Team notebook

HCMUS-IdentityImbalance

December 2, 2021

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;
}
```

2 DP Optimizations

2.1 convex hull trick

```
/**
 * Problems:
 *   http://codeforces.com/problemset/problem/319/C
 *   http://codeforces.com/contest/311/problem/B
 *   https://csacademy.com/contest/archive/task/squared-ends
 *   http://codeforces.com/contest/932/problem/F
 * */

struct line {
  long long m, b;
  line (long long a, long long c) : m(a), b(c) {}
  long long eval(long long x) {
    return m * x + b;
  }
};

long double inter(line a, line b) {
  long double den = a.m - b.m;
  long double num = b.b - a.b;
  return num / den;
}

/**
 * min m_i * x_j + b_i, for all i.
 *     x_j <= x_{j + 1}
 *     m_i >= m_{j + 1}
 * */
struct ordered_cht {
  vector<line> ch;
  int idx; // id of last "best" in query
  ordered_cht() {
    idx = 0;
  }
```

```
  void insert_line(long long m, long long b) {
    line cur(m, b);
    // new line's slope is less than all the previous
    while (ch.size() > 1 &&
       (inter(cur, ch[ch.size() - 2]) >= inter(cur, ch[ch.size() - 1]))) {
        // f(x) is better in interval [inter(ch.back(), cur), inf)
        ch.pop_back();
    }

    ch.push_back(cur);
  }

  long long eval(long long x) { // minimum
    // current x is greater than all the previous x,
    // if that is not the case we can make binary search.
    idx = min<int>(idx, ch.size() - 1);
    while (idx + 1 < (int)ch.size() && ch[idx + 1].eval(x) <=
         ch[idx].eval(x))
      idx++;

    return ch[idx].eval(x);
  }
};


/**
 * Dynammic convex hull trick
 * */

typedef long long int64;
typedef long double float128;

const int64 is_query = -(1LL<<62), inf = 1e18;

struct Line {
  int64 m, b;
  mutable function<const Line*()> succ;
  bool operator<(const Line& rhs) const {
    if (rhs.b != is_query) return m < rhs.m;
    const Line* s = succ();
    if (!s) return 0;
    int64 x = rhs.m;
    return b - s->b < (s->m - m) * x;
  }
};

struct HullDynamic : public multiset<Line> { // will maintain upper hull
    for maximum
  bool bad(iterator y) {
    auto z = next(y);
    if (y == begin()) {
      if (z == end()) return 0;
      return y->m == z->m && y->b <= z->b;
    }
    auto x = prev(y);
    if (z == end()) return y->m == x->m && y->b <= x->b;
    return (float128)(x->b - y->b)*(z->m - y->m) >= (float128)(y->b -
        z->b)*(y->m - x->m);
  }
  void insert_line(int64 m, int64 b) {
    auto y = insert({ m, b });
    y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
    if (bad(y)) { erase(y); return; }
    while (next(y) != end() && bad(next(y))) erase(next(y));
    while (y != begin() && bad(prev(y))) erase(prev(y));
  }

  int64 eval(int64 x) {
    auto l = *lower_bound((Line) { x, is_query });
    return l.m * x + l.b;
  }
};
```

2.2 divide and conquer

```
/**
 * recurrence:
 *    dp[k][i] = min dp[k-1][j] + c[i][j - 1], for all j > i;
 *
 * "comp" computes dp[k][i] for all i in O(n log n) (k is fixed)
 *
 * Problems:
 *  https://icpc.kattis.com/problems/branch
 *  http://codeforces.com/contest/321/problem/E
 * */

void comp(int l, int r, int le, int re) {
```

```
  if (l > r) return;

  int mid = (l + r) >> 1;

  int best = max(mid + 1, le);
  dp[cur][mid] = dp[cur ^ 1][best] + cost(mid, best - 1);
  for (int i = best; i <= re; i++) {
    if (dp[cur][mid] > dp[cur ^ 1][i] + cost(mid, i - 1)) {
      best = i;
      dp[cur][mid] = dp[cur ^ 1][i] + cost(mid, i - 1);
    }
  }

  comp(l, mid - 1, le, best);
  comp(mid + 1, r, best, re);
}
```

2.3 dp on trees

```
/**
 * This trick is very useful when doing DP on trees, basically, you can
     save
 * the answer for each node as if it was the root of the tree. Partial
     results
 * are also stored in order to query subtrees (taking the root and
     exclude some
 * child).
 *
 * problems:
 *   - http://codeforces.com/gym/101161, problem I : Sky tax
 *   - http://codeforces.com/contest/791/problem/D
 * */


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

struct state {
  bool seen;
  long long missing;
  long long total;
  vector<long long> partial;

  state() { clear(); }

  void clear() {
    seen = false;
    missing = 0;
    total = 0;
    partial.clear();
  }
};

void add_edge(int u, int v) {
  int id_u_v = g[u].size();
  int id_v_u = g[v].size();
  g[u].emplace_back(v, id_v_u); // id of the parent in the child's list
      (g[v][id] -> u)
  g[v].emplace_back(u, id_u_v); // id of the parent in the child's list
      (g[u][id] -> v)
}


int go(int node, int id_parent) {

  state &s = dp[node];

  if (!s.seen) {
    int ans = 1;
    s.partial.assign(g[node].size(), 0); // create the list of partial
        results.
    for (int i = 0; i < int(g[node].size()); i++) {
      int to = g[node][i].to;
      int pid = g[node][i].p_id;
      if (i != id_parent) {
        int tmp = go(to, pid);
        ans += tmp;
        s.partial[i] = tmp;
      }
    }

    s.missing = id_parent;
    s.total = ans;
    s.seen = true;

    return ans;
```

```
  } else {

    if (s.missing == id_parent) { // the same id_parent than before, so
        we can not complete the results yet
      return s.total;
    }

    if (s.missing != -1) { // only one missing and is different of
        'id_parent'
      int tmp = go(g[node][s.missing].to, g[node][s.missing].p_id);
      s.partial[s.missing] = tmp;
      s.total += tmp;
      s.missing = -1;
    }

    int extra = (id_parent == -1) ? 0 : s.partial[id_parent];
    return s.total - extra;
  }
}
```

3 Data structures

3.1 STL Treap

```
#include <ext/rope> //header with rope
using namespace std;
using namespace __gnu_cxx; //namespace with rope and some additional stuff
int main()
{
    ios_base::sync_with_stdio(false);
    rope <int> v; //use as usual STL container
    int n, m;
    cin >> n >> m;
    for(int i = 1; i <= n; ++i)
        v.push_back(i); //initialization
    int l, r;
    for(int i = 0; i < m; ++i)
    {
        cin >> l >> r;
        --l, --r;
        rope <int> cur = v.substr(l, r - l + 1);
        v.erase(l, r - l + 1);
        v.insert(v.mutable_begin(), cur);
    }
    for(rope <int>::iterator it = v.mutable_begin(); it !=
        v.mutable_end(); ++it)
        cout << *it << " ";
    return 0;
}
```

3.2 STL order statistics tree II

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

typedef tree<int,null_type,less<int>,rb_tree_tag,
tree_order_statistics_node_update> order_set;

order_set X;

int get(int y) {
  int l=0,r=1e9+1;
  while(l<r) {
    int m=l+((r-l)>>1);
    if(m-X.order_of_key(m+1)<y)
      l=m+1;
    else
      r=m;
  }
  return l;
}

main(){
  ios::sync_with_stdio(0);
  cin.tie(0);
  int n,m;
  cin>>n>>m;

  for(int i=0;i<m;i++) {
    char a;
    int b;
```

```
    cin>>a>>b;
    if(a=='L')
      cout<<get(b)<<endl;
    else
      X.insert(get(b));
  }
}

/***
Input
20 7
L 5
D 5
L 4
L 5
D 5
L 4
L 5

Output
5
4
6
4
7
***/
```

3.3 STL order statistics tree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <bits/stdc++.h>

using namespace __gnu_pbds;
using namespace std;

typedef
tree<
  pair<int,int>,
  null_type,
  less<pair<int,int>>,
  rb_tree_tag,
  tree_order_statistics_node_update>
ordered_set;

main()
 {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int n;
    int sz=0;
    cin>>n;
    vector<int> ans(n,0);

    ordered_set t;
    int x,y;
    for(int i=0;i<n;i++)
    {
        cin>>x>>y;
        ans[t.order_of_key({x,++sz})]++;
        t.insert({x,sz});
    }

    for(int i=0;i<n;i++)
        cout<<ans[i]<<'\n';
}

/***
Input
5
1 1
5 1
7 1
3 3
5 5

Output
1
2
1
1
0
***/
```

3.4 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;
        }
    }
};
```

3.5 fenwick tree

```
template <typename T>
class FenwickTree{
  vector <T> fenw;
  int n;
public:
  void initialize(int _n){
    this->n = _n;
    fenw.resize(n+1);
  }

  void update(int id, T val) {
    while (id <= n) {
      fenw[id] += val;
      id += id&(-id);
    }
  }

  T get(int id){
    T ans{};
    while(id >= 1){
      ans += fenw[id];
      id -= id&(-id);
    }
    return ans;
  }
};
```

3.6 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));
  }
};
```

3.7 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]; // depht
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);
  }
};
```

3.8 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);
}
```

3.9 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];
```

3.10 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);
}
```

3.11 segment tree

```cpp
// 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, l, r);
    }
#undef m
#undef lc
#undef rc
};
```

3.12 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]);
    }
};
```

3.13 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;};
};
```

3.14 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;
  }

};
```

3.15 wavelet tree

```
// this can be tested in the problem:
    http://www.spoj.com/problems/ILKQUERY/

struct wavelet {
  vector<int> values, ori;
  vector<int> map_left, map_right;
  int l, r, m;
  wavelet *left, *right;
  wavelet() : left(NULL), right(NULL) {}
  wavelet(int a, int b, int c) : l(a), r(b), m(c), left(NULL),
      right(NULL) {}
};

wavelet *init(vector<int> &data, vector<int> &ind, int lo, int hi) {
  if (lo > hi || (data.size() == 0)) return NULL;
  int mid = ((long long)(lo) + hi) / 2;
  if (lo + 1 == hi) mid = lo; // handle negative values

  wavelet *node = new wavelet(lo, hi, mid);
  vector<int> data_l, data_r, ind_l, ind_r;
  int ls = 0, rs = 0;
  for (int i = 0; i < int(data.size()); i++) {
    int value = data[i];
    if (value <= mid) {
      data_l.emplace_back(value);
      ind_l.emplace_back(ind[i]);
      ls++;
    } else {
      data_r.emplace_back(value);
      ind_r.emplace_back(ind[i]);
      rs++;
    }
    node->map_left.emplace_back(ls);
    node->map_right.emplace_back(rs);
    node->values.emplace_back(value);
    node->ori.emplace_back(ind[i]);
  }

  if (lo < hi) {
    node->left = init(data_l, ind_l, lo, mid);
    node->right = init(data_r, ind_r, mid + 1, hi);
  }
  return node;
}

int kth(wavelet *node, int to, int k) {
  // returns the kth element in the sorted version of (a[0], ..., a[to])
  if (node->l == node->r) return node->m;
  int c = node->map_left[to];
  if (k < c)
    return kth(node->left, c - 1, k);
  return kth(node->right, node->map_right[to] - 1, k - c);
}

int pos_kth_ocurrence(wavelet *node, int val, int k) {
  // returns the position on the original array of the kth ocurrence of
      the value "val"
  if (!node) return -1;

  if (node->l == node->r) {
    if (int(node->ori.size()) <= k)
      return -1;
    return node->ori[k];
  }
```

```
  if (val <= node->m)
    return pos_kth_ocurrence(node->left, val, k);
  return pos_kth_ocurrence(node->right, val, k);
}
```

4 Geometry

4.1 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;
}
```

4.2 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);
}
```

4.3 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;
}
```

4.4 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:

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

5 Graphs

5.1 SCC kosaraju

```
struct SCC {
  vector<vector<int> > g, gr;
  vector<bool> used;
  vector<int> order, component;
  int total_components;

  SCC(vector<vector<int> > &adj) {
    g = adj;
    int n = g.size();
    gr.resize(n);
    for (int i = 0; i < n; i++)
      for (auto to : g[i])
        gr[to].push_back(i);

    used.assign(n, false);
    for (int i = 0; i < n; i++)
      if (!used[i])
        GenTime(i);

    used.assign(n, false);
    component.assign(n, -1);
    total_components = 0;
    for (int i = n - 1; i >= 0; i--) {
      int v = order[i];
      if (!used[v]) {
        vector<int> cur_component;
        Dfs(cur_component, v);
        for (auto node : cur_component)
          component[node] = total_components;
        total_components++;
      }
    }
  }

  void GenTime(int node) {
    used[node] = true;
```

```
    for (auto to : g[node])
      if (!used[to])
        GenTime(to);
    order.push_back(node);
  }

  void Dfs(vector<int> &cur, int node) {
    used[node] = true;
    cur.push_back(node);
    for (auto to : gr[node])
      if (!used[to])
        Dfs(cur, to);
  }

  vector<vector<int>> CondensedGraph() {
    vector<vector<int>> ans(total_components);
    for (int i = 0; i < int(g.size()); i++) {
      for (int to : g[i]) {
        int u = component[i], v = component[to];
        if (u != v)
          ans[u].push_back(v);
      }
    }
    return ans;
  }
};
```

5.2 board

```
struct board {
  int n, m, r;
  board(int a, int b, int c = 1) : n(a), m(b), r(c) {}

  long long frec(int x, int y) {
    // returns how many squares of r x r contain the cell (x, y)
    long long a = min(x, n - r) - max(x - r + 1, 0) + 1;
    long long b = min(y, m - r) - max(y - r + 1, 0) + 1;
    return a * b;
  }

  bool valid(int x, int y) {
    return x >= 0 && x < n && y >= 0 && y < m;
  }
```

```
};
```

5.3 bridges

```
struct Graph {
  vector<vector<Edge>> g;
  vector<int> vi, low, d, pi, is_b;
  int bridges_computed;

  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;
    bridges_computed = 0;
  }

  void AddEdge(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 < (int)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]);
      }
```

```
    }
  }

  // Multiple edges from a to b are not allowed.
  // (they could be detected as a bridge).
  // If you need to handle this, just count
  // how many edges there are from a to b.
  void CompBridges() {
    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 < (int)g.size(); ++i)
      if (!vi[i]) Dfs(i);
    bridges_computed = true;
  }

  map<int, vector<Edge>> BridgesTree() {
    if (!bridges_computed) CompBridges();
    int n = g.size();
    Dsu dsu(g.size());
    for (int i = 0; i < n; i++)
      for (auto e : g[i])
        if (!is_b[e.id]) dsu.Join(i, e.to);

    map<int, vector<Edge>> tree;
    for (int i = 0; i < n; i++)
      for (auto e : g[i])
        if (is_b[e.id])
          tree[dsu.Find(i)].emplace_back(dsu.Find(e.to), e.id);

    return tree;
  }
};
```

5.4 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;
}
```

5.5 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;
}
```

5.6 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;
}
```

5.7 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

5.8 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.

5.9 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:

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

5.10 query with lca

```
struct lowest_ca {
  int T[MN], L[MN], W[MN];
  int P[MN][ML], MI[MN][ML], MA[MN][ML];

  void dfs(vector<vector<edge> > &g, int root, int pi = -1) {
    if (pi == -1) {
```

```
    L[root] = W[root] = 0;
    T[root] = -1;
  }
  for (int i = 0; i < (int)g[root].size(); ++i) {
    int to = g[root][i].v;
    if (to != pi) {
      T[to] = root;
      W[to] = g[root][i].w;
      L[to] = L[root] + 1;
      dfs(g, to, root);
    }
  }
}

void init(vector<vector<edge> > &g, int root) {
  // g is undirected
  dfs(g, root);
  int N = g.size(), i, j;

  for (i = 0; i < N; i++) {
    for (j = 0; 1 << j < N; j++) {
      P[i][j] = -1;
      MI[i][j] = inf;
    }
  }

  for (i = 0; i < N; i++) {
    P[i][0] = T[i];
    MI[i][0] = W[i];
  }

  for (j = 1; 1 << j < N; j++)
    for (i = 0; i < N; i++)
      if (P[i][j - 1] != -1) {
        P[i][j] = P[P[i][j - 1]][j - 1];
        MI[i][j] = min(MI[i][j-1], MI[ P[i][j - 1] ][j - 1]);
      }
}

int query(int p, int q) {
  int tmp, log, i;

  int mmin = inf;
  if (L[p] < L[q])
    tmp = p, p = q, q = tmp;

  for (log = 1; 1 << log <= L[p]; log++);
  log--;

  for (i = log; i >= 0; i--)
    if (L[p] - (1 << i) >= L[q]) {
      mmin = min(mmin, MI[p][i]);
      p = P[p][i];
    }

  if (p == q)
    // return p;
    return mmin;

  for (i = log; i >= 0; i--)
    if (P[p][i] != -1 && P[p][i] != P[q][i]) {
      mmin = min(mmin, min(MI[p][i], MI[q][i]));
      p = P[p][i], q = P[q][i];
    }

  // return T[p];
  return min(mmin, min(MI[p][0], MI[q][0]));
}

int get_child(int p, int q) { // p is ancestor of q
  if (p == q) return -1;

  int i, log;
  for (log = 1; 1 << log <= L[q]; log++) {}
  log--;

  for (i = log; i >= 0; i--)
    if (L[q] - (1 << i) > L[p]) {
      q = P[q][i];
    }

  assert(P[q][0] == p);
  return q;
}

int is_ancestor(int p, int q) {
  if (L[p] >= L[q])
    return false;

  int dist = L[q] - L[p];
```

```
    int cur = q;
    int step = 0;
    while (dist) {
      if (dist & 1)
        cur = P[cur][step];
      step++;
      dist >>= 1;
    }

    return cur == p;
  }
};
```

5.11 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++;
    }
  }
};
```

5.12 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;
}
```

6 Math

6.1 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 $m \leq n$.

6.2 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;
}
```

6.3 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;
}
```

6.4 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.

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

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

6.5 others

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

6.6 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;
  }
};
```

6.7 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:

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

7 Matrix

7.1 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;
  }
}
```

8 Misc

8.1 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();
    }

}
```

8.2 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];
}
```

8.3 fraction

```
struct frac{
  long long x, y;
  frac(long long a, long long b) {
    long long g = __gcd(a, b);
    x = a / g;
    y = b / g;
  }
  bool operator < (const frac &o) const {
    return (x * o.y < y * o.x);
  }
};
```

8.4 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; }
};
```

9 Number theory

9.1 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;
}
```

9.2 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;
}
```

9.3 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;
}
```

9.4 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;
}
```

9.5 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;
  }
}
```

9.6 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;
}
```

9.7 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;
}
```

9.8 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;
```

9.9 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;
}
```

9.10 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;
}
```

9.11 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;
}
```

9.12 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]);
  }
}
```

9.13 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;
}
```

9.14 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;
  }
}
```

9.15 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;
```

9.16 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;
}
```

10 Strings

10.1 Incremental Aho Corasick

```
class IncrementalAhoCorasic {
  static const int Alphabets = 26;
  static const int AlphabetBase = 'a';
  struct Node {
    Node *fail;
    Node *next[Alphabets];
    int sum;
    Node() : fail(NULL), next{}, sum(0) { }
  };

  struct String {
    string str;
    int sign;
  };

public:
  //totalLen = sum of (len + 1)
  void init(int totalLen) {
    nodes.resize(totalLen);
    nNodes = 0;
    strings.clear();
    roots.clear();
    sizes.clear();
    que.resize(totalLen);
  }

  void insert(const string &str, int sign) {
    strings.push_back(String{ str, sign });
    roots.push_back(nodes.data() + nNodes);
    sizes.push_back(1);
    nNodes += (int)str.size() + 1;
    auto check = [&]() { return sizes.size() > 1 && sizes.end()[-1] ==
        sizes.end()[-2]; };
    if(!check())
```

```
    makePMA(strings.end() - 1, strings.end(), roots.back(), que);
   while(check()) {
    int m = sizes.back();
    roots.pop_back();
    sizes.pop_back();
    sizes.back() += m;
    if(!check())
     makePMA(strings.end() - m * 2, strings.end(), roots.back(), que);
   }
  }

  int match(const string &str) const {
   int res = 0;
   for(const Node *t : roots)
    res += matchPMA(t, str);
   return res;
  }

private:
  static void makePMA(vector<String>::const_iterator begin,
      vector<String>::const_iterator end, Node *nodes, vector<Node*>
      &que) {
   int nNodes = 0;
   Node *root = new(&nodes[nNodes ++]) Node();
   for(auto it = begin; it != end; ++ it) {
    Node *t = root;
    for(char c : it->str) {
     Node *&n = t->next[c - AlphabetBase];
     if(n == nullptr)
      n = new(&nodes[nNodes ++]) Node();
     t = n;
    }
    t->sum += it->sign;
   }
   int qt = 0;
   for(Node *&n : root->next) {
    if(n != nullptr) {
     n->fail = root;
     que[qt ++] = n;
    } else {
     n = root;
    }
   }
   for(int qh = 0; qh != qt; ++ qh) {
    Node *t = que[qh];
    int a = 0;
    for(Node *n : t->next) {
     if(n != nullptr) {
      que[qt ++] = n;
      Node *r = t->fail;
      while(r->next[a] == nullptr)
       r = r->fail;
      n->fail = r->next[a];
      n->sum += r->next[a]->sum;
     }
     ++ a;
    }
   }
  }

  static int matchPMA(const Node *t, const string &str) {
   int res = 0;
   for(char c : str) {
    int a = c - AlphabetBase;
    while(t->next[a] == nullptr)
     t = t->fail;
    t = t->next[a];
    res += t->sum;
   }
   return res;
  }


  vector<Node> nodes;
  int nNodes;
  vector<String> strings;
  vector<Node*> roots;
  vector<int> sizes;
  vector<Node*> que;
};

int main() {
  int m;
  while(~scanf("%d", &m)) {
   IncrementalAhoCorasic iac;
   iac.init(600000);
   rep(i, m) {
    int ty;
    char s[300001];
    scanf("%d%s", &ty, s);
```

```
    if(ty == 1) {
      iac.insert(s, +1);
    } else if(ty == 2) {
      iac.insert(s, -1);
    } else if(ty == 3) {
      int ans = iac.match(s);
      printf("%d\n", ans);
      fflush(stdout);
    } else {
      abort();
    }
   }
  }
  return 0;
}
```

10.2 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;
}
```

10.3 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;
  }
};
```

10.4 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/view/1530
 * */

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;
}
```

10.5 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;
}
```