inline void push_up(int rt,int l,int r) {

```

```

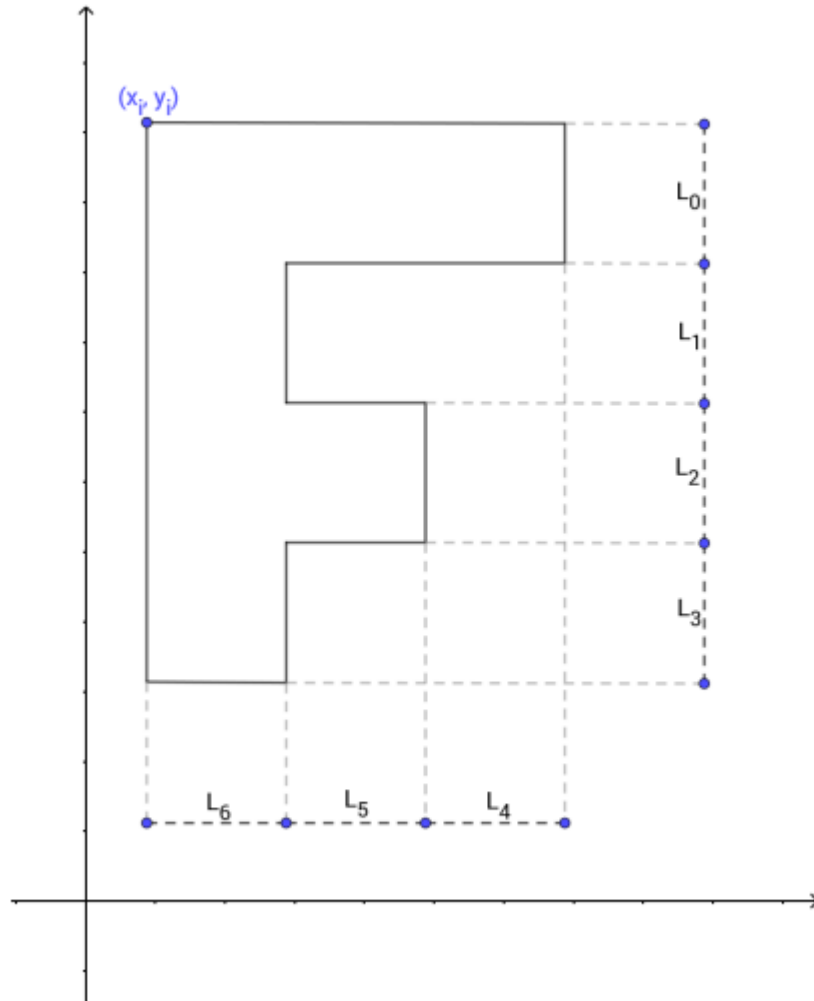
if (c[rt]>=2)
    one[rt]=two[rt]=id[r+1]-id[l];
else if (c[rt]==1)
    one[rt]=id[r+1]-id[l],
    two[rt]=l==r?0:one[rt<<1]+one[rt<<1|1];
else
    one[rt]=l==r?0:one[rt<<1]+one[rt<<1|1],
    two[rt]=l==r?0:two[rt<<1]+two[rt<<1|1];
}

void update(int rt,int l,int r,int L,int R,int v) {
    if (L<=l&&R<=r) {
        c[rt]+=v;
        push_up(rt,l,r);
        return;
    }
    int m=l+r>>1;
    if (L<=m) update(lson,L,R,v);
    if (m<R) update(rson,L,R,v);
    push_up(rt,l,r);
}

int main() {
    int T,n;
    scanf("%d",&T);
    while (T--) {
        scanf("%d",&n);
        nume=numid=0;
        memset(c,0,sizeof c);
        memset(one,0,sizeof one);
        memset(two,0,sizeof two);
        for (int i=1;i<=n;++i) {
            double xa,xb,ya,yb;
            scanf("%lf%lf%lf%lf",&xa,&ya,&xb,&yb);
            id[++numid]=ya;
            id[++numid]=yb;
            e[++nume]=edge(xa,ya,yb,1);
            e[++nume]=edge(xb,ya,yb,-1);
        }
        sort(e+1,e+1+nume,cmp);
        sort(id+1,id+1+numid);
        numid=unique(id+1,id+1+numid)-id-1;
        double ans=0;
        for (int i=1;i<nume;++i) {
            update(1,1,numid-1,getid(e[i].ya),getid(e[i].yb)-1,e[i].lr);
            ans+=two[1]*(e[i+1].x-e[i].x);
        }
        printf("%.2f\n",ans);
    }
    return 0;
}

```

不相交图形个数



给你图形F的几个区域长度 $L_0, L_1, L_2, L_3, L_4, L_5, L_6$, 然后给你N个点, 为图形F的左上角坐标, 问有几个F图形和其他图形都不相交。 ($1 \leq N \leq 1e5$, $1 \leq L \leq 1e3$, $1 \leq |x_i|, |y_i| \leq 1e9$, 保证 (x_i, y_i) 不同)

对于数据 1 1 1 1 1 1 2 0 0 1 3, 我们从左向右扫一遍能去掉第二个, 从右向左也只能去掉第二个。

发现图形F具有偏序关系, 相同的图形可以按他们的左上角坐标旋转180的, 然后从右往左扫, 就改变了偏序关系

```
#include<bits/stdc++.h>
#define lson rt<<1,l,m
#define rson rt<<1|1,m+1,r
using namespace std;
typedef long long ll;
const int maxn=1e5+5;

int id[maxn<<2],numid,nume;
int getid(int x) { return lower_bound(id+1,id+1+numid,x)-id; }
struct edge {
    int x,ya,yb,lr,idx;
    edge() {}
    edge(int x,int ya,int yb,int lr,int idx):x(x),ya(ya),yb(yb),lr(lr),idx(idx) {}
} e[maxn<<3];
bool cmp(edge a,edge b) {
    if (a.x==b.x&&a.lr==b.lr) return a.idx<b.idx;
    if (a.x==b.x) return a.lr>b.lr;
```

```

    return a.x<b.x;
}

ll t[maxn<<4],lz[maxn<<4];
inline void push_up(int rt) { t[rt]=t[rt<<1]|t[rt<<1|1]; }
inline void push_down(int rt,int l,int r) {
    if (lz[rt]) {
        int m=l+r>>1;
        t[rt<<1]+=lz[rt];
        t[rt<<1|1]+=lz[rt];
        lz[rt<<1]+=lz[rt];
        lz[rt<<1|1]+=lz[rt];
        lz[rt]=0;
    }
}

void update(int rt,int l,int r,int L,int R,int v) {
    if (L<=l&&r<=R) { t[rt]+=v; lz[rt]+=v; return; }
    push_down(rt,l,r);
    int m=l+r>>1;
    if (L<=m) update(lson,L,R,v);
    if (m<R) update(rson,L,R,v);
    push_up(rt);
}

ll query(int rt,int l,int r,int L,int R) {
    if (L<=l&&r<=R) return t[rt];
    push_down(rt,l,r);
    int m=l+r>>1;
    int res=0;
    if (L<=m) res+=query(lson,L,R);
    if (m<R) res+=query(rson,L,R);
    return res;
}

int n,w[11],x[maxn],y[maxn],vis[maxn];
int main () {
    for (int i=0;i<=6;++i) scanf("%d",&w[i]);
    for (int i=1;i<=3;++i) w[i]+=w[i-1];
    for (int i=5;i>=4;--i) w[i]+=w[i+1];
    scanf("%d",&n);
    for (int i=1;i<=n;++i) scanf("%d%d",&x[i],&y[i]);
    for (int i=1;i<=n;++i) {
        e[++nume]=edge(x[i], y[i]-w[3],y[i],0,i);
        e[++nume]=edge(x[i]+w[6],y[i]-w[3],y[i],1,i);
        e[++nume]=edge(x[i]+w[6],y[i]-w[2],y[i],0,i);
        e[++nume]=edge(x[i]+w[5],y[i]-w[2],y[i],1,i);
        e[++nume]=edge(x[i]+w[5],y[i]-w[0],y[i],0,i);
        e[++nume]=edge(x[i]+w[4],y[i]-w[0],y[i],1,i);
        id[++numid]=y[i];
        id[++numid]=y[i]-w[0];
        id[++numid]=y[i]-w[2];
        id[++numid]=y[i]-w[3];
    }
    sort(e+1,e+1+nume,cmp);
    sort(id+1,id+1+numid);
}

```

```

numid=unique(id+1,id+1+numid)-id-1;
for (int i=1;i<nume;++i) {
    int L=getid(e[i].ya),R=getid(e[i].yb)-1;
    if (!e[i].lr) {
        if (query(1,1,numid-1,L,R)) vis[e[i].idx]=1;
        update(1,1,numid-1,L,R,1);
    }
    else
        update(1,1,numid-1,L,R,-1);
}

nume=numid=0;
memset(t,0,sizeof t);
memset(lz,0,sizeof lz);
for (int i=1;i<=n;++i) {
    e[++nume]=edge(x[i],y[i],y[i]+w[3],1,i);
    e[++nume]=edge(x[i]-w[6],y[i],y[i]+w[3],0,i);
    e[++nume]=edge(x[i]-w[6],y[i],y[i]+w[2],1,i);
    e[++nume]=edge(x[i]-w[5],y[i],y[i]+w[2],0,i);
    e[++nume]=edge(x[i]-w[5],y[i],y[i]+w[0],1,i);
    e[++nume]=edge(x[i]-w[4],y[i],y[i]+w[0],0,i);
    id[++numid]=y[i];
    id[++numid]=y[i]+w[0];
    id[++numid]=y[i]+w[2];
    id[++numid]=y[i]+w[3];
}
sort(e+1,e+1+nume,cmp);
sort(id+1,id+1+numid);
numid=unique(id+1,id+1+numid)-id-1;
for (int i=nume;i-->0) {
    int L=getid(e[i].ya),R=getid(e[i].yb)-1;
    if (e[i].lr) {
        if (query(1,1,numid-1,L,R)) vis[e[i].idx]=1;
        update(1,1,numid-1,L,R,1);
    }
    else
        update(1,1,numid-1,L,R,-1);
}
int ans=0;
for (int i=1;i<=n;++i) if(!vis[i]) ans++;
printf("%d\n",ans);
return 0;
}

```

替罪羊树

树堆

```
int rt,tot,ch[maxn][2],key[maxn],val[maxn],sz[maxn],cnt[maxn];
/* 根 节点 子节点      关键字      值      子节点个数 重复值个数*/
```

1.push

```
inline void push_up(int x) { sz[x]=cnt[x]+sz[ch[x][0]]+sz[ch[x][1]]; }
```

2.rotate

```
inline void rotate(int &x,int d) {
    int y=ch[x][d^1];
    ch[x][d^1]=ch[y][d]; ch[y][d]=x; x=y;
    push_up(ch[x][d]); push_up(x)
}
```

bzoj3224

1插入x 2删除x 3查询x的排名 4查询排名为x的数 5求x的前驱 6求x的后继

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=1e6+20;
const int INF = 1e9;
int rt,tot,ch[maxn][2],key[maxn],val[maxn],sz[maxn],cnt[maxn];
inline int get_key() { static int x=471; return x=(48271LL*x+1)%2147483647; }
inline void push_up(int x) { sz[x]=cnt[x]+sz[ch[x][0]]+sz[ch[x][1]]; }
inline void rotate(int &x,int d) {
    int y=ch[x][d^1];
    ch[x][d^1]=ch[y][d]; ch[y][d]=x; x=y;
    push_up(ch[x][d]); push_up(x);
}
void add(int &x,int v) {
    if (!x) {
        x=++tot;
        key[x]=get_key(); val[x]=v;
        ch[x][0]=ch[x][1]=0;
        sz[x]=cnt[x]=1;
        return;
    }
    if (v==val[x]) { ++cnt[x],++sz[x]; return; }
    int d=v>val[x];
    add(ch[x][d],v);
    if (key[x]<key[ch[x][d]]) rotate(x,d^1);
    push_up(x);
}
void del(int &x,int v) {
    if (!x) return;
    if (v==val[x]) {
```

```

        if (cnt[x]>1) { --cnt[x],--sz[x]; return; }
        if (!ch[x][0]&&!ch[x][1]) { x=0; return; }
        if (!ch[x][1]||key[ch[x][0]]>key[ch[x][1]])
            rotate(x,1),del(ch[x][1],v);
        else
            rotate(x,0),del(ch[x][0],v);
        push_up(x);
        return;
    }
    if (v<val[x]) del(ch[x][0],v);
    else          del(ch[x][1],v);
    push_up(x);
}
inline int rankv(int v) {
    int x=rt,res=0;
    while (x) {
        if (v<val[x]) { x=ch[x][0]; continue; }
        res+=sz[ch[x][0]];
        if (v==val[x]) break;
        res+=cnt[x]; x=ch[x][1];
    }
    return res+1;
}
inline int findp(int p) {
    int x=rt;
    while (x) {
        if (p<=sz[ch[x][0]]) { x=ch[x][0]; continue; }
        int tmp=sz[ch[x][0]]+cnt[x];
        if (p<=tmp) break;
        p-=tmp; x=ch[x][1];
    }
    return val[x];
}
inline int get_pre(int v) {
    int x=rt,y=0;
    while (x) {
        if (val[x]<v) y=x,x=ch[x][1];
        else x=ch[x][0];
    }
    return val[y];
}
inline int get_next(int v) {
    int x=rt,y=0;
    while (x) {
        if (val[x]>v) y=x,x=ch[x][0];
        else x=ch[x][1];
    }
    return val[y];
}
int main() {
    int n,op,x;
    scanf("%d",&n);
    for (int i=1;i<=n;++i) {

```

```

scanf("%d%d",&op,&x);
if (op==1) add(rt,x);
if (op==2) del(rt,x);
if (op==3) printf("%d\n",rankv(x));
if (op==4) printf("%d\n",findp(x));
if (op==5) printf("%d\n",get_pre(x));
if (op==6) printf("%d\n",get_next(x));
}
return 0;
}

```

伸展树

```

int n,m,a[maxn];
int rt,tot,fa[maxn],ch[maxn][2],key[maxn],cnt[maxn],sz[maxn],lz[maxn],rvs[maxn];
/* 根    父节点    子节点    关键字    值相同个数    子节点个数    懒惰标记    反转标记*/

```

1.push

```

inline void push_up(int x) {
    //子节点个数
    sz[x]=cnt[x]+sz[ch[x][0]]+sz[ch[x][1]];
    //最小值的最小位置
    minp[x]=x;
    if (key[minp[ch[x][0]]]==key[minp[x]]) minp[x]=min(minp[x],minp[ch[x][0]]);
    if (key[minp[ch[x][0]]]< key[minp[x]]) minp[x]=minp[ch[x][0]];
    if (key[minp[ch[x][1]]]==key[minp[x]]) minp[x]=min(minp[x],minp[ch[x][1]]);
    if (key[minp[ch[x][1]]]< key[minp[x]]) minp[x]=minp[ch[x][1]];
}
inline void push_down(int x) {
    if (lz[x]) { //赋值标记
        if (ch[x][0]) key[ch[x][0]]=lz[ch[x][0]]=lz[x];
        if (ch[x][1]) key[ch[x][1]]=lz[ch[x][1]]=lz[x];
        lz[x]=0;
    }
    if (rvs[x]){ //反转标记
        rvs[ch[x][0]]^=1; rvs[ch[x][1]]^=1;
        swap(ch[ch[x][0]][0],ch[ch[x][0]][1]);
        swap(ch[ch[x][1]][0],ch[ch[x][1]][1]);
        rvs[x]=0;
    }
}
}

```

2.build

```

int build(int f,int l,int r,int *arr=a) {
    if (l>r) return 0;
    int m=(l+r)>>1,x=++tot;
    fa[x]=f; key[x]=arr[m]; sz[x]=0; cnt[x]=1; lz[x]=0; rvs[x]=0;
}

```

```

    ch[x][0]=build(x,l,m-1,arr);
    ch[x][1]=build(x,m+1,r,arr);
    push_up(x);
    return x;
}
//先初始化可用内存池，节省空间，加快速度
queue<int> que;
inline int newp() {
    int x=que.front(); que.pop();
    if (ch[x][0]) que.push(ch[x][0]);
    if (ch[x][1]) que.push(ch[x][1]);
    return x;
}
int build(int f,int l,int r,int *arr=a) {
    if (l>r) return 0;
    int m=(l+r)>>1,x=newp();
    fa[x]=f; key[x]=arr[m]; sz[x]=0; cnt[x]=1; lz[x]=0; rvs[x]=0;
    ch[x][0]=build(x,l,m-1,arr);
    ch[x][1]=build(x,m+1,r,arr);
    push_up(x);
    return x;
}

```

3.splay

```

inline bool get(int x) { return ch[fa[x]][1]==x; }
inline void rotate(int x) {
    push_down(fa[x]); push_down(x);
    int y=fa[x],z=fa[y],son=get(x);
    if (z) ch[z][get(y)]=x; fa[x]=z;
    ch[y][son]=ch[x][son^1]; if(ch[x][son^1]) fa[ch[x][son^1]]=y;
    ch[x][son^1]=y; fa[y]=x;
    push_up(y); push_up(x);
}
inline void splay(int x,int tar=0) {
    for (int y;(y=fa[x])!=tar;rotate(x))
        if (fa[y]!=tar)
            rotate(get(x)==get(y)?y:x);
    if (!tar) rt=x;
}

```

4.单点操作

```

//前驱lst(ch[rt][0]) 后继fst(ch[rt][1]) push_down!!
inline int fst(int x) { push_down(x);while(ch[x][0])x=ch[x][0],push_down(x);return x; }
inline int lst(int x) { push_down(x);while(ch[x][1])x=ch[x][1],push_down(x);return x; }
//查找值为v的排名
inline int rankv(int v) {
    int res=0,x=rt;
    while (x) {
        push_down(x);

```

```

        if (v<key[x]) { x=ch[x][0]; continue; }
        res+=sz[ch[x][0]];
        if (v==key[x]) { splay(x); break; }
        res+=cnt[x]; x=ch[x][1];
    }
    return res;
}
//查找值为v的节点
inline int findv(int v) {
    int x=rt;
    while (x) {
        push_down(x);
        if (v==key[x]) { splay(x); return x; }
        x=ch[x][key[x]<v];
    }
    return -1;
}
//查找第p个节点
inline int findp(int p) {
    int x=rt;
    while (x) {
        push_down(x);
        if (p<=sz[ch[x][0]]) { x=ch[x][0]; continue; }
        int tmp=sz[ch[x][0]]+cnt[x];
        if (p<=tmp) return x;
        p-=tmp; x=ch[x][1];
    }
    return -1;
}
inline void add(int v) {
    int x=rt,y;
    while (x) {
        push_down(x);
        if (key[x]==v) { ++cnt[x]; ++sz[x]; splay(x); return; }
        y=x; x=ch[x][key[x]<v];
    }
    rt=x+++tot;
    fa[x]=y; ch[x][0]=ch[x][1]=0; key[x]=v; cnt[x]=sz[x]=1;
    if(y) ch[y][key[y]<v]=x,splay(x);
}
inline void del(int v) {
    int x=findv(v);
    if (cnt[rt]>1) { --cnt[rt];--sz[rt]; return; }
    if (!ch[rt][0]&&!ch[rt][1]) { tot=rt=0; return; }
    if (!ch[rt][0]) { rt=ch[rt][1]; fa[rt]=0; return; }
    if (!ch[rt][1]) { rt=ch[rt][0]; fa[rt]=0; return; }
    splay(fst(ch[x][1]));
    ch[rt][0]=ch[x][0];
    fa[ch[x][0]]=rt;
    push_up(rt);
}

```

5.区间操作

添加首尾节点方便操作，坐标全都加一！

```
//将区间[l+1,r-1]旋转到位 rt:l y=ch[rt][1]:r x=ch[y][0]:[l+1,r-1]
inline int inter(int l,int r) {
    splay(findp(l));
    splay(findp(r),rt);
    return ch[rt][1];
}
inline void adds(int p,int n) {
    int x=build(0,1,n);
    int y=inter(p,p+1);
    ch[y][0]=x; fa[x]=y;
    push_up(y); push_up(rt);
}
inline void dels(int l,int r) {
    int y=inter(l-1,r+1);
    que.push(ch[y][0]); ch[y][0]=0;
    push_up(y); push_up(rt);
}
inline void update(int l,int r,int v) {
    int y=inter(l-1,r+1),x=ch[y][0];
    key[x]=lz[x]=v; rvs[x]=0;
    push_up(y); push_up(rt);
}
inline void reverse(int l,int r) {
    int y=inter(l-1,r+1),x=ch[y][0];
    rvs[x]^=1; swap(ch[x][0],ch[x][1]);
}
}
```

6.debug

```
inline void debug(int x) {
    if (!x) return;
    if (x==rt) printf("=====\n");
    printf("%2d %2d fa:%2d ls:%2d rs:%2d\n", x,key[x],fa[x],ch[x][0],ch[x][1]);
    push_down(x); debug(ch[x][0]); debug(ch[x][1]);
}
}
```

bzoj3224

1插入x 2删除x 3查询x的排名 4查询排名为x的数 5求x的前驱 6求x的后继

```
#include<cstdio>
using namespace std;
const int maxn = 1e6+5;
int rt,tot,fa[maxn],ch[maxn][2],key[maxn],cnt[maxn],sz[maxn];
inline void push_up(int x) { sz[x]=cnt[x]+sz[ch[x][0]]+sz[ch[x][1]]; }
inline bool get(int x) { return ch[fa[x]][1]==x; }
inline void rotate(int x) {
    int y=fa[x],z=fa[y],son=get(x);
    ch[y][son]=ch[x][son^1]; fa[ch[x][son^1]]=y;
}
```

```

    if (z) ch[z][get(y)]=x; fa[x]=z;
    ch[x][son^1]=y; fa[y]=x;
    push_up(y); push_up(x);
}
inline void splay(int x,int target=0) {
    for (int y;(y=fa[x])!=target;rotate(x))
        if (fa[y]!=target)
            rotate(get(x)==get(y)?y:x);
    if (!target) rt=x;
}
inline int fst(int x=rt) { while (ch[x][0]) x=ch[x][0]; return x; }
inline int lst(int x=rt) { while (ch[x][1]) x=ch[x][1]; return x; }
inline int rankv(int v) {
    int x=rt,res=0;
    while (x) {
        if (v<key[x]) { x=ch[x][0]; continue; }
        res+=sz[ch[x][0]];
        if (v==key[x]) { splay(x); return res+1; }
        res+=cnt[x]; x=ch[x][1];
    }
    return -1;
}
inline int findv(int v) {
    int x=rt;
    while (x) {
        if (v==key[x]) { splay(x); return x; }
        x=ch[x][key[x]<v];
    }
    return -1;
}
inline int findp(int p) {
    int x=rt;
    while (x) {
        if (p<=sz[ch[x][0]]) { x=ch[x][0]; continue; }
        int res=sz[ch[x][0]]+cnt[x];
        if (p<=res) return x;
        p-=res; x=ch[x][1];
    }
    return -1;
}
inline void add(int v) {
    int x=rt,y=0;
    while (x) {
        if (key[x]==v) { ++cnt[x]; ++sz[x]; splay(x); return; }
        y=x,x=ch[x][key[x]<v];
    }
    rt=x+++tot;
    fa[x]=y; ch[x][0]=ch[x][1]=0; key[x]=v; cnt[x]=sz[x]=1;
    if(y) ch[y][key[y]<v]=x,splay(x);
}
inline void del(int v) {
    int x=findv(v);
    if (cnt[rt]>1) { --cnt[rt];--sz[rt]; return; }

```

```

    if (!ch[rt][0]&&!ch[rt][1]) { tot=rt=0; return; }
    if (!ch[rt][0]) { rt=ch[rt][1]; fa[rt]=0; return; }
    if (!ch[rt][1]) { rt=ch[rt][0]; fa[rt]=0; return; }
    splay(fst(ch[x][1]));
    ch[rt][0]=ch[x][0];
    fa[ch[x][0]]=rt;
    push_up(rt);
}
int main() {
    int n,op,x;
    scanf("%d",&n);
    for (int i=1;i<=n;i++) {
        scanf("%d%d",&op,&x);
        if (op==1) add(x);
        if (op==2) del(x);
        if (op==3) printf("%d\n",rankv(x));
        if (op==4) printf("%d\n",key[findp(x)]);
        if (op==5) { add(x); printf("%d\n",key[lst(ch[rt][0]))); del(x); }
        if (op==6) { add(x); printf("%d\n",key[fst(ch[rt][1]))); del(x); }
    }
    return 0;
}

```

bzoj1500

1. INSERT_post_tot_c₁c₂...c_{tot}, 在当前数列的第post个数字后插入tot个数字
2. DELETE_post_tot, 从当前数列的第post个数字开始删除tot个数字
3. MAKE-SAME_post_tot_c, 将当前数列的第post个数字开始的连续tot个数字改为c
4. REVERSE_post_tot, 取出从当前数列的第post个数字开始的tot个数字, 翻转后放入原位置
5. GET-SUM_post_tot, 计算从当前数列的第post个数字开始的tot个数字的和并输出
6. MAX-SUM, 求出当前数列中和最大的一段子列, 并输出最大和

```

#include<queue>
#include<cstdio>
#include<cstring>
using namespace std;
const int inf = 0x3f3f3f3f;
const int maxn = 5e5+5;
char op[20];
int n,m,p,v,a[maxn],sum[maxn],lsum[maxn],rsum[maxn],msum[maxn];
int rt,fa[maxn],ch[maxn][2],key[maxn],sz[maxn],lz[maxn],rvs[maxn];
inline void push_up(int x) {
    sz[x]=sz[ch[x][0]]+1+sz[ch[x][1]];
    sum[x]=sum[ch[x][0]]+key[x]+sum[ch[x][1]];
    lsum[x]=max(lsum[ch[x][0]],sum[ch[x][0]]+key[x]+lsum[ch[x][1]]);
    rsum[x]=max(rsum[ch[x][1]],sum[ch[x][1]]+key[x]+rsum[ch[x][0]]);
    msum[x]=max(msum[ch[x][0]],msum[ch[x][1]]);
    msum[x]=max(msum[x],rsum[ch[x][0]]+key[x]+lsum[ch[x][1]]);
}
inline void push_down(int x) {
    if (lz[x]!=inf) {
        if (ch[x][0]) {

```

```

        key[ch[x][0]]=lz[ch[x][0]]=lz[x];
        sum[ch[x][0]]=lz[x]*sz[ch[x][0]];
        if (lz[x]>=0) lsum[ch[x][0]]=rsum[ch[x][0]]=msum[ch[x][0]]=sum[ch[x][0]];
        else          lsum[ch[x][0]]=rsum[ch[x][0]]=0,msum[ch[x][0]]=lz[x];
    }
    if (ch[x][1]) {
        key[ch[x][1]]=lz[ch[x][1]]=lz[x];
        sum[ch[x][1]]=lz[x]*sz[ch[x][1]];
        if (lz[x]>=0) lsum[ch[x][1]]=rsum[ch[x][1]]=msum[ch[x][1]]=sum[ch[x][1]];
        else          lsum[ch[x][1]]=rsum[ch[x][1]]=0,msum[ch[x][1]]=lz[x];
    }
    lz[x]=inf;
}
if (rvs[x]){
    rvs[ch[x][0]]^=1,rvs[ch[x][1]]^=1;
    swap(ch[ch[x][0]][0],ch[ch[x][0]][1]);
    swap(ch[ch[x][1]][0],ch[ch[x][1]][1]);
    swap(lsum[ch[x][0]],rsum[ch[x][0]]);
    swap(lsum[ch[x][1]],rsum[ch[x][1]]);
    rvs[x]=0;
}
}
queue<int> que;
inline int newp() {
    int x=que.front(); que.pop();
    if (ch[x][0]) que.push(ch[x][0]);
    if (ch[x][1]) que.push(ch[x][1]);
    return x;
}
int build(int f,int l,int r) {
    if (l>r) return 0;
    int m=(l+r)>>1,x=newp();
    fa[x]=f; key[x]=a[m]; sz[x]=0; lz[x]=inf; rvs[x]=0;
    sum[x]=lsum[x]=rsum[x]=0; msum[x]=a[m];
    ch[x][0]=build(x,l,m-1);
    ch[x][1]=build(x,m+1,r);
    push_up(x);
    return x;
}
inline bool get(int x) { return ch[fa[x]][1]==x; }
inline void rotate(int x) {
    push_down(fa[x]); push_down(x);
    int y=fa[x],z=fa[y],son=get(x);
    ch[y][son]=ch[x][son^1]; fa[ch[x][son^1]]=y;
    if (z) ch[z][get(y)]=x; fa[x]=z;
    ch[x][son^1]=y; fa[y]=x;
    push_up(y); push_up(x);
}
inline void splay(int x,int tar=0) {
    if (x==-1) return; //findp返回-1
    for (int y;(y=fa[x])!=tar;rotate(x))
        if (fa[y]!=tar)
            rotate(get(x)==get(y)?y:x);
}

```

```

    if (!tar) rt=x;
}
inline int fst(int x) { push_down(x);while(ch[x][0])x=ch[x][0],push_down(x);return x; }
inline int lst(int x) { push_down(x);while(ch[x][1])x=ch[x][1],push_down(x);return x; }
inline int findp(int p) {
    int x=rt;
    while (x) {
        push_down(x);
        if (p<=sz[ch[x][0]]) { x=ch[x][0]; continue; }
        if (p==sz[ch[x][0]]+1) return x;
        p-=sz[ch[x][0]]+1; x=ch[x][1];
    }
    return -1;
}
inline int inter(int l,int r) {
    splay(findp(l));
    splay(findp(r),rt);
    return ch[rt][1];
}
inline void adds(int p,int n) {
    int x=build(0,1,n);
    int y=inter(p,p+1);
    ch[y][0]=x; fa[x]=y;
    push_up(y); push_up(rt);
}
inline void dels(int l,int r) {
    int y=inter(l-1,r+1);
    que.push(ch[y][0]); ch[y][0]=0;
    push_up(y); push_up(rt);
}
inline void update(int l,int r,int v) {
    int y=inter(l-1,r+1),x=ch[y][0];
    key[x]=lz[x]=v; rvs[x]=0; sum[x]=v*sz[x];
    if (v>=0) lsum[x]=rsum[x]=msum[x]=sum[x];
    else      lsum[x]=rsum[x]=0,msum[x]=v;
    push_up(y); push_up(rt);
}
inline void reverse(int l,int r) {
    int y=inter(l-1,r+1),x=ch[y][0];
    rvs[x]^=1;
    swap(ch[x][0],ch[x][1]);
    swap(lsum[x],rsum[x]);
}
int main() {
    scanf("%d%d",&n,&m); msum[0]=-inf;
    for (int i=1;i<=n;i++) scanf("%d",&a[i+1]);
    for (int i=1;i<=500000;i++) que.push(i);
    rt=build(0,1,n+2);
    for (int i=1;i<=m;i++) {
        scanf("%s",op);
        if (op[2]=='S') {
            scanf("%d%d",&p,&n);
            for (int i=1;i<=n;i++) scanf("%d",&a[i]);

```

```

        adds(p+1,n);
    }
    if (op[2]=='L') {
        scanf("%d%d",&p,&n);
        dels(p+1,p+n);
    }
    if (op[2]=='K') {
        scanf("%d%d%d",&p,&n,&v);
        update(p+1,p+n,v);
    }
    if (op[2]=='V') {
        scanf("%d%d",&p,&n);
        rvserse(p+1,p+n);
    }
    if (op[2]=='T') {
        scanf("%d%d",&p,&n);
        int y=inter(p,p+n+1),x=ch[y][0];
        printf("%d\n",sum[x]);
    }
    if (op[2]=='X') {
        splay(fst(rt)); splay(lst(rt),rt);
        printf("%d\n",msum[ch[ch[rt][1]][0]]);
    }
}
return 0;
}

```

主席树

静态区间K大值

```

#include<cstdio>
#include<algorithm>
using namespace std;
#define lson ls[x],ls[y],l,m
#define rson rs[x],rs[y],m+1,r
const int maxn=1e5+5;
const int maxm=2e6+5;

int id[maxn],numid;
inline int getid(int x) { return lower_bound(id+1,id+1+numid,x)-id; }

int a[maxn],n,m;
int rt[maxn],ls[maxm],rs[maxm],sum[maxm],tot;
void update(int x,int &y,int l,int r,int v) {
    y=++tot;
    if(l==r) { sum[y]=sum[x]+1; return; }
    int m=l+r>>1;
    if(v<=m) rs[y]=rs[x],update(ls[x],ls[y],l,m,v);
    else ls[y]=ls[x],update(rs[x],rs[y],m+1,r,v);
    sum[y]=sum[ls[y]]+sum[rs[y]];
}

```

```

}
int query(int x,int y,int l,int r,int k) {
    if(l==r) return l;
    int m=l+r>>1;
    int res=sum[ls[y]]-sum[ls[x]];
    if(res>=k) return query(ls[x],ls[y],l,m,k);
    return query(rs[x],rs[y],m+1,r,k-res);
}
int main() {
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;++i) scanf("%d",&a[i]);
    //离散化
    for(int i=1;i<=n;++i) id[++numid]=a[i];
    sort(id+1,id+1+numid);
    numid=unique(id+1,id+1+numid)-id-1;

    for(int i=1;i<=n;++i) update(rt[i-1],rt[i],1,numid,getid(a[i]));
    for(int i=1;i<=m;++i) {
        int x,y,k;
        scanf("%d%d%d",&x,&y,&k);
        printf("%d\n",id[query(rt[x-1],rt[y],1,numid,k)]);
    }
    return 0;
}

```

持久化并查集

当作线段树，每次只修改一个点，查询一个点

1. 合并x,y
2. 退回到第x个版本
3. 询问x,y是否为同一集合

```

#include<bits/stdc++.h>
using namespace std;
const int maxn=1e6+5;
const int maxm=2e7+5;

int n,m,a[maxn];
int rt[maxn],ls[maxn],rs[maxn],fa[maxn],dep[maxn],tot;
void build(int &x,int l,int r) {
    x=++tot;
    if (l==r) { fa[x]=l; return; }
    int m=(l+r)>>1;
    build(ls[x],l,m);
    build(rs[x],m+1,r);
}
void add(int x,int l,int r,int p) {
    if (l==r) { ++dep[x]; return; }
    int m=l+r>>1;
    if (p<=m) add(ls[x],l,m,p);
    else      add(rs[x],m+1,r,p);
}

```

```

}
void update(int x,int &y,int l,int r,int p,int v) {
    y=++tot;
    if (l==r) { fa[y]=v; dep[y]=dep[x]; return; }
    int m=(l+r)>>1;
    if (p<=m) rs[y]=rs[x],update(ls[x],ls[y],l,m,p,v);
    else     ls[y]=ls[x],update(rs[x],rs[y],m+1,r,p,v);
}
int query(int rt,int l,int r,int p) {
    if (l==r) return rt;
    int m=(l+r)>>1;
    if (p<=m) return query(ls[rt],l,m,p);
    return query(rs[rt],m+1,r,p);
}
int Find(int rt,int p) {
    int q=query(rt,1,n,p);
    return p==fa[q]?q:Find(rt,fa[q]);
}
void Union(int i,int x,int y) {
    int fx=Find(rt[i],x);
    int fy=Find(rt[i],y);
    if (fa[fx]==fa[fy]) return;
    if (dep[fx]>dep[fy]) swap(fx,fy);
    update(rt[i-1],rt[i],1,n,fa[fx],fa[fy]);
    if (dep[fx]==dep[fy]) add(rt[i],1,n,fa[fy]);
}
int main() {
    scanf("%d%d",&n,&m);
    build(rt[0],1,n);
    for (int i=1;i<=m;++i) {
        int op,x,y; scanf("%d",&op);
        if (op==1) {
            rt[i]=rt[i-1];
            scanf("%d%d",&x,&y);
            Union(i,x,y);
        }
        else if (op==2) {
            scanf("%d",&x);
            rt[i]=rt[x];
        } else {
            rt[i]=rt[i-1];
            scanf("%d%d",&x,&y);
            int fx=Find(rt[i],x);
            int fy=Find(rt[i],y);
            if (fa[fx]==fa[fy]) puts("1");
            else puts("0");
        }
    }
    return 0;
}

```

树套数

主席树套字典树

1. Or X, 区间[1,n]或X
2. And X, 区间[1,n]与X
3. Xor X, 区间[1,n]异或X
4. Ask L R K, 查询区间[L,R]的第K小

某位相同后，以后都相同，变相同时暴力重构

```
#include<cstdio>
const int maxn=5e4+5;
int n,m,val,a[maxn]; bool vis[35];
int rt[maxn<<5],t[maxn<<5][2],sum[maxn<<5],tot;
inline void update(int x,int &y,int v) {
    int z=y++tot;
    for (int i=30;i>=0;--i) {
        if ((v>>i)&1) t[z][0]=t[x][0],t[z][1]++;tot;
        else t[z][1]=t[x][1],t[z][0]++;tot;
        z=t[z][(v>>i)&1];
        x=t[x][(v>>i)&1];
        sum[z]=sum[x]+1;
    }
}
inline void rebuild(int p,int v) {
    if (!vis[p]) {
        for (int i=1;i<=n;++i)
            if ((a[i]>>p)&1)
                a[i]--1<<p;
        vis[p]=1; tot=0;
        for (int i=1;i<=n;++i)
            update(rt[i-1],rt[i],a[i]);
    }
    if (v==1) val|=1<<p;
    else if ((val>>p)&1) val--1<<p;
}
inline int query(int l,int r,int k) {
    int res=0;
    for (int i=30;i>=0;--i) {
        int p=(val>>i)&1;
        if (sum[t[r][p]]-sum[t[l][p]]>=k)
            l=t[l][p],r=t[r][p];
        else {
            k-=sum[t[r][p]]-sum[t[l][p]];
            l=t[l][p^1],r=t[r][p^1];
            res+=1<<i;
        }
    }
    return res;
}
int main() {
#ifdef LOCAL
    freopen("in.txt","r",stdin);
#endif
}
```

```

scanf("%d%d",&n,&m);
for (int i=1;i<=n;++i) scanf("%d",&a[i]);
for (int i=1;i<=n;++i) update(rt[i-1],rt[i],a[i]);
for (int i=1;i<=m;++i) {
    int l,r,x; char op[11];
    scanf(" %s",op);
    if (op[1]=='r') {
        scanf("%d",&x);
        for (int i=30;i>=0;--i) {
            if ((x>>i)&1)
                rebuild(i,1);
        }
    }
    if (op[1]=='n') {
        scanf("%d",&x);
        for (int i=30;i>=0;--i) {
            if ((x>>i)&1) continue;
            rebuild(i,0);
        }
    }
    if (op[1]=='o') {
        scanf("%d",&x);
        val^=x;
    }
    if (op[1]=='s') {
        scanf("%d%d%d",&l,&r,&x);
        printf("%d\n",query(rt[l-1],rt[r],x));
    }
}
return 0;
}

```

线段树套伸展树

LG3380二逼平衡树

1. 查询k在区间内的排名
2. 查询区间内排名为k的值
3. 修改某一位值上的数值
4. 查询k在区间内的前驱(前驱定义为严格小于x，且最大的数，若不存在输出-2147483647)
5. 查询k在区间内的后继(后继定义为严格大于x，且最小的数，若不存在输出2147483647)

```

#include<bits/stdc++.h>
using namespace std;
const int maxn=4e6+5;
const int inf=2147483647;
int n,m,ans,MAX,a[maxn];
namespace SPLAY {
    int tot,fa[maxn],ch[maxn][2],key[maxn],sz[maxn],cnt[maxn];
    inline void push_up(int x) { sz[x]=cnt[x]+sz[ch[x][0]]+sz[ch[x][1]]; }
    inline int get(int x) { return ch[fa[x]][1]==x; }
    inline void rotate(int x) {

```

```

    int y=fa[x],z=fa[y],son=get(x);
    if (z) ch[z][get(y)]=x; fa[x]=z;
    ch[y][son]=ch[x][son^1];
    if(ch[x][son^1]) fa[ch[x][son^1]]=y;
    ch[x][son^1]=y; fa[y]=x;
    push_up(y); push_up(x);
}

inline void splay(int &rt,int x,int tar=0) {
    for (int y;(y=fa[x])!=tar;rotate(x))
        if (fa[y]!=tar)
            rotate(get(x)==get(y)?y:x);
    if (!tar) rt=x;
}

inline int fst(int x) { while (ch[x][0]) x=ch[x][0]; return x; }
inline int lst(int x) { while (ch[x][1]) x=ch[x][1]; return x; }
inline int rankv(int &rt,int v) {
    int x=rt,res=0;
    while (x) {
        if (v<key[x]) { x=ch[x][0]; continue; }
        res+=sz[ch[x][0]];
        if (v==key[x]) break;
        res+=cnt[x];
        x=ch[x][1];
    }
    return res;
}

inline int findv(int &rt,int v) {
    int x=rt;
    while (x) {
        if (v==key[x]) { splay(rt,x); return x; }
        x=ch[x][key[x]<v];
    }
    return -1;
}

inline void add(int &rt,int v,int y=0) {
    int x=rt;
    while (x) {
        if (key[x]==v) { ++cnt[x]; ++sz[x]; splay(rt,x); return; }
        y=x; x=ch[x][key[x]<v];
    }
    rt=x++tot;
    fa[x]=y; ch[x][0]=ch[x][1]=0; key[x]=v; cnt[x]=sz[x]=1;
    if (y) ch[y][key[y]<v]=x,splay(rt,x);
}

inline void del(int &rt,int v) {
    int x=findv(rt,v);
    if (cnt[rt]>1) { --cnt[rt];--sz[rt]; return; }
    if (!ch[rt][0]&&!ch[rt][1]) { rt=0; return; }
    if (!ch[rt][0]) { rt=ch[rt][1]; fa[rt]=0; return; }
    if (!ch[rt][1]) { rt=ch[rt][0]; fa[rt]=0; return; }
    splay(rt,fst(ch[x][1]));
    ch[rt][0]=ch[x][0];
    fa[ch[x][0]]=rt;
}

```

```

        push_up(rt);
    }
    inline int pre(int &rt,int v) {
        int x=rt,res=-inf;
        while (x) {
            if (v<=key[x]) { x=ch[x][0]; continue; }
            if (res<key[x]) res=key[x];
            x=ch[x][1];
        }
        return res;
    }
    inline int nex(int &rt,int v) {
        int x=rt,res=inf;
        while (x) {
            if (key[x]<=v) { x=ch[x][1]; continue; }
            if (res>key[x]) res=key[x];
            x=ch[x][0];
        }
        return res;
    }
}

namespace SEG {
#define lson rt<<1,l,m
#define rson rt<<1|1,m+1,r
    int root[maxn];
    void add(int rt,int l,int r,int p,int v) {
        SPLAY::add(root[rt],v);
        if (l==r) return;
        int m=l+r>>1;
        if (p<=m) add(lson,p,v);
        else      add(rson,p,v);
    }
    int rankv(int rt,int l,int r,int L,int R,int v) {
        if (L<=l&&r<=R) return SPLAY::rankv(root[rt],v);
        int res=0;
        int m=l+r>>1;
        if (L<=m) res+=rankv(lson,L,R,v);
        if (m<R)  res+=rankv(rson,L,R,v);
        return res;
    }
    void update(int rt,int l,int r,int p,int v) {
        SPLAY::del(root[rt],a[p]);
        SPLAY::add(root[rt],v);
        if (l==r) { a[p]=v; return; }
        int m=l+r>>1;
        if (p<=m) update(lson,p,v);
        else      update(rson,p,v);
    }
    void pre(int rt,int l,int r,int L,int R,int v) {
        if (L<=l&&r<=R) { ans=max(ans,SPLAY::pre(root[rt],v)); return; }
        int m=l+r>>1;
        if (L<=m) pre(lson,L,R,v);
        if (m<R)  pre(rson,L,R,v);
    }
}

```

```

}
void nex(int rt,int l,int r,int L,int R,int v) {
    if (L<=l&&r<=R) { ans=min(ans,SPLAY::nex(root[rt],v)); return; }
    int m=l+r>>1;
    if (L<=m) nex(lson,L,R,v);
    if (m<R) nex(rson,L,R,v);
}
int kth(int L,int R,int k) {
    int l=0,r=MAX+1;
    while (l+1<r) {
        int m=l+r>>1;
        ans=rankv(l,l,n,L,R,m);
        if (ans<k) l=m;
        else r=m;
    }
    return l;
}
}
int main() {
    scanf("%d%d",&n,&m);
    for (int i=1;i<=n;++i) {
        scanf("%d",&a[i]);
        SEG::add(1,1,n,i,a[i]);
        if (a[i]>MAX) MAX=a[i];
    }
    for (int i=1;i<=m;++i) {
        int op,x,y,z;
        scanf("%d%d%d",&op,&x,&y);
        if (op==1) scanf("%d",&z),printf("%d\n",SEG::rankv(1,1,n,x,y,z)+1);
        if (op==2) scanf("%d",&z),printf("%d\n",SEG::kth(x,y,z));
        if (op==3) SEG::update(1,1,n,x,y);
        if (op==4) scanf("%d",&z),ans=-inf,SEG::pre(1,1,n,x,y,z),printf("%d\n",ans);
        if (op==5) scanf("%d",&z),ans= inf,SEG::nex(1,1,n,x,y,z),printf("%d\n",ans);
    }
    return 0;
}

```

珂朵莉树

```

#define IT set<odtnode>::iterator
struct odtnode {
    int l,r; mutable ll v;
    odtnode(int l,int r=-1,ll v=0):l(l),r(r),v(v) {}
    bool operator<(const odtnode& b) const { return l<b.l; }
};
set<odtnode> s;
inline ll pow_mod(ll a,ll b,ll m) {
    ll res=1;
    while (b) {
        if (b&1) res=res*a%m;
        a=a*a%m;
        b>>=1;
    }
}

```

```

    }
    return res;
}
inline IT split(int pos) {
    IT it=s.lower_bound(odtnode(pos));
    if (it!=s.end() && it->l==pos) return it;
    --it;
    int l=it->l, r=it->r; ll v=it->v;
    s.erase(it);
    s.insert(odtnode(l, pos-1, v));
    return s.insert(odtnode(pos, r, v)).first;
}
inline void add(int l, int r, ll v) {
    IT itr=split(r+1), itl=split(l);
    for (; itl!=itr; ++itl) itl->v+=v;
}
inline void update(int l, int r, ll v) {
    IT itr=split(r+1), itl=split(l);
    s.erase(itl, itr);
    s.insert(odtnode(l, r, v));
}
inline ll ranks(int l, int r, int k) {
    vector<pair<ll, int> > v(0);
    IT itr=split(r+1), itl=split(l);
    for (; itl!=itr; ++itl)
        v.push_back(make_pair(itl->v, itl->r-itl->l+1));
    sort(v.begin(), v.end());
    for (int i=0; i<v.size(); i++) {
        k-=v[i].second;
        if(k<=0) return v[i].first;
    }
}
inline ll sum(int l, int r, int n, int m) {
    IT itr=split(r+1), itl=split(l);
    ll res=0;
    for (; itl!=itr; ++itl)
        res=(res+1ll*(itl->r-itl->l+1)*pow_mod(itl->v, n, m))%mod;
    return res;
}

```

DFS序

DFS序是指DFS搜索过程对应的节点序列，DFS序列有多种

规定dep[]为深度，in[]为入栈时序，out[]为出栈时序，fa[]=dp[][0]为父节点，操作都是树状数组的

对于树：A[B[C,D], E]

DFS序：ABCDE

```
//      A  B  C  D  E
//in    1  2  3  4  5
//out   5  4  3  4  5
int in[maxn],out[maxn],idx;
void dfs(int u,int f) {
    in[u]++;idx;
    for (int i=head[u];~i;i=e[i].next) {
        int v=e[i].v;
        if (v==f) continue;
        dfs(v,u);
    }
    out[u]=idx;
}
```

点x的子树连续且为[in[x],out[x]]

- 单点修改，子树求和

```
add(in[x],w);
query(out[x])-query(in[x]-1);
```

- 子树修改，单点查询

```
add(in[x],w); add(out[x]+1,-w);
query(in[x])
```

- 子树修改，子树求和

```
add_xy(in[x],out[x],w);
query_xy(in[x],out[x]);
```

- 单点修改，树链求和

点存到根的距离，单点修改其子树都修改，转变为子树修改，单点查询

```
add(in[x],w); add(out[x]+1,-w);
int lcauv=lca(u,v);
query(in[u])+query(in[v])-query(in[lcauv])-query(in[fa[lcauv]]);
```

- 子树修改，树链求和

点存到根的距离，点x对其子节点y有贡献，且贡献为 $w[x] * (dep[y] - dep[x] + 1)$

分离变量得 $w[x] * (dep[y] + 1) - w[x] * dep[x]$ ，转变为子树修改，单点查询

```
inline void add_uv(int x,int w) {
    add(t1,in[x],w);
    add(t1,out[x]+1,-w);
    add(t2,in[x],-w*dep[x]);
    add(t2,out[x],w*dep[x]);
}
```

```

inline ll query_uv(int u,int v) {
    int lcauv=lca(u,v);
    ll res=0;
    res+=(dep[u]+1)*query(t1,in[u]);
    res+=(dep[v]+1)*query(t1,in[v]);
    res-=(dep[lcauv]+1)*query(t1,in[lcauv]);
    res-=(dep[lcauv])*query(t1,in[fa[lcauv]]);

    res+=query(t2,in[u]);
    res+=query(t2,in[v]);
    res-=query(t2,in[lcauv]);
    res-=query(t2,in[fa[lcauv]]);
    return res;
}

```

- 树链修改，单点查询

点存对其祖先贡献，单点修改其祖先都修改，转变为单点修改，子树和查询

```

int lcauv=lca(u,v);
add(in[u],w); add(in[v],w); add(in[lcauv],-w); add(in[fa[lcauv]],-w)
query(out[x])-query(in[x]-1)

```

- 树链修改，子树求和

点存对其祖先贡献，点x在y的子树内会产生贡献，且贡献为 $w[x]*(dep[x]-dep[y]+1)$

分离变量得 $w[x]*dep[x]+(-w[x])*(dep[y]-1)$ ，转变为单点修改，子树和查询

```

inline void add_uv(int u,int v,int w) {
    int lcauv=lca(u,v);
    add(t1,in[u],w*dep[u]);
    add(t1,in[v],w*dep[v]);
    add(t1,in[lcauv],-w*dep[lcauv]);
    add(t1,in[fa[lcauv]],-w*dep[fa[lcauv]]);
    add(t2,in[u],-w);
    add(t2,in[v],-w);
    add(t2,in[lcauv],w);
    add(t2,in[fa[lcauv]],w);
}

inline ll query_uv(int x) {
    ll res=0;
    res+=query(t1,out[x]);
    res-=query(t1,in[x]-1);
    res+=(dep[x]-1)*query(t2,out[x]);
    res-=(dep[x]-1)*query(t2,in[x]-1);
    return res;
}

```

欧拉序: ABCCDDDBEEA


```
//      A  B  C  D  E
//in    1  2  3  5  8
//out  10  7  3  6  9
int in[maxn],out[maxn],idx;
void dfs(int u,int f) {
    in[u]++;idx;
    for (int i=head[u];~i;i=e[i].next) {
        int v=e[i].v;
        if (v==f) continue;
        dfs(v,u);
    }
    outu[++]idx;
}
```

树链剖分

1. dfs1处理fa父节点, sz子树大小, dep深度, son重儿子
2. dfs2处理top链顶, idx时间戳, rid映射idx
3. $[idx[u], idx[u]+sz[u]-1]$ 是u的子树在线段树中的区间

```
int fa[maxn],sz[maxn],dep[maxn],son[maxn];
int top[maxn],idx[maxn],rid[maxn],tot;
inline void init() { tot=0; }
inline void dfs1(int u,int f,int d) {
    fa[u]=f; sz[u]=1; dep[u]=d; son[u]=0;
    for (int i=head[u];~i;i=e[i].next) {
        int v=e[i].v;
        if (v==f) continue;
        dfs1(v,u,d+1);
        sz[u]+=sz[v];
        if (sz[son[u]]<sz[v]) son[u]=v;
    }
}
inline void dfs2(int u,int rt) {
    top[u]=rt; idx[u]++;tot; rid[tot]=u;
    if (!son[u]) return;
    dfs2(son[u],rt);
    for (int i=head[u];~i;i=e[i].next) {
        int v=e[i].v;
        if (v==son[u]||v==fa[u]) continue;
        dfs2(v,v);
    }
}
//点权维护
inline void update_chain(int x,int y,int v) {
    while (top[x]!=top[y]) {
        if (dep[top[x]]<dep[top[y]]) swap(x,y);
        update(1,1,n,idx[top[x]],idx[x],v);
        x=fa[top[x]];
    }
    if (dep[x]>dep[y]) swap(x,y);
}
```

```

    update(1,1,n,idx[x],idx[y],v);
}
inline ll query_chain(int x,int y) {
    int res=0;
    while (top[x]!=top[y]) {
        if (dep[top[x]]<dep[top[y]]) swap(x,y);
        res+=query(1,1,n,idx[top[x]],idx[x]);
        x=fa[top[x]];
    }
    if (dep[x]>dep[y]) swap(x,y);
    res+=query(1,1,n,idx[x],idx[y]);
    return res;
}
//边权维护 把边的权值落到其深度大的节点上,求链时不能访问LCA节点
inline void update_chain(int x,int y) {
    while (top[x]!=top[y]) {
        if (dep[top[x]]<dep[top[y]]) swap(x,y);
        update(1,1,n,idx[top[x]],idx[x]);
        x=fa[top[x]];
    }
    if (dep[x]>dep[y]) swap(x,y);
    if (x!=y) update(1,1,n,idx[x]+1,idx[y]);
}
inline int query_chain(int x,int y) {
    if (x==y) return 0;
    int res=0xf0f0f0f0;
    while (top[x]!=top[y]) {
        if (dep[top[x]]<dep[top[y]]) swap(x,y);
        res=max(res,query(1,1,n,idx[top[x]],idx[x]));
        x=fa[top[x]];
    }
    if (dep[x]>dep[y]) swap(x,y);
    if (x!=y) res=max(res,query(1,1,n,idx[x]+1,idx[y]));
    return res;
}

```

LG3384

- 1 x y z x到y的路径上所有节点的值加z
- 2 x y 求x到y的路径上所有节点的和
- 3 x z 将x的子树内的所有节点值加z
- 4 x 求x的子树内的所有节点值的和

```

#include<bits/stdc++.h>
using namespace std;
#define lson    rt<<1,l,m
#define rson    rt<<1|1,m+1,r
const int inf=0x3f3f3f3f;
int mod=1e9+7;
const int maxn=1e5+5;

```

```

const int maxm=1e6+5;
int a[maxn];
int head[maxn],nume;
struct edge { int v,next; } e[maxm];
void init_edge() { memset(head,-1,sizeof head); nume=0; }
void add_edge(int u,int v) { e[nume].v=v; e[nume].next=head[u]; head[u]=nume++; }

int fa[maxn],sz[maxn],dep[maxn],son[maxn];
int top[maxn],idx[maxn],rid[maxn],tot;
inline void dfs1(int u,int f,int d) {
    fa[u]=f; sz[u]=1; dep[u]=d;
    for (int i=head[u];~i;i=e[i].next) {
        int v=e[i].v;
        if (v==f) continue;
        dfs1(v,u,d+1);
        sz[u]+=sz[v];
        if (sz[son[u]]<sz[v]) son[u]=v;
    }
}
inline void dfs2(int u,int ff) {
    top[u]=ff; idx[u]=++tot; rid[tot]=u;
    if (!son[u]) return;
    dfs2(son[u],ff);
    for (int i=head[u];~i;i=e[i].next) {
        int v=e[i].v;
        if (v==son[u]||v==fa[u]) continue;
        dfs2(v,v);
    }
}

int sum[maxn<<2],lz[maxn<<2];
inline void push_up(int rt) { sum[rt]=(sum[rt<<1]+sum[rt<<1|1])%mod; }
inline void push_down(int rt,int l,int r) {
    if (!lz[rt]) {
        int m=(l+r)>>1;
        sum[rt<<1]+=(m-l+1)*lz[rt]; sum[rt<<1]%mod;
        sum[rt<<1|1]+=(r-m)*lz[rt]; sum[rt<<1|1]%mod;
        lz[rt<<1]+=lz[rt];
        lz[rt<<1|1]+=lz[rt];
        lz[rt]=0;
    }
}
inline void build(int rt,int l,int r){
    if(l==r){ sum[rt]=a[ rid[l] ]%mod; return; }
    int m=(l+r)>>1;
    build(lson);
    build(rson);
    push_up(rt);
}
inline void update(int rt,int l,int r,int L,int R,int v) {
    if (L<=l&&r<=R) { sum[rt]=(sum[rt]+(r-l+1)*v)%mod; lz[rt]=(lz[rt]+v)%mod; return; }
    push_down(rt,l,r);
    int m=(l+r)>>1;

```

```

    if (L<=m) update(lson,L,R,v);
    if (m<R) update(rson,L,R,v);
    push_up(rt);
}

inline int query(int rt,int l,int r,int L,int R) {
    if (L<=l&&r<=R) return sum[rt] ;
    push_down(rt,l,r);
    int m=l+r>>1;
    int res=0;
    if (L<=m) res+=query(lson,L,R);
    if (m<R) res+=query(rson,L,R);
    return res%mod;
}

int n,m,r;
inline void update_chain(int x,int y,int v) {
    while (top[x]!=top[y]) {
        if (dep[top[x]]<dep[top[y]]) swap(x,y);
        update(1,1,n,idx[top[x]],idx[x],v);
        x=fa[top[x]];
    }
    if (dep[x]>dep[y]) swap(x,y);
    update(1,1,n,idx[x],idx[y],v);
}

inline int query_chain(int x,int y) {
    int res=0;
    while (top[x]!=top[y]) {
        if (dep[top[x]]<dep[top[y]]) swap(x,y);
        res+=query(1,1,n,idx[top[x]],idx[x]);
        res%=mod;
        x=fa[top[x]];
    }
    if (dep[x]>dep[y]) swap(x,y);
    res+=query(1,1,n,idx[x],idx[y]);
    return res%mod;
}

int main () {
    init_edge();
    scanf("%d%d%d%d",&n,&m,&r,&mod);
    for (int i=1;i<=n;++i) scanf("%d",&a[i]);
    for (int i=1;i<n;++i) {
        int u,v;
        scanf("%d%d",&u,&v);
        add_edge(u,v);
        add_edge(v,u);
    }
    dfs1(r,0,1);
    dfs2(r,r);
    build(1,1,n);

    for (int i=1;i<=m;++i) {
        int op,x,y,w;

```

```

scanf("%d",&op);
if (op==1) {
    scanf("%d%d%d",&x,&y,&w);
    update_chain(x,y,w);
}
if (op==2) {
    scanf("%d%d",&x,&y);
    printf("%d\n",query_chain(x,y));
}
if (op==3) {
    scanf("%d%d",&x,&w);
    update(1,1,n,idx[x],idx[x]+sz[x]-1,w);
}
if (op==4) {
    scanf("%d",&x);
    printf("%d\n",query(1,1,n,idx[x],idx[x]+sz[x]-1));
}
}
return 0;
}

```

POJ3237

- 1 i v 将第i条边的权值改为v
- 2 x y 将x到y的最短路上边的权值取反
- 3 x y 查询x到y的最短路上的的最大边权

```

#include<cstdio>
#include<cstring>
#include<algorithm>
#define lson rt<<1,l,m
#define rson rt<<1|1,m+1,r
using namespace std;
const int maxn=1e4+5;

int head[maxn],nume;
struct edge { int v,next; } e[maxn];
void init_edge() { memset(head,-1,sizeof head); nume=0; }
void add_edge(int u,int v) { e[nume].v=v; e[nume].next=head[u]; head[u]=nume++; }
int maxv[maxn],minv[maxn],lz[maxn];
inline void push_up(int rt) {
    maxv[rt]=max(maxv[rt<<1],maxv[rt<<1|1]);
    minv[rt]=min(minv[rt<<1],minv[rt<<1|1]);
}
inline void push_down(int rt) {
    if (lz[rt]) {
        lz[rt<<1]^=lz[rt];
        swap(maxv[rt<<1],minv[rt<<1]);
        maxv[rt<<1]=-maxv[rt<<1];
        minv[rt<<1]=-minv[rt<<1];
    }
}

```

```

        lz[rt<<1|1]^=lz[rt];
        swap(maxv[rt<<1|1],minv[rt<<1|1]);
        maxv[rt<<1|1]=-maxv[rt<<1|1];
        minv[rt<<1|1]=-minv[rt<<1|1];
        lz[rt]=0;
    }
}

inline void change(int rt,int l,int r,int p,int v) {
    if (l==r) { maxv[rt]=minv[rt]=v; return; }
    push_down(rt);
    int m=(l+r)>>1;
    if (p<=m) change(lson,p,v);
    else      change(rson,p,v);
    push_up(rt);
}

inline void update(int rt,int l,int r,int L,int R) {
    if (L<=l&&r<=R) {
        lz[rt]^=1;
        swap(maxv[rt],minv[rt]);
        maxv[rt]=-maxv[rt];
        minv[rt]=-minv[rt];
        return;
    }
    push_down(rt);
    int m=l+r>>1;
    if (L<=m) update(lson,L,R);
    if (m<R)  update(rson,L,R);
    push_up(rt);
}

inline int query(int rt,int l,int r,int L,int R) {
    if (L<=l&&r<=R) return maxv[rt];
    push_down(rt);
    int m=l+r>>1;
    if (R<=m) return query(lson,L,R);
    if (m<L)  return query(rson,L,R);
    return max(query(lson,L,R),query(rson,L,R));
}

int n,m,x,y,u[maxn],v[maxn],w[maxn];
int fa[maxn],sz[maxn],dep[maxn],son[maxn];
int top[maxn],idx[maxn],rid[maxn],tot;
inline void dfs1(int u,int f,int d) {
    fa[u]=f; sz[u]=1; dep[u]=d; son[u]=0;
    for (int i=head[u];~i;i=e[i].next) {
        int v=e[i].v;
        if (v==f) continue;
        dfs1(v,u,d+1);
        sz[u]+=sz[v];
        if (sz[son[u]]<sz[v]) son[u]=v;
    }
}

inline void dfs2(int u,int rt) {
    top[u]=rt; idx[u]++;tot; rid[tot]=u;

```

```

    if (!son[u]) return;
    dfs2(son[u],rt);
    for (int i=head[u];~i;i=e[i].next) {
        int v=e[i].v;
        if (v==fa[u]||v==son[u]) continue;
        dfs2(v,v);
    }
}

inline void update_chain(int x,int y) {
    while (top[x]!=top[y]) {
        if (dep[top[x]]<dep[top[y]]) swap(x,y);
        update(1,1,n,idx[top[x]],idx[x]);
        x=fa[top[x]];
    }
    if (dep[x]>dep[y]) swap(x,y);
    if (x!=y) update(1,1,n,idx[x]+1,idx[y]);
}

inline int query_chain(int x,int y) {
    if (x==y) return 0;
    int res=0xf0f0f0f0;
    while (top[x]!=top[y]) {
        if (dep[top[x]]<dep[top[y]]) swap(x,y);
        res=max(res,query(1,1,n,idx[top[x]],idx[x]));
        x=fa[top[x]];
    }
    if (dep[x]>dep[y]) swap(x,y);
    if (x!=y) res=max(res,query(1,1,n,idx[x]+1,idx[y]));
    return res;
}

int main() {
    int T;
    scanf("%d",&T);
    while (T--) {
        init_edge(); tot=0;
        memset(lz,0,sizeof lz);
        scanf("%d",&n);
        for (int i=1;i<n;i++) {
            scanf("%d%d%d",&u[i],&v[i],&w[i]);
            add_edge(u[i],v[i]);
            add_edge(v[i],u[i]);
        }
        dfs1(1,0,0);
        dfs2(1,1);
        for (int i=1;i<n;i++) {
            if (dep[u[i]]<dep[v[i]]) swap(u[i],v[i]);
            change(1,1,n,idx[u[i]],w[i]);
        }
        while (1) {
            char s[11];
            scanf(" %s",s);
            if (s[0]=='D') break;
            scanf("%d%d",&x,&y);
            if (s[0]=='Q') printf("%d\n",query_chain(x,y));
        }
    }
}

```

```

        if (s[0]=='N') update_chain(x,y);
        if (s[0]=='C') change(1,1,n,idx[u[x]],y);
    }
}
return 0;
}

```

动态树

bzoj2049

Query x y 查询两个点是否有边

Connect x y 连接两个点

Destroy x y 断开两个点的边

```

#include<cstdio>
#include<stdlib.h>
using namespace std;
int _;
struct T{
    int l,r,f;
    bool rev;
    void swap(){
        _=l;
        l=r;
        r=_;
    }
}tree[10005];
void add_rev(int x)
{
    tree[x].swap();
    tree[x].rev=!tree[x].rev;
}
bool root(int x)
{
    int f=tree[x].f;
    return tree[f].l!=x&&tree[f].r!=x;
}
void down(int x)
{
    if(tree[x].rev)
    {
        add_rev(tree[x].l);
        add_rev(tree[x].r);
        tree[x].rev=false;
    }
}
void clean(int x)
{
    int f=tree[x].f;
    // printf("clean->%d\n",x);
}

```



```

    if(root(x))
    {
        down(x);
        return;
    }
    clean(f);
    down(x);
}

void up1(int wei)
{
    if(tree[wei].f==0)
        return;
    int f=tree[wei].f;
    if(wei==tree[f].l)
    {
        tree[wei].f=tree[f].f;
        int F=tree[f].f;
        if(tree[F].l==f)
            tree[F].l=wei;
        else
            if(tree[F].r==f)
                tree[F].r=wei;
        tree[f].f=wei;
        tree[f].l=tree[wei].r;
        if(tree[wei].r!=-1)
            tree[tree[wei].r].f=f;
        tree[wei].r=f;
    }
    else
    {
        tree[wei].f=tree[f].f;
        int F=tree[f].f;
        if(tree[F].l==f)
            tree[F].l=wei;
        else
            if(tree[F].r==f)
                tree[F].r=wei;
        tree[f].f=wei;
        tree[f].r=tree[wei].l;
        if(tree[wei].l!=-1)
            tree[tree[wei].l].f=f;
        tree[wei].l=f;
    }
}

void splay(int x)
{
    clean(x);
    while(!root(x))
    {
        up1(x);
    }
    // printf("splay->%d\n",x);
}

```

```

void access(int x)
{
    splay(x);
    tree[x].r=-1;
    while(tree[x].f>0)
    {
        int f=tree[x].f;
        splay(f);
        tree[f].r=x;
        splay(x);
    }
    // printf("access->%d\n",x);
}
void mackroot(int x)
{
    access(x);
    add_rev(x);
}
int findroot(int x)
{
    access(x);
    down(x);
    while(tree[x].l>0)
    {
        x=tree[x].l;
        down(x);
    }
    return x;
}
void link(int x,int y)
{
    mackroot(y);
    tree[y].f=x;
}
void cut(int x,int y)
{
    mackroot(y);
    down(y);
    splay(x);
    if(tree[x].l!=y)
    {
        tree[x].f=0;
    }
    else
    {
        tree[y].f=0;
        tree[x].l=-1;
    }
}
int main()
{
    T null_tree;
    null_tree.l=null_tree.r=-1;
}

```

```

null_tree.f=0;
null_tree.rev=false;
int n,m;
char c[20];
int x,y;
scanf("%d%d",&n,&m);
for(int i=1;i<=n;i++)
{
    tree[i]=null_tree;
}
while(m--)
{
    scanf("%s",c);
    scanf("%d%d",&x,&y);
    switch(c[0])
    {
        case 'C':
            link(x,y);
            break;
        case 'D':
            cut(x,y);
            break;
        case 'Q':
            if(findroot(x)==findroot(y))
                printf("Yes\n");
            else
                printf("No\n");
            break;
    }
}
}

```