Team Note of 020304

sharaelong, serverrepairman, whqkrkt04

Compiled on July 22, 2022

Contents

문제가 너무 쉽거나 어려우면 조건을 잘못 읽은 게 아닌지 확인하기!!

문제를 잘못 읽으면 팀노트의 모든 내용이 의미가 없어진다는 사실을 꼭 기억하기!!!

누군가가 컴퓨터를 잡고 디버깅을 길게 하고 있으면 계속 확인해주기!!!

자신이 컴퓨터를 잡고 있는 동안 시간의 가치가 3배로 늘어난다는 사실을 언제나 생각하기!!!

스코어보드에서 많이 풀린 순으로 푸는 게 언제나 옳다!!!

GLHF Good Luck Have Fun!

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
typedef pair<int, pi> piii;
typedef pair<ll, ll> pll;
typedef pair<ll, pl> plll;
#define fi first
#define se second
const int INF = 1e9+1;
const int P = 1000000007;
const ll LLINF = (ll)1e18+1;
template <typename T>
ostream& operator<<(ostream& os, const vector<T>& v) { for(auto i : v) os << i << " "; os <<
"\n"; return os; }
template <typename T1, typename T2>
ostream& operator<<(ostream& os, const pair<T1, T2>& p) { os << p.fi << " " << p.se; return
os; }
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
#define rnd(x, y) uniform_int_distribution<int>(x, y)(rng)

ll mod(ll a, ll b) { return ((a%b) + b) % b; }
ll ext_gcd(ll a, ll b, ll &x, ll &y) {
    ll g = a; x = 1, y = 0;
    if(b) g = ext_gcd(b, a % b, y, x), y -= a / b * x;
    return g;
}
ll inv(ll a, ll m) {
    ll x, y; ll g = ext_gcd(a, m, x, y);
    if(g > 1) return -1;
    return mod(x, m);
}

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

    return 0;
}
```

1 Data Structures

1.1 HLD

```
class HLD {
private:
    static constexpr int M = 500001;
    vector<int> adj[M];
    int in[M], sz[M], par[M], top[M], depth[M];

    void traverse1(int u) {
        sz[u] = 1;
        for (int &v: adj[u]) {
            adj[v].erase(find(adj[v].begin(), adj[v].end(), u));
            depth[v] = depth[u] + 1;
            traverse1(v);
            par[v] = u;
            sz[u] += sz[v];
            if (sz[v] > sz[adj[u][0]]) swap(v, adj[u][0]);
        }
    }
    void traverse2(int u) {
        static int n = 0;
        in[u] = n++;
        for (int v: adj[u]) {
            top[v] = (v == adj[u][0] ? top[u] : v);
            traverse2(v);
        }
    }
public:
    void link(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void init() {
        top[1] = 1;
        traverse1(1);
        traverse2(1);
    }
    pint subtree(int u) {
        return {in[u], in[u] + sz[u]};
    }
    vector<pint> path(int u, int v) {
        vector<pint> res;
        while (top[u] != top[v]) {
            if (depth[top[u]] < depth[top[v]]) swap(u, v);
            res.emplace_back(in[top[u]], in[u] + 1);
            u = par[top[u]];
        }
        res.emplace_back(min(in[u], in[v]), max(in[u], in[v]) + 1);
        return res;
    }
};
```

1.2 Non-recursive Segment Tree

```
// (s, e], 0-indexed
class SegTree {
private:
    constexpr static int M = 131072;
    int t[M * 2];
public:
    void update(int i, int d) {
        for (i += M; i > 0; i >>= 1) t[i] += d;
    }
    int sum(int s, int e) {
        int res = 0;
        for (s += M, e += M; s < e; s >>= 1, e >>= 1) {
            if (s & 1) res += t[s++];
            if (e & 1) res += t[--e];
        }
        return res;
    }
};
```

1.3 Li-Chao Tree

```
typedef long long lint;
const lint INF = 4e18;
const lint X_INF = 1e12 + 10;

struct LiChao {
    struct Line {
        lint a, b;
        lint at(lint x) { return a * x + b; }
    };
    struct Node {
        Node *l, *r;
        Line line;
    } *root;

    LiChao() {
        root = new Node {nullptr, nullptr, {0, -INF}};
    }
    void update(Line line) {
        _update(line, root, -X_INF, X_INF);
    }
    void _update(Line line, Node *p, lint s, lint e) {
        Line l1 = p->line, l2 = line;
        if (l1.at(s) > l2.at(s)) swap(l1, l2);
        if (l1.at(e - 1) <= l2.at(e - 1)) {
            p->line = l2;
            return;
        }
        lint mid = (s + e) / 2;
        if (l1.at(mid) >= l2.at(mid)) {
            p->line = l1;
            if (!p->l) p->l = new Node {nullptr, nullptr, l2};
            _update(l2, p->l, s, mid);
        }
        else {
            p->line = l2;
            if (!p->r) p->r = new Node {nullptr, nullptr, l1};
            _update(l1, p->r, mid, e);
        }
    }
    lint query(lint x) {
        return _query(x, root, -X_INF, X_INF);
    }
    lint _query(lint x, Node *p, lint s, lint e) {
        if (!p || x < s || x >= e) return -INF;
        lint mid = (s + e) / 2;
        return max({ p->line.at(x), _query(x, p->l, s, mid), _query(x, p->r, mid, e) });
    }
} T;
```

1.4 Splay Tree

```
typedef ll TCON; // content
const TCON initval = 0;
typedef ll TV; // subtree value
const TV id = 0;
typedef ll TLAZ; // lazy value
const TLAZ S_unused = 0;

struct Snode{
    Snode *l, *r, *p;
    int cnt;

    TCON content = initval;
    TV val;
    TLAZ lazy = S_unused;

    void init(){
        // Initialize value using CONTENT
        val = content;
    }
    TV combine(TV a, TV b){
        // Real value when a <-- b
        return a+b;
    }
    TLAZ combineL(TLAZ a, TLAZ b){
        // Lazy value when a <-- b
        return a+b;
    }
    void unlazy_inner(){
        // Update CONTENT and VAL using LAZY
        content+= lazy;
        val+= lazy * cnt;
    }

    void update(){
        cnt = 1;
        init();
        if(l) l->unlazy(), cnt+= l->cnt,
            val = combine(l->val, val);
        if(r) r->unlazy(), cnt+= r->cnt,
            val = combine(val, r->val);
    }

    void lazy_add(TLAZ x){lazy = combineL(lazy, x);}
    void unlazy(){
        if(lazy == S_unused) return;
        unlazy_inner();
        if(l) l->lazy_add(lazy);
        if(r) r->lazy_add(lazy);
        lazy = S_unused;
    }

    void debug_inorder(){
        unlazy();
        if(l) l->debug_inorder();
        //cout << content << ' ';
        if(r) r->debug_inorder();
    }
};

// 1-indexed; has sentinel nodes on both ends
struct Splay{
    Snode *root;

    Splay(int n){
        Snode *x;
        root = x = new Snode;
        x->l = x->r = x->p = NULL;
        x->cnt = n, x->lazy = S_unused;
        x->init();
        for(int i=1; i<n+2; i++){
            x->r = new Snode;
            x->r->p = x; x = x->r;
            x->l = x->r = NULL;
            x->cnt = n-i, x->lazy = S_unused;
            x->init();
        }
    }

    void rotate(Snode *x){
        // x goes to parent of x
        Snode *p = x->p, *b = NULL;
        if(x == p->l) p->l = b = x->r, x->r = p;
        else p->r = b = x->l, x->l = p;
        x->p = p->p, p->p = x;
        if(b) b->p = p;
        (x->p ? p == x->p->l ? x->p->l : x->p->r : root) = x;
        p->update(), x->update();
    }

    Snode* splay(Snode *x){
        // x becomes the root
        while(x->p){
            Snode *p = x->p, *g = p->p;
            if(g) rotate((x == p->l) == (p == g->l) ? p : x);
            rotate(x);
        }
        return root = x;
    }

    Snode* kth(int k){
        // kth becomes the root
        // DO NOT USE IT FOR POINT UPDATE!! USE INTERVAL(k,k)!!!!
        Snode *x = root; x->unlazy();
        while(1){
            while(x->l && x->l->cnt > k) (x = x->l)->unlazy();
            if(x->l) k-= x->l->cnt;
            if(!k--) break;
            (x = x->r)->unlazy();
        }
        return splay(x);
    }

    Snode* interval(int l, int r){
        // l to r goes to root->r->l
        kth(l-1);
        Snode *x = root;
        root = x->r; root->p = NULL;
        kth(r-l+1);
        x->r= root; root->p = x; root = x;
        (x = root->r->l)->unlazy();
        return x;
    }

    void insert(int k, TCON v){
        // insert CONTENT v at index k, which becomes root
        kth(k);
        Snode *x = new Snode;
        if(root->l) root->l->p = x;
        x->l = root->l; root->l = x; x->p = root; x->r = NULL;
        x->content = v; x->init();
        splay(x);
    }

    void remove(int k){
        // remove k-th node
        kth(k);
        Snode *p = root;
        p->unlazy();
        if(p->l){
            if(p->r){
                root = p->l; root->p = NULL;
                Snode *cur = root;
                cur->unlazy();
                while (cur->r) cur = cur->r, cur->unlazy();
                cur->r = p->r; p->r->p = cur;
                splay(cur); delete p;
            }
            else{root = p->l; root->p = NULL; delete p;}
        }
        else{root = p->r; if(root) root->p = NULL; delete p;}
    }
};
```

1.5 PBDS RBTree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
#include <functional>
using namespace ¨gnu˙pbds;
using ordered_set = tree<int, null_type, less<int>,
  rb_tree_tag, tree_order_statistics_node_update>;

int main(){
  ordered_set X;
  for (int i=1; i<10; i+=2) X.insert(i); // 1 3 5 7 9
  cout << *X.find_by_order(2) << endl; // 5
  cout << X.order_of_key(6) << endl; // 3
  cout << X.order_of_key(7) << endl; // 3
  X.erase(3);
}
```

1.6 Rope

```
#include <ext/rope>
using namespace std;
using namespace ¨gnu˙cxx;

int main()
{
    crope rp;
    // operator[], substr, operator+=
}
```

2 Flows, Matching, Graph

2.1 Hopcroft-Karp Bipartite Matching

```
class BipartiteMatching {
private:
    constexpr static int N = 201, M = 201;
    int level[N];
    vector<int> adj[N];

    void bfs() {
        queue<int> q;
        for (int i = 0; i < N; i++) {
            if (A[i] == -1) {
                q.push(i);
                level[i] = 0;
            }
            else level[i] = -1;
        }
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (int v: adj[u]) {
                if (B[v] == -1 || level[B[v]] != -1) continue;
                level[B[v]] = level[u] + 1;
                q.push(B[v]);
            }
        }
    }
    bool dfs(int u) {
        for (int v: adj[u]) {
            if (B[v] != -1 && (level[B[v]] != level[u] + 1 || !dfs(B[v]))) continue;
            A[u] = v;
            B[v] = u;
            return true;
        }
        return false;
    }
public:
    int A[N], B[M];

    inline void add_edge(int u, int v) { adj[u].push_back(v); }
    int match() {
        fill(A, A + N, -1);
        fill(B, B + M, -1);

        int res = 0;
        while (true) {
            bfs();

            int found = 0;
            for (int i = 0; i < N; i++)
                if (A[i] == -1) found += dfs(i);

            if (found == 0) return res;
            res += found;
        }
    }
};
```

2.2 Dinic's Algorithm

```
class DinicGraph {
private:
    constexpr static int INF = 0x3f3f3f3f;
    struct Edge {
        int oppo, capa, flow;
        Edge* dual;
    };

    vector<vector<Edge*> > adj;
    vector<int> level, search_cache;

    int dfs(int here, int dest, int extra_capa) {
        if (here == dest) { return extra_capa; }
        for (int &i = search_cache[here]; i < adj[here].size(); ++i) {
            Edge* e = adj[here][i];
            if (level[here] + 1 != level[e->oppo] || e->flow == e->capa) { continue; }

            int flow_from_here = dfs(e->oppo, dest, min(extra_capa, e->capa - e->flow));
            if (flow_from_here > 0) {
                e->flow += flow_from_here;
                e->dual->flow -= flow_from_here;
                return flow_from_here;
            }
        }
        return 0;
    }

public:
    DinicGraph(int n): adj(n), level(n), search_cache(n) {}
    ~DinicGraph() {
        for (auto &v: adj) {
            for (Edge* e: v) {
                delete e;
            }
    }
  }

  void add_edge(int u, int v, int capa) {
    Edge* e1 = new Edge {v, capa, 0, nullptr};
    Edge* e2 = new Edge {u, 0, 0, e1};
    e1->dual = e2;
    adj[u].push_back(e1);
    adj[v].push_back(e2);
  }

  int max_flow(int s, int t) {
    int ret = 0;
    while (true) {
      fill(level.begin(), level.end(), -1);
      fill(search_cache.begin(), search_cache.end(), 0);
      level[s] = 0;
      queue<int> q;
      q.push(s);
      while (!q.empty()) {
        int here = q.front();
        q.pop();
        for (Edge* e: adj[here]) {
          if (level[e->oppo] != -1 || e->flow == e->capa) { continue; }
          level[e->oppo] = level[here] + 1;
          q.push(e->oppo);
        }
      }
      if (level[t] == -1) { return ret; }

      while (true) {
        int flow = dfs(s, t, INF);
        if (flow == 0) { break; }
        ret += flow;
      }
    }
    // not reachable!
    return -1;
  }
};
```

2.3 Stoer-Wagner

```
struct Mincut{
 int n; vector<vector<int>> graph;
 Mincut(int n): n{n}, graph(n, vector<int>(n)) {}
 void connect(int a, int b, int w)
  {if(a != b) graph[a][b]+= w, graph[b][a]+= w;}

 pair<int, pair<int, int>> stmin(vector<int> &active){
  vector<int> key(n), v(n);
  int s = -1, t = -1;
  for(size_t i=0; i<active.size(); i++){
   int maxv = -1, cur = -1;
   for(auto j: active) if(!v[j] && maxv < key[j])
    maxv = key[j], cur = j;
   t = s, s = cur; v[cur] = 1;
   for(auto j: active) key[j]+= graph[cur][j];
  }
  return make_pair(key[s], make_pair(s, t));
 }

 vector<int> cut;
 int solve(){
  int res = numeric_limits<int>::max();
  vector<vector<int>> grps; vector<int> active;
  cut.resize(n);
  for(int i=0; i<n; i++) grps.emplace_back(1, i);
  for(int i=0; i<n; i++) active.push_back(i);
  while(active.size() >= 2){
   auto stcut = stmin(active);
   if(stcut.first < res){
    res = stcut.first;
    fill(entire(cut), 0);
    for(auto v: grps[stcut.second.first]) cut[v] = 1;
   }
   int s, t; tie(s, t) = stcut.second;
   if(grps[s].size() < grps[t].size()) swap(s, t);
   active.erase(find(entire(active), t));
   grps[s].insert(grps[s].end(), entire(grps[t]));
   for(int i=0; i<n; i++)
    graph[i][s]+= graph[i][t], graph[i][t] = 0;
   for(int i=0; i<n; i++)
    graph[s][i]+= graph[t][i], graph[t][i] = 0;
   graph[s][s] = 0;
  }
  return res;
 }
};
```

2.4 Hungarian Algorithm

```
struct Hungarian {
  int n;
  int Cost[M][M], S[M], T[M];
  int A[M], B[M], C[M], Bt[M], k;

  void set_cost(int u, int v, int cost) {
    Cost[u][v] = cost;
  }
  int func(int u, int v) {
    return Cost[u][v] - S[u] - T[v];
  }
  bool match(int cur) {
    for (int i = 1; i <= n; i++) {
      if (func(C[i], i) > func(cur, i)) C[i] = cur;
    }
    for (int i = 1; i <= n; i++) if (!func(cur, i) && !Bt[i]) {
      Bt[i] = cur;
      if (!B[i] || match(B[i])) return true;
    }
    return false;
  }
  void augment() {
    for (int i = 1; i <= n; i++) if (Bt[i] && !B[i]) {
      k = i; break;
```

```
        }
        for (int i = k; i; i = k) {
            B[i] = Bt[i]; k = A[Bt[i]]; A[Bt[i]] = i;
        }
    }
    void solve(int x) {
        for (int i = 1; i <= n; i++) C[i] = x;
        if (match(x)) {
            augment(); return;
        }
        while (1) {
            k = -1;
            for (int i = 1; i <= n; i++) if (!Bt[i]) {
                if (k == -1 || func(C[k], k) > func(C[i], i)) k = i;
            }
            int v = func(C[k], k);
            for (int i = 1; i <= n; i++) {
                if (i == x || Bt[A[i]]) S[i] += v;
                if (Bt[i]) T[i] -= v;
            }
            Bt[k] = C[k];
            if (!B[k] || match(B[k])) {
                augment(); break;
            }
        }
        memset(Bt, 0, sizeof Bt);
    }
    int get_match() {
        int ans = 0;
        for (int i = 1; i <= n; i++) solve(i);
        for (int i = 1; i <= n; i++) {
            ans += Cost[i][A[i]];
        }
        return ans;
    }
} G;
```

2.5 Min Cost Max Flow

```
struct MCMFGraph {
    struct Edge {
        int from, to;
        lint c, f, cost;
        Edge *dual;
    };
    vector<lint> dist;
    vector<bool> InQ;
    vector<Edge*> bt;
    vector<vector<Edge*>> G;

    MCMFGraph(int n) {
        G.resize(n + 1); InQ.resize(n + 1);
        dist.resize(n + 1); bt.resize(n + 1);
    }
    void add_edge(int u, int v, lint c, lint cost) {
        Edge *e1 = new Edge {u, v, c, 0, cost, nullptr};
        Edge *e2 = new Edge {v, u, 0, 0, -cost, nullptr};
        e1->dual = e2; e2->dual = e1;
        G[u].push_back(e1); G[v].push_back(e2);
    }

    pint solve(int src, int snk) {
        lint total_cost = 0, total_flow = 0;
        while (1) {
            fill(dist.begin(), dist.end(), 1e18);
            fill(bt.begin(), bt.end(), nullptr);
            fill(InQ.begin(), InQ.end(), 0);

            queue<int> Q;
            dist[src] = 0;
            Q.push(src); InQ[src] = 1;
            while (!Q.empty()) {
                int x = Q.front(); Q.pop();
                InQ[x] = 0;

                for (Edge *e : G[x]) {
                    if (e->c == e->f) continue;
                    if (dist[e->to] > dist[x] + e->cost) {
                        dist[e->to] = dist[x] + e->cost;
                        bt[e->to] = e;
                        if (!InQ[e->to]) {
                            Q.push(e->to); InQ[e->to] = 1;
                        }
                    }
                }
            }
            if (dist[snk] == 1e18) break;
            lint x = 0, f = 1e18;
            for (auto e = bt[snk]; e; e = bt[e->from]) {
                f = min(f, e->c - e->f);
            }
            for (auto e = bt[snk]; e; e = bt[e->from]) {
                e->f += f;
                e->dual->f -= f;
                total_cost += f * e->cost;
            }
            total_flow += f;
        }
        return {total_flow, total_cost};
    }
};
```

2.6 2-SAT

```
struct TwoSat { // 1-based
    int n;
    vector<vector<int>> G, H;
    vector<vector<int>> SCC;
    vector<int> Chk, Stk, Id;

    TwoSat(int _n) {
        n = 2 * _n + 1;
        G.resize(n); H.resize(n);
        Chk.resize(n); Id.resize(n);
    }
    void add(int x, int y) {
        if (x < 0) x += n;
        if (y < 0) y += n;
        G[n - x].push_back(y);
        H[y].push_back(n - x);
        G[n - y].push_back(x);
        H[x].push_back(n - y);
    }
    void dfs(int cur) {
        if (Chk[cur]) return; Chk[cur] = true;
        for (int x : G[cur]) dfs(x);
        Stk.push_back(cur);
    }
    void back_dfs(int cur, int idx) {
        if (Chk[cur]) return; Chk[cur] = true;
        SCC[idx].push_back(cur);
        Id[cur] = idx;
        for (int x : H[cur]) back_dfs(x, idx);
    }
    void get_SCC() {
        for (int i = 1; i < n; i++) {
            if (!Chk[i]) dfs(i);
        }
        fill(Chk.begin(), Chk.end(), false);
        while (!Stk.empty()) {
            int x = Stk.back(); Stk.pop_back();
            if (Chk[x]) continue;
            SCC.push_back(vector<int>(0));
            back_dfs(x, SCC.size() - 1);
        }
    }
    vector<bool> solve() {
        get_SCC();
        for (int i = 1; i <= n / 2; i++) {
            if (Id[i] == Id[n - i]) return vector<bool>(0);
        }
        vector<int> Ans(n, -1);
        for (int i = 0; i < SCC.size(); i++) {
            if (Ans[SCC[i][0]] != -1) continue;
            for (int x : SCC[i]) {
                Ans[x] = 0; Ans[n - x] = 1;
            }
        }
        vector<bool> ret(Ans.begin() + 1, Ans.begin() + 1 + n / 2);
        return ret;
    }
};
```

2.7 BCC

```
class BCCGraph {
private:
    constexpr static int M = 10000;
    int dfsc = 0, dfsn[M];
    vector<int> adj[M];
    vector<pint> st;
public:
    BCCGraph() { fill(dfsn, dfsn + M, 0); }
    vector<vector<pint>> bccs;
    inline void add_edge(int u, int v) { adj[u].push_back(v); adj[v].push_back(u); }
    int decompose(int u, int p = -1) {
        int m_dfsn = dfsn[u] = ++dfsc;

        for (int v: adj[u]) {
            if (v == p) continue;

            if (dfsn[u] > dfsn[v]) st.emplace_back(u, v);
            if (dfsn[v] != 0) m_dfsn = min(m_dfsn, v);
            else {
                int t_dfsn = decompose(v, u);
                m_dfsn = min(m_dfsn, t_dfsn);

                if (t_dfsn >= dfsn[u]) {
                    vector<pint> bcc;
                    while (!st.empty() && st.back() != pint {u, v}) {
                    bcc.push_back(st.back()); st.pop_back(); }
                    bcc.push_back(st.back()); st.pop_back();
                    bccs.push_back(bcc);
                }
            }
        }

        return m_dfsn;
    }
};
```

2.8 Bridge

```
class BridgeGraph {
private:
    constexpr static int M = 10000;
    vector<int> adj[M];
    int depth[M], highest[M];
public:
    vector<pint> res;
    BridgeGraph() { fill(depth + 1, depth + M, -1); }
    void add_edge(int u, int v) { adj[u].push_back(v); adj[v].push_back(u); }
    void find_bridge(int u = 0, int p = -1) {
        for (int v: adj[u]) {
            if (v == p) continue;
            if (depth[v] == -1) {
                depth[v] = highest[v] = depth[u] + 1;
                find_bridge(v, u);
                if (depth[v] == highest[v]) res.emplace_back(u, v);
            }
            highest[u] = min(highest[u], highest[v]);
        }
    };
};
```

2.9 Cactus decomposition

```
// adj, ring, sub_branch store index of edge
// sub_ring stores index of ring
vector<ituple> edges;
vector<int> adj[M], sub_branch[M], sub_ring[M], ring[M];
int ring_cnt[M];
```

```
void decompose(int u, int p = -1) {
    static vector<int> st;
    static bitset<M> visited, finished;
    static int rcnt = 0;

    visited[u] = true;

    for (int e: adj[u]) {
        int v = u ^ get<0>(edges[e]) ^ get<1>(edges[e]);
        if (e == p || finished[v]) continue;
        st.push_back(e);

        if (!visited[v]) {
            int cnt = ring_cnt[u];
            decompose(v, e);
            if (ring_cnt[u] == cnt) sub_branch[u].push_back(e);
        }
        else {
            auto it = st.rbegin();
            int p = v;
            do {
                ring_cnt[p]++;
                p ^= get<0>(edges[*it]) ^ get<1>(edges[*it]);
                it++;
            } while (p != v);
            ring[rcnt].insert(ring[rcnt].end(), st.rbegin(), it);
            sub_ring[v].push_back(rcnt++);
        }
        st.pop_back();
    }

    finished[u] = true;
}
```

3 Number Theory

3.1 FFT, Polynomial Multiplication

```
const double PI = acos(-1);
typedef complex<double> C;

void FFT(vector<C> &F, bool inv) {
    int n = F.size();
    for (int i = 1, j = 0; i < n; i++) {
        int b = n >> 1;
        for (; j >= b; b >>= 1) j -= b;
        j += b;
        if (i < j) swap(F[i], F[j]);
    }
    for (int i = 1; i < n; i <<= 1) {
        C w(cos(PI / i), sin(PI / i));
        if (inv) w = conj(w);

        for (int j = 0; j < n; j += 2 * i) {
            C x(1, 0);
            for (int k = j; k < j + i; k++) {
                C a = F[k], b = F[k + i];
                F[k] = a + b * x;
                F[k + i] = a - b * x;
                x = x * w;
            }
        }
    }
    if (inv) {
        for (int i = 0; i < n; i++) F[i] /= n;
    }
}
void multiply(vector<int> F, vector<int> G, vector<int> &Res) {
    int n = 2;
    while (n < F.size() + G.size()) n <<= 1;

    vector<C> P(n);
    for (int i = 0; i < F.size(); i++) P[i].real(F[i]);
    for (int i = 0; i < G.size(); i++) P[i].imag(G[i]);
    FFT(P, 0);

    for (int i = 0; i < n / 2; i++) {
        C a = P[i], b = P[(n - i) % n];
        P[i] = (a * a - conj(b) * conj(b)) * C(0, -0.25);
        P[(n - i) % n] = (b * b - conj(a) * conj(a)) * C(0, -0.25);
    }
    FFT(P, 1);

    Res.resize(n);
    for (int i = 0; i < n; i++) {
        Res[i] = (int)round(P[i].real());
    }
}
```

3.2 NTT

```
// TODO
```

3.3 Miller-Rabin test, Pollard-Rho Algorithm

```
mt19937 rng(1010101);
lint randInt(lint l, lint r) {
    return uniform_int_distribution<lint>(l, r)(rng);
}

namespace NT {
    const lint Base[12] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 };
    const lint NAIVE_MAX = 1'000'000'000;

    lint add(lint a, lint b, const lint mod) {
        if (a + b >= mod) return a + b - mod;
        return a + b;
    }
    lint mul(lint a, lint b, const lint mod) {
        return (__int128_t)a * b % mod;
    }
    lint _pow(lint a, lint b, const lint mod) {
        lint ret = 1;
        while (b) {
            if (b & 1) ret = mul(ret, a, mod);
            a = mul(a, a, mod); b /= 2;
        }
        return ret;
    }
    bool naive_prime(lint n) {
        for (int i = 2; i * i <= n; i++) {
            if (n % i == 0) return false;
        }
        return true;
    }
    bool is_prime(lint n) {
        if (n <= NAIVE_MAX) {
            return naive_prime(n);
        }
        if (n % 2 == 0) return false;
        // Miller-Rabin Primality test
        lint s = 0, d = n - 1;
        while (d % 2 == 0) {
            s += 1; d /= 2;
        }

        // When n < 2^64, it is okay to test only prime bases <= 37
        for (lint base : Base) {
            lint x = _pow(base, d, n), f = 0;
            if (x == 1) f = 1;
            for (int i = 0; i < s; i++) {
                if (x == n - 1) {
                    f = 1;
                }
                x = mul(x, x, n);
            }
            if (!f) return false;
        }
        return true;
    }
    lint run(lint n, lint x0, lint c) {
        function<lint(lint)> f = [c, n](lint x) {
            return NT::add(NT::mul(x, x, n), c, n);
        };
        lint x = x0, y = x0, g = 1;
        while (g == 1) {
            x = f(x);
            y = f(y); y = f(y);
            g = __gcd(abs(x - y), n);
        }
        return g;
    }
    // Res is NOT sorted after this call
    void factorize(lint n, vector<lint> &Res) {
        if (n == 1) return;
        if (n % 2 == 0) {
            Res.push_back(2); factorize(n / 2, Res);
            return;
        }
        if (is_prime(n)) {
            Res.push_back(n); return;
        }
        while (1) {
            lint x0 = randInt(1, n - 1), c = randInt(1, 20) % (n - 1) + 1;
            lint g = run(n, x0, c);
            if (g != n) {
                factorize(n / g, Res); factorize(g, Res);
                return;
            }
        }
    }
};
```

3.4 CRT, Diophantine Equation

```
typedef long long lint;
typedef pair<lint, lint> pint;

// returns (x0, y0) where x0 >= 0, x0 = -1 if solution does not exist
pint solve(lint a, lint b, lint c) {
    lint g = __gcd(a, b);
    if (c % g != 0) return pint(-1, 0);
    c /= g; a /= g; b /= g;

    vector<lint> V;
    while (b != 0) {
        lint q = a / b, r = a % b;
        V.push_back(q);
        a = b; b = r;
    }
    lint x = c, y = 0;
    while (!V.empty()) {
        lint q = V.back(); V.pop_back();
        b += q * a; swap(a, b);
        x -= q * y; swap(x, y);
    }
    lint r = (x - (b + x % b) % b) / b;
    x -= b * r; y += a * r;

    return pint(x, y);
}
// returns (x, period of x), x = -1 if solution doesn't exist
pint CRT(lint a1, lint m1, lint a2, lint m2) {
    auto sol = solve(m1, m2, a2 - a1);
    if (sol.va == -1) return pint(-1, 0);

    lint g = __gcd(m1, m2); m2 /= g;
    return pint((m1 * sol.va + a1) % (m1 * m2), m1 * m2);
}
```

3.5 Linear sieve

```
void linear_sieve() {
    vector<int> p(M), pr;
    vector<int> mu(M), phi(M);
    for (int i = 2; i < M; i++) {
        if (!p[i]) {
            pr.push_back(i);
            mu[i] = -1;
            phi[i] = i - 1; // value of multiplicative function for prime
        }
        for (int j = 0; j < pr.size() && i * pr[j] < M; j++) {
            p[i * pr[j]] = 1;
            if (i % pr[j] == 0) {
                mu[i * pr[j]] = 0;
                phi[i * pr[j]] = phi[i] * pr[j];
                break;
            }
            else {
                mu[i * pr[j]] = mu[i] * mu[pr[j]];
                phi[i * pr[j]] = phi[i] * phi[pr[j]];
            }
        }
    }
    for (int i = 2; i < 50; i++) {
        cout << "mu(" << i << ") = " << mu[i] << ' ';
        cout << "phi(" << i << ") = " << phi[i] << '\n';
    }
}
```

3.6 Mobius inversion

If $f(n) = \sum_{d|n} g(d)$ and f, g are multiplicative functions, $g(n) = \sum_{d|n} \mu(d) g(n/d)$.

$\sum_{d|n} \mu(d) = [n = 1]$

$\sum_{i=1}^{n} \sum_{j=1}^{n} [gcd(i, j) = 1] = \sum_{d=1}^{n} \lfloor \frac{n}{d} \rfloor^2$

$\sum_{i=1}^{n} \sum_{j=1}^{n} gcd(i, j) = \sum_{d=1}^{n} \phi(d) \lfloor \frac{n}{d} \rfloor^2$

4 String

4.1 Suffix array nlogn

```
void suffix_array(string S, vector<int> &sa, vector<int> &lcp) {
    int n = S.size();
    vector<int> r(n), nr(n), pos(n), ind(n);
    sa.resize(n); lcp.resize(n);

    for (int i = 0; i < n; i++) sa[i] = i;
    sort(sa.begin(), sa.end(), [&](int a, int b) { return S[a] < S[b]; });
    for (int i = 1; i < n; i++) r[sa[i]] = r[sa[i - 1]] + (S[sa[i - 1]] != S[sa[i]]);

    for (int d = 1; d < n; d <<= 1) {
        for (int i = n - 1; i >= 0; i--) {
            pos[r[sa[i]]] = i;
        }
        int j = 0;
        for (int i = n - d; i < n; i++) ind[j++] = i;
        for (int i = 0; i < n; i++) {
            if (sa[i] >= d) ind[j++] = sa[i] - d;
        }
        for (int i = 0; i < n; i++) {
            sa[pos[r[ind[i]]]++] = ind[i];
        }
        nr[sa[0]] = 0;
        for (int i = 1; i < n; i++) {
            if (r[sa[i]] != r[sa[i - 1]]) {
                nr[sa[i]] = nr[sa[i - 1]] + 1;
            }
            else {
                int prv = (sa[i - 1] + d >= n ? -1 : r[sa[i - 1] + d]);
                int cur = (sa[i] + d >= n ? -1 : r[sa[i] + d]);
                nr[sa[i]] = nr[sa[i - 1]] + (prv != cur);
            }
        }
        swap(r, nr);
    }
    for (int i = 0, len = 0; i < n; ++i, len = max(len - 1, 0)) {
        if (r[i] == n - 1) continue;
        for (int j = sa[r[i] + 1]; S[i + len] == S[j + len]; ++len);
        lcp[r[i]] = len;
    }
}
```

4.2 KMP, Failure Function

```
vector<int> failure_function(string S) {
    int n = S.size();
    vector<int> F(n);
    for (int i = 1; i < n; i++) {
        for (int j = F[i - 1]; j != 0; j = F[j - 1]) {
            if (S[j] == S[i]) {
                F[i] = j + 1; break;
            }
        }
        if (S[0] == S[i]) F[i] = max(1, F[i]);
    }
    return F;
}
// All occurences of P in S (0-base)
vector<int> search(string S, string P) {
    int n = S.size(), m = P.size();
    vector<int> F = failure_function(P);
    vector<int> Ans;

    for (int i = 0, p = 0; i < n; i++) {
        while (p && (p == m || S[i] != P[p])) p = F[p - 1];
        if (S[i] == P[p]) p += 1;
        if (p == m) Ans.push_back(i - m + 1);
    }
    return Ans;
}
```

4.3 Aho-Corasick

```
struct Node {
    Node *go[26], *fail;
    bool end;
    Node() : fail(nullptr), end(false) { fill(go, go + 26, nullptr); }
    ~Node() {
        for (Node *next: go)
            if (next) delete next;
    }
};

Node * build_trie(vector<string> &patterns) {
    Node *root = new Node();

    for (string &p: patterns) {
        Node *curr = root;
        for (char c: p) {
            if (!curr->go[c - 'a']) curr->go[c - 'a'] = new Node();
            curr = curr->go[c - 'a'];
        }
        curr->end = true;
    }

    queue<Node *> q; q.push(root);
    root->fail = root;

    while (!q.empty()) {
        Node *curr = q.front(); q.pop();
        for (int i = 0; i < 26; i++) {
            Node *next = curr->go[i];
            if (!next) continue;
            q.push(next);

            if (curr == root) next->fail = root;
            else {
                Node *dest = curr->fail;
                while (dest != root && !dest->go[i]) dest = dest->fail;
                if (dest->go[i]) dest = dest->go[i];
                next->fail = dest;
                next->end |= dest->end;
            }
        }
    }

    return root;
}


bool find_trie(Node *trie, string &s) {
    Node *curr = trie;
    for (char c: s) {
        while (curr != trie && !curr->go[c - 'a']) curr = curr->fail;
        if (curr->go[c - 'a']) curr = curr->go[c - 'a'];
        if (curr->end) return true;
    }
    return false;
}
```

4.4 Manacher's Algorithm

```
vector<int> manacher(string S) {
    int n = S.size();
    vector<int> L(2 * n + 1), T(2 * n + 1);

    for (int i = 0; i < n; i++) {
        T[2 * i + 1] = S[i];
    }
    for (int i = 0, s = 0; i < 2 * n + 1; i++) {
        if (s + L[s] > i) {
            L[i] = min(L[2 * s - i], s + L[s] - i);
        }
        while (i - L[i] >= 0 && i + L[i] < 2 * n + 1 &&
            T[i + L[i]] == T[i - L[i]]) L[i] += 1;
        if (s + L[s] < i + L[i]) s = i;
    }
    for (int i = 0; i < L.size(); i++) {
        L[i] /= 2;
    }
    return L;
}
```

5 Geometry

5.1 Template, CCW, Segment intersection

```
typedef long double L; // typedef long long L;
typedef pair<L, L> pi;
const L eps = 1e-10;

pi operator+(pi a, pi b) { return pi(a.va + b.va, a.vb + b.vb); }
pi operator-(pi a, pi b) { return pi(a.va - b.va, a.vb - b.vb); }
pi operator*(L k, pi a) { return pi(k * a.va, k * a.vb); }
L operator*(pi a, pi b) { return a.va * b.va + a.vb * b.vb; }
L operator/(pi a, pi b) { return a.va * b.vb - a.vb * b.va; }

int ccw(pi a, pi b, pi c) {
    L x = (b - a) / (c - a);
    return x == 0 ? 0 : (x > 0 ? 1 : -1);
}

struct Line {
    pi s, e, d;
    Line(pi s, pi e) : s(s), e(e), d(e - s) {}
};
int intersect(Line a, Line b) {
    if (a.d / b.d == 0) {
        if ((a.s - b.s) / a.d != 0) return 0;
        if (a.e < b.s || b.e < a.s) return 0;
        return a.e == b.s || a.s == b.e ? 1 : 2;
    }
    return ccw(a.s, a.e, b.s) != ccw(a.s, a.e, b.e) &&
        ccw(b.s, b.e, a.s) != ccw(b.s, b.e, a.e);
}
pi operator&(Line a, Line b) {
    return a.s + ((b.s - a.s) / b.d) / (a.d / b.d) * a.d;
}
```

5.2 Convex Hull

```
int ccw(pi a, pi b, pi c) {
    L k = (b - a) / (c - a);
    return k == 0 ? 0 : (k > 0 ? 1 : -1);
}
vector<pi> convex_hull(vector<pi> P) {
    swap(P[0], *min_element(P.begin(), P.end()));
    sort(P.begin() + 1, P.end(), [&](pi a, pi b) {
        int k = ccw(P[0], a, b);
        if (k != 0) return k > 0;
        return a < b;
    });

    /* if (ccw(P[0], P[1], P.back()) == 0) {
        return P; // P is a line
    }
    int p = P.size() - 1;
    while (p >= 2 && ccw(P[0], P[p - 1], P[p]) == 0) p -= 1;
    reverse(P.begin() + p, P.end()); */

    vector<pi> H;
    for (int i = 0; i < P.size(); i++) {
    // If hull should contain all points on the edges, change ccw(...) <= 0 to ccw(...) < 0
        while (H.size() >= 2 && ccw(H[H.size() - 2], H.back(), P[i]) < 0) {
            H.pop_back();
        }
        H.push_back(P[i]);
    }
    return H;
}
```

5.3 Rotating Calipers

```
void rotating_calipers(vector<pi> A) {
    // A[0] should be mininum element, A should be a convex polygon and sorted in ccw
    int n = A.size(); A.push_back(A[0]);
    int l = 0, r = max_element(A.begin(), A.end()) - A.begin();

    while (1) {
        // A[l], A[r] are antipodal points

        if (n == 2 || r == n) break;
        if ((A[l + 1] - A[l]) / (A[r + 1] - A[r]) <= 0) {
            l += 1;
        }
        else r += 1;
    }
}
```

6 Miscellaneous

6.1 Random

```
mt19937 rng(1010101);
lint randInt(lint l, lint r) {
    return uniform_int_distribution<lint>(l, r)(rng);
}
```

6.2 Random in Python

```
import random
random.randrange(s, e)  # random integer from [s, e)
random.random()         # random float from [0, 1)
random.uniform(a, b)    # random float from [a, b]
random.shuffle(list)    # shuffle list
random.sample(list, n)  # sampling without replacement
```

6.3 Some primes for NTT, Hashing

$998244353 = 119 \times 2^{23} + 1$, Primitive root $= 3$
$985661441 = 235 \times 2^{22} + 1$, Primitive root $= 3$
$1012924417 = 483 \times 2^{21} + 1$, Primitive root $= 5$

6.4 Optimization

```
#pragma GCC optimize("O3")
#pragma GCC optimize("Ofast")
#pragma GCC optimize("unroll-loops")
```

6.5 Bit operations

```
int __builtin_clz(int x); // Number of leading zeros 0010 = 2
int __builtin_ctz(int x); // Number of trailing zeros 0010 = 1
int __builtin_popcount(int x); // Number of 1-bits in x 01011 = 3

int lsb(int n) { return n & -n; } // Smallest bit
int remove_lsb(int n) { return n & (n - 1); } // n - lsb(n)

// Subset iteration, used in O(3^n) dp
for (int i = x; ; i = (i - 1) & x) {
    // i is a subset of x, decreasing in terms of integer value
    if (i == 0) break;
}
```

6.6 Tricks

```
// floor(n / 1), floor(n / 2), ... has at most 2 * sqrt(n) different values
for (int l = 1; l <= n; ) {
    int q = n / l;
    int r = n / q;
    // floor(n / x) = q for x in [l, r]
    l = r + 1;
}
```

6.7 Mathematics

Pick's theorem : $A = i + b/2 - 1$

6.8 FastIO

```
static char buf[1 << 19]; // size : any number >1024
static int idx = 0;
static int bytes = 0;
static inline int _read() {
  if (!bytes || idx == bytes) {
    bytes = (int)fread(buf, sizeof(buf[0]),
      sizeof(buf), stdin);
    idx = 0;
  }
  return buf[idx++];
}
static inline int _readInt() {
  int x = 0, s = 1;
  int c = _read();
  while (c <= 32) c = _read();
  if (c == '-') s = -1, c = _read();
  while (c > 32) x = 10 * x + (c - '0'), c = _read();
  if (s < 0) x = -x;
  return x;
}
```

6.9 Randomly generated tree

Caution:

```
vector<int> parent;
int root(int x) {
    return (x == parent[x]) ? x : parent[x] = root(parent[x]);
}

void gen_tree(int nodes, vector<pair<int, int> >& edges) {
    parent.resize(nodes);
    for (int i=0; i<nodes; ++i) { parent[i] = i; }

    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<int> dist(0, nodes-1);
    edges.resize(nodes - 1);
    for (int i=0; i<nodes-1; ++i) {
        int u, v;
        do {
            u = dist(gen);
            v = dist(gen);
        } while (root(u) == root(v));
        parent[root(u)] = root(v);
        edges[i].first = u;
        edges[i].second = v;
    }
}
```

6.10 Stress test

Caution: If checker's answer is NO, assert false to make RuntimeError in checker.cpp

```
import os

while True:
    with open("input.txt", "w") as f:
        # Generate Data
        pass

    os.system("source.exe < input.txt > output.txt")
    if os.system("checker.exe < input.txt") != 0:
        break
```

6.11 Print Template

```
template<class T, size_t... I>
std::ostream& print_tup(std::ostream& out, const T& tup, std::index_sequence<I...>)
{
    out << "(";
    (..., (out << (I == 0? "" : " ") << std::get<I>(tup)));
    out << ")";
    return out;
}

template <class... T>
std::ostream& operator <<(std::ostream& out, const std::tuple<T...>& tup) {
    print_tup(out, tup, std::make_index_sequence<sizeof...(T)>());
    return out;
}

template <typename T, typename S>
std::ostream& operator <<(std::ostream& out, const std::pair<T, S>& p) {
    out << "(" << p.first << " " << p.second << ")";
    return out;
}

template <typename T>
std::ostream& operator <<(std::ostream& out, const std::vector<T>& v) {
    out << "[";
    for (int i = 0; i < (int)v.size(); i++) {
        out << v[i];
        if (i != (int)v.size()-1) out << " ";
    }
    out << "]";
    return out;
}

template <typename T, size_t size>
std::ostream& operator <<(std::ostream& out, const std::array<T,size>& v) {
    out << '[';
    for (int i = 0; i < (int)v.size(); i++) {
        out << v[i];
        if (i != (int)v.size()-1) out << ' ';
    }
    out << ']';
    return out;
}

template <typename T>
std::ostream& operator <<(std::ostream& out, const std::vector<std::vector<T> >& m) {
    out << '\n';
    for (int i=0; i<m.size(); ++i) {
        out << m[i] << "\n";
    }
    return out;
}

template <typename T>
std::ostream& operator <<(std::ostream& out, const std::set<T>& s) {
    out << '{';
    for (T x: s) {
        out << x << ' ';
    }
    out << '}';
    return out;
}

#define print(...) show(cout, #__VA_ARGS__, __VA_ARGS__)

template<typename H1>
std::ostream& show(std::ostream& out, const char* label, H1&& value) {
    return out << label << " = " << value << std::endl;
}

template<typename H1, typename ...T>
std::ostream& show(std::ostream& out, const char* label, H1&& value, T&&... rest) {
    const char* first_comma = strchr(label, ',');
    const char* left_parenthesis = strchr(label, '(');
    if (left_parenthesis != nullptr && left_parenthesis < first_comma) {
        const char* right_parenthesis = strchr(left_parenthesis, ')');
        assert(right_parenthesis != nullptr);
        const char* pcomma = strchr(right_parenthesis, ',');
        return show(out.write(label, pcomma - label) << " = " << value << ',', pcomma + 1,
        rest...);
    }
    return show(out.write(label, first_comma - label) << " = " << value << ',', first_comma
    + 1, rest...);
}
```