**Dinic’s Algorithm**

struct FlowEdge {

int v, u;

long long cap, flow = 0;

FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}

};

struct Dinic {

const long long flow\_inf = 1e18;

vector<FlowEdge> edges;

vector<vector<int>> adj;

int n, m = 0;

int s, t;

vector<int> level, ptr;

queue<int> q;

Dinic(int n, int s, int t) : n(n), s(s), t(t) {

adj.resize(n);

level.resize(n);

ptr.resize(n);

}

void add\_edge(int v, int u, long long cap) {

edges.emplace\_back(v, u, cap);

edges.emplace\_back(u, v, 0);

adj[v].push\_back(m);

adj[u].push\_back(m + 1);

m += 2;

}

bool bfs() {

while (!q.empty()) {

int v = q.front();

q.pop();

for (int id : adj[v]) {

if (edges[id].cap - edges[id].flow < 1)

continue;

if (level[edges[id].u] != -1)

continue;

level[edges[id].u] = level[v] + 1;

q.push(edges[id].u);

}

}

return level[t] != -1;

}

long long dfs(int v, long long pushed) {

if (pushed == 0)

return 0;

if (v == t)

return pushed;

for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {

int id = adj[v][cid];

int u = edges[id].u;

if (level[v] + 1 != level[u] || edges[id].cap - edges[id].flow < 1)

continue;

long long tr = dfs(u, min(pushed, edges[id].cap - edges[id].flow));

if (tr == 0)

continue;

edges[id].flow += tr;

edges[id ^ 1].flow -= tr;

return tr;

}

return 0;

}

long long flow() {

long long f = 0;

while (true) {

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

level[s] = 0;

q.push(s);

if (!bfs())

break;

fill(ptr.begin(), ptr.end(), 0);

while (long long pushed = dfs(s, flow\_inf)) {

f += pushed;

}

}

return f;

}

};