目录

**2SAT1**

**BGSG1**

**高斯消元2**

**求二次剩余2**

**线性基3**

**一般图最大匹配3**

**主席树5**

2SAT

//当我们想知道A选不选，只需要比较A和!A所属联通块的编号就OK了

//选择那个编号更大，即拓扑序更靠后的决策

int n, m;

cin >> n >> m;

graph<int> g(2 \* n);

for(int i = 0; i < m; i++) {

int u, v;

cin >> u >> v;

--u, --v;

g.add(u, cus(v));

g.add(v, cus(u));

}

auto v = find\_scc(g);

for(int i = 0; i < 2 \* n; i += 2) {

if(v[i] == v[i + 1]) {

cout << "NIE" << '\n'; return ;

}

}

for(int i = 0; i < 2 \* n; i += 2) {

if(v[i] > v[i + 1]) {

cout << i + 1 << '\n';

}

else {

cout << i + 2 << '\n';

}

}

BGSG

//给定 a,p,b求满足 a^x≡b mod p 的最小自然数 x ,1<= a,p,b≤10^9

const int INF = 0x3f3f3f3f;

int a, b, p;

unordered\_map<int, int> hs;

int exgcd(int a, int b, int &x, int &y) {

if (!b) {

x = 1, y = 0;

return a;

}

int d = exgcd(b, a % b, y, x);

y -= a / b \* x;

return d;

}

int BSGS(int a, int b, int p) {

if (1 % p == b % p) return 0;

int k = sqrt(p) + 1;

hs.clear();

for (int y = 0, r = b % p; y < k; y++) {

hs[r] = y;

r = (LL)r \* a % p;

}

int ak = 1;

for (int i = 1; i <= k; i++) ak = (LL)ak \* a % p;

for (int x = 1, l = ak; x <= k; x++) {

if (hs.count(l)) return k \* x - hs[l];

l = (LL)l \* ak % p;

}

return -INF;

}

int exBSGS(int a, int b, int p) {

b = (b % p + p) % p;

if (1 % p == b % p) return 0;

int x, y;

int d = exgcd(a, p, x, y);

if (d > 1) {

if (b % d) return -INF;

exgcd(a / d, p / d, x, y);

return exBSGS(a, (LL)b / d \* x % (p / d), p / d) + 1;

}

return BSGS(a, b, p);

}

void solve() {

while (~scanf("%d%d%d", &a, &p, &b), a || b || p) {

int res = exBSGS(a, b, p);

if (res < 0) puts("No Solution");

else printf("%d\n", res);

}

}

高斯消元

const double eps = 1e-5;

int n; //可以跑 n=668的情况

double a[105][105];

void Gauss() {

for (int i = 1; i <= n; ++i) {

int p = i;

for (int j = i + 1; j <= n; ++j)

if (fabs(a[j][i]) > fabs(a[j][p]))

p = j;

if (fabs(a[i][i]) < eps) {

cout << "No Solution" << '\n';;

return ;

}

for (int j = 1; j <= n + 1; ++j)

swap(a[i][j], a[p][j]);

for (int j = 1; j <= n; ++j) {

if (i == j) continue;

double t = a[j][i] / a[i][i];

for (int k = i; k <= n + 1; ++k)

a[j][k] -= a[i][k] \* t;

}

}

for(int i = 1; i <= n; i++) {

a[i][n + 1] /= a[i][i];

}

for(int i = 1; i <= n; i++) {

cout << fixed << setprecision(2) << a[i][n + 1] << '\n';

}

}

void solve() {

cin>> n;

for (int i = 1; i <= n; ++i)

for (int j = 1; j <= n + 1; ++j)

cin >> a[i][j];

Gauss();

求二次剩余

using ll = long long int;

#define int long long

ll qpow(ll x, ll n, ll mod) {

ll res = 1;

for (x %= mod; n > 0; n >>= 1, x = x \* x % mod) {

if (n & 1LL) res = res \* x % mod;

}

return (res + mod) % mod;

}

// Tonelli-Shanks

ll ModSqrt(ll a, ll p) {

if (a == 0 || p <= 2) return a;

if (qpow(a, (p - 1) / 2, p) != 1) return -1;

if (p % 4 == 3) return qpow(a, (p + 1) / 4, p);

ll k = \_\_builtin\_ctz(p - 1), h = (p - 1) >> k;

ll N = 2; while (qpow(N, (p - 1) / 2, p) == 1) N += 1;

ll x = qpow(a, (h + 1) / 2, p), g = qpow(N, h, p), b = qpow(a, h, p);

for (ll m = 0;; k = m) {

ll t = b;

for (m = 0; m < k && t != 1; m++) t = t \* t % p;

if (m == 0) return x;

ll gs = qpow(g, 1 << (k - m - 1), p);

g = gs \* gs % p, b = b \* g % p, x = x \* gs % p;

}

return -1;//复杂度粗略估计为log^2(p)

}

线性基

//给定 n 个整数（数字可能重复），求在这些数中选取任意个，使得他们的异或和最大

typedef long long ll;

const ll N = 55, bit = 50;

ll n, a, p[N];

void insert(ll k) {

for(ll i=bit;i>=0;i--) {

if(!(k&(1LL<<i))) continue;

if(!p[i]) return p[i] = k, void();

k ^= p[i];

}

}

ll maxXor() {

ll res = 0;

for(ll i=bit;i>=0;i--) res = max(res, (res^p[i]));

return res;

}

void solve() {

scanf("%lld", &n);

while(n--) {scanf("%lld", &a); insert(a);}

printf("%lld\n", maxXor());

}

一般图最大匹配

template <typename T>

class graph {

public:

struct edge {

int from;

int to;

T cost;

};

vector<edge> edges;

vector<vector<int> > g;

int n;

graph(int \_n) : n(\_n) { g.resize(n); }

virtual int add(int from, int to, T cost) = 0;

};

// undirectedgraph

template <typename T>

class undirectedgraph : public graph<T> {

public:

using graph<T>::edges;

using graph<T>::g;

using graph<T>::n;

undirectedgraph(int \_n) : graph<T>(\_n) {}

int add(int from, int to, T cost = 1) {

assert(0 <= from && from < n && 0 <= to && to < n);

int id = (int)edges.size();

g[from].push\_back(id);

g[to].push\_back(id);

edges.push\_back({from, to, cost});

return id;

}

};

// blossom / find\_max\_unweighted\_matching

template <typename T>

vector<int> find\_max\_unweighted\_matching(const undirectedgraph<T> &g) {

std::mt19937 rng(chrono::steady\_clock::now().time\_since\_epoch().count());

vector<int> match(g.n, -1); // 匹配

vector<int> aux(g.n, -1); // 时间戳记

vector<int> label(g.n); // "o" or "i"

vector<int> orig(g.n); // 花根

vector<int> parent(g.n, -1); // 父节点

queue<int> q;

int aux\_time = -1;

auto lca = [&](int v, int u) {

aux\_time++;

while (true) {

if (v != -1) {

if (aux[v] == aux\_time) { // 找到拜访过的点 也就是LCA

return v;

}

aux[v] = aux\_time;

if (match[v] == -1) {

v = -1;

} else {

v = orig[parent[match[v]]]; // 以匹配点的父节点继续寻找

}

}

swap(v, u);

}

}; // lca

auto blossom = [&](int v, int u, int a) {

while (orig[v] != a) {

parent[v] = u;

u = match[v];

if (label[u] == 1) { // 初始点设为"o" 找增广路

label[u] = 0;

q.push(u);

}

orig[v] = orig[u] = a; // 缩花

v = parent[u];

}

}; // blossom

auto augment = [&](int v) {

while (v != -1) {

int pv = parent[v];

int next\_v = match[pv];

match[v] = pv;

match[pv] = v;

v = next\_v;

}

}; // augment

auto bfs = [&](int root) {

fill(label.begin(), label.end(), -1);

iota(orig.begin(), orig.end(), 0);

while (!q.empty()) {

q.pop();

}

q.push(root);

// 初始点设为 "o", 这里以"0"代替"o", "1"代替"i"

label[root] = 0;

while (!q.empty()) {

int v = q.front();

q.pop();

for (int id : g.g[v]) {

auto &e = g.edges[id];

int u = e.from ^ e.to ^ v;

if (label[u] == -1) { // 找到未拜访点

label[u] = 1; // 标记 "i"

parent[u] = v;

if (match[u] == -1) { // 找到未匹配点

augment(u); // 寻找增广路径

return true;

}

// 找到已匹配点 将与她匹配的点丢入queue 延伸交错树

label[match[u]] = 0;

q.push(match[u]);

continue;

} else if (label[u] == 0 && orig[v] != orig[u]) {

// 找到已拜访点 且标记同为"o" 代表找到"花"

int a = lca(orig[v], orig[u]);

// 找LCA 然后缩花

blossom(u, v, a);

blossom(v, u, a);

}

}

}

return false;

}; // bfs

auto greedy = [&]() {

vector<int> order(g.n);

// 随机打乱 order

iota(order.begin(), order.end(), 0);

shuffle(order.begin(), order.end(), rng);

// 将可以匹配的点匹配

for (int i : order) {

if (match[i] == -1) {

for (auto id : g.g[i]) {

auto &e = g.edges[id];

int to = e.from ^ e.to ^ i;

if (match[to] == -1) {

match[i] = to;

match[to] = i;

break;

}

}

}

}

}; // greedy

// 一开始先随机匹配

greedy();

// 对未匹配点找增广路

for (int i = 0; i < g.n; i++) {

if (match[i] == -1) {

bfs(i);

}

}

//match[i] 表示 第i个人和match[i]号匹配，若为-1则无匹配。

return match;

}

主席树

namespace Hjt {

#define ls lson[now\_]

#define rs rson[now\_]

const int MAXN = 1e5 + 5;

int root[MAXN];

int lson[MAXN << 5], rson[MAXN << 5], val[MAXN << 5], node\_num = 0;

void Insert(int &now\_, int pre\_, int l, int r, int pos, int val\_) {

now\_ = ++ node\_num;

val[now\_] = val[pre\_] + val\_;

ls = lson[pre\_], rs = rson[pre\_];

if(l == r) return ;

int mid = (l + r) >> 1;

if(pos <= mid) Insert(ls, lson[pre\_], l, mid, pos, val\_);

else Insert(rs, rson[pre\_], mid + 1, r, pos, val\_);

}

int Query(int now\_, int pre\_, int l, int r, int L, int R) {

if(!(val[now\_] - val[pre\_])) return 0;

if(L <= l && r <= R) return val[now\_] - val[pre\_];

int mid = (l + r) >> 1, res = 0;

if(mid >= L) res += Query(ls, lson[pre\_], l, mid, L, R);

if(mid < R) res += Query(rs, rson[pre\_], mid + 1, r, L, R);

return res;

}

}