Standard Code Library

Kyooma

April 21, 2025

Contents

一切的开始

快读

```cpp
#define gc()(is==it?it=(is=in)+fread(in,1,Q,stdin),(is==it?EOF:*is++):*is++)
const int Q=(1<<24)+1;
char in[Q],*is=in,*it=in,c;
void read(long long &n){
    for(n=0;(c=gc())<'0'||c>'9';);
    for(;c<='9'&&c>='0';c=gc())n=n*10+c-48;
}
```

代码模板

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

using namespace std;
#define dbg(x...) \
    do { \
        cout << #x << " -> "; \
        err(x); \
    } while (0)

void err() {
    cout << endl;
}

template<class T, class... Ts>
void err(T arg, Ts &... args) {
    cout << arg << ' ';
    err(args...);
}

typedef long long ll;
typedef pair<int, int> pii;
const int N = 1e5 + 10, Log = 20, inf = 0x3f3f3f3f;

void solve() {

}

int main() {
    int T = 1;
    ios::sync_with_stdio(false);
    cin >> T;
    while (T--) solve();
    return 0;
}
```

数据结构

st 表

$st[i][j]$ 表示区间 $[i, i+2^j-1]$ 的 gcd

```cpp
int st[N][Log + 5], logx[N];

void init(int n) {
    logx[0] = -1;
    for (int i = 1; i <= n; i++)logx[i] = logx[i >> 1] + 1;
    for (int i = 1; i <= n; i++)st[i][0] = i;
    for (int j = 1; (1 << j) <= n; j++) {
        for (int i = 1; i + (1 << j) - 1 <= n; i++) {
            st[i][j] = __gcd(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
        }
    }
}

int query(int l, int r){
```

```
    int k = logx[r - l + 1];
    return __gcd(st[l][k], st[r - (1 << k) + 1][k]);
}
```

树状数组

```
template <typename T>
struct Fenwick {
    const int n;
    vector<T> a;
    Fenwick(int n) : n(n), a(n + 1) {}
    void add(int x, T v) {
        while(x <= n){
            a[x] += v;
            x += x & -x;
        }
    }
    T sum(int x) {
        T ans = 0;
        for (int i = x; i; i -= i & -i) {
            ans += a[i];
        }
        return ans;
    }
    T rangeSum(int l, int r) {
        return sum(r) - sum(l - 1);
    }
};
```

主席树

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

using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
const int N = 200010;
int root[N], tot = 0, num[N], len, a[N];
struct Info {
    int sum, l, r;
} info[N << 5];

int getid(int x) {
    return lower_bound(num + 1, num + len + 1, x) - num;
}

void build(int &x, int l, int r) {//创建空树
    x = ++tot;
    info[x].sum = 0;
    if (l == r) return;
    int mid = (l + r) / 2;
    build(info[x].l, l, mid);
    build(info[x].r, mid + 1, r);
}

void update(int pre, int &now, int l, int r, int q) {//更新
    now = ++tot;
    info[now] = info[pre];
    info[now].sum++;
    if (l == r) return;
    int mid = (l + r) / 2;
    if (mid >= q) update(info[pre].l, info[now].l, l, mid, q);
    else update(info[pre].r, info[now].r, mid + 1, r, q);
}

int query(int pre, int now, int l, int r, int k) {//求第 k 小
    if (l == r) return l;
    int delta = info[info[now].l].sum - info[info[pre].l].sum;
    int mid = (l + r) / 2;
    if (delta >= k) return query(info[pre].l, info[now].l, l, mid, k);
```

```
    else return query(info[pre].r, info[now].r, mid + 1, r, k - delta);
}

int query_sum(int pre, int now, int l, int r, int k) {// 求小于等于 k 的个数
    if (l == r) return info[now].sum - info[pre].sum;
    int mid = (l + r) >> 1;
    if (k <= mid) return query_sum(info[pre].l, info[now].l, l, mid, k);
    else return (info[info[now].l].sum - info[info[pre].l].sum) + query_sum(info[pre].r, info[now].r, mid + 1, r, k);
}
/*
先进行离散化
sort(num + 1, num + 1 + n);
len = unique(num + 1, num + 1 + n) - num - 1;
建空树
build(root[0], 1, len);
更新
update(root[i - 1], root[i], 1, len, getid(a[i]));
查询 [l, r]
query_sum(root[l - 1], root[r], 1, len, k)
query(root[l - 1], root[r], 1, len, k)
*/
```

树链剖分

```
vector<int> e[N];

int n;
int sz[N], f[N], son[N];
int top[N], dfn[N], rk[N], tot, ru[N];
int a[N];

void dfs(int u, int fa){
    sz[u] = 1;
    f[u] = fa;
    son[u] = -1;
    for(int i : e[u]){
        if(i == fa) continue;
        dfs(i, u);
        sz[u] += sz[i];
        if(son[u] == -1 || sz[i] > sz[son[u]]) son[u] = i;
    }
}

void dfs1(int u, int t){
    top[u] = t;
    dfn[u] = ++tot;
    rk[tot] = u;
    if(son[u] == -1){
        ru[u] = dfn[u];
        return;
    }
    dfs1(son[u], t);
    for(int i : e[u]){
        if(i == f[u] || i == son[u]) continue;
        dfs1(i, i);
    }
    ru[u] = tot;
}

template<typename T>
struct SegmentTree{
    T sum[N << 2], lz[N << 2];
    void apply(int k, int l, int r, T x){
        sum[k] += (r - l + 1) * x;
        lz[k] += x;
    }
    void pd(int k, int l, int r){// push down
        int mid = (l + r) >> 1;
        apply(k << 1, l, mid, lz[k]);
        apply(k << 1 | 1, mid + 1, r, lz[k]);
        lz[k] = 0;
```

```
    }
    void pu(int k){// push up
        sum[k] = sum[k << 1] + sum[k << 1 | 1];
    }
    void build(int k, int l, int r){
        if(l == r){
            sum[k] = a[rk[l]];
            lz[k] = 0;
            return;
        }
        int mid = (l + r) >> 1;
        build(k << 1, l, mid);
        build(k << 1 | 1, mid + 1, r);
        pu(k);
    }
    void mdf(int k, int l, int r, int ql, int qr, T x){// modify [ql, qr] add x
        if(l > qr || r < ql) return;
        if(l >= ql && r <= qr){
            sum[k] += (r - l + 1) * x;
            lz[k] += x;
            return;
        }
        pd(k, l, r);
        int mid = (l + r) >> 1;
        mdf(k << 1, l, mid, ql, qr, x);
        mdf(k << 1 | 1, mid + 1, r, ql, qr, x);
        pu(k);
    }
    T query(int k, int l, int r, int ql, int qr){
        if(l > qr || r < ql) return 0;
        if(l >= ql && r <= qr){
            return sum[k];
        }
        pd(k, l, r);
        int mid = (l + r) >> 1;
        return query(k << 1, l, mid, ql, qr) + query(k << 1 | 1, mid + 1, r, ql, qr);
    }
};

SegmentTree<ll> seg;

int qrysum(int u, int v){
    int fu = top[u], fv = top[v], ret = 0;
    while(fu != fv){
        if(dfn[fu] > dfn[fv]){
            ret += seg.query(1, 1, n, dfn[fu], dfn[u]);
            u = f[fu];
        }else{
            ret += seg.query(1, 1, n, dfn[fv], dfn[v]);
            v = f[fv];
        }
        fu = top[u];
        fv = top[v];
    }
    if(dfn[u] > dfn[v]) swap(u, v);
    ret += seg.query(1, 1, n, dfn[u], dfn[v]);
    return ret;
}

void solve() {
    int m, rt;
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> a[i];
    for(int i = 0, u, v; i < n - 1; i++){
        cin >> u >> v;
        e[u].push_back(v);
        e[v].push_back(u);
    }
    dfs(1, 0);
    dfs1(1, 1);
    seg.build(1, 1, n);
```

```
}
```

平衡树 Treap

普通平衡树

```cpp
mt19937 mt(chrono::steady_clock::now().time_since_epoch().count());

int rng(int l, int r) {
    uniform_int_distribution<int> uni(l, r);
    return uni(mt);
}

struct Node{
    Node *lt, *rt; // 左右子结点
    int val, prio; // 值, 优先级
    int cnt, sz; // 重复次数, 子树大小

    Node(int _val) : val(_val), cnt(1), sz(1) {
        lt = rt = nullptr;
        prio = rng(1, 1e9);
    }

    void upd(){
        sz = cnt;
        if(lt != nullptr) sz += lt->sz;
        if(rt != nullptr) sz += rt->sz;
    }
};

struct Treap{
    int siz(Node *p){
        if(p == nullptr) return 0;
        return p->sz;
    }

    Node *root;

    pair<Node *, Node *> split(Node *cur, int key) { // 根据 val 分裂成 小于等于 key 和 大于 key 的两个 treap
        if (cur == nullptr) return {nullptr, nullptr};
        if (cur->val <= key) {// 当前属于第一个 treap
            auto temp = split(cur->rt, key);
            cur->rt = temp.first;
            cur->upd();
            return {cur, temp.second};
        } else {// 当前属于第二个 treap
            auto temp = split(cur->lt, key);
            cur->lt = temp.second;
            cur->upd();
            return {temp.first, cur};
        }
    }

    tuple<Node *, Node *, Node *> split_by_rk(Node *cur, int rk) { // 根据 rk 分裂成 小于 rk 和 等于 rk 和 大于 rk 的三个 treap, 其中第二个只有一个结点
        if (cur == nullptr) return {nullptr, nullptr, nullptr};
        int ls_siz = siz(cur->lt); // 左子树大小
        if (rk <= ls_siz) {// 当前属于第三个 treap
            Node *l, *mid, *r;
            tie(l, mid, r) = split_by_rk(cur->lt, rk);
            cur->lt = r;
            cur->upd();
            return {l, mid, cur};
        } else if (rk <= ls_siz + cur->cnt) {// 当前属于第二个 treap
            Node *lt = cur->lt;
            Node *rt = cur->rt;
            cur->lt = cur->rt = nullptr;
            return {lt, cur, rt};
        } else {// 当前属于第一个 treap
            Node *l, *mid, *r;
            tie(l, mid, r) = split_by_rk(cur->rt, rk - ls_siz - cur->cnt);
```

```
            cur->rt = l;
            cur->upd();
            return {cur, mid, r};
        }
    }

    Node *merge(Node *u, Node *v) {// 按照 prio 小根堆合并
        if (u == nullptr && v == nullptr) return nullptr;
        if (u != nullptr && v == nullptr) return u;
        if (v != nullptr && u == nullptr) return v;
        if (u->prio < v->prio) {
            u->rt = merge(u->rt, v);
            u->upd();
            return u;
        } else {
            v->lt = merge(u, v->lt);
            v->upd();
            return v;
        }
    }

    void insert(int val) {// 插入
        auto temp = split(root, val);
        auto l_tr = split(temp.first, val - 1);
        Node *new_node;
        if (l_tr.second == nullptr) {
            new_node = new Node(val);
        } else {
            l_tr.second->cnt++;
            l_tr.second->upd();
        }
        Node *l_tr_combined = merge(l_tr.first, l_tr.second == nullptr ? new_node : l_tr.second);
        root = merge(l_tr_combined, temp.second);
    }

    void del(int val) {// 删除
        auto temp = split(root, val);
        auto l_tr = split(temp.first, val - 1);
        if (l_tr.second == nullptr){
            root = merge(l_tr.first, temp.second);
            return;
        }
        if (l_tr.second->cnt > 1) {
            l_tr.second->cnt--;
            l_tr.second->upd();
            l_tr.first = merge(l_tr.first, l_tr.second);
        } else {
            if (temp.first == l_tr.second) {
                temp.first = nullptr;
            }
            delete l_tr.second;
            l_tr.second = nullptr;
        }
        root = merge(l_tr.first, temp.second);
    }

    int qrank_by_val(Node *cur, int val) { // 查询 val 的 rk
        auto temp = split(cur, val - 1);
        int ret = siz(temp.first) + 1;
        root = merge(temp.first, temp.second);
        return ret;
    }

    int qval_by_rank(Node *cur, int rk) { // 查询 rk 的 val 第 rk 大的值
        Node *l, *mid, *r;
        tie(l, mid, r) = split_by_rk(cur, rk);
        int ret = (mid == nullptr ? -114514 : mid->val);
        root = merge(merge(l, mid), r);
        return ret;
    }

```

```
    int qprev(int val) { // 查询第一个比 val 小的值
        auto temp = split(root, val - 1);
        int ret = qval_by_rank(temp.first, temp.first->sz);
        root = merge(temp.first, temp.second);
        return ret;
    }

    int qnex(int val) { // 查询第一个比 val 大的值
        auto temp = split(root, val);
        int ret = qval_by_rank(temp.second, 1);
        root = merge(temp.first, temp.second);
        return ret;
    }
};
```

区间翻转

```
mt19937 mt(chrono::steady_clock::now().time_since_epoch().count());

int rng(int l, int r) {
    uniform_int_distribution<int> uni(l, r);
    return uni(mt);
}

struct Node{
    Node *lt, *rt; // 左右子结点
    int val, prio; // 值, 优先级
    int cnt, sz; // 重复次数, 子树大小
    bool rev;// 是否翻转

    Node(int _val) : val(_val), cnt(1), sz(1) {
        lt = rt = nullptr;
        rev = false;
        prio = rng(1, 1e9);
    }

    void pu(){
        sz = cnt;
        if(lt != nullptr) sz += lt->sz;
        if(rt != nullptr) sz += rt->sz;
    }

    void pd(){
        if(rev){
            swap(lt, rt);
            if(lt != nullptr) lt->rev ^= 1;
            if(rt != nullptr) rt->rev ^= 1;
            rev = false;
        }
    }
};

struct Treap{
    Node* root;
    int siz(Node *p){
        if(p == nullptr) return 0;
        return p->sz;
    }

    pair<Node *, Node *> split(Node *cur, int sz){
        if(cur == nullptr) return {nullptr, nullptr};
        cur->pd();
        int lc = siz(cur->lt);
        if(sz <= lc){
            auto temp = split(cur->lt, sz);
            cur->lt = temp.second;
            cur->pu();
            return {temp.first, cur};
        }else{
            auto temp = split(cur->rt, sz - lc - cur->cnt);
            cur->rt = temp.first;
```

```
            cur->pu();
            return {cur,  temp.second};
        }
    }

    Node* merge(Node* u, Node* v) { // u 小 v 大
        if (u == nullptr && v == nullptr) return nullptr;
        if (u != nullptr && v == nullptr) return u;
        if (u == nullptr && v != nullptr) return v;
        u->pd(), v->pd();
        if (u->prio < v->prio) { // u 为根
            u->rt = merge(u->rt, v);
            u->pu();
            return u;
        } else {
            v->lt = merge(u, v->lt);
            v->pu();
            return v;
        }
    }

    void insert(int val){
        root = merge(root, new Node(val));
    }

    void seg_rev(int l, int r) {
        auto res = split(root, l - 1); // [1, l - 1] [l, n]
        auto ans = split(res.second, r - l + 1); // [l, r] [r + 1, n]
        ans.first->rev = true;
        root = merge(res.first, merge(ans.first, ans.second));
    }

    void print(Node* cur) {
        if (cur == nullptr) return;
        cur->pd();
        print(cur->lt);
        printf("%d ", cur->val);
        print(cur->rt);
    }
};
```

zkw 线段树

```
//懒标记可下放的 zkw 线段树 (P3372 模板)
struct SegmentTree{
    ll sum[N << 2], cnt[N << 2];
    ll lz[N << 2];
    int p, dep;

    void clearTag(int u){
        lz[u] = 0;
    }

    void pu(int u){// push up
        sum[u] = (sum[u << 1] + sum[u << 1 | 1]);
        cnt[u] = (cnt[u << 1] + cnt[u << 1 | 1]);
    }

    void apply(int u, ll x){
        sum[u] += x * cnt[u];
        lz[u] += x;
    }

    void pd(int u){// push down
        apply(u << 1, lz[u]);
        apply(u << 1 | 1, lz[u]);
        clearTag(u);
    }

    void build(int n){
        for(p = 1, dep = 0; p < n + 2; p <<= 1, dep++);
```

```
        for(int i = 0; i < p; i++){
            if(i >= 1 && i <= n){
                sum[i + p] = a[i];
                cnt[i + p] = 1;
            }else{
                sum[i + p] = 0;
                cnt[i + p] = 0;
            }
            clearTag(i + p);
        }
        for(int i = p - 1; i; i--){
            clearTag(i);
            pu(i);
        }
    }

    void handle(int u, ll x){
        //处理 u 节点的修改
        sum[u] += cnt[u] * x;
        lz[u] += x;
    }

    void upd(int lx, int rx, ll x){
        int l = lx + p - 1, r = rx + p + 1;
        for(int i = dep; i; i--){
            pd(l >> i);
            pd(r >> i);
        }
        while(l ^ r ^ 1){
            if(~ l & 1) handle(l ^ 1, x);
            if(r & 1) handle(r ^ 1, x);
            l >>= 1;
            r >>= 1;
            pu(l);pu(r);
        }
        for(l >>= 1; l; l >>= 1) pu(l);
    }

    ll qry(int lx, int rx){
        int l = lx + p - 1, r = rx + p + 1;
        for(int i = dep; i; i--){
            pd(l >> i);
            pd(r >> i);
        }
        ll ans = 0;
        while(l ^ r ^ 1){
            if(~ l & 1) ans += sum[l ^ 1];
            if(r & 1) ans += sum[r ^ 1];
            l >>= 1;
            r >>= 1;
        }
        return ans;
    }
}seg;
```

可持久化 Trie

01 字典树处理 $\max_{l\le i\le r}(x \oplus a_i)$

```
struct Trie{
    int tr[N][2], rt[M], tot;
    int sum[N][2];
    void init(int n) {
        for (int i = 1; i <= n; i++) rt[i] = 0;
        for (int i = 1; i <= tot; i++) {
            tr[i][0] = tr[i][1] = 0;
            sum[i][0] = sum[i][1] = 0;
        }
        tot = 0;
    }

```

```
    void insert(int p, int x){ // 第 p 个 插入 x
        int rt0 = rt[p - 1], rt1 = rt[p] = ++tot;

        for(int i = Log; i >= 0; i--){
            for(int j = 0; j < 2; j++){
                sum[rt1][j] = sum[rt0][j];
                tr[rt1][j] = tr[rt0][j];
            }
            int c = x >> i & 1;
            sum[rt1][c]++;
            tr[rt1][c] = ++tot;
            rt1 = tr[rt1][c];
            rt0 = tr[rt0][c];
        }
    }

    int qry(int x, int l, int r){
        int rt0 = rt[l - 1], rt1 = rt[r];
        int ans = 0;
        for(int i = Log; i >= 0; i--){
            int c = x >> i & 1;

            if(sum[rt1][c ^ 1] - sum[rt0][c ^ 1] > 0){
                ans |= 1 << i;
                rt0 = tr[rt0][c ^ 1];
                rt1 = tr[rt1][c ^ 1];
            }else{
                rt0 = tr[rt0][c];
                rt1 = tr[rt1][c];
            }
        }
        return ans;
    }
}trie;
```

数学

组合数

```
ll f[N], inv[N];

ll qpow(ll a, ll b) {
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b /= 2;
    }
    return res;
}

ll C(ll n, ll m) {
    return f[n] * inv[m] % mod * inv[n - m] % mod;
}

void init(int M) {
    f[0] = 1;
    for (int i = 1; i <= M; i++) f[i] = f[i - 1] * i % mod;
    inv[M] = qpow(f[M], mod - 2);
    for (int i = M - 1; i >= 0; i--) inv[i] = inv[i + 1] * (i + 1) % mod;
}
```

Exgcd

求解 $xa + yb = c$

有解需满足 $gcd(a, b)|c$

设解出的一组特解为 x_0, y_0 则通解为 $x = x_0 + tb, y = y_0 - ta$

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

ll upper(ll m, ll n) {//向上取整
    if (m <= 0) return m / n;
    return (m - 1) / n + 1;
}

ll lower(ll m, ll n) {//向下取整
    if (m >= 0) return m / n;
    return (m + 1) / n - 1;
}
```

Lucas 定理

适用于模数为小质数

$$C_n^m \ mod \ p = C_{n \ mod \ p}^{m \ mod \ p} \times C_{\lfloor \frac{n}{p} \rfloor}^{\lfloor \frac{m}{p} \rfloor} \ mod \ p$$

```cpp
ll C(ll n, ll r, ll p) {
    if (r > n || r < 0) return 0;
    return f[n] * inv[r] % p * inv[n - r] % p;
}

ll Lucas(ll n, ll m, ll p) {
    if (m == 0) return 1;
    return (C(n % p, m % p, p) * Lucas(n / p, m / p, p)) % p;
}
```

欧拉筛

```cpp
const int N = 1e4 + 10, M = 10000;
vector<int> p;
int vis[N];

void init() {
    for (int i = 2; i <= M; i++) {
        if (!vis[i]) {
            p.push_back(i);
        }
        for (int j = 0; j < p.size() && p[j] * i <= M; j++) {
            vis[p[j] * i] = 1;
            if (i % p[j] == 0) {
                break;
            }
        }
    }
}
```

求欧拉函数: $phi(n) = n\prod(1 - \frac{1}{p_i})$

```cpp
const int N = 1e4 + 10, M = 10000;
vector<int> p;
int phi[N], vis[N];

void rua() {//欧拉筛 以及 求欧拉函数
    for (int i = 2; i <= M; i++) {
        if (!vis[i]) {
```

```
            p.push_back(i);
            phi[i] = i - 1;
        }
        for (int j = 0; j < p.size() && p[j] * i <= M; j++) {
            vis[p[j] * i] = 1;
            if (i % p[j] == 0) {
                phi[i * p[j]] = phi[i] * p[j];
                break;
            } else {
                phi[i * p[j]] = phi[i] * phi[p[j]];
            }
        }
    }
}
```

线性基

线性基是一个数的集合，并且每个序列都拥有至少一个线性基，取线性基中若干个数异或起来可以得到原序列中的任何一个数。原序列里面的任意一个数都可以由线性基里面的一些数异或得到线性基里面的任意一些数异或起来都不能得到 0 线性基里面的数的个数唯一，并且在保持性质一的前提下，数的个数是最少的

```
ll d[Log + 5];

void add(ll x){// 线性基插入
    for(int i = Log; i >= 0; i--){
        if((x >> i) & 1){
            if(d[i]) x ^= d[i];
            else{
                d[i] = x; // 插入成功
                break;
            }
        }
    }
}
```

欧拉降幂

$$a^b \pmod m \equiv a^{b \bmod \phi(m)+\phi(m)} \pmod m [b \ge \phi(m)]$$

以下代码以计算 $a_l^{a_{l+1}^{a_{l+2}^{\cdots^{a_r}}}}$ 为例

```
unordered_map<ll, ll> mp;
ll a[N];
ll MOD(ll x, ll mod) {return x < mod ? x : x % mod + mod;}
ll qpow(ll a, ll b, ll mod) {
    ll res = 1;
    while (b) {
        if (b & 1) res = MOD(res * a, mod);
        b /= 2;
        a = MOD(a * a, mod);
    }
    return res;
}
ll phi(ll x) {
    if (mp[x]) return mp[x];
    ll res = x;
    for (ll i = 2; i * i <= x; i++) {
        if (x % i == 0) {
            res -= res / i;
            while (x % i == 0) x /= i;
        }
    }
    if (x > 1) {
        res -= res / x;
    }
    return mp[x] = res;
}
ll solve(int l, int r, ll p) {
    if (p == 1) return MOD(a[l], p);
```

```cpp
    if (l == r) return MOD(a[l], p);

    return qpow(a[l], solve(l + 1, r, phi(p)), p);
}
```

矩阵快速幂

```cpp
const int MOD = 1e9 + 7;

struct mat {
    int n;
    vector<vector<int>> a;

    mat(int n): n(n), a(n, vector<int>(n)){}

    mat operator*(const mat& b) const {
        mat res(n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    (res.a[i][j] += 1ll * a[i][k] * b.a[k][j] % MOD) %= MOD;
                }
            }
        }
        return res;
    }

    void print(){
        for(int i = 0; i < n; i++){
            for(int j = 0; j < n; j++){
                cout << a[i][j] << ' ';
            }
            cout << '\n';
        }
        cout << '\n';
    }
};

mat qpow(mat a, ll b) {
    mat res(a.n);
    for (int i = 0; i < a.n; i++) {
        res.a[i][i] = 1;
    }
    while (b) {
        if (b & 1) res = res * a;
        a = a * a, b >>= 1;
    }
    return res;
}
```

中国剩余定理

$x = num_i (mod\ r_i)$

```cpp
ll CRT(int n) {//适用于 ri 两两互质
    ll N = 1, res = 0;
    for (int i = 1; i <= n; i++) N *= r[i];
    for (int i = 1; i <= n; i++) {
        ll m = N / r[i], x, y;
        exgcd(m, r[i], x, y);
        res = (res + num[i] * m % N * x % N) % N;
    }
    return (res + N) % N;
}
```

通解解法

$x = a_1 (mod\ m_1)$

$x = a_2 (mod\ m_2)$

$x = k_1 \times m_1 + a_1 = k_2 \times m_2 + a_2$

$k_1 \times m_1 - k_2 \times m_2 = a_2 - a_1$

运用 exgcd 可求得一组解 (k1,k2) 可将上述两方程化为

$x = k_1 \times m_1 + a_1 (mod\ lcm(m_1, m_2))$

若有多个方程依次两两合并即可

整除分块

$\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor$

```
ans = 0;
for(int l = 1, r; l <= n; l = r + 1) {
    r = n / (n / l);
    ans += n / l * (r - l + 1);
}
```

拉格朗日插值

设要求的 n 次多项式为 $f(k)$, 已知 $f(x_i)\ \ (1 \le i \le n+1)$

$f(k) = \sum_{i=1}^{n+1} f(x_i) \prod_{j \neq i} \frac{k - x_j}{x_i - x_j}$

设要求的 n 次多项式为 $f(k)$, 已知 $f(i)\ \ (1 \le i \le n+1)$

$f(k) = \sum_{i=1}^{n+1} f(i) \times \frac{\prod_{j=1}^{n+1} (x-j)}{(x-i) \times (-1)^{n+1-i} \times (i-1)! \times (n+1-i)!}$

以下代码求 $\sum_{i=1}^{n} i^k$

```
ll f[N], inv[N];

ll qpow(ll a, ll b) {
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b /= 2;
    }
    return res;
}

ll C(ll n, ll m) {
    return f[n] * inv[m] % mod * inv[n - m] % mod;
}

void init(int M) {
    f[0] = 1;
    for (int i = 1; i <= M; i++) f[i] = f[i - 1] * i % mod;
    inv[M] = qpow(f[M], mod - 2);
    for (int i = M - 1; i >= 0; i--) inv[i] = inv[i + 1] * (i + 1) % mod;
}

void solve() { // 对 k+1 次多项式插值, 且横坐标连续
    int n, k;
    cin >> n >> k;
    vector<ll> y(k + 3);
    for(int i = 1; i <= k + 2; i++){ // 前 k+2 项
        y[i] = (y[i - 1] + qpow(i, k)) % mod;
    }
    if(n <= k + 2){
        cout << y[n] << '\n';
        return;
    }
    init(2e6);
```

```
    vector<ll> p(k + 3);
    ll sum = 1;
    for(int i = 1; i <= k + 2; i++){
        p[i] = qpow(n - i, mod - 2);
        sum = sum * (n - i) % mod;
    }
    ll ans = 0;
    for(int i = 1; i <= k + 2; i++){
        ll tmp = y[i] * sum % mod * p[i] % mod * inv[i - 1] % mod * inv[k + 2 - i] % mod;
        if((k + 2 - i) & 1) ans -= tmp;
        else ans += tmp;
        ans %= mod;
        if(ans < 0) ans += mod;
    }
    cout << ans;
}
```

FFT

```
typedef vector<int> vi;
typedef long long ll;
typedef pair<int, int> pii;

typedef complex<double> C;
typedef vector<double> vd;

void fft(vector<C> &a) {
    int n = (int) a.size(), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1);
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n);
        rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        for (int i = k; i < 2 * k; i++) {
            rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
        }
    }
    vi rev(n);
    for (int i = 0; i < n; i++) {
        rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    }
    for (int i = 0; i < n; i++) {
        if (i < rev[i]) swap(a[i], a[rev[i]]);
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                C z = rt[j + k] * a[i + j + k];
                a[i + j + k] = a[i + j] - z;
                a[i + j] += z;
            }
        }
    }
}

vd conv(const vd &a, const vd &b) {
    if (a.empty() || b.empty()) return {};
    vd res((int) a.size() + (int) b.size() - 1);
    int L = 32 - __builtin_clz((int) res.size()), n = 1 << L;
    vector<C> in(n), out(n);
    copy(a.begin(), a.end(), begin(in));
    for (int i = 0; i < (int) b.size(); i++) {
        in[i].imag(b[i]);
    }
    fft(in);
    for (C &x: in) x *= x;
    for (int i = 0; i < n; i++) {
        out[i] = in[-i & (n - 1)] - conj(in[i]);
    }
    fft(out);
```

```cpp
    for (int i = 0; i < (int) res.size(); i++) {
        res[i] = imag(out[i]) / (4 * n);
    }
    return res;
}

using vll = vector<ll>;

vll gao(const vi &a, const vi &b) { //a 和 b 的卷积
    vd aa((int) a.size()), bb((int) b.size());
    for (int i = 0; i < (int) a.size(); i++) aa[i] = a[i];
    for (int j = 0; j < (int) b.size(); j++) bb[j] = b[j];

    vd cc = conv(aa, bb);
    vll c((int) cc.size());
    for (int i = 0; i < (int) c.size(); i++) c[i] = round(cc[i]);
    return c;
}
```

NTT

```cpp
typedef long long ll;
typedef __int128 i128;
const int N = 1e6 + 5;
const ll mod = 4179340454199820289, G = 3, Gi = 1393113484733273430;

ll qpow(ll a, ll b, ll p) {
    ll res = 1;
    while (b) {
        if (b & 1) res = (i128) res * a % p;
        b >>= 1;
        a = (i128) a * a % p;
    }
    return res;
}

ll f[N], g[N];
int bit, tot, rev[N];

void NTT(ll a[], int type) {
    for (int i = 0; i < tot; i++)
        if (i > rev[i])
            swap(a[i], a[rev[i]]);
    for (int mid = 1; mid < tot; mid <<= 1) {
        ll w1 = qpow(type == 1 ? G : Gi, (mod - 1) / (mid * 2), mod);
        for (int i = 0; i < tot; i += mid * 2) {
            ll wk = 1;
            for (int j = 0; j < mid; j++, wk = (i128) wk * w1 % mod) {
                ll x = a[i + j], y = (i128) wk * a[i + j + mid] % mod;
                a[i + j] = (x + y) % mod, a[i + j + mid] = (x - y + mod) % mod;
            }
        }
    }
    if (type == -1) {
        ll inv = qpow(tot, mod - 2, mod);
        for (int i = 0; i < tot; i++)
            a[i] = (i128) a[i] * inv % mod;
    }
}

void gao(){
    int n = 0, m = 0;// f, g 长度
    for(int i = 0; i < n; i++) f[i] = 0;
    for (int i = 0; i < m; i++) g[i] = 0;
    while ((1 << bit) <= n + m) bit++;
    tot = 1 << bit;
    for (int i = 0; i < tot; i++)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));
    for (int i = n; i < tot; i++) f[i] = 0;
    for (int i = m; i < tot; i++) g[i] = 0;
    NTT(f, 1), NTT(g, 1);
```

```
    for (int i = 0; i < tot; i++) f[i] = (i128) f[i] * g[i] % mod;
    NTT(f, -1);
}
```

高斯消元

```
const double eps = 1e-8;

int sgn(double x) {
    if (fabs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}

double a[4][4], x[4], b[4][4], y[4];
int equ, var;

int Gauss() {
    int i, j, k, col, max_r;
    for (k = 0, col = 0; k < equ && col < var; ++k, ++col) {
        max_r = k;
        for (i = k + 1; i < equ; ++i) if (fabs(a[i][col]) > fabs(a[max_r][col])) max_r = i;
        if (fabs(a[max_r][col]) < eps) return 0;
        if (k != max_r) {
            for (j = col; j < var; ++j) swap(a[k][j], a[max_r][j]);
            swap(x[k], x[max_r]);
        }
        x[k] /= a[k][col];
        for (j = col + 1; j < var; ++j) a[k][j] /= a[k][col];
        a[k][col] = 1;
        for (i = 0; i < equ; ++i) {
            if (i != k) {
                x[i] -= x[k] * a[i][col];
                for (j = col + 1; j < var; ++j) a[i][j] -= a[k][j] * a[i][col];
                a[i][col] = 0;
            }
        }
    }
    return 1;
}

int Gauss(int n, int m){// equ var
    equ = n;
    var = m;
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            a[i][j] = b[i][j];
        }
        x[i] = y[i];
    }
    if(!Gauss()) return 0;
    for(int i = 0; i < n; i++){
        double res = 0;
        for(int j = 0; j < m; j++){
            res += x[j] * b[i][j];
        }
        if(sgn(res - y[i])) return 0;
    }
    return 1;
}
```

图论

LCA

倍增求法

```
const int N = 100010, Log = 20;
int anc[N][Log + 5], depth[N];
```

```cpp
vector<int> e[N];

void dfs(int u, int fa) {
    anc[u][0] = fa;
    depth[u] = depth[fa] + 1;
    for (int i : e[u]) {
        if(i == fa) continue;
        dfs(i, u);
    }
}

void init(int root, int n) {//初始化
    depth[0] = 0;
    dfs(root, 0);
    for (int j = 1; j <= Log; j++) {
        for (int i = 1; i <= n; i++) {
            anc[i][j] = anc[anc[i][j - 1]][j - 1];
        }
    }
}

int rush(int u, int h) {
    for (int i = 0; i <= Log; i++) {
        if (h >> i & 1) u = anc[u][i];
    }
    return u;
}

int qry(int x, int y) {// 求 x 和 y 的 lca
    if (depth[x] < depth[y]) swap(x, y);
    x = rush(x, depth[x] - depth[y]);
    if (x == y) return x;
    for (int i = Log; i >= 0; i--) {
        if (anc[x][i] != anc[y][i]) {
            x = anc[x][i];
            y = anc[y][i];
        }
    }
    return anc[x][0];
}
```

欧拉序求法

```cpp
const int N = 100010, Log = 20;
int logx[N], st[N][Log];//logx[i] 即 log(i) 向下取整  st[i][j] 表示 i 为起点长度为 2^j 区间最值
int first[N], id[N], tot, depth[N];//id 为访问时间戳
vector<int> e[N];

void dfs(int u, int fa, int d) {
    id[++tot] = u; //存储欧拉序所对应的树的节点编号
    depth[tot] = d; //存储每个 dfs 遍历序列号的深度
    first[u] = tot; //表示树的第 x 节点在序列第一次出现的时间戳 y
    for(int v : e[u]){
        if(v == fa) continue;
        dfs(v, u, d + 1);
        id[++tot] = u;
        depth[tot] = d;
    }
}

int Min(int x, int y) {
    return depth[x] > depth[y] ? y : x;
}

void init(int root, int n) {
    dfs(root, 0, 0);
    n = n * 2 - 1; // 欧拉序长度
    logx[0] = -1;
    for (int i = 1; i <= n; i++) logx[i] = logx[i >> 1] + 1;
    for (int i = 1; i <= n; i++) st[i][0] = i;
    for (int j = 1; (1 << j) <= n; j++) {
```

```cpp
        for (int i = 1; i + (1 << j) - 1 <= n; i++) {
            st[i][j] = Min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
        }
    }
}

int qry(int u, int v) {//求 u 和 v 节点的 lca
    int l = first[u], r = first[v];
    if (l > r) swap(l, r);
    int k = logx[r - l + 1];
    return id[Min(st[l][k], st[r - (1 << k) + 1][k])];
}
```

Tarjan

求割点割边点双

```cpp
// 无向图
const int N = 1e3 + 10, M = 1e6 + 10;

struct Edge{
    int v, id;
};

vector<Edge> e[N];
vector<int> bcc[N];//点双
bool cut[N], cut_edge[M];// 割点 割边
int low[N], dfn[N], tot, bcc_cnt, sta[N], top;

void tarjan(int u, int fa) {
    low[u] = dfn[u] = ++tot;
    sta[++top] = u;
    int child = 0, x;
    for (Edge i : e[u]) {
        int v = i.v, id = i.id;
        if (!dfn[v]) {
            child++;
            tarjan(v, u);
            low[u] = min(low[v], low[u]);
            if ((!fa && child > 1) || (fa && low[v] >= dfn[u])) {//割点
                cut[u] = true;
            }
            if (low[v] > dfn[u]) {//割边
                cut_edge[id] = true;
            }
            if (low[v] >= dfn[u]) {//点双
                bcc_cnt++;
                do{
                    x = sta[top--];
                    bcc[bcc_cnt].push_back(x);
                }while(x != v);
                bcc[bcc_cnt].push_back(u);
            }
        } else if (v != fa) {
            low[u] = min(low[u], dfn[v]);
        }
    }
}

void solve() {
    int n, m;
    cin >> n >> m;
    for(int i = 0, u, v; i < m; i++){
        cin >> u >> v;
        e[u].push_back({v, i});
        e[v].push_back({u, i});
    }
    for(int i = 1; i <= n; i++){
        if(!dfn[i]){
            top = 0;
            tarjan(i, 0);
```

```
        }
    }
}
```

求有向图强连通分量 (scc)

2-sat 问题:

- $a \vee b : \neg a \longrightarrow b, \neg b \longrightarrow a$
- $\neg(a \wedge b) : a \longrightarrow \neg b, b \longrightarrow \neg a$

```
//有 n 对点, 每对点只能选一个, m 对关系, 每对关系给出 u, v 两点, 表示 u 和 v 不能同时选
//输出方案或不成立 (NIE)
//编号为 2i-1 和 2i 的代表属于第 i 对点
#include <bits/stdc++.h>

using namespace std;

const int N = 1e5 + 10, M = 1e6 + 10;

vector<int> e[N];
int low[N], dfn[N], tot, sta[N], top;
int scc_cnt, scc[N], in[N];

void tarjan(int u) {
    low[u] = dfn[u] = ++tot;
    sta[++top] = u;
    int x;
    in[u] = 1;
    for (int v : e[u]) {
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[v], low[u]);
        } else if (in[v]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {// scc 强连通分量
        scc_cnt++;
        do {
            x = sta[top--];
            in[x] = 0;
            scc[x] = scc_cnt;  // 染色
        } while (x != u);
    }
}

int re(int x){
    return ((x & 1) ? (x + 1) : (x - 1));
}

void solve() {
    int n, m;
    cin >> n >> m;
    for(int i = 0, u, v; i < m; i++){
        cin >> u >> v;
        e[u].push_back(re(v));
        e[v].push_back(re(u));
    }
    for(int i = 1; i <= n * 2; i++){
        if(!dfn[i]){
            top = 0;
            tarjan(i);
        }
    }
    for(int i = 1; i <= n * 2; i += 2){
        if(scc[i] == scc[i + 1]){
            cout << "NIE\n";
            return;
        }
    }
```

```cpp
    for(int i = 1; i <= n * 2; i += 2){
        int f1 = scc[i], f2 = scc[i + 1];
        if(f1 < f2){
            cout << i << '\n';
        }else{
            cout << i + 1 << '\n';
        }
    }
}

int main() {
    int T = 1;
    ios::sync_with_stdio(false);
    //cin >> T;
    while (T--) solve();
    return 0;
}
```

```cpp
/*
落谷 P6378
n 个点 m 条边的无向图被分成 k 个部分 (点集)。每个部分包含一些点。
请选择一些关键点，使得每个部分恰有一个关键点，且每条边至少有一个端点是关键点。

做法:
每条边就是 (u or v) 的关系。
对于每个点集来说使用前缀和优化可以变为线性复杂度。
设当前点为 u, 前一个点为 v, preu 和 prev 为对应点前缀和点, re 表示不选。
u -> preu, re(preu) -> re(u)
prev -> preu, re(preu) -> re(prev)
prev -> re(u), u -> re(prev)
*/
#include <bits/stdc++.h>

using namespace std;
const int N = 4e6 + 10, Log = 20, inf = 0x3f3f3f3f;

int n, m, k;

vector<int> e[N];
int low[N], dfn[N], tot, sta[N], top;
int scc_cnt, scc[N], in[N];

void tarjan(int u) {
    low[u] = dfn[u] = ++tot;
    sta[++top] = u;
    int x;
    in[u] = 1;
    for (int v : e[u]) {
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[v], low[u]);
        } else if (in[v]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {// scc 强连通分量
        scc_cnt++;
        do {
            x = sta[top--];
            in[x] = 0;
            scc[x] = scc_cnt;  // 染色
        } while (x != u);
    }
}

int re(int x){
    return x > n ? x - n : x + n;
}

void solve() {
    scanf("%d%d%d", &n, &m, &k);
    n *= 2;
```

```cpp
    for(int i = 0, u, v; i < m; i++){
        scanf("%d%d", &u, &v);
        e[re(u)].push_back(v);
        e[re(v)].push_back(u);
    }
    for(int i = 0, cnt; i < k; i++){
        scanf("%d", &cnt);
        int pre = -1;
        for(int j = 0, u; j < cnt; j++){
            scanf("%d", &u);
            int prefix = u + n / 2;
            e[u].push_back(prefix);
            e[re(prefix)].push_back(re(u));
            if(pre != -1){
                e[pre].push_back(prefix);
                e[re(prefix)].push_back(re(pre));
                e[pre].push_back(re(u));
                e[u].push_back(re(pre));
            }
            pre = prefix;
        }
    }
    for(int i = 1; i <= n * 2; i++){
        if(!dfn[i]) tarjan(i);
    }
    for(int i = 1; i <= n; i++){
        if(scc[i] == scc[i + n]){
            puts("NIE");
            return;
        }
    }
    puts("TAK");
}

int main() {
    int T = 1;
    // ios::sync_with_stdio(false);
    // cin >> T;
    while (T--) solve();
    return 0;
}
```

判断仙人掌图

```cpp
// 无向图判断仙人掌
vector<int> e[N];
int dfn[N], dep[N], fa[N], low[N], tot;
int du[N], is_cactus = true;

void DP(int rt, int v){
    int num = dep[v] - dep[rt] + 1;// 环大小
    for(int i = v; i != rt; i = fa[i]){
        du[i]++;
        if(du[i] > 1){
            is_cactus = false;
            return;
        }
    }
}

void tarjan(int x){
    dfn[x] = low[x] = ++tot;
    cc.push_back(x);
    for(int v : e[x]){
        if(v == fa[x]) continue;
        if(!dfn[v]){
            fa[v] = x;
            dep[v] = dep[x] + 1;
            tarjan(v);
            low[x] = min(low[x], low[v]);
        }else{
```

```
            low[x] = min(low[x], dfn[v]);
        }
    }
    if(!is_cactus) return;
    for(int v : e[x]){
        if(fa[v] != x && dfn[x] < dfn[v]) DP(x, v);
    }
}
```

树哈希

p_i 表示第 i 个质数。

- $f_u = 1 + \sum\limits_{v \in son_u} f_v \times p_{sz_v}$
- $f_u = \prod\limits_{v \in son_u} f_v + p_{sz_v}$

对于无根树，找重心 (对于一棵树 n 个节点的无根树，找到一个点，将无根树变为以该点为根的有根树时，最大子树的结点数最小，这个点就是重心)，一颗树的重心最多只有两个，分别比较即可。

以下为另一种树哈希。

```
//copy from https://uoj.ac/submission/579874
//树哈希
#include <cstdio>
#include <cctype>
#include <chrono>
#include <random>
#include <algorithm>

typedef unsigned uint;
typedef long long unsigned uint64;

uint64 xorshift(uint64 x) {
    x ^= x << 13;
    x ^= x >> 7;
    x ^= x << 17;
    return x;
}

const int Max_N = 1000005;

std::mt19937_64 engine((std::random_device())() ^ std::chrono::steady_clock::now().time_since_epoch().count() ^
 size_t(new char));
uint64 S = std::uniform_int_distribution<uint64>(1, -1)(engine);

int N;
int first[Max_N];
int Next[Max_N * 2], To[Max_N * 2];
int ecnt;

int hfirst[1 << 20];
int hNext[Max_N];
uint64 hTo[Max_N];
int hcnt;

void insert(uint64 x) {
    int b = x & ((1 << 20) - 1);
    for (int e = hfirst[b]; e; e = hNext[e])
        if (x == hTo[e])
            return;
    ++hcnt, hNext[hcnt] = hfirst[b], hTo[hcnt] = x, hfirst[b] = hcnt;
}

uint64 dfs(const int v, const int p) {
    uint64 h = S;
    for (int e = first[v]; e; e = Next[e])
        if (To[e] != p)
            h += xorshift(dfs(To[e], v));
    insert(h);
```

```cpp
    return h;
}

int main(int argc, char **argv) {
    cin >> N;
    ecnt = 0;
    for (int i = 1; i != N; ++i) {
        int v, w;
        cin >> v >> w;
        ++ecnt, Next[ecnt] = first[v], To[ecnt] = w, first[v] = ecnt;
        ++ecnt, Next[ecnt] = first[w], To[ecnt] = v, first[w] = ecnt;
    }

    dfs(1, 0);

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

    return 0;
}
```

二分图

最大匹配 (匈牙利)

- k-正则图：各顶点的度均为 k 的无向简单图
- 最大匹配数：最大匹配的匹配边的数目
- 最大独立集数：选取最多的点集，使点集中任意两点均不相连
- 最小点覆盖数：选取最少的点集，使任意一条边都至少有一个端点在点集中
- 最大匹配数 = 最小点覆盖数
- 最大独立集数 = 顶点数 - 最大匹配数

```cpp
int n, m;
int mp[N][N], link[N];// 存图  link i 右部图 i 点在左部图的连接点
bool vis[N];// 是否在交替路中

bool dfs(int u){
    for(int v = 1; v <= m; v++){
        if(vis[v] || !mp[u][v]) continue;
        vis[v] = true;
        if(link[v] == -1 || dfs(link[v])){
            link[v] = u;
            return true;
        }
    }
    return false;
}

int hungarian(){
    int ans = 0;
    for(int i = 1; i <= m; i++) link[i] = -1;
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= m; j++) vis[j] = false;
        if(dfs(i)) ans++;
    }
    return ans;
}

void solve() {
    int e;
    cin >> n >> m >> e;
    for(int i = 0, u, v; i < e; i++){
        cin >> u >> v;
        mp[u][v] = true;
    }
    cout << hungarian();
}
```

也可建立一个源点和汇点, 将源点连向所有左部点, 左部点连向右部点, 右部点连向汇点, 且所有流量为 1, 然后跑最大流即为最大匹配

最大权匹配

KM (时间复杂度 n^3)

适用于二分图的最大权完美匹配，若两部分点个数不同，可以增加一些虚点并将边权置 0 。

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

using namespace std;

typedef long long ll;
typedef pair<int, int> pii;
//Data
const int N = 500 + 10;
const ll inf = 1e11;
int nx;

//KM
ll c[N], e[N][N], kb[N], ka[N];
int mb[N], p[N], vb[N];

void Bfs(int u) {
    int a, v, vl = 0;
    ll d;
    for (int i = 1; i <= nx; i++) p[i] = 0, c[i] = inf;
    mb[v] = u;
    do {
        a = mb[v], d = inf, vb[v] = 1;
        for (int b = 1; b <= nx; b++)
            if (!vb[b]) {
                if (c[b] > ka[a] + kb[b] - e[a][b])
                    c[b] = ka[a] + kb[b] - e[a][b], p[b] = v;
                if (c[b] < d) d = c[b], vl = b;
            }
        for (int b = 0; b <= nx; b++)
            if (vb[b]) ka[mb[b]] -= d, kb[b] += d;
            else c[b] -= d;
        v = vl;
    } while (mb[v]);
    while (v) mb[v] = mb[p[v]], v = p[v];
}

ll KM() {
    for (int i = 1; i <= nx; i++) mb[i] = 0, ka[i] = kb[i] = 0;
    for (int a = 1; a <= nx; a++) {
        for (int b = 1; b <= nx; b++) vb[b] = 0;
        Bfs(a);
    }
    ll res = 0;
    for (int b = 1; b <= nx; b++) res += e[mb[b]][b];
    return res;
}

void solve() {
    int n, m;
    scanf("%d%d", &n, &m);
    nx = n;
    for (int a = 1; a <= nx; a++)
        for (int b = 1; b <= nx; b++) e[a][b] = -inf;
    for (int i = 1, u, v, w; i <= m; i++) {
        scanf("%d%d%d", &u, &v, &w);
        e[u][v] = max(e[u][v], w * 1ll);
    }
    printf("%lld\n", KM());
    for (int u = 1; u <= ny; u++) printf("%d ", mb[u]);
    puts("");
}

int main() {
    solve();
    return 0;
}
```

费用流（时间复杂度 $n \times e \times f$ 或 $e \times log(n) \times f$）

欧拉回路

```
// 若有奇数度数的点 可先建若干条虚边使其度数变为偶数
const int N = 5e5 + 10;

struct Edge{
    int to, next;
    int index; // 边在图中编号
    int dir; // 方向
    bool flag;
}edge[N];
int head[N], tot;

void init(){
    memset(head, -1, sizeof(head));
    tot = 0;
}

void add(int u, int v, int index){
    edge[tot] = {v, head[u], index, 0, false};
    head[u] = tot++;
    edge[tot] = {u, head[v], index, 1, false};
    head[v] = tot++;
}

int du[N];// 点的度
vector<int> ans;

void dfs(int u){
    for(int i = head[u]; i != -1; i = edge[i].next){
        if(!edge[i].flag){
            edge[i].flag = true;
            edge[i ^ 1].flag = true;
            dfs(edge[i].to);
            ans.push_back(i);
        }
    }
}
```

最大流

```
template<int V>
struct MF {
    using U = int; //流量类型
    const U INF = 0x3f3f3f3f;
    struct Edge {
        int v, inv; //有向边指向 v 点, 反向边在 v 邻接表中的位置 inv
        U w; //流量
    };
    int s, t, n; //源点、汇点、总数
    vector<Edge> e[V];
    int cur[V], qe[V];
    U dis[V];
    bool vis[V];

    void add(int u, int v, U w, U rw = 0) {
        e[u].push_back({v, (int) e[v].size(), w});
        e[v].push_back({u, (int) e[u].size() - 1, rw});
    }

    bool bfs() {
        for (int i = 1; i <= n; i++) vis[i] = false;
        int lt = 0, rt = 0;
        qe[rt++] = s;
        vis[s] = true;
        dis[s] = 0;
        while (lt < rt) {
            int u = qe[lt++];
```

```cpp
            for(auto [v, inv, w] : e[u]){
                if(!vis[v] && w){
                    vis[v] = true;
                    dis[v] = dis[u] + 1;
                    qe[rt++] = v;
                }
            }
        }
        return vis[t];
    }

    U dfs(int x, U flow) {
        if (x == t || !flow) return flow;
        U delta = 0, f;
        for (int i = cur[x]; i < e[x].size(); i++) {
            auto [v, inv, w] = e[x][i];
            cur[x] = i;
            if (dis[v] == dis[x] + 1 && (f = dfs(v, min(flow, w))) > 0) {
                e[x][i].w -= f;
                e[v][inv].w += f;
                flow -= f;
                delta += f;
                if (flow == 0) break;
            }
        }
        return delta;
    }

    U MaxFlow() {
        U ans = 0;
        while (bfs()) {
            for (int i = 1; i <= n; i++) cur[i] = 0;
            ans += dfs(s, inf);
        }
        return ans;
    }

    void init(){
        for(int i = 1; i <= n; i++){
            vector<Edge>().swap(e[i]);
        }
    }
};
```

最小费用最大流 (费用流)

SPFA(时间复杂度 $n \times e \times f$)

```cpp
const int inf = 0x3f3f3f3f, N = 100010;
struct edge {
    int to, next;
    ll w, fee;//w 为流量  fee 为费用
} e[N];
int head[N], idx;
int pre[N], id[N];//pre 前一个节点   id 当前节点的边的 idx
int s, t, n;
ll dist[N], flow[N];//dist 费用 (距离)  flow 流量
bool vis[N];

void init() {
    idx = 0;
    for (int i = 1; i <= n; i++) head[i] = -1;
}

void add(int a, int b, ll c, ll fee) {
    e[idx].to = b;
    e[idx].w = c;
    e[idx].next = head[a];
    e[idx].fee = fee;
    head[a] = idx++;
}
```

```cpp

bool spfa() {
    for (int i = 1; i <= n; i++) {
        vis[i] = false;
        dist[i] = inf;
        flow[i] = inf;
    }
    queue<int> q;
    q.push(s);
    vis[s] = true;
    pre[t] = -1;
    dist[s] = 0;
    while (!q.empty()) {
        int x = q.front();
        q.pop();
        vis[x] = false;
        for (int i = head[x]; i != -1; i = e[i].next) {
            int to = e[i].to;
            ll w = e[i].w, fee = e[i].fee;
            if (w && dist[to] > dist[x] + fee) {
                dist[to] = dist[x] + fee;
                flow[to] = min(flow[x], w);
                pre[to] = x;
                id[to] = i;
                if (!vis[to]) {
                    q.push(to);
                    vis[to] = true;
                }
            }
        }
    }
    return dist[t] != inf;
}

void MinFee() {
    ll minfee = 0, maxflow = 0;
    while (spfa()) {
        int now = t;
        maxflow += flow[t];
        minfee += flow[t] * dist[t];
        while (now != s) {
            e[id[now]].w -= flow[t];
            e[id[now] ^ 1].w += flow[t];
            now = pre[now];
        }
    }
    printf("%lld %lld\n", maxflow, minfee);
}
```

dij (边权为正, 时间复杂度 $e \times log(n) \times f$)

```cpp
//适用于正边权
template<int V>
struct MCMF {
    using U = int; //流量类型
    using T = int; //费用类型
    using P = pair<T, int>;
    const U INF = 0x3f3f3f3f;
    const T FINF = 0x3f3f3f3f;
    struct Edge {
        int v, inv; //有向边指向 v 点, 反向边在 v 邻接表中的位置 inv
        U w; //流量
        T fee; //费用
    };
    int s, t, n; //源点、汇点、总数
    vector<Edge> e[V];
    int pre[V], id[V];//边: pre[v] -> v, id[v] 表示边在 pre[v] 中的下标
    U flow[V];
    T dis[V], h[V];
    bool vis[V];

```

```cpp
    void add(int u, int v, U w, T fee) {
        e[u].push_back({v, (int) e[v].size(), w, fee});
        e[v].push_back({u, (int) e[u].size() - 1, 0, -fee});
    }

    bool dij() {
        for (int i = 1; i <= n; i++) {
            vis[i] = false;
            dis[i] = FINF;
            flow[i] = INF;
        }
        dis[s] = 0;
        priority_queue<P, vector<P>, greater<>> q;
        q.push({dis[s], s});
        while (!q.empty()) {
            auto [d, u] = q.top();
            q.pop();
            if (vis[u]) continue;
            vis[u] = true;
            for (int i = 0; i < e[u].size(); i++) {
                auto [v, inv, w, fee] = e[u][i];
                if (w && dis[v] > dis[u] + h[u] - h[v] + fee) {
                    dis[v] = dis[u] + h[u] - h[v] + fee;
                    flow[v] = min(flow[u], w);
                    pre[v] = u;
                    id[v] = i;
                    q.push({dis[v], v});
                }
            }
        }
        return dis[t] < FINF;
    }

    void init(){
        for(int i = 1; i <= n; i++){
            h[i] = FINF;
            vis[i] = false;
        }
        queue<int> q;
        h[s] = 0, vis[s] = true;
        q.push(s);
        while(!q.empty()){
            int u = q.front();
            q.pop();
            vis[u] = false;
            for(auto [v, inv, w, fee] : e[u]){
                if(w && h[v] > h[u] + fee){
                    h[v] = h[u] + fee;
                    if(!vis[v]){
                        vis[v] = true;
                        q.push(v);
                    }
                }
            }
        }
    }

    pair<U, T> MinCostMaxFlow() {
        T minfee = 0;
        U maxflow = 0;
        init();
        while (dij()) {
            for (int i = 1; i <= n; i++) {
                h[i] += dis[i];
            }
            int v = t;
            maxflow += flow[t];
            minfee += flow[t] * h[t];
            while (v != s) {
                int u = pre[v];
                //u -> v
```

```
                e[u][id[v]].w -= flow[t];
                int inv = e[u][id[v]].inv;
                e[v][inv].w += flow[t];
                v = u;
            }
        }
        return {maxflow, minfee};
    }
};
```

```
//from reborn
//适用于正边权
#pragma GCC optimize(2)
#pragma GCC optimize(3,"Ofast","inline")
#include<bits/stdc++.h>
using namespace std;
const int N = 30005;
typedef long long ll;
 ll tot = 2;
#define int long long

const int INF = 0x3f3f3f3f3f3f3f3f;

#define V vector
#define pb push_back
#define eb emplace_back
#define sz(x) (int)size(x)

// Min cost max flow {{{
template<typename flow_t = int, typename cost_t = int>
struct MinCostFlow {
    struct Edge {
        cost_t c;
        flow_t f;  // DO NOT USE THIS DIRECTLY. SEE getFlow(Edge const& e)
        int to, rev;
        Edge(int _to, cost_t _c, flow_t _f, int _rev) : c(_c), f(_f), to(_to), rev(_rev) {}
    };

    int N, S, T;
    vector<vector<Edge> > G;
    MinCostFlow(int _N, int _S, int _T) : N(_N), S(_S), T(_T), G(_N), eps(0) {}

    void AddEdge(int a, int b, flow_t cap, cost_t cost) {
    assert(cap >= 0);
        assert(a >= 0 && a < N && b >= 0 && b < N);
        if (a == b) { assert(cost >= 0); return; }
        cost *= N;
        eps = max(eps, abs(cost));
        G[a].emplace_back(b, cost, cap, G[b].size());
        G[b].emplace_back(a, -cost, 0, G[a].size() - 1);
    }

    flow_t getFlow(Edge const &e) {
        return G[e.to][e.rev].f;
    }

    pair<flow_t, cost_t> minCostMaxFlow() {
        cost_t retCost = 0;
        for (int i = 0; i<N; ++i) {
            for (Edge &e : G[i]) {
                retCost += e.c*(e.f);
            }
        }
        //find max-flow
        flow_t retFlow = max_flow();
        h.assign(N, 0); ex.assign(N, 0);
        isq.assign(N, 0); cur.assign(N, 0);
        queue<int> q;
        for (; eps; eps >>= scale) {
            //refine
            fill(cur.begin(), cur.end(), 0);
            for (int i = 0; i < N; ++i) {
```

```
                for (auto &e : G[i]) {
                    if (h[i] + e.c - h[e.to] < 0 && e.f) push(e, e.f);
                }
            }
            for (int i = 0; i < N; ++i) {
                if (ex[i] > 0){
                    q.push(i);
                    isq[i] = 1;
                }
            }
            // make flow feasible
            while (!q.empty()) {
                int u = q.front(); q.pop();
                isq[u]=0;
                while (ex[u] > 0) {
                    if (cur[u] == G[u].size()) {
                        relabel(u);
                    }
                    for (unsigned int &i=cur[u], max_i = G[u].size(); i < max_i; ++i) {
                        Edge &e = G[u][i];
                        if (h[u] + e.c - h[e.to] < 0) {
                            push(e, ex[u]);
                            if (ex[e.to] > 0 && isq[e.to] == 0) {
                                q.push(e.to);
                                isq[e.to] = 1;
                            }
                            if (ex[u] == 0) break;
                        }
                    }
                }
            }
            if (eps > 1 && eps>>scale == 0) {
                eps = 1<<scale;
            }
        }
        for (int i = 0; i < N; ++i) {
            for (Edge &e : G[i]) {
                retCost -= e.c*(e.f);
            }
        }
        return make_pair(retFlow, retCost / 2 / N);
    }

private:
    static constexpr cost_t INFCOST = numeric_limits<cost_t>::max()/2;
    static constexpr int scale = 2;

    cost_t eps;
    vector<unsigned int> isq, cur;
    vector<flow_t> ex;
    vector<cost_t> h;
    vector<vector<int> > hs;
    vector<int> co;

    void add_flow(Edge& e, flow_t f) {
        Edge &back = G[e.to][e.rev];
        if (!ex[e.to] && f) {
            hs[h[e.to]].push_back(e.to);
        }
        e.f -= f; ex[e.to] += f;
        back.f += f; ex[back.to] -= f;
    }

    void push(Edge &e, flow_t amt) {
        if (e.f < amt) amt = e.f;
        e.f -= amt; ex[e.to] += amt;
        G[e.to][e.rev].f += amt; ex[G[e.to][e.rev].to] -= amt;
    }

    void relabel(int vertex){
        cost_t newHeight = -INFCOST;
```

```
        for (unsigned int i = 0; i < G[vertex].size(); ++i){
            Edge const&e = G[vertex][i];
            if(e.f && newHeight < h[e.to] - e.c){
                newHeight = h[e.to] - e.c;
                cur[vertex] = i;
            }
        }
        h[vertex] = newHeight - eps;
    }

    flow_t max_flow() {
        ex.assign(N, 0);
        h.assign(N, 0); hs.resize(2*N);
        co.assign(2*N, 0); cur.assign(N, 0);
        h[S] = N;
        ex[T] = 1;
        co[0] = N-1;
        for (auto &e : G[S]) {
            add_flow(e, e.f);
        }
        if (hs[0].size()) {
            for (int hi = 0; hi>=0;) {
                int u = hs[hi].back();
                hs[hi].pop_back();
                while (ex[u] > 0) { // discharge u
                    if (cur[u] == G[u].size()) {
                        h[u] = 1e9;
                        for(unsigned int i = 0; i < G[u].size(); ++i) {
                            auto &e = G[u][i];
                            if (e.f && h[u] > h[e.to]+1) {
                                h[u] = h[e.to]+1, cur[u] = i;
                            }
                        }
                        if (++co[h[u]], !--co[hi] && hi < N) {
                            for (int i = 0; i < N; ++i) {
                                if (hi < h[i] && h[i] < N) {
                                    --co[h[i]];
                                    h[i] = N + 1;
                                }
                            }
                        }
                        hi = h[u];
                    } else if (G[u][cur[u]].f && h[u] == h[G[u][cur[u]].to]+1) {
                        add_flow(G[u][cur[u]], min(ex[u], G[u][cur[u]].f));
                    } else {
                        ++cur[u];
                    }
                }
                while (hi>=0 && hs[hi].empty()) {
                    --hi;
                }
            }
        }
        return -ex[S];
    }
};
```

```
//from bbg
//可以跑负权
#include <bits/stdc++.h>

using ll = long long;
using ull = unsigned long long;

struct MCMF {
    struct Edge {
        int nxt, to;
        ll cap, cost;
    };
    std::vector<Edge> edges;
    std::vector<int> head, fa, fe;
    std::vector<ll> dual, mark, cyc;
```

```
    ll ti, sum;

    MCMF(int n) : head(n, 0), fa(n), fe(n), dual(n), mark(n), cyc(n + 1), ti(0) {
        edges.push_back({0, 0, 0, 0});
        edges.push_back({0, 0, 0, 0});
    }

    int addEdge(int u, int v, ll cap, ll cost) {
        sum += std::abs(cost);
        assert(edges.size() % 2 == 0);
        int e = edges.size();
        edges.push_back({head[u], v, cap, cost});
        head[u] = e;
        edges.push_back({head[v], u, 0, -cost});
        head[v] = e + 1;
        return e;
    }

    void initTree(int x) {
        mark[x] = 1;
        for (int i = head[x]; i; i = edges[i].nxt) {
            int v = edges[i].to;
            if (!mark[v] and edges[i].cap) {
                fa[v] = x, fe[v] = i;
                initTree(v);
            }
        }
    }

    int phi(int x) {
        if (mark[x] == ti)
            return dual[x];
        return mark[x] = ti, dual[x] = phi(fa[x]) - edges[fe[x]].cost;
    }

    void pushFlow(int e, ll &cost) {
        int pen = edges[e ^ 1].to, lca = edges[e].to;
        ti++;
        while (pen)
            mark[pen] = ti, pen = fa[pen];
        while (mark[lca] != ti)
            mark[lca] = ti, lca = fa[lca];

        int e2 = 0;
        ll f = edges[e].cap;
        int path = 2, clen = 0;
        for (int i = edges[e ^ 1].to; i != lca; i = fa[i]) {
            cyc[++clen] = fe[i];
            if (edges[fe[i]].cap < f)
                f = edges[fe[e2 = i] ^ (path = 0)].cap;
        }
        for (int i = edges[e].to; i != lca; i = fa[i]) {
            cyc[++clen] = fe[i] ^ 1;
            if (edges[fe[i] ^ 1].cap <= f)
                f = edges[fe[e2 = i] ^ (path = 1)].cap;
        }
        cyc[++clen] = e;

        for (int i = 1; i <= clen; ++i) {
            edges[cyc[i]].cap -= f, edges[cyc[i] ^ 1].cap += f;
            cost += edges[cyc[i]].cost * f;
        }
        if (path == 2)
            return;

        int laste = e ^ path, last = edges[laste].to, cur = edges[laste ^ 1].to;
        while (last != e2) {
            mark[cur]--;
            laste ^= 1;
            std::swap(laste, fe[cur]);
            std::swap(last, fa[cur]);
```

```cpp
            std::swap(last, cur);
        }
    }

    std::pair<ll, ll> compute(int s, int t) {
        ll tot = sum;
        int ed = addEdge(t, s, 1e18, -tot);
        ll cost = 0;
        initTree(0);
        mark[0] = ti = 2;
        fa[0] = cost = 0;
        int ncnt = edges.size() - 1;
        for (int i = 2, pre = ncnt; i != pre; i = i == ncnt ? 2 : i + 1) {
            if (edges[i].cap and
                edges[i].cost < phi(edges[i ^ 1].to) - phi(edges[i].to))
                pushFlow(pre = i, cost);
        }
        ll flow = edges[ed ^ 1].cap;
        cost += tot * flow;
        return {cost, flow};
    }
};

void run(int tCase) {
    int n, m, s, t;
    std::cin >> n >> m >> s >> t;
    s--, t--;
    MCMF mcmf(n);
    for (int i = 0; i < m; ++i) {
        int u, v, cap, cost;
        std::cin >> u >> v >> cap >> cost;
        u--, v--;
        mcmf.addEdge(u, v, cap, cost);
    }
    auto [cost, flow] = mcmf.compute(s, t);
    std::cout << flow << ' ' << cost << '\n';
}

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int T = 1;
//    std::cin >> T;
    for (int t = 1; t <= T; ++t) {
        run(t);
    }
    return 0;
}
```

竞赛图

一张完全有向图。其中一条由 u 指向 v 的边表示 u 能打败 v。

经典结论：

1. 缩点之后是一条链，拓扑序唯一。
2. 拓扑序在前的 SCC 的任意节点的入度严格小于拓扑序在后的 SCC 的节点。
3. 若按照入度从小到大排序后，前 i 个点的入度和等于 $\frac{i\times(i-1)}{2}$ 时，说明出现了一个新的 SCC。

```cpp
//cf1498E
#include <bits/stdc++.h>

using namespace std;
typedef pair<int, int> pii;
const int N = 1e5 + 10, Log = 20, inf = 0x3f3f3f3f;

void solve() {
    int n;
    cin >> n;
    string s(n, '1');
```

```cpp
    vector<pii> v;
    for(int i = 1, x; i <= n; i++){
        cin >> x;
        v.push_back({x, i});
    }
    sort(v.begin(), v.end());
    int x = 0, y = 0, mx = -1;
    for(int i = 0, sum = 0, l = 0; i < n; i++){
        sum += v[i].first;
        if(sum == (i + 1) * i / 2){
            if(i > l && v[i].first - v[l].first > mx){
                mx = v[i].first - v[l].first;
                x = v[l].second;
                y = v[i].second;
            }
            l = i + 1;
        }
    }
    cout << "! " << x << ' ' << y << endl;
}

int main() {
    int T = 1;
    ios::sync_with_stdio(false);
    // cin >> T;
    while (T--) solve();
    return 0;
}
```

计算几何

二维几何

```cpp
namespace Geometry {
using T = ll;
constexpr T eps = 0;

bool eq(const T &x, const T &y) { return abs(x - y) <= eps; }
inline constexpr int type(T x, T y) {
    if(x == 0 and y == 0) return 0;
    if(y < 0 or (y == 0 and x > 0)) return -1;
    return 1;
}
struct Point {
    T x, y;
    constexpr Point(T _x = 0, T _y = 0) : x(_x), y(_y) {}
    constexpr Point operator+() const noexcept { return *this; }
    constexpr Point operator-() const noexcept { return Point(-x, -y); }
    constexpr Point operator+(const Point &p) const { return Point(x + p.x, y + p.y); }
    constexpr Point operator-(const Point &p) const { return Point(x - p.x, y - p.y); }
    constexpr Point &operator+=(const Point &p) { return x += p.x, y += p.y, *this; }
    constexpr Point &operator-=(const Point &p) { return x -= p.x, y -= p.y, *this; }
    constexpr T operator*(const Point &p) const { return x * p.x + y * p.y; }
    constexpr Point &operator*=(const T &k) { return x *= k, y *= k, *this; }
    constexpr Point operator*(const T &k) { return Point(x * k, y * k); }
    constexpr bool operator==(const Point &r) const noexcept { return r.x == x and r.y == y; }
    constexpr T cross(const Point &r) const { return x * r.y - y * r.x; }

    constexpr bool operator<(const Point &r) const { return pair(x, y) < pair(r.x, r.y); }

    // 1 : left, 0 : same, -1 : right
    constexpr int toleft(const Point &r) const {
        auto t = cross(r);
        return t > eps ? 1 : t < -eps ? -1 : 0;
    }

    constexpr bool arg_cmp(const Point &r) const {
        int L = type(x, y), R = type(r.x, r.y);
        if(L != R) return L < R;

```

```
        T X = x * r.y, Y = r.x * y;
        if(X != Y) return X > Y;
        return x < r.x;
    }
};
bool arg_cmp(const Point &l, const Point &r) { return l.arg_cmp(r); }
ostream &operator<<(ostream &os, const Point &p) { return os << p.x << " " << p.y; }
istream &operator>>(istream &is, Point &p) {
    is >> p.x >> p.y;
    return is;
}

struct Line {
    Point a, b;
    Line() = default;
    Line(Point a, Point b) : a(a), b(b) {}
    // ax + by = c
    Line(T A, T B, T C) {
        if(A == 0) {
            a = Point(0, C / B), b = Point(1, C / B);
        } else if(B == 0) {
            a = Point(C / A, 0), b = Point(C / A, 1);
        } else {
            a = Point(0, C / B), b = Point(C / A, 0);
        }
    }
    // 1 : left, 0 : same, -1 : right
    constexpr int toleft(const Point &r) const {
        auto t = (b - a).cross(r - a);
        return t > eps ? 1 : t < -eps ? -1 : 0;
    }

    friend std::ostream &operator<<(std::ostream &os, Line &ls) {
        return os << "{"
                  << "(" << ls.a.x << ", " << ls.a.y << "), (" << ls.b.x << ", " << ls.b.y << ")}";
    }
};
istream &operator>>(istream &is, Line &p) { return is >> p.a >> p.b; }

struct Segment : Line {
    Segment() = default;
    Segment(Point a, Point b) : Line(a, b) {}
};

ostream &operator<<(ostream &os, Segment &p) { return os << p.a << " to " << p.b; }
istream &operator>>(istream &is, Segment &p) {
    is >> p.a >> p.b;
    return is;
}

struct Circle {
    Point p;
    T r;
    Circle() = default;
    Circle(Point p, T r) : p(p), r(r) {}
};

using pt = Point;
using Points = vector<pt>;
using Polygon = Points;
T cross(const pt &x, const pt &y) { return x.x * y.y - x.y * y.x; }
T dot(const pt &x, const pt &y) { return x.x * y.x + x.y * y.y; }

T abs2(const pt &x) { return dot(x, x); }
// http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_1_C
// 点の回転方向
int ccw(const Point &a, Point b, Point c) {
    b = b - a, c = c - a;
    if(cross(b, c) > 0) return +1;   // "COUNTER_CLOCKWISE"
    if(cross(b, c) < 0) return -1;   // "CLOCKWISE"
    if(dot(b, c) < 0) return +2;     // "ONLINE_BACK"
```

```cpp
    if(abs2(b) < abs2(c)) return -2; // "ONLINE_FRONT"
    return 0;                         // "ON_SEGMENT"
}

// http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_2_A
// 平行判定
bool parallel(const Line &a, const Line &b) { return (cross(a.b - a.a, b.b - b.a) == 0); }

// http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_2_A
// 垂直判定
bool orthogonal(const Line &a, const Line &b) { return (dot(a.a - a.b, b.a - b.b) == 0); }

bool intersect(const Line &l, const Point &p) { return abs(ccw(l.a, l.b, p)) != 1; }

bool intersect(const Line &l, const Line &m) { return !parallel(l, m); }

bool intersect(const Segment &s, const Point &p) { return ccw(s.a, s.b, p) == 0; }

bool intersect(const Line &l, const Segment &s) { return cross(l.b - l.a, s.a - l.a) * cross(l.b - l.a, s.b - l.a) <= 0; }

// http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_2_B
bool intersect(const Segment &s, const Segment &t) { return ccw(s.a, s.b, t.a) * ccw(s.a, s.b, t.b) <= 0 && ccw(t.a, t.b, s.a) * ccw(t.a, t.b, s.b) <= 0; }

bool intersect(const Polygon &ps, const Polygon &qs) {
    int pl = si(ps), ql = si(qs), i = 0, j = 0;
    while((i < pl or j < ql) and (i < 2 * pl) and (j < 2 * ql)) {
        auto ps0 = ps[(i + pl - 1) % pl], ps1 = ps[i % pl];
        auto qs0 = qs[(j + ql - 1) % ql], qs1 = qs[j % ql];
        if(intersect(Segment(ps0, ps1), Segment(qs0, qs1))) return true;
        Point a = ps1 - ps0;
        Point b = qs1 - qs0;
        T v = cross(a, b);
        T va = cross(qs1 - qs0, ps1 - qs0);
        T vb = cross(ps1 - ps0, qs1 - ps0);

        if(!v and va < 0 and vb < 0) return false;
        if(!v and !va and !vb) {
            i += 1;
        } else if(v >= 0) {
            if(vb > 0)
                i += 1;
            else
                j += 1;
        } else {
            if(va > 0)
                j += 1;
            else
                i += 1;
        }
    }
    return false;
}

T norm(const Point &p) { return p.x * p.x + p.y * p.y; }
Point projection(const Segment &l, const Point &p) {
    T t = dot(p - l.a, l.a - l.b) / norm(l.a - l.b);
    return l.a + (l.a - l.b) * t;
}

Point crosspoint(const Line &l, const Line &m) {
    T A = cross(l.b - l.a, m.b - m.a);
    T B = cross(l.b - l.a, l.b - m.a);
    if(A == 0 and B == 0) return m.a;
    return m.a + (m.b - m.a) * (B / A);
}

// http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_2_C
Point crosspoint(const Segment &l, const Segment &m) { return crosspoint(Line(l), Line(m)); }
```

```cpp
// http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_3_B
// 凸性判定
bool is_convex(const Points &p) {
    int n = (int)p.size();
    for(int i = 0; i < n; i++) {
        if(ccw(p[(i + n - 1) % n], p[i], p[(i + 1) % n]) == -1) return false;
    }
    return true;
}

Points convex_hull(Points p) {
    int n = p.size(), k = 0;
    if(n <= 2) return p;
    sort(begin(p), end(p), [](pt x, pt y) { return (x.x != y.x ? x.x < y.x : x.y < y.y); });
    Points ch(2 * n);
    for(int i = 0; i < n; ch[k++] = p[i++]) {
        while(k >= 2 && cross(ch[k - 1] - ch[k - 2], p[i] - ch[k - 1]) <= 0) --k;
    }
    for(int i = n - 2, t = k + 1; i >= 0; ch[k++] = p[i--]) {
        while(k >= t && cross(ch[k - 1] - ch[k - 2], p[i] - ch[k - 1]) <= 0) --k;
    }
    ch.resize(k - 1);
    return ch;
}

// 面積の 2 倍
T area2(const Points &p) {
    T res = 0;
    rep(i, si(p)) { res += cross(p[i], p[i == si(p) - 1 ? 0 : i + 1]); }
    return res;
}

enum { _OUT, _ON, _IN };

int contains(const Polygon &Q, const Point &p) {
    bool in = false;
    for(int i = 0; i < Q.size(); i++) {
        Point a = Q[i] - p, b = Q[(i + 1) % Q.size()] - p;
        if(a.y > b.y) swap(a, b);
        if(a.y <= 0 && 0 < b.y && cross(a, b) < 0) in = !in;
        if(cross(a, b) == 0 && dot(a, b) <= 0) return _ON;
    }
    return in ? _IN : _OUT;
}

Polygon Minkowski_sum(const Polygon &P, const Polygon &Q) {
    vector&lt;Segment&gt; e1(P.size()), e2(Q.size()), ed(P.size() + Q.size());
    const auto cmp = [](const Segment &u, const Segment &v) { return (u.b - u.a).arg_cmp(v.b - v.a); };
    rep(i, P.size()) e1[i] = {P[i], P[(i + 1) % P.size()]};
    rep(i, Q.size()) e2[i] = {Q[i], Q[(i + 1) % Q.size()]};
    rotate(begin(e1), min_element(all(e1), cmp), end(e1));
    rotate(begin(e2), min_element(all(e2), cmp), end(e2));
    merge(all(e1), all(e2), begin(ed), cmp);
    const auto check = [](const Points &res, const Point &u) {
        const auto back1 = res.back(), back2 = *prev(end(res), 2);
        return eq(cross(back1 - back2, u - back2), eps) and dot(back1 - back2, u - back1) >= -eps;
    };
    auto u = e1[0].a + e2[0].a;
    Points res{u};
    res.reserve(P.size() + Q.size());
    for(const auto &v : ed) {
        u = u + v.b - v.a;
        while(si(res) >= 2 and check(res, u)) res.pop_back();
        res.eb(u);
    }
    if(res.size() and check(res, res[0])) res.pop_back();
    return res;
}

// -1 : on, 0 : out, 1 : in
// O(log(n))
```

```cpp
int is_in(const Polygon &p, const Point &a) {
    if(p.size() == 1) return a == p[0] ? -1 : 0;
    if(p.size() == 2) return intersect(Segment(p[0], p[1]), a);
    if(a == p[0]) return -1;
    if((p[1] - p[0]).toleft(a - p[0]) == -1 || (p.back() - p[0]).toleft(a - p[0]) == 1) return 0;
    const auto cmp = [&](const Point &u, const Point &v) { return (u - p[0]).toleft(v - p[0]) == 1; };
    const size_t i = lower_bound(p.begin() + 1, p.end(), a, cmp) - p.begin();
    if(i == 1) return intersect(Segment(p[0], p[i]), a) ? -1 : 0;
    if(i == p.size() - 1 && intersect(Segment(p[0], p[i]), a)) return -1;
    if(intersect(Segment(p[i - 1], p[i]), a)) return -1;
    return (p[i] - p[i - 1]).toleft(a - p[i - 1]) > 0;
}

Points halfplane_intersection(vector<Line> L, const T inf = 1e9) {
    Point box[4] = {Point(inf, inf), Point(-inf, inf), Point(-inf, -inf), Point(inf, -inf)};
    rep(i, 4) { L.emplace_back(box[i], box[(i + 1) % 4]); }
    sort(all(L), [](const Line &l, const Line &r) { return (l.b - l.a).arg_cmp(r.b - r.a); });
    deque<Line> dq;
    int len = 0;
    auto check = [](const Line &a, const Line &b, const Line &c) { return a.toleft(crosspoint(b, c)) == -1; };
    rep(i, L.size()) {
        while(dq.size() > 1 and check(L[i], *(end(dq) - 2), *(end(dq) - 1))) dq.pop_back();
        while(dq.size() > 1 and check(L[i], dq[0], dq[1])) dq.pop_front();
        // dump(L[i], si(dq));

        if(dq.size() and eq(cross(L[i].b - L[i].a, dq.back().b - dq.back().a), 0)) {
            if(dot(L[i].b - L[i].a, dq.back().b - dq.back().a) < eps) return {};
            if(L[i].toleft(dq.back().a) == -1)
                dq.pop_back();
            else
                continue;
        }
        dq.emplace_back(L[i]);
    }

    while(dq.size() > 2 and check(dq[0], *(end(dq) - 2), *(end(dq) - 1))) dq.pop_back();
    while(dq.size() > 2 and check(dq.back(), dq[0], dq[1])) dq.pop_front();
    if(si(dq) < 3) return {};
    Polygon ret(dq.size());
    rep(i, dq.size()) ret[i] = crosspoint(dq[i], dq[(i + 1) % dq.size()]);
    return ret;
}
} // namespace Geometry

using namespace Geometry;
```

Andrew

```cpp
const double eps = 1e-9, pi = acos(-1.0);
const int N = 1e5 + 10;

int n, cnt, m;

int sgn(double x) {
    if(fabs(x) < eps)   return 0;
    if(x > 0)   return 1;
    return -1;
}

struct point {
    double x, y;
    point(double a = 0.0, double b = 0.0) : x(a), y(b) {}
    bool operator < (point t) {
        if(sgn(x - t.x) == 0)  return y < t.y;
        return x < t.x;
    }
    point operator - (point p){
        return {x - p.x, y - p.y};
    }
    double operator ^ (point p){
        return x * p.y - y * p.x;
```

```
    }
}p[N], ans[N];

double dis(point a, point b) {
    a = a - b;
    return sqrt(a.x * a.x + a.y * a.y);
}

void Andrew() {
    sort(p, p + n);
    int p1 = 0, p2;
    for(int i = 0; i < n; i++) {
        while(p1 > 1 && sgn((ans[p1] - ans[p1 - 1]) ^ (p[i] - ans[p1 - 1])) <= 0)   p1--;
        ans[++p1] = p[i];
    }
    p2 = p1;
    for(int i = n - 2; i >= 0; i--) {
        while(p2 > p1 && sgn((ans[p2] - ans[p2 - 1]) ^ (p[i] - ans[p2 - 1])) <= 0)  p2--;
        ans[++p2] = p[i];
    }
    double target = 0.0;
    for(int i = 1; i < p2; i++){
        target += dis(ans[i], ans[i + 1]);
    }
    printf("%.2f\n", target);
}
/*
usage:
scanf("%d", &n);
for(int i = 0; i < n; i++)
    scanf("%lf%lf", &p[i].x, &p[i].y);
Andrew();
*/
```

CHT

```
// 维护上凸壳
struct Line {
    ll k, b;
    double intersect(Line l) {
        //交点 x 坐标
        double db = l.b - b;
        double dk = k - l.k;
        return db / dk;
    }

    ll calc (int x) {
        return k * x + b;
    }
};

struct CHT {
    vector<double> x; // 相邻线交点
    vector<Line> line; // 线

    void init(Line l) {
        x.push_back(-inf);
        line.push_back(l);
    }

    void addLine(Line l) {
        while (line.size() >= 2 && l.intersect(line[line.size() - 2]) <= x.back()) {
            x.pop_back();
            line.pop_back();
        }
        x.push_back(l.intersect(line.back()));
        line.push_back(l);
    }

    ll query(int qx) {
        int id = upper_bound(x.begin(), x.end(), qx) - x.begin() - 1; // 计算点属于的线 id
```

```cpp
        return line[id].calc(qx);
    }
};
```

字符串

KMP

```cpp
int nxt[N];
string a, b;
//a 为模式串  b 为匹配串

int kmp(int n, int m){
    int res = 0;
    nxt[0] = -1;
    for(int j = -1, i = 0; i < n;){
        if(j == -1 || a[j] == a[i]){
            i++;j++;
            nxt[i] = j;
        }else{
            j = nxt[j];
        }
    }
    //i 模式串  j 匹配串
    for(int i = 0, j = 0; j < m; ){
        if(i == -1 || a[i] == b[j]){
            i++;j++;
        }else i = nxt[i];
        if(i == n){
            res += 1;
            // position:: j - n + 1
            i = nxt[i];
        }
    }
    return res;
}
```

序列自动机

```cpp
构建:
for(int i = n; i >= 1; i--){
    for(int j = 0; j < 26; j++) ne[i - 1][j] = ne[i][j];
    ne[i - 1][s[i - 1] - 'a'] = i;
}

求三 (或多个) 个串的公共子序列个数:
int dfs(int p1, int p2, int p3){
    if(f[p1][p2][p3]) return f[p1][p2][p3];
    for(int i = 0; i < 26; i++){
        if(ne[0][p1][i] && ne[1][p2][i] && ne[2][p3][i]){
            f[p1][p2][p3] = (f[p1][p2][p3] + dfs(ne[0][p1][i], ne[1][p2][i], ne[2][p3][i])) % mod;
        }
    }
    f[p1][p2][p3] = (f[p1][p2][p3] + 1) % mod;
    return f[p1][p2][p3];
}
```

字符串双哈希

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

using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;

const int N = 1e5 + 10;
const pii mod = {1e9 + 7, 1e9 + 9};
```

```cpp
const pii base = {131, 251};
pll pw[N];

pll operator*(const pll &p1, const pll &p2) {
    return {p1.first * p2.first % mod.first, p1.second * p2.second % mod.second};
}

pll operator+(const pll &p1, const pll &p2) {
    return {(p1.first + p2.first) % mod.first, (p1.second + p2.second) % mod.second};
}

pll operator-(const pll &p1, const pll &p2) {
    return {(p1.first - p2.first + mod.first) % mod.first, (p1.second - p2.second + mod.second) % mod.second};
}

struct Hash {
    vector<pll> f;
    int n{};

    void init(ll ss[], int _n) {
        n = _n;
        f.resize(n + 1, {0, 0});
        for (int i = 1; i <= n; i++) {
            ll ch = ss[i];
            f[i] = f[i - 1] * base + pll{ch, ch};
        }
    }

    pll ask(int l, int r) {//[l, r]
        return f[r] - f[l - 1] * pw[r - l + 1];
    }
};
//记得初始化 pw
//pw[0] = {1, 1};
//for (int i = 1; i <= n; i++) pw[i] = pw[i - 1] * base;
```

马拉车

```cpp
struct Manacher {
    int f[N], m;
    char s[N];

    void init(const std::string& a, int n){
        // a: 1-index
        for(int i = 1; i <= n; i++){
            s[i << 1] = a[i];
            s[i << 1 | 1] = '#';
            f[i << 1] = f[i << 1 | 1] = 0;
        }
        f[0] = f[1] = 0;
        int r, p, i;
        s[0] = '$', s[1] = '#', s[m = (n + 1) << 1] = '@';
        for(r = p = 0, f[1] = 1, i = 2; i < m; i += 1){
            f[i] = r > i ? std::min(r - i, f[p * 2 - i]) : 1;
            for(; s[i - f[i]] == s[i + f[i]]; f[i]++);
            if(i + f[i] > r) r = i + f[i], p = i;
        }
    }

    // i in [2, m - 1]
    std::pair<int, int> query(int i) {
        // the longest palindrome centered on i
        return {(i - f[i] + 2) / 2, (i + f[i] - 1) / 2};
    }
}manacher;
```

杂项

莫队

时间复杂度 $O(\frac{n^2}{S}+mS)$, n 为长度, m 个询问, 块长为 S (一般取 $\sqrt{n}$ 或 $\frac{n}{\sqrt{m}}$)

```cpp
int unit;
int a[N];

struct node {
    int l, r, id;

    bool operator < (const node &k) const {
        if (l / unit != k.l / unit) return l / unit < k.l / unit;
        return r < k.r;
    }
} q[N];
void add(int i) {

}

void sub(int i) {

}
void solve(){
    unit = (int)sqrt(m);// m 个区间
    sort(q + 1, q + 1 + m);
    int L = 1, R = 0;
    for (int i = 1; i <= m; i++) {
        while (R < q[i].r) {
            R++;
            add(R);
        }
        while (R > q[i].r) {
            sub(R);
            R--;
        }
        while (L > q[i].l) {
            L--;
            add(L);
        }
        while (L < q[i].l) {
            sub(L);
            L++;
        }
    }
}
```

unordered_map

```cpp
struct HashFunc{
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    template<typename T, typename U>
    size_t operator()(const std::pair<T, U>& p) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(p.first + FIXED_RANDOM) ^ splitmix64(p.second + FIXED_RANDOM);
    }
};

// 键值比较，哈希碰撞的比较定义，需要直到两个自定义对象是否相等
struct EqualKey {
    template<typename T, typename U>
    bool operator ()(const std::pair<T, U>& p1, const std::pair<T, U>& p2) const {
        return p1.first == p2.first && p1.second == p2.second;
```

```
    }
};
unordered_map<pii, int, HashFunc, EqualKey> mp;
```

DSU

```
struct DSU{
    int f[N];
    void init(int n){
        for(int i = 0; i <= n; i++) f[i] = -1;
    }
    int find(int x){
        return f[x] < 0 ? x : f[x] = find(f[x]);
    }
    void merge(int x, int y){
        int fx = find(x), fy = find(y);
        if(fx == fy) return;
        if(f[fx] > f[fy]) swap(fx, fy);
        f[fx] += f[fy];
        f[fy] = fx;
    }
}dsu;
```

Floyd 判圈

```
//适用于每个点出度唯一的图找环
const ll mod = 1099511627776;

ll calc(ll x){
    return (x + (x >> 20) + 12345) % mod;
}

void Floyd_Cycle_Detection_Algorithm(){
    ll p1 = 1611516670, p2 = 1611516670; // 起始点
    do{
        p1 = calc(p1); // 移动一次
        p2 = calc(calc(p2)); // 移动两次
    }while(p1 != p2);
    // 存在环
    ll len = 0;// 环长
    do{
        p2 = calc(p2);
        len++;
    }while(p1 != p2);
    p1 = 1611516670;// 寻找环起点
    ll c1 = 0; // 起点到环起点的距离
    while(p1 != p2){
        p1 = calc(p1);
        p2 = calc(p2);
        c1++;
    }
    cout << p1 << ' ' << len << ' ' << c1 << '\n';
}
```

三分搜索

```
auto ternary_search = [&](ld l, ld r) {
    int it = 300;              //set the error limit here
    while (it--) {
        ld m1 = l + (r - l) / 3;
        ld m2 = r - (r - l) / 3;
        ld f1 = f(m1);      //evaluates the function at m1
        ld f2 = f(m2);      //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return l;                    //return the maximum of f(x) in [l, r]
};
```

```cpp
auto ternary_search = [&](int l, int r) {
    int it = 300;              //set the error limit here
    while (it--) {
        int m1 = (l + r) >> 1;
        int m2 = m1 + 1;
        ld f1 = f(m1);      //evaluates the function at m1
        ld f2 = f(m2);      //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return l;                   //return the maximum of f(x) in [l, r]
};
```

随机器

```cpp
// random shuffle
random_device rd;
mt19937 rng(rd());
shuffle(a + 1, a + 1 + n, rng);
```

```cpp
// 区间随机
mt19937 mt(chrono::steady_clock::now().time_since_epoch().count());

int rng(int l, int r) {
    uniform_int_distribution<int> uni(l, r);
    return uni(mt);
}
```

启发式合并 (DSU ON TREE)

```cpp
/*
题意翻译：树的节点有颜色，一种颜色占领了一个子树，
当且仅当没有其他颜色在这个子树中出现得比它多。
求占领每个子树的所有颜色之和。
*/
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair<int, int> pii;
const int N = 1e5 + 10, Log = 20, inf = 0x3f3f3f3f;

int sz[N], lt[N], rt[N], son[N], tot;
vector<int> e[N];
int cnt[N], a[N], rk[N];
ll sum, ans[N], mx;

void dfs(int u, int fa){
    sz[u] = 1;
    lt[u] = ++tot;
    rk[tot] = u;
    for(int i : e[u]){
        if(i == fa) continue;
        dfs(i, u);
        if(!son[u] || sz[i] > sz[son[u]]){
            son[u] = i;
        }
        sz[u] += sz[i];
    }
    rt[u] = tot;
}

// need detailed
void add(int u, int x){
    cnt[u] += x;
    if(cnt[u] > mx){
        mx = cnt[u];
```

```cpp
        sum = u;
    }else if(cnt[u] == mx){
        sum += u;
    }
}

void gao(int u, int fa, bool f){
    for(int v : e[u]){
        if(v == fa || v == son[u]) continue;
        gao(v, u, true);
    }
    if(son[u]) gao(son[u], u, false);
    for(int v : e[u]){
        if(v == fa || v == son[u]) continue;
        for(int i = lt[v]; i <= rt[v]; i++){
            add(a[rk[i]], 1);
        }
    }
    {// need detailed
        add(a[u], 1);
        ans[u] = sum;
    }
    if(f){// need detailed
        for(int i = lt[u]; i <= rt[u]; i++){
            add(a[rk[i]], -1);
        }
        sum = mx = 0;
    }
}

void solve() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++){
        cin >> a[i];
    }
    for(int i = 0, u, v; i + 1 < n; i++){
        cin >> u >> v;
        e[u].push_back(v);
        e[v].push_back(u);
    }
    dfs(1, 0);
    gao(1, 0, true);
    for(int i = 1; i <= n; i++){
        cout << ans[i] << " \n"[i == n];
    }
}

int main() {
    int T = 1;
    ios::sync_with_stdio(false);
    // cin >> T;
    while (T--) solve();
    return 0;
}
```