

## GEMP - UFC Quixadá - ICPC Library

## Contents

<b>1</b>	<b>Data Structures</b>	<b>1</b>
1.1	BIT	1
1.2	BIT 2D	2
1.3	BIT In Range	2
1.4	Dynamic Median	2
1.5	Dynamic Wavelet Tree	2
1.6	Implicit Treap	4
1.7	LiChao Tree	5
1.8	Policy Based Tree	6
1.9	Queue Query	6
1.10	Range Color	6
1.11	Segment Tree	7
1.12	Segment Tree 2D	8
1.13	Segment Tree Iterative	8
1.14	Segment Tree Lazy	8
1.15	Sparse Table	9
1.16	SQRT Decomposition	9
1.17	SQRT Tree	10
1.18	Stack Query	11
1.19	Treap	11
1.20	Union Find	12
1.21	Wavelet Tree	12
<b>2</b>	<b>Graph Algorithms</b>	<b>13</b>
2.1	2-SAT	13
2.2	Centroid Decomposition	14
2.3	Dinic	14
2.4	Flow With Demand	15
2.5	Kruskal	15
2.6	HLD	16
2.7	Minimum Cost Maximum Flow	16
2.8	Strongly Connected Component	17
2.9	Topological Sort	18
<b>3</b>	<b>Dynamic Programming</b>	<b>18</b>
3.1	Divide and Conquer Optimization	18
3.2	Divide and Conquer Optimization Implementation	18
3.3	Knuth Optimization	19
3.4	Knuth Optimization Implementation	19
<b>4</b>	<b>Math</b>	<b>19</b>
4.1	Basic Math	19
4.2	BigInt	19
4.3	Binomial Coefficients	21
4.4	Chinese Remainder Theorem	22
4.5	Euler's totient	22
4.6	Extended Euclidean	23
4.7	Fraction	23
4.8	Gray Code	24
4.9	Matrix	24
4.10	Modular Arithmetic	24
4.11	Montgomery Multiplication	24
4.12	Prime Number	25
<b>5</b>	<b>Geometry</b>	<b>25</b>
5.1	Basic Geometry	25
5.2	Circle Area Union	29
5.3	Count Lattices	30
5.4	Point 3D	30
<b>6</b>	<b>String Algorithms</b>	<b>31</b>
6.1	Min Cyclic String	31
6.2	Suffix Automaton	31

<b>7</b>	<b>Miscellaneous</b>	<b>32</b>
7.1	Longest Increasing Subsequence	32
7.2	Mo Algorithm	32
<b>8</b>	<b>Theorems and Formulas</b>	<b>32</b>
8.1	Binomial Coefficients	32
8.2	Catalan Number	33
8.3	Euler's Totient	33
8.4	Formulas	33
8.5	Manhattan Distance	33
8.6	Primes	34

## 1 Data Structures

## 1.1 BIT

```

#include <bits/stdc++.h>
using namespace std;
class Bit{
private:
    typedef long long t_bit;
    int nBit;
    int nLog;
    vector<t_bit> bit;
public:
    Bit(int n){
        nBit = n;
        nLog = 20;
        bit.resize(nBit + 1, 0);
    }
    //1-indexed
    t_bit get(int i){
        t_bit s = 0;
        for (; i > 0; i -= (i & -i))
            s += bit[i];
        return s;
    }
    //1-indexed [l, r]
    t_bit get(int l, int r){
        return get(r) - get(l - 1);
    }
    //1-indexed
    void add(int i, t_bit value){
        assert(i > 0);
        for (; i <= nBit; i += (i & -i))
            bit[i] += value;
    }
    t_bit lower_bound(t_bit value){
        t_bit sum = 0;
        int pos = 0;
        for (int i = nLog; i >= 0; i--){
            if ((pos + (1 << i) <= nBit) and (sum + bit[pos + (1 << i)] <
                value)){
                sum += bit[pos + (1 << i)];
                pos += (1 << i);
            }
        }
        return pos + 1;
    }
};

```

## 1.2 BIT 2D

```
#include <bits/stdc++.h>
using namespace std;
class Bit2d{
private:
    typedef long long t_bit;
    vector<vector<t_bit>> bit;
    int nBit, mBit;
public:
    Bit2d(int n, int m){
        nBit = n;
        mBit = m;
        bit.resize(nBit + 1, vector<t_bit>(mBit + 1, 0));
    }
    //1-indexed
    t_bit get(int i, int j){
        t_bit sum = 0;
        for (int a = i; a > 0; a -= (a & -a))
            for (int b = j; b > 0; b -= (b & -b))
                sum += bit[a][b];
        return sum;
    }
    //1-indexed
    t_bit get(int a1, int b1, int a2, int b2){
        return get(a2, b2) - get(a2, b1 - 1) - get(a1 - 1, b2) + get(a1 - 1, b1 - 1);
    }
    //1-indexed [i, j]
    void add(int i, int j, t_bit value){
        for (int a = i; a <= nBit; a += (a & -a))
            for (int b = j; b <= mBit; b += (b & -b))
                bit[a][b] += value;
    }
};
```

## 1.3 BIT In Range

```
#include <bits/stdc++.h>
using namespace std;
class BitRange{
private:
    typedef long long t_bit;
    vector<t_bit> bit1, bit2;
    t_bit get(vector<t_bit> &bit, int i){
        t_bit sum = 0;
        for (; i > 0; i -= (i & -i))
            sum += bit[i];
        return sum;
    }
    void add(vector<t_bit> &bit, int i, t_bit value){
        for (; i < (int)bit.size(); i += (i & -i))
            bit[i] += value;
    }
public:
    BitRange(int n){
        bit1.assign(n + 1, 0);
        bit2.assign(n + 1, 0);
    }
};
```

```

    }
    //1-indexed [i, j]
    void add(int i, int j, t_bit v){
        add(bit1, i, v);
        add(bit1, j + 1, -v);
        add(bit2, i, v * (i - 1));
        add(bit2, j + 1, -v * j);
    }
    //1-indexed
    t_bit get(int i){
        return get(bit1, i) * i - get(bit2, i);
    }
    //1-indexed [i, j]
    t_bit get(int i, int j){
        return get(j) - get(i - 1);
    }
};
```

## 1.4 Dynamic Median

```
#include <bits/stdc++.h>
using namespace std;
class DinamicMedian{
    typedef int t_median;
private:
    priority_queue<t_median> mn;
    priority_queue<t_median, vector<t_median>, greater<t_median>> mx;
public:
    double median(){
        if (mn.size() > mx.size())
            return mn.top();
        else
            return (mn.top() + mx.top()) / 2.0;
    }
    void push(t_median x){
        if (mn.size() <= mx.size())
            mn.push(x);
        else
            mx.push(x);
        if ((!mx.empty()) and (!mn.empty())){
            while (mn.top() > mx.top()){
                t_median a = mx.top();
                mx.pop();
                t_median b = mn.top();
                mn.pop();
                mx.push(b);
                mn.push(a);
            }
        }
    }
};
```

## 1.5 Dynamic Wavelet Tree

```
#include <bits/stdc++.h>
using namespace std;
struct SplayTree{
    struct Node{
```

```

int x, y, s;
Node *p = 0;
Node *l = 0;
Node *r = 0;
Node(int v){
    x = v;
    y = v;
    s = 1;
}
void upd(){
    s = 1;
    y = x;
    if (l){
        y += l->y;
        s += l->s;
    }
    if (r){
        y += r->y;
        s += r->s;
    }
}
int left_size(){
    return l ? l->s : 0;
}
};
Node *root = 0;
void rot(Node *c){
    auto p = c->p;
    auto g = p->p;
    if (g)
        (g->l == p ? g->l : g->r) = c;
    if (p->l == c){
        p->l = c->r;
        c->r = p;
        if (p->l)
            p->l->p = p;
    }
    else{
        p->r = c->l;
        c->l = p;
        if (p->r)
            p->r->p = p;
    }
    p->p = c;
    c->p = g;
    p->upd();
    c->upd();
}
void splay(Node *c){
    while (c->p){
        auto p = c->p;
        auto g = p->p;
        if (g)
            rot((g->l == p) == (p->l == c) ? p : c);
        rot(c);
    }
    c->upd();
    root = c;
}
Node *join(Node *l, Node *r){
    if (not l)
        return r;
    if (not r)
        return l;
    while (l->r){
        l = l->r;
        splay(l);
        r->p = l;
        l->r = r;
        l->upd();
        return l;
    }
}
pair<Node *, Node *> split(Node *p, int idx){
    if (not p)
        return make_pair(nullptr, nullptr);
    if (idx < 0)
        return make_pair(nullptr, p);
    if (idx >= p->s)
        return make_pair(p, nullptr);
    for (int lf = p->left_size(); idx != lf; lf = p->left_size()){
        if (idx < lf)
            p = p->l;
        else
            p = p->r, idx -= lf + 1;
    }
    splay(p);
    Node *l = p;
    Node *r = p->r;
    if (r){
        l->r = r->p = 0;
        l->upd();
    }
    return make_pair(l, r);
}
Node *get(int idx){
    auto p = root;
    for (int lf = p->left_size(); idx != lf; lf = p->left_size()){
        if (idx < lf)
            p = p->l;
        else
            p = p->r, idx -= lf + 1;
    }
    splay(p);
    return p;
}
int insert(int idx, int x){
    Node *l, *r;
    tie(l, r) = split(root, idx - 1);
    int v = l ? l->y : 0;
    root = join(l, join(new Node(x), r));
    return v;
}
void erase(int idx){
    Node *l, *r;
    tie(l, r) = split(root, idx);
    root = join(l->l, r);
    delete l;
}
int rank(int idx){
    Node *l, *r;
    tie(l, r) = split(root, idx);
    int x = (l && l->l ? l->l->y : 0);

```

```

    root = join(l, r);
    return x;
}
int operator[](int idx){
    return rank(idx);
}
~SplayTree(){
    if (!root)
        return;
    vector<Node *> nodes{root};
    while (nodes.size()){
        auto u = nodes.back();
        nodes.pop_back();
        if (u->l)
            nodes.emplace_back(u->l);
        if (u->r)
            nodes.emplace_back(u->r);
        delete u;
    }
};
class WaveletTree{
private:
    int lo, hi;
    WaveletTree *l = 0;
    WaveletTree *r = 0;
    SplayTree b;
public:
    WaveletTree(int min_value, int max_value){
        lo = min_value;
        hi = max_value;
        b.insert(0, 0);
    }
    ~WaveletTree(){
        delete l;
        delete r;
    }
    //0-indexed
    void insert(int idx, int x){
        if (lo >= hi)
            return;
        int mid = (lo + hi - 1) / 2;
        if (x <= mid){
            l = l ? new WaveletTree(lo, mid);
            l->insert(b.insert(idx, 1), x);
        }else{
            r = r ? new WaveletTree(mid + 1, hi);
            r->insert(idx - b.insert(idx, 0), x);
        }
    }
    //0-indexed
    void erase(int idx){
        if (lo == hi)
            return;
        auto p = b.get(idx);
        int lf = p->l ? p->l->y : 0;
        int x = p->x;
        b.erase(idx);
        if (x == 1)
            l->erase(lf);
        else

```

```

        r->erase(idx - lf);
    }
    //kth smallest element in range [i, j]
    //0-indexed
    int kth(int i, int j, int k){
        if (i >= j)
            return 0;
        if (lo == hi)
            return lo;
        int x = b.rank(i);
        int y = b.rank(j);
        if (k <= y - x)
            return l->kth(x, y, k);
        else
            return r->kth(i - x, j - y, k - (y - x));
    }
    //Amount of numbers in the range [i, j] Less than or equal to k
    //0-indexed
    int lte(int i, int j, int k){
        if (i >= j or k < lo)
            return 0;
        if (hi <= k)
            return j - i;
        int x = b.rank(i);
        int y = b.rank(j);
        return l->lte(x, y, k) + r->lte(i - x, j - y, k);
    }
    //Amount of numbers in the range [i, j] equal to k
    //0-indexed
    int count(int i, int j, int k){
        if (i >= j or k < lo or k > hi)
            return 0;
        if (lo == hi)
            return j - i;
        int mid = (lo + hi - 1) / 2;
        int x = b.rank(i);
        int y = b.rank(j);
        if (k <= mid)
            return l->count(x, y, k);
        return r->count(i - x, j - y, k);
    }
    //0-indexed
    int get(int idx){
        return kth(idx, idx + 1, 1);
    }
};

```

## 1.6 Implicit Treap

```

#include <bits/stdc++.h>
using namespace std;
namespace ITreap{
    const int N = 500010;
    typedef long long treap_t;
    treap_t X[N];
    int en = 1, Y[N], sz[N], L[N], R[N], root;

    const treap_t neutral = 0;
    treap_t op_val[N];
    bool rev[N];

```

```

inline treap_t join(treap_t a, treap_t b, treap_t c){
    return a + b + c;
}
void calc(int u) { // update node given children info
    sz[u] = sz[L[u]] + 1 + sz[R[u]];
    // code here, no recursion
    op_val[u] = join(op_val[L[u]], X[u], op_val[R[u]]);
}
void unlaze(int u) {
    if(!u) return;
    // code here, no recursion
    if (rev[u]){
        if(L[u])
            rev[L[u]] ^= rev[u];
        if(R[u])
            rev[R[u]] ^= rev[u];
        swap(L[u], R[u]);
        rev[u] = false;
    }
}
void split(int u, int s, int &l, int &r) { // l gets first s, r gets remaining
    unlaze(u);
    if(!u) return (void) (l = r = 0);
    if(sz[L[u]] < s) { split(R[u], s - sz[L[u]] - 1, l, r); R[u] = l;
        l = u; }
    else { split(L[u], s, l, r); L[u] = r; r = u; }
    calc(u);
}
int merge(int l, int r) { // els on l <= els on r
    unlaze(l); unlaze(r);
    if(!l || !r) return l + r;
    int u;
    if(Y[l] > Y[r]) { R[l] = merge(R[l], r); u = l; }
    else { L[r] = merge(l, L[r]); u = r; }
    calc(u);
    return u;
}
int new_node(treap_t x){
    X[en] = x;
    op_val[en] = x;
    rev[en] = false;
    return en++;
}
int nth(int u, int idx){
    if(!u)
        return 0;
    unlaze(u);
    if(idx <= sz[L[u]])
        return nth(L[u], idx);
    else if(idx == sz[L[u]] + 1)
        return u;
    else
        return nth(R[u], idx - sz[L[u]] - 1);
}
//Public
void init(int n=N-1) { // call before using other funcs
    //init position 0
    sz[0] = 0;
    op_val[0] = neutral;
    //init Treap

```

```

    root = 0;
    for(int i = en = 1; i <= n; i++) { Y[i] = i; sz[i] = 1; L[i] = R[i]
        = 0; }
    random_shuffle(Y + 1, Y + n + 1);
}
//0-indexed
void insert(int idx, int val){
    int a, b;
    split(root, idx, a, b);
    root = merge(merge(a, new_node(val)), b);
}
//0-indexed
void erase(int idx){
    int a, b, c, d;
    split(root, idx, a, b);
    split(b, 1, c, d);
    root = merge(a, d);
}
//0-indexed
treap_t nth(int idx){
    int u = nth(root, idx+1);
    return X[u];
}
//0-indexed [l, r]
treap_t query(int l, int r){
    if(l > r) swap(l, r);
    int a, b, c, d;
    split(root, l, a, d);
    split(d, r - l + 1, b, c);
    treap_t ans = op_val[b];
    root = merge(a, merge(b, c));
    return ans;
}
//0-indexed [l, r]
void reverse(int l, int r){
    if (l > r) swap(l, r);
    int a, b, c, d;
    split(root, l, a, d);
    split(d, r - l + 1, b, c);
    if(b)
        rev[b] ^= 1;
    root = merge(a, merge(b, c));
}
};

```

## 1.7 LiChao Tree

```

#include <bits/stdc++.h>
using namespace std;
const int INF = 0x3f3f3f3f;
class LiChaoTree{
private:
    typedef int t_line;
    struct Line{
        t_line k, b;
        Line() {}
        Line(t_line k, t_line b) : k(k), b(b) {}
    };
    int n_tree, min_x, max_x;
    vector<Line> li_tree;

```

```

t_line f(Line l, int x){
    return l.k * x + l.b;
}
void add(Line nw, int v, int l, int r){
    int m = (l + r) / 2;
    bool lef = f(nw, l) > f(li_tree[v], l);
    bool mid = f(nw, m) > f(li_tree[v], m);
    if (mid)
        swap(li_tree[v], nw);
    if (r - l == 1)
        return;
    else if (lef != mid)
        add(nw, 2 * v, l, m);
    else
        add(nw, 2 * v + 1, m, r);
}
int get(int x, int v, int l, int r){
    int m = (l + r) / 2;
    if (r - l == 1)
        return f(li_tree[v], x);
    else if (x < m)
        return max(f(li_tree[v], x), get(x, 2 * v, l, m));
    else
        return max(f(li_tree[v], x), get(x, 2 * v + 1, m, r));
}
public:
LiChaoTree(int mn_x, int mx_x){
    min_x = mn_x;
    max_x = mx_x;
    n_tree = max_x - min_x + 5;
    li_tree.resize(4 * n_tree, Line(0, -INF));
}
void add(t_line k, t_line b){
    add(Line(k, b), 1, min_x, max_x);
}
t_line get(int x){
    return get(x, 1, min_x, max_x);
}
};

```

## 1.8 Policy Based Tree

```

#include <bits/stdc++.h>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using namespace std;
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> OrderedSet;
typedef tree<int, int, less<int>, rb_tree_tag,
tree_order_statistics_node_update> OrderedMap;
//order_of_key(k) : Number of items strictly smaller than k .
//find_by_order(k) : K-th element in a set (counting from zero).

```

## 1.9 Queue Query

```

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

```

```

class QueueQuery{
private:
    typedef long long t_queue;
    stack<pair<t_queue, t_queue>> s1, s2;
    t_queue cmp(t_queue a, t_queue b){
        return min(a, b);
    }
    void move(){
        if (s2.empty()){
            while (!s1.empty()){
                t_queue element = s1.top().first;
                s1.pop();
                t_queue result = s2.empty() ? element : cmp(element, s2.top().second);
                s2.push({element, result});
            }
        }
    }
public:
    void push(t_queue x){
        t_queue result = s1.empty() ? x : cmp(x, s1.top().second);
        s1.push({x, result});
    }
    void pop(){
        move();
        s2.pop();
    }
    t_queue front(){
        move();
        return s2.top().first;
    }
    t_queue query(){
        if (s1.empty() || s2.empty())
            return s1.empty() ? s2.top().second : s1.top().second;
        else
            return cmp(s1.top().second, s2.top().second);
    }
    t_queue size(){
        return s1.size() + s2.size();
    }
};

```

## 1.10 Range Color

```

#include <bits/stdc++.h>
using namespace std;
class RangeColor{
private:
    typedef long long ll;
    struct Node{
        ll l, r;
        int color;
        Node() {}
        Node(ll l, ll r, int color) : l(l), r(r), color(color) {}
    };
    struct cmp{
        bool operator()(Node a, Node b){
            return a.r < b.r;
        }
    };
};

```

```

std::set<Node, cmp> st;
vector<ll> ans;
public:
RangeColor(ll first, ll last, int maxColor){
    ans.resize(maxColor + 1);
    ans[0] = last - first + 1LL;
    st.insert(Node(first, last, 0));
}
//get color in position x
int get(ll x){
    auto p = st.upper_bound(Node(0, x - 1LL, -1));
    return p->color;
}
//set newColor in [a, b]
void set(ll a, ll b, int newColor){
    auto p = st.upper_bound(Node(0, a - 1LL, -1));
    assert(p != st.end());
    ll l = p->l;
    ll r = p->r;
    int oldColor = p->color;
    ans[oldColor] -= (r - l + 1LL);
    p = st.erase(p);
    if (l < a){
        ans[oldColor] += (a - l);
        st.insert(Node(l, a - 1LL, oldColor));
    }
    if (b < r){
        ans[oldColor] += (r - b);
        st.insert(Node(b + 1LL, r, oldColor));
    }
    while ((p != st.end()) and (p->l <= b)){
        l = p->l;
        r = p->r;
        oldColor = p->color;
        ans[oldColor] -= (r - l + 1LL);
        if (b < r){
            ans[oldColor] += (r - b);
            st.erase(p);
            st.insert(Node(b + 1LL, r, oldColor));
            break;
        }else{
            p = st.erase(p);
        }
    }
    ans[newColor] += (b - a + 1LL);
    st.insert(Node(a, b, newColor));
}
ll countColor(int x){
    return ans[x];
}
};

```

## 1.11 Segment Tree

```

#include <bits/stdc++.h>
using namespace std;
class SegTree{
private:
    typedef long long Node;
    Node neutral = 0;

```

```

vector<Node> st;
vector<int> v;
int n;
Node join(Node a, Node b){
    return (a + b);
}
void build(int node, int i, int j){
    if (i == j){
        st[node] = v[i];
        return;
    }
    int m = (i + j) / 2;
    int l = (node << 1);
    int r = l + 1;
    build(l, i, m);
    build(r, m + 1, j);
    st[node] = join(st[l], st[r]);
}
Node query(int node, int i, int j, int a, int b){
    if ((i > b) or (j < a))
        return neutral;
    if ((a <= i) and (j <= b))
        return st[node];
    int m = (i + j) / 2;
    int l = (node << 1);
    int r = l + 1;
    return join(query(l, i, m, a, b), query(r, m + 1, j, a, b));
}
void update(int node, int i, int j, int idx, Node value){
    if (i == j){
        st[node] = value;
        return;
    }
    int m = (i + j) / 2;
    int l = (node << 1);
    int r = l + 1;
    if (idx <= m)
        update(l, i, m, idx, value);
    else
        update(r, m + 1, j, idx, value);
    st[node] = join(st[l], st[r]);
}
public:
template <class MyIterator>
SegTree(MyIterator begin, MyIterator end){
    n = end - begin;
    v = vector<int>(begin, end);
    st.resize(4 * n + 5);
    build(1, 0, n - 1);
}
//0-indexed [a, b]
Node query(int a, int b){
    return query(1, 0, n - 1, a, b);
}
//0-indexed
void update(int idx, int value){
    update(1, 0, n - 1, idx, value);
}
};

```

## 1.12 Segment Tree 2D

```
#include <bits/stdc++.h>
using namespace std;
struct SegTree2D{
private:
    int n, m;
    typedef int Node;
    Node neutral = -0x3f3f3f3f;
    vector<vector<Node>> seg;
    Node join(Node a, Node b){
        return max(a, b);
    }
public:
    SegTree2D(int n1, int m1){
        n = n1, m = m1;
        seg.assign(2 * n, vector<Node>(2 * m, 0));
    }
    void update(int x, int y, int val){
        assert(0 <= x && x < n && 0 <= y && y < m);
        x += n, y += m;
        seg[x][y] = val;
        for (int j = y / 2; j > 0; j /= 2)
            seg[x][j] = join(seg[x][2 * j], seg[x][2 * j + 1]);
        for (x /= 2; x > 0; x /= 2){
            seg[x][y] = join(seg[2 * x][y], seg[2 * x + 1][y]);
            for (int j = y / 2; j > 0; j /= 2){
                seg[x][j] = join(seg[x][2 * j], seg[x][2 * j + 1]);
            }
        }
    }
    vector<int> getCover(int l, int r, int N){
        l = std::max(0, l);
        r = std::min(N, r);
        vector<int> ans;
        for (l += N, r += N; l < r; l /= 2, r /= 2){
            if (l & 1)
                ans.push_back(l++);
            if (r & 1)
                ans.push_back(--r);
        }
        return ans;
    }
    Node query(int x1, int y1, int x2, int y2){
        auto c1 = getCover(x1, x2 + 1, n);
        auto c2 = getCover(y1, y2 + 1, m);
        Node ans = neutral;
        for (auto i : c1){
            for (auto j : c2){
                ans = join(ans, seg[i][j]);
            }
        }
        return ans;
    }
};
```

## 1.13 Segment Tree Iterative

```
#include <bits/stdc++.h>
using namespace std;
class SegTreeIterative{
private:
    typedef long long Node;
    Node neutral = 0;
    vector<Node> st;
    int n;
    inline Node join(Node a, Node b){
        return a + b;
    }
public:
    template <class MyIterator>
    SegTreeIterative(MyIterator begin, MyIterator end){
        int sz = end - begin;
        for (n = 1; n < sz; n <= 1);
        st.assign(n <= 1, neutral);
        for (int i = 0; i < sz; i++, begin++)
            st[i + n] = (*begin);
        for (int i = n + sz - 1; i > 1; i--)
            st[i >> 1] = join(st[i >> 1], st[i]);
    }
    //0-indexed
    void update(int i, Node x){
        st[i += n] = x;
        for (i >= 1; i; i >= 1)
            st[i] = join(st[i << 1], st[1 + (i << 1)]);
    }
    //0-indexed [l, r]
    Node query(int l, int r){
        Node ans = neutral;
        for (l += n, r += n + 1; l < r; l >= 1, r >= 1){
            if (l & 1)
                ans = join(ans, st[l++]);
            if (r & 1)
                ans = join(ans, st[--r]);
        }
        return ans;
    }
};
```

## 1.14 Segment Tree Lazy

```
#include <bits/stdc++.h>
using namespace std;
class SegTreeLazy{
private:
    typedef long long Node;
    vector<Node> st;
    vector<long long> lazy;
    vector<int> v;
    int n;
    Node neutral = 0;
    inline Node join(Node a, Node b){
        return a + b;
    }
    inline void upLazy(int &node, int &i, int &j){
        if (lazy[node] != 0){
            st[node] += lazy[node] * (j - i + 1);
            //st[node] += lazy[node];
        }
    }
};
```



```

    if (i != j){
        lazy[(node << 1)] += lazy[node];
        lazy[(node << 1) + 1] += lazy[node];
    }
    lazy[node] = 0;
}
}
void build(int node, int i, int j){
    if (i == j){
        st[node] = v[i];
        return;
    }
    int m = (i + j) / 2;
    int l = (node << 1);
    int r = l + 1;
    build(l, i, m);
    build(r, m + 1, j);
    st[node] = join(st[l], st[r]);
}
Node query(int node, int i, int j, int a, int b){
    upLazy(node, i, j);
    if ((i > b) or (j < a))
        return neutral;
    if ((a <= i) and (j <= b)){
        return st[node];
    }
    int m = (i + j) / 2;
    int l = (node << 1);
    int r = l + 1;
    return join(query(l, i, m, a, b), query(r, m + 1, j, a, b));
}
void update(int node, int i, int j, int a, int b, int value){
    upLazy(node, i, j);
    if ((i > j) or (i > b) or (j < a))
        return;
    if ((a <= i) and (j <= b)){
        lazy[node] = value;
        upLazy(node, i, j);
    }else{
        int m = (i + j) / 2;
        int l = (node << 1);
        int r = l + 1;
        update(l, i, m, a, b, value);
        update(r, m + 1, j, a, b, value);
        st[node] = join(st[l], st[r]);
    }
}
}
public:
    template <class MyIterator>
    SegTreeLazy(MyIterator begin, MyIterator end){
        n = end - begin;
        v = vector<int>(begin, end);
        st.resize(4 * n + 5);
        lazy.assign(4 * n + 5, 0);
        build(1, 0, n - 1);
    }
    //0-indexed [a, b]
    Node query(int a, int b){
        return query(1, 0, n - 1, a, b);
    }
    //0-indexed [a, b]

```

```

    void update(int a, int b, int value){
        update(1, 0, n - 1, a, b, value);
    }
};

```

## 1.15 Sparse Table

```

#include <bits/stdc++.h>
using namespace std;
class SparseTable{
private:
    typedef int t_st;
    vector<vector<t_st>> st;
    vector<int> log2;
    t_st neutral = 0x3f3f3f3f;
    int nLog;
    t_st join(t_st a, t_st b){
        return min(a, b);
    }
public:
    template <class MyIterator>
    SparseTable(MyIterator begin, MyIterator end){
        int n = end - begin;
        nLog = 20;
        log2.resize(n + 1);
        log2[1] = 0;
        for (int i = 2; i <= n; i++)
            log2[i] = log2[i / 2] + 1;
        st.resize(n, vector<t_st>(nLog, neutral));
        for (int i = 0; i < n; i++, begin++){
            st[i][0] = (*begin);
            for (int j = 1; j < nLog; j++){
                for (int i = 0; (i + (1 << (j - 1))) < n; i++){
                    st[i][j] = join(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
                }
            }
        }
        //0-indexed [a, b]
        t_st query(int a, int b){
            int d = b - a + 1;
            t_st ans = neutral;
            for (int j = nLog - 1; j >= 0; j--){
                if (d & (1 << j)){
                    ans = join(ans, st[a][j]);
                    a = a + (1 << (j));
                }
            }
            return ans;
        }
        //0-indexed [a, b]
        t_st queryRMQ(int a, int b){
            int j = log2[b - a + 1];
            return join(st[a][j], st[b - (1 << j) + 1][j]);
        }
    }
};

```

## 1.16 SQRT Decomposition

```

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

```

```

struct SqrtDecomposition{
    typedef long long t_sqrt;
    int sqrtLen;
    vector<t_sqrt> block;
    vector<t_sqrt> v;
    template <class MyIterator>
    SqrtDecomposition(MyIterator begin, MyIterator end){
        int n = end - begin;
        sqrtLen = (int)sqrt(n + .0) + 1;
        v.resize(n);
        block.resize(sqrtLen + 5);
        for (int i = 0; i < n; i++, begin++){
            v[i] = (*begin);
            block[i / sqrtLen] += v[i];
        }
    }
    //0-indexed
    void update(int idx, t_sqrt new_value){
        t_sqrt d = new_value - v[idx];
        v[idx] += d;
        block[idx / sqrtLen] += d;
    }
    //0-indexed [l, r]
    t_sqrt query(int l, int r){
        t_sqrt sum = 0;
        int c_l = l / sqrtLen, c_r = r / sqrtLen;
        if (c_l == c_r){
            for (int i = l; i <= r; i++)
                sum += v[i];
        }else{
            for (int i = l, end = (c_l + 1) * sqrtLen - 1; i <= end; i++)
                sum += v[i];
            for (int i = c_l + 1; i <= c_r - 1; i++)
                sum += block[i];
            for (int i = c_r * sqrtLen; i <= r; i++)
                sum += v[i];
        }
        return sum;
    }
};

```

## 1.17 SQRT Tree

```

#include <bits/stdc++.h>
using namespace std;
class SqrtTree{
private:
    typedef long long t_sqrt;
    t_sqrt op(const t_sqrt &a, const t_sqrt &b){
        return a | b;
    }
    inline int log2Up(int n){
        int res = 0;
        while ((1 << res) < n)
            res++;
        return res;
    }
    int n, lg, indexSz;
    vector<t_sqrt> v;
    vector<int> clz, layers, onLayer;

```

```

    vector<vector<t_sqrt>> pref, suf, between;
    inline void buildBlock(int layer, int l, int r){
        pref[layer][l] = v[l];
        for (int i = l + 1; i < r; i++)
            pref[layer][i] = op(pref[layer][i - 1], v[i]);
        suf[layer][r - 1] = v[r - 1];
        for (int i = r - 2; i >= l; i--)
            suf[layer][i] = op(v[i], suf[layer][i + 1]);
    }
    inline void buildBetween(int layer, int lBound, int rBound, int
        betweenOffs){
        int bSzLog = (layers[layer] + 1) >> 1;
        int bCntLog = layers[layer] >> 1;
        int bSz = 1 << bSzLog;
        int bCnt = (rBound - lBound + bSz - 1) >> bSzLog;
        for (int i = 0; i < bCnt; i++){
            t_sqrt ans;
            for (int j = i; j < bCnt; j++){
                t_sqrt add = suf[layer][lBound + (j << bSzLog)];
                ans = (i == j) ? add : op(ans, add);
                between[layer - 1][betweenOffs + lBound + (i << bCntLog) + j]
                    = ans;
            }
        }
    }
    inline void buildBetweenZero(){
        int bSzLog = (lg + 1) >> 1;
        for (int i = 0; i < indexSz; i++){
            v[n + i] = suf[0][i << bSzLog];
        }
        build(1, n, n + indexSz, (1 << lg) - n);
    }
    inline void updateBetweenZero(int bid){
        int bSzLog = (lg + 1) >> 1;
        v[n + bid] = suf[0][bid << bSzLog];
        update(1, n, n + indexSz, (1 << lg) - n, n + bid);
    }
    void build(int layer, int lBound, int rBound, int betweenOffs){
        if (layer >= (int)layers.size())
            return;
        int bSz = 1 << ((layers[layer] + 1) >> 1);
        for (int l = lBound; l < rBound; l += bSz){
            int r = min(l + bSz, rBound);
            buildBlock(layer, l, r);
            build(layer + 1, l, r, betweenOffs);
        }
        if (layer == 0)
            buildBetweenZero();
        else
            buildBetween(layer, lBound, rBound, betweenOffs);
    }
    void update(int layer, int lBound, int rBound, int betweenOffs, int
        x){
        if (layer >= (int)layers.size())
            return;
        int bSzLog = (layers[layer] + 1) >> 1;
        int bSz = 1 << bSzLog;
        int blockIdx = (x - lBound) >> bSzLog;
        int l = lBound + (blockIdx << bSzLog);
        int r = min(l + bSz, rBound);
        buildBlock(layer, l, r);

```

```

    if (layer == 0)
        updateBetweenZero(blockIdx);
    else
        buildBetween(layer, lBound, rBound, betweenOffs);
    update(layer + 1, l, r, betweenOffs, x);
}

inline t_sqrt query(int l, int r, int betweenOffs, int base){
    if (l == r)
        return v[l];
    if (l + 1 == r)
        return op(v[l], v[r]);
    int layer = onLayer[clz[(l - base) ^ (r - base)]];
    int bSzLog = (layers[layer] + 1) >> 1;
    int bCntLog = layers[layer] >> 1;
    int lBound = ((l - base) >> layers[layer]) << layers[layer] +
        base;
    int lBlock = ((l - lBound) >> bSzLog) + 1;
    int rBlock = ((r - lBound) >> bSzLog) - 1;
    t_sqrt ans = suf[layer][l];
    if (lBlock <= rBlock){
        t_sqrt add;
        if (layer == 0)
            add = query(n + lBlock, n + rBlock, (1 << lg) - n, n);
        else
            add = between[layer - 1][betweenOffs + lBound + (lBlock <<
                bCntLog) + rBlock];
        ans = op(ans, add);
    }
    ans = op(ans, pref[layer][r]);
    return ans;
}

public:
    template <class MyIterator>
    SqrtTree(MyIterator begin, MyIterator end){
        n = end - begin;
        v.resize(n);
        for (int i = 0; i < n; i++, begin++)
            v[i] = (*begin);
        lg = log2Up(n);
        clz.resize(1 << lg);
        onLayer.resize(lg + 1);
        clz[0] = 0;
        for (int i = 1; i < (int)clz.size(); i++)
            clz[i] = clz[i >> 1] + 1;
        int tlg = lg;
        while (tlg > 1){
            onLayer[tlg] = (int)layers.size();
            layers.push_back(tlg);
            tlg = (tlg + 1) >> 1;
        }
        for (int i = lg - 1; i >= 0; i--)
            onLayer[i] = max(onLayer[i], onLayer[i + 1]);
        int betweenLayers = max(0, (int)layers.size() - 1);
        int bSzLog = (lg + 1) >> 1;
        int bSz = 1 << bSzLog;
        indexSz = (n + bSz - 1) >> bSzLog;
        v.resize(n + indexSz);
        pref.assign(layers.size(), vector<t_sqrt>(n + indexSz));
        suf.assign(layers.size(), vector<t_sqrt>(n + indexSz));
        between.assign(betweenLayers, vector<t_sqrt>((1 << lg) + bSz));
        build(0, 0, n, 0);
    }

```

```

    }
    //0-indexed
    inline void update(int x, const t_sqrt &item){
        v[x] = item;
        update(0, 0, n, 0, x);
    }
    //0-indexed [l, r]
    inline t_sqrt query(int l, int r){
        return query(l, r, 0, 0);
    }
};

```

## 1.18 Stack Query

```

#include <bits/stdc++.h>
using namespace std;
struct StackQuery{
    typedef int t_stack;
    stack<pair<t_stack, t_stack>> st;
    t_stack cmp(t_stack a, t_stack b){
        return min(a, b);
    }
    void push(t_stack x){
        t_stack new_value = st.empty() ? x : cmp(x, st.top().second);
        st.push({x, new_value});
    }
    void pop(){
        st.pop();
    }
    t_stack top(){
        return st.top().first;
    }
    t_stack query(){
        return st.top().second;
    }
    t_stack size(){
        return st.size();
    }
};

```

## 1.19 Treap

```

#include <bits/stdc++.h>
using namespace std;
namespace Treap{
    const int N = 500010;
    typedef long long treap_t;
    treap_t X[N];
    int en = 1, Y[N], sz[N], L[N], R[N], root;

    const treap_t neutral = 0;
    treap_t op_val[N];
    inline treap_t join(treap_t a, treap_t b, treap_t c){
        return a + b + c;
    }
    void calc(int u) { // update node given children info
        sz[u] = sz[L[u]] + 1 + sz[R[u]];
        // code here, no recursion
    }
}

```

```

    op_val[u] = join(op_val[L[u]], X[u], op_val[R[u]]);
}
void unlaze(int u) {
    if(!u) return;
    // code here, no recursion
}
void split(int u, treap_t x, int &l, int &r) { // l gets <= x, r
    gets > x
    unlaze(u);
    if(!u) return (void) (l = r = 0);
    if(X[u] <= x) { split(R[u], x, l, r); R[u] = l; l = u; }
    else { split(L[u], x, l, r); L[u] = r; r = u; }
    calc(u);
}
int merge(int l, int r) { // els on l <= els on r
    unlaze(l); unlaze(r);
    if(!l || !r) return l + r;
    int u;
    if(Y[l] > Y[r]) { R[l] = merge(R[l], r); u = l; }
    else { L[r] = merge(l, L[r]); u = r; }
    calc(u);
    return u;
}
int new_node(treap_t x){
    X[en] = x;
    op_val[en] = x;
    return en++;
}
int nth(int u, int idx){
    if(!u)
        return 0;
    unlaze(u);
    if(idx <= sz[L[u]])
        return nth(L[u], idx);
    else if(idx == sz[L[u]] + 1)
        return u;
    else
        return nth(R[u], idx - sz[L[u]] - 1);
}
//Public
void init(int n=N-1) { // call before using other funcs
    //init position 0
    sz[0] = 0;
    op_val[0] = neutral;
    //init Treap
    root = 0;
    for(int i = en = 1; i <= n; i++) { Y[i] = i; sz[i] = 1; L[i] = R[i]
        ] = 0; }
    random_shuffle(Y + 1, Y + n + 1);
}
void insert(treap_t x){
    int a, b;
    split(root, x, a, b);
    root = merge(merge(a, new_node(x)), b);
}
void erase(treap_t x){
    int a, b, c, d;
    split(root, x-1, a, b);
    split(b, x, c, d);
    root = merge(a, d);
}

```

```

int count(treap_t x){
    int a, b, c, d;
    split(root, x-1, a, b);
    split(b, x, c, d);
    int ans = sz[c];
    root = merge(a, merge(c, d));
    return ans;
}
int size(){ return sz[root];}
//0-indexed
treap_t nth(int idx){
    int u = nth(root, idx + 1);
    return X[u];
}
};

```

## 1.20 Union Find

```

#include <bits/stdc++.h>
using namespace std;
class UnionFind{
private:
    vector<int> p, w, sz;
public:
    UnionFind(int n){
        w.resize(n + 1, 1);
        sz.resize(n + 1, 1);
        p.resize(n + 1);
        for (int i = 0; i <= n; i++)
            p[i] = i;
    }
    int find(int x){
        if (p[x] == x)
            return x;
        return p[x] = find(p[x]);
    }
    void join(int x, int y){
        x = find(x);
        y = find(y);
        if (x == y)
            return;
        if (w[x] > w[y])
            swap(x, y);
        p[x] = y;
        sz[y] += sz[x];
        if (w[x] == w[y])
            w[y]++;
    }
    bool isSame(int x, int y){
        return find(x) == find(y);
    }
    int size(int x){
        return sz[find(x)];
    }
};

```

## 1.21 Wavelet Tree

```

#include <bits/stdc++.h>
using namespace std;
struct WaveletTree{
private:
    typedef int t_wavelet;
    t_wavelet lo, hi;
    WaveletTree *l = nullptr, *r = nullptr;
    vector<t_wavelet> a;
public:
    template <class MyIterator>
    WaveletTree(MyIterator begin, MyIterator end, t_wavelet minX,
        t_wavelet maxX){
        lo = minX, hi = maxX;
        if (lo == hi or begin >= end)
            return;
        t_wavelet mid = (lo + hi - 1) / 2;
        auto f = [mid](int x) {
            return x <= mid;
        };
        a.reserve(end - begin + 2);
        a.push_back(0);
        for (auto it = begin; it != end; it++)
            a.push_back(a.back() + f(*it));
        auto pivot = stable_partition(begin, end, f);
        l = new WaveletTree(begin, pivot, lo, mid);
        r = new WaveletTree(pivot, end, mid + 1, hi);
    }
    inline int b(int i){
        return i - a[i];
    }
    //kth smallest element in range [i, j]
    //1-indexed
    int kth(int i, int j, int k){
        if (i > j)
            return 0;
        if (lo == hi)
            return lo;
        int inLeft = a[j] - a[i - 1];
        int i1 = a[i - 1] + 1, j1 = a[j];
        int i2 = b(i - 1) + 1, j2 = b(j);
        if (k <= inLeft)
            return l->kth(i1, j1, k);
        return r->kth(i2, j2, k - inLeft);
    }
    //Amount of numbers in the range [i, j] Less than or equal to k
    //1-indexed
    int lte(int i, int j, int k){
        if (i > j or k < lo)
            return 0;
        if (hi <= k)
            return j - i + 1;
        int i1 = a[i - 1] + 1, j1 = a[j];
        int i2 = b(i - 1) + 1, j2 = b(j);
        return l->lte(i1, j1, k) + r->lte(i2, j2, k);
    }
    //Amount of numbers in the range [i, j] equal to k
    //1-indexed
    int count(int i, int j, int k){
        if (i > j or k < lo or k > hi)
            return 0;
        if (lo == hi)

```

```

        return j - i + 1;
        t_wavelet mid = (lo + hi - 1) / 2;
        int i1 = a[i - 1] + 1, j1 = a[j];
        int i2 = b(i - 1) + 1, j2 = b(j);
        if (k <= mid)
            return l->count(i1, j1, k);
        return r->count(i2, j2, k);
    }
    //swap v[i] with v[i+1]
    //1-indexed
    void swap(int i){
        if (lo == hi or a.size() <= 2)
            return;
        if (a[i - 1] + 1 == a[i] and a[i] + 1 == a[i + 1])
            l->swap(a[i]);
        else if (b(i - 1) + 1 == b(i) and b(i) + 1 == b(i + 1))
            r->swap(b(i));
        else if (a[i - 1] + 1 == a[i])
            a[i]--;
        else
            a[i]++;
    }
    ~WaveletTree(){
        if (l) delete l;
        if (r) delete r;
    }
};

```

## 2 Graph Algorithms

### 2.1 2-SAT

```

#include "strongly_connected_component.h"
using namespace std;
struct SAT{
    typedef pair<int, int> pii;
    vector<pii> edges;
    int n;
    SAT(int size){
        n = 2 * size;
    }
    vector<bool> solve2SAT(){
        vector<bool> vAns(n / 2, false);
        vector<int> comp = SCC::scc(n, edges);
        for (int i = 0; i < n; i += 2){
            if (comp[i] == comp[i + 1])
                return vector<bool>();
            vAns[i / 2] = (comp[i] > comp[i + 1]);
        }
        return vAns;
    }
    int v(int x){
        if (x >= 0)
            return (x << 1);
        x = ~x;
        return (x << 1) ^ 1;
    }
    void add(int a, int b){

```

```

    edges.push_back(pii(a, b));
}
void addOr(int a, int b){
    add(v(~a), v(b));
    add(v(~b), v(a));
}
void addImp(int a, int b){
    addOr(~a, b);
}
void addEqual(int a, int b){
    addOr(a, ~b);
    addOr(~a, b);
}
void addDiff(int a, int b){
    addEqual(a, ~b);
}
};

```

## 2.2 Centroid Decomposition

```

#include <bits/stdc++.h>
using namespace std;
// O(N*log(N))
struct CentroidDecomposition{
    vector<vector<int>> adj;
    vector<int> dad, sub;
    vector<bool> rem;
    int centroidRoot, n;
    void init(int _n){
        n = _n;
        adj.resize(n);
        dad.resize(n);
        sub.resize(n);
        rem.assign(n, false);
    }
    // Return Centroid Decomposition Tree
    vector<vector<int>> build(){
        assert(n > 0);
        centroidRoot = decomp(0, -1);
        vector<vector<int>> ret(n);
        for (int u = 0; u < n; u++){
            if (dad[u] != u)
                ret[dad[u]].push_back(u);
        }
        return ret;
    }
    void addEdge(int a, int b){
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    int decomp(int u, int p){
        int sz = dfs(u, p);
        int c = centroid(u, p, sz);
        if (p == -1)
            p = c;
        dad[c] = p;
        rem[c] = true;
        for (auto to : adj[c]){
            if (!rem[to])
                decomp(to, c);
        }
    }
};

```

```

    }
    return c;
}
int dfs(int u, int p){
    sub[u] = 1;
    for (int to : adj[u]){
        if (!rem[to] and to != p)
            sub[u] += dfs(to, u);
    }
    return sub[u];
}
int centroid(int u, int p, int sz){
    for (auto to : adj[u])
        if (!rem[to] and to != p and sub[to] > sz / 2)
            return centroid(to, u, sz);
    return u;
}
int operator[](int i){
    return dad[i];
}
};

```

## 2.3 Dinic

```

#include <bits/stdc++.h>
using namespace std;
//O(V^2*E): for generic graph
//O(sqrt(V)*E): for bipartite graph
template <typename flow_t>
struct Dinic{
    struct FlowEdge{
        int from, to;
        flow_t cap, flow = 0;
        FlowEdge(int from, int to, flow_t cap) : from(from), to(to), cap(
            cap) {}
    };
    const flow_t flow_inf = numeric_limits<flow_t>::max();
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;
    bool bfs(){
        while (!q.empty()){
            int u = q.front();
            q.pop();
            for (int id : adj[u]){
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].to] != -1)
                    continue;
                level[edges[id].to] = level[u] + 1;
                q.push(edges[id].to);
            }
        }
        return level[t] != -1;
    }
    flow_t dfs(int u, flow_t pushed){
        if (pushed == 0)
            return 0;
        if (u == t)
            return pushed;
        for (int id : adj[u])
            if (edges[id].cap - edges[id].flow > 0)
                pushed = dfs(edges[id].to, pushed);
        edges[u].flow += pushed;
        for (int id : adj[u])
            edges[id].flow -= pushed;
        return pushed;
    }
};

```

```

    return 0;
if (u == t)
    return pushed;
for (int &cid = ptr[u]; cid < (int)adj[u].size(); cid++){
    int id = adj[u][cid];
    int to = edges[id].to;
    if (level[u] + 1 != level[to] || edges[id].cap - edges[id].flow
        < 1)
        continue;
    flow_t tr = dfs(to, min(pushed, edges[id].cap - edges[id].flow))
    ;
    if (tr == 0)
        continue;
    edges[id].flow += tr;
    edges[id ^ 1].flow -= tr;
    return tr;
}
return 0;
}
Dinic(){}
void init(int _n){
    n = _n;
    adj.resize(n);
    level.resize(n);
    ptr.resize(n);
}
void addEdge(int from, int to, flow_t cap){
    assert(n>0);
    edges.push_back(FlowEdge(from, to, cap));
    edges.push_back(FlowEdge(to, from, 0));
    adj[from].push_back(m);
    adj[to].push_back(m + 1);
    m += 2;
}
flow_t maxFlow(int s1, int t1){
    s = s1, t = t1;
    flow_t f = 0;
    for(int i=0; i<m; i++){
        edges[i].flow = 0;
    }
    while (true){
        level.assign(n, -1);
        level[s] = 0;
        q.push(s);
        if (!bfs())
            break;
        ptr.assign(n, 0);
        while (flow_t pushed = dfs(s, flow_inf))
            f += pushed;
    }
    return f;
}
};
typedef pair<int, int> pii;
vector<pii> recoverCut(Dinic<int> &d){
    vector<int> level(d.n, 0);
    vector<pii> rc;
    queue<int> q;
    q.push(d.s);
    level[d.s] = 1;
    while (!q.empty()){
        int u = q.front();

```

```

        q.pop();
        for (int id : d.adj[u]){
            if ((id & 1) == 1)
                continue;
            if (d.edges[id].cap == d.edges[id].flow){
                rc.push_back(pii(d.edges[id].from, d.edges[id].to));
            }else{
                if (level[d.edges[id].to] == 0){
                    q.push(d.edges[id].to);
                    level[d.edges[id].to] = 1;
                }
            }
        }
    }
    vector<pii> ans;
    for (pii p : rc)
        if ((level[p.first] == 0) or (level[p.second] == 0))
            ans.push_back(p);
    return ans;
}

```

## 2.4 Flow With Demand

```

#include "dinic.h"
using namespace std;
template <typename flow_t>
struct MaxFlowEdgeDemands{
    Dinic<flow_t> mf;
    vector<flow_t> ind, outd;
    flow_t D;
    int n;
    MaxFlowEdgeDemands(int n) : n(n){
        D = 0;
        mf.init(n + 2);
        ind.assign(n, 0);
        outd.assign(n, 0);
    }
    void addEdge(int a, int b, flow_t cap, flow_t demands){
        mf.addEdge(a, b, cap - demands);
        D += demands;
        ind[b] += demands;
        outd[a] += demands;
    }
    bool solve(int s, int t){
        mf.addEdge(t, s, numeric_limits<flow_t>::max());
        for (int i = 0; i < n; i++){
            if (ind[i]) mf.addEdge(n, i, ind[i]);
            if (outd[i]) mf.addEdge(i, n + 1, outd[i]);
        }
        return mf.maxFlow(n, n + 1) == D;
    }
};

```

## 2.5 Kruskal

```

#include "../data_structures/union_find.h"
typedef long long ll;
struct Edge{

```

```

    int u, v; ll w;
    Edge() {}
    Edge(int u, int v, ll w):u(u), v(v), w(w) {}
};
ll kruskal(vector<Edge> v, int nVet){
    ll cost = 0;
    UnionFind uf(nVet);
    sort(v.begin(), v.end(), [&](Edge a, Edge b){
        return a.w < b.w;
    });
    for(Edge &e: v){
        if(!uf.isSame(e.u, e.v)){
            cost += e.w;
            uf.join(e.u, e.v);
        }
    }
    return cost;
}

```

## 2.6 HLD

```

#include <bits/stdc++.h>
#include "../data_structures/bit_range.h"
using namespace std;
#define F first
#define S second
using hld_t = long long;
using pii = pair<int, hld_t>;
struct HLD{
    vector<vector<pii>> adj;
    vector<int> sz, h, dad, pos;
    vector<hld_t> val, v;
    int t;
    bool edge;
    //Begin Internal Data Structure
    BitRange *bit;
    hld_t neutral = 0;
    inline hld_t join(hld_t a, hld_t b){
        return a+b;
    }
    inline void update(int a, int b, hld_t x){
        bit->add(a+1, b+1, x);
    }
    inline hld_t query(int a, int b){
        return bit->get(a+1, b+1);
    }
    //End Internal Data Structure
    void init(int n){
        dad.resize(n); pos.resize(n); val.resize(n); v.resize(n);
        adj.resize(n); sz.resize(n); h.resize(n);
        bit = new BitRange(n);
    }
    void dfs(int u, int p = -1){
        sz[u] = 1;
        for(pii &to: adj[u]) if(to.F != p){
            if(edge) val[to.F] = to.S;
            dfs(to.F, u);
            sz[u] += sz[to.F];
            if(sz[to.F] > sz[adj[u][0].F] or adj[u][0].F == p)
                swap(to, adj[u][0]);
        }
    }
}

```

```

    }
}
void build_hld(int u, int p=-1){
    dad[u] = p;
    pos[u] = t++;
    v[pos[u]] = val[u];
    for(pii to: adj[u]) if(to.F != p){
        h[to.F] = (to == adj[u][0]) ? h[u] : to.F;
        build_hld(to.F, u);
    }
}
void addEdge(int a, int b, hld_t w = 0){
    adj[a].emplace_back(b, w);
    adj[b].emplace_back(a, w);
}
void build(int root, bool is_edge){
    assert(!adj.empty());
    edge = is_edge;
    t = 0;
    h[root] = 0;
    dfs(root);
    build_hld(root);
    //Init Internal Data Structure
    for(int i=0; i<t; i++)
        update(i, i, v[i]);
}
hld_t query_path(int a, int b) {
    if (edge and a == b) return neutral;
    if (pos[a] < pos[b]) swap(a, b);
    if (h[a] == h[b]) return query(pos[b]+edge, pos[a]);
    return join(query(pos[h[a]], pos[a]), query_path(dad[h[a]], b));
}
void update_path(int a, int b, hld_t x) {
    if (edge and a == b) return;
    if (pos[a] < pos[b]) swap(a, b);
    if (h[a] == h[b]) return (void)update(pos[b]+edge, pos[a], x);
    update(pos[h[a]], pos[a], x); update_path(dad[h[a]], b, x);
}
hld_t query_subtree(int a) {
    if (edge and sz[a] == 1) return neutral;
    return query(pos[a]+edge, pos[a]+sz[a]-1);
}
void update_subtree(int a, hld_t x) {
    if (edge and sz[a] == 1) return;
    update(pos[a] + edge, pos[a]+sz[a]-1, x);
}
int lca(int a, int b) {
    if (pos[a] < pos[b]) swap(a, b);
    return h[a] == h[b] ? b : lca(dad[h[a]], b);
}
};

```

## 2.7 Minimum Cost Maximum Flow

```

#include <bits/stdc++.h>
using namespace std;
template <class T = int>
class MCMF{
private:
    struct Edge{

```



```

    int to;
    T cap, cost;
    Edge(int a, T b, T c) : to(a), cap(b), cost(c) {}
};
int n;
vector<vector<int>> edges;
vector<Edge> list;
vector<int> from;
vector<T> dist, pot;
vector<bool> visit;
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 = std::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;
}
void fixPot(){
    T INF = numeric_limits<T>::max();
    for (int i = 0; i < n; i++){
        if (dist[i] < INF)
            pot[i] += dist[i];
    }
}
public:
MCMF(int size){
    n = size;
    edges.resize(n);

```

```

    pot.assign(n, 0);
    dist.resize(n);
    visit.assign(n, false);
}
pair<T, T> solve(int src, int sink){
    pair<T, T> ans(0, 0);
    // Can use dijkstra to speed up depending on the graph
    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));
}
};
/*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;
    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;
}*/

```

## 2.8 Strongly Connected Component

```

#include "topological_sort.h"
using namespace std;
namespace SCC{
    typedef pair<int, int> pii;

```

```

vector<vector<int>> revAdj;
vector<int> component;
void dfs(int u, int c){
    component[u] = c;
    for (int to : revAdj[u]){
        if (component[to] == -1)
            dfs(to, c);
    }
}
vector<int> scc(int n, vector<pii> &edges){
    revAdj.assign(n, vector<int>());
    for (pii p : edges)
        revAdj[p.second].push_back(p.first);
    vector<int> tp = TopologicalSort::order(n, edges);
    component.assign(n, -1);
    int comp = 0;
    for (int u : tp){
        if (component[u] == -1)
            dfs(u, comp++);
    }
    return component;
}
} // namespace SCC

```

## 2.9 Topological Sort

```

#include <bits/stdc++.h>
using namespace std;
namespace TopologicalSort{
    typedef pair<int, int> pii;
    vector<vector<int>> adj;
    vector<bool> visited;
    vector<int> vAns;
    void dfs(int u){
        visited[u] = true;
        for (int to : adj[u]){
            if (!visited[to])
                dfs(to);
        }
        vAns.push_back(u);
    }
    vector<int> order(int n, vector<pii> &edges){
        adj.assign(n, vector<int>());
        for (pii p : edges)
            adj[p.first].push_back(p.second);
        visited.assign(n, false);
        vAns.clear();
        for (int i = 0; i < n; i++){
            if (!visited[i])
                dfs(i);
        }
        reverse(vAns.begin(), vAns.end());
        return vAns;
    }
}; // namespace TopologicalSort

```

## 3 Dynamic Programming

### 3.1 Divide and Conquer Optimization

Reduces the complexity from  $O(n^2k)$  to  $O(nk \log n)$  of PD's in the following ways (and other variants):

$$dp[n][k] = \max_{0 \leq i < n} (dp[i][k-1] + C[i+1][n]), \text{ base case : } dp[0][j], dp[i][0] \quad (1)$$

- $C[i][j]$  = the cost only depends on  $i$  and  $j$ .
- $opt[n][k] = i$  is the optimal value that maximizes  $dp[n][k]$ .

It is necessary that  $opt$  is increasing along each column:  $opt[j][k] \leq opt[j+1][k]$ .

### 3.2 Divide and Conquer Optimization Implementation

```

#include <bits/stdc++.h>
using namespace std;
int C(int i, int j);
const int MAXN = 100010;
const int MAXK = 110;
const int INF = 0x3f3f3f3f;
int dp[MAXN][MAXK];
void calculateDP(int l, int r, int k, int opt_l, int opt_r){
    if (l > r)
        return;
    int mid = (l + r) >> 1;
    int ans = -INF, opt = mid;
    // int ans = dp[mid][k-1], opt=mid; //If you accept empty subsegment
    for (int i = opt_l; i <= min(opt_r, mid - 1); i++){
        if (ans < dp[i][k-1] + C(i+1, mid)){
            opt = i;
            ans = dp[i][k-1] + C(i+1, mid);
        }
    }
    dp[mid][k] = ans;
    calculateDP(l, mid - 1, k, opt_l, opt);
    calculateDP(mid + 1, r, k, opt, opt_r);
}
int solve(int n, int k){
    for (int i = 0; i <= n; i++)
        dp[i][0] = -INF;
    for (int j = 0; j <= k; j++)
        dp[0][j] = -INF;
    dp[0][0] = 0;
    for (int j = 1; j <= k; j++)
        calculateDP(1, n, j, 0, n - 1);
    return dp[n][k];
}

```

### 3.3 Knuth Optimization

Reduces the complexity from  $O(n^3)$  to  $O(n^2)$  of PD's in the following ways (and other variants):

$$dp[i][j] = C[i][j] + \min_{i < k < j} (dp[i][k] + dp[k][j]), \text{ caso base : } dp[i][i] \quad (2)$$

$$dp[i][j] = \min_{i < k < j} (dp[i][k] + C[i][k]), \text{ caso base : } dp[i][i] \quad (3)$$

- $C[i][j]$  = the cost only depends on  $i$  and  $j$ .
- $opt[i][j] = k$  is the optimal value that maximizes  $dp[i][j]$ .

The following conditions must be met:

- Foursquare inequality on  $C$ :  $C[a][c] + C[b][d] \leq C[a][d] + C[b][c]$ ,  $a \leq b \leq c \leq d$ .
- Monotonicity on  $C$ :  $C[b][c] \leq C[a][d]$ ,  $a \leq b \leq c \leq d$ .

Or the following condition:

- $opt$  increasing in rows and columns:  $opt[i][j-1] \leq opt[i][j] \leq opt[i+1][j]$ .

### 3.4 Knuth Optimization Implementation

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAXN = 1009;
const ll INFLL = 0x3f3f3f3f3f3f3f3f;
ll C(int a, int b);
ll dp[MAXN][MAXN];
int opt[MAXN][MAXN];
ll knuth(int n){
    for (int i = 0; i < n; i++){
        dp[i][i] = 0;
        opt[i][i] = i;
    }
    for (int s = 1; s < n; s++){
        for (int i = 0, j; (i + s) < n; i++){
            j = i + s;
            dp[i][j] = INFLL;
            for (int k = opt[i][j-1]; k < min(j, opt[i+1][j]+1); k++){
                ll cur = dp[i][k] + dp[k+1][j] + C(i, j);
                if (dp[i][j] > cur){
                    dp[i][j] = cur;
                    opt[i][j] = k;
                }
            }
        }
    }
    return dp[0][n-1];
}
```

## 4 Math

### 4.1 Basic Math

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef unsigned long long ull;

ull fastPow(ull base, ull exp, ull mod){
    base %= mod;
    //exp %= phi(mod) if base and mod are relatively prime
    ull ans = 1LL;
    while (exp > 0){
        if (exp & 1LL)
            ans = (ans * (__int128_t)base) % mod;
        base = (base * (__int128_t)base) % mod;
        exp >>= 1;
    }
    return ans;
}

ll gcd(ll a, ll b){ return __gcd(a, b); }
ll lcm(ll a, ll b){ return (a / gcd(a, b)) * b; }

void enumeratingAllSubmasks(int mask){
    for (int s = mask; s; s = (s-1) & mask)
        cout << s << endl;
}

//MOD to Hash
namespace ModHash{
    const uint64_t MOD = (1ll<<61) - 1;
    uint64_t modmul(uint64_t a, uint64_t b){
        uint64_t l1 = (uint32_t)a, h1 = a>>32, l2 = (uint32_t)b, h2 = b
        >>32;
        uint64_t l = l1*l2, m = l1*h2 + l2*h1, h = h1*h2;
        uint64_t ret = (l&MOD) + (l>>61) + (h << 3) + (m >> 29) + ((m <<
        35) >> 3) + 1;
        ret = (ret & MOD) + (ret>>61);
        ret = (ret & MOD) + (ret>>61);
        return ret-1;
    }
};
```

### 4.2 BigInt

```
#include <bits/stdc++.h>
using namespace std;
typedef int32_t intB;
typedef int64_t longB;
typedef vector<intB> vib;
class BigInt{
private:
    vib vb;
    bool neg;
    const int BASE_DIGIT = 9;
    const intB base = 1000000LL*1000; //000LL*1000000LL;
    void fromString(string &s){
        if(s[0] == '-'){
```

```

    neg = true;
    s = s.substr(1);
} else {
    neg = false;
}
vb.clear();
vb.reserve((s.size()+BASE_DIGIT-1)/BASE_DIGIT);
for(int i=(int)s.length(); i>0; i-=BASE_DIGIT){
    if(i < BASE_DIGIT)
        vb.push_back(stol(s.substr(0, i)));
    else
        vb.push_back(stol(s.substr(i-BASE_DIGIT, BASE_DIGIT)));
}
fix(vb);
}
void fix(vib &v){
    while(v.size()>1 && v.back()==0)
        v.pop_back();
    if(v.size() == 0)
        neg = false;
}
bool comp(vib &a, vib &b){
    fix(a); fix(b);
    if(a.size() != b.size()) return a.size() < b.size();
    for(int i=(int)a.size()-1; i>=0; i--){
        if(a[i] != b[i]) return a[i] < b[i];
    }
    return false;
}
vib sum(vib a, vib b){
    int carry = 0;
    for(size_t i=0; i<max(a.size(), b.size()) or carry; i++){
        if(i == a.size())
            a.push_back(0);
        a[i] += carry + (i<b.size() ? b[i] : 0);
        carry = (a[i] >= base);
        if(carry) a[i] -= base;
    }
    fix(a);
    return a;
}
vib sub(vib a, vib b){
    int carry = 0;
    for(size_t i=0; i<b.size() or carry; i++){
        a[i] -= carry + (i<b.size() ? b[i] : 0);
        carry = a[i] < 0;
        if(carry) a[i] += base;
    }
    fix(a);
    return a;
}

public:
    BigInt(){}
    BigInt(intB n){
        neg = (n<0);
        vb.push_back(abs(n));
        fix(vb);
    }
    BigInt(string s){
        fromString(s);

```

```

    }
    BigInt operator =(BigInt oth){
        this->neg = oth.neg;
        this->vb = oth.vb;
        return *this;
    }
    BigInt operator +(BigInt &oth){
        vib &a = vb, &b = oth.vb;
        BigInt ans;
        if(neg == oth.neg){
            ans.vb = sum(vb, oth.vb);
            ans.neg = neg;
        } else {
            if(comp(a, b)){
                ans.vb = sub(b, a);
                ans.neg = oth.neg;
            } else {
                ans.vb = sub(a, b);
                ans.neg = neg;
            }
        }
        return ans;
    }
    BigInt operator -(BigInt oth){
        oth.neg ^= true;
        return (*this) + oth;
    }
    BigInt operator *(intB b){
        bool negB = false;
        if(b < 0){
            negB = true;
            b = -b;
        }
        BigInt ans = *this;
        auto &a = ans.vb;
        intB carry = 0;
        for(size_t i=0; i<a.size() or carry; i++){
            if(i == a.size()) a.push_back(0);
            longB cur = carry + a[i] *(longB) b;
            a[i] = intB(cur%base);
            carry = intB(cur/base);
        }
        ans.neg ^= negB;
        fix(ans.vb);
        return ans;
    }
    BigInt operator *(BigInt &oth){
        BigInt ans;
        auto a = vb, &b = oth.vb, &c = ans.vb;
        c.assign(a.size() + b.size(), 0);
        for(size_t i=0; i<a.size(); i++){
            intB carry=0;
            for(size_t j=0; j<b.size() or carry; j++){
                longB cur = c[i+j] + a[i]*(longB)(j<b.size() ? b[j] : 0);
                cur += carry;
                c[i+j] = intB(cur%base);
                carry = intB(cur/base);
            }
        }
        ans.neg = neg^oth.neg;
        fix(ans.vb);

```

```

    return ans;
}
BigInt operator / (intB b){
    bool negB = false;
    if(b < 0){
        negB = true;
        b = -b;
    }
    BigInt ans = *this;
    auto &a = ans.vb;
    intB carry = 0;
    for(int i=(int)a.size()-1; i>=0; i--){
        longB cur = a[i] + (longB)carry * base;
        a[i] = intB(cur/b);
        carry = intB(cur%b);
    }
    ans.neg ^= negB;
    fix(ans.vb);
    return ans;
}
void shiftL(int b){
    vb.resize(vb.size() + b);
    for(int i=(int)vb.size()-1; i>=0; i--){
        if(i>=b) vb[i] = vb[i-b];
        else vb[i] = 0;
    }
    fix(vb);
}
void shiftR(int b){
    if((int)vb.size() <= b){
        vb.clear();
        vb.push_back(0);
        return;
    }
    for(int i=0; i<((int)vb.size() - b); i++){
        vb[i] = vb[i+b];
    }
    vb.resize((int)vb.size() - b);
    fix(vb);
}
void divide(BigInt a, BigInt b, BigInt &q, BigInt &r){
    BigInt z(0), p(1);
    while(b < a) {
        p.shiftL(max(1, (int)(a.vb.size()-b.vb.size())));
        b.shiftL(max(1, (int)(a.vb.size()-b.vb.size())));
    }
    while(true) {
        while ((a < b) && (z < p)) {
            p = p/10;
            b = b/10;
        }
        if(!(z < p)) break;
        a = a - b;
        q = q + p;
    }
    r = a;
}
BigInt operator % (BigInt &oth){
    BigInt q, r;
    divide(*this, oth, q, r);
    return q;
}

```

```

BigInt operator % (BigInt &oth){
    BigInt q, r;
    divide(*this, oth, q, r);
    return r;
}
bool operator < (BigInt &oth){
    BigInt ans = (*this) - oth;
    return ans.neg;
}
bool operator == (BigInt &oth){
    BigInt ans = (*this) - oth;
    return (ans.vb.size()==1) and (ans.vb.back()==0);
}
friend ostream &operator<<(ostream &out, const BigInt &D){
    if(D.neg)
        out << '-';
    out << (D.vb.empty() ? 0 : D.vb.back());
    for(int i=(int)D.vb.size()-2; i>=0; i--){
        out << setfill('0') << setw(D.BASE_DIGIT) << D.vb[i];
    }
    return out;
}
string to_string(){
    std::stringstream ss;
    ss << (*this);
    return ss.str();
}
friend istream &operator>>(istream &input, BigInt &D) {
    string s;
    input >> s;
    D.fromString(s);
    return input;
}
};

```

### 4.3 Binomial Coefficients

```

#include <bits/stdc++.h>
#include "../basic_math.h"
#include "../modular.h"
using namespace std;
typedef long long ll;
//O(k)
ll C1(int n, int k){
    ll res = 1LL;
    for (int i = 1; i <= k; ++i)
        res = (res * (n - k + i)) / i;
    return res;
}
//O(n^2)
vector<vector<ll>> C2(int maxn, int mod){
    vector<vector<ll>> mat(maxn + 1, vector<ll>(maxn + 1, 0));
    mat[0][0] = 1;
    for (int n = 1; n <= maxn; n++){
        mat[n][0] = mat[n][n] = 1;
        for (int k = 1; k < n; k++){
            mat[n][k] = (mat[n - 1][k - 1] + mat[n - 1][k]) % mod;
        }
    }
    return mat;
}
//O(N)

```

```

vector<int> factorial, inv_factorial;
void prevC3(int maxn, int mod){
    factorial.resize(maxn + 1);
    factorial[0] = 1;
    for (int i = 1; i <= maxn; i++){
        factorial[i] = (factorial[i - 1] * 1LL * i) % mod;
    }
    inv_factorial.resize(maxn + 1);
    inv_factorial[maxn] = fastPow(factorial[maxn], mod - 2, mod);
    for (int i = maxn - 1; i >= 0; i--){
        inv_factorial[i] = (inv_factorial[i + 1] * 1LL * (i + 1)) % mod;
    }
}
int C3(int n, int k, int mod){
    if (n < k)
        return 0;
    return (((factorial[n] * 1LL * inv_factorial[k]) % mod) * 1LL *
            inv_factorial[n - k]) % mod;
}
//O(P*log(P))
//C4(n, k, p) = Comb(n, k)%p
vector<int> changeBase(int n, int p){
    vector<int> v;
    while (n > 0){
        v.push_back(n % p);
        n /= p;
    }
    return v;
}
int C4(int n, int k, int p){
    auto vn = changeBase(n, p);
    auto vk = changeBase(k, p);
    int mx = max(vn.size(), vk.size());
    vn.resize(mx, 0);
    vk.resize(mx, 0);
    prevC3(p - 1, p);
    int ans = 1;
    for (int i = 0; i < mx; i++){
        ans = (ans * 1LL * C3(vn[i], vk[i], p)) % p;
    }
    return ans;
}
//O(P^k)
//C5(n, k, p, pk) = Comb(n, k)%(p^k)
int fat_p(ll n, int p, int pk){
    vector<int> fatl(pk, 1);
    int res = 1;
    for(int i=1; i<pk; i++){
        if(i%p == 0)
            fatl[i] = fatl[i-1];
        else
            fatl[i] = (fatl[i-1]*1LL*i)%pk;
    }
    while(n > 1){
        res = (res*1LL*fastPow(fatl[pk-1], n/pk, pk))%pk;
        res = (res*1LL*fatl[n%pk])%pk;
        n /= p;
    }
    return res;
}
ll cnt(ll n, int p){
    ll ans = 0;
    while(n > 1){
        ans += n/p;

```

```

        n/=p;
    }
    return ans;
}
int C5(ll n, ll k, int p, int pk){
    ll exp = cnt(n, p) - cnt(n-k, p) - cnt(k, p);
    int d = (fat_p(n-k, p, pk)*1LL*fat_p(k, p, pk))%pk;
    int ans = (fat_p(n, p, pk)*1LL*inv(d, pk))%pk;
    return (ans*1LL*fastPow(p, exp, pk))%pk;
}

```

## 4.4 Chinese Remainder Theorem

```

#include <bits/stdc++.h>
#include "extended_euclidean.h"
using namespace std;
typedef long long ll;
namespace CRT{
    inline ll normalize(ll x, ll mod){
        x %= mod;
        if (x < 0)
            x += mod;
        return x;
    }
    ll solve(vector<ll> a, vector<ll> m){
        int n = a.size();
        for (int i = 0; i < n; i++)
            normalize(a[i], m[i]);
        ll ans = a[0];
        ll lcm1 = m[0];
        for (int i = 1; i < n; i++){
            ll x, y;
            ll g = extGcd(lcm1, m[i], x, y);
            if ((a[i] - ans) % g != 0)
                return -1;
            ans = normalize(ans + (((a[i] - ans) / g) * x) % (m[i] / g)) *
                        lcm1, (lcm1 / g) * m[i]);
            lcm1 = (lcm1 / g) * m[i]; //lcm(lcm1, m[i]);
        }
        return ans;
    }
} // namespace CRT

```

## 4.5 Euler's totient

```

#include <bits/stdc++.h>
using namespace std;
int nthPhi(int n){
    int result = n;
    for (int i = 2; i <= n / i; i++){
        if (n % i == 0){
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
}

```

```

    return result;
}
vector<int> phiFromltoN(int n){
    vector<int> vPhi(n + 1);
    vPhi[0] = 0;
    vPhi[1] = 1;
    for (int i = 2; i <= n; i++){
        vPhi[i] = i;
        for (int i = 2; i <= n; i++){
            if (vPhi[i] == i){
                for (int j = i; j <= n; j += i)
                    vPhi[j] -= vPhi[j] / i;
            }
        }
    }
    return vPhi;
}

```

## 4.6 Extended Euclidean

```

#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll extGcd(ll a, ll b, ll &x, ll &y){
    if (b == 0){
        x = 1, y = 0;
        return a;
    }else{
        ll g = extGcd(b, a % b, y, x);
        y -= (a / b) * x;
        return g;
    }
}
//a*x + b*y = g
//a*(x-(b/g)*k) + b*(y+(a/g)*k) = g
bool dioEq(ll a, ll b, ll c, ll &x0, ll &y0, ll &g){
    g = extGcd(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;
}
inline void shift_solution(ll &x, ll &y, ll a, ll b, ll cnt){
    x += cnt * b;
    y -= cnt * a;
}
ll findAllSolutions(ll a, ll b, ll c, ll minx, ll maxx, ll miny, ll maxy){
    ll x, y, g;
    if(a==0 or b==0){
        if(a==0 and b==0)
            return (c==0)*(maxx-minx+1)*(maxy-miny+1);
        if(a == 0)
            return (c%b == 0)*(maxx-minx+1)*(miny<=c/b and c/b<=maxy);
        return (c%a == 0)*(minx<=c/a and c/a<=maxx)*(maxy-miny+1);
    }
    if (!dioEq(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;
    ll lx1 = x;
    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    ll 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;
    ll lx2 = x;
    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    ll rx2 = x;
    if (lx2 > rx2)
        swap(lx2, rx2);
    ll lx = max(lx1, lx2);
    ll rx = min(rx1, rx2);
    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}

```

## 4.7 Fraction

```

#include <bits/stdc++.h>
using namespace std;
typedef long long f_type;
//Representation of the a/b
struct Fraction {
    f_type a, b;
    Fraction(f_type _a = 0): a(_a), b(1){}
    Fraction(f_type _a, f_type _b) {
        f_type g = __gcd(_a, _b);
        a = _a/g;
        b = _b/g;
        if(b < 0){
            a = -a;
            b = -b;
        }
    }
    Fraction operator+(Fraction oth) {
        return Fraction(a*oth.b + oth.a*b, b*oth.b);
    }
    Fraction operator-(Fraction oth) {
        return Fraction(a*oth.b - oth.a*b, b*oth.b);
    }
    Fraction operator*(Fraction oth) {
        return Fraction(a*oth.a, b*oth.b);
    }
    Fraction operator/(Fraction oth) {

```

```

    return Fraction(a*oth.b, b*oth.a);
}
bool operator>=(Fraction oth){
    return ((*this) - oth).a >= 0;
}
bool operator==(Fraction oth){
    return a == oth.a and b == oth.b;
}
operator f_type() {return a/b;}
operator double() {return double(a)/b;}
};

```

## 4.8 Gray Code

```

int grayCode(int nth){
    return nth ^ (nth >> 1);
}
int revGrayCode(int g){
    int nth = 0;
    for (; g > 0; g >>= 1)
        nth ^= g;
    return nth;
}

```

## 4.9 Matrix

```

#include <bits/stdc++.h>
#include "modular.h"
using namespace std;
const int D = 3;
struct Matrix{
    int m[D][D];
    Matrix(bool identify = false){
        memset(m, 0, sizeof(m));
        for (int i = 0; i < D; i++)
            m[i][i] = identify;
    }
    Matrix(vector<vector<int>> mat){
        for(int i=0; i<D; i++)
            for(int j=0; j<D; j++)
                m[i][j] = mat[i][j];
    }
    int * operator[] (int pos){
        return m[pos];
    }
    Matrix operator*(Matrix oth){
        Matrix ans;
        for (int i = 0; i < D; i++){
            for (int j = 0; j < D; j++){
                int &sum = ans[i][j];
                for (int k = 0; k < D; k++)
                    sum = modSum(sum, modMul(m[i][k], oth[k][j]));
            }
        }
        return ans;
    }
};

```

## 4.10 Modular Arithmetic

```

#include <bits/stdc++.h>
#include "extended_euclidean.h"
using namespace std;
const int MOD = 1000000007;
inline int modSum(int a, int b, int mod = MOD){
    int ans = a+b;
    if(ans > mod) ans -= mod;
    return ans;
}
inline int modSub(int a, int b, int mod = MOD){
    int ans = a-b;
    if(ans < 0) ans += mod;
    return ans;
}
inline int modMul(int a, int b, int mod = MOD){
    return (a*1LL*b)%mod;
}
int inv(int a, int mod=MOD){
    ll inv_x, y;
    extGcd(a, mod, inv_x, y);
    return (inv_x%mod + mod)%mod;
}
int modDiv(int a, int b, int mod = MOD){
    return modMul(a, inv(b, mod));
}

```

## 4.11 Montgomery Multiplication

```

#include <bits/stdc++.h>
using namespace std;
using u64 = uint64_t;
using u128 = __uint128_t;
using i128 = __int128_t;
struct u256{
    u128 high, low;
    static u256 mult(u128 x, u128 y){
        u64 a = x >> 64, b = x;
        u64 c = y >> 64, d = y;
        u128 ac = (u128)a * c;
        u128 ad = (u128)a * d;
        u128 bc = (u128)b * c;
        u128 bd = (u128)b * d;
        u128 carry = (u128)(u64)ad + (u128)(u64)bc + (bd >> 64u);
        u128 high = ac + (ad >> 64u) + (bc >> 64u) + (carry >> 64u);
        u128 low = (ad << 64u) + (bc << 64u) + bd;
        return {high, low};
    }
};
//x_m := x*r mod n
struct Montgomery{
    u128 mod, inv, r2;
    //the N will be an odd number
    Montgomery(u128 n) : mod(n), inv(1), r2((-n % n){
        for (int i = 0; i < 7; i++)
            inv *= 2 - n * inv;
        for (int i = 0; i < 4; i++){

```



```

    r2 <= 1;
    if (r2 >= mod)
        r2 -= mod;
}
for (int i = 0; i < 5; i++)
    r2 = mult(r2, r2);
}
ul28 init(ul28 x){
    return mult(x, r2);
}
ul28 reduce(u256 x){
    ul28 q = x.low * inv;
    i128 a = x.high - u256::mult(q, mod).high;
    if (a < 0)
        a += mod;
    return a;
}
ul28 mult(ul28 a, ul28 b){
    return reduce(u256::mult(a, b));
}
};

```

## 4.12 Prime Number

```

#include <bits/stdc++.h>
#include "basic_math.h"
using namespace std;
typedef unsigned long long ull;
ull modMul(ull a, ull b, ull mod){
    return (a * (__uint128_t)b) % mod;
}
bool checkComposite(ull n, ull a, ull d, int s){
    ull x = fastPow(a, d, n);
    if (x == 1 or x == n - 1)
        return false;
    for (int r = 1; r < s; r++){
        x = modMul(x, x, n);
        if (x == n - 1LL)
            return false;
    }
    return true;
};
bool millerRabin(ull n){
    if (n < 2)
        return false;
    int r = 0;
    ull d = n - 1LL;
    while ((d & 1LL) == 0){
        d >>= 1;
        r++;
    }
    for (ull a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}){
        if (n == a)
            return true;
        if (checkComposite(n, a, d, r))
            return false;
    }
    return true;
}
ull pollard(ull n){

```

```

auto f = [n](ull x) { return modMul(x, x, n) + 1; };
ull x = 0, y = 0, t = 0, prd = 2, i = 1, q;
while (t++ % 40 || __gcd(prd, n) == 1){
    if (x == y)
        x = ++i, y = f(x);
    if ((q = modMul(prd, max(x, y) - min(x, y), n))
        prd = q;
        x = f(x), y = f(f(y));
    }
    return __gcd(prd, n);
}
vector<ull> factor(ull n){
    if (n == 1)
        return {};
    if (millerRabin(n))
        return {n};
    ull x = pollard(n);
    auto l = factor(x), r = factor(n / x);
    l.insert(l.end(), r.begin(), r.end());
    return l;
}

```

## 5 Geometry

### 5.1 Basic Geometry

```

#include <bits/stdc++.h>
using namespace std;
#define POINT_DOUBLE
#ifdef POINT_DOUBLE
    typedef double ftype;
    typedef long double ftLong;
    const double EPS = 1e-9;
    #define eq(a, b) (abs(a - b) < EPS)
    #define lt(a, b) ((a + EPS) < b)
    #define gt(a, b) (a > (b + EPS))
    #define le(a, b) (a < (b + EPS))
    #define ge(a, b) ((a + EPS) > b)
#else
    typedef int32_t ftype;
    typedef int64_t ftLong;
    #define eq(a, b) (a == b)
    #define lt(a, b) (a < b)
    #define gt(a, b) (a > b)
    #define le(a, b) (a <= b)
    #define ge(a, b) (a >= b)
#endif
//Begin Point 2D
struct Point2d{
    ftype x, y;
    Point2d() {}
    Point2d(ftype x, ftype y) : x(x), y(y) {}
    Point2d operator+(const Point2d &t){
        return Point2d(x + t.x, y + t.y);
    }
    Point2d operator-(const Point2d &t){
        return Point2d(x - t.x, y - t.y);
    }
}

```

```

Point2d operator*(ftype t){
    return Point2d(x * t, y * t);
}
Point2d operator/(ftype t){
    return Point2d(x / t, y / t);
}
bool operator<(const Point2d &o) const{
    return lt(x, o.x) or (eq(x, o.x) and lt(y, o.y));
}
bool operator==(const Point2d &o) const{
    return eq(x, o.x) and eq(y, o.y);
}
};
ftLong pw2(ftype a){
    return a * (ftLong)a;
}
//Scalar product
ftLong dot(Point2d a, Point2d b){
    return a.x*(ftLong)b.x + a.y*(ftLong)b.y;
}
ftLong norm(Point2d a){
    return dot(a, a);
}
double len(Point2d a){
    return sqrtl(dot(a, a));
}
double dist(Point2d a, Point2d b){
    return len(a - b);
}
//Vector product
ftLong cross(Point2d a, Point2d b){
    return a.x * (ftLong)b.y - a.y * (ftLong)b.x;
}
//Projection size from A to B
double proj(Point2d a, Point2d b){
    return dot(a, b) / len(b);
}
//The angle between A and B
double angle(Point2d a, Point2d b){
    return acos(dot(a, b) / len(a) / len(b));
}
//Left rotation. Angle in radian
Point2d rotateL(Point2d p, double ang){
    return Point2d(p.x * cos(ang) - p.y * sin(ang), p.x * sin(ang) + p.y * cos(ang));
}
//90 degree left rotation
Point2d perpL(Point2d a){
    return Point2d(-a.y, a.x);
}
//0-> 1o,2o quadrant, 1-> 3o,4o
int half(Point2d &p){
    if (gt(p.y, 0) or (eq(p.y, 0) and ge(p.x, 0)))
        return 0;
    else
        return 1;
}
//angle(a) < angle(b)
bool cmpByAngle(Point2d a, Point2d b){
    int ha = half(a), hb = half(b);
    if (ha != hb)

```

```

        return ha < hb;
    else
        return gt(cross(a, b), 0);
}
inline int sgn(ftLong x){
    return ge(x, 0) ? (eq(x, 0) ? 0 : 1) : -1;
}
// -1: angle(a, b) < angle(b, c)
// 0: angle(a, b) = angle(b, c)
// +1: angle(a, b) > angle(b, c)
int cmpAngleBetweenVectors(Point2d a, Point2d b, Point2d c){
    ftLong dotAB = dot(a, b), dotBC = dot(b, c);
    int sgnAB = sgn(dotAB), sgnBC = sgn(dotBC);
    if (sgnAB == sgnBC){
        ftLong l = pw2(dotAB)*dot(c, c), r = pw2(dotBC)*dot(a, a);
        if (l == r)
            return 0;
        if (sgnAB == 1)
            return (l > r)? -1 : +1;
        return (l < r)? -1 : +1;
    }else{
        return (sgnAB > sgnBC)? -1 : +1;
    }
}
//Line parameterized: r1 = a1 + d1*t
//This function can be generalized to 3D
Point2d intersect(Point2d a1, Point2d d1, Point2d a2, Point2d d2){
    return a1 + d1 * (cross(a2 - a1, d2) / cross(d1, d2));
}
//distance between the point(a) e line(p11, p12)
//This function can be generalized to 3D
double dist(Point2d a, Point2d p11, Point2d p12){
    //crs = parallelogram area
    double crs = cross(Point2d(a - p11), Point2d(p12 - p11));
    //h = area/base
    return abs(crs / dist(p11, p12));
}
double area(vector<Point2d> p){
    double ret = 0;
    for (int i = 2; i < (int)p.size(); i++){
        ret += cross(p[i] - p[0], p[i - 1] - p[0]) / 2.0;
    }
    return abs(ret);
}
ftLong signed_area_parallelogram(Point2d p1, Point2d p2, Point2d p3){
    return cross(p2 - p1, p3 - p2);
}
double triangle_area(Point2d p1, Point2d p2, Point2d p3){
    return abs(signed_area_parallelogram(p1, p2, p3)) / 2.0;
}
bool pointInTriangle(Point2d a, Point2d b, Point2d c, Point2d p){
    ftLong s1 = abs(cross(b - a, c - a));
    ftLong s2 = abs(cross(a - p, b - p)) + abs(cross(b - p, c - p)) +
        abs(cross(c - p, a - p));
    return eq(s1, s2);
}
bool clockwise(Point2d p1, Point2d p2, Point2d p3){
    return lt(signed_area_parallelogram(p1, p2, p3), 0);
}
bool counter_clockwise(Point2d p1, Point2d p2, Point2d p3){
    return gt(signed_area_parallelogram(p1, p2, p3), 0);
}

```

```

//End Point 2D

//Begin Line
ftLong det(ftype a, ftype b, ftype c, ftype d){
    return a * (ftLong)d - b * (ftLong)c;
}
struct Line{
    ftype a, b, c;
    Line() {}
    Line(ftype a, ftype b, ftype c) : a(a), b(b), c(c){
        normalize();
    }
    Line(Point2d p1, Point2d p2){
        a = p1.y - p2.y;
        b = p2.x - p1.x;
        c = -a * p1.x - b * p1.y;
        normalize();
    }
    void normalize(){
#ifdef POINT_DOUBLE
        ftype z = sqrt(pw2(a) + pw2(b));
#else
        ftype z = __gcd(abs(a), __gcd(abs(b), abs(c)));
#endif
        a /= z;
        b /= z;
        c /= z;
        if (lt(a, 0) or (eq(a, 0) and lt(b, 0))){
            a = -a;
            b = -b;
            c = -c;
        }
    }
};
bool intersect(Line m, Line n, Point2d &res){
    ftype zn = det(m.a, m.b, n.a, n.b);
    if (eq(zn, 0))
        return false;
    res.x = -det(m.c, m.b, n.c, n.b) / zn;
    res.y = -det(m.a, m.c, n.a, n.c) / zn;
    return true;
}
bool parallel(Line m, Line n){
    return eq(det(m.a, m.b, n.a, n.b), 0);
}
bool equivalent(Line m, Line n){
    return eq(det(m.a, m.b, n.a, n.b), 0) &&
        eq(det(m.a, m.c, n.a, n.c), 0) &&
        eq(det(m.b, m.c, n.b, n.c), 0);
}
//Distance from a point(x, y) to a line m
double dist(Line m, ftype x, ftype y){
    return abs(m.a * (ftLong)x + m.b * (ftLong)y + m.c) /
        sqrt(m.a * (ftLong)m.a + m.b * (ftLong)m.b);
}
//End Line

//Begin Segment
struct Segment{
    Point2d a, b;
    Segment() {}

```

```

    Segment(Point2d a, Point2d b) : a(a), b(b) {}
};
bool interld(ftype a, ftype b, ftype c, ftype d){
    if (a > b)
        swap(a, b);
    if (c > d)
        swap(c, d);
    return le(max(a, c), min(b, d));
}
bool check_intersection(Segment s1, Segment s2){
    Point2d a = s1.a, b = s1.b, c = s2.a, d = s2.b;
    if (cross(a - c, d - c) == 0 && cross(b - c, d - c) == 0)
        return interld(a.x, b.x, c.x, d.x) && interld(a.y, b.y, c.y, d.y);
    return sgn(cross(b - a, c - a)) != sgn(cross(b - a, d - a)) &&
        sgn(cross(d - c, a - c)) != sgn(cross(d - c, b - c));
}
inline bool betw(ftype l, ftype r, ftype x){
    return le(min(l, r), x) and le(x, max(l, r));
}
bool intersect(Segment s1, Segment s2, Segment &ans){
    Point2d a = s1.a, b = s1.b, c = s2.a, d = s2.b;
    if (!interld(a.x, b.x, c.x, d.x) || !interld(a.y, b.y, c.y, d.y))
        return false;
    Line m(a, b);
    Line n(c, d);
    if (parallel(m, n)){
        if (!equivalent(m, n))
            return false;
        if (b < a)
            swap(a, b);
        if (d < c)
            swap(c, d);
        ans = Segment(max(a, c), min(b, d));
        return true;
    }else{
        Point2d p(0, 0);
        intersect(m, n, p);
        ans = Segment(p, p);
        return betw(a.x, b.x, p.x) && betw(a.y, b.y, p.y) &&
            betw(c.x, d.x, p.x) && betw(c.y, d.y, p.y);
    }
}
//End Segment

//Begin Circle
struct Circle{
    ftype x, y, r;
    Circle() {}
    Circle(ftype x, ftype y, ftype r) : x(x), y(y), r(r){};
};
bool pointInCircle(Circle c, Point2d p){
    return ge(c.r, dist(Point2d(c.x, c.y), p));
}
//CircumCircle of a triangle is a circle that passes through all the
//vertices
Circle circumCircle(Point2d a, Point2d b, Point2d c){
    Point2d u((b - a).y, -((b - a).x));
    Point2d v((c - a).y, -((c - a).x));
    Point2d n = (c - b) * 0.5;
    double t = cross(u, n) / cross(v, u);
    Point2d ct = (((a + c) * 0.5) + (v * t));

```

```

    double r = dist(ct, a);
    return Circle(ct.x, ct.y, r);
}

//InCircle is the largest circle contained in the triangle
Circle inCircle(Point2d a, Point2d b, Point2d c){
    double m1 = dist(a, b);
    double m2 = dist(a, c);
    double m3 = dist(b, c);
    Point2d ct = ((c * m1) + (b * m2) + a * (m3)) / (m1 + m2 + m3);
    double sp = 0.5 * (m1 + m2 + m3);
    double r = sqrt(sp * (sp - m1) * (sp - m2) * (sp - m3)) / sp;
    return Circle(ct.x, ct.y, r);
}

//Minimum enclosing circle, O(n)
Circle minimumCircle(vector<Point2d> p){
    random_shuffle(p.begin(), p.end());
    Circle c = Circle(p[0].x, p[0].y, 0.0);
    for (int i = 0; i < (int)p.size(); i++){
        if (pointInCircle(c, p[i]))
            continue;
        c = Circle(p[i].x, p[i].y, 0.0);
        for (int j = 0; j < i; j++){
            if (pointInCircle(c, p[j]))
                continue;
            c = Circle((p[j].x + p[i].x) * 0.5, (p[j].y + p[i].y) * 0.5, 0.5 * dist(p[j], p[i]));
            for (int k = 0; k < j; k++){
                if (pointInCircle(c, p[k]))
                    continue;
                c = circumCircle(p[j], p[i], p[k]);
            }
        }
    }
    return c;
}

//Return the number of the intersection
int circle_line_intersection(Circle circ, Line line, Point2d &p1, Point2d &p2){
    ftLong r = circ.r;
    ftLong a = line.a, b = line.b, c = line.c + line.a * circ.x + line.b * circ.y; //take a circle to the (0, 0)
    ftLong x0 = -a * c / (pw2(a) + pw2(b)), y0 = -b * c / (pw2(a) + pw2(b)); // (x0, y0) is the shortest distance point of the line for (0, 0)
    if (gt(pw2(c), pw2(r) * (pw2(a) + pw2(b)))){
        return 0;
    }
    else if (eq(pw2(c), pw2(r) * (pw2(a) + pw2(b)))){
        p1.x = x0 + b * mult + circ.x;
        p1.y = y0 + a * mult + circ.y;
        return 1;
    }
    else{
        ftLong d_2 = pw2(r) - pw2(c) / (pw2(a) + pw2(b));
        ftLong mult = sqrt(d_2 / (pw2(a) + pw2(b)));
        p1.x = x0 + b * mult + circ.x;
        p2.x = x0 - b * mult + circ.x;
        p1.y = y0 - a * mult + circ.y;
        p2.y = y0 + a * mult + circ.y;
        return 2;
    }
}

```

```

//Return the number of the intersection
int circle_intersection(Circle c1, Circle c2, Point2d &p1, Point2d &p2){
    if (eq(c1.x, c2.x) and eq(c1.y, c2.y)){
        if (eq(c1.r, c2.r))
            return -1; //INF
        else
            return 0;
    }
    else{
        Circle circ(0, 0, c1.r);
        Line line;
        line.a = -2 * (c2.x - c1.x);
        line.b = -2 * (c2.y - c1.y);
        line.c = pw2(c2.x - c1.x) + pw2(c2.y - c1.y) + pw2(c1.r) - pw2(c2.r);
        int sz = circle_line_intersection(circ, line, p1, p2);
        p1.x += c1.x;
        p2.x += c1.x;
        p1.y += c1.y;
        p2.y += c1.y;
        return sz;
    }
}

bool checkIfTheSegmentIsCompletelyCoveredByCircles(vector<Circle> &vc, Segment s){
    vector<Point2d> v = {s.a, s.b};
    Line l(s.a, s.b);
    for (Circle c : vc){
        Point2d p1, p2;
        int inter = circle_line_intersection(c, l, p1, p2);
        if (inter >= 1 and betw(s.a.x, s.b.x, p1.x) and betw(s.a.y, s.b.y, p1.y))
            v.push_back(p1);
        if (inter == 2 and betw(s.a.x, s.b.x, p2.x) and betw(s.a.y, s.b.y, p2.y))
            v.push_back(p2);
    }
    sort(v.begin(), v.end());
    bool ans = true;
    for (int i = 1; i < (int)v.size(); i++){
        bool has = false;
        for (Circle c : vc){
            if (pointInCircle(c, v[i - 1]) and pointInCircle(c, v[i])){
                has = true;
                break;
            }
        }
        ans = ans && has;
    }
    return ans;
}

void tangents(Point2d c, double r1, double r2, vector<Line> &ans){
    double r = r2 - r1;
    double z = pw2(c.x) + pw2(c.y);
    double d = z - pw2(r);
    if (lt(d, 0))
        return;
    d = sqrt(abs(d));
    Line l;

```

```

    l.a = (c.x * r + c.y * d) / z;
    l.b = (c.y * r - c.x * d) / z;
    l.c = r1;
    ans.push_back(l);
}
vector<Line> tangents(Circle a, Circle b){
    vector<Line> ans;
    for (int i = -1; i <= 1; i += 2)
        for (int j = -1; j <= 1; j += 2)
            tangents(Point2d(b.x - a.x, b.y - a.y), a.r * i, b.r * j, ans);
    for (size_t i = 0; i < ans.size(); ++i){
        ans[i].c -= ans[i].a * a.x + ans[i].b * a.y;
        ans[i].normalize();
    }
    return ans;
}
//End Circle

```

## 5.2 Circle Area Union

```

#include "basic_geometry.h"
using namespace std;

const double PI = acos(-1);
pair<double, double> isCC(Circle circ1, Circle circ2){
    Point2d c1(circ1.x, circ1.y), c2(circ2.x, circ2.y);
    double r1 = circ1.r, r2 = circ2.r;
    double d = dist(c1, c2);
    double x1 = c1.x, x2 = c2.x, y1 = c1.y, y2 = c2.y;
    double mid = atan2(y2 - y1, x2 - x1);
    double a = r1, c = r2;
    double t = acos((a * a + d * d - c * c) / (2 * a * d));
    return make_pair(mid - t, mid + t);
}

int testCC(Circle circ1, Circle circ2){
    Point2d c1(circ1.x, circ1.y), c2(circ2.x, circ2.y);
    double r1 = circ1.r, r2 = circ2.r;
    double d = dist(c1, c2);
    if (le(r1 + r2, d))
        return 1; // not intersected or tged
    if (le(r1 + d, r2))
        return 2; // C1 inside C2
    if (le(r2 + d, r1))
        return 3; // C2 inside C1
    return 0; // intersected
}

struct event_t{
    double theta;
    int delta;
    event_t(double t, int d) : theta(t), delta(d) {}
    bool operator<(const event_t &r) const{
        if (fabs(theta - r.theta) < EPS)
            return delta > r.delta;
        return theta < r.theta;
    }
};

vector<event_t> e;
void add(double begin, double end){
    if (begin <= -PI)
        begin += 2 * PI, end += 2 * PI;
}

```

```

if (end > PI){
    e.push_back(event_t(begin, 1));
    e.push_back(event_t(PI, -1));
    e.push_back(event_t(-PI, 1));
    e.push_back(event_t(end - 2 * PI, -1));
}
else{
    e.push_back(event_t(begin, 1));
    e.push_back(event_t(end, -1));
}
}

double calc(Point2d c, double r, double a1, double a2){
    double da = a2 - a1;
    double aa = r * r * (da - sin(da)) / 2;
    Point2d p1 = Point2d(cos(a1), sin(a1)) * r + c;
    Point2d p2 = Point2d(cos(a2), sin(a2)) * r + c;
    return cross(p1, p2) / 2 + aa;
}

/* O(n^2 log n), please remove coincided circles first. */
double circle_union(vector<Circle> &vc){
    int n = vc.size();
    for (int i = n - 1; i >= 0; i--){
        if (eq(vc[i].r, 0)){
            swap(vc[i], vc[n - 1]);
            n--;
            continue;
        }
        for (int j = 0; j < i; j++){
            if (eq(vc[i].x, vc[j].x) and eq(vc[i].y, vc[j].y) and eq(vc[i].r,
                , vc[j].r)){
                swap(vc[i], vc[n - 1]);
                n--;
            }
        }
    }
    if (n == 0)
        return 0;
    vc.resize(n);
    vector<double> cntarea(2 * n, 0);
    for (int c = 0; c < n; c++){
        int cvrcnt = 0;
        e.clear();
        for (int i = 0; i < n; i++){
            if (i != c){
                int r = testCC(vc[c], vc[i]);
                if (r == 2){
                    cvrcnt++;
                } else if (r == 0){
                    auto paa = isCC(vc[c], vc[i]);
                    add(paa.first, paa.second);
                }
            }
        }
        if (e.size() == 0){
            double a = PI * vc[c].r * vc[c].r;
            cntarea[cvrcnt] += a;
            cntarea[cvrcnt + 1] += a;
        } else {
            e.push_back(event_t(-PI, 1));
            e.push_back(event_t(PI, -2));
            sort(e.begin(), e.end());
            for (int i = 0; i < int(e.size()) - 1; i++){

```

```

    cvrcnt += e[i].delta;
    double a = calc(Point2d(vc[c].x, vc[c].y), vc[c].r, e[i].theta
        , e[i + 1].theta);
    cntarea[cvrcnt - 1] -= a;
    cntarea[cvrcnt] += a;
}
}
double ans = 0;
for(int i=1; i<=n; i++)
    ans += cntarea[i];
return ans;
}

```

## 5.3 Count Lattices

```

#include "../code/math/fraction.h"
Fraction f_1 = 1;
//Calculates number of integer points (x,y) such for 0<=x<n and 0<y<=
    piso(k*x+b)
//O(log(N)*log(MAXV))
f_type count_lattices(Fraction k, Fraction b, f_type n) {
    auto fk = (f_type)k;
    auto fb = (f_type)b;
    auto cnt = 0LL;

    if (k >= f_1 || b >= f_1) {
        cnt += (fk * (n - 1) + 2 * fb) * n / 2;
        k = k - Fraction(fk, 1);
        b = b - Fraction(fb, 1);
    }
    auto t = k * Fraction(n, 1) + b;
    auto ft = (f_type)t;
    if (ft >= 1) {
        cnt += count_lattices(f_1 / k, (t - Fraction((f_type)t, 1)) / k, (
            f_type)t);
    }
    return cnt;
}

```

## 5.4 Point 3D

```

#include <bits/stdc++.h>
using namespace std;
//#define POINT_DOUBLE
#ifdef POINT_DOUBLE
    typedef double ftype;
    typedef long double ftLong;
    const double EPS = 1e-9;
    #define eq(a, b) (abs(a-b)<EPS)
    #define lt(a, b) ((a+EPS)<b)
    #define gt(a, b) (a>(b+EPS))
    #define le(a, b) (a<(b+EPS))
    #define ge(a, b) ((a+EPS)>b)
#else
    typedef int32_t ftype;
    typedef int64_t ftLong;
    #define eq(a, b) (a==b)

```

```

#define lt(a, b) (a<b)
#define gt(a, b) (a>b)
#define le(a, b) (a<=b)
#define ge(a, b) (a>=b)
#endif
//Point3D
struct Point3d{
    ftype x, y, z;
    Point3d() {}
    Point3d(ftype x, ftype y, ftype z) : x(x), y(y), z(z) {}
    Point3d operator+(Point3d t){
        return Point3d(x + t.x, y + t.y, z + t.z);
    }
    Point3d operator-(Point3d t){
        return Point3d(x - t.x, y - t.y, z - t.z);
    }
    Point3d operator*(ftype t){
        return Point3d(x * t, y * t, z * t);
    }
    Point3d operator/(ftype t){
        return Point3d(x / t, y / t, z / t);
    }
};
ftLong dot(Point3d a, Point3d b){
    return a.x * (ftLong)b.x + a.y * (ftLong)b.y + a.z * (ftLong)b.z;
}
double len(Point3d a){
    return sqrt(dot(a, a));
}
double dist(Point3d a, Point3d b){
    return len(a-b);
}
double proj(Point3d a, Point3d b){
    return dot(a, b) / len(b);
}
double angle(Point3d a, Point3d b){
    return acos(dot(a, b) / len(a) / len(b));
}
Point3d cross(Point3d a, Point3d b){
    return Point3d(a.y * b.z - a.z * b.y,
        a.z * b.x - a.x * b.z,
        a.x * b.y - a.y * b.x);
}
ftLong triple(Point3d a, Point3d b, Point3d c){
    return dot(a, cross(b, c));
}
Point3d planeIntersect(Point3d a1, Point3d n1, Point3d a2, Point3d n2,
    Point3d a3, Point3d n3){
    Point3d x(n1.x, n2.x, n3.x);
    Point3d y(n1.y, n2.y, n3.y);
    Point3d z(n1.z, n2.z, n3.z);
    Point3d d(dot(a1, n1), dot(a2, n2), dot(a3, n3));
    return Point3d(triple(d, y, z),
        triple(x, d, z),
        triple(x, y, d)) / triple(n1, n2, n3);
}
struct Sphere{
    ftype x, y, z, r;
    Sphere(){}
    Sphere(ftype x, ftype y, ftype z, ftype r):x(x), y(y), z(z), r(r){}
};

```

```
//Minimum enclosing Sphere, O(n*70000)
//It is also possible to do with ternary search in the 3 dimensions
Sphere minimumSphere(vector<Point3d> vp){
    Point3d ans(0, 0, 0);
    int n = vp.size();
    for(Point3d p: vp)
        ans = ans + p;
    ans = ans/n;
    double P = 0.1;
    double d = 0, e = 0;
    for(int i = 0; i < 70000; i++){
        int f = 0;
        d = dist(ans, vp[0]);
        for (int j = 1; j < n; j++) {
            e = dist(ans, vp[j]);
            if (d < e) {
                d = e;
                f = j;
            }
        }
        ans = ans + (vp[f]-ans)*P;
        P *= 0.998;
    }
    return Sphere(ans.x, ans.y, ans.z, d);
}
```

## 6 String Algorithms

### 6.1 Min Cyclic String

```
#include <bits/stdc++.h>
using namespace std;
string min_cyclic_string(string s){
    s += s;
    int n = s.size();
    int i = 0, ans = 0;
    while (i < n / 2){
        ans = i;
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]){
            if (s[k] < s[j])
                k = i;
            else
                k++;
            j++;
        }
        while (i <= k)
            i += j - k;
    }
    return s.substr(ans, n / 2);
}
```

### 6.2 Suffix Automaton

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

```
struct SuffixAutomaton{
    struct state{
        int len, link, first_pos;
        bool is_clone = false;
        map<char, int> next;
    };
    vector<state> st;
    int sz, last;
    SuffixAutomaton(string s){
        st.resize(2 * s.size() + 10);
        st[0].len = 0;
        st[0].link = -1;
        st[0].is_clone = false;
        sz = 1;
        last = 0;
        for (char c : s)
            insert(c);
        preCompute();
    }
    void insert(char c){
        int cur = sz++;
        st[cur].len = st[last].len + 1;
        st[cur].first_pos = st[cur].len - 1;
        st[cur].is_clone = false;
        int p = last;
        while (p != -1 && !st[p].next.count(c)){
            st[p].next[c] = cur;
            p = st[p].link;
        }
        if (p == -1){
            st[cur].link = 0;
        }
        else{
            int q = st[p].next[c];
            if (st[p].len + 1 == st[q].len){
                st[cur].link = q;
            }
            else{
                int clone = sz++;
                st[clone].len = st[p].len + 1;
                st[clone].next = st[q].next;
                st[clone].link = st[q].link;
                st[clone].first_pos = st[q].first_pos;
                st[clone].is_clone = true;
                while (p != -1 && st[p].next[c] == q){
                    st[p].next[c] = clone;
                    p = st[p].link;
                }
                st[q].link = st[cur].link = clone;
            }
        }
        last = cur;
    }
    string lcs(string s){
        int v = 0, l = 0, best = 0, bestpos = 0;
        for (int i = 0; i < (int)s.size(); i++){
            while (v && !st[v].next.count(s[i])){
                v = st[v].link;
                l = st[v].len;
            }
            if (st[v].next.count(s[i])){
                v = st[v].next[s[i]];
                l++;
            }
        }
    }
}
```

```

    }
    if (l > best){
        best = l;
        bestpos = i;
    }
}
return s.substr(bestpos - best + 1, best);
}
vector<ll> dp;
vector<int> cnt;
ll dfsPre(int s){
    if (dp[s] != -1)
        return dp[s];
    dp[s] = cnt[s]; //Accepts repeated substrings
    //dp[s] = 1; //Does not accept repeated substrings
    for (auto p : st[s].next)
        dp[s] += dfsPre(p.second);
    return dp[s];
}
void preCompute(){
    cnt.assign(sz, 0);
    vector<pair<int, int>> v(sz);
    for (int i = 0; i < sz; i++){
        cnt[i] = !st[i].is_clone;
        v[i] = make_pair(st[i].len, i);
    }
    sort(v.begin(), v.end(), greater<pair<int, int>>());
    for (int i = 0; i < sz - 1; i++){
        cnt[st[v[i].second].link] += cnt[v[i].second];
    }
    dp.assign(sz, -1);
    dfsPre(0);
}
};

```

## 7 Miscellaneous

### 7.1 Longest Increasing Subsequence

```

#include <bits/stdc++.h>
using namespace std;
vector<int> lis(vector<int> &v){
    vector<int> st, ans;
    vector<int> pos(v.size()+1), dad(v.size()+1);
    for(int i=0; i < (int)v.size(); i++){
        auto it = lower_bound(st.begin(), st.end(), v[i]); // Do not
        accept repeated values
        //auto it = upper_bound(st.begin(), st.end(), v[i]); //Accept
        repeated values
        int p = it-st.begin();
        if(it==st.end())
            st.push_back(v[i]);
        else
            *it = v[i];
        pos[p] = i;
        dad[i] = (p==0)? -1 : pos[p-1];
    }
    int p = pos[st.size() - 1];
    while(p >= 0){

```

```

        ans.push_back(v[p]);
        p=dad[p];
    }
    reverse(ans.begin(), ans.end());
    return ans;
}

```

### 7.2 Mo Algorithm

```

#include <bits/stdc++.h>
using namespace std;
const int BLOCK_SIZE = 700;
void remove(int idx);
void add(int idx);
void clearAnswer();
int getAnswer();
struct Query{
    int l, r, idx;
    bool operator<(Query other) const{
        if (l / BLOCK_SIZE != other.l / BLOCK_SIZE)
            return l < other.l;
        return (l / BLOCK_SIZE & 1) ? (r < other.r) : (r > other.r);
    }
};
vector<int> mo_s_algorithm(vector<Query> queries){
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());
    clearAnswer();
    int L = 0, R = 0;
    add(0);
    for(Query q : queries){
        while(q.l < L) add(--L);
        while(R < q.r) add(++R);
        while(L < q.l) remove(L--);
        while(q.r < R) remove(R--);
        answers[q.idx] = getAnswer();
    }
    return answers;
}

```

## 8 Theorems and Formulas

### 8.1 Binomial Coefficients

$$(a+b)^n = \binom{n}{0}a^n + \binom{n}{1}a^{n-1}b + \binom{n}{2}a^{n-2}b^2 + \dots + \binom{n}{k}a^{n-k}b^k + \dots + \binom{n}{n}b^n$$

$$\text{Pascal's Triangle: } \binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

$$\text{Symmetry rule: } \binom{n}{k} = \binom{n}{n-k}$$

$$\text{Factoring in: } \binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}$$

$$\text{Sum over } k: \sum_{k=0}^n \binom{n}{k} = 2^n$$

$$\text{Sum over } n: \sum_{m=0}^n \binom{m}{k} = \binom{n+1}{k+1}$$

$$\text{Sum over } n \text{ and } k: \sum_{k=0}^m \binom{n+k}{k} = \binom{n+m+1}{m}$$

$$\text{Sum of the squares: } \binom{n}{0}^2 + \binom{n}{1}^2 + \dots + \binom{n}{n}^2 = \binom{2n}{n}$$

$$\text{Weighted sum: } 1\binom{n}{1} + 2\binom{n}{2} + \dots + n\binom{n}{n} = n2^{n-1}$$



Connection with the Fibonacci numbers:  $\binom{n}{0} + \binom{n-1}{1} + \dots + \binom{n-k}{k} + \dots + \binom{0}{n} = F_{n+1}$

More formulas:  $\sum_{k=0}^m (-1)^k \cdot \binom{n}{k} = (-1)^m \cdot \binom{n-1}{m}$  \_\_\_\_\_

## 8.2 Catalan Number

Recursive formula:  $C_0 = C_1 = 1$

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}, n \geq 2$$

$$\text{Analytical formula: } C_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{1}{n+1} \binom{2n}{n}, n \geq 0$$

The first few numbers Catalan numbers,  $C_n$  (starting from zero):  
1, 1, 2, 5, 14, 42, 132, 429, 1430, ...

The Catalan number  $C_n$  is the solution for:

- Number of correct bracket sequence consisting of  $n$  opening and  $n$  closing brackets.
- The number of rooted full binary trees with  $n + 1$  leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.
- The number of ways to completely parenthesize  $n + 1$  factors.
- The number of triangulations of a convex polygon with  $n + 2$  sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
- The number of ways to connect the  $2n$  points on a circle to form  $n$  disjoint chords.
- The number of non-isomorphic full binary trees with  $n$  internal nodes (i.e. nodes having at least one son).
- The number of monotonic lattice paths from point  $(0, 0)$  to point  $(n, n)$  in a square lattice of size  $n \times n$ , which do not pass above the main diagonal (i.e. connecting  $(0, 0)$  to  $(n, n)$ ).
- Number of permutations of length  $n$  that can be stack sorted (i.e. it can be shown that the rearrangement is stack sorted if and only if there is no such index  $i < j < k$ , such that  $a_k < a_i < a_j$ ).
- The number of non-crossing partitions of a set of  $n$  elements.
- The number of ways to cover the ladder  $1 \dots n$  using  $n$  rectangles (The ladder consists of  $n$  columns, where  $i^{th}$  column has a height  $i$ ).

## 8.3 Euler's Totient

If  $p$  is a prime number:  $\phi(p) = p - 1$  and  $\phi(p^k) = p^k - p^{k-1}$

If  $a$  and  $b$  are relatively prime, then:  $\phi(ab) = \phi(a) \cdot \phi(b)$

$$\text{In general: } \phi(ab) = \phi(a) \cdot \phi(b) \cdot \frac{gcd(a, b)}{\phi(gcd(a, b))}$$

This interesting property was established by Gauss:  $\sum_{d|n} \phi(d) = n$ , Here the sum is over all positive divisors  $d$  of  $n$ .

Euler's theorem:  $a^{\phi(m)} \equiv 1 \pmod{m}$ , if  $a$  and  $m$  are relatively prime.

Generalization:  $a^n \equiv a^{\phi(m) + [n \bmod \phi(m)]} \pmod{m}$ , for arbitrary  $a$ ,  $m$  and  $n$   
 $\geq \log_2(m)$ . \_\_\_\_\_

## 8.4 Formulas

Count the number of ways to partition a set of  $n$  labelled objects into  $k$  nonempty labelled subsets.

$$f(n, k) = \sum_{i=0}^k (-1)^i \binom{k}{i} (k-i)^n$$

Stirling Number 2nd: Partitions of an  $n$  element set into  $k$  not-empty set. Or count the number of ways to partition a set of  $n$  labelled objects into  $k$  nonempty unlabelled subsets.

$$S_{2nd}(n, k) = \left\{ \begin{matrix} n \\ k \end{matrix} \right\} = \frac{1}{k!} \sum_{i=0}^k (-1)^i \binom{k}{i} (k-i)^n$$

Euler's formula:  $f = e - v + 2$

Euler's formula to  $n$  Lines or Segment if there is no three lines/segments that contains the same point:  $R = \text{intersects} + \text{component} - n$

Number of regions in a planar graph:  $R = E - V + C + 1$  where  $C$  is the number of connected components

Given  $a$  and  $b$  co-prime,  $n = a \cdot x + b \cdot y$  where  $x \geq 0$  and  $y \geq 0$ . You are required to find the least value of  $n$ , such that all currency values greater than or equal to  $n$  can be made using any number of coins of denomination  $a$  and  $b$ :  
 $n = (a - 1) * (b - 1)$

generalization of the above problem,  $n$  is multiple of  $gcd(a, b)$ :  $n = lcm(a, b) - a - b + gcd(a, b)$  \_\_\_\_\_

## 8.5 Manhattan Distance

Transformation of the manhattan distance to 2 dimensions between  $P_1 = (x_1, y_1)$  and  $P_2 = (x_2, y_2)$ :

$$|x_1 - x_2| + |y_1 - y_2| = \max(|A_1 - B_1|, |A_2 - B_2|) \text{ where } A = (x_1 + y_1, x_1 - y_1) \text{ e } B = (x_2 + y_2, x_2 - y_2)$$

Transformation of the manhattan distance to 3 dimensions between  $P_1 = (x_1, y_1, z_1)$  and  $P_2 = (x_2, y_2, z_2)$ :

$|x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2| = \max(|A_1 - B_1|, |A_2 - B_2|, |A_3 - B_3|, |A_4 - B_4|)$   
 where  $A = (x_1 + y_1 + z_1, x_1 + y_1 - z_1, x_1 - y_1 + z_1, -x_1 + y_1 + z_1)$  e  $B = (x_2 + y_2 + z_2, x_2 + y_2 - z_2, x_2 - y_2 + z_2, -x_2 + y_2 + z_2)$

Transformation of the manhattan distance to D dimensions between  $P_1$  and  $P_2$ :

$isSet(i, x) = 1$  if the i-th bit is setted in  $x$  and 0 otherwise.

$$A[i] = \sum_{j=0}^{d-1} (-1)^{isSet(j,i)} P_1[j]$$

$$B[i] = \sum_{j=0}^{d-1} (-1)^{isSet(j,i)} P_2[j]$$

$$\sum_{i=0}^{d-1} |P_1[i] - P_2[i]| = \max_{i=0}^{2^d-1} |A_i - B_i|$$

## 8.6 Primes

If  $n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$ , then:

Number of divisors is  $d(n) = (e_1 + 1) \cdot (e_2 + 1) \cdots (e_k + 1)$ .

Sum of divisors is  $\sigma(n) = \frac{p_1^{e_1+1}-1}{p_1-1} \cdot \frac{p_2^{e_2+1}-1}{p_2-1} \cdots \frac{p_k^{e_k+1}-1}{p_k-1}$