ICPC Template Manual

University of Jinan

Awesome

May 22, 2022

Contents

0

1 Math

1.1 素数

1.1.1 质因数分解

```
vector<pair<ll, int>> getFactors(ll x) {
    vector<pair<ll, int>> fact;
    for (int i = 0; prime[i] <= x / prime[i]; i++) {
        if (x % prime[i] == 0) {
            fact.emplace_back(prime[i], 0);
            while (x % prime[i] == 0) fact.back().second++, x /= prime[i];
        }
    }
    if (x != 1) fact.emplace_back(x, 1);
    return fact;
}
```

1.1.2 Miller Rabin 素数测试

$O(s \log n)$ 内判定 2^{63} 内的数是不是素数，s 为测定次数

```
bool Miller_Rabin(ll n, int s) { // s : 判定次数, 一般 8 ~ 10 次就够
    if (n == 2) return 1;
    if (n < 2 || !(n & 1)) return 0;
    int t = 0;
    ll  x, y, u = n - 1;
    while ((u & 1) == 0) t++, u >>= 1;
    for (int i = 0; i < s; i++) {
        ll a = rand() % (n - 1) + 1;
        ll x = qpow(a, u, n);   // 快速幂(套龟速乘)
        for (int j = 0; j < t; j++) {
            ll y = smul(x, x, n);   // 龟速乘
            if (y == 1 && x != 1 && x != n - 1) return 0;
            x = y;
        }
        if (x != 1) return 0;
    }
    return 1;
}
```

1.1.3 区间筛

对区间 $[a, b)$ 内的整数执行筛法。
函数返回区间内素数个数
`is_prime[i-a]=true` 表示 i 是素数
$1 < a < b \leq 10^{12}, b - a \leq 10^6$

```
const int N = "Edit";
bool is_prime_small[N], is_prime[N];
ll prime[N];
int segment_sieve(ll a, ll b) {
    int tot = 0;
    for (ll i = 0; i * i < b; ++i) is_prime_small[i] = true;
    for (ll i = 0; i < b - a; ++i) is_prime[i] = true;
    for (ll i = 2; i * i < b; ++i)
        if (is_prime_small[i]) {
            for (ll j = 2 * i; j * j < b; j += i)
```

```
                is_prime_small[j] = false;
            for (ll j = max(2LL, (a + i - 1) / i) * i; j < b; j += i)
                is_prime[j - a] = false;
        }
    for (ll i = 0; i < b - a; ++i)
        if (is_prime[i]) prime[tot++] = i + a;
    return tot;
}
```

1.1.4 杜教筛

```
#define N 6000000 //杜教筛求解phi和mu前缀和
ll phi[N], p[N];
int mu[N], vis[N], tot;
unordered_map<int, int> sum_mu;
unordered_map<int, ll> sum_phi;
inline void init() {
    mu[1] = phi[1] = 1;
    for (int i = 2; i < N; ++i) {
        if(!vis[i]) { p[++tot] = i; mu[i] = -1; phi[i] = i - 1;}
        for (int j = 1; j <= tot && i * p[j] < N; ++j) {
            vis[i * p[j]] = 1;
            if (i % p[j] == 0) { phi[i * p[j]] = phi[i] * p[j]; break;}
            else { mu[i * p[j]] = -mu[i]; phi[i * p[j]] = phi[i] * phi[p[j]];}
        }
    }
    for (int i = 1; i < N; ++i) { //预处理线性时间可处理的前缀和
        mu[i] += mu[i - 1]; phi[i] += phi[i - 1];
    }
}
inline int get_smu(ll x) {
    if (x < N) return mu[x];
    if (sum_mu[x]) return sum_mu[x];
    int ans = 1;
    for (ll l = 2, r; l <= x; l = r + 1) { //从2开始!!!!!!!!!!!!!!!!!!!!!!!!!
        r = x / (x / l);
        ans -= (r - l + 1) * get_smu(x / l);
    }
    return sum_mu[x] = ans;
}
inline ll get_sphi(ll x) {
    if (x < N) return phi[x];
    if (sum_phi[x]) return sum_phi[x];
    ll ans = 1ll * x * (x + 1ll) / 2ll;
    for (ll l = 2, r; l <= x; l = r + 1) {
        r = x / (x / l);
        ans -= 1ll * (r - l + 1) * get_sphi(x / l);
    }
    return sum_phi[x] = ans;
}
```

1.2 欧拉函数

1.2.1 函数

```
ll euler(ll n) { // 小于等于 n 和 n 互质的数的个数
    ll rt = n;
    for (int i = 2; i * i <= n; i++)
```

```
        if (n % i == 0) {
            rt -= rt / i;
            while (n % i == 0) n /= i;
        }
    if (n > 1) rt -= rt / n;
    return rt;
}
```

1.2.2 埃式筛

```
const int N = "Edit";
int phi[N] = {0, 1};
void caleuler() {
    for (int i = 2; i < N; i++)
        if (!phi[i])
            for (int j = i * i; j < N; j += i) { // i * i 优化
                if (!phi[j]) phi[j] = j;
                phi[j] = phi[j] / i * (i - 1);
            }
}
```

1.2.3 线性筛

```
inline void init() {
    mu[1] = phi[1] = 1;
    for (int i = 2; i < N; ++i) {
        if(!vis[i]) {
            p[++tot] = i;
            mu[i] = -1;
            phi[i] = i - 1;
        }
        for (int j = 1; j <= tot && i * p[j] < N; ++j) {
            vis[i * p[j]] = 1;
            if (i % p[j] == 0) {
                phi[i * p[j]] = phi[i] * p[j];
                break;
            } else {
                mu[i * p[j]] = -mu[i];
                phi[i * p[j]] = phi[i] * phi[p[j]];
            }
        }
    }
    for (int i = 1; i < N; ++i) { //预处理线性时间可处理的前缀和
        mu[i] += mu[i - 1];       //其余的留给后面处理
        phi[i] += phi[i - 1];
    }
}
```

1.3 筛法

1.3.1 埃式筛

$O(n \log\log n)$ 筛出 N 内所有素数
$nprime[i] = 0/1$ 0 为素数 1 为非素数

```
const int N = "Edit";
bool nprime[N] = {1, 1};   // 0 && 1 为非素数
void GetPrime() {
    for (int i = 2; i < N; ++i)
        if (!nprime[i] && i <= N / i)  // 筛到√n为止
            for (int j = i * i; j < N; j += i) nprime[j] = 1;
}
```

1.3.2 欧拉筛

$O(n)$ 得到欧拉函数 $npri[]$、素数表 $pri[]$、素数个数 cnt

```
const int N = "Edit";
bool vis[N];
int cnt, npri[N], pri[N];
void CalPhi() {
    npri[1] = 1;
    for (int i = 2; i < N; ++i) {
        if (!vis[i]) {
            pri[++cnt] = i;
            npri[i] = i - 1;
        }
        for (int j = 1; j < cnt && i * pri[j] <= N; ++j) {
            if (i * pri[j] > N) break;
            vis[i * pri[j]] = 1;
            if (i % pri[j] == 0) {
                npri[i * pri[j]] = npri[i] * pri[j];
                break;
            }
            else
                npri[i * pri[j]] = npri[i] * (pri[j] - 1);
        }
    }
}
```

$d(n)$ 函数

```
const int N = "Edit";
int pri[N], cnt;
int d[N], e[N]; //d正除数个数，e最小质因子个数
bool check[N];
void CalD() {
    d[1] = 1;
    for (int i = 2; i < N; i++) {
        if (!check[i]) {
            pri[cnt++] = i;
            e[i] = 1, d[i] = 2;
        }
        for (int j = 0; j < cnt; j++) {
            if (i * pri[j] >= N) break;
            check[i * pri[j]] = true;
            if (i % pri[j] == 0) {
                e[i * pri[j]] = e[i] + 1;
                d[i * pri[j]] = d[i] / e[i] * (e[i] + 1);
                break;
            }
            else {
                e[i * pri[j]] = 1;
```

```
                d[i * pri[j]] = 2 * d[i];
            }
        }
    }
}
```

$\sigma\lambda(n)$ 函数，$\lambda = 1$

```
const int N = "Edit";
int pri[N], cnt;
int sig[N], e[N]; //sig正除数，e不含能整除i的最小质因子的正除数和
bool check[N];
void CalSig() {
    sig[1] = 1;
    for (int i = 2; i < N; i++) {
        if (!check[i]) {
            pri[cnt++] = i;
            e[i] = 1, sig[i] = i + 1;
        }
        for (int j = 0; j < cnt; j++) {
            if (i * pri[j] >= N) break;
            check[i * pri[j]] = true;
            if (i % pri[j] == 0) {
                sig[i * pri[j]] = sig[i] * pri[j] + e[i];
                e[i * pri[j]] = e[i];
                break;
            }
            else {
                sig[i * pri[j]] = sig[i] * (pri[j] + 1);
                e[i * pri[j]] = sig[i];
            }
        }
    }
}

int np[N], pri[N], n, cnt;
void Calpri() {
    np[1] = 1;
    for (int i = 2; i <= n; ++i) {
        if (!np[i]) pri[++cnt] = i;
        for (int j = 1; j <= cnt && i * pri[j] <= n; ++j) {
            np[i * pri[j]] = 1;
            if (!(i % pri[j])) break;
        }
    }
}
```

1.3.3 筛法求四类积性函数

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
#define N 100000

bool vis[N];
int p[N], mu[N], tot;
ll phi[N];
int num[N], d[N];
```

```
ll sig[N], g[N];

inline void seive() {
    mu[1] = phi[1] = sig[1] = g[1] = d[1] = 1;
    for (int i = 2; i < N; ++i) {
        if (!vis[i]) {
            p[++tot] = i;
            mu[i] = -1;
            phi[i] = i - 1;
            num[i] = 1, d[i] = 2;
            g[i] = sig[i] = i + 1;
        }
        for (int j = 1; j <= tot && i * p[j] < N; ++j) {
            vis[i * p[j]] = 1;
            if (i % p[j] == 0) {
                phi[i * p[j]] = phi[i] * p[j];
                num[i * p[j]] = num[i] + 1;
                d[i * p[j]] = d[i] / num[i * p[j]] * (num[i * p[j]] + 1);
                g[i * p[j]] = g[i] * p[j] + 1;
                sig[i * p[j]] = sig[i] / g[i] * g[i * p[j]];
                break;
            }
            mu[i * p[j]] = - mu[i];
            phi[i * p[j]] = phi[i] * phi[p[j]];
            num[i * p[j]] = 1;
            d[i * p[j]] = d[i] * 2;
            g[i * p[j]] = g[i] * (1 + p[j]);
            sig[i * p[j]] = sig[i] * sig[p[j]];
        }
    }
}
int main() {
    seive();
    for (int i = 1; i < 20; ++i) {
        cout << i << " : " << mu[i] << ' ' << phi[i] << ' ' << d[i] << ' ' << sig[i] <<
     endl;
    }
    return 0;
}
```

1.3.4 $\mathbf{Min}_2 5 sieve$

```
//sieve sum phi && sum mu
#include <bits/stdc++.h>
#define LL long long
#define N (1000005)
using namespace std;
int T,n,pri,t,unit;
int prime[N],q[N],id1[N],id2[N];
LL g[N],h[N],sum[N];
bool nprime[N];
template <typename T> void read(T&t) {
    t=0;
    bool fl=true;
    char p=getchar();
    while (!isdigit(p)) {
        if (p=='-') fl=false;
        p=getchar();
```

```
17      }
18      do {
19          (t*=10)+=p-48;p=getchar();
20      }while (isdigit(p));
21      if (!fl) t=-t;
22  }
23  inline void init(){
24      pri=t=0;
25      unit=sqrt(n);
26      for (int i=2;i<=unit;i++){
27          if (!nprime[i]) prime[++pri]=i,sum[pri]=sum[pri-1]+i;
28          for (int j=1;j<=pri&&i*prime[j]<=unit;j++){
29              nprime[i*prime[j]]=1;
30              if (i%prime[j]==0) break;
31          }
32      }
33      for (int i=1;i<=n;){
34          int v=n/i,R=n/v;
35          q[++t]=v;
36          if (v<=unit) id1[v]=t;
37          else id2[R]=t;
38          g[t]=1ll*v*(v+1)/2-1;
39          h[t]=v-1;
40          i=R+1;
41      }
42  }
43  inline int id(int x){
44      return (x<=unit)?id1[x]:id2[n/x];
45  }
46  LL S(int n,int m){
47      if (n<=1||prime[m]>n) return 0;
48      LL ret=g[id(n)]-sum[m-1]+m-1;
49      for (int k=m;prime[k]*prime[k]<=n&&k<=pri;k++){
50          for (int v=prime[k],p1=prime[k]-1;1ll*v*prime[k]<=n;v=v*prime[k],p1=p1*prime[k
        ])
51          ret+=(S(n/v,k+1)+prime[k])*p1;
52      }
53      return ret;
54  }
55  LL D(int n,int m){
56      if (n<=1||prime[m]>n) return 0;
57      LL ret=h[id(n)]+(m-1);
58      for (int k=m;prime[k]*prime[k]<=n&&k<=pri;k++){
59          ret-=D(n/prime[k],k+1);
60      }
61      return ret;
62  }
63  signed main(){
64      read(T);
65      while (T--){
66          read(n);
67          init();
68          for (int i=1;i<=pri;i++){
69              int v=prime[i]*prime[i];
70              for (int j=1;j<=t&&v<=q[j];j++){
71                  int k=id(q[j]/prime[i]);
72                  g[j]-=1ll*prime[i]*(g[k]-sum[i-1]);
73                  h[j]-=h[k]-i+1;
74              }
```

```
        }
        for (int i=1;i<=t;i++) g[i]-=h[i],h[i]=-h[i];
        printf("%lld %lld\n",S(n,1)+1,D(n,1)+1);
    }
    return 0;
}
```

1.3.5 线性筛约数和

```
#include <iostream>

using namespace std;
int g[100999], f[100000], p[100000], tot, n = 100000, v[100000];

void pre() {
    g[1] = f[1] = 1;
    for (int i = 2; i <= n; ++i) {
        if (!v[i]) v[i] = 1, p[++tot] = i, g[i] = i + 1, f[i] = i + 1;
        for (int j = 1; j <= tot && i <= n / p[j]; ++j) {
            v[p[j] * i] = 1;
            if (i % p[j] == 0) {
                g[i * p[j]] = g[i] * p[j] + 1;
                f[i * p[j]] = f[i] / g[i] * g[i * p[j]];
                break;
            } else {
                f[i * p[j]] = f[i] * f[p[j]];
                g[i * p[j]] = 1 + p[j];
            }
        }
    }
    //for (int i = 1; i <= n; ++i) f[i] = (f[i - 1] + f[i]);
}
int main(int argc, char *argv[]) {
    pre();
    cout << f[69073];
}
```

1.4 基本数论

1.4.1 exgcd

```
// 仅需满足 gcd(a, p) = 1 , p 可不为质数
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (!b) { x = 1, y = 0; return a; }
    ll d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
```

1.4.2 ax+by=c

引用返回通解: $X = x + k * dx, Y = y–k * dy$
引用返回的 x 是最小非负整数解，方程无解函数返回 0

```
#define Mod(a, b) (((a) % (b) + (b)) % (b))
bool solve(ll a, ll b, ll c, ll& x, ll& y, ll& dx, ll& dy) {
    if (a == 0 && b == 0) return 0;
```

```
    ll x0, y0;
    ll d = exgcd(a, b, x0, y0);
    if (c % d != 0) return 0;
    dx = b / d, dy = a / d;
    x = Mod(x0 * c / d, dx);
    y = (c - a * x) / b;
    //  y = Mod(y0 * c / d, dy); x = (c - b * y) / a;
    return 1;
}
```

1.4.3 乘法逆元

利用 exgcd 求 a 在模 p 下的逆元，需要保证 $\gcd(a,p) == 1$.

```
ll inv(ll a, ll m) {
    ll x, y;
    ll d = exgcd(a, p, x, y);
    return d == 1 ? (x + p) % p : -1;
}
```

$a < p$ 且 p 为素数时，有以下两种求法
费马小定理

```
ll inv(ll a, ll p) { return qpow(a, p - 2, p); }
```

线性筛

```
// 线性递推求解乘法逆元  条件: p 是奇质数
inline void inv_(int n, ll p) {
    inv[1] = 1;
    for (int i = 2; i <= n; ++i) {
        inv[i] = (p - p / i) * inv[p % i] % p;
    }
}
// 线性递推求阶乘逆元
void finv_(int n, ll p) {
    fac[1] = 1;
    for (int i = 2; i <= n; ++i) {
        fac[i] = fac[i - 1] * i % p;
    }
    // 费马小定理
    finv[n] = qpow(fac[n], p - 2, p);
    for (int i = n - 1; i; --i) {
        finv[i] = finv[i + 1] * (i + 1) % p;
    }
}
// 用线性逆元解阶乘逆元
void get_inv(int n, ll p) {
    // inv[0] = inv[1] = finv[1] = fac[1] = 1;
    inv[0] = inv[1] = finv[1] = 1;
    for (int i = 2; i <= n; ++i) {
        // fac[i] = fac[i - 1] * i % p; // 将 3 个一起求出来就可以用下面的组合数函数了
        inv[i] = (p - p / i) * inv[p % i] % p;
        finv[i] = finv[i - 1] * inv[i] % p;
    }
}
// 求组合数 C(a,b) 在 mod p 意义下的值
ll C(ll a, ll b, ll p) {
    return fac[b] * finv[a] % p * finv[b - a] % p;
}
```

1.4.4 Discrete Logarithm

求解 $a^x \equiv b \pmod{p}$，p 可以不是质数

```
ll exbsgs(ll a, ll b, ll p) {
    if (b == 1LL) return 0;
    ll t, d = 1, k = 0;
    while ((t = gcd(a, p)) != 1) {
        if (b % t) return -1;
        ++k, b /= t, p /= t, d = d * (a / t) % p;
        if (b == d) return k;
    }
    map<ll, ll> dic;
    ll m = ceil(sqrt(p));
    ll a_m = Pow(a, m, p), mul = b;
    for (ll j = 1; j <= m; ++j) mul = mul * a % p, dic[mul] = j;
    for (ll i = 1; i <= m; ++i) {
        d = d * a_m % p;
        if (dic[d]) return i * m - dic[d] + k;
    }
    return -1;
}
```

1.4.5 BSGS

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll a,b,p;
ll power(ll a, ll b, ll c) {//快速幂
    if(b==0) return 1%c;
    ll ans=1,t=a;
    while(b>0) {
        if(b%2==1) ans=ans*t%c;
        b/=2; t=t*t%c;
    }
    return ans;
}
ll bsgs(ll a,ll b,ll p) {//bsgs
    map<ll,ll> hash; hash.clear();//建立一个Hash表
    b%=p;
    ll t=sqrt(p)+1;
    for(ll i=0;i<t;++i) hash[(ll)b*power(a,i,p)%p]=i;//将每个j对应的值插入Hash表
    a=power(a,t,p);
    if(!a) return b==0?1:-1;//特判
    for(ll i=1;i<=t;++i) {//在Hash表中查找是否有i对应的j值
        ll val=power(a,i,p);
        int j=hash.find(val)==hash.end()?-1:hash[val];
        if(j>=0&&i*t-j>=0) return i*t-j;
    }
    return -1;//无解返回-1
}
signed main() {
    scanf("%lld%lld%lld",&p,&a,&b);
    ll ans=bsgs(a,b,p);
    if(ans==-1) printf("no solution\n");
    else printf("%lld\n",ans);
    return 0;
}
```

1.4.6 二次剩余

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
ll n, p, w;

struct num { ll x, y; };

inline num mul(num a, num b) {
    num ans = {0, 0};
    ans.x = ((a.x * b.x % p + a.y * b.y % p * w % p) + p) % p;
    ans.y = ((a.x * b.y % p + a.y * b.x % p) % p + p) % p;
    return ans;
}

inline ll qpow_real(ll a, ll b) {
    ll ans = 1;
    while (b) {
        if (b & 1) (ans *= a) %= p;
        b >>= 1, (a *= a) %= p;
    }
    return ans;
}

inline ll qpow_imag(num a, ll b) {
    num ans = {1, 0};
    while (b) {
        if (b & 1) ans = mul(ans, a);
        b >>= 1, a = mul(a, a);
    }
    return ans.x;
}

inline ll cipolla(ll n, ll p) {
    n %= p;
    if (qpow_real(n, (p-1) >> 1) == p - 1) return -1;
    ll a;
    while (1) {
        a = rand() % p;
        w = ((a * a % p - n) % p + p) % p;
        if (qpow_real(w, (p-1) >> 1) == p - 1) break;
    }
    num t = {a, 1};
    return qpow_imag(t, (p + 1) >> 1);
}

int main() {
    int t; cin >> t;
    while (t--) {
        cin >> n >> p;
        if (n==0) {
            cout << 0 << endl;
            continue;
        }
        ll x0 = cipolla(n, p);
        if (x0 == -1) cout << "Hola!" << endl;
        else {
```

```
            ll x1 = (p - x0 + p) % p;
            if (x0 != x1) cout << min(x0, x1) << ' ' << max(x0, x1) << endl;
            else cout << x0 << endl;
        }
    }
    return 0;
}
```

1.5 Combinatorics

1.5.1 Combination

$0 \le m \le n \le 1000$

```
const int maxn = 1010;
ll C[maxn][maxn];
void CalComb() {
    C[0][0] = 1;
    for (int i = 1; i < maxn; i++) {
        C[i][0] = 1;
        for (int j = 1; j <= i; j++) C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % mod;
    }
}
```

$0 \le m \le n \le 10^5$, 模 p 为素数

```
const int maxn = 100010;
ll f[maxn];
ll inv[maxn]; // 阶乘的逆元
void CalFact() {
    f[0] = 1;
    for (int i = 1; i < maxn; i++) f[i] = (f[i - 1] * i) % p;
    inv[maxn - 1] = Pow(f[maxn - 1], p - 2, p);
    for (int i = maxn - 2; ~i; i--) inv[i] = inv[i + 1] * (i + 1) % p;
}
ll C(int n, int m) { return f[n] * inv[m] % p * inv[n - m] % p; }
```

1.5.2 Lucas

$1 \le n, m \le 1000000000, 1 < p < 100000$, p 是素数

```
const int maxp = 100010;
ll f[maxn];
ll inv[maxn]; // 阶乘的逆元
void CalFact() {
    f[0] = 1;
    for (int i = 1; i < maxn; i++) f[i] = (f[i - 1] * i) % p;
    inv[maxn - 1] = Pow(f[maxn - 1], p - 2, p);
    for (int i = maxn - 2; ~i; i--) inv[i] = inv[i + 1] * (i + 1) % p;
}
ll Lucas(ll n, ll m, ll p) {
    ll ret = 1;
    while (n && m) {
        ll a = n % p, b = m % p;
        if (a < b) return 0;
        ret = ret * f[a] % p * inv[b] % p * inv[a - b] % p;
        n /= p, m /= p;
    }
    return ret;
}
```

1.5.3 exLucas

```cpp
#include<bits/stdc++.h>

using namespace std;
typedef long long ll;

ll exgcd(ll a,ll b,ll &x,ll &y) {
    if(!b) {
        x=1;
        y=0;
        return a;
    }
    ll res=exgcd(b,a%b,x,y),t;
    t=x;
    x=y;
    y=t-a/b*y;
    return res;
}

ll p;

inline ll power(ll a,ll b,ll mod) {
    ll sm;
    for(sm=1; b; b>>=1,a=a*a%mod)if(b&1) sm=sm*a%mod;
    return sm;
}

ll fac(ll n,ll pi,ll pk) {
    if(!n)return 1;
    ll res=1;
    for(ll i=2; i<=pk; ++i)
        if(i%pi)(res*=i)%=pk;
    res=power(res,n/pk,pk);
    for(ll i=2; i<=n%pk; ++i)
        if(i%pi)(res*=i)%=pk;
    return res*fac(n/pi,pi,pk)%pk;
}

inline ll inv(ll n,ll mod) {
    ll x,y;
    exgcd(n,mod,x,y);
    return (x+=mod)>mod?x-mod:x;
}

inline ll CRT(ll b,ll mod) {
    return b*inv(p/mod,mod)%p*(p/mod)%p;
}

const int MAXN=11;
static ll n,m;
static ll w[MAXN];

inline ll C(ll n,ll m,ll pi,ll pk) {
    ll up=fac(n,pi,pk),d1=fac(m,pi,pk),d2=fac(n-m,pi,pk);
    ll k=0;
    for(ll i=n; i; i/=pi)k+=i/pi;
    for(ll i=m; i; i/=pi)k-=i/pi;
    for(ll i=n-m; i; i/=pi)k-=i/pi;
```

```
58      return up*inv(d1,pk)%pk*inv(d2,pk)%pk*power(pi,k,pk)%pk;
59  }
60
61  inline ll exlucus(ll n,ll m) {
62      ll res=0,tmp=p,pk;
63      static int lim=sqrt(p)+5;
64      for(int i=2; i<=lim; ++i)if(tmp%i==0) {
65              pk=1;
66              while(tmp%i==0)pk*=i,tmp/=i;
67              (res+=CRT(C(n,m,i,pk),pk))%=p;
68          }
69      if(tmp>1)(res+=CRT(C(n,m,tmp,tmp),tmp))%=p;
70      return res;
71  }
72
73  int main() {
74      scanf("%lld%lld%d",&n,&m,&p);
75      printf("%d\n",exlucus(n,m));
76      return 0;
77  }
```

1.5.4 exLucas.cpp.

```
1   #include<bits/stdc++.h>
2
3   using namespace std;
4   typedef long long ll;
5
6   ll exgcd(ll a,ll b,ll &x,ll &y) {
7       if(!b){x=1;y=0;return a;}
8       ll res=exgcd(b,a%b,x,y),t;
9       t=x;x=y;y=t-a/b*y;
10      return res;
11  }
12
13  ll p;
14
15  inline ll power(ll a,ll b,ll mod) {
16      ll sm;
17      for(sm=1;b;b>>=1,a=a*a%mod)if(b&1) sm=sm*a%mod;
18      return sm;
19  }
20
21  ll fac(ll n,ll pi,ll pk) {
22      if(!n)return 1;
23      ll res=1;
24      for(ll i=2;i<=pk;++i)
25          if(i%pi)(res*=i)%=pk;
26      res=power(res,n/pk,pk);
27      for(ll i=2;i<=n%pk;++i)
28          if(i%pi)(res*=i)%=pk;
29      return res*fac(n/pi,pi,pk)%pk;
30  }
31
32  inline ll inv(ll n,ll mod)
33  {
34      ll x,y;
35      exgcd(n,mod,x,y);
```

```
    return (x+=mod)>mod?x-mod:x;
}

inline ll CRT(ll b,ll mod){return b*inv(p/mod,mod)%p*(p/mod)%p;}

const int MAXN=11;
static ll n,m;
static ll w[MAXN];

inline ll C(ll n,ll m,ll pi,ll pk) {
    ll up=fac(n,pi,pk),d1=fac(m,pi,pk),d2=fac(n-m,pi,pk);
    ll k=0;
    for(ll i=n;i;i/=pi)k+=i/pi;
    for(ll i=m;i;i/=pi)k-=i/pi;
    for(ll i=n-m;i;i/=pi)k-=i/pi;
    return up*inv(d1,pk)%pk*inv(d2,pk)%pk*power(pi,k,pk)%pk;
}

inline ll exlucus(ll n,ll m) {
    ll res=0,tmp=p,pk;
    static int lim=sqrt(p)+5;
    for(int i=2;i<=lim;++i)if(tmp%i==0) {
        pk=1;while(tmp%i==0)pk*=i,tmp/=i;
        (res+=CRT(C(n,m,i,pk),pk))%=p;
    }
    if(tmp>1)(res+=CRT(C(n,m,tmp,tmp),tmp))%=p;
    return res;
}

int main() {
    scanf("%lld%lld%d",&n,&m,&p);
    printf("%d\n",exlucus(n,m));
    return 0;
}
```

1.5.5 Big Combination

$0 \le n \le 10^9, 0 \le m \le 10^4, 1 \le k \le 10^9 + 7$

```
vector<int> v; int dp[110];
ll Cal(int l, int r, int k, int dis) {
    ll res = 1;
    for (int i = l; i <= r; i++) {
        int t = i;
        for (int j = 0; j < v.size(); j++) {
            int y = v[j];
            while (t % y == 0) dp[j] += dis, t /= y;
        }
        res = res * (ll)t % k;
    }
    return res;
}
ll Comb(int n, int m, int k) {
    memset(dp, 0, sizeof(dp));
    v.clear();
    int tmp = k;
    for (int i = 2; i * i <= tmp; i++)
        if (tmp % i == 0) {
```

```
            int num = 0;
            while (tmp % i == 0) tmp /= i, num++;
            v.push_back(i);
        }
    if (tmp != 1) v.push_back(tmp);
    ll ans = Cal(n - m + 1, n, k, 1);
    for (int j = 0; j < v.size(); j++) ans = ans * Pow(v[j], dp[j], k) % k;
    ans = ans * inv(Cal(2, m, k, -1), k) % k;
    return ans;
}
```

1.5.6 Polya

推论：一共 n 个置换，第 i 个置换的循环节个数为 $gcd(i,n)$

$N*N$ 的正方形格子，$c^{n^2}+2c^{\frac{n^2+3}{4}}+c^{\frac{n^2+1}{2}}+2c^{n\frac{n+1}{2}}+2c^{\frac{n(n+1)}{2}}$

正六面体，$\frac{m^8+17m^4+6m^2}{24}$ 正四面体，$\frac{m^4+11m^2}{12}$

长度为 n 的项链串用 c 种颜色染 $\sum_{d|n}\frac{\varphi(n/d)c^d}{n}$

```
ll solve(int c, int n) {
    if (n == 0) return 0;
    ll ans = 0;
    for (int i = 1; i <= n; i++) ans += Pow(c, __gcd(i, n));
    if (n & 1) ans += n * Pow(c, n + 1 >> 1);
    else ans += n / 2 * (1 + c) * Pow(c, n >> 1);
    return ans / n / 2;
}
```

每种颜色至少涂多少个，求方案数

```
ll polya(int a) { //a为循环节长度
    ll dp[65][65] = {0}; //前者为颜色，后者为未填充格子个数
    int tot = 60 / a, limit = 0;
    dp[0][tot] = 1;
    for (int i = 1; i <= n; i++) {
        int tmp = (c[i] + a - 1) / a;
        int up2 = tot - limit;
        int up1 = up2 - tmp;            //最多空tot-(limit + tmp)
        for (int j = 0; j <= up1; j++) { //最少空0个，即填满
            for (int k = tmp; j + k <= up2; k++) //至少选tmp个，最多选tot - limit -j
                (dp[i][j] += dp[i - 1][j + k] * C[j + k][k]) %= p;
        }
        limit += tmp;
    }
    return dp[n][0];
}
```

每种颜色要有多少个，求恰好满足的方案数

```
bool check(int b) { //a[i]是每种颜色有多少个，b是循环节长度
    for (int i = 0; i < n; i++)
        if (a[i] % b) return false;
    return true;
}
ll solve(int tot, int b) { //tot是总数，b是循环节长度
    if (!check(b)) return 0;
    ll res = 1, cnt = tot / b; //cnt循环节个数
    for (int i = 0; i < 6; i++) {
        res *= C[cnt][a[i] / b];
```

```
            cnt -= a[i] / b;
        }
        return res;
}
```

1.5.7 other

```
ll C(int a, int b) { // 无素数取模，求具体值的情况
    b = min(b, a - b);
    ll ans = 1;
    for (int i = a, j = 1; j <= b; ++j) {
        ans = ans * i / j;
        if (ans > up) { // 题目条件限制最大值，避免爆 long long
            return -1;
        }
    }
    return ans;
}
```

1.6 Modulo Linear Equation

1.6.1 Chinese Remainder Theory

$X \equiv r_i \pmod{m_i}$; 要求 m_i 两两互质
引用返回通解 $X = re + k * mo$

```
void crt(ll r[], ll m[], ll n, ll &re, ll &mo) {
    mo = 1, re = 0;
    for (int i = 0; i < n; i++) mo *= m[i];
    for (int i = 0; i < n; i++) {
        ll x, y,  tm = mo / m[i];
        ll d = exgcd(tm, m[i], x, y);
        re = (re + tm * x * r[i]) % mo;
    }
    re = (re + mo) % mo;
}
```

1.6.2 ExCRT

$X \equiv r_i \pmod{m_i}$; m_i 可以不两两互质
引用返回通解 $X = re + k * mo$; 函数返回是否有解

```
bool excrt(ll r[], ll m[], ll n, ll &re, ll &mo) {
    ll x, y;
    mo = m[0], re = r[0];
    for (int i = 1; i < n; i++) {
        ll d = exgcd(mo, m[i],  x, y);
        if ((r[i] - re) % d != 0) return 0;
        x = (r[i] - re) / d * x % (m[i] / d);
        re += x * mo;
        mo = mo / d * m[i];
        re %= mo;
    }
    re = (re + mo) % mo;
    return 1;
}
```

1.7 快速

```
inline ll Mul(ll a, ll b, ll m) {
    if (m <= 1000000000) return a * b % m;
    else if (m <= 100000000000000ll)
        return (((a * (b >> 20) % m) << 20) + (a * (b & ((1 << 20) - 1)))) % m;
    else {
        ll d = (ll)floor(a * (long double)b / m + 0.5);
        ll ret = (a * b - d * m) % m;
        if (ret < 0) ret += m;
        return ret;
    }
}
ll smul(ll a, ll b, ll p) { // 龟速乘
    a %= p;
    ll ans = 0;
    while (b) {
        if (b & 1) ans = (ans + a) % p;
        a = a * 2 % p, b >>= 1;
    }
    return ans;
}
// O(1), 数值过大时容易因为精度引起误差
ll qmul(ll a, ll b, ll p) { // 快速乘
    a %= p, b %= p;
    ll c = (long double)a * b / p;
    ll ans = a * b - c * p;
    if (ans < 0) return ans + p;
    if (ans >= p) return ans - p;
    return ans;
}
ll qpow(ll a, ll b, ll p) {
    a %= p;
    ll ans = 1;
    while (b) {
        if (b & 1) ans = smul(ans, a, p);   //  p > 1e9 || a > 1e9
                   // ans = (ans * a) % p;
        a = smul(a, a, p);  //  p > 1e9 || a > 1e9
        // a = a * a % p;
        b >>= 1;
    }
    return ans;
}
```

1.8 Mobius Inversion

1.8.1 Mobius

$F(n) = \sum_{d|n} f(d) \Rightarrow f(n) = \sum_{d|n} \mu(d) F(\frac{n}{d})$
$F(n) = \sum_{n|d} f(d) \Rightarrow f(n) = \sum_{n|d} \mu(\frac{d}{n}) F(d)$

```
const int maxn = "Edit";
int prime[maxn], tot, mu[maxn];
bool check[maxn];
void CalMu() {
    mu[1] = 1;
    for (int i = 2; i < maxn; i++) {
        if (!check[i]) prime[tot++] = i, mu[i] = -1;
```

```
        for (int j = 0; j < tot; j++) {
            if (i * prime[j] >= maxn) break;
            check[i * prime[j]] = true;
            if (i % prime[j] == 0) {
                mu[i * prime[j]] = 0;
                break;
            }
            else mu[i * prime[j]] = -mu[i];
        }
    }
}
```

1.8.2 Examples

有 n 个数 $(n \leq 100000, 1 \leq a_i \leq 10^6)$，问这 n 个数中互质的数的对数

```
const int maxn = "Edit";
int b[maxn];
ll solve(int n){
    ll ans = 0;
    for (int i = 0, x; i < n; i++) scanf("%d", &x), b[x]++;
    for (int i = 1; i < maxn; i++) {
        int cnt = 0;
        for (int j = i; j < maxn; j += i) cnt += b[j];
        ans += 1LL * mu[i] * cnt * cnt;
    }
    return (ans - b[1]) / 2;
}
```

$\gcd(x, y) = 1$ 的对数, $x \leq n, y \leq m$

```
ll solve(int n, int m){
    if (n > m) swap(n, m);
    ll ans = 0;
    for (int i = 1; i <= n; i++) ans += (ll)mu[i] * (n / i) * (m / i);
    /*
    数论分块写法(sum为莫比乌斯函数的前缀和)
    for (int i = 1; i <= n; i = pos + 1)
    {
        pos = min(n / (n / i), m / (m / i));
        ans += 1LL * (sum[pos] - sum[i - 1]) * (n / i) * (m / i);
    }
    */
    return ans;
}
```

1.9 Fast Transformation

```
//#pragma GCC optimize(2)
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

const int N = 3000007;
const int p = 998244353, gg = 3, ig = 332738118, img = 86583718;
const int mod = 998244353;

template <typename T>void read(T &x) {
```

```
    x = 0;
    register int f = 1;
    register char ch = getchar();
    while(ch < '0' || ch > '9') {if(ch == '-')f = -1;ch = getchar();}
    while(ch >= '0' && ch <= '9') {x = x * 10 + ch - '0';ch = getchar();}
    x *= f;
}

int qpow(int a, int b) {
    int res = 1;
    while(b) {
        if(b & 1) res = 1ll * res * a % mod;
        a = 1ll * a * a % mod;
        b >>= 1;
    }
    return res;
}

namespace Poly {
    #define mul(x, y) (1ll * x * y >= mod ? 1ll * x * y % mod : 1ll * x * y)
    #define minus(x, y) (1ll * x - y < 0 ? 1ll * x - y + mod : 1ll * x - y)
    #define plus(x, y) (1ll * x + y >= mod ? 1ll * x + y - mod : 1ll * x + y)
    #define ck(x) (x >= mod ? x - mod : x)//取模运算太慢了

    typedef vector<int> poly;
    const int G = 5;
    const int inv_G = qpow(G, mod - 2);
    int RR[N], deer[2][19][N], inv[N];

    void init(const int t) {//预处理出来NTT里需要的w和wn，砍掉了一个log的时间
        for(int p = 1; p <= t; ++ p) {
            int buf1 = qpow(G, (mod - 1) / (1 << p));
            int buf0 = qpow(inv_G, (mod - 1) / (1 << p));
            deer[0][p][0] = deer[1][p][0] = 1;
            for(int i = 1; i < (1 << p); ++ i) {
                deer[0][p][i] = 1ll * deer[0][p][i - 1] * buf0 % mod;//逆
                deer[1][p][i] = 1ll * deer[1][p][i - 1] * buf1 % mod;
            }
        }
        inv[1] = 1;
        for(int i = 2; i <= (1 << t); ++ i)
            inv[i] = 1ll * inv[mod % i] * (mod - mod / i) % mod;
    }

    int NTT_init(int n) {//快速数论变换预处理
        int limit = 1, L = 0;
        while(limit < n) limit <<= 1, L ++ ;
        for(int i = 0; i < limit; ++ i)
            RR[i] = (RR[i >> 1] >> 1) | ((i & 1) << (L - 1));
        return limit;
    }

    void NTT(poly &A, int type, int limit) {//快速数论变换
        A.resize(limit);
        for(int i = 0; i < limit; ++ i)
            if(i < RR[i])
                swap(A[i], A[RR[i]]);
        for(int mid = 2, j = 1; mid <= limit; mid <<= 1, ++ j) {
            int len = mid >> 1;
```

```
            for(int pos = 0; pos < limit; pos += mid) {
                int *wn = deer[type][j];
                for(int i = pos; i < pos + len; ++ i, ++ wn) {
                    int tmp = 1ll * (*wn) * A[i + len] % mod;
                    A[i + len] = ck(A[i] - tmp + mod);
                    A[i] = ck(A[i] + tmp);
                }
            }
        }
        if(type == 0) {
            for(int i = 0; i < limit; ++ i)
                A[i] = 1ll * A[i] * inv[limit] % mod;
        }
    }

    poly poly_mul(poly A, poly B) {//多项式乘法
        int deg = A.size() + B.size() - 1;
        int limit = NTT_init(deg);
        poly C(limit);
        NTT(A, 1, limit);
        NTT(B, 1, limit);
        for(int i = 0; i < limit; ++ i)
            C[i] = 1ll * A[i] * B[i] % mod;
        NTT(C, 0, limit);
        C.resize(deg);
        return C;
    }

    poly poly_inv(poly &f, int deg) {//多项式求逆
        if(deg == 1) return poly(1, qpow(f[0], mod - 2));

        poly A(f.begin(), f.begin() + deg);
        poly B = poly_inv(f, (deg + 1) >> 1);
        int limit = NTT_init(deg << 1);
        NTT(A, 1, limit), NTT(B, 1, limit);
        for(int i = 0; i < limit; ++ i)
            A[i] = B[i] * (2 - 1ll * A[i] * B[i] % mod + mod) % mod;
        NTT(A, 0, limit);
        A.resize(deg);
        return A;
    }

    poly poly_dev(poly f) {//多项式求导
        int n = f.size();
        for(int i = 1; i < n; ++ i) f[i - 1] = 1ll * f[i] * i % mod;
        return f.resize(n - 1), f;//f[0] = 0, 这里直接扔了,从1开始
    }

    poly poly_idev(poly f) {//多项式求积分
        int n = f.size();
        for(int i = n - 1; i ; -- i) f[i] = 1ll * f[i - 1] * inv[i] % mod;
        return f[0] = 0, f;
    }

    poly poly_ln(poly f, int deg) {//多项式求对数
        poly A = poly_idev(poly_mul(poly_dev(f), poly_inv(f, deg)));
        return A.resize(deg), A;
    }

```

```
129     poly poly_exp(poly &f, int deg) {//多项式求指数
130         if(deg == 1) return poly(1, 1);
131 
132         poly B = poly_exp(f, (deg + 1) >> 1);
133         B.resize(deg);
134         poly lnB = poly_ln(B, deg);
135         for(int i = 0; i < deg; ++ i)
136             lnB[i] = ck(f[i] - lnB[i] + mod);
137 
138         int limit = NTT_init(deg << 1);//n -> n^2
139         NTT(B, 1, limit), NTT(lnB, 1, limit);
140         for(int i = 0; i < limit; ++ i)
141             B[i] = 1ll * B[i] * (1 + lnB[i]) % mod;
142         NTT(B, 0, limit);
143         B.resize(deg);
144         return B;
145     }
146 
147     poly poly_sqrt(poly &f, int deg) {//多项式开方
148         if(deg == 1) return poly(1, 1);
149         poly A(f.begin(), f.begin() + deg);
150         poly B = poly_sqrt(f, (deg + 1) >> 1);
151         poly IB = poly_inv(B, deg);
152         int limit = NTT_init(deg << 1);
153         NTT(A, 1, limit), NTT(IB, 1, limit);
154         for(int i = 0; i < limit; ++ i)
155             A[i] = 1ll * A[i] * IB[i] % mod;
156         NTT(A, 0, limit);
157         for(int i =0; i < deg; ++ i)
158             A[i] = 1ll * (A[i] + B[i]) * inv[2] % mod;
159         A.resize(deg);
160         return A;
161     }
162 
163     poly poly_pow(poly f, int k) {//多项式快速幂
164         f = poly_ln(f, f.size());
165         for(auto &x : f) x = 1ll * x * k % mod;
166         return poly_exp(f, f.size());
167     }
168 
169     poly poly_cos(poly f, int deg) {//多项式三角函数 (cos)
170         poly A(f.begin(), f.begin() + deg);
171         poly B(deg), C(deg);
172         for(int i = 0; i < deg; ++ i)
173             A[i] = 1ll * A[i] * img % mod;
174 
175         B = poly_exp(A, deg);
176         C = poly_inv(B, deg);
177         int inv2 = qpow(2, mod - 2);
178         for(int i = 0; i < deg; ++ i)
179             A[i] = 1ll * (1ll * B[i] + C[i]) % mod * inv2 % mod;
180         return A;
181     }
182 
183     poly poly_sin(poly f, int deg) {//多项式三角函数 (sin)
184         poly A(f.begin(), f.begin() + deg);
185         poly B(deg), C(deg);
186         for(int i = 0; i < deg; ++ i)
187             A[i] = 1ll * A[i] * img % mod;
```

```
188
189         B = poly_exp(A, deg);
190         C = poly_inv(B, deg);
191         int inv2i = qpow(img << 1, mod - 2);
192         for(int i = 0; i < deg; ++ i)
193             A[i] = 1ll * (1ll * B[i] - C[i] + mod) % mod * inv2i % mod;
194         return A;
195     }
196
197     poly poly_arcsin(poly f, int deg) {
198         poly A(f.size()), B(f.size()), C(f.size());
199         A = poly_dev(f);
200         B = poly_mul(f, f);
201         for(int i = 0; i < deg; ++ i)
202             B[i] = minus(mod, B[i]);
203         B[0] = plus(B[0], 1);
204         C = poly_sqrt(B, deg);
205         C = poly_inv(C, deg);
206         C = poly_mul(A, C);
207         C = poly_idev(C);
208         return C;
209     }
210
211     poly poly_arctan(poly f, int deg) {
212         poly A(f.size()), B(f.size()), C(f.size());
213         A = poly_dev(f);
214         B = poly_mul(f, f);
215         B[0] = plus(B[0], 1);
216         C = poly_inv(B, deg);
217         C = poly_mul(A, C);
218         C = poly_idev(C);
219         return C;
220     }
221 }
222
223 using Poly::poly;
224 using Poly::poly_arcsin;
225 using Poly::poly_arctan;
226
227 int n, m, x, k, type;
228 poly f, g;
229 char s[N];
230
231 int main() {
232     Poly::init(18);//2^21 = 2,097,152,根据题目数据多项式项数的大小自由调整，注意大小需要跟deer数
        组同步(21+1=22)
233
234     read(n), read(type);
235
236     for(int i = 0; i < n; ++ i) read(x), f.push_back(x);
237
238     if(type == 0) g = poly_arcsin(f, n);
239     else g = poly_arctan(f, n);
240
241     for(int i = 0; i < n; ++i) printf("%d ", g[i]);
242     return 0;
243 }
```

1.9.1 FFT

```
const double PI = acos(-1.0);
//复数结构体
struct Complex {
    double x, y; //实部和虚部 x+yi
    Complex(double _x = 0.0, double _y = 0.0) { x = _x, y = _y; }
    Complex operator-(const Complex& b) const { return Complex(x - b.x, y - b.y); }
    Complex operator+(const Complex& b) const { return Complex(x + b.x, y + b.y); }
    Complex operator*(const Complex& b) const { return Complex(x * b.x - y * b.y, x * b
    .y + y * b.x); }
};
void change(Complex y[], int len) {
    for (int i = 1, j = len / 2; i < len - 1; i++) {
        if (i < j) swap(y[i], y[j]);
        int k = len / 2;
        while (j >= k) j -= k, k /= 2;
        if (j < k) j += k;
    }
}
/*
* len必须为2^k形式,
* on==1时是DFT, on==-1时是IDFT
*/
void fft(Complex y[], int len, int on) {
    change(y, len);
    for (int h = 2; h <= len; h <<= 1) {
        Complex wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI / h));
        for (int j = 0; j < len; j += h) {
            Complex w(1, 0);
            for (int k = j; k < j + h / 2; k++) {
                Complex u = y[k];
                Complex t = w * y[k + h / 2];
                y[k] = u + t, y[k + h / 2] = u - t;
                w = w * wn;
            }
        }
    }
    if (on == -1) for (int i = 0; i < len; i++) y[i].x /= len;
}
```

1.9.2 NTT

模数 P 为费马素数，G 为 P 的原根。$G^{\frac{P-1}{n}}$ 具有和 $w_n = e^{\frac{2i\pi}{n}}$ 相似的性质。具体的 P 和 G 可参考 1.11

```
const int mod = 119 << 23 | 1;
const int G = 3;
int wn[20];
void getwn() { //  千万不要忘记
    for (int i = 0; i < 20; i++) wn[i] = Pow(G, (mod - 1) / (1 << i), mod);
}
void change(int y[], int len) {
    for (int i = 1, j = len / 2; i < len - 1; i++) {
        if (i < j) swap(y[i], y[j]);
        int k = len / 2;
        while (j >= k) j -= k, k /= 2;
        if (j < k) j += k;
    }
```

```
}
void ntt(int y[], int len, int on) {
    change(y, len);
    for (int h = 2, id = 1; h <= len; h <<= 1, id++) {
        for (int j = 0; j < len; j += h) {
            int w = 1;
            for (int k = j; k < j + h / 2; k++) {
                int u = y[k] % mod;
                int t = 1LL * w * (y[k + h / 2] % mod) % mod;
                y[k] = (u + t) % mod, y[k + h / 2] = ((u - t) % mod + mod) % mod;
                w = 1LL * w * wn[id] % mod;
            }
        }
    }
    if (on == -1) {
        //  原本的除法要用逆元
        int inv = Pow(len, mod - 2, mod);
        for (int i = 1; i < len / 2; i++) swap(y[i], y[len - i]);
        for (int i = 0; i < len; i++) y[i] = 1LL * y[i] * inv % mod;
    }
}
```

1.9.3 FWT

```
void fwt(int f[], int m) {
    int n = __builtin_ctz(m);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            if (j & (1 << i)) {
                int l = f[j ^ (1 << i)], r = f[j];
                f[j ^ (1 << i)] = l + r, f[j] = l - r;
                // or: f[j] += f[j ^ (1 << i)];
                // and: f[j ^ (1 << i)] += f[j];
            }
}
void ifwt(int f[], int m) {
    int n = __builtin_ctz(m);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            if (j & (1 << i)) {
                int l = f[j ^ (1 << i)], r = f[j];
                f[j ^ (1 << i)] = (l + r) / 2, f[j] = (l - r) / 2;
                // 如果有取模需要使用逆元
                // or: f[j] -= f[j ^ (1 << i)];
                // and: f[j ^ (1 << i)] -= f[j];
            }
}
```

1.9.4 多项式对数指数函数

```
#include<bits/stdc++.h>
#define maxn 100010
#define N (262144 | 3)

const int mod=998244353;
const int G = 3;
int n,m;
int inv[N],a[N],b[N];
```

```
int num[N], maxv;

inline void init() {
    inv[1] = 1;
    for (int i = 2; i < N; ++i) {
        inv[i] = 1ll * (mod - mod / i) * inv[mod % i] % mod;
    }
}

namespace Poly{
    inline int qpow(int a, int b) {
        int ans = 1;
        while (b) {
            if (b & 1) ans = 1ll * a * ans % mod;
            b >>= 1;
            a = 1ll * a * a % mod;
        }
        return ans;
    }
    inline int Inv(int x){return qpow(x, mod - 2);}
    int lim, ilim, L, wn[N], R[N];
    inline void init(int n) {
        lim = 1, L = 0; while (lim < n) lim <<= 1, L++; ilim = inv[lim];
        for (int i = 0; i < lim; i++) R[i] = R[i >> 1] >> 1 | (i & 1) << (L - 1);
        int t = qpow(G, (mod - 1) / lim);
        wn[0] = 1; for (int i = 1; i <= lim; i++) wn[i] = 1ll * wn[i - 1] * t % mod;
    }
    inline void add(int &a, int b) {if ((a += b) >= mod) a -= mod;}
    inline void NTT(int *A, int type = 1) {
        for (int i = 0; i < lim; i++) if (i < R[i]) std::swap(A[i], A[R[i]]);
        for (int mid = 1; mid < lim; mid <<= 1) {
            int t = lim / mid >> 1;
            for (int i = 0; i < lim; i += mid << 1) {
                for (int j = 0; j < mid; j++) {
                    int W = type ? wn[t * j] : wn[lim - t * j];
                    int X = A[i + j], Y = 1ll * A[i + j + mid] * W % mod;
                    add(A[i + j], Y), add(A[i + j + mid] = X, mod - Y);
                }
            }
        }
        if (!type) for (int i = 0; i < lim; i++) A[i] = 1ll * A[i] * ilim % mod;
    }
    inline void DER(int *A, int *B, int n) {
        B[n - 1] = 0; for (int i = 1; i < n; i++) B[i - 1] = 1ll * A[i] * i % mod;
    }
    inline void INT(int *A, int *B, int n) {
        B[0] = 0; for (int i = 1; i < n; i++) B[i] = 1ll * A[i - 1] * inv[i] % mod;
    }
    int C[N];
    inline void INV(int *A, int *B, int n) {
        if (n == 1) {B[0] = Inv(A[0]); return ;}
        INV(A, B, (n + 1) >> 1), init(n << 1);
        for (int i = 0; i < n; i++) C[i] = A[i];
        for (int i = n; i < lim; i++) C[i] = B[i] = 0;
        NTT(B), NTT(C);
        for (int i = 0; i < lim; i++) B[i] = (2 + mod - 1ll * B[i] * C[i] % mod) * B[i]
 % mod;
        NTT(B, 0);
        for (int i = n; i < lim; i++) B[i] = 0;
```

```
    }
    int D[N];
    inline void LN(int *A, int *B, int n) {
        DER(A, D, n), INV(A, B, n);
        init(n << 1);
        NTT(B), NTT(D);
        for (int i = 0; i < lim; i++) D[i] = 1ll * B[i] * D[i] % mod;
        NTT(D, 0), INT(D, B, n);
        for (int i = n; i < lim; i++) B[i] = 0;
    }
    int E[N], F[N];
    void EXP(int *A, int *B, int n) {
        if (n == 1) {B[0] = 1; return ;}
        EXP(A, B, (n + 1) >> 1);
        for (int i = 0; i < n << 1; i++) E[i] = F[i] = 0;
        LN(B, E, n);
        for (int i = 0; i < n; i++) F[i] = A[i];
        NTT(B), NTT(E), NTT(F);
        for (int i = 0; i < lim; i++) B[i] = (1ll + mod - E[i] + F[i]) * B[i] % mod;
        NTT(B, 0);
        for (int i = n; i < lim; i++) B[i] = 0;
    }
}

int main() {
    init();
    scanf("%d%d", &n, &m); m++;
    for (int i = 0; i < n; ++i) {
        int u; scanf("%d", &u);
        num[u]++;
        maxv = std::max(maxv, u);
    }
    for (int i = 1; i <= maxv; ++i) {
        if (num[i]) {
            for (int j = 1; j * i <= m; ++j) {
                a[j * i] = (a[j * i] + 1ll * inv[j] * num[i] % mod) % mod;
            }
        }
    }
    Poly::EXP(a, b, m);
    for (int i = 1; i < m; i++) printf("%d\n", b[i]);
    return 0;
}
```

1.10 Numerical Integration

1.10.1 Adaptive Simpson's Rule

$\int_a^b f(x)dx \approx \frac{b-a}{6}[f(a)+4f(\frac{a+b}{2})+f(b)]$
$|S(a,c)+S(c,b)-S(a,b)|/15 < \epsilon$

```
double F(double x) {}
double simpson(double a, double b)
{ // 三点Simpson法
    double c = a + (b - a) / 2;
    return (F(a) + 4 * F(c) + F(b)) * (b - a) / 6;
}
double asr(double a, double b, double eps, double A)
{ //自适应Simpson公式（递归过程）。已知整个区间[a,b]上的三点Simpson值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) / 15.0;
    return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
}
double asr(double a, double b, double eps) { return asr(a, b, eps, simpson(a, b)); }
```

1.10.2 Berlekamp-Massey

```
const int maxn = 1 << 14;
ll res[maxn], base[maxn], _c[maxn], _md[maxn];
vector<int> Md;
void mul(ll* a, ll* b, int k) {
    for (int i = 0; i < k + k; i++) _c[i] = 0;
    for (int i = 0; i < k; i++)
        if (a[i]) for (int j = 0; j < k; j++)
            _c[i + j] = (_c[i + j] + a[i] * b[j]) % mod;
    for (int i = k + k - 1; i >= k; i--)
        if (_c[i]) for (int j = 0; j < Md.size(); j++)
            _c[i - k + Md[j]] = (_c[i - k + Md[j]] - _c[i] * _md[Md[j]]) % mod;
    for (int i = 0; i < k; i++) a[i] = _c[i];
}
int solve(ll n, VI a, VI b) {
    ll ans = 0, pnt = 0;
    int k = a.size();
    assert(a.size() == b.size());
    for (int i = 0; i < k; i++) _md[k - 1 - i] = -a[i];
    _md[k] = 1;
    Md.clear();
    for (int i = 0; i < k; i++)
        if (_md[i] != 0) Md.push_back(i);
    for (int i = 0; i < k; i++) 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;
            for (int j = 0; j < Md.size(); j++) res[Md[j]] = (res[Md[j]] - res[k] * _md
    [Md[j]]) % mod;
        }
    }
    for (int i = 0; i < k; i++) 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;
    for (int n = 0; n < s.size(); n++) {
        ll d = 0;
        for (int i = 0; i <= L; i++) d = (d + (ll)C[i] * s[n - i]) % mod;
        if (d == 0) ++m;
        else if (2 * L <= n) {
            VI T = C;
            ll c = mod - d * Pow(b, mod - 2) % mod;
            while (C.size() < B.size() + m) C.push_back(0);
```

```
            for (int i = 0; i < B.size(); i++) 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 * Pow(b, mod - 2) % mod;
            while (C.size() < B.size() + m) C.push_back(0);
            for (int i = 0; i < B.size(); i++) 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());
    for (int i = 0; i < c.size(); i++) c[i] = (mod - c[i]) % mod;
    return solve(n, c, VI(a.begin(), a.begin() + c.size()));
}
```

1.10.3 Simplex

输入矩阵 a 描述线性规划的标准形式。

a 为 $m+1$ 行 $n+1$ 列，其中行 $0 \sim m-1$ 为不等式，行 m 为目标函数（最大化）。

列 $0 \sim n-1$ 为变量 $0 \sim n-1$ 的系数，列 n 为常数项。

约束为 $a_{i,0}x_0 + a_{i,1}x_1 + \cdots \le a_{i,n}$，目标为 $\max(a_{m,0}x_0 + a_{m,1}x_1 + \cdots + a_{m,n-1}x_{n-1} - a_{m,n})$

注意：变量均有非负约束 $x[i] \ge 0$

```
const int maxm = 500; // 约束数目上限
const int maxn = 500; // 变量数目上限
const double INF = 1e100;
const double eps = 1e-10;
struct Simplex {
    int n;                 // 变量个数
    int m;                 // 约束个数
    double a[maxm][maxn]; // 输入矩阵
    int B[maxm], N[maxn]; // 算法辅助变量
    void pivot(int r, int c) {
        swap(N[c], B[r]);
        a[r][c] = 1 / a[r][c];
        for (int j = 0; j <= n; j++)
            if (j != c) a[r][j] *= a[r][c];
        for (int i = 0; i <= m; i++)
            if (i != r) {
                for (int j = 0; j <= n; j++)
                    if (j != c) a[i][j] -= a[i][c] * a[r][j];
                a[i][c] = -a[i][c] * a[r][c];
            }
    }
    bool feasible() {
        for (;;) {
            int r, c;
            double p = INF;
            for (int i = 0; i < m; i++)
                if (a[i][n] < p) p = a[r = i][n];
```

```
            if (p > -eps) return true;
            p = 0;
            for (int i = 0; i < n; i++)
                if (a[r][i] < p) p = a[r][c = i];
            if (p > -eps) return false;
            p = a[r][n] / a[r][c];
            for (int i = r + 1; i < m; i++)
                if (a[i][c] > eps)
                {
                    double v = a[i][n] / a[i][c];
                    if (v < p) r = i, p = v;
                }
            pivot(r, c);
        }
    }
    // 解有界返回1，无解返回0，无界返回-1。b[i]为x[i]的值，ret为目标函数的值
    int simplex(int n, int m, double x[maxn], double& ret) {
        this->n = n, this->m = m;
        for (int i = 0; i < n; i++) N[i] = i;
        for (int i = 0; i < m; i++) B[i] = n + i;
        if (!feasible()) return 0;
        for (;;) {
            int r, c;
            double p = 0;
            for (int i = 0; i < n; i++)
                if (a[m][i] > p) p = a[m][c = i];
            if (p < eps) {
                for (int i = 0; i < n; i++)
                    if (N[i] < n) x[N[i]] = 0;
                for (int i = 0; i < m; i++)
                    if (B[i] < n) x[B[i]] = a[i][n];
                ret = -a[m][n];
                return 1;
            }
            p = INF;
            for (int i = 0; i < m; i++)
                if (a[i][c] > eps) {
                    double v = a[i][n] / a[i][c];
                    if (v < p) r = i, p = v;
                }
            if (p == INF) return -1;
            pivot(r, c);
        }
    }
};
```

1.11 其他

约瑟夫问题
n 个人围成一圈，从第一个开始报数，第 m 个将被杀掉

```
int josephus(int n, int m) {
    int r = 0;
    for (int k = 1; k <= n; ++k) r = (r + m) % k;
    return r + 1;
}
```

n^n 最左边一位数

```cpp
int leftmost(int n) {
    double m = n * log10((double)n);
    double g = m - (ll)m;
    return (int)pow(10.0, g);
}
```

$n!$ 位数

```cpp
int count(ll n) {
    if (n == 1) return 1;
    return (int)ceil(0.5 * log10(2 * M_PI * n) + n * log10(n) - n * log10(M_E));
}
```

1.12 公式

1. 约数定理：若 $n = \prod_{i=1}^{k} p_i^{a_i}$，则
   (a) 约数个数 $f(n) = \prod_{i=1}^{k}(a_i + 1)$
   (b) 约数和 $g(n) = \prod_{i=1}^{k}(\sum_{j=0}^{a_i} p_i^j)$
2. 小于 n 且互素的数之和为 $n\varphi(n)/2$
3. 若 $\gcd(n, i) = 1$，则 $\gcd(n, n-i) = 1 (1 \le i \le n)$
4. 错排公式：$D(n) = (n-1)(D(n-2) + D(n-1)) = \sum_{i=2}^{n} \frac{(-1)^k n!}{k!} = [\frac{n!}{e} + 0.5]$
5. 威尔逊定理：$p\ is\ prime\ \Rightarrow (p-1)! \equiv -1 \pmod p$
6. 欧拉定理：$\gcd(a, n) = 1 \Rightarrow a^{\varphi(n)} \equiv 1 \pmod n$
7. 欧拉定理推广：$\gcd(n, p) = 1 \Rightarrow a^n \equiv a^{n\%\varphi(p)} \pmod p$
8. 模的幂公式：$a^n \pmod m = \begin{cases} a^n \mod m & n < \varphi(m) \\ a^{n\%\varphi(m)+\varphi(m)} \mod m & n \ge \varphi(m) \end{cases}$
9. 素数定理：对于不大于 n 的素数个数 $\pi(n)$，$\lim_{n\to\infty} \pi(n) = \frac{n}{\ln n}$
10. 位数公式：正整数 x 的位数 $N = \log_{10}(n) + 1$
11. 斯特灵公式 $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$
12. 设 $a > 1, m, n > 0$，则 $\gcd(a^m - 1, a^n - 1) = a^{\gcd(m,n)} - 1$
13. 设 $a > b, \gcd(a, b) = 1$，则 $\gcd(a^m - b^m, a^n - b^n) = a^{\gcd(m,n)} - b^{\gcd(m,n)}$

$$G = \gcd(C_n^1, C_n^2, ..., C_n^{n-1}) = \begin{cases} n, & n \text{ is prime} \\ 1, & n \text{ has multy prime factors} \\ p, & n \text{ has single prime factor } p \end{cases}$$

$\gcd(Fib(m), Fib(n)) = Fib(\gcd(m, n))$

14. 若 $\gcd(m, n) = 1$，则:
   (a) 最大不能组合的数为 $m * n - m - n$
   (b) 不能组合数个数 $N = \frac{(m-1)(n-1)}{2}$
15. $(n+1)lcm(C_n^0, C_n^1, ..., C_n^{n-1}, C_n^n) = lcm(1, 2, ..., n+1)$
16. 若 p 为素数，则 $(x + y + ... + w)^p \equiv x^p + y^p + ... + w^p \pmod p$
17. 卡特兰数：1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012
    $h(0) = h(1) = 1, h(n) = \frac{(4n-2)h(n-1)}{n+1} = \frac{C_{2n}^n}{n+1} = C_{2n}^n - C_{2n}^{n-1}$
18. 伯努利数：$B_n = -\frac{1}{n+1}\sum_{i=0}^{n-1} C_{n+1}^i B_i$

$$\sum_{i=1}^{n} i^k = \frac{1}{k+1}\sum_{i=1}^{k+1} C_{k+1}^i B_{k+1-i}(n+1)^i$$

19. 二项式反演:

$$f_n = \sum_{i=0}^{n} (-1)^i \binom{n}{i} g_i \Leftrightarrow g_n = \sum_{i=0}^{n} (-1)^i \binom{n}{i} f_i$$

$$f_n = \sum_{i=0}^{n} \binom{n}{i} g_i \Leftrightarrow g_n = \sum_{i=0}^{n} (-1)^{n-i} \binom{n}{i} f_i$$

20. FFT 常用素数

$r\ 2^k + 1$	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
998244353	119	23	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
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

g(1)S(n) $= \sum_{i=1}^{n} (f * g)(i) - \sum_{i=2}^{n} g(i) S(\lfloor \frac{n}{i} \rfloor)$

其中 S(n) $= \sum_{i=1}^{n} f(i)$, 关键在于寻找 g

使得 $(f * g)(i)$ 为我们方便求解的形式

$\mu * 1 = \epsilon$
$\varphi * 1 = id$
$\mu * id = \phi$

Pick therom : $A = i + \frac{b}{2} - 1$
i : 多边形内部格点数
b : 多边形边上格点数

2 String Processing

2.1 哈希

2.1.1 一般哈希

```
/* 1 x : 插入一个数 x
 * 2 x : 询问 x 是否在集合中出现过 */
const int p = 100003; // p < N
typedef unsigned long long ull;
int n, x, cnt, h[N], nx[N], e[N];
void init() { memset(h, 1, sizeof(h)); }
// 开放寻址法 (1e5 数据下比 unordered_map 快)
// 如果 x 在哈希表中, 返回 x 的下标, 否则返回 x 应该插入的位置
int get_pos(int x) { // 2
    int t = (x % p + p) % p;
    while (h[t] && h[t] != x) if (++t == p) t = 0;
    return t;
}
void insert(int x) { h[find(x)] = x; }
int find(int x) { return h[find(x)] == x; }
```

2.1.2 字符串哈希

```
// 查询 n 个字符串中共有多少个不同的字符串 // 也可用trie来做
ull base = 13331; // 取 131 或 13331 不容易冲突
ull Hash(char *s, ull ans = 0) {
    for (int i = 0; s[i]; ++i)
        ans = ans * base + (ull)s[i];
    return ans;
}
for (int i = 0; i < n; ++i) hs[i] = Hash(s[i]);
sort(hs, hs + n);
printf("%ld\n", unique(hs, hs + n) - hs);
```

2.1.3 字符串哈希 (区间)

```
// 判断 [l1, r1], [l2, r2] 这两个区间所包含的字符串字串是否完全相同
const int base = 13331; // base 用 131 或 13331 不容易冲突
ull h[N], p[N];
inline void Hash() {
    *p = 1;
    for (int i = 1; i <= n; ++i) {
        p[i] = p[i - 1] * base;
        h[i] = h[i - 1] * base + s[i];
    }
}
inline ull get(int l, int r) { // 区间 [l, r] 的 Hash 值
    return h[r] - h[l - 1] * p[r - l + 1];
}
int check(int l1, int r1, int l2, int r2) { // 判断
    return get(l1, r1) == get(l2, r2);
}
```

2.1.4 ELF HASH

```
uint ELF_Hash(char *s) {
    uint x, hs = 1; // hs = 0 在树上哈希会出错, 故统一令 hs = 1
    while (*s) {
        hs = (hs << 4) + *s++;
        (x = hs & 0xf0000000) && (hs ^= (x >> 24), hs &= ~x); // 注意是 ~ x , 不是 - x
    }
    return (hs & 0x7fffffff);
}
```

2.2 KMP

```
// 返回 y 在 x 中的出现次数
const int N = "Edit";
int kmp[N];
void init(char x[], int m) {
    int j = 0;
    for (int i = 1; i < m; ++i) {
        while (j && x[i] != x[j]) j = kmp[j - 1];
        if (x[i] == x[j]) ++j;
        kmp[i] = j;
    }
}
int KMP(char x[], int n, char y[], int m) {
    init(y, m);
    int j = 0, ans = 0;
    for (int i = 0; i < n; ++i) {
        while (j && x[i] != y[j]) j = kmp[j - 1];
        if (x[i] == y[j] && ++j == m) {
            ++ans, j = kmp[j - 1];
        }
    }
    return ans;
}
```

2.3 ExKMP

```
//kmp[i]:x[i...m-1]与x[0...m-1]的最长公共前缀
//extend[i]:y[i...n-1]与x[0...m-1]的最长公共前缀
const int N = "Edit";
int kmp[N], extend[N];
void pre_ekmp(char x[], int m) {
    kmp[0] = m;
    int j = 0;
    while (j + 1 < m && x[j] == x[j + 1]) j++;
    kmp[1] = j;
    int k = 1;
    for (int i = 2; i < m; i++) {
        int p = kmp[k] + k - 1;
        int L = kmp[i - k];
        if (i + L < p + 1) kmp[i] = L;
        else {
            j = max(0, p - i + 1);
            while (i + j < m && x[i + j] == x[j]) j++;
            kmp[i] = j, k = i;
        }
    }
}
```

```
void ekmp(char x[], int m, char y[], int n) {
    pre_ekmp(x, m, kmp);
    int j = 0;
    while (j < n && j < m && x[j] == y[j]) j++;
    extend[0] = j;
    int k = 0;
    for (int i = 1; i < n; i++) {
        int p = extend[k] + k - 1;
        int L = kmp[i - k];
        if (i + L < p + 1) extend[i] = L;
        else {
            j = max(0, p - i + 1);
            while (i + j < n && j < m && y[i + j] == x[j]) j++;
            extend[i] = j, k = i;
        }
    }
}
```

2.4 Manacher

$O(n)$ 求解最长回文子串

```
const int N = "Edit";
char s[N], str[N << 1];
int p[N << 1]; // p[i] : 以 i 为中心的最长回文半径
void Manacher(char s[], int& n) {
    str[0] = '$', str[1] = '#';
    for (int i = 0; i < n; i++) str[(i << 1) + 2] = s[i], str[(i << 1) + 3] = '#';
    n = 2 * n + 2;
    str[n] = 0;
    int mx = 0, id;
    for (int i = 1; i < n; i++) {
        p[i] = mx > i ? min(p[2 * id - i], mx - i) : 1;
        while (str[i - p[i]] == str[i + p[i]]) p[i]++;
        if (p[i] + i > mx) mx = p[i] + i, id = i;
    }
}
int solve(char s[]) {
    int n = strlen(s);
    Manacher(s, n);
    return *max_elememt(p, p + n) - 1;
}
```

2.5 AC 自动机

```
const int N = "Edit";
struct Trie {
    int ch[N][26], f[N], val[N];
    int sz, rt;
    int New() { memset(ch[sz], -1, sizeof(ch[sz])), val[sz] = 0; return sz++; }
    void init() { sz = 0, rt = New(); }
    inline int idx(char c) { return c - 'A'; };
    void insert(const char* s) {
        int u = 0;
        for (int i = 0; s[i]; i++) {
            int c = idx(s[i]);
            if (ch[u][c] == -1) ch[u][c] = New();
```

```
            u = ch[u][c];
        }
        val[u]++;
    }
    void build() {
        queue<int> q;
        f[rt] = rt;
        for (int c = 0; c < 26; c++) {
            if (~ch[rt][c])
                f[ch[rt][c]] = rt, q.push(ch[rt][c]);
            else
                ch[rt][c] = rt;
        }
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            // val[u] |= val[f[u]];
            for (int c = 0; c < 26; c++) {
                if (~ch[u][c])
                    f[ch[u][c]] = ch[f[u]][c], q.push(ch[u][c]);
                else
                    ch[u][c] = ch[f[u]][c];
            }
        }
    }
    //返回主串中有多少模式串
    int query(const char* s) {
        int u = rt;
        int res = 0;
        for (int i = 0; s[i]; i++) {
            int c = idx(s[i]);
            u = ch[u][c];
            int tmp = u;
            while (tmp != rt) {
                res += val[tmp];
                val[tmp] = 0;
                tmp = f[tmp];
            }
        }
        return res;
    }
};
```

2.6 后缀数组 (SA)

```
//倍增算法构造后缀数组,复杂度O(nlogn)
const int N = "Edit";
struct Suffix_Array {
    char s[N];
    // rk[i] 表示后缀 i 的排名
    // sa[i] 表示将所有后缀排序后第 i 小的后缀的编号
    int sa[N], t[N], t2[N], c[N], rk[N], height[N];
    void build_sa(int m, int n) { //n为字符串的长度,字符集的值为0~m-1
        n++;
        int *x = t, *y = t2;
        //基数排序
        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; i--) sa[--c[x[i]]] = i;
        for (int k = 1; k <= n; k <<= 1) { //直接利用sa数组排序第二关键字
            int p = 0;
            for (int i = n - k; i < n; i++) y[p++] = i;
            for (int i = 0; i < n; i++)
                if (sa[i] >= k) y[p++] = sa[i] - k;
            //基数排序第一关键字
            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; i--) sa[--c[x[y[i]]]] = y[i];
            //根据sa和y数组计算新的x数组
            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] + k] == y[sa[i] + k]
     ? p - 1 : p++;
            if (p >= n) break; //以后即使继续倍增，sa也不会改变，推出
            m = p;             //下次基数排序的最大值
        }
        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 dp[N][30];
    void initrmq(int n) {
        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 rmq(int l, int r) {
        int k = 31 - __builtin_clz(r - l + 1);
        return min(dp[l][k], dp[r - (1 << k) + 1][k]);
    }
    int lcp(int a, int b) { // 求两个后缀的最长公共前缀
        a = rk[a], b = rk[b];
        if (a > b) swap(a, b);
        return rmq(a + 1, b);
    }
};
```

2.7 后缀自动机 (SAM)

```
const int N = "Edit";
struct SAM {
    int len[N << 1], link[N << 1], ch[N << 1][26];
    int num[N << 1]; //每个结点所代表的字符串的出现次数
```

```
    int sz, rt, last;
    int newnode(int x = 0) {
        len[sz] = x;
        link[sz] = -1;
        memset(ch[sz], -1, sizeof(ch[sz]));
        return sz++;
    }
    void init() { sz = last = 0, rt = newnode(); }
    void reset() { last = 0; }
    void extend(int c) {
        int np = newnode(len[last] + 1);
        int p;
        for (p = last; ~p && ch[p][c] == -1; p = link[p]) ch[p][c] = np;
        if (p == -1) link[np] = rt;
        else {
            int q = ch[p][c];
            if (len[p] + 1 == len[q]) link[np] = q;
            else {
                int nq = newnode(len[p] + 1);
                memcpy(ch[nq], ch[q], sizeof(ch[q]));
                link[nq] = link[q], link[q] = link[np] = nq;
                for (; ~p && ch[p][c] == q; p = link[p]) ch[p][c] = nq;
            }
        }
        last = np;
    }
    int topcnt[N], topsam[N << 1];
    void build(const char* s) { // 加入串后拓扑排序
        memset(topcnt, 0, sizeof(topcnt));
        for (int i = 0; i < sz; i++) topcnt[len[i]]++;
        for (int i = 0; i < N - 1; i++) topcnt[i + 1] += topcnt[i];
        for (int i = 0; i < sz; i++) topsam[--topcnt[len[i]]] = i;
        int u = rt;
        for (int i = 0; s[i]; i++) num[u = ch[u][s[i] - 'a']] = 1;
        for (int i = sz - 1; ~i; i--) {
            int u = topsam[i];
            if (~link[u]) num[link[u]] += num[u];
        }
    }
};
```

2.8 回文自动机（树）

```
const int N = "Edit";
struct Palindromic_Tree {
    int ch[N][26], f[N], len[N], s[N];
    int cnt[N]; // 结点表示的本质不同的回文串的个数(调用count()后)
    int num[N]; // 结点表示的最长回文串的最右端点为回文串结尾的回文串个数
    int last, sz, n;
    int newnode(int x) {
        memset(ch[sz], 0, sizeof(ch[sz]));
        cnt[sz] = num[sz] = 0, len[sz] = x;
        return sz++;
    }
    void init() {
        sz = 0;
        newnode(0), newnode(-1);
        last = n = 0, s[0] = -1, f[0] = 1;
```

```
16      }
17      int get_fail(int u) {
18          while (s[n - len[u] - 1] != s[n]) u = f[u];
19          return u;
20      }
21      void add(int c) { // c-='a'
22          s[++n] = c;
23          int u = get_fail(last);
24          if (!ch[u][c]) {
25              int np = newnode(len[u] + 2);
26              f[np] = ch[get_fail(f[u])][c];
27              num[np] = num[f[np]] + 1;
28              ch[u][c] = np;
29          }
30          last = ch[u][c];
31          cnt[last]++;
32      }
33      void count() {
34          for (int i = sz - 1; ~i; i--) cnt[f[i]] += cnt[i];
35      }
36  };
```

3 Data Structure

3.1 并查集

3.1.1 并查集 (启发式合并)

```cpp
int pre[N], rk[N];
void init() {
    for (int i = 1; i <= n; ++i) pre[i] = i, rk[i] = 1;
}
int find(int x) {
    return pre[x] == x ? x : pre[x] = find(pre[x]);
}
void merge(int x, int y) { // 启发式合并 (按秩合并)
    x = find(x), y = find(y);
    if (x == y) return;
    if (rk[y] < rk[x]) {
        pre[y] = x;
    } else {
        pre[x] = y;
        if (rk[x] == rk[y]) ++rk[y];
    }
}
```

3.1.2 带权并查集

```cpp
int find(int x) {
    if (x == pre[x]) return x;
    int fx = pre[x];
    pre[x] = find(pre[x]); // root
    val[x] += val[fx];
    // val[x] : x 到父节点的权值, val[fx] : 父节点到根的权值
    // x 的父亲从 fx 变为 pre[x] (根), 则新 val[x] = 旧 val[x] + val[fx]
    return pre[x]; // root
}
void join(int x, int y, int s) { // s 代表 x 到 y 的权值
    int fx = find(x), fy = find(y);
    if (fx != fy) {
        pre[fx] = fy;
        val[fx] = val[y] - val[x] + s;
        // 更新 val[fx], 可通过向量关系(可画图得出)得到计算公式
        // 公式不一定是上面这个, 可能还附带取模操作, 需视具体情况进行分析
    }
}
```

3.1.3 种类并查集

```cpp
int find(int x){
    if (pre[x] == x) return x;
    // x : 孙, fx : 子, pre[x] : 父(根)
    int  fx = pre[x];
    pre[x] = find(pre[x]);
    w[x] = cal_1(w[x], w[fx]);
    /* w[x] : x 与父亲的"关系"(由题目决定)
     * 路径压缩后 x 的父亲从 fx 变为 pre[x]
     * 需通过 w[x] 和 w[fx] 计算新的 w[x]
     * 可将情况列出来, 然后找规律 */
    return pre[x];
```

```
}
void join(int x, int y) {
    int fx = find(x), fy = find(y);
    if (fx != fy) {
        pre[fx] = fy;
        w[fx] = cal_2(w[x], w[y]);
        /* fx 的父亲变更为 fy
         * 需通过 w[x], w[y], x 与 y 的关系(join 操作隐含) 更新 w[fx]
         * 也可将情况列出来, 然后找规律 */
    }
}
```

3.1.4 可持续化并查集

```
/* 给定 n 个集合, 第 i 个集合内初始状态下只有一个数, 为 i
 * 有 m 次操作. 操作分为 3 种:
 * 1 a b : 合并 a, b 所在集合
 * 2 k   : 回到第 k 次操作(执行三种操作中的任意一种都记为一次操作)之后的状态
 * 3 a b : 询问 a, b 是否属于同一集合, 如果是则输出 1 , 否则输出 0  */
// time: O(mlog2n) splae: O(mlogn)  n: 结点数, m: 操作数
#define M 200005
#define L(p) t[p].l
#define R(p) t[p].r
struct Node { int l, r, fa, dep; } t[M * 20]; // O(mlogn)
int rt[M];
int n, m, opt, x, y, fx, fy, cnt;
void build(int &p, int l, int r) { // 初始化并查集
    p = ++cnt;
    if (l == r) { t[p].fa = l; return; }
    int mid = (l + r) >> 1;
    build(L(p), l, mid);
    build(R(p), mid + 1, r);
}
// 合并 x, fa 集合 (x 向 fa 合并)
void merge(int &now, int pre, int x, int fa, int l = 1, int r = n) {
    t[now = ++cnt] = t[pre];
    if (l == r) { t[now].fa = fa; return; }
    int mid = (l + r) >> 1;
    if (x <= mid) {
        merge(L(now), L(pre), x, fa, l, mid);
    } else {
        merge(R(now), R(pre), x, fa, mid + 1, r);
    }
}
void update(int p, int x, int l = 1, int r = n) { // dep[x] + 1
    if (l == r) { ++t[p].dep; return; }
    int mid = (l + r) >> 1;
    if (x <= mid) {
        update(L(p), x, l, mid);
    } else {
        update(R(p), x, mid + 1, r);
    }
}
int query(int p, int x, int l = 1, int r = n) { // num(x)
    if (l == r) return p;
    int mid = (l + r) >> 1;
    if (x <= mid) return query(L(p), x, l, mid);
    return query(R(p), x, mid + 1,  r);
```

```
}

int find(int tim, int x) { // 找根(无路径压缩), 返回结点编号
    int p = query(tim, x); // 线段树结点
    return t[p].fa ^ x ? find(tim, t[p].fa) : p;
} // 对照: return pre[x] ^ x ? find(pre[x]) : x

int main() {
    scanf("%d%d", &n, &m);
    build(*rt, 1, n); // 建立并查集
    for (int i = 1; i <= m; ++i) {
        scanf("%d%d", &opt, &x);
        switch (opt) {
            case 1:
                scanf("%d", &y);
                rt[i] = rt[i - 1];
                fx = find(rt[i], x), fy = find(rt[i], y);
                if (fx == fy) break;
                if (t[fx].dep > t[fy].dep) swap(fx, fy);
                merge(rt[i], rt[i - 1], t[fx].fa, t[fy].fa); // 启发式合并, fa[fx] = fy;
                if (t[fx].dep == t[fy].dep) update(rt[i], t[fy].fa);
                // dep 相同, 则 dep = dep + 1
                break;
            case 2: rt[i] = rt[x]; break;
            case 3:
                scanf("%d", &y);
                rt[i] = rt[i - 1];
                puts(find(rt[i], x) ^ find(rt[i], y) ? "0" : "1");
                break;
        }
    }
    return 0;
}
```

3.1.5 可撤销并查集

```
// 用按秩合并实现, 不能路径压缩
#include <bits/stdc++.h>
using namespace std;
class UFS {
    private:
        int *fa, *rk;
        stack<pair<int*, int> > stk;
    public:
        UFS() {}
        UFS(int n)  {
            fa = new int [(const int) n + 1];
            rk = new int [(const int) n + 1];
            memset(rk, 0, sizeof rk);
            for (int i = 1; i <= n; ++i) fa[i] = i;
        }
        inline int find(int x) {
            while (x ^ fa[x]) x = fa[x];
            return x;
        }
        inline int merge(int x, int y)  {
            x = find(x), y = find(y);
            if (x == y) return 0;
```

```
            if (rk[x] <= rk[y]) {
                stk.push({fa + x, fa[x]});
                fa[x] = y;
                if (rk[x] == rk[y])  {
                    stk.push({rk + y, rk[y]});
                    ++rk[y];
                    return 2;
                }
                return 1;
            }
            stk.push({fa + y, fa[y]});
            return fa[y] = x, 1;
        }
        inline void undo()  { // 撤销
            *stk.top().first = stk.top().second;
            stk.pop();
        }
} T;
```

3.1.6 跳点

```
// 例: 跳过值为 1 的数
l = find(l);
while (l <= r) {
    a[l] = sqrt(a[l]);
    if (a[l] <= 1) { // 满足跳过条件
        pre[l] = l + 1; // 将他连向下一个数
    }
    l = find(l + 1); // 跳到 l + 1 连向的数
}
```

3.1.7 习题 1

```
// 共两个集合, 每次将两个人划分到不同的两个集合中, 如果两个人已在一个集合则结束
int pre[N], foe[N];
// find, merge 为基本操作
{ // x, y 两人
    if (find(x) == find(y)) { // 两个人已在一个集合
        puts("END"), exit(0);
    }
    if (!foe[x]) foe[x] = y; // x 与 y 不在一个集合
    if (!foe[y]) foe[y] = x; // 同理
    merge(foe[x], y); // x 的敌人与 y 在一个集合, 因为 x 与 y 不在一个集合
    merge(foe[y], x); // 同理
}
```

3.1.8 习题 2

```
/*
   一个长度为 n 的大数, 用 S1 S2 S3 · · · Sn 表示, 其中 Si 表示数的第 i 位, S1 是数的最高位.
   告诉你一些限制条件, 每个条件 表示为四个数, l1, r1, l2, r2,
   即两个长度相同的区间, 表示子串 Sl1 Sl1+1 Sl1+2 · · · Sr1 与 Sl2 Sl2+1 Sl2+2 · · · Sr2 完全
    相同。
   问满足以上所有条件的数有多少个.
 */
// 并查集 + ST 表
int fa[N][20]; // f[i][j] 表示 [i, i + 2 ^ j - 1] 的所属联通块
```

```
inline int find(int x, int k) { return fa[x][k] == x ? x : fa[x][k] = find(fa[x][k], k)
    ; }
inline void merge(int x, int y, int k) {
    int fx = find(fa[x][k], k);
    int fy = find(fa[y][k], k);
    if (fx != fy) fa[fx][k] = fy;
}

int main() {
    n = rd(), m = rd(), mx = lg[n]; // lg[] 需预处理
    for (int i = 1; i <= n; ++i) {
        for (int j = 0; j <= mx; ++j) { // 0
            fa[i][j] = i; // 并查集初始化
        }
    }
    for (int i = 1; i <= m; ++i) {
        l1 = rd(), r1 = rd(), l2 = rd(), r2 = rd();
        int k = lg[r1 - l1 + 1];
        merge(l1, l2, k); // 将两个区间都拆成两部分进行合并
        merge(r1 - (1 << k) + 1, r2 - (1 << k) + 1, k);
    }
    for (int j = mx; j; --j) { // 从大区间往小区间合并
        for (int i = 1; i + (1 << j) - 1 <= n; ++i) {
            int d = find(i, j);
            if (d != i) {
                merge(i, d, j - 1); // 用大区间的联通块编号 d 合并小区间
                merge(i + (1 << (j - 1)), d + (1 << (j - 1)), j - 1);
            }
        }
    }
    for (int i = 1; i <= n; ++i) {
        if (fa[i][0] == i) ++cnt; // 统计联通块个数
    }
    // 10 ^ (联通块个数 - 1) 即为答案
    printf("%lld\n", ans * qpow(10, cnt - 1) % P);
    return 0;
}
```

3.2 单调栈

```
// 求出数列中第 i 个元素之后第一个大于 a[i] 的元素下标
int a[N], st[N], ans[N], t; // 数列, 栈, 结果
for (int i = n; i; --i) { // 正向读数据, 反向处理
    while (cnt && a[i] >= a[st[t]]) --t; // 保证栈单调递增
    ans[i] = st[t];
    st[++t] = i;
}
```

3.3 单调队列

```
/*
 * 有一个长为 n 的序列 a, 以及一个大小为 k 的窗口.
 * 现在这个从左边开始向右滑动, 每次滑动一个单位,
 * 求出每次滑动后窗口中的最大值和最小值.
 */
#include <cstdio>
#define N 1000006
```

```cpp
int n, k, cnt1, cnt2; // n : 序列长度, k : 窗口大小
int a[N], mx[N], mn[N];
struct Deque { // 双端队列
    int front_, rear, a[N]; // front_ 在左, rear 在右
    Deque() { front_ = 0, rear = -1; }
    void push(int x) { a[++rear] = x; } // 从尾部加
    void pop_front() { ++front_; }
    void pop_back()  { --rear; }
    int  front() { return a[front_]; }
    int  back()  { return a[rear]; }
    int  size()  { return rear - front_ + 1; }
    bool empty() { return rear < front_; }
} qx, qn;
for (int i = 1; i <= n; ++i) { // 滑动窗口
    while (!qx.empty() &&   qx.front() <= i - k) qx.pop_front(); // 最早加入的下标在窗口外
    while (!qn.empty() &&   qn.front() <= i - k) qn.pop_front(); // 最晚加入的下标在窗口外
    while (!qx.empty() && a[qx.back()] <= a[i] ) qx.pop_back(); // 单调递减
    while (!qn.empty() && a[qn.back()] >= a[i] ) qn.pop_back(); // 单调递增
    qx.push(i), qn.push(i); // 插入下标
    if (i >= k) {
        mx[++*mx] = a[qx.front()]; // 最大值
        mn[++*mn] = a[qn.front()]; // 最小值
    }
}
```

3.4 对顶堆

3.4.1 维护第 k 小 (大)

```cpp
// 时间 : O(logn)
// 若求第 k 大, 则使 qn.size() 为 k , qn.top()即为第k大
// 若求第 k 小, 则使 qx.size() 为 k , qx.top()即为第k小
// https://www.luogu.com.cn/problem/P1801
struct Heap { // 大顶堆
    int siz, a[N];
    Heap() { siz = 0; memset(a, 0, sizeof(a)); }
    void push(int x) {
        a[++siz] = x;
        int p = siz, q = p >> 1;
        while (q && a[p] > a[q]) {
            swap(a[p], a[q]);
            p = q, q >>= 1;
        }
    }
    int top() { return a[1]; }
    void pop() {
        a[1] = a[siz--];
        int p = 1, l = 2, r = 3;
        while (l <= siz) {
            int q = l;
            if (r <= siz && a[r] > a[l]) q = r;
            if (a[p] > a[q]) return;
            swap(a[p], a[q]);
            p = q, l = p << 1, r = p << 1 ^ 1;
        }
    }
    int size() { return siz; }
    bool empty() { return !siz; }
} qx, qn; // 大, 小根堆
```

```cpp
int n, m, x, t1, t2, a[N];
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i)
        scanf("%d", &a[i]);
    for (int i = 1; i <= m; ++i) {
        scanf("%d", &x);
        while (t1 < x) qx.push(a[++t1]);
        ++t2;
        while (qx.size() > t2) {
            qn.push(-qx.top()), qx.pop();
        }
        printf("%d\n", qx.top()); // 第 k 小
        // 假设这一次为求第 k1 小, 下一次为求第 k2 小
        // 若 k2 > k1 , 则需将 qn 中的 k2 - k1 个数放回 qx 中
        // 避免第 k2 小在 qn 中的情况
        //
        // 因为下一次是求第 t2 + 1 小, 所以要预先往 qx 放回一个数
        // 避免第 t2 + 1 小在 qn 中的情况
        if (!qn.empty()) {
            qx.push(-qn.top()), qn.pop();
        }
    }
    return 0;
}
```

3.4.2 维护中位数

```cpp
// 动态求中位数
struct Queue {...}; // 大根堆
Queue qx, qn; // 大, 小根堆
// 小根堆在上(顶朝下), 大根堆在下(顶朝上), 形成漏斗状结构
qx.push(x); // 先放一个
for (int i = 2; i <= n; ++i) {
    scanf("%d", &x);
    if (x < qx.top()) qx.push(x); // 插入
    else qn.push(-x);
    if (qn.size() > qx.size() + 1) { // 调整
        qx.push(-qn.top()), qn.pop();
    }
    else if (qx.size() > qn.size() + 1) { // 调整
        qn.push(-qx.top()), qx.pop();
    }
    if (i & 1) // 每当插入奇数个数字时求中位数
        ans[++*ans] = qx.size() > qn.size() ? qx.top() : -qn.top();
}
```

3.5 ST 表

3.5.1 一维 RMQ

```cpp
// 作用 : 寻找区间最值
// 空间 O(nlogn)
int n, m;
int lg[N], d[N][20];
// N : 区间长度
// 2^16 =  65536 < N = 1e5
// 2^17 = 131072 > N = 1e5
```

```
// d[i][j] => i ~ (i + 2^j - 1)
void init() { // O(nlogn) 预处理
    for (int j = 1; j <= 16; ++j) // j
        for (int i = 1; i + (1 << j) - 1 <= n; ++i) // i
            d[i][j] = max(d[i][j - 1], d[i + (1 << (j - 1))][j - 1]);
}
int query(int l, int r) { // O(1) 查询 [l, r] 的最大值
    int k = lg[r - l + 1]; // 需要 O(n) 预处理 lg[]
    // int k = 31 - __builtin_clz(r - l + 1);
    // x = l, y = r - 2^k + 1 <== y + 2^k - 1 = r
    return max(d[l][k], d[r - (1 << k) + 1][k]);
}
```

3.5.2 二维 RMQ

```
// 作用 : 求子矩阵的最值问题
int d[N][20][N][20];
 // d[i][j][k][l] : 代表i ~ i + 2 ^ k  j ~ j + 2 ^ l 这个矩阵中的最大值
void init(int n, int m) { // O(n * m * logn * logm)
    for (int i = 0; (1 << i) <= n; i++)
        for (int j = 0; (1 << j) <= m; j++) {
            if (i == 0 && j == 0) continue;
            for (int row = 1; row + (1 << i) - 1 <= n; row++)
                for (int col = 1; col + (1 << j) - 1 <= m; col++)
                    if (i) {
                        d[row][col][i][j] = max(d[row][col][i - 1][j],
                                d[row + (1 << (i - 1))][col][i - 1][j]);
                    } else {
                        d[row][col][i][j] = max(d[row][col][i][j - 1],
                                d[row][col + (1 << (j - 1))][i][j - 1]);
                    }
        }
}
int query(int x1, int y1, int x2, int y2) { // O(1)
    int kx = lg[x2 - x1 + 1];
    int ky = lg[y2 - y1 + 1];
    // int kx = 31 - __builtin_clz(x2 - x1 + 1);
    // int ky = 31 - __builtin_clz(y2 - y1 + 1);
    int m1 = d[x1][y1][kx][ky];
    int m2 = d[x2 - (1 << kx) + 1][y1][kx][ky];
    int m3 = d[x1][y2 - (1 << ky) + 1][kx][ky];
    int m4 = d[x2 - (1 << kx) + 1][y2 - (1 << ky) + 1][kx][ky];
    return max({m1, m2, m3, m4});
}
```

3.6 线性基

3.6.1 线性基

```
// 给定 n 个整数(数字可能重复), 求在这些数中选取任意个, 使得他们的异或和最大
namespace Linear_Basis {
    const int mx = 50; // 数字 <= 2 ^ mx
    int n, tot; // n: 总数字个数, tot: 线性基中数字个数
    ll p[mx + 1], x, ans; // p[] : 线性基
    inline void insert(ll x) { // 常规写法, 往线性基插入一个数
        for (int i = 50; i >= 0; --i) if (x >> i) {
            if (!p[i]) return p[i] = x, ++tot, void();
            x ^= p[i];
```

```
10          }
11      }
12      inline void insert(ll x) { // 个人改进写法 ( 也许常数会小一些 )
13          while (x) {
14              int i = log2(x); // x 在 int 范围内才能将 log2(x) 替换为 31 - __builtin_clz(x)
15              if (!p[i]) return p[i] = x, ++tot, void();
16              x ^= p[i];
17          }
18      }
19      // 判断一个数是否能被线性基中的元素异或得到出来
20      // 如果能被表示出来, 则满足的子序列个数为
21      // 2 ^ (线性有关的数字个数 = 总数字个数(尝试插入的数字个数) - 线性无关的数字个数(线性基中数字个
        数))
22      // = 2 ^ (n - tot)
23      inline int check(int x) {
24          while (x) {
25              int i = log2(x);
26              if (!p[i]) return 0; // 不能表示
27              x ^= p[i];
28          }
29          return 1; // 能表示
30      }
31      // 在一个序列中, 取若干个数, 使得它们的异或和最大
32      inline ll query_max() {
33          ll ans = 0;
34          for (int i = mx; i >= 0; --i) {
35              if ((ans ^ p[i]) > ans) ans ^= p[i];
36          }
37          return ans;
38      }
39      // 在一个序列中, 取若干个数, 使得它们的异或和最小
40      inline ll query_min() {
41          if (tot < n) return 0; // 原来的序列可以异或出 0
42          for (int i = 0; i <= mx; ++i) if (p[i]) return p[i];
43      }
44      inline void rebuild() { // kth 的预处理 (对 p[] 进行处理)
45          all = (1ll << tot) - 1; // 线性基所有不同异或和的数量
46          for (int i = 1; i <= mx; ++i) {
47              for (int j = 0; j < i; ++j) {
48                  if (p[i] >> j & 1) p[i] ^= p[j];
49              }
50          }
51      }
52      // 一个序列中取任意个元素进行异或, 查询能异或出的所有数字中的第 k 小
53      inline ll kth(ll k) { // 需要先调用一次 rebuild
54          if (tot < n && !--k) return 0; // 能异或出 0, k = 1 直接输出 0, 否则去掉 0 这种情况
55          if (k > all) return -1;
56          ll ans = 0;
57          for (int i = 0; i <= mx; ++i) if (p[i]) {
58              if (k & 1) ans ^= p[i];
59              k >>= 1;
60          }
61          return ans;
62      }
63      // 获取一个数在能异或出的所有数字中的排名
64      inline ll query_rank(ll x) {
65          ll k = 0, now = 1;
66          for (int i = 0; i <= mx; ++i) if (p[i]) {
67              if (x >> i & 1) k += now;
```

```
            now <<= 1;
        }
        if (tot == n) return k;
        return k * qpow(2, n - tot) + 1;
    }
}
```

3.6.2 实数线性基

```
/* n 件装备，每件装备 m 个属性，每件装备还有个价格.
 * 如果手里有的装备的每一项属性为它们分配系数(实数)后可以相加得到某件装备，
 * 则不必要买这件装备. 求能购买的最多装备数量及最多装备下的最小花费.  */
#include <bits/stdc++.h>
using namespace std;
typedef long double LD;
const LD eps = 1e-6;
struct Node {
    int w; LD x[N];
    bool operator < (const Node &b) const { return w < b.w; }
} t[N];
int p[N];
int n, m, cnt, sum;
void insert(int i) {
    for (int j = 1; j <= m; ++j) {
        if (fabs(t[i].x[j]) > eps) {
            if (!p[j]) {
                p[j] = i;
                sum += t[i].w, ++cnt;
                return;
            }
            LD tmp = t[i].x[j] / t[p[j]].x[j];
            for (int k = j; k <= m; ++k)
                t[i].x[k] -= t[p[j]].x[k] * tmp;
        }
    }
}
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j)
            scanf("%LF", &t[i].x[j]); // 第 i 件装备的第 j 个属性
    for (int i = 1; i <= n; ++i)
        scanf("%d", &t[i].w); // 第 i 件装备的花费
    sort(t + 1, t + 1 + n);
    for (int i = 1; i <= n; ++i) insert(i);
    printf("%d %d\n", cnt, sum);
    return 0;
}
```

3.6.3 前缀线性基 (区间线性基)

```
// 前缀线性基 (在线算法)  O(nlogn)
// 查询: 给定 l, r, 在 [l, r] 中选取任意个, 使得他们的异或和最大
#include <bits/stdc++.h>
using namespace std;
#define N 1000005
const int mx = 19;
int n, m, l, r;
```

```
struct P { // Prefix_Linear_Basis 前缀线性基
    int p[mx + 1], pos[mx + 1];
    inline void init() { memset(p, 0, sizeof(p)), memset(pos, 0, sizeof(pos)); }
    inline void insert(const P &a, int now, int x) {
        *this = a; // 别忘了
        for (int i = mx; i >= 0; --i) {
            if (x >> i) {
                if (!p[i]) return p[i] = x, pos[i] = now, void();
                if (pos[i] < now) swap(pos[i], now), swap(p[i], x);
                x ^= p[i]; // 拿 pos 小的替换 pos 大的继续插入, 确保线性基中 pos 值尽量大
            }
        }
    }
    inline int query(int L) { // 查询该线性基中所有 ≥L 的最大值
        int ans = 0;
        for (int i = mx; i >= 0; --i) {
            if (p[i] && pos[i] >= L) ans = max(ans, ans ^ p[i]);
        }
        return ans;
    }
} a[N];
inline int rd();
int main() {
    while (T--) {
        for (int i = 1; i <= n; ++i) a[i].init();
        n = rd();
        for (int i = 1; i <= n; ++i) {
            a[i].insert(a[i - 1], i, rd()); // 构建前缀线性基
        }
        m = rd();
        while (m--) {
            if (rd()) { // 往数列末尾添加一个数字
                ++n, a[n].insert(a[n - 1], n, rd());
            } else {
                l = rd(), r = rd();
                printf("%d\n", a[r].query(l)); // 查询
            }
        }
    }
    return 0;
}
```

3.6.4 线段树合并线性基 (区间线性基)

```
// 线段树维护线性基, 暴力合并 O(n logn logn logn)
// 查询: 给定 l, r, 在 [l, r] 中选取任意个, 使得他们的异或和最大
#include <bits/stdc++.h>
using namespace std;

#define N 500005
#define L t[p].l
#define R t[p].r
#define ls(p) (p << 1)
#define rs(p) (p << 1 ^ 1)
#define tls(p) t[ls(p)]
#define trs(p) t[rs(p)]

const int mx = 19;
```

```cpp

struct Node { int l, r, p[mx + 1]; } t[N << 2];

int n, m, x, y;
int a[N], tmp[mx + 1];

// inline int log(long long x) { return 63 - __builtin_clzll(x); } // 64位
inline int log(int x) { return 31 - __builtin_clz(x); } // 32位

inline void insert(int *p, int x) {
    while (x) {
        int i = log(x);
        if (!p[i]) return p[i] = x, void();
        x ^= p[i];
    }
}

void build(int l, int r, int p) {
    L = l, R = r;
    for (int i = l; i <= r; ++i) insert(t[p].p, a[i]);
    if (l == r) return;
    int mid = (l + r) >> 1;
    build(l, mid, ls(p)), build(mid + 1, r, rs(p));
}

void update(int x, int k, int p = 1) { // 往桶 x 插入一个数
    insert(t[p].p, k); if (L == R) return;
    update(x, k, x <= tls(p).r ? ls(p) : rs(p));
}

void query(int x, int y, int p = 1) {
    if (x > R || y < L) return;
    if (x <= L && y >= R) {
        for (int i = mx; i >= 0; --i) insert(tmp, t[p].p[i]);
        return;
    }
    query(x, y, ls(p)), query(x, y, rs(p));
}

inline int ask(int l, int r) {
    memset(tmp, 0, sizeof(tmp));
    int ans = 0; query(l, r);
    for (int i = mx; i >= 0; --i) {
        if ((ans ^ tmp[i]) > ans) ans ^= tmp[i];
    }
    return ans;
}

inline int rd();
int main() {
    n = rd();
    for (int i = 1; i <= n; ++i) a[i] = rd();
    build(1, n, 1);
    m = rd();
    while (m--) {
        x = rd(), y = rd();
        printf("%d\n", ask(x, y));
    }
    return 0;
```

```
}
```

3.6.5 最大 xor 和路径

```
// 求一条 1 到 n 的路径，使得路径上经过的边的权值的 xor 和最大，输出这个 最大 xor 和 (luogu P4145 )
// 做法：找出所有环，扔进线性基，随便找一条链，以它作为初值求最大异或和就可以了
#include <bits/stdc++.h>
using namespace std;

#define N 50004
#define M 100005

typedef long long ll;

struct Edge { int to, nx; ll w; } e[M << 1];

ll z, ans;
ll p[65], w[N];
int h[N], vis[N];
int n, m, x, y, cnt;

inline void add(int u, int v, ll w) { e[++cnt] = {v, h[u], w}, h[u] = cnt; }
inline void insert(ll x) {
    for (int i = 60; i >= 0; --i) if (x >> i) {
        if (!p[i]) return p[i] = x, void();
        x ^= p[i];
    }
}

void dfs(int u) {
    vis[u] = 1;
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (!vis[v]) w[v] = w[u] ^ e[i].w, dfs(v);
        else insert(w[u] ^ e[i].w ^ w[v]);
    }
}

int main() {
    scanf("%d%d", &n, &m);
    while (m--) {
        scanf("%d%d%lld", &x, &y, &z);
        add(x, y, z), add(y, x, z);
    }
    dfs(1), ans = w[n];
    for (int i = 60; i >= 0; --i) {
        if ((ans ^ p[i]) > ans) ans ^= p[i];
    }
    printf("%lld\n", ans);
    return 0;
}
```

3.7 舞蹈链

```
// 精确覆盖问题
/* 给定一个 N行 M列的矩阵，矩阵中每个元素要么是 1，要么是 0
```

```
 * 你需要在矩阵中挑选出若干行，使得对于矩阵的每一列 j，
 * 在你挑选的这些行中，有且仅有一行的第 j 个元素为 1 */
#define N 250505 // n * m + m <= N
int n, m, x, cnt; // n : 行, m : 列
int h[N], siz[N], ans[N], l[N], r[N], u[N], d[N], row[N], col[N];
// h[]: 每行的头结点，siz[]: 每列的节点数，ans[]: 选了那些集合
// l, r, u, d, row, col 每个点的左右上下指针，所在行列
void init(int m) { // 在第 0 行插入 (m + 1) 列
    for (int i = 0; i <= m; ++i) { // 0 到 m
        r[i] = i + 1;
        l[i] = i - 1;
        u[i] = d[i] = i;
    }
    cnt = m + 1;
    *l = m, r[m] = 0;
    memset(h, -1, sizeof(h));
    memset(siz, 0, sizeof(siz));
}
void link(int R, int C) { // 在 R 行 C 列插入点
    ++siz[C];
    u[cnt] = C, d[cnt] = d[C];
    u[d[C]] = cnt, d[C] = cnt;
    row[cnt] = R, col[cnt] = C;
    if (~h[R]) {
        r[cnt] = h[R], l[cnt] = l[h[R]];
        r[l[h[R]]] = cnt, l[h[R]] = cnt;
    } else { // 该行没点，直接插入
        h[R] = l[cnt] = r[cnt] = cnt;
    }
    ++cnt;
}
void remove(int C) { // 删除涉及 C 列的集合
    r[l[C]] = r[C], l[r[C]] = l[C];
    for (int i = d[C]; i != C; i = d[i])
        for (int j = r[i]; j != i; j = r[j]) {
            u[d[j]] = u[j], d[u[j]] = d[j];
            --siz[col[j]];
        }
}
void resume(int C) { // 恢复涉及 C 列的集合
    for (int i = u[C]; i != C; i = u[i]) {
        for (int j = l[i]; j != i; j = l[j]) {
            u[d[j]] = j, d[u[j]] = j;
            ++siz[col[j]];
        }
    }
    r[l[C]] = C, l[r[C]] = C;
}
bool dance(int dep) {
    if (!*r) {
        for (int i = 0; i < dep; ++i)
            printf("%d ", ans[i]);
        return 1;
    }
    int c = *r;
    for (int i = c; i; i = r[i]) {
        // 从元素最少的列开始处理
        if (siz[i] < siz[c]) c = i;
    }
```

```
    remove(c); // 删除该列的行
    for (int i = d[c]; i != c; i = d[i]) {
        ans[dep] = row[i]; // 选择保留该行
        // 删除被选择行所包含列涉及的所有行
        for (int j = r[i]; j != i; j = r[j]) remove(col[j]);
        if (dance(dep + 1)) return 1;
        for (int j = l[i]; j != i; j = l[j]) resume(col[j]);
        // 恢复被选择行所包含列涉及的所有行
    }
    resume(c); // 恢复删除该列的行
    return 0;
}
int main() {
    n = rd(), m = rd(), init(m);
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j) {
            x = rd();
            if (x) link(i, j);
        }
    if (!dance(0)) puts("No Solution!");
    return 0;
}
```

3.8 线段树

3.8.1 区间乘加区间和

```
struct Node { int l, r; ll sum, add, mul; } t[N << 2];
inline void pushup(int p) { t[p].sum = (tls(p).sum + trs(p).sum) % P; }
void build(int l, int r, int p) {
    L = l, R = r, t[p].mul = 1; // 初始化为 1
    if (l == r) { t[p].sum = a[l] % P; return; }
    int mid = (l + r) >> 1;
    build(l, mid, ls(p));
    build(mid + 1, r, rs(p));
    pushup(p);
}
void push(int p, ll mul, ll add) {
    t[p].sum = (t[p].sum * mul + add * (R - L + 1)) % P;
    t[p].add = (t[p].add * mul + add) % P;
    t[p].mul = (t[p].mul * mul) % P;
}
void pushdown(int p) {
    push(ls(p), t[p].mul, t[p].add);
    push(rs(p), t[p].mul, t[p].add);
    t[p].mul = 1, t[p].add = 0;
}
void updMul(int p, int x, int y, int k) { // [x, y] mul k
    if (x > R || y < L) return;
    if (x <= L && y >= R) {
        t[p].add = (t[p].add * k) % P;
        t[p].mul = (t[p].mul * k) % P;
        t[p].sum = (t[p].sum * k) % P;
        return;
    }
    pushdown(p), updMul(ls(p), x, y, k), updMul(rs(p), x, y, k), pushup(p);
}
void updAdd(int p, int x, int y, int k) { // [x, y] add k
    if (x > R || y < L) return;
```

```
    if (x <= L && y >= R) {
        t[p].add = (t[p].add + k) % P;
        t[p].sum = (t[p].sum + k * (R - L + 1)) % P;
        return;
    }
    pushdown(p), updAdd(ls(p), x, y, k), updAdd(rs(p), x, y, k), pushup(p);
}
```

3.8.2 zkw 线段树

```
ll sum[N << 2], mx[N << 2], mn[N << 2], add[N << 2];
// 若有区间修改操作, 则 sum 不需要差分, mx 和 mn 需要差分
void build(int n) { // 建树
    M = 1 << (32 - __builtin_clz(n)); // 1 << (log(n) + 1)
    for (int i = M + 1; i <= M + n; ++i)
        sum[i] = mx[i] = mn[i] = rd();
    for (int i = M - 1; i; --i) {
        int L = i << 1, R = L ^ 1;
        sum[i] = sum[L] + sum[R];
        mn[i] = min(mn[L], mn[R]);
        mx[i] = max(mx[L], mx[R]);
        mn[R] -= mn[i], mn[R] -= mn[i];
        mx[L] -= mx[i], mx[R] -= mx[i];
    }
}
// ls, rs : 左右两颗子树已覆盖的区间长度
// len : 该层每个节点的子树大小(一个节点覆盖的区间长度)
void update(int x, int y, int k) { // [x, y] add k
    ll A = 0, ls = 0, rs = 0, len = 1;
    for (x += M - 1, y += M + 1; x ^ y ^ 1; x >>= 1, y >>= 1, len <<= 1) {
        if (~x & 1) {
            add[x ^ 1] += k;
            mx[x ^ 1] += k;
            mn[x ^ 1] += k;
            ls += len;
        }
        if (y & 1) {
            add[y ^ 1] += k;
            mx[y ^ 1] += k;
            mn[y ^ 1] += k;
            rs += len;
        }
        sum[x >> 1] += k * ls; // 该节点表示的区间覆盖了 [x, y] 中 ls 个点
        sum[y >> 1] += k * rs; // 该节点表示的区间覆盖了 [x, y] 中 rs 个点
        A = min(mn[x], mn[x ^ 1]);
        mn[x] -= A, mn[x ^ 1] -= A, mn[x >> 1] += A;
        A = min(mn[y], mn[y ^ 1]);
        mn[y] -= A, mn[y ^ 1] -= A, mn[y >> 1] += A;
        A = max(mx[x], mx[x ^ 1]);
        mx[x] -= A, mx[x ^ 1] -= A, mx[x >> 1] += A;
        A = max(mx[y], mx[y ^ 1]);
        mx[y] -= A, mx[y ^ 1] -= A, mx[y >> 1] += A;
    }
    for (ls += rs; x; x >>= 1) {
        sum[x >> 1] += k * ls;
        A = min(mn[x], mn[x ^ 1]);
        mn[x] -= A, mn[x ^ 1] -= A, mn[x >> 1] += A;
        A = max(mx[x], mx[x ^ 1]);
```

```
        mx[x] -= A, mx[x ^ 1] -= A, mx[x >> 1] += A;
    }
}
ll qsum(int x, int y) { // sum[x, y]
    ll ls = 0, rs = 0, len = 1, ans = 0;
    for (x += M - 1, y += M + 1; x ^ y ^ 1; x >>= 1, y >>= 1, len <<= 1) {
        if (~x & 1) {
            ans += sum[x ^ 1] + len * add[x ^ 1];
            ls += len;
        }
        if (y & 1) {
            ans += sum[y ^ 1] + len * add[y ^ 1];
            rs += len;
        }
        if (add[x >> 1])
            ans += add[x >> 1] * ls;
        if (add[y >> 1])
            ans += add[y >> 1] * rs;
    }
    for (ls += rs, x >>= 1; x; x >>= 1)
        if (add[x])
            ans += add[x] * ls;
    return ans;
}
int qmin(int x, int y) { // min[x, y]
    ll L = 0, R = 0;
    if (x == y)   // 单点查询
        for (x += M; x; x >>= 1) L += mn[x];
        return L;
    }
    for (x = x + M - 1, y = y + M + 1; x ^ y ^ 1; x >>= 1, y >>= 1) {
        L += mn[x], R += mn[y];
        if (~x & 1) L = min(L, mn[x ^ 1]);
        if (y & 1) R = min(R, mn[y ^ 1]);
    }
    L += mn[x], R += mn[y];
    ll ans = min(L, R);
    while (x) ans += mn[x >>= 1];
    return ans;
}
// qmax 即把 qmin 中的 max, mx 换成 min, mn
int qmax(int x, int y) {...};
```

3.8.3 线段树合并

```
// 开新节点的合并方式, 不需要离线, 一边询问一边用
// 缺点 : 空间开销巨大  时空: O(nlogn)
int merge(int a, int b, int l = 1, int r = n) {
    if (!a || !b) return a | b;
    int p = ++cnt; // 新节点
    if (l == r) {
        t[p].sum = t[a].sum + t[b].sum;
        // merge meesege
        return;
    }
    int mid = (l + r) >> 1;
    L(p) = merge(L(a), L(b), l, mid);
    R(p) = merge(R(a), R(b), mid + 1, r);
```

```
    pushup(p);
    return p;
}
// 不开新结点的合并方式，只适合离线下来，合并完成后立即询问
int merge(int a, int b, int l = 1, int r = n) {
    if (!a || !b) return a | b;
    // t[a].sum += t[b].sum;
    // 向 sum 这种信息也可以直接在路径上加，这样就可以省去 l == r 了
    if (l == r) {
        t[a].sum += t[b].sum;
        // merge meesege
        return a;
    }
    int mid = (l + r) >> 1;
    L(a) = merge(L(a), L(b), l, mid);
    R(a) = merge(R(a), R(b), mid + 1, r);
    pushup(a);
    return a;
}
void dfs(int u, int pre) { // 树上差分的线段树合并操作
    for (auto &v : e[u])
        if (v != pre) {
            dfs(v, u);
            rt[u] = merge(rt[u], rt[v]);
        }
    if (t[rt[u]].sum) ans[u] = t[rt[u]].pos;
}
```

3.8.4 线段树合并 (空间优化)

```
// 线段树合并 ( 空间优化为 O(N * 2) )
// 合并的时候进行垃圾回收，update 中用自定义 New() 创建新结点
inline int New() { return rb ? rub[rb--] : ++cnt; }
inline void Del(int p) { rub[++rb] = p, t[p].l = t[p].r = t[p].tot = 0; }
void update(int &p, int x, int l = 1, int r = mxd) {
    if (!p) p = New(); //
    ++t[p].tot; // ... update messages ... //
    if (l == r) {
        // ... update messages ... //
        return;
    }
    int mid = (l + r) >> 1;
    if (x <= mid) update(t[p].l, x, l, mid);
    else update(t[p].r, x, mid + 1, r);
}
int merge(int a, int b, int l = 1, int r = mxd) {
    if (!a || !b) return a | b;
    t[a].tot += t[b].tot; // ... merge messages ... //
    if (l == r) {
        // ... merge messages ... //
        Del(b); // 回收
        return a;
    }
    L(a) = merge(L(a), L(b), l, mid);
    R(a) = merge(R(a), R(b), mid + 1, r);
    Del(b); // 回收
    return a;
}
```

```cpp
// 关键优化: 遍历儿子时先遍历 siz 大的儿子, 并且先 merge 再 update
int tmp[N], tp;
void dfs(int u) {
    int l = tp + 1; // + 1
    for (int i = h[u]; i; i = e[i].nx) // tmp 存儿子
        tmp[++tp] = e[i].to;
    int r = tp; //
    sort(tmp + l, tmp + 1 + r, [&] (int x, int y) { return siz[x] > siz[y]; }); // 按儿子 siz 从大到小排序
    for (int i = l; i <= r; ++i) {
        dfs(tmp[i]);
        rt[u] = merge(rt[u], rt[tmp[i]]); // 先 merge (不开新节点版)
    }
    update(rt[u], dep[u]); // 再 update
    ans[...] = query(...); // ... 记录答案 ... //
}
```

3.8.5 线段树分裂

```cpp
inline int New() { return rb ? rub[rb--] : ++cnt; }
inline void Del(int x) { t[x].l = t[x].r = t[x].tot = 0, rub[++rb] = x; }
// 不开新节点的合并, 把 b 合并给 a
int merge(int a, int b) {
    if (!a || !b) return a | b;
    t[a].tot += t[b].tot;
    L(a) = merge(L(a), L(b));
    R(a) = merge(R(a), R(b));
    DEL(b);
    return a;
}
// 将权值线段树 a [x ~ y]小 分裂为 a [x ~ k]小 和 b [k + 1, y]小
void split(int a, int &b, ll k) {
    if (!a) return;
    b = NEW(); // 新节点
    ll tot = t[L(a)].tot;
    if (k > tot) split(R(a), R(b), k - tot);
    else {
        R(b) = R(a);
        R(a) = 0;
        if (k < tot) split(L(a), L(b), k);
    }
    t[b].tot = t[a].tot - k;
    t[a].tot = k;
}
void SP(int x, int k1, int k2) { // 将树 x 中第 [k1 ~ k2] 小取出, 放入新树
    split(rt[x], rt[++cnt], k1);
    split(rt[cnt], tmp, k2);
    rt[x] = merge(rt[x], tmp);
}
```

3.8.6 线段树分裂例题 1

```cpp
// P2824 : 对于一个 1 ~ n 的排列, 询问 m 次区间局部排序后某个位置的数 (在线)
struct Node { // 权值线段树
    int l, r, tot; // tot 记录该区间数字个数
} t[N * 80]; // 每个询问有两次分裂操作, 故空间翻倍
set<int> s;
int rt[N], op[N];
```

```cpp
int n, m, opt, x, y, cnt;
/* 设 set 内每相邻两个元素表示为 from, nx
 * set 存区间, set 内相邻每两个元素构成一个区间 [from, next), 初始 set 内为 1 ~ n + 1
 * rt[from] : [from, next) 这个区间的线段树根节点
 * op[from] : [from, next) 这个区间的线段树的操作(tag), 0 表升序操作, 1 表降序操作 */
void update(int &p, int x, int l = 0, int r = n) {
    p = ++cnt;
    ++t[p].tot;
    if (l == r) return;
    int mid = (l + r) >> 1;
    if (x <= mid)
        update(L(p), x, l, mid);
    else
        update(R(p), x, mid + 1, r);
}
// 因为这颗树被分裂到只剩一个权值了, 所以可以直接采用以下查询方式
int query(int p, int l = 0, int r = n) {
    if (l == r) return l; // 返回权值
    int mid = (l + r) >> 1;
    if (L(p)) return query(L(p), l, mid);
    return query(R(p), mid + 1, r);
}
int merge(int a, int b) {
    if (!a || !b)
        return a | b;
    t[a].tot += t[b].tot;
    L(a) = merge(L(a), L(b));
    R(a) = merge(R(a), R(b));
    return a;
}
// k 表示分裂后旧树的区间元素个数
void split(int a, int &b, int k, int opt) {
    if (t[a].tot == k) return;
    b = ++cnt;
    t[b].tot = t[a].tot - k; // 新树元素个数
    t[a].tot = k; // 旧树元素个数
    if (opt) { // 降序
        if (k <= t[R(a)].tot) { // 把小的数分到右边
            split(R(a), R(b), k, opt);
            L(b) = L(a), L(a) = 0;
        } else {
            split(L(a), L(b), k - t[R(a)].tot, opt); // k - t[L(a)].tot
        }
    } else { // 升序
        if (k <= t[L(a)].tot) { // 把小的数分到左边
            split(L(a), L(b), k, opt);
            R(b) = R(a), R(a) = 0;
        } else {
            split(R(a), R(b), k - t[L(a)].tot, opt); // k - t[L(a)].tot
        }
    }
}
set<int>::iterator SP(int x) { // 将区间分裂为 [left, x), [x, right]
    auto p = s.lower_bound(x); // 找 *p >= x
    if (*p == x) return p // *p = x, 无需分裂
    --p; // 往左移动一位, 此时 *p < x
    op[x] = op[*p]; // copy
    split(rt[*p], rt[x], x - *p, op[x]); // x - *p : 分裂后旧树的元素个数
    return s.insert(x).first; // 位置 x 的 iterator
```

```
}
int main() {
    n = rd(), m = rd();
    s.insert(n + 1); // 插入 n + 1 防止在 sp 函数中 lower_bound 返回 s.end()
    for (int i = 1; i <= n; ++i) {
        x = rd();
        update(rt[i], x);
        s.insert(i); // 初始化
    }
    while (m--) {
        opt = rd(), x = rd(), y = rd();
        auto l = SP(x), r = SP(y + 1); // x, y + 1
        for (auto i = ++l; i != r; ++i) // 注意这里 i = ++l; i != r, 所以是对区间 (x, y] 内的元素合并
            rt[x] = merge(rt[x], rt[*i]);
        s.erase(l, r); // 删除 [l, r) 位置的元素
        op[x] = opt; // 记录 [x, y) 这个区间的操作
    }
    SP(pos, SP(pos + 1);
    printf("%d\n", query(rt[x])); // 查询 pos 这个位置的数
    return 0;
}
```

3.8.7 线段树分裂例题 2

```
// CF558E : 对一个只含小写字母的字符串进行 M 次区间局部排序, 输出最终的字符串 (在线)
set<int> s;
char ss[N];
int n, m, x, y, k, tot;
int rt[N], cnt[26], opt[N];
struct Node { int l, r, tot; } t[N * 80]; // ... 表示与例题一操作相同
void update(int &p, int x, int l = 0, int r = 25) { ... }
int merge(int a, int b) { ... }
void split(int a, int &b, int k, int op) {
    ...
    t[a].tot = k;
    if (!L(a) && !R(a)) return; // 如果有重复元素, 则需要额外加上这句
    if (op)  ...
    else ...
}
set<int>::iterator sp(int x) { ... }
void query(int p, int l = 0, int r = 25) { // 查询
    if (!p) return;
    if (l == r || !t[p].tot) {
        cnt[l] = t[p].tot; // cnt 记录每个字母出现的次数
        return;
    }
    int mid = (l + r) >> 1;
    query(L(p), l, mid);
    query(R(p), mid + 1, r);
}
int main() {
    n = rd(), m = rd();
    scanf("%s", ss + 1);
    s.insert(n + 1);
    for (int i = 1; i <= n; ++i) {
        update(rt[i], ss[i] - 'a');
        s.insert(i);
```

```
    }
    while (m--) { ... } // m 次排序
    for (auto &x : s) { // 输出结果串
        memset(cnt, 0, sizeof(cnt)); //
        query(rt[x]);
        if (opt[x]) { // 升序
            for (int i = 0; i <= 25; ++i)
                for (int j = 1; j <= cnt[i]; ++j)
                    putchar(i + 'a');
        else // 降序
            for (int i = 25; i >= 0; --i)
                for (int j = 1; j <= cnt[i]; ++j)
                    putchar(i + 'a');
}
```

3.8.8 吉老师线段树

```
/* 给出一个长度为 n 的数列 A, 同时定义一个辅助数组 B, B 开始与 A 完全相同
 * 1 l r k : [l ~ r] add k
 * 2 l r v : a[i] = min(a[i], v) i ∈ [l, r] // 区间最小值操作
 * 3 l r   : ask sum a[l ~ r]
 * 4 l r   : ask max a[l ~ r] // 区间最大值
 * 5 l r   : ask max b[l ~ r] // 历史最大值的区间最大值
 * 在每一次操作后, 都令所有的 b[i] = max(b[i], a[i]) // b[] : 历史最大值
 */
struct Node {
    int ls, rs;
    // 最大值, 次大值, 历史最大值, 最大值个数, 和
    ll mx, se, hx, cnt, sum;
    ll amx, namx, ahx, nahx;
    // mx, 非 mx, 最大 hmx, 非最大 hmx 的懒标记
} t[N << 2];
void pushup(int p) {
    t[p].mx = max(tls(p).mx, trs(p).mx);
    t[p].hx = max(tls(p).hx, trs(p).hx);
    t[p].sum = tls(p).sum + trs(p).sum;
    if (tls(p).mx == trs(p).mx) {
        t[p].cnt = tls(p).cnt + trs(p).cnt;
        t[p].se = max(tls(p).se, trs(p).se);
    } else if (tls(p).mx > trs(p).mx) {
        t[p].cnt = tls(p).cnt;
        t[p].se = max(tls(p).se, trs(p).mx);
    } else {
        t[p].cnt = trs(p).cnt;
        t[p].se = max(trs(p).se, tls(p).mx);
    }
}
void build(int l, int r, int p) {
    L = l, R = r;
    if (l == r) {
        t[p].mx = t[p].hx = t[p].sum = a[l];
        t[p].se = -INF; // 区间次大值初始化
        t[p].cnt = 1;  // 区间最大值个数初始化
        return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, ls(p)), build(mid + 1, r, rs(p)), pushup(p);
}
```

```
// 最大值，最大历史最大值，非最大值，非最大历史最大值 要加的数
void update(ll amx, ll ahx, ll namx, ll nahx, int p) {
    t[p].sum += amx * t[p].cnt + namx * (R - L + 1 - t[p].cnt);
    t[p].hx = max(t[p].hx, t[p].mx + ahx);
    t[p].ahx = max(t[p].ahx, t[p].amx + ahx);
    t[p].nahx = max(t[p].nahx, t[p].namx + nahx);
    // ahx 维护在标记下传前历史上加的最多的一次的加了多少
    // nahx 在下传给非最大值的儿子时变为该儿子的 ahx 标记
    // 后续 nahx 儿子可能变为历史最大值
    /* 先更新上面的式子，才能更新下面三个式子 */
    t[p].mx += amx, t[p].amx += amx, t[p].namx += namx;
    if (t[p].se != -INF) t[p].se += namx;
}
void pushdown(int p) {
    ll mx = max(tls(p).mx, trs(p).mx);
    // 分配懒标记给儿子
    if (tls(p).mx == mx) // 最大值在 ls(p), ls(p) 的 amx 和 ahx 为 amx 和 ahx
        update(t[p].amx, t[p].ahx, t[p].namx, t[p].nahx, ls(p));
    else // 最大值不在 ls(p), ls(p) 的 amx 和 ahx 为 namx 和 nahx
        update(t[p].namx, t[p].nahx, t[p].namx, t[p].nahx, ls(p));
    if (trs(p).mx == mx) // 同上
        update(t[p].amx, t[p].ahx, t[p].namx, t[p].nahx, rs(p));
    else // 同上
        update(t[p].namx, t[p].nahx, t[p].namx, t[p].nahx, rs(p));
    t[p].amx = t[p].namx = t[p].ahx = t[p].nahx = 0;
}
void update_add(int x, int y, int k, int p = 1) { // 区间加
    if (x > R || y < L) return;
    if (x <= L && y >= R) return update(k, k, k, k, p);
    pushdown(p), update_add(x, y, k, ls(p)), update_add(x, y, k, rs(p)), pushup(p);
}
void update_min(int x, int y, int k, int p = 1) { // 区间最值操作
    if (x > R || y < L || k >= t[p].mx) return;
    if (x <= L && y >= R && k > t[p].se) { // t[p].se < k < t[p].mx
        // 把 min(t[p].mx, k) 操作转化成了 t[p].mx -= (t[p].mx - k)
        // 即加法操作 t[p].mx += (k - t[p].mx) => t[p].mx = k;
        // 故 update_add 和 update_min 操作共用懒标记 t[p].amx, t[p].namx
        // 显然 min(t[p].mx, k) 对 历史最大值(hx) 没有任何影响
        // 只有 update_add 操作才 有可能(add 正数时) 改变 历史最大值(hx)
        update(k - t[p].mx, k - t[p].mx, 0, 0, p);
        return;
    }
    pushdown(p), update_min(x, y, k, ls(p)), update_min(x, y, k, rs(p)), pushup(p);
}
// 三个 query (sum, mx, hx) , 与线段树一般 query 写法一致
ll query_sum(int x, int y, int p = 1) {...}
ll query_mx(int x, int y, int p = 1) {...}
ll query_hx(int x, int y, int p = 1) {...}
```

3.8.9 吉老师线段树 (例题)

```
/* Q l r   : 查询区间最大值
 * A l r   : 查询区间历史最大值
 * P l r x : 区间加
 * C l r x : 区间赋值
 */
const ll NINF = -1e16;
struct Node { int l, r; ll mx, hx, amx, ahx, cmx, chx; } t[N << 2];
```

```
int n, m, x, y;
inline void pushup(int p) {
    t[p].mx = max(tls(p).mx, trs(p).mx);
    t[p].hx = max(tls(p).hx, trs(p).hx);
}
void build(int l, int r, int p) {
    t[p].cmx = t[p].chx = NINF;
    L = l, R = r;
    if (l == r) { t[p].mx = t[p].hx = rd(); return; }
    int mid = (l + r) >> 1;
    build(l, mid, ls(p));
    build(mid + 1, r, rs(p));
    pushup(p);
}
inline void pusha(int p, int m, int h) {
    t[p].hx = max(t[p].hx, t[p].mx + h);
    t[p].mx += m;
    if (t[p].cmx ^ NINF) {
        t[p].chx = max(t[p].chx, t[p].cmx + h);
        t[p].cmx += m;
    } else {
        t[p].ahx = max(t[p].ahx, t[p].amx + h);
        t[p].amx += m;
    }
}
inline void pushc(int p, int m, int h) {
    t[p].hx = max(t[p].hx, h);
    t[p].chx = max(t[p].chx, h);
    t[p].mx = t[p].cmx = m;
}
inline void pushdown(int p) {
    if (t[p].amx) {
        pusha(ls(p), t[p].amx, t[p].ahx);
        pusha(rs(p), t[p].amx, t[p].ahx);
        t[p].amx = t[p].ahx = 0;
    }
    if (t[p].cmx ^ NINF) {
        pushc(ls(p), t[p].cmx, t[p].chx);
        pushc(rs(p), t[p].cmx, t[p].chx);
        t[p].cmx = t[p].chx = NINF;
    }
}
void add(int x, int y, int k, int p = 1) {
    if (x > R || y < L) return;
    if (x <= L && y >= R) return pusha(p, k, max(k, 0));
    pushdown(p);
    if (x <= tls(p).r) add(x, y, k, ls(p));
    if (y > tls(p).r) add(x, y, k, rs(p));
    pushup(p);
}
void chg(int x, int y, int k, int p = 1) {
    if (x > R || y < L) return;
    if (x <= L && y >= R) { pushc(p, k, k); return; }
    pushdown(p);
    if (x <= tls(p).r) chg(x, y, k, ls(p));
    if (y > tls(p).r) chg(x, y, k, rs(p));
    pushup(p);
}
ll qmx(int x, int y, int p = 1); // 区间最值
```

```
ll qhx(int x, int y, int p = 1); // 区间历史最值
int main() {
    n = rd(), build(1, n, 1), m = rd();
    while (m--) {
        scanf("%s", s);
        x = rd(), y = rd();
        switch (*s) {
            case 'Q': printf("%lld\n", qmx(x, y)); break;
            case 'A': printf("%lld\n", qhx(x, y)); break;
            case 'P': add(x, y, rd()); break;
            case 'C': chg(x, y, rd());
        }
    }
    return 0;
}
```

3.8.10 李超线段树

```
// 可强制在线
// 插入 O(logn logn), 查询 O(logn)
// 两个操作 1: 插入线段, 2: 查询与 x = k 相交的线段的最大纵坐标

/* 最优线段定义:
 * 1. 该线段的定义域<完整覆盖>了区间
 * 2. 该线段在<区间中点处><最优>
 */
typedef pair<db, int> pdi; // y 值, 线段编号
struct Line { db k, b; } p[N]; // 线段
int t[N << 2]; // SEG: 存区间最优线段的编号
int n, x1, y1, x2, y2, opt, cnt, lans;
il db cal(int id, int x) { // 计算 y = kx + b
    return p[id].k * x + p[id].b;
}
il void add(int x1, int y1, int x2, int y2) { // 通过两点坐标存线段
    ++cnt; // 线段编号
    if (x1 == x2) { // 斜率不存在
        p[cnt].k = 0;
        p[cnt].b = max(y1, y2);
    } else {
        p[cnt].k = (db)(y1 - y2) / (x1 - x2);
        p[cnt].b = y1 - p[cnt].k * x1;
    }
}
// 插入线段, [x, y] 是线段在 x 轴上的左右端点; 如果是插入直线的话就只需要写全部插入的部分
void update(int x, int y, int u, int l = 1, int r = MX, int rt = 1) { // O(logn logn)
    int mid = (l + r) >> 1;
    if (x <= l && y >= r) { // 线段在这个区间里, 全部插入
        int v = t[rt]; // 原来的最优线段编号
        db yu = cal(u, mid); // 新线段中点处 y 值
        db yv = cal(v, mid); // 旧线段中点处 y 值
        if (l == r) {
            if (yu > yv) // 直接比较 y 轴
                t[rt] = u;
            return;
        }
        if (p[u].k > p[v].k) { // 当前斜率大于原最优线段斜率
            if (yu > yv) { // 比较中点处 y 值
                t[rt] = u;
```

```
                update(x, y, v, l, mid, ls(rt));
            } else {
                update(x, y, u, mid + 1, r, rs(rt));
            }
        } else if (p[u].k < p[v].k) { // 当前斜率小于原最优线段斜率
            if (yu > yv) {
                t[rt] = u;
                update(x, y, v, mid + 1, r, rs(rt));
            } else {
                update(x, y, u, l, mid, ls(rt));
            }
        } else if (p[u].b > p[v].b) // 斜率相等直接比较截距
                t[rt] = u;
        return; //
    }
    if (x <= mid) { // 部分插入
        update(x, y, u, l, mid, ls(rt));
    }
    if (y > mid) { // 部分插入
        update(x, y, u, mid + 1, r, rs(rt));
    }
}
il pdi pmax(pdi x, pdi y) {
    if (x.fi == y.fi) // y 值相同时优先选编号小的(本题)
        return x.se > y.se ? y : x;
    return x.fi < y.fi ? y : x;
}
// 将所有包含 x 区间的最优线段拿出来，在这些线段中比较，从而得出最优线段
pdi query(int x, int l = 1, int r = MX, int rt = 1) { // O(logn)
    double ans = cal(t[rt], x);
    if (l == r)
        return pdi(ans, t[rt]);
    pdi ans;
    int mid = (l + r) >> 1;
    if (x <= mid) {
        return pmax(pdi(ans, t[rt]), query(x, l, mid, ls(rt)));
    }
    return pmx(pdi(ans, t[rt]), query(x, mid + 1, r, rs(rt)));
}
main {
    if (opt) { // 插入线段
        if (x1 > x2) swap(x1, x2), swap(y1, y2);
        add(x1, y1, x2, y2);
        update(x1, x2, cnt);
    } else { // 查询与 x = k 相交的线段的最大纵坐标
        lans = query(k).se;
    }
}
```

3.8.11 扫描线-面积并

```
int v[N << 1]; ll ans;
int n, x1, y1, x2, y2;
struct Line { int l, r, h, d; } a[N << 1];
struct Tree { ll len; int l, r, sum; } t[N << 4]; // 空间开到 3 会 RE
void build(int l, int r, int p) {
    L = v[l], R = v[r];
    if (r - l < 2) return;
```

```
    int mid = (l + r) >> 1;
    build(l, mid, ls(p));
    build(mid, r, rs(p));
}
inline void pushup(int p) {
    if (t[p].sum) t[p].len = R - L;
    else t[p].len = tls(p).len + trs(p).len;
}
void update(int x, int y, int k, int p = 1) {
    if (x <= L && y >= R) {
        t[p].sum += k;
        pushup(p);
        return;
    }
    // 下面这样写比较快
    if (x < tls(p).r) update(x, y, k, ls(p));
    if (y > tls(p).r) update(x, y, k, rs(p));
    pushup(p);
}
int main() {
    n = rd();
    for (int i = 1; i <= n; ++i) {
        x1 = rd(), y1 = rd(), x2 = rd(), y2 = rd();
        v[i] = x1, v[i + n] = x2;
        a[i] = (Line) {x1, x2, y1, 1};
        a[i + n] = (Line) {x1, x2, y2, -1};
    }
    n <<= 1; //
    sort(v + 1, v + 1 + n);
    sort(a + 1, a + 1 + n, [&] (Line a, Line b) { return a.h < b.h; });
    build(1, n, 1);
    for (int i = 1; i < n; ++i) {
        update(a[i].l, a[i].r, a[i].d);
        ans += t[1].len * (a[i + 1].h - a[i].h);
    }
    printf("%lld\n", ans);
    return 0;
}
```

3.8.12 扫描线-周长并

```
int val[N << 1];
int n, x1, y1, x2, y2, pre, res;

struct Line {
    int l, r, h, f;
    bool operator < (const Line &b) const { return h ^ b.h ? h < b.h : f > b.f; }
    // 如果出现了两条高度相同的扫描线，也就是两矩形相邻
    // 那么需要先扫底边再扫顶边，否则就会多算这条边
    // 这个对面积并无影响但对周长并有影响
} line[N];

struct Seg {
    int l, r, sum, len, tot; // tot : 线段条数
    bool lc, rc; // 左右区间是否完全覆盖
} t[N << 2];

void build(int l, int r, int p) {
```

```
18      L = val[l], R = val[r];
19      if (r - l < 2) return;
20      int mid = (l + r) >> 1;
21      build(l, mid, ls(p));
22      build(mid, r, rs(p));
23  }
24
25  inline void pushup(int p) {
26      int ls = ls(p), rs = rs(p);
27      if (t[p].sum) {
28          t[p].len = R - L;
29          t[p].tot = t[p].lc = t[p].rc = 1;
30      } else {
31          t[p].len = t[ls].len + t[rs].len;
32          t[p].tot = t[ls].tot + t[rs].tot;
33          t[p].lc = t[ls].lc, t[p].rc = t[rs].rc;
34          // 如果左儿子左端点被覆盖, 那么自己的左端点也肯定被覆盖; 右儿子同理
35          if (t[ls].rc && t[rs].lc) --t[p].tot;
36          // 如果左儿子右端点和右儿子左端点都被覆盖,
37          // 那么中间就是连续的一段, 所以要 -= 1
38      }
39  }
40
41  void update(int x, int y, int k, int p = 1) {
42      if (x <= L && y >= R) {
43          t[p].sum += k;
44          pushup(p);
45          return;
46      }
47      if (x < t[ls(p)].r) update(x, y, k, ls(p));
48      if (y > t[ls(p)].r) update(x, y, k, rs(p));
49      pushup(p);
50  }
51
52  int main() {
53      n = rd();
54      for (int i = 1; i <= n; ++i) {
55          x1 = rd(), y1 = rd(), x2 = rd(), y2 = rd();
56          val[i] = x1, val[i + n] = x2;
57          line[i] = (Line) {x1, x2, y1, 1};
58          line[i + n] = (Line) {x1, x2, y2, -1};
59      }
60      n <<= 1;
61      sort(val + 1, val + 1 + n);
62      sort(line + 1, line + 1 + n);
63      build(1, n, 1);
64      for (int i = 1; i < n; ++i) {
65          update(line[i].l, line[i].r, line[i].f);
66          res += abs(pre - t[1].len); // 横边
67          pre = t[1].len;
68          res += 2 * t[1].tot * (line[i + 1].h - line[i].h); // 纵边
69      }
70      res += line[n].r - line[n].l;
71      //  特判一下枚举不到的最后一条扫描线
72      printf("%d\n", res);
73      return 0;
74  }
```

3.9 树状数组

3.9.1 单点更新区间查询

```
int lowbit(int &x) { return x & -x; }
void add(int i, int k) { // 点 i 加上 x
    for (; i <= n; i += i & -i) c[i] += k;
}
int ask(int i) { // [l, r] 区间和 => ask(r) - ask(l - 1)
    int ans = 0;
    for (; i; i ^= i & -i) ans += c[i];
    return ans;
}
```

3.9.2 区间更新单点查询

```
void add(int i, int k) { for (; i <= n; i += i & -i) c[i] += k; }
int ask(int i) { // 返回 c[1] ~ c[i] 的和(即得 a[i] 的值)
    int ans = 0;
    for (; i; i ^= i & -i) ans += c[i];
    return ans;
}
{
    for (int i = 1; i <= n; ++i) add(i, a[i] - a[i - 1]); // 差分数组更新
    if (t & 1) { // [x, y] 的数加上 k
        add(x, k); // a[x] - a[x - 1] 增加了 k
        add(y + 1, -k); // a[y + 1] - a[y] 减少了 k
    } else { // 查询 x 的值
        printf("%d\n", ask(x));
    }
}
```

3.9.3 区间更新区间查询

```
/*
 设
    d[n]       =  a[n] - a[n - 1]  □
    sd[n]      =  (n - 1) * d[n]   □
    sum(a[n])  =  a[1] + a[2] + ... + a[n]
    sum(d[n])  =  d[1] + d[2] + ... + d[n] = a[n]
    sum(sd[n]) =  sd[1] + sd[2] + ... + sd[n]
 */ /*
 则 sum(a[n])
    =  a[1] + a[2] + ... + a[n]
    =  sum(d[1]) + sum(d[2]) + ... + sum(d[n])
    =  d[1] + (d[1] + d[2]) + ... + (d[1] + d[2] + ... + d[n])
    =  n * d[1] + (n - 1) * d[2] + ... + d[n]
    =  n * (d[1] + d[2] + ... + d[n]) -
       (0 * d[1] + 1 * d[2] + ... + (n - 1) * d[n])
    =  n * sum(d[n]) - sum(sd[n])
*/
// 综上, 需要维护 d[n], sd[n] 这两个数组
// sum(a[n]) = n * sum(d[n]) - sum(sd[n])  □
ll a[N], d[N], sd[N], pd[N], psd[N]; // 前缀和, 空间换时间
void build() { // O(n) 建树
    for (int i = 1; i <= n; ++i) {
        d[i] = pd[i] - pd[i - lowbit(i)];
        sd[i] = psd[i] - psd[i - lowbit(i)];
```

```
    }
}
void add(int i, int k) { // O(logn)
    int p = i - 1;
    for (; i <= n; i += i & -i) d[i] += k, sd[i] += k * p;
}
ll ask(int i) {
    ll p = i, s1 = 0, s2 = 0;
    for (; i; i ^= i & -i) s1 += d[i], s2 += sd[i];
    return p * s1 - s2;
}
{
    for (int i = 1; i <= n; ++i) {
        ll q = a[i] - a[i - 1];
        pd[i] = pd[i - 1] + q; // d[i] 前缀和
        psd[i] = psd[i - 1] + q * (i - 1); // sd[i] 前缀和
        // add(i, a[i] - a[i - 1]);
    }
    build();
    if (t & 1) { // [x, y] add k
        scanf("%d", &k);
        add(x, k), add(y + 1, -k);
    } else { // ask [x, y]
        printf("%lld\n", ask(y) - ask(x - 1));
    }
}
```

3.9.4 逆序对

```
// 权值树状数组 求逆序对
ll c[N];
ll n, x, y, ans;
struct Node {
    ll val; int num;
    bool operator < (const Node &b) const { return val > b.val; }
} t[N];
void add(int i) {
    for (; i <= n; i += i & -i) ++c[i];
}
ll ask(int i) {
    int ans = 0;
    for (; i; i ^= i & -i) ans += c[i];
    return ans;
}
int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) {
        scanf("%lld", &t[i].val);
        t[i].num = i;
    }
    sort(t + 1, t + 1 + n); // 结构体离散化
    for (int i = 1; i <= n; ++i) {
        add(t[i].num); // 晚加入的下标 t[i].num 指向的数字 t[i].val 小于早加入的
        x = ask(t[i].num - 1); // t[i].val 左边比 t[i].val 大的数字个数
        y = i - x - 1; // t[i].val 右边比 t[i].val 大的数字个数
        ans += x * y;
    }
    printf("%lld ", ans); // V 的个数, 减增减的数对个数
```

```
    ans = 0;
    memset(c, 0, sizeof(c));
    for (int i = n; i >= 1; --i) {
        add(t[i].num); // 晚加入的下标 t[i].num 指向的数字 t[i].val 大于早加入的
        x = ask(t[i].num - 1); // t[i].val 左边比 t[i].val 小的数字个数
        y = n - i - x; // t[i].val 右边比 t[i].val 小的数字个数
        // y = (n - i + 1) - x - 1
        ans += x * y;
    }
    printf("%lld\n", ans); // ∧的个数, 增减增的数对个数
    return 0;
}
```

3.9.5 线性复杂度建树

```
void build1(){ // 方式一
    for (int i = 1; i <= n; ++i) pre[i] = pre[i - 1] + a[i]; // 前缀和
    for (int i = 1; i <= n; ++i) c[i] = pre[i] - pre[i - lowbit(i)];
}
void build2() { // 方式二, 推荐
    for (int i = 1; i <= n; ++i) {
        c[i] += a[i];
        int j = i + lowbit(i);
        if (j <= n) c[j] += c[i];
    }
}
```

3.9.6 维护后缀信息

```
// 将 add 与 ask 的 for 循环遍历方式交换一下就行
void add(int i, int k) { for (; i; i ^= i & -i) c[i] += k; }
int ask(int i) {
    int ans = 0;
    for (; i <= n; i += i & -i) ans += c[i];
    return ans;
}
```

3.9.7 权值树状数组【可求全局第 k 小】

```
int mx, mn; // 值域上下限
void build() { // 建树
    for (int i = 1; i <= n; ++i) ++c[a[i]];
}
void presum() { // 前缀和
    for (int i = mn + 1; i <= mx; ++i) pre[i] = pre[i - 1] + a[i];
    // pre[i] 表示原数组中小于等于 i 的元素个数
}
// 基本操作
inline int lowbit(int x) { return x & -x; }
inline int getsum(int i) { // 小于等于 i 的元素的数目
    int ans = 0;
    for (; i; i -= lowbit(i)) ans += c[i];
    return ans;
}
inline void update(int i, int k) { // 将值为 i 的元素增加 k 个
    for (; i <= mx; i += lowbit(i)) c[i] += k;
}
```

```cpp
// 求全局第 k 小, 注意下标应从 1 开始
int kth(int k) {
    int tot = 0, ans = 0;
    for (int i = log2(n); ~i; --i) {
        ans ^= 1 << i;          // 尝试扩展
        if (ans >= n || tot + c[ans] >= k) ans ^= 1 << i; // 如果扩展失败
        else tot += c[ans];  // 扩展成功后 要更新之前求和的值
    }
    return ans + 1;
}
```

3.10 主席树

解决静态区间第 k 小问题, 视情况进行离散化空间: O(nlogn), 时间: O(nlogn)

```cpp
#define L(x) t[x].l
#define R(x) t[x].r
int n, m, cnt;
int rt[N], a[N], b[N]; // rt[i] : (a[1] ~ a[i]) 权值线段树的根节点编号
// l, r : 左, 右节点编号, tot : 该区间内数字个数
struct Node { int l, r, tot; } t[N * 20]; // O(nlogn)
void update(int &now, int pre, int x, int l = 1, int r = n) { // &now
    t[now = ++cnt] = t[pre], ++t[cnt].tot; // 可持续化核心句
    if (l == r) return;
    int mid = (l + r) >> 1;
    if (x <= mid) update(L(now), L(pre), x, l, mid);
    else update(R(now), R(pre), x, mid + 1, r);
}
// 另一种写法, 先在 update1 前 rt[now] = rt[pre]
void update1(int &p, int x, int l = 1, int r = n) {
    t[++cnt] = t[p], ++t[p = cnt].tot;
    if (l == r) return;
    int mid = (l + r) >> 1;
    if (x <= mid) update(L(p), x, l, mid);
    else update(R(p), x, mid + 1, r);
}
int query(int tl, int tr, int k, int l = 1, int r = n) {
    if (l == r) return l;
    // 差分思想 : 利用t[r] - t[l - 1] 得到区间 [l, r] 的权值线段树
    int tot = t[L(tr)].tot - t[L(tl)].tot; // 左子树区间内的数字个数
    int mid = (l + r) >> 1;
    if (k <= tot) return query(L(tl), L(tr), k, l, mid);
    return query(R(tl), R(tr), k - tot, mid + 1, r);
}
int main() {
    n = rd(), m = rd();
    for (int i = 1; i <= n; ++i) a[i] = b[i] = rd();
    sort(b + 1, b + 1 + n);
    int len = unique(b + 1, b + 1 + n) - b - 1;
    for (int i = 1; i <= n; ++i) {
        int id = lower_bound(b + 1, b + 1 + len, a[i]) - b;
        // update(rt[i], rt[i - 1], id);
        rt[i] = rt[i - 1], update1(rt[i], id);
    }
    while (m--) {
        int l = rd(), r = rd(), k = rd();
        printf("%d\n", b[query(rt[l - 1], rt[r], k)]);
    }
```

```
    return 0;
}
```

3.10.1 求区间内不同数的个数

```
// 在线算法
#define N 200005
struct Node { int l, r, tot; } t[N << 5];
int rt[N], p[N], pre[N], cnt;
void build(int &p, int l, int r) {
    t[p = ++cnt].tot = 0;
    if (l == r) return;
    int mid = (l + r) >> 1;
    build(L(p), l, mid);
    build(R(p), mid + 1, r);
}
void update(int &p, int x, int k, int l = 1, int r = n) {
    t[++cnt] = t[p];
    t[p = cnt].tot += k;
    if (l == r) return;
    if (x <= mid) update(L(p), x, k, l, mid);
    else update(R(p), x, k, mid + 1, r);
}
int query(int p, int x, int l = 1, int r = n) {
    if (l == r) return t[p].tot;
    int mid = (l + r) >> 1;
    if (x <= mid) return query(L(p), x, l, mid) + t[R(p)].tot;
    return query(R(p), x, mid + 1, r);
}
void init() {
    // build(*rt, 1, n);
    fill(pre, 0, (n + 1) << 2);
}
void UPDATE() {
    for (int i = 1; i <= n; ++i) {
        if (p[x = rd()]) pre[i] = p[x];
        p[x] = i;
    }
    for (int i = 1; i <= n; ++i) {
        rt[i] = rt[i - 1], update(rt[i], i, 1);
        // 如果 a[i] 已出现过
        if (pre[i]) update(rt[i], pre[i], -1);
    }
}
void QUERY() {
    l = rd(), r = rd();
    printf("%d\n", query(rt[r], l)); // 查询区间 [l, r] 内不同数的个数
}
{
    init(), UPDATE(), QUERY();
}
```

3.10.2 区间修改懒标记

```
/*
   C l r d: 区间 [l, r] 中的数都加 d , 同时当前的时间戳加 1
   Q l r: 查询当前时间戳区间 [l, r] 中所有数的和
   H l r t: 查询时间戳 t 区间 [l, r] 的和
```

```
5       B t: 将当前时间戳置为 t
6    */
7   struct Node { int l, r; ll sum, tag; } t[N * 40]; // 懒标记 tag
8   char s[2];
9   int a[N], rt[N];
10  int n, m, cnt, tim;
11  inline void pushup(int p) { t[p].sum = t[L(p)].sum + t[R(p)].sum; }
12  void build(int &p, int l, int r) {
13      p = ++cnt;
14      if (l == r) { t[p] = (Node) {0, 0, a[l], 0}; return; }
15      int mid = (l + r) >> 1;
16      build(L, l, mid);
17      build(R, mid + 1, r);
18      pushup(p);
19  }
20  void update(int &p, int x, int y, ll k, int l = 1, int r = n) {
21      t[++cnt] = t[p];
22      t[p = cnt].sum += k *  (min(y, r) - max(x, l) + 1); // 加区间交集
23      if (x <= l && y >= r) {
24          t[p].tag += k; //
25          return;
26      }
27      int mid = (l + r) >> 1;
28      if (x <= mid) update(L(p), x, y, k, l, mid);
29      if (y >  mid) update(R(p), x, y, k, mid + 1, r);
30  }
31  ll query(int p, int x, int y, ll tag = 0, int l = 1, int r = n) {
32      if (x <= l && y >= r) return t[p].sum + tag * (r - l + 1);
33      ll ans = 0; int mid = (l + r) >> 1;
34      if (x <= mid) ans = query(L(p), x, y, tag + t[p].tag, l, mid); // 带着懒标记查询
35      if (y >  mid) ans += query(R(p), x, y, tag + t[p].tag, mid + 1, r);
36      return ans;
37  }
38  int main() {
39      n = rd(), m = rd();
40      for (int i = 1; i <= n; ++i) a[i] = rd();
41      build(*rt, 1, n); // build
42      int l, r, d, t;
43      while (m--) {
44          scanf("%s", s);
45          switch (*s) {
46              case 'C':
47                  l = rd(), r = rd(), d = rd();
48                  ++tim, rt[tim] = rt[tim - 1];
49                  update(rt[tim], l, r, d);
50                  break;
51              case 'Q':
52                  l = rd(), r = rd();
53                  printf("%lld\n", query(rt[tim], l, r));
54                  break;
55              case 'H':
56                  l = rd(), r = rd();
57                  printf("%lld\n", query(rt[rd()], l, r));
58                  break;
59              case 'B': tim = rd();
60          }
61      }
62      return 0;
63  }
```

3.10.3 经典例题 middle

```cpp
/*
    给你一个长度为 n 的序列 s
    回答 Q 个这样的询问: s 的左端点在 [a, b] 之间, 右端点在 [c, d] 之间的子区间中, 最大的中位数
 */
/*
    二分一个答案 mid, 把所有 >= mid 的数值设成 1 , < mid 的值设为 -1
    查询区间内的和是否 >= 0 (本题偶数项的中位数取靠后的那个)
    是, 中位数应该更大, 否则, 中位数只能更小
    ( 要让中位数尽量大, 所以应该取尽可能多的 1 )
*/
struct Node { int l, r, sum, lmx, rmx; } t[N * 40];

vector<int> pos[N];
int a[N], b[N], rt[N];
int n, m, cnt, len, lans;

inline void pushup(int p) {
    t[p].sum = tls(p).sum + trs(p).sum;
    t[p].lmx = max(tls(p).lmx, tls(p).sum + trs(p).lmx);
    t[p].rmx = max(trs(p).rmx, trs(p).sum + tls(p).rmx);
}

void build(int &p, int l, int r) { // 初始化全为 1
    p = ++cnt;
    if (l == r) {
        t[p] = (Node) {0, 0, 1, 1, 1}; // 1
        return;
    }
    int mid = (l + r) >> 1;
    build(L(p), l, mid);
    build(R(p), mid + 1, r);
    pushup(p);
}

void update(int &p, int x, int l = 1, int r = n) { // 把 x 位置设置为 -1
    t[++cnt] = t[p], p = cnt;
    if (l == r) {
        t[p] = (Node) {0, 0, -1, -1, -1}; // -1
        return;
    }
    int mid = (l + r) >> 1;
    if (x <= mid) {
        update(L(p), x, l, mid);
    } else {
        update(R(p), x, mid + 1, r);
    }
    pushup(p);
}

int qsum(int p, int x, int y, int l = 1, int r = n) { // 和
    if (x > y) return 0;
    if (x <= l && y >= r) return t[p].sum;
    int ans = 0;
    int mid = (l + r) >> 1;
    if (x <= mid) ans = qsum(L(p), x, y, l, mid);
    if (y >  mid) ans += qsum(R(p), x, y, mid + 1, r);
    return ans;
```

```
}

Node qlmx(int p, int x, int y, int l = 1, int r = n) { // 最大前缀
    if (x <= l && y >= r) return t[p];
    Node tt, tl, tr;
    int mid = (l + r) >> 1;
    if (x <= mid) tt = qlmx(L(p), x, y, l, mid);
    if (y > mid) {
        if (x <= mid) {
            tl = tt;
            tr = qlmx(R(p), x, y, mid + 1, r);
            tt.sum = tl.sum + tr.sum;
            tt.lmx = max(tl.lmx, tl.sum + tr.lmx);
        } else {
            tt = qlmx(R(p), x, y, mid + 1, r);
        }
    }
    return tt;
}

Node qrmx(int p, int x, int y, int l = 1, int r = n) { // 最大后缀
    if (x <= l && y >= r) return t[p];
    Node tt, tl, tr;
    int mid = (l + r) >> 1;
    if (x <= mid) tt = qrmx(L(p), x, y, l, mid);
    if (y > mid) {
        if (x <= mid) {
            tl = tt;
            tr = qrmx(R(p), x, y, mid + 1, r);
            tt.sum = tl.sum + tr.sum;
            tt.rmx = max(tr.rmx, tr.sum + tl.rmx);
        } else {
            tt = qrmx(R(p), x, y, mid + 1, r);
        }
    }
    return tt;
}

int check(int mid) {
    int x = qrmx(rt[mid], A, B).rmx;
    int z = qlmx(rt[mid], C, D).lmx;
    int y = qsum(rt[mid], B + 1, C - 1);
    return x + y + z >= 0;
}

int main() {
    n = rd();
    for (int i = 1; i <= n; ++i) a[i] = b[i] = rd();
    sort(b + 1, b + 1 + n);
    len = unique(b + 1, b + 1 + n) - b - 1;
    for (int i = 1; i <= n; ++i) {
        a[i] = lower_bound(b + 1, b + 1 + n, a[i]) - b;
        pos[a[i]].push_back(i);
    }
    build(rt[1], 1, n); // 先全部建为 1
    for (int i = 2; i <= len; ++i) {
        rt[i] = rt[i - 1]; // rt[]: 权值, 主席树区间: 位置
        for (auto &x : pos[i - 1]) { // 把 rt[i] 中 < i 的位置 x 修改为 -1
            update(rt[i], x);
```

```
        }
    }
    m = rd();
    while (m--) {
        A = rd(), B = rd(), C = rd(), D = rd(); // 题目要求强制在线，此处略去
        int l = 1, r = len, ans;
        while (l <= r) { // 对中位数进行二分
            int mid = (l + r) >> 1;
            if (check(mid)) { // 检查该中位数合法性
                ans = mid;
                l = mid + 1;
            } else {
                r = mid - 1;
            }
        }
        printf("%d\n", lans = b[ans]);
    }
    return 0;
}
```

3.11 平衡树

树套树可用常数较小的 Treap LCT 必须用 Splay (与普通的 Splay 写法上有区别) 可持续化只能用 FHQ Treap

3.11.1 Splay

```
#define ls(x) t[x].ch[0]
#define rs(x) t[x].ch[1]
#define tls(x) t[ls(x)]
#define trs(x) t[rs(x)]
struct Splay {
    int ch[2];
    int fa, siz, tot;
    int mx, val, tag, rev;
} t[N];
int n, m, rt, cnt, num, a[N];
iv init() {
    t[0].siz = t[0].tot = t[0].tag = t[0].rev = 0;
    ls(0) = rs(0) = t[0].fa = t[0].val = 0;
    t[0].mx = -INF; //
}
ii id(int x) { return x == rs(t[x].fa); }
ii New(int v, int fa) {
    int x = ++cnt;
    t[x].fa = fa;
    t[x].val = t[x].mx = v;
    t[x].siz = t[x].tot = 1;
    // ls(x) = rs(x) = t[x].tag = t[x].rev = 0;
    return x;
}
iv destroy(int x) { ls(x) = rs(x) = t[x].fa = t[x].siz = t[x].tot = t[x].val = 0; }
iv pushup(int x) {
    t[x].siz = tls(x).siz + trs(x).siz + t[x].tot;
    t[x].mx = max(t[x].val, max(tls(x).mx, trs(x).mx)); // 注意初始化 t[0].mx = -INF
}
iv add(int x, int k) { t[x].mx += k, t[x].val += k, t[x].tag += k; }
```

```
iv rev(int x) { swap(ls(x), rs(x)), t[x].rev ^= 1; }
iv pushdown(int x) {
    if (t[x].tag) {
        if (ls(x)) add(ls(x), t[x].tag);
        if (rs(x)) add(rs(x), t[x].tag);
        t[x].tag = 0;
    }
    if (t[x].rev) {
        rev(ls(x));
        rev(rs(x));
        t[x].rev = 0;
    }
}
iv rotate(int x) {
    int f = t[x].fa, ff = t[f].fa;
    pushdown(x), pushdown(f); // pushdown
    int d = id(x), son = t[x].ch[d ^ 1];
    if (ff) t[ff].ch[id(f)] = x;
    t[x].ch[d ^ 1] = f, t[f].ch[d] = son;
    t[son].fa = f, t[f].fa = x, t[x].fa = ff;
    pushup(f) // pushup
}
iv splay(int x, int to) {
    while (t[x].fa ^ to) {
        int f = t[x].fa;
        if (t[f].fa ^ to) {
            rotate(id(f) ^ id(x) ? x : f);
        }
        rotate(x);
    }
    pushup(x);
    if (!to) rt = x;
}

// 直接用 a[] 数组建树
int build(int l, int r, int fa) {
    if (l > r) return 0;
    int mid = (l + r) >> 1;
    int x = ++cnt;
    t[x].fa = fa;
    t[x].val = t[x].mx = a[mid];
    ls(x) = build(l, mid - 1, x);
    rs(x) = build(mid + 1, r, x);
    pushup(x);
    return x;
}

// 找到第 k 个数在树中的编号
ii find(int k) {
    int x = rt;
    while (1) {
        pushdown(x); //
        if (x <= tls(x).siz) x = ls(x);
        else if (x <= tls(x).siz + t[x].tot) break;
        else {
            k -= tls(x).siz + t[x].tot;
            x = rs(x);
        }
    }
```

```
90      if (x) splay(x, 0);
91      return x;
92  }
93
94  // 找到权值为 v 的数在树中的编号
95  ii find1(int v) {
96      int x = rt;
97      while (x && v != t[x].val) {
98          pushdown(x); //
99          x = v < t[x].val ? ls(x) : rs(x);
100     }
101     if (x) splay(x, 0);
102     return x;
103 }
104
105 // 插入权值为 v 的数
106 void insert(int v) {
107     ++num;
108     if (!rt) { rt = New(v, 0); return; }
109     int x = rt;
110     while (1) {
111         pushdown(x); //
112         ++t[x].siz;
113         if (t[x].val == v) {
114             ++t[x].tot;
115             splay(x, 0);
116             return;
117         }
118         int d = v <= t[x].val ? 0 : 1;
119         if (!t[x].ch[d]) {
120             t[x].ch[d] = New(v, x);
121             splay(t[x].ch[d], 0);
122             return;
123         }
124         x = t[x].ch[d];
125     }
126 }
127
128 // 删除权值为 v 的数
129 iv del(int v) { // 几个 pushdown 操作自己脑补的，暂时不知道是否完全正确
130     int x = find1(v); // find(v) 附带了 splay 操作
131     if (!x) return;
132     --num;
133     if (t[x].tot > 1) {
134         --t[x].tot;
135         --t[x].siz;
136         return;
137     }
138     pushdown(x); //
139     if (!ls(x)) {
140         rt = rs(x);
141         t[rt].fa = 0;
142     } else {
143         int L = ls(x), R = rs(x);
144         pushdown(L); //
145         while (rs(L)) {
146             L = rs(L);
147             pushdown(L); //
148         }
```

```
149         splay(L, 0);
150         rs(0) = L, rs(L) = R;
151         t[R].fa = L, t[L].fa = 0;
152         pushup(L);
153     }
154     destroy(x);
155 }
156 
157 // 获得权值为 v 的数的 <排名 + 1>
158 ii get_rk(int v) {
159     int x = rt, k = 0;
160     while (x) {
161         pushdown(x); //
162         if (v == t[x].val) {
163             k += tls(x).siz + 1;
164             break;
165         }
166         if (v < t[x].val) x = ls(x);
167         else {
168             k += tls(x).siz + t[x].tot;
169             x = rs(x);
170         }
171     }
172     if (x) splay(x, 0);
173     return k;
174 }
175 
176 // 获得排名第 k 的数的权值
177 ii kth(int k) { // 查询排名第 k 为 kth(k + 1)
178     if (k > num) return -1;
179     int x = rt;
180     while (1) {
181         pushdown(x); //
182         if (k <= tls(x).siz) x = ls(x);
183         else if (k <= tls(x).siz + t[x].tot) break;
184         else {
185             k -= tls(x).siz + t[x].tot;
186             x = rs(x);
187         }
188     }
189     if (x) splay(x, 0); //
190     return t[x].val;
191 }
192 
193 // 获取翻转的区间
194 ii split(int l, int r) {
195     l = find(l), r = find(r + 2);
196     splay(l, 0), splay(r, l);
197     return ls(rs(rt));
198 }
199 
200 main() {
201     t[0].mx = a[1] = a[n + 2] = -INF; // t[0].mx 也需置为 -INF !
202     rt = build(1, n + 2, 0); // 注意是 n + 2, 因为多了两个哨兵节点
203     { // OPT
204         int pos = split(l, r); // 获取翻转的区间
205         switch (opt) {
206             case 1: add(pos, v); break; // 区间加
207             case 2: rev(pos); break; // 区间翻转
```

```
208             case 3: printf("%d\n", t[pos].mx); // 查区间最大值
209         }
210     }
211 }
```

3.11.2 Treap

```
1  int n, x, cnt, rt;
2  struct Treap { int ch[2], val, dat, siz, tot; } t[N];
3  iv pushup(int x) { t[x].siz = tls(x).siz + trs(x).siz + t[x].tot; }
4  ii New(int v) {
5      t[++cnt].val = v;
6      t[cnt].dat = rand(); // 随机权值，父大子小
7      t[cnt].siz = t[cnt].tot = 1;
8      return cnt;
9  }
10 iv rotate(int &x, int d) {
11     int p = t[x].ch[d ^ 1];
12     t[x].ch[d ^ 1] = t[p].ch[d];
13     t[p].ch[d] = x, x = p;
14     pushup(t[x].ch[d]);
15     pushup(x);
16 }
17 void insert(int &x, int v) {
18     if (!x) { x = New(v); return; }
19     if (t[x].val == v) {
20         ++t[x].tot;
21         ++t[x].siz;
22         return;
23     }
24     int d = v < t[x].val ? 0 : 1;
25     insert(t[x].ch[d], v);
26     if (t[t[x].ch[d]].dat > t[x].dat) rotate(x, d ^ 1);
27     pushup(x);
28 }
29 void del(int &x, int v) {
30     if (!x) return;
31     if (t[x].val == v) {
32         if (t[x].tot > 1) {
33             --t[x].tot;
34             --t[x].siz;
35             return;
36         }
37         if (t[x].ch[0] || t[x].ch[1]) {
38             if (!t[x].ch[1] || t[t[x].ch[0]].dat > t[t[x].ch[1]].dat) {
39                 rotate(x, 1);
40                 del(t[x].ch[1], v);
41             } else {
42                 rotate(x, 0);
43                 del(t[x].ch[0], v);
44             }
45             pushup(x);
46         } else x = 0;
47         return;
48     }
49     int d = v < t[x].val ? 0 : 1;
50     del(t[x].ch[d], v);
51     pushup(x);
```

```
}

// iv build() {
    // rt = New(-INF);
    // t[rt].ch[1] = New(INF);
    // ++t[rt].siz;
// }

#define s t[t[x].ch[0]].siz
// 求的是有多少个数小于 val, 排名最后需 +1 //
ii get_rk(int x, int v) { // 获取排名 (get_rk(x, v) + 1)
    int rk = 0;
    while (x) {
        if (v == t[x].val) { rk += s; break; }
        if (v < t[x].val) { // <
            x = t[x].ch[0];
        } else {
            rk += s + t[x].tot;
            x = t[x].ch[1];
        }
    }
    return rk;
}
ii get_val(int x, int rk) {
    // if (!rk) {
        // return -INF;
    // }
    while (x) {
        if (rk <= s)
            x = t[x].ch[0];
        else if (rk <= s + t[x].tot)
            break;
        else {
            rk -= s + t[x].tot;
            x = t[x].ch[1];
        }
    }
    // if (!x) { // rk > 区间数字个数
        // return INF;
    // }
    return t[x].val;
}
#undef s
ii get_pre(int x, int v) {
    int pre = -INF;
    while (x) {
        if (t[x].val < v) {
            pre = t[x].val;
            x = t[x].ch[1];
        } else {
            x = t[x].ch[0];
        }
    }
    return pre;
}
ii get_nx(int x, int v) {
    int nx = INF;
    while (x) {
        if (t[x].val > v) {
```

```
            nx = t[x].val;
            x = t[x].ch[0];
        } else {
            x = t[x].ch[1];
        }
    }
    return nx;
}
```

3.11.3 FHQ Treap

```
struct Node { int ls, rs, val, siz, dat; } t[N];
void pushup(int x) { t[x].siz = tls(x).siz + trs(x).siz + 1; }
int New(int v) {
    t[++cnt].dat = rand(); // 与 Treap 一样有个 dat
    t[cnt].val = v;
    t[cnt].siz = 1;
    return cnt;
}
// 按权值 v 分裂, x : [mn ~ v], y : [v + 1, mx]
void split(int p, int v, int &x, int &y) {
    if (!p) { x = y = 0; return; }
    if (v >= t[p].val) { // 往右子树继续分裂
        x = p;
        split(rs(p), v, rs(p), y);
    } else {
        y = p;
        split(ls(p), v, x, ls(p));
    }
    pushup(p);
}
int merge(int x, int y) { //合并
    if (!x || !y) return x | y;
    if (t[x].dat < t[y].dat) {
        rs(x) = merge(rs(x), y);
        pushup(x);
        return x;
    } else {
        ls(y) = merge(x, ls(y));
        pushup(y);
        return y;
    }
}
int x, y, z;
void insert(int v) { // 插入数 v
    split(rt, v, x, y); // x : [mn ~ v], y : [v + 1, mx]
    rt = merge(merge(x, New(v)), y);
}
void del(int v) { // 删除数 v
    split(rt, v, x, z); // x : [mn ~ v], z : [v + 1, mx]
    split(x, v - 1, x, y); // x : [mn ~ v - 1], y : [ v ]
    // 不要写成 split(rt, v - 1, x, y), split(y, v, y, z);
    y = merge(ls(y), rs(y));
    rt = merge(merge(x, y), z);
}
void get_rk(int v) { // 查询权值为 v 的数的排名
    split(rt, v - 1, x, y);
    printf("%d\n", t[x].siz + 1);
```

```
    rt = merge(x, y);
}
int get_val(int rk) { // 查询排名为 rk 的数的权值
    int x = rk;
    while (1) {
        if (rk == tls(x).siz + 1) return t[x].val;
        if (rk <= tls(x).siz) x = ls(x);
        else {
            rk -= tls(x).siz + 1;
            x = rs(x);
        }
    }
}
int get_pre(int v) { // 数 v 的前驱
    split(rt, v - 1, x, y);
    printf("%d\n", get_val(x, t[x].siz)); // x 中排名最后
    rt = merge(x, y);
}
int get_nx(int v) { // 数 v 的后继
    split(rt, v, x, y);
    printf("%d\n", get_val(y, 1)); // y 中排名第 1
    rt = merge(x, y);
}
```

3.11.4 FHQ Treap 区间翻转

```
struct Node { int ls, rs, val, dat, siz, tag; } t[N];
void pushup(int x) { t[x].siz = tls(x).siz + trs(x).siz + 1; }
void pushdown(int x) {
    if (t[x].tag) {
        swap(ls(x), rs(x));
        tls(x).tag ^= 1;
        trs(x).tag ^= 1;
        t[x].tag = 0;
    }
}
int New(int v) {
    t[++cnt].dat = rand();
    t[cnt].val = v;
    t[cnt].siz = 1;
    return cnt;
}
int build(int l, int r) {
    if (l > r) return 0;
    int mid = (l + r) >> 1;
    int x = New(mid - 1); //  mid - 1
    ls(x) = build(l, mid - 1);
    rs(x) = build(mid + 1, r);
    pushup(x);
    return x;
}
// x : [1 ~ k], y : [k + 1 ~ mx]
void split(int p, int k, int &x, int &y) { // 按个数 k 分裂
    if (!p) { x = y = 0; return; }
    pushdown(p);
    if (k <= tls(p).siz) {
        y = p;
        split(ls(p), k, x, ls(p));
```

```
    } else {
        x = p;    // k - tls(p).siz - 1
        split(rs(p), k - tls(p).siz - 1, rs(p), y);
    }
    pushup(p);
}
int merge(int x, int y) {
    if (!x || !y) return x | y;
    pushdown(x), pushdown(y);
    if (t[x].dat < t[y].dat) {
        rs(x) = merge(rs(x), y);
        pushup(x);
        return x;
    } else {
        ls(y) = merge(x, ls(y));
        pushup(y);
        return y;
    }
}
void rev(int l, int r) { // 翻转区间 [l, r]
    // 因为树中多了个权值 0, 故 split 的时候需加 1
    split(rt, r + 1, x, z); // r
    split(x, l, x, y); // l - 1
    t[y].tag ^= 1;
    rt = merge(merge(x, y), z);
}
void print(int x) { // 中序遍历
    pushdown(x); // 记得清空标记
    if (ls(x)) print(ls(x));
    if (t[x].val) printf("%d ", t[x].val);
    if (rs(x)) print(rs(x));
}
{
    rt = build(1, n + 1); // 在数中插入了序列 [0, n]
    // 所需序列为 1 ~ n, 故额外插入了一个 0
}
```

3.11.5 FHQ Treap 可持续化

```
struct Node { int ls, rs, val, siz, dat; } t[N << 6];
int rt[N]; // 各版本的根节点
int New(int v) {
    t[++cnt].dat = rand();
    t[cnt].val = v;
    t[cnt].siz = 1;
    return cnt;
}
void pushup(int x) { t[x].siz = tls(x).siz + trs(x).siz + 1; }
void split(int p, int v, int &x, int &y) { // 按权值 v 分裂
    if (!p) { x = y = 0; return; }
    // 可持续化平衡树和可持续化文艺平衡树需将 pushup() 写在 if-else 结构内
    // 而平衡树和文艺平衡树可直接在 if-else 后写一个 pushup(p)
    // 易错 : 将 t[p].val 打成 t[x].val
    if (v <= t[p].val) {
        y = ++cnt;
        t[y] = t[p];
        split(ls(y), v, x, ls(y));
        pushup(y);
```

```
20      } else {
21          x = ++cnt;
22          t[x] = t[p];
23          split(rs(x), v, rs(x), y);
24          pushup(x);
25      }
26      // pushup(p);  错误, t[p] 已经是历史版本了, 新版本为 t[x] 或 t[y]
27  }
28  int merge(int x, int y) { // 合并
29      if (!x || !y) return x + y;
30      if (t[x].dat < t[y].dat) {
31          rs(x) = merge(rs(x), y);
32          pushup(x);
33          return x;
34      } else {
35          ls(y) = merge(x, ls(y));
36          pushup(y);
37          return y;
38      }
39  }
40  int get_val(int x, int rk) { // 求版本 x 下排名为 rk 的数对应的权值
41      if (rk == tls(x).siz + 1) return t[x].val;
42      if (rk <= tls(x).siz) return get_val(ls(x), rk);
43      return get_val(rs(x), rk - tls(x).siz - 1);
44  }
45  {
46      rt[i] = rt[p]; // 回到的版本 p
47      switch (opt) {
48          case 1:     // 插入数 v
49              split(rt[i], v, x, y);
50              rt[i] = merge(merge(x, New(v)), y);
51          case 2:     // 删除数 v
52              split(rt[i], v + 1, x, z);
53              split(x, v, x, y);
54              y = merge(ls(y), rs(y));
55              rt[i] = merge(merge(x, y), z);
56          case 3:     // 权值为 v 的数的排名
57              split(rt[i], v, x, y);
58              printf("%d\n", t[x].siz + 1); // 注意要 + 1
59              rt[i] = merge(x, y);
60          case 4:     // 排名为 rk 的数的权值
61              printf("%d\n", get_val(rt[i], rk));
62          case 5:     // 前驱
63              split(rt[i], v, x, y);
64              if (!x) printf("%d\n", -INF);
65              else {
66                  printf("%d\n", get_val(x, t[x].siz));
67                  rt[i] = merge(x, y);
68              }
69          case 6:     // 后继
70              split(rt[i], v + 1, x, y);
71              if (!y) printf("%d\n", INF);
72              else {
73                  printf("%d\n", get_val(y, 1));
74                  rt[i] = merge(x, y);
75              }
76      }
77  }
```

3.11.6 FHQ Treap 可持续化区间翻转

```
struct Node {
    ll val, sum;
    int ls, rs, siz, dat, tag;
} t[N << 6];
ll a, b, lastans;
int n, v, x, y, z, opt, cnt, rt[N];
int New(ll v) {
    t[++cnt].dat = rand();
    t[cnt].val = t[cnt].sum = v;
    t[cnt].siz = 1;
    return cnt;
}
int cpy(int x) { t[++cnt] = t[x]; return cnt; } // 新版本复制版本 x
void update(int x) {
    t[x].siz = tls(x).siz + trs(x).siz + 1;
    t[x].sum = tls(x).sum + trs(x).sum + t[x].val;
}
void pushdown(int x) {
    if (t[x].tag) {
        t[x].tag = 0;
        // 可持续化文艺平衡树与可持续化平衡树相比
        // 不仅要在 split 时保存历史版本
        // 还需在翻转区间 pushdown() 时保存历史版本
        if (ls(x)) ls(x) = cpy(ls(x));
        if (rs(x)) rs(x) = cpy(rs(x));
        swap(ls(x), rs(x));
        tls(x).tag ^= 1;
        trs(x).tag ^= 1;
    }
}
void split(int p, int k, int &x, int &y) {
    if (!p) { x = y = 0; return; }
    pushdown(p);
    if (k <= tls(p).siz) {
        y = cpy(p);
        split(ls(y), k, x, ls(y));
        update(y);
    } else {
        x = cpy(p);
        split(rs(x), k - tls(x).siz - 1, rs(x), y);
        update(x);
    }
}
int merge(int x, int y) {
    if (!x || !y) return x + y;
    pushdown(x), pushdown(y);
    if (t[x].dat < t[y].dat) {
        rs(x) = merge(rs(x), y);
        update(x);
        return x;
    } else {
        ls(y) = merge(x, ls(y));
        update(y);
        return y;
    }
}
{
```

```
    for (int i = 1; i <= n; ++i) {
        p = rd(), opt = rd();
        a = rd() ^ lastans;
        rt[i] = rt[p];
        switch (opt)  {
            case 1: // 在第 a 个数后插入 b
                b = rd() ^ lastans;
                split(rt[i], a, x, y);
                rt[i] = merge(merge(x, New(b)), y);
            case 2:  // 删除第 a 个数
                split(rt[i], a, x, z);
                split(x, a - 1, x, y);
                rt[i] = merge(x, z);
                /* 因为是按照 k split 的, 所以 y 没有子树
                 * 而如果是按照 val 来 split, 且树中含重复权值的话
                 * y 就有可能有子树, 需将 rt[i] = merge(x, z) 替换为以下语句
                 \ y = merge(ls(y), rs(y)),
                 \ rt[i] = merge(merge(x, y), z)
                 * 参见 FHQ Treap 可持续化的 del 操作 */
            case 3: // 翻转区间 [a, b]
                b = rd() ^ lastans;
                split(rt[i], b, x, z);
                split(x, a - 1, x, y);
                t[y].tag ^= 1;
                rt[i] = merge(merge(x, y), z);
            case 4: // 查询区间 [a, b] 所有数的和
                b = rd() ^ lastans;
                split(rt[i], b, x, z);
                split(x, a - 1, x, y);
                lastans = t[y].sum;
                prdntf("%lld\n", lastans);
                rt[i] = merge(merge(x, y), z);
        }
    }
}
```

3.11.7 01Trie

```
namespace Trie {
    int cnt;
    int w[N * 19], siz[N * 19], ch[N * 19][2];
    inline void init() { ch[0][0] = ch[0][1] = cnt = 0; }
    inline void insert(int x, int k) {
        int u = 0;
        for (int i = 18; i >= 0; --i) {
            int d = (x >> i) & 1;
            if (!ch[u][d]) {
                ch[u][d] = ++cnt;
                ch[cnt][0] = ch[cnt][1] = siz[cnt] = 0; // init
            }
            u = ch[u][d];
            siz[u] += k;
        }
        w[u] = x;
    }
    inline int kth(int k) { // 返回排名为 k 的数
        int u = 0;
        for (int i = 18; i >= 0; --i) {
```

```
            if (k <= siz[ch[u][0]]) {
                u = ch[u][0];
            } else {
                k -= siz[ch[u][0]];
                u = ch[u][1];
            }
        }
        return w[u];
    }
    inline int get_rk(int x) { // 返回小于 x 的数字个数
        int u = 0, ans = 0;
        for (int i = 18; i >= 0; --i) {
            if ((x >> i) & 1) {
                ans += siz[ch[u][0]];
                u = ch[u][1];
            } else {
                u = ch[u][0];
            }
            if (!u) break; // 不存在 x
        }
        return ans;
    }
    inline int pre(int x) { return kth(get_rk(x)); }
    inline int suf(int x) { return kth(get_rk(x + 1) + 1); }
}
using namespace Trie;
int n, opt, x;
int main() {
    scanf("%d", &n);
    while (n--) {
        scanf("%d%d", &opt, &x);
        switch (opt) {
            case 1: insert(x + mx, 1); break; // 整体加 mx, 避免负数
            case 2: insert(x + mx, -1); break; // 删数
            case 3: printf("%d\n", get_rk(x + mx) + 1); break;
            case 4: printf("%d\n", kth(x) - mx); break;
            case 5: printf("%d\n", pre(x + mx) - mx); break;
            case 6: printf("%d\n", suf(x + mx) - mx);
        }
    }
    return 0;
}
```

3.11.8 Splay 最大子序列和

```
/*
   1.插入: 在当前数列的第 posi 个数字后插入 tot 个数字: c1, c2 · · · ctot; 若在数列首插入, 则
    posi 为 0
   2.删除: 从当前数列的第 posi 个数字开始连续删除 tot 个数字
   3.修改: 从当前数列的第 posi 个数字开始的连续 tot 个数字统一修改为 c
   4.翻转: 取出从当前数列的第 posi 个数字开始的 tot 个数字, 翻转后放入原来的位置
   5.求和: 计算从当前数列的第 posi 个数字开始的 tot 个数字的和并输出
   6.求最大子列和: 求出当前数列中和最大的一段子列, 并输出最大和
 */
struct Splay {
    int ch[2];
    int fa, val, tag, rev;
    int mx, lmx, rmx, sum, siz; } t[N];
```

```
13 char s[10];
14 int a[N], num[N], rub[N];
15 int n, m, p, len, x, v, rb, rt, cnt, _;
16 ii id(int x) { return x == rs(t[x].fa); }
17 ii New(int x, int v, int fa) {
18     t[x].val = t[x].sum = t[x].mx = v;
19     t[x].lmx = t[x].rmx = max(0, v); //
20     t[x].tag = -INF;
21     t[x].fa = fa;
22     t[x].siz = 1;
23     return x;
24 }
25 iv destroy(int &x) { // &
26     t[x].val = t[x].sum = t[x].rev = t[x].lmx = t[x].rmx = 0;
27     t[x].ch[0] = t[x].ch[1] = t[x].fa = t[x].siz = 0;
28     t[x].mx = t[x].tag = -INF; //
29     rub[++rb] = x;
30     x = 0;
31 }
32 iv pushup(int x) {
33     t[x].siz = tls(x).siz + trs(x).siz + 1;
34     t[x].sum = tls(x).sum + trs(x).sum + t[x].val;
35     t[x].lmx = max(tls(x).lmx, tls(x).sum + t[x].val + trs(x).lmx);
36     t[x].rmx = max(trs(x).rmx, trs(x).sum + t[x].val + tls(x).rmx);
37     t[x].mx = max(tls(x).rmx + t[x].val + trs(x).lmx, max(tls(x).mx, trs(x).mx));
38 }
39 iv update_val(int x, int k) {
40     t[x].val = t[x].tag = k;
41     t[x].sum = t[x].siz * k;
42     t[x].mx = k > 0 ? t[x].sum : k;
43     t[x].lmx = t[x].rmx = max(0, t[x].sum);
44 }
45 iv rev(int x) {
46     swap(t[x].lmx, t[x].rmx);
47     swap(ls(x), rs(x));
48     t[x].rev ^= 1;
49 }
50 iv pushdown(int x) {
51     if (t[x].tag != -INF) {
52         if (ls(x)) update_val(ls(x), t[x].tag);
53         if (rs(x)) update_val(rs(x), t[x].tag);
54         t[x].tag = -INF;
55     }
56     if (t[x].rev) {
57         rev(ls(x)), rev(rs(x));
58         t[x].rev = 0;
59     }
60 }
61 iv rotate(int x) {
62     int f = t[x].fa, ff = t[f].fa;
63     int d = id(x), son = t[x].ch[d ^ 1];
64     t[ff].ch[id(f)] = x, t[x].fa = ff;
65     t[x].ch[d ^ 1] = f, t[f].fa = x;
66     t[f].ch[d] = son, t[son].fa = f;
67     pushup(f);
68 }
69 iv splay(int x, int to) {
70     while (t[x].fa != to) {
71         int f = t[x].fa;
```

```
72          if (t[f].fa != to) {
73              rotate(id(f) ^ id(x) ? x : f);
74          }
75          rotate(x);
76      }
77      pushup(x); //
78      if (!to) rt = x;
79  }
80  int build(int l, int r, int fa) {
81      if (l > r) return 0;
82      int mid = (l + r) >> 1;
83      int x = num[mid];
84      New(x, a[mid], fa);
85      ls(x) = build(l, mid - 1, x);
86      rs(x) = build(mid + 1, r, x);
87      pushup(x);
88      return x;
89  }
90  ii find(int k) {
91      int x = rt;
92      while (x) {
93          pushdown(x);
94          if (k == tls(x).siz + 1) break;
95          if (k <= tls(x).siz) {
96              x = ls(x);
97          } else {
98              k -= tls(x).siz + 1;
99              x = rs(x);
100         }
101     }
102     return x;
103 }
104 ii split(int l, int r) {
105     l = find(l), r = find(r + 2);
106     splay(l, 0),  splay(r, l);
107     return rs(rt);
108 }
109 iv del(int &x) { // &
110     pushdown(x);
111     if (ls(x)) del(ls(x));
112     if (rs(x)) del(rs(x));
113     destroy(x);
114 }
115 int main() {
116     n = rd(), m = rd();
117     t[0].mx = a[1] = a[n + 2] = -INF; // t[0].mx
118     for (int i = 1; i <= n; ++i) a[i + 1] = rd();
119     for (int i = 1; i <= n + 2; ++i) num[i] = i;
120     rt = build(1, n + 2, 0); // rt = build()
121     cnt = n + 2; //
122     while (m--) {
123         scanf("%s", s);
124         switch (s[2]) {
125             case 'S': // INSERT
126                 p = rd(), len = rd();
127                 for (int i = 1; i <= len; ++i) {
128                     a[i] = rd();
129                     num[i] = rb ? rub[rb--] : ++cnt;
130                 }
```

```
131                 x = split(p + 1, p);
132                 ls(x) = build(1, len, x);
133                 pushup(x), pushup(rt);
134                 break;
135             case 'L': // DELETE
136                 p = rd(), len = rd();
137                 x = split(p, p + len - 1);
138                 del(ls(x)), pushup(x), pushup(rt);
139                 break;
140             case 'K': // MAKE-SOME
141                 p = rd(), len = rd(), v = rd();
142                 x = ls(split(p, p + len - 1));
143                 update_val(x, v), pushup(rs(rt)), pushup(rt);
144                 break;
145             case 'V': // REVERSE
146                 p = rd(), len = rd();
147                 x = ls(split(p, p + len - 1));
148                 if (t[x].tag == -INF) {
149                     rev(x), pushup(rs(rt)), pushup(rt);
150                 }
151                 break;
152             case 'T': // GET-SUM
153                 p = rd(), len = rd();
154                 x = ls(split(p, p + len - 1)); // ls
155                 printf("%d\n", t[x].sum);
156                 break;
157             case 'X': // MAX-SUM
158                 printf("%d\n", t[rt].mx);
159                 break;
160         }
161     }
162     return 0;
163 }
```

3.12 动态树

```
1  // 操作均摊复杂度 O(log(n) ^ 2)
2  #define N 300005
3  #define ls(x) t[x].ch[0]
4  #define rs(x) t[x].ch[1]
5  struct Node { int ch[2], fa, tag, val, sum; } t[N];
6  int id(int x) { return x == rs(t[x].fa); }
7  inline int nroot(int x) { // 不是 splay 的根
8      return ls(t[x].fa) == x || rs(t[x].fa) == x;
9  }
10 void pushup(int x) { // 维护信息
11     t[x].sum = tls(x).sum ^ trs(x).sum ^ t[x].val;
12     t[x] = t[ls(x)]. ^ t[rs(x)] ^ w[x];
13 }
14 void rev(int x) { swap(ls(x), rs(x)), t[x].tag ^= 1; }
15 void pushdown(int x) {
16     if (t[x].tag) {
17         if (ls(x)) rev(ls(x));
18         if (rs(x)) rev(rs(x));
19         t[x].tag = 0;
20     }
21 }
22 void push(int x) {
```

```
23      if (nroot(x)) push(t[x].fa);
24      pushdown(x);
25  }
26  void rotate(int x) {
27      int f = t[x].fa, ff = t[f].fa;
28      int d = id(x), son = t[x].ch[d ^ 1];
29      if (nroot(f)) t[ff].ch[id(f)] = x;
30      t[x].ch[d ^ 1] = f, t[f].ch[d] = son;
31      t[son].fa = f, t[f].fa = x, t[x].fa = ff;
32      pushup(f);
33  }
34  void splay(int x) {
35      push(x);
36      while (nroot(x)) {
37          int f = t[x].fa;
38          if (nroot(f)) {
39              rotate(id(x) ^ id(f) ? x : f);
40          }
41          rotate(x);
42      }
43      pushup(x); // 这里一定要 pushup 一下 !!!
44      // 因为 rotate 的时候只 pushup 了 f, 没有 pushup x !
45  }
46  void access(int x) { // 核心操作, 打通 LCT 的根到 x 的路径
47      for (int y = 0; x; y = x, x = fa[x])
48          splay(x), rs(x) = y, pushup(x);
49  }
50  void makeroot(int x) { // 使 x 为 LCT 的根
51      access(x), splay(x), rev(x);
52  }
53  // 找到 LCT 的根, 并将它 splay 到 LCT 顶部
54  int findroot(int x) {
55      access(x), splay(x);
56      while (ls(x)) {
57          pushdown(x);
58          x = ls(x);
59      }
60      splay(x);
61      return x;
62  }
63  void split(int x, int y) {
64      makeroot(x), access(y), splay(y);
65  }
66  void link(int x, int y) { // 连接 x, y 两点
67      makeroot(x);
68      if (findroot(y) ^ x) t[x].fa = y;
69  }
70  void link1(int x, int y) { // 题目保证连边合法
71      // makeroot(x), fa[x] = y;
72      split(x, y), t[x].fa = y; // 维护子树信息时必须写成 split(x, y)
73      // 因为要保证 t[y].siz 是 y 所在树的子树大小, 以此更新 t[x].si (x 的虚子树大小)
74  }
75  void cut(int x, int y) { // 断开 Edge(x, y)
76      split(x, y);
77      if (ls(y) == x && !rs(x)) {
78          ls(y) = t[x].fa = 0;
79          pushup(y);
80      }
81  }
```

```
void cut1(int x, int y) { // 题目保证断边合法
    split(x, y);
    ls(y) = t[x].fa = 0;
    pushup(y); // pushup
}
void modify(int x, int y) { // 把 x 点的权值改为 y
    splay(x), t[x].val = y;
    // access(x), splay(x), t[x].val = y; // 维护子树信息时要这样写
}
int query(int x, int y) { // 查询 [x, y] 的信息
    split(x, y);
    return t[y].sum;
}
```

3.12.1 求 MST

```
// 一道例题: https://www.luogu.com.cn/problem/P2387
#define N 205005 // O(n + m)
struct LCT {
    int ch[2];
    int id, mx, fa, tag; // id : 最大点权所在点编号, mx : 最大点权
} t[N]; // O(n + m)
int w[N]; // O(n + m)
int n, m, x, y, z, num, cnt, ans, _;
ii id(int x) { return x == rs(t[x].fa); }
iv pushup(int x) {
    t[x].id = x, t[x].mx = w[x];
    if (t[x].mx < tls(x).mx) {
        t[x].mx = tls(x).mx;
        t[x].id = tls(x).id;
    }
    if (t[x].mx < trs(x).mx) {
        t[x].mx = trs(x).mx;
        t[x].id = trs(x).id;
    }
}
iv rev(int x) { ... }
iv pushdown(int x) { ... }
ii nroot(int x) { ... }
iv rotate(int x) { ... }
void push(int x) { ... }
iv splay(int x) { ... }
iv access(int x) { ... }
iv makeroot(int x) { ... }
ii findroot(int x) { ... }
iv split(int x, int y) { ... }
iv link(int x, int y) { makeroot(x), t[x].fa = y; } // 只会在不联通的情况下连边
ii check(int x, int y) { // 判断是否不联通
    makeroot(x);
    return x ^ findroot(y);
}
int main() {
    cnt = n = rd(), m = rd(); // cnt = n
    while (m--) {
        x = rd(), y = rd(), z = rd();
        if (check(x, y)) { // 不连通就直接连边
            ans += z; // MST
            w[++cnt] = z; // 边权化为点权, 将 x -> y 的边权变为 cnt 的点权
```

```
            link(x, cnt), link(cnt, y); // 连边 x -> cnt -> y
        } else { // 已联通
            split(x, y);
            if (z < t[y].mx) { // 如果当前边比 LCT 中的最大边小，替换最大边
                ans += z - t[y].mx; // 更新 MST
                splay(num = t[y].id);
                // 这里要预存一下 t[y].id，因为 splay 后 t[y].id 可能会变化
                tls(num).fa = trs(num).fa = 0;
                w[++cnt] = z;
                link(x, cnt), link(cnt, y);
            }
        }
    }
    printf("%d\n", ans);
    return 0;
}
```

3.12.2 求 LCA

```
// 动态求 LCA，可换根
int n, m, s, cnt, _;
int fa[N], tag[N], ch[N][2];
ii id(int x) { ... }
ii nroot(int x) { ... }
iv rotate(int x) { ... }
iv rev(int x) { ... }
iv pushdown(int x) { ... }
iv push(int x) { ... }
iv splay(int x) { ... }
iv access(int x) { ... }
iv makeroot(int x) { ... } // 设置根
iv link(int x, int y) { makeroot(x), fa[x] = y; } // 题目保证是一颗树
ii LCA(int u, int x) {
    access(u); // 先 access u
    int lca = 0;
    // 在 access x 的过程中记录
    for (int y = 0; x; lca = y = x, x = fa[x]) {
        splay(x), rs(x) = y;
    }
    return lca;
}
/* LCA(x, y)
 * 如果先 access(x)，那么 x 和根在一个 Splay 中，LCA 显然也在这颗 Splay 中，
 * 那么再 access(y)，因为 LCA 和 x 在一颗 Splay 中，所以 y 到 LCA 一定是一条虚边，
 * 故可在 access(y) 的时候每次记录跳过的虚边是跳到了谁那里，
 * 然后将最后一次跑虚边的点(父亲)，作为 LCA 输出.
 */
int main() {
    n = rd(), m = rd(), s = rd();
    for (int i = 1; i < n; ++i) link(rd(), rd());
    makeroot(s); // 将根设置为 s
    while (m--) printf("%d\n", LCA(rd(), rd()));
    return 0;
}
```

3.12.3 维护子树信息

```
// 维护子树大小和（需通过维护虚树子树大小和）
// https://www.luogu.com.cn/problem/P4219
```

```
struct LCT {
    int ch[2];
    int fa, tag, siz, si; // siz : 子树大小和, si : 虚子树大小和
} t[N];
char s[2];
int n, m, x, y, _, cnt;
ii id(int x) { ... }
ii nroot(int x) { ... }
iv pushup(int x) {
    t[x].siz = tls(x).siz + trs(x).siz + 1 + t[x].si; // 需要额外加上虚子树大小和
}
iv rotate(int x) { ... }
iv rev(int x) { ... }
iv pushdown(int x) { ... }
void push(int x) { ... }
iv splay(int x) { ... }
iv access(int x) {
    for (int y = 0; x; y = x, x = t[x].fa) {
        splay(x);
        t[x].si += trs(x).siz - t[y].siz;
        // y 的虚子树和减去原右子树子树和, 加上新右子树子树和
        rs(x) = y;
        pushup(x);
    }
}
iv makeroot(int x) { ... }
iv link(int x, int y) {
    makeroot(x), makeroot(y);
    t[x].fa = y;
    t[y].si += t[x].siz; // y 的虚子树和增加了 x 的子树和
    pushup(y);
}
iv split(int x, int y) { ... }
int main() {
    n = rd(), m = rd();
    while (m--) {
        scanf("%s", s);
        x = rd(), y = rd();
        switch (*s) {
            case 'A': link(x, y); break;
            case 'Q':
                split(x, y); // 因为 x 与 y 直接相连
                // 则 split 后 x 仅有一条与 y 相连的实边, y 同理仅有 1 条
                // 故 x 的虚子树和加上自己的个数 1 即得 x 的子树和, y 同理可得
                printf("%lld\n", (t[x].si + 1ll) * (t[y].si + 1));
                break;
        }
    }
    return 0;
}
```

3.12.4 求重心

```
// 给一颗以 1 为根的树, 输出以 i 为根的子树的重心 (1 <= i <= n)
struct Node { int ch[2], l, r, siz, si, tag, fa; } t[N << 1];
struct Edge { int to, nx; } e[N << 1];
int n, x, y, cnt;
int h[N], pre[N], dep[N], ans[N][2];
```

```
// pre[i] : 并查集, 记录以 i 为根的子树的重心, ans[i][2] : 记录节点两个重心
inline void add(int u, int v) { e[++cnt] = (Edge) {v, h[u]}, h[u] = cnt; }
namespace LCT {
    ii id(int x) { ... }
    ii nroot(int x) { ... }
    iv pushup(int x) { t[x].siz = tls(x).siz + trs(x).siz + 1 + t[x].si; }
    iv rotate(int x) { ... }
    iv rev(int x) { ... }
    iv pushdown(int x) { ... }
    void push(int x) { ... }
    iv splay(int x) { ... }
    iv access(int x) {
        for (int y = 0; x; y = x, x = t[x].fa) {
            splay(x);
            t[x].si += trs(x).siz - t[y].siz; // 更新虚子树大小
            rs(x) = y;
            pushup(x);
        }
    }
    iv makeroot(int x) { ... }
    iv split(int x, int y) { ... }
    iv link(int x, int y) {
        split(x, y);
        t[x].fa = y;
        t[y].si += t[x].siz; // 更新虚子树大小
        pushup(y);
    }

    int zx[3], ff; // 记录重心和重心个数
    ii getc(int x) {
        int odd = t[x].siz & 1, half = t[x].siz >> 1;
        int lsum = 0, rsum = 0, now = 0, L, R; // now 要初始化
        ff = 0;
        while (x) {
            pushdown(x); //
            L = t[ls(x)].siz + lsum, R = t[rs(x)].siz + rsum;
            if (L <= half && R <= half) { // 重心性质
                if (odd) { // 大小为奇数, 则只有一个重心
                    zx[++ff] = x;
                    now = x;
                    break;
                } else { // 否则有两个重心
                    zx[++ff] = x;
                    now = dep[now] < dep[x] ? x : now;
                    // 要让并查集指向深度大的那个重心, 才能保证后续合并找重心正确
                    if (ff == 2) break; // 找完两个重心
                }
            }
            if (L < R) {
                lsum += t[ls(x)].siz + t[x].si + 1;
                x = rs(x);
            } else {
                rsum += t[rs(x)].siz + t[x].si + 1;
                x = ls(x);
            }
        }
        splay(now);
        return now;
    }
```

```cpp
}
using namespace LCT;
int find(int x) { return pre[x] == x ? x : pre[x] = find(pre[x]); } // 并查集
void dfs(int u, int p) {
    int f = 0;
    dep[u] = dep[p] + 1;
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (v != p) {
            f = 1;
            dfs(v, u);
            link(u, v); // 合并两棵树
            int fu = find(u), fv = find(v); // 记录两颗旧树的重心
            split(fu, fv); // 新的重心在两个旧的重心的路径上
            int z = getc(fv); // 找到新树的重心
            pre[fu] = pre[fv] = pre[z] = z; // 并查集记录重心
        }
    }
    if (f) { // 有子树
        ans[u][0] = zx[1];
        if (ff == 2) { // 有两个重心
            ans[u][1] = zx[2];
            if (ans[u][0] > ans[u][1]) {
                swap(ans[u][0], ans[u][1]); // 题目要求按重心按递增序输出
            }
        }
    } else ans[u][0] = u; // 无子树, 重心是自己
}
int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) {
        t[i].siz = 1, pre[i] = i;
    }
    for (int i = 1; i < n; ++i) {
        scanf("%d%d", &x, &y);
        add(x, y), add(y, x);
    }
    dfs(1, 0);
    for (int i = 1; i <= n; ++i) { // 输出每个 1 ~ n 节点的重心
        if (ans[i][1]) { // 有两个重心
            printf("%d %d\n", ans[i][0], ans[i][1]);
        } else { // 一个重心
            printf("%d\n", ans[i][0]);
        }
    }
    return 0;
}
```

3.12.5 求直径

```cpp
// 动态维护树的直径
struct DIA { int x, y, dis; } d[N]; // Diameter, 记录直径两个端点和直径的长度
char s[2];
int n, x, y, dis, cnt, _, pre[N];
namespace LCT {
    struct Node { int ch[2], fa, siz, tag; } t[N];
    ii id(int x) { ... }
    ii nroot(int x) { ... }
```

```
    iv pushup(int x) { ... }
    iv rev(int x) { ... }
    iv pushdown(int x) { ... }
    void push(int x) { ... }
    iv rotate(int x) { ...  }
    iv splay(int x) { ... }
    iv access(int x) { ... }
    iv makeroot(int x) { ... }
    iv split(int x, int y) { ... }
    iv link(int x, int y) { ... }
}
using namespace LCT;
int find(int x) { return pre[x] ? pre[x] = find(pre[x]) : x; }
int query(int x, int y) { // 查询 x, y 之间的距离
    split(x, y);
    return t[y].siz - 1;
}
void update(int x, int y) { // 更新直径
    x = find(x), pre[y] = x;
    int a = d[x].x, b = d[x].y;
    dis = query(a, y);
    if (dis > d[x].dis) d[x] = (DIA) {a, y, dis};
    dis = query(b, y);
    if (dis > d[x].dis) d[x] = (DIA) {b, y, dis};
}
int main() {
    n = rd();
    while (n--) {
        scanf("%s", s);
        x = rd();
        if (*s == 'B') { // 新建一个节点, 如果 x != -1, 则新节点与 x 相连
            y = ++cnt;
            d[y] = (DIA) {y, y, 0}; // 初始化 y 的直径
            if (~x) { // 连接
                link(x, y);
                update(x, y);
            }
        } else { // 查询在 x 节点所在的连通块中, 距它最远的点的距离
            // 显然最远距离为 x 到直径两个端点之一
            y = find(x);
            printf("%d\n", max(query(x, d[y].x), query(x, d[y].y)));
        }
    }
    return 0;
}
```

3.12.6 LCT+ 随机化（对子集 hash）

```
struct Edge { int x, y, w; } e[M];
int n, m, opt, x, y, u, v, S, cnt, _, w[N];
namespace LCT {
    struct Node { int fa, sum, su, tag, ch[2]; } t[N]; // 维护子树异或和
    ii id(int x) { ... }
    ii nroot(int x) { ... }
    iv pushup(int x) { t[x].sum = tls(x).sum ^ trs(x).sum ^ w[x] ^ t[x].su; } //
    iv rev(int x) { ... }
    iv pushdown(int x) { ... }
    void push(int x) { ... }
```

```
11      iv rotate(int x) { ... }
12      iv splay(int x) { ... }
13      iv access(int x) {
14          for (int y = 0; x; y = x, x = t[x].fa) {
15              splay(x);
16              t[x].su ^= trs(x).sum ^ t[y].sum; //
17              rs(x) = y;
18              pushup(x);
19          }
20      }
21      iv makeroot(int x) { ... }
22      ii findroot(int x) { ... }
23      iv split(int x, int y) { ... }
24      iv link(int x, int y) {
25          split(x, y); // 维护子树信息要把 y 也转上去
26          t[x].fa = y;
27          t[y].su ^= t[x].sum; //
28          pushup(y);
29      }
30      iv cut(int x, int y) { ... }
31  }
32  using namespace LCT;
33  int seed = 19260817;
34  inline int rnd() { return seed = (seed * 7 + 13) ^ (seed / 13 - 7); } // 随机数
35  int main() {
36      rd(), n = rd(), m = rd();
37      for (int i = 1; i < n; ++i) {
38          link(rd(), rd()); // 初始图连边
39      }
40      while (m--) {
41          opt = rd(), x = rd();
42          switch (opt) {
43              case 1:
44                  y = rd(), u = rd(), v = rd();
45                  cut(x, y), link(u, v); // 断开边 x, y ; 连接边 u, v
46                  break;
47              case 2: // 标记一条路径 (x - y)
48                  e[++cnt] = (Edge) {x, y = rd(), rnd()}; // 记录第 cnt 条边的两个端点及随机权
    值
49                  S ^= e[cnt].w; // S 异或上随机权值
50                  access(x), splay(x); // 先 access 和 splay , 再更新权值 !
51                  w[x] ^= e[cnt].w; // LCT 中端点也异或上这个随机权值
52                  access(y), splay(y);
53                  w[y] ^= e[cnt].w;
54                  break;
55              case 3: // 删除第 x 次标记路径的标记
56                  u = e[x].x, v = e[x].y; // 该路径的两个端点
57                  S ^= e[x].w; // 再次异或该路径端点的随机权值
58                  access(u), splay(u); // 更新 LCT
59                  w[u] ^= e[x].w; // 更新 LCT
60                  access(v), splay(v);
61                  w[v] ^= e[x].w;
62                  break;
63              case 4: // 查询所有标记的路径是否都经过边 (x - y)
64                  split(x, y = rd());
65                  /*
66                   * 查询时先将 y 选到顶部, 这样若有标记路径经过边 (x - y),
67                   * 则路径必有一个端点在子树 y 内, 所以每标记一条路径
68                   * 就对该路径的两个端点随机一个较大的权值,
```

```
                 * 并将 S 异或上这个权值 (S 初始为 0),
                 * 这样如果查询时 x 子树内的权值异或和等于 S,
                 * 就表示所有标记路径都经过边 (x - y),
                 * 该法在随机权值为 1e9 级别下的错误率十分低
                */
                puts(t[x].sum ^ S ? "NO" : "YES");
                break;
            }
        }
        return 0;
}
```

3.12.7 LCT+ 主席树维护区间联通块个数

```
/*
 * 你有一个无向图 G(V, E), E 中每一个元素用一个二元组 (u, v) 表示.
 * 现在把 E 中的元素排成一个长度为 m 序列 A.
 * 然后给你 q 个询问二元组 (l, r),
 * 表示询问由序列 A 中 [l, r] 位置的边组成的图的联通块个数.
 */
// N : 点数, M : 序列长度上界
struct Edge { int x, y; } e[M];
int n, m, q, f, x, y, num, cnt, lans, _, rt[M];

namespace LCT { // LCT 维护最早加入边的编号
    struct Node { int val, mn, fa, tag, ch[2]; } t[N + M];
    ii id(int x) { ... }
    ii nroot(int x) { ... }
    iv rev(int x) { ... }
    iv pushup(int x) { t[x].mn = min(t[x].val, min(tls(x).mn, trs(x).mn)); } // 维护最早加入的边的编号
    iv pushdown(int x) { ... }
    void push(int x) { ... }
    iv rotate(int x) { ... }
    iv splay(int x) { ... }
    iv access(int x) { .. }
    ii findroot(int x) { ... }
    iv makeroot(int x) { ... }
    iv split(int x, int y) { ... }
    ii check(int x, int y) { ... }
    iv link(int x, int y) { ... }
    iv cut(int x, int y) { ... } // 最后不要忘了 pushup(y)
}
using namespace LCT;

namespace Tree { // 主席树维护区间联通块减少个数
    struct Node { int l, r, tot; } t[M * 40];
    void update(int &now, int x, int k, int l = 1, int r = m) {
        t[++cnt] = t[now];
        t[cnt].tot += k;
        now = cnt;
        if (l == r) return;
        int mid = (l + r) >> 1;
        if (x <= mid) {
            update(t[now].l, x, k, l, mid);
        } else {
            update(t[now].r, x, k, mid + 1, r);
        }
```

```
    }
    int query(int p, int x, int y, int l = 1, int r = m) {
        if (x <= l && y >= r) return t[p].tot;
        int ans = 0, mid = (l + r) >> 1;
        if (x <= mid) ans = query(t[p].l, x, y, l, mid);
        if (y > mid) ans += query(t[p].r, x, y, mid + 1, r);
        return ans;
    }
}

int main() {
    n = rd(), m = rd(), q = rd(), f = rd();
    for (int i = 0; i <= n; ++i) { // 0 ~ n
        t[i].mn = t[i].val = INF; // init
    }
    // 注意: rt[i] 存储的是询问右端点为 i 的答案
    for (int i = 1; i <= m; ++i) { //  依次更新询问右端点为 i 的主席树
        x = rd(), y = rd();
        rt[i] = rt[i - 1]; // 可持续化操作
        if (x == y) continue; // 自环
        e[i] = (Edge) {x, y};
        t[i + n].val = t[i + n].mn = i;
        // 在边中间加入一个新点 i + n, 并令点权为边的编号 i (加入顺序)
        Tree::update(rt[i], i, 1); // 1 ~ i 联通块减少一个
        // 因为每次查询 rt[i] 的右端点都是 i
        // 所以相当于对所有右端点为 i 的查询结果都加了 1
        if (check(x, y)) { // 联通
            split(x, y);
            num = t[y].mn; // x - y 路径上最早加入的边
            cut(e[num].x, num + n), cut(num + n, e[num].y); // 间接断开边 e[num].x, e[num
    ].y
            Tree::update(rt[i], num, -1);
            /*
               原来就是联通的. 因为之前在 rt[num] 已经对 num 这个位置加过 1 了,
               所以显然 rt[i] 的 num 位置也是加了 1 的,
               (以下为询问右端点为 i 时的情况)
               所以询问左端点为 [1, num] 的区间加了 2, 左端点为 (num, i] 的区间加了 1
               故现在需要在 num 这个位置减 1, 使得询问左端点为 [1, i] 的结果区间都是加 1
               */
        }
        link(x, i + n), link(i + n, y); // x -> i + n -> y
    }
    while (q--) {
        x = rd(), y = rd();
        if (f) { // 强制在线参数
            x = (x + f * lans) % m + 1;
            y = (y + f * lans) % m + 1;
        }
        if (x > y) sawp(x, y);
        // 注意查询 rt[y] 的 [x, y] 区间
        printf("%d\n", lans = n - Tree::query(rt[y], x, y)); // 转化为 n - 联通块减少个数
        // 不要忘了让 lans = 新结果
    }
    return 0;
}
```

3.12.8 路径乘, 路径加

```
char s[2];
```

```
int n, m, x, y, a, b, _;
namespace LCT {
    struct Node {
        int ch[2];
        int fa, siz, rev;
        ll sum, add, mul, val;
    } t[N];
    ii id(int x) { ... }
    ii nroot(int x) { ... }
    iv pushup(int x) {
        t[x].siz = tls(x).siz + trs(x).siz + 1;
        t[x].sum = (tls(x).sum + trs(x).sum + t[x].val) % P;
    }
    iv pushr(int x) {
        if (!x) return;
        swap(ls(x), rs(x));
        t[x].rev ^= 1;
    }
    iv pushm(int x, ll mul) {
        if (!x) return;
        t[x].sum = t[x].sum * mul % P;
        t[x].mul = t[x].mul * mul % P;
        t[x].add = t[x].add * mul % P;
        t[x].val = t[x].val * mul % P;
    }
    iv pusha(int x, ll add) {
        if (!x) return;
        t[x].sum = (t[x].sum + add * t[x].siz) % P;
        t[x].add = (t[x].add + add) % P;
        t[x].val = (t[x].val + add) % P;
    }
    iv pushdown(int x) {
        if (t[x].mul != 1) { // 先下放 mul 标记
            pushm(ls(x), t[x].mul);
            pushm(rs(x), t[x].mul);
            t[x].mul = 1;
        }
        if (t[x].add) { // 再下放 add 标记
            pusha(ls(x), t[x].add);
            pusha(rs(x), t[x].add);
            t[x].add = 0;
        }
        if (t[x].rev) {
            pushr(ls(x));
            pushr(rs(x));
            t[x].rev = 0;
        }
    }
    iv push(int x) { ... }
    iv rotate(int x) { ... }
    iv splay(int x) { ... }
    iv access(int x) { ... }
    iv makeroot(int x) { ... }
    iv split(int x, int y) { ... }
    iv link(int x, int y) { ... }
    iv cut(int x, int y) { ... }
}
using namespace LCT;
int main() {
```

```
    n = rd(), m = rd();
    for (int i = 1; i <= n; ++i) t[i].val = t[i].mul = 1; // 初始化点权(本题给定为 1) 和 mul 标记
    for (int i = 1; i < n; ++i) link(rd(), rd()); // 连边
    while (m--) {
        scanf("%s%d%d", s, &x, &y);
        switch (*s) {
            case '+': // x, y 路径上点权都加 a
                split(x, y);
                pusha(y, a = rd());
                break;
            case '-': // 删边 x, y, 加边 a, b
                cut(x, y), link(a = rd(), b = rd());
                break;
            case '*':  // x, y 路径上点权都乘 a
                scanf("%d", &a);
                split(x, y);
                pushm(y, a = rd());
                break;
            case '/': // 询问 x, y 路径点权和
                split(x, y);
                printf("%lld\n", t[y].sum);
        }
    }
    return 0;
}
```

3.12.9 维护双联通分量

```
int n, m, opt, fx, x, y, cnt;
int pre[N], scc[N], w[N]; // w[] 为原始节点的点权
// 两个并查集: pre[] 维护联通性, scc[] 维护双联通分量
int find(int x) { return pre[x] ^ x ? pre[x] = find(pre[x]) : x; }
int find2(int x) { return scc[x] ^ x ? scc[x] = find2(scc[x]) : x; }
namespace LCT {
    struct Node { int fa, rev, val, sum, ch[2]; } t[N << 1];
    ii id(int x) { ... }
    ii nroot(int x) { ... }
    iv pushup(int p) { t[p].sum = tls(p).sum + trs(p).sum + t[p].val; }
    iv pushr(int x) { ... }
    iv pushdown(int x) { ... }
    void push(int x) { ... }
    iv rotate(int x) { ... }
    iv splay(int x) { ... }
    iv access(int x) {
        for (int y = 0; x; y = x, x = t[x].fa = find2(t[x].fa)) { // 注意这里要更新 x 的父亲
            splay(x), rs(x) = y, pushup(x);
        }
    }
    iv makeroot(int x) { ... }
    iv split(int x, int y) { ... }
    iv link(int x, int y) {
        makeroot(x), t[x].fa = y, pre[find(x)] = find(y); // 并查集维护联通性
    }
}
using namespace LCT;
void dfs(int x, int tpf) { // 对一个双联通分量内的点进行缩点
```

```
    if (!x) return;
    scc[find2(x)] = tpf; // 记录双联通分量
    dfs(ls(x), tpf);
    dfs(rs(x), tpf);
}
int main() {
    n = rd(), m = rd();
    for (int i = 1; i <= n; ++i) {
        t[i].val = w[i] = rd(); // init
        pre[i] = scc[i] = i; // init
    }
    while (m--) {
        opt = rd(), x = rd(), y = rd();
        switch (opt) {
            case 1: // 连边 x - y
                x = find2(x), y = find2(y);
                if (x == y) break; // 双联通
                if (find(x) ^ find(y)) { // 不联通
                    link(x, y);
                } else { // 单联通
                    split(x, y);
                    t[y].val = t[y].sum; // 缩点后的点权
                    dfs(y, y); // 缩点
                    ls(y) = rs(y) = 0;
                }
                break;
            case 2: // 将点 x 的点权修改为 y
                fx = find2(x);
                access(fx), splay(fx);
                t[fx].val += y - w[x];
                w[x] = y, pushup(fx);
                break;
            case 3: // 询问 x, y 间的最大点权和
                x = find2(x), y = find2(y);
                if (find(x) ^ find(y)) { // x, y 不联通
                    puts("-1");
                } else {
                    split(x, y);
                    printf("%d\n", t[y].sum);
                }
        }
    }
    return 0;
}
```

3.13 笛卡尔树

```
/*  笛卡尔树性质:
    1. 每个节点的下标满足二叉搜索树的性质
    2. 每个节点的权值满足小根堆的性质
 */
// 应用: 求元素的左右延伸区间; 确定分治的边界; ...
int n, w[N], ls[N], rs[N], st[N];
// w[] : 初始权值, ls[], rs[] : 笛卡尔树的左右儿子
void build1() { // 标准构造笛卡尔树
    int top = 0, t = 0;
    for (int i = 1; i <= n; ++i) {
        t = top;
```

```
        while (t && w[st[t]] > w[i]) --t;
        if (t) rs[st[t]] = i;
        if (t < top) ls[i] = st[t + 1];
        st[++t] = i, top = t;
    }
}
void build() { // 个人精简版，只有当结点编号从 1 开始时才能这么用!
    int t = 0; // top
    for (int i = 1; i <= n; ++i) {
        while (t && w[st[t]] > w[i]) ls[i] = st[t--];
        rs[st[t]] = i, st[++t] = i;
    }
    rt = st[1];
}
```

3.14 左偏树 (可并堆)

```
/*
 * 一开始有 n 个小根堆，每个堆包含且仅包含一个数. 接下来需要支持两种操作:
 * 1 x y : 将第 x 个数和第 y 个数所在的小根堆合并
 *  (若第 x 或第 y 个数已经被删除或第 x 和第 y 个数在用一个堆内，则无视此操作)
 * 2 x : 输出第 x 个数所在的堆最小数，并将这个最小数删除
 * (若有多个最小数，优先删除先输入的；若第 x 个数已经被删除，则输出 -1 并无视删除操作)
 */
struct Tree {
    int ls, rs, w, d, fa, del;
    // w : 权值, d => dist : 离自己最近的子树距离, fa => rt : 堆顶, del : 标记删除
} t[N];
int n, m, opt, x, y;
int find(int x) { // 找根，也就是找堆顶
    return t[x].fa == x ? x : t[x].fa = find(t[x].fa);
}
int merge(int x, int y) { // 合并两个堆
    if (!x || !y) return x | y;
    // 满足小根堆性质及题目约束
    if (t[x].w > t[y].w || (t[x].w == t[y].w && x > y)) swap(x, y);
    rs(x) = merge(rs(x), y);
    if (trs(x).d > tls(x).d) swap(ls(x), rs(x)); // 左偏树性质
    tls(x).fa = trs(x).fa = x;
    t[x].d = trs(x).d + 1; // 左偏树性质
    return x;
}
inline void pop(int x) { // 删数
    t[x].del = 1; // 标记删除
    t[x].fa = tls(x).fa = trs(x).fa = merge(ls(x), rs(x));
}
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i) { // 初始化堆
        scanf("%d", &t[i].w);
        t[i].fa = i;
    }
    while (m--) {
        scanf("%d%d", &opt, &x);
        if (opt == 1) {
            scanf("%d", &y);
            if (t[x].del || t[y].del) continue; // 至少一个被删
            x = find(x), y = find(y);
```

```
            if (x != y) merge(x, y);
        } else {
            if (t[x].del) puts("-1");
            else printf("%d\n", t[find(x)].w), pop(find(x));
        }
    }
    return 0;
}

//
```

3.15 Trie 树

```
string ss;
char s[55];
int cnt, vis[N], trie[N][26]; // N 大小为 <字符串个数> * <字符串长度>
void insert(string s) { // string 的输入方式
    int p = 0;
    for (auto &c : s) {
        int x = c - 'a';
        if (!trie[p][x]) trie[p][x] = ++cnt;
        p = trie[p][x];
    }
    vis[p] = 1;
}
// 插入字符串
void insert(char *s) { // char[] 的输入方式
    int p = 0;
    for (int i = 0; s[i]; ++i) {
        int x = s[i] - 'a';
        if (!trie[p][x]) trie[p][x] = ++cnt;
        p = trie[p][x];
    }
    vis[p] = 1;
}
int search(char *s) { // 搜索 trie 树是否有 s 这个串
    int p = 0, len = strlen(s);
    for (int i = 0; i < len; ++i) {
        int x = s[i] - 'a';
        if (!trie[p][x]) return 0;
        p = trie[p][x];
    }
    return vis[p] ? vis[p]++ : vis[p];
    // vis =  0 : 没有这个串, 1 : 有这个串, 2 : 有这个串但重复查询了
}
```

3.15.1 01Trie 平衡树

```
const int mx = 1e7; // 数的绝对值最大值
namespace Trie {
    int w[N * 19], siz[N * 19], ch[N * 19][2], cnt;
    inline void init() { ch[0][0] = ch[0][1] = cnt = 0; }
    inline void insert(int x, int k) {
        int u = 0;
        for (int i = 18; i >= 0; --i) {
            int d = (x >> i) & 1;
            if (!ch[u][d]) {
```

```
                ch[u][d] = ++cnt;
                ch[cnt][0] = ch[cnt][1] = siz[cnt] = 0;
            }
            u = ch[u][d];
            siz[u] += k;
        }
        w[u] = x;
    }
    inline int kth(int k) {
        int u = 0;
        for (int i = 18; i >= 0; --i) {
            if (k <= siz[ch[u][0]]) {
                u = ch[u][0];
            } else {
                k -= siz[ch[u][0]];
                u = ch[u][1];
            }
        }
        return w[u];
    }
    inline int get_rk(int x) { // 返回小于 x 的数字个数
        int u = 0, ans = 0;
        for (int i = 18; i >= 0; --i) {
            if ((x >> i) & 1) {
                ans += siz[ch[u][0]];
                u = ch[u][1];
            } else {
                u = ch[u][0];
            }
            if (!u) break; // x 不存在
        }
        return ans;
    }
    inline int pre(int x) { return kth(get_rk(x)); }
    inline int suf(int x) { return kth(get_rk(x + 1) + 1); }
}
using namespace Trie;
int n, opt, x;
int main() {
    scanf("%d", &n);
    while (n--) {
        scanf("%d%d", &opt, &x);
        switch (opt) {
            case 1: insert(x + mx, 1); break; // 整体加 mx, 避免负数
            case 2: insert(x + mx, -1); break; // 删除
            case 3: printf("%d\n", get_rk(x + mx) + 1); break; // x 的排名
            case 4: printf("%d\n", kth(x) - mx); break; // 排名为 x 的数
            case 5: printf("%d\n", pre(x + mx) - mx); break; // 前驱
            case 6: printf("%d\n", suf(x + mx) - mx); // 后继
        }
    }
    return 0;
}
```

3.15.2 可持续化 Trie 树

```
/*
 * 给定一个非负整数序列 {a}, 初始长度为 n.
```

```
 * 有 m 个操作，有以下两种操作类型：
 * A x：添加操作，表示在序列末尾添加一个数 x，序列的长度 n + 1.
 * Q l r x：询问操作，你需要找到一个位置 p，满足 l ≤ p ≤ r，
 * 使得： a[p] ⊕ a[p + 1] ⊕ ... ⊕ a[N] ⊕ x 最大，输出最大是多少.
 * */
char c[2];
int n, m, l, r, x, cnt;
int s[N], rt[N], t[N * 25][2], last[N * 25];
void insert(int &now, int pre, int i, int k = 23) {
    now = ++cnt;
    if (k < 0) {
        last[now] = i; // 子树下标最大值
        return;
    }
    int d = s[i] >> k & 1;
    if (pre) t[now][d ^ 1] = t[pre][d ^ 1];
    insert(t[now][d], t[pre][d], i, k - 1);
    last[now] = max(last[t[now][0]], last[t[now][1]]); // 上传
}
int query(int now, int x, int L) {
    for (int i = 23; i >= 0; --i) {
        int d = x >> i & 1;
        if (last[t[now][d ^ 1]] >= L) // 查询最大下标 >= l - 1 的子树
            now = t[now][d ^ 1];
        else now = t[now][d];
    }
    return s[last[now]] ^ x;
}
int main() {
    *last = -1; // 必加，初始化未建节点最大下标为 -1
    n = rd(), m = rd();
    insert(*rt, 0, 0); //
    for (int i = 1; i <= n; ++i) {
        x = rd();
        s[i] = s[i - 1] ^ x;
        insert(rt[i], rt[i - 1], i);
    }
    while (m--) {
        scanf("%s", c);
        if (*c == 'A') {
            ++n;
            x = rd();
            s[n] = s[n - 1] ^ x;
            insert(rt[n], rt[n - 1], n);
        } else {
            l = rd(), r = rd(), x = rd();
            printf("%d\n", query(rt[r - 1], s[n] ^ x, l - 1)); //
        }
    }
    return 0;
}
```

3.16 虚树

```
// 1.建虚树  2.虚树上DP
int st[N], t; // 栈
inline void build(int s) { // 建立虚树
```

```
    sort(a + 1, a + 1 + k, [&] (int x, int y) { return id[x] < id[y]; }); // 关键点按
    dfn 排序
    st[t = 1] = s, h1[s] = cnt1 = 0; // 加入 s, 并清空 s 的邻接表
    for (int i = 1; i <= k; ++i) {
        if (a[i] == s) continue; // s 不能重复加入
        int lca = LCA(st[t], a[i]);
        if (lca ^ st[t]) { // a[i] 与栈顶不在一条链上
            while (id[st[t - 1]] > id[lca]) add1(st[t - 1], st[t]), --t;
            if (id[lca] > id[st[t - 1]]) { // lca 还未入过栈, 连接并入栈
                h1[lca] = 0, add1(lca, st[t]), st[t] = lca;
            } else {
                add1(lca, st[t--]); // 相等, lca 已入过栈, 直接连接
            }
        }
        h1[a[i]] = 0, st[++t] = a[i]; // a[i] 第一次加入, 需先清空 a[i] 的邻接表
    }
    for (int i = 1; i < t; ++i) add1(st[i], st[i + 1]); // 将剩下在一条链上的点依次连接
}
```

3.17 柯朵莉树

```
// 数据随机的情况下  O(nlog(log n))
// 支持区间加, 区间赋值, 查询区间第 k 小与区间每个数字的 x 次方的和模 y 的值

struct Node {
    ll l, r; mutable ll v; // mutable !!
    bool operator < (const Node &b) const { return l < b.l; }
};
set<Node> s; // 核心为 set
auto split(int pos) { // ODT 核心操作 1, 按位置 pos 分裂
    auto it = s.lower_bound({pos, 0, 0});
    if (it != s.end() && it->l == pos) return it;
    --it; ll l = it->l, r = it->r, v = it->v;
    s.erase(it), s.insert({l, pos - 1, v});
    return s.insert({pos, r, v}).first; // .first
} // 返回结构体左端点为 pos 的迭代器
void assign(ll l, ll r, ll v) { // ODT 核心操作 2, 区间赋值
    auto R = split(r + 1), L = split(l); // 注意 split 顺序
    s.erase(L, R), s.insert({l, r, v});
}
void add(ll l, ll r, ll v) { // 区间加
    auto R = split(r + 1);
    for (auto it = split(l); it != R; ++it) it->v += v;
}
ll kth(ll l, ll r, ll k) { // 区间第 k 大
    vector<pair<ll, ll> > v;
    auto R = split(r + 1);
    for (auto it = split(l); it != R; ++it) {
        v.push_back({it->v, it->r - it->l + 1});
    }
    sort(v.begin(), v.end());
    for (auto &x : v) {
        k -= x.second;
        if (k <= 0) return x.first;
    }
    return -1;
}
ll psum(ll l, ll r, ll x, ll y) { // 区间每个数字的 x 次方的和模 y 的值
```

```
    ll ans = 0; auto R = split(r + 1);
    for (auto it = split(l); it != R; ++it) {
        ans = (ans + qpow(it->v, x, y) * (it->r - it->l + 1)) % y;
    }
    return ans;
}

s.insert({i, i, w}); // 初始化 i 位置为 w ( 插入数据 )
```

3.18 树套树

3.18.1 线段树套线段树【单点修改 + 矩形最值】

```
// 空间 O(n ^ 2 * 32)
/* c x y z : 将 (x, y) 改为 z
 * q x1 y1 x1 y2 : 找出矩阵 (x1, y1) ~ (x2, y2) 的 max 和 min */
#include <bits/stdc++.h>
using namespace std;

#define N 810
#define ls(x) (x << 1)
#define rs(x) (x << 1 ^ 1)

char s[2];
int n, q, mx1, mn1;
int a[N][N], mx[N << 2][N << 2], mn[N << 2][N << 2];

inline void pushup(int x, int y) {
    mx[x][y] = max(mx[x][ls(y)], mx[x][rs(y)]);
    mn[x][y] = min(mn[x][ls(y)], mn[x][rs(y)]);
}

void buildy(int l, int r, int px, int py, int ff) {
    if (l == r) {
        if (~ff) { // ff 用于建树
            mx[px][py] = mn[px][py] = a[ff][l];
            // mx[px][py] = INT_MIN, mn[px][py] = INT_MAX;
        } else {
            mx[px][py] = max(mx[ls(px)][py], mx[rs(px)][py]);
            mn[px][py] = min(mn[ls(px)][py], mn[rs(px)][py]);
        }
        return;
    }
    int mid = (l + r) >> 1;
    buildy(l, mid, px, ls(py), ff);
    buildy(mid + 1, r, px, rs(py), ff);
    pushup(px, py);
}

void buildx(int l, int r, int p) {
    if (l == r) {
        buildy(1, n, p, 1, l);
        return;
    }
    int mid = (l + r) >> 1;
    buildx(l, mid, ls(p));
    buildx(mid + 1, r, rs(p));
    buildy(1, n, p, 1, -1); // ff = -1
}
```

```
47
48  void updatey(int l, int r, int px, int py, int x, int k, int ff) {
49      if (l == r) {
50          if (~ff) {
51              mx[px][py] = mn[px][py] = k;
52          } else {
53              mx[px][py] = max(mx[ls(px)][py], mx[rs(px)][py]);
54              mn[px][py] = min(mn[ls(px)][py], mn[rs(px)][py]);
55          }
56          return;
57      }
58      int mid = (l + r) >> 1;
59      if (x <= mid) {
60          updatey(l, mid, px, ls(py), x, k, ff);
61      } else {
62          updatey(mid + 1, r, px, rs(py), x, k, ff);
63      }
64      pushup(px, py);
65  }
66
67  void updatex(int l, int r, int p, int x, int y, int k) {
68      if (l == r) {
69          updatey(1, n, p, 1, y, k, 0);
70          return;
71      }
72      int mid = (l + r) >> 1;
73      if (x <= mid) {
74          updatex(l, mid, ls(p), x, y, k);
75      } else {
76          updatex(mid + 1, r, rs(p), x, y, k);
77      }
78      updatey(1, n, p, 1, y, k, -1);
79  }
80
81  void queryy(int l, int r, int px, int py, int x, int y) {
82      if (x <= l && y >= r) {
83          mx1 = max(mx1, mx[px][py]);
84          mn1 = min(mn1, mn[px][py]);
85          return;
86      }
87      int mid = (l + r) >> 1;
88      if (x <= mid) queryy(l, mid, px, ls(py), x, y);
89      if (y >  mid) queryy(mid + 1, r, px, rs(py), x, y);
90  }
91
92  void queryx(int l, int r, int p, int x1, int y1, int x2, int y2) {
93      if (x1 <= l && x2 >= r) {
94          queryy(1, n, p, 1, y1, y2);
95          return;
96      }
97      int mid = (l + r) >> 1;
98      if (x1 <= mid) queryx(l, mid, ls(p), x1, y1, x2, y2);
99      if (x2 >  mid) queryx(mid + 1, r, rs(p), x1, y1, x2, y2);
100 }
101
102 int main() {
103     scanf("%d", &n);
104     for (int i = 1; i <= n; ++i) {
105         for (int j = 1; j <= n; ++j) {
```

```
            scanf("%d", &a[i][j]);
        }
    }
    buildx(1, n, 1); // O(n ^ 2) 建树
    scanf("%d", &q);
    int x, y, z, x1, y1, x2, y2;
    while (q--) {
        scanf("%s", s);
        if (*s == 'c') {
            scanf("%d%d%d", &x, &y, &z);
            updatex(1, n, 1, x, y, z); // O(nlognlogn)
        } else {
            scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
            mx1 = INT_MIN, mn1 = INT_MAX;
            queryx(1, n, 1, x1, y1, x2, y2); // O(nlognlogn)
            printf("%d %d\n", mx1, mn1);
        }
    }
    return 0;
}
```

3.18.2 线段树套平衡树【区间限制 + 平衡树操作】

```
// 线段树套 Treap
// 空间 : O(n logn)
// 时间 : O(n logn logn)
// 在普通平衡树的操作上加上了区间的限制
namespace Treap { // 平衡树
    struct treap {
        int ch[2], val, dat, siz, tot;
    } t[N * 20]; // 空间 O(nlogn)
    int n, opt, x, cnt, rt;
    iv pushup(int x) {
        t[x].siz = tls(x).siz + trs(x).siz + t[x].tot;
    }
    ii New(int v) {
        t[++cnt].val = v;
        t[cnt].dat = rand();
        t[cnt].siz = t[cnt].tot = 1;
        return cnt;
    }
    iv rotate(int &x, int d) {
        // x 相当于 splay 中 rotate 的 f 节点
        int p = t[x].ch[d ^ 1];
        t[x].ch[d ^ 1] = t[p].ch[d];
        t[p].ch[d] = x;
        x = p;
        pushup(t[x].ch[d]);
        pushup(x);
    }
    void insert(int &x, int v) {
        if (!x) {
            x = New(v);
            return;
        }
        if (t[x].val == v) {
            ++t[x].tot;
            ++t[x].siz;
```

```
36              return;
37          }
38          int d = v < t[x].val ? 0 : 1;
39          insert(t[x].ch[d], v);
40          if (t[t[x].ch[d]].dat > t[x].dat)
41              rotate(x, d ^ 1); // 1 顺时针转, 0 逆时针转
42          pushup(x);
43      }
44      void remove(int &x, int v) {
45          if (!x) return;
46          if (t[x].val == v) {
47              if (t[x].tot > 1) {
48                  --t[x].tot;
49                  --t[x].siz;
50                  return;
51              }
52              if (t[x].ch[0] || t[x].ch[1]) {
53                  if (!t[x].ch[1] || t[t[x].ch[0]].dat > t[t[x].ch[1]].dat) {
54                      rotate(x, 1);
55                      remove(t[x].ch[1], v);
56                  } else {
57                      rotate(x, 0);
58                      remove(t[x].ch[0], v);
59                  }
60                  pushup(x);
61              } else x = 0;
62              return;
63          }
64          int d = v < t[x].val ? 0 : 1;
65          remove(t[x].ch[d], v);
66          pushup(x);
67      }
68      // 求的是有多少个数小于 val, 排名最后需 +1
69      ii get_rk(int x, int v) {
70          int rk = 0;
71          while (x) {
72              if (v == t[x].val) {
73                  rk += t[t[x].ch[0]].siz;
74                  break;
75              }
76              if (v < t[x].val) {
77                  x = t[x].ch[0];
78              } else {
79                  rk += t[t[x].ch[0]].siz + t[x].tot;
80                  x = t[x].ch[1];
81              }
82          }
83          return rk;
84      }
85      ii get_pre(int x, int v) { // 前驱
86          int pre = -INF;
87          while (x) {
88              if (t[x].val < v) {
89                  pre = t[x].val;
90                  x = t[x].ch[1];
91              } else {
92                  x = t[x].ch[0];
93              }
94          }
```

```
95          return pre;
96      }
97      ii get_nx(int x, int v) { // 后继
98          int nx = INF;
99          while (x) {
100             if (t[x].val > v) {
101                 nx = t[x].val;
102                 x = t[x].ch[0];
103             } else {
104                 x = t[x].ch[1];
105             }
106         }
107         return nx;
108     }
109 }
110
111 namespace SEG { // 外层线段树
112     struct seg { int l, r, rt; } t[N << 2]; // rt : 内层平衡树对应的根节点
113     void build(int l, int r, int p) {
114         t[p].l = l, t[p].r = r;
115         // 对每个线段树上的节点建一颗对应区间的平衡树
116         for (int i = l; i <= r; ++i) // 区间 [l, r]
117             Treap::insert(t[p].rt, a[i]); // 共执行 nlogn 次 insert 操作
118         if (l == r) return;
119         int mid = (l + r) >> 1;
120         build(l, mid, ls(p));
121         build(mid + 1, r, rs(p));
122     }
123     // O(logn logn)
124     // 求的是有多少个数小于 val, 排名最后需 +1
125     int get_rk(int x, int y, int val, int p = 1) { // 求 [x, y] 权值 val 对应的排名
126         if (x > t[p].r || y < t[p].l) return 0;
127         if (x <= t[p].l && y >= t[p].r) return Treap::get_rk(t[p].rt, val);
128         return get_rk(x, y, val, ls(p)) + get_rk(x, y, val, rs(p));
129     }
130     // O(logn logn logn)
131     int get_val(int x, int y, int k) { // 求 [x, y] 内排名 k 对应的权值
132         int l = 0, r = 1e8; // 数据的范围 [0, 1e8]
133         // 对每个二分值求它的排名
134         while (l < r) {
135             int mid = (l + r + 1) >> 1;
136             // 向上取整往右逼近
137             // 满足 get_rk() < k 的最大二分值即为所求
138             if (get_rk(x, y, mid) < k)
139                 l = mid;
140             else
141                 r = mid - 1;
142         }
143         return r;
144     }
145     // O(logn logn)
146     // 单点修改操作: 就是先删除权值 a[x], 再添加权值 y
147     void modify(int x, int y, int p = 1) {
148         // 对覆盖到 x 的每颗平衡树都进行修改
149         Treap::remove(t[p].rt, a[x]);
150         Treap::insert(t[p].rt, y);
151         if (t[p].l == t[p].r) return;
152         int mid = (t[p].l + t[p].r) >> 1;
153         if (x <= mid)
```

```
            modify(x, y, ls(p));
        else
            modify(x, y, rs(p));
    }
    // O(logn logn)
    int get_pre(int x, int y, int v, int p = 1) { // 求前驱
        if (x > t[p].r || y < t[p].l) return -INF;
        if (x <= t[p].l && y >= t[p].r) return Treap::get_pre(t[p].rt, v);
        return max(get_pre(x, y, v, ls(p)), get_pre(x, y, v, rs(p))); // 取 max
    }
    // O(logn logn)
    int get_nx(int x, int y, int v, int p = 1) { // 求后继
        if (x > t[p].r || y < t[p].l) return INF;
        if (x <= t[p].l && y >= t[p].r) return Treap::get_nx(t[p].rt, v);
        return min(get_nx(x, y, v, ls(p)), get_nx(x, y, v, rs(p))); // 取 min
    }
}
using namespace SEG;
{
    build(1, n, 1);
    case 1: printf("%d\n", get_rk(x, y, z) + 1); // 查询 z 在 [x, y] 中的排名
    case 3: /// 单点修改
            modify(x, y);
            a[x] = y; // a[x] 也修改一下
    // 其他操作只需直接输出函数即可
}
```

3.18.3 权值线段树套线段树【带修区间桶内第 k 大】

```
/* 维护 n 个可重整数集，集合的编号从 1 到 n. 这些集合初始都是空集，有 m 个操作:
// 1 l r c: 表示将 c 加入到编号在 [l, r] 内的集合中
// 2 l r c: 表示查询编号在 [l, r] 内的集合的并集中，第 c 大的数是多少 */
#include <bits/stdc++.h>
using namespace std;

#define N 50004
#define L(x) t[x].l
#define R(x) t[x].r

typedef long long ll;

ll c[N];
int n, m, len;
int a[N], opt[N], l[N], r[N];

// 内层区间线段树 [1, n]
namespace SEG{
    struct seg {
        int l, r, tg;
        ll w;
    } t[N * 404]; // 空间 O(n logn logn)
    int tot; // 节点编号
    void insert(int &now, int x, int y, int l = 1, int r = n) {
        if (x > y) return;
        if (!now) now = ++tot;
        t[now].w += min(y, r) - max(x, l) + 1; // [x, y] 与 [l, r] 的交集长度
        if (x <= l && y >= r) {
            ++t[now].tg; // 标记永久化
```

```
30              return;
31          }
32          int mid = (l + r) >> 1;
33          if (x <= mid) insert(L(now), x, y, l, mid);
34          if (y >  mid) insert(R(now), x, y, mid + 1, r);
35      }
36      ll query(int &now, int x, int y, ll tg = 0, int l = 1, int r = n) {
37          if (x > r || y < l) return 0;
38          if (x <= l && y >= r) return t[now].w + tg * (r - l + 1); // 标记永久化
39          int mid = (l + r) >> 1;
40          return query(L(now), x, y, tg + t[now].tg, l, mid) + query(R(now), x, y, tg + t
    [now].tg, mid + 1, r);
41      }
42  }
43
44  // 外层权值线段树 [1, n]
45  namespace SEG_V {
46      struct seg { int l, r, rt; } t[N << 2];
47      int rt, tot;
48      void insert(int &now, int x, int y, int k, int l = 1, int r = len) { // 区间 [x, y],
     权值 k
49          if (!now) now = ++tot;
50          SEG::insert(R(now)t, x, y); // 加入该权值 k 下的内层区间线段树
51          if (l == r) return;
52          int mid = (l + r) >> 1;
53          if (k <= mid) insert(L(now), x, y, k, l, mid);
54          else insert(R(now), x, y, k, mid + 1, r);
55      }
56      int query(int &now, int x, int y, ll k, int l = 1, int r = len) { // 区间 [x, y], 第
     k 大
57          if (l == r) return l;
58          if (!now) now = ++tot;
59          int mid = (l + r) >> 1;
60          ll tot = SEG::query(t[R(now)].rt, x, y);
61          if (k <= tot) return query(R(now), x, y, k, mid + 1, r);
62          return query(L(now), x, y, k - tot, l, mid);
63      }
64  }
65  using namespace SEG_V;
66  int main() {
67      n = rd(), m = rd();
68      for (int i = 1; i <= m; ++i) {
69          opt[i] = rd(), l[i] = rd();
70          r[i] = rd(), c[i] = rd();
71          if (opt[i] & 1) a[++len] = c[i];
72      }
73      sort(a + 1, a + 1 + len);
74      len = unique(a + 1, a + 1 + len) - a - 1;  // 离散化
75      for (int i = 1; i <= m; ++i) {
76          if (opt[i] & 1) c[i] = lower_bound(a + 1, a + 1 + len, c[i]) - a;
77      }
78      for (int i = 1; i <= m; ++i)
79          if (opt[i] & 1) insert(rt, l[i], r[i], c[i]);
80          else printf("%d\n", a[query(rt, l[i], r[i], c[i])]);
81      return 0;
82  }
```

3.18.4 树状数组套树状数组【矩形加 + 矩形和】

```cpp
// 维护矩形加，查询矩形和
int n, m, x1, y1, x2, y2; // #define y1 yl
namespace BIT { // 二维 BIT
    int t[N][N], ti[N][N], tj[N][N], tij[N][N];
    inline int lowbit(int x) { return x & -x; }
    inline void add(int x, int y, int k) { // O(lognlogn)
        int ki = k * x, kj = k * y, kij = ki * y;
        for (int i = x; i <= n; i += lowbit(i)) {
            for (int j = y; j <= m; j += lowbit(j)) {
                t[i][j] += k, ti[i][j] += ki, tj[i][j] += kj, tij[i][j] += kij;
            }
        }
    }
    inline int ask(int x, int y) { // O(lognlogn)
        int s = 0, si = 0, sj = 0, sij = 0;
        for (int i = x; i; i -= lowbit(i)) {
            for (int j = y; j; j -= lowbit(j)) {
                s += t[i][j], si += ti[i][j], sj += tj[i][j], sij += tij[i][j];
            }
        }
        return s * (x * y + x + y + 1) - si * (y + 1) - sj * (x + 1) + sij;
    }
}
using namespace BIT;
inline void update(int x1, int y1, int x2, int y2, int k) { // 矩阵加，拆成单点修改
    add(x1, y1, k), add(x1, y2 + 1, -k), add(x2 + 1, y1, -k), add(x2 + 1, y2 + 1, k);
}
inline int query(int x1, int y1, int x2, int y2) { // 矩阵和，拆成单点查询
    return ask(x2, y2) - ask(x1 - 1, y2) - ask(x2, y1 - 1) + ask(x1 - 1, y1 - 1);
}
int main() {
    scanf("%d%d", &n, &m);
    while (~scanf("%s", s)) {
        x1 = rd(), y1 = rd(), x2 = rd(), y2 = rd();
        if (*s == 'L') { // 矩形区域修改
            update(x1, y1, x2, y2, rd());
        } else { // 询问矩形区域和
            printf("%d\n", query(x1, y1, x2, y2));
        }
    }
    return 0;
}
```

3.18.5 树状数组套树状数组【特殊题，矩形改 + 矩形最值】

```cpp
// 例题：https://ac.nowcoder.com/acm/contest/35285/K
// 区间插入线段(x, y, z)，要求查询区间 (L, R) 内完整线段的 max - min
// 插入线段对矩形 (x, 1), (up, y) 产生贡献，查询矩阵为 (1, y), (x, up)
#include <bits/stdc++.h>
using namespace std;

#define N 3003

const int up = 3000; // 上界

int n, m;
int mn[N][N], mx[N][N];

```

```cpp
inline void add(int x, int y, int w) { // 矩形修改 (x, 1), (up, y)
    for (int i = x; i; i ^= i & -i) { // 视维护的矩形决定维护前/后缀
        for (int j = y; j <= up; j += j & -j) {
            mx[i][j] = max(mx[i][j], w);
            mn[i][j] = min(mn[i][j], w);
        }
    }
}

inline int ask(int x, int y) { // 矩形查询 (1, y), (x, up)
    int mx1 = 0, mn1 = INT_MAX; // 如果初始化 mx[][], 那么 mx1 = INT_MIN
    for (int i = x; i <= up; i += i & -i) {
        for (int j = y; j; j ^= j & -j) {
            mx1 = max(mx1, mx[i][j]);
            mn1 = min(mn1, mn[i][j]);
        }
    }
    return !mx1 ? 0 : mx1 - mn1;
}

int main() {
    scanf("%d%d", &n, &m);
    memset(mn, 127, sizeof(mn));
    // memset(mx, 128, sizeof(mx)); // 视情况决定是否初始化 mx
    int opt, x, y, z;
    for (int i = 1; i <= n; ++i) {
        scanf("%d%d%d", &x, &y, &z);
        add(x, y, z);
    }
    while (m--) {
        scanf("%d%d%d", &opt, &x, &y);
        if (opt & 1) {
            scanf("%d", &z);
            add(x, y, z);
        } else {
            printf("%d\n", ask(x, y));
        }
    }
    return 0;
}
```

3.18.6 树状数组套主席树【带修区间第 k 小】

动态区间第 k 小 (带修区间第 k 小, 修改操作采用先删后加的方式实现)

```cpp
// 空间 : O(n logn logn)
// 时间 : O(n logn logn)
/* 查询排名时间复杂度优于 线段树套平衡树 (后者查询排名是 O(logn logn logn) 的)
 * 但空间复杂度会多一个 logn */
// 核心函数 add(), kth()
#define L(x) t[x].l
#define R(x) t[x].r
struct Tree {
    int l, r, w;
} t[N * 404]; // O(nlognlogn)
struct Q { // 离线询问
    int l, r, k, opt, x, y;
} q[N];
```

```
14 int n, m, opt, x, len, cnt;
15 int a[N], b[N << 1], rt[N], tl[22], tr[22];
16 // b 对所有初始值和修改后的值离散化, 空间 : O(n * n)
17 // tl, tr : O(logn) 空间
18 void update(int &now, int x, int k, int l = 1, int r = len) {
19     if (!now) now = ++cnt;
20     t[now].w += k; // 主席树的前缀和
21     if (l == r) return;
22     int mid = (l + r) >> 1;
23     if (x <= mid)
24         update(L(now), x, k, l, mid);
25     else
26         update(R(now), x, k, mid + 1, r);
27 }
28 iv add(int i, int x, int k) { // k: 1, i 位置加数;  x: -1, i 位置减数 x
29     while (i <= n) {
30         update(rt[i], x, k); // 树状数组的区间前缀和
31         i += lowbit(i);
32     }
33 }
34 // 递归版
35 // 树状数组存的是区间前缀和, 每个区间对应一颗主席树, 主席树内维护权值前缀
36 // 故将区间 [1 ~ r] 的主席树的 ls - [1 ~ l - 1] 的主席树的 ls 即得 [l, r] 的主席树的 ls
37 int query(int k, int l = 1, int r = len) {
38     if (l == r) return l;
39     int mid = (l + r) >> 1, tot = 0;
40     for (int i = 1; i <= *tr; ++i) tot += t[L(tr[i])].w; // 区间 [1 ~ r] 的主席树
41     for (int i = 1; i <= *tl; ++i) tot -= t[L(tl[i])].w; // 区间 [1 ~ l] 的主席树
42     // 差分 : 用 [1 ~ r] 的左子树 - [1, l - 1] 的左子树
43     if (k <= tot) { // 在左子树
44         for (int i = 1; i <= *tl; ++i) tl[i] = L(tl[i]);
45         for (int i = 1; i <= *tr; ++i) tr[i] = L(tr[i]);
46         return query(k, l, mid);
47     } else { // 在右子树
48         for (int i = 1; i <= *tl; ++i) tl[i] = R(tl[i]);
49         for (int i = 1; i <= *tr; ++i) tr[i] = R(tr[i]);
50         return query(k - tot, mid + 1, r); // k - tot
51     }
52 }
53 int kth(int l, int r, int k) {
54     *tl = *tr = 0;
55     for (int i =   r  ; i; i -= lowbit(i)) tr[++*tr] = rt[i]; // 把 [1 ~ r] 的根都存下
56     for (int i = l - 1; i; i -= lowbit(i)) tl[++*tl] = rt[i]; // 把 [1 ~ l - 1] 的根都存
       下
57     return query(k);
58 }
59 // 非递归版
60 int kth(int l, int r, int k) { // 非递归版, 时间上更优
61     *tl = *tr = 0;
62     for (int i = l - 1; i; i -= lowbit(i)) tl[++*tl] = rt[i];
63     for (int i =   r  ; i; i -= lowbit(i)) tr[++*tr] = rt[i];
64     l = 1, r = len;
65     while (l < r) {
66         int tot = 0, mid = (l + r) >> 1;
67         for (int i = 1; i <= *tl; ++i) tot -= t[L(tl[i])].w;
68         for (int i = 1; i <= *tr; ++i) tot += t[L(tr[i])].w;
69         if (k <= tot) {
70             for (int i = 1; i <= *tl; ++i) tl[i] = L(tl[i]);
71             for (int i = 1; i <= *tr; ++i) tr[i] = L(tr[i]);
```

```
            r = mid;
        } else {
            for (int i = 1; i <= *tl; ++i) tl[i] = R(tl[i]);
            for (int i = 1; i <= *tr; ++i) tr[i] = R(tr[i]);
            l = mid + 1;
            k -= tot;
        }
    }
    return l;
}
int main() {
    len = n = rd(), m = rd();
    for (int i = 1; i <= n; ++i) a[i] = b[i] = rd();
    for (int i = 1; i <= m; ++i) {
        scanf("%s", s);
        q[i].opt = *s == 'Q';
        if (q[i].opt) {
            q[i].l = rd(), q[i].r = rd(), q[i].k = rd();
        } else {
            q[i].x = rd();
            q[i].y = b[++len] = rd();
        }
    }
    // 离线下来进行离散化
    sort(b + 1, b + 1 + len);
    len = unique(b + 1, b + 1 + len) - b - 1;
    for (int i = 1; i <= n; ++i)
        add(i, lower_bound(b + 1, b + 1 + len, a[i]) - b, 1); // 存入树中
    for (int i = 1; i <= m; ++i) {
        if (q[i].opt) { // 查询 [l, r] 中第 k 小的数
            printf("%d\n", b[kth(q[i].l, q[i].r, q[i].k)]);
        } else { // 将 a[x] 修改为 y
            x = lower_bound(b + 1, b + 1 + len, a[q[i].x]) - b;
            add(q[i].x, x, -1); // 删数 x
            a[q[i].x] = q[i].y;
            x = lower_bound(b + 1, b + 1 + len, a[q[i].x]) - b;
            add(q[i].x, x, 1); /// 加数 y
        }
    }
    return 0;
}
```

3.18.7 树状数组套主席树【动态逆序对个数】

```
// 时空 : O(nlognlogn)
// 动态逆序对个数
// 每删一个数, 统计一次逆序对个数
#define L(x) t[x].l
#define R(x) t[x].r
struct Tree { int l, r, w; } t[N * 404]; // O(nlognlogn)
int a[N], rt[N], pos[N], tl[22], tr[22];
int n, m, x, cnt;
ll ans;
void update(int &now, int x, int k, int l = 1, int r = n) {
    if (!now) now = ++cnt;
    t[now].w += k;
    if (l == r) return;
    int mid = (l + r) >> 1;
```

```
    if (x <= mid)
        update(L(now), x, k, l, mid);
    else
        update(R(now), x, k, mid + 1, r);
}
void add(int i, int x, int k) { // k: 1, i 位置加数;  x: -1, i 位置减数 x
    while (i <= n) {
        update(rt[i], x, k);
        i += lowbit(i);
    }
}
ll query(int l, int r, int x, int d) {
    *tl = *tr = 0;
    for (int i = l - 1; i; i -= lowbit(i)) tl[++*tl] = rt[i]; // 注意 l - 1
    for (int i =   r  ; i; i -= lowbit(i)) tr[++*tr] = rt[i];
    l = 1, r = n;
    ll sum = 0;
    while (l < r) {
        int mid = (l + r) >> 1;
        if (x <= mid) {
            if (!d) { // 查询比 x 大的数的个数, 统计右区间答案
                for (int i = 1; i <= *tl; ++i) sum -= t[R(tl[i])].w;
                for (int i = 1; i <= *tr; ++i) sum += t[R(tr[i])].w;
            }
            for (int i = 1; i <= *tl; ++i) tl[i] = L(tl[i]);
            for (int i = 1; i <= *tr; ++i) tr[i] = L(tr[i]);
            r = mid;
        } else {
            if (d) { // 查询比 x 小的数的个数, 统计左区间答案
                for (int i = 1; i <= *tl; ++i) sum -= t[L(tl[i])].w;
                for (int i = 1; i <= *tr; ++i) sum += t[L(tr[i])].w;
            }
            for (int i = 1; i <= *tl; ++i) tl[i] = R(tl[i]);
            for (int i = 1; i <= *tr; ++i) tr[i] = R(tr[i]);
            l = mid + 1;
        }
    }
    return sum;
}
{
    for (int i = 1; i <= n; ++i) {
        a[i] = rd();
        pos[a[i]] = i; // 权值为 a[i] 的数的位置
        ans += query(1, i - 1, a[i], 0);
        // 逆序对增加了 i 前面比 a[i] 大的数的个数
        add(i, a[i], 1); // i 这个位置对主席树的 a[i] 位置加 1
    }
    while (--m) {
        x = rd();  // 删去权值为 x 的数
        ans -= query(1, pos[x] - 1, x, 0);
        ans -= query(pos[x] + 1, n, x, 1);
        // 每次删数逆序对减少了 pos[x] 前面比 x 大, pos[x] 后面比 x 小的数的个数
        printf("%lld\n", ans);
        add(pos[x], x, -1); // pos[x] 这个位置对主席树的 x 位置减 1
    }
}
```

3.18.8 树状数组套主席树树【带修链上第 k 大】

```
// BIT 套主席树(在线)  O(n logn logn) 或 树上带修莫队+值域分块(离线)  O(n^(5/3))
#define N 100005
const int mx = 1e8;
int n, m, k, x, y, cnt;
int h[N], a[N], rt[N], dfn[N];
int siz[N], fa[N], dep[N], son[N], top[N];
int lowbit(int x) { return x & -x; }
struct Edge { int to, nx; } e[N << 1];
struct Node { int l, r, tot; } t[N * 404];
struct Tmp {
    int n, a[N];
    void init(int i) {
        n = 0;
        while (i) {
            a[++n] = rt[i];
            i -= lowbit(i);
        }
    }
    int cal(int ans = 0) {
        for (int i = 1; i <= n; ++i) {
            ans += t[t[a[i]].l].tot; // 计算左子树 tot
        }
        return ans;
    }
    void ls() {
        for (int i = 1; i <= n; ++i) a[i] = t[a[i]].l;
    }
    void rs() {
        for (int i = 1; i <= n; ++i) a[i] = t[a[i]].r;
    }
} tl, tr, ta, tf;
void add(int u, int v) {
    static int cnt = 0;
    e[++cnt].to = v;
    e[cnt].nx = h[u];
    h[u] = cnt;
}
void dfs1(int u, int pre) {
    siz[u] = 1;
    fa[u] = pre;
    dfn[u] = ++cnt;
    dep[u] = dep[pre] + 1;
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (v != pre) {
            dfs1(v, u);
            siz[u] += siz[v];
            if (siz[son[u]] < siz[v]) {
                son[u] = v;
            }
        }
    }
}
void dfs2(int u, int tpf) {
    top[u] = tpf;
    if (!son[u]) return;
    dfs2(son[u], tpf);
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
```

```
60          if (v != fa[u] && v != son[u]) {
61              dfs2(v, v);
62          }
63      }
64  }
65  int LCA(int x, int y) {
66      while (top[x] ^ top[y]) {
67          dep[top[x]] < dep[top[y]] ? y = fa[top[y]] : x = fa[top[x]];
68      }
69      return dep[x] > dep[y] ? y : x;
70  }
71  void update(int &now, int x, int k, int l = 1, int r =  mx) {
72      if (!now) now = ++cnt;
73      t[now].tot += k;
74      if (l == r) return;
75      int mid = (l + r) >> 1;
76      if (x <= mid) {
77          update(t[now].l, x, k, l, mid);
78      } else {
79          update(t[now].r, x, k, mid + 1, r);
80      }
81  }
82  void add(int i, int x, int k) {
83      while (i <= n) {
84          update(rt[i], x, k);
85          i += lowbit(i);
86      }
87  }
88  int query(int k) { // 查第 k 小
89      int l = 1, r = mx;
90      while (l < r) {
91          int mid = (l + r) >> 1;
92          int tot = tl.cal() + tr.cal() - ta.cal() - tf.cal();
93          if (k <= tot) { // 注意是 <=
94              tl.ls(), tr.ls(), ta.ls(), tf.ls();
95              r = mid;
96          } else {
97              tl.rs(), tr.rs(), ta.rs(), tf.rs();
98              l = mid + 1;
99              k -= tot; //
100         }
101     }
102     return l; //
103 }
104 int main() {
105     n = rd(), m = rd();
106     for (int i = 1; i <= n; ++i) a[i] = rd();
107     for (int i = 1; i <  n; ++i) {
108         x = rd(), y = rd();
109         add(x, y), add(y, x);
110     }
111     dfs1(1, 0), dfs2(1, 1);
112     for (int i = 1; i <= n; ++i) { // 对 dfn 序进行可持续化
113         add(dfn[i], a[i], 1); // 对 i 子树内的 a[i] 加 1
114         add(dfn[i] + siz[i], a[i], -1); // i 子树外的 a[i] 不用加 1 , 要减去前面加的 1
115     }
116     while (m--) {
117         k = rd(), x = rd(), y = rd();
118         if (k) { // 求 x - y 链上的第 k 大
```

```
            int lca = LCA(x, y), flca = fa[lca];
            int len = dep[x] + dep[y] - dep[lca] - dep[flca];
            if (k > len) { puts("invalid request!"); continue; } // 不存在
            k = len - k + 1; // 转为求第 k 小
            tl.init(dfn[x]), tr.init(dfn[y]); // 把 rt[] 先提出来
            ta.init(dfn[lca]), tf.init(dfn[flca]);
            printf("%d\n", query(k));
        } else { // 修改 a[x] 为 y
            add(dfn[x], a[x], -1);
            add(dfn[x] + siz[x], a[x], 1);
            a[x] = y;
            add(dfn[x], a[x], 1);
            add(dfn[x] + siz[x], a[x], -1);
        }
    }
    return 0;
}
```

3.19 树链剖分

3.19.1 点权

```
struct Edge { int to, nx; } e[M << 1];
int n, x, y, z, cnt, tot;
int h[N], w[N], wt[N], t[N << 2], tag[N << 2];
int dep[N], siz[N], top[N], son[N], fa[N], id[N];
void dfs1(int u, int pre) {
    dep[u] = dep[pre] + 1, fa[u] = pre, siz[u] = 1;
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (v != pre) {
            dfs1(v, u);
            siz[u] += siz[v];
            if (siz[son[u]] < siz[v]) son[u] = v;
        }
}
void dfs2(int u, int tpf) {
    top[u] = tpf, wt[id[u] = ++tot] = w[u];
    if (!son[u]) return;
    dfs2(son[u], tpf);
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (v != fa[u] && v != son[u]) dfs2(v, v);
    }
}
void build(int l, int r, int p) {
    if (l == r) return t[p] = wt[l], void(); // 线段树部分以 wt[] 来建树
    /* */
}
void updRange(int x, int y, int z, int q) { // 用 z 修改 x 到 y 路径上的权值
    while (top[x] != top[y]) {
        if (dep[top[x]] < dep[top[y]]) swap(x, y);
        update(1, n, 1, id[top[x]], id[x], z, q);
        x = fa[top[x]];
    }
    if (dep[x] > dep[y]) swap(x, y);
    update(1, n, 1, id[x], id[y], z, q);
}
int qRange(int x, int y) { // 求 x 到 y 路径上的权值
```

```
    int ans = 0;
    while (top[x] ^ top[y]) {
        if (dep[top[x]] < dep[top[y]]) swap(x, y);
        ans += query(1, n, 1, id[top[x]], id[x]);
        x = fa[top[x]];
    }
    if (dep[x] > dep[y]) swap(x, y);
    ans += query(1, n, 1, id[x], id[y], z, q);
    return ans;
}
void updSon(int x, int z) { // 用 z 修改 x 子树
    update(1, n, 1, id[x], id[x] + siz[x] - 1, z);
}
int qSon(int x, int y, int z) { // 求 x 子树的权值
    return query(1, n, 1, id[x], id[x] + siz[x] - 1]);
}
```

3.19.2 边权

```
struct Node {
    int to, nx, w; // 维护边权
} e[N << 1];
int n, x, y, z, cnt, tot;
int h[N], w[N], wt[N], t[N << 2], tag[N << 2];
int dep[N], siz[N], top[N], son[N], fa[N], id[N];
void dfs1(int u, int pre) {
    dep[u] = dep[pre] + 1, fa[u] = pre, siz[u] = 1;
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (v != pre) {
            dfs1(v, u);
            w[v] = e[i].w; // 边权作为儿子的点权
            siz[u] += siz[v];
            if (siz[son[u]] < siz[v]) son[u] = v;
        }
    }
}
void dfs2(int u, int tpf) { /* 与点权版一致 */ }
void updRange(int x, int y, int z) {
    while (top[x] != top[y]) {
        if (dep[top[x]] < dep[top[y]]) swap(x, y);
        update(1, n, 1, id[top[x]], id[x], z);
        x = fa[top[x]];
    }
    if (x == y) return; // 边权版需增加该判断
    if (dep[x] > dep[y]) swap(x, y);
    update(1, n, 1, id[son[x]], id[y], z); // id[son[x]] 边权与点权区别
}
void updSon(int x, int z) { // 用 z 修改 x 子树
    update(1, n, 1, id[son[x]], id[x] + siz[x] - 1, z); // id[son[x]] 边权与点权区别
}
// qRange 与 qSon 对 updRange 和 updSon 进行简单修改即可
```

3.20 最近公共祖先 (LCA)

3.20.1 倍增版

```cpp
vector<int> e[N];
int fa[N][22], dep[N], lg[N];
void build(int u, int pre) { // build(rt, 0)
    fa[u][0] = pre;
    dep[u] = dep[pre] + 1;
    for (int i = 1; i <= lg[dep[u]]; ++i)
        fa[u][i] = fa[fa[u][i - 1]][i - 1];
    for (auto &v : e[u]) if (v != pre) build(v, u);
}
int LCA(int x, int y) { // lca = LCA(x, y)
    if (dep[x] < dep[y]) swap(x, y);
    while (dep[x] > dep[y]) x = fa[x][lg[dep[x] - dep[y]]];
    if (x == y) return x;
    for (int i = lg[dep[x]]; i >= 0; --i) // i >= 0
        if (fa[x][i] != fa[y][i]) {
            x = fa[x][i], y = fa[y][i];
        }
    return fa[x][0];
}
void init() { // lg[] 初始化
    *lg = -1;
    for (int i = 1; i <= n; ++i) lg[i] = lg[i >> 1] + 1;
}
```

3.20.2 树剖版

```cpp
// 先树剖预处理
void dfs1(int u, int pre);
void dfs2(int u, int tpf);
int LCA(int x, int y) {
    while (top[x] ^ top[y]) {
        dep[top[x]] < dep[top[y]] ? y = fa[top[y]] : x = fa[top[x]];
    }
    return dep[x] > dep[y] ? dep[y] : dep[x];
}
```

3.20.3 Tarjan 版（离线）

```cpp
// 离线算法
struct { int u, v, lca; } q[N << 1]; // 离线询问
int vis[N], pre[N], h[N];
vector<int> e[N]; // 边
int find(int x) {
    return pre[x] == x ? x : pre[x] = find(pre[x]);
}
void merge(int x, int y) {
    x = find(x), y = find(y);
    if (x ^ y) pre[x] = y;
}
void init() {
    fill(h, 0, (h + 1) << 2);
    for (int i = 1; i <= n; ++i) pre[i] = i;
}
void Tarjan(int u) {
    vis[u] = 1;
    for (auto &v : e[u]) if (!vis[v]) Tarjan(v), merge(u, v);
    for (int i = h[u]; i; i = q[i].nx) {
        int v = q[i].to;
```

```
        if (vis[v]) q[i].lca = q[i ^ 1].lca = find(v); // 计算出每个询问的 LCA
    }
}
```

3.20.4 LCT 版

```
// 详见 LCT 分类中的求 LCA
```

3.21 树上 k 级祖先

```
// 求 x 的第 k 级祖先
// 重链剖分版 O(n) - O(logn)
// 先树链剖分预处理
void dfs1(int u, int pre);
void dfs2(int u, int tpf) {
    top[u] = tpf;
    id[u] = ++tot;
    pos[tot] = u; // 记录位置
    /*...*/
}
int ask(int x, int k) {
    while (k >= id[x] - id[top[x]] + 1 && fa[x]) {
        k -= id[x] - id[top[x]] + 1;
        x = fa[top[x]];
    }
    return pos[id[x] - k];
}

// 长链剖分版 O(nlogn) - O(1)
// mxd : 当前节点的儿子最大深度, son : 长儿子
// 随机数据下比重链剖分版慢 1 倍, 好像非随机长链剖分也比重链剖分慢?
void dfs1(int u, int pre) {
    mxd[u] = dep[u] = dep[pre] + 1;
    fa[u][0] = pre;
    for (int i = 1; i <= lg[dep[u]]; ++i)
        fa[u][i] = fa[fa[u][i - 1]][i - 1];
    for (auto &v : e[u]) {
        if (v != pre) {
            dfs1(v, u);
            if (mxd[u] < mxd[v]) {
                mxd[u] = mxd[v];
                son[u] = v;
            }
        }
    }
}
void dfs2(int u, int tpf) {
    top[u] = tpf;
    if (u == tpf) { // 在每条链的顶点记录在它上面和下面的节点
        for (int i = 0, x = u; i <= mxd[u] - dep[u]; ++i)
            up[u].push_back(x), x = fa[x][0];
        for (int i = 0, x = u; i <= mxd[u] - dep[u]; ++i)
            down[u].push_back(x), x = son[x];
    }
    if (!son[u]) return;
    dfs2(son[u], tpf);
    for (auto &v : e[u]) if (v != son[u] && v != fa[u][0]) dfs2(v, v);
```

```
}
inline int ask(int x, int k) {
    if (!k) return x;
    x = fa[x][lg[k]], k -= 1 << lg[k]; // 先用倍增往上跳
    k -= dep[x] - dep[top[x]], x = top[x]; // 再跳到所在链顶端
    return k >= 0 ? up[x][k] : down[x][-k]; // 最后用链顶的信息确定位置
}
```

3.22 树同构

```
/* 对于两个树 T1 和 T2，如果能够把树 T1 的所有点重新标号，
 * 使得树 T1 和树 T2 完全相同，那么这两个树是同构的. 也就是说，它们具有相同的形态.
 * 现在，给你 M 个无根树，请你把它们按同构关系分成若干个等价类. */
int x; uint ans[N][N]; // ans[i][j] : i 树以 j 为根的 Hash 值
uint ELF_Hash(int u, int pre) {
    uint st[N], hs = 1, t = 0;
    for (int i = h[u]; i; i = e[i].nx)
        if (e[i].to != pre)
            st[++t] = ELF_Hash(e[i].to, u);
    sort(st + 1, st + 1 + t); // 对子树 Hash 值排序
    for (int i = 1; i <= t; ++i) { // ELF Hash
        hs = (hs << 4) + st[i];
        (x = hs & 0xf0000000) && (hs ^= (x >> 24), hs &= ~x);
    }
    return (hs & 0x7fffffff);
}
int main() {
    scanf("%d", &m);
    for (int i = 1; i <= m; ++i) {
        memset(h, 0, sizeof(h));
        scanf("%d", &n);
        cnt = 0;
        for (int j = 1; j <= n; ++j) {
            scanf("%d", &x);
            if (x) add(x, j), add(j, x);
        }
        for (int j = 1; j <= n; ++j)
            ans[i][j] = ELF_Hash(j, 0);
        sort(ans[i] + 1, ans[i] + 1 + n); // 对各根的 Hash 值排序
        for (int j = 1; j <= i; ++j) {
            int k = 0;
            while (k++ <= n && ans[i][k] == ans[j][k]);
            if (k > n) { printf("%d\n", j); break; }
        }
    }
    return 0;
}
```

3.23 树的重心

3.23.1 重心

```
void dfs(int u, int pre) {
    size[u] = 1, mx[u] = 0;
    for (auto &v : e[u]) {
        if(v != pre)  {
            dfs(v, u);
            size[u] += size[v];
```

```
            mx[u] = max(mx[u], size[v]);
        }
        mx[u] = max(mx[u], n - size[u]);
        //
        if(mx[u] <= n / 2) rt[*rt] = u; // 可能有两个重心
        // 或者求任意一个重心可将以上 if 语句改为以下语句
        if (mx[u] < mx[rt]) rt = u; //
    }
}
```

3.23.2 带权重心

```
// https://www.luogu.com.cn/problem/P1364
int w[N], dp[N];
void dfs(int u, int pre, int d) {
    dp[1] += d * w[u];
    for (auto &v : e[u]) {
        if (v != pre) dfs(v, u, d + 1);
        w[u] += w[v];
    }
}

void dfs1(int u, int pre) {
    ans = ans > dp[u] ? dp[u] : ans;
    for (auto &v : e[u])
        if (v != pre) {
            dp[v] = dp[u] + w[1] - (w[v] << 1);
            dfs1(v, u);
        }
}
dfs(1, 0, 0), dfs1(1, 0);
```

3.24 K-D Tree

```
// K-D Tree
// 空间: O(n)
// 时间: 建树, 插入 O(nlogn), 单次查询 O(pow(n, (k - 1) / k))
// https://www.luogu.com.cn/problem/P4148
// 操作: 插入一个点; 查询矩阵权值和
// 强制在线, 空间限制
#define L t[o].l
#define R t[o].r
#define y1 yl
const double al = 0.75; // 平衡因子, 一般设定为 0.6 ~ 0.9
int rub[N]; // 回收栈
int n, K, rt, rb, cnt, lans, opt;
struct Point {
    int x[2]; // 横纵坐标值
    ll w; // 权值, 变量顺序不能反
    bool operator < (const Point &b) const {
        return x[K] < b.x[K]; // 以第 K 维的坐标值来排序
    }
} p[N];
struct Node {
    ll sum;
    Point p;
    int l, r, siz;
```

```
    int mx[2], mn[2]; // 第 i 维区间的上下界
} t[N];
// 建立一个新节点
inline int New() { return rb ? rub[rb--] : ++cnt; }
// 更新子树最大/最小值，子树大小，权值和
inline void pushup(int o) {
    for (int i = 0; i <= 1; ++i) {
        t[o].mn[i] = min(t[o].p.x[i], min(t[L].mn[i], t[R].mn[i]));
        t[o].mx[i] = max(t[o].p.x[i], max(t[L].mx[i], t[R].mx[i]));
    }
    t[o].sum = t[L].sum + t[R].sum + t[o].p.w;
    t[o].siz = t[L].siz + t[R].siz + 1;
}
// 把树还原成序列
void pia(int o, int lsum) {
    if (L) pia(L, lsum);
    p[t[L].siz + lsum + 1] = t[o].p; // 将节点存为序列
    rub[++rb] = o; // 回收节点编号
    if (R) pia(R, t[L].siz + lsum + 1);
}
// 通过序列建树
int rebuild(int l, int r, int k) { // O(nlogn)
    if (l > r) return 0;
    K = k; // 令 p[] 按照第 K 维排序
    int mid = (l + r) >> 1, o = New();
    nth_element(p + l, p + mid, p + r + 1); // 按中位数划分
    t[o].p = p[mid];
    L = rebuild(l, mid - 1, k ^ 1); // 按照不同维度交替建树
    R = rebuild(mid + 1, r, k ^ 1);
    pushup(o);
    return o;
}
// 检查树是否平衡，不平衡则重建
inline void check(int &o, int k) {
    if (t[o].siz * al < t[L].siz || t[o].siz * al < t[R].siz) { // 判断平衡
        pia(o, 0); // 转化为序列
        o = rebuild(1, t[o].siz, k); // 重建这颗子树
    }
}
// 插入
void insert(int &o, Point tmp, int k) {
    if (!o) { // 插入点
        o = New();
        L = R = 0;
        t[o].p = tmp;
        pushup(o);
        return;
    }
    if (tmp.x[k] <= t[o].p.x[k]) // 按照坐标值插入
        insert(L, tmp, k ^ 1);
    else
        insert(R, tmp, k ^ 1);
    pushup(o); // 先更新信息
    check(o, k); // 然后检查检查平衡
}
// x : 查询的范围，X : 当前节点覆盖的范围，X 要在 x 里面
// 检查是否在查询矩形内
inline int in(int x1, int y1, int x2, int y2, int X1, int Y1, int X2, int Y2) {
    return x1 <= X1 && y1 <= Y1 && x2 >= X2 && y2 >= Y2;
```

```
}
// 检查是否在查询矩形外
inline int out(int x1, int y1, int x2, int y2, int X1, int Y1, int X2, int Y2) {
    return x1 > X2 || y1 > Y2 || x2 < X1 || y2 < Y1;
}
// 查询矩阵和
ll query(int o, int x1, int y1, int x2, int y2) {
    if (!o) return 0;
    if (out(x1, y1, x2, y2, t[o].mn[0], t[o].mn[1], t[o].mx[0], t[o].mx[1])) return 0;
    if ( in(x1, y1, x2, y2, t[o].mn[0], t[o].mn[1], t[o].mx[0], t[o].mx[1])) return t[o
    ].sum;
    ll ans = 0;
    if (in(x1, y1, x2, y2, t[o].p.x[0], t[o].p.x[1], t[o].p.x[0], t[o].p.x[1]))
        ans = t[o].p.w; // 判断当前点是否在查询矩阵内
    return ans + query(L, x1, y1, x2, y2) + query(R, x1, y1, x2, y2);
}
{
    t[0].mn[0] = row_max, t[0].mn[1] = col_max;
    t[0].mx[0] = row_min, t[0].mx[1] = col_min;
    if (opt & 1) {
        insert(rt, (Point) {x, y, A}, 0);
    } else {
        // (x1, y1) 左下, (x2, y2) 右上
        lans = query(rt, x1, y1, x2, y2);
    }
}
```

3.24.1 二维最近距离

```
// 小常数 K-D Tree 写法
// 空间回收 -> n, 不回收 -> n * 3 ~ n * 4
const int INF = 2e9;
const double al = 0.75;
int rub[N], tmp[N];
int n, m, opt, x, y, K, rb, rt, tot, cnt, ans, now;
struct Point { int x[2]; } p[N];
struct Node {
    int l, r, k, id, siz;
    int mn[2], mx[2];
    Point p;
} t[N];
inline bool cmp(int i, int j) { return p[i].x[K] < p[j].x[K]; }
inline int New() { return rb ? rub[rb--] : ++cnt; }
inline void pushup(int o) {
    t[o].mn[0] = min(t[o].p.x[0], min(t[L].mn[0], t[R].mn[0]));
    t[o].mx[0] = max(t[o].p.x[0], max(t[L].mx[0], t[R].mx[0]));
    t[o].mn[1] = min(t[o].p.x[1], min(t[L].mn[1], t[R].mn[1]));
    t[o].mx[1] = max(t[o].p.x[1], max(t[L].mx[1], t[R].mx[1]));
    t[o].siz = t[L].siz + t[R].siz + 1;
}
void pia(int o) {
    if (L) pia(L);
    tmp[++tot] = t[o].id;
    rub[++rb] = o; //
    if (R) pia(R);
}
int build(int l, int r, int k) {
    if (l > r) return 0;
```

```
30      K = k;
31      int mid = (l + r) >> 1, o = New();
32      nth_element(tmp + l, tmp + mid, tmp + r + 1, cmp);
33      t[o].p = p[tmp[mid]];
34      t[o].id = tmp[mid];
35      t[o].k = k;
36      L = build(l, mid - 1, k ^ 1);
37      R = build(mid + 1, r, k ^ 1);
38      pushup(o);
39      return o;
40  }
41  inline void check(int &o) {
42      if (t[o].siz * al < max(t[L].siz, t[R].siz)) {
43          tot = 0, pia(o);
44          o = build(1, tot, 0);
45      }
46  }
47  short seed = 2333; // 卡常
48  inline short rnd() { return seed = (seed * 7 + 13) ^ (seed / 13 - 7); }
49  void insert(int &o, int id) {
50      if (!o) {
51          o = New();
52          L = R = 0;
53          t[o].id = id;
54          t[o].p = p[id];
55          t[o].k = rnd() & 1;
56          pushup(o);
57          return;
58      }
59      K = t[o].k;
60      if (cmp(id, t[o].id)) insert(L, id);
61      else insert(R, id);
62      pushup(o), check(o);
63  }
64  int cal(int o, int id) {
65      int ans = 0;
66      ans += max(0, p[id].x[0] - t[o].mx[0]) + max(0, t[o].mn[0] - p[id].x[0]);
67      ans += max(0, p[id].x[1] - t[o].mx[1]) + max(0, t[o].mn[1] - p[id].x[1]);
68      return ans;
69  }
70  int query(int o, int id) {
71      if (id ^ t[o].id)
72          ans = min(ans, abs(t[o].p.x[0] - p[id].x[0]) + abs(t[o].p.x[1] - p[id].x[1]));
73      int ld = L ? cal(L, id) : INF;
74      int rd = R ? cal(R, id) : INF;
75      if (ld < rd) {
76          if (ld < ans) query(L, id);
77          if (rd < ans) query(R, id);
78      } else {
79          if (rd < ans) query(R, id);
80          if (ld < ans) query(L, id);
81      }
82      return ans;
83  }
84  int main() {
85      n = rd(), m = rd();
86      t[0].mn[0] = t[0].mn[1] = INF;
87      for (int i = 1; i <= n; ++i) {
88          tmp[i] = i;
```

```
89          p[i].x[0] = rd();
90          p[i].x[1] = rd();
91      }
92      rt = build(1, n, 0); // 直接建树
93      now = n;
94      while (m--) {
95          opt = rd(), x = rd(), y = rd();
96          p[++now] = (Point) {x, y};
97          if (opt & 1) {
98              insert(rt, now);
99          } else {
100             ans = INF;
101             query(rt, now);
102             printf("%d\n", ans);
103         }
104     }
105     return 0;
106 }
```

3.25 树上启发式合并 (DSU)

```
1  // O(nlogn) * max(O(add), O(del), O(getans))
2  int siz[N], son[N], st[N], ed[N], ans[N], rdfn[N], dfn;
3  void dfs1(int u, int pre) {
4      siz[u] = 1, st[u] = ++dfn, rdfn[dfn] = u;
5      for (int &v : e[u]) {
6          if (v != pre) {
7              dfs1(v, u);
8              siz[u] += siz[v];
9              if (siz[v] > siz[son[u]]) son[u] = v;
10         }
11     }
12     ed[u] = dfn;
13 }
14 void dfs2(int u, int pre, int keep) { // keep 控制是否保留修改
15     // 计算轻儿子的答案
16     for (int &v : e[u]) if (v != pre && v != son[u]) dfs2(v, u, 0);
17     // 计算重儿子答案并保留计算过程中的修改(用于继承)
18     if (son[u]) dfs2(son[u], u, 1);
19     for (int &v : e[u]) {
20         if (v != pre && v != son[u]) {
21             // 子树结点的 dfs 序构成一段连续区间, 可以直接遍历
22             for (int i = st[v]; i <= ed[v]; i++) add(rdfn[i]); // 修改
23         }
24     }
25     add(u); // 修改
26     ans[u] = getAns(); // 记录答案
27     // 轻儿子不保留修改, 回溯操作
28     if (!keep) for (int i = st[u]; i <= ed[u]; i++) del(rdfn[i]); // 回溯
29 }
30
31 {
32     dfs1(1, 0), dfs2(1, 0, 0);
33 }
```

4 Graph Theory

4.1 最短路

技巧

平面图转对偶图。平面图最小割 = 对偶图最短路。

分层图最短路。$dis[], vis[]$ 改为多维数组记录当前状态下的 dis, vis，队列多存储下状态，即可进行转移。

状态压缩最短路。将状态用二进制存储下来，然后跑分层图最短路即可。

同余最短路。

4.1.1 Dijkstra

```
namespace Dij {
    typedef pair<int, int> p;
    struct Node {
        int to, nx, dis;
    } e[N << 1];
    int n, m, s, t, cnt;
    bool vis[N];
    int h[N], dis[N], pre[N]; // pre : 存路径
    void init(int n) { memset(h, 0, sizeof(h)), cnt = 0; }
    void add(int u, int v, int dis) { // 无向图需调用两次
        e[++cnt] = (Edge) {v, h[u], dis}, h[u] = cnt;
    }

    // 暴力 O(n * n)
    void dij(int s, int t) {
        memset(dis, 0x3f, sizeof(dis));
        memset(vis, 0, sizeof(vis));
        dis[s] = 0;
        for (int k = 1; k <= n; ++k) {
            int u = 0, mn = 0x3f3f3f3f;
            for (int i = 1; i <= n; ++i) {
                if (!vis[i] && dis[i] < mn) {
                    mn = dis[i], u = i;
                }
            }
            vis[u] = 1;
            for (int i = h[u]; i; i = e[i].nx) {
                int v = e[i].to;
                if (dis[v] > dis[u] + e[i].dis) {
                    dis[v] = dis[u] + e[i].dis;
                }
            }
        }
    }

    // 优先队列优化 O(mlogm)
    void dij(int s, int t) { // s, t : 起点, 终点
        priority_queue<p, vector<p>, greater<p> > que;
        memset(dis, 0x3f, sizeof(dis));
        memset(vis, 0, sizeof(vis));
        que.push(p(0, s));
        dis[s] = 0;
        while (!que.empty()) {
            int u = que.top().second;
```

```
            que.pop();
            if (vis[u]) continue;
            vis[u] = 1;
            for (int i = h[u]; i; i = e[i].nx) {
                int v = e[i].to;
                if (dis[v] > dis[u] + e[i].dis) {
                    dis[v] = dis[u] + e[i].dis;
                    pre[v] = u; // 路径
                    if (!vis[v] && v != t) {
                        que.push(p(dis[v], v));
                    }
                }
            }
        }
    }

    void build(int l, int r, int p) {
        L = l, R = r;
        if (L == R) { t[p].d = l; return; }
        int mid = (l + r) >> 1;
        build(l, mid, ls(p));
        build(mid + 1, r, rs(p));
    }
    inline int cmp(int x, int y) { return dis[x] < dis[y] ? x : y; }
    void update(int x, int p = 1) {
        if (L == R) return;
        update(x, x <= tls(p).r ? ls(p) : rs(p));
        t[p].d = cmp(tls(p).d, trs(p).d);
    }
    // 线段树优化 O(mlogn)
    inline void dij(int s) {
        memset(dis, 0x3f, sizeof(dis));
        build(1, n, 1), dis[s] = 0;
        for (int u = s, k = 1; k <= n; ++k) {
            const int disu = dis[u];
            ans[u] = disu;
            dis[u] = INF, update(u);
            for (int i = h[u]; i; i = e[i].nx) {
                int v = e[i].to;
                if (dis[v] < INF && dis[v] > disu + e[i].dis) {
                    dis[v] = disu + e[i].dis, update(v);
                }
            }
            u = t[1].d;
        }
    }
}
```

4.1.2 Bellman-Ford

```
// 有边数限制的最短路
const int INF = 0x3f3f3f3f;
struct Edge { int u, v, dis; } e[M];
int n, m, k; // 节点数, 边数, 限制经过的边数
int dis[N], dis1[N];
// s, t : 起点, 终点 , m : 边数 , k : 限定经过边数
inline int Bellman_Ford(int s, int t, int m, int k) {
    memset(dis, 0x3f, sizeof(dis)), dis[s] = 0;
```

```
    for (re i = 1; i <= k; ++i) { // 每循环一次, 松弛一次
        memcpy(dis1, dis, sizeof(dis)); // cpy
        for (re j = 1; j <= m; ++j) {
            int u = e[j].u, v = e[j].v;
            if (dis1[u] < INF && dis[v] > dis1[u] + e[j].dis) {
                dis[v] = dis1[u] + e[j].dis;
            }
        }
    }
    if (dis[t] < INF) return dis[t]; // 最短路
    return -1;
}
```

4.1.3 Floyd

```
void init(int n) {
    memset(dis, 0x3f, sizeof(dis));
    for (int i = 1; i <= n; ++i) dis[i][i] = 0;
}
void add(int u, int v, int d) { dis[u][v] = d; } // 无向图需 add 两次
void Floyd(int n) {
    for (int k = 1; k <= n; ++k)
        for (int i = 1; i <= n; ++i)
            if (i != k) // 节省时间
            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];
}
```

4.1.4 SPFA 判负环

```
// O(n * m)
bool spfa(int s) {
    memset(dis, 0x3f, sizeof(dis));
    memset(num, 0, sizeof(num));
    memset(vis, 0, sizeof(vis));
    queue<int> que, dis[s] = 0;
    while (!que.empty()) {
        int u = que.front();
        que.pop(), vis[u] = 0;
        for (int i = h[u]; i; i = e[i].nx) {
            int v = e[i].to;
            if (dis[v] > dis[u] + e[i].dis) {
                dis[v] = dis[u] + e[i].dis;
                if (!vis[v]) {
                    num[v] = num[u] + 1; // 经过的节点数
                    if (num[v] > n) return true; // 有负环
                    vis[v] = 1, que.push(v);
                }
            }
        }
    }
    return false; // 无负环
}
```

4.1.5 SPFA

```cpp
// O(n * m)
// 有负权，判断能否从 s 到 t
int spfa(int s, int t) {
    memset(dis, 0x3f, sizeof(dis));
    memset(vis, 0, sizeof(vis));
    memset(num, 0, sizeof(num));
    queue<int>que;
    que.push(s);
    dis[s] = 0;
    while (!que.empty()) {
        Ci u = que.front();
        que.pop(), vis[u] = 0;
        for (int i = h[u]; i; i = e[i].nx) {
            int v = e[i].to;
            if (dis[v] > dis[u] + e[i].dis) {
                dis[v] = dis[u] + e[i].dis;
                if (vis[v]) continue;
                vis[v] = 1;
                num[v] = num[u] + 1;
                if (num[v] > n) continue; // 负环
                if (v != t) que.push(v);
            }
        }
    }
    return dis[t] == INF ? -1 : dis[t];
}
```

4.1.6 SPFA(SLF 优化)

```cpp
// 判负环不可加SLF优化，可被卡成指数级别复杂度！！！
void spfa(int s) {
    memset(dis, 0x3f, sizeof(dis));
    memset(vis, 0, sizeof(vis));
    deque<int> que; // 双端队列优化
    que.push_back(s);
    dis[s] = 0;
    while (!que.empty()) {
        int u = que.front();
        que.pop_front();
        vis[u] = 0;
        for (re i = h[u]; i; i = e[i].nx) {
            int v = e[i].to;
            if (dis[v] > dis[u] + e[i].dis) {
                dis[v] = dis[u] + e[i].dis;
                if (!vis[v]) {
                    vis[v] = 1;
                    if (!que.empty() && dis[v] < dis[que.front()]) {
                        que.push_front(v);
                    } else {
                        que.push_back(v);
                    }
                }
            }
        }
    }
}
```

4.1.7 Johnson 全源最短路

```cpp
// 时间复杂度 O(n * m * logm)
// 可求出每对结点之间的最短路
// 但时间复杂度可比 O(n ^ 3) 的 Floyd 优秀
// 适用于无负环图，可有负权
/* 对于负权图的处理：
 * 1. SPFA 预处理负权
 * 2. 跑 n 遍堆优化 Dijkstra */
#define N maxn
#define M (maxn + maxm)
// 有向图邻接表需开 (maxn + maxm) 大小，其中 maxn 用于超级源点连边
struct Edge { int to, nx, dis; } e[N + M]; // 注意 N + M
int n, m, cnt;
ll dis[N], hh[N];
int h[N], vis[N], num[N];
int spfa(int s) {
    memset(hh, 0x3f, sizeof(hh));
    hh[s] = 0, vis[s] = 1;
    queue<int> que;
    que.push(s);
    while (!que.empty()) {
        int u = que.front();
        que.pop(), vis[u] = 0;
        for (int i = h[u]; i; i = e[i].nx) {
            int v = e[i].to;
            if (hh[v] > hh[u] + e[i].dis) {
                hh[v] = hh[u] + e[i].dis;
                if (!vis[v]) {
                    vis[v] = 1;
                    num[v] = num[u] + 1;
                    if (num[v] > n) return 0; // 负环
                    que.push(v);
                }
            }
        }
    }
    return 1;
}

void dij(int s) {
    priority_queue<p, vector<p>, greater<p> > que;
    for (int i = 1; i <= n; ++i) dis[i] = INF;
    memset(vis, 0, sizeof(vis));
    que.push(p(0, s));
    dis[s] = 0;
    while (!que.empty()) {
        int u = que.top().second;
        que.pop();
        if (vis[u]) continue;
        vis[u] = 1;
        for (int i = h[u]; i; i = e[i].nx) {
            int v = e[i].to;
            if (dis[v] > dis[u] + e[i].dis) {
                dis[v] = dis[u] + e[i].dis;
                if (!vis[v]) que.push(p(dis[v], v));
            }
        }
    }
```

```
}
int main() {
    n = rd(), m = rd();
    while (m--) {
        u = rd(), v = rd(), w = rd();
        add(u, v, w); // 有向图
    }
    for (int i = 1; i <= n; ++i) add(0, i, 0); // 超级源点
    if (!spfa(0)) puts("-1"), exit(0); // 有负环
    for (int u = 1; u <= n; ++u)
        for (int i = h[u]; i; i = e[i].nx)
            e[i].dis += hh[u] - hh[e[i].to]; // 处理负权
    for (int i = 1; i <= n; ++i) {
        dij(i);
        for (int j = 1; j <= n; ++j)
            if (dis[j] == INF) ans[i][j] = -1; // 不可达
            else ans[i][j] = dis[j] + hh[j] - hh[i]; // 减回去得到真实距离
    }
    return 0;
}
```

4.1.8 线段树优化建图

```
/* 要求连边
 * 1: u -> v
 * 2: u -> [l, r]
 * 3: [l, r] -> v
 */
ll dis[N * 3]; // 原始节点 N + 线段树节点 N * 2
int h[N * 3], vis[N * 3];
int n, m, s, opt, x, y, w, l, r, cnt;
namespace Graph {
    struct Edge { int to, nx, dis; } e[N * 40];
    // build O(N * 4) + update O(N * 31), 空间开到 35 以上最保险
    inline void add(int u, int v, int d) {
        static int cnt = 0;
        e[++cnt] = (Edge) {v, h[u], dis}, h[u] = cnt;
    }
    inline void dij() {} // 常规 dijkstra
}
using namespace Graph;

namespace Seg {
    struct Node { int l, r, in, out; } t[N << 2];
    void build(int l, int r, int p) {
        L = l, R = r;
        if (l == r) { t[p].in = t[p].out = l; return; }
        int mid = (l + r) >> 1;
        build(l, mid, ls(p));
        build(mid + 1, r, rs(p));
        t[p].in = ++cnt;
        t[p].out = ++cnt;
        add(t[p].in, tls(p).in, 0);
        add(t[p].in, trs(p).in, 0);
        add(tls(p).out, t[p].out, 0);
        add(trs(p).out, t[p].out, 0);
    }
    void updIn(int x, int y, int from, int dis, int p = 1) {
```

```
        // update(2, 99999) 时, 调用 31 次 add (上限)
        if (x <= L && y >= R) return add(from, t[p].in, dis);
        if (x <= tls(p).r) updIn(x, y, from, dis, ls(p));
        if (y >  tls(p).r) updIn(x, y, from, dis, rs(p));
    }
    void updOut(int x, int y, int from, int dis, int p = 1) {
        if (x <= L && y >= R) return add(t[p].out, from, dis);
        if (x <= tls(p).r) updOut(x, y, from, dis, ls(p));
        if (y >  tls(p).r) updOut(x, y, from, dis, rs(p));
    }
}
using namespace Seg;

int main() {
    cnt = n = rd(), m = rd(), s = rd();
    build(1, n, 1);
    while (m--) {
        opt = rd(), x = rd();
        switch (opt) {
            case 1: y = rd(), w = rd(); add(x, y, w); break; // x -> y
            case 2: l = rd(), r = rd(), w = rd(); updIn (l, r, x, w); break; // x -> [l
    , r]
            case 3: l = rd(), r = rd(), w = rd(); updOut(l, r, x, w); break; // [l, r]
    -> x
        }
    }
    /* dijkstra */
}
```

4.1.9 无向图最小环

```
// O(n ^ 3)
int w[N][N];  // 原图的邻接矩阵
inline int floyd(const int &n) {
  static int dis[N][N];  // 最短路矩阵
  repn(i, 1, n) repn(j, 1, n) dis[i][j] = w[i][j];  // 初始化最短路矩阵
  int ans = INF;
  repn(k, 1, n) {
      rep(i, 1, k) rep(j, 1, i)
          ans = min(ans, dis[i][j] + w[i][k] + w[k][j]);  // 更新答案
      repn(i, 1, n) repn(j, 1, n)
          dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);  // 正常的 floyd 更新最短路矩
    阵
  }
  return ans;
}
```

4.1.10 k 短路

```
// 要求前 k 短路的长度和小于 m, 求 k 的值
// 可持续化可并堆优化
// time: O(nlogm)  space: O((n + k)logm)
#include <bits/stdc++.h>
using namespace std;
const int N = 200010;
int n, m, s, t, k, x, y, ww, cnt, fa[N];
struct Edge {
    int cur, h[N], nxt[N], p[N], w[N];
```

```
    void add_edge(int x, int y, int z) {
        nxt[++cur] = h[x], h[x] = cur, p[cur] = y, w[cur] = z;
    }
} e1, e2;
int dist[N];
bool tf[N], vis[N], ontree[N];
struct Node {
    int x, v;
    Node* operator = (Node a) { x = a.x, v = a.v; return this; }
    bool operator < (Node a) const { return v > a.v; }
} a;
priority_queue<Node> que;
void dfs(int x) {
    vis[x] = 1;
    for (int j = e2.h[x]; j; j = e2.nxt[j])
        if (!vis[e2.p[j]] && dist[e2.p[j]] == dist[x] + e2.w[j])
            fa[e2.p[j]] = x, ontree[j] = 1, dfs(e2.p[j]);
}
struct LeftistTree {
    int cnt, rt[N], lc[N * 20], rc[N * 20], dist[N * 20];
    Node v[N * 20];
    LeftistTree() { dist[0] = -1; }
    int newNode(Node w) { v[++cnt] = w; return cnt; }
    int merge(int x, int y) {
        if (!x || !y) return x + y;
        if (v[x] < v[y]) swap(x, y);
        int p = ++cnt;
        lc[p] = lc[x], v[p] = v[x];
        rc[p] = merge(rc[x], y);
        if (dist[lc[p]] < dist[rc[p]]) swap(lc[p], rc[p]);
        dist[p] = dist[rc[p]] + 1;
        return p;
    }
} st;
void dfs2(int x) {
    vis[x] = true;
    if (fa[x]) st.rt[x] = st.merge(st.rt[x], st.rt[fa[x]]);
    for (int j = e2.h[x]; j; j = e2.nxt[j])
        if (fa[e2.p[j]] == x && !vis[e2.p[j]]) dfs2(e2.p[j]);
}
int main() {
    scanf("%d%d%d%d%d", &n, &m, &s, &t, &k);
    for (int i = 1; i <= m; i++)
        scanf("%d%d%d", &x, &y, &ww), e1.add_edge(x, y, ww), e2.add_edge(y, x, ww);
    que.push({t, 0});
    while (!que.empty()) {
        a = que.top(), que.pop();
        if (tf[a.x]) continue;
        tf[a.x] = 1, dist[a.x] = a.v;
        for (int j = e2.h[a.x]; j; j = e2.nxt[j]) que.push({e2.p[j], a.v + e2.w[j]});
    }
    if (k == 1) {
        if (tf[s]) printf("%d\n", dist[s]);
        else printf("-1\n");
        return 0;
    }
    dfs(t);
    for (int i = 1; i <= n; i++)
        if (tf[i]) for (int j = e1.h[i]; j; j = e1.nxt[j])
```

```
            if (!ontree[j] && tf[e1.p[j]])
                st.rt[i] = st.merge( st.rt[i],
                        st.newNode({e1.p[j], dist[e1.p[j]] + e1.w[j] - dist[i]}));
    for (int i = 1; i <= n; i++) vis[i] = 0;
    dfs2(t);
    if (st.rt[s]) que.push({st.rt[s], dist[s] + st.v[st.rt[s]].v});
    while (!que.empty()) {
        a = que.top(), que.pop(), ++cnt;
        if (cnt == k - 1) return !printf("%d\n", a.v);
        if (st.lc[a.x]) que.push({st.lc[a.x], a.v - st.v[a.x].v + st.v[st.lc[a.x]].v});
        if (st.rc[a.x]) que.push({st.rc[a.x], a.v - st.v[a.x].v + st.v[st.rc[a.x]].v});
        x = st.rt[st.v[a.x].x];
        if (x) que.push({x, a.v + st.v[x].v});
    }
    printf("-1\n");
    return 0;
}
```

4.2 生成树

4.2.1 Kruskal 最小生成树

```
// O(mlogm)
struct Edge { int u, v, w; } e[M];
int pre[N]; // 并查集
void kruskal(){
    sort(e + 1, e + 1 + m, [&] (Edge a, Edge b) { return w < b.w; });
    for (int i = 1; i <= m; ++i) {
        int fu = find(e[i].u), fv = find(e[i].v);
        if (fu == fv) continue;
        pre[fu] = fv, ans += e[i].w;
        if (++cnt == n - 1) break;
    }
}
if (cnt < n - 1) puts("No"); // 图不连通，无最小生成树
```

4.2.2 Prim 最小生成树

```
// prim 更适用于稠密图，可适用于重边，自环，负边权
// 暴力 O(n * n + m)
inline int prim() {
    memset(dis, 0x3f, sizeof(dis));
    for (int i = h[1]; i; i = e[i].nx) { // 处理重边
        dis[e[i].to] = min(dis[e[i].to], e[i].dis);
    }
    int u = 1, ans = 0;
    while (++tot < n) {
        int mn = INF; vis[u] = 1;
        for (int i = 1; i <= n; ++i) {
            if (!vis[i] && mn > dis[i]) {
                mn = dis[i], u = i;
            }
        }
        if (vis[u]) return -1; // 不连通
        ans += mn;
        for (int i = h[u]; i; i = e[i].nx) {
            int v = e[i].to;
            if (!vis[v] && dis[v] > e[i].dis) {
```

```
                dis[v] = e[i].dis;
            }
        }
    }
    return ans;
}

// 堆优化 O(mlogn)
int n, m, cnt, cnt1, ans;
int h[N], dis[N], vis[N]; // 邻接表存图
struct P {
    int dis, id;
    bool operator < (const P &b) const { return d > b.d; }
};
namespace Prim {
    inline void init() { memset(h, 0, sizeof(h)), cnt = 0; }
    inline void prim(int s) {
        priority_queue<P>que;
        memset(dis, 0x3f, sizeof(dis));
        memset(vis, 0, sizeof(vis));
        dis[s] = 0;
        que.push(p(0, s));
        while (!que.empty() && cnt1 < n) { //在确定为连通图的情况下可删 !que.empty()&&
            P p = que.top(); que.pop();
            int u = p.id;
            if (vis[u]) continue;
            vis[u] = 1;
            ans += p.dis, ++cnt1; // 若最终 cnt1 = n, 则答案为 ans, 否则图不连通
            for (int i = h[u]; i; i = e[i].nx) {
                int v = e[i].to;
                if (dis[v] > e[i].dis) {
                    dis[v] = e[i].dis;
                    if (!vis[v]) que.push((P) {dis[v], v});
                }
            }
        }
    }
}
```

4.2.3 Zhu Liu 最小树形图

```
// 有向图最小生成树 ( 最小树形图 )
const int maxn = "Edit";
// 固定根的最小树型图, 邻接矩阵写法, 结点标号从 0 开始
struct MDST {  // 最小直径生成树
    int n;
    int w[maxn][maxn]; // 边权
    int vis[maxn];     // 访问标记, 仅用来判断无解
    int ans;           // 计算答案
    int removed[maxn]; // 每个点是否被删除
    int cid[maxn];     // 所在圈编号
    int pre[maxn];     // 最小入边的起点
    int iw[maxn];      // 最小入边的权值
    int max_cid;       // 最大圈编号
    void init(int n) {
        this->n = n;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) w[i][j] = INF;
```

```
18      }
19      void AddEdge(int u, int v, int cost) {
20          w[u][v] = min(w[u][v], cost); // 重边取权最小的
21      }
22      // 从s出发能到达多少个结点
23      int dfs(int s) {
24          vis[s] = 1;
25          int ans = 1;
26          for (int i = 0; i < n; i++)
27              if (!vis[i] && w[s][i] < INF) ans += dfs(i);
28          return ans;
29      }
30      // 从u出发沿着pre指针找圈
31      bool cycle(int u) {
32          max_cid++;
33          int v = u;
34          while (cid[v] != max_cid) {
35              cid[v] = max_cid;
36              v = pre[v];
37          }
38          return v == u;
39      }
40      // 计算u的最小入弧，入弧起点不得在圈c中
41      void update(int u) {
42          iw[u] = INF;
43          for (int i = 0; i < n; i++)
44              if (!removed[i] && w[i][u] < iw[u]) {
45                  iw[u] = w[i][u];
46                  pre[u] = i;
47              }
48      }
49      // 根结点为s，如果失败则返回false
50      bool solve(int s) {
51          memset(vis, 0, sizeof(vis));
52          if (dfs(s) != n) return false;
53          memset(removed, 0, sizeof(removed));
54          memset(cid, 0, sizeof(cid));
55          for (int u = 0; u < n; u++) update(u);
56          pre[s] = s;
57          iw[s] = 0; // 根结点特殊处理
58          ans = max_cid = 0;
59          for (;;) {
60              bool have_cycle = false;
61              for (int u = 0; u < n; u++)
62                  if (u != s && !removed[u] && cycle(u)) {
63                      have_cycle = true;
64                      // 以下代码缩圈，圈上除了u之外的结点均删除
65                      int v = u;
66                      do {
67                          if (v != u) removed[v] = 1;
68                          ans += iw[v];
69                          // 对于圈外点i，把边i->v改成i->u（并调整权值）；v->i改为u->i
70                          // 注意圈上可能还有一个v'使得i->v'或者v'->i存在，
71                          // 因此只保留权值最小的i->u和u->i
72                          for (int i = 0; i < n; i++)
73                              if (cid[i] != cid[u] && !removed[i]) {
74                                  if (w[i][v] < INF)
75                                      w[i][u] = min(w[i][u], w[i][v] - iw[v]);
76                                  w[u][i] = min(w[u][i], w[v][i]);
```

```
                                if (pre[i] == v) pre[i] = u;
                            }
                        v = pre[v];
                    } while (v != u);
                    update(u);
                    break;
                }
            if (!have_cycle) break;
        }
        for (int i = 0; i < n; i++)
            if (!removed[i]) ans += iw[i];
        return true;
    }
} T;
int n, m, r, u, v, w;
int main() {
    scanf("%d%d%d", &n, &m, &r);
    T.init(n);
    while (m--) {
        scanf("%d%d%d", &u, &v, &w);
        T.AddEdge(u - 1, v - 1, w); // 注意结点编号从 0 开始
    }
    int w = T.solve(r - 1);
    if (w) printf("%d\n", T.ans);
    else puts("-1"); // 不存在
    return 0;
}
```

4.2.4 严格次小生成树

```
// 对每一条不在 mst 内的边(u, v, w), 替换 mst 内 (u, v) 之间的最大边, 即可得到不严格次小生成树
// 若最大边权值等于 w, 则替换严格次大边, 这样得到的就是严格次小生成树了

struct P { int x, y, w; } ee[M];
struct Edge { int to, nx, w; } e[N << 1];

int n, m, tot, cnt, d = INF; ll sum; // mx 最大值, sx 严格次大值
int h[N], lg[N], pre[N], dep[N], vis[N], fa[N][22], mx[N][22], sx[N][22];
int find(int x) { return pre[x] ? pre[x] = find(pre[x]) : x; }

inline void add(int u, int v, int w) {
    e[++cnt] = (Edge) {v, h[u], w}, h[u] = cnt;
    e[++cnt] = (Edge) {u, h[v], w}, h[v] = cnt;
}

inline void kruskal() {
    sort(ee + 1, ee + 1 + m, [&] (P a, P b) { return a.w < b.w; });
    for (int i = 1; i <= m; ++i) {
        const P &e = ee[i];
        if (find(e.x) ^ find(e.y)) {
            pre[find(e.x)] = find(e.y);
            vis[i] = 1, sum += e.w;
            add(e.x, e.y, e.w);
            if (++tot == n - 1) break;
        }
    }
}

```

```cpp
void dfs(int u, int pre) {
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (v == pre) continue;
        dep[v] = dep[u] + 1;
        mx[v][0] = e[i].w;
        fa[v][0] = u;
        for (int i = 1; i <= lg[dep[v]]; ++i) {
            int pv = fa[v][i - 1];
            int x = mx[v][i - 1], y = mx[pv][i - 1];
            fa[v][i] = fa[pv][i - 1];
            mx[v][i] = max(x, y);
            if (x ^ y) {
                if (x > y) {
                    sx[v][i] = max(y, sx[v][i - 1]);
                } else {
                    sx[v][i] = max(x, sx[pv][i - 1]);
                }
            } else {
                sx[v][i] = max(sx[v][i - 1], sx[pv][i - 1]);
            }
        }
        dfs(v, u);
    }
}

int mx_, sx_;
inline void cmp(int x) {
    if (x > mx_) sx_ = mx_, mx_ = x;
    else if (x > sx_ && x < mx_) sx_ = x;
}
inline void upd(int a, int b) { cmp(mx[a][b]), cmp(sx[a][b]); }
inline int check(int x, int y, int w) { // 找不与 w 相等的最大边
    mx_ = sx_ = -1;
    if (dep[x] < dep[y]) swap(x, y);
    while (dep[x] > dep[y]) {
        upd(x, lg[dep[x] - dep[y]]);
        x = fa[x][lg[dep[x] - dep[y]]];
    }
    if (x ^ y) {
        for (int i = lg[dep[x]]; i >= 0; --i) {
            if (fa[x][i] ^ fa[y][i]) {
                upd(x, i), upd(y, i);
                x = fa[x][i], y = fa[y][i];
            }
        }
        cmp(mx[x][0]), cmp(mx[y][0]);
    }
    if (~mx_) {
        if (w ^ mx_) return w - mx_;
        if (~sx_) return w - sx_;
    }
    return INF;
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; ++i) { // edge
        scanf("%d%d%d", &ee[i].x, &ee[i].y, &ee[i].w);
```

```
    }
    *lg = -1;
    for (int i = 1; i <= n; ++i) { lg[i] = lg[i >> 1] + 1; }
    memset(sx, -1, sizeof(sx));
    kruskal(), dfs(1, 0);
    for (int i = 1; i <= m; ++i) {
        if (!vis[i]) d = min(d, check(ee[i].x, ee[i].y, ee[i].w));
    }
    printf("%lld\n", sum + d);
    return 0;
}
```

4.2.5 Kruskal 重构树

```
// kruskal 重构树用于解决瓶颈生成树问题
// 经过权值 <= w 的边能到达的第 k 大点 => 在 Kruskal 重构树 dfs 序上建主席树即可
#define N 20004 // 记得点要开两倍空间
struct Input { int x, y, w; } ee[M];
struct Edge { int to, nx; } e[N]; // 2n - 2

int n, m, q, x, y, z, tot, cnt;
int h[N], w[N], fa[N], pre[N], dep[N], top[N], son[N], siz[N];

inline void add(int u, int v) {
    e[++cnt] = (Edge) {v, h[u]}, h[u] = cnt;
}

int find(int x) {
    return pre[x] ? pre[x] = find(pre[x]) : x;
}

// 建立 kruskal 重构树 (新增 n - 1 个点和 2n - 2 条边)
inline void kruskal() {
    sort(ee + 1, ee + 1 + m, [&] (Input a, Input b) { return a.w > b.w; });
    tot = n; // 边权从大到小排, 则根的点权最小, 重构树点权从下到上单调不增
    for (int i = 1; i <= m; ++i) {
        x = find(ee[i].x), y = find(ee[i].y);
        if (x == y) continue;
        w[++tot] = ee[i].w; // x - y 的边权变为了 LCA(x, y) 的点权
        pre[x] = pre[y] = tot;
        add(tot, x), add(tot, y);
    }
    for (int i = tot; i; --i) { // 倒序
        if (!dep[i]) dfs1(i, 0), dfs2(i, i); // 建树 (如果不用求 LCA, 那就不用树剖了)
    }
}
// x - y 路径上边权最小值的最大值等于 Kruskal 重构树的 w[LCA(x, y)]

inline int query(int x, int k, ...) { // 对于某些题目的查询需要用树上倍增解决
    for (int y, i = lg[dep[x]]; i >= 0; --i) {
        if ((y = fa[x][i]) && w[fa[x][i]] <= k) x = y;
    }
}
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; ++i)
        scanf("%d%d%d", &ee[i].x, &ee[i].y, &ee[i].w);
    kruskal(); while (q--) // query
```

```
    return 0;
}
```

4.2.6 最小直径生成树

```
#include <bits/stdc++.h>
using namespace std;
const int N = 502;
typedef long long ll;
typedef pair<int, int> pii;
#define repn(i, x, y) for (int i = x; i <= y; ++i)
ll d[N][N], dd[N][N], rk[N][N], val[N];
const ll INF = 1e17;
int n, m;
bool cmp(int a, int b) { return val[a] < val[b]; }
void floyd() {
    repn(k, 1, n) repn(i, 1, n) repn(j, 1, n)
        d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
}
struct node {
    ll u, v, w;
} a[N * (N - 1) / 2];
inline void init() {
    repn(i, 1, 501) repn(j, 1, 501) d[i][j] = INF;
    repn(i, 1, 501) d[i][i] = 0;
}
inline void solve() {
    //求图的绝对中心
    floyd();
    repn(i, 1, n) {
        repn(j, 1, n) rk[i][j] = j, val[j] = d[i][j];
        sort(rk[i] + 1, rk[i] + 1 + n, cmp);
    }
    ll P = 0, ansP = INF;
    //在点上
    repn(i, 1, n) if (d[i][rk[i][n]] * 2 < ansP) {
        ansP = d[i][rk[i][n]] * 2, P = i;
    }
    //在边上
    int f1 = 0, f2 = 0;
    ll disu = INT_MIN, disv = INT_MIN, ansL = INF;
    repn(i, 1, m) {
        ll u = a[i].u, v = a[i].v, w = a[i].w;
        for (int p = n, i = n - 1; i >= 1; i--) {
            if (d[v][rk[u][i]] > d[v][rk[u][p]]) {
                if (d[u][rk[u][i]] + d[v][rk[u][p]] + w < ansL) {
                    ansL = d[u][rk[u][i]] + d[v][rk[u][p]] + w;
                    f1 = u, f2 = v;
                    disu = (d[u][rk[u][i]] + d[v][rk[u][p]] + w) / 2 - d[u][rk[u][i]];
                    disv = w - disu;
                }
                p = i;
            }
        }
    }
    cout << min(ansP, ansL) / 2 << '\n';
    //最小路径生成树
    vector<pii> pp;
```

```
    init();
    if (ansP <= ansL) {
        repn(j, 1, n) repn(i, 1, m) {
            ll u = a[i].u, v = a[i].v, w = a[i].w;
            if (dd[P][u] + w == d[P][v] && dd[P][u] + w < dd[P][v]) {
                dd[P][v] = dd[P][u] + w;
                pp.push_back({u, v});
            }
            u = a[i].v, v = a[i].u, w = a[i].w;
            if (dd[P][u] + w == d[P][v] && dd[P][u] + w < dd[P][v]) {
                dd[P][v] = dd[P][u] + w;
                pp.push_back({u, v});
            }
        }
        for (auto [x, y] : pp) cout << x << ' ' << y << '\n';
    } else {
        d[n + 1][f1] = disu, d[f1][n + 1] = disu;
        d[n + 1][f2] = disv, d[f2][n + 1] = disv;
        a[m + 1].u = n + 1, a[m + 1].v = f1, a[m + 1].w = disu;
        a[m + 2].u = n + 1, a[m + 2].v = f2, a[m + 2].w = disv;
        n += 1, m += 2, floyd(), P = n;
        repn(j, 1, n) repn(i, 1, m) {
            ll u = a[i].u, v = a[i].v, w = a[i].w;
            if (dd[P][u] + w == d[P][v] && dd[P][u] + w < dd[P][v]) {
                dd[P][v] = dd[P][u] + w;
                pp.push_back({u, v});
            }
            u = a[i].v, v = a[i].u, w = a[i].w;
            if (dd[P][u] + w == d[P][v] && dd[P][u] + w < dd[P][v]) {
                dd[P][v] = dd[P][u] + w;
                pp.push_back({u, v});
            }
        }
        cout << f1 << ' ' << f2 << '\n';
        for (auto [x, y] : pp)
            if (x != n && y != n) cout << x << ' ' << y << '\n';
    }
}
int main() {
    init();
    cin >> n >> m;
    repn(i, 1, m) {
        ll u, v, w;
        cin >> u >> v >> w;
        w *= 2, d[u][v] = w, d[v][u] = w;
        a[i].u = u, a[i].v = v, a[i].w = w;
    }
    solve();
    return 0;
}
```

4.2.7 最小异或生成树

```
// 完全图, 边权为两点间 xor 值, 求 MST
#include <bits/stdc++.h>
#define ll long long
using std::min;
using std::max;
```

```cpp
const int N=2e5+10;
const int inf=0x3f3f3f3f;
#define ls ch[now][0]
#define rs ch[now][1]
int L[N*40],R[N*40],ch[N*40][2],tot;
int n,a[N],root;
void Insert(int &now,int x,int dep) {
    if(!now) now=++tot;
    L[now]=min(L[now],x),R[now]=max(R[now],x);
    if(dep<0) return;
    int bit=a[x]>>dep&1;
    Insert(ch[now][bit],x,dep-1);
}
int query(int now,int val,int dep) {
    if(dep<0) return 0;
    int bit=val>>dep&1;
    if(ch[now][bit]) return query(ch[now][bit],val,dep-1);
    else return query(ch[now][bit^1],val,dep-1)+(1<<dep);
}
ll dfs(int now,int dep) {
    if(dep<0) return 0;
    if(R[ls]&&R[rs]) {
        int mi=inf;
        for(int i=L[ls];i<=R[ls];i++) mi=min(mi,query(rs,a[i],dep-1));
        return dfs(ls,dep-1)+dfs(rs,dep-1)+mi+(1<<dep);
    }
    if(R[ls]) return dfs(ls,dep-1);
    if(R[rs]) return dfs(rs,dep-1);
    return 0;
}
int main() {
    scanf("%d",&n);
    for(int i=1;i<=n;i++) scanf("%d",a+i);
    std::sort(a+1,a+1+n);
    memset(L,0x3f,sizeof(L));
    for(int i=1;i<=n;i++) Insert(root,i,30);
    printf("%lld\n",dfs(root,30));
    return 0;
}
```

4.3 树的直径

```cpp
// 树a 与 树b 合并, 新树c 直径为 dc = (da + 1) / 2 + (db + 1) / 2 + 1
int s;
void dfs1(int u, int pre) {
    for (int &v : e[u]) {
        if (v != pre) {
            d[v] = d[u] + e[i].dis;
            if (d[v] > d[s]) s = v;
            dfs1(v, u);
        }
    }
}
inline int work() { // 只适用于正边权  O(n)
    dfs1(s = 1, 0), dfs1(s, d[s] = 0);
    return d[s]; // 直径
}

```

```cpp
void dfs2(int u, int pre) { // 适用于正|负边权  O(n)
    for (int &v : e[u]) {
        if (v != pre) {
            dfs2(v, u);
            ans = max(ans, f[u] + f[v] + e[i].dis); // 直径
            f[u] = max(f[u], f[v] + e[i].dis);
        }
    }
}
```

4.4 拓扑排序

```cpp
int q[N], l, r;
inline void tuopu() { // 有向图
    for (int i = 1; i <= n; ++i) {
        if (!in[i]) q[++r] = i;
    }
    while (l < r) {
        int u = q[++l];
        for (auto &v : e[u]) {
            if (in[v]) {
                // do sth
                if (!--in[v]) q[++r] = v;
            }
        }
    }
}
// 无向图入队条件修改 !in[x] => (in[x] == 1)
```

4.5 欧拉路径

- 基本概念:
  - 欧拉图: 能够没有重复地一次遍历所有边的图。(必须是连通图)
  - 欧拉路: 上述遍历的路径就是欧拉路。
  - 欧拉回路: 若欧拉路是闭合的（一个圈，从起点开始遍历最终又回到起点），则为欧拉回路。
- 无向图 G 有欧拉路径的充要条件
  - G 是连通图
  - G 中奇顶点（连接边的数量为奇数）的数量等于 0 或 2.
- 无向图 G 有欧拉回路的充要条件
  - G 是连通图
  - G 中每个顶点都是偶顶点
- 有向图 G 有欧拉路径的充要条件
  - G 是连通图
  - u 的出度比入度大 1，v 的出度比入度小 1，其他所有点出度和入度相同。(u 为起点，v 为终点)
- 有向图 G 有欧拉回路的充要条件
  - G 是连通图
  - G 中每个顶点的出度等于入度

4.5.1 欧拉路径

若有两个点的度数是奇数，则此时这两个点只能作为欧拉路径的起点和终点。

```

int in[N], out[N], cur[N], st[M], top;
inline void add(int u, int v) { // 邻接矩阵写法
    ++d[u], ++d[v], ++g[u][v], ++g[v][u]; // 无向图
    ++out[u], ++in[v], ++g[u][v]; // 有向图
}
inline void add(int u, int v) { // 邻接表写法
    ++out[u], ++in[v], e[u].emplace_back(v); // 有向图
}
void dfs(int u) { // 邻接矩阵写法
    for (int i = 1; i <= n; ++i) {
        if (g[u][i]) {
            --g[u][i], --g[i][u]; // 无向图; 有向图只需删一条边
            dfs(i);
        }
    }
    st[++top] = u; // 一定要先存下来
}
void dfs(int u) { // 邻接表写法(有向图), 若要保证字典最小, 则需要对邻接表排序
    for (int i = cur[u]; i < e[u].size(); i = cur[u]) { // 邻接表
        cur[u] = i + 1, dfs(e[u][i]);
    }
    st[++top] = u;
}
// 有多个自环, 并且要保证路径边权字典序最小的情况
// 先对输入边按边权排序, 然后再加边
void dfs(int now, int u) {
    if (now == m) return;
    for (int num, i = 0; i < e[u].size(); ++i) {
        if (vis[num = e[u][i].num]) continue; // num: 边的编号
        vis[num] = 1, dfs(now + 1, e[u][i].to);
        st[++top] = num; // 一定要存好然后在 dfs 外倒序输出!!!
    }
}
dfs(0, start);
```

4.6 差分约束

建模技巧

最短路求最大解，最长路求最小解。
最短路 $a <= b + c => add(b, a, c)$;

最长路 $a >= b + c => add(b, a, c)$;

$a = b => add(a, b, 0), add(b, a, 0)$;

$a < b + w => \mathbf{a <= b + w - 1} => add(b, a, w - 1)$;

$a > b + w => \mathbf{b <= a - w - 1} => add(a, b, -w - 1)$;
两种处理方式 1 一开始将所有点放入队列中而不是只放一个起点，这样每个点都被作为起点讨论过 2 建一个虚拟源点 0，从这个虚拟源点向其他所有点都连一条长度为 0 的边，以虚拟源点为起点放入队列建图后 $SPFA$ 判负环，有负环代表无解，否则各点的 dis 值即为一组解。

4.7 连通性相关

4.7.1 缩点

```
// tarjan  O(n + m)
int n, m, x, y, top, cnt, cnt1, tim;
int h[N], h1[N], w[N], f[N], in[N];
int st[N], vis[N], dfn[N], low[N], dis[N];
struct Edge { int from, to, nx; } e[M], e1[M];
inline void add(int u, int v) { e[++cnt] = (Edge) {u, v, h[u]}, h[u] = cnt; }
inline void add1(int u, int v) { e1[++cnt1] = (Edge) {u, v, h1[u]}, h1[u] = cnt1; }
inline void tarjan(int u) {
    dfn[u] = low[u] = ++tim;
    st[++top] = u, vis[u] = 1;
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (vis[v]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {
        int v = st[top--];
        while (v) {
            f[v] = u, vis[v] = 0;
            if (u == v) break;
            w[u] += w[v];
            v = st[top--];
        }
    }
}

int main() {
    n = rd(), m = rd();
    for (int i = 1; i <= n; ++i) w[i] = rd();
    for (int i = 1; i <= m; ++i) x = rd(), y = rd(), add(x, y);
    for (int i = 1; i <= n; ++i) if (!dfn[i]) tarjan(i);
    for (int i = 1; i <= m; ++i) {
        int u = f[e[i].from], v = f[e[i].to];
        if (u ^ v) add1(u, v), ++in[v]; // 建缩点后的图
    }
    return 0;
}
```

4.7.2 割点

```
// tarjan  O(n + m)
int n, m, x, y, cnt, tim, tot;
int h[N], dfn[N], low[N], cut[N];
struct Edge { int to, nx; } e[M << 1];
inline void add(int u, int v) { e[++cnt] = (Edge) {v, h[u]}, h[u] = cnt; }
void tarjan(int u, int rt) {
    dfn[u] = low[u] = ++tim;
    int child = 0;
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (!dfn[v]) {
```

```
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            if (u == rt) ++child;
            else if (low[v] >= dfn[u]) cut[u] = 1; // 标记割点
        } else {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (u == rt && child >= 2) cut[u] = 1; // 标记割点
}
int main() {
    scanf("%d%d", &n, &m);
    while (m--) scanf("%d%d", &x, &y), add(x, y), add(y, x);
    for (int i = 1; i <= n; ++i) if (!dfn[i]) tarjan(i, i);
    for (int i = 1; i <= n; ++i) if (cut[i]) ++tot;
    printf("%d\n", tot); // 割点数量
    for (int i = 1; i <= n; ++i) if (cut[i]) printf("%d ", i); puts("");
    return 0;
}
```

4.7.3 割边（桥）

```
int low[N], dfn[N],  fa[N], isbridge[N], tot;
void tarjan(int u, int pre) {
  low[u] = dfn[u] = ++tim, fa[u] = pre;
  for (int i = h[u]; i; i = e[i].nx) {
      int v = e[i].to;
      if (!dfn[v]) {
          tarjan(v, u);
          low[u] = min(low[u], low[v]);
          if (low[v] > dfn[u]) {
              isbridge[v] = true; // 标记割边(桥)
              ++tot; // 桥的数量
          }
      } else if (dfn[v] < dfn[u] && v != pre) {
          low[u] = min(low[u], dfn[v]);
      }
  }
}
```

4.7.4 圆方树

```
// 解决点双问题
struct Edge { int to, nx; } e[M];
int h[N], fa[N], st[N], dfn[N], low[N], vis[N]; // N = maxn << 1
int n, a, b, x, y, top, tim, cnt, num, tot, ans = 1e9;
inline void add(int u, int v) { e[++cnt] = (Edge) {v, h[u]}, h[u] = cnt; }
void tarjan(int u) {
    dfn[u] = low[u] = ++tim, st[++top] = u;
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
            if (dfn[u] == low[v]) { // 构建圆方树
                ++num; // 圆方树附加结点编号
                for (int x = 0; x ^ v; --top) {
                    x = st[top];
```

```
                    add(x, num), add(num, x);
                }
                add(u, num), add(num, u);
            }
        } else low[u] = min(low[u], dfn[v]);
    }
}
void dfs(int u) {
    if (u == b) return;
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (vis[v]) continue;
        if ((u <= n && v > n) || (u > n && v <= n)) {
            fa[v] = u, vis[v] = 1, dfs(v);
        }
    }
}
int main() {
    scanf("%d", &n), num = n;
    while (scanf("%d%d", &x, &y) && x) {
        add(x, y), add(y, x);
    }
    // 求使得 a, b 两点不连通的编号最小的一个点
    scanf("%d%d", &a, &b);
    for (int i = 1; i <= n; ++i) {
        if (!dfn[i]) tarjan(i), --top; // --top
    }
    dfs(a);
    if (fa[fa[b]] == a || !vis[b]) puts("No solution"), exit(0);
    while (fa[b] ^ a) {
        b = fa[b];
        if (b <= n) ans = min(ans, b);
    }
    printf("%d\n", ans);
    return 0;
}
```

4.7.5 无向图全局最小割

```
// Stoer-Wagner 算法
#include <bits/stdc++.h>
using namespace std;
const int N = 610, INF = 1e9;
int n, m, x, y, z, s, t;
int dis[N][N], w[N], dap[N], vis[N], ord[N];
inline int proc(int x) {
    memset(vis, 0, (n + 1) << 3);
    memset(w, 0, (n + 1) << 3), *w = -1;
    for (int i = 1; i <= n - x + 1; i++) {
        int mx = 0;
        for (int j = 1; j <= n; j++) {
            if (!dap[j] && !vis[j] && w[j] > w[mx]) mx = j;
        }
        vis[mx] = 1, ord[i] = mx;
        for (int j = 1; j <= n; j++) {
            if (!dap[j] && !vis[j]) {
                w[j] += dis[mx][j];
            }
```

```
        }
    }
    s = ord[n - x], t = ord[n - x + 1];
    return w[t];
}
inline int sw() {
    int ans = INF;
    for (int i = 1; i < n; i++) {
        ans = min(ans, proc(i));
        dap[t] = 1;
        for (int j = 1; j <= n; j++) {
            dis[s][j] += dis[t][j];
            dis[j][s] += dis[j][t];
        }
    }
    return ans; // 如果图不连通, ans = 0
}
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++) {
        scanf("%d%d%d", &x, &y, &z);
        dis[x][y] += z, dis[y][x] += z;
    }
    printf("%d\n", sw());
    return 0;
}
```

4.7.6 最小斯坦纳树

```
// 花最小代价, 连通给定的 k 个关键点
#include <bits/stdc++.h>
using namespace std;
#define N 510
int n, m, k, x, y, z, eg, cnt;
int p[N], h[N], vis[N], dp[N][4200];
struct Edge { int to, nx, dis; } e[N << 1];
struct P {
    int dis, id;
    P(int d, int x) : dis(d), id(x) {}
    bool operator < (const P &b) const { return dis > b.dis; }
};
priority_queue<P> que;
inline void add(int u, int v, int w) {
    e[++cnt] = (Edge) {v, h[u], w}, h[u] = cnt;
}
void dij(int s) {
    memset(vis, 0, sizeof(vis));
    while (!que.empty()) {
        int u = que.top().id;
        que.pop();
        if (vis[u]) continue;
        vis[u] = 1;
        for (int i = h[u]; i; i = e[i].nx) {
            int v = e[i].to;
            if (dp[v][s] > dp[u][s] + e[i].dis) {
                dp[v][s] = dp[u][s] + e[i].dis;
                que.push(P(dp[v][s], v));
            }
```

```
        }
    }
}
int main() {
    memset(dp, 0x3f, sizeof(dp));
    scanf("%d%d%d", &n, &m, &k);
    for (int i = 1; i <= m; ++i) {
        scanf("%d%d%d", &x, &y, &z);
        add(x, y, z), add(y, x, z);
    }
    for (int i = 1; i <= k; ++i) {
        scanf("%d", &p[i]); // 关键点
        dp[p[i]][1 << (i - 1)] = 0;
    }
    for (int s = 1; s < (1 << k); ++s) {
        for (int i = 1; i <= n; ++i) {
            for (int subs = s & (s - 1); subs; subs = s & (subs - 1)) {
                dp[i][s] = min(dp[i][s], dp[i][subs] + dp[i][s ^ subs]);
            }
            if (dp[i][s] != 0x3f3f3f3f) que.push(P(dp[i][s], i));
        }
        dij(s);
    }
    printf("%d\n", dp[p[1]][(1 << k) - 1]);
    return 0;
}
```

4.7.7 最小割树

```
// 最小割树, q 次询问 u, v 之间的最小割
#include <bits/stdc++.h>
using namespace std;
#define N 505
#define M 1505
#define LOGN 10
const int INF = 1e9;
int n, m, q;
namespace Dinic {
    int h[N], cur[N], dep[N], cnt = 1;
    struct Edge { int to, nx, f; } e[M << 2]; //
    inline void add(int u, int v, int f) {
        e[++cnt] = (Edge) {v, h[u], f}, h[u] = cnt;
        e[++cnt] = (Edge) {u, h[v], 0}, h[v] = cnt;
    }
    int q[N], l, r;
    inline int bfs(int s, int t) {
        memset(dep, 0, (n + 1) << 2);
        dep[s] = 1, cur[s] = h[s];
        l = 0, q[r = 1] = s;
        while (l < r) {
            int u = q[++l];
            for (int i = h[u]; i; i = e[i].nx) {
                int v = e[i].to;
                if (e[i].f && !dep[v]) {
                    dep[v] = dep[u] + 1;
                    cur[v] = h[v];
                    q[++r] = v;
                }
```

```
            }
        }
        return dep[t];
    }
    int dfs(int u, int t, int flow = INT_MAX) {
        if (!flow || u == t) return flow;
        int f, add = 0;
        for (int &i = cur[u]; i; i = e[i].nx) {
            int v = e[i].to;
            if (dep[v] == dep[u] + 1 && (f = dfs(v, t, min(flow, e[i].f)))) {
                e[i].f -= f, e[i ^ 1].f += f;
                flow -= f, add += f;
                if (!flow) break;
            }
        }
        return add;
    }
    inline void init() {
        for (int i = 2; i <= cnt; i += 2) {
            e[i].f += e[i ^ 1].f;
            e[i ^ 1].f = 0;
        }
    }
    inline int main(int s, int t) {
        init(); int ans = 0, f;
        while (bfs(s, t)) {
            while ((f = dfs(s, t))) ans += f;
        }
        return ans;
    }
}

namespace Mincut_Tree {
    struct Edge { int to, nx, len; } e[N << 1];
    int h[N], cnt = 1;
    inline void add(int u, int v, int w) {
        e[++cnt] = (Edge) {v, h[u], w}, h[u] = cnt;
        e[++cnt] = (Edge) {u, h[v], w}, h[v] = cnt;
    }
    int node[N], tmp1[N], tmp2[N];
    void build(int l, int r) {
        if (l == r) return;
        int s = node[l], t = node[l + 1];
        int cut = Dinic::main(s, t);
        add(s, t, cut);
        int cnt1 = 0, cnt2 = 0;
        for (int i = l; i <= r; ++i) {
            if (Dinic::dep[node[i]]) tmp1[++cnt1] = node[i];
            else tmp2[++cnt2] = node[i];
        }
        for (int i = l; i <= l + cnt1 - 1; ++i) node[i] = tmp1[i - l + 1];
        for (int i = l + cnt1; i <= r; ++i) node[i] = tmp2[i - cnt1 - l + 1];
        build(l, l + cnt1 - 1);
        build(l + cnt1, r);
    }
    int dep[N], anc[N][LOGN], mnl[N][LOGN], log2n;
    void dfs(int u, int pre) {
        dep[u] = dep[pre] + 1;
        for (int i = 1; i <= log2n; i++) {
```

```
89              anc[u][i] = anc[anc[u][i - 1]][i - 1];
90              mnl[u][i] = min(mnl[u][i - 1], mnl[anc[u][i - 1]][i - 1]);
91          }
92          for (int i = h[u]; i; i = e[i].nx) {
93              int v = e[i].to;
94              if (v != pre) {
95                  anc[v][0] = u;
96                  mnl[v][0] = e[i].len;
97                  dfs(v, u);
98              }
99          }
100     }
101     inline void work() {
102         log2n = log2(n) + 1;
103         for (int i = 1; i <= n; i++) node[i] = i;
104         build(1, n), dfs(1, 0);
105     }
106     inline int query(int x, int y) {
107         int ans = INF;
108         if (dep[x] < dep[y]) swap(x, y);
109         for (int i = log2n; i >= 0; i--) {
110             if (dep[anc[x][i]] >= dep[y]) {
111                 ans = min(ans, mnl[x][i]);
112                 x = anc[x][i];
113             }
114         }
115         if (x == y) return ans;
116         for (int i = log2n; i >= 0; i--) {
117             if (anc[x][i] != anc[y][i]) {
118                 ans = min(ans, mnl[x][i]);
119                 ans = min(ans, mnl[y][i]);
120                 x = anc[x][i], y = anc[y][i];
121             }
122         }
123         ans = min(ans, mnl[x][0]);
124         ans = min(ans, mnl[y][0]);
125         return ans;
126     }
127 }
128
129 int main() {
130     scanf("%d %d", &n, &m);
131     for (int u, v, w, i = 1; i <= m; i++) {
132         scanf("%d %d %d", &u, &v, &w);
133         Dinic::add(u, v, w);
134         Dinic::add(v, u, w);
135     }
136     Mincut_Tree::work();
137     scanf("%d", &q);
138     for (int u, v, i = 1; i <= q; i++) {
139         scanf("%d %d", &u, &v);
140         int ans = Mincut_Tree::query(u, v);
141         if (ans == INF) ans = -1;
142         printf("%d\n", ans);
143     }
144     return 0;
145 }
```

4.8 2-SAT

```cpp
#define N 2000006 // maxn << 1
struct Edge { int to, nx; } e[N];
int n, m, i, a, j, b, cnt, top, scc, dfncnt;
int h[N], st[N], vis[N], dfn[N], low[N], col[N];
inline void add(int u, int v) { e[++cnt] = (Edge) {v, h[u]}, h[u] = cnt; }
void tarjan(int u) {
    st[++top] = u, vis[u] = 1;
    dfn[u] = low[u] = ++dfncnt;
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (vis[v]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {
        int v = st[top--]; ++scc;
        while (1) {
            col[v] = scc, vis[v] = 0;
            if (u == v) break;
            v = st[top--];
        }
    }
}
int main() {
    scanf("%d%d", &n, &m);
    while (m--) { // (i = a) V (j = b)
        scanf("%d%d%d%d", &i, &a, &j, &b); // 建图
        add(i + n * (a & 1), j + n * (b ^ 1));
        add(j + n * (b & 1), i + n * (a ^ 1));
    }                   // n << 1
    for (int i = 1; i <= (n << 1); ++i) if (!dfn[i]) tarjan(i);
    for (int i = 1; i <= n; ++i) {
        if (col[i] == col[i + n]) puts("IMPOSSIBLE"), exit(0); // 无解
    }
    puts("POSSIBLE");
    for (int i = 1; i <= n; ++i) { // 选项转换 0 => i, 1 => i + n
        printf("%d", col[i] < col[i + n]); // 选 scc 编号较小的点所代表的选项
    }
    return 0;
}
```

4.9 二分图匹配

1. 一个二分图中的最大匹配数等于这个图中的最小点覆盖数
2. 最小路径覆盖 $=|G|$-最大匹配数

   在一个 $N \times N$ 的有向图中, 路径覆盖就是在图中找一些路经, 使之覆盖了图中的所有顶点, 且任何一个顶点有且只有一条路径与之关联;

   (如果把这些路径中的每条路径从它的起始点走到它的终点, 那么恰好可以经过图中的每个顶点一次且仅一次); 如果不考虑图中存在回路, 那么每每条路径就是一个弱连通子集.

   由上面可以得出:

   (a) 一个单独的顶点是一条路径;

(b) 如果存在一路径 $p_1, p_2, ...p_k$, 其中 p_1 为起点,p_k 为终点, 那么在覆盖图中, 顶点 $p_1, p2, ...p_k$ 不再与其它的顶点之间存在有向边.

最小路径覆盖就是找出最小的路径条数, 使之成为 G 的一个路径覆盖.
路径覆盖与二分图匹配的关系: 最小路径覆盖 =|G|-最大匹配数;

3. 二分图最大独立集 = 顶点数-二分图最大匹配
独立集: 图中任意两个顶点都不相连的顶点集合。

4.9.1 匈牙利算法

时间复杂度:$O(VE)$.
建图时连左部点到右部点的有向边

```
vector<int> e[N];
int vis[N], mch[N];
int n, m, k, x, y, t, ans;
int dfs(int u) {
    for (auto &v: e[u]) {
        if (vis[v] ^ t) {
            vis[v] = t;
            if (!mch[v] || dfs(mch[v])) return mch[v] = u, 1;
        }
    }
    return 0;
}
inline int work() {
    int ans = 0;
    for (int i = 1; i <= n; ++i) {
        ++t; if (dfs(i)) ++ans;
    }
    return ans;
}
```

4.9.2 Hopcroft-Carp

二分图匹配复杂度 $O(\sqrt{n} * E)$
uN 为左端的顶点数, 使用前赋值 (点编号 0 开始)

```
const int maxn = "Edit";
vector<int> G[maxn];
int uN, dis;
int Mx[maxn], My[maxn];
int dx[maxn], dy[maxn];
bool used[maxn];
inline void init(int n) {
    for (int i = 0; i < n; i++) G[i].clear();
}
inline void addedge(int u, int v) { G[u].push_back(v); }
bool bfs() {
    queue<int> q;
    dis = INF;
    memset(dx, -1, sizeof(dx)), memset(dy, -1, sizeof(dy));
    for (int i = 0; i < uN; i++)
        if (Mx[i] == -1) q.push(i), dx[i] = 0;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        if (dx[u] > dis) break;
```

```
        for (auto& v : G[u]) {
            if (dy[v] == -1) {
                dy[v] = dx[u] + 1;
                if (My[v] == -1) dis = dy[v];
                else {
                    dx[My[v]] = dy[v] + 1;
                    q.push(My[v]);
                }
            }
        }
    }
    return dis != INF;
}
bool dfs(int u) {
    for (auto& v : G[u]) {
        if (!used[v] && dy[v] == dx[u] + 1) {
            used[v] = true;
            if (My[v] != -1 && dy[v] == dis) continue;
            if (My[v] == -1 || dfs(My[v])) {
                My[v] = u, Mx[u] = v;
                return true;
            }
        }
    }
    return false;
}
int MaxMatch() {
    int res = 0;
    memset(Mx, -1, sizeof(Mx)), memset(My, -1, sizeof(My));
    while (bfs()) {
        memset(used, false, sizeof(used));
        for (int i = 0; i < uN; i++)
            if (Mx[i] == -1 && dfs(i)) res++;
    }
    return res;
}
```

4.9.3 Hungry(Multiple)

```
const int maxn = "Edit";
const int maxm = "Edit";
int uN, vN;        //u,v的数目,使用前面必须赋值
int g[maxn][maxm]; //邻接矩阵
int linker[maxm][maxn];
bool used[maxm];
int num[maxm]; //右边最大的匹配数
bool dfs(int u)
{
    for (int v = 0; v < vN; v++)
        if (g[u][v] && !used[v])
        {
            used[v] = true;
            if (linker[v][0] < num[v])
            {
                linker[v][++linker[v][0]] = u;
                return true;
            }
            for (int i = 1; i <= num[0]; i++)
```

```
                if (dfs(linker[v][i]))
                {
                    linker[v][i] = u;
                    return true;
                }
        }
    return false;
}
int hungary()
{
    int res = 0;
    for (int i = 0; i < vN; i++) linker[i][0] = 0;
    for (int u = 0; u < uN; u++)
    {
        memset(used, 0, sizeof(used));
        if (dfs(u)) res++;
    }
    return res;
}
```

4.9.4 KM 算法

```
// 二分图最大权完美匹配  KM 算法  O(n ^ 3)
#include <bits/stdc++.h>
using namespace std;

#define N 505
#define repn(i, x, y) for (int i = x; i <= y; ++i)

typedef long long ll;
const ll INF = 1e14;

// vx[], vy[]: 左,右部点遍历标记
// mx[], my[]: 左,右部点匹配编号
// lx[], ly[]: 左,右部点顶标
// sl[]: 松弛量, w[][]: 边权

bool vx[N], vy[N]; // visx, visy
int n, m, x, y, l, r;
int q[N], mx[N], my[N], pre[N]; // matchx, matchy
ll lx[N], ly[N], sl[N], w[N][N]; // slack

inline int check(int y) {
    vy[y] = 1;
    if (my[y]) {
        if (!vx[my[y]]) vx[q[++r] = my[y]] = 1;
        return 0;
    }
    while (y) swap(y, mx[my[y] = pre[y]]);
    return 1;
}

inline void bfs(int s) {
    fill(sl, sl + 1 + n, INF);
    fill(vx, vx + 1 + n, 0);
    fill(vy, vy + 1 + n, 0);
    l = 0, q[r = 1] = s, vx[s] = 1;
    while (1) {
```

```
        while (l < r) {
            int x = q[++l];
            repn(y, 1, n) {
                ll d = lx[x] + ly[y] - w[x][y];
                if (!vy[y] && d <= sl[y]) {
                    sl[y] = d, pre[y] = x;
                    if (!d && check(y)) return;
                }
            }
        }
        ll dt = INF; // delta
        repn(i, 1, n) if (!vy[i] && sl[i]) dt = min(dt, sl[i]);
        repn(i, 1, n) {
            if (vx[i]) lx[i] -= dt;
            if (vy[i]) ly[i] += dt;
            else sl[i] -= dt;
        }
        l = r = 0;
        repn(i, 1, n) if (!vy[i] && !sl[i] && check(i)) return;
    }
}

inline void init() {
    fill(mx, mx + 1 + n, 0), fill(my, my + 1 + n, 0);
    fill(lx, lx + 1 + n, 0), fill(ly, ly + 1 + n, 0);
    repn(i, 1, n) repn(j, 1, n) w[i][j] = -INF;
}

inline void work() {
    repn(i, 1, n) repn(j, 1, n) lx[i] = max(lx[i], w[i][j]);
    repn(i, 1, n) bfs(i);
}

inline void output() {
    ll ans = 0;
    repn(i, 1, n) ans += lx[i] + ly[i];
    printf("%lld\n", ans); // 最大权
    repn(i, 1, n) printf("%d ", my[i]); puts("");
}

int main() {
    int T = 1;
    while (T--) {
        n = rd(), m = rd(), init();
        while (m--) x = rd(), y = rd(), w[x][y] = rd();
        work(), output();
    }
    return 0;
}
```

4.10 网络流

建模技巧

二分图带权最大独立集。给出一个二分图，每个结点上有一个正权值。要求选出一些点，使得这些点之间没有边相连，且权值和最大。

解： 在二分图的基础上添加源点 S 和汇点 T，然后从 S 向所有 X 集合中的点连一条边，所有 Y 集合中的点向 T 连一条边，容量均为该点的权值。X 结点与 Y 结点之间的边的容量均为无穷大。这样，对于图中的任意一个割，将割中的

边对应的结点删掉就是一个符合要求的解，权和为所有权减去割的容量。因此，只需要求出最小割，就能求出最大权和。

公平分配问题。把 m 个任务分配给 n 个处理器。其中每个任务有两个候选处理器，可以任选一个分配。要求所有处理器中，任务数最多的那个处理器所分配的任务数尽量少。不同任务的候选处理器集 $\{p_1, p_2\}$ 保证不同。

解： 本题有一个比较明显的二分图模型，即 X 结点是任务，Y 结点是处理器。二分答案 x，然后构图，首先从源点 S 出发向所有的任务结点引一条边，容量等于 1，然后从每个任务结点出发引两条边，分别到达它所能分配到的两个处理器结点，容量为 1，最后从每个处理器结点出发引一条边到汇点 T，容量为 x，表示选择该处理器的任务不能超过 x。这样网络中的每个单位流量都是从 S 流到一个任务结点，再到处理器结点，最后到汇点 T。只有当网络中的总流量等于 m 时才意味着所有任务都选择了一个处理器。这样，我们通过 $O(\log m)$ 次最大流便算出了答案。

区间 k 覆盖问题。数轴上有一些带权值的左闭右开区间。选出权和尽量大的一些区间，使得任意一个数最多被 k 个区间覆盖。

解： 本题可以用最小费用流解决，构图方法是把每个数作为一个结点，然后对于权值为 w 的区间 $[u, v)$ 加边 $u \to v$，容量为 1，费用为 $-w$。再对所有相邻的点加边 $i \to i+1$，容量为 k，费用为 0。最后，求最左点到最右点的最小费用最大流即可，其中每个流量对应一组互不相交的区间。如果数值范围太大，可以先进行离散化。

最大闭合子图。给定带权图 G（权值可正可负），求一个权和最大的点集，使得起点在该点集中的任意弧，终点也在该点集中。

解： 新增附加源 s 和附加汇 t，从 s 向所有正权点引一条边，容量为权值；从所有负权点向汇点引一条边，容量为权值的相反数。求出最小割以后，$S - \{s\}$ 就是最大闭合子图。

最大密度子图。给出一个无向图，找一个点集，使得这些点之间的边数除以点数的值（称为子图的密度）最大。

解： 如果两个端点都选了，就必然要选边，这就是一种推导。如果把每个点和每条边都看成新图中的结点，可以把问题转化为最大闭合子图。

4.10.1 EK

```
// EK  O(n * m * m)
struct Edge { int from, to, nx, w; } e[M << 1];
int h[N], pr[N], flow[N];
int n, m, s, t, x, y, cnt = 1; // 1
int q[N], l, r;
inline int bfs() {
    memset(flow, 0, sizeof(flow));
    l = r = 0, q[++r] = s;
    flow[s] = INT_MAX;
    while (l < r) {
        int u = q[++l];
        for (int i = h[u]; i; i = e[i].nx) {
            int v = e[i].to;
            if (!flow[v] && e[i].w > 0) {
                flow[v] = min(flow[u], e[i].w);
                pr[v] = i, q[++r] = v;
            }
        }
        if (flow[t]) break;
    }
    return flow[t];
}
inline ll ek() {
    int delta; ll ans = 0;
    while ((delta = bfs())) {
        ans += delta;
        for (int i = t; i ^ s; i = e[pr[i]].from) {
            e[pr[i]].w -= delta;
```

```
            e[pr[i] ^ 1].w += delta;
        }
    }
    return ans;
}
```

4.10.2 Dinic

```
// Dinic  O(n * n * m)
typedef long long ll;
namespace Dinic {
    int n, s, t, cnt; ll mxf;
    int h[N], cur[N], dep[N];
    struct Edge { int to, nx; ll f; } e[M];
    inline void add(int u, int v, ll f, bool o = 1) {
        e[++cnt] = (Edge) {v, h[u], f}, h[u] = cnt;
        if (o) e[++cnt] = (Edge) {u, h[v], 0}, h[v] = cnt;
    }
    int q[N], l, r;
    inline int bfs() {
        memset(dep, 0, (n + 1) << 2);
        dep[s] = 1, cur[s] = h[s];
        l = 0, q[r = 1] = s;
        while (l < r) {
            int u = q[++l];
            for (int i = h[u]; i; i = e[i].nx) {
                int v = e[i].to;
                if (e[i].f && !dep[v]) {
                    dep[v] = dep[u] + 1;
                    cur[v] = h[v];
                    q[++r] = v;
                }
            }
        }
        return dep[t];
    }
    ll dfs(int u, ll flow = LONG_MAX) {
        if (!flow || u == t) return flow;
        ll f, add = 0;
        for (int &i = cur[u]; i; i = e[i].nx) {
            int v = e[i].to;
            if (dep[v] == dep[u] + 1 && (f = dfs(v, min(flow, e[i].f)))) {
                e[i].f -= f, e[i ^ 1].f += f;
                flow -= f, add += f;
                if (!flow) break;
            }
        }
        return add;
    }
    inline void main() { while (bfs()) mxf += dfs(s); }
    inline void init(int _n, int _s, int _t) {
        n = _n, s = _s, t = _t, mxf = 0, cnt = 1;
        memset(h, 0, (n + 1) << 2);
    }
}

// 一个小技巧，跑完 Dinic 后 dep 不为 0 的点
// 表示最终与 s（源点）连通，在最小割中该点被划分到 s 那边
```

4.10.3 ISAP

```cpp
// ISAP  O(n * n * m)
typedef long long ll;
namespace ISAP {
    const int N = 2e5 + 5;
    const int M = 1e6 + 6;
    int n, s, t, cnt; ll mxf;
    int h[N], cur[N], dep[N], gap[N];
    struct Edge { int to, nx; ll f; } e[M];
    inline void add(int u, int v, ll f, bool o = 1) {
        e[++cnt] = (Edge) {v, h[u], f}, h[u] = cnt;
        if (o) e[++cnt] = (Edge) {u, h[v], 0}, h[v] = cnt;
    }
    int q[N], l, r;
    inline void bfs() {
        memset(dep, 0, (n + 1) << 2);
        memset(gap, 0, (n + 1) << 2);
        gap[dep[t] = 1] = 1, l = 0, q[r = 1] = t;
        while (l < r) {
            int u = q[++l];
            for (int i = h[u]; i; i = e[i].nx) {
                int v = e[i].to;
                if (!dep[v]) {
                    dep[v] = dep[u] + 1;
                    ++gap[dep[v]];
                    q[++r] = v;
                }
            }
        }
    }
    ll dfs(int u, ll flow = LONG_MAX) {
        if (!flow || u == t) return flow;
        ll f, add = 0;
        for (int &i = cur[u]; i; i = e[i].nx) { // & cur
            int v = e[i].to;
            if (dep[v] == dep[u] - 1) {
                if ((f = dfs(v, min(flow, e[i].f)))) {
                    e[i].f -= f, e[i ^ 1].f += f;
                    flow -= f, add += f;
                }
                if (!flow) return add;
            }
        }
        --gap[dep[u]] ? ++gap[++dep[u]] : dep[s] = n + 1;
        return add;
    }
    inline void main() {
        bfs();
        while (dep[s] <= n) {
            memcpy(cur, h, (n + 1) << 2);
            mxf += dfs(s);
        }
    }
    inline void init(int _n, int _s, int _t) {
        n = _n, s = _s, t = _t, mxf = 0, cnt = 1;
        memset(h, 0, (n + 1) << 2);
    }
}
```

```
int main() {
    int n = rd(), m = rd(), s = rd(), t = rd();
    ISAP::init(n, s, t);
    while (m--) {
        x = rd(), y = rd(), w = rd();
        ISAP::add(x, y, w);
    }
    printf("%lld\n", ISAP::main());
    return 0;
}
```

4.10.4 费用流

```
// 基于 SPFA & EK
// MCMF  O(n * m * maxflow)
typedef long long ll;
namespace MCMF {
    const int N = 5e3 + 3;
    const int M = 1e5 + 5;
    int n, s, t, cnt; ll mxf, mnc;
    int h[N], pr[N], pre[N], vis[N]; ll dis[N], flow[N]; // ll
    struct Edge { int to, nx; ll f, dis; } e[M];
    inline void add(int u, int v, ll f, ll w, bool o = 1) {
        e[++cnt] = (Edge) {v, h[u], f, w}, h[u] = cnt;
        if (o) e[++cnt] = (Edge) {u, h[v], 0, -w}, h[v] = cnt;
    }
    inline int spfa() {
        memset(flow, 127, (n + 1) << 3);
        memset(dis, 63, (n + 1) << 3);
        memset(vis, 0, (n + 1) << 2);
        dis[s] = 0, pre[t] = -1;
        queue<int> que; que.push(s);
        while (!que.empty()) {
            int u = que.front(); que.pop(); vis[u] = 0;
            for (int i = h[u]; i; i = e[i].nx) {
                int v = e[i].to;
                if (e[i].f && dis[v] > dis[u] + e[i].dis) {
                    flow[v] = min(flow[u], e[i].f);
                    dis[v] = dis[u] + e[i].dis;
                    pre[v] = u, pr[v] = i;
                    if (!vis[v]) vis[v] = 1, que.push(v);
                }
            }
        }
        return ~pre[t];
    }
    inline void main() {
        while (spfa()) {
            const ll f = flow[t];
            mxf += f, mnc += f * dis[t];
            for (int u = t; u ^ s; u = pre[u]) {
                e[pr[u]].f -= f, e[pr[u] ^ 1].f += f;
            }
        }
    }
    inline void init(int _n, int _s, int _t) {
        n = _n, s = _s, t = _t, mnc = mxf = 0, cnt = 1;
        memset(h, 0, (n + 1) << 2);
```

```
    }
}
int main() {
    int n = rd(), m = rd(), s = rd(), t = rd();
    MCMF::init(n, s, t);
    while (m--) {
        int u = rd(), v = rd(), f = rd(), w = rd();
        MCMF::add(u, v, f, w);
    }
    MCMF::main();
    printf("%d %lld\n", MCMF::mxf, MCMF::mnc);
    return 0;
}
```

4.10.5 上下界网络流建图方法

上下界网络流建图方法

记号说明

- $f(u,v)$ 表示 $u \to v$ 的实际流量
- $b(u,v)$ 表示 $u \to v$ 的流量下界
- $c(u,v)$ 表示 $u \to v$ 的流量上界

无源汇可行流

建图

- 新建附加源点 S 和 T
- 原图中的边 $u \to v$，限制为 $[b,c]$，建边 $u \to v$，容量为 $c-b$
- 记 $d(i) = \sum b(u,i) - \sum b(i,v)$
- 若 $d(i) > 0$，建边 $S \to i$，流量为 $d(i)$
- 若 $d(i) < 0$，建边 $i \to T$，流量为 $-d(i)$

求解

- 跑 $S \to T$ 的最大流，如果满流，则原图存在可行流。
- 此时，原图中每一条边的流量为新图中对应边的流量加上这条边的下界。

有源汇可行流

建图

- 在原图中建边 $t \to s$，流量限制为 $[0,+\infty)$，这样就改造成了无源汇的网络流图。
- 之后就可以像求解无源汇可行流一样建图了。

求解 同无源汇可行流

有源汇最大流

建图 同有源汇可行流

求解

- 先跑一遍 $S \to T$ 的最大流，求出可行流
- 记此时 $\sum f(s,i) = sum_1$
- 将 $t \to s$ 这条边拆掉，在新图上跑 $s \to t$ 的最大流
- 记此时 $\sum f(s,i) = sum_2$
- 最终答案即为 $sum_1 + sum_2$

有源汇最小流

建图 同无源汇可行流

求解

- 求 $S \to T$ 最大流
- 建边 $t \to s$，容量为 $+\infty$
- 再跑一遍 $S \to T$ 的最大流，答案即为 $f(t,s)$

有源汇的最大流和最小流也可以通过二分答案求得，即二分 $t \to s$ 的下界（最大流）和上界（最小流）复杂度多了个 $O(\log n)$ 这里不再赘述。

蓝书上的做法

- 先用无源汇可行流建图的方法求出可行流，然后用传统 $s-t$ 增广路算法即可得到最大流。把 t 看成源点，s 看成汇点后求出的 $t-s$ 最大流就是最小流。
- 注意：原先每条弧 $u \to v$ 的反向弧容量为 0，而在有容量下界的情形中，反向弧的容量应该等于流量下界。

有源汇费用流

建图

- 新建附加源点 S 和 T
- 原图中的边 $u \to v$，限制为 $[b,c]$，费用为 $cost$，建边 $u \to v$，容量为 $c-b$，费用为 $cost$
- 记 $d(i) = \sum b(u,i) - \sum b(i,v)$
- 若 $d(i) > 0$，建边 $S \to i$，流量为 $d(i)$，**费用为** 0
- 若 $d(i) < 0$，建边 $i \to T$，流量为 $-d(i)$，**费用为** 0
- 建边 $t \to s$，流量为 $+\infty$，费用为 0。

求解

- 跑 $S \to T$ 的最小费用最大流
- 答案为求出的费用加上原图中边的下界乘以边的费用

4.10.6 无源汇上下界可行流

```
// 上下界可行流
const int M = 2e6 + 6;
typedef long long ll;
namespace ISAP { }
namespace no_ok_flow {
    const int N = 2e5 + 5;
    int n, S, T; ll d[N], s;
    inline void add(int x, int y, ll l, ll r) {
        ISAP::add(x, y, r - l), d[y] += l, d[x] -= l;
    }
    inline bool main() {
        for (int i = 1; i <= n; ++i) {
            if (d[i] > 0) ISAP::add(S, i, d[i]), s += d[i];
            else if (d[i] < 0) ISAP::add(i, T, -d[i]);
        }
        ISAP::main();
        return ISAP::mxf == s;
    }
    inline void init(int _n) {
        n = _n, s = 0, S = n + 1, T = n + 2;
        memset(d, 0, (n + 1) << 3); // 3
        ISAP::init(n + 2, S, T);
```

```
    }
}
int n, m, l[M];
int main() {
    scanf("%d%d", &n, &m), no_ok_flow::init(n);
    for (int i = 1, x, y, l, r; i <= m; ++i) {
        scanf("%d%d%d%d", &x, &y, &l, &r);
        no_ok_flow::add(x, y, l, r), ::l[i] = l;
    }
    if (!no_ok_flow::main()) puts("NO"), exit(0);
    puts("YES"); // 可行
    for (int i = 1, j = 3; i <= m; ++i, j += 2) {
        printf("%lld\n", l[i] + ISAP::e[j].f); // 输出每条边流量
    }
    return 0;
}
```

4.10.7 有源汇上下界最大流

```
// 有源汇上下界最大流
typedef long long ll;
namespace ISAP { }
namespace no_ok_flow { }
namespace yes_max_flow {
    int n, S, T;
    inline bool main() {
        no_ok_flow::add(T, S, 0, LONG_MAX);
        if (!no_ok_flow::main()) return 0;
        ISAP::s = S, ISAP::t = T, ISAP::mxf = 0;
        return ISAP::main(), 1;
    }
    inline void init(int _n, int _s, int _t) {
        n = _n, S = _s, T = _t;
        no_ok_flow::init(n);
    }
}

int n, m, s, t;
int main() {
    scanf("%d%d%d%d", &n, &m, &s, &t);
    yes_max_flow::init(n, s, t);
    for (int i = 1, x, y, l, r; i <= m; ++i) {
        scanf("%d%d%d%d", &x, &y, &l, &r);
        no_ok_flow::add(x, y, l, r);
    }
    if (!yes_max_flow::main()) puts("please go home to sleep"), exit(0);
    printf("%lld\n", ISAP::mxf);
    return 0;
}
```

4.10.8 有源汇上下界最小流

```
// 有源汇上下界最小流
typedef long long ll;
namespace ISAP { }
namespace no_ok_flow { }
namespace yes_min_flow {
    int n, S, T;
```

```
    inline bool main() {
        no_ok_flow::main();
        no_ok_flow::add(T, S, 0, LONG_MAX);
        ISAP::main();
        return ISAP::mxf == no_ok_flow::s;
    }
    inline void init(int _n, int _s, int _t) {
        n = _n, S = _s, T = _t;
        no_ok_flow::init(n);
    }
}
int n, m, s, t;
int main() {
    scanf("%d%d%d%d", &n, &m, &s, &t);
    yes_min_flow::init(n, s, t);
    for (int i = 1, x, y, l, r; i <= m; ++i) {
        scanf("%d%d%d%d", &x, &y, &l, &r);
        no_ok_flow::add(x, y, l, r);
    }
    if (!yes_min_flow::main()) puts("please go home to sleep"), exit(0);
    printf("%lld\n", ISAP::e[ISAP::cnt].f);
    return 0;
}
```

4.10.9 无源汇上下界最小费用可行流

```
// 无源汇上下界最小费用可行流
// 有源汇做法: addedge(t, s, 0, INF, 0) 转为无源汇模型
typedef long long ll;
namespace MCMF { }
namespace no_min_cost_flow {
    const int N = 310;
    int n, S, T; ll d[N], s, sum;
    inline void add(int x, int y, ll l, ll r, ll w) {
        MCMF::add(x, y, r - l, w), d[y] += l, d[x] -= l, sum += l * w;
    }
    inline bool main() {
        for (int i = 1; i <= n; ++i) {
            if (d[i] > 0) MCMF::add(S, i, d[i], 0), s += d[i];
            else if (d[i] < 0) MCMF::add(i, T, -d[i], 0);
        }
        MCMF::main();
        return MCMF::mxf == s;
    }
    // 最终答案还需要加上 sum = 每条边的 (下界流量 * 费用)
    inline ll mnc() { return MCMF::mnc + sum; }
    inline void init(int _n) {
        n = _n, s = 0, S = n + 1, T = n + 2;
        MCMF::init(n + 2, S, T);
        memset(d, 0, (n + 1) << 3);
    }
}
int n, m, s, t;
int main() {
    using no_min_cost_flow::add;
    scanf("%d%d", &n, &m);
    no_min_cost_flow::init(node_tot);
    // addedge
```

```
33      add(t, s, 0, INF, 0); // 有源汇转无源汇
34      no_min_cost_flow::main();
35      printf("%lld\n", no_min_cost_flow::mnc());
36      return 0;
37  }
```

5 Computational Geometry

5.1 Basic Function

```
#define zero(x) ((fabs(x) < eps ? 1 : 0))
#define sgn(x) (fabs(x) < eps ? 0 : ((x) < 0 ? -1 : 1))

struct point
{
    double x, y;
    point(double a = 0, double b = 0) { x = a, y = b; }
    point operator-(const point& b) const { return point(x - b.x, y - b.y); }
    point operator+(const point& b) const { return point(x + b.x, y + b.y); }
    // 两点是否重合
    bool operator==(point& b) { return zero(x - b.x) && zero(y - b.y); }
    // 点积(以原点为基准)
    double operator*(const point& b) const { return x * b.x + y * b.y; }
    // 叉积(以原点为基准)
    double operator^(const point& b) const { return x * b.y - y * b.x; }
    // 绕P点逆时针旋转a弧度后的点
    point rotate(point b, double a)
    {
        double dx, dy;
        (*this - b).split(dx, dy);
        double tx = dx * cos(a) - dy * sin(a);
        double ty = dx * sin(a) + dy * cos(a);
        return point(tx, ty) + b;
    }
    // 点坐标分别赋值到a和b
    void split(double& a, double& b) { a = x, b = y; }
};
struct line
{
    point s, e;
    line() {}
    line(point ss, point ee) { s = ss, e = ee; }
};
```

5.2 Position

5.2.1 Point-Point

```
double dist(point a, point b) { return sqrt((a - b) * (a - b)); }
```

5.2.2 Line-Line

```
// <0, *> 表示重合; <1, *> 表示平行; <2, P> 表示交点是P;
pair<int, point> spoint(line l1, line l2)
{
    point res = l1.s;
    if (sgn((l1.s - l1.e) ^ (l2.s - l2.e)) == 0)
        return {sgn((l1.s - l2.e) ^ (l2.s - l2.e)) != 0, res};
    double t = ((l1.s - l2.s) ^ (l2.s - l2.e)) / ((l1.s - l1.e) ^ (l2.s - l2.e));
    res.x += (l1.e.x - l1.s.x) * t;
    res.y += (l1.e.y - l1.s.y) * t;
    return {2, res};
}
```

5.2.3 Segment-Segment

```
bool segxseg(line l1, line l2)
{
    return
        max(l1.s.x, l1.e.x) >= min(l2.s.x, l2.e.x) &&
        max(l2.s.x, l2.e.x) >= min(l1.s.x, l1.e.x) &&
        max(l1.s.y, l1.e.y) >= min(l2.s.y, l2.e.y) &&
        max(l2.s.y, l2.e.y) >= min(l1.s.y, l1.e.y) &&
        sgn((l2.s - l1.e) ^ (l1.s - l1.e)) * sgn((l2.e-l1.e) ^ (l1.s - l1.e)) <= 0 &&
        sgn((l1.s - l2.e) ^ (l2.s - l2.e)) * sgn((l1.e-l2.e) ^ (l2.s - l2.e)) <= 0;
}
```

5.2.4 Line-Segment

```
//l1是直线,l2是线段
bool segxline(line l1, line l2)
{
    return sgn((l2.s - l1.e) ^ (l1.s - l1.e)) * sgn((l2.e - l1.e) ^ (l1.s - l1.e)) <=
    0;
}
```

5.2.5 Point-Line

```
double pointtoline(point p, line l)
{
    point res;
    double t = ((p - l.s) * (l.e - l.s)) / ((l.e - l.s) * (l.e - l.s));
    res.x = l.s.x + (l.e.x - l.s.x) * t, res.y = l.s.y + (l.e.y - l.s.y) * t;
    return dist(p, res);
}
```

5.2.6 Point-Segment

```
double pointtosegment(point p, line l)
{
    point res;
    double t = ((p - l.s) * (l.e - l.s)) / ((l.e - l.s) * (l.e - l.s));
    if (t >= 0 && t <= 1)
        res.x = l.s.x + (l.e.x - l.s.x) * t, res.y = l.s.y + (l.e.y - l.s.y) * t;
    else
        res = dist(p, l.s) < dist(p, l.e) ? l.s : l.e;
    return dist(p, res);
}
```

5.2.7 Point on Segment

```
bool PointOnSeg(point p, line l)
{
    return
        sgn((l.s - p) ^ (l.e-p)) == 0 &&
        sgn((p.x - l.s.x) * (p.x - l.e.x)) <= 0 &&
        sgn((p.y - l.s.y) * (p.y - l.e.y)) <= 0;
}
```

5.3 Polygon

5.3.1 Area

```
double area(point p[], int n)
{
    double res = 0;
    for (int i = 0; i < n; i++) res += (p[i] ^ p[(i + 1) % n]) / 2;
    return fabs(res);
}
```

5.3.2 Point in Convex

```
// 点形成一个凸包，而且按逆时针排序(如果是顺时针把里面的<0改为>0)
// 点的编号 : [0,n)
// -1 : 点在凸多边形外
// 0  : 点在凸多边形边界上
// 1  : 点在凸多边形内
int PointInConvex(point a, point p[], int n)
{
    for (int i = 0; i < n; i++)
        if (sgn((p[i] - a) ^ (p[(i + 1) % n] - a)) < 0)
            return -1;
        else if (PointOnSeg(a, line(p[i], p[(i + 1) % n])))
            return 0;
    return 1;
}
```

5.3.3 Point in Polygon

```
// 射线法,poly[]的顶点数要大于等于3,点的编号0~n-1
// -1 : 点在凸多边形外
// 0  : 点在凸多边形边界上
// 1  : 点在凸多边形内
int PointInPoly(point p, point poly[], int n)
{
    int cnt;
    line ray, side;
    cnt = 0;
    ray.s = p;
    ray.e.y = p.y;
    ray.e.x = -100000000000.0; // -INF,注意取值防止越界
    for (int i = 0; i < n; i++)
    {
        side.s = poly[i], side.e = poly[(i + 1) % n];
        if (PointOnSeg(p, side)) return 0;
        //如果平行轴则不考虑
        if (sgn(side.s.y - side.e.y) == 0)
            continue;
        if (PointOnSeg(sid e.s, r ay))
            cnt += (sgn(side.s.y - side.e.y) > 0);
        else if (PointOnSeg(side.e, ray))
            cnt += (sgn(side.e.y - side.s.y) > 0);
        else if (segxseg(ray, side))
            cnt++;
    }
    return cnt % 2 == 1 ? 1 : -1;
}
```

5.3.4 Judge Convex

```
//点可以是顺时针给出也可以是逆时针给出
//点的编号1~n-1
bool isconvex(point poly[], int n)
{
    bool s[3];
    memset(s, 0, sizeof(s));
    for (int i = 0; i < n; i++)
    {
        s[sgn((poly[(i + 1) % n] - poly[i]) ^ (poly[(i + 2) % n] - poly[i])) + 1] = 1;
        if (s[0] && s[2]) return 0;
    }
    return 1;
}
```

5.4 Integer Points

5.4.1 On Segment

```
int OnSegment(line l) { return __gcd(fabs(l.s.x - l.e.x), fabs(l.s.y - l.e.y)) + 1; }
```

5.4.2 On Polygon Edge

```
int OnEdge(point p[], int n)
{
    int i, ret = 0;
    for (i = 0; i < n; i++)
        ret += __gcd(fabs(p[i].x - p[(i + 1) % n].x), fabs(p[i].y - p[(i + 1) % n].y));
    return ret;
}
```

5.4.3 Inside Polygon

```
int InSide(point p[], int n)
{
    int i, area = 0;
    for (i = 0; i < n; i++)
        area += p[(i + 1) % n].y * (p[i].x - p[(i + 2) % n].x);
    return (fabs(area) - OnEdge(p, n)) / 2 + 1;
}
```

5.5 Circle

5.5.1 Circumcenter

```
point waixin(point a, point b, point c)
{
    double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1 * a1 + b1 * b1) / 2;
    double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2 * a2 + b2 * b2) / 2;
    double d = a1 * b2 - a2 * b1;
    return point(a.x + (c1 * b2 - c2 * b1) / d, a.y + (a1 * c2 - a2 * c1) / d);
}
```

5.6 RuJia Liu's

5.6.1 Point

```
struct Point
{
    double x, y;
    Point(double x = 0, double y = 0) : x(x), y(y) {}
};

typedef Point Vector;

//向量+向量=向量，点+向量=点
Vector operator+(Vector A, Vector B) { return Vector(A.x + B.x, A.y + B.y); }
//点-点=向量
Vector operator-(Point A, Point B) { return Vector(A.x - B.x, A.y - B.y); }
//向量*数=向量
Vector operator*(Vector A, double p) { return Vector(A.x * p, A.y * p); }
//向量/数=向量
Vector operator/(Vector A, double p) { return Vector(A.x / p, A.y / p); }

bool operator<(const Point& a, const Point& b)
{
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}

const double eps = 1e-10;
double dcmp(double x)
{
    if (fabs(x) < eps)
        return 0;
    else
        return x < 0 ? -1 : 1;
}

bool operator==(const Point& a, const Point& b)
{
    return dcmp(a.x - b.x) == 0 && dcmp(a.y - b.y) == 0;
}

/*
 * 基本运算:
 * 点积
 * 叉积
 * 向量旋转
 */
double Dot(Vector A, Vector B) { return A.x * B.x + A.y * B.y; }
double Length(Vector A) { return sqrt(Dot(A, A)); }
double Angle(Vector A, Vector B) { return acos(Dot(A, B) / Length(A) / Length(B)); }

double Cross(Vector A, Vector B) { return A.x * B.y - A.y * B.x; }
double Area2(Point A, Point B, Point C) { return Cross(B - A, C - A); }

//rad是弧度
Vector Rotate(Vector A, double rad)
{
    return Vector(A.x * cos(rad) - A.y * sin(rad),
                  A.x * sin(rad) + A.y * cos(rad));
}
```

```
56
57  //调用前请确保A不是零向量
58  Vector Normal(Vector A)
59  {
60      double L = Length(A);
61      return Vector(-A.y / L, A.x / L);
62  }
63
64  /*
65   * 点和直线:
66   * 两直线交点
67   * 点到直线的距离
68   * 点到线段的距离
69   * 点在直线上的投影
70   * 线段相交判定
71   * 点在线段上判定
72   */
73
74  //调用前保证两条直线P+tv和Q+tw有唯一交点。当且仅当Cross(v, w)非0
75  Point GetLineIntersection(Point P, Vector v, Point Q, Vector w)
76  {
77      Vector u = P - Q;
78      double t = Cross(w, u) / Cross(v, w);
79      return P + v * t;
80  }
81
82  double DistanceToLine(Point P, Point A, Point B)
83  {
84      Vector v1 = B - A, v2 = P - A;
85      return fabs(Cross(v1, v2)) / Length(v1); //如果不取绝对值，得到的是有向距离
86  }
87
88  double DistanceToSegment(Point P, Point A, Point B)
89  {
90      if (A == B) return Length(P - A);
91      Vector v1 = B - A, v2 = P - A, v3 = P - B;
92      if (dcmp(Dot(v1, v2)) < 0) return Length(v2);
93      if (dcmp(Dot(v1, v3)) > 0) return Length(v3);
94      return fabs(Cross(v1, v2)) / Length(v1);
95  }
96
97  Point GetLineProjection(Point P, Point A, Point B)
98  {
99      Vector v = B - A;
100     return A + v * (Dot(v, P - A) / Dot(v, v));
101 }
102
103 bool SegmentProperIntersection(Point a1, Point a2, Point b1, Point b2)
104 {
105     double c1 = Cross(a2 - a1, b1 - a1), c2 = Cross(a2 - a1, b2 - b1),
106            c3 = Cross(b2 - b1, a1 - b1), c4 = Cross(b2 - b1, a2 - b1);
107     return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) < 0;
108 }
109
110 bool OnSegment(Point p, Point a1, Point a2)
111 {
112     return dcmp(Cross(a1 - p, a2 - p)) == 0 && dcmp(Dot(a1 - p, a2 - p)) < 0;
113 }
```

5.6.2 Circle

```
struct Line
{
    Point p;    //直线上任意一点
    Vector v;   //方向向量。它的左边就是对应的半平面
    double ang; //极角。即从x正半轴旋转到向量v所需要的角（弧度）
    Line() {}
    Line(Point p, Vector v) : p(p), v(v) { ang = atan2(v.y, v.x); }
    bool operator<(const Line& L) const // 排序用的比较运算符
    {
        return ang < L.ang;
    }
    Point point(double t) { return p + v * t; }
};

struct Circle
{
    Point c;
    double r;
    Circle(Point c, double r) : c(c), r(r) {}
    Point point(double a) { return c.x + cos(a) * r, c.y + sin(a) * r; }
};

int getLineCircleIntersection(Line L, Circle C, double& t1, double& t2, vector<Point>&
    sol)
{
    double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;
    double e = a * a + c * c, f = 2 * (a * b + c * d), g = b * b + d * d - C.r * C.r;
    double delta = f * f - 4 * e * g; //判别式
    if (dcmp(delta) < 0) return 0;    //相离
    if (dcmp(delta) == 0)             //相切
    {
        t1 = t2 = -f / (2 * e);
        sol.push_back(L.point(t1));
        return 1;
    }
    //相交
    t1 = (-f - sqrt(delta)) / (2 * e);
    t2 = (-f + sqrt(delta)) / (2 * e);
    sol.push_back(t1);
    sol.push_back(t2);
    return 2;
}

double angle(Vector v) { return atan2(v.y, v.x); }

int getCircleCircleIntersection(Circle C1, Circle C2, vector<Point>& sol)
{
    double d = Length(C1.c - C2.c);
    if (dcmp(d) == 0)
    {
        if (dcmp(C1.r - C2.r) == 0) return -1; //两圆重合
        return 0;
    }
    if (dcmp(C1.r + C2.r - d) < 0) return 0;       //内含
    if (dcmp(fabs(C1.r - C2.r) - d) > 0) return 0; //外离

    double a = angle(C2.c - C1.c); //向量C1C2的极角
```

```
57      double da = acos((C1.r * C1.r + d * d - C2.r * C2.r) / (2 * C1.r * d));
58      //C1C2到C1P1的角
59      Point p1 = C1.point(a - da), p2 = C1.point(a + da);
60
61      sol.push_back(p1);
62      if (p1 == p2) return 1;
63      sol.push_back(p2);
64      return 2;
65  }
66
67  //过点p到圆C的切线，v[i]是第i条切线的向量，返回切线条数
68  int getTangents(Point p, Circle C, Vector* v)
69  {
70      Vector u = C.c - p;
71      double dist = Length(u);
72      if (dist < C.r)
73          return 0;
74      else if (dcmp(dist - C.r) == 0)
75      { //p在圆上，只有一条切线
76          v[0] = Rotate(u, M_PI / 2);
77          return 1;
78      }
79      else
80      {
81          double ang = asin(C.r / dist);
82          v[0] = Rotate(u, -ang);
83          v[1] = Rotate(u, +ang);
84          return 2;
85      }
86  }
87
88  //两圆的公切线
89  //返回切线的条数。-1表示无穷条切线。
90  //a[i]和b[i]分别是第i条切线在圆A和圆B上的切点
91  int getTangents(Circle A, Circle B, Point* a, Point* b)
92  {
93      int cnt = 0;
94      if (A.r < B.r)
95      {
96          swap(A, B);
97          swap(a, b);
98      }
99      int d2 = (A.c.x - B.c.x) * (A.c.x - B.c.x) + (A.c.y - B.c.y) * (A.c.y - B.c.y);
100     int rdiff = A.r - B.r;
101     int rsum = A.r + B.r;
102     if (d2 < rdiff * rdiff) return 0; //内含
103     double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
104     if (d2 == 0 && A.r == B.r) return -1; //无限多条切线
105     if (d2 == rdiff * rdiff)
106     { //内切，一条切线
107         a[cnt] = A.point(base);
108         b[cnt] = B.point(base);
109         cnt++;
110         return 1;
111     }
112     //有外共切线
113     double ang = acos((A.r - B.r) / sqrt(d2));
114     a[cnt] = A.point(base + ang);
115     b[cnt] = B.point(base + ang);
```

```
    cnt++;
    a[cnt] = A.point(base + ang);
    b[cnt] = B.point(base - ang);
    cnt++;
    if (d2 == rsum * rsum)
    {
        a[cnt] = A.point(base);
        b[cnt] = B.point(M_PI + base);
        cnt++;
    }
    else if (d2 > rsum * rsum)
    {
        double ang = acos((A.r + B.r) / sqrt(d2));
        a[cnt] = A.point(base + ang);
        b[cnt] = B.point(M_PI + base + ang);
        cnt++;
        a[cnt] = A.point(base - ang);
        b[cnt] = B.point(M_PI + base - ang);
        cnt++;
    }
    return cnt;
}

//三角形外接圆（三点保证不共线）
Circle CircumscribedCircle(Point p1, Point p2, Point p3)
{
    double Bx = p2.x - p1.x, By = p2.y - p1.y;
    double Cx = p3.x - p1.x, Cy = p3.y - p1.y;
    double D = 2 * (Bx * Cy - By * Cx);
    double cx = (Cy * (Bx * Bx + By * By) - By * (Cx * Cx + Cy * Cy)) / D + p1.x;
    double cy = (Bx * (Cx * Cx + Cy * Cy) - Cx * (Bx * Bx + By * By)) / D + p1.y;
    Point p = Point(cx, cy);
    return Circle(p, Length(p1 - p));
}

//三角形内切圆
Circle InscribedCircle(Point p1, Point p2, Point p3)
{
    double a = Length(p2 - p3);
    double b = Length(p3 - p1);
    double c = Length(p1 - p2);
    Point p = (p1 * a + p2 * b + p3 * c) / (a + b + c);
    return Circle(p, DistanceToLine(p, p1, p2));
}
```

5.6.3 Polygon

```
typedef vector<Point> Polygon;
//多边形的有向面积
double PolygonArea(Polygon po)
{
    int n = po.size();
    double area = 0.0;
    for (int i = 1; i < n - 1; i++)
        area += Cross(po[i] - po[0], po[i + 1] - po[0]);
    return area / 2;
}

```

```cpp
//点在多边形内判定
int isPointInPolygon(Point p, Polygon poly)
{
    int wn = 0; //绕数
    int n = poly.size();
    for (int i = 0; i < n; i++)
    {
        if (OnSegment(p, poly[i], poly[(i + 1) % n])) return -1; //边界上
        int k = dcmp(Cross(poly[(i + 1) % n] - poly[i], p - poly[i]));
        int d1 = dcmp(poly[i].y - p.y);
        int d2 = dcmp(poly[(i + 1) % n].y - p.y);
        if (k > 0 && d1 <= 0 && d2 > 0) wn++;
        if (k < 0 && d2 <= 0 && d1 > 0) wn--;
    }
    if (wn != 0) return 1; //内部
    return 0;              //外部
}

//凸包(Andrew算法)
//如果不希望在凸包的边上有输入点，把两个 <= 改成 <
//如果不介意点集被修改，可以改成传递引用
Polygon ConvexHull(vector<Point> p)
{
    sort(p.begin(), p.end());
    p.erase(unique(p.begin(), p.end()), p.end());
    int n = p.size(), m = 0;
    Polygon res(n + 1);
    for (int i = 0; i < n; i++)
    {
        while (m > 1 && Cross(res[m - 1] - res[m - 2], p[i] - res[m - 2]) <= 0) m--;
        res[m++] = p[i];
    }
    int k = m;
    for (int i = n - 2; i >= 0; i--)
    {
        while (m > k && Cross(res[m - 1] - res[m - 2], p[i] - res[m - 2]) <= 0) m--;
        res[m++] = p[i];
    }
    m -= n > 1;
    res.resize(m);
    return res;
}

//半平面交
vector<Point> HalfplaneIntersection(vector<Line>& L)
{
    int n = L.size();
    sort(L.begin(), L.end()); // 按极角排序

    int first, last;    // 双端队列的第一个元素和最后一个元素的下标
    vector<Point> p(n); // p[i]为q[i]和q[i+1]的交点
    vector<Line> q(n);  // 双端队列
    vector<Point> ans;  // 结果

    q[first = last = 0] = L[0]; // 双端队列初始化为只有一个半平面L[0]
    for (int i = 1; i < n; i++)
    {
        while (first < last && !OnLeft(L[i], p[last - 1])) last--;
        while (first < last && !OnLeft(L[i], p[first])) first++;
```

```
71          q[++last] = L[i];
72          if (fabs(Cross(q[last].v, q[last - 1].v)) < eps)
73          { // 两向量平行且同向，取内侧的一个
74              last--;
75              if (OnLeft(q[last], L[i].p)) q[last] = L[i];
76          }
77          if (first < last) p[last - 1] = GetLineIntersection(q[last - 1], q[last]);
78      }
79      while (first < last && !OnLeft(q[first], p[last - 1])) last--; // 删除无用平面
80      if (last - first <= 1) return vector<Point>();                 // 空集
81      p[last] = GetLineIntersection(q[last], q[first]);              // 计算首尾两个半平面的
        交点
82
83      return vector<Point>(q.begin() + first, q.begin() + last + 1);
84  }
```

5.7 Convex$_H ull$

5.7.1 Convex$_H ull$

```
1   #include <bits/stdc++.h>
2   using namespace std;
3
4   #define N 500000
5   const double PI = acos(-1.0);
6   // #define PI 3.14159265357
7
8   struct Point {
9       double x, y;
10      Point(double x = 0, double y = 0): x(x), y(y) {}
11      Point operator + (const Point &rhs) const {
12          return Point(x + rhs.x, y + rhs.y);
13      }
14      Point operator - (const Point &rhs) const {
15          return Point(x - rhs.x, y - rhs.y);
16      }
17      double operator * (const Point &rhs) const {
18          return x * rhs.x + y * rhs.y;
19      }
20      double operator ^ (const Point &rhs) const {
21          return x * rhs.y - y * rhs.x;
22      }
23  }p[N], st[N];
24
25  typedef Point Vector;
26
27  Vector Rotate(Vector A, double rad) {
28      return Vector(A.x * cos(rad) - A.y * sin(rad), A.x * sin(rad) + A.y * cos(rad));
29  }
30
31  namespace Convex_Hull {
32      int n, tp = 0;
33      double cross(Point p1, Point p2, Point p0) {
34          return (p1 - p0) ^ (p2 - p0);
35      }
36      double dist(Point p1, Point p2) {
37          return sqrt((p1 - p2) * (p1 - p2));
38      }
39      bool cmp(Point p1, Point p2) {
```

```
40          double tmp = cross(p1, p2, p[1]);
41          if (tmp > 0) return 1;
42          else if (tmp == 0 && dist(p1, p[1]) < dist(p2, p[1])) return 1;
43          return 0;
44      }
45      void insert(Point u) {
46          n++;
47          p[n].x = u.x, p[n].y = u.y;
48          if (p[n].y < p[1].y || (p[n].y == p[1].y && p[n].x < p[1].x)) {
49              swap(p[1], p[n]);
50          }
51      }
52      void graham() {
53          sort(p + 2, p + n + 1, cmp);
54          st[1] = p[1]; tp = 1;
55          // n++; p[n] = p[1];
56          for (int i = 2; i <= n; ++i) {
57              while (tp > 1 && cross(st[tp], p[i], st[tp - 1]) < 0) tp--;
58              st[++tp] = p[i];
59          } st[tp + 1] = p[1];
60      }
61      double area() {
62          double ans = 0.0;
63          for (int i = 2; i < tp; ++i) {
64              ans += (st[i] - st[1]) ^ (st[i + 1] - st[1]);
65          }
66          return ans / 2;
67      }
68  }
69
70  double torad(double deg) {
71      return deg / 180 * PI;
72  }
73
74  int n;
75  double x, y, theta, w, h;
76
77  int main() {
78      int t; scanf("%d", &t);
79      while (t--) {
80          scanf("%d", &n);
81          double area = 0;
82          Convex_Hull::n = Convex_Hull::tp = 0;
83          for (int i = 0; i < n; ++i) {
84              scanf("%lf%lf%lf%lf%lf", &x, &y, &w, &h, &theta);
85              area += w * h;
86              Point u = Point(x, y);
87              double ang = -torad(theta);
88              Convex_Hull::insert(u + Rotate(Vector(-w / 2, -h / 2), ang));
89              Convex_Hull::insert(u + Rotate(Vector(w / 2, h / 2), ang));
90              Convex_Hull::insert(u + Rotate(Vector(-w / 2, h / 2), ang));
91              Convex_Hull::insert(u + Rotate(Vector(w / 2, -h / 2), ang));
92          }
93          Convex_Hull::graham();
94          printf("%.1lf %%\n", area * 100.0 / Convex_Hull::area());
95      }
96      return 0;
97  }
```

5.7.2 Half$_{P}$lane$_{I}$nsertion

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 1000;
const double eps = 1e-10;

struct Point {
    double x, y;
    Point(double x = 0, double y = 0): x(x), y(y) {}
    Point operator + (const Point &rhs) const {
        return Point(x + rhs.x, y + rhs.y);
    }
    Point operator - (const Point &rhs) const {
        return Point(x - rhs.x, y - rhs.y);
    }
    Point operator * (double p) const {
        return Point(x * p, y * p);
    }
    double operator * (const Point &rhs) const {
        return x * rhs.x + y * rhs.y;
    }
    double operator ^ (const Point &rhs) const {
        return x * rhs.y - y * rhs.x;
    }
}p[N], q[N];
// p初始的点集
// q生成的点集

typedef Point Vec;

struct Line {
    Point A;
    Vec B;
    double K;
    Line(Point A = Point(), Point B = Point()): A(A), B(B) {K = atan2(B.y, B.x);}
    bool operator < (const Line &rhs) const {
        return K < rhs.K;
    }
}P[N], Q[N];
//初始边集
//Q队列里边集

inline Point get(Line A, Line B) {
    Vec C = A.A - B.A;
    double t = (B.B ^ C) / (A.B ^ B.B);
    return A.A + A.B * t;
}

int l, r;

int n, m, tot;

inline void HalfPlaneInseration() {
    l = r = 1;
    Q[1] = P[1];
    for (int i = 2; i <= tot; ++i) {
        while (l < r && (P[i].B ^ (q[r - 1] - P[i].A)) <= eps) --r;
```

```
        while (l < r && (P[i].B ^ (q[l] - P[i].A)) <= eps) ++l;
        Q[++r] = P[i];
        if (fabs(Q[r].B ^ Q[r - 1].B) <= eps) {
            --r;
            if ((Q[r].B ^ (P[i].A - Q[r].A)) > eps) Q[r] = P[i];
        }
        if (l < r) {
            q[r - 1] = get(Q[r - 1], Q[r]);
        }
    }
    while (l < r && (Q[l].B ^ (q[r - 1] - Q[l].A)) <= eps) --r;
    if (r - l <= 1) return;
    q[r] = get(Q[l], Q[r]);
}

inline void solve() {
    double area = 0.0;
    for (int i = l + 1; i < r; ++i) {
        area += (q[i] - q[l]) ^ (q[i + 1] - q[l]);
    }
    printf("%.3lf\n", area / 2);
}

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &m);
        for (int j = 1; j <= m; ++j) {
            scanf("%lf %lf", &p[j].x, &p[j].y);
        }
        for (int j = 1; j <= m; ++j) {
            int to = j + 1;
            if (j == m) to = 1;
            P[++tot] = Line(p[j], p[to] - p[j]);
        }
    }
    sort(P + 1, P + tot + 1);
    HalfPlaneInseration();
    solve();
    return 0;
}
```

5.7.3 旋转卡壳

```
#include <bits/stdc++.h>
using namespace std;

#define N 500000
#define ll long long
#define PI 3.14159265358

namespace Convex_Hull {
    struct Point {
        double x, y;
        Point(double x = 0, double y = 0): x(x), y(y) {}
        Point operator - (const Point &rhs) const {
            return Point(x - rhs.x, y - rhs.y);
        }
```

```
15          double operator * (const Point &rhs) const {
16              return x * rhs.x + y * rhs.y;
17          }
18          double operator ^ (const Point &rhs) const {
19              return x * rhs.y - y * rhs.x;
20          }
21      }p[N], st[N];
22      int n, tp = 0;
23      double cross(Point p1, Point p2, Point p0) {
24          return (p1 - p0) ^ (p2 - p0);
25      }
26      double dist(Point p1, Point p0) {
27          return sqrt((p1 - p0) * (p1 - p0));
28      }
29      ll dist2(Point p1, Point p2) {
30          return (p1 - p2) * (p1 - p2);
31      }
32      bool cmp(Point p1, Point p2) {
33          double tmp = cross(p1, p2, p[1]);
34          if (tmp > 0) return 1;
35          if (tmp == 0 && dist(p1, p[1]) < dist(p2, p[1])) return 1;
36          return 0;
37      }
38      void insert(double x, double y) {
39          n++;
40          p[n].x = x, p[n].y = y;
41          if (p[n].y < p[1].y || (p[n].y == p[1].y && p[n].x < p[1].x)) {
42              Point t = p[1]; p[1] = p[n]; p[n] = t;
43          }
44      }
45      void graham() {
46          sort(p + 2, p + n + 1, cmp);
47          st[1] = p[1]; st[2] = p[2];
48          tp = 2;
49          for (int i = 3; i <= n; ++i) {
50              while (tp > 1 && cross(st[tp], p[i], st[tp - 1]) <= 0) tp--;
51              st[++tp] = p[i];
52          }
53          st[tp + 1] = p[1];
54      }
55      double solve() {
56          double ans = 0.0;
57          for (int i = 2; i <= tp; ++i) {
58              ans = ans + dist(st[i], st[i - 1]);
59          }
60          return ans;
61      }
62      ll get_diameter() {
63          if (tp == 1) return 0;
64          if (tp == 2) {
65              return dist2(st[1], st[2]);
66          }
67          int j = 2;
68          ll ans = 0;
69          for (int i = 1; i <= tp; ++i) {
70              while (cross(st[i], st[i + 1], st[j]) <= cross(st[i], st[i + 1], st[j + 1])
    )
71                  j = (j == tp ? 1 : j + 1);
72              ans = max(ans, max(dist2(st[j], st[i]), dist2(st[j], st[i + 1])));
```

```
        }
        return ans;
    }
}

int n, L;
double a, b, r;
double x, y, theta;

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        int u, v;
        scanf("%d%d", &u, &v);
        Convex_Hull::insert(u, v);
    }
    Convex_Hull::graham();
    printf("%lld\n", Convex_Hull::get_diameter());
    return 0;
}
```

5.8 ScanLine

5.8.1 ScanLine

```
#include <cstdio>
#include <cmath>
#include <algorithm>

using namespace std;

#define ls (rt << 1)
#define rs (rt << 1 | 1)

typedef long long ll;
const int N = 1e6;

int n;
ll u1, v1, u2, v2, X[N << 1], ans;

struct Scanline {
    ll l, r, h;
    int f;
    Scanline(ll l = 0, ll r = 0, ll h = 0, int f = 0) :
        l(l), r(r), h(h), f(f) {}
    bool operator < (const Scanline &rhs) const {
        return h < rhs.h;
    }
}Line[N << 1];

struct node {
    int l, r, sum;
    ll len;
}tree[N << 2];

inline void build(int rt, int l, int r) {
    tree[rt].l = l, tree[rt].r = r;
    if (l == r) return ;
    int mid = l + ((r - l) >> 1);
```

```
35      build(ls, l, mid);
36      build(rs, mid + 1, r);
37      return ;
38  }
39
40  inline void pushup(int rt) {
41      int l = tree[rt].l, r = tree[rt].r;
42      if (tree[rt].sum) {
43          tree[rt].len = X[r + 1] - X[l];
44      } else {
45          tree[rt].len = tree[ls].len + tree[rs].len;
46      }
47  }
48
49  inline void update(int rt, int L, int R, int f) {
50      int l = tree[rt].l, r = tree[rt].r;
51      if (X[l] >= R || X[r + 1] <= L) return ;
52      if (L <= X[l] && X[r + 1] <= R) {
53          tree[rt].sum += f;
54          pushup(rt);
55          return ;
56      }
57      update(ls, L, R, f);
58      update(rs, L, R, f);
59      pushup(rt);
60  }
61
62  int main() {
63      scanf("%d", &n);
64      for (int i = 1; i <= n; ++i) {
65          scanf("%lld %lld %lld %lld", &u1, &v1, &u2, &v2);
66          Line[2 * i - 1] = Scanline(u1, u2, v1, 1);
67          Line[2 * i] = Scanline(u1, u2, v2, -1);
68          X[2 * i - 1] = u1, X[2 * i] = u2;
69      }
70      n <<= 1;
71      sort(X + 1, X + n + 1);
72      sort(Line + 1, Line + n + 1);
73      int tot = unique(X + 1, X + n + 1) - X - 1;
74      build(1, 1, tot - 1);
75      for (int i = 1; i < n; ++i) {
76          update(1, Line[i].l, Line[i].r, Line[i].f);
77          ans += tree[1].len * (Line[i + 1].h - Line[i].h);
78      }
79      printf("%lld\n", ans);
80      return 0;
81  }
```

6 Dynamic Programming

6.1 序列

6.1.1 最大子序列和

```
// 传入序列a和长度n，返回最大子序列和
int MaxSeqSum(int a[], int n)
{
    int ans = 0, sum = 0;
    for (int i = 0; i < n; i++)
        sum += a[i], ans = max(sum, ans), sum = max(0, sum);
    return ans;
}
```

6.1.2 LIS

```
// 最长不上升子序列 LIS
// 序列下标从 1 开始，LIS()返回长度，序列存在lis[]中
const int N = "Edit";
int len, a[N], b[N], dp[N];
int Find(int p, int l, int r) {
    while (l <= r) {
        int mid = (l + r) >> 1;
        if (a[p] > b[mid]) l = mid + 1;
        else r = mid - 1;
    }
    return dp[p] = l;
}
int LIS(int lis[], int n) {
    int len = 1;
    dp[1] = 1, b[1] = a[1];
    for (int i = 2; i <= n; i++) {
        if (a[i] > b[len])
            b[++len] = a[i], dp[i] = len;
        else
            b[Find(i, 1, len)] = a[i];
    }
    for (int i = n, t = len; i && t; i--)
        if (dp[i] == t) lis[--t] = a[i];
    return len;
}

// 简单写法(下标从 0 开始，只返回长度)
int dp[N];
int LIS(int a[], int n) {
    memset(dp, 0x3f, sizeof(dp));
    for (int i = 0; i < n; i++) *lower_bound(dp, dp + n, a[i]) = a[i];
    return lower_bound(dp, dp + n, INF) - dp;
}
```

6.1.3 LCS

```
// 序列下标从1开始
int LCS(int a[], int b[], int n, int m) {
    memset(dp, 0, sizeof(dp));
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
```

```
            if (a[i] == b[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[n][m];
}
```

6.2 数位 DP

```
int a[20];
ll dp[20][state];
ll dfs(int pos, /*state变量*/, bool lead /*前导零*/, bool limit /*数位上界变量*/) {
    //递归边界，既然是按位枚举，最低位是0，那么pos==-1说明这个数枚举完了
    if (pos == -1) return 1;
    /*这里一般返回1，表示枚举的这个数是合法的，那么这里就需要在枚举时必须每一位都要满足题目条件，
    也就是说当前枚举到pos位，一定要保证前面已经枚举的数位是合法的。*/
    if (!limit && !lead && dp[pos][state] != -1) return dp[pos][state];
    /*常规写法都是在没有限制的条件记忆化，这里与下面记录状态是对应*/
    int up = limit ? a[pos] : 9; //根据limit判断枚举的上界up
    ll ans = 0;
    for (int i = 0; i <= up; i++) { //枚举，然后把不同情况的个数加到ans就可以了
        if () ...
        else if () ...
        ans += dfs(pos - 1, /*状态转移*/, lead && i == 0, limit && i == a[pos])
        //最后两个变量传参都是这样写的
        /*当前数位枚举的数是i，然后根据题目的约束条件分类讨论
        去计算不同情况下的个数，还有要根据state变量来保证i的合法性*/
    }
    //计算完，记录状态
    if (!limit && !lead) dp[pos][state] = ans;
    /*这里对应上面的记忆化，在一定条件下时记录，保证一致性，
    当然如果约束条件不需要考虑lead，这里就是lead就完全不用考虑了*/
    return ans;
}
ll solve(ll x) {
    int pos = 0;
    do //把数位都分解出来
        a[pos++] = x % 10;
    while (x /= 10);
    return dfs(pos - 1 /*从最高位开始枚举*/, /*一系列状态 */, true, true);
    //刚开始最高位都是有限制并且有前导零的，显然比最高位还要高的一位视为0
}
```

6.2.1 example

```
//洛谷P4124手机号码
//48不同时出现，至少有个数连续出现三次
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
int num[12]; // the number has 11 digits
ll dp[12][10][10][2][2][2][2];
//current pos, digt, value of higher pos, [value cur < value number], [one number
    appear at least 3 times], [4 exist], [8 exist];

inline ll f(int p, int a, int b, bool c, bool d, bool e4, bool e8) {
```

```
    if (e4 && e8) return 0;
    if (p == 1) return d;
    if (~dp[p][a][b][c][d][e4][e8]) return dp[p][a][b][c][d][e4][e8];
    int mx = (c ? 9 : num[p - 1]);
    ll ans = 0;
    for (int i = 0; i <= mx; ++i) {
        ans += f(p - 1, i, a, c || (i < mx), d || (i == a && i == b), e4 || (i == 4),
    e8 || (i == 8));
    }
    return dp[p][a][b][c][d][e4][e8] = ans;
}

inline ll calc(ll n) {
    if (n < 1e10) return 0;
    memset(dp, -1, sizeof(dp));
    int len = 0;
    while (n) num[++len] = n % 10, n /= 10;
    ll ans = 0;
    for (int i = 1; i <= num[len]; ++i) {
        ans += f(11, i, -1, i < num[len], 0, (i == 4), (i == 8));
    }
    return ans;
}
int main() {
    ll l, r;
    scanf("%lld%lld", &l, &r);
    printf("%lld\n", calc(r) - calc(l - 1));
    return 0;
}
```

6.3 区间 DP

```
for (int p = 1; p < n; ++p) { // 区间长度
    for (int i = 1, j = i + p; j < n2; j = ++i + p) { // 区间 [i, j]
        dp[i][j] = INF;
        for (int k = i; k < j; ++k) { // 分割点 k, 分为 [i, k] , [k + 1, j]
            dp[i][j] = min(dp[i][j], dp[i][k] + dp[k + 1][j] + /* 区间合并消耗 */ );
        }
    }
}
// 技巧: 断环为链, 复制原区间
```

6.4 树形 DP

6.4.1 换根 DP

```
// 先 dfs1 得到根的 dp 值, 以其他结点为根的 dp 值通过 dfs2 转移
void dfs1(int u, int pre) { /* dfs1 */ }
void dfs2(int u, int pre) {
    for (int &v : e[u]) {
        if (v != pre) {
            dp[v] = dp[u] + /* 转移 */;
            dfs2(v, u);
        }
    }
}
/* ... */ dfs1(1, 0); /* ... */ dfs2(1, 0); /* ... */
```

6.4.2 环形 DP

```
// 1: 固定 1 的状态，由 1 的状态确定 1 的 DP 初始值
// 2: 对 1 的每种状态跑一次 DP，并统计在 1 对应状态下 n 能取的 DP 值
{
    fill(dp, ..., ...), dp[1][0] = a0, dp[1][1] = b0, /* ... */; // init0
    DP(), ans0 = max(dp[n][...], dp[n][...], /* ... */);

    fill(dp, ..., ...), dp[1][0] = a1, dp[1][1] = b1, /* ... */; // init1
    DP(), ans1 = max(dp[n][...], dp[n][...], /* ... */);
    /* ... */
    ans = max(ans0, ans1, /* ... */, ansd); // 对 1 取每种状态的 DP 值取最值
}
```

6.4.3 基环树 DP

```
// 1: 拓扑排序
// 2: 对环上对点 dfs，存为连续的序列
// 3: 以环上每个点为根，对不在环上的点 dfs，得到这个根各状态的 DP 值
// 4: 对存下的序列跑环形 DP
int n, x, y, tot, cnt, ans;
int h[N], a[N], in[N], vis[N], f[N][2], dp[N][2];
void dfs1(int u, int pre) {
    vis[u] = 1, a[++tot] = u; // 存为序列
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (v != pre && !vis[v] && in[v] == 2) { dfs1(v, u); return; }
    }
}
void dfs2(int u, int pre) {
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (v != pre && !vis[v]) {
            dfs2(v, u);
            /* 状态转移 */
            // f[u][1] += f[v][0];
            // f[u][0] += max(f[v][0], f[v][1]);
        }
    }
}
{
    while (m--) /* addedge */
    tuopu();
    for (int i = 1; i <= n; ++i) {
        if (in[i] == 2) { dfs1(i, 0); break; } // 存为序列
    }
    for (int i = 1; i <= n; ++i) { // 得到环上点的 DP 值
        if (vis[i]) dfs2(i, 0);
    }
    /* 环形DP */
}
```

6.5 决策单调性优化

问题 设 $f(i)=\min(y[k]-s[i]\times x[k]), k\in[1,i-1]$, 现在要求出所有 $f(i), i\in[1,n]$

考虑两个决策 j 和 k，如果 j 比 k 优，则

$$y[j]-s[i]\times x[j]<y[k]-s[i]\times x[k]$$

化简得:

$$\frac{y_j - y_k}{x_j - x_k} < s_i$$

不等式左边是个斜率，我们把它设为 slope(j,k)

我们可以维护一个单调递增的队列，为什么呢?

因为如果 slope$(q[i-1],q[i])$ > slope$(q[i],q[i+1])$，那么当前者成立时，后者必定成立。即 $q[i]$ 决策优于 $q[i-1]$ 决策时，$q[i+1]$ 必然优于 $q[i]$，因此 $q[i]$ 就没有存在的必要了。所以我们要维护递增的队列。

那么每次的决策点 i，都要满足

$$\begin{cases} \text{slope}(q[i-1],q[i]) < s[i] \\ \text{slope}(q[i],q[i+1]) \geq s[i] \end{cases}$$

一般情况去二分这个 i 即可。

如果 $s[i]$ 是单调不降的，那么对于决策 j 和 $k(j<k)$ 来说，如果决策 k 优于决策 j，那么对于 $i \in [k+1,n]$，都存在决策 k 优于决策 j，因此决策 j 就可以舍弃了。这样的话我们可以用单调队列进行优化，可以少个 log。

单调队列滑动窗口最大值

```
// k为滑动窗口的大小
deque<int> q;
for (int i = 0, j = 0; i + k <= d; i++)
{
    while (j < i + k)
    {
        while (!q.empty() && a[q.back()] < a[j]) q.pop_back();
        q.push_back(j++);
    }
    while (q.front() < i) q.pop_front();
    // a[q.front()]为当前滑动窗口的最大值
}
```

6.5.1 Slope$_O$*ptimization*

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
const int N = 50005;

int n, L;
int q[N], h, t;
ll sum[N], dp[N];

inline ll a(int x) {return sum[x] + x;}
inline ll b(int x) {return sum[x] + x + L + 1;}
inline ll X(int x) {return b(x);}
inline ll Y(int x) {return dp[x] + b(x) * b(x);} //根据题意自定义X和Y
inline double slope(int i, int j) {return (Y(j) - Y(i)) / (X(j) - X(i));}

int main() {
    scanf("%d%d", &n, &L);
    for (int i = 1; i <= n; ++i) {
        scanf("%lld", &sum[i]);
        sum[i] += sum[i - 1];
    }
    h = t = 1;
    for (int i = 1; i <= n; ++i) {
        while (h < t && slope(q[h], q[h + 1]) < 2 * a(i)) h++; // a[i] = sum[i] + i 单调增，当前淘汰后不影响后续状态
```

```
        dp[i] = dp[q[h]] + 1ll * (a(i) - b(q[h])) * (a(i) - b(q[h]));
        while (h < t && slope(q[t], q[t - 1]) > slope(q[t], i)) t--; // 在凸包内的点，必然不会是最优解
        q[++t] = i;
    }
    printf("%lld\n", dp[n]);
    return 0;
}
```

6.6 多重背包优化

```
{ // 单调队列优化  O( n * V )
    scanf("%d%d", &n, &V);
    for (int i = 1; i <= n; ++i) {
        scanf("%d%d%d", &w, &v, &m); // 价值, 重量, 数量
        if (!v) { ans += m * w; continue; } // 重量为 0, 全选
        m = min(m, V / v);
        for (int d = 0; d < v; ++d) {
            l = 0, r = -1, k = (V - d) / v;
            for (int j = 0; j <= k; ++j) {
                now = dp[j * v + d] - j * w;
                while (l <= r && que[r] <= now) --r;
                num[++r] = j, que[r] = now;
                while (l <= r && num[l] < j - m) ++l;
                dp[j * v + d] = max(dp[j * v + d], que[l] + j * w);
            }
        }
    }
    printf("%d\n", ans + dp[V]);
}
{ // 二进制优化  O( n * V * ∑logm )
    scanf("%d%d", &n, &V);
    for (int i = 1; i <= n; ++i) {
        scanf("%d%d%d", &w, &v, &m);
        if (!v) { ans += m * w; continue; }
        for (p = 1; m > p; p <<= 1) {
            m -= p, t[++cnt] = (Node) {v * p, w * p};
        }
        t[++cnt] = (Node) {v * m, w * m};
    }
    for (int i = 1; i <= cnt; ++i) {
        for (int j = V; j >= t[i].v; --j) {
            dp[j] = max(dp[j], dp[j - t[i].v] + t[i].w);
        }
    }
    printf("%d\n", ans + dp[V]);
}
```

7 Offline Algorithm

7.1 莫队

7.1.1 普通莫队

```cpp
// 询问在区间 [l, r] 中大小在 [a, b] 中的数及数值的个数 (莫队+值域分块) P4396
// N, M, num ∈ [1, 1e5]

int n, m, l, r, x, y, B, mx;
int a[N], d[N], id[N], cnt[N], sum1[N], sum2[N], ans1[N], ans2[N];

struct Q {
    int l, r, a, b, p;
    bool operator < (const Q &b) const {
        // return d[l] ^ d[b.l] ? l < b.l : d[l] & 1 ? r < b.r : r > b.r;
        return d[l] ^ d[b.l] ? l < b.l : d[l] & 1 ? r < b.r : r > b.r;
        // 奇偶化排序优化, 一般能快 30 %, 有被手造数据卡的可能性
    }
} q[N];

inline void add(int x) {
    ++sum1[id[x]];
    if (++cnt[x] == 1) ++sum2[id[x]];
}

inline void del(int x) {
    --sum1[id[x]];
    if (--cnt[x] == 0) --sum2[id[x]];
}

inline int get_ans1(int l, int r) { // 统计出现过的数的个数
    int sid = id[l], eid = id[r], ans = 0;
    if (sid == eid) {
        for (int i = l; i <= r; ++i) ans += cnt[i];
        return ans;
    }
    for (int i = l; id[i] == sid; ++i) ans += cnt[i];
    for (int i = r; id[i] == eid; --i) ans += cnt[i];
    for (int i = sid + 1; i < eid; ++i) ans += sum1[i];
    return ans;
}

inline int get_ans2(int l, int r) { // 统计出现过的数值的个数
    int sid = id[l], eid = id[r], ans = 0;
    if (sid == eid) {
        for (int i = l; i <= r; ++i) ans += (bool)cnt[i];
        return ans;
    }
    for (int i = l; id[i] == sid; ++i) ans += (bool)cnt[i];
    for (int i = r; id[i] == eid; --i) ans += (bool)cnt[i];
    for (int i = sid + 1; i < eid; ++i) ans += sum2[i];
    return ans;
}

inline void Mo() {
    int l = 1, r = 0;
    sort(q + 1, q + 1 + m);
    for (int i = 1; i <= m; ++i) {
```

```
        const int &L = q[i].l, &R = q[i].r;
        while (l > L) add(a[--l]);
        while (r < R) add(a[++r]);
        while (l < L) del(a[l++]);
        while (r > R) del(a[r--]);
        ans1[q[i].p] = get_ans1(q[i].a, q[i].b);
        ans2[q[i].p] = get_ans2(q[i].a, q[i].b);
    }
}

int main() {
    n = rd(), m = rd(), B = n / sqrt(m) + 1; // +1 避免块长为 0
    for (int i = 1; i <= n; ++i) {
        a[i] = rd(), d[i] = i / B; // 区间分块
        // mx = mx < a[i] ? a[i] : mx; // 不需要, 只对查询值域分块即可
    }
    for (int i = 1; i <= m; ++i) {
        l = rd(), r = rd(), x = rd(), y = rd();
        q[i] = (Q) {l, r, x, y, i}; // 离线
        mx = mx < y ? y : mx;
    }
    B = sqrt(mx);
    for (int i = 1; i <= mx; ++i) {
        id[i] = (i - 1) / B + 1; // 值域分块
    }
    Mo();
    for (int i = 1; i <= m; ++i) {
        printf("%d %d\n", ans1[i], ans2[i]);
    }
    return 0;
}
```

7.1.2 树上莫队-路径

```
// 求路径上结点不同颜色数, 欧拉序

int n, m, l, r, B, rt, lca, res, _;
int st[N], ed[N], fa[N], dep[N], siz[N], top[N], son[N];
int a[N], h[N], id[N << 1], cnt[N], ans[N], vis[N], num[N << 1];

struct In {
    int x, d;
    bool operator < (const In &b) const {
        return x < b.x;
    }
} t[N];

struct Edge { int to, nx; } e[N << 1];

struct Q {
    int l, r, d, lca;
    bool operator < (const Q &b) const {
        return id[l] ^ id[b.l] ? l < b.l : id[l] & 1 ? r < b.r : r > b.r;
    }
} q[N];

inline void add(int u, int v) {
    static int cnt = 0;
```

```
25      e[++cnt].to = v;
26      e[cnt].nx = h[u];
27      h[u] = cnt;
28  }
29
30  inline void init() {
31      static int cnt = 0;
32      sort(t + 1, t + 1 + n);
33      for (int i = 1; i <= n; ++i) {
34          if (t[i].x ^ t[i - 1].x) ++cnt;
35          a[t[i].d] = cnt;
36      }
37      for (int i = 1; i <= n << 1; ++i) {
38          id[i] = (i - 1) / B + 1;
39      }
40  }
41
42  void dfs1(int u, int pre) {
43      static int cnt = 0;
44      dep[u] = dep[pre] + 1;
45      fa[u] = pre;
46      siz[u] = 1;
47      st[u] = ++cnt, num[cnt] = u; // 欧拉序
48      for (int i = h[u]; i; i = e[i].nx) {
49          int v = e[i].to;
50          if (v != pre) {
51              dfs1(v, u);
52              siz[u] += siz[v];
53              if (siz[son[u]] < siz[v]) {
54                  son[u] = v;
55              }
56          }
57      }
58      ed[u] = ++cnt, num[cnt] = u;; // 欧拉序
59  }
60
61  void dfs2(int u, int tpf) {
62      top[u] = tpf;
63      if (!son[u]) return;
64      dfs2(son[u], tpf);
65      for (int i = h[u]; i; i = e[i].nx) {
66          int v = e[i].to;
67          if (v != fa[u] && v != son[u]) {
68              dfs2(v, v);
69          }
70      }
71  }
72
73  inline int LCA(int x, int y) {
74      while (top[x] ^ top[y]) {
75          dep[top[x]] < dep[top[y]] ? y = fa[top[y]] : x = fa[top[x]];
76      }
77      return dep[x] > dep[y] ? y : x;
78  }
79
80  inline void add(int x) {
81      if (++cnt[a[x]] == 1) ++res;
82  }
83
```

```
inline void del(int x) {
    if (--cnt[a[x]] == 0) --res;
}

inline void move(int x) { //
    vis[x] ? del(x) : add(x);
    vis[x] ^= 1;
}

inline void Mo() {
    int l = 1, r = 0;
    sort(q + 1, q + 1 + m);
    for (int i = 1; i <= m; ++i) {
        const int &L = q[i].l, &R = q[i].r;
        while (l > L) move(num[--l]);
        while (r < R) move(num[++r]);
        while (l < L) move(num[l++]);
        while (r > R) move(num[r--]);
        if (q[i].lca) move(q[i].lca); //
        ans[q[i].d] = res;
        if (q[i].lca) move(q[i].lca); //
    }
}

int main() {
    srand(19260817);
    n = rd(), m = rd();
    B = n / sqrt(m) + 1;
    rt = rand() % n + 1;
    for (int i = 1; i <= n; ++i) {
        t[i] = (In) {rd(), i}; // 存下颜色进行离散化
    }
    init();
    for (int i = 1; i < n; ++i) {
        l = rd(), r = rd(); // 边
        add(l, r), add(r, l);
    }
    dfs1(rt, 0), dfs2(rt, rt);
    for (int i = 1; i <= m; ++i) {
        l = rd(), r = rd();
        if (st[l] > st[r]) _ = l, l = r, r = _; //
        lca = LCA(l, r);
        if (lca ^ l && lca ^ r) {
            q[i] = (Q) {ed[l], st[r], i, lca}; //
        } else {
            q[i] = (Q) {st[l], st[r], i}; //
        }
    }
    Mo();
    for (int i = 1; i <= m; ++i) {
        printf("%d\n", ans[i]);
    }
    return 0;
}
```

7.1.3 树上莫队-子树

```
// 求子树内出现次数 >= y 的数有多少种, dfn 序
```

```

int n, m, x, y, B;
int st[N], ed[N], cnt[N], tot[N];
int a[N], w[N], h[N], id[N], ans[N];

struct Q {
    int l, r, k, d;
    bool operator < (const Q &b) const {
        return id[l] ^ id[b.l] ? l < b.l : id[l] & 1 ? r < b.r : r > b.r;
    }
} q[N];

struct Edge { int to, nx; } e[N << 1];

inline void add(int u, int v) {
    static int cnt = 0;
    e[++cnt].to = v;
    e[cnt].nx = h[u];
    h[u] = cnt;
}

void dfs(int u, int pre) {
    static int cnt = 0;
    st[u] = ++cnt, w[cnt] = a[u]; // dfn 序
    for (int i = h[u]; i; i = e[i].nx) {
        if (e[i].to != pre) {
            dfs(e[i].to, u);
        }
    }
    ed[u] = cnt; // dfn 序
}

inline void add(int x) { ++tot[++cnt[x]]; }
inline void del(int x) { --tot[cnt[x]--]; }

inline void Mo() {
    int l = 1, r = 0;
    sort(q + 1, q + 1 + m);
    for (int i = 1; i <= m; ++i) {
        const int &L = q[i].l, &R = q[i].r;
        while (l > L) add(w[--l]);
        while (r < R) add(w[++r]);
        while (l < L) del(w[l++]);
        while (r > R) del(w[r--]);
        ans[q[i].d] = tot[q[i].k];
    }
}

int main() {
    n = rd(), m = rd();
    B = n / sqrt(m) + 1;
    for (int i = 1; i <= n; ++i) {
        a[i] = rd();
        id[i] = i / B;
    }
    for (int i = 1; i < n; ++i) {
        x = rd(), y = rd();
        add(x, y), add(y, x);
    }
```

```
    dfs(1, 0);
    for (int i = 1; i <= m; ++i) {
        x = rd(), y = rd();
        q[i] = (Q) {st[x], ed[x], y, i};
    }
    Mo();
    for (int i = 1; i <= m; ++i) {
        printf("%d\n", ans[i]);
    }
    return 0;
}
```

7.1.4 带修莫队

```
// 数颜色个数, 修改颜色

#define N 150005
#define M 1000006

char s[2];
int a[N], id[N], cnt[M], ans[N];
int n, m, B, x, y, t1, t2, res, _;

struct Q { // 查询
    int l, r, i, t; // t : 上一次修改的编号
    bool operator < (const Q &b) const {
        return id[l] ^ id[b.l] ? l < b.l : id[r] ^ id[b.r] ? r < b.r : t < b.t;
    }
} q[N];

struct C { int i, to; } c[N]; // 修改

inline void add(int x) { if (++cnt[x] == 1) ++res; }
inline void del(int x) { if (--cnt[x] == 0) --res; }

inline void work(int t, int i) {
    if (q[i].l <= c[t].i && c[t].i <= q[i].r) { // 要修改的点在当前区间内
        del(a[c[t].i]), add(c[t].to);
    }
    swap(a[c[t].i], c[t].to); // 妙呀!
}

inline void Mo() {
    int l = 1, r = 0, t = 0; // t 当前修改编号
    sort(q + 1, q + 1 + t1);
    for (int i = 1; i <= t1; ++i) {
        const int &L = q[i].l, &R = q[i].r;
        while (l > L) add(a[--l]);
        while (r < R) add(a[++r]);
        while (l < L) del(a[l++]);
        while (r > R) del(a[r--]);
        while (t < q[i].t) work(++t, i); // 修改
        while (t > q[i].t) work(t--, i); // 还原
        ans[q[i].i] = res;
    }
}

int main() {
```

```
    n = rd(), m = rd();
    B = pow(n, 2.0 / 3); // 注意块长
    for (int i = 1; i <= n; ++i) {
        a[i] = rd();
        id[i] = i / B;
    }
    while (m--) {
        scanf("%s", s);
        x = rd(), y = rd();
        if (*s == 'Q') {
            ++t1, q[t1] = (Q) {x, y, t1, t2};
        } else {
            c[++t2] = (C) {x, y};
        }
    }
    Mo();
    for (int i = 1; i <= t1; ++i) {
        printf("%d\n", ans[i]);
    }
    return 0;
}
```

7.1.5 回滚莫队

```
// 维护相同的数的最远间隔距离
// 回滚莫队(不删除莫队)

int n, m, x, y, B, up, tot, res;
int a[N], L[N], R[N], st[N], ed[N], st_[N], ed_[N], ans[N];

struct In {
    int i, x;
    bool operator < (const In &b) const { return x < b.x; }
} t[N];

struct Q {
    int l, r, d, i;
    bool operator < (const Q &b) const { return d ^ b.d ? l < b.l : r < b.r; }
} q[N];

inline void add(int i, int st[], int ed[]) { // 修改
    const int &x = a[i];
    if (st[x]) {
        st[x] = min(st[x], i);
        ed[x] = max(ed[x], i);
        res = max(res, ed[x] - st[x]);
    } else {
        st[x] = ed[x] = i;
    }
}

inline void resume(int x) { // 清空
    st[x] = st_[x] = ed[x] = ed_[x] = 0;
}

inline void copy(int x) { // 备份
    st_[x] = st[x], ed_[x] = ed[x];
}
```

```cpp

inline void Mo() {
    sort(q + 1, q + 1 + m);
    int l = 1, r = 0, lst = 0;
    for (int i = 1; i <= m; ++i) {
        const int &ql = q[i].l, &qr = q[i].r;
        if (lst < q[i].d) { // 当前左端点块号小于询问左端点块号
            while (l <= R[q[i].d]) {  // <=
                if (l <= r) resume(a[l]);
                ++l;
            }
            while (r > L[q[i].d]) { // L[q[i].d]
                if (l <= r) resume(a[r]);
                --r;
            }
            lst = q[i].d, res = 0; //
        }
        // 右端点单调，直接同步修改原数组，并备份到临时数组
        while (r < qr) if (l <= ++r) add(r, st, ed), copy(a[r]);
        int tmp = res; // 备份左端点移动前的结果
        while (l > ql) if (--l <= r) add(l, st_, ed_); // 修改临时数组
        ans[q[i].i] = res; // 记录临时数组修改的结果
        const int l_ = R[q[i].d] + 1;
        while (l < l_) { // 回滚，恢复临时数组到原数组
            if (l <= r) copy(a[l]);
            ++l;
        }
        res = tmp; // 恢复到左端点移动前的结果
    }
}

inline void init() {
    sort(t + 1, t + 1 + n);
    for (int i = 1; i <= n; ++i) {
        if (t[i].x ^ t[i - 1].x) ++tot;
        a[t[i].i] = tot;
    }
    B = sqrt(n);
    up = (n - 1) / B + 1;
    for (int i = 1; i <= up; ++i) {
        L[i] = R[i - 1] + 1;
        R[i] = i * B;
    }
    R[up] = n;
}

int main() {
    n = rd();
    for (int i = 1; i <= n; ++i) {
        t[i] = (In) {i, rd()};
    }
    init(), m = rd();
    for (int i = 1; i <= m; ++i) {
        x = rd(), y = rd();
        q[i] = (Q) {x, y, (x - 1) / B + 1, i};
    }
    Mo();
    for (int i = 1; i <= m; ++i) {
        printf("%d\n", ans[i]);
```

```
    }
    return 0;
}
```

7.1.6 bitset 上莫队

```
/*
 * 给你一个序列 a, 长度为 n, 有 m 次操作,
 * 每次询问一个区间是否可以选出两个数它们的差 or 和 or 乘积 or 商(没有余数)为 w
 * 这四个操作分别为操作 1, 2, 3, 4  选出的这两个数可以是同一个位置的数
 */
#include <bits/stdc++.h>
using namespace std;
#define N 100005
const int mx = 1e5;
bitset<N> b1, b2;
int n, m, opt, x, y, w, B, t;
int a[N], id[N], cnt[N], ans[N];
struct Q {
    int opt, l, r, x, i;
    bool operator < (const Q &b) const {
        return id[l] ^ id[b.l] ? l < b.l : id[l] & 1 ? r < b.r : r > b.r;
    }
} q[N];
inline void add(int x) { if (++cnt[x] == 1) b1[x] = b2[mx - x] = 1; }
inline void del(int x) { if (--cnt[x] == 0) b1[x] = b2[mx - x] = 0; }
inline int qsub(int x) { return (b1 & (b1 << x)).any(); }
inline int qadd(int x) { return (b1 & (b2 >> (mx - x))).any(); }
inline int qmul(int x) {
    if (!x) return b1[0];
    for (int i = 1; i * i <= x; ++i) {
        if (x % i == 0 && (b1[i] && b1[x / i])) return 1;
    }
    return 0;
}
inline int qdiv(int x) { // x >= B
    if (!x) return b1[0] && (b1 >> 1).any();
    for (int i = 1; i * x <= mx; ++i) {
        if (b1[i] && b1[i * x]) return 1;
    }
    return 0;
}
inline void Mo() {
    int l = 1, r = 0;
    sort(q + 1, q + 1 + t);
    for (int i = 1; i <= t; ++i) {
        const int &L = q[i].l, &R = q[i].r;
        while (l > L) add(a[--l]);
        while (r < R) add(a[++r]);
        while (l < L) del(a[l++]);
        while (r > R) del(a[r--]);
        switch (q[i].opt) {
            case 1: ans[q[i].i] = qsub(q[i].x); break;
            case 2: ans[q[i].i] = qadd(q[i].x); break;
            case 3: ans[q[i].i] = qmul(q[i].x); break;
            case 4: ans[q[i].i] = qdiv(q[i].x); break;
        }
    }
```

```
}
struct Div { int i, l, r; };
vector<Div> v[N];
int pre[N], res[N];
inline void solve() { // 处理 w < B 的部分的商操作
    for (int i = 1; i < B; ++i) {
        if (v[i].empty()) continue;
        memset(pre, 0, sizeof(pre));
        memset(res, 0, sizeof(res));
        int now = 0;
        for (int j = 1; j <= n; ++j) {
            pre[a[j]] = j;
            if (a[j] * i <= mx) now = max(now, pre[a[j] * i]);
            if (a[j] % i == 0 ) now = max(now, pre[a[j] / i]);
            res[j] = now;
        }
        for (auto &q : v[i]) {
            ans[q.i] = q.l <= res[q.r];
        }
    }
}
inline int rd();
int main() {
    n = rd(), m = rd(), B = sqrt(n) + 1;
    for (int i = 1; i <= n; ++i) id[i] = (i - 1) / B + 1;
    for (int i = 1; i <= n; ++i) a[i] = rd();
    for (int i = 1; i <= m; ++i) {
        opt = rd(), x = rd(), y = rd(), w = rd();
        if (opt == 4 && w && w < B) { // 根号分治
            v[w].push_back((Div) {i, x, y});
        } else {
            q[++t] = (Q) {opt, x, y, w, i};
        }
    }
    Mo(), solve();
    for (int i = 1; i <= m; ++i) {
        puts(ans[i] ? "yuno" : "yumi");
    }
    return 0;
}
```

7.1.7 二次离线莫队 1

```
// 给你一个长为 n 的序列 a, m 次询问, 每次查询一个区间的逆序对数
#include <bits/stdc++.h>
using namespace std;

#define N 100005
#define eb emplace_back

typedef long long ll;

int n, m, x, y, B, up, tot;
ll pre1[N], pre2[N], ans[N];
int a[N], cnt[N], tag[N], id[N], L[N], R[N], num1[N], num2[N];

struct In {
    int i, x;
```

```
16      bool operator < (const In &b) const { return x < b.x; }
17  } in[N];
18
19  struct Q {
20      int l, r, i;
21      bool operator < (const Q &b) const {
22          return id[l] ^ id[b.l] ? l < b.l : r < b.r;
23      }
24  } q[N];
25
26  vector<tuple<int, int, int, int> > v1[N], v2[N];
27
28  inline int rd();
29  namespace BIT {
30      int c[N];
31      inline int lowbit(int x) {
32          return x & -x;
33      }
34      inline void add(int i) {
35          while (i <= tot) {
36              ++c[i];
37              i += lowbit(i);
38          }
39      }
40      inline int ask(int i, int ans = 0) {
41          while (i) {
42              ans += c[i];
43              i -= lowbit(i);
44          }
45          return ans;
46      }
47  }
48  using namespace BIT;
49
50  inline void init1() {
51      sort(in + 1, in + 1 + n);
52      if (!in[1].x) tot = 1;
53      for (int i = 1; i <= n; ++i) { // 离散化
54          if (in[i].x ^ in[i - 1].x) ++tot;
55          a[in[i].i] = tot;
56      }
57      for (int i = 1; i <= n; ++i) {
58          num1[i] = i - 1 - ask(a[i]);
59          pre1[i] = pre1[i - 1] + num1[i];
60          add(a[i]);
61      }
62      memset(c, 0, sizeof(c));
63      for (int i = n; i; --i) {
64          num2[i] = ask(a[i] - 1);
65          pre2[i] = pre2[i + 1] + num2[i];
66          add(a[i]);
67      }
68      B = sqrt(n);
69      for (int i = 1; i <= n; ++i) {
70          id[i] = (i - 1) / B + 1;
71      }
72  }
73
74  inline void Mo1() {
```

```
    int l = 1, r = 0;
    sort(q + 1, q + 1 + m);
    for (int i = 1; i <= m; ++i) {
        const int &ql = q[i].l, &qr = q[i].r;
        if (r > qr) {
            ans[q[i].i] -= pre1[r] - pre1[qr];
            v1[l].eb(qr + 1, r, q[i].i, 1), r = qr; // 离线 m 次端点移动
        }
        if (r < qr) {
            ans[q[i].i] += pre1[qr] - pre1[r];
            v1[l].eb(r + 1, qr, q[i].i, -1), r = qr;
        }
        if (l > ql) {
            ans[q[i].i] += pre2[ql] - pre2[l];
            v2[r].eb(ql, l - 1, q[i].i, -1), l = ql;
        }
        if (l < ql) {
            ans[q[i].i] -= pre2[l] - pre2[ql];
            v2[r].eb(l, ql - 1, q[i].i, 1), l = ql;
        }
    }
}

inline void init2() { // 分块初始化
    B = sqrt(++tot); // 值域分块会查询到 x + 1 位置
    for (int i = 1; i <= tot; ++i) {
        id[i] = (i - 1) / B + 1;
    }
    up = id[tot];
    for (int i = 1; i <= up; ++i) {
        L[i] = R[i - 1] + 1;
        R[i] = i * B;
    }
    R[up] = tot;
}

inline void pushdown(int d) {
    if (tag[d]) {
        for (int i = L[d]; i <= R[d]; ++i) cnt[i] += tag[d];
        tag[d] = 0;
    }
}

inline void ins1(int x) {
    pushdown(id[x]);
    for (int i = L[id[x]]; i <= x; ++i) ++cnt[i];
    for (int i = 1; i < id[x]; ++i) ++tag[i];
}

inline void ins2(int x) {
    pushdown(id[x]);
    for (int i = R[id[x]]; i >= x; --i) ++cnt[i];
    for (int i = up; i > id[x]; --i) ++tag[i];
}

inline void Mo2() { // 分块 O(√n) - O(1), 总 O(n√n) - O(n√n)
    int l, r, d, f;
    for (int i = 1; i <= n; ++i) { // O(n)
        for (auto &x : v1[i]) {
```

```
            tie(l, r, d, f) = x;
            for (int j = l; j <= r; ++j) { // 总移动次数 O(n√n)
                ans[d] += (ll)f * (tag[id[a[j] + 1]] + cnt[a[j] + 1]); // O(1) 查询
            }
        }
        ins1(a[i]); // 分块 O(√n) 修改
    }
    memset(cnt, 0, sizeof(cnt));
    memset(tag, 0, sizeof(tag));
    for (int i = n; i; --i) {
        for (auto &x : v2[i]) {
            tie(l, r, d, f) = x;
            for (int j = l; j <= r; ++j) {
                ans[d] += (ll)f * (tag[id[a[j] - 1]] + cnt[a[j] - 1]);
            }
        }
        ins2(a[i]);
    }
}

int main() {
    n = rd(), m = rd();
    for (int i = 1; i <= n; ++i) in[i] = (In) {i, rd()};
    for (int i = 1; i <= m; ++i) {
        x = rd(), y = rd();
        q[i] = (Q) {x, y, i};
    }
    init1(), Mo1(); // 离线端点移动共 O(n√n) 次
    init2(), Mo2(); // 分块扫描线平衡复杂度
    // ans 为每次操作后的答案差分, 需再做一次前缀和
    for (int i = 1; i <= m; ++i) ans[q[i].i] += ans[q[i - 1].i];
    for (int i = 1; i <= m; ++i) printf("%lld\n", ans[i]);
    return 0;
}
```

7.1.8 二次离线莫队 2

```
// 每次查询区间内 ai ^ aj 有 k 个 1 的二元组个数
#include <bits/stdc++.h>
using namespace std;

#define N 100005
#define eb emplace_back

const int mx = 16384;

typedef long long ll;

ll ans[N];
int n, m, k, x, y, d, B;
int a[N], id[N], cnt[N], pre[N];

struct Q {
    int l, r, i;
    ll ans;
    bool operator < (const Q &b) const {
        return id[l] ^ id[b.l] ? l < b.l : r < b.r;
    }
```

```
} q[N];

vector<int> b;
vector<tuple<int, int, int> > v[N];

inline int rd();
int main() {
    n = rd(), m = rd(), k = rd();
    if (k > 14) { while (m--) puts("0"); exit(0); } // 特判
    for (int i = 1; i <= n; ++i) a[i] = rd();
    for (int i = 1; i <= m; ++i) {
        x = rd(), y = rd();
        q[i] = (Q) {x, y, i};
    }
    for (int i = 0; i < mx; ++i) { // [0, mx) QwQ
        if (__builtin_popcount(i) == k) b.eb(i); // init
    }
    B = sqrt(n);
    for (int i = 1; i <= n; ++i) id[i] = (i - 1) / B + 1;
    for (int i = 1; i <= n; ++i) {
        for (auto &x : b) ++cnt[a[i] ^ x];
        pre[i] = cnt[a[i + 1]]; // i + 1 位置对 [1, i] 的贡献
    }
    int l = 1, r = 0;
    sort(q + 1, q + 1 + m);
    for (int i = 1; i <= m; ++i) {
        const int &L = q[i].l, &R = q[i].r;
        if (r > R) {
            v[l - 1].eb(R + 1, r, i); // 二次离线
            while (r > R) q[i].ans -= pre[r - 1], --r; // 计算不用离线部分的贡献
        }
        if (r < R) {
            v[l - 1].eb(r + 1, R, -i);
            while (r < R) q[i].ans += pre[r], ++r;
        }
        if (l > L) {
            v[r].eb(L, l - 1, i);
            while (l > L) q[i].ans -= pre[l - 2], --l;
        }
        if (l < L) {
            v[r].eb(l, L - 1, -i);
            while (l < L) q[i].ans += pre[l - 1], ++l;
        }
    }
    memset(cnt, 0, sizeof(cnt));
    for (int i = 1; i <= m; ++i) {
        for (auto &x : b) ++cnt[a[i] ^ x];
        for (auto &x : v[i]) {
            tie(l, r, d) = x;
            if (d > 0) {
                for (int tmp, j = l; j <= r; ++j) {
                    tmp = cnt[a[j]];
                    if (j <= i && !k) --tmp;
                    q[d].ans += tmp;
                }
            } else {
                for (int tmp, j = l; j <= r; ++j) {
                    tmp = cnt[a[j]];
                    if (j <= i && !k) --tmp;
```

```
                    q[-d].ans -= tmp;
                }
            }
        }
    }
    for (int i = 1; i <= m; ++i) q[i].ans += q[i - 1].ans; // 前缀和
    for (int i = 1; i <= m; ++i) ans[q[i].i] = q[i].ans;
    for (int i = 1; i <= m; ++i) printf("%lld\n", ans[i]);
    return 0;
}
```

7.2 CDQ 分治

7.2.1 动态逆序对

```
// cdq  O(n logn logn)
#include <bits/stdc++.h>
using namespace std;

#define N 100005

struct P { int pos, val, add, id; } a[N << 1];

int n, m;
int c[N], pos[N];
long long ans[N];

inline void add(int i, int k) {
    for (; i <= n; i += i & -i) c[i] += k;
}
inline int ask(int i) {
    int ans = 0;
    for (; i; i -= i & -i) ans += c[i];
    return ans;
}

inline bool cmp(const P &a, const P &b) { return a.pos ^ b.pos ? a.pos < b.pos : a.val > b.val; }

void cdq(int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    cdq(l, mid), cdq(mid + 1, r);
    sort(a + l, a + mid + 1, cmp);
    sort(a + mid + 1, a + r + 1, cmp);
    // tim[j] < tim[i] && pos[j] < pos[i] && val[j] > val[i]
    int j = l, i = mid + 1;
    while (i <= r) {
        while (j <= mid && a[j].pos <= a[i].pos) {
            add(a[j].val, a[j].add), ++j;
        }
        ans[a[i].id] += a[i].add * (ask(n) - ask(a[i].val)), ++i;
    }
    while (--j >= l) add(a[j].val, -a[j].add);
    // tim[j] < tim[i] && pos[j] > pos[i] && val[j] < val[i]
    j = mid, i = r;
    while (i > mid) {
        while (j >= l && a[j].pos >= a[i].pos) {
            add(a[j].val, a[j].add), --j;
```

```
        }
        ans[a[i].id] += a[i].add * ask(a[i].val - 1), --i;
    }
    while (++j <= mid) add(a[j].val, -a[j].add);
}

inline int rd();
int main() {
    n = rd(), m = rd();
    for (int x, i = 1; i <= n; ++i) {
        x = rd(), a[i] = {pos[x] = i, x, 1, 0};
    }
    for (int x, i = 1; i < m; ++i) {
        x = rd(), a[i + n] = {pos[x], x, -1, i};
    }
    // 1d 自然形成时间序, 无须 sort
    cdq(1, n + m - 1);
    for (int i = 0; i < m; ++i) {
        printf("%lld\n", ans[i] += ans[i - 1]);
    }
    return 0;
}
```

7.2.2 三维 CDQ 优化 DP

```
/* 给定一个长度为 n 的序列 a. 同时这个序列还可能发生变化, 每一种变化 (x[i], y[i]) 对应着 a[x[i] 可能变成 y[i].
 * 不会同时发生两种变化. 需要找出一个最长的子序列, 使得这个子序列在任意一种变化下都是不降的.
 * 只需要求出这个子序列的长度即可. */
#include <bits/stdc++.h>
using namespace std;

#define N 100005
#define repn(i, x, y) for (int i = x; i <= y; ++i)

struct P { int w, mn, mx, id, ans; } a[N], tmp[N];

int c[N];
int n, m, x, y, mxv, ans;

inline void add(int i, int x) {
    for (; i <= mxv; i += i & -i) c[i] = max(c[i], x);
}
inline void del(int i) {
    for (; i <= mxv; i += i & -i) c[i] = 0;
}
inline int ask(int i) {
    int ans = 0;
    for (; i; i -= i & -i) ans = max(ans, c[i]);
    return ans;
}

// j <= i && a[j].mx <= a[i].w && a[j].w <= a[i].mn
inline bool cmp1(const P &a, const P &b) { return a.mx < b.mx; }
inline bool cmp2(const P &a, const P &b) { return a.w < b.w; }
void cdq(int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
```

```
    cdq(l, mid);
    sort(a + l, a + mid + 1, cmp1); // sort_cmp1
    sort(a + mid + 1, a + r + 1, cmp2); // sort_cmp2
    int j = l, i = mid + 1;
    while (i <= r) {
        while (j <= mid && a[j].mx <= a[i].w) {
            add(a[j].w, a[j].ans), ++j;
        }
        a[i].ans = max(a[i].ans, ask(a[i].mn) + 1), ++i;
    }
    while (--j >= l) del(a[j].w);
    repn(i, l, r) tmp[a[i].id] = a[i];
    repn(i, l, r) a[i] = tmp[i];
    cdq(mid + 1, r);
}

inline int rd();
int main() {
    n = rd(), m = rd();
    repn(i, 1, n) x = rd(), a[i] = {x, x, x, i, 1};
    repn(i, 1, m) {
        x = rd(), y = rd();
        a[x].mn = min(a[x].mn, y);
        a[x].mx = max(a[x].mx, y);
    }
    repn(i, 1, n) mxv = max(mxv, a[i].mx);
    cdq(1, n); // 1d 时间序
    repn(i, 1, n) ans = max(ans, a[i].ans);
    printf("%d\n", ans);
    return 0;
}
```

7.2.3 四维 CDQ 优化 DP

```
// dp1: 最大价值和, dp2: 满足条件的方案个数
#include <bits/stdc++.h>
using namespace std;

#define N 80004
#define repn(i, x, y) for (int i = x; i <= y; ++i)

typedef long long ll;

const int P = 998244353;

struct Node {
    ll mx; int cnt; // mx: longest path
    Node operator + (const ll &val) const {
        return {mx + val, cnt};
    }
};
// x[j] <= x[i] && y[j] <= y[i] && z[j] <= z[i] && w[j] <= w[i]
struct Data {
    int x, y, z, w, id;
    ll val; Node dp; bool ok; // ok: 1d tag
} a[N], tmp[N];

Node c[N];
```

```
25 int w[N], pos[N];
26 int n, m, len, tot, cnt;
27
28 inline Node merge(const Node &a, const Node &b) { // dp 转移
29     if (a.mx ^ b.mx) return a.mx < b.mx ? b : a;
30     return {a.mx, (a.cnt + b.cnt) % P};
31 }
32
33 inline void add(int i, const Node &k) {
34     for (; i <= len; i += i & -i) c[i] = merge(c[i], k);
35 }
36 inline void del(int i) {
37     for (; i <= len; i += i & -i) c[i] = {0, 0};
38 }
39 inline Node ask(int i) {
40     Node ans = {0, 0};
41     for (; i; i -= i & -i) ans = merge(ans, c[i]);
42     return ans;
43 }
44
45 inline bool cmp1(const Data &a, const Data &b) { // x -> y -> z -> w
46     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 : a.w
       < b.w;
47 }
48 inline bool cmp2(const Data &a, const Data &b) { // y -> z -> w -> x
49     return a.y ^ b.y ? a.y < b.y : a.z ^ b.z ? a.z < b.z : a.w ^ b.w ? a.w < b.w : a.x
       < b.x;
50 }
51 inline bool cmp3(const Data &a, const Data &b) { // z -> w -> x -> y
52     return a.z ^ b.z ? a.z < b.z : a.w ^ b.w ? a.w < b.w : a.x ^ b.w ? a.x < b.x : a.y
       < b.y;
53 }
54
55 void cdq2(int l, int r) {
56     if (l == r) return;
57     int mid = (l + r) >> 1;
58     cdq2(l, mid);
59     sort(a + l, a + mid + 1, cmp3);
60     sort(a + mid + 1, a + r + 1, cmp3);
61     int j = l, i = mid + 1;
62     while (i <= r) {
63         while (j <= mid && a[j].z <= a[i].z) {
64             if (a[j].ok) add(a[j].w, a[j].dp); ++j;
65         }
66         if (!a[i].ok) a[i].dp = merge(a[i].dp, ask(a[i].w) + a[i].val); ++i;
67     }
68     while (--j >= l) if (a[j].ok) del(a[j].w); // undo
69     repn(i, l, r) tmp[pos[a[i].id]] = a[i];
70     repn(i, l, r) a[i] = tmp[i]; // resume to sort_cmp2
71     cdq2(mid + 1, r);
72 }
73
74 void cdq1(int l, int r) {
75     if (l == r) return;
76     int mid = (l + r) >> 1;
77     cdq1(l, mid);
78     repn(i, l, mid) a[i].ok = 1; // 1d tag
79     repn(i, mid + 1, r) a[i].ok = 0; // 1d tag
80     sort(a + l, a + r + 1, cmp2);
```

```
    repn(i, l, r) pos[a[i].id] = i; // backup for sort_cmp2
    cdq2(l, r);
    repn(i, l, r) tmp[a[i].id] = a[i];
    repn(i, l, r) a[i] = tmp[i]; // resume to sort_cmp1
    cdq1(mid + 1, r);
}

inline int rd();
int main() {
    n = rd(), m = rd();
    repn(i, 1, n) {
        a[i].x = rd(), a[i].y = rd(), a[i].z = rd();
        a[i].w = w[i] = rd(), a[i].val = rd();
    }
    sort(w + 1, w + 1 + n);
    len = unique(w + 1, w + 1 + n) - w - 1;
    repn(i, 1, n) a[i].w = lower_bound(w + 1, w + 1 + len, a[i].w) - w;
    sort(a + 1, a + 1 + n, cmp1); // 1d sort
    repn(i, 1, n) { // unique
        if (a[i].x ^ a[i + 1].x || a[i].y ^ a[i + 1].y || a[i].z ^ a[i + 1].z || a[i].w
 ^ a[i + 1].w) {
            a[++cnt] = a[i];
        } else {
            a[i + 1].val += a[i].val;
        }
    }
    repn(i, 1, cnt) a[i].dp = {a[i].val, 1}, a[i].id = i; // backup for sort_cmp1
    cdq1(1, cnt);
    Node ans = {0, 0};
    repn(i, 1, cnt) ans = merge(ans, a[i].dp);
    printf("%lld\n%d\n", ans.mx, ans.cnt);
    return 0;
}
```

7.3 树分治

7.3.1 点分治模版

```
// 给定一棵有 n 个点的树，询问树上距离为 k 的点对是否存在  time: O(nlogn)
// https://www.luogu.com.cn/problem/P3806
const int INF = 2e9;
const int mxd = 1e7;
struct Node { int to, nx, dis; } e[N << 1];
bitset<mxd + 1> tf;
int n, m, x, y, z, rt, cnt, tot;
int h[N], mx[N], vis[N], siz[N], ans[N], q[110];
void getr(int u, int pre) { // 获取重心，重新计算 mx[], siz[]
    mx[u] = 0, siz[u] = 1;
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (v != pre && !vis[v]) { // !vis[v]
            getr(v, u);
            siz[u] += siz[v];
            mx[u] = max(mx[u], siz[v]);
        }
    }
    mx[u] = max(mx[u], tot - siz[u]);
    if (mx[u] < mx[rt]) rt = u; // 更新重心，错误求法 mx[u] <= tot / 2
}
```

```cpp
int tmp[N], d[N], t, ct;
// dis[i] : i 到 calc() 中 v 的距离
// tmp[] : 线性存储所有 dis[i] (i 是 v 子树中的节点)
void getd(int u, int pre, int dis) {
    if (dis > mxd) return; // 剪枝!  k <= mxd, 故只存储 dis <= mxd 的路径!
    tmp[++t] = d[++ct] = dis;
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (v != pre && !vis[v]) { // !vis[v]
            getd(v, u, dis + e[i].dis);
        }
    }
}
// tmp[] 存储以当前处理的儿子 v 为根的树的所有 dis[] 信息
// d[] 存储 rt 为根的树的所有 dis[] 信息, 用于处理完该树后清空 tf[]
// ct, t 作为 d, tmp 的数组长度
void calc(int u) {
    ct = 0;
    for (int i = h[u]; i; i = e[i].nx) { // 处理 u 的每个子树
        int v = e[i].to;
        if (vis[v]) continue;
        t = 0, getd(v, u, e[i].dis); // 计算的是子树 v 中的节点到 u 的距离
        for (int k = 1; k <= t; ++k)
            for (int t = 1; t <= m; ++t)
                 // tmp[] 存了所有 u 子树节点到 u 的距离
                 // 如果 tmp[k] > q[t], 路径肯定不存在
                // if (tmp[k] <= q[t])
                    ans[t] |= tf[q[t] - tmp[k]];
                    // 若 tmp[] 中一个数 + tf[] 中一个数 = k, 就说明路径存在
        for (int k = 1; k <= t; ++k) tf[tmp[k]] = 1; // 更新桶, 表示存在长度为 tmp[k] 的路径
    }
    // 找完一颗 以 rt 为根的树后, 清空 tf[]
    // 不可以用 memset 暴力清空 tf, 会超时
    for (int i = 1; i <= ct; ++i) tf[d[i]] = 0;
}
void solve(int u) {
    vis[u] = tf[0] = 1; // vis[] : 避免重复计算, 注意要设置 tf[0] 为 1
    calc(u);
    for (int all = tot, i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (vis[v]) continue;
        tot = siz[v] < siz[u] ? siz[v] : all - siz[u]; // 子树的节点数量
        getr(v, rt = 0), solve(rt); // 处理子树
    }
}
int main() {
    n = rd(), m = rd();
    for (int i = 1; i < n; ++i) {
        x = rd(), y = rd(), z = rd();
        add(x, y, z), add(y, x, z);
    }
    for (int i = 1; i <= m; ++i) q[i] = rd();
    tot = *mx = n; // 注意设置 *mx = n
    getr(1, rt = 0); // 获取重心
    solve(rt);
    for (int i = 1; i <= m; ++i) puts(ans[i] ? "AYE" : "NAY");
    return 0;
}
```

7.3.2 点分治例题

```
// 给定一棵 n 个节点的树，每条边有 边权 ，求出树上两点距离 <= k 的点对数量。
// 双指针维护，运用容斥减去重复贡献.
// 也可使用 BIT 维护，但有 k 大小限制，不然 BIT 数组开不下
// ( 须在 getd 时碰到 dis[v] > k 时剪枝，不然 BIT 空间是 O(n * k) 的 )
// https://www.luogu.com.cn/problem/P4178
struct Node { int to, nx, dis; } e[N << 1];
int n, k, x, y, z, rt, cnt, tot, ans;
int h[N], vis[N], dis[N], d[N], mx[N], siz[N];
void add(int u, int v, int dis);
void getr(int u, int pre) {
    mx[u] = 0, siz[u] = 1;
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (v != pre && !vis[v]) {
            getr(v, u);
            siz[u] += siz[v];
            mx[u] = max(mx[u], siz[v]);
        }
    }
    mx[u] = max(mx[u], tot - siz[u]);
    if (mx[u] < mx[rt]) rt = u;
}
void getd(int u, int pre, int dis) {
    if (dis > k) return; // 剪枝! 优化空间与时间
    d[++ct] = dis;
    for (int i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (v != pre && !vis[v]) {
            getd(v, u, dis + e[i].dis);
        }
    }
}
inline int geta(int l, int r) { // 双指针维护答案
    int res = 0;
    sort(d + l, d + r + 1);
    while (l <= r) d[l] + d[r] <= k ? res += r - l++ : --r;
    return res;
}
inline void calc(int u) {
    for (int p, i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (vis[v]) continue;
        p = t, getd(v, u, e[i].dis);
        ans -= geta(p + 1, t); // 减去重复贡献 ( 同一子树内两个结点的贡献 )
    }
    d[++t] = 0, ans += geta(1, t), t = 0;
}
void solve(int u) {
    vis[u] = 1, calc(u);
    for (int all = tot, i = h[u]; i; i = e[i].nx) {
        int v = e[i].to;
        if (vis[v]) continue;
        tot = siz[v] < siz[u] ? siz[v] : all - siz[u];
        getr(v, rt = 0), solve(rt);
    }
}
inline int rd();
```

```
int main() {
    n = rd();
    for (int i = 1; i < n; ++i) {
        x = rd(), y = rd(), z = rd();
        add(x, y, z), add(y, x, z);
    }
    k = rd(), *mx = tot = n;
    getr(1, 0), solve(rt);
    printf("%d\n", ans);
    return 0;
}
```

7.4 整体二分

7.4.1 带修区间第 k 小

```
// O(nlogn * O(BIT)) = O(n logn logn)
#include <bits/stdc++.h>
using namespace std;

#define N 100005
#define repn(i, x, y) for (int i = x; i <= y; ++i)

struct P { int x, y, k; } t[N];
struct Q { int x, y, k, id; } q[N * 3], q1[N * 3], q2[N * 3];

char s[2];
int n, m, x, y, len, cnt, tot;
int a[N], b[N << 1], c[N], ans[N];

inline void add(int i, int k) { for (; i <= n; i += i & -i) c[i] += k; }
inline int ask(int i) { int ans = 0; for (; i; i -= i & -i) ans += c[i]; return ans; }

void solve(int l, int r, int ql, int qr) { // l, r: 值域,  ql, qr: 询问
    if (ql > qr) return;
    if (l == r) { repn(i, ql, qr) if (q[i].id) ans[q[i].id] = l; return; }
    int mid = (l + r) >> 1, t1 = 0, t2 = 0;
    repn(i, ql, qr) {
        if (!q[i].id) {
            if (q[i].y <= mid) { // 修改部分按权值化分为 q1: [l, mid], q2: [mid + 1, r]
                q1[++t1] = q[i], add(q[i].x, q[i].k);
            } else { // BIT 维护权值 <= mid 的部分的区间数量前缀和
                q2[++t2] = q[i];
            }
        } else {
            int x = ask(q[i].y) - ask(q[i].x - 1); // q[i] 询问区间内权值 <= mid 的数的数量
            if (q[i].k <= x) {
                q1[++t1] = q[i];
            } else {
                q[i].k -= x, q2[++t2] = q[i];
            }
        }
    }
    repn(i, 1, t1) if (!q1[i].id) add(q1[i].x, -q1[i].k); // 还原
    repn(i, 1, t1) q[i + ql - 1] = q1[i];
    repn(i, 1, t2) q[i + ql - 1 + t1] = q2[i];
    solve(l, mid, ql, ql + t1 - 1), solve(mid + 1, r, ql + t1, qr);
}

```

```
inline int rd();
int main() {
    len = n = rd(), m = rd();
    repn(i, 1, n) a[i] = b[i] = rd();
    repn(i, 1, m) {
        scanf("%s", s), x = rd(), y = rd();
        if (*s == 'Q') {
            t[i] = {x, y, rd()};
        } else {
            t[i] = {x, b[++len] = y};
        }
    }
    sort(b + 1, b + 1 + len);
    len = unique(b + 1, b + 1 + len) - b - 1;
    repn(i, 1, n) a[i] = lower_bound(b + 1, b + 1 + len, a[i]) - b;
    repn(i, 1, n) q[++cnt] = {i, a[i], 1}; // 原始数值按插入操作处理
    repn(i, 1, m) {
        if (t[i].k) { // 询问
            q[++cnt] = {t[i].x, t[i].y, t[i].k, ++tot};
        } else { // 将修改拆成先删除后插入
            int y = lower_bound(b + 1, b + 1 + len, t[i].y) - b;
            q[++cnt] = {t[i].x, a[t[i].x], -1};
            q[++cnt] = {t[i].x, a[t[i].x] = y, 1};
        }
    }
    solve(0, len + 1, 1, cnt);
    repn(i, 1, tot) printf("%d\n", b[ans[i]]);
    return 0;
}
```

7.4.2 矩形区域第 k 小

```
#include <bits/stdc++.h>
using namespace std;

#define N 250004
#define repn(i, x, y) for (int i = x; i <= y; ++i)

struct P { int x, y, k, add, id; } q[N << 1], q1[N << 1], q2[N << 1];

int n, m, len, cnt;
int b[N], c[N], sum[N], ans[N];

inline void add(int i, int k) { for (; i <= n; i += i & -i) c[i] += k; }
inline int ask(int i) { int ans = 0; for (; i; i -= i & -i) ans += c[i]; return ans; }
inline bool cmp(const P &a, const P &b) { return a.x ^ b.x ? a.x < b.x : a.id < b.id; }

// O( n n logn logn )  n 为矩形边长
void solve(int l, int r, int ql, int qr) { // l, r 为当前二分答案边界, ql, qr 为处在当前答案范围内的询问
    if (ql > qr) return;
    if (l == r) { repn(i, ql, qr) if (q[i].id) ans[q[i].id] = l; return; }
    int mid = (l + r) >> 1, t1 = 0, t2 = 0; // mid: 当前二分答案
    repn(i, ql, qr) {
        if (!q[i].id) {
            if (q[i].k <= mid) add(q[i].y, 1); // add
        } else {
            sum[q[i].id] += ask(q[i].y) * q[i].add;
```

```
        }
    }
    // 统计完答案再划分询问，保证询问按照 x 轴单调
    repn(i, ql, qr) {
        if (!q[i].id) {
            if (q[i].k <= mid) q1[++t1] = q[i], add(q[i].y, -1); // resume
            else q2[++t2] = q[i];
        } else {
            if (q[i].k <= sum[q[i].id]) q1[++t1] = q[i];
            else q[i].k -= sum[q[i].id], q2[++t2] = q[i];
        }
    }
    repn(i, ql, qr) if (q[i].id) sum[q[i].id] = 0; // resume
    repn(i, 1, t1) q[i + ql - 1] = q1[i]; // 划分到左区间的
    repn(i, 1, t2) q[i + ql - 1 + t1] = q2[i]; // 划分到右区间的
    solve(l, mid, ql, ql + t1 - 1), solve(mid + 1, r, ql + t1, qr);
}

inline int rd();
int main() {
    n = rd(), m = rd(), cnt = n * n;
    repn(i, 1, n) repn(j, 1, n) {
        ++len, q[len] = {i, j, b[len] = rd()};
    }
    sort(b + 1, b + 1 + len);
    len = unique(b + 1, b + 1 + len) - b - 1;
    repn(i, 1, cnt) q[i].k = lower_bound(b + 1, b + 1 + len, q[i].k) - b;
    repn(i, 1, m) {
        int x1 = rd(), y1 = rd(), x2 = rd(), y2 = rd(), k = rd();
        q[++cnt] = {x2, y2, k, 1, i}; // 拆解询问，转化为 2 维数点问题
        if (x1 > 1) q[++cnt] = {x1 - 1, y2, k, -1, i};
        if (y1 > 1) q[++cnt] = {x2, y1 - 1, k, -1, i};
        if (x1 > 1 && y1 > 1) q[++cnt] = {x1 - 1, y1 - 1, k, 1, i};
    }
    sort(q + 1, q + 1 + cnt, cmp); // 按 x 轴排序
    solve(0, len + 1, 1, cnt);
    repn(i, 1, m) printf("%d\n", b[ans[i]]);
    return 0;
}
```

8 Others

8.1 模拟退火

8.1.1 费马点

```cpp
#include <bits/stdc++.h>
#define rate 0.997
const int maxn = 105;
using namespace std;

typedef long long ll;
struct node {
    int x, y;
}a[maxn];
int n;
double ansx,ansy, anse;
inline double energy(double x, double y) {
    double ans = 0;
    for(int i = 1; i <= n; ++i) {
        double dx = x - a[i].x;
        double dy = y - a[i].y;
        ans += sqrt(dx * dx + dy * dy);
    }
    return ans;
}
inline void SA(){
    double t = 3000;
    while(t > 1e-15) {
        double tx = ansx + (rand() * 2 - RAND_MAX) * t;
        double ty = ansy + (rand() * 2 - RAND_MAX) * t;
        double te = energy(tx, ty);
        double de = te - anse;
        if(de < 0) {
            ansx = tx , ansy = ty , anse = te;
        }else if(exp(-de / t) * RAND_MAX > rand()) {
            ansx = tx , ansy = ty;
        }
        t *= rate;
    }
}
inline void solve(){
    SA(); SA(); SA(); SA();
}
int main() {
    int t; scanf("%d", &t);
    while (t--) {
        scanf("%d",&n);
        for(int i = 1; i <= n; ++i) {
            scanf("%d%d", &a[i].x, &a[i].y);
            ansx += a[i].x , ansy += a[i].y;
        }
        ansx /= n , ansy /= n;
        anse = energy(ansx, ansy);
        solve();
        printf("%lld\n", (ll)(round(anse)));
        if (t) puts("");
    }
    return 0;
```

```
}
```

8.2 矩阵

8.2.1 矩阵快速幂

```
struct Matrix {
    ll a[N][N]; // 矩阵大小
    Matrix() { memset(a, 0, sizeof(a)); }
    inline void build() { // 构造单位矩阵
        for (int i = 1; i <= n; ++i)
            a[i][i] = 1;
    }
};

Matrix operator * (const Matrix &x, const Matrix & y) {
    Matrix z;
    // k 放中间循环, 充分利用 cpu cache
    for (int i = 1; i <= n; ++i)
        // if (x.a[i][k]) // 对稀疏矩阵的优化
            for (int k = 1; k <= n; ++k)
                for (int j = 1; j <= n; ++j)
                    z.a[i][j] = (z.a[i][j] + x.a[i][k] * y.a[k][j] % P) % P;
    return z;
}

Matrix Pow(Matrix a, ll b) {
    Matrix ans;
    ans.build();
    b %= P;
    while (b) {
        if (b & 1) ans = ans * a;
        a = a * a; // 定义了 * , 没定义 *=
        b >>= 1;
    }
    return ans;
}
```

8.2.2 高斯消元

```
void gauss()
{
    int now = 1, to;
    double t;
    for (int i = 1; i <= n; i++, now++)
    {
        /*for (to = now; !a[to][i] && to <= n; to++);
        //做除法时减小误差, 可不写
        if (to != now)
            for (int j = 1; j <= n + 1; j++)
                swap(a[to][j], a[now][j]);*/
        t = a[now][i];
        for (int j = 1; j <= n + 1; j++) a[now][j] /= t;
        for (int j = 1; j <= n; j++)
            if (j != now)
            {
                t = a[j][i];
                for (int k = 1; k <= n + 1; k++) a[j][k] -= t * a[now][k];
```

```
            }
    }
}
```

8.2.3 gauss

```
#include <bits/stdc++.h>
const int maxn = 105;
using namespace std;
double a[maxn][maxn];
int main(){
    int n; scanf("%d",&n);
    for(int i=1;i<=n;++i){
        for(int j=1;j<=n+1;++j){
            scanf("%lf",&a[i][j]);
        }
    }
    for(int j=1;j<=n;++j) { //列
        int mx = j;
        for(int i=j+1;i<=n;++i){
            if(fabs(a[i][j]) > fabs(a[mx][j])) mx = i;
        }
        for(int i=1;i<=n+1;++i){
            swap(a[mx][i],a[j][i]);
        }
        if(!a[j][j]){
            puts("No Solution\n");
            return 0;
        }
        for(int i=1;i<=n;++i){ //行
            if(i==j) continue;
            double tmp = a[i][j] / a[j][j];
            for(int k=1;k<=n+1;++k){
                a[i][k] -= tmp * a[j][k];
            }
        }
    }
    for(int i=1;i<=n;++i){
        printf("%.2lf\n",a[i][n+1]/a[i][i]);
    }
}
```

8.3 技巧

8.3.1 解决爆栈问题

```
// 解决爆栈问题
#pragma comment(linker, "/STACK:1024000000,1024000000")
```

8.3.2 卡常

```
int main() {
    ios_base::sync_with_stdio(false), cin.tie(0);
}
namespace IO { // 快读
    const int in = 1 << 22, out = 1 << 22;
    static char buf[in], *p1 = buf, *p2 = buf, duf[out], *q1 = duf, ch[20];
```

```
#define gcf() (p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, in, stdin), p1 == p2) ? EOF
    : *p1++)
    template <typename T> inline int rd(T &x) {
        x = 0; char c; int f = 0;
        while (!isdigit(c = gcf())) f |= c == '-';
        while ( isdigit(c)) x = (x << 1) + (x << 3) + (c ^ 48), c = gcf();
        return f ? -x : x;
    }
    template <typename T, typename... Args>
    inline void rd(T& t, Args&... args) { rd(t), rd(args...); }
    template <typename T> inline void wr(T x) {
        int i = 0;
        (x < 0) && (x = -x, *q1++ = '-');
        (x) || (*q1++ = '0');
        while (x) ch[i++] = x % 10 + 48, x /= 10;
        while (i) *q1++ = ch[--i];
    }
    inline void flush() { fwrite(duf, q1 - duf, 1, stdout); }
}
using IO::rd;
using IO::wr;
// 输出字符 *IO::q1++='要输出的字符';
// 程序结尾需要加上 IO::flush();

short seed = 2333;
inline short rnd() { // 自制随机数生成器
    return seed = (seed * 7 + 13) ^ (seed / 13 - 7);
}
```

8.3.3 Strok-Sscanf

```
// 空格作为分隔输入,读取一行的整数
fgets(buf, BUFSIZE, stdin);
int v;
char *p = strtok(buf, " ");
while (p) {
    sscanf(p, "%d", &v);
    p = strtok(NULL," ");
}
```

8.3.4 对拍

```
#include <cstdio>
#include <cstdlib>
int main() {
    int tmp = 0;
    for (int i = 1; i <= 1000; ++i) {
        system("./generate"); // 数据生成器
        system("./tt"); // 对拍程序
        system("./a"); // 我的程序
        if (i / 100 > tmp) {
            printf("-- %d --\n", i);
            tmp = i / 100;
        }
        if (system("diff test.out std.out")) { // 对比结果
            printf("wrong in --> %d \n", i);
            break;
        }
```

```
    }
    return 0;
}
```

8.3.5 树生成器

```
// 生成一颗树
#include <bits/stdc++.h>
using namespace std;

const int N = 3e5;

int main() {
    freopen("test.in", "w", stdout);
    srand(time(NULL));
    const int n = rand() % N + 1; // 节点数
    const int m = rand() % N + 1; // 询问数
    printf("%d %d\n", n, m);
    for (int i = 1; i < n; ++i) { // 边
        printf("%d %d\n", rand() % i + 1, i + 1);
    }
    for (int i = 1; i <= m; ++i) { // 询问(根据题目来码)
        printf("%d %d\n", rand() % n + 1, rand() % n + 1);
    }
    return 0;
}
```

8.3.6 double 4 舍 5 入

```
double round1(double x) {
    const double sd = 100; // 2
    return int(x * sd + (x < 0 ? -0.5 : 0.5)) / sd;
}
double round2(double x) { // 防爆整数范围版本
    const double sd = 100; // 2
    ll y = x; x -= y;
    return y + ll(x * sd + (x < 0 ? -0.5 : 0.5)) / sd;
}
```

8.4 大数

8.4.1 High-precision

```
// 加法 乘法 小于号 输出
struct bint {
    int l;
    short int w[100];
    bint(int x = 0) {
        l = x == 0, memset(w, 0, sizeof(w));
        while (x) w[l++] = x % 10, x /= 10;
    }
    bool operator<(const bint& x) const {
        if (l != x.l) return l < x.l;
        int i = l - 1;
        while (i >= 0 && w[i] == x.w[i]) i--;
        return (i >= 0 && w[i] < x.w[i]);
    }
```

```
    bint operator+(const bint& x) const {
        bint ans;
        ans.l = l > x.l ? l : x.l;
        for (int i = 0; i < ans.l; i++)
        {
            ans.w[i] += w[i] + x.w[i];
            ans.w[i + 1] += ans.w[i] / 10;
            ans.w[i] = ans.w[i] % 10;
        }
        if (ans.w[ans.l] != 0) ans.l++;
        return ans;
    }
    bint operator*(const bint& x) const {
        bint res;
        int up, tmp;
        for (int i = 0; i < l; i++)
        {
            up = 0;
            for (int j = 0; j < x.l; j++)
            {
                tmp = w[i] * x.w[j] + res.w[i + j] + up;
                res.w[i + j] = tmp % 10;
                up = tmp / 10;
            }
            if (up != 0) res.w[i + x.l] = up;
        }
        res.l = l + x.l;
        while (res.w[res.l - 1] == 0 && res.l > 1) res.l--;
        return res;
    }
    void print() {
        for (int i = l - 1; ~i; i--) printf("%d", w[i]);
        puts("");
    }
};
```

8.4.2 H÷L

```
char s[N];
int a[N], c[N], b;
{
    scanf("%s%d", s, &b); // 高精除低精
    int la = strlen(s);
    for (int i = 1; i <= la; ++i) a[i] = s[i - 1] ^ 48;
    int t = 0;
    for (int i = 1; i <= la; ++i) {
        t = (t << 1) + (t << 3) + a[i];
        c[i] = t / b, t %= b;
    }
    int len = 1;
    while (!c[len]) ++len;
    for (int i = len; i <= la; ++i) printf("%d", c[i]); puts("");
}
```

8.4.3 H÷H

```
char a[N], b[N], c[N];
void sub() {
```

```
    int len = 0;
    while (a[len] == '0') ++len;
    for (int i = lb - 1; i >= len; --i) {
        a[i] -= b[i] ^ 48;
        if (a[i] < '0') --a[i - 1], a[i] += 10;
    }
}
int main() {
    scanf("%s%s", a, b);
    la = strlen(a), lb = strlen(b);
    if (la < lb || (la == lb && strcmp(a, b) < 0)) {
        puts("0"), puts(a), exit(0);
    }
    int len = 1;
    while (1) {
        while (strncmp(a, b, lb) >= 0) ++c[len], sub();
        if (la == lb) break;
        ++len;
        for (int i = lb; i; --i) b[i] = b[i - 1];
        *b = '0', ++lb;
    }
    int i = 1;
    // 输出商
    while (!c[i] && i < len) ++i;
    while (i <= len) printf("%d", c[i++]); puts("");
    // 输出余数
    i = 0;
    while (a[i] == '0' && i < la - 1) ++i;
    puts(a + i);
    return 0;
}
```

8.5 Misc

8.5.1 Standard Template Library

```
template <class InputIterator, class OutputIterator>
  OutputIterator copy (InputIterator first, InputIterator last, OutputIterator result);

template <class InputIterator1, class InputIterator2,
          class OutputIterator, class Compare>
  OutputIterator merge (InputIterator1 first1, InputIterator1 last1,
                        InputIterator2 first2, InputIterator2 last2,
                        OutputIterator result, Compare comp);

template <class InputIterator, class Function>
   Function for_each (InputIterator first, InputIterator last, Function fn);

template <class InputIterator, class OutputIterator, class UnaryOperation>
  OutputIterator transform (InputIterator first1, InputIterator last1,
                            OutputIterator result, UnaryOperation op);

template< class ForwardIterator, class T >
void iota( ForwardIterator first, ForwardIterator last, T value );
```

8.5.2 Policy-Based Data Structures

红黑树

声明/头文件

```
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
typedef tree<pt, null_type, less<pt>, rb_tree_tag, tree_order_statistics_node_update>
    rbtree;
```

使用方法

```
pt                                  // 关键字类型
null_type                           // 无映射(低版本g++为null_mapped_type)
less<int>                           // 从小到大排序
rb_tree_tag                         // 红黑树 (splay_tree_tag)
tree_order_statistics_node_update   // 结点更新
T.insert(val);                      // 插入
T.erase(iterator);                  // 删除
T.order_of_key();                   // 查找有多少数比它小
T.find_by_order(k);                 // 有k个数比它小的数是多少
a.join(b);                          // b并入a 前提是两棵树的key的取值范围不相交
a.split(v, b);                      // key小于等于v的元素属于a, 其余的属于b
T.lower_bound(x);                   // >=x的min的迭代器
T.upper_bound((x);                  // >x的min的迭代器
```

8.5.3 Subset Enumeration

枚举真子集

```
for (int s = (S - 1) & S; s; s = (s - 1) & S)
```

枚举大小为 k 的子集

```
void subset(int k, int n) {
    int t = (1 << k) - 1;
    while (t < (1 << n)) {
        // do something
        int x = t & -t, y = t + x;
        t = ((t & ~y) / x >> 1) | y;
    }
}
```

8.5.4 Date Magic

```
string dayOfWeek[] = {"Mo", "Tu", "We", "Th", "Fr", "Sa", "Su"};

// converts Gregorian date to integer (Julian day number)
int DateToInt(int m, int d, int y)
{
    return 1461 * (y + 4800 + (m - 14) / 12) / 4
           + 367 * (m - 2 - (m - 14) / 12 * 12) / 12
           - 3 * ((y + 4900 + (m - 14) / 12) / 100) / 4
           + d - 32075;
}

// converts integer (Julian day number) to Gregorian date: month/day/year
void IntToDate(int jd, int& m, int& d, int& y)
{
    int x, n, i, j;
    x = jd + 68569;
```

```
    n = 4 * x / 146097;
    x -= (146097 * n + 3) / 4;
    i = (4000 * (x + 1)) / 1461001;
    x -= 1461 * i / 4 - 31;
    j = 80 * x / 2447;
    d = x - 2447 * j / 80;
    x = j / 11;
    m = j + 2 - 12 * x;
    y = 100 * (n - 49) + i + x;
}

// converts integer (Julian day number) to day of week
string IntToDay(int jd) { return dayOfWeek[jd % 7]; }
```

8.5.5 进制转换

```
// 输入x, y, old, 然后调用trans(), 此时 a 即为转换后的字符串
char old[N], a[N]; // 转换前、后的字符串
int t[N], A[N]; // old, a 的数组
int x, y; // 转换前、后的进制
int idx(char c) { ... } // 将字符转为数字
char idc(int x) { ... } // 将数字转为字符
void trans() {
    int len = strlen(old);
    for (int i = 0; i < len; ++i) {
        t[i] = idx(old[len - i - 1]);
    }
    int k = 0;
    while (len) {
        for (int i = len - 1; i; --i) {
            t[i - 1] += t[i] % y * x;
            t[i] /= y;
        }
        A[k++] = *t % y; // 此时最后一位模 y 就是进制转换后的倒数第 k + 1 位
        *t /= y;
        while (len && !t[len - 1]) --len;
    }
    a[k] = 0;
    for (int i = 0; i < k; ++i) {
        a[i] = idc(A[k - i - 1]);
    }
}
int main() {
    scanf("%d%d%s", &x, &y, old);
    trans(), puts(a);
    return 0;
}
```

8.5.6 区间加等差数列求和

```
// 静态维护
/* 维护一个数组, 先进行 m 次操作, 然后查询每个位置的值
 * 每个操作给定四个参数 l, r, a, k 表示从 l 到 r 依次加上一个首项为 a, 公差为 k 的等差数列.
 * 方法: 维护 d2 数组, 表示原数组的二阶差分
*/
void add(int l, int r, int a, int k) {
    d2[l] += a;
    d2[l + 1] += k - a;
```

```
    d2[r + 1] -= (r - l + 1) * k + a;
    d2[r + 2] -= (l - r) * k - a;
}
void pre_sum() { // 做前缀和
    for (int i = 1; i <= n; ++i) d2[i] += d2[i - 1];
}
int main() {
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= m; ++i) {
        scanf("%d %d %d %d", &l, &r, &a, &k); // [l, r], an = a + k(n - 1)
        add(l, r, a, k);
    }
    pre_sum(), pre_sum();
    // d2[] => d1[] => a[] , 做两次前缀和还原数组
    for (int i = 1; i <= n; ++i) {
        printf("%d\n", d2[i]);
    }
    return 0;
}

// 动态维护

// 1. 差分方法
/* 操作 l, r, k, d , 记 sum[i] 为差分数组 a[i] - a[i - 1]
 * 对于  l     : sum[l] = sum[l] + k
 * 对于 (l, r] : sum[i] = sum[i] + d
 * 对于  r + 1 : sum[r + 1] = sum[r + 1] - (k + (r - l) * d)
 * 每次查询输出 a[x] + (sum[1] + ... + sum[x])
 */
struct Node { int l, r, sum, tag; } t[N << 2];
int a[N], n, m, opt, l, r, k, d, x;
inline void pushup(int p) { t[p].sum = tls(p).sum + trs(p).sum; }
inline void add(int p, int k) {
    t[p].tag += k;
    t[p].sum += (R - L + 1) * k;
}
inline void pushdown(int p) {
    if (t[p].tag) {
        add(ls(p), t[p].tag);
        add(rs(p), t[p].tag);
        t[p].tag = 0;
    }
}
void build(int l, int r, int p) {
    L = l, R = r;
    if (l == r) return;
    int mid = (l + r) >> 1;
    build(l, mid, ls(p));
    build(mid + 1, r, rs(p));
}
void update(int x, int y, int k, int p = 1) {
    if (x <= L && y >= R) return add(p, k);
    pushdown(p);
    if (x <= tls(p).r) update(x, y, k, ls(p));
    if (y >  tls(p).r) update(x, y, k, rs(p));
    pushup(p);
}
int query(int x, int y, int p = 1) {
    if (x >  R || y <  L) return 0;
```

```
    if (x <= L && y >= R) return t[p].sum;
    pushdown(p);
    return query(x, y, ls(p)) + query(x, y, rs(p));
}
int main() {
    n = rd(), m = rd();
    for (int i = 1; i <= n; ++i) a[i] = rd();
    build(1, n, 1);
    while (m--) {
        opt = rd();
        if (opt & 1) {
            l = rd(), r = rd(), k = rd(), d = rd();
            update(l, l, k);
            if (l < r) update(l + 1, r, d);
            if (r ^ n) update(r + 1, r + 1, -(k + (r - l) * d));
        } else {
            x = rd();
            printf("%d\n", a[x] + query(1, x));
        }
    }
    return 0;
}

// 2. 非差分方法, 线段树(两个 tag)
struct Node {
    int l, r;
    ll k, d, sum; // k, d : 首项, 公差的 tag
} t[N << 2];
int n, m, opt, l, r, k, d;
void build(int l, int r, int p) {
    L = l, R = r;
    if (l == r) { t[p].sum = rd(); return; }
    int mid = (l + r) >> 1;
    build(l, mid, ls(p));
    build(mid + 1, r, rs(p));
}
inline void pushup(int p) { t[p].sum = tls(p).sum + trs(p).sum; }
inline void add(int p, ll k, ll d) {
    int n = (R - L + 1);
    t[p].k += k, t[p].d += d;
    t[p].sum += n * k + n * (n - 1) / 2 * d;
}
inline void pushdown(int p) {
    if (t[p].k || t[p].d) {
        add(ls(p), t[p].k, t[p].d);
        t[p].k += (tls(p).r - L + 1) * t[p].d;
        add(rs(p), t[p].k, t[p].d);
        t[p].k = t[p].d = 0;
    }
}
void update(int x, int y, ll k, ll d, int p = 1) {
    if (x <= L && y >= R) {
        k += (L - x) * d;
        add(p, k, d);
        return;
    }
    pushdown(p);
    if (x <= tls(p).r) update(x, y, k, d, ls(p));
    if (y >  tls(p).r) update(x, y, k, d, rs(p));
```

```
    pushup(p);
}
ll query(int x, int p = 1) {
    if (L == R) return t[p].sum;
    pushdown(p);
    if (x <= tls(p).r) return query(x, ls(p));
    return query(x, rs(p));
}
int main() {
    n = rd(), m = rd(), build(1, n, 1);
    while (m--) {
        if ((opt = rd()) & 1) update(l = rd(), r = rd(), k = rd(), d = rd());
        else printf("%lld\n", query(rd()));
    }
    return 0;
}
```

8.6 配置

8.6.1 vim

```
sy on
set et nu is sc ai hls cin udf
set bs=2 sw=4 so=999 hi=10000 sts=4 mouse=a cb=unnamed
set noeb vb t_vb=
colo gruvbox
"colo evening

let mapleader = ","
nnoremap <C-e> :enew<CR>:e
nnoremap <silent> <C-n> :bn<CR>
nnoremap <silent> <C-p> :bpre<CR>
nnoremap <leader>w :w<cr><esc>
nnoremap <leader>v mmggVG
nnoremap <leader>y mmggVG"+y`m
nnoremap <leader>r :call Run()<cr>
nnoremap <leader>tr :call Run_T()<cr>
nnoremap <silent><leader>l :!clear<CR><CR>
"vnoremap <leader><leader> mm^o^<C-v>I// <ESC>`m
"vnoremap <leader>d  mm^o^<C-v>lld<ESC>`m

set noesckeys

"map<c-y> mmggVG"+y`m
"map<f5> :call Run()<cr>
"vnoremap \\ mm^o^<C-v>I//<ESC>`m
"vnoremap \d mm^o^<C-v>ld<ESC>`m

noremap <C-j> <C-W>j
noremap <C-k> <C-W>k
noremap <C-h> <C-W>h
noremap <C-l> <C-W>l

func! Run()
    exec "w"
    exec "!g++ -w % -o %< && ./%<"
endfunc

func! Run_T()
```

```
    exec "w"
    exec "!g++ % -o %< && time ./%<"
endfunc

inoremap {<cr> {<cr>}<esc>O
"inoremap ( ()<esc>i
"inoremap { {}<esc>i
"inoremap [ []<esc>i
"inoremap ) <c-r>=ClosePair(')')<cr>
"inoremap } <c-r>=ClosePair('}')<cr>
"inoremap ] <c-r>=ClosePair(']')<cr>
"
"func ClosePair(char)
"    if getline('.')[col('.') - 1] == a:char
"        return "\<right>"
"    else
"        return a:char
"    endif
"endfunc
```

8.6.2 精简配置

```
sy on
"set noeb vb t_vb=
set et nu is sc ai hls cin udf
set bs=2 sw=4 so=999 hi=10000 sts=4 mouse=a cb=unnamed
"colo gruvbox
colo evening

let mapleader = ","
nnoremap <leader>y mmggVG"+y`m
nnoremap <leader>r :call Run()<cr>
vnoremap <leader>r :call Run()<cr>

"map<c-y> mmggVG"+y`m
"map<f5> :call Run()<cr>
"vnoremap \\ mm^o^<C-v>I//<ESC>`m
"vnoremap \d mm^o^<C-v>ld<ESC>`m

func! Run()
    exec "w"
    exec "!g++ -w % -o %< && ./%<"
endfunc

inoremap {<cr> {<cr>}<esc>O
```