

# Team notebook

January 18, 2016

## Contents

<b>1 Algorithms</b>	<b>1</b>
1.1 sliding window . . . . .	1
<b>2 Data structures</b>	<b>1</b>
2.1 hash table . . . . .	1
2.2 heavy light decomposition . . . . .	1
2.3 segment tree . . . . .	2
2.4 sparse table . . . . .	3
2.5 splay tree . . . . .	4
2.6 trie . . . . .	5
<b>3 Geometry</b>	<b>5</b>
3.1 squares . . . . .	5
<b>4 Graphs</b>	<b>6</b>
4.1 bridges . . . . .	6
4.2 directed mst . . . . .	7
4.3 eulerian path . . . . .	8
4.4 karp min mean cycle . . . . .	8
4.5 konig's theorem . . . . .	9
4.6 minimum path cover in DAG . . . . .	9
4.7 tarjan scc . . . . .	9
4.8 two sat (with kosaraju) . . . . .	10
<b>5 Math</b>	<b>11</b>
5.1 fft . . . . .	11
<b>6 Matrix</b>	<b>13</b>
6.1 matrix . . . . .	13

<b>7 Misc</b>	<b>13</b>
7.1 Template Java . . . . .	13
7.2 io . . . . .	14
<b>8 Number theory</b>	<b>15</b>
8.1 convolution . . . . .	15
8.2 crt . . . . .	15
8.3 discrete logarithm . . . . .	16
8.4 ext euclidean . . . . .	16
8.5 highest exponent factorial . . . . .	16
8.6 miller rabin . . . . .	16
8.7 mod inv . . . . .	17
8.8 mod mul . . . . .	17
8.9 mod pow . . . . .	17
8.10 number theoretic transform . . . . .	17
8.11 pollard rho factorize . . . . .	18
8.12 totient sieve . . . . .	18
8.13 totient . . . . .	19
<b>9 Strings</b>	<b>19</b>
9.1 minimal string rotation . . . . .	19
9.2 suffix array . . . . .	19
9.3 z algorithm . . . . .	20

## 1 Algorithms

### 1.1 sliding window

---

/\*

```

* Given an array ARR and an integer K, the problem boils down to
  computing for each index i: min(ARR[i], ARR[i-1], ..., ARR[i-K+1]).
* if mx == true, returns the maximum.
* http://people.cs.uct.ac.za/~ksmith/articles/sliding_window_minimum.html
* */

```

```

vector<int> sliding_window_minmax(vector<int> & ARR, int K, bool mx) {
    deque< pair<int, int> > window;
    vector<int> ans;
    for (int i = 0; i < ARR.size(); i++) {
        if (mx) {
            while (!window.empty() && window.back().first <= ARR[i])
                window.pop_back();
        } else {
            while (!window.empty() && window.back().first >= ARR[i])
                window.pop_back();
        }
        window.push_back(make_pair(ARR[i], i));

        while(window.front().second <= i - K)
            window.pop_front();

        ans.push_back(window.front().first);
    }
    return ans;
}

```

## 2 Data structures

### 2.1 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));
}
};

```

### 2.2 heavy light decomposition

// Heavy-Light Decomposition

```

struct TreeDecomposition {
    vector<int> g[MAXN], c[MAXN];
    int s[MAXN]; // subtree size
    int p[MAXN]; // parent id
    int r[MAXN]; // chain root id
    int t[MAXN]; // index used in segtree/bit/...
    int d[MAXN]; // depth
    int ts;
}

```

```

void dfs(int v, int f) {
    p[v] = f;
    s[v] = 1;
    if (f != -1) d[v] = d[f] + 1;
    else d[v] = 0;

    for (int i = 0; i < g[v].size(); ++i) {
        int w = g[v][i];
        if (w != f) {
            dfs(w, v);
            s[v] += s[w];
        }
    }
}

```

```

void hld(int v, int f, int k) {
    t[v] = ts++;
    c[k].push_back(v);
    r[v] = k;

    int x = 0, y = -1;
    for (int i = 0; i < g[v].size(); ++i) {
        int w = g[v][i];
        if (w != f) {

```

```

        if (s[w] > x) {
            x = s[w];
            y = w;
        }
    }
}
if (y != -1) {
    hld(y, v, k);
}

for (int i = 0; i < g[v].size(); ++i) {
    int w = g[v][i];
    if (w != f && w != y) {
        hld(w, v, w);
    }
}
}

void init(int n) {
    for (int i = 0; i < n; ++i) {
        g[i].clear();
    }
}

void add(int a, int b) {
    g[a].push_back(b);
    g[b].push_back(a);
}

void build() {
    ts = 0;
    dfs(0, -1);
    hld(0, 0, 0);
}
};

```

## 2.3 segment tree

```

/**
 * Taken from: http://codeforces.com/blog/entry/18051
 */

```

```

const int N = 1e5; // limit for array size

```

```

int n; // array size
int t[2 * N];

void build() { // build the tree
    for (int i = n - 1; i > 0; --i) t[i] = t[i<<1] + t[i<<1|1];
}

// Single modification, range query.
void modify(int p, int value) { // set value at position p
    for (t[p += n] = value; p > 1; p >>= 1) t[p>>1] = t[p] + t[p^1];
}

int query(int l, int r) { // sum on interval [l, r)
    int res = 0;
    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
        if (l&1) res += t[l++];
        if (r&1) res += t[--r];
    }
    return res;
}

// Range modification, single query.

void modify(int l, int r, int value) {
    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
        if (l&1) t[l++] += value;
        if (r&1) t[--r] += value;
    }
}

int query(int p) {
    int res = 0;
    for (p += n; p > 0; p >>= 1) res += t[p];
    return res;
}

/**
 * If at some point after modifications we need to inspect all the
 * elements in the array, we can push all the modifications to the
 * leaves using the following code. After that we can just traverse
 * elements starting with index n. This way we reduce the complexity
 * from O(n log(n)) to O(n) similarly to using build instead of n
 * modifications.
 */

```

```

void push() {
    for (int i = 1; i < n; ++i) {
        t[i<<1] += t[i];
        t[i<<1|1] += t[i];
        t[i] = 0;
    }
}

// Non commutative combiner functions.

void modify(int p, const S& value) {
    for (t[p += n] = value; p >>= 1; ) t[p] = combine(t[p<<1], t[p<<1|1]);
}

S query(int l, int r) {
    S resl, resr;
    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
        if (l&1) resl = combine(resl, t[l++]);
        if (r&1) resr = combine(t[--r], resr);
    }
    return combine(resl, resr);
}

// To be continued ...

```

## 2.4 sparse table

```

// RMQ.
const int MN = 100000 + 10; // Max number of elements
const int ML = 18; // ceil(log2(MN));

struct st {
    int data[MN];
    int M[MN][ML];
    int n;

    void read(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i)
            cin >> data[i];
    }

    void build() {

```

```

        for (int i = 0; i < n; ++i)
            M[i][0] = data[i];
        for (int j = 1, p = 2, q = 1; p <= n; ++j, p <= 1, q <= 1)
            for (int i = 0; i + p - 1 < n; ++i)
                M[i][j] = max(M[i][j - 1], M[i + q][j - 1]);
    }
    int query(int b, int e) {
        int k = log2(e - b + 1);
        return max(M[b][k], M[e + 1 - (1<<k)][k]);
    }
};

```

## 2.5 splay tree

```

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

typedef int T;

struct node{
    node *left, *right, *parent;
    T key;
    node (T k) : key(k), left(0), right(0), parent(0) {}
};

struct splay_tree{

    node *root;

    void right_rot(node *x) {
        node *p = x->parent;
        if (x->parent = p->parent) {
            if (x->parent->left == p) x->parent->left = x;
            if (x->parent->right == p) x->parent->right = x;
        }
        if (p->left = x->right) p->left->parent = p;
        x->right = p;
        p->parent = x;
    }

    void left_rot(node *x) {

```

```

node *p = x->parent;
if (x->parent == p->parent) {
    if (x->parent->left == p) x->parent->left = x;
    if (x->parent->right == p) x->parent->right = x;
}
if (p->right == x->left) p->right->parent = p;
x->left = p;
p->parent = x;
}

```

```

void splay(node *x, node *fa = 0) {

```

```

    while( x->parent != fa and x->parent != 0) {
        node *p = x->parent;
        if (p->parent == fa)
            if (p->right == x)
                left_rot(x);
            else
                right_rot(x);
        else {
            node *gp = p->parent; //grand parent
            if (gp->left == p)
                if (p->left == x)
                    right_rot(x), right_rot(x);
                else
                    left_rot(x), right_rot(x);
            else
                if (p->left == x)
                    right_rot(x), left_rot(x);
                else
                    left_rot(x), left_rot(x);
        }
    }
    if (fa == 0) root = x;
}

```

```

void insert(T key) {
    node *cur = root;
    node *pcur = 0;
    while (cur) {
        pcur = cur;
        if (key > cur->key) cur = cur->right;
        else cur = cur->left;
    }
    cur = new node(key);

```

```

        cur->parent = pcur;
        if (!pcur) root = cur;
        else if (key > pcur->key) pcur->right = cur;
        else pcur->left = cur;
        splay(cur);
    }

```

```

node *find(T key) {
    node *cur = root;
    while (cur) {
        if (key > cur->key) cur = cur->right;
        else if (key < cur->key) cur = cur->left;
        else return cur;
    }
    return 0;
}

```

```

splay_tree(){ root = 0;};
};

```

---

## 2.6 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 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;
}

```

---

## 4 Graphs

### 4.1 bridges

```

struct edge{
    int to, id;
    edge(int a, int b) : to(a), id(b) {}
};

struct graph {
    vector<vector<edge> > g;

```

```

    vector<int> vi, low, d, pi, is_b;

    int ticks, edges;

    graph(int n, int m) {
        g.assign(n, vector<edge>());
        is_b.assign(m, 0);
        vi.resize(n);
        low.resize(n);
        d.resize(n);
        pi.resize(n);
        edges = 0;
    }

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

    void dfs(int u) {
        vi[u] = true;
        d[u] = low[u] = ticks++;
        for (int i = 0; i < g[u].size(); ++i) {
            int v = g[u][i].to;
            if (v == pi[u]) continue;
            if (!vi[v]) {
                pi[v] = u;
                dfs(v);
                if (d[u] < low[v])
                    is_b[g[u][i].id] = true;

                low[u] = min(low[u], low[v]);
            } else {
                low[u] = min(low[u], d[v]);
            }
        }
    }

    void comp_bridges() {
        fill(pi.begin(), pi.end(), -1);
        fill(vi.begin(), vi.end(), 0);
        fill(low.begin(), low.end(), 0);
        fill(d.begin(), d.end(), 0);
    }
}

```

```

    ticks = 0;
    for (int i = 0; i < g.size(); ++i)
        if (!vi[i]) dfs(i);
}
};

```

---

## 4.2 directed mst

---

```

const int inf = 1000000 + 10;

struct edge {
    int u, v, w;
    edge() {}
    edge(int a, int b, int c) : u(a), v(b), w(c) {}
};

/**
 * Computes the minimum spanning tree for a directed graph
 * - edges : Graph description in the form of list of edges.
 *   each edge is: From node u to node v with cost w
 * - root : Id of the node to start the DMST.
 * - n : Number of nodes in the graph.
 */

int dmst(vector<edge> &edges, int root, int n) {
    int ans = 0;
    int cur_nodes = n;
    while (true) {
        vector<int> lo(cur_nodes, inf), pi(cur_nodes, inf);
        for (int i = 0; i < edges.size(); ++i) {
            int u = edges[i].u, v = edges[i].v, w = edges[i].w;
            if (w < lo[v] and u != v) {
                lo[v] = w;
                pi[v] = u;
            }
        }

        lo[root] = 0;
        for (int i = 0; i < lo.size(); ++i) {
            if (i == root) continue;
            if (lo[i] == inf) return -1;
        }
        int cur_id = 0;

```

```

        vector<int> id(cur_nodes, -1), mark(cur_nodes, -1);
        for (int i = 0; i < cur_nodes; ++i) {
            ans += lo[i];
            int u = i;
            while (u != root and id[u] < 0 and mark[u] != i) {
                mark[u] = i;
                u = pi[u];
            }
            if (u != root and id[u] < 0) { // Cycle
                for (int v = pi[u]; v != u; v = pi[v])
                    id[v] = cur_id;
                id[u] = cur_id++;
            }
        }

        if (cur_id == 0)
            break;

        for (int i = 0; i < cur_nodes; ++i)
            if (id[i] < 0) id[i] = cur_id++;

        for (int i = 0; i < edges.size(); ++i) {
            int u = edges[i].u, v = edges[i].v, w = edges[i].w;
            edges[i].u = id[u];
            edges[i].v = id[v];
            if (id[u] != id[v])
                edges[i].w -= lo[v];
        }
        cur_nodes = cur_id;
        root = id[root];
    }

    return ans;
}

```

---

## 4.3 eulerian path

---

// Taken from <https://github.com/lbv/pc-code/blob/master/code/graph.cpp>  
// Eulerian Trail

```

struct Euler {
    ELV adj; IV t;
    Euler(ELV Adj) : adj(Adj) {}

```



```

void build(int u) {
    while(! adj[u].empty()) {
        int v = adj[u].front().v;
        adj[u].erase(adj[u].begin());
        build(v);
    }
    t.push_back(u);
}
};
bool eulerian_trail(IV &trail) {
    Euler e(adj);
    int odd = 0, s = 0;
    /*
        for (int v = 0; v < n; v++) {
            int diff = abs(in[v] - out[v]);
            if (diff > 1) return false;
            if (diff == 1) {
                if (++odd > 2) return false;
                if (out[v] > in[v]) start = v;
            }
        }
    */
    e.build(s);
    reverse(e.t.begin(), e.t.end());
    trail = e.t;
    return true;
}

```

## 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 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  $G' = (V_{out} \cup V_{in}, E')$  from  $G$ , where :

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

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

$$E' = \{(u, v) \in V_{out} \times V_{in} : (u, v) \in E\}$$

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 matching in  $G'$ .

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

**NOTE:** If the paths are not necessarily disjoint, find the transitive closure and solve the problem for disjoint paths.

## 4.7 tarjan scc

---

```

const int MN = 20002;

struct tarjan_scc {

```

```

    int scc[MN], low[MN], d[MN], stacked[MN];
    int ticks, current_scc;
    deque<int> s; // used as stack.

    tarjan_scc() {}

    void init () {
        memset(scc, -1, sizeof scc);
        memset(d, -1, sizeof d);
        memset(stacked, 0, sizeof stacked);
        s.clear();
        ticks = current_scc = 0;
    }

    void compute(vector<vector<int> > &g, int u) {
        d[u] = low[u] = ticks++;
        s.push_back(u);
        stacked[u] = true;
        for (int i = 0; i < g[u].size(); ++i) {
            int v = g[u][i];
            if (d[v] == -1)
                compute(g, v);
            if (stacked[v]) {
                low[u] = min(low[u], low[v]);
            }
        }

        if (d[u] == low[u]) { // root
            int v;
            do {
                v = s.back(); s.pop_back();
                stacked[v] = false;
                scc[v] = current_scc;
            } while (u != v);
            current_scc++;
        }
    }
};

```

---

## 4.8 two sat (with kosaraju)

---

```

/**
 * 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
**/

```

```

#include<bits/stdc++.h>
using namespace std;
#define MAX 100000
#define endl '\n'

```

```

vector<int> G[MAX];
vector<int> GT[MAX];
vector<int> Ftime;
vector<vector<int>> > SCC;
bool visited[MAX];
int n;

```

```

void dfs1(int n){
    visited[n] = 1;

    for (int i = 0; i < G[n].size(); ++i) {
        int curr = G[n][i];
        if (visited[curr]) continue;
        dfs1(curr);
    }
}

```

```

Ftime.push_back(n);
}

```

```

void dfs2(int n, vector<int> &scc) {
    visited[n] = 1;
    scc.push_back(n);

    for (int i = 0; i < GT[n].size(); ++i) {
        int curr = GT[n][i];
        if (visited[curr]) continue;
        dfs2(curr, scc);
    }
}

```

```

void kosaraju() {
    memset(visited, 0, sizeof visited);

```

```

    for (int i = 0; i < 2 * n ; ++i) {

```

```

        if (!visited[i]) dfs1(i);
    }

```

```

    memset(visited, 0, sizeof visited);
    for (int i = Ftime.size() - 1; i >= 0; i--) {
        if (visited[Ftime[i]]) continue;
        vector<int> _scc;
        dfs2(Ftime[i], _scc);
        SCC.push_back(_scc);
    }
}

```

```

/**
 * After having the SCC, we must traverse each scc, if in one SCC are -b
 * y b, there is not a solution.
 * Otherwise we build a solution, making the first "node" that we find
 * truth and its complement false.
 */

```

```

bool two_sat(vector<int> &val) {
    kosaraju();
    for (int i = 0; i < SCC.size(); ++i) {
        vector<bool> tmpvisited(2 * n, false);
        for (int j = 0; j < SCC[i].size(); ++j) {
            if (tmpvisited[SCC[i][j] ^ 1]) return 0;
            if (val[SCC[i][j]] != -1) continue;
            else {
                val[SCC[i][j]] = 0;
                val[SCC[i][j] ^ 1] = 1;
            }
            tmpvisited[SCC[i][j]] = 1;
        }
    }
    return 1;
}

```

```

// Example of use

```

```

int main() {

    int m, u, v, nc = 0, t; cin >> t;
    // n = "nodes" number, m = clauses number

    while (t--) {

```

```

cin >> m >> n;
Ftime.clear();
SCC.clear();
for (int i = 0; i < 2 * n; ++i) {
    G[i].clear();
    GT[i].clear();
}

// (a1 v a2) = (a1 -> a2) = (a2 -> a1)
for (int i = 0; i < m; ++i) {
    cin >> u >> v;
    int t1 = abs(u) - 1;
    int t2 = abs(v) - 1;
    int p = t1 * 2 + ((u < 0)? 1 : 0);
    int q = t2 * 2 + ((v < 0)? 1 : 0);
    G[p ^ 1].push_back(q);
    G[q ^ 1].push_back(p);
    GT[p].push_back(q ^ 1);
    GT[q].push_back(p ^ 1);
}

vector<int> val(2 * n, -1);
cout << "Case " << ++nc << ": ";
if (two_sat(val)) {
    cout << "Yes" << endl;
    vector<int> sol;
    for (int i = 0; i < 2 * n; ++i)
        if (i % 2 == 0 and val[i] == 1)
            sol.push_back(i / 2 + 1);
    cout << sol.size() ;

    for (int i = 0; i < sol.size(); ++i) {
        cout << " " << sol[i];
    }
    cout << endl;
} else {
    cout << "No" << endl;
}
}
return 0;
}

```

## 5 Math

### 5.1 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;
}

```

## 6 Matrix

### 6.1 matrix

```

const int MN = 111;
const int mod = 10000;

struct matrix {
    int r, c;
    int m[MN][MN];

    matrix (int _r, int _c) : r (_r), c (_c) {
        memset(m, 0, sizeof m);
    }

    void print() {
        for (int i = 0; i < r; ++i) {
            for (int j = 0; j < c; ++j)
                cout << m[i][j] << " ";
            cout << endl;
        }
    }

    int x[MN][MN];
    matrix & operator *= (const matrix &o) {
        memset(x, 0, sizeof x);
        for (int i = 0; i < r; ++i)
            for (int k = 0; k < c; ++k)
                if (m[i][k] != 0)
                    for (int j = 0; j < c; ++j) {
                        x[i][j] = (x[i][j] + ((m[i][k] * o.m[k][j]) % mod) ) % mod;
                    }
        memcpy(m, x, sizeof(m));
        return *this;
    }
};

void matrix_pow(matrix b, long long e, matrix &res) {
    memset(res.m, 0, sizeof res.m);
    for (int i = 0; i < b.r; ++i)
        res.m[i][i] = 1;

    if (e == 0) return;
    while (true) {
        if (e & 1) res *= b;
        if ((e >>= 1) == 0) break;
        b *= b;
    }
}

```

## 7 Misc

### 7.1 Template Java

---

```

import java.io.*;
import java.util.StringTokenizer;

public class Template {

    public static void main(String []args) throws IOException {
        Scanner in = new Scanner(System.in);
        OutputWriter out = new OutputWriter(System.out);
        Task solver = new Task();
        solver.solve(in, out);
        out.close();
    }

}

class Task{
    public void solve(Scanner in, OutputWriter out){

    }
}

class Scanner{
    public BufferedReader reader;
    public StringTokenizer st;

    public Scanner(InputStream stream){
        reader = new BufferedReader(new InputStreamReader(stream));
        st = null;
    }

    public String next(){
        while(st == null || !st.hasMoreTokens()){
            try{
                String line = reader.readLine();
                if(line == null) return null;
                st = new StringTokenizer(line);
            }catch (Exception e){
                throw (new RuntimeException());
            }
        }
    }
}

```

```

        return st.nextToken();
    }

    public int nextInt(){
        return Integer.parseInt(next());
    }
    public long nextLong(){
        return Long.parseLong(next());
    }
    public double nextDouble(){
        return Double.parseDouble(next());
    }
}

class OutputWriter{
    BufferedWriter writer;

    public OutputWriter(OutputStream stream){
        writer = new BufferedWriter(new OutputStreamWriter(stream));
    }

    public void print(int i) throws IOException {
        writer.write(i);
    }

    public void print(String s) throws IOException {
        writer.write(s);
    }

    public void print(char []c) throws IOException {
        writer.write(c);
    }
    public void close() throws IOException {
        writer.close();
    }
}

```

## 7.2 io

// taken from :  
<https://github.com/lbv/pc-code/blob/master/solved/c-e/diablo/diablo.cpp>

```

// this is very fast as well :
https://github.com/lbv/pc-code/blob/master/code/input.cpp

typedef unsigned int u32;
#define BUF 524288
struct Reader {
    char buf[BUF]; char b; int bi, bz;
    Reader() { bi=bz=0; read(); }
    void read() {
        if (bi==bz) { bi=0; bz = fread(buf, 1, BUF, stdin); }
        b = bz ? buf[bi++] : 0; }
    void skip() { while (b > 0 && b <= 32) read(); }
    u32 next_u32() {
        u32 v = 0; for (skip(); b > 32; read()) v = v*10 + b-48; return v; }
    int next_int() {
        int v = 0; bool s = false;
        skip(); if (b == '-') { s = true; read(); }
        for (; 48<=b&&b<=57; read()) v = v*10 + b-48; return s ? -v : v; }
    char next_char() { skip(); char c = b; read(); return c; }
};

```

## 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 discrete logarithm

// Computes x which  $a^x = b \bmod n$ .

```

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;
}

```



```
}
```

---

## 8.4 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;
    }
}
```

---

## 8.5 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;
}
```

---

## 8.6 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(9999999999999997LL) == 1);
// D(miller_rabin(999999999999971LL) == 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;
}
```

---

## 8.7 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;
}
```

---

## 8.8 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;
}
```

---

## 8.9 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;
}
```

---

## 8.10 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]);
    }
}
```

---

## 8.11 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;
}
```

---

## 8.12 totient sieve

---

```
for (int i = 0; i < MP; ++i) phi[i] = i;
```

```
for (int i = 0; primes[i] <= 5000000; ++i) {
    phi[primes[i]] = primes[i] - 1;
    for (int j = 2 * primes[i]; j <= 5000000; j += primes[i]) {
        phi[j] = phi[j] * (primes[i] - 1);
        phi[j] = phi[j] / primes[i];
    }
}
```

---

## 8.13 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 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;
}

```

## 9.2 suffix array

```

/**
 * O (n log^2 (n))
 * See http://web.stanford.edu/class/cs97si/suffix-array.pdf for reference
 */

struct entry{
    int a, b, p;
    entry(){}
    entry(int x, int y, int z): a(x), b(y), p(z){}
    bool operator < (const entry &o) const {
        return (a == o.a) ? (b == o.b) ? (p < o.p) : (b < o.b) : (a < o.a);
    }
};

struct SuffixArray{
    const int N;
    string s;
    vector<vector<int>> > P;
    vector<entry> M;

    SuffixArray(const string &s) : N(s.length()) , s(s), P(1, vector<int>
        (N, 0)), M(N) {
        for (int i = 0; i < N; ++i)

```

```

        P[0][i] = (int) s[i];

        for (int skip = 1, level = 1; skip < N; skip *= 2, level++) {
            P.push_back(vector<int>(N, 0));
            for (int i = 0 ; i < N; ++i) {
                int next = ((i + skip) < N) ? P[level - 1][i + skip] : -10000;
                M[i] = entry(P[level - 1][i], next, i);
            }
            sort(M.begin(), M.end());
            for (int i = 0; i < N; ++i)
                P[level][M[i].p] = (i > 0 and M[i].a == M[i - 1].a and M[i].b ==
                    M[i - 1].b) ? P[level][M[i - 1].p] : i;
        }
    }

    vector<int> getSuffixArray(){
        vector<int> &rank = P.back();
        vector<pair<int, int> > inv(rank.size());
        for (int i = 0; i < rank.size(); ++i)
            inv[i] = make_pair(rank[i], i);
        sort(inv.begin(), inv.end());
        vector<int> sa(rank.size());
        for (int i = 0; i < rank.size(); ++i)
            sa[i] = inv[i].second;
        return sa;
    }

    // returns the length of the longest common prefix of s[i...L-1] and
    // s[j...L-1]
    int lcp(int i, int j) {
        int len = 0;
        if (i == j) return N - i;
        for (int k = P.size() - 1; k >= 0 && i < N && j < N; --k) {
            if (P[k][i] == P[k][j]) {
                i += 1 << k;
                j += 1 << k;
                len += 1 << k;
            }
        }
        return len;
    }
};

```

## 9.3 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 - 1] == 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;
}
```