Team notebook

CatsOnTrees

October 2, 2022

Contents

1 Data-Structures

1.1 mergeSortTree

```
const int maxn = 2e5 + 5;
vecl a(maxn);
```

```
vecl t[4*maxn];
void merge(vecl& temp1, vecl& temp2,
    vecl& final){
    int i = 0, j = 0;
    while(i < sz(temp1) && j <
        sz(temp2)){
        if(temp1[i] <= temp2[j]) {
            final.pb(temp1[i]);
            i++;
        } else {
            final.pb(temp2[j]);
            j++;
        }
    }
    while(i < sz(temp1)){
        final.pb(temp1[i]);
        i++;
    }
    while(j < sz(temp2)){
        final.pb(temp2[j]);
        j++;
    }
}
void build(int ind, int tl, int tr){
    if(tl == tr){
        t[ind].pb(a[tl]);
        return;
    }
    int tm = (tl + tr) / 2;
    build(2 * ind, tl, tm);
    build(2 * ind + 1, tm + 1, tr);
    merge(t[2 * ind], t[2 * ind + 1],
        t[ind]);
}
```

```
int query(int ind, int tl, int tr, int
    l, int r, int valuetoCompare){ //
    query for elements strictly greater
    than k
    if(l > r){
        return 0;
    }
    if(l == tl && r == tr){
        return t[ind].end() -
            upper_bound(all(t[ind]),
            valuetoCompare);
    }
    int tm = (tl + tr) / 2;
    return (query(2 * ind, tl, tm, l,
        min(r, tm), valuetoCompare) +
            query(2 * ind + 1, tm + 1,
                tr, max(l, tm + 1), r,
                valuetoCompare));
}
```

1.2 policybased

```
/*
find_by_order(k): return iterator to
    k'th element(counting from zero)
order_of_key(k) : number of items < k in
    O(logn) time.
*/
include <ext/pb_ds/assoc_container.hpp>
include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>,
rb_tree_tag,
    tree_order_statistics_node_update>
    ordered_set;
```

1.3 segmentTree

```
/*
Usage: Node* tr = new Node(v, 0, sz(v));
finds max in a range, [L, R)
*/
// Either globally or in a single class:
static char buf[450 << 20];
void* operator new(size_t s) {
        static size_t i = sizeof buf;
        assert(s < i);
        return (void*)&buf[i -= s];
}
void operator delete(void*) {}

const int inf = 1e9;
struct Node {
        Node *l = 0, *r = 0;
        ll lo, hi, mset = inf, madd = 0,
            val = -inf;
        Node(int lo,int
            hi):lo(lo),hi(hi){} // Large
            interval of -inf
        Node(vll& v, int lo, int hi) :
            lo(lo), hi(hi) {
                if (lo + 1 < hi) {
                        int mid = lo + (hi
                            - lo)/2;
                        l = new Node(v,
                            lo, mid); r =
                            new Node(v,
                            mid, hi);
```

```
                val = max(l->val,
                    r->val);
        }
        else val = v[lo];
}
ll query(int L, int R) {
        if (R <= lo || hi <= L)
            return -inf;
        if (L <= lo && hi <= R)
            return val;
        push();
        return max(l->query(L, R),
            r->query(L, R));
}
void set(int L, int R, ll x) {
        if (R <= lo || hi <= L)
            return;
        if (L <= lo && hi <= R)
            mset = val = x, madd =
            0;
        else {
                push(), l->set(L,
                    R, x),
                    r->set(L, R, x);
                val = max(l->val,
                    r->val);
        }
}
void add(int L, int R, ll x) {
        if (R <= lo || hi <= L)
            return;
        if (L <= lo && hi <= R) {
                if (mset != inf)
                    mset += x;
                else madd += x;
                val += x;
        }
        else {
                push(), l->add(L,
                    R, x),
                    r->add(L, R, x);
                val = max(l->val,
                    r->val);
        }
    }
    void push() {
        if (!l) {
            int mid = lo + (hi
                - lo)/2;
            l = new Node(lo,
                mid); r = new
                Node(mid, hi);
        }
        if (mset != inf)
            l->set(lo,hi,mset),
                r->set(lo,hi,mset),
                mset = inf;
        else if (madd)
            l->add(lo,hi,madd),
                r->add(lo,hi,madd),
                madd = 0;
    }
};
```

1.4 treap

```
struct Node {
    char val;
    int weight, size;
    Node *left, *right;
    Node(char c) : val(c),
        weight(rand()), size(1),
        left(NULL), right(NULL) {}
} *root;

inline int size(Node *treap) {
    return treap ? treap->size : 0;
}
void split(Node *treap, Node *&left,
    Node *&right, int val) {
    if (!treap) {
        left = right = NULL;
        return;
    }
    if (size(treap->left) < val) {
        split(treap->right, treap->right,
            right, val -
            size(treap->left) - 1);
        left = treap;
    } else {
        split(treap->left, left,
            treap->left, val);
        right = treap;
    }
    treap->size = 1 + size(treap->left)
        + size(treap->right);
}
void merge(Node *&treap, Node *left,
    Node *right) {
    if (left == NULL) {
        treap = right;
        return;
    }
    if (right == NULL) {
```

```
        treap = left;
        return;
    }

    if (left->weight < right->weight) {
        merge(left->right, left->right,
            right);
        treap = left;
    } else {
        merge(right->left, left,
            right->left);
        treap = right;
    }
    treap->size = 1 + size(treap->left)
        + size(treap->right);
}
ostream& operator<<(ostream &os, Node
    *n) {
    if (!n) return os;
    os << n->left; os << n->val; os <<
        n->right;
    return os;
}
void solve() { // USAGE:
    // get integer n, q, and s
    for(auto c: s) merge(root, root, new
        Node(c));
    while(q--) {
        int l, r; cin >> l >> r;
        Node *a, *b;
        split(root, a, b, l - 1);
        Node *c, *d;
        split(b, c, d, r - l + 1);
        merge(root, a, d);
        merge(root, root, c);
    }
    cout << root << nl;
}
```

1.5 treapLazy

```
struct Node {
    char val;
    int weight, size;
    Node *left, *right;
    int toinvert;
    Node(char c) : val(c),
        weight(rand()), size(1),
        left(NULL), right(NULL),
        toinvert(0) {}
} *root;
inline int size(Node *treap) {
    return treap ? treap->size : 0;
}
void push(Node *treap) {
    if(treap == NULL) return;
    if(treap->toinvert == 0) return;
    Node *temp = treap->left;
    treap->left = treap->right;
    treap->right = temp;
    treap->toinvert = 0;
    if(treap->left != NULL)
        treap->left->toinvert ^= 1;
    if(treap->right != NULL)
        treap->right->toinvert ^= 1;
}
void split(Node *treap, Node *&left,
    Node *&right, int val) {
    if (!treap) {
        left = right = NULL; return;
    }
    push(treap);
    if (size(treap->left) < val) {
        split(treap->right, treap->right,
            right, val -
            size(treap->left) - 1);
        left = treap;
    } else {
        split(treap->left, left,
            treap->left, val);
        right = treap;
    }
    treap->size = 1 + size(treap->left)
        + size(treap->right);
}
void merge(Node *&treap, Node *left,
    Node *right) {
    if (left == NULL)
        treap = right; return;
    if (right == NULL)
        treap = left; return;
    push(left); push(right);
    if (left->weight < right->weight) {
        merge(left->right, left->right,
            right); treap = left;
    } else {
        merge(right->left, left,
            right->left); treap = right;
    }
    treap->size = 1 + size(treap->left)
        + size(treap->right);
}
void solve() { //USAGE:
    //get integers n,q and string s
```

```
    for(auto c: s) merge(root, root, new
        Node(c));
    while(q--) {
        int l, r; cin >> l >> r;
        Node *a, *b;
        split(root, a, b, l - 1);
        Node *c, *d;
        split(b, c, d, r - l + 1);
        c->toinvert ^= 1;
        merge(root, a, c);
        merge(root, root, d);
    }
    cout << root << nl;
}
```

2 DP-Optimizations

2.1 convexhull

```
const ll is_query = -(1LL<<62);
struct line {
        ll m, b;
        mutable function<const line*()>
            succ;
        bool operator<(const line& rhs)
            const {
                if (rhs.b != is_query)
                    return m < rhs.m;
                const line* s = succ();
                if (!s) return 0;
                ll x = rhs.m;
                return b - s->b < (s->m -
                    m) * x;
        }
};
struct dynamic_hull : public
    multiset<line> { // will maintain
    upper hull for maximum
        const ll inf = LLONG_MAX;
        bool bad(iterator y) {
                auto z = next(y);
                if (y == begin()) {
                        if (z == end())
                            return 0;
                        return y->m ==
                            z->m && y->b <=
                            z->b;
                }
                auto x = prev(y);
                if (z == end()) return
                    y->m == x->m && y->b
                    <= x->b;
                /* compare two lines by
                    slope, make sure
                    denominator is not 0 */
                ll v1 = (x->b - y->b);
                if (y->m == x->m) v1 =
                    x->b > y->b ? inf :
                    -inf;
                else v1 /= (y->m - x->m);
                ll v2 = (y->b - z->b);
                if (z->m == y->m) v2 =
                    y->b > z->b ? inf :
                    -inf;
                else v2 /= (z->m - y->m);
                return v1 >= v2;
        }
        void insert_line(ll m, ll b) {
                auto y = insert({ m, b });
                y->succ = [=] { return
                    next(y) == end() ? 0 :
                    &*next(y); };
                if (bad(y)) { erase(y);
                    return; }
                while (next(y) != end() &&
                    bad(next(y)))
                    erase(next(y));
                while (y != begin() &&
                    bad(prev(y)))
                    erase(prev(y));
        }
        ll eval(ll x) { //maximum at
            point x
                auto l =
                    *lower_bound((line) {
                    x, is_query });
                return l.m * x + l.b;
        }
};
```

2.2 divideNconquer

```
//dp[i][j]  = min  of
    dp[ i - 1 ][k]  + C [k][j] forall k
    < j,
//and optk[i][j] <= optk[i][j+1]
//optk is optimial k that gives you
    answer.
// compute dp_cur[l], ... dp_cur[r]
    (inclusive)
// C(a,c) + C(b,d) <= C(a,d) + C(b,c)
    for all a<=b<=c<=d
```

```
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);
}
// notebook-generator ./ --author
    "CatsOnTrees" --initials UTP --size
    11 --columns 3 --paper a4paper
```

2.3 knuth

```
//dp[i][j] = min { dp[i][k] + dp[k][j] +
    C(i, j) } : for i <= k < j
// O(n^3) -> O(n^2) if optk(i,j-1) <=
    opt(i,j) <= opt(i+1,j)
// criteria to see:
/* for a<=b<=c<=d
1) C(b,c) <= C(a,d)
2) C(a,c) + C(b,d) <= C(a,d) + C(b,c)
eg: C(i,j) = cost of arr[i..j] if all
    elements +ve */
```

```
int solve() {
    int N;
    ... // read N and input
    int dp[N][N], opt[N][N];
    auto C = [&](int i, int j) {
        ... // Implement cost function C.
    };
    for (int i = 0; i < N; i++) {
        opt[i][i] = i;
        ... // Initialize dp[i][i]
            according to the problem
    }
    for (int i = N-2; i >= 0; i--) {
        for (int j = i+1; j < N; j++) {
            int mn = INT_MAX;
            int cost = C(i, j);
            for (int k = opt[i][j-1]; k
                <= min(j-1, opt[i+1][j]);
                k++) {
                if (mn >= dp[i][k] +
                    dp[k+1][j] + cost) {
                    opt[i][j] = k;
                    mn = dp[i][k] +
                        dp[k+1][j] + cost;
                }
            }
            dp[i][j] = mn;
        }
    }
    cout << dp[0][N-1] << endl;
}
```

2.4 matrixExponentiation

```
vector<vector<int>> indentity(int n) {
    vector<vector<int>> i(n,
        vector<int>(n, 0));
    for(int j = 0; j < n; j++) {
        i[j][j] = 1;
    }
    return i;
}
ll mod_mul(ll a, ll b){
    a = a%M; b = b%M;
    return ((a*b)%M + M)%M;
}
vector<vector<int>>
    mul(vector<vector<int>>& a,
    vector<vector<int>>& b, int mod) {
    int n = sz(a);
    vector<vector<int>> toreturn(n,
        vector<int>(n, 0));
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            int ans = 0;
            for(int k = 0; k < n; k++) {
                ans += mod_mul(a[i][k] ,
                    b[k][j]);
                ans = ans%mod;
            }
            toreturn[i][j] = ans;
        }
    }
    return toreturn;
}
vector<vector<int>>
    expo(vector<vector<int>> &mat, int
    pow, int mod) {
```

```
    int n = mat.size();
    if(pow == 0) return indentity(n);
    vector<vector<int>> temp =
        indentity(n);
    auto Exp = expo(mat, pow/2, mod);
    if(pow % 2) {
        temp = mul(temp, mat, mod);
        vector<vector<int>> result = Exp;
        result = mul(result, result, mod);
        temp = mul(temp, result, mod);
    } else {
        vector<vector<int>> result = Exp;
        result = mul(result, result, mod);
        temp = mul(temp, result, mod);
    }
    return temp;
}
```

2.5 sosDP

```
for(int i = 0; i<(1<<N); ++i)F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask =
    0; mask < (1<<N); ++mask){
        if(mask & (1<<i)) F[mask] +=
            F[mask^(1<<i)];
}
```

3 Geometry

3.1 point2d

```
struct point2d {
    ll x, y;
    point2d() {}
    point2d(ll x, ll y): x(x), y(y) {}
point2d& operator+=(const point2d &t) {
    x += t.x; y += t.y;
    return *this;
}
point2d& operator-=(const point2d &t) {
    x -= t.x; y -= t.y;
    return *this;
}
point2d& operator*=(ll t) {
    x *= t; y *= t;
    return *this;
}
point2d& operator/=(ll t) {
    x /= t; y /= t;
    return *this;
}
point2d operator+(const point2d &t)
    const {
    return point2d(*this) += t;
}
point2d operator-(const point2d &t)
    const {
    return point2d(*this) -= t;
}
point2d operator*(ll t) const {
    return point2d(*this) *= t;
}
point2d operator/(ll t) const {
    return point2d(*this) /= t;
}
long long cross(const point2d& p) const
{ return x * p.y - y * p.x; }
long long cross(const point2d& a, const
    point2d& b) const
{ return (a - *this).cross(b - *this); }
};
point2d operator*(ll a, point2d b) {
    return b * a;
}
```

3.2 point3d

```
struct point3d {
    ftype x, y, z; //replace ftype
    point3d() {}
    point3d(ftype x, ftype y, ftype z):
        x(x), y(y), z(z) {}
point3d& operator+=(const point3d &t) {
    x += t.x; y += t.y; z += t.z;
    return *this;
}
point3d& operator-=(const point3d &t) {
    x -= t.x; y -= t.y; z -= t.z;
    return *this;
}
point3d& operator*=(ftype t) {
    x *= t; y *= t; z *= t;
    return *this;
}
point3d& operator/=(ftype t) {
    x /= t; y /= t; z /= t;
    return *this;
}
point3d operator+(const point3d &t)
    const {
```

```
    return point3d(*this) += t;
}
point3d operator-(const point3d &t)
    const {
    return point3d(*this) -= t;
}
point3d operator*(ftype t) const {
    return point3d(*this) *= t;
}
point3d operator/(ftype t) const {
    return point3d(*this) /= t;
}
};
point3d operator*(ftype a, point3d b) {
    return b * a;
}
```

3.3 Useful fnx

```
ftype dot(point2d a, point2d b) {
return a.x * b.x + a.y * b.y;
}
ftype dot(point3d a, point3d b) {
return a.x * b.x + a.y * b.y + a.z * b.z;
}
point3d cross(point3d a, point3d b) {
    return point3d(a.y * b.z - a.z * b.y,
        a.z * b.x - a.x * b.z,
        a.x * b.y - a.y * b.x);
}
ftype triple(point3d a, point3d b,
    point3d c) {
    return dot(a, cross(b, c));
}
ftype cross(point2d a, point2d b) {
    return a.x * b.y - a.y * b.x;
}
// line intersection:
point2d intersect(point2d a1, point2d
    d1, point2d a2, point2d d2) {
    return a1 + cross(a2 - a1, d2) /
        cross(d1, d2) * d1;
}
// plane intersection:
point3d intersect(point3d a1, point3d
    n1, point3d a2, point3d n2, point3d
    a3, point3d n3) {
    point3d x(n1.x, n2.x, n3.x);
    point3d y(n1.y, n2.y, n3.y);
    point3d z(n1.z, n2.z, n3.z);
    point3d d(dot(a1, n1), dot(a2, n2),
        dot(a3, n3));
    return point3d(triple(d, y, z),
    triple(x, d, z),
    triple(x, y, d)) / triple(n1, n2,
        n3);
}

//line segment intersection (a,b), (c,d):
int sgn(const long long& x)
{ return x >= 0 ? x ? 1 : 0 : -1; }
bool inter1(ll a, ll b, ll c, ll d) {
    if (a > b) swap(a, b);
    if (c > d) swap(c, d);
    return max(a, c) <= min(b, d);
}
bool check_inter(const pt& a, const pt&
    b, const pt& c, const pt& d) {
    if (cross(a-c, d-c) == 0 &&
        cross(b-c, d-c) == 0)
     return inter1(a.x, b.x, c.x, d.x) &&
         inter1(a.y, b.y, c.y, d.y);
    return sgn(cross(b-a, c-a)) !=
        sgn(cross(b-a, d-a)) &&
     sgn(cross(d-c, a-c)) !=
         sgn(cross(d-c, b-c));
}
```

4 Graphs

4.1 2sat

```
/*
(x     y)     ( x      y) so add edge
    between ~x ---> y and ~y ---> x
id[x] < id[x] => x = false
Both variables must have the same value
    is equivalent to:
( x      y)   (x      y ).
*/
struct two_sat { int n;
    vector<vector<int>> g, gr; // gr is
        the reversed graph
    vector<int> comp, topological_order,
        answer; // comp[v]: ID of the SCC
        containing node v
    vector<bool> vis;
    two_sat() {}
    two_sat(int _n) { init(_n); }
    void init(int _n) {
        n = _n;
```

```
    g.assign(2 * n, vector<int>());
    gr.assign(2 * n, vector<int>());
    comp.resize(2 * n);
    vis.resize(2 * n);
    answer.resize(2 * n);
}
void add_edge(int u, int v) {
    g[u].push_back(v);
    gr[v].push_back(u);
}
// For the following three functions
// int x, bool val: if 'val' is
    true, we take the variable to be
    x. Otherwise we take it to be x's
    complement.
// At least one of them is true
void add_clause_or(int i, bool f,
    int j, bool g) {
    add_edge(i + (f ? n : 0), j + (g
        ? 0 : n));
    add_edge(j + (g ? n : 0), i + (f
        ? 0 : n));
}
// Only one of them is true
void add_clause_xor(int i, bool f,
    int j, bool g) {
    add_clause_or(i, f, j, g);
    add_clause_or(i, !f, j, !g);
}
// Both of them have the same value
void add_clause_and(int i, bool f,
    int j, bool g) {
    add_clause_xor(i, !f, j, g);
}
// Topological sort
void dfs(int u) {
    vis[u] = true;
    for (const auto &v : g[u])
        if (!vis[v]) dfs(v);
    topological_order.push_back(u);
}
// Extracting strongly connected
    components
void scc(int u, int id) {
    vis[u] = true;
    comp[u] = id;
    for (const auto &v : gr[u])
        if (!vis[v]) scc(v, id);
}
// Returns true if the given
    proposition is satisfiable and
    constructs a valid assignment
bool satisfiable() {
    fill(vis.begin(), vis.end(),
        false);
    for (int i = 0; i < 2 * n; i++)
        if (!vis[i]) dfs(i);
    fill(vis.begin(), vis.end(),
        false);
    reverse(topological_order.begin(),
        topological_order.end());
    int id = 0;
    for (const auto &v :
        topological_order)
        if (!vis[v]) scc(v, id++);
    // Constructing the answer
    for (int i = 0; i < n; i++) {
        if (comp[i] == comp[i + n])
            return false;
            answer[i] = (comp[i] > comp[i
                + n] ? 1 : 0);
        }
        return true;
    }
};
```

4.2 bellmanford

```
struct edge
{ int a, b, cost;};
int n, m, v;
vector<edge> e;
const int INF = 1000000000;
void solve(){
    vector<int> d (n, INF);
    d[v] = 0;
    vector<int> p (n, - 1);
    int x;
for (int i=0; i<n; ++i){
        x = -1;
    for (int j=0; j<m; ++j)
    if (d[e[j].a] < INF)
    if (d[e[j].b] > d[e[j].a] +
        e[j].cost)
    {
     d[e[j].b] = max (-INF, d[e[j].a] +
         e[j].cost);
     p[e[j].b] = e[j].a;
     x = e[j].b;
    }
}
    if (x == -1)
```

```
        cout << "No negative cycle from "
            << v;
    else
    {
    int y = x;
    for (int i=0; i<n; ++i)
        y = p[y];
    vector<int> path;
    for (int cur=y; ; cur=p[cur])
    {
     path.push_back (cur);
     if (cur == y && path.size() > 1)
        break;
     }
     reverse (path.begin(), path.end());
    cout << "Negative cycle: ";
    for (size_t i=0; i<path.size(); ++i)
     cout << path[i] << ' ';
    }
}
```

4.3 dijkstra

```
vector<ll> dist;
vector<bool> vis;
//Single source shortest path algorithm
void dijkstra(vvpll& graph, ll start){
        int n = graph.size();
        vis.assign(n,false);
        dist.assign(n,1e18);
        //priority queue stores distance
            , current
        priority_queue<pair<ll,ll>, vpll,
            greater<pair<ll,ll>>> peq;
        dist[start] = 0;
        vis[start] = 0;
        peq.push(MP(0,start));
        while(!peq.empty()){
                ll curr = peq.top().S;
                ll currdist = peq.top().F;
                peq.pop();
                if(vis[curr]) continue;
                vis[curr] = true;
                //update all the children
                for(auto cpx: graph[curr]){
                        if(vis[cpx.F])
                            continue;
                        ll newDist =
                            currdist+cpx.S;
                        //relaxation
                        if(dist[cpx.F] >
                            newDist){
                        dist[cpx.F] =
                            newDist;
                        peq.push(MP(newDist,
                            cpx.F));
                        }
                }}}
```

4.4 dinics

```
struct Dinics{
        struct Edge{
                int to, revidx;
                ll cap, ocap;
                Edge(int to, int revidx,
                    ll cap, ll
                    ocap):to(to),
                    revidx(revidx),
                    cap(cap), ocap(ocap){}
                Edge(){}
                ll flow(){
                 return max(ocap - cap,
                     0ll);
                }
};
vector<vector<Edge>> adj;
vector<int> level, next;
int n;
Dinics(int n):n(n){
        level.assign(n, 0),
            next.assign(n,0);
        adj.assign(n,
            vector<Edge>(0));
}
void addEdge(int u, int v, ll
    cap, ll rev = 0){
        adj[u].push_back(Edge(v,
            adj[v].size(), cap,
            cap));
        adj[v].push_back(Edge(u,
            adj[u].size() - 1,
            rev, rev));
}
ll dfs(int curr, int t, ll flow){
        // cout<<curr<<" "<<t<<"
            "<<flow<<"\n";
        if(curr == t || !flow)
            return flow;
        for(int& i = next[curr]; i
            < adj[curr].size();
            i++){
         Edge &edge = adj[curr][i];
```

```
            if(level[edge.to] !=
                level[curr]+1)
                continue;
            ll actualflow;
            actualflow = dfs(edge.to,
                t, min(flow,
                edge.cap));
            if(actualflow){
             edge.cap -= actualflow;
           adj[edge.to][edge.revidx].cap
               += actualflow;
             return actualflow;
            }
           }
           return 0;
}
ll calc(int src, int t){
           ll flow = 0;
           const ll inf = 1e16;
           //capacity scaling
           for(int L = 30; L >= 0;
               L--){ do{
           level.assign(n, 0);
           next.assign(n, 0);
           //level assignment
           queue<int> q;
           level[src] = 1;
           q.push(src);
           while(!q.empty() &&
               !level[t]){
           int curr = q.front();
               q.pop();
           for(int i = 0; i <
               adj[curr].size(); i++){
           Edge &e = adj[curr][i];
               if(!level[e.to] && (e.cap
                   >> L)){
                level[e.to] = level[curr]
                    + 1;
                q.push(e.to);
               }}}
               //flows
               ll curflow;
               while(curflow = dfs(src,
                   t, inf)) flow +=
                   curflow;
               } while(level[t] != 0);
               }
               return flow;
           }
};
```

5 Maths

5.1 crt

```
/**
 * Find z such that z % x[i] = a[i] for
     all i.
 * */
long long crt(vector<long long> &a,
    vector<long long> &x) {
  long long z = 0;
  long long n = 1;
  for (int i = 0; i < x.size(); ++i)
    n *= x[i];
  for (int i = 0; i < a.size(); ++i) {
    long long tmp = (a[i] * (n / x[i]))
        % n;
    tmp = (tmp * mod_inv(n / x[i],
        x[i])) % n;
    z = (z + tmp) % n;
  }
  return (z + n) % n;
}
```

5.2 eulerToitientNlog(logn))

```
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
     phi[i] = i;
    for (int i = 2; i <= n; i++) {
     if (phi[i] == i) {
         for (int j = i; j <= n; j += i)
          phi[j] -= phi[j] / i;
         }
    }}
```

5.3 eulerToitientRootn

```
// counts the number of integers between
    1 and n inclusive, which are coprime
    to n
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
         while (n % i == 0)
```

```
            n /= i;
         result -= result / i;
        }
    }
    if (n > 1) result -= result / n;
    return result;
}
```

5.4 extendedgcd

```
int gcd(int a, int b, int& x, int& y) {
    x = 1, y = 0;
    int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        int q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q
            * x1);
        tie(y, y1) = make_tuple(y1, y - q
            * y1);
        tie(a1, b1) = make_tuple(b1, a1 -
            q * b1);
    } return a1;
}
int modinv(int a, int m){
        int x, y;
        int g = gcd(a, m, x, y);
        if(g != 1) return -100;
        else return (x%m + m)%m;
}
```

5.5 linearSieve

```
const int maxn = 100000;
int lp[maxn+1], mobius[maxn+1];
int twopow[maxn+1];
vector<int> primes(0);
void lsieve(){
        lp[1] = 1;
        mobius[1] = 1;
        for(int i = 2; i <= maxn; i++){
                if(lp[i] == 0){
                 lp[i] = i;
                 primes.push_back(i);
                }
                for(int j = 0;
                    (j<primes.size()) &&
                    primes[j]<=lp[i] &&
                    i*primes[j] <= maxn;
                    j++){
                 lp[i*primes[j]] =
                     primes[j];
                }
                if(lp[i] == i) mobius[i] =
                    -1;
                else {
                 int x = i/lp[i];
                 if(x%lp[i] == 0)
                     mobius[i] = 0;
                 else mobius[i] =
                     mobius[x]*mobius[lp[i]];
                }
        }}
```

5.6 modInt

```
struct mi {
ll v; explicit operator ll() const {
    return v % mod; }
mi() { v = 0; }
mi(ll _v) {
    v = (-mod < _v && _v < mod) ? _v
        : _v % mod;
    if (v < 0) v += mod;
}
friend bool operator==(const mi& a,
    const mi& b) {
    return a.v == b.v; }
friend bool operator!=(const mi& a,
    const mi& b) {
    return !(a == b); }
friend bool operator<(const mi& a,
    const mi& b) {
    return a.v < b.v; }
mi& operator+=(const mi& m) {
    if ((v += m.v) >= mod) v -= mod;
    return *this; }
mi& operator-=(const mi& m) {
    if ((v -= m.v) < 0) v += mod;
    return *this; }
mi& operator*=(const mi& m) {
    v = v*m.v%mod; return *this; }
mi& operator/=(const mi& m) { return
    (*this) *= inv(m); }
friend mi pow(mi a, ll p) {
    mi ans = 1; assert(p >= 0);
    for (; p; p /= 2, a *= a) if
        (p&1) ans *= a;
    return ans;
}
friend mi inv(const mi& a) {
    assert(a.v != 0);
```

```
        return pow(a,mod-2); }

    mi operator-() const { return
        mi(-v); }
    mi& operator++() { return *this +=
        1; }
    mi& operator--() { return *this -=
        1; }
    mi operator++(int) { mi temp; temp.v
        = v++; return temp; }
    mi operator--(int) { mi temp; temp.v
        = v--; return temp; }
    friend mi operator+(mi a, const mi&
        b) { return a += b; }
    friend mi operator-(mi a, const mi&
        b) { return a -= b; }
    friend mi operator*(mi a, const mi&
        b) { return a *= b; }
    friend mi operator/(mi a, const mi&
        b) { return a /= b; }
    friend ostream& operator<<(ostream&
        os, const mi& m) {
        os << m.v; return os;
    }
    friend istream& operator>>(istream&
        is, mi& m) {
        ll x; is >> x;
        m.v = x;
        return is;
    }
};
#define vm vector<mi>
```

6 strings

6.1 manacher

```
/*
 * For each position in a string,
    computes p[0][i] = half length of
 * longest even palindrome around pos i,
 * p[1][i] = longest odd (half rounded
    down). where i is right center
    (verify)
 * O(n)
*/
vi<vi> manacher(const string& s) {
        int n = sz(s);
        vi<vi> p = {vi(n+1), vi(n)};
        rep(z,0,2) for (int i=0,l=0,r=0;
            i < n; i++) {
                int t = r-i+!z;
                if (i<r) p[z][i] = min(t,
                    p[z][l+t]);
                int L = i-p[z][i], R =
                    i+p[z][i]-!z;
                while (L>=1 && R+1<n &&
                    s[L-1] == s[R+1])
                        p[z][i]++, L--,
                            R++;
                if (R>r) l=L, r=R;
        }
        return p;
}
```

6.2 trie

```
typedef struct trie{
        typedef struct node{
                node* nxt[2]; int cnt = 0;
                node(){
                 nxt[0] = nxt[1] = NULL;
                 cnt = 0;
                }
        }Node;
        Node* head;
        trie() { head = new Node(); }
        void insert(int x){
                Node* cur = head;
                for(int i = 30; i >= 0;
                    i--){
                 int b = (x >> i) & 1;
                 if(!cur -> nxt[b])
                        cur -> nxt[b] =
                            new Node();
                 cur = cur -> nxt[b];
                 cur -> cnt++;
                }
        }
        void remove(int x){
                Node* cur = head;
                for(int i = 30; i >= 0;
                    i--){
                 int b = (x >> i) & 1;
                 cur = cur -> nxt[b];
                 cur -> cnt--;
                }
        }
        int maxxor(int x){
                Node* cur = head;
                int ans = 0;
```

```
            for(int i = 30; i >= 0;
                i--){
             int b = (x >> i) & 1;
             if(cur -> nxt[!b] && cur
                -> nxt[!b] -> cnt > 0)
            {
             ans += (1LL << i);
             cur = cur -> nxt[!b];
            }
            else
             cur = cur -> nxt[b];
            }
            return ans;
        }
}Trie;
```

6.3 Z algo

```
// z[i] is the length of the longest
    string that is,
// at the same time, a prefix of s and
// a prefix of the suffix of s starting
    at i
vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n;
        ++i) {
        if (i <= r)
         z[i] = min (r - i + 1, 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;
    } // for number of occurences of t
        in s
    return z; //str = s+$+t
}
```

7 Trees

7.1 centroid decomposition

```
vvll tree;
//return distance between a and b
int dist(int a, int b){
        // return depth[a] + depth[b] -
            2*depth[lca];
}
//find depth and parent
void dfs1(int v, int p, int d = 0){
        depth[v] = d;
        for(ll &nbr: tree[v])
        {if(nbr == p) continue; dfs1(nbr,
            v, d+1);}
}
vvll centroidTree; //actual tree
int croot; //centroid root
vector<bool> vis; //helper
vll cpar; vll sz;
//construct centroid tree
int markSizeOfUnvisited(int v, int p =
    -1){
        sz[v] = 1;
        for(int nbr: tree[v]){
                if(!vis[nbr] && nbr != p){
                sz[v] +=
                markSizeOfUnvisited(nbr,
                    v);
                }
        }
        return sz[v];
}
int findCentroid(int v, int p, int num){
        for(int nbr: tree[v]){
                if(!vis[nbr] && nbr != p
                    && sz[nbr] > num/2){
                 return findCentroid(nbr,
                    v, num);
                }
        }
        return v;
}
void constructCentroidTree(int v, int p){
        markSizeOfUnvisited(v, p);
        int cvex = findCentroid(v, p,
            sz[v]);
        if(p == -1) croot = cvex;
        else{
                cpar[cvex] = p;
                centroidTree[cvex].push_back(p);
                centroidTree[p].push_back(cvex);
        }
        vis[cvex] = true;
        for(int nbr: tree[cvex]){
                if(vis[nbr]) continue;
                constructCentroidTree(nbr,
                    cvex);
        }
}
vll opt;
```

```
int query(int vx){ int res = 1e8;
        for(int px = vx; px != -1; px =
            cpar[px]){
                res = min(res,
                    (int)opt[px] +
                    dist(px, vx));
        } return res;
}
void update(int vx){
        for(int px = vx; px != -1; px =
            cpar[px]){
                opt[px] =
                    min((int)opt[px],
                    dist(vx, px));
        }
}
void init(int n){
        tree.assign(n, vll(0));
        centroidTree.assign(n, vll(0));
        cpar.assign(n, -1); sz.assign(n,
            0);
        vis.assign(n, false);
}
void solve(){
        int n, m; cin>>n>>m; init(n);
        for(int i = 0; i < n-1; i++){
                int u, v; cin>>u>>v; u--;
                    v--;
                tree[u].push_back(v);
                tree[v].push_back(u);
        }
        constructCentroidTree(0,-1);
            //constructs tree
        // other dfs stuff
        const int inf = 1e8;
        opt.clear(); opt.resize(n, inf);
        update(0); //color node 0
        for(int i = 0; i < m; i++){
                int type; cin>>type; int
                    vx; cin>>vx; vx--;
                if(type == 1) update(vx);
                else cout<<query(vx)<<"\n";
        }
}
```

7.2 hld

```
//import lca
//import segment tree
const int maxn = 200005;
vvll tree;
vvll par;
ll sz[maxn], pos[maxn], moola[maxn],
    depth[maxn];
ll heavy[maxn], chain[maxn];
int num = 0;
//assigns first parent, subtree size,
    heavy child and depth
int dfs(int curr, int p, int d = 0){
        par[curr][0] = p; sz[curr] = 1;
        depth[curr] = d;
        int maxchild = -1, msize = 0;
        for(auto &child: tree[curr]){
                if(child == p) continue;
                sz[curr] += dfs(child,
                    curr, d+1);
                if(sz[child] > msize){
                        maxchild = child;
                            msize =
                                sz[child];
                }
        }
        heavy[curr] = maxchild;
        return sz[curr];
}
//assign pos (in segtree), and chaintop
void decompose(int curr, int p, bool
    isheavy = false){
        pos[curr] = num++;
        if(isheavy == true) chain[curr] =
            chain[p];
        else chain[curr] = curr;

        if(heavy[curr] != -1){
                decompose(heavy[curr],
                    curr, true);
        }
        for(auto &child: tree[curr]){
                if(child == p || child ==
                    heavy[curr]) continue;
                decompose(child, curr,
                    false);
        }
}

int query(Node* stree, int from, int p){
        int res = 0;
        while(chain[from] != chain[p]){
                int top = pos[chain[from]];
                int till = pos[from];
                res =
                    max(stree->query(top,
                    till+1), res);
                //jump to the above one
```

```
            from = chain[from];
            from = par[from][0];
        }
        int top = pos[p];
        int till = pos[from];
        res = max(res, stree->query(top,
            till+1));
        return res;
}
//call dfs(0,-1) then
    decompose(0,-1,false)
//then query(node, child, lca), segtree
    is of size n (vertices)
```

7.3 lca

```
#define sz(x) (int)(x).size()
template<class T>
struct RMQ {
        vector<vector<T>> jmp;
        RMQ(const vector<T>& V) : jmp(1,
            V) {
                for (int pw = 1, k = 1; pw
                    * 2 <= sz(V); pw *= 2,
                    ++k) {
                 jmp.emplace_back(sz(V) -
                     pw * 2 + 1);
                 rep(j,0,sz(jmp[k]))
                  jmp[k][j] = min(jmp[k -
                      1][j], jmp[k - 1][j
                      + pw]);
                }
        }
        T query(int a, int b) {
            assert(a < b); // or
                return inf if a == b
            int dep = 31 -
                __builtin_clz(b - a);
            return min(jmp[dep][a],
                jmp[dep][b - (1 <<
                dep)]);
        }
};
struct LCA {
        int T = 0;
        vi time, path, ret;
        RMQ<int> rmq;
        //pass in adjacency list, 0-based
            tree, root at 0
        LCA(vector<vi>& C) : time(sz(C)),
            rmq((dfs(C,0,-1), ret)) {}
        void dfs(vector<vi>& C, int v,
            int par) {
                time[v] = T++;
                for (int y : C[v]) if (y
                    != par) {
                 path.push_back(v),
                     ret.push_back(time[v]);
                 dfs(C, y, v);
                }
        }
        int lca(int a, int b) {
                if (a == b) return a;
                tie(a, b) =
                    minmax(time[a],
                    time[b]);
                return path[rmq.query(a,
                    b)];
        }
        //dist(a,b){return depth[a] +
            depth[b] - 2*depth[lca(a,b)];}
};
```

8 X Extras

8.1 customHash

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t
        x) {
        //
            http://xorshift.di.unimi.it/splitmix
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) *
            0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) *
            0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM
=
    chrono::steady_clock::now().time_since_epoc
        return splitmix64(x +
            FIXED_RANDOM);
    }
};
```

8.2 dynamicConnectivity

```
// DSU with Rollback
vll par; vll sz;
vector<pll> hist; int cc = 0;
int findRoot(int v){
	while(par[v] != v) v = par[v];
	return v;
}
void unite(int u, int v){
	u = findRoot(u); v = findRoot(v);
	if(u != v){
		cc--;
		if(sz[u] > sz[v]) swap(u,
		v);
		hist.push_back({~v,
		sz[v]}); sz[v] +=
		sz[u];
		hist.push_back({u,
		par[u]}); par[u] = v;
	}}
void rollback(int save){
	while((int)hist.size() > save){
		int v = hist.back().first;
		int val =
		hist.back().second;
		hist.pop_back();
		if(v < 0){
			cc++; sz[~v] = val;
		} else {
			par[v] = val;
		}}}
int n, m, q;
void init(){
	par.assign(n, 0);
	iota(par.begin(), par.end(), 0);
	sz.assign(n, 1);cc = n; }
vll answer;
void dnc(int l, int r, vector<ppll>&
    redge){
	vector<ppll> partialoverlap;
	int save = hist.size();
	for(auto &val: redge){
		int cl = val.first.first;
		int cr = val.first.second;
		int u = val.second.first;
		int v = val.second.second;
		if(cr <= l || cl >
		r){/*nothing*/}
		else if(cl <= l && r < cr){
			unite(u, v);
		} else {
		partialoverlap.push_back(val);
		}
	}
	if(l == r){
		//answer query:
		answer[l] = cc;
	} else {
		int mid = (l+r)/2;
		dnc(l, mid,
		partialoverlap);
		dnc(mid+1, r,
		partialoverlap);
	}
	rollback(save);
}
int main(){
	cin>>n>>m>>q; init();//initialize
	dsu
	vector<pair<pll, ll>> edgelist;
	//already existing edges
	for(int i = 0; i < m; i++){
		int u,v;cin>>u>>v;u--;v--;
		if(u > v) swap(u, v);
		edgelist.push_back({{u,v},
		0});
	}
	answer.assign(q+1, -1);
	// if u,v then remove else add
	edge
	for(int i = 1; i <= q; i++){
		int u,v; cin>>u>>v; u--;
		v--;
		if(u > v) swap(u, v);
		edgelist.push_back({{u,v},
		i});
	}
	sort(edgelist.begin(),
	edgelist.end());
	int esize = edgelist.size();
	vector<ppll> rangeedge;
	for(int i = 0; i < esize; i++){
		auto &val = edgelist[i];
		int u = val.first.first, v
		= val.first.second;
		int aq = val.second; int
		en;
		if(i+1>=esize || val.first
		!=
		edgelist[i+1].first){
			en = q+1;
		} else {
			en =
			edgelist[i+1].second;
			i++;
```

```
            }
            rangeedge.push_back({{aq,en},
                {u,v}});
        }
        dnc(0,q, rangeedge);
        for(int i = 0; i <= q; i++){
            cout<<answer[i]<<" ";
                //number of cc
        }}
```

8.3 mathystuff

```
/*
Burnside lemma:
---------------
(1/n)*sum(c[0]...c[n-1])
where n ways to change positions and
    c[k] is
combinations that remain unchanged when
    k'th way applied
c[0] = total combinations usually

Derangement rec:
----------------
D[n] = (n-1)(D[n-1] + D[n-2])
D[1] = 0, D[2] = 1

Catalan Number
--------------
2n_C_n - 2n_C_(n+1)
C[n] = no. of binary trees of n nodes (l
    and r diff)
C[n] = no of trees of n+1 nodes (l and r
    same)

Fermats theorem
---------------
a^p = a (mod p)
(when a,p coprime)
so a^(p-1) mod p = 1

Wilson's Theorem
----------------
if p is prime then
(p-1)! = -1 or p-1 (mod p)

Game Theory
-----------
Parital Games - Eg: Chess, TicTacToe
Impartial - Moves only depend on state
    of the game.

Types of Impartial Games -
a) Normal Game: Player who plays the
    last move wins
b) Misere Game: Player who plays the
    last move loses

NIM GAME- Impartial Normal Game;
rules-
Stone piles: [a_1, a_2, a_3..... a_n]
In each turn, a player : choose one pile
    -> remove atleast one or more stones
The player who takes the last stone wins.
Alice moves first.
If XOR(a_1, a_2 .... a_n) == 0
    Bob wins
else
    Alice wins

What is Sprague-Grundy Theorem?
Suppose there is a composite game (more
    than one sub-game) made up of N
    sub-games and two players, A and B.
    Then Sprague-Grundy Theorem says that
    if both A and B play optimally (i.e.,
    they dont make any mistakes), then
    the player starting first is
    guaranteed to win if the XOR of the
    grundy numbers of position in each
    sub-games at the beginning of the
    game is non-zero. Otherwise, if the
    XOR evaluates to zero, then player A
    will lose definitely, no matter what.
We can apply Sprague-Grundy Theorem in
    any impartial game and solve it. The
    basic steps are listed as follows:
-Break the composite game into sub-games.
-Then for each sub-game, calculate the
    Grundy Number at that position.
-Then calculate the XOR of all the
    calculated Grundy Numbers.
-If the XOR value is non-zero, then the
    player who is going to make the turn
    (First Player) will win else he is
    destined to lose, no matter what.

 A Dynamic Programming
     (Memoization-based) approach to
 calculate Grundy Number of a Game
 Game Description-
 Just like a one-pile version of Nim,
     the game starts with
```

```
 a pile of n stones, and the player to
    move may take any
 positive number of stones.
 The last player to move wins. Which
    player wins the game?

A Function to calculate Mex of all the
   values in that set
This function remains same
*/
int calculateMex(unordered_set<int> Set)
    {
        int Mex = 0;
        while (Set.find (Mex) !=
            Set.end())
                Mex++;
        return (Mex);
}

// A function to Compute Grundy Number
   of 'n'
// Only this function varies according
   to the game
int calculateGrundy(int n, int Grundy[])
    {
        if (n == 0)
                return (0);
        if (Grundy[n] != -1)
                return (Grundy[n]);
        unordered_set<int> Set; // A Hash
            Table
        for (int i=0; i<=n-1; i++)
                Set.insert(calculateGrundy(i,
                    Grundy));
        // Store the result
        Grundy[n] = calculateMex (Set);
        return(Grundy[n]);
}
//Driver program to test above functions
int main() {
        int n = 10;
        // An array to cache the
            sub-problems so that
        // re-computation of same
            sub-problems is avoided
        int Grundy[n+1];
        memset (Grundy, -1,
            sizeof(Grundy));
        printf ("%d", calculateGrundy(n,
            Grundy));
        return (0);
}
```

8.4 random rng

```
  //include <random> and <chrono>
  mt19937
      rng(chrono::steady_clock::now().time_since_epoch().count());
//   mt19937 rng((uint64_t) new char);
```

8.5 submaskOfBitmask

```
for (int m=0; m<(1<<n); ++m)
    for (int s=m; s; s=(s-1)&m)
 ... s and m ...
```

8.6 template

```
#include <bits/stdc++.h>
using namespace std;
#define rep(i, a, b) for(int i = a; i <
    (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;
#define MOD1 1000000007
#define MOD2 1000000009
#define MOD3 1000000021
#define MOD4 1000000033
#define MOD5 1000000087
#define MOD6 1000000093
#define MOD7 1000000097
int main(){
        ios::sync_with_stdio(false);
        cin.tie(0);
        return 0;
}
```