Team Note of Baka Team

Shine - QioCas - Asamai

Compiled on December 9, 2024

Contents

1 Template

1.1 Template

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

using namespace std;
using ll = long long;

#define MASK(k) 1LL << (k)
#define BIT(x, k) ((x) >> (k) & 1)
#define all(x) (x).begin(), (x).end()

template<class T> bool minimize(T& a, const T& b) {
    if(a > b) return a = b, true;
    return false;
}

template<class T> bool maximize(T& a, const T& b) {
    if(a < b) return a = b, true;
    return false;
}

ll fdiv(ll a, ll b) {
    assert(b != 0);
    if(b < 0) a *= -1, b *= -1;
    return a >= 0 ? a / b : (a + 1) / b - 1;
}

ll cdiv(ll a, ll b) {
    assert(b != 0);
    if(b < 0) a *= -1, b *= -1;
    return a <= 0 ? a / b : (a - 1) / b + 1;
}
```

1.2 Debug

```cpp
void debug_utils() {}

template<class T, class ...U> void debug_utils(T a, U... b) {
    cerr << a;
    if(sizeof...(b)) { cerr << ", "; debug_utils(b...);}
}

#define debug(...) { cerr << #__VA_ARGS__ << " = ";
debug_utils(__VA_ARGS__); cerr << "\n"; }

template<class Tp1, class Tp2>
ostream& operator << (ostream& cout, pair<Tp1, Tp2> val) {
    return cout << val.first << " " << val.second << "\n";
}

template<class Data, class Tp =
decltype(declval<Data>().begin())>
typename enable_if<!is_same<Data, string>::value,
ostream&>::type
operator << (ostream& cout, Data val) {
    cout << "[";
    for(auto i = val.begin(); i != val.end(); ++i)
        cout << (i == val.begin() ? "" : " ") << *i;
    return cout << "]";
}

template<class Data, class = decltype(declval<Data>().top())>
ostream& operator << (ostream& cout, Data val) {
    cout << "[";
    for(; val.size(); val.pop())
        cout << val.top() << (val.size() == 1 ? "" : " ");
    return cout << "]";
}

template<class Tp> ostream& operator << (ostream& cout,
queue<Tp> val) {
    cout << "[";
    for(; val.size(); val.pop())
        cout << val.front() << (val.size() == 1 ? "" : " ");
    return cout << "]";
}
```

1.3 Generate

```cpp
mt19937_64
rng(chrono::steady_clock::now().time_since_epoch().count());

ll randint(ll a, ll b) {
    return uniform_int_distribution<ll> (a, b) (rng);
}
```

2 Data Structure

2.1 Mutidimensional Vector

```
template<class Tp, int D = 1>
struct Tvector : public vector<Tvector<Tp, D - 1>> {
    template <class... Args>
    Tvector(int n = 0, Args... args) : vector<Tvector<Tp, D -
    1>>(n, Tvector<Tp, D - 1>(args...)) {}
};

template <class Tp>
struct Tvector<Tp, 1> : public vector<Tp> {
    Tvector(int n = 0, Tp val = Tp()) : vector<Tp>(n, val) {}
};
```

2.2 Rollback

```
vector<pair<int*, int>> event;

void assign(int* u, int v) {
    event.push_back({u, exchange(*u, v)});
}

void rollback(int t) {
    for(; (int) event.size() > t; event.pop_back()) {
        *event.back().first = event.back().second;
    }
}
```

2.3 Wavelet Tree

```
struct wavelet {
    wavelet *left, *right;
    vector<ll> pref;
    int wl, wr;

    wavelet() { }
    wavelet(int tl, int tr, int pL, int pR, vector<ll> &v) {
        wl = tl, wr = tr;
        if (wl == wr || pL > pR) return;

        int mid = (wl + wr) >> 1;

        pref.pb(0);
        for (int i = pL; i <= pR; i++)
            pref.pb(pref.back() + (v[i] <= mid));

        ll piv = stable_partition(v.begin() + pL, v.begin() +
        pR + 1
                    , [&](int x){ return x <= mid; }) -
                    v.begin() - 1;

        left = new wavelet(wl, mid, pL, piv, v);
        right = new wavelet(mid + 1, wr, piv + 1, pR, v);
    }

    ll findKth(int k, int l, int r) {
        if (wl == wr) return wl;
        // cout << wl << " " << wr << " " << k << '\n';

        int amt = pref[r] - pref[l - 1];
        int lBound = pref[l - 1];
        int rBound = pref[r];

        if (amt >= k) return left->findKth(k, lBound + 1,
        rBound);

        return right->findKth(k - amt, l - lBound, r -
        rBound);
    }
};
```

2.4 Sparse lichao tree

```
struct Line {
  ll m, b;
  Line(ll _m = 0, ll _b = INF * 8) : m(_m), b(_b) {}
};

ll F(Line l, ll x) {
  return l.m * x + l.b;
}
struct lichao_t {
  lichao_t *left = nullptr, *right = nullptr;
  Line mn;
  lichao_t(ll tl = 0, ll tr = 0) {}
  void Update(ll tl, ll tr, Line nLine) {
    ll mid = (tl + tr) >> 1;
    bool pLeft = (F(nLine, tl) < F(mn, tl));
    bool pMid = (F(nLine, mid) < F(mn, mid));
    if (pMid)
      swap(mn, nLine);
    if (tl == tr)
      return;
    if (pLeft != pMid) {
      if (left == nullptr)
        left = new lichao_t();
      left->Update(tl, mid, nLine);
    } else {
      if (right == nullptr)
        right = new lichao_t();
      right->Update(mid + 1, tr, nLine);
    }
  }
  ll Query(ll tl, ll tr, ll x) {
    if (tl == tr)
      return F(mn, x);
    ll mid = (tl + tr) >> 1;
    ll res = F(mn, x);
    if (x <= mid) {
      if (left != nullptr)
        minimize(res, left->Query(tl, mid, x));
    } else {
      if (right != nullptr)
        minimize(res, right->Query(mid + 1, tr, x));
    }
    return res;
  }
};
```

2.5 Line Container

```
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

2.6 Ordered Set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;

template<class T>
using OrderedTree = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
```

2.7 Treap

```
struct Lazy {
    // [...]
};

struct Node {
    // [...]
};

Node operator + (Node u, Node v) {
    // [...]
    return Node{};
}

struct Tnode {
    Tnode *l = NULL, *r = NULL;
    Node node, key;
    Lazy lazy;
    int size = 1, prior = 0;
    Tnode(Node key = Node{}, int prior = randint(-1 << 30, 1
    << 30)): node(key), key(key), prior(prior) {}
};

using Pnode = Tnode*;

Pnode IgnoreNode;

// PreProcess.
void InitIgnoreNode() {
    IgnoreNode = new Tnode{};
    IgnoreNode->size = 0;
}

#define NODE(x) (x ? x : IgnoreNode)

Pnode FIX(Pnode u) {
    if(u) {
        u->size = NODE(u->l)->size + 1 + NODE(u->r)->size;
        u->node = NODE(u->l)->node + NODE(u)->key +
        NODE(u->r)->node;
    }
    return u;
}

void update_node(Pnode u, Lazy val) {
    if(!u) return;
    // [...]
}

void Down(Pnode t) {

}

Pnode merge(Pnode l, Pnode r) {
    if(!l || !r) return (l ? l : r);
    Down(l); Down(r);
    if(l->prior > r->prior) {
        l->r = merge(l->r, r);
        return FIX(l);
    } else {
        r->l = merge(l, r->l);
        return FIX(r);
    }
}

pair<Pnode, Pnode> split(Pnode t, int k) {
    if(!t) return {NULL, NULL};
    else Down(t);
    Pnode l = NULL, r = NULL;
    if(k <= NODE(t->l)->size) tie(l, t->l) = split(t->l, k), r
    = t;
    else                tie(t->r, r) = split(t->r, k - 1 -
    NODE(t->l)->size), l = t;
    FIX(t);
    return {l, r};
}

tuple<Pnode, Pnode, Pnode> split(Pnode t, int u, int v) {
    if(!t) return {NULL, NULL, NULL};
    Pnode l = NULL, m = NULL, r = NULL;
    tie(t, r) = split(t, v + 1);
    tie(l, m) = split(t, u);
    return {l, m, r};
}

void DFS(Pnode t) {
    if(!t) return;
    Down(t);
    DFS(t->l);
    // [...]
    DFS(t->r);
}
```

3 Graphs

3.1 Max Matching (Hopcroft)

```
/*
hopcroft karp for finding maximum matching on bipartite graphs
time complexity : O(E.sqrt(V))
layL[i] is the bfs layer of the ith vertex of left partition
layR[i] is for the ith vertex of the right partition
mtR[i] is the vertex matched with the ith vertex of right
partition, -1 if unmatched
adj[i] is list of neighbours of ith vertex of left partition
*/

struct hopcroft {
    ll nl, nr;

    // adj list of the left partition
    vector<vector<int>> adj;
    vector<int> layL, layR, mtR, cur, nxt;

    vector<bool> vis[2], mark;

    hopcroft(int n, int m) : nl(n), nr(m) {
        adj.assign(n, {});
    }

    bool dfs(int u, int len) {
        if(layL[u] != len) return 0;

        layL[u] = -1;
        for (int v : adj[u]) {
            if (layR[v] == len + 1) {
                layR[v] = 0;
                if (mtR[v] == -1 || dfs(mtR[v], len + 1))
                    return mtR[v] = u, 1;
            }
        }

        return 0;
    }

    ll max_matching() {
        layL.assign(nl, 0);
        layR.assign(nr, 0);
        mtR.assign(nr, -1);

        ll res = 0;

        while (true) {
            fill(all(layL), 0);
```

```
        fill(all(layR), 0);
        cur.clear();

        for (int u : mtR)
            if(u != -1) layL[u] = -1;

        for (int i = 0; i < sz(adj); i++)
            if (layL[i] != -1)
                cur.pb(i);

        bool isLast = false;
        for (int lay = 1; ; lay++) {
            nxt.clear();

            for (int u : cur) {
                for (int v : adj[u]) {
                    if (mtR[v] == -1) {
                        layR[v] = lay;
                        isLast = true;
                    }
                    else if (mtR[v] != u && !layR[v]) {
                        layR[v] = lay;
                        nxt.pb(mtR[v]);
                    }
                }
            }

            if (isLast) break;

            if (nxt.empty()) return res;

            for (int u : nxt)
                layL[u] = lay;

            swap(cur, nxt);
        }

        for (int i = 0; i < sz(adj); i++)
            res += dfs(i, 0);
    }
}

void dfs2(int u, int l) {
    vis[l][u] = true;

    if (!l) {
        for (int v : adj[u]) {
            if (!vis[1][v])
                dfs2(v, 1);
        }
    }
    else {
        if (mtR[u] != -1 && !vis[0][mtR[u]])
            dfs2(mtR[u], 0);
    }
}

//edges in matching -> right to left, else left to right
//return {left/right, index} of minimum cover
vector<pll> minCover() {
    vis[0].assign(nl, false);
    vis[1].assign(nr, false);
    mark.assign(nl, false);

    vector<pll> res;
    for (int i = 0; i < nr; i++)
        if (mtR[i] != -1)
            mark[mtR[i]] = true;

    for (int i = 0; i < nl; i++)
        if (!mark[i])
            dfs2(i, 0);

    //unvisited of the left and visited of the right is in
    min cover
    for (int i = 0; i < nl; i++)
        if (!vis[0][i])
            res.pb({0, i});

    for (int i = 0; i < nr; i++)
        if (vis[1][i])
            res.pb({1, i});

    return res;
}
};
```

3.2 Max Matching (Blossom)

```
/* Complexity: O(E*sqrt(V))
*/
struct Blossom {
    static const int MAXV = 1e3 + 5;
    static const int MAXE = 1e6 + 5;
    int n, E, lst[MAXV], next[MAXE], adj[MAXE];
    int nxt[MAXV], mat[MAXV], dad[MAXV], col[MAXV];
    int que[MAXV], qh, qt;
    int vis[MAXV], act[MAXV];
    int tag, total;

    void init(int n) {
        this->n = n;
        for (int i = 0; i <= n; i++) {
            lst[i] = nxt[i] = mat[i] = vis[i] = 0;
        }
        E = 1, tag = total = 0;
    }
    void add(int u,int v) {
        if (!mat[u] && !mat[v]) mat[u] = v, mat[v] = u,
        total++;
        E++, adj[E] = v, next[E] = lst[u], lst[u] = E;
        E++, adj[E] = u, next[E] = lst[v], lst[v] = E;
    }
    int lca(int u, int v) {
        tag++;
        for(; ; swap(u, v)) {
            if (u) {
                if (vis[u = dad[u]] == tag) {
                    return u;
                }
                vis[u] = tag;
                u = nxt[mat[u]];
            }
        }
    }
    void blossom(int u, int v, int g) {
        while (dad[u] != g) {
            nxt[u] = v;
            if (col[mat[u]] == 2) {
                col[mat[u]] = 1;
                que[++qt] = mat[u];
            }
            if (u == dad[u]) dad[u] = g;
            if (mat[u] == dad[mat[u]]) dad[mat[u]] = g;
            v = mat[u];
            u = nxt[v];
        }
}

    int augument(int s) {
        for (int i = 1; i <= n; i++) {
            col[i] = 0;
            dad[i] = i;
        }
        qh = 0; que[qt = 1] = s; col[s] = 1;
        for (int u, v, i; qh < qt; ) {
            act[u = que[++qh]] = 1;
            for (i = lst[u];i ; i = next[i]) {
                v = adj[i];
                if (col[v] == 0) {
                    nxt[v] = u;
                    col[v] = 2;
                    if (!mat[v]) {
                        for (; v; v = u) {
                            u = mat[nxt[v]];
                            mat[v] = nxt[v];
                            mat[nxt[v]] = v;
                        }
                        return 1;
```

```
                }
                col[mat[v]] = 1;
                que[++qt] = mat[v];
            }
            else if (dad[u] != dad[v] && col[v] == 1) {
                int g = lca(u, v);
                blossom(u, v, g);
                blossom(v, u, g);
                for (int j = 1; j <= n; j++) {
                    dad[j] = dad[dad[j]];
                }
            }
        }
    }
    return 0;
}
int maxmat() {
    for (int i = 1; i <= n; i++) {
        if (!mat[i]) {
            total += augument(i);
        }
    }
    return total;
}
}
```

3.3 Max Flow (Push Relabel)

```
/**
 * Author: Simon Lindholm
 * Date: 2015-02-24
 * License: CC0
 * Source: Wikipedia, tinyKACTL
 * Description: Push-relabel using the highest label selection
rule and the gap heuristic. Quite fast in practice.
 *  To obtain the actual flow, look at positive values only.
 * Time: $O(V^2\sqrt E)$
 * Status: Tested on Kattis and SPOJ, and stress-tested
 */
#pragma once


struct PushRelabel {
  struct Edge {
    int dest, back;
    ll f, c;
  };
  vector<vector<Edge>> g;
  vector<ll> ec;
  vector<Edge*> cur;
  vector<vi> hs; vi H;
  PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}

  void addEdge(int s, int t, ll cap, ll rcap=0) {
    if (s == t) return;
    g[s].push_back({t, sz(g[t]), 0, cap});
    g[t].push_back({s, sz(g[s])-1, 0, rcap});
  }

  void addFlow(Edge& e, ll f) {
    Edge &back = g[e.dest][e.back];
    if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
    e.f += f; e.c -= f; ec[e.dest] += f;
    back.f -= f; back.c += f; ec[back.dest] -= f;
  }
  ll calc(int s, int t) {
    int v = sz(g); H[s] = v; ec[t] = 1;
    vi co(2*v); co[0] = v-1;
    rep(i,0,v) cur[i] = g[i].data();
    for (Edge& e : g[s]) addFlow(e, e.c);

    for (int hi = 0;;) {
      while (hs[hi].empty()) if (!hi--) return -ec[s];
      int u = hs[hi].back(); hs[hi].pop_back();
      while (ec[u] > 0)  // discharge u
        if (cur[u] == g[u].data() + sz(g[u])) {
          H[u] = 1e9;
          for (Edge& e : g[u]) if (e.c && H[u] > H[e.dest]+1)
            H[u] = H[e.dest]+1, cur[u] = &e;
          if (++co[H[u]], !--co[hi] && hi < v)
            rep(i,0,v) if (hi < H[i] && H[i] < v)
              --co[H[i]], H[i] = v + 1;
          hi = H[u];
        } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
          addFlow(*cur[u], min(ec[u], cur[u]->c));
        else ++cur[u];
    }
  }
  bool leftOfMinCut(int a) { return H[a] >= sz(g); }
};
```

3.4 Gomory Hu

```
const int INF = 1000000000;

struct Edge {
    int a, b, cap, flow;
};

struct MaxFlow {
    int n, s, t;
    vector<int> d, ptr, q;
    vector< Edge > e;
    vector< vector<int> > g;
    MaxFlow() = default;
    MaxFlow(int _n) : n(_n), d(_n), ptr(_n), q(_n), g(_n) {
        e.clear();
        for (int i = 0; i < n; i++) {
            g[i].clear();
            ptr[i] = 0;
        }
    }
    void addEdge(int a, int b, int cap) {
        Edge e1 = { a, b, cap, 0 };
        Edge e2 = { b, a, 0, 0 };
        g[a].push_back( (int) e.size() );
        e.push_back(e1);
        g[b].push_back( (int) e.size() );
        e.push_back(e2);
    }
    int getMaxFlow(int _s, int _t) {
        s = _s; t = _t;
        int flow = 0;
        for (;;) {
            if (!bfs()) break;
            std::fill(ptr.begin(), ptr.end(), 0);
            while (int pushed = dfs(s, INF))
                flow += pushed;
        }
        return flow;
    }
private:
    bool bfs() {
        int qh = 0, qt = 0;
        q[qt++] = s;
        std::fill(d.begin(), d.end(), -1);
        d[s] = 0;
        while (qh < qt && d[t] == -1) {
            int v = q[qh++];
            for (int i = 0; i < (int) g[v].size(); i++) {
                int id = g[v][i], to = e[id].b;
                if (d[to] == -1 && e[id].flow < e[id].cap) {
                    q[qt++] = to;
                    d[to] = d[v] + 1;
                }
            }
        }
        return d[t] != -1;
    }
    int dfs (int v, int flow) {
        if (!flow) return 0;
        if (v == t) return flow;
        for (; ptr[v] < (int)g[v].size(); ++ptr[v]) {
            int id = g[v][ptr[v]],
                to = e[id].b;
            if (d[to] != d[v] + 1) continue;
            int pushed = dfs(to, min(flow, e[id].cap -
            e[id].flow));
```

```
            if (pushed) {
                e[id].flow += pushed;
                e[id^1].flow -= pushed;
                return pushed;
            }
        }
        return 0;
    }
};

const int N = 202;
int ok[N], cap[N][N];
int answer[N][N], parent[N];
int n;
MaxFlow flow;

void Init(int vertices) {
    n = vertices;
    flow = MaxFlow(vertices);
    for(int i = 0; i < vertices; ++i) ok[i] = parent[i] = 0;
    for(int i = 0; i < vertices; ++i)
        for(int j = 0; j < vertices; ++j)
            cap[i][j] = 0, answer[i][j] = INF;
}

void bfs(int start) {
    memset(ok,0,sizeof ok);
    queue<int> qu;
    qu.push(start);
    while (!qu.empty()) {
        int u=qu.front(); qu.pop();
        for(int xid = 0; xid < (int) flow.g[u].size(); ++xid)
        {
            int id = flow.g[u][xid];
            int v = flow.e[id].b, F = flow.e[id].flow, C =
            flow.e[id].cap;
            if (!ok[v] && F < C) {
                ok[v]=1;
                qu.push(v);
            }
        }
    }
}
void FindMaxFlow() {
    for(int i = 1; i <= n-1; ++i) {
        flow = MaxFlow(n);
        for(int u = 0; u < n; ++u)
            for(int v = 0; v < n; ++v)
            if (cap[u][v])
                flow.addEdge(u, v, cap[u][v]);

        int f = flow.getMaxFlow(i, parent[i]);

        bfs(i);
        for(int j = i+1; j < n; ++j)
            if (ok[j] && parent[j]==parent[i])
                parent[j]=i;

        answer[i][parent[i]] = answer[parent[i]][i] = f;
        for(int j = 0; j < i; ++j)

            answer[i][j]=answer[j][i]=min(f,answer[parent[i]][j]);
    }
}
```

3.5 Min Cost Max Flow

```
int n, m, k, source, sink;
struct FlowEdge {
    int to, rev, id, flow, cap, cost;
};
vector<FlowEdge> adj[MAX_N];
int dist[MAX_N];
bool inQueue[MAX_N];
pii trc[MAX_N];
queue<int> q;
int ans;

void addEdge(int u, int v, int cost, int cap) {
    int szU = adj[u].size();
    int szV = adj[v].size();
    adj[u].pb({v, szV, szU, 0, cap, cost});
    adj[v].pb({u, szU, szV, 0, cap, cost});
}

bool BellmanFord() {
    for (int i = 1; i <= n; i++) {
        dist[i] = inf;
    }
    dist[source] = 0;
    q.push(source);
    inQueue[source] = true;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inQueue[u] = false;
        for (auto e : adj[u]) {
            int v = e.to;
            int c = (e.flow >= 0 ? 1 : -1) * e.cost;
            if (e.flow < e.cap && dist[u] + c < dist[v]) {
                dist[v] = dist[u] + c;
                trc[v] = {u, e.id};
                if (!inQueue[v]) {
                    q.push(v);
                }
            }
        }
    }
    return dist[sink] < inf;
}

void inc() {
    int incFlow = inf;

    for (int i = sink; i != source; i = trc[i].fi) {
        int u = trc[i].fi;
        int id = trc[i].se;
        minimize(incFlow, (adj[u][id].flow >= 0 ?
        adj[u][id].cap - adj[u][id].flow : -adj[u][id].flow));
    }

    minimize(incFlow, k);


    for (int i = sink; i != source; i = trc[i].fi) {
        int u = trc[i].fi;
        int id = trc[i].se;
        adj[u][id].flow += incFlow;
        adj[i][adj[u][id].rev].flow -= incFlow;
    }

    ans += incFlow * dist[sink];
    k -= incFlow;

    if (!k) {
        cout << ans << '\n';
        for (int i = 1; i <= n; i++) {
            for (auto e : adj[i]) {
                if (e.flow > 0) {
                    cout << i << " " << e.to << " " << e.flow
                    << '\n';
                }
            }
        }
        cout << "0 0 0";
        exit(0);
    }
}

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    cin >> n >> m >> k >> source >> sink;
    for (int i = 1; i <= m; i++) {
        int u, v, c, d;
        cin >> u >> v >> c >> d;
        addEdge(u, v, c, d);
    }
```

```
    while (BellmanFord()) {
        inc();
    }

    cout << -1;

    return 0;
}
```

3.6 Weighted Matching (Hungarian)

```
/**
 * Author: Benjamin Qi, chilli
 * Date: 2020-04-04
 * License: CC0
 * Description: Given a weighted bipartite graph, matches
every node on
 * the left with a node on the right such that no
 * nodes are in two matchings and the sum of the edge weights
is minimal. Takes
 * cost[N][M], where cost[i][j] = cost for L[i] to be matched
with R[j] and
 * returns (min cost, match), where L[i] is matched with
 * R[match[i]]. Negate costs for max cost. Requires N <= M.
 * Time: O(N^2M)
 * Status: Tested on kattis:cordonbleu, stress-tested
 */
#pragma once

pair<int, vi> hungarian(const vector<vi> &a) {
  if (a.empty()) return {0, {}};
  int n = sz(a) + 1, m = sz(a[0]) + 1;
  vi u(n), v(m), p(m), ans(n - 1);
  rep(i,1,n) {
    p[0] = i;
    int j0 = 0; // add "dummy" worker 0
    vi dist(m, INT_MAX), pre(m, -1);
    vector<bool> done(m + 1);
    do { // dijkstra
      done[j0] = true;
      int i0 = p[j0], j1, delta = INT_MAX;
      rep(j,1,m) if (!done[j]) {
        auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
        if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
        if (dist[j] < delta) delta = dist[j], j1 = j;
      }
      rep(j,0,m) {
        if (done[j]) u[p[j]] += delta, v[j] -= delta;
        else dist[j] -= delta;
      }
      j0 = j1;
    } while (p[j0]);
    while (j0) { // update alternating path
      int j1 = pre[j0];
      p[j0] = p[j1], j0 = j1;
    }
  }
  rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
  return {-v[0], ans}; // min cost
}
```

3.7 Max Clique

```
template <size_t max_n>
class Clique
{
    using bits = bitset<max_n>;
    bits MASK, ZERO, ans;
    const bits *e;
    int N;
    // int64_t calls;
void bk_init()
    {
        ans = ZERO;
        MASK = ZERO;
        MASK.flip();
        // calls = 0;
    }
    void bk(bits use, bits can_start, bits can_other)
    {
        // ++calls;
        if (can_start.none() && can_other.none())
        {
            if (use.count() > ans.count())
                ans = use;
            return;
        }
        bits r = can_start;
        bool fi = 1;
        for (int i = 0; i < N; ++i)
        {
            if (r[i])
            {
                if (fi)
                {
                    fi = 0;
                    r &= e[i] ^ MASK;
                }
                use[i] = 1;
                bk(use, can_start & e[i], can_other & e[i]);
                use[i] = 0;
                can_start[i] = 0;
                can_other[i] = 1;
            }
        }
    }
    static Clique &get()
    {
        static Clique c;
        return c;
    }

public:
    static bits find_clique(bits const *g, const int &n)
    {
        Clique &c = get();
        c.e = g;
        c.N = n;
        c.bk_init();
        bits me;
        c.bk(me, c.MASK, c.ZERO);
        // cerr << "Calls: " << c.calls << "\n";
        // c.calls = 0;
        return c.ans;
    }
    static bits find_clique(vector<bits> const &g)
    {
        return find_clique(g.data(), g.size());
    }
    static bits find_clique(array<bits, max_n> const &g, const
    int &n)
    {
        return find_clique(g.data(), n);
    }
};
```

3.8 2 SAT

```
//source: https://wiki.vnoi.info/vi/algo/graph-theory/2-SAT

#include <bits/stdc++.h>

using namespace std;

const int maxN = 500500;

int n, m;

// Lu đ th G
vector<int> G[maxN << 1];

// Ly giá tr ph đnh ca x
int NOT(int x) {
    return x + (x <= n ? n : -n); // -x
}
```

```
// Thêm điu kin u OR v
void add_clause(int u, int v) {
    G[NOT(u)].push_back(v); // -u -> v
    G[NOT(v)].push_back(u); // -v -> u
}

// Tìm thành phn liên thông mnh
int id[maxN << 1];
int num[maxN << 1], low[maxN << 1];
int timeDFS = 0, scc = 0;
int st[maxN << 1];

void dfs(int u) {
    num[u] = low[u] = ++timeDFS;
    st[++st[0]] = u;
    for(const int& v : G[u]) {
        if(id[v] != 0) continue;
        if(num[v] == 0) {
            dfs(v);
            low[u] = min(low[u], low[v]);
        } else low[u] = min(low[u], num[v]);
    }

    if(num[u] == low[u]) {
        for(++scc; true; ) {
            int v = st[st[0]--];
            id[v] = scc;
            if(v == u) break;
        }
    }
}

int main() {
    cin.tie(0)->sync_with_stdio(0);

    cin >> n >> m;
    for(int i = 1; i <= m; ++i) {
        int u, v; cin >> u >> v;
        add_clause(u, v);
    }

    // Thut toán Tarjan
    for(int i = 1; i <= 2 * n; ++i) {
        if(!id[i]) dfs(i);
    }

    bool answer = 1;
    for(int i = 1; i <= n; ++i) {
        // Kim tra điu kin tn ti phng án
        if(id[i] == id[NOT(i)]) answer = 0;
    }
    if(!answer) {
        cout << "IMPOSSIBLE"; // Thông báo bài toán vô nghim
        return 0;
    }
    // In tp giá tr a1, a2, ..., an
    for(int i = 1; i <= n; ++i) cout << (id[i] < id[NOT(i)])
    << " ";
    return 0;
}
```

4 DP

4.1 Divide And Conquer Optimization

```
void compute(int l, int r, int optl, int optr) {
    if (l > r)
        return;

    int mid = (l + r) >> 1;
    pair<long long, int> best = {LLONG_MAX, -1};

    for (int k = optl; k <= min(mid, optr); k++) {
        best = min(best, {(k ? dp_before[k - 1] : 0) + C(k,
        mid), k});
    }

    dp_cur[mid] = best.first;
    int opt = best.second;

    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}
```

5 Strings

5.1 KMP

```
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

5.2 Z Function

```
vector<int> zfunction(const string& s) {
    int n = (int) s.size();
    vector<int> z(n);
    for(int i = 1, l = 0, r = 0; i < n; ++i) {
         if(i <= r) z[i] = min(r - i, z[i - l]);
         while(i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
         if(i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    } return z;
}
```

5.3 Aho Corasick (static)

```
const int CAP = 1003, ALPHABET = 26;
int cntTrie = 1;
int fail[CAP], to[CAP][ALPHABET];
bool ending[CAP];

void add_string(const string& s) {
    int u = 1;
    for(const char& c : s) {
        int x = c - 'a';
        if(!to[u][x]) {
            to[u][x] = ++cntTrie;
        }
        u = to[u][x];
    }
    ending[u] = true;
}

void aho_corasick() {
    queue<int> q; q.push(1);
    while(q.size()) {
        int u = q.front(); q.pop();
        for(int x = 0; x < ALPHABET; ++x) {
            int& v = to[u][x];
            if(!v) {
                v = u == 1 ? 1 : to[fail[u]][x];
            } else {
                if(!fail[v]) fail[v] = fail[u];
                fail[v] = u == 1 ? 1 : to[fail[v]][x];
                ending[v] |= ending[fail[v]];
                q.push(v);
            }
        }
    }
}
```

5.4 Aho Corasick (vector)

```
struct TrieNode {
  int pi = 0;
  int child[26] = {0};
};
vector<TrieNode> trie;
vector<vector<int>> adj;

int TrieInsert(const string& s) {
  int p = 0;
  for (int i = 0; i < s.size(); i++) {
    if (!trie[p].child[s[i] - 'a']) {
      trie[p].child[s[i] - 'a'] = trie.size();
      trie.pb(TrieNode());
    }
    p = trie[p].child[s[i] - 'a'];
  }
  return p;
}

void AhoCorasickBuild() {
  queue<int> q;
  for (int i = 0; i < 26; i++) {
    if (trie[0].child[i]) {
      q.push(trie[0].child[i]);
    }
  }
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    for (int i = 0; i < 26; i++) {
      if (!trie[u].child[i]) continue;
      int j = trie[u].pi;
      while (!trie[j].child[i]) {
        if (!j) break;
        j = trie[j].pi;
      }
      trie[trie[u].child[i]].pi = trie[j].child[i];
      q.push(trie[u].child[i]);
    }
    adj[trie[u].pi].pb(u);
  }
}

signed main() {
  trie.pb(TrieNode());


  adj.resize(trie.size());
  AhoCorasickBuild();
}
```

---

6 Math

6.1 Chinese Remainder Theorem

```
struct gcd_t { ll x, y, d; };

gcd_t e_gcd(ll a, ll b) {
    if (b == 0) return {1, 0, a};

    gcd_t res = e_gcd(b, a % b);

    return {res.y, res.x - res.y * (a / b), res.d};
}

pll crt(vector<ll> r, vector<ll> m) {
    //find x such that for (1 <= i <= n): x = r[i] (mod m[i])
    //return {y, z} where x = y (mod z), z = lcm of vector m
    //all solutions are congruent modulo z

    ll y = r[0], z = m[0];
    for (int i = 1; i < sz(r); i++) {
        gcd_t cur = e_gcd(z, m[i]);

        ll x = cur.x, d = cur.d;

        if((r[i] - y) % d != 0) return {-1, -1};

        //ka = kb (mod kc)   =>   a = b (mod c) if (gcd(k, c)
        = 1)
        //add (x * (r[i] - y) / d * z) to result (with moduli
        lcm(z, m[i]))
        ll tmp = (x * (r[i] - y) / d) % (m[i] / d);
        y = y + tmp * z;
        z = z / d * m[i];

        y %= z;
        if (y < 0) y += z;
    }

    return {y, z};
}

ll inverse(ll a, ll m) {
    gcd_t cur = e_gcd(a, m);

    return (cur.x % m + m) % m;
}
```

6.2 Miller Rabin

```
// From
https://github.com/SnapDragon64/ContestLibrary/blob/master/math.h
// which also has specialized versions for 32-bit and 42-bit
//
// Tested:
// - https://oj.vnoi.info/problem/icpc22_national_c (fastest
solution)
// - https://www.spoj.com/problems/PON/

// Rabin miller {{{
inline uint64_t mod_mult64(uint64_t a, uint64_t b, uint64_t m)
{
    return __int128_t(a) * b % m;
}
uint64_t mod_pow64(uint64_t a, uint64_t b, uint64_t m) {
    uint64_t ret = (m > 1);
    for (;;) {
        if (b & 1) ret = mod_mult64(ret, a, m);
        if (!(b >>= 1)) return ret;
        a = mod_mult64(a, a, m);
    }
}

// Works for all primes p < 2^64
bool is_prime(uint64_t n) {
    if (n <= 3) return (n >= 2);
    static const uint64_t small[] = {
        2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
        47, 53, 59, 61, 67,
        71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127,
        131, 137, 139,
        149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197,
        199,
    };
    for (size_t i = 0; i < sizeof(small) / sizeof(uint64_t);
    ++i) {
        if (n % small[i] == 0) return n == small[i];
    }

    // Makes use of the known bounds for Miller-Rabin
    pseudoprimes.
    static const uint64_t millerrabin[] = {
        2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
    };
    static const uint64_t A014233[] = {  // From OEIS.
        2047LL, 1373653LL, 25326001LL, 3215031751LL,
        2152302898747LL,
        3474749660383LL, 341550071728321LL, 341550071728321LL,
        3825123056546413051LL, 3825123056546413051LL,
        3825123056546413051LL, 0,
    };
    uint64_t s = n-1, r = 0;
    while (s % 2 == 0) {
        s /= 2;
        r++;
```

```
    }
    for (size_t i = 0, j; i < sizeof(millerrabin) /
    sizeof(uint64_t); i++) {
        uint64_t md = mod_pow64(millerrabin[i], s, n);
        if (md != 1) {
            for (j = 1; j < r; j++) {
                if (md == n-1) break;
                md = mod_mult64(md, md, n);
            }
            if (md != n-1) return false;
        }
        if (n < A014233[i]) return true;
    }
    return true;
}
// }}}
```

6.3 Discrete Logarithm

```
// Computes x which a ^ x = b mod n.

long long d_log(long long a, long long b, long long n) {
    long long m = ceil(sqrt(n));
    long long aj = 1;
    map<long long, long long> M;
    for (int i = 0; i < m; ++i) {
        if (!M.count(aj))
            M[aj] = i;
        aj = (aj * a) % n;
    }

    long long coef = mod_pow(a, n - 2, n);
    coef = mod_pow(coef, m, n);
    // coef =  a ^ (-m)
    long long gamma = b;
    for (int i = 0; i < m; ++i) {
        if (M.count(gamma)) {
            return i * m + M[gamma];
        } else {
            gamma = (gamma * coef) % n;
        }
    }
    return -1;
}
```

6.4 Fast Fourier Transform

```
typedef complex<double> cmplx;
typedef vector<complex<double> > VC;
const double PI = acos(-1);
struct FFT {
    static void fft(VC &u, int sign) {
        int n = u.size();
        double theta = 2. * PI * sign / n;
        for (int m = n; m >= 2; m >>= 1, theta *= 2.) {
            cmplx w(1, 0), wDelta = polar(1., theta);
            for (int i = 0, mh = m >> 1; i < mh; i++) {
                for (int j = i; j < n; j += m) {
                    int k = j + mh;
                    cmplx temp = u[j] - u[k];
                    u[j] += u[k];
                    u[k] = w * temp;
                }
                w *= wDelta;
            }
        }
        for (int i = 1, j = 0; i < n; i++) {
            for (int k = n >> 1; k > (j ^= k); k >>= 1);
            if (j < i) {
                swap(u[i], u[j]);
            }
        }
    }

    static vector<ll> mul(const vector<int> &a, const
    vector<int> &b) {
        int newSz = a.size() + b.size() - 1;
        int fftSz = 1;
        while (fftSz < newSz) fftSz <<= 1;
        VC aa(fftSz, 0.), bb(fftSz, 0.);
        for (int i = 0; i < a.size(); i++) aa[i] = a[i];
        for (int i = 0; i < b.size(); i++) bb[i] = b[i];
        fft(aa, 1), fft(bb, 1);
        for (int i = 0; i < fftSz; i++) aa[i] *= bb[i];
        fft(aa, -1);
        vector<ll> res(newSz);
        for (int i = 0; i < newSz; i++)
            res[i] = (ll)(aa[i].real() / fftSz + 0.5);
        return res;
    }
};
```

6.5 Berlekamp massey

```
template<typename T> vector<T> berlekampMassey(const vector<T>
&sequence) {
    int n = (int)sequence.size(), len = 0, m = 1;
    vector<T> prevBest(n), coefficients(n);
    T prevDelta = prevBest[0] = coefficients[0] = 1;
    for (int i = 0; i < n; i++, m++) {
        T delta = sequence[i];
        for (int j = 1; j <= len; j++) delta +=
        coefficients[j] * sequence[i - j];
        if ((long long)delta == 0) continue;
        vector<T> temp = coefficients;
        T coef = delta / prevDelta;
        for (int j = m; j < n; j++) coefficients[j] -= coef *
        prevBest[j - m];
        if ((len << 1) <= i)
            len = i + 1 - len, prevBest = temp, prevDelta =
            delta, m = 0;
    }
    coefficients.resize(len + 1);
    coefficients.erase(coefficients.begin());
    for (T &x : coefficients) x = -x;
    return coefficients;
}

template<typename T> T calcKthTerm(
    const vector<T> &coefficients, const vector<T> &sequence,
    long long k
) {
    assert(coefficients.size() <= sequence.size());
    int n = (int)coefficients.size();

    auto mul = [&](const vector<T> &a, const vector<T> &b) {
        vector<T> res(a.size() + b.size() - 1u);
        for (int i = 0; i < (int)a.size(); i++)
            for (int j = 0; j < (int)b.size(); j++)
                res[i + j] += a[i] * b[j];
        for (int i = (int)res.size() - 1; i >= n; i--)
            for (int j = n - 1; j >= 0; j--)
                res[i - j - 1] += res[i] * coefficients[j];
        res.resize(min((int)res.size(), n));
        return res;
    };

    vector<T> a = (n == 1 ? vector<T>{coefficients[0]} :
    vector<T>{0, 1}), x{1};
    for (; k; k >>= 1) {
        if (k & 1) x = mul(x, a);
        a = mul(a, a);
    }
    x.resize(n);
    T res = 0;
    for (int i = 0; i < n; i++) res += x[i] * sequence[i];
    return res;
}

// Usual: cout << calcKthTerm(berlekampMassey(ans), ans, n) <<
"\n";
```

7 Geometry

7.1 Geomtry Point

```
struct Point{
    typedef ll T;
    T x, y;
    Point(T _x = 0, T _y = 0) : x(_x), y(_y) {}

    bool operator < (Point p) const { return tie(x, y) <
    tie(p.x, p.y); }
    bool operator > (Point p) const { return tie(x, y) >
    tie(p.x, p.y); }
    bool operator == (Point p) const { return tie(x, y) ==
    tie(p.x, p.y); }
    bool operator != (Point p) const { return tie(x, y) !=
    tie(p.x, p.y); }

    Point operator + (Point p) const { return Point(x + p.x, y
    + p.y); }
    Point operator - (Point p) const { return Point(x - p.x, y
    - p.y); }
    T operator * (Point p) const { return x * p.x + y * p.y; }
    T operator ^ (Point p) const { return x * p.y - y * p.x; }

    Point operator * (T d) const { return Point(x * d, y * d);
    }
    Point operator / (T d) const { return Point(x / d, y / d);
    }

    T len2() const { return x * x + y * y; }
    double len() const { return sqrt((double)len2()); }
    Point perp() { return Point(-y, x); }
    friend ostream& operator << (ostream &os, const Point &p)
    {
        return os << "(" << p.x << ", " << p.y << ")"; }
};

ll ccw(const Point &P0, const Point &P1, const Point &P2){
    return (P1 - P0) ^ (P2 - P1);
}

ll sgn(const ll &x) { return (x >= 0 ? (x ? 1 : 0) : -1); }
```

7.2 Convex Hull

```
vector<Point> convexHull(vector<Point> dots) {
    sort(dots.begin(), dots.end());
    vector<Point> A(1, dots[0]);
    const int sz = dots.size();
    for(int c = 0; c < 2; reverse(all(dots)), c++)
    for(int i = 1, t = A.size(); i < sz;
    A.emplace_back(dots[i++]))
      while (A.size() > t and ccw(A[A.size()-2], A.back(),
      dots[i]) < 0)
        A.pop_back();
    A.pop_back(); return A;
}
```

7.3 Manhattan MST

```
vector<array<ll, 3>> manhattanMST(vector<Point> ps) {
    vector<int> id(sz(ps));
    iota(all(id), 0);
    vector<array<ll, 3>> edges;
    for (int k = 0; k < 4; k++) {
            sort(all(id), [&](int i, int j) { return (ps[i] -
            ps[j]).x < (ps[j] - ps[i]).y; });
            map<int, int> sweep;
            for (int i : id) {
                for (auto it = sweep.lower_bound(-ps[i].y);
                    it != sweep.end(); sweep.erase(it++)) {
                    int j = it->second;
                    Point d = ps[i] - ps[j];
                    if (d.y > d.x) break;
                    edges.push_back({d.y + d.x, i, j});
                }
                sweep[-ps[i].y] = i;
            }
            for (Point &p : ps)
                if (k & 1)
                    p.x = -p.x;
                else
                    swap(p.x, p.y);
    }
    return edges;
}
```

7.4 Some Common Geometry Operations

```
ll ccw(const Point &P0, const Point &P1, const Point &P2){
    return (P1 - P0) ^ (P2 - P1);
}

bool on_segment(Point &p, Point &p0, Point &p1){
    if((p1 - p0) * (p - p1) > 0) return false;
    if((p0 - p1) * (p - p0) > 0) return false;
    return (ccw(p, p0, p1) == 0);
}

db dist_segment(Point &p, Point &p0, Point &p1){
    if((p1 - p0) * (p - p1) >= 0) return (p - p1).len();
    if((p0 - p1) * (p - p0) >= 0) return (p - p0).len();
    return abs((db)((p1 - p0) ^ (p - p0)) / (p1 - p0).len());
}

bool insideConvex(Point p, vector<Point> &poly){
    // clock wise
    int n = sz(poly);
    if(ccw(poly[0], poly[1], p) >= 0) return false;
    if(ccw(poly[n - 1], poly[0], p) >= 0) return false;

    ll l = 1, r = n-1;
    while(l < r){
        ll mid = (l+r+1)/2;
        if(ccw(poly[0], p, poly[mid]) >= 0) l = mid;
        else r = mid - 1;
    }
    r = l + 1;

    return (ccw(poly[l], p, poly[r]) > 0);
}

ll wn_poly(Point p, vector<Point> &poly){
    // 1 if inside 0 if outside, INF if on boundary
    // counter clock wise
    const ll on_boundary = INF;
    ll wn = 0;

    int n = sz(poly);
    for(int i = 0; i < n; i++){
        if(p == poly[i]) return on_boundary;

        int j = (i + 1 != n ? i + 1 : 0);

        if(poly[i].y == p.y && poly[j].y == p.y){
            if(min(poly[i].x, poly[j].x) <= p.x
                 && p.x <= max(poly[i].x, poly[j].x))
                return on_boundary;
        }
        else{
            bool below = (poly[i].y <= p.y);
            //different sides of horizontal ray
            if (below != (poly[j].y <= p.y)){
                ll orientation = ccw(p, poly[i], poly[j]);

                if (orientation == 0) return on_boundary;
                if (below == (orientation > 0)) wn += (below ?
                1 : -1);
            }
        }
    }

    return wn;
}

bool line_intersect(pii a, pii b, pii c, pii d) {
```

```
  if (!ccw(c, a, b) || !ccw(d, a, b) || !ccw(a, c, d) ||
  !ccw(b, c, d)) {
    if (!ccw(c, a, b) && dot_product(a, c, b) <= 0) {
      return true;
    }
    if (!ccw(d, a, b) && dot_product(a, d, b) <= 0) {
      return true;
    }
    if (!ccw(a, c, d) && dot_product(c, a, d) <= 0) {
      return true;
    }
    if (!ccw(b, c, d) && dot_product(c, b, d) <= 0) {
      return true;
    }
    return false;
  }
  return (ccw(a, b, c) * ccw(a, b, d) < 0 && ccw(c, d, a) *
  ccw(c, d, b) < 0);
```

8 Miscellaneous

8.1 Hilber Order for Mo's

```
inline ll gilbertOrder(int x, int y, int pow, int rotate) {
    if (pow == 0) {
        return 0;
    }
    int hpow = 1 << (pow - 1);
    int seg = (x < hpow) ? ((y < hpow) ? 0 : 3) : ((y < hpow)
    ? 1 : 2);
    seg = (seg + rotate) & 3;
    const int rotateDelta[4] = {3, 0, 0, 1};
    int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
    int nrot = (rotate + rotateDelta[seg]) & 3;
    ll subSquareSize = ll(1) << (2 * pow - 2);
    ll ans = seg * subSquareSize;
    ll add = gilbertOrder(nx, ny, pow - 1, nrot);
    ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add
    - 1);
    return ans;
}

struct Query {
    ll l, r, id, ord;
    Query(int _l, int _r, int _id) : l(_l), r(_r), id(_id) {}
    inline void calc() {
        ord = gilbertOrder(l, r, 21, 0);
    }
};

inline bool operator<(const Query &a, const Query &b)
{
    return a.ord < b.ord;
}
```