Inimigos do Beto - ICPC Library

Contents

1 Data Structures 1
- 1.1 BIT - Binary Indexed Tree . . . 1
- 1.2 BIT 2D Comprimida . . . 1
- 1.3 Iterative Segment Tree . . . 2
- 1.4 Iterative Segment Tree with Lazy Propagation . . . 2
- 1.5 Segment Tree with Lazy Propagation . . . 3
- 1.6 Sparse Table . . . 4
- 1.7 Policy Based Structures . . . 4
- 1.8 Max Queue . . . 4
- 1.9 Color Update . . . 4
- 1.10 KD-Tree . . . 5
- 1.11 Merge Sort Tree . . . 6

2 Graph Algorithms 6
- 2.1 Simple Disjoint Set . . . 6
- 2.2 Dinic Max Flow . . . 6
- 2.3 Minimum Vertex Cover . . . 7
- 2.4 Min Cost Max Flow . . . 7
- 2.5 Articulation Points/Bridges/Biconnected Components . . . 8
- 2.6 SCC - Strongly Connected Components / 2SAT . . . 9
- 2.7 LCA - Lowest Common Ancestor . . . 9
- 2.8 Heavy Light Decomposition . . . 9
- 2.9 Centroid Decomposition . . . 10
- 2.10 Match Algorithm Biparite . . . 10
- 2.11 Hungarian Algorithm - Maximum Cost Matching . . . 10

3 Dynamic Programming 11
- 3.1 CHT . . . 11
- 3.2 SOSDP . . . 12

4 Math 13
- 4.1 Chinese Remainder Theorem . . . 13
- 4.2 Diophantine Equations . . . 13
- 4.3 Discrete Logarithm . . . 14
- 4.4 Discrete Root . . . 14
- 4.5 Primitive Root . . . 14
- 4.6 Division Trick . . . 14
- 4.7 Prime Functions . . . 14
- 4.8 Matrix Fast Exponentiation . . . 15
- 4.9 FFT - Fast Fourier Transform . . . 15
- 4.10 NTT - Number Theoretic Transform . . . 17
- 4.11 Determinant using Mod . . . 17
- 4.12 Gauss Elimination . . . 18
- 4.13 SPF . . . 18

5 Geometry 19
- 5.1 Geometry . . . 19
- 5.2 Convex Hull . . . 21
- 5.3 Cut Polygon . . . 21
- 5.4 Smallest Enclosing Circle . . . 21
- 5.5 Minkowski . . . 21
- 5.6 Half Plane Intersection . . . 22
- 5.7 Closest Pair . . . 22
- 5.8 Maximum Scalar Point-Poly . . . 22

6 String Algorithms 23
- 6.1 KMP . . . 23
- 6.2 KMP Automaton . . . 23
- 6.3 Trie . . . 23
- 6.4 Aho-Corasick . . . 24
- 6.5 Algoritmo de Z . . . 24
- 6.6 Suffix Array . . . 24

7 Miscellaneous 25
- 7.1 Ternary Search . . . 25
- 7.2 Random Number Generator . . . 25
- 7.3 Dates . . . 25
- 7.4 Max Histogram . . . 26
- 7.5 Mo Algorithm . . . 26
- 7.6 Angular Sweep . . . 27

8 Teoremas e formulas uteis 27
- 8.1 Grafos . . . 27
- 8.2 Math . . . 28
- 8.3 Geometry . . . 28

1 Data Structures

1.1 BIT - Binary Indexed Tree

```
int bit[ms], n;

void update(int v, int idx) {
    while(idx <= n) {
        bit[idx] += v;
        idx += idx & -idx;
    }
}

int query(int idx) {
    int r = 0;
    while(idx > 0) {
        r += bit[idx];
        idx -= idx & -idx;
    }
    return r;
}
```

1.2 BIT 2D Comprimida

```
//by tfg
typedef pair<int, int> ii;
template<typename T>
struct Bit2D {
public:
    Bit2D(vector<ii> pts) {
        sort(pts.begin(), pts.end());
        for(auto a : pts) {
            if(ord.empty() || a.first != ord.back())
                ord.push_back(a.first);
        }
        fw.resize(ord.size() + 1);
        coord.resize(fw.size());
        for(auto &a : pts)
            swap(a.first, a.second);
        sort(pts.begin(), pts.end());
        for(auto &a : pts) {
            swap(a.first, a.second);
            for(int on = upper_bound(ord.begin(), ord.end(), a.first)
                - ord.begin(); on < fw.size(); on += on & -on) {
                if(coord[on].empty() || coord[on].back() != a.second)
                    coord[on].push_back(a.second);
```

```
            }
        }
        for(int i = 0; i < fw.size(); i++) {
            fw[i].assign(coord[i].size() + 1, 0);
        }
    }

    void upd(int x, int y, T v) {
        for(int xx = upper_bound(ord.begin(), ord.end(), x) - ord.
            begin(); xx < fw.size(); xx += xx & -xx) {
            for(int yy = upper_bound(coord[xx].begin(), coord[xx].end
                (), y) - coord[xx].begin(); yy < fw[xx].size(); yy +=
                yy & -yy) {
                fw[xx][yy] = max(fw[xx][yy], v);
            }
        }
    }

    T qry(int x, int y) {
        T ans = 0;
        for(int xx = upper_bound(ord.begin(), ord.end(), x) - ord.
            begin(); xx > 0; xx -= xx & -xx) {
            for(int yy = upper_bound(coord[xx].begin(), coord[xx].end
                (), y) - coord[xx].begin(); yy > 0; yy -= yy & -yy) {
                ans = max(ans, fw[xx][yy]);
            }
        }
        return ans;
    }

    void add_rect (int x1, int y1, int x2, int y2, T val) {
        upd(x1, y1, val);
        upd(x1, y2+1, -val);
        upd(x2+1, y1,  -val);
        upd(x2+1, y2+1, val);
    }

    T get_rect (int x1, int y1, int x2, int y2) {
        T ret = qry(x2, y2);
        ret = ret - qry(x1-1, y2);
        ret = ret - qry(x2, y1-1);
        ret = ret + qry(x1-1, y1-1);
        return ret;
    }

private:
    vector<int> ord;
    vector<vector<T>> fw, coord;
};
```

1.3 Iterative Segment Tree

```
struct Node {
    Node() {
        // empty constructor
    }

    Node(int v) {
        // init
    }

    Node(Node l, Node r) {
        // merge
    }

    // var
};

int n;
int a[ms];
Node seg[2*ms];

void build() {
    for(int i = 0; i < n; ++i) seg[i + n] = Node(a[i]);
    for(int i = n - 1; i > 0; --i) seg[i] = Node(seg[i<<1], seg[i
        <<1|1]); // Merge
}

void upd(int p, int value) { // set value at position p
    for(seg[p += n] = Node(value); p > 1; p >>= 1) seg[p>>1] = Node(
        seg[p], seg[p^1]); // Merge
}

Node qry(int l, int r) {
    Node lp, rp;
    for(l += n, r += n+1; l < r; l >>= 1, r >>= 1) {
        if(l&1) lp = Node(lp, seg[l++]); // Merge
        if(r&1) rp = Node(seg[--r], rp); // Merge
    }
    return Node(lp, rp);
}
```

1.4 Iterative Segment Tree with Lazy Propagation

```
struct LazyContext {
    LazyContext() {

    }

    void reset() {

    }

    void operator += (LazyContext o) {

    }
};

struct Node {
    Node() {

    }

    Node(ll c) {

    }

    Node(Node &l, Node &r) {

    }
```

```
    void apply(LazyContext lazy) {

    }
};

Node tree[2*ms];
LazyContext lazy[ms];
bool dirty[ms];
int n, h, a[ms];

void init() {
    h = 0;
    while((1 << h) < n) h++;
    for(int i = 0; i < n; i++) {
        tree[i + n] = Node(a[i]);
    }
    for(int i = n - 1; i > 0; i--) {
        tree[i] = Node(tree[i + i], tree[i + i + 1]);
        lazy[i].reset();
        dirty[i] = 0;
    }
}

void apply(int p, LazyContext &lc) {
    tree[p].apply(lc);
    if(p < n) {
        dirty[p] = true;
        lazy[p] += lc;
    }
}

void push(int p) {
    for(int s = h; s > 0; s--) {
        int i = p >> s;
        if(dirty[i]) {
            apply(i + i, lazy[i]);
            apply(i + i + 1, lazy[i]);
            lazy[i].reset();
            dirty[i] = false;
        }
    }
}

void build(int p) {
    for(p /= 2; p > 0; p /= 2) {
        tree[p] = Node(tree[p + p], tree[p + p + 1]);
        if(dirty[p]) {
            tree[p].apply(lazy[p]);
        }
    }
}

Node qry(int l, int r) {
    if(l > r) return Node();
    l += n, r += n+1;
    push(l);
    push(r - 1);
    Node lp, rp;
    for(; l < r; l /= 2, r /= 2) {
        if(l & 1) lp = Node(lp, tree[l++]);
        if(r & 1) rp = Node(tree[--r], rp);
    }
    return Node(lp, rp);
}

void upd(int l, int r, LazyContext lc) {
    if(l > r) return;
    l += n, r += n+1;
    push(l);
    push(r - 1);
    int l0 = l, r0 = r;
    for(; l < r; l /= 2, r /= 2) {
        if(l & 1) apply(l++, lc);
        if(r & 1) apply(--r, lc);
    }
    build(l0);
    build(r0 - 1);
}
```

1.5 Segment Tree with Lazy Propagation

```
int tree[4*MAXN], lazy[4*MAXN];

void build(int on = 1, int l = 0, int r = n - 1) {
  lazy[on] = 0;
  if (l == r) {
    tree[on] = a[l];
    return;
  }
  int mid = (l + r) / 2;
  build(2 * on, l, mid);
  build(2 * on + 1, mid + 1, r);
  tree[on] = tree[2*on] | tree[2*on + 1];
}

void propagate(int on, int l, int r) {
  if (lazy[on]) {
    tree[on] = lazy[on];
    if (l != r) {
      lazy[2 * on] = lazy[on];
      lazy[2 * on + 1] = lazy[on];
    }
    lazy[on] = 0;
  }
}

int query(int left, int right, int on = 1, int l = 0, int r = n - 1) {
  propagate(on, l, r);
  if (right < l || left > r) return 0;
  if (l >= left && r <= right) {
    return tree[on];
  }
  int mid = (l + r) / 2;
  int x = query(left, right, 2 * on, l, mid);
  int y = query(left, right, 2 * on + 1, mid + 1, r);
  return x | y;
}

void update(int left, int right, int val, int on = 1, int l = 0, int r
    = n - 1) {
```

```cpp
    propagate(on, l, r);
    if (right < l || left > r) return;
    if (l >= left && r <= right) {
      lazy[on] = val;
      propagate(on, l, r);
      return;
    }
    int mid = (l + r) / 2;
    update(left, right, val, 2 * on, l, mid);
    update(left, right, val, 2 * on + 1, mid + 1, r);
    tree[on] = tree[2*on] | tree[2*on + 1];
  }
```

1.6 Sparse Table

```cpp
struct Merger {
    int operator() (int a, int b) { return min(a, b); }
};

template <class T, class Merger>
class SparseTable {
public:
    void init(vector<T> a) {
        int e = 0;
        int n = a.size();
        while((1 << e) / 2 < a.size()) {
            e++;
        }
        table.resize(e, vector<T>(n));
        get.assign(n + 1, -1);
        for(int i = 0; i < n; i++) {
            table[0][i] = a[i];
            get[i+1] = get[(i+1)/2] + 1;
        }
        for(int i = 0; i + 1 < e; i++) {
            for(int j = 0; j + (1 << i) < n; j++) {
                table[i+1][j] = merge(table[i][j], table[i][j + (1 <<
                    i)]);
            }
        }
    }

    T qry(int l, int r) {
        int e = get[r - l];
        return merge(table[e][l], table[e][r - (1 << e)]);
    }
private:
    vector<vector<T>> table;
    vector<int> get;
    Merger merge;
};
```

1.7 Policy Based Structures

```cpp
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including
    tree_order_statistics_node_update
using namespace __gnu_pbds;

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

ordered_set X;
X.insert(1);
X.find_by_order(0); //pointer 0-indexed element
X.order_of_key(-5); //number of items strictly smaller than
end(X), begin(X);
```

1.8 Max Queue

```cpp
// src: tfg50
template <class T, class C = less<T>>
struct MaxQueue {
    MaxQueue() {
        clear();
    }

    void clear() {
        id = 0;
        q.clear();
    }

    void push(T x) {
        pair<int, T> nxt(1, x);
        while(q.size() > id && cmp(q.back().second, x)) {
            nxt.first += q.back().first;
            q.pop_back();
        }
        q.push_back(nxt);
    }

    T qry() {
        return q[id].second;
    }

    void pop() {
        q[id].first--;
        if(q[id].first == 0) {
            id++;
        }
    }

    bool empty() {
        if(id == q.size()) return true;
        return false;
    }

private:
    vector<pair<int, T>> q;
    int id;
    C cmp;
};
```

1.9 Color Update

```
struct range {
    int l, r;
    int v;
    range(int l = 0, int r = 0, int v = 0) : l(l), r(r), v(v) {}

    bool operator < (const range &a) const {
        return l < a.l;
    }
};

set<range> ranges;

vector<range> update(int l, int r, int v) { // [l, r)
    vector<range> ans;
    if(l >= r) return ans;
    auto it = ranges.lower_bound(l);
    if(it != ranges.begin()) {
        it--;
        if(it->r > l) {
            auto cur = *it;
            ranges.erase(it);
            ranges.insert(range(cur.l, l, cur.v));
            ranges.insert(range(l, cur.r, cur.v));
        }
    }
    it = ranges.lower_bound(r);
    if(it != ranges.begin()) {
        it--;
        if(it->r > r) {
            auto cur = *it;
            ranges.erase(it);
            ranges.insert(range(cur.l, r, cur.v));
            ranges.insert(range(r, cur.r, cur.v));
        }
    }
    for(it = ranges.lower_bound(l); it != ranges.end() && it->l < r;
        it++) {
        ans.push_back(*it);
    }
    ranges.erase(ranges.lower_bound(l), ranges.lower_bound(r));
    ranges.insert(range(l, r, v));
    return ans;
}

int query(int v) { // Substituir -1 por flag para quando nao houver
     resposta
    auto it = ranges.upper_bound(v);
    if(it == ranges.begin()) {
        return -1;
    }
    it--;
    return it->r > v ? it->v : -1;
}
```

1.10 KD-Tree

```
int d;
long long getValue(const PT &a) {return (d & 1) == 0 ? a.x : a.y; }
bool comp(const PT &a, const PT &b) {
    if((d & 1) == 0) { return a.x < b.x; }
    else { return a.y < b.y; }
}
long long sqrDist(PT a, PT b) { return (a - b) * (a - b); }

class KD_Tree {
public:
    struct Node {
        PT point;
        Node *left, *right;
    };

    void init(vector<PT> pts) {
        if(pts.size() == 0) {
            return;
        }
        int n = 0;
        tree.resize(2 * pts.size());
        build(pts.begin(), pts.end(), n);
        //assert(n <= (int) tree.size());
    }

    long long nearestNeighbor(PT point) {
        // assert(tree.size() > 0);
        long long ans = (long long) 1e18;
        nearestNeighbor(&tree[0], point, 0, ans);
        return ans;
    }
private:
    vector<Node> tree;

    Node* build(vector<PT>::iterator l, vector<PT>::iterator r, int &n
        , int h = 0) {
        int id = n++;
        if(r - l == 1) {
            tree[id].left = tree[id].right = NULL;
            tree[id].point = *l;
        } else if(r - l > 1) {
            vector<PT>::iterator mid = l + ((r - l) / 2);
            d = h;
            nth_element(l, mid - 1, r, comp);
            tree[id].point = *(mid - 1);
            // BE CAREFUL!
            // DO EVERYTHING BEFORE BUILDING THE LOWER PART!
            tree[id].left = build(l, mid, n, h^1);
            tree[id].right = build(mid, r, n, h^1);
        }
        return &tree[id];
    }

    void nearestNeighbor(Node* node, PT point, int h, long long &ans)
        {
        if(!node) {
            return;
        }
        if(point != node->point) {
            // THIS WAS FOR A PROBLEM
            // THAT YOU DON'T CONSIDER THE DISTANCE TO ITSELF!
            ans = min(ans, sqrDist(point, node->point));
        }
        d = h;
        long long delta = getValue(point) - getValue(node->point);
```

```
        if(delta <= 0) {
            nearestNeighbor(node->left, point, h^1, ans);
            if(ans > delta * delta) {
                nearestNeighbor(node->right, point, h^1, ans);
            }
        } else {
            nearestNeighbor(node->right, point, h^1, ans);
            if(ans > delta * delta) {
                nearestNeighbor(node->left, point, h^1, ans);
            }
        }
    }
};
```

1.11 Merge Sort Tree

```
const int mx = 512345;
vector<long long> tree[2*mx];
int n;
void init() {
    for(int i = n - 1; i >= 1; i--) {
        merge(all(tree[i + i]), all(tree[i + i + 1]), back_inserter(
            tree[i]));
    }
}
// Count the numbers in range [l, r] smaller or equal to k
int get(int l, int r, long long k) {
    int ans = 0; //colocar a base
    for(l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
        if (l & 1) {
            ans += upper_bound(all(tree[l]), k) - tree[l].begin();
            l++;
        }
        if (r & 1) {
            r--;
            ans += upper_bound(all(tree[r]), k) - tree[r].begin();
        }
    }
    return ans;
}

int main() {
    cin >> n;
    vector<ll> v(n);
    for (int i = n; i < 2*n; i++) {
        cin >> v[i - n];
        tree[i].push_back(v[i - n]);
    }
  init();
  //get(0, n - 1, x);
}
```

2 Graph Algorithms

2.1 Simple Disjoint Set

```
int ds[ms], sz[ms];

void dsBuild(){
    for(int i = 0; i < n; ++i){
        ds[i] = i;
        sz[i] = 1;
    }
}

int dsFind(int i){
    if(ds[i] != i) return ds[i] = dsFind(ds[i]);
    return ds[i];
}

void dsUnion(int a, int b){
    a = dsFind(a);
    b = dsFind(b);
    if(sz[a] < sz[b]) swap(a, b);
    if(a != b) sz[a] += sz[b];
    ds[b] = a;
}
```

2.2 Dinic Max Flow

```
const int ms = 1e3; // Quantidade maxima de vertices
const int me = 1e5; // Quantidade maxima de arestas

int adj[ms], to[me], ant[me], wt[me], z, n;
int copy_adj[ms], fila[ms], level[ms];

void clear() { // Lembrar de chamar no main
    memset(adj, -1, sizeof adj);
    z = 0;
}

void add(int u, int v, int k) {
    to[z] = v;
    ant[z] = adj[u];
    wt[z] = k;
    adj[u] = z++;
    swap(u, v);
    to[z] = v;
    ant[z] = adj[u];
    wt[z] = 0; // Lembrar de colocar = 0
    adj[u] = z++;
}

int bfs(int source, int sink) {
    memset(level, -1, sizeof level);
    level[source] = 0;
    int front = 0, size = 0, v;
    fila[size++] = source;
    while(front < size) {
        v = fila[front++];
        for(int i = adj[v]; i != -1; i = ant[i]) {
            if(wt[i] && level[to[i]] == -1) {
                level[to[i]] = level[v] + 1;
                fila[size++] = to[i];
            }
        }
```

```
    }
    return level[sink] != -1;
}

int dfs(int v, int sink, int flow) {
    if(v == sink) return flow;
    int f;
    for(int &i = copy_adj[v]; i != -1; i = ant[i]) {
        if(wt[i] && level[to[i]] == level[v] + 1 && (f = dfs(to[i],
            sink, min(flow, wt[i])))) {
            wt[i] -= f;
            wt[i ^ 1] += f;
            return f;
        }
    }
    return 0;
}

int maxflow(int source, int sink) {
    int ret = 0, flow;
    while(bfs(source, sink)) {
        memcpy(copy_adj, adj, sizeof adj);
        while((flow = dfs(source, sink, 1 << 30))) {
            ret += flow;
        }
    }
    return ret;
}
```

2.3 Minimum Vertex Cover

```
// + Dinic
vector<int> coverU, U, coverV, V; // ITA - P a r t i  o U LEFT,
    p a r t i  o V RIGHT, 0 indexed
bool Zu[mx], Zv[mx];
int pairU[mx], pairV[mx];
void getreach(int u) {
    if (u == -1 || Zu[u]) return;
    Zu[u] = true;
    for (int i = adj[u]; ~i; i = ant[i]) {
        int v = to[i];
        if (v == SOURCE || v == pairU[u]) continue;
        Zv[v] = true;
        getreach(pairV[v]);
    }
}

void minimumcover () {
    memset(pairU, -1, sizeof pairU);
    memset(pairV, -1, sizeof pairV);
    for (auto i : U) {
        for (int j = adj[i]; ~j; j = ant[j]) {
            if (!(j&1) && !wt[j]) {
                pairU[i] = to[j], pairV[to[j]] = i;
            }
        }
    }
    memset(Zu, 0, sizeof Zu);
    memset(Zv, 0, sizeof Zv);
    for (auto u : U) {
        if (pairU[u] == -1) getreach(u);
    }
    coverU.clear(), coverV.clear();
    for (auto u : U) {
        if (!Zu[u]) coverU.push_back(u);
    }
    for (auto v : V) {
        if (Zv[v]) coverV.push_back(v);
    }
}
```

2.4 Min Cost Max Flow

```
template <class T = int>
class MCMF {
public:
    struct Edge {
        Edge(int a, T b, T c) : to(a), cap(b), cost(c) {}
        int to;
        T cap, cost;
    };
    MCMF(int size) {
        n = size;
        edges.resize(n);
        pot.assign(n, 0);
        dist.resize(n);
        visit.assign(n, false);
    }
    pair<T, T> mcmf(int src, int sink) {
        pair<T, T> ans(0, 0);
        if(!SPFA(src, sink)) return ans;
        fixPot();
        // can use dijkstra to speed up depending on the graph
        while(SPFA(src, sink)) {
            auto flow = augment(src, sink);
            ans.first += flow.first;
            ans.second += flow.first * flow.second;
            fixPot();
        }
        return ans;
    }
    void addEdge(int from, int to, T cap, T cost) {
        edges[from].push_back(list.size());
        list.push_back(Edge(to, cap, cost));
        edges[to].push_back(list.size());
        list.push_back(Edge(from, 0, -cost));
    }
private:
    int n;
    vector<vector<int>> edges;
    vector<Edge> list;
    vector<int> from;
    vector<T> dist, pot;
    vector<bool> visit;
    /*bool dij(int src, int sink) {
        T INF = numeric_limits<T>::max();
        dist.assign(n, INF);
        from.assign(n, -1);
        visit.assign(n, false);
        dist[src] = 0;
```

```cpp
    for(int i = 0; i < n; i++) {
        int best = -1;
        for(int j = 0; j < n; j++) {
            if(visit[j]) continue;
            if(best == -1 || dist[best] > dist[j]) best = j;
        }
        if(dist[best] >= INF) break;
        visit[best] = true;
        for(auto e : edges[best]) {
            auto ed = list[e];
            if(ed.cap == 0) continue;
            T toDist = dist[best] + ed.cost + pot[best] - pot[ed.
                to];
            assert(toDist >= dist[best]);
            if(toDist < dist[ed.to]) {
                dist[ed.to] = toDist;
                from[ed.to] = e;
            }
        }
    }
    return dist[sink] < INF;
}*/
pair<T, T> augment(int src, int sink) {
    pair<T, T> flow = {list[from[sink]].cap, 0};
    for(int v = sink; v != src; v = list[from[v]^1].to) {
        flow.first = min(flow.first, list[from[v]].cap);
        flow.second += list[from[v]].cost;
    }
    for(int v = sink; v != src; v = list[from[v]^1].to) {
        list[from[v]].cap -= flow.first;
        list[from[v]^1].cap += flow.first;
    }
    return flow;
}
queue<int> q;
bool SPFA(int src, int sink) {
    T INF = numeric_limits<T>::max();
    dist.assign(n, INF);
    from.assign(n, -1);
    q.push(src);
    dist[src] = 0;
    while(!q.empty()) {
        int on = q.front();
        q.pop();
        visit[on] = false;
        for(auto e : edges[on]) {
            auto ed = list[e];
            if(ed.cap == 0) continue;
            T toDist = dist[on] + ed.cost + pot[on] - pot[ed.to];
            if(toDist < dist[ed.to]) {
                dist[ed.to] = toDist;
                from[ed.to] = e;
                if(!visit[ed.to]) {
                    visit[ed.to] = true;
                    q.push(ed.to);
                }
            }
        }
    }
    return dist[sink] < INF;
}
```

```cpp
    void fixPot() {
        T INF = numeric_limits<T>::max();
        for(int i = 0; i < n; i++) {
            if(dist[i] < INF) pot[i] += dist[i];
        }
    }
};
```

2.5 Articulation Points/Bridges/Biconnected Components

```cpp
int adj[ms], to[me], ant[me], z;
int num[ms], low[ms], timer;
int art[ms], bridge[me], rch;
int bc[ms], nbc;
stack<int> st;
bool f[me];

void clear() { // Lembrar de chamar no main
    memset(adj, -1, sizeof adj);
    z = 0;
}

void add(int u, int v) {
    to[z] = v;
    ant[z] = adj[u];
    adj[u] = z++;
}

void generateBc (int v) {
    while (!st.empty()) {
        int u = st.top();
        st.pop();
        bc[u] = nbc;
        if (v == u) break;
    }
    ++nbc;
}

void dfs (int v, int p) {
    st.push(v);
    low[v] = num[v] = ++timer;
    for (int i = adj[v]; i != -1; i = ant[i]) {
        if (f[i] || f[i^1]) continue;
        f[i] = 1;
        int u = to[i];
        if (num[u] == -1) {
            dfs(u, v);
            if (low[u] > num[v]) bridge[i] = bridge[i^1] = 1;
            art[v] |= p != -1 && low[u] >= num[v];
            if (p == -1 && rch > 1) art[v] = 1;
            else rch ++;
            low[v] = min(low[v], low[u]);
        } else {
            low[v] = min(low[v], num[u]);
        }
    }
    if (low[v] == num[v]) generateBc(v);
}

void biCon (int n) {
```

```
    nbc = 0, timer = 0;
    memset(num, -1, sizeof num);
    memset(bc, -1, sizeof bc);
    memset(bridge, 0, sizeof bridge);
    memset(art, 0, sizeof art);
    memset(f, 0, sizeof f);
    for (int i = 0; i < n; i++) {
        if (num[i] == -1) dfs(i, 0);
    }
}
```

2.6 SCC - Strongly Connected Components / 2SAT

```
vector<int> g[ms];
int idx[ms], low[ms], z, comp[ms], ncomp, n;

stack<int> st;
// Operacoes comuns de 2-sat
int NOT(int x) { return x < n ? x + n : x - n; }
void addImp(int a, int b) { g[a].push_back(b); }
void addOr(int a, int b) { addImp(NOT(a), b); addImp(NOT(b), a); }
void addEqual(int a, int b) { addOr(a, NOT(b)); addOr(NOT(a), b); }
void addDiff(int a, int b) { addEqual(a, NOT(b)); }
// valoracao: value[v] = comp[trad(v)] < comp[trad(~v)]

int dfs(int u) {
    if(~idx[u]) return idx[u] ? idx[u] : z;
    low[u] = idx[u] = z++;
    st.push(u);
    for(int v : g[u]) {
        low[u] = min(low[u], dfs(v));
    }
    if(low[u] == idx[u]) {
        while(st.top() != u) {
            int v = st.top();
            idx[v] = 0;
            low[v] = low[u];
            comp[v] = ncomp;
            st.pop();
        }
        idx[st.top()] = 0;
        st.pop();
        comp[u] = ncomp++;
    }
    return low[u];
}

bool solveSat() {
    memset(idx, -1, sizeof idx);
    z = 1; ncomp = 0;
    for(int i = 0; i < 2*n; i++) dfs(i);
    for(int i = 0; i < n; i++) if(comp[i] == comp[NOT(i)]) return
        false;
    return true;
}
```

2.7 LCA - Lowest Common Ancestor

```
int par[ms][mlg + 1], lvl[ms];
vector<int> g[ms];

void dfs(int v, int p, int l = 0){ // chamar dfs(parent, parent)
    lvl[v] = l;
    par[v][0] = p;
    for(int k = 1; k <= mlg; ++k){
        par[v][k] = par[par[v][k-1]][k-1];
    }
    for(auto u : g[v]){
        if(u != p){
            dfs(u, v, l+1);
        }
    }
}

int lca(int a, int b){
    if(lvl[b] > lvl[a]) swap(a, b);
    for(int i = mlg; i >= 0; --i){
        if(lvl[a] - (1 << i) >= lvl[b]) a = par[a][i];
    }

    if(a == b) return a;

    for(int i = mlg; i>= 0; --i){
        if(par[a][i] != par[b][i]) a = par[a][i], b = par[b][i];
    }
    return par[a][0];
}
```

2.8 Heavy Light Decomposition

```
//HLD + ETT by adamant http://codeforces.com/blog/entry/53170
//query of path and subtree of p (in[p], out[p]) [l, r)
int sz[ms], par[ms], h[ms];
int t, in[ms], out[ms], rin[ms], nxt[ms];

void dfs_sz(int v = 0, int p = -1){
    sz[v] = 1;
    for(int i = 0; i < g[v].size(); ++i){
        int &u = g[v][i];
        if(u == p) continue;
        h[u] = h[v]+1, par[u] = v;
        dfs_sz(u, v);
        sz[v] += sz[u];
        if(g[v][0] == p || sz[u] > sz[g[v][0]]){
            swap(u, g[v][0]);
        }
    }
}

void dfs_hld(int v = 0, int p = -1){
    in[v] = t++;
    rin[in[v]] = v;
    for(auto u : g[v]){
        if(u == p) continue;
        nxt[u] = u == g[v][0] ? nxt[v] : u;
        dfs_hld(u, v);
    }
    out[v] = t;
```

```
}

int up(int v){
    return (nxt[v] != v) ? nxt[v] : (~par[v] ? par[v] : v);
}

int getLCA(int a, int b){
    while(nxt[a] != nxt[b]){
        if(h[a] == 0 || h[up(a)] < h[up(b)]) swap(a, b);
        a = up(a);
    }
    return h[a] < h[b] ? a : b;
}

vector<ii> getPathAncestor(int a, int anc){
    vector<ii> ans;
    while(nxt[a] != nxt[anc]){
        ans.emplace_back(in[nxt[a]], in[a]);
        a = par[nxt[a]];
    }
    ans.emplace_back(in[anc], in[a]);
    return ans;
}

int queryPath(int a, int b){
    int res = 0;
    while(nxt[a] != nxt[b]){
        if(h[nxt[a]] > h[nxt[b]]) swap(a, b);
        int cur = qry(in[nxt[b]], in[b]);
        res = max(res, cur);
        b = par[nxt[b]];
    }
    if(h[a] > h[b]) swap(a, b);
    int cur = qry(in[a], in[b]); // in[a] + 1 dont include LCA
    res = max(res, cur);

    return res;
}
```

2.9 Centroid Decomposition

```
const int MAXN = 1e5 + 7;

set<int> adj[MAXN];
int parent[MAXN], sz[MAXN];

void dfsSubtree(int u, int p) {
  sz[u] = 1;
  for (auto v : adj[u]) {
    if (v != p && !removed[v]) {
      dfsSubtree(v, u);
      sz[u] += sz[v];
    }
  }
}

int getCentroid(int u, int p, int size) {
  for (auto v : adj[u]) {
    if (v != p && !removed[v] && sz[v] * 2 >= size) return getCentroid
        (v, u, size);
  }
  return u;
}

void decompose(int u, int p) {
  dfsSubtree(u, -1);
  int ctr = getCentroid(u, -1, sz[u]);
  if (p == -1) {
    p = ctr;
  }
  parent[ctr] = p;
  removed[ctr] = 1;
  for (auto v : adj[ctr]) {
    if (v != p && !removed[v]) {
      decompose(v, ctr);
    }
  }
}
```

2.10 Match Algorithm Biparite

```
// ADACITY - Matching
const int INF = 0x3f3f3f3f;
const int MAXN = 500 + 7;

int friends[MAXN], match[MAXN], adj[MAXN][MAXN];
bool vis[MAXN];
vector<int> v[MAXN];
int n, m, f, t;

int solve(int u) {
  if(vis[u]) return 0;
  vis[u] = true;
  for(int i = 0; i < v[u].size(); i++) {
    int w = v[u][i];
    if(match[w] == -1 || solve(match[w])) {
      match[w] = u;
      return 1;
    }
  }
  return 0;
}

void clear() {
  for(int i = 1; i <= MAXN; i++) {
    v[i].clear();
  }
  for(int i = 1; i <= MAXN; i++) {
    for(int j = 1;j <= MAXN; j++) {
      adj[i][j] = INF;
    }
    adj[i][i] = 0;
  }
  memset(match, -1, sizeof(match));
}
```

2.11 Hungarian Algorithm - Maximum Cost Matching

```
// 1-indexed by ITA
const int INF = 0x3f3f3f3f;
const int MAXN = 2009, MAXM = 2009;

int n, m;
int pu[MAXN], pv[MAXN], cost[MAXN][MAXM];
int pairV[MAXM], way[MAXM], minv[MAXM];  //pairV[i] = id of worker assigned to do job i or 0
bool used[MAXM];

void clear () {
    memset(pu, 0, sizeof pu);
    memset(pv, 0, sizeof pv);
    memset(way, 0, sizeof way);
    memset(cost, 0, sizeof cost); // remember to change (0 for max, 0x3f for min)
    memset(cost[0], 0, sizeof cost[0]);
}

void hungarian () {
    memset(pairV, 0, sizeof pairV);
    for (int i = 1, j0 = 0; i <=n; i++) {
        pairV[0] = i;
        memset(minv, INF, sizeof minv);
        memset(used, false, sizeof used);
        do {
            used[j0] = true;
            int i0 = pairV[j0], delta = INF, j1;
            for (int j = 1; j <= m; j++) {
                if (used[j]) continue;
                int cur = cost[i0][j] - pu[i0] - pv[j];
                if (cur < minv[j])
                    minv[j] = cur, way[j] = j0;
                if (minv[j] < delta)
                    delta = minv[j], j1=j;
            }
            for (int j = 0; j <= m; j++) {
                if (used[j])
                    pu[pairV[j]] += delta, pv[j] -= delta;
                else minv[j] -= delta;
            }
            j0 = j1;
        } while (pairV[j0] != 0);
        do {
            int j1 = way[j0];
            pairV[j0] = pairV[j1];
            j0 = j1;
        } while (j0);
    }
}
```

3 Dynamic Programming

3.1 CHT

```
typedef long double ldouble_t;
typedef long long ll;

class HullDynamic {
public:
    const ldouble_t inf = 1e9;

    struct Line {
        ll m, b;
        ldouble_t start;
        bool is_query;

        Line() {}

        Line(ll _m, ll _b, ldouble_t _start, bool _is_query) : m(_m),
            b(_b), start(_start), is_query(_is_query) {}

        ll eval(ll x) {
            return m * x + b;
        }

        ldouble_t intersect(const Line& l) const {
            return (ldouble_t) (l.b - b) / (m - l.m);
        }

        bool operator< (const Line& l) const {
            if (is_query == 0) return m < l.m; //     > min     < max
            return (start < l.start);
        }
    };

    typedef set<Line>::iterator iterator_t;

    bool has_prev(iterator_t it) {
        return (it != hull.begin());
    }

    bool has_next(iterator_t it) {
        return (++it != hull.end());
    }

    bool irrelevant(iterator_t it) {
        if (!has_prev(it) || !has_next(it)) return 0;
        iterator_t prev = it, next = it;
        prev--;
        next++;
        return next->intersect(*prev) <= it->intersect(*prev);
    }

    void update_left(iterator_t it) {
        if (it == hull.begin()) return;
        iterator_t pos = it;
        --it;
        vector<Line> rem;
        while(has_prev(it)) {
            iterator_t prev = it;
            --prev;
            if (prev->intersect(*pos) <= prev->intersect(*it)) {
                rem.push_back(*it);
            } else {
                break;
            }
            --it;
        }
```

```
        ldouble_t start = pos->intersect(*it);
        Line f = *pos;
        for (Line r : rem) hull.erase(r);
        hull.erase(f);
        f.start = start;
        hull.insert(f);
    }

    void update_right(iterator_t it) {
        if (!has_next(it)) return;
        iterator_t pos = it;
        ++it;
        vector<Line> rem;
        while(has_next(it)) {
            iterator_t next = it;
            ++next;
            if (next->intersect(*pos) <= pos->intersect(*it)) {
                rem.push_back(*it);
            } else {
                break;
            }
            ++it;
        }
        ldouble_t start = pos->intersect(*it);
        Line f = *it;
        for (Line r : rem) hull.erase(r);
        hull.erase(f);
        f.start = start;
        hull.insert(f);
    }

    void add(ll m, ll b) {
        Line f(m, b, -inf, 0);
        iterator_t it = hull.lower_bound(f);
        if (it != hull.end() && it->m == f.m) {
            if (it->b <= f.b) {
                return;
            } else if (it->b > f.b) {
                hull.erase(it);
            }
        }
        hull.insert(f);
        it = hull.lower_bound(f);
        if (irrelevant(it)) {
            hull.erase(it);
            return;
        }
        update_left(it);
        it = hull.lower_bound(f);
        update_right(it);
    }

    ll query(ll x) {
        Line f(0, 0, x, 1);
        iterator_t it = hull.upper_bound(f);
        assert(it != hull.begin());
        --it;
        return it->m * x + it->b;
    }

private:
    set<Line> hull;
};

//mais rapido

bool Q;
struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const {
        return Q ? p < o.p : k < o.k;
    }
};

struct HullDynamic : multiset<Line> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    const ll inf = LLONG_MAX;

    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b);
    }

    bool isect(iterator x, iterator y) {
        if (y == end()) { x->p = inf; return false; }
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }

    void add(ll k, ll m) { // para min multiplicar por -1
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }

    ll query(ll x) {
        assert(!empty());
        Q = 1; auto l = *lower_bound({0,0,x}); Q = 0;
        return l.k * x + l.m; // para min multiplicar por -1
    }
};
```

3.2 SOSDP

```
// F[i] = Sum of all A[j] where j is a submask of i
for(int i = 0; i<(1<<N); ++i)
  F[i] = A[i];
for(int i = 0;i < N; ++i){
    for(int mask = 0; mask < (1<<N); ++mask){
        if(mask & (1<<i))
            F[mask] += F[mask^(1<<i)];
    }
}

// Submasks
for (int s=m; ; s=(s-1)&m) {

    if (s==0)  break;
}
```

4 Math

4.1 Chinese Remainder Theorem

```
const long long N = 20;

long long GCD(long long a, long long b) {
  return (b == 0) ? a : GCD(b, a % b);
}
inline long long get_LCM(long long a, long long b) {
  return a / GCD(a, b) * b;
}
inline long long normalize(long long x, long long mod) {
  x %= mod;
  if (x < 0) x += mod;
  return x;
}

struct GCD_type {
  long long x, y, d;
};
GCD_type ex_GCD(long long a, long long b){
  if (b == 0) return {1, 0, a};
  GCD_type pom = ex_GCD(b, a % b);
  return {pom.y, pom.x - a / b * pom.y, pom.d};
}

long long testCases;
long long t;
long long a[N], n[N], ans, LCM;

int main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  t = 2;
  long long T;
  cin >> T;
  while(T--) {
    for(long long i = 1; i <= t; i++) {
      cin >> a[i] >> n[i];
      normalize(a[i], n[i]);
    }
    ans = a[1];
    LCM = n[1];
    bool impossible = false;
    for(long long i = 2; i <= t; i++) {
      auto pom = ex_GCD(LCM, n[i]);
      long long x1 = pom.x;
      long long d = pom.d;
      if((a[i] - ans) % d != 0) {
        impossible = true;
      }
      ans = normalize(ans + x1 * (a[i] - ans) / d % (n[i] / d) * LCM,
          LCM * n[i] / d);
      LCM = get_LCM(LCM, n[i]);
    }
    if (impossible) cout << "no solution\n";
    else cout << ans << " " << LCM << endl;
  }
  return 0;
}
```

4.2 Diophantine Equations

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

bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &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 (int & x, int & y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}

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

    int sign_a = a>0 ? +1 : -1;
    int 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;
    int lx1 = x;

    shift_solution (x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution (x, y, a, b, -sign_b);
    int 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;
    int lx2 = x;

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

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

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

4.3 Discrete Logarithm

```
ll discreteLog(ll a, ll b, ll m) {
    // a^ans == b mod m
    // ou -1 se nao existir
    ll cur = a, on = 1;
    for(int i = 0; i < 100; i++) {
        cur = cur * a % m;
    }
    while(on * on <= m) {
        cur = cur * a % m;
        on++;
    }
    map<ll, ll> position;
    for(ll i = 0, x = 1; i * i <= m; i++) {
        position[x] = i * on;
        x = x * cur % m;
    }
    for(ll i = 0; i <= on + 20; i++) {
        if(position.count(b)) {
            return position[b] - i;
        }
        b = b * a % m;
    }
    return -1;
}
```

4.4 Discrete Root

```
//x^k = a % mod
ll discreteRoot(ll k, ll a, ll mod) {
    ll g = primitiveRoot(mod);
    ll y = discreteLog(fexp(g, k, mod), a, mod);
    if (y == -1) {
        return y;
    }
    return fexp(g, y, mod);
}
```

4.5 Primitive Root

```
int fexp(int x, int y, int p){
  int ans = 1;
  x = x % p;
  while (y) {
    if (y & 1) ans = (ans * x) % p;
    x = (x * x) % p;
    y = y >> 1;
  }
  return ans;
}
// If p has a primitive root, then there are phi(phi(p)) primitives
    roots of p
int primitiveRoot(int p) {
  vector<int> factors;
  int phi = p - 1; // phi(n)
  int n = phi;
  for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      factors.push_back(i);
      while (n % i == 0) n /= i;
    }
  }
  if (n > 1) factors.push_back(n);
  for (int i = 2; i <= p; i++) {
    bool ok = true;
    for (int j = 0; j < factors.size() && ok; j++) {
      ok &= (fexp(i, phi / factors[j], p) != 1);
    }
    if (ok) return i;
  }
  return -1;
}
```

4.6 Division Trick

```
for(int l = 1, r; l <= n; l = r + 1) {
    r = n / (n / l);
    // n / i has the same value for l <= i <= r
}
```

4.7 Prime Functions

```
vector<int> prime;
bool notPrime[ms];

//O(nlogn)
void sieve(){
    for(int i = 2; i<ms; ++i){
        if(!notPrime[i]){
            prime.push_back(i);
            for(int j = 2*i; j < ms; j += i){
```

```cpp
                    notPrime[j] = true;
                }
            }
        }
}

//O(nloglogn)
void sieve(){
    prime.push_back(2);
    for(int i = 4; i<ms; i += 2){
        notPrime[i] = true;
    }
    for(ll i = 3; i<ms; i += 2){
        if(!notPrime[i]){
            prime.push_back(i);
            for(ll j = i*i; j < ms; j += i+i){
                notPrime[j] = true;
            }
        }
    }
}

vector<int> fact(int x){
    vector<int> ans;
    int idx = 0, pf = prime[idx];
    while(pf * pf <= x){
        while(x%pf == 0){
            x /= pf;
            ans.push_back(pf);
        }
        pf = prime[++idx];
    }
    if(x != 1){
        ans.push_back(x);
    }
    return ans;
}
```

4.8 Matrix Fast Exponentiation

```cpp
struct Matrix{
    long long mat[m][m];

    Matrix operator * (const Matrix &p){
        Matrix ans;
        for(int i = 0; i < m; ++i){
            for(int j = 0; j < m; ++j){
                for(int k = ans.mat[i][j] = 0; k < m; ++k){
                    ans.mat[i][j] = (ans.mat[i][j] + (mat[i][k] * p.
                        mat[k][j]) % mod) % mod;
                }
            }
        }
        return ans;
    }
};

Matrix fExp(Matrix a, long long b){
    Matrix ans;
    for(int i = 0; i < m; ++i) for(int j = 0; j < m; ++j) ans.mat[i][j
        ] = (i == j);

    while(b){
        if(b & 1) ans = ans * a;
        a = a*a;
        b >>= 1;
    }
    return ans;
}


// precompute
for(int k = 1;k <= 62;k++){
    memcpy(T[k], T[k-1], sizeof(T[k-1]));
    T[k] = T[k] * T[k];
}
for(int i = 0; i < 63 && (1LL<<i) <= N; i++){
        if(N&(1LL<<i)) S = S * T[i];
}
```

4.9 FFT - Fast Fourier Transform

```cpp
//by TFG
typedef double ld;

const ld PI = acos(-1);

struct Complex {
        ld real, imag;
        Complex conj() { return Complex(real, -imag); }
        Complex(ld a = 0, ld b = 0) : real(a), imag(b) {}
        Complex operator + (const Complex &o) const { return Complex(
            real + o.real, imag + o.imag); }
        Complex operator - (const Complex &o) const { return Complex(
            real - o.real, imag - o.imag); }
        Complex operator * (const Complex &o) const { return Complex(
            real * o.real - imag * o.imag, real * o.imag + imag * o.
            real); }
        Complex operator / (ld o) const { return Complex(real / o,
            imag / o); }
        void operator *= (Complex o) { *this = *this * o; }
        void operator /= (ld o) { real /= o, imag /= o; }
};

typedef vector<Complex> CVector;

const int ms = 1 << 22;

int bits[ms];
Complex root[ms];

void initFFT() {
        root[1] = Complex(1);
        for(int len = 2; len < ms; len += len) {
                Complex z(cos(PI / len), sin(PI / len));
                for(int i = len / 2; i < len; i++) {
                        root[2 * i] = root[i];
                        root[2 * i + 1] = root[i] * z;
                }
```

```
        }
}

void pre(int n) {
        int LOG = 0;
        while(1 << (LOG + 1) < n) {
                LOG++;
        }
        for(int i = 1; i < n; i++) {
                bits[i] = (bits[i >> 1] >> 1) | ((i & 1) << LOG);
        }
}


CVector fft(CVector a, bool inv = false) {
        int n = a.size();
        pre(n);
        if(inv) {
                reverse(a.begin() + 1, a.end());
        }
        for(int i = 0; i < n; i++) {
                int to = bits[i];
                if(to > i) {
                        swap(a[to], a[i]);
                }
        }
        for(int len = 1; len < n; len *= 2) {
                for(int i = 0; i < n; i += 2 * len) {
                        for(int j = 0; j < len; j++) {
                                Complex u = a[i + j], v = a[i + j +
                                    len] * root[len + j];
                                a[i + j] = u + v;
                                a[i + j + len] = u - v;
                        }
                }
        }
        if(inv) {
                for(int i = 0; i < n; i++)
                        a[i] /= n;
        }
        return a;
}

void fft2in1(CVector &a, CVector &b) {
        int n = (int) a.size();
        for(int i = 0; i < n; i++) {
                a[i] = Complex(a[i].real, b[i].real);
        }
        auto c = fft(a);
        for(int i = 0; i < n; i++) {
                a[i] = (c[i] + c[(n-i) % n].conj()) * Complex(0.5, 0);
                b[i] = (c[i] - c[(n-i) % n].conj()) * Complex(0, -0.5)
                    ;
        }
}

void ifft2in1(CVector &a, CVector &b) {
        int n = (int) a.size();
        for(int i = 0; i < n; i++) {
                a[i] = a[i] + b[i] * Complex(0, 1);
        }
        a = fft(a, true);
        for(int i = 0; i < n; i++) {
                b[i] = Complex(a[i].imag, 0);
                a[i] = Complex(a[i].real, 0);
        }
}


vector<long long> mod_mul(const vector<long long> &a, const vector<
    long long> &b, long long cut = 1 << 15) {
        // TODO cut memory here by /2
        int n = (int) a.size();
        CVector C[4];
        for(int i = 0; i < 4; i++) {
                C[i].resize(n);
        }
        for(int i = 0; i < n; i++) {
                C[0][i] = a[i] % cut;
                C[1][i] = a[i] / cut;
                C[2][i] = b[i] % cut;
                C[3][i] = b[i] / cut;
        }
        fft2in1(C[0], C[1]);
        fft2in1(C[2], C[3]);
        for(int i = 0; i < n; i++) {
                // 00, 01, 10, 11
                Complex cur[4];
                for(int j = 0; j < 4; j++) cur[j] = C[j/2+2][i] * C[j
                    % 2][i];
                for(int j = 0; j < 4; j++) C[j][i] = cur[j];
        }
        ifft2in1(C[0], C[1]);
        ifft2in1(C[2], C[3]);
        vector<long long> ans(n, 0);
        for(int i = 0; i < n; i++) {
                // if there are negative values, care with rounding
                ans[i] += (long long) (C[0][i].real + 0.5);
                ans[i] += (long long) (C[1][i].real + C[2][i].real +
                    0.5) * cut;
                ans[i] += (long long) (C[3][i].real + 0.5) * cut * cut
                    ;
        }
        return ans;
}

vector<int> mul(const vector<int> &a, const vector<int> &b) {
        int n = 1;
        while (n - 1 < (int) a.size() + (int) b.size() - 2) n += n;
        CVector poly(n);
        for(int i = 0; i < n; i++) {
                if(i < (int) a.size()) {
                        poly[i].real = a[i];
                }
                if(i < (int) b.size()) {
                        poly[i].imag = b[i];
                }
        }
        poly = fft(poly);
        for(int i = 0; i < n; i++) {
                poly[i] *= poly[i];
        }
```

```
        poly = fft(poly, true);
        vector<int> c(n, 0);
        for(int i = 0; i < n; i++) {
                c[i] = (int) (poly[i].imag / 2 + 0.5);
        }
        while (c.size() > 0 && c.back() == 0) c.pop_back();
        return c;
}
```

4.10 NTT - Number Theoretic Transform

```
//by TFG
typedef long long ll;

const int MOD = 998244353;
const int me = 15;
const int ms = 1 << me;
ll fexp(ll x, ll e, ll mod = MOD) {
        ll ans = 1;
        x %= mod;
        for(; e > 0; e /= 2) {
                if(e & 1) {
                        ans = ans * x % mod;
                }
                x = x * x % mod;
        }
        return ans;
}
//is n primitive root of p ?
bool test(ll x, ll p) {
        ll m = p - 1;
        for(int i = 2; i * i <= m; ++i) if(!(m % i)) {
                if(fexp(x, i, p) == 1) return false;
                if(fexp(x, m / i, p) == 1) return false;
        }
        return true;
}
//find the largest primitive root for p
int search(int p) {
        for(int i = p - 1; i >= 2; --i) if(test(i, p)) return i;
        return -1;
}
map<int, int> roots;
int get_root(int p) {
        if(roots[p]) {
                return roots[p];
        } else {
                roots[p]=search(p);
                return roots[p];
        }
}

#define add(x, y) x+y>=MOD?x+y-MOD:x+y

const int gen = search(MOD);
int bits[ms], root[ms];

void initFFT() {
        root[1] = 1;
        for(int len = 2; len < ms; len += len) {
                int z = fexp(gen, (MOD - 1) / len / 2);
                for(int i = len / 2; i < len; i++) {
                        root[2 * i] = root[i];
                        root[2 * i + 1] = (long long) root[i] * z %
                            MOD;
                }
        }
}

void pre(int n) {
        int LOG = 0;
        while(1 << (LOG + 1) < n) {
                LOG++;
        }
        for(int i = 1; i < n; i++) {
                bits[i] = (bits[i >> 1] >> 1) | ((i & 1) << LOG);
        }
}

vector<int> fft(vector<int> a, int mod, bool inv = false) {
        int n = (int) a.size();
        pre(n);
        if(inv) {
                reverse(a.begin() + 1, a.end());
        }
        for(int i = 0; i < n; i++) {
                int to = bits[i];
                if(i < to) { swap(a[i], a[to]); }
        }
        for(int len = 1; len < n; len *= 2) {
                for(int i = 0; i < n; i += len * 2) {
                        for(int j = 0; j < len; j++) {
                                int u = a[i + j], v = (ll) a[i + j +
                                    len] * root[len + j] % mod;
                                a[i + j] = add(u, v);
                                a[i + j + len] = add(u, mod - v);
                        }
                }
        }
        if(inv) {
                int rev = fexp(n, mod-2, mod);
                for(int i = 0; i < n; i++)
                        a[i] = (ll) a[i] * rev % mod;
        }
        return a;
}
```

4.11 Determinant using Mod

```
// by zchao1995
// Determinante com coordenadas inteiras usando Mod

ll mat[ms][ms];

ll det (int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            mat[i][j] %= mod;
        }
    }
```

```
    ll res = 1;
    for (int i = 0; i < n; i++) {
        if (!mat[i][i]) {
            bool flag = false;
            for (int j = i + 1; j < n; j++) {
                if (mat[j][i]) {
                    flag = true;
                    for (int k = i; k < n; k++) {
                        swap (mat[i][k], mat[j][k]);
                    }
                    res = -res;
                    break;
                }
            }
            if (!flag) {
                return 0;
            }
        }
        for (int j = i + 1; j < n; j++) {
            while (mat[j][i]) {
                ll t = mat[i][i] / mat[j][i];
                for (int k = i; k < n; k++) {
                mat[i][k] = (mat[i][k] - t * mat[j][k]) % mod;
                swap (mat[i][k], mat[j][k]);
                }
                res = -res;
            }
        }
        res = (res * mat[i][i]) % mod;
    }
    return (res + mod) % mod;
}
```

4.12 Gauss Elimination

```
const double eps = 1e-7;

int gauss (vector<vector<double>> a, vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i){
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        }
        if (abs (a[sel][col]) < eps) continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;
        for (int i=0; i<n; ++i){
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        }
        ++row;
    }
    ans.assign (m, 0);
    for (int i=0; i<m; ++i){
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    }
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > eps)
            return 0;
    }
    for (int i=0; i<m; ++i){
        if (where[i] == -1)
            return INF;
    }
    return 1;
}

// mod 2 (xor);
int gauss (vector <bitset<ms>> a, int m) {
    int n = (int) a.size();
    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        for (int i=row; i<n; ++i){
            if (a[i][col]) {
                swap (a[i], a[row]);
                break;
            }
        }
        if (!a[row][col]) continue;
        where[col] = row;
        for (int i=0; i<n; ++i){
            if (i != row && a[i][col])
                a[i] ^= a[row];
        }
        ++row;
    }
    //same above
}
```

4.13 SPF

```
int spf[MAXN];

// Complexity: O(nloglogn)
void sieve() {
  spf[1] = 1;
  for (int i = 2; i * i < MAXN; i++) {
    if (!spf[i]) {
      for (int j = i; j < MAXN; j += i) {
        if (!spf[j]) spf[j] = i;
      }
    }
  }
}

// Complexity: O(log n)
vector<int> getFactors(int n) {
```

```
    vector<int> ans;
    while (n != 1) {
      ans.push_back(spf[n]);
      n = n / spf[n];
    }
    return ans;
  }
```

5 Geometry

5.1 Geometry

```
const double inf = 1e100, eps = 1e-9;
const double PI = acos(-1.0L);

int cmp (double a, double b = 0) {
    if (abs(a-b) < eps) return 0;
    return (a < b) ? -1 : +1;
}

struct PT {
    double x, y;
    PT(double x = 0, double y = 0) : x(x), y(y) {}
    PT operator + (const PT &p) const { return PT(x+p.x, y+p.y); }
    PT operator - (const PT &p) const { return PT(x-p.x, y-p.y); }
    PT operator * (double c) const { return PT(x*c, y*c); }
    PT operator / (double c) const { return PT(x/c, y/c); }

    bool operator <(const PT &p) const {
        if(cmp(x, p.x) != 0) return x < p.x;
        return cmp(y, p.y) < 0;
    }
    bool operator ==(const PT &p) const {
        return !cmp(x, p.x) && !cmp(y, p.y);
    }
    bool operator !=(const PT &p) const {
        return !(p == *this);
    }
};

double dot (PT p, PT q) { return p.x * q.x + p.y*q.y; }
double cross (PT p, PT q) { return p.x * q.y - p.y*q.x; }
double dist2 (PT p, PT q = PT(0, 0)) { return dot(p-q, p-q); }
double dist (PT p, PT q) { return hypot(p.x-q.x, p.y-q.y); }
double norm (PT p) { return hypot(p.x, p.y); }
PT normalize (PT p) { return p/hypot(p.x, p.y); }
double angle (PT p, PT q) { return atan2(cross(p, q), dot(p, q)); }
double angle (PT p) { return atan2(p.y, p.x); }
double polarAngle (PT p) {
    double a = atan2(p.y,p.x);
    return a < 0 ? a + 2*PI : a;
}

// - p.y*sen(+90), p.x*sen(+90)
PT rotateCCW90 (PT p) { return PT(-p.y, p.x); }
// - p.y*sen(-90), p.x*sen(-90)
PT rotateCW90 (PT p) { return PT(p.y, -p.x); }

PT rotateCCW (PT p, double t) {
    return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}

// a.b = |a| cost * |b|
//ponto c
PT projectPointLine (PT a, PT b, PT c) {
    return a + (b-a) * dot(b-a, c-a)/dot(b-a, b-a);
}

PT reflectPointLine (PT a, PT b, PT c) {
    PT p = projectPointLine(a, b, c);
    return p*2 - c;
}

PT projectPointSegment (PT a, PT b, PT c) {
    double r = dot(b-a, b-a);
    if (cmp(r) == 0) return a;
    r = dot(b-a, c-a)/r;
    if (cmp(r, 0) == 0) return a;
    if (cmp(r, 1) == 0) return b;
    return a + (b - a) * r;
}

double distancePointSegment (PT a, PT b, PT c) {
    return dist(c, projectPointSegment(a, b, c));
}

// Parallel and opposite directions
bool ptInSegment (PT a, PT b, PT c) {
    if (a == b) return a == c;
    a = a-c, b = b-c;
    return cmp(cross(a, b)) == 0 && cmp(dot(a, b)) <= 0;
}

bool parallel (PT a, PT b, PT c, PT d) {
    return cmp(cross(b - a, c - d)) == 0;
}

bool collinear (PT a, PT b, PT c, PT d) {
    return parallel(a, b, c, d) && cmp(cross(a - b, a - c)) == 0 &&
        cmp(cross(c - d, c - a)) == 0;
}

bool segmentsIntersect (PT a, PT b, PT c, PT d) {
    if (collinear(a, b, c, d)) {
        if (cmp(dist(a, c)) == 0 || cmp(dist(a, d)) == 0 || cmp(dist(b
            , c)) == 0 || cmp(dist(b, d)) == 0) return true;
        if (cmp(dot(c - a, c - b)) > 0 && cmp(dot(d - a, d - b)) > 0
            && cmp(dot(c - b, d - b)) > 0) return false;
        return true;
    }
    if (cmp(cross(d - a, b - a) * cross(c - a, b - a)) > 0) return
        false;
    if (cmp(cross(a - c, d - c) * cross(b - c, d - c)) > 0) return
        false;
    return true;
}

// r = a1 + t*d1, (r - a2) x d2 = 0
PT computeLineIntersection (PT a, PT b, PT c, PT d) {
```

```
    b = b - a; d = c - d; c = c - a;
    assert(cmp(cross(b, d)) != 0); // checar sse sao paralelos
    return a + b * cross(c, d) / cross(b, d);
}

// Determina se o ponto p esta dentro do triangulo (a, b, c)
bool ptInsideTriangle(PT p, PT a, PT b, PT c) {
    if(cross(b-a, c-b) < 0) swap(a, b);
    ll x = cross(b-a, p-b);
    ll y = cross(c-b, p-c);
    ll z = cross(a-c, p-a);
    if(x > 0 && y > 0 && z > 0) return true;
    if(!x) return ptInSegment(a,b,p);
    if(!y) return ptInSegment(b,c,p);
    if(!z) return ptInSegment(c,a,p);
    return false;
}

// Determina se o ponto esta num poligono convexo em O(lgn)
bool pointInConvexPolygon(const vector<PT> &p, PT q) {
    PT pivot = p[0];
    int x = 1, y = p.size();
    while(y-x != 1) {
        int z = (x+y)/2;
        PT diagonal = pivot - p[z];
        if(cross(p[x] - pivot, q - pivot) * cross(q-pivot, p[z] -
            pivot) >= 0) y = z;
        else x = z;
    }
    return ptInsideTriangle(q, p[x], p[y], pivot);
}

// Determina se o ponto esta num poligono possivelmente nao-convexo
// Retorna 1 para pontos estritamente dentro, 0 para pontos
    estritamente fora do poligno
// e 0 ou 1 para os pontos restantes
// Eh possivel converter num teste exato usando inteiros e tomando
    cuidado com a divisao
// e entao usar testes exatos para checar se esta na borda do poligno
bool pointInPolygon(const vector<PT> &p, PT q) {
    bool c = 0;
    for(int i = 0; i < p.size(); i++){
        int j = (i + 1) % p.size();
        if((p[i].y <= q.y && q.y < p[j].y || p[j].y <= q.y && q.y < p[
            i].y) &&
            q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].
                y - p[i].y))
            c = !c;
    }
    return c;
}

// Determina se o ponto esta na borda do poligno
bool pointOnPolygon(const vector<PT> &p, PT q) {
    for(int i = 0; i < p.size(); i++)
        if(dist2(projectPointSegment(p[i], p[(i + 1) % p.size()], q),
            q) < eps)
            return true;
    return false;
}

//circunferencia com tres pontos
PT computeCircleCenter (PT a, PT b, PT c) {
    b = (a + b) / 2; // bissector
    c = (a + c) / 2; // bissector
    return computeLineIntersection(b, b + rotateCW90(a - b), c, c +
        rotateCW90(a - c));
}

//circunferencia com dois pontos e o raio, ate duas
vector<PT> circle2PtsRad (PT p1, PT p2, double r) {
    vector<PT> ret;
    double d2 = dist2(p1, p2);
    double det = r * r / d2 - 0.25;
    if (det < 0.0) return ret;
    double h = sqrt(det);
    for (int i = 0; i < 2; i++) {
        double x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * h;
        double y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * h;
        ret.push_back(PT(x, y));
        swap(p1, p2);
    }
    return ret;
}

bool circleLineIntersection(PT a, PT b, PT c, double r) {
    return cmp(dist(c, projectPointLine(a, b, c)), r) <= 0;
}

vector<PT> circleLine (PT a, PT b, PT c, double r) {
    vector<PT> ret;
    PT p = projectPointLine(a, b, c), p1;
    double h = norm(c-p);
    if (cmp(h,r) == 0) {
        ret.push_back(p);
    } else if (cmp(h,r) < 0) {
        double k = sqrt(r*r - h*h);
        p1 = p + (b-a)/(norm(b-a))*k;
        ret.push_back(p1);
        p1 = p - (b-a)/(norm(b-a))*k;
        ret.push_back(p1);
    }
    return ret;
}

vector<PT> circleCircle (PT a, double r, PT b, double R) {
        vector<PT> ret;
        double d = norm(a-b);
        if (d > r + R || d + min(r, R) < max(r, R)) return ret;
        double x = (d*d - R*R + r*r) / (2*d); // x = r*cos(R opposite
            angle)
        double y = sqrt(r*r - x*x);
        PT v = (b - a)/d;
        ret.push_back(a + v*x + rotateCCW90(v)*y);
        if (cmp(y) > 0)
                ret.push_back(a + v*x - rotateCCW90(v)*y);
        return ret;
}

double computeSignedArea (const vector<PT> &p) {
    double area = 0;
    for (int i = 0; i < p.size(); i++) {
```

```
        int j = (i+1) % p.size();
        area += cross(p[i], p[j]);
    }
    return area / 2.0;
}

double computeArea (const vector<PT> &p) {
    return abs(computeSignedArea(p));
}

PT computeCentroid(const vector<PT> &p) {
    PT c(0,0);
    double scale = 6.0 * computeSignedArea(p);
    for(int i = 0; i < p.size(); i++){
        int j = (i + 1) % p.size();
        c = c + (p[i] + p[j]) * (p[i].x * p[j].y - p[j].x * p[i].y);
    }
    return c / scale;
}
```

5.2 Convex Hull

```
vector<PT> convexHull (vector<PT> p) {
    int n = p.size(), k = 0;
    vector<PT> h(2 * n);
    sort(p.begin(), p.end());
    for(int i = 0; i < n; i++) {
        while(k >= 2 && cross(h[k - 1] - h[k - 2], p[i] - h[k - 2]) <=
             0) k--;
        h[k++] = p[i];
    }
    for(int i = n - 2, t = k + 1; i >= 0; i--) {
        while(k >= t && cross(h[k - 1] - h[k - 2], p[i] - h[k - 2]) <=
             0) k--;
        h[k++] = p[i];
    }
    h.resize(k); // n+1 points where the first is equal to the last
    return h;
}
```

5.3 Cut Polygon

```
vector<PT> cutPolygon (vector<PT> Q, PT a, PT b) {
    PT vec = normalize(rotateCW90((b-a)));
    vector<PT> resp;
    for(int i=0; i<Q.size(); i++){
        int j = (i+1)%Q.size();
        double n1 = (Q[i]-a)*(vec);
        double n2 = (Q[j]-a)*(vec);
        if(n1>=-eps) resp.push_back(Q[i]);
        if((n1<-eps && n2>eps) || (n1>eps && n2<-eps)){
            resp.push_back(computeLineIntersection(a, b, Q[i], Q[j]));
        }
    }
    return resp;
}
```

5.4 Smallest Enclosing Circle

```
typedef pair<PT, double> circle;
bool inCircle (circle c, PT p){
    return cmp(dist(c.first, p), c.second) <= 0;
}

PT circumcenter (PT p, PT q, PT r){
    PT a = p-r, b = q-r;
    PT c = PT(dot(a, p+r)/2, dot(b, q+r)/2);
    return PT(cross(c, PT(a.y,b.y)), cross(PT(a.x,b.x), c)) / cross(a,
         b);
}

circle spanningCircle (vector<PT> &v) {
    int n = v.size();
    random_shuffle(v.begin(), v.end());
    circle C(PT(), -1);
    for (int i = 0; i < n; i++) if (!inCircle(C, v[i])) {
        C = circle(v[i], 0);
        for (int j = 0; j < i; j++) if (!inCircle(C, v[j])) {
            C = circle((v[i]+v[j])/2, dist(v[i], v[j])/2);
            for(int k = 0; k < j; k++) if (!inCircle(C, v[k])){
                PT o = circumcenter(v[i], v[j], v[k]);
                C = circle(o,dist(o,v[k]));
            }
        }
    }
    return C;
}
```

5.5 Minkowski

```
bool comp(PT a, PT b){
    int hp1 = (a.x < 0 || (a.x==0 && a.y<0));
    int hp2 = (b.x < 0 || (b.x==0 && b.y<0));
    if(hp1 != hp2) return hp1 < hp2;
    long long R = cross(a, b);
    if(R) return R > 0;
    return dot(a, a) < dot(b, b);
}

vector<PT> minkowskiSum(const vector<PT> &a, const vector<PT> &b){
    if(a.empty() || b.empty()) return vector<PT>(0);
    vector<PT> ret;
    int n1 = a.size(), n2 = b.size();
    if(min(n1, n2) < 2){
        for(int i = 0; i < n1; i++) {
            for(int j = 0; j < n2; j++) {
                ret.push_back(a[i]+b[j]);
            }
        }
        return ret;
    }
    auto insert = [&](PT p) {
        while(ret.size() >= 2 && cmp(cross(p-ret.back(), p-ret[(int)
             ret.size()-2])) == 0) {
        // removing colinear points
```

```
            // needs the scalar product stuff it the result is a line
                ret.pop_back();
            }
            ret.push_back(p);
        };
        PT v1, v2, p = a[0]+b[0];
        ret.push_back(p);
        for (int i = 0, j = 0; i + j + 1 < n1+n2; ){
            v1 = a[(i+1)%n1]-a[i];
            v2 = b[(j+1)%n2]-b[j];
            if(j == n2 || (i < n1 && comp(v1, v2))) p = p + v1, i++;
            else p = p + v2, j++;
            insert(p);
        }
        return ret;
    }
```

5.6 Half Plane Intersection

```
    struct L {
        PT a, b;
        L(){}
        L(PT a, PT b) : a(a), b(b) {}
    };

    double angle (L la) { return atan2(-(la.a.y - la.b.y), la.b.x - la.a.x
        ); }

    bool comp (L la, L lb) {
        if (cmp(angle(la), angle(lb)) == 0) return cross((lb.b - lb.a), (
            la.b - lb.a)) > eps;
        return cmp(angle(la), angle(lb)) < 0;
    }

    PT computeLineIntersection (L la, L lb) {
        return computeLineIntersection(la.a, la.b, lb.a, lb.b);
    }

    bool check (L la, L lb, L lc) {
        PT p = computeLineIntersection(lb, lc);
        double det = cross((la.b - la.a),  (p - la.a));
        return cmp(det) < 0;
    }

    vector<PT> hpi (vector<L> line) { // salvar (i, j) CCW, (j, i) CW
        sort(line.begin(), line.end(), comp);
        vector<L> pl(1, line[0]);
        for (int i = 0; i < (int)line.size(); ++i) if (cmp(angle(line[i]),
             angle(pl.back())) != 0) pl.push_back(line[i]);
        deque<int> dq;
        dq.push_back(0);
        dq.push_back(1);
        for (int i = 2; i < (int)pl.size(); ++i) {
            while ((int)dq.size() > 1 && check(pl[i], pl[dq.back()], pl[dq
                [dq.size() - 2]])) dq.pop_back();
            while ((int)dq.size() > 1 && check(pl[i], pl[dq[0]], pl[dq
                [1]])) dq.pop_front();
            dq.push_back(i);
        }
        while ((int)dq.size() > 1 && check(pl[dq[0]], pl[dq.back()], pl[dq
            [dq.size() - 2]])) dq.pop_back();
        while ((int)dq.size() > 1 && check(pl[dq.back()], pl[dq[0]], pl[dq
            [1]])) dq.pop_front();
        vector<PT> res;
        for (int i = 0; i < (int)dq.size(); ++i){
            res.emplace_back(computeLineIntersection(pl[dq[i]], pl[dq[(i +
                 1) % dq.size()]]));
        }
        return res; // condicao checar res.size() > 2
    }
```

5.7 Closest Pair

```
    double closestPair(vector<PT> p) {
        int n = p.size(), k = 0;
        sort(p.begin(), p.end());
        double d = inf;
        set<PT> ptsInv;
        for(int i = 0; i < n; i++) {
        while(k < i && p[k].x < p[i].x - d) {
            ptsInv.erase(swapCoord(p[k++]));
        }
        for(auto it = ptsInv.lower_bound(PT(p[i].y - d, p[i].x - d));
            it != ptsInv.end() && it->x <= p[i].y + d; it++) {
            d = min(d, dist(p[i] - swapCoord(*it), PT(0, 0)));
        }
        ptsInv.insert(swapCoord(p[i]));
        }
        return d;
    }
```

5.8 Maximum Scalar Point-Poly

```
    //double operator * (const PT p) const { return p.x * q.x + p.y*q.y; }

    int maximizeScalarProduct(const vector<PT> &hull, PT vec) {
        int ans = 0;
        int n = hull.size();
        if(n < 20) {
            for(int i = 0; i < n; i++) {
                if(hull[i] * vec > hull[ans] * vec) {
                    ans = i;
                }
            }
        } else {
            int diff = 1;
            if(hull[0] * vec == hull[1] * vec) {
                int l = 2, r = n - 1;
                while(l != r) {
                    int mid = (l + r) / 2;
                    if((hull[1] - hull[0]) * (hull[mid] - hull[0]) > 0 &&
                        (hull[1] - hull[0]) % (hull[mid] - hull[0]) == 0)
                        {
                        l = mid + 1;
                    } else {
                        r = mid;
                    }
```

```
            }
            diff = 1;
            //diff = 2;
        }
        if(hull[0] * vec < hull[diff] * vec) {
            int l = diff, r = n - 1;
            while(l != r) {
                int mid = (l + r + 1) / 2;
                if(hull[mid] * vec >= hull[mid - 1] * vec && hull[mid]
                     * vec >= hull[0] * vec) {
                    l = mid;
                } else {
                    r = mid - 1;
                }
            }
            if(hull[0] * vec < hull[l] * vec) {
                ans = l;
            }
        } else {
            int l = diff, r = n - 1;
            while(l != r) {
                int mid = (l + r + 1) / 2;
                if(hull[mid] * vec >= hull[mid - 1] * vec || hull[mid
                    - 1] * vec < hull[0] * vec) {
                    l = mid;
                } else {
                    r = mid - 1;
                }
            }
            if(hull[0] * vec < hull[l] * vec) {
                ans = l;
            }
        }
    }
    return ans;
}
```

6 String Algorithms

6.1 KMP

```
int b[ms];

void kmpPreprocess(string p) {
    int m = p.size();
    int i = 0, j = -1;
    b[0] = -1;
    while(i < m) {
        while(j >= 0 && p[i] != p[j]) j = b[j];
        b[++i] = ++j;
    }
}

int kmpSearch(string p, string s) {
    int n = s.size(), m = p.size();
    int i = 0, j = 0, ans = 0;
    while(i < n) {
        while(j >= 0 && s[i] != p[j]) j = b[j];
        i++; j++;
        if(j == m) {
            //ocorrencia aqui comecando em i - j
            ans++;
            j = b[j];
        }
    }
    return ans;
}
```

6.2 KMP Automaton

```
int pre[ms][limit];

void build_automaton(string s){
    int n = (int) s.size();
    for(int i = 0; i < limit; ++i){
        pre[0][i] = 0;
    }
    pre[0][s[0] - 'A'] = 1;
    int fail = 0;
    for(int i = 1; i <= n; ++i){
        for(int j = 0; j < limit; ++j){
            pre[i][j] = pre[fail][j];
        }
        if(i == n) continue;
        pre[i][s[i] - 'A'] = i + 1;
        fail = pre[fail][s[i] - 'A'];
    }
}
```

6.3 Trie

```
int trie[ms][sigma], terminal[ms], z = 1;

int get_id(char c) {
    return c - 'a';
}

void insert(string &p) {
    int cur = 0;
    for(int i = 0; i < p.size(); i++) {
        int id = get_id(p[i]);
        if(trie[cur][id] == 0) {
            trie[cur][id] = z++;
        }
        cur = trie[cur][id];
    }
    terminal[cur]++;
}

int count(string &p) {
    int cur = 0;
    for(int i = 0; i < p.size(); i++) {
        int id = get_id(p[i]);
        if(trie[cur][id] == 0) {
            return false;
        }
```

```
        cur = trie[cur][id];
    }
    return terminal[cur];
}
```

6.4 Aho-Corasick

```
template<const int ALPHA = 26, class T = string, const int off = 'a'>
struct Aho {
    struct Node {
        int to[ALPHA];
        int size;
        int fail, pfail;
        bool present;

        Node() {
            for(int i = 0; i < ALPHA; i++) {
                to[i] = 0;
            }
            size = 0;
            pfail = fail = 0;
            present = false;
            // maybe initialize some other stuff here
        }

        // maybe add some other stuff here
    };

    Aho() {
        nodes.push_back(Node());
    }

    int addString(const T &str) {
        int on = 0;
        for(auto ch : str) {
            if(nodes[on].to[ch-off] == 0) {
                nodes[on].to[ch-off] = (int) nodes.size();
                nodes.push_back(Node());
                nodes.back().size = 1 + nodes[on].size;
            }
            on = nodes[on].to[ch-off];
        }
        // makes this node present
        nodes[on].present = true;
        return on;
    }

    void build() {
        queue<int> que;
        que.push(0);
        while(!que.empty()) {
            int on = que.front();
            que.pop();
            nodes[on].pfail = nodes[nodes[on].fail].present ? nodes[on
                ].fail : nodes[nodes[on].fail].pfail;
            // do stuff that carries over with fail here
            for(int i = 0; i < ALPHA; i++) {
                int &to = nodes[on].to[i];
                if(to) {
                    nodes[to].fail = on == 0 ? 0 : nodes[nodes[on].
                        fail].to[i];
                    que.push(to);
                } else {
                    to = nodes[nodes[on].fail].to[i];
                }
            }
        }
    }
    vector<Node> nodes;
};
```

6.5 Algoritmo de Z

```
// credits to FMota/UFCG
template <class T>
struct ZFunc {
    vector<int> z; // z[i] = match a partir de i com a posicao 0
    ZFunc(const vector<T> &v) : z(v.size()) {
        int n = (int) v.size(), a = 0, b = 0;
        if (!z.empty()) z[0] = n;
        for (int i = 1; i < n; i++) {
            int end = i; if (i < b) end = min(i + z[i - a], b);
            while(end < n && v[end] == v[end - i]) ++end;
            z[i] = end - i; if(end > b) a = i, b = end;
        }
    }
};
```

6.6 Suffix Array

```
typedef pair<int, int> ii;

vector<int> buildSA(string s) {
    int n = (int) s.size();
    vector<int> ids(n), pos(n);
    vector<ii> pairs(n);
    for(int i = 0; i < n; i++) {
        ids[i] = i;
        pairs[i] = ii(s[i], -1);
    }
    sort(ids.begin(), ids.end(), [&](int a, int b) -> bool {
        return pairs[a] < pairs[b];
    });
    int on = 0;
    for(int i = 0; i < n; i++) {
        if (i && pairs[ids[i - 1]] != pairs[ids[i]]) on++;
        pos[ids[i]] = on;
    }
    for(int offset = 1; offset < n; offset <<= 1) {
        for(int i = 0; i < n; i++) {
            pairs[i].first = pos[i];
            if (i + offset < n) {
                pairs[i].second = pos[i + offset];
            } else {
                pairs[i].second = -1;
            }
        }
```

```
        sort(ids.begin(), ids.end(), [&](int a, int b) -> bool {
            return pairs[a] < pairs[b];
        });
        int on = 0;
        for(int i = 0; i < n; i++) {
            if (i && pairs[ids[i - 1]] != pairs[ids[i]]) on++;
            pos[ids[i]] = on;
        }
    }
    return ids;
}

vector<int> buildLCP(string s, vector<int> sa) {
    int n = (int) s.size();
    vector<int> pos(n), lcp(n, 0);
    for(int i = 0; i < n; i++) {
        pos[sa[i]] = i;
    }
    int k = 0;
    for(int i = 0; i < n; i++) {
        if (pos[i] + 1 == n) {
            k = 0;
            continue;
        }
        int j = sa[pos[i] + 1];
        while(i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
        lcp[pos[i]] = k;
        k = max(k - 1, 0);
    }
    return lcp;
}

int distinctSubstrings(string s) {
    vector<int> sa = buildSA(s);
    vector<int> pref = buildLCP(s, sa);
    int n = s.size();
    int ans = n - sa[0];
    for (int i = 1; i < s.size(); i++) {
        ans += (n - sa[i]) - pref[i - 1];
    }
    return ans;
}

void kthLexicographicalSubstring() {
    string s;
    cin >> s;
    int n = s.size();
    vector<int> sa = buildSA(s);
    vector<int> pref = buildLCP(s, sa);
    vector<int> prefAcum(n);
    prefAcum[0] = n - sa[0];
    for (int i = 1; i < n; i++) {
        prefAcum[i] = prefAcum[i - 1] + ((n - sa[i]) - pref[i - 1]);
    }
    int m;
    cin >> m;
    for (int i = 0; i < m; i++) {
        int k;
        cin >> k;
        int pos = lower_bound(prefAcum.begin(), prefAcum.end(), k) -
            prefAcum.begin();
        string ans;
        ans = s.substr(sa[pos], pref[pos - 1] + (k - prefAcum[pos -
            1]));
        cout << ans << '\n';
    }
}
```

7 Miscellaneous

7.1 Ternary Search

```
// R
for(int i = 0; i < LOG; i++){
    long double m1 = (A * 2 + B) / 3.0;
    long double m2 = (A + 2 * B) / 3.0;

    if(f(m1) > f(m2))
        A = m1;
    else
        B = m2;
}
ans = f(A);

// Z
while(B - A > 4){
    int m1 = (A + B) / 2;
    int m2 = (A + B) / 2 + 1;
    if(f(m1) > f(m2))
        A = m1;
    else
        B = m2;
}
ans = inf;
for(int i = A; i <= B; i++) ans = min(ans , f(i));
```

7.2 Random Number Generator

```
// mt19937_64 se LL
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
// Random_Shuffle
shuffle(v.begin(), v.end(), rng);
// Random number in interval
int randomInt = uniform_int_distribution(0, i)(rng);
double randomDouble = uniform_real_distribution(0, 1)(rng);
// bernoulli_distribution, binomial_distribution,
    geometric_distribution
// normal_distribution, poisson_distribution, exponential_distribution
```

7.3 Dates

```
string dayOfWeek[] = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"
    };

// converts Gregorian date to integer (Julian day number)
int dateToInt (int m, int d, int y){
```

```
    return
    1461 * (y + 4800 + (m - 14) / 12) / 4 +
    367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
    3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
    d - 32075;
}

// converts integer (Julian day number) to Gregorian date: month/day/
    year
void intToDate (int jd, int &m, int &d, int &y){
    int x, n, i, j;

    x = jd + 68569;
    n = 4 * x / 146097;
    x -= (146097 * n + 3) / 4;
    i = (4000 * (x + 1)) / 1461001;
    x -= 1461 * i / 4 - 31;
    j = 80 * x / 2447;
    d = x - 2447 * j / 80;
    x = j / 11;
    m = j + 2 - 12 * x;
    y = 100 * (n - 49) + i + x;
}

// converts integer (Julian day number) to day of week
string intToDay (int jd){
    return dayOfWeek[jd % 7];
}
```

7.4 Max Histogram

```
int getMaxArea(int hist[], int n) {
  stack<int> s;
  int max_area = 0;
  int tp;
  int area_with_top;

  int i = 0;
  while (i < n) {
    if (s.empty() || hist[s.top()] <= hist[i]) {
      s.push(i++);
    } else {
      tp = s.top();
      s.pop();
      area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);
      if (max_area < area_with_top) {
        max_area = area_with_top;
      }
    }
  }
  while (s.empty() == false) {
    tp = s.top();
    s.pop();
    area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);
    if (max_area < area_with_top) {
      max_area = area_with_top;
    }
  }
  return max_area;
}

int main() {
  int hist[] = {6, 2, 5, 4, 5, 1, 6};
  int n = sizeof(hist)/sizeof(hist[0]);
  cout << "Maximum area is " << getMaxArea(hist, n);
  return 0;
}
```

7.5 Mo Algorithm

```
const int blk_sz = 170;

struct Query {
  int l, r, idx;
  bool operator < (Query a) {
    if (l / blk_sz == a.l / blk_sz) {
      return r < a.r;
    }
    return (l / blk_sz) < (a.l / blk_sz);
  }
};

vector<Query> queries;
int a[MAXN], ans[MAXN], qnt[1000010];
int diff = 0;

void add(int x) {
  x = a[x];
  if (qnt[x] == 0) {
    diff++;
  }
  qnt[x]++;
}

void remove(int x) {
  x = a[x];
  qnt[x]--;
  if (qnt[x] == 0) {
    diff--;
  }
}

void mos() {
  int curr_l = 0, curr_r = -1;
  sort(queries.begin(), queries.end());
  for (Query q : queries) {
    while (curr_l > q.l) {
      curr_l--;
      add(curr_l);
    }
    while (curr_r < q.r) {
      curr_r++;
      add(curr_r);
    }
    while (curr_l < q.l) {
      remove(curr_l);
      curr_l++;
    }
    while (curr_r > q.r) {
      remove(curr_r);
```

```
      curr_r--;
    }
    ans[q.idx] = diff;
  }
}
```

7.6 Angular Sweep

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

point pt[MAXN];
int n;

double dist(point a, point b) {
  return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}

int solve() {
  vector<pair<double, int>> events;
  int ans = 1;
  for (int i = 0; i < n; i++) {
    events.clear();
    for (int j = 0; j < n; j++) {
      if (i == j || dist(pt[i], pt[j]) > 2.0 * R) continue;
      double A = atan2(pt[j].y - pt[i].y, pt[j].x - pt[i].x);
      if (A < 0) A += 2*PI;
      double B = acos(dist(pt[i], pt[j]) / (2.0 * R));
      if (A - B >= 0) {
        events.push_back({A - B, -1});
      } else {
        events.push_back({0.0, -1});
        events.push_back({+2*PI + (A - B), +1});
      }
      if (A + B <= 2*PI) {
        events.push_back({A + B, +1});
      } else {
        events.push_back({0.0, -1});
        events.push_back({-2*PI + (A + B), +1});
      }
    }
    sort(events.begin(), events.end());
    int cnt = 1;
    for (int i = 0; i < events.size(); i++) {
      if (events[i].second < 0) {
        cnt++;
      } else {
        cnt--;
      }
      ans = max(ans, cnt);
    }
  }
  return ans;
}
```

8 Teoremas e formulas uteis

8.1 Grafos

```
Formula de Euler: V - E + F = 2 (para grafo planar)
Handshaking: Numero par de vertices tem grau impar
Kirchhoff's Theorem: Monta matriz onde Mi,i = Grau[i] e Mi,j = -1 se
    houver aresta i-j ou 0 caso contrario, remove uma linha e uma
    coluna qualquer e o numero de spanning trees nesse grafo eh o det
    da matriz

Grafo contem caminho hamiltoniano se:
Dirac's theorem: Se o grau de cada vertice for pelo menos n/2
Ore's theorem: Se a soma dos graus que cada par nao-adjacente de
    vertices for pelo menos n

Grafo (bidirecional) cont m circuito euleriano se todos v rtices tem
     grau par
Grafo (bidirecional) cont m caminho euleriano se tem no m ximo dois
    v rtices de grau  mpar

Grafo (direcional) cont m circuito euleriano se in = out para todo
    v rtice
Grafo (direcional) cont m caminho euleriano se for circuito euleriano
     OU (possui UM v rtice com in - out = 1 e UM v rtice com in -
    out = -1)
OBS: Checar se fazem parte do mesmo grupo

Trees:
Tem Catalan(N) Binary trees de N vertices
Tem Catalan(N-1) Arvores enraizadas com N vertices
Caley Formula: n^(n-2) arvores em N vertices com label
Prufer code: Cada etapa voce remove a folha com menor label e o label
    do vizinho eh adicionado ao codigo ate ter 2 vertices
Prufer theorem: Toda  rvore pode ser representada por um vetor de N-2
     posi es

Flow:
Max Edge-disjoint paths: Max flow com arestas com peso 1
Max Node-disjoint paths: Faz a mesma coisa mas separa cada vertice em
    um com as arestas de chegadas e um com as arestas de saida e uma
    aresta de peso 1 conectando o vertice com aresta de chegada com
    ele mesmo com arestas de saida
Konig's Theorem: minimum node cover = maximum matching se o grafo for
    bipartido, complemento eh o maximum independent set
Min Node disjoint path cover: formar grafo bipartido de vertices
    duplicados, onde aresta sai do vertice tipo A e chega em tipo B,
    entao o path cover eh N - matching
Min General path cover: Mesma coisa mas colocando arestas de A pra B
    sempre que houver caminho de A pra B
Dilworth's Theorem: Min General Path cover = Max Antichain (set de
    vertices tal que nao existe caminho no grafo entre vertices desse
    set)
Hall's marriage: um grafo tem um matching completo do lado X se para
    cada subconjunto W de X,
    |W| <= |vizinhosW| onde |W| eh quantos vertices tem em W
```

8.2 Math

```
Goldbach's: todo numero par n > 2 pode ser representado com n = a + b
    onde a e b sao primos
Twin prime: existem infinitos pares p, p + 2 onde ambos sao primos
Legendre's: sempre tem um primo entre n^2 e (n+1)^2
Lagrange's: todo numero inteiro pode ser inscrito como a soma de 4
    quadrados
Zeckendorf's: todo numero pode ser representado pela soma de dois
    numeros de fibonnacis diferentes e nao consecutivos
Euclid's: toda tripla de pitagoras primitiva pode ser gerada com
    (n^2 - m^2, 2nm, n^2+m^2) onde n, m sao coprimos e um deles eh par
Wilson's: n eh primo quando (n-1)! mod n = n - 1
Mcnugget: Para dois coprimos x, y o maior inteiro que nao pode ser
    escrito como ax + by eh (x-1)(y-1)/2

Fermat: Se p eh primo entao a^(p-1) % p = 1
Se x e m tambem forem coprimos entao x^k % m = x^(k mod(m-1)) % m
Euler's theorem: x^(phi(m)) mod m = 1 onde phi(m) eh o totiente de
    euler

Chinese remainder theorem:
Para equacoes no formato x = a1 mod m1, ... , x = an mod mn onde todos
     os pares m1, ... , mn sao coprimos
Deixe Xk = m1*m2*..*mn/mk e Xk^-1 mod mk = inverso de Xk mod mk, entao
x = somatorio com k de 1 ate n de ak*Xk*(Xk,mk^-1 mod mk)
Para achar outra solucao so somar m1*m2*..*mn a solucao existente

Catalan number: exemplo expressoes de parenteses bem formadas
C0 = 1, Cn = somatorio de i=0 -> n-1 de Ci*C(n-1+1)
outra forma: Cn = (2n escolhe n)/(n+1)
Bertrand's ballot theorem: p votos tipo A e q votos tipo B com p>q,
    prob de em todo ponto ter mais As do que Bs antes dele = (p-q)/(p+
    q)
Se puder empates entao prob = (p+1-q)/(p+1), para achar quantidade de
    possibilidades nos dois casos basta multiplicar por (p + q escolhe
     q)

Propriedades de Coeficientes Binomiais:
Somatorio de k = 0 -> m de (-1)^k * (n escolhe k) = (-1)^m* (n -1
    escolhe m)
(N escolhe K) = (N escolhe N-K)
(N escolhe K) = N/K * (n-1 escolhe k-1)
Somatorio de k = 0 -> n de (n escolhe k) = 2^n
Somatorio de m = 0 -> n de (m escolhe k) = (n+1 escolhe k + 1)
Somatorio de k = 0 -> m de (n+k escolhe k) = (n+m+1 escolhe m)
Somatorio de k = 0 -> n de (n escolhe k)^2 = (2n escolhe n)
Somatorio de k = 0 ou 1 -> n de k*(n escolhe k) = n * 2^(n-1)
Somatorio de k = 0 -> n de (n-k escolhe k) = Fib(n+1)

Hockey-stick: Somatorio de i = r -> n de (i escolhe r) = (n + 1
    escolhe r + 1)
Vandermonde: (m+n escolhe r) = somatorio de k = 0 -> r de (m escolhe k
    ) * (n escolhe r - k)

Burnside lemma: colares diferentes nao contando rotacoes quando m =
    cores e n = comprimento
(m^n + somatorio i =1 - > n-1 de m^gcd(i, n))/n

Distribuicao uniforme a,a+1, ..., b Expected[X] = (a+b)/2
Distribuicao binomial com n tentativas de probabilidade p, X =
    sucessos:
    P(X = x) = p^x * (1-p)^(n-x) * (n escolhe x) e E[X] = p*n
Distribuicao geometrica onde continuamos ate ter sucesso, X =
    tentativas:
    P(X = x) = (1-p)^(x-1) * p e E[X] = 1/p
Linearity of expectation: Tendo duas variaveis X e Y e constantes a e
    b, o valor esperado de aX + bY = a*E[X] + b*E[X]

Primos de Mersenne 2^n - 1
Lista de Ns que resultam nos primeiros 41 primos de Mersenne:
2; 3; 5; 7; 13; 17; 19; 31; 61; 89; 107; 127; 521; 607; 1.279; 2.203;
    2.281; 3.217; 4.253; 4.423; 9.689; 9.941; 11.213; 19.937; 21.701;
    23.209; 44.497; 86.243; 110.503; 132.049; 216.091; 756.839;
    859.433; 1.257.787; 1.398.269; 2.976.221; 3.021.377; 6.972.593;
    13.466.917; 20.996.011; 24.036.583;
```

8.3 Geometry

```
Formula de Euler: V - E + F = 2
Pick Theorem: Para achar pontos em coords inteiras num poligono Area =
     i + b/2 - 1 onde i eh o o numero de pontos dentro do poligono e b
     de pontos no perimetro do poligono
Two ears theorem: Todo poligono simples com mais de 3 vertices tem
    pelo menos 2 orelhas, vertices que podem ser removidos sem criar
    um crossing, remover orelhas repetidamente triangula o poligono
Incentro triangulo:  (a(Xa, Ya) + b(Xb, Yb) + c(Xc, Yc))/(a+b+c) onde
    a = lado oposto ao vertice a, incentro eh onde cruzam as
    bissetrizes, eh o centro da circunferencia inscrita e eh
    equidistante aos lados

Delaunay Triangulation: Triangulacao onde nenhum ponto esta dentro de
    nenhum circulo circunscrito nos triangulos
Eh uma triangulacao que maximiza o menor angulo e a MST euclidiana de
    um conjunto de pontos eh um subconjunto da triangulacao

Brahmagupta s formula: Area cyclic quadrilateral (maximum)
s = (a+b+c+d)/2
area = sqrt((s-a)*(s-b)*(s-c)*(s-d))
d = 0 => (Heron) area = sqrt((s-a)*(s-b)*(s-c)*s)
```