Standard Code Library

A cup of latte

ZUCC

August 16, 2022

Contents

一切的开始

快读

```
#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

```
#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

```
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)
*/
```

数学

组合数预处理

```
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 $x = x_0 + tb, y = y_0 - ta$

```
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

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

```
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

```
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: $phi(n) = n \prod (1 - \frac{1}{p_i})$

```
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

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

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

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

矩阵快速幂

```
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

$x = num_i (mod\ r_i)$

```
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);
}
```

差分推 x+y 组合数方案

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

typedef long long ll;
using namespace std;
const int maxn = 2000005;

/*
A <= x <= B
C <= y <= D
s[i] 表示 x+y=i 的方案数
*/

int s[maxn];

int main() {

    int A, B, C, D;
    A = ; B = ;
    C = ; D = ;

    s[A + C]++;
    s[A + D + 1]--;
    s[B + C + 1]--;
    s[B + D + 2]++;

    for (int i = 1; i < maxn; i++) s[i] += s[i - 1];
    for (int i = 1; i < maxn; i++) s[i] += s[i - 1];

    for (int i = A + C; i <= B + D + 2; i++) {
        printf("s[%d] = %d\n", i, s[i]);
    }

    return 0;
}
```

拉格朗日插值

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

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

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

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

FFT

```
const double PI = acos(-1.0);

struct Complex {
    double x, y;

    Complex(double _x = 0.0, double _y = 0.0) {
        x = _x;
```

```
        y = _y;
    }

    Complex operator-(const Complex &b) const {
        return {x - b.x, y - b.y};
    }

    Complex operator+(const Complex &b) const {
        return {x + b.x, y + b.y};
    }

    Complex operator*(const Complex &b) const {
        return {x * b.x - y * b.y, x * b.y + y * b.x};
    }
};

/*
 * 进行 FFT 和 IFFT 前的反置变换
 * 位置 i 和 i 的二进制反转后的位置互换
 *len 必须为 2 的幂
 */
void change(Complex y[], int len) {
    int i, j, k;

    for (i = 1, j = len / 2; i < len - 1; i++) {
        if (i < j) swap(y[i], y[j]);

        // 交换互为小标反转的元素, i<j 保证交换一次
        // i 做正常的 + 1, j 做反转类型的 + 1, 始终保持 i 和 j 是反转的
        k = len / 2;

        while (j >= k) {
            j = j - k;
            k = k / 2;
        }

        if (j < k) j += k;
    }
}

/*
 * 做 FFT
 * len 必须是 2^k 形式
 * on == 1 时是 DFT, on == -1 时是 IDFT
 * DFT: 系数 -> 点值表示  IDFT: 点值表示 -> 系数
 */
void fft(Complex y[], int len, int on) {
    change(y, len);

    for (int h = 2; h <= len; h <<= 1) {
        Complex wn(cos(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;
        }
    }
}
```

图论

最小生成树

Prim

```cpp
typedef pair<int, int> pii;
const int N = 400009, inf = 0x3f3f3f3f;
int n, m;
struct edge {
    int to, w, next;
} e[N];

struct Prim {
    int head[N], idx;
    int dist[N];
    bool vis[N];

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

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

    int prim(int x) {
        int cnt = 0, sum = 0;//cnt 为加点数 sum 为总边权和
        for (int i = 1; i <= n; i++) dist[i] = inf, vis[i] = false;
        dist[x] = 0;
        priority_queue<pii, vector<pii>, greater<pii>> q;
        q.push({dist[x], x});
        while (!q.empty() && cnt < n) {
            int t = q.top().second;
            int dis = q.top().first;
            q.pop();
            if (vis[t]) continue;
            cnt++;
            sum += dis;
            vis[t] = true;
            for (int i = head[t]; i != -1; i = e[i].next) {
                int tar = e[i].to;
                if (dist[tar] > e[i].w) {
                    dist[tar] = e[i].w;
                    q.push({dist[tar], tar});
                }
            }
        }
        if (cnt == n) return sum;
        return -1;//非联通
    }
} prim;
```

Kruskal

```cpp
typedef pair<int, int> pii;
const int N = 400009, inf = 0x3f3f3f3f;
int n, m;

struct Edge {
    int x, y, w;

    bool operator<(const Edge &k) const {
        return w < k.w;
    }
} edge[N];

```

```cpp
int f[N];

int find(int x) {
    int r = x;
    while (x != f[x]) x = f[x];
    while (r != x) {
        int j = f[r];
        f[r] = x;
        r = j;
    }
    return x;
}

int Kruskal() {
    for (int i = 1; i <= n; i++) f[i] = i;
    int cnt = 0, sum = 0;
    sort(edge + 1, edge + 1 + m);
    for (int i = 1; i <= m; i++) {
        int x = find(edge[i].x), y = find(edge[i].y);
        if (x != y) {
            f[x] = y;
            cnt++;
            sum += edge[i].w;
        }
    }
    if (cnt == n - 1) return sum;
    return -1;
}
```

Dijkstra

```cpp
typedef pair<int, int> pii;
const int N = 100009, inf = 0x3f3f3f3f;
int n, m;
struct edge {
    int to, w, next;
} e[N];

struct dijkstra {
    int head[N], idx;
    int dist[N];
    bool vis[N];

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

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

    void dij(int x) {
        for (int i = 1; i <= n; i++) dist[i] = inf, vis[i] = false;
        dist[x] = 0;
        priority_queue<pii, vector<pii >, greater<pii>> q;
        q.push({dist[x], x});
        while (!q.empty()) {
            int t = q.top().second;
            int dis = q.top().first;
            q.pop();
            if (vis[t]) continue;
            vis[t] = true;
            for (int i = head[t]; i != -1; i = e[i].next) {
                int tar = e[i].to;
                if (dist[tar] > e[i].w + dis) {
                    dist[tar] = e[i].w + dis;
                    q.push({dist[tar], tar});
```

```
                }
            }
        }
    }
} dij;
```

LCA

倍增求法

```
const int inf = 0x3f3f3f3f, N = 100010, Log = 20;
int anc[N][Log + 5], depth[N];
vector<int> e[N];

void dfs(int k, int fa) {
    anc[k][0] = fa;
    depth[k] = depth[fa] + 1;
    for (int i = 0; i < e[k].size(); i++) {
        int to = e[k][i];
        if (to != fa) {
            dfs(to, k);
        }
    }
}

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 k, int h) {//从节点 k 往上找 h 个祖先
    for (int j = 1, i = 0; j <= h; j <<= 1, i++) {
        if (j & h) k = anc[k][i];
    }
    return k;
}

int query(int x, int y) {//询问 x 和 y 的最小公共祖先
    if (depth[x] < depth[y]) swap(x, y);
    x = rush(x, depth[x] - depth[y]);//调整为相同深度
    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

```
const int N = 100010, Log = 30;
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, deep[N];//id 为欧拉序
vector<int> f[N];

void dfs(int k, int fa, int d) {
    id[++tot] = k; //id[] 存储欧拉序所对应的树的节点编号
    deep[tot] = d; //deep[] 存储每个 dfs 遍历序列号的深度
    first[k] = tot; //first[x]=y 表示树的第 x 号节点在 dfs 遍历序列第一次出现的位置 y
    for (int i = 0; i < f[k].size(); i++) {
        int u = f[k][i];
        if (u != fa) {
            dfs(u, k, d + 1);
```

```
            id[++tot] = k;
            deep[tot] = d;
        }
    }
}

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

void init(int n) {//更新 st 表和 logx
    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] = Min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
        }
    }
}

int LCA(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

```
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);
        }
    }
}
```

求强连通分量 (scc)

2-sat 问题对于一对互斥关系 (a, b) 将 a 与!b 连边, b 与!a 连边, 跑 scc 即可

```
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;
        }
    }
    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;
}
```

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

```
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;
}

```

```cpp
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)

```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, ny;

//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, ny = 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

```
const int inf = 0x3f3f3f3f, N = 20000, M = 2e5 + 10;
struct edge {
    int to, next;
    ll w;//w 为流量
} e[M];
int head[N], idx, cur[N];
int dist[N], s, t, n;
bool vis[N];

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

```

```cpp
void _add(int a, int b, ll c) {
    e[idx] = {b, head[a], c};
    head[a] = idx++;
}

void add(int a, int b, ll c){
    _add(a, b, c);
    _add(b, a, 0);
}

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

ll dfs(int x, ll flow) {
    if (x == t || !flow) return flow;
    ll delta = 0, f;
    for (int i = cur[x]; i != -1; i = e[i].next) {
        int to = e[i].to;
        ll w = e[i].w;
        cur[x] = i;
        if (dist[to] == dist[x] + 1 && (f = dfs(to, min(flow, w))) > 0) {
            e[i].w -= f;
            e[i ^ 1].w += f;
            flow -= f;
            delta += f;
            if (flow == 0) break;
        }
    }
    return delta;
}

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

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++;
}

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 $e \times log(n) \times f$)

```
typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, int> pll;
const int N = 1e6 + 10, M = 2e7;
const ll inf = 1e10;
```

```
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], h[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++;
}

bool dij() {
    for (int i = 1; i <= n; i++) {
        vis[i] = false;
        dist[i] = inf;
        flow[i] = inf;
    }
    dist[s] = 0;
    pre[t] = -1;
    priority_queue<pll, vector<pll>, greater<pll>> q;
    q.push({dist[s], s});
    while(!q.empty()){
        int x = q.top().second;
        q.pop();
        if(vis[x]) continue;
        vis[x] = true;
        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] + h[x] - h[to] + fee) {
                dist[to] = dist[x] + h[x] - h[to] + fee;
                flow[to] = min(flow[x], w);
                pre[to] = x;
                id[to] = i;
                q.push({dist[to], to});
            }
        }
    }
    return dist[t] != inf;
}

ll MinFee() {
    ll minfee = 0, maxflow = 0;
    while (dij()) {
        for(int i = 1; i <= n; i++) {
            h[i] += dist[i];
        }
        int now = t;
        maxflow += flow[t];
        minfee += flow[t] * h[t];
        while (now != s) {
            e[id[now]].w -= flow[t];
            e[id[now] ^ 1].w += flow[t];
            now = pre[now];
        }
    }
    return minfee;
}
```

```
from oiwiki
```

```cpp

#include <algorithm>
#include <cstdio>
#include <cstring>
#include <queue>
#define INF 0x3f3f3f3f
using namespace std;

struct edge {
  int v, f, c, next;
} e[100005];

struct node {
  int v, e;
} p[10005];

struct mypair {
  int dis, id;

  bool operator<(const mypair& a) const { return dis > a.dis; }

  mypair(int d, int x) { dis = d, id = x; }
};

int head[5005], dis[5005], vis[5005], h[5005];
int n, m, s, t, cnt = 1, maxf, minc;

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

bool dijkstra() {
  priority_queue<mypair> q;
  for (int i = 1; i <= n; i++) dis[i] = INF;
  memset(vis, 0, sizeof(vis));
  dis[s] = 0;
  q.push(mypair(0, s));
  while (!q.empty()) {
    int u = q.top().id;
    q.pop();
    if (vis[u]) continue;
    vis[u] = 1;
    for (int i = head[u]; i; i = e[i].next) {
      int v = e[i].v, nc = e[i].c + h[u] - h[v];
      if (e[i].f && dis[v] > dis[u] + nc) {
        dis[v] = dis[u] + nc;
        p[v].v = u;
        p[v].e = i;
        if (!vis[v]) q.push(mypair(dis[v], v));
      }
    }
  }
  return dis[t] != INF;
}

void spfa() {
  queue<int> q;
  memset(h, 63, sizeof(h));
  h[s] = 0, vis[s] = 1;
  q.push(s);
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    vis[u] = 0;
    for (int i = head[u]; i; i = e[i].next) {
      int v = e[i].v;
      if (e[i].f && h[v] > h[u] + e[i].c) {
```

```
          h[v] = h[u] + e[i].c;
          if (!vis[v]) {
            vis[v] = 1;
            q.push(v);
          }
        }
      }
    }
}

int main() {
  scanf("%d%d%d%d", &n, &m, &s, &t);
  for (int i = 1; i <= m; i++) {
    int u, v, f, c;
    scanf("%d%d%d%d", &u, &v, &f, &c);
    addedge(u, v, f, c);
    addedge(v, u, 0, -c);
  }
  spfa();  // 先求出初始势能
  while (dijkstra()) {
    int minf = INF;
    for (int i = 1; i <= n; i++) h[i] += dis[i];
    for (int i = t; i != s; i = p[i].v) minf = min(minf, e[p[i].e].f);
    for (int i = t; i != s; i = p[i].v) {
      e[p[i].e].f -= minf;
      e[p[i].e ^ 1].f += minf;
    }
    maxf += minf;
    minc += minf * h[t];
  }
  printf("%d %d\n", maxf, minc);
  return 0;
}
```

计算几何

二维几何：点与向量

```
#define y1 yy1
#define nxt(i) ((i + 1) % s.size())
typedef double LD;
const LD PI = 3.14159265358979323846;
const LD eps = 1E-10;
int sgn(LD x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }
struct L;
struct P;
typedef P V;
struct P {
    LD x, y;
    explicit P(LD x = 0, LD y = 0): x(x), y(y) {}
    explicit P(const L& l);
};
struct L {
    P s, t;
    L() {}
    L(P s, P t): s(s), t(t) {}
};

P operator + (const P& a, const P& b) { return P(a.x + b.x, a.y + b.y); }
P operator - (const P& a, const P& b) { return P(a.x - b.x, a.y - b.y); }
P operator * (const P& a, LD k) { return P(a.x * k, a.y * k); }
P operator / (const P& a, LD k) { return P(a.x / k, a.y / k); }
inline bool operator < (const P& a, const P& b) {
    return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0);
}
bool operator == (const P& a, const P& b) { return !sgn(a.x - b.x) && !sgn(a.y - b.y); }
P::P(const L& l) { *this = l.t - l.s; }
ostream &operator << (ostream &os, const P &p) {
    return (os << "(" << p.x << "," << p.y << ")");
}
```

```
istream &operator >> (istream &is, P &p) {
    return (is >> p.x >> p.y);
}

LD dist(const P& p) { return sqrt(p.x * p.x + p.y * p.y); }
LD dot(const V& a, const V& b) { return a.x * b.x + a.y * b.y; }
LD det(const V& a, const V& b) { return a.x * b.y - a.y * b.x; }
LD cross(const P& s, const P& t, const P& o = P()) { return det(s - o, t - o); }
// --------------------------------------------
```

字符串

KMP

```
const int N = 100010;
int ne[N];
string s, t;//s 为原串 t 为匹配串
void getNext() {//求 next 数组 ne[i] 表示长度为 i 的最长公共前后缀长度 (ne[i]<i)
    ne[0] = -1;
    int k = -1, j = 0;
    while (j < t.size()) {
        if (k == -1 || t[k] == t[j]) {
            j++;
            k++;
            ne[j] = k;
        } else k = ne[k];
    }
}

int kmp() {//返回匹配下标
    int i = 0, j = 0;
    int n = (int)s.size(), m = (int)t.size();
    while (i < n && j < m) {
        if (j == -1 || s[i] == t[j]) {
            i++;
            j++;
        } else {
            j = ne[j];
        }
    }
    if (j == m) return i - m;
    return -1;
}
```

序列自动机

```
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];
}
```

字典树

```
//对数排序 查找排序后第 k 个数 每个数 <= 1e9
const int N = 5000010;// 总长度
```

```
int trie[N][10], tot, sum[N][10], ssum = 0;
int color[N];

void insert(string s) {//插入
    int p = 0;
    ssum++;
    for (char i: s) {
        int c = i - '0';
        sum[p][c]++;
        if (!trie[p][c]) trie[p][c] = ++tot;
        p = trie[p][c];
    }
    color[p]++;
}

void insert(int x) {//x 转换为字符
    string s;
    while (x) {
        char t = x % 10 + '0';
        s += t;
        x /= 10;
    }
    while (s.size() <= 10) s += '0';
    reverse(s.begin(), s.end());
    insert(s);
}

int find(int k) {//查找第 k 个数
    int res = 0, p = 0;
    while (k > 0) {
        for (int i = 0; i < 10; i++) {
            if (sum[p][i] < k) k -= sum[p][i];
            else {
                res = res * 10 + i;
                p = trie[p][i];
                k -= color[p];
                break;
            }
        }
    }
    return res;
}
```

字符串双哈希

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

using namespace std;
typedef long long ll;
const int N = 1e5 + 10;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
const pii mod = {1e9 + 7, 1e9 + 9};

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

杂项

莫队

```
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 重写哈希函数
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;
```