struct Edge {

int from, to, cap, flow, index;

Edge(int from, int to, int cap, int flow, int index) :

from(from), to(to), cap(cap), flow(flow), index(index) {}

};

struct PushRelabel {

int N;

vector<vector<Edge> > G;

vector<ll> excess;

vector<int> dist, active, count;

queue<int> Q;

PushRelabel(int N) : N(N), G(N), excess(N), dist(N), active(N), count(2\*N) {}

void AddEdge(int from, int to, int cap){

G[from].push\_back(Edge(from, to, cap, 0, G[to].size()));

if(from == to) G[from].back().index++;

G[to].push\_back(Edge(to, from, 0, 0, G[from].size() - 1)); // for bidirectional set cap.

}

void Enqueue(int v){

if(!active[v] && excess[v] > 0) { active[v] = true; Q.push(v); }

}

void Push(Edge &e){

int amt = min(excess[e.from], ll(e.cap - e.flow));

if(dist[e.from] <= dist[e.to] || amt == 0) return;

e.flow += amt;

G[e.to][e.index].flow -= amt;

excess[e.to] += amt;

excess[e.from] -= amt;

Enqueue(e.to);

}

void Gap(int k){

fr(v, 0, N - 1){

if(dist[v] < k) continue;

count[dist[v]]--;

dist[v] = max(dist[v], N + 1);

count[dist[v]]++;

Enqueue(v);

}

}

void Relabel(int v){

count[dist[v]]--;

dist[v] = 2 \* N;

fr(i, 0, G[v].size() - 1)

if(G[v][i].cap - G[v][i].flow > 0)

dist[v] = min(dist[v], dist[G[v][i].to] + 1);

count[dist[v]]++;

Enqueue(v);

}

void Discharge(int v){

for(int i = 0; excess[v] > 0 && i < G[v].size(); i++) Push(G[v][i]);

if(excess[v] > 0){

if(count[dist[v]] == 1) Gap(dist[v]);

else Relabel(v);

}

}

ll GetMaxFlow(int s, int t) {

count[0] = N - 1;

count[N] = 1;

dist[s] = N;

active[s] = active[t] = 1;

fr(i, 0, (int)G[s].size() - 1){

excess[s] += G[s][i].cap;

Push(G[s][i]);

}

while(!Q.empty()){

int v = Q.front();

Q.pop();

active[v] = 0;

Discharge(v);

}

ll totflow = 0;

fr(i, 0, (int)G[s].size() - 1) totflow += G[s][i].flow;

return totflow;

}

};