Team notebook

HCMUS-IdentityImbalance

October 28, 2022

Contents

1 Algorithms

1.1 Mo's algorithm on trees

```
/*
--------
Problem:
--------
https://www.spoj.com/problems/COT2/
Given a tree with N nodes and Q queries. Each node
    has an integer weight.
Each query provides two numbers u and v, ask for
    how many different integers weight of nodes
there are on path from u to v.
```

```
----------
Modify DFS:
----------
For each node u, maintain the start and the end DFS
     time. Let's call them ST(u) and EN(u).
=> For each query, a node is considered if its
     occurrence count is one.

--------------
Query solving:
--------------
Let's query be (u, v). Assume that ST(u) <= ST(v).
     Denotes P as LCA(u, v).

Case 1: P = u
Our query would be in range [ST(u), ST(v)].

Case 2: P != u
Our query would be in range [EN(u), ST(v)] +
     [ST(p), ST(p)]
*/

void update(int &L, int &R, int qL, int qR){
    while (L > qL) add(--L);
    while (R < qR) add(++R);

    while (L < qL) del(L++);
    while (R > qR) del(R--);
}

vector <int> MoQueries(int n, vector <query> Q){
    block_size = sqrt((int)nodes.size());
    sort(Q.begin(), Q.end(), [](const query &A,
         const query &B){
        return (ST[A.l]/block_size !=
             ST[B.l]/block_size)?
             (ST[A.l]/block_size <
             ST[B.l]/block_size) : (ST[A.r] <
             ST[B.r]);
    });
    vector <int> res;
    res.resize((int)Q.size());

    LCA lca;
    lca.initialize(n);

    int L = 1, R = 0;
    for(query q: Q){
        int u = q.l, v = q.r;
        if(ST[u] > ST[v]) swap(u, v); // assume that
             S[u] <= S[v]
        int parent = lca.get(u, v);

        if(parent == u){
            int qL = ST[u], qR = ST[v];
            update(L, R, qL, qR);
        }else{
            int qL = EN[u], qR = ST[v];
            update(L, R, qL, qR);
            if(cnt_val[a[parent]] == 0)
                res[q.pos] += 1;
        }

        res[q.pos] += cur_ans;
    }
    return res;
}
```

1.2 Mo's algorithm

```
/*
    https://www.spoj.com/problems/FREQ2/
*/
vector <int> MoQueries(int n, vector <query> Q){

    block_size = sqrt(n);
    sort(Q.begin(), Q.end(), [](const query &A,
         const query &B){
        return (A.l/block_size != B.l/block_size)?
             (A.l/block_size < B.l/block_size) :
             (A.r < B.r);
    });
    vector <int> res;
    res.resize((int)Q.size());

    int L = 1, R = 0;
    for(query q: Q){
        while (L > q.l) add(--L);
        while (R < q.r) add(++R);

        while (L < q.l) del(L++);
        while (R > q.r) del(R--);

        res[q.pos] = calc(1, R-L+1);
    }
    return res;
}
```

1.3 parallel binary search

```
int lo[N], mid[N], hi[N];
vector<int> vec[N];

// Reset
void clear() { memset(bit, 0, sizeof(bit)); }

// Apply ith update/query
void apply(int idx) {
    if (ql[idx] <= qr[idx])
        update(ql[idx], qa[idx]), update(qr[idx] +
             1, -qa[idx]);
    else {
        update(1, qa[idx]);
        update(qr[idx] + 1, -qa[idx]);
        update(ql[idx], qa[idx]);
    }
}

// Check if the condition is satisfied
bool check(int idx) {
    int req = reqd[idx];
    for (auto &it : owns[idx]) {
        req -= pref(it);
        if (req < 0)
            break;
    }
    if (req <= 0)
        return 1;
    return 0;
}

void work() {
    for (int i = 1; i <= q; i++)
        vec[i].clear();
    for (int i = 1; i <= n; i++)
        if (mid[i] > 0)
            vec[mid[i]].push_back(i);
    clear();
    for (int i = 1; i <= q; i++) {
        apply(i);
        for (auto &it : vec[i]) // Add appropriate
             check conditions
        {
            if (check(it))
                hi[it] = i;
            else
                lo[it] = i + 1;
        }
    }
}

void parallel_binary() {
    for (int i = 1; i <= n; i++)
        lo[i] = 1, hi[i] = q + 1;
    bool changed = 1;
    while (changed) {
        changed = 0;
        for (int i = 1; i <= n; i++) {
            if (lo[i] < hi[i]) {
                changed = 1;
```

```
            mid[i] = (lo[i] + hi[i]) / 2;
        } else
            mid[i] = -1;
    }
    work();
  }
}
```

2 Data structures

2.1 dsu with undo

```
/**
 * Author: Lukas Polacek, Simon Lindholm
 * Date: 2019-12-26
 * License: CC0
 * Source: folklore
 * Description: Disjoint-set data structure with
     undo.
 * If undo is not needed, skip st, time() and
     rollback().
 * Usage: int t = uf.time(); ...; uf.rollback(t);
 * Time: O(log(N))
 */
#pragma once

struct RollbackUF {
  vi e; vector<pii> st;
  RollbackUF(int n) : e(n, -1) {}
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x :
      find(e[x]); }
  int time() { return sz(st); }
  void rollback(int t) {
    for (int i = time(); i --> t;)
      e[st[i].first] = st[i].second;
    st.resize(t);
  }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.emb(a, e[a]); st.emb(b, e[b]);
    e[a] += e[b]; e[b] = a;
    return true;
  }
};
```

2.2 dsu

```
class DSU{
public:
    vector <int> parent;
    void initialize(int n){
        parent.resize(n+1, -1);
    }

    int findSet(int u){
        while(parent[u] > 0)
            u = parent[u];
        return u;
    }

    void Union(int u, int v){
        int x = parent[u] + parent[v];
        if(parent[u] > parent[v]){
            parent[v] = x;
            parent[u] = v;
        }else{
            parent[u] = x;
            parent[v] = u;
        }
    }
};
```

2.3 fake fenwick tree update

```
vector <int> fake_bit[MAXN];

void fake_update(int x, int y, int limit_x){
    for(int i = x; i < limit_x; i += i&(-i))
        fake_bit[i].pb(y);
}

void fake_get(int x, int y){
    for(int i = x; i >= 1; i -= i&(-i))
        fake_bit[i].pb(y);
}

vector <int> bit[MAXN];

void update(int x, int y, int limit_x, int val){
    for(int i = x; i < limit_x; i += i&(-i)){
        for(int j = lower_bound(fake_bit[i].begin(),
             fake_bit[i].end(), y) -
             fake_bit[i].begin(); j <
             fake_bit[i].size(); j += j&(-j))
            bit[i][j] = max(bit[i][j], val);
    }
}

int get(int x, int y){
    int ans = 0;
    for(int i = x; i >= 1; i -= i&(-i)){
        for(int j = lower_bound(fake_bit[i].begin(),
             fake_bit[i].end(), y) -
             fake_bit[i].begin(); j >= 1; j -=
             j&(-j))
            ans = max(ans, bit[i][j]);
    }
    return ans;
}

int main(){
    int n; cin >> n;
    vector <int> Sx, Sy;
    for(int i = 1; i <= n; i++){
        cin >> a[i].fi >> a[i].se;
        Sx.pb(a[i].fi);
        Sy.pb(a[i].se);
    }
    compress(Sx);
    compress(Sy);
    // unique all value
    for(int i = 1; i <= n; i++){
        a[i].fi = lower_bound(Sx.begin(), Sx.end(),
             a[i].fi) - Sx.begin();
        a[i].se = lower_bound(Sy.begin(), Sy.end(),
             a[i].se) - Sy.begin();
    }

    // do fake BIT update and get operator
    for(int i = 1; i <= n; i++){
        fake_get(a[i].fi-1, a[i].se-1);
        fake_update(a[i].fi, a[i].se,
             (int)Sx.size());
    }

    for(int i = 0; i < Sx.size(); i++){
        fake_bit[i].pb(INT_MIN); // avoid zero
        sort(fake_bit[i].begin(), fake_bit[i].end());
        fake_bit[i].resize(unique(fake_bit[i].begin(),
             fake_bit[i].end()) -
             fake_bit[i].begin());
        bit[i].resize((int)fake_bit[i].size(), 0);
    }

    // real update, get operator
    int res = 0;
    for(int i = 1; i <= n; i++){
        int maxCurLen = get(a[i].fi-1, a[i].se-1) +
             1;
        res = max(res, maxCurLen);
        update(a[i].fi, a[i].se, (int)Sx.size(),
             maxCurLen);
    }
}
```

2.4 hash table

```
/*
 * Micro hash table, can be used as a set.
 *   Very efficient vs std::set
 *
 */

const int MN = 1001;
struct ht {
  int _s[(MN + 10) >> 5];
  int len;
  void set(int id) {
    len++;
    _s[id >> 5] |= (1LL << (id & 31));
  }
  bool is_set(int id) {
    return _s[id >> 5] & (1LL << (id & 31));
  }
};
```

2.5 heavy light decomposition

```
/*
 * Problem: Given a graph, there are 2 type of query
 * 1: update weight of vertex u
 * 2: find maximum weight of vertices from a to b
*/
const int N = 2e5+5;
const int D = 19;
const int S = (1<<D);

int n, q, v[N];
vector<int> adj[N];

int sz[N], p[N], dep[N];
int st[S], id[N], tp[N];

void update(int idx, int val) {
        st[idx += n] = val;
        for (idx /= 2; idx; idx /= 2)
                st[idx] = max(st[2 * idx], st[2 *
                        idx + 1]);
}

int query(int lo, int hi) {
        int ra = 0, rb = 0;
        for (lo += n, hi += n + 1; lo < hi; lo /= 2,
             hi /= 2) {
                if (lo & 1)
                        ra = max(ra, st[lo++]);
                if (hi & 1)
                        rb = max(rb, st[--hi]);
        }
        return max(ra, rb);
}

int dfs_sz(int cur, int par) {
        sz[cur] = 1;
        p[cur] = par;
        for(int chi : adj[cur]) {
                if(chi == par) continue;
                dep[chi] = dep[cur] + 1;
                p[chi] = cur;
                sz[cur] += dfs_sz(chi, cur);
        }
        return sz[cur];
}

int ct = 1;

void dfs_hld(int cur, int par, int top) {
        id[cur] = ct++;
        tp[cur] = top;
        update(id[cur], v[cur]);
        int h_chi = -1, h_sz = -1;
        for(int chi : adj[cur]) {
                if(chi == par) continue;
                if(sz[chi] > h_sz) {
                        h_sz = sz[chi];
                        h_chi = chi;
                }
        }
        if(h_chi == -1) return;
        dfs_hld(h_chi, cur, top);
        for(int chi : adj[cur]) {
                if(chi == par || chi == h_chi)
                        continue;
                dfs_hld(chi, cur, chi);
        }
}

int path(int x, int y){
        int ret = 0;
        while(tp[x] != tp[y]){
                if(dep[tp[x]] < dep[tp[y]])swap(x,y);
                ret = max(ret,
                        query(id[tp[x]],id[x]));
                x = p[tp[x]];
        }
        if(dep[x] > dep[y])swap(x,y);
        ret = max(ret, query(id[x],id[y]));
        return ret;
}

int main() {
        // input ...
        dfs_sz(1, 1);
        dfs_hld(1, 1, 1);
        // query
}
```

2.6 hull optimization

```
/**
 * Author: hieplpvip
 * Date: 2020-10-17
 * License: CC0
 * Source: own work
 * Description: Add line in decreasing slope, query
     in increasing x
 * Time: O(\log N)
 * Status: untested
 */
#pragma once

template <typename T = long long> struct MinHull {
    struct Line {
        T a, b;
        Line(T a, T b) : a(a), b(b) {}
        T calc(T x) { return a * x + b; }
    };
    vector<Line> dq;
    size_t seen;
    bool overlap(Line &p1, Line &p2, Line &p3) {
        return 1.0 * (p3.b - p1.b) / (p1.a - p3.a) <=
              1.0 * (p2.b - p1.b) / (p1.a - p2.a);
    }
    void addLine(T a, T b) {
        Line newLine(a, b);
        while (dq.size() > seen + 1 &&
              overlap(dq[(int)dq.size() - 2],
                  dq.back(), newLine))
            dq.pop_back();
        dq.pb(newLine);
    }
    T query(T x) {
        // change >= to <= this to get MaxHull
        while (seen + 1 < dq.size() &&
             dq[seen].calc(x) >= dq[seen +
             1].calc(x))
            ++seen;
        return dq[seen].calc(x);
    }
};
```

2.7 line container

```
/**
 * Author: Simon Lindholm
 * Date: 2017-04-20
 * License: CC0
 * Source: own work
 * Description: Container where you can add lines
     of the form kx+m, and query
 * maximum values at points x. Useful for dynamic
     programming (``convex hull
 * trick''). Time: O(\log N) Status: stress-tested
 */
#pragma once

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;
    }
};
```

2.8 order statistic tree

```
/**
 * Author: Simon Lindholm
 * Date: 2016-03-22
 * License: CC0
 * Source: hacKIT, NWERC 2015
 * Description: A set (not multiset!) with support
     for finding the n'th
 * element, and finding the index of an element.
 * To get a map, change \texttt{null\_type}.
 * Time: O(log N)
 */
#pragma once

#include <bits/extc++.h> /** keep-include */
using namespace __gnu_pbds;

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

void example() {
    Tree<int> t, t2;
    t.insert(8);
    auto it = t.insert(10).first;
    assert(it == t.lower_bound(9));
    assert(t.order_of_key(10) == 1);
    assert(t.order_of_key(11) == 2);
    assert(*t.find_by_order(0) == 8);
    t.join(t2); // assuming T < T2 or T > T2, merge
        t2 into t
}
```

2.9 persistent array

```
struct node {
    node *l, *r;
    int val;

    node(int x) : l(NULL), r(NULL), val(x) {}
    node() : l(NULL), r(NULL), val(-1) {}
};

typedef node *pnode;

pnode update(pnode cur, int l, int r, int at, int
    what) {
    pnode ans = new node();

    if (cur != NULL) {
        *ans = *cur;
    }
    if (l == r) {
        ans->val = what;
        return ans;
    }
    int m = (l + r) >> 1;
    if (at <= m)
        ans->l = update(ans->l, l, m, at, what);
    else
        ans->r = update(ans->r, m + 1, r, at, what);
    return ans;
}

int get(pnode cur, int l, int r, int at) {
    if (cur == NULL)
        return 0;
    if (l == r)
        return cur->val;
    int m = (l + r) >> 1;
    if (at <= m)
        return get(cur->l, l, m, at);
    else
        return get(cur->r, m + 1, r, at);
}
```

2.10 persistent seg tree

```
/* Problem: https://cses.fi/problemset/task/1737/
 * Your task is to maintain a list of arrays which
     initially has a single array. You have to
     process the following types of queries:
 * Query 1: Set the value a in array k to x.
 * Query 2: Calculate the sum of values in range
     [a,b] in array k.
 * Query 3: Create a copy of array k and add it to
     the end of the list.
 * Idea to create a persistent segment tree to save
     all version of array.
*/

vector <int> a;

struct Node{
    int val;
    Node *left, *right;
    Node(){
        left = right = NULL;
        val = 0;
    }
    Node(Node* l, Node *r, int v){
        left = l;
        right = r;
        val = v;
```

```
    }
};

void build(Node* &cur, int l, int r){
    if(l == r){
        cur->val = a[l];
        return;
    }
    int mid = (l+r) >> 1;
    cur->left = new Node();
    cur->right = new Node();
    build(cur->left, l, mid);
    build(cur->right, mid+1, r);
    cur->val = cur->left->val + cur->right->val;
}

void update(Node* prev, Node* &cur, int l, int r,
     int i, int val){
    if(i < l || r < i)
        return;
    if(l == r && l == i){
        cur->val = val;
        return;
    }
    int mid = (l+r) >> 1;
    if(i <= mid){
        cur->right = prev->right;
        cur->left = new Node();
        update(prev->left, cur->left, l, mid, i,
             val);
    }else{
        cur->left = prev->left;
        cur->right = new Node();
        update(prev->right, cur->right, mid+1, r, i,
             val);
    }
    cur->val = cur->left->val + cur->right->val;
}

int get(Node* cur, int l, int r, int u, int v){
    if(v < l || r < u)
        return 0;
    if(u <= l && r <= v){
        return cur->val;
    }
    int mid = (l+r) >> 1;
    int L = get(cur->left, l, mid, u, v);
    int R = get(cur->right, mid+1, r, u, v);
    return L + R;
}

Node* ver[MAXN];
```

2.11 persistent segment (v2)

```
/*
    Find distinct numbers in a range (online query
         with persistent array)
*/
struct Node{
    int lnode, rnode;
    int sum;
    Node(){
        lnode = rnode = sum = 0;
    }
}ver[MAXN * 120];
int sz = 0;

int build_new_node(int l, int r){
    int next = ++sz;
    if(l != r){
        int mid = (l+r) >> 1;
        ver[next].lnode = build_new_node(l, mid);
        ver[next].rnode = build_new_node(mid+1, r);
    }
    return next;
}

int update(int cur, int l, int r, int pos, int val){
    int next = ++sz;
    ver[next] = ver[cur];
    if(l == r){
        ver[next].sum = val;
        return next;
    }
    else{
        int mid = (l+r) >> 1;
        if(pos <= mid)
            ver[next].lnode = update(ver[cur].lnode,
                 l, mid, pos, val);
        else
            ver[next].rnode = update(ver[cur].rnode
                 , mid+1, r, pos, val);
    }
    ver[next].sum = ver[ver[next].lnode].sum +
         ver[ver[next].rnode].sum;
    return next;
}

int get(int cur, int l, int r, int u, int v){
    if(r < u ||v < l)
        return 0;
    if(u <= l && r <= v){
        return ver[cur].sum;
    }
    int mid = (l+r) >> 1;
    return get(ver[cur].lnode, l, mid, u, v) +
         get(ver[cur].rnode, mid+1, r, u, v);
}
```

2.12 persistent trie

```
// both tries can be tested with the problem:
     http://codeforces.com/problemset/problem/916/D

// Persistent binary trie (BST for integers)
const int MD = 31;

struct node_bin {
  node_bin *child[2];
  int val;

  node_bin() : val(0) {
    child[0] = child[1] = NULL;
  }
};

typedef node_bin* pnode_bin;

pnode_bin copy_node(pnode_bin cur) {
  pnode_bin ans = new node_bin();
  if (cur) *ans = *cur;
  return ans;
}

pnode_bin modify(pnode_bin cur, int key, int inc,
     int id = MD) {
  pnode_bin ans = copy_node(cur);
  ans->val += inc;
  if (id >= 0) {
    int to = (key >> id) & 1;
    ans->child[to] = modify(ans->child[to], key,
         inc, id - 1);
  }
  return ans;
}

int sum_smaller(pnode_bin cur, int key, int id =
     MD) {
  if (cur == NULL) return 0;
  if (id < 0) return 0; // strictly smaller
  // if (id == - 1) return cur->val; // smaller or
       equal

  int ans = 0;
  int to = (key >> id) & 1;
  if (to) {
    if (cur->child[0]) ans += cur->child[0]->val;
    ans += sum_smaller(cur->child[1], key, id - 1);
```

```
  } else {
    ans = sum_smaller(cur->child[0], key, id - 1);
  }
  return ans;
}


// Persistent trie for strings.
const int MAX_CHILD = 26;
struct node {
  node *child[MAX_CHILD];
  int val;
  node() : val(-1) {
    for (int i = 0; i < MAX_CHILD; i++) {
      child[i] = NULL;
    }
  }
};

typedef node* pnode;

pnode copy_node(pnode cur) {
  pnode ans = new node();
  if (cur) *ans = *cur;
  return ans;
}

pnode set_val(pnode cur, string &key, int val, int
    id = 0) {
  pnode ans = copy_node(cur);
  if (id >= int(key.size())) {
    ans->val = val;
  } else {
    int t = key[id] - 'a';
    ans->child[t] = set_val(ans->child[t], key,
        val, id + 1);
  }
  return ans;
}

pnode get(pnode cur, string &key, int id = 0) {
  if (id >= int(key.size()) || !cur)
    return cur;
  int t = key[id] - 'a';
  return get(cur->child[t], key, id + 1);
}
```

2.13 segment tree

```
// Problem:
    https://codeforces.com/edu/course/2/lesson/4/1/practice/contest/273169/problem/B
struct SegmentTree {
#define m ((l + r) >> 1)
#define lc (i << 1)
#define rc (i << 1 | 1)
    vector<int> mn;
    int n;

    SegmentTree(int n = 0) : n(n){
        mn.resize(4 * n + 1, 0);
    }

    SegmentTree(const vector<int> &a) : n(a.size())
        {
        mn.resize(4 * n + 1, 0);
        function<void(int, int, int)> build =
            [&](int i, int l, int r){
            if (l == r){
                mn[i] = a[l - 1];
                return;
            }
            build(lc, l, m); build(rc, m + 1, r);
            mn[i] = min(mn[lc], mn[rc]);
        };
        build(1, 1, n);
    }

    void update(int i, int l, int r, int p, long
        val){
        if (l == r){
            mn[i] = val;
            return;
        }
        if (p <= m) update(lc, l, m, p, val);
        else update(rc, m + 1, r, p, val);
        mn[i] = min(mn[lc], mn[rc]);
    }

    int get(int i, int l, int r, int u, int v){
        if (v < l || r < u) return INF;
        if (u <= l && r <= v) return mn[i];
        return min(get(lc, l, m, u, v), get(rc, m +
            1, r, u, v));
    }

    void update(int p, long val){
        update(1, 1, n, p, val);
    }

    int get(int l, int r){
        return get(1, 1, n, l, r);
    }
#undef m
#undef lc
#undef rc
};


// Problem: There are two operations:
// 1 l r val: add the value val to the segment from
    l to r
// 2 l v: calculate the minimum of elements from l
    to r
struct LazySegmentTree {
#define m ((l + r) >> 1)
#define lc (i << 1)
#define rc (i << 1 | 1)
    vector<int> mn, lazy;
    int n;

    LazySegmentTree(int n = 0) : n(n){
        mn.resize(4 * n + 1, 0);
        lazy.resize(4 * n + 1, 0);
    }

    void push(int i, int l, int r){
        if (lazy[i] == 0) return;
        mn[i] += lazy[i];
        if (l != r){
            lazy[lc] += lazy[i];
            lazy[rc] += lazy[i];
        }
        lazy[i] = 0;
    }

    void update(int i, int l, int r, int u, int v,
        int val){
        push(i, l, r);
        if (v < l || r < u) return;
        if (u <= l && r <= v){
            lazy[i] += val;
            push(i, l, r);
            return;
        }
        update(lc, l, m, u, v, val); update(rc, m +
            1, r, u, v, val);
        mn[i] = min(mn[lc], mn[rc]);
    }

    int get(int i, int l, int r, int u, int v){
        push(i, l, r);
        if (v < l || r < u) return INF;
        if (u <= l && r <= v) return mn[i];
        return min(get(lc, l, m, u, v), get(rc, m +
            1, r, u, v));
    }

    void update(int l, int r, int val){
        update(1, 1, n, l, r, val);
    }

    int get(int l, int r){
```

```
        return get(1, 1, n, 1, r);
    }
#undef m
#undef lc
#undef rc
};
```

2.14 sparse table

```
template <typename T, typename func =
     function<T(const T, const T)>>
struct SparseTable {
    func calc;
    int n;
    vector<vector<T>> ans;

    SparseTable() {}

    SparseTable(const vector<T>& a, const func& f)
         : n(a.size()), calc(f) {
        int last = trunc(log2(n)) + 1;
        ans.resize(n);
        for (int i = 0; i < n; i++){
            ans[i].resize(last);
        }
        for (int i = 0; i < n; i++){
            ans[i][0] = a[i];
        }
        for (int j = 1; j < last; j++){
            for (int i = 0; i <= n - (1 << j); i++){
                ans[i][j] = calc(ans[i][j - 1],
                     ans[i + (1 << (j - 1))][j - 1]);
            }
        }
    }

    T query(int l, int r){
        assert(0 <= l && l <= r && r < n);
        int k = trunc(log2(r - l + 1));
        return calc(ans[l][k], ans[r - (1 << k) +
             1][k]);
    }
};
```

2.15 trie

```
const int MN = 26; // size of alphabet
const int MS = 100010; // Number of states.

struct trie{
  struct node{
    int c;
    int a[MN];
  };

  node tree[MS];
  int nodes;

  void clear(){
    tree[nodes].c = 0;
    memset(tree[nodes].a, -1, sizeof tree[nodes].a);
    nodes++;
  }

  void init(){
    nodes = 0;
    clear();
  }

  int add(const string &s, bool query = 0){
    int cur_node = 0;
    for(int i = 0; i < s.size(); ++i){
      int id = gid(s[i]);
      if(tree[cur_node].a[id] == -1){
        if(query) return 0;
        tree[cur_node].a[id] = nodes;
        clear();
      }
      cur_node = tree[cur_node].a[id];
    }
    if(!query) tree[cur_node].c++;
    return tree[cur_node].c;
  }

};
```

3 Geometry

3.1 center 2 points + radious

```
vector<point> find_center(point a, point b, long
     double r) {
  point d = (a - b) * 0.5;
  if (d.dot(d) > r * r) {
    return vector<point> ();
  }
  point e = b + d;
  long double fac = sqrt(r * r - d.dot(d));
  vector<point> ans;
  point x = point(-d.y, d.x);
  long double l = sqrt(x.dot(x));
  x = x * (fac / l);
  ans.push_back(e + x);
  x = point(d.y, -d.x);
  x = x * (fac / l);
  ans.push_back(e + x);
  return ans;
}
```

3.2 closest pair problem

```
struct point {
  double x, y;
  int id;
  point() {}
  point (double a, double b) : x(a), y(b) {}
};

double dist(const point &o, const point &p) {
  double a = p.x - o.x, b = p.y - o.y;
  return sqrt(a * a + b * b);
}

double cp(vector<point> &p, vector<point> &x,
     vector<point> &y) {
  if (p.size() < 4) {
    double best = 1e100;
    for (int i = 0; i < p.size(); ++i)
      for (int j = i + 1; j < p.size(); ++j)
        best = min(best, dist(p[i], p[j]));
    return best;
  }

  int ls = (p.size() + 1) >> 1;
  double l = (p[ls - 1].x + p[ls].x) * 0.5;
  vector<point> xl(ls), xr(p.size() - ls);
  unordered_set<int> left;
  for (int i = 0; i < ls; ++i) {
    xl[i] = x[i];
    left.insert(x[i].id);
  }
  for (int i = ls; i < p.size(); ++i) {
    xr[i - ls] = x[i];
  }

  vector<point> yl, yr;
  vector<point> pl, pr;
  yl.reserve(ls); yr.reserve(p.size() - ls);
  pl.reserve(ls); pr.reserve(p.size() - ls);
  for (int i = 0; i < p.size(); ++i) {
    if (left.count(y[i].id))
      yl.push_back(y[i]);
    else
      yr.push_back(y[i]);
```

```
    if (left.count(p[i].id))
      pl.push_back(p[i]);
    else
      pr.push_back(p[i]);
  }

  double dl = cp(pl, xl, yl);
  double dr = cp(pr, xr, yr);
  double d = min(dl, dr);
  vector<point> yp; yp.reserve(p.size());
  for (int i = 0; i < p.size(); ++i) {
    if (fabs(y[i].x - l) < d)
      yp.push_back(y[i]);
  }
  for (int i = 0; i < yp.size(); ++i) {
    for (int j = i + 1; j < yp.size() && j < i + 7;
        ++j) {
      d = min(d, dist(yp[i], yp[j]));
    }
  }
  return d;
}

double closest_pair(vector<point> &p) {
  vector<point> x(p.begin(), p.end());
  sort(x.begin(), x.end(), [](const point &a, const
      point &b) {
    return a.x < b.x;
  });
  vector<point> y(p.begin(), p.end());
  sort(y.begin(), y.end(), [](const point &a, const
      point &b) {
    return a.y < b.y;
  });
  return cp(p, x, y);
}
```

3.3 convex diameter

```
struct point{
    int x, y;
};

struct vec{
    int x, y;
};

vec operator - (const point &A, const point &B){
    return vec{A.x - B.x, A.y - B.y};
}

int cross(vec A, vec B){
    return A.x*B.y - A.y*B.x;
}

int cross(point A, point B, point C){
    int val = A.x*(B.y - C.y) + B.x*(C.y - A.y) +
        C.x*(A.y - B.y);
    if(val == 0)
        return 0; // coline
    if(val < 0)
        return 1; // clockwise
    return -1; //counter clockwise
}

vector <point> findConvexHull(vector <point>
     points){
    vector <point> convex;
    sort(points.begin(), points.end(), [](const
        point &A, const point &B){
        return (A.x == B.x)? (A.y < B.y): (A.x <
            B.x);
    });
    vector <point> Up, Down;
    point A = points[0], B = points.back();
    Up.push_back(A);
    Down.push_back(A);

    for(int i = 0; i < points.size(); i++){
        if(i == points.size()-1 || cross(A,
            points[i], B) > 0){
            while(Up.size() > 2 &&
                cross(Up[Up.size()-2],
                Up[Up.size()-1], points[i]) <= 0)
               Up.pop_back();
            Up.push_back(points[i]);
        }
        if(i == points.size()-1 || cross(A,
            points[i], B) < 0){
            while(Down.size() > 2 &&
                cross(Down[Down.size()-2],
                Down[Down.size()-1], points[i]) >=
                0)
               Down.pop_back();
            Down.push_back(points[i]);
        }
    }
    for(int i = 0; i < Up.size(); i++)
        convex.push_back(Up[i]);
    for(int i = Down.size()-2; i > 0; i--)
        convex.push_back(Down[i]);
    return convex;
}

int dist(point A, point B){
    return (A.x - B.x)*(A.x - B.x) + (A.y -
        B.y)*(A.y - B.y);
}

double findConvexDiameter(vector <point>
     convexHull){
    int n = convexHull.size();

    int is = 0, js = 0;
    for(int i = 1; i < n; i++){
        if(convexHull[i].y > convexHull[is].y)
            is = i;
        if(convexHull[js].y > convexHull[i].y)
            js = i;
    }

    int maxd = dist(convexHull[is], convexHull[js]);
    int i, maxi, j, maxj;
    i = maxi = is;
    j = maxj = js;
    do{
        int ni = (i+1)%n, nj = (j+1)%n;
        if(cross(convexHull[ni] - convexHull[i],
             convexHull[nj] - convexHull[j]) <= 0){
            j = nj;
        }else{
            i = ni;
        }
        int d = dist(convexHull[i], convexHull[j]);
        if(d > maxd){
            maxd = d;
            maxi = i;
            maxj = j;
        }
    }while(i != is || j != js);
    return sqrt(maxd);
}
```

3.4 pick theorem

```
struct point{
    ll x, y;
};

//Pick: S = I + B/2 - 1

ld polygonArea(vector <point> &points){
    int n = (int)points.size();
    ld area = 0.0;
    int j = n-1;
    for(int i = 0; i < n; i++){
```

```
        area += (points[j].x + points[i].x) *
            (points[j].y - points[i].y);
        j = i;
    }

    return abs(area/2.0);
}

ll boundary(vector <point> points){
    int n = (int)points.size();
    ll num_bound = 0;
    for(int i = 0; i < n; i++){
        ll dx = (points[i].x - points[(i+1)%n].x);
        ll dy = (points[i].y - points[(i+1)%n].y);
        num_bound += abs(__gcd(dx, dy)) - 1;
    }
    return num_bound;
}
```

3.5 squares

```
typedef long double ld;

const ld eps = 1e-12;
int cmp(ld x, ld y = 0, ld tol = eps) {
    return ( x <= y + tol) ? (x + tol < y) ? -1 : 0
        : 1;
}

struct point{
  ld x, y;
  point(ld a, ld b) : x(a), y(b) {}
  point() {}
};

struct square{
  ld x1, x2, y1, y2,
     a, b, c;
  point edges[4];
  square(ld _a, ld _b, ld _c) {
    a = _a, b = _b, c = _c;
    x1 = a - c * 0.5;
    x2 = a + c * 0.5;
    y1 = b - c * 0.5;
    y2 = b + c * 0.5;
    edges[0] = point(x1, y1);
    edges[1] = point(x2, y1);
    edges[2] = point(x2, y2);
    edges[3] = point(x1, y2);
  }
};

ld min_dist(point &a, point &b) {
  ld x = a.x - b.x,
     y = a.y - b.y;
  return sqrt(x * x + y * y);
}

bool point_in_box(square s1, point p) {
  if (cmp(s1.x1, p.x) != 1 && cmp(s1.x2, p.x) != -1
      &&
      cmp(s1.y1, p.y) != 1 && cmp(s1.y2, p.y) != -1)
    return true;
  return false;
}

bool inside(square &s1, square &s2) {
  for (int i = 0; i < 4; ++i)
    if (point_in_box(s2, s1.edges[i]))
      return true;

  return false;
}

bool inside_vert(square &s1, square &s2) {
  if ((cmp(s1.y1, s2.y1) != -1 && cmp(s1.y1, s2.y2)
      != 1) ||
      (cmp(s1.y2, s2.y1) != -1 && cmp(s1.y2, s2.y2)
      != 1))
    return true;
 return false;
}

bool inside_hori(square &s1, square &s2) {
  if ((cmp(s1.x1, s2.x1) != -1 && cmp(s1.x1, s2.x2)
      != 1) ||
      (cmp(s1.x2, s2.x1) != -1 && cmp(s1.x2, s2.x2)
      != 1))
    return true;
 return false;
}

ld min_dist(square &s1, square &s2) {
  if (inside(s1, s2) || inside(s2, s1))
    return 0;

  ld ans = 1e100;
  for (int i = 0; i < 4; ++i)
    for (int j = 0; j < 4; ++j)
      ans = min(ans, min_dist(s1.edges[i],
          s2.edges[j]));


  if (inside_hori(s1, s2) || inside_hori(s2, s1)) {
    if (cmp(s1.y1, s2.y2) != -1)
      ans = min(ans, s1.y1 - s2.y2);
    else
    if (cmp(s2.y1, s1.y2) != -1)
      ans = min(ans, s2.y1 - s1.y2);
  }

  if (inside_vert(s1, s2) || inside_vert(s2, s1)) {
    if (cmp(s1.x1, s2.x2) != -1)
      ans = min(ans, s1.x1 - s2.x2);
    else
    if (cmp(s2.x1, s1.x2) != -1)
      ans = min(ans, s2.x1 - s1.x2);
  }

  return ans;
}
```

3.6 template

```
#define EPS 1e-6
const double PI = acos(-1.0);

double DEG_TO_RAD(double d) { return d * PI /
    180.0; }
double RAD_TO_DEG(double r) { return r * 180.0 /
    PI; }

inline int cmp(double a, double b) {
    return (a < b - EPS) ? -1 : ((a > b + EPS) ? 1
        : 0);
}

struct Point{
    double x, y;
    Point(){
        x = y = 0.0;
    }
    Point(double x, double y): x(x), y(y) {}

    Point operator + (const Point& a) const {
        return Point(x+a.x, y+a.y); }
    Point operator - (const Point& a) const {
        return Point(x-a.x, y-a.y); }
    Point operator * (double k) const { return
        Point(x*k, y*k); }
    Point operator / (double k) const { return
        Point(x/k, y/k); }

    double dot(const Point& a) const { return x*a.x
        + y*a.y; } // dot product
    double cross(const Point& a) const { return
        x*a.y - y*a.x; } // cross product
```

```
    int cmp(const Point& q) const {
        if (x != q.x) return ::cmp(x, q.x);
        return ::cmp(y, q.y);
    }

    #define Comp(x) bool operator x (Point q) const
         { return cmp(q) x 0; }
    Comp(>) Comp(<) Comp(==) Comp(>=) Comp(<=)
         Comp(!=)
    #undef Comp

    double norm() { return x*x + y*y; }
    double len() { return sqrt(norm()); }

    // Rotate vector
    Point rotate(double alpha) {
        double cosa = cos(alpha), sina = sin(alpha);
        return Point(x * cosa - y * sina, x * sina +
             y * cosa);
    }
};

istream& operator >> (istream& cin, Point& p) {
    cin >> p.x >> p.y;
    return cin;
}
ostream& operator << (ostream& cout, Point& p) {
    cout << p.x << ' ' << p.y;
    return cout;
}

struct Line{
    double a, b, c;
    Point A, B;

    Line(double a, double b, double c): a(a), b(b),
         c(c) {}

    Line(Point A, Point B): A(A), B(B) {
        a = B.y - A.y;
        b = A.x - B.x;
        c = -(a * A.x + b * A.y);
    }

    // initialize a line with slope k
    Line(Point P, double k) {
        a = -k;
        b = 1;
        c = k * P.x - P.y;
    }

    double f(Point A){
        return a * A.x + b * A.y + c;
    }
};

bool areParallel(Line l1, Line l2) {
    return cmp(l1.a*l2.b, l1.b*l2.a) == 0;
}

bool areSame(Line l1, Line l2) {
    return areParallel(l1, l2) && cmp(l1.c*l2.a,
         l2.c*l1.a) == 0
           && cmp(l1.c*l2.b, l1.b*l2.c) == 0;
}

bool areIntersect(Line l1, Line l2, Point &p) {
    if (areParallel(l1, l2))
        return false;
    double dx = l1.b*l2.c - l2.b*l1.c;
    double dy = l1.c*l2.a - l2.c*l1.a;
    double d = l1.a*l2.b - l2.a*l1.b;
    p = Point(dx / d, dy / d);
    return true;
}

// distance from p to line ab
double distToLine(Point p, Point a, Point b, Point
     &c) {
    Point ap = p - a, ab = b - a;
    double k = ap.dot(ab) / ab.norm();
    c = a + (ab * k);
    return (p - c).len();
}

// closest point from p in line l.
void closestPoint(Line l, Point p, Point &ans) {
    if (fabs(l.b) < EPS) {
        ans.x = -(l.c) / l.a; ans.y = p.y;
        return;
    }
    if (fabs(l.a) < EPS) {
        ans.x = p.x; ans.y = -(l.c) / l.b;
        return;
    }
    Line perp(l.b, -l.a, - (l.b*p.x - l.a*p.y));
    areIntersect(l, perp, ans);
}

// reflect point p over line l
void reflectionPoint(Line l, Point p, Point &ans) {
    Point b;
    closestPoint(l, p, b);
    ans = p + (b - p) * 2;
}
```

3.7 triangles

Let a, b, c be length of the three sides of a triangle.

$$p = (a + b + c) * 0.5$$

The inradius is defined by:

$$iR = \sqrt{\frac{(p-a)(p-b)(p-c)}{p}}$$

The radius of its circumcircle is given by the formula:

$$cR = \frac{abc}{\sqrt{(a+b+c)(a+b-c)(a+c-b)(b+c-a)}}$$

4 Graphs

4.1 bridges

```
vector<int> G[MAXN];

int cnt = 0;
int low[MAXN], num[MAXN];
int numChild[MAXN], criVertex[MAXN], bridgeCnt = 0;

void DFS(int u, int pre) {
    num[u] = ++cnt;
    low[u] = INT_MAX;

    for(int v: G[u]) {
        if(v == pre) continue;
        if(num[v]) {
            low[u] = min(low[u], num[v]);
        }else {
            numChild[u]++;
            DFS(v, u);
            if(low[v] >= num[u]) criVertex[u] = 1;
            if(low[v] > num[u]) bridgeCnt++;
            low[u] = min(low[u], low[v]);
        }
    }
}

int main(){
    // input
    for(int i = 1; i <= n; i++)
        if(!num[i]) {
            DFS(i, 0);
            if(numChild[i] < 2)
```

```
            criVertex[i] = 0;
        }
    int criVertexCnt = 0;
    for(int i = 1; i <= n; i++)
        criVertexCnt += criVertex[i];
    cout << criVertexCnt << ' ' << bridgeCnt <<
        '\n';
}
```

4.2 delete on dsu

```
struct dsu_save {
    int u, v;
    int par_u, par_v;

    dsu_save() {}

    dsu_save(int _v, int _par_v, int _u, int _par_u)
        : v(_v), par_v(_par_v), u(_u), par_u(_par_u)
            {}
};

class dsu_rollback {
  public:
    vector<int> parent;
    int comps;
    stack<dsu_save> st_op;

    dsu_rollback(){};

    dsu_rollback(int n) {
        parent.resize(n + 1, -1);
        comps = n;
    }

    int find_set(int u) {
        while (parent[u] > 0)
            u = parent[u];
        return u;
    }

    bool Union(int u, int v) {
        int U = find_set(u);
        int V = find_set(v);
        if (U == V)
            return false;
        comps--;
        st_op.push(dsu_save(U, parent[U], V,
             parent[V]));
        int x = parent[U] + parent[V];
        if (parent[U] > parent[V]) {
            parent[U] = V;
            parent[V] = x;
        } else {
            parent[U] = x;
            parent[V] = U;
        }
        return true;
    }

    void rollback() {
        if (st_op.empty())
            return;
        dsu_save x = st_op.top();
        st_op.pop();
        comps++;
        parent[x.u] = x.par_u;
        parent[x.v] = x.par_v;
    }
};

struct query {
    int u, v;
    bool united;
};

class QueryTree {
    vector<vector<query>> t;
    dsu_rollback dsu;
    int T;

  public:
    QueryTree(int _T, int n) {
        this->T = _T;
        this->dsu = dsu_rollback(n);
        t.resize(4 * T + 4);
    }

    void add_to_tree(int id, int l, int r, int u,
         int v, query q) {
        if (v < l || r < u || u > v)
            return;
        if (u <= l && r <= v) {
            t[id].push_back(q);
            return;
        }
        int mid = (l + r) >> 1;
        add_to_tree(2 * id, l, mid, u, v, q);
        add_to_tree(2 * id + 1, mid + 1, r, u, v, q);
    }

    void add_query(query q, int l, int r) {
         add_to_tree(1, 0, T - 1, l, r, q); }

    void DFS(int id, int l, int r, vector<int>
         &ans) {
        for (query &q : t[id])
            q.united = dsu.Union(q.u, q.v);

        if (l == r) {
            ans[l] = dsu.comps;
        } else {
            int mid = (l + r) >> 1;
            DFS(2 * id, l, mid, ans);
            DFS(2 * id + 1, mid + 1, r, ans);
        }

        for (query &q : t[id])
            if (q.united)
                dsu.rollback();
    }

    vector<int> compute() {
        vector<int> ans(T); // T query
        DFS(1, 0, T - 1, ans);
        return ans;
    }
};
```

4.3 euler path

```
struct DirectedEulerPath {
    int n;
    vector<vector<int>> g;
    vector<int> path;

    void init(int _n) {
        n = _n;
        g = vector<vector<int>>(n + 1,
             vector<int>());
        path.clear();
    }

    void add_edge(int u, int v) {
         g[u].push_back(v); }

    void dfs(int u) {
        while (g[u].size()) {
            int v = g[u].back();
            g[u].pop_back();
            dfs(v);
        }
        path.push_back(u);
    }

    bool getPath() {
        int ctEdges = 0;
        vector<int> outDeg, inDeg;
        outDeg = inDeg = vector<int>(n + 1, 0);
```

```
        for (int i = 1; i <= n; i++) {
            ctEdges += g[i].size();
            outDeg[i] += g[i].size();
            for (auto &u : g[i])
                inDeg[u]++;
        }
        int ctMiddle = 0, src = 1;
        for (int i = 1; i <= n; i++) {
            if (abs(inDeg[i] - outDeg[i]) > 1)
                return 0;
            if (inDeg[i] == outDeg[i])
                ctMiddle++;
            if (outDeg[i] > inDeg[i])
                src = i;
        }
        if (ctMiddle != n && ctMiddle + 2 != n)
            return 0;
        dfs(src);
        reverse(path.begin(), path.end());
        return (path.size() == ctEdges + 1);
    }
};
```

4.4 karp min mean cycle

```
/**
 * Finds the min mean cycle, if you need the max
     mean cycle
 * just add all the edges with negative cost and
     print
 * ans * -1
 *
 * test: uva, 11090 - Going in Cycle!!
 * */

const int MN = 1000;
struct edge {
    int v;
    long long w;
    edge() {}
    edge(int v, int w) : v(v), w(w) {}
};

long long d[MN][MN];
// This is a copy of g because increments the size
// pass as reference if this does not matter.
int karp(vector<vector<edge>> g) {
    int n = g.size();

    g.resize(n + 1); // this is important

    for (int i = 0; i < n; ++i)
        if (!g[i].empty())
            g[n].push_back(edge(i, 0));
    ++n;

    for (int i = 0; i < n; ++i)
        fill(d[i], d[i] + (n + 1), INT_MAX);

    d[n - 1][0] = 0;

    for (int k = 1; k <= n; ++k)
        for (int u = 0; u < n; ++u) {
            if (d[u][k - 1] == INT_MAX)
                continue;
            for (int i = g[u].size() - 1; i >= 0;
                 --i)
                d[g[u][i].v][k] =
                    min(d[g[u][i].v][k], d[u][k - 1]
                    + g[u][i].w);
        }

    bool flag = true;

    for (int i = 0; i < n && flag; ++i)
        if (d[i][n] != INT_MAX)
            flag = false;

    if (flag) {
        return true; // return true if there is no a
             cycle.
    }

    double ans = 1e15;

    for (int u = 0; u + 1 < n; ++u) {
        if (d[u][n] == INT_MAX)
            continue;
        double W = -1e15;

        for (int k = 0; k < n; ++k)
            if (d[u][k] != INT_MAX)
                W = max(W, (double)(d[u][n] -
                    d[u][k]) / (n - k));

        ans = min(ans, W);
    }

    // printf("%.2lf\n", ans);
    cout << fixed << setprecision(2) << ans << endl;

    return false;
}
```

4.5 konig's theorem

In any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover

4.6 matching

```
struct Hopcroft_Karp {
    static const int inf = 1e9;

    int n;
    vector<int> matchL, matchR, dist;
    vector<vector<int>> g;

    Hopcroft_Karp(int n)
        : n(n), matchL(n + 1), matchR(n + 1), dist(n
            + 1), g(n + 1) {}

    void addEdge(int u, int v) { g[u].push_back(v);
        }

    bool bfs() {
        queue<int> q;
        for (int u = 1; u <= n; u++) {
            if (!matchL[u]) {
                dist[u] = 0;
                q.push(u);
            } else
                dist[u] = inf;
        }
        dist[0] = inf;

        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (auto v : g[u]) {
                if (dist[matchR[v]] == inf) {
                    dist[matchR[v]] = dist[u] + 1;
                    q.push(matchR[v]);
                }
            }
        }

        return (dist[0] != inf);
    }

    bool dfs(int u) {
        if (!u)
            return true;
        for (auto v : g[u]) {
            if (dist[matchR[v]] == dist[u] + 1 &&
                dfs(matchR[v])) {
```

```
            matchL[u] = v;
            matchR[v] = u;
            return true;
        }
    }
    dist[u] = inf;
    return false;
}

int max_matching() {
    int matching = 0;
    while (bfs()) {
        for (int u = 1; u <= n; u++) {
            if (!matchL[u])
                if (dfs(u))
                    matching++;
        }
    }
    return matching;
}
};
```

4.7 max flow min cost

```
struct edge {
    long long x, y, cap, flow, cost;
};

struct MinCostMaxFlow {
    long long n, S, T;
    vector<vector<long long>> a;
    vector<long long> dist, prev, done, pot;
    vector<edge> e;

    MinCostMaxFlow() {}
    MinCostMaxFlow(long long _n, long long _S, long
         long _T) {
        n = _n;
        S = _S;
        T = _T;
        a = vector<vector<long long>>(n + 1);
        dist = vector<long long>(n + 1);
        prev = vector<long long>(n + 1);
        done = vector<long long>(n + 1);
        pot = vector<long long>(n + 1, 0);
    }

    void addEdge(long long x, long long y, long
         long _cap, long long _cost) {
        edge e1 = {x, y, _cap, 0, _cost};
        edge e2 = {y, x, 0, 0, -_cost};
        a[x].push_back(e.size());
        e.push_back(e1);
        a[y].push_back(e.size());
        e.push_back(e2);
    }

    pair<long long, long long> dijkstra() {
        long long flow = 0, cost = 0;
        for (long long i = 1; i <= n; i++)
            done[i] = 0, dist[i] = oo;
        priority_queue<pair<long long, long long>> q;
        dist[S] = 0;
        prev[S] = -1;
        q.push(make_pair(0, S));
        while (!q.empty()) {
            long long x = q.top().second;
            q.pop();
            if (done[x])
                continue;
            done[x] = 1;
            for (int i = 0; i < int(a[x].size());
                 i++) {
                long long id = a[x][i], y = e[id].y;
                if (e[id].flow < e[id].cap) {
                    long long D = dist[x] +
                        e[id].cost + pot[x] - pot[y];
                    if (!done[y] && D < dist[y]) {
                        dist[y] = D;
                        prev[y] = id;
                        q.push(make_pair(-dist[y],
                            y));
                    }
                }
            }
        }

        for (long long i = 1; i <= n; i++)
            pot[i] += dist[i];

        if (done[T]) {
            flow = oo;
            for (long long id = prev[T]; id >= 0; id
                 = prev[e[id].x])
                flow = min(flow, e[id].cap -
                    e[id].flow);
            for (long long id = prev[T]; id >= 0; id
                 = prev[e[id].x]) {
                cost += e[id].cost * flow;
                e[id].flow += flow;
                e[id ^ 1].flow -= flow;
            }
        }

        return make_pair(flow, cost);
    }

    pair<long long, long long> minCostMaxFlow() {
        long long totalFlow = 0, totalCost = 0;
        while (1) {
            pair<long long, long long> u =
                dijkstra();
            if (!done[T])
                break;
            totalFlow += u.first;
            totalCost += u.second;
        }
        return make_pair(totalFlow, totalCost);
    }
};
```

4.8 minimum path cover in DAG

Given a directed acyclic graph $G = (V, E)$, we are to find the minimum number of vertex-disjoint paths to cover each vertex in V.

We can construct a bipartite graph $G' = (Vout \cup Vin, E')$ from G, where :

$$Vout = \{v \in V : v\ has\ positive\ out - degree\}$$

$$Vin = \{v \in V : v\ has\ positive\ in - degree\}$$

$$E' = \{(u, v) \in Vout \times Vin : (u, v) \in E\}$$

Then it can be shown, via König's theorem, that G' has a matching of size m if and only if there exists $n - m$ vertex-disjoint paths that cover each vertex in G, where n is the number of vertices in G and m is the maximum cardinality bipartite mathching in G'.

Therefore, the problem can be solved by finding the maximum cardinality matching in G' instead.

NOTE: If the paths are note necesarily disjoints, find the transitive closure and solve the problem for disjoint paths.

4.9 planar graph (euler)

Euler's formula states that if a finite, connected, planar graph is drawn in the plane without any edge intersections, and v is the number of vertices, e is the number of edges and f is the number of faces (regions bounded by edges, including the outer, infinitely large region), then:

$$f + v = e + 2$$

It can be extended to non connected planar graphs with c connected components:

$$f + v = e + c + 1$$

4.10 two sat

```
/**
 * Given a set of clauses (a1 v a2)^(a2 v a3)....
 * this algorithm find a solution to it set of
     clauses.
 * test:
     http://lightoj.com/volume_showproblem.php?problem=1251
 **/

vector<int> G[MAXN];
vector<int> Gv2[MAXN];

int low[MAXN], num[MAXN];
int cntTime = 0, cntSCC = 0, SCC[MAXN];
vector<int> inSCC[MAXN];
stack<int> st;
queue<int> q;
// storing topo order with queue instead of stack
// because we need to go from back to begin of topo
    order

void DFS(int u) {
    low[u] = num[u] = ++cntTime;
    st.push(u);
    for (int v : G[u])
        if (num[v])
            low[u] = min(low[u], num[v]);
        else {
            DFS(v);
            low[u] = min(low[u], low[v]);
        }
    if (low[u] == num[u]) {
        int v;
        cntSCC++;
        do {
            v = st.top();
            st.pop();
            SCC[v] = cntSCC;
            inSCC[cntSCC].push_back(v);
            low[v] = num[v] = INT_MAX;
        } while (u != v);
    }
}

void DFS_topo(int u) {
    num[u] = 1;
    for (int v : Gv2[u])
        if (!num[v])
            DFS_topo(v);
    q.push(u);
}

int main() {
    int n, m;
    cin >> m >> n;
    auto getNot = [&](int u) -> int {
        if (u > n)
            return u - n;
        return u + n;
    };

    while (m--) {
        char c1, c2;
        int u, v;
        cin >> c1 >> u >> c2 >> v;
        if (c1 == '-')
            u += n;
        if (c2 == '-')
            v += n;
        // add (-v -> u) and (-u -> v)
        G[getNot(u)].push_back(v);
        G[getNot(v)].push_back(u);
    }

    // using tarjan's algorithm to find SCC.
    for (int i = 1; i <= 2 * n; i++)
        if (!num[i])
            DFS(i);

    vector<int> notSCC(2 * n + 1);

    // check if exist u and -u are in the same
        component
    for (int i = 1; i <= n; i++)
        if (SCC[i] == SCC[i + n])
            return cout << "IMPOSSIBLE", 0;
        else {
            // store the opposite component.
            notSCC[SCC[i]] = SCC[i + n];
            notSCC[SCC[i + n]] = SCC[i];
        }

    // build new graph
    for (int i = 1; i <= 2 * n; i++)
        for (int v : G[i])
            if (SCC[i] != SCC[v]) {
                Gv2[SCC[i]].push_back(SCC[v]);
            }

    // topological sort
    fill(num + 1, num + 1 + 2 * n, 0);
    for (int i = 1; i <= cntSCC; i++)
        if (!num[i])
            DFS_topo(i);

    vector<int> ansSCC(2 * n + 1, -1);
    vector<int> ans(2 * n + 1, 0);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        if (ansSCC[u] == -1) { // not pick
            // if u = 1 then -u must be 0
            ansSCC[u] = 1;
            ansSCC[notSCC[u]] = 0;
        }

        // set value of all nodes in the current SCC
        for (int v : inSCC[u]) {
            ans[v] = ansSCC[u];
        }
    }

    for (int i = 1; i <= n; i++)
        cout << ((ans[i]) ? '+' : '-') << ' ';
}
```

5 Math

5.1 Lucas theorem

For non-negative integers m and n and a prime p, the following congruence relation holds: :

$$\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} \pmod p,$$

where :

$$m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_1 p + m_0,$$

and :

$$n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_1 p + n_0$$

are the base p expansions of m and n respectively. This uses the convention that $\binom{m}{n} = 0$ if $m \le n$.

5.2 cumulative sum of divisors

```
/*
The function SOD(n) (sum of divisors) is defined
as the summation of all the actual divisors of
an integer number n. For example,

  SOD(24) = 2+3+4+6+8+12 = 35.

The function CSOD(n) (cumulative SOD) of an integer
     n, is defined as below:

  csod(n) = \sum_{i = 1}^{n} sod(i)

It can be computed in O(sqrt(n)):
*/

long long csod(long long n) {
  long long ans = 0;
  for (long long i = 2; i * i <= n; ++i) {
    long long j = n / i;
    ans += (i + j) * (j - i + 1) / 2;
    ans += i * (j - i);
  }
  return ans;
}
```

5.3 fft

```
/**
 * Fast Fourier Transform.
 * Useful to compute convolutions.
 * computes:
 *    C(f star g)[n] = sum_m(f[m] * g[n - m])
 * for all n.
 * test: icpc live archive, 6886 - Golf Bot
 * */


using namespace std;
#include <bits/stdc++.h>
#define D(x) cout << #x " = " << (x) << endl
#define endl '\n'

const int MN = 262144 << 1;
int d[MN + 10], d2[MN + 10];


const double PI = acos(-1.0);

struct cpx {
  double real, image;
  cpx(double _real, double _image) {
    real = _real;
    image = _image;
  }
  cpx(){}
};

cpx operator + (const cpx &c1, const cpx &c2) {
  return cpx(c1.real + c2.real, c1.image +
      c2.image);
}

cpx operator - (const cpx &c1, const cpx &c2) {
  return cpx(c1.real - c2.real, c1.image -
      c2.image);
}

cpx operator * (const cpx &c1, const cpx &c2) {
  return cpx(c1.real*c2.real - c1.image*c2.image,
      c1.real*c2.image + c1.image*c2.real);
}

int rev(int id, int len) {
  int ret = 0;
  for (int i = 0; (1 << i) < len; i++) {
    ret <<= 1;
    if (id & (1 << i)) ret |= 1;
  }
  return ret;
}

cpx A[1 << 20];

void FFT(cpx *a, int len, int DFT) {
  for (int i = 0; i < len; i++)
    A[rev(i, len)] = a[i];
  for (int s = 1; (1 << s) <= len; s++) {
    int m = (1 << s);
    cpx wm = cpx(cos( DFT * 2 * PI / m), sin(DFT *
        2 * PI / m));
    for(int k = 0; k < len; k += m) {
      cpx w = cpx(1, 0);
      for(int j = 0; j < (m >> 1); j++) {
        cpx t = w * A[k + j + (m >> 1)];
        cpx u = A[k + j];
        A[k + j] = u + t;
        A[k + j + (m >> 1)] = u - t;
        w = w * wm;
      }
    }
  }
  if (DFT == -1) for (int i = 0; i < len; i++)
      A[i].real /= len, A[i].image /= len;
  for (int i = 0; i < len; i++) a[i] = A[i];
  return;
}

cpx in[1 << 20];

void solve(int n) {
  memset(d, 0, sizeof d);
  int t;
  for (int i = 0; i < n; ++i) {
    cin >> t;
    d[t] = true;
  }
  int m;
  cin  >> m;
  vector<int> q(m);
  for (int i = 0; i < m; ++i)
    cin >> q[i];

  for (int i = 0; i < MN; ++i) {
    if (d[i])
      in[i] = cpx(1, 0);
    else
      in[i] = cpx(0, 0);
  }

  FFT(in, MN, 1);
  for (int i = 0; i < MN; ++i) {
    in[i] = in[i] * in[i];
  }
  FFT(in, MN, -1);

  int ans = 0;
  for (int i = 0; i < q.size(); ++i) {
    if (in[q[i]].real > 0.5 || d[q[i]]) {
      ans++;
    }
  }
  cout << ans << endl;
}

int main() {
  ios_base::sync_with_stdio(false);cin.tie(NULL);
  int n;
  while (cin >> n)
    solve(n);
  return 0;
}
```

5.4 fibonacci properties

Let A, B and n be integer numbers.

$$k = A - B \tag{1}$$

$$F_A F_B = F_{k+1} F_A^2 + F_k F_A F_{A-1} \tag{2}$$

$$\sum_{i=0}^{n} F_i^2 = F_{n+1} F_n \tag{3}$$

$ev(n) =$ returns 1 if n is even.

$$\sum_{i=0}^{n} F_i F_{i+1} = F_{n+1}^2 - ev(n) \tag{4}$$

$$\sum_{i=0}^{n} F_i F_{i-1} = \sum_{i=0}^{n-1} F_i F_{i+1} \tag{5}$$

5.5 gauss

```
const int inf = 1e9;
const double eps = 1e-6;

/*
 * Input:
 *     a: the coefficients of the system
 *     ans: storing answer
 * Output:
 *     The number of roots
*/
int gauss(vector <vector <double>> a, vector
     <double> &ans){
    int n = (int)a.size();
    int m = (int)a[0].size() - 1;

    vector <int> where(m, -1);

    for(int col = 0, row = 0; col < m && row < n;
         col++){
        // Choosing the pivot row is done with
             heuristic:
        // choosing maximum value in the current
             column
        int pivot = row;
        for(int i = row; i < n; i++)
            if(abs(a[i][col]) > abs(a[pivot][col]))
                pivot = i;

        for(int i = col; i <= m; i++)
            swap(a[pivot][i], a[row][i]);
        where[col] = row;

        for(int i = 0; i < n; i++)
            if(i != row){
                double c = a[i][col] / a[row][col];
                for(int j = col; j <= m; j++)
                    a[i][j] -= a[row][j] * c;
            }
        row++;
    }

    ans.assign(m, 0);
    for(int i = 0; i < m; i++)
        if(where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];

    // calculate the number of roots by re-checking
         the system of equations.
    for(int i = 0; i < n; i++){
        double sum = 0;
        for(int j = 0; j < m; j++)
            sum += ans[j] * a[i][j];
        if(abs(sum - a[i][m]) > eps)
            return 0;
    }

    for(int i = 0; i < m; i++)
        if(where[i] == -1)
            return inf;

    return 1;
}
```

5.6 others

Approximate factorial

$$n! = \sqrt{2.\pi.n}.(\frac{n}{e})^n \tag{6}$$

5.7 polynomials

```
// TODO: what's this ?
const double pi = acos(-1);
struct poly {
  deque <double> coef;
  double x_lo, x_hi;

  double evaluate(double x) {
    double ans = 0;
    for (auto it : coef)
      ans = (ans * x + it);
    return ans;
  }

  double volume(double x, double dx=1e-6) {
    dx = (x_hi - x_lo) / 1000000.0;
    double ans = 0;
    for (double ix = x_lo; ix <= x; ix += dx) {
      double rad = evaluate(ix);
      ans += pi * rad * rad * dx;
    }
    return ans;
  }
};
```

5.8 sigma function

the sigma function is defined as:

$$\sigma_x(n) = \sum_{d|n} d^x$$

when $x = 0$ is called the divisor function, that counts the number of positive divisors of n.

Now, we are interested in find

$$\sum_{d|n} \sigma_0(d)$$

if n is written as prime factorization:

$$n = \prod_{i=1}^{k} P_i^{e_k}$$

we can demonstrate that:

$$\sum_{d|n} \sigma_0(d) = \prod_{i=1}^{k} g(e_k + 1)$$

where $g(x)$ is the sum of the first x positive numbers:

$$g(x) = (x * (x + 1))/2$$

5.9 system different constraints

```
/* http://poj.org/problem?id=2983 */
/*
## Problem
Given a system of inequations of the form x_j - x_i
     <= w_ij.
Find any solution x_1, x_2, ..., x_n or show that
     the system has no solution.

## Solution
```

```
We construct a n-vertex graph (vertext i represents
     variable x_i). For each inequation x_j - x_i
     <= w_ij,
we add an edge from i to j with weight w_ij.

If the graph has negative cycle, there's no
     solution.
Else, create a virtual vertex s, add edge with
     weight 0 from s to every x_i,
the solution is the shortest path from s to n
     vertices.
*/
typedef long long ll;

struct edge{
    int u, v, c;
};

/*
    check if negative cycle
*/
bool bellman_ford(int n, vector <edge> edges){
    int m = (int)edges.size();
    vector <ll> dist(n+1);
    for(int i = 1; i < n; i++)
        for(int j = 0; j < m; j++){
            int u = edges[j].u;
            int v = edges[j].v;
            int c = edges[j].c;
            if(dist[v] > dist[u] + c)
                dist[v] = dist[u] + c;
        }

    for(int j = 0; j < m; j++){
        int u = edges[j].u;
        int v = edges[j].v;
        int c = edges[j].c;
        if(dist[v] > dist[u] + c)
            return true;
    }
    return false;
}

void solve(int n, int m){
    vector <edge> edges;
    while(m--){
        char t;
        cin >> t;
        if(t == 'P'){
            int u, v, c;
            cin >> u >> v >> c;
            edges.push_back({u, v, c});
            edges.push_back({v, u, -c});
        }else{
            int u, v; cin >> u >> v;
            edges.push_back({v, u, -1});
        }
    }
    if(bellman_ford(n, edges))
        cout << "Unreliable" << '\n';
    else cout << "Reliable" << '\n';
}
```

6 Matrix

6.1 matrix

```
const int dim = 10;

struct matrix {
    vector<vector<long long>> a;
    matrix() {
        a.resize(dim);
        for (int i = 1; i < dim; i++)
            a[i].resize(dim, 0);
    }
};

matrix Identity() {
    matrix A;
    for (int i = 1; i < dim; i++)
        A.a[i][i] = 1;
    return A;
}

matrix operator*(const matrix &A, const matrix &B) {
    matrix mul;
    for (int k = 1; k < dim; k++)
        for (int i = 1; i < dim; i++)
            for (int j = 1; j < dim; j++)
                mul.a[i][j] += A.a[i][k] * B.a[k][j];
    return mul;
}

matrix fastPow(matrix A, long long b) {
    if (b == 0)
        return Identity();
    if (b == 1)
        return A;
    matrix t = fastPow(A, b / 2);
    t = t * t;
    if (b % 2 == 1)
        t = t * A;
    return t;
}
```

7 Misc

7.1 dates

```
//
// Time - Leap years
//

// A[i] has the accumulated number of days from
     months previous to i
const int A[13] = { 0, 0, 31, 59, 90, 120, 151,
     181, 212, 243, 273, 304, 334 };
// same as A, but for a leap year
const int B[13] = { 0, 0, 31, 60, 91, 121, 152,
     182, 213, 244, 274, 305, 335 };
// returns number of leap years up to, and
     including, y
int leap_years(int y) { return y / 4 - y / 100 + y
     / 400; }
bool is_leap(int y) { return y % 400 == 0 || (y % 4
     == 0 && y % 100 != 0); }
// number of days in blocks of years
const int p400 = 400*365 + leap_years(400);
const int p100 = 100*365 + leap_years(100);
const int p4  = 4*365 + 1;
const int p1  = 365;
int date_to_days(int d, int m, int y)
{
  return (y - 1) * 365 + leap_years(y - 1) +
       (is_leap(y) ? B[m] : A[m]) + d;
}
void days_to_date(int days, int &d, int &m, int &y)
{
  bool top100; // are we in the top 100 years of a
       400 block?
  bool top4;   // are we in the top 4 years of a
       100 block?
  bool top1;   // are we in the top year of a 4
       block?

  y = 1;
  top100 = top4 = top1 = false;

  y += ((days-1) / p400) * 400;
  d = (days-1) % p400 + 1;

  if (d > p100*3) top100 = true, d -= 3*p100, y +=
       300;
  else y += ((d-1) / p100) * 100, d = (d-1) % p100
       + 1;

  if (d > p4*24) top4 = true, d -= 24*p4, y += 24*4;
  else y += ((d-1) / p4) * 4, d = (d-1) % p4 + 1;
```

```
  if (d > p1*3) top1 = true, d -= p1*3, y += 3;
  else y += (d-1) / p1, d = (d-1) % p1 + 1;

  const int *ac = top1 && (!top4 || top100) ? B : A;
  for (m = 1; m < 12; ++m) if (d <= ac[m + 1])
      break;
  d -= ac[m];
}
```

7.2 fast input

```
inline char gc() { // like getchar()
  static char buf[1 << 16];
  static size_t bc, be;
  if (bc >= be) {
    buf[0] = 0, bc = 0;
    be = fread(buf, 1, sizeof(buf), stdin);
  }
  return buf[bc++]; // returns 0 on EOF
}
int readInt() {
  int a, c;
  while ((a = gc()) < 40);
  if (a == '-') return -readInt();
  while ((c = gc()) >= 48) a = a * 10 + c - 480;
  return a - 48;
}
```

7.3 fast knapsack

```
/**
 * Author: Mrten Wiman
 * License: CC0
 * Source: Pisinger 1999, "Linear Time Algorithms
     for Knapsack Problems with
 * Bounded Weights" Description: Given N
     non-negative integer weights w and a
 * non-negative target t, computes the maximum S <=
     t such that S is the sum of
 * some subset of the weights. Time: O(N \max(w_i))
     Status: Tested on
 * kattis:eavesdropperevasion, stress-tested
 */
#pragma once

int knapsack(vi w, int t) {
    int a = 0, b = 0, x;
    while (b < sz(w) && a + w[b] <= t)
        a += w[b++];
    if (b == sz(w))
        return a;
    int m = *max_element(all(w));
    vi u, v(2 * m, -1);
    v[a + m - t] = b;
    rep(i, b, sz(w)) {
        u = v;
        rep(x, 0, m) v[x + w[i]] = max(v[x + w[i]],
            u[x]);
        for (x = 2 * m; --x > m;)
            rep(j, max(0, u[x]), v[x]) v[x - w[j]] =
                max(v[x - w[j]], j);
    }
    for (a = t; v[a + m - t] < 0; a--)
        ;
    return a;
}
```

8 Number theory

8.1 convolution

```
typedef long long int LL;
typedef pair<LL, LL> PLL;

inline bool is_pow2(LL x) { return (x & (x - 1)) ==
     0; }

inline int ceil_log2(LL x) {
    int ans = 0;
    --x;
    while (x != 0) {
        x >>= 1;
        ans++;
    }
    return ans;
}

/* Returns the convolution of the two given vectors
     in time proportional to
 * n*log(n). The number of roots of unity to use
     nroots_unity must be set so
 * that the product of the first nroots_unity
     primes of the vector
 * nth_roots_unity is greater than the maximum
     value of the convolution. Never
 * use sizes of vectors bigger than 2^24, if you
     need to change the values of
 * the nth roots of unity to appropriate primes for
     those sizes.
 */
vector<LL> convolve(const vector<LL> &a, const
     vector<LL> &b,
                    int nroots_unity = 2) {
    int N = 1 << ceil_log2(a.size() + b.size());
    vector<LL> ans(N, 0), fA(N), fB(N), fC(N);
    LL modulo = 1;
    for (int times = 0; times < nroots_unity;
         times++) {
        fill(fA.begin(), fA.end(), 0);
        fill(fB.begin(), fB.end(), 0);
        for (int i = 0; i < a.size(); i++)
            fA[i] = a[i];
        for (int i = 0; i < b.size(); i++)
            fB[i] = b[i];
        LL prime = nth_roots_unity[times].first;
        LL inv_modulo = mod_inv(modulo % prime,
             prime);
        LL normalize = mod_inv(N, prime);
        ntfft(fA, 1, nth_roots_unity[times]);
        ntfft(fB, 1, nth_roots_unity[times]);
        for (int i = 0; i < N; i++)
            fC[i] = (fA[i] * fB[i]) % prime;
        ntfft(fC, -1, nth_roots_unity[times]);
        for (int i = 0; i < N; i++) {
            LL curr = (fC[i] * normalize) % prime;
            LL k = (curr - (ans[i] % prime) + prime)
                % prime;
            k = (k * inv_modulo) % prime;
            ans[i] += modulo * k;
        }
        modulo *= prime;
    }
    return ans;
}
```

8.2 crt

```
/**
 * Chinese remainder theorem.
 * 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;
}
```

8.3 diophantine equations

```
long long gcd(long long a, long long b, long long
    &x, long long &y) {
    if (a == 0) {
        x = 0;
        y = 1;
        return b;
    }
    long long x1, y1;
    long long d = gcd(b % a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return d;
}

bool find_any_solution(long long a, long long b,
    long long c, long long &x0,
                         long long &y0, long long &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0)
        x0 = -x0;
    if (b < 0)
        y0 = -y0;
    return true;
}

void shift_solution(long long &x, long long &y,
    long long a, long long b,
                     long long cnt) {
    x += cnt * b;
    y -= cnt * a;
}

long long find_all_solutions(long long a, long long
    b, long long c,
                             long long minx, long long
                                maxx, long long miny,
                             long long maxy) {
    long long x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;

    long long sign_a = a > 0 ? +1 : -1;
    long long sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    long long lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    long long rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    long long lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    long long rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    long long lx = max(lx1, lx2);
    long long rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

8.4 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;
}
```

8.5 ext euclidean

```
void ext_euclid(long long a, long long b, long long
    &x, long long &y,
                long long &g) {
    x = 0, y = 1, g = b;
    long long m, n, q, r;
    for (long long u = 1, v = 0; a != 0; g = a, a =
        r) {
        q = g / a, r = g % a;
        m = x - u * q, n = y - v * q;
        x = u, y = v, u = m, v = n;
    }
}
```

8.6 highest exponent factorial

```
int highest_exponent(int p, const int &n) {
    int ans = 0;
    int t = p;
    while (t <= n) {
        ans += n / t;
        t *= p;
    }
    return ans;
}
```

8.7 miller rabin

```
const int rounds = 20;
```

```
// checks whether a is a witness that n is not
     prime, 1 < a < n
bool witness(long long a, long long n) {
    // check as in Miller Rabin Primality Test
         described
    long long u = n - 1;
    int t = 0;
    while (u % 2 == 0) {
        t++;
        u >>= 1;
    }
    long long next = mod_pow(a, u, n);
    if (next == 1)
        return false;
    long long last;
    for (int i = 0; i < t; ++i) {
        last = next;
        next = mod_mul(last, last, n);
        if (next == 1) {
            return last != n - 1;
        }
    }
    return next != 1;
}

// Checks if a number is prime with prob 1 - 1 / (2
     ^ it)
//  D(miller_rabin(99999999999999997LL) == 1);
//  D(miller_rabin(9999999999971LL) == 1);
//  D(miller_rabin(7907) == 1);
bool miller_rabin(long long n, int it = rounds) {
    if (n <= 1)
        return false;
    if (n == 2)
        return true;
    if (n % 2 == 0)
        return false;
    for (int i = 0; i < it; ++i) {
        long long a = rand() % (n - 1) + 1;
        if (witness(a, n)) {
            return false;
        }
    }
    return true;
}
```

8.8 mod integer

```
template <class T, T mod> struct mint_t {
    T val;
    mint_t() : val(0) {}
    mint_t(T v) : val(v % mod) {}

    mint_t operator+(const mint_t &o) const {
         return (val + o.val) % mod; }
    mint_t operator-(const mint_t &o) const {
         return (val - o.val) % mod; }
    mint_t operator*(const mint_t &o) const {
         return (val * o.val) % mod; }
};

typedef mint_t<long long, 998244353> mint;
```

8.9 mod inv

```
long long mod_inv(long long n, long long m) {
    long long x, y, gcd;
    ext_euclid(n, m, x, y, gcd);
    if (gcd != 1)
        return 0;
    return (x + m) % m;
}
```

8.10 mod mul

```
// Computes (a * b) % mod
long long mod_mul(long long a, long long b, long
     long mod) {
    long long x = 0, y = a % mod;
    while (b > 0) {
        if (b & 1)
            x = (x + y) % mod;
        y = (y * 2) % mod;
        b /= 2;
    }
    return x % mod;
}
```

8.11 mod pow

```
// Computes ( a ^ exp ) % mod.
long long mod_pow(long long a, long long exp, long
     long mod) {
    long long ans = 1;
    while (exp > 0) {
        if (exp & 1)
            ans = mod_mul(ans, a, mod);
        a = mod_mul(a, a, mod);
        exp >>= 1;
    }
    return ans;
}
```

8.12 number theoretic transform

```
typedef long long int LL;
typedef pair<LL, LL> PLL;

/* The following vector of pairs contains pairs
     (prime, generator)
 * where the prime has an Nth root of unity for N
      being a power of two.
 * The generator is a number g s.t g^(p-1)=1 (mod p)
 * but is different from 1 for all smaller powers */
vector<PLL> nth_roots_unity{{1224736769,
     330732430}, {1711276033, 927759239},
                          {167772161, 167489322},
                              {469762049,
                              343261969},
                          {754974721, 643797295},
                              {1107296257,
                              883865065}};

PLL ext_euclid(LL a, LL b) {
    if (b == 0)
        return make_pair(1, 0);
    pair<LL, LL> rc = ext_euclid(b, a % b);
    return make_pair(rc.second, rc.first - (a / b)
         * rc.second);
}

// returns -1 if there is no unique modular inverse
LL mod_inv(LL x, LL modulo) {
    PLL p = ext_euclid(x, modulo);
    if ((p.first * x + p.second * modulo) != 1)
        return -1;
    return (p.first + modulo) % modulo;
}

// Number theory fft. The size of a must be a power
     of 2
void ntfft(vector<LL> &a, int dir, const PLL
     &root_unity) {
    int n = a.size();
    LL prime = root_unity.first;
    LL basew = mod_pow(root_unity.second, (prime -
         1) / n, prime);
    if (dir < 0)
        basew = mod_inv(basew, prime);
    for (int m = n; m >= 2; m >>= 1) {
```

```
        int mh = m >> 1;
        LL w = 1;
        for (int i = 0; i < mh; i++) {
            for (int j = i; j < n; j += m) {
                int k = j + mh;
                LL x = (a[j] - a[k] + prime) % prime;
                a[j] = (a[j] + a[k]) % prime;
                a[k] = (w * x) % prime;
            }
            w = (w * basew) % prime;
        }
        basew = (basew * basew) % prime;
    }
    int i = 0;
    for (int j = 1; j < n - 1; j++) {
        for (int k = n >> 1; k > (i ^= k); k >>= 1)
            ;
        if (j < i)
            swap(a[i], a[j]);
    }
}
```

8.13 pollard rho factorize

```
long long pollard_rho(long long n) {
    long long x, y, i = 1, k = 2, d;
    x = y = rand() % n;
    while (1) {
        ++i;
        x = mod_mul(x, x, n);
        x += 2;
        if (x >= n)
            x -= n;
        if (x == y)
            return 1;
        d = __gcd(abs(x - y), n);
        if (d != 1)
            return d;
        if (i == k) {
            y = x;
            k *= 2;
        }
    }
    return 1;
}

// Returns a list with the prime divisors of n
vector<long long> factorize(long long n) {
    vector<long long> ans;
    if (n == 1)
        return ans;
    if (miller_rabin(n)) {
        ans.push_back(n);
    } else {
        long long d = 1;
        while (d == 1)
            d = pollard_rho(n);
        vector<long long> dd = factorize(d);
        ans = factorize(n / d);
        for (int i = 0; i < dd.size(); ++i)
            ans.push_back(dd[i]);
    }
    return ans;
}
```

8.14 primes

```
namespace primes {
const int MP = 100001;
bool sieve[MP];
long long primes[MP];
int num_p;
void fill_sieve() {
    num_p = 0;
    sieve[0] = sieve[1] = true;
    for (long long i = 2; i < MP; ++i) {
        if (!sieve[i]) {
            primes[num_p++] = i;
            for (long long j = i * i; j < MP; j += i)
                sieve[j] = true;
        }
    }
}

// Finds prime numbers between a and b, using basic
    primes up to sqrt(b)
// a must be greater than 1.
vector<long long> seg_sieve(long long a, long long
    b) {
    long long ant = a;
    a = max(a, 3LL);
    vector<bool> pmap(b - a + 1);
    long long sqrt_b = sqrt(b);
    for (int i = 0; i < num_p; ++i) {
        long long p = primes[i];
        if (p > sqrt_b)
            break;
        long long j = (a + p - 1) / p;
        for (long long v = (j == 1) ? p + p : j * p;
             v <= b; v += p) {
            pmap[v - a] = true;
        }
    }
    vector<long long> ans;
    if (ant == 2)
        ans.push_back(2);
    int start = a % 2 ? 0 : 1;
    for (int i = start, I = b - a + 1; i < I; i +=
         2)
        if (pmap[i] == false)
            ans.push_back(a + i);
    return ans;
}

vector<pair<int, int>> factor(int n) {
    vector<pair<int, int>> ans;
    if (n == 0)
        return ans;
    for (int i = 0; primes[i] * primes[i] <= n;
         ++i) {
        if ((n % primes[i]) == 0) {
            int expo = 0;
            while ((n % primes[i]) == 0) {
                expo++;
                n /= primes[i];
            }
            ans.emplace_back(primes[i], expo);
        }
    }

    if (n > 1) {
        ans.emplace_back(n, 1);
    }
    return ans;
}
} // namespace primes
```

8.15 totient sieve

```
for (int i = 1; i < MN; i++)
    phi[i] = i;

for (int i = 1; i < MN; i++)
    if (!sieve[i]) // is prime
        for (int j = i; j < MN; j += i)
            phi[j] -= phi[j] / i;
```

8.16 totient

```
long long totient(long long n) {
    if (n == 1)
        return 0;
    long long ans = n;
```

```
    for (int i = 0; primes[i] * primes[i] <= n;
         ++i) {
        if ((n % primes[i]) == 0) {
            while ((n % primes[i]) == 0)
                n /= primes[i];
            ans -= ans / primes[i];
        }
    }
    if (n > 1) {
        ans -= ans / n;
    }
    return ans;
}
```

9 Strings

9.1 hashing codeforces

```
/**
 * Author: Simon Lindholm
 * Date: 2015-03-15
 * License: CC0
 * Source: own work
 * Description: Various self-explanatory methods
     for string hashing.
 * Use on Codeforces, which lacks 64-bit support
     and where solutions can be
 * hacked.
 */
#pragma once

static int C; // initialized below

// Arithmetic mod two primes and 2^32
     simultaneously.
// "typedef uint64_t H;" instead if Thue-Morse does
     not apply.
template <int M, class B> struct A {
    int x;
    B b;
    A(int x = 0) : x(x), b(x) {}
    A(int x, B b) : x(x), b(b) {}
    A operator+(A o) {
        int y = x + o.x;
        return {y - (y >= M) * M, b + o.b};
    }
    A operator-(A o) {
        int y = x - o.x;
        return {y + (y < 0) * M, b - o.b};
    }
    A operator*(A o) { return {(int)(1LL * x * o.x
         % M), b * o.b}; }
    explicit operator ull() { return x ^ (ull)b <<
         21; }
    bool operator==(A o) const { return (ull) *
         this == (ull)o; }
    bool operator<(A o) const { return (ull) * this
         < (ull)o; }
};
typedef A<1000000007, A<1000000009, unsigned>> H;

struct HashInterval {
    vector<H> ha, pw;
    HashInterval(string &str) : ha(sz(str) + 1),
         pw(ha) {
        pw[0] = 1;
        rep(i, 0, sz(str)) ha[i + 1] = ha[i] * C +
            str[i],
                               pw[i + 1] = pw[i] *
                                   C;
    }
    H hashInterval(int a, int b) { // hash [a, b)
        return ha[b] - ha[a] * pw[b - a];
    }
};

vector<H> getHashes(string &str, int length) {
    if (sz(str) < length)
        return {};
    H h = 0, pw = 1;
    rep(i, 0, length) h = h * C + str[i], pw = pw *
         C;
    vector<H> ret = {h};
    rep(i, length, sz(str)) {
        ret.pb(h = h * C + str[i] - pw * str[i -
            length]);
    }
    return ret;
}

H hashString(string &s) {
    H h{};
    for (char c : s)
        h = h * C + c;
    return h;
}

#include <sys/time.h>
int main() {
    timeval tp;
    gettimeofday(&tp, 0);
    C = (int)tp.tv_usec; // (less than modulo)
    assert((ull)(H(1) * 2 + 1 - 3) == 0);
    // ...
}
```

9.2 kmp

```
/**
 * Author: Johan Sannemo
 * Date: 2016-12-15
 * License: CC0
 * Description: pi[x] computes the length of the
     longest prefix of s that ends
 * at x, other than s[0...x] itself (abacaba ->
     0010123). Can be used to find
 * all occurrences of a string.
 * Time: O(n) Status:
 */
#pragma once

vi pi(const string &s) {
    vi p(sz(s));
    rep(i, 1, sz(s)) {
        int g = p[i - 1];
        while (g && s[i] != s[g])
            g = p[g - 1];
        p[i] = g + (s[i] == s[g]);
    }
    return p;
}
void compute_automaton(const string &s, vector<vi>
     &aut) {
    vi p = pi(s);
    aut.assign(sz(s), vi(26));
    rep(i, 0, sz(s)) rep(c, 0, 26) if (i > 0 &&
         s[i] != 'a' + c) aut[i][c] =
        aut[p[i - 1]][c];
    else aut[i][c] = i + (s[i] == 'a' + c);
}
vi match(const string &s, const string &pat) {
    vi p = pi(pat + '\0' + s), res;
    rep(i, sz(p) - sz(s), sz(p)) if (p[i] ==
         sz(pat)) res.emb(i - 2 * sz(pat));
    return res;
}
```

9.3 mancher

```
/**
 * Author: User adamant on CodeForces
 * Source: http://codeforces.com/blog/entry/12143
 * Description: 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).
 * Time: O(N) Status: Stress-tested
```

```
*/
#pragma once

array<vi, 2> manacher(const string &s) {
    int n = sz(s);
    array<vi, 2> 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;
}
```

9.4 minimal string rotation

```
// Lexicographically minimal string rotation
int lmsr() {
    string s;
    cin >> s;
    int n = s.size();
    s += s;
    vector<int> f(s.size(), -1);
    int k = 0;
    for (int j = 1; j < 2 * n; ++j) {
        int i = f[j - k - 1];
        while (i != -1 && s[j] != s[k + i + 1]) {
            if (s[j] < s[k + i + 1])
                k = j - i - 1;
            i = f[i];
        }
        if (i == -1 && s[j] != s[k + i + 1]) {
            if (s[j] < s[k + i + 1]) {
                k = j;
            }
            f[j - k] = -1;
        } else {
            f[j - k] = i + 1;
        }
    }
    return k;
}
```

9.5 suffix array

```
const int MAXN = 200005;

const int MAX_DIGIT = 256;
void countingSort(vector<int> &SA, vector<int> &RA,
    int k = 0) {
    int n = SA.size();
    vector<int> cnt(max(MAX_DIGIT, n), 0);
    for (int i = 0; i < n; i++)
        if (i + k < n)
            cnt[RA[i + k]]++;
        else
            cnt[0]++;
    for (int i = 1; i < cnt.size(); i++)
        cnt[i] += cnt[i - 1];
    vector<int> tempSA(n);
    for (int i = n - 1; i >= 0; i--)
        if (SA[i] + k < n)
            tempSA[--cnt[RA[SA[i] + k]]] = SA[i];
        else
            tempSA[--cnt[0]] = SA[i];
    SA = tempSA;
}

vector<int> constructSA(string s) {
    int n = s.length();
    vector<int> SA(n);
    vector<int> RA(n);
    vector<int> tempRA(n);
    for (int i = 0; i < n; i++) {
        RA[i] = s[i];
        SA[i] = i;
    }
    for (int step = 1; step < n; step <<= 1) {
        countingSort(SA, RA, step);
        countingSort(SA, RA, 0);
        int c = 0;
        tempRA[SA[0]] = c;
        for (int i = 1; i < n; i++) {
            if (RA[SA[i]] == RA[SA[i - 1]] &&
                RA[SA[i] + step] == RA[SA[i - 1] +
                    step])
                tempRA[SA[i]] = tempRA[SA[i - 1]];
            else
                tempRA[SA[i]] = tempRA[SA[i - 1]] +
                    1;
        }
        RA = tempRA;
        if (RA[SA[n - 1]] == n - 1)
            break;
    }
    return SA;
}
```

```
vector<int> computeLCP(const string &s, const
    vector<int> &SA) {
    int n = SA.size();
    vector<int> LCP(n), PLCP(n), c(n, 0);
    for (int i = 0; i < n; i++)
        c[SA[i]] = i;
    int k = 0;
    for (int j, i = 0; i < n - 1; i++) {
        if (c[i] - 1 < 0)
            continue;
        j = SA[c[i] - 1];
        k = max(k - 1, 0);
        while (i + k < n && j + k < n && s[i + k] ==
            s[j + k])
            k++;
        PLCP[i] = k;
    }
    for (int i = 0; i < n; i++)
        LCP[i] = PLCP[SA[i]];
    return LCP;
}
```

9.6 suffix automaton

```
/*
 * Suffix automaton:
 * This implementation was extended to maintain
    (online) the
 * number of different substrings. This is
    equivalent to compute
 * the number of paths from the initial state to
    all the other
 * states.
 *
 * The overall complexity is O(n)
 * can be tested here:
 *
    https://www.urionlinejudge.com.br/judge/en/problems/v
 * */

struct state {
    int len, link;
    long long num_paths;
    map<int, int> next;
};

const int MN = 200011;
state sa[MN << 1];
int sz, last;
long long tot_paths;
```

```
void sa_init() {
    sz = 1;
    last = 0;
    sa[0].len = 0;
    sa[0].link = -1;
    sa[0].next.clear();
    sa[0].num_paths = 1;
    tot_paths = 0;
}

void sa_extend(int c) {
    int cur = sz++;
    sa[cur].len = sa[last].len + 1;
    sa[cur].next.clear();
    sa[cur].num_paths = 0;
    int p;
    for (p = last; p != -1 && !sa[p].next.count(c);
         p = sa[p].link) {
        sa[p].next[c] = cur;
        sa[cur].num_paths += sa[p].num_paths;
        tot_paths += sa[p].num_paths;
    }

    if (p == -1) {
        sa[cur].link = 0;
    } else {
        int q = sa[p].next[c];
        if (sa[p].len + 1 == sa[q].len) {
            sa[cur].link = q;
        } else {
            int clone = sz++;
            sa[clone].len = sa[p].len + 1;
            sa[clone].next = sa[q].next;
            sa[clone].num_paths = 0;
            sa[clone].link = sa[q].link;
            for (; p != -1 && sa[p].next[c] == q; p
                 = sa[p].link) {
                sa[p].next[c] = clone;
                sa[q].num_paths -= sa[p].num_paths;
                sa[clone].num_paths +=
                     sa[p].num_paths;
            }
            sa[q].link = sa[cur].link = clone;
        }
    }
    last = cur;
}
```

9.7 z algorithm

```
using namespace std;
#include <bits/stdc++.h>

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

int main() {

    // string line;cin>>line;
    string line = "alfalfa";
    vector<int> z = compute_z(line);

    for (int i = 0; i < z.size(); ++i) {
        if (i)
            cout << " ";
        cout << z[i];
    }
    cout << endl;

    // must print "0 0 0 4 0 0 1"

    return 0;
}
```