Harbour.Space University notebook : x^3

Compiled on April 7, 2023

Contents

1 Contest

1.1 template

```
#include <bits/stdc++.h>
using namespace std;
#pragma GCC optimize("O3","Ofast","no-stack-
protector","unroll-loops","omit-frame-pointer","inline")
//Optimization flags
#pragma GCC
option("arch=native","tune=native","no-zero-upper") //Enable
AVX
#pragma GCC target("avx2") //Enable AVX
#define db(x) cerr << #x << ": " << x << '\n';
#define dbv(v) cerr << #v << ": ["; for(auto i:v) cerr << i <<
' ';  cerr << "]\n";
#define dbp(p) cerr << #p << ": (" << p.first << ", " <<
p.second << ") ";
#define all(x) (x).begin(), (x).end()
using ll = long long;
mt19937_64
rng(chrono::steady_clock::now().time_since_epoch().count());
int main() {
    ios_base::sync_with_stdio(0); cin.tie(0);
    cout << setprecision(9) << fixed;
    // clock() >= 2.5 * CLOCKS_PER_SEC
    return 0;
}
```

2 Combinatorial

2.1 K - PIE

```
//Count permutations with at least K fixed points:
//comb(n,k)*(n-k)! * f(k,k) +
//comb(n,k+1)*(n-(k+1))! * f(k+1,k) +
//comb(n,k+1)*(n-(k+2))! * f(k+2,k) ...
ll intersection_value(ll N, ll K) {
    if(N < K)return 0;
    if(K == 0) {
        if(N == 0)return 1;
        return 0;
    }
    if((N-K)&1)return mod(-comb(N-1,K-1));
    return comb(N-1,K-1);
}
```

2.2 Extended Binomial

```
// Solve sum 0 <= i <= n comb(n, i) * x^n * i^k
// if x = mod-1 then when N > K the ans is always 0
// otherwise just try bruteforce
ll extended_binomial(ll n, ll x, int k) {
    //x != (MOD-1)
    x = mod(x);
    if(x == 0 && k == 0)return 1;
    x = (inv(x)+1ll)%MOD;
    ll val = inv(x), res = 0, pw = 1;
    for(int i = 0 ; i <= k ; i++) {
        res = (res + stirling2_dp(k,i)*pw)%MOD;
        pw = (pw*(n-i)%MOD)*val%MOD;
    }
    res = res*qpow(x*inv((x+MOD-1)%MOD)%MOD,n)%MOD;
    return res;
}
```

3 Data Structures

3.1 Venice Trick

```
template<typename T>
struct VENICE_SET {
    multiset<T> st;
    T acum = 0;
    void insert(T x) {
        st.insert(x - acum);
    }
    void erase(T x) {
        st.erase(st.find(x - acum));
    }
    void update(T x) {
        acum += x;
    }
    T getmin() {
        assert(st.size());
        return *st.begin() + acum;
    }
    T getmax() {
        assert(st.size());
        return *(--st.end()) + acum;
    }
    int size() {
        return st.size();
    }
};
```

3.2 Monotonic Queue

```
struct MONOTONIC_QUEUE {
    deque<pair<T,int>> D;
    int sz = 0, pt = 0;
    void insert(T x) {
        sz++;
        while(!D.empty() &&
!F()(D.back().first,x))D.pop_back();
        D.push_back({x,pt++});
    }
    void erase() {
        assert(sz); sz--;
        if(pt-D.front().second > sz)D.pop_front();
    }
    T get() {
        assert(sz);
        return D.front().first;
    }
};
```

3.3 Monotonic Stack

```
struct MONOTONIC_STACK {
    stack<pair<T,int>> S;
    int sz = 0;
    void insert(T x) {
        sz++;
        if(S.empty() || F()(x,S.top().first))S.push({x,sz});
    }
    void erase() {
        assert(sz);
        sz--;
        if(S.top().second > sz)S.pop();
    }
    int get() {
        assert(sz);
        return S.top().first;
    }
};
```

3.4 Monotonic Stack - Hard

```
template<typename T, class F = less<T>, class G = less<T>>
struct MONOTONIC_STACK {
    const T oo = (G()(-4e18,4e18) ? 4e18 : -4e18);
    vector<pair<int,T>> S;
    vector<int> res;
    vector<T> vect;
    void insert(T x) {
        S.push_back({vect.size(),oo});
        vect.push_back(x);
        if(res.size()) {
            if(F()(vect[S[res.back()].first] +
S[res.back()].second, vect[S.back().first]
             + S.back().second))res.push_back(res.back());
            else res.push_back(res.size());
        } else res.push_back(res.size());
    }
    void update(T x) {
        if(vect.size() == 0)return;
        int best = -1;
        while(S.size() && G()(x,S.back().second)) {
            if(best == -1) best = S.back().first;
            else {
                if(F()(vect[S.back().first], vect[best]))best =
S.back().first;
            }
            S.pop_back();
            res.pop_back();
        }
        if(best == -1)return;
        S.push_back({best,x});
        if(res.size()) {
            if(F()(vect[S[res.back()].first] +
S[res.back()].second, vect[S.back().first]
             + S.back().second))res.push_back(res.back());
            else res.push_back(res.size());
        } else res.push_back(res.size());
    }
    T get() {
        assert(vect.size());
        return vect[S[res.back()].first] +
S[res.back()].second;
    }
};
```

3.5 DSU on trees

```
void DSU_ON_TREE_ADD(int);
void DSU_ON_TREE_DELETE(int);
void DSU_ON_TREE_QUERY(int);
void DSU_ON_TREE(vector<vector<int>> &L, int root = 0) {
    int N = L.size();
    vector<int> sz(N), st(N), ft(N), ver(N);
    int contg = -1;
    function<void(int,int)> precal = [&](int nodo, int padre) {
        ver[++contg] = nodo;
        st[nodo] = contg;
        sz[nodo] = 1;
        for(auto v : L[nodo]) {
            if(v != padre) {
                precal(v,nodo);
                sz[nodo] += sz[v];
            }
        }
        ft[nodo] = contg;
    };
    precal(root,-1);
    function<void(int,int,bool)> DSUT = [&](int nodo, int
padre, bool keep) {
        int may = -1, big = -1;
        for(auto v : L[nodo])
            if(v != padre && sz[v] > may)
                may = sz[v], big = v;
        for(auto v : L[nodo])
            if(v != padre && v != big)
                DSUT(v,nodo,0);
        if(big != -1)DSUT(big,nodo,1);
        for(auto v : L[nodo])
            if(v != padre && v != big)
                for(int i = st[v] ; i <= ft[v] ; i++)
                    DSU_ON_TREE_ADD(ver[i]);
        DSU_ON_TREE_ADD(nodo);
        DSU_ON_TREE_QUERY(nodo);
        if(!keep)
            for(int i = st[nodo] ; i <= ft[nodo] ; i++)
                DSU_ON_TREE_DELETE(ver[i]);
    };
    DSUT(root,-1,false);
}
```

3.6 Sparse Table 2D

```
template<typename T, class F = less<T>>
struct RMQ_2D {
    vector<vector<vector<vector<T>>>> A;
    RMQ_2D(const vector<vector<T>> &arr) {
        if(!arr.size() || !arr[0].size())return;
        int N = arr.size(), M = arr[0].size();
        A.resize(N);
        for(int i = 0 ; i < A.size() ; i++) {
            A[i].resize(31-__builtin_popcount(N)+1);
            for(int j = 0 ; j < A[i].size() ; j++) {
                A[i][j].resize(M);
                for(int k = 0 ; k < A[i][j].size() ; k++)
A[i][j][k].resize(31-__builtin_popcount(M)+1);
            }
        }
        for(int i = 0 ; i < N ; i++) {
            for(int j = 0 ; j < M ; j++)A[i][0][j][0] =
arr[i][j];
            for(int y = 1 ; y <= 31-__builtin_popcount(M) ;
y++)
                for(int j = 0 ; j+(1<<y)-1 < M ; j++)

if(F()(A[i][0][j][y-1],A[i][0][j+(1<<(y-1))][y-1]))
                        A[i][0][j][y] = A[i][0][j][y-1];
                    else A[i][0][j][y] =
A[i][0][j+(1<<(y-1))][y-1];
        }
        for(int x = 1 ; x <= 31-__builtin_popcount(N) ; x++)
            for(int i = 0 ; i+(1<<x)-1 < N ; i++)
                for(int y = 0 ; y <= 31-__builtin_popcount(M);
y++)
                    for(int j = 0 ; j+(1<<y)-1 < M ; j++)

if(F()(A[i][x-1][j][y],A[i+(1<<(x-1))][x-1][j][y]))
                            A[i][x][j][x] = A[i][x-1][j][y];
                        else A[i][x][j][y] =
A[i+(1<<(x-1))][x-1][j][y];

    }
    T query(int x1, int y1, int x2, int y2) {
        if(x1 > x2)swap(x1,x2);
        if(y1 > y2)swap(y1,y2);
        int kx = 31-__builtin_popcount(x2-x1+1);
        int ky = 31-__builtin_popcount(y2-y1+1);
        T r1, r2;

if(F()(A[x1][kx][y1][ky],A[x1][kx][y2-(1<<ky)+1][ky]))r1 =
A[x1][kx][y1][ky];
        else r1 = A[x1][kx][y2-(1<<ky)+1][ky];
        if(F()(A[x2-(1<<kx)+1][kx][y1][ky],A[x2-(1<<kx)+1][kx]
[y2-(1<<ky)+1][ky]))
            r2 = A[x2-(1<<kx)+1][kx][y1][ky];
        else r2 = A[x2-(1<<kx)+1][kx][y2-(1<<ky)+1][ky];
        if(F()(r1,r2))return r1;
        else return r2;
    }
};
```

3.7 DSU Rollback

```
struct DSU_WITH_ROLLBACKS {
    struct dsu_save {
        int u, urank, v, vrank, comps;
        bool bipart, bipart_comp;
    };
    stack<dsu_save> op; vector<int> rnk;
    vector<bool> col, bip; bool bipartite; int comps;
    DSU_WITH_ROLLBACKS() {}
    DSU_WITH_ROLLBACKS(int N) { init(N); }
    void init(int N) {
        rnk.clear(), col.clear(), bip.clear();
        while (!op.empty())
            op.pop();
        rnk.resize(N), col.resize(N), bip.resize(N);
        for (int i = 0; i < N; i++) {
            rnk[i] = 1, col[i] = false, bip[i] = true;
        }
        bipartite = true, comps = N;
    }
    pair<int, bool> find(int x) {
        if (rnk[x] > 0) return {x, col[x]};
        pair<int, bool> tp = find(-rnk[x]);
        tp.second ^= col[x];
        return tp;
    }
    void join(int a, int b) {
        pair<int, bool> ta = find(a), tb = find(b);
        a = ta.first, b = tb.first;
        if (a == b) {
            op.push({a, rnk[a], b, rnk[b], comps, bipartite,
bip[a]});
            if (ta.second == tb.second)
                bipartite = false, bip[a] = false;
            return;
        }
        if (rnk[a] < rnk[b])
            swap(a, b);
        op.push({a, rnk[a], b, rnk[b], comps, bipartite,
bip[a]});
        comps--;
        rnk[a] += rnk[b];
        rnk[b] = -a;
        col[b] = col[b] ^ (ta.second == tb.second);
        bip[a] = bip[a] & bip[b];
    }
    void rollback() {
        if (op.empty()) return;
        dsu_save x = op.top(); op.pop();
        comps = x.comps, bipartite = x.bipart;
        rnk[x.u] = x.urank, col[x.u] = false, bip[x.u] =
x.bipart_comp;
        rnk[x.v] = x.vrank, col[x.v] = false;
    }
};
```

3.8 Fenwick Tree 2D

```
//Complexity (O(nlog^2)) first call fakeUpdate, prepare, and
//then start with real updates and queries. Computes sums
//a[i,j] for all i<I, j<J, and increases single elements
a[i,j]
struct FT {
    vector<ll> ft;
    FT () {}
    FT(int n) : ft(n + 1) {}
    void update(int id, ll val){
        for (id ++; id < ft.size(); id += (id&-id)) ft[id] += val;
    }
    ll query(int id) { if (id < 0) return 0;
        ll res = 0;
        for (id ++; id > 0; id -= (id&-id))
            res += ft[id];
        return res; }
};
struct FT2D {
    vector<FT> fts; vector<vector<int>> ys; int szx;
    FT2D (int _szx) : szx(_szx + 1), ys(_szx + 1), fts(_szx + 1) {}
    void fakeUpdate(int x,int y) {
        for (x ++; x < szx; x += (x&-x))
            ys[x].push_back(y);
    }
    int index(int x,int y) {
        return lower_bound(ys[x].begin(), ys[x].end(), y) - ys[x].begin();
    }
    int index2(int x,int y) {
        return upper_bound(ys[x].begin(), ys[x].end(), y) - ys[x].begin();
    }
    void prepare () { int count = 0;
        for (auto& vy : ys) {
            sort(vy.begin(), vy.end());
            fts[count++].ft.resize(vy.size() + 2);
        }
    }
    void update (int x,int y, ll val) {
        for (x ++; x < szx; x += (x&-x))
            fts[x].update(index(x, y), val);
    }
    ll query(int x,int y) { ll res = 0;
        for (x ++; x > 0; x -= (x&-x))
            res += fts[x].query(index2(x, y) - 1);
        return res;
    }
};
```

3.9 Implicit Treap

```
struct Node {
    int prio, sz;
    Node *l, *r;
    ll val, sum;
    ll lazy;
    Node (ll _val) : l(NULL), r(NULL), sz(1),
        prio(rng()), val(_val), lazy(0), sum(_val) {}
};
struct Treap {
    inline int size(Node *u) { return u ? u->sz : 0; }
    inline void push(Node *u) {
        if (u && u->lazy) {
            // How u->lazy affects u
            u->val += u->lazy;
            u->sum += u->lazy * u->sz;
            if (u->l) u->l->lazy += u->lazy;
            if (u->r) u->r->lazy += u->lazy;
            u->lazy = 0;
        }
    }
    Node* update(Node *u){
        // Recompute u based on merging its children
        u->sum = u->val, u->sz = 1;
        if (u->l) u->sum += u->l->sum, u->sz += u->l->sz;
        if (u->r) u->sum += u->r->sum, u->sz += u->r->sz;
        return u;
    }
    pair<Node*, Node*> split(Node* u, int k){
        if (!u) return { u, u };
        push(u);
        if (size(u->l) >= k){
            auto s = split(u->l, k);
            u->l = s.second;
            return { s.first, update(u) };
        }
        auto s = split(u->r, k - size(u->l) - 1);
        u->r = s.first;
        return { update(u), s.second };
    }
    Node* merge(Node *u, Node *v){
        if (!u || !v) return u ? u : v;
        if (u->prio > v->prio){
            push(u);
            u->r = merge(u->r, v);
            return update(u);
        }
        push(v);
        v->l = merge(u, v->l);
        return update(v);
    }
    Node* kth(Node *u, int k) {
        while (u && size(u->l) + 1 != k){
            push(u);
            if (size(u->l) >= k) u = u->l;
            else k -= size(u->l) + 1, u = u->r;
        }
        push(u);
        return u;
    }
};
```

3.10 LiChao Tree

```
template<typename T, class F = less <T>>
struct LiChaoTree {
    bool sz = true;
    struct node {
        T func; node *L = NULL, *R = NULL;
    } ST;
    ll Li, Ri;
    LiChaoTree(ll a, ll b) { Li = a, Ri = b; }
    void add_function(T nw) {
        if (sz)ST.func = nw, sz = false;
        else li_chao_tree_update(nw, &ST, Li, Ri);
    }
    T get(ll x) { assert(!sz);
        return li_chao_tree_query(x, &ST, Li, Ri);
    }
    void li_chao_tree_update(T nw, node *pt, ll l, ll r) {
        ll m = (l + r) / 2;
        bool lef = F()(nw.f(l), pt->func.f(l));
        bool mid = F()(nw.f(m), pt->func.f(m));
        if (mid)swap(nw, pt->func);
        if (l == r) return;
        if (lef != mid) {
            if (pt->L == NULL)pt->L = new node(), pt->L->func = nw;
            else li_chao_tree_update(nw, pt->L, l, m);
        } else {
            if (pt->R == NULL)pt->R = new node(), pt->R->func = nw;
            else li_chao_tree_update(nw, pt->R, m + 1, r);
        }
    }

    T li_chao_tree_query(ll x, node *pt, ll l, ll r) {
        if (l == r)return pt->func;
        ll m = (l + r) / 2;
        T tp = pt->func;
        if (x <= m && pt->L != NULL) {
            T qu = li_chao_tree_query(x, pt->L, l, m);
            if (F()(qu.f(x), tp.f(x)))tp = qu;
        } else if (pt->R != NULL) {
            T qu = li_chao_tree_query(x, pt->R, m + 1, r);
            if (F()(qu.f(x), tp.f(x)))tp = qu;
        }
        return tp;
    }
};
struct line {
    ll m, n;
    ll f(ll x) { return m * x + n; }
};
```

3.11 Link Cut Tree

```
// Usage: FOR(i,1,N)LCT[i]=new snode(i);
typedef struct snode* sn;
struct snode { //////// VARIABLES
  sn p, c[2]; // parent, children
  sn extra; // extra cycle node for "The Applicant"
  bool flip = 0; // subtree flipped or not
  int val, sz; // value in node, # nodes in current splay tree
  int sub, vsub = 0; // vsub stores sum of virtual children
  snode(int _val) : val(_val) {
    p = c[0] = c[1] = extra = NULL; calc(); }
  friend int getSz(sn x) { return x?x->sz:0; }
  friend int getSub(sn x) { return x?x->sub:0; }
  void prop() { // lazy prop
    if (!flip) return;
    swap(c[0],c[1]); flip = 0;
    for(int i = 0 ; i < 2 ; i++) if (c[i]) c[i]->flip ^= 1;
  }
  void calc() { // recalc vals
    for(int i = 0 ; i < 2 ; i++) if (c[i]) c[i]->prop();
    sz = 1+getSz(c[0])+getSz(c[1]);
    sub = 1+getSub(c[0])+getSub(c[1])+vsub;
  }
  //////// SPLAY TREE OPERATIONS
  int dir() {
    if (!p) return -2;
    for(int i = 0 ; i < 2 ; i++) if (p->c[i] == this) return i;
    return -1; // p is path-parent pointer
  } // -> not in current splay tree
  // test if root of current splay tree
  bool isRoot() { return dir() < 0; }
  friend void setLink(sn x, sn y, int d) {
    if (y) y->p = x;
    if (d >= 0) x->c[d] = y; }
  void rot() { // assume p and p->p propagated
    assert(!isRoot()); int x = dir(); sn pa = p;
    setLink(pa->p, this, pa->dir());
    setLink(pa, c[x^1], x); setLink(this, pa, x^1);
    pa->calc();
  }
  void splay() {
    while (!isRoot() && !p->isRoot()) {
      p->p->prop(), p->prop(), prop();
      dir() == p->dir() ? p->rot() : rot();
      rot();
    }
    if (!isRoot()) p->prop(), prop(), rot();
    prop(); calc();
  }
  sn fbo(int b) { // find by order
    prop(); int z = getSz(c[0]); // of splay tree
    if (b == z) { splay(); return this; }
    return b < z ? c[0]->fbo(b) : c[1] -> fbo(b-z-1);
  }
  //////// BASE OPERATIONS
  void access() { // bring this to top of tree, propagate
    for (sn v = this, pre = NULL; v; v = v->p) {
      v->splay(); // now switch virtual children
      if (pre) v->vsub -= pre->sub;
      if (v->c[1]) v->vsub += v->c[1]->sub;
      v->c[1] = pre; v->calc(); pre = v;
    }
    splay(); assert(!c[1]); // right subtree is empty
  }
  void makeRoot() {
    access(); flip ^= 1; access(); assert(!c[0] && !c[1]); }
  //////// QUERIES
  friend sn lca(sn x, sn y) {
    if (x == y) return x;
    x->access(), y->access(); if (!x->p) return NULL;
    x->splay(); return x->p?:x; // y was below x in latter case
  } // access at y did not affect x -> not connected
  friend bool connected(sn x, sn y) { return lca(x,y); }
  // # nodes above
  int distRoot() { access(); return getSz(c[0]); }
  sn getRoot() { // get root of LCT component
    access(); sn a = this;
    while (a->c[0]) a = a->c[0], a->prop();
    a->access(); return a;
  }
  sn getPar(int b) { // get b-th parent on path to root
    access(); b = getSz(c[0])-b; assert(b >= 0);
    return fbo(b);
  } // can also get min, max on path to root, etc
  //////// MODIFICATIONS
  void set(int v) { access(); val = v; calc(); }
  friend void link(sn x, sn y, bool force = 0) {
    assert(!connected(x,y));
    if (force) y->makeRoot(); // make x par of y
    else { y->access(); assert(!y->c[0]); }
    x->access(); setLink(y,x,0); y->calc();
  }
  friend void cut(sn y) { // cut y from its parent
    y->access(); assert(y->c[0]);
    y->c[0]->p = NULL; y->c[0] = NULL; y->calc(); }
  friend void cut(sn x, sn y) { // if x, y adj in tree
    x->makeRoot(); y->access();
    assert(y->c[0] == x && !x->c[0] && !x->c[1]); cut(y); }
};
sn LCT[MX];
```

3.12 Mo's algorithm

```
void MO_S_ADD(int);
void MO_S_REMOVE(int);
ll MO_S_GET_ANSWER();
vector<ll> MO_S_ALGORITHM(vector<pair<int,int>> &queries) {
    int Q = queries.size();
    //M = N/sqrt(Q)
    const int BLOCK_SIZE = 500;
    struct Query { int l, r, id; };
    vector<Query> que(Q);
    for(int i = 0 ; i < Q ; i++) {
        que[i].l = queries[i].first;
        que[i].r = queries[i].second;
        que[i].id = i;
    }
    sort(que.begin(),que.end(),[&](const Query &x, const Query
&y) {
        if(x.l/BLOCK_SIZE != y.l/BLOCK_SIZE)return x.l < y.l;
        return (x.l/BLOCK_SIZE&1) ? (x.r < y.r) : (x.r > y.r);
    });
    int cur_l = 0, cur_r = -1;
    vector<ll> res(Q);
    for(Query q : que) {
        while(cur_l > q.l) {
            cur_l--, MO_S_ADD(cur_l);
        }
        while(cur_r < q.r) {
            cur_r++, MO_S_ADD(cur_r);
        }
        while(cur_l < q.l) {
            MO_S_REMOVE(cur_l), cur_l++;
        }
        while(cur_r > q.r) {
            MO_S_REMOVE(cur_r), cur_r--;
        }
        res[q.id] = MO_S_GET_ANSWER();
    }
    return res;
}
```

3.13 Normal Treap

```
template<typename T>
struct treap {
    struct node {
        T key; int sz; int prior; node *l,*r;
        node(T key) : l(0),r(0),key(key),sz(1),prior(rng()) {}
    };
    typedef node* pnode;
    pnode root;
    treap() : root(0){}
    int sz(pnode p) { return p ? p->sz : 0; }
    void update(pnode p) {
        if(p) p->sz=1+sz(p->l)+sz(p->r);
    }
    //left subtree will contain elements less than or equal to
key
    void split(pnode p,T key,pnode &l,pnode &r) {
        if(!p) l = r = 0;
        else if(key<p->key) {
            split(p->l,key,l,p->l); r=p;
        } else {
            split(p->r,key,p->r,r); l=p;
        }
        update(p);
    }
    void insert(T key) {
        pnode add=new node(key); insert(root,add);
    }
    void erase(T key) {
        int pos=order_of_key(key);
        if(pos && key==find_kth(pos))
            erase(root,key);
    }
    ///returns the number of elements less than or equal to
the key
    int order_of_key(T key) { return order_of_key(root,key); }
    ///returns the k-th(1-indexed) in the ordered set of keys
of the treap
    T find_kth(int k) {return find_kth(root,k);}
    int size() { return sz(root); }
    int order_of_key(pnode p,T key) {
        if(!p) return 0;
        if(p->key<=key) return
sz(p->l)+1+order_of_key(p->r,key);
        return order_of_key(p->l,key);
    }
    T find_kth(pnode p,int k) {
        int pos=sz(p->l)+1;
        if(pos==k) return p->key;
        if(pos>k) return find_kth(p->l,k);
        return find_kth(p->r,k-pos);
    }
    void insert(pnode &p,pnode it) {
        if(!p) p=it;
        else if(it->prior>p->prior) {
            split(p,it->key,it->l,it->r); p=it;
        } else {
            if(it->key<p->key) insert(p->l,it);
            else insert(p->r,it);
        }
        update(p);
    }
    void merge(pnode &p,pnode l,pnode r) {
        if(!l || !r) {
            if(l) p=l;
            else p=r;
        } else if(l->prior>r->prior) {
            merge(l->r,l->r,r); p=l;
        } else {
            merge(r->l,l,r->l); p=r;
        }
        update(p);
    }
    void erase(pnode &p,T key) {
        if(p->key==key)
            merge(p,p->l,p->r);
        else if(key<p->key)
            erase(p->l,key);
        else
            erase(p->r,key);
        update(p);
    }
};
```

3.14 Ordered Stats

```
template<class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
      tree_order_statistics_node_update>;
```

3.15 Persistent SegTree

```
typedef int T;
struct node{
    int l, r;T v;
    node () {}
};
vector<node> pool; int actual;
int next() {
    pool.push_back(node());
    return actual ++;
}
struct pst {
    vector<int> versions; int n;
    pst(vector<T> &a) : n(a.size()) {
        versions.push_back(build(0, n - 1, a));
    }
    T merge(T v1, T v2) { /*Merge v1 and v2*/ }
    void up(int p, T v) { /*How v affects pool[p].v*/ }
    int build(int l, int r, vector<T> &a){
        int ans = next();
        if (l == r){
            pool[ans].v = a[l];
            return ans;
        }
        int mid = (l + r) >> 1;
        pool[ans].l = build(l, mid, a);
        pool[ans].r = build(mid + 1, r, a);
        pool[ans].v = merge(pool[pool[ans].l].v,
pool[pool[ans].r].v);
        return ans;
    }
    int clone(int p){
        int ans = next();
        pool[ans] = pool[p];
        return ans;
    }
    void update(int ver, int pos, T v) {
        versions.push_back(update(versions[ver], 0, n - 1,
pos, v));
    }
    void update(int pos, T v) {
        versions.push_back(update(versions.back(), 0, n - 1,
pos, v));
    }
    int update(int p, int l, int r, int pos, T v){
        p = clone(p);
        if (r == l){
            up(p, v);
            return p;
        }
        int mid = (l + r) >> 1;
        if (pos <= mid) pool[p].l = update(pool[p].l, l, mid,
pos, v);
        else pool[p].r = update(pool[p].r, mid + 1, r, pos, v);
        pool[p].v = merge(pool[pool[p].l].v,
pool[pool[p].r].v);
        return p;
    }
    T query(int t, int l, int r) {
        return query(versions[t], 0, n - 1, l, r);
    }
    T query(int p, int l, int r, int L, int R){
        if (L <= l && r <= R) return pool[p].v;
        int mid = (l + r) >> 1;
        if (R <= mid) return query(pool[p].l, l, mid, L, R);
        if (L > mid) return query(pool[p].r, mid + 1, r, L, R);
        return merge(query(pool[p].l, l, mid, L, R),
                     query(pool[p].r, mid + 1, r, L, R));
    }
};
```

3.16 Rollback Trick

```
template<typename F, typename Q, typename T, typename D>
struct ROLLBACK_TRICK {
    typedef long long ll;
    vector<vector<T>> ST; vector<vector<Q>> arr; int N;
    void update(T &x, D &DS) { DS.join(x.first, x.second); }
    F query(Q &x, D &DS) { return DS.rnk[DS.find(x).first]; }
    void rollback(D &DS) { DS.rollback(); }
    void add_to_tree(int nodo, int l, int r, int a, int b, T
&qu) {
        if(r < a || l > b) return;
        if(a <= l && r <= b) {
            ST[nodo].push_back(qu); return;
        }
        int mid = (l+r)/2;
        add_to_tree(nodo*2, l, mid, a, b, qu);
        add_to_tree(nodo*2+1, mid+1, r, a, b, qu);
    }

    void add_query(T &qu, int l, int r) {
        add_to_tree(1, 0, N-1, l, r, qu);
    }

    void dfs(int nodo, int l, int r, vector<F> &ans, D &DS) {
        for(auto x : ST[nodo])update(x, DS);
        if(l == r)
            for(auto x : arr[l])ans.push_back(query(x, DS));
        else {
            int mid = (l+r)/2;
            dfs(nodo*2, l, mid, ans, DS);
            dfs(nodo*2+1, mid+1, r, ans, DS);
        }
        for(int i = 0 ; i < ST[nodo].size() ; i++)
rollback(DS);
    }
    vector<F> solve(vector<Q> &qu, vector<ll> &querys,
        vector<T> &vect, vector<pair<ll,ll>> &ranges, D &DS) {
        assert(qu.size() == querys.size()),
            assert(vect.size() == ranges.size());
        vector<pair<ll,ll>> mp;
        for(int i = 0 ; i < querys.size() ; i++) {
            mp.push_back({querys[i], i});
        }
        for(int i = 0 ; i < ranges.size() ; i++) {
            mp.push_back({ranges[i].first, i+querys.size()});
            mp.push_back({ranges[i].second,
i+querys.size()+ranges.size()});
        }
        sort(mp.begin(), mp.end());
        ll temp = 0;
        int cont = 0;
        if(mp.size())temp = mp[0].first;
        for(int i = 0 ; i < mp.size() ; i++) {
            if(mp[i].first != temp)cont++;
            if(mp[i].second <
querys.size())querys[mp[i].second] = cont;
            else if(mp[i].second >= querys.size() +
ranges.size())

ranges[mp[i].second-querys.size()-ranges.size()].second =
cont;
            else ranges[mp[i].second-querys.size()].first =
cont;
            temp = mp[i].first;
        }
        N = cont+1; ST.clear(), ST.resize(4*(N+4));
        for(int i = 0 ; i < vect.size() ; i++)
            add_query(vect[i], ranges[i].first,
ranges[i].second);
        arr.clear(), arr.resize(N);
        for(int i = 0 ; i < querys.size() ; i++)
            arr[querys[i]].push_back(qu[i]);
        vector<F> ans;
        dfs(1, 0, N-1, ans, DS);
        return ans;
    }
};
```

3.17 Segment Tree

```
//Must implement merge_ function and push
//For use without lazy, remove stuff that contains if-lazy
template <typename Tnode,typename Tlazy>
struct ST{
    vector<Tnode> st; int sz;
    vector<Tlazy> lazy; Tlazy neutraL; // if-lazy
    ST (int n,Tlazy _neutraL) : sz(n), st(2*n),
                                lazy(2*n), neutraL(_neutraL)
{} // if-lazy
    Tnode merge_(Tnode a, Tnode b) {}
    void push (int nod,int l,int r){ /** IF-LAZY */
        int mi = (l + r) >> 1;
        /// how lazy[nod] affects st[nod]
        if( l != r ){
            /// how lazy[nod] affects lazy[nod+1] and
lazy[nod+2*(mi-l+1)]
        }
        lazy[nod] = neutraL;
    }
    void build (vector<Tnode> &arr){build(0, 0, sz-1, arr);}
    void build (int nod,int l,int r,vector<Tnode> &arr){
        if (l == r) {
            st[nod] = arr[l];
            lazy[nod] = neutraL; // if-lazy
            return;
        }
        int mi = (l + r) >> 1;
        build(nod+1,l,mi,arr);
        build(nod+((mi-l+1)<<1), mi+1, r, arr);
        st[nod] = merge_(st[nod+1], st[nod+((mi-l+1)<<1)]);
        lazy[nod] = neutraL; // if-lazy
    }
    void updateRange (int x,int y,Tlazy v) { /** IF-LAZY */
        updateRange(0, 0, sz-1, x, y, v);
    }
    void updateRange (int nod,int l,int r,int x,int y,Tlazy v){
        push(nod, l, r);
        if (l >= x && r <= y) {
            lazy[nod] = v;
            push(nod, l, r); return;
        }
        int mi = (l + r) >> 1;
        if (x <= mi) updateRange(nod+1, l, mi, x, y, v);
        else push(nod+1, l, mi);
        if (y > mi) updateRange(nod+((mi-l+1)<<1), mi+1, r, x,
y, v);
        else push(nod+((mi-l+1)<<1), mi+1, r);
        st[nod] = merge_(st[nod+1] , st[nod+((mi-l+1)<<1)]);
    }
    void updatePoint(int id, Tnode val) {
        updatePoint(0, 0, sz - 1, id, val);
    }
    void updatePoint(int nod,int l,int r,int id, Tnode val) {
        push(nod, l, r); // if-lazy
        if (l == r) {
            st[nod] = val; return;
        }
        int mi = (l + r) >> 1;
        if (id <= mi) {
            updatePoint(nod+1, l, mi, id, val);
            push(nod+((mi-l+1)<<1), mi+1, r); // if-lazy
        } else {
            updatePoint(nod+((mi-l+1)<<1), mi+1, r, id, val);
            push(nod+1, l, mi); // if-lazy
        }
        st[nod] = merge_(st[nod+1], st[nod+((mi-l+1)<<1)]);
    }
    Tnode query (int l,int r) {return query(0, 0, sz-1, l, r);}
    Tnode query (int nod,int l,int r,int x,int y) {
        push(nod, l, r); // if-lazy
        if (l >= x && r <= y) return st[nod];
        int mi = (l + r) >> 1;
        if (y <= mi) return query(nod+1, l, mi, x, y);
        if (x > mi) return query(nod+((mi-l+1)<<1), mi+1, r, x,
y);
        return merge_(query(nod+1, l, mi, x, y),
                      query(nod+((mi-l+1)<<1), mi+1, r, x,
y));
    }
};
```

3.18 Sparse Table

```
template <typename T, class F = less<T>>
struct RMQ {
    vector<vector<T>> A;
    RMQ(const vector<T> &arr) {
        A.resize(arr.size());
        for (int i = 0; i < A.size(); i++)
            A[i].resize(31 - __builtin_clz(arr.size()) + 1),
A[i][0] = arr[i];
        for (int j = 1; j <= 31 - __builtin_clz(arr.size());
j++)
            for (int i = 0; i + (1 << j) - 1 < A.size(); i++)
                if (F()(A[i][j - 1], A[i + (1 << (j - 1))][j -
1])) A[i][j] = A[i][j - 1];
                else A[i][j] = A[i + (1 << (j - 1))][j - 1];
    }
    T query(int a, int b) {
        if (a > b) swap(a, b);
        int k = 31 - __builtin_clz(b - a + 1);
        if (F()(A[a][k], A[b - (1 << k) + 1][k])) return
A[a][k];
        else return A[b - (1 << k) + 1][k];
    }
};
```

3.19 Cartesian Tree

```
template<typename T, class F = less<T>>
int CARTESIAN_TREE(vector<vector<int>> &L, const vector<T>
&arr) {
    int N = arr.size();
    L.clear(); L.resize(N);
    stack<int> st; vector<int> parent(N,-1);
    for(int i = 0 ; i < N ; i++) {
        int last = -1;
        while(!st.empty() && F()(arr[i],arr[st.top()]))
            last = st.top(), st.pop();
        if(!st.empty())parent[i] = st.top();
        if(last >= 0)parent[last] = i;
        st.push(i);
    }
    int root = -1;
    for(int i = 0 ; i < N ; i++) {
        if(parent[i] == -1)root = i;
        else L[parent[i]].push_back(i);
    }
    return root;
}
```

4 Graph

4.1 Hungarian

```
//  Description:
//  - We are given a cost table of size n times m with n <= m.
//  - It finds a maximum cost assignment, i.e.,
//        max sum_{ij} c(i,j) x(i,j)
//    where   sum_{i in [n]} x(i,j) = 1,
//        sum_{j in [n]} x(i,j) <= 1.
//  Complexity: O(n^3)
template<typename T>
T max_assignment(const vector<vector<T>> &a) {
    int n = a.size(), m = a[0].size();
    assert(n <= m);
    vector<int> x(n, -1), y(m, -1);
    vector<T> px(n, numeric_limits<T>::min()), py(m, 0);
    for (int u = 0; u < n; ++u)
        for (int v = 0; v < m; ++v) px[u] = max(px[u],
a[u][v]);
    for (int u = 0, p, q; u < n;) {
        vector<int> s(n + 1, u), t(m, -1);
        for (p = q = 0; p <= q && x[u] < 0; ++p)
            for (int k = s[p], v = 0; v < m && x[u] < 0; ++v)
                if (px[k] + py[v] == a[k][v] && t[v] < 0) {
                    s[++q] = y[v], t[v] = k;
                    if (s[q] < 0)
                        for (p = v; p >= 0; v = p)
                            y[v] = k = t[v], p = x[k], x[k] =
v;
                }
        if (x[u] < 0) {
            T delta = numeric_limits<T>::max();
            for (int i = 0; i <= q; ++i)
                for (int v = 0; v < m; ++v) if (t[v] < 0)
                        delta = min(delta, px[s[i]] + py[v] -
a[s[i]][v]);
            for (int i = 0; i <= q; ++i)
                px[s[i]] -= delta;
            for (int v = 0; v < m; ++v)
                py[v] += (t[v] < 0 ? 0 : delta);
        } else ++u;
    }
    T cost = 0;
    for (int u = 0; u < n; ++u)
        cost += a[u][x[u]];
    return cost;
}
```

4.2 Complementary Graph Components

```
vector<vector<int>>
INVERSE_GRAPH_COMPONENTS(vector<vector<int>> &L) {
    int N = L.size();
    vector<vector<int>> ady(N), comps;
    vector<bool> in(N);
    list<int> ls;
    for(int i = 0 ; i < N ; i++)
        ls.push_back(i);
    while(!ls.empty()) {
        queue<int> Q;
        vector<int> cmp;
        Q.push(*ls.begin());
        cmp.push_back(*ls.begin());
        ls.erase(ls.begin());
        while(!Q.empty()) {
            int nodo = Q.front();
            Q.pop();
            for(auto v : L[nodo])
                in[v] = true;
            vector<list<int>::iterator> temp;
            for(auto it = ls.begin() ; it != ls.end() ; it++)
                if(!in[*it])
                    temp.push_back(it);
            for(auto x : temp) {
                ady[nodo].push_back(*x);
                ady[*x].push_back(nodo);
                Q.push(*x);
                cmp.push_back(*x);
                ls.erase(x);
            }
            for(auto v : L[nodo])
                in[v] = false;
        }
        comps.push_back(cmp);
    }
    return comps;
}
```

4.3 Directed Euler Path

```
int CHECK_EULER_TOUR(int N, vector<pair<int,int>> &edge,
    bool undigraph, int troot = -1) {
    assert(troot == -1 || (troot >= 0 && troot < N));
    vector<vector<int>> L(N);
    for(int i = 0 ; i < edge.size() ; i++) {
        assert(edge[i].first >= 0 && edge[i].first < N);
        assert(edge[i].second >= 0 && edge[i].second < N);
        if(undigraph) {
            L[edge[i].first].push_back(edge[i].second);
            L[edge[i].second].push_back(edge[i].first);
        } else {
            L[edge[i].first].push_back(edge[i].second);
        }
    }
    vector<int> ingree(N), outgree(N);
    vector<bool> mark(N);
    int total_edge = 0;
    for(int i = 0 ; i < N ; i++) {
        total_edge += L[i].size();
        for(auto v : L[i]) {
            outgree[i]++;
            ingree[v]++;
        }
    }
    int init = -1;
    vector<pair<int,int>> odd;
    for(int i = 0 ; i < N ; i++) {
        if(init == -1 && outgree[i])init = i;
        if(undigraph) {
            if(outgree[i]&1)odd.push_back({i,0});
        } else {
            if(outgree[i] != ingree[i])
                odd.push_back({i,outgree[i]-ingree[i]});
        }
    }
    int root = -1;
    if(odd.size()) {
        if(undigraph) {
            if(odd.size() != 2)return -1;

            if(odd[0].first > odd[1].first)swap(odd[0],odd[1]);

            if(troot != -1) {
                if(troot == odd[0].first)root = troot;
                else if(troot == odd[1].first)root = troot;
                else return -1;
            } else root = odd[0].first;
        } else {
            if(odd.size() != 2)return -1;
            if(odd[0].second <
odd[1].second)swap(odd[0],odd[1]);
            if(odd[0].second == 1 && odd[1].second == -1) {
                if(troot != -1) {
                    if(troot == odd[0].first)root = troot;
                    else return -1;
                } else root = odd[0].first;
            } else return -1;
        }
    } else {
        if(troot != -1)root = troot;
        else root = init;
    }
    queue<int> Q;
    mark[root] = true;
    Q.push(root);
    int temp_edge = 0;
    while(!Q.empty()) {
        int nodo = Q.front();
        Q.pop();
        temp_edge += L[nodo].size();
        for(auto v : L[nodo]) {
            if(!mark[v]) {
                mark[v] = true;
                Q.push(v);
            }
        }
    }
    if(total_edge != temp_edge)return -1;
    return root;
}
vector<int> EULER_TOUR(int N, vector<pair<int,int>> &edge,
    bool undigraph, int troot = -1) {
    vector<int> res;
    int root = CHECK_EULER_TOUR(N,edge,undigraph,troot);
    if(root == -1) {
        return res;
    }
    vector<vector<int>> L(N), id(N), inv(N);
    for(int i = 0 ; i < edge.size() ; i++) {
        if(undigraph) {
            L[edge[i].first].push_back(edge[i].second);
            L[edge[i].second].push_back(edge[i].first);
            id[edge[i].first].push_back(i);
            id[edge[i].second].push_back(i+edge.size());
            if(edge[i].first == edge[i].second) {
                int sz = inv[edge[i].first].size();
                inv[edge[i].first].push_back(sz+1);
                inv[edge[i].second].push_back(sz);
            } else {
                int sz2 = inv[edge[i].second].size();
                int sz1 = inv[edge[i].first].size();
                inv[edge[i].first].push_back(sz2);
                inv[edge[i].second].push_back(sz1);
            }
        } else {
            L[edge[i].first].push_back(edge[i].second);
            id[edge[i].first].push_back(i);
        }
    }
    vector<vector<bool>> mark(N);
    vector<int> ptr(N);
    for(int i = 0 ; i < N ; i++) {
        mark[i].resize(L[i].size());
    }
    stack<pair<int,int>> S;
    S.push({root,-1});
    while(!S.empty()) {
        int nodo = S.top().first;
        while(ptr[nodo] < L[nodo].size() &&
mark[nodo][ptr[nodo]])
            ptr[nodo]++;
        if(ptr[nodo] == L[nodo].size()) {
            res.push_back(S.top().second);
            S.pop();
        } else {
            mark[nodo][ptr[nodo]] = true;
            if(undigraph)
                mark[L[nodo][ptr[nodo]]][inv[nodo][ptr[nodo]]]
= true;
            S.push({L[nodo][ptr[nodo]],id[nodo][ptr[nodo]]});
        }
    }
    reverse(res.begin(),res.end());
    return res;
}
```

4.4 2-SAT

```
struct satisfiability_twosat {
    int n; vector<vector<int>> imp;
    satisfiability_twosat(int n) : n(n), imp(2 * n) {}
    void add_edge(int u, int v) { imp[u].push_back(v); }
    int neg(int u) { return (n << 1) - u - 1; }
    void implication(int u, int v) {
        add_edge(u, v);
        add_edge(neg(v), neg(u));
    }
    vector<bool> solve() {
        int size = 2 * n;
        vector<int> S, B, I(size);
        function<void(int)> dfs = [&](int u) {
            B.push_back(I[u] = S.size());
            S.push_back(u);
            for (int v : imp[u])
                if (!I[v]) dfs(v);
                else while (I[v] < B.back()) B.pop_back();
            if (I[u] == B.back())
                for (B.pop_back(), ++size; I[u] < S.size();
S.pop_back())
                    I[S.back()] = size;
        };
        for (int u = 0; u < 2 * n; ++u)
            if (!I[u]) dfs(u);
        vector<bool> values(n);
        for (int u = 0; u < n; ++u)
            if (I[u] == I[neg(u)]) return {};
            else values[u] = I[u] < I[neg(u)];
        return values;
    }
};
```

4.5 BCC Art Points

```
struct graph {
    int n; vector<vector<int>> adj;
    graph(int n) : n(n), adj(n) {}
    void add_edge(int u, int v) {
        adj[u].push_back(v); adj[v].push_back(u);
    }
    int add_node() { adj.push_back({}); return n++; }
    vector<int>& operator[](int u) { return adj[u]; }
};
vector<vector<int>> biconnected_components(graph &adj) {
    int n = adj.n;
    vector<int> num(n), low(n), art(n), stk;
    vector<vector<int>> comps;
    function<void(int, int, int&)> dfs = [&](int u, int p, int
&t)
    {
        num[u] = low[u] = ++t; stk.push_back(u);
        for (int v : adj[u])
            if (v != p) {
                if (!num[v]) {
                    dfs(v, u, t);
                    low[u] = min(low[u], low[v]);
                    if (low[v] >= num[u]) {
                        art[u] = (num[u] > 1 || num[v] > 2);
                        comps.push_back({u});
                        while (comps.back().back() != v)

comps.back().push_back(stk.back()),
                                    stk.pop_back();
                    }
                }
                else low[u] = min(low[u], num[v]);
            }
    };
    for (int u = 0, t; u < n; ++u)
        if (!num[u]) dfs(u, -1, t = 0);
    /// build the block cut tree
    function<graph()> build_tree = [&]() {
        graph tree(0); vector<int> id(n);
        for (int u = 0; u < n; ++u)
            if (art[u]) id[u] = tree.add_node();
        for (auto &comp : comps) {
            int node = tree.add_node();
            for (int u : comp)
                if (!art[u]) id[u] = node;
                else tree.add_edge(node, id[u]);
        }
        return tree;
    };
    return comps;
}
```

4.6 Bridges

```
struct graph {
    int n; vector<vector<int>> adj;
    graph(int n) : n(n), adj(n) {}
    void add_edge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    vector<int>& operator[](int u) { return adj[u]; }
};
vector<vector<int>> bridge_blocks(graph &adj) {
    int n = adj.n;
    vector<int> num(n), low(n), stk;
    vector<vector<int>> comps;
    vector<pair<int, int>> bridges;
    function<void(int, int, int&)> dfs = [&](int u, int p, int
&t) {
        num[u] = low[u] = ++t;
        stk.push_back(u);
        // remove if there isn't parallel edges
        sort(adj[u].begin(), adj[u].end());
        for (int i = 0, sz = adj[u].size(); i < sz; ++i) {
            int v = adj[u][i];
            if (v == p) {
                if (i + 1 < sz && adj[u][i + 1] == v)
                    low[u] = min(low[u], num[v]);
                continue;
            }
            if (!num[v]) {
                dfs(v, u, t);
                low[u] = min(low[u], low[v]);

                if (low[v] == num[v])
                    bridges.push_back({u, v});
            }
            else low[u] = min(low[u], num[v]);
        }
        if (num[u] == low[u])
        {
            comps.push_back({});
            for (int v = -1; v != u; stk.pop_back())
                comps.back().push_back(v = stk.back());
        }
    };
    for (int u = 0, t; u < n; ++u)
        if (!num[u]) dfs(u, -1, t = 0);
    // this is for build the bridge-block tree
    function<graph()> build_tree = [&]() {
        vector<int> id(n);
        for (int i = 0; i < (int) comps.size(); ++i)
            for (int u : comps[i]) id[u] = i;
        graph tree(comps.size());
        for (auto &e : bridges)
            tree.add_edge(id[e.first], id[e.second]);
        return tree;
    };
    return comps;
}
```

4.7 Centroid

```
int Centroid(vector<vector<int>> &Lcent, vector<vector<int>>
&L, int root) {
    int N = L.size();
    vector<bool> mk(N); vector<int> sz(N);
    Lcent.clear(); Lcent.resize(N);
    function<void(int,int)> DFS = [&](int nodo, int padre) {
        sz[nodo] = 1;
        for(auto v : L[nodo]) {
            if(v != padre && !mk[v]) {
                DFS(v, nodo); sz[nodo] += sz[v];
            }
        }
    };
    function<int(int,int,int)> centroid = [&](int nodo, int
padre, int tam) {
        for(auto v : L[nodo])
            if(v != padre && !mk[v] && sz[v] > tam/2)
                return centroid(v, nodo, tam);
        return nodo;
    };
    function<int(int)> solve = [&](int nodo) {
        DFS(nodo, -1);
        int cent = centroid(nodo, -1, sz[nodo]); mk[cent] =
true;
        for(auto v : L[cent])
            if(!mk[v]) Lcent[cent].push_back(solve(v));
        return cent;
    };
    return solve(root);
}
```

4.8 Dinic

```
template<typename T>
struct dinic {
    struct edge {
        int src, dst; T cap, flow; int rev;
    };
    int n; vector< vector<edge> > adj;
    dinic(int n) : n(n), adj(n) {}
    void add_edge(int src, int dst, T cap) {
        adj[src].push_back({ src, dst, cap, 0, (int)
adj[dst].size() });
        if (src == dst)
            adj[src].back().rev++;
        adj[dst].push_back({ dst, src, 0, 0, (int)
adj[src].size() - 1 });
    }
    vector<int> level, iter;
    T augment(int u, int t, T cur) {
        if (u == t)
            return cur;
        for (int &i = iter[u]; i < (int) adj[u].size(); ++i) {
            edge &e = adj[u][i];
            if (e.cap - e.flow > 0 && level[u] > level[e.dst])
{
                T f = augment(e.dst, t, min(cur, e.cap -
e.flow));
                if (f > 0) {
                    e.flow += f;
                    adj[e.dst][e.rev].flow -= f;
                    return f;
                }
            }
        }
        return 0;
    }
    int bfs(int s, int t) {
        level.assign(n, n);
        level[t] = 0;
        queue<int> Q;
        for (Q.push(t); !Q.empty(); Q.pop()) {
            int u = Q.front();
            if (u == s)
                break;
            for (int i = 0; i < (int)adj[u].size(); ++i) {
                edge &e = adj[u][i];
                edge &erev = adj[e.dst][e.rev];
                if (erev.cap - erev.flow > 0
                    && level[e.dst] > level[u] + 1) {
                    Q.push(e.dst);
                    level[e.dst] = level[u] + 1;
                }
            }
        }
        return level[s];
    }
    const T oo = numeric_limits<T>::max();
    T max_flow(int s, int t) {
        for (int u = 0; u < n; ++u) // initialize
            for (int i = 0; i < (int)adj[u].size(); ++i) {
                edge &e = adj[u][i];
                e.flow = 0;
            }
        T flow = 0;
        while (bfs(s, t) < n) {
            iter.assign(n, 0);
            for (T f; (f = augment(s, t, oo)) > 0;)
                flow += f;
        } // level[u] == n ==> s-side
        return flow;
    }
};
```

4.9 Euler Path

```
vector<int> euler_path(const vector<vector<pair<int, int>>> &G,
int s = 0) {
    int n = G.size(), odd = 0, m = 0;
    for (int i = 0; i < n; ++i) {
        odd += G[i].size() & 1;
        m += G[i].size();
    }
    vector<int> path;
    if (odd == 0 || (odd == 2 && (G[s].size() & 1) == 1)) {
        vector<int> pos(n); vector<bool> mark(m / 2);
        function<void(int)> visit = [&](int u) {
            for (int v, id; pos[u] < G[u].size();) {
                tie(v, id) = G[u][pos[u]++];
                if (!mark[id]) {
                    mark[id] = true;
                    visit(v);
                }
            }
            path.push_back(u);
        };
        visit(s);
        reverse(path.begin(), path.end());
        if (path.size() != m / 2 + 1)
            path.clear();
    }
    return path;
}
```

4.10 Flow Lower Bound

```
template<typename T>
struct dinic {
    struct edge {
        int src, dst; T low, cap, flow; int rev;
    };
    int n; vector<vector<edge>> adj;
    dinic(int n) : n(n), adj(n + 2) {}
    void add_edge(int src, int dst, T low, T cap) {
        adj[src].push_back({ src, dst, low, cap, 0,
(int)adj[dst].size()});
        if (src == dst) adj[src].back().rev++;
        adj[dst].push_back({ dst, src, 0, 0, 0,
(int)adj[src].size() - 1});
    }
    vector<int> level, iter;
    T augment(int u, int t, T cur) {
        if (u == t) return cur;
        for (int &i = iter[u]; i < (int) adj[u].size(); ++i) {
            edge &e = adj[u][i];
            if (e.cap - e.flow > 0 && level[u] > level[e.dst])
{
                T f = augment(e.dst, t, min(cur, e.cap -
e.flow));
                if (f > 0) {
                    e.flow += f; adj[e.dst][e.rev].flow -= f;
                    return f;
                }
            }
        }
        return 0;
    }
    int bfs(int s, int t) {
        level.assign(n + 2, n + 2); level[t] = 0; queue<int> Q;
        for (Q.push(t); !Q.empty(); Q.pop()) {
            int u = Q.front(); if (u == s) break;
            for (edge &e : adj[u]) {
                edge &erev = adj[e.dst][e.rev];
                if (erev.cap - erev.flow > 0
                    && level[e.dst] > level[u] + 1) {
                    Q.push(e.dst); level[e.dst] = level[u] +
1;
                }
            }
        }
        return level[s];
    }
    const T oo = numeric_limits<T>::max();
    T max_flow(int source, int sink) {
        vector<T> delta(n + 2);
        for (int u = 0; u < n; ++u) // initialize
            for (auto &e : adj[u]) {
                delta[e.src] -= e.low;
                delta[e.dst] += e.low;
                e.cap -= e.low;
                e.flow = 0;
            }
        T sum = 0; int s = n, t = n + 1;
        for (int u = 0; u < n; ++u) {
            if (delta[u] > 0) {
                add_edge(s, u, 0, delta[u]);
                sum += delta[u];
            }
            else if (delta[u] < 0) add_edge(u, t, 0,
-delta[u]);
        }
        add_edge(sink, source, 0, oo); T flow = 0;
        while (bfs(s, t) < n + 2) {
            iter.assign(n + 2, 0);
            for (T f; (f = augment(s, t, oo)) > 0;) flow += f;
        }
        if (flow != sum) return -1; // no solution
        for (int u = 0; u < n; ++u)
            for (auto &e : adj[u]) {
                e.cap += e.low; e.flow += e.low;
                edge &erev = adj[e.dst][e.rev];
                erev.cap -= e.low; erev.flow -= e.low;
            }
        adj[sink].pop_back(); adj[source].pop_back();
        while (bfs(source, sink) < n + 2) {
            iter.assign(n + 2, 0);
            for (T f; (f = augment(source, sink, oo)) > 0;)
                flow += f;
        } // level[u] == n + 2 ==> s-side
        return flow;
    }
};
```

4.11 HLD

```
struct HLD {
    int N;
    vector<int> heavy, root, depth, pos, ipos, parent;
    int currentPos;
    int precal(const vector<vector<int>> &L, int nodo, int
padre) {
        parent[nodo] = padre, heavy[nodo] = -1;
        int size = 1, maxSubtree = 0;
        for(auto v : L[nodo]) {
            if(v != padre) {
                depth[v] = depth[nodo] + 1;
                int subtree = precal(L, v, nodo);
                if(subtree > maxSubtree)
                    heavy[nodo] = v, maxSubtree = subtree;
                size += subtree;
            }
        }
        return size;
    }
    void DFS(const vector<vector<int>> &L, int nodo, int padre)
{
        pos[nodo] = ++currentPos; ipos[currentPos] = nodo;
        if(heavy[nodo] != -1) {
            root[heavy[nodo]] = root[nodo];
            DFS(L, heavy[nodo], nodo);
        }
        for(auto v : L[nodo]) {
            if(v != padre && v != heavy[nodo]) {
                root[v] = v;
                DFS(L, v, nodo);
            }
        }
    }
    HLD(const vector<vector<int>> &L, int ROOT = 0) {
        N = L.size();
        heavy.resize(N); root.resize(N); depth.resize(N);
        pos.resize(N); ipos.resize(N); parent.resize(N);
        depth[ROOT] = 0; currentPos = -1; root[ROOT] = ROOT;
        precal(L,ROOT,-1); DFS(L,ROOT,-1);
    }
    vector<pair<int,int>> get_paths(int u, int v) {
        vector<pair<int,int>> paths;
        for(; root[u] != root[v] ; v = parent[root[v]]) {
            if(depth[root[u]] > depth[root[v]])swap(u, v);
            paths.push_back({pos[root[v]], pos[v]});
        }
        if(depth[u] > depth[v])swap(u, v);
        paths.push_back({pos[u], pos[v]});
        return paths;
    }
    int lca(int a, int b) {
        for (; root[a] != root[b]; b = parent[root[b]])
            if (depth[root[a]] > depth[root[b]]) swap(a, b);
        if (depth[a] > depth[b]) swap(a, b);
        return a;
    }
    int go_up(int u, int k) {
        for(; pos[u] - pos[root[u]] < k ; u = parent[root[u]])
            k -= pos[u] - pos[root[u]] + 1;
        return ipos[pos[u] - k];
    }
};
```

4.12 Hopcroft Karp

```
struct graph {
  int L, R; vector<vector<int>> adj;
  graph(int L, int R) : L(L), R(R), adj(L + R) {}
  void add_edge(int u, int v) {
    adj[u].push_back(v + L);
    adj[v + L].push_back(u);
  }
  int maximum_matching() {
    vector<int> level(L), mate(L + R, -1);
    function<bool(void)> levelize = [&]() {
      queue<int> Q;
      for (int u = 0; u < L; ++u) {
        level[u] = -1;
        if (mate[u] < 0) {
          level[u] = 0;
          Q.push(u);
        }
      }
      while (!Q.empty()) {
        int u = Q.front(); Q.pop();
        for (int w : adj[u]) {
          int v = mate[w];
          if (v < 0) return true;
          if (level[v] < 0) {
            level[v] = level[u] + 1;
            Q.push(v);
          }
        }
      }
      return false;
    };
    function<bool(int)> augment = [&](int u) {
      for (int w : adj[u]) {
        int v = mate[w];
        if (v < 0 || (level[v] > level[u] && augment(v))) {
          mate[u] = w;
          mate[w] = u;
          return true;
        }
      }
      return false;
    };
    int match = 0;
    while (levelize())
      for (int u = 0; u < L; ++u)
        if (mate[u] < 0 && augment(u))
          ++match;
    return match;
  }
};
```

4.13 LCA

```
struct LCA {
    int LOG = 0;
    vector<vector<int>> T; vector<int> height;
    vector<bool> visited; int n;
    LCA(const vector<vector<int>> &adj, int root = 0) {
        n = adj.size(); int temp = n-1;
        while(temp)temp /= 2, LOG++;
        height.resize(n); visited.assign(n,false);
T.resize(n);
        for(int i = 0 ; i < T.size() ; i++)T[i].resize(LOG+1);
        T[root][0] = -1; DFS(adj,root); precal();
    }
    void DFS(const vector<vector<int>> &adj, int nodo) {
        visited[nodo] = true;
        for(auto v : adj[nodo])
            if(!visited[v]) {
                T[v][0] = nodo; height[v] = height[nodo]+1;
                DFS(adj,v);
            }
    }
    void precal() {
        for(int j = 1 ; j <= LOG ; j++)
            for(int i = 0 ; i < n ; i++)
                if(T[i][j-1] != -1)T[i][j] = T[T[i][j-1]][j-1];
                else T[i][j] = -1;
    }
    int query(int u, int v) {
        if(height[u] < height[v])swap(u,v);
        for(int i = LOG ; i >= 0 ; i--)
            if(height[u]-(1<<i) >= height[v])
                u = T[u][i];
        if(u == v)return u;
        for(int i = LOG ; i >= 0 ; i--)
            if(T[u][i] != T[v][i])
                u = T[u][i], v = T[v][i];
        return T[u][0];
    }
};
```

4.14 MCMF

```
template <typename T, typename C>
class mcmf {
public:
    static constexpr T eps = (T) 1e-9;
    struct edge { int from; int to; T c; T f; C cost; };
    vector< vector<int> > g; vector<edge> edges;
    vector<C> d; vector<int> q;
    vector<bool> in_queue; vector<int> pe;
    int n; int st, fin; T flow; C cost;
    mcmf(int _n, int _st, int _fin) : n(_n), st(_st), fin(_fin)
{
        assert(0 <= st && st < n && 0 <= fin && fin < n && st
!= fin);
        g.resize(n); d.resize(n);
        in_queue.resize(n); pe.resize(n);
        flow = 0; cost = 0;
    }
    void clear_flow() {
        for (const edge &e : edges) { e.f = 0; } flow = 0;
    }
    void add(int from, int to, T forward_cap, T backward_cap, C
cost) {
        assert(0 <= from && from < n && 0 <= to && to < n);
        g[from].push_back((int) edges.size());
        edges.push_back({from, to, forward_cap, 0, cost});
        g[to].push_back((int) edges.size());
        edges.push_back({to, from, backward_cap, 0, -cost});
    }
    bool expath() {
        fill(d.begin(), d.end(), numeric_limits<C>::max());
        q.clear(); q.push_back(st); d[st] = 0;
        in_queue[st] = true; int beg = 0; bool found = false;
        while (beg < (int) q.size()) {
            int i = q[beg++];
            if (i == fin) { found = true; }
            in_queue[i] = false;
            for (int id : g[i]) {
                const edge &e = edges[id];
                if (e.c - e.f > eps && d[i] + e.cost <
d[e.to]) {
                    d[e.to] = d[i] + e.cost; pe[e.to] = id;
                    if (!in_queue[e.to]) {
                        q.push_back(e.to); in_queue[e.to] =
true;
                    }
                }
            }
        }
        if (found) {
            T push = numeric_limits<T>::max(); int v = fin;
            while (v != st) {
                const edge &e = edges[pe[v]];
                push = min(push, e.c - e.f); v = e.from;
            }
            v = fin;
            while (v != st) {
                edge &e = edges[pe[v]]; e.f += push;
                edge &back = edges[pe[v] ^ 1];
                back.f -= push; v = e.from;
            }
            flow += push; cost += push * d[fin];
        }
        return found;
    }
    pair<T, C> max_flow_min_cost() {
        while (expath()) {}
        return make_pair(flow, cost);
    }
};
```

4.15 Push Relabel

```
template<typename flow_type>
struct goldberg_tarjan {
    struct edge {
        size_t src, dst, rev;
        flow_type flow, cap;
    };
    int n; vector<vector<edge>> adj;
    goldberg_tarjan(int n) : n(n), adj(n) {}
    void add_edge(size_t src, size_t dst, flow_type cap,
flow_type rcap = 0) {
        adj[src].push_back({ src, dst, adj[dst].size(), 0, cap
});
        if (src == dst)  adj[src].back().rev++;
        adj[dst].push_back({ dst, src, adj[src].size() - 1, 0,
rcap });
    }
    flow_type max_flow(int source, int sink) {
        vector<flow_type> excess(n);
        vector<int> dist(n), active(n), count(2 * n);
        queue<int> q;
        auto enqueue = [&](int v) {
            if (!active[v] && excess[v] > 0) {
                active[v] = true;
                q.push(v);
            }
        };
        auto push = [&](edge &e) {
            flow_type f = min(excess[e.src], e.cap - e.flow);
            if (dist[e.src] <= dist[e.dst] || f == 0) return;
            e.flow += f;
            adj[e.dst][e.rev].flow -= f;
            excess[e.dst] += f; excess[e.src] -= f;
            enqueue(e.dst);
        };
        dist[source] = n; active[source] = active[sink] =
true;
        count[0] = n - 1; count[n] = 1;
        for (int u = 0; u < n; ++u)
            for (edge &e : adj[u]) e.flow = 0;
        for (edge &e : adj[source]) {
            excess[source] += e.cap; push(e);
        }
        for (int u; !q.empty(); q.pop()) {
            active[u = q.front()] = false;
            for (auto &e : adj[u]) push(e);
            if (excess[u] > 0) {
                if (count[dist[u]] == 1) {
                    int k = dist[u]; // Gap Heuristics
                    for (int v = 0; v < n; v++) {
                        if (dist[v] < k)
                            continue;
                        count[dist[v]]--;
                        dist[v] = max(dist[v], n + 1);
                        count[dist[v]]++;
                        enqueue(v);
                    }
                } else {
                    count[dist[u]]--; // Relabel
                    dist[u] = 2 * n;
                    for (edge &e : adj[u])
                        if (e.cap > e.flow)
                            dist[u] = min(dist[u], dist[e.dst]
+ 1);

                    count[dist[u]]++;
                    enqueue(u);
                }
            }
        }
        flow_type flow = 0;
        for (edge e : adj[source]) flow += e.flow;
        return flow;
    }
};
```

4.16 SCC Gabow

```
struct graph {
    int n; vector<vector<int>> adj;
    graph(int n) : n(n), adj(n) {}
    void add_edge(int u, int v) {
        adj[u].push_back(v);
    }
    vector<int>& operator[](int u) { return adj[u]; }
};
```

```
vector<vector<int>> scc_gabow(graph &adj) {
    int n = adj.n;
    vector<vector<int>> scc;
    vector<int> S, B, I(n);
    function<void(int)> dfs = [&](int u) {
        B.push_back(I[u] = S.size());
        S.push_back(u);
        for (int v : adj[u])
            if (!I[v]) dfs(v);
            else while (I[v] < B.back()) B.pop_back();
        if (I[u] == B.back()) {
            scc.push_back({});
            for (B.pop_back(); I[u] < S.size(); S.pop_back()) {
                scc.back().push_back(S.back());
                I[S.back()] = n + scc.size();
            }
        }
    };
    for (int u = 0; u < n; ++u)
        if (!I[u]) dfs(u);
    return scc; // in reverse topological order
}
```

5 Number Theory

5.1 Chinese Reminder Theorem

```
bool crt(ll k1, ll m1, ll k2, ll m2, ll &k, ll &m) {
    k1 %= m1;
    if (k1 < 0) k1 += m1;
    k2 %= m2;
    if (k2 < 0) k2 += m2;
    ll x, y, g;
    if (!find_any_solution(m1, -m2, k2 - k1, x, y, g)) {
        return false;
    }
    ll dx = m2 / g;
    ll delta = x / dx - (x % dx < 0);
    k = m1 * (x - dx * delta) + k1;
    m = m1 / g * m2;
    assert(0 <= k && k < m);
    return true;
}
long long chinese_remainder(vector<int> rem, vector<int> mod) {
    long long ans = rem[0],m = mod[0]; int n = rem.size();
    for(int i=1; i<n; ++i) {
        int a = modular_inverse(m,mod[i]);
        int b = modular_inverse(mod[i],m);
        ans = (ans*b*mod[i]+rem[i]*a*m)%(m*mod[i]);
        m *= mod[i];
    }
    return ans;
}
```

5.2 Diophantine

```
template<typename T>
bool find_any_solution(T a, T b, T c, T &x, T &y, T &g) {
    if (a == 0 && b == 0) {
        if (c == 0) {
            x = y = g = 0;
            return true;
        }
        return false;
    }
    if (a == 0) {
        if (c % b == 0) {
            x = 0, y = c / b, g = abs(b);
            return true;
        }
        return false;
    }
    if (b == 0) {
        if (c % a == 0) {
            x = c / a, y = 0, g = abs(a);
            return true;
        }
        return false;
    }
    g = extgcd(a, b, x, y);
    if (c % g != 0) {
        return false;
    }
    T dx = c / a; c -= dx * a;
    T dy = c / b; c -= dy * b;
    x = dx + (T) ((__int128) x * (c / g) % b);
    y = dy + (T) ((__int128) y * (c / g) % a);
    g = abs(g);
    return true;
    // |x|, |y| <= max(|a|, |b|, |c|) [tested]
}

template<typename T>
void shift_solution(T & x, T & y, T a, T b, T cnt) {
    x += cnt * b, y -= cnt * a;
}

template<typename T>
T find_all_solutions(T a, T b, T c, T minx, T maxx, T miny, T
maxy) {
    T x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g, b /= g;
    T sign_a = a > 0 ? +1 : -1, 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;
    T lx1 = x;
    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    T 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;
    T lx2 = x;
    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    T rx2 = x;
    if (lx2 > rx2)
        swap(lx2, rx2);
    T lx = max(lx1, lx2), rx = min(rx1, rx2);
    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

5.3 Discrete Log

```
// Solve a^x=b (mod M)
ll dlog(ll a, ll b, ll M) {
    map<ll, ll> _hash;
    ll n = euler_phi(M), k = sqrt(n);
    for(ll i = 0, t = 1; i < k; ++i) {
        _hash[t] = i;
        t = mul(t, a, M);
    }
    ll c = pow(a, n - k, M);
    for(ll i = 0; i * k < n; i++) {
        if(_hash.find(b) != _hash.end())
            return i * k + _hash[b];
        b = mul(b, c, M);
    }
    return -1;
}
```

5.4 Discrete Root

```
// Solve x^k=a (mod n)
vector<ll> discrete_root(ll k, ll a, ll n) {
    if (a == 0)
        return {0};

    ll g = primitive_root(n);
    ll sq = (ll) sqrt(n + .0) + 1;
    vector<pair<ll, ll>> dec(sq);
    for (ll i = 1; i <= sq; ++i)
        dec[i - 1] = {pow(g, ll(i * sq * 1ll * k % (n - 1)),
n), i};
    sort(dec.begin(), dec.end());
    ll any_ans = -1;
    for (int i = 0; i < sq; ++i) {
        ll my = ll(pow(g, ll(i * 1ll * k % (n - 1)), n) * 1ll
* a % n);
        auto it = lower_bound(dec.begin(), dec.end(),
make_pair(my, 0ll));
        if (it != dec.end() && it->first == my) {
            any_ans = it->second * sq - i;
            break;
        }
    }
    if (any_ans == -1)
        return {};
    ll delta = (n - 1) / __gcd(k, n - 1);
    vector<ll> ans;
    for (ll cur = any_ans % delta; cur < n - 1; cur += delta)
        ans.push_back(pow(g, cur, n));
    sort(ans.begin(), ans.end());
    return ans;
}
```

5.5 Divisor Sigma

```
ll divisor_sigma(ll n) {
    ll sigma = 0, d = 1;
    for (; d * d < n; ++d)
        if (n % d == 0)
            sigma += d + n / d;
    if (d * d == n) sigma += d;
    return sigma;
}
vector<ll> divisor_sigma(ll lo, ll hi) {
    vector<ll> ps = primes(sqrt(hi) + 1);
    vector<ll> res(hi - lo), sigma(hi - lo, 1);
    iota(res.begin(), res.end(), lo);
    for (ll p : ps)
        for (ll k = ((lo + (p - 1)) / p) * p; k < hi; k += p)
{
            ll b = 1;
            while (res[k - lo] > 1 && res[k - lo] % p == 0) {
                res[k - lo] /= p;
                b = 1 + b * p;
            }
            sigma[k - lo] *= b;
        }
    for (ll k = lo; k < hi; ++k)
        if (res[k - lo] > 1)
            sigma[k - lo] *= (1 + res[k - lo]);
    return sigma; // sigma[k-lo] = sigma(k)
}
```

5.6 Euler Phi

```
ll euler_phi(ll n) {
    if (n == 0) return 0;
    ll ans = n;
    for (ll x = 2; x * x <= n; ++x)
        if (n % x == 0) {
            ans -= ans / x;
            while (n % x == 0)
                n /= x;
        }
    if (n > 1) ans -= ans / n;
    return ans;
}
vector<ll> euler_phi(ll lo, ll hi) {
    vector<ll> ps = primes(sqrt(hi) + 1);
    vector<ll> res(hi - lo), phi(hi - lo, 1);
    iota(res.begin(), res.end(), lo);
    for (ll p : ps)
        for (ll k = ((lo + (p - 1)) / p) * p; k < hi; k += p)
{
            if (res[k - lo] < p) continue;
            phi[k - lo] *= (p - 1);
            res[k - lo] /= p;
            while (res[k - lo] > 1 && res[k - lo] % p == 0) {
                phi[k - lo] *= p;
                res[k - lo] /= p;
            }
        }
    for (ll k = lo; k < hi; ++k)
        if (res[k - lo] > 1)
            phi[k - lo] *= (res[k - lo] - 1);
    return phi; // phi[k-lo] = phi(k)
}
```

5.7 Extended GCD

```
template<typename T>
T extgcd(T a, T b, T &x, T &y) {
    if (a == 0) {
        x = 0, y = 1;
        return b;
    }
    T p = b / a;
    T g = extgcd(b - p * a, a, y, x);
    x -= p * y;
    return g;
}
```

5.8 Floor Sum

```
// sum of (A * i + B) / M
ll solve(ll N, ll M, ll A, ll B) { assert(M > 0);
    if (B < 0) {
        ll at_least = (-B + M - 1) / M;
        return solve(N, M, A, B + at_least * M) - N * at_least;
    }
    if (A == 0) return (B / M) * N;
    assert(0 <= B && B < M);
    if (A >= M) {
        return solve(N, M, A % M, B)+(A / M)*((N * (N - 1)) /
2);
    }
    ll up = (A * (N - 1) + B) / M; ll val = N * up;
    val -= (B / A);
    return val - solve(up + 1, A, M, A - 1 - B);
}
```

5.9 Floyd Cycle Finding

```
pair<int,int> find_cycle() {
    int t = f(x0), h = f(t), mu = 0, lam = 1;
    while (t != h) t = f(t), h = f(f(h));
    h = x0;
    while (t != h) t = f(t), h = f(h), mu++;
    h = f(t);
    while (t != h) h = f(h), lam++;
    return {mu, lam};
}
```

5.10 Harmonic Partition

```
vector<pair<ll,ll>> HarmonicSeriesPartition(ll N) {
    vector<pair<ll,ll>> res;
    ll temp = 0;
    for(ll i = 1, la ; i <= N ; i = la+1) {
        la = N/(N/i);
        res.push_back({temp+1,la});
        temp = la;
    }
    return res;
}
```

5.11 Linear Congruence

```
//  Solve x=ai(mod mi), for any i and j, (mi,mj)|ai-aj
//  Return (x0,M) M=[m1..mn]. All solutions are x=x0+t*M
//  Note: be carful with the overflow in the multiplication
pair<ll, ll> linear_congruences(const vector<ll> &a, const
vector<ll> &m) {
    int n = a.size();
    ll u = a[0], v = m[0], p, q;
    for (int i = 1; i < n; ++i) {
        ll r = gcd(v, m[i], p, q);
        ll t = v;
        if ((a[i] - u) % r)
            return {-1, 0}; // no solution
        v = v / r * m[i];
        u = ((a[i] - u) / r * p * t + u) % v;
    }
    if (u < 0)
        u += v;
    return {u, v};
}
```

5.12 Miller Rabin

```
bool witness(ll a, ll s, ll d, ll n) {
    ll x = pow(a, d, n);
    if (x == 1 || x == n - 1) return 0;
    for (int i = 0; i < s - 1; i++) {
        x = mul(x, x, n);
        if (x == 1) return 1;
        if (x == n - 1) return 0;
    }
    return 1;
}
bool miller_rabin(ll n) {
    if (n < 2) return 0;
    if (n == 2) return 1;
    if (n % 2 == 0) return 0;
    ll d = n - 1, s = 0;
    while (d % 2 == 0)++s, d /= 2;
    vector<ll> test = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
37};
    for (ll p : test)
        if (p >= n) break;
        else if (witness(p, s, d, n)) return 0;
    return 1;
}
```

5.13 Pollard Rho

```
// Note: n shouldn't be prime
ll pollard_rho(ll n) {
    if (!(n & 1)) return 2;
    while (1) {
        ll x = (ll) rand() % n, y = x;
        ll c = rand() % n;
        if (c == 0 || c == 2) c = 1;
        for (int i = 1, k = 2;; i++) {
            x = mul(x, x, n);
            if (x >= c) x -= c; else x += n - c;
            if (x == n) x = 0;
            if (x == 0) x = n - 1; else x--;
            ll d = __gcd(x > y ? x - y : y - x, n);
            if (d == n) break;
            if (d != 1) return d;
            if (i == k) { y = x; k <<= 1;}
        }
    }
    return 0;
}
```

5.14 Mobius Mu

```
ll mobius_mu(ll n) {
    if (n == 0) return 0;
    ll mu = 1;
    for (ll x = 2; x * x <= n; ++x)
        if (n % x == 0) {
            mu = -mu, n /= x;
            if (n % x == 0) return 0;
        }
    return n > 1 ? -mu : mu;
}
vector<ll> mobius_mu(ll lo, ll hi) {
    vector<ll> ps = primes(sqrt(hi) + 1);
    vector<ll> res(hi - lo), mu(hi - lo, 1);
    iota(res.begin(), res.end(), lo);
    for (ll p : ps)
        for (ll k = ((lo + (p - 1)) / p) * p; k < hi; k += p)
{
            mu[k - lo] = -mu[k - lo];
            if (res[k - lo] % p == 0) {
                res[k - lo] /= p;
                if (res[k - lo] % p == 0) {
                    mu[k - lo] = 0, res[k - lo] = 1;
                }
            }
        }
    for (ll k = lo; k < hi; ++k)
        if (res[k - lo] > 1)
            mu[k - lo] = -mu[k - lo];
    return mu; // mu[k-lo] = mu(k)
}
```

5.15 Primitive Root

```
// Note: Only 2, 4, p^n, 2p^n have primitive roots
ll primitive_root(ll m) {
    if (m == 1) return 0;
    if (m == 2) return 1;
    if (m == 4) return 3;
    auto pr = primes(0, sqrt(m) + 1); // fix upper bound
    ll t = m;
    if (!(t & 1))
        t >>= 1;
    for (ll p : pr) {
        if(p > t) break;
        if (t % p) continue;
        do t /= p;
        while (t % p == 0);
        if (t > 1 || p == 2) return 0;
    }
    ll x = euler_phi(m), y = x, n = 0;
    vector<ll> f(32);
    for (ll p : pr) {
        if (p > y) break;
        if (y % p) continue;
        do y /= p;
        while (y % p == 0);
        f[n++] = p;
    }
    if (y > 1) f[n++] = y;
    for (ll i = 1; i < m; ++i) {
        if (__gcd(i, m) > 1) continue;
        bool flag = 1;
        for (ll j = 0; j < n; ++j) {
            if (pow(i, x / f[j], m) == 1) {
                flag = 0;
                break;
            }
        }
        if (flag) return i;
    }
    return 0;
}
```

5.16 Sieve

```
struct SieveOfEratosthenes {
    vector<int> primes, least, cnt, pw, phi, sigma, mu;
    SieveOfEratosthenes(int N) {
        if(N < 1)N = 1;
        least.resize(N+1), cnt.resize(N+1), pw.resize(N+1);
        phi.resize(N+1), sigma.resize(N+1), mu.resize(N+1);
        pw[1] = 0;
        least[1] = cnt[1] = phi[1] = sigma[1] = mu[1] = 1;
        for(int i = 4 ; i <= N ; i+=2) least[i] = 2;
        for(int i = 3 ; i*i <= N ; i+=2) {
            if(least[i] == 0) {
                for(int j = i*i ; j <= N ; j+=i*2) {
                    if(least[j] == 0)least[j] = i;
                }
            }
        }
        for(int i = 2 ; i <= N ; i++) {
            if(least[i] == 0) {
                primes.push_back(i);
                least[i] = i, cnt[i] = i, pw[i] = 1;
                phi[i] = i-1, sigma[i] = i+1, mu[i] = -1;
            } else {
                int x = i/least[i];
                if(least[i] == least[x])cnt[i] =
cnt[x]*least[i], pw[i] = pw[x]+1;
                else cnt[i] = least[i], pw[i] = 1;
                if(i == cnt[i]) {
                    phi[i] = cnt[i]-cnt[i]/least[i];
                    sigma[i] =
(cnt[i]*least[i]-1)/(least[i]-1);
                    mu[i] = 0;
                } else {
                    phi[i] = phi[i/cnt[i]]*phi[cnt[i]];
                    sigma[i] = sigma[i/cnt[i]]*sigma[cnt[i]];
                    mu[i] = mu[i/cnt[i]]*mu[cnt[i]];
                }
            }
        }
    }
};
```

6 Numerical

6.1 Berlekamp Massey

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int mod = 1e9 + 7;
void add_self(int & a, int b) { a += b; if(a >= mod) a -= mod;
}
void sub_self(int & a, int b) { a -= b; if(a < 0) a += mod; }
int mul(int a, int b) { return (ll) a * b % mod; }
int my_pow(int a, int b) {
    int r = 1;
    while(b) {
        if(b & 1) r = mul(r, a);
        a = mul(a, a);
        b >>= 1;
    }
    return r;
}
int my_inv(int a) { return my_pow(a, mod - 2); }

struct Massey { // Berlekamp-Massey by Errichto / Warsaw U
    vector<int> start, coef; // 3 optional lines
    vector<vector<int>> powers;
    int memo_inv;
    // Start here and write the next ~25 lines until "STOP"

    int L; // L == coef.size() <= start.size()
    Massey(vector<int> in) { // O(N^2)
        L = 0;
        const int N = in.size();
        vector<int> C{1}, B{1};
        for(int n = 0; n < N; ++n) {
            assert(0 <= in[n] && in[n] < mod); // invalid
input
            B.insert(B.begin(), 0);
            int d = 0;
            for(int i = 0; i <= L; ++i)
                add_self(d, mul(C[i], in[n-i]));
            if(d == 0) continue;
            vector<int> T = C;
            C.resize(max(B.size(), C.size()));
            for(int i = 0; i < (int) B.size(); ++i)
                sub_self(C[i], mul(d, B[i]));
            if(2 * L <= n) {
                L = n + 1 - L;
                B = T;
                d = my_inv(d);
                for(int & x : B) x = mul(x, d);
            }
        }
        assert(2 * L <= N - 2); // NO RELATION FOUND :(
        // === STOP ===
        for(int i = 1; i < (int) C.size(); ++i)
            coef.push_back((mod - C[i]) % mod);
        assert((int) coef.size() == L);
        for(int i = 0; i < L; ++i)
            start.push_back(in[i]);
        while(!coef.empty() && !coef.back()) { coef.pop_back();
--L; }
        if(!coef.empty()) memo_inv = my_inv(coef.back());
        powers.push_back(coef);
    }

    vector<int> mul_cut(vector<int> a, vector<int> b) {
        vector<int> r(2 * L - 1);
        for(int i = 0; i < L; ++i)
            for(int j = 0; j < L; ++j)
                add_self(r[i+j], mul(a[i], b[j]));
        while((int) r.size() > L) {
            int value = mul(r.back(), memo_inv); //
div(r.back(), coef.back());
            const int X = r.size();
            add_self(r[X-L-1], value);
            for(int i = 0; i < L; ++i)
                sub_self(r[X-L+i], mul(value, coef[i]));
            assert(r.back() == 0);
            r.pop_back();
        }
        return r;
    }
    int get(ll k) { // O(L^2 * log(k))
        if(k < (int) start.size()) return start[k];
        if(L == 0) return 0;
        k -= start.size();
        vector<int> vec = coef;
        for(int i = 0; (1LL << i) <= k; ++i) {
            if(i == (int) powers.size())
                powers.push_back(mul_cut(powers.back(),
powers.back()));
            if(k & (1LL << i))
                vec = mul_cut(vec, powers[i]);
        }
        int total = 0;
        for(int i = 0; i < L; ++i)
            add_self(total, mul(vec[i],
start[(int)start.size()-1-i]));
        return total;
    }
};
int main() {
    // f[n] = 3 * f[n-1] + f[n-3] ---> coef: [3, 0, 1]
    vector<int> in{10, 0, 1, 0, 0, 1, 3, 9, 28, 87};
    Massey massey(in);
    for(int i = 0; i < 30; ++i) printf("%d ", massey.get(i));
    puts(""); // 10 0 ... 951398949 883208606 modulo 1e9+7
}
```

6.2 FFT

```
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
  int n = a.size(), L = 31 - __builtin_clz(n);
  static vector<complex<long double>> R(2, 1);
  static vector<C> rt(2, 1);  // (^ 10% faster if double)
  for (static int k = 2; k < n; k *= 2) {
    R.resize(n); rt.resize(n);
    auto x = polar(1.0L, acos(-1.0L) / k);
    for(int i = k; i < 2*k; i++)
      rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
  }
  vector<int> rev(n);
  for(int i = 0; i < n; i++)
    rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
  for(int i = 0; i < n; i++)
    if (i < rev[i])
      swap(a[i], a[rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k)
      for(int j = 0; j < k; j++) {
        // C z = rt[j+k] * a[i+j+k]; // (25% faster if
hand-rolled)  /// include-line
        auto x = (double *)&rt[j+k], y = (double *)&a[i+j+k];
/// exclude-line
      C z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);
/// exclude-line
```

```
      a[i + j + k] = a[i + j] - z;
      a[i + j] += z;
    }
}
vector<double> conv(const vector<double>& a, const
vector<double>& b) {
  if (a.empty() || b.empty()) return {};
  vector<double> res((int)a.size() + (int)b.size() - 1);
  int L = 32 - __builtin_clz(res.size()), n = 1 << L;
  vector<C> in(n), out(n);
  copy(all(a), begin(in));
  for(int i = 0; i < (int)b.size(); i++)
    in[i].imag(b[i]);
  fft(in);
  for (C& x : in) x *= x;
  for(int i = 0; i < n; i++)
    out[i] = in[-i & (n - 1)] - conj(in[i]);
  fft(out);
  for(int i = 0; i < (int)res.size(); i++)
    res[i] = imag(out[i]) / (4 * n);
  return res;
}
```

6.3 FFT Mod

```
template<int M> vector<ll> convMod(const vector<ll> &a, const
vector<ll> &b) {
  if (a.empty() || b.empty()) return {};
  vector<ll> res(a.size() + b.size() - 1);
  int B=32-__builtin_clz(res.size()), n=1<<B, cut=int(sqrt(M));
  vector<C> L(n), R(n), outs(n), outl(n);
  for(int i = 0; i < (int)a.size(); i++)
    L[i] = C((int)a[i] / cut, (int)a[i] % cut);
  for(int i = 0; i < (int)b.size(); i++)
    R[i] = C((int)b[i] / cut, (int)b[i] % cut);
  fft(L), fft(R);
  for(int i = 0; i < n; i++){
    int j = -i & (n - 1);
    outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
    outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
  }
  fft(outl), fft(outs);
  for(int i = 0; i < (int)res.size(); i++) {
    ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
    ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
    res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
  }
  return res;
}
```

6.4 FWHT

```
// Notes: if you use mod make sure 0 <= a[i], b[i] < mod when
you call convolve
enum bit_op { AND, OR, XOR };

namespace bitwise_transform {
    template<int P, typename T>
    inline void add(T &x, T y) {
        x += y;
        if (P != 0 && x >= P) x -= P;
    }
    template<bit_op B, int P, bool inv = false, typename T>
    void transform(T a[], int n) {
        for (int len = 1; len < n; len <<= 1)
            for (int i = 0; i < n; i += len << 1)
                for (int j = i; j < i + len; ++j) {
                    T u = a[j], v = a[j + len];
                    if (B == AND) add<P>(a[j], inv ? P-v : v);
                    if (B == OR) add<P>(a[j + len], inv ? P-u
: u);
                    if (B == XOR)
                        add<P>(a[j], v),
                            add<P>(a[j + len] = u, P-v);
                }
        if (B == XOR && inv) {
            int in = pow_mod(n, P-2, P);


            for (int i = 0; i < n; ++i) {
                if (P == 0) a[i] /= n;
                else a[i] = (ll)a[i] * in % P;
            }
        }
    }
    template<bit_op B, int P = 0, typename T>
    vector<T> convolve(vector<T> a, vector<T> b) {
        int n = max(a.size(), b.size()), sz = 1;
        while (sz < n) sz <<= 1;
        a.resize(sz);
        b.resize(sz);
        transform<B, P>(a.data(), sz);
        transform<B, P>(b.data(), sz);
        for (int i = 0; i < sz; ++i) {
            if (P == 0) a[i] *= b[i];
            else a[i] = (ll)a[i] * b[i] % P;
        }
        transform<B, P, true>(a.data(), sz);
        return a;
    }
}
```

6.5 Gaussian Elimination

```
int GAUSSIAN_ELIMINATION(vector<vector<ld>> A, vector<ld> &ans)
{
    const ld EPS = 1e-9;
    int N = (int)A.size(), M = (int)A[0].size()-1;
    vector<int> where(M, -1);
    for(int col = 0, row = 0 ; col < M && row < N ; col++) {
        int sel = row;
        for(int i = row ; i < N ; i++) {
            if(abs(A[i][col]) > abs(A[sel][col]))
                sel = i;
        }
        if(abs(A[sel][col]) < EPS)
            continue;
        swap(A[sel], A[row]);
        where[col] = row;
        for(int i = row+1 ; i < N ; i++) {
            ld c = A[i][col]/A[row][col];
            for(int j = col ; j <= M ; j++)
                A[i][j] -= A[row][j]*c;
        }
        row++;
    }
    ans.assign(M, 0);
    vector<ld> vect(N);
    for(int i = 0 ; i < N ; i++)
        vect[i] = A[i][M];
    for(int j = M-1 ; j >= 0 ; j--) {
        if(where[j] != -1)
            ans[j] = vect[where[j]]/A[where[j]][j];
        for(int i = 0 ; i < N ; i++)
            vect[i] -= ans[j]*A[i][j];
    }
    for(int i = 0 ; i < N ; i++) {
        ld sum = 0;
        for(int j = 0 ; j < M ; j++)
            sum += ans[j]*A[i][j];
        if(abs(sum - A[i][M]) > EPS)
            return 0;
    }
    for(int i = 0 ; i < M ; i++)
        if(where[i] == -1)
            return 2;
    return 1;
}
```

6.6 Golden Section Search

```
/*** Usage:
  double func(double x) { return 4+x+.3*x*x; }
  double xmin = gss(-1000,1000,func);
 */
const double gold = (sqrt(5) - 1) / 2;
const double eps = 1e-7;
/// It is important for gold to be precise, otherwise we don't
necessarily maintain the inequality a < x1 < x2 < b.
double gss(double a, double b, double (*f)(double)) {
  double x1 = b - gold*(b-a), x2 = a + gold*(b-a);
  double f1 = f(x1), f2 = f(x2);
  while (b-a > eps)
    if (f1 < f2) { //change to > to find maximum
      b = x2; x2 = x1; f2 = f1;
      x1 = b - gold*(b-a); f1 = f(x1);
    } else {
      a = x1; x1 = x2; f1 = f2;
      x2 = a + gold*(b-a); f2 = f(x2);
    }
  return a;
}
```

6.7 Lagrange Interpolation

```
ll lagrange_interpolation_arithmetic
    (const vector <ll> &y, ll T, ll a = 0, ll d = 1) {
    T = mod(T), a = mod(a), d = mod(d);
    int N = (int) y.size() - 1;
    if (a == 0 && d == 1 && T <= N) return y[T];
    vector <ll> dp(N + 1, 1), pd(N + 1, 1);
    for (int i = 0; i < N; i++)
        dp[i + 1] = dp[i] * (T + MOD - (a + d * i) % MOD) %
MOD;
    for (int i = N; i > 0; i--)
        pd[i - 1] = pd[i] * (T + MOD - (a + d * i) % MOD) %
MOD;
    vector <ll> fact(N + 1, 1), finv(N + 1, 1);
    for (int i = 1; i <= N; i++)
        fact[i] = fact[i - 1] * d % MOD * i % MOD;
    finv[N] = finv[N] * inv(fact[N]) % MOD;
    for (int i = N; i >= 1; i--)
        finv[i - 1] = finv[i] * d % MOD * i % MOD;
    ll ret = 0;
    for (int i = 0; i <= N; i++) {
        ll tmp = y[i] * dp[i] % MOD * pd[i] % MOD *
                finv[i] % MOD * finv[N - i] % MOD;
        if ((N - i) & 1)ret = (ret + MOD - tmp) % MOD;
        else ret = (ret + tmp) % MOD;
    }
    return ret;
}
ll lagrange_interpolation(vector <pair<ll, ll>> p, ll z) {
    int n = p.size(); ll l, val = 0;
    for (int i = 0; i < n; i++) {
        p[i].first = mod(p[i].first);
        p[i].second = mod(p[i].second);
    }
    z = mod(z);
    for (int i = 0; i < n; i++) {
        l = p[i].second;
          for (int j = 0; j < n; j++)
            if (i != j)
                l = l * (z + MOD - p[j].first) % MOD *
                    inv(p[i].first + MOD - p[j].first) % MOD;
        val = (val + l) % MOD;
    }
    return val;
}
```

6.8 Newton's Method

```
template<class F, class G>
long double NEWTON_METHOD(F f, G df, long double x) {
    for (int i = 0; i < 100; i++) {
        long double fx = f(x), dfx = df(x);
        x = x - fx / dfx;
        if (abs(f(x)) < 1e-12) break;
    }
    return x;
}
```

6.9 NTT

```
ll modpow(ll b, ll e) {
  ll ans = 1;
  for (; e; b = b * b % mod, e >>= 1)
    if (e & 1) ans = ans * b % mod;
  return ans;
}

const ll mod = (119 << 23) + 1, root = 62; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
void ntt(vector<ll> &a) {
  int n = (int)a.size(), L = 31 - __builtin_clz(n);
  static vector<ll> rt(2, 1);
  for (static int k = 2, s = 2; k < n; k *= 2, s++) {
    rt.resize(n);
    ll z[] = {1, modpow(root, mod >> s)};
    for(int i = k; i < 2*k; i++)
      rt[i] = rt[i / 2] * z[i & 1] % mod;
  }
  vector<int> rev(n);
  for(int i = 0; i < n; i++)
    rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
  for(int i = 0; i < n; i++)
    if (i < rev[i])
      swap(a[i], a[rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k)
      for(int j = 0; j < k; j++) {
        ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
        a[i + j + k] = ai - z + (z > ai ? mod : 0);
        ai += (ai + z >= mod ? z - mod : z);
      }
}
vector<ll> conv(const vector<ll> &a, const vector<ll> &b) {
  if (a.empty() || b.empty()) return {};
  int s = (int)a.size() + (int)b.size() - 1, B = 32 -
__builtin_clz(s), n = 1 << B;
  int inv = modpow(n, mod - 2);
  vector<ll> L(a), R(b), out(n);
  L.resize(n), R.resize(n);
  ntt(L), ntt(R);
  for(int i = 0; i < n; i++)
    out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
  ntt(out);
  return {out.begin(), out.begin() + s};
}
```

6.10 Simplex

```
const double eps = 1e-9, oo =
numeric_limits<double>::infinity();
typedef vector<double> vec; typedef vector <vec> mat;
double SIMPLEX_METHOD_PD(vector <vector<double>> &A,
     vector<double> &b, vector<double> &c, vector<double> &x) {
    int n = c.size(), m = b.size();
    vector <vector<double>> T(m + 1, vector<double>(n + m +
1));
    vector<int> base(n + m), row(m); x.clear();
    for (int j = 0; j < m; j++) {
        for (int i = 0; i < n; i++)
            T[j][i] = A[j][i];
        T[j][n + j] = 1; base[row[j] = n + j] = 1;
        T[j][n + m] = b[j];
    }
    for (int i = 0; i < n; i++) T[m][i] = c[i];
    while (1) {
        int p = 0, q = 0;
        for (int i = 0; i < n + m; i++)
            if (T[m][i] <= T[m][p]) p = i;
        for (int j = 0; j < m; j++)
            if (T[j][n + m] <= T[q][n + m]) q = j;
        double t = min(T[m][p], T[q][n + m]);
        if (t >= -eps) {
            x.resize(n);
            for (int i = 0; i < m; i++)
                if (row[i] < n)x[row[i]] = T[i][n + m];
            // x is the solution
            return -T[m][n + m]; // optimal
        }
        if (t < T[q][n + m]) { // tight on c -> primal update
            for (int j = 0; j < m; j++)
                if (T[j][p] >= eps)
                    if (T[j][p] * (T[q][n + m] - t) >=
                        T[q][p] * (T[j][n + m] - t)) q = j;
            if (T[q][p] <= eps) return oo;//primal infeasible
        } else { // tight on b -> dual update
            for (int i = 0; i < n + m + 1; i++)
                T[q][i] = -T[q][i];
            for (int i = 0; i < n + m; i++)
                if (T[q][i] >= eps)
                    if (T[q][i] * (T[m][p] - t)
                        >= T[q][p] * (T[m][i] - t)) p = i;
            if (T[q][p] <= eps) return -oo;//dual infeasible
        }
        for (int i = 0; i < m + n + 1; i++)
            if (i != p) T[q][i] /= T[q][p];
        T[q][p] = 1; // pivot(q, p)
        base[p] = 1; base[row[q]] = 0; row[q] = p;
        for (int j = 0; j < m + 1; j++)
            if (j != q) {
                double alpha = T[j][p];
                for (int i = 0; i < n + m + 1; i++)
                    T[j][i] -= T[q][i] * alpha;
            }
    } return oo;
}
bool SIMPLEX_METHOD_INTEGER(vector <vector<double>> &A,
    vector<double> &B, vector<double> &C, long long &best,
    vector<long long> &solution) {
    vector<double> x; double v = SIMPLEX_METHOD_PD(A, B, C, x);
    if (v == oo || v == -oo) return false;// Infeasible
    if ((long long) ceil(v) >= best)return true;
    for (int i = 0; i < (int) x.size(); i++) {
        double a = floor(x[i]);
        double b = ceil(x[i]);
        if (min(x[i] - a, b - x[i]) >= eps) {
            vector <vector<double>> NA = A;
            vector<double> NB = B; int vars = C.size();
            vector<double> nv(vars); nv[i] = -1;
            NA.push_back(nv); NB.push_back(-b);
            bool
ok=SIMPLEX_METHOD_INTEGER(NA,NB,C,best,solution);
            NA.pop_back(); NB.pop_back();
            nv[i] = 1; NA.push_back(nv); NB.push_back(a);
            ok|=SIMPLEX_METHOD_INTEGER(NA,NB,C,best,solution);
            return ok;
        }
    }
    //Solution stored in x, assume that it will be integer.
    long long cur_value = (long long) round(v);
    if (cur_value < best) {
        best = cur_value;
        solution = vector<long long>(x.begin(), x.end());
    }
    return true;
}
```

6.11 Simpson's Rule

```
template<class F>
double simpson(F f, double a, double b, int n = 2000) {
    double h = (b - a) / (2 * n), fa = f(a), nfa, res = 0;
    for (int i = 0; i < n; ++i, fa = nfa) {
        nfa = f(a + 2 * h);
        res += (fa + 4 * f(a + h) + nfa); a += 2 * h;
    }
    res = res * h / 3; return res;
}
```

6.12 Xor Basis

```
struct XOR_BASIS {
    vector <bitset<__sz_bset>> basis; int sz;int D;
    XOR_BASIS() { init(0); }
    XOR_BASIS(int n) { init(n); }
    void init(int n) {
        D = n; sz = 0;
        basis.clear(); basis.resize(D);
    }
    void insert(bitset <__sz_bset> mask) {
        for (int i = 0; i < D; i++) {
            if (mask[i] == 0)continue;
            if (basis[i].none()) {
                basis[i] = mask;
                sz++; return;
            }
            mask ^= basis[i];
        }
    }
};
```

7 Strings

7.1 Aho-Corasick

```
template<typename T, typename F>
struct AHO_CORASICK {
    vector<pair<int,int>> lim;
    vector<vector<int>> glink, words;
    vector<unordered_map<F,pair<int,bool>>> tgo;
    vector<int> pt_string, dict_link, vtime, link, abi, ch, p;
    int cant_string = 0, int sz = 0;
    bool ok_match = false, ok_link_tree = false;
    void init() {
        lim.clear(), glink.clear(), words.clear(),
tgo.clear(), pt_string.clear();
        dict_link.clear(), vtime.clear(), link.clear(),
abi.clear(), ch.clear(), p.clear();
        cant_string = 0, sz = 0;
        ok_match = false, ok_link_tree = false;
        new_node(-1,-1);
    }
    void build(vector<T> &vect) {
        init();
        for(int i = 0 ; i < vect.size() ; i++)
            add_string(vect[i]);
    }
    int size() {return sz;}
    void new_node(int tp, int tch) {
        sz++;
        tgo.emplace_back(), dict_link.push_back(-1),
words.emplace_back();
        link.push_back(-1), ch.push_back(tch),
p.push_back(tp);
    }
    AHO_CORASICK() { init(); }
    void add_string(T cad) {
        int pt = 0;
        for(int i = 0 ; i < cad.size() ; i++) {
            F tch = cad[i];
            auto it = &tgo[pt][tch];
            if((*it).second == false) {
                (*it).second = true;
                (*it).first = sz;
                new_node(pt, tch);
            }
            pt = (*it).first;
        }
        words[pt].push_back(pt_string.size());
        pt_string.push_back(pt);
    }
    void build_link_tree() {
        ok_link_tree = true;
        glink.resize(sz);
        for(int i = 1 ; i < sz ; i++) {
            glink[get_link(i)].push_back(i);
        }
    }
    void build_match() {
        ok_match = true;
        if(ok_link_tree == false)build_link_tree();
        vtime.resize(sz), lim.resize(sz), abi.resize(sz+1);
        int ct = 0;
        function<void(int)> DFS = [&](int nodo) {
            vtime[nodo] = ++ct;
            for(auto v : glink[nodo]) DFS(v);
            lim[nodo] = {vtime[nodo], ct};
        };
        DFS(0);
    }
    void abi_update(int x, int v) {
        while(x < abi.size()) {
            abi[x] += v; x += x&-x;
        }
    }
    int abi_query(int x) {
        int res = 0;
        while(x) {
            res += abi[x]; x -= x&-x;
        }
        return res;
    }
    int abi_range(int a, int b) {
        return abi_query(b) - abi_query(a-1);
    }
    vector<int> match(vector<T> vcad) {
        if(ok_match == false)build_match();
        vector<int> ups;
        for(auto cad : vcad) {
            int pt = 0;
            for(int i = 0 ; i < cad.size() ; i++) {
                pt = go(pt, cad[i]);
                abi_update(vtime[pt], 1);
                ups.push_back(vtime[pt]);
            }
        }
        vector<int> res(pt_string.size());
        for(int i = 0 ; i < pt_string.size() ; i++)
            res[i] = abi_range(lim[pt_string[i]].first,
lim[pt_string[i]].second);
        for(auto x : ups)
            abi_update(x, -1);
        return res;
    }
    vector<int> match_offline(vector<T> vcad) {
        if(ok_link_tree == false)build_link_tree();
        vector<int> dp(sz);
        for(auto cad : vcad) {
            int pt = 0;
            for(int i = 0 ; i < cad.size() ; i++)
                pt = go(pt, cad[i]), dp[pt]++;
        }
        function<void(int)> DFS = [&](int nodo) {
            for(auto v : glink[nodo]) {
                DFS(v), dp[nodo] += dp[v];
            }
        };
        DFS(0);
        vector<int> res(pt_string.size());
        for(int i = 0 ; i < pt_string.size() ; i++)
            res[i] = dp[pt_string[i]];
        return res;
    }
    int get_dictionary_link(int nodo) {
        if(dict_link[nodo] == -1) {
            if(nodo == 0 || p[nodo] == 0)dict_link[nodo] = 0;
            else {
                int v = get_link(nodo);
                if(words[v].size())dict_link[nodo] = v;
                else dict_link[nodo] = get_dictionary_link(v);
            }
        }
        return dict_link[nodo];
    }
    int get_link(int nodo) {
        if(link[nodo] == -1) {
            if(nodo == 0 || p[nodo] == 0)link[nodo] = 0;
            else link[nodo] = go(get_link(p[nodo]), ch[nodo]);
        }
        return link[nodo];
    }
    int go(int nodo, F tch) {
        auto it = &tgo[nodo][tch];
        if((*it).second == false) {
            (*it).second = true;
            (*it).first = (nodo == 0) ? 0 :
go(get_link(nodo),tch);
        }
        return (*it).first;
    }
};
```

7.2 Hashing

```
// Hashing mod 2^64-1 (Approved by kactl and errichto and
tested)
// Use using H = unsigned long long; instead of H for mod 2^64
// Two hashes modulo primes below
// Use .get() instead of actual values to get the hashes
typedef unsigned long long ull;
struct H {
    ull x; H(ull x = 0) : x(x) {}
    H operator +(H o) { return x + o.x + (x + o.x < x); }
    H operator -(H o) { return *this + ~o.x; }
    H operator *(H o) { auto m = (__uint128_t)x * o.x;
        return H((ull)m) + (ull)(m >> 64); }
    ull get() const { return x + !~x; }
    bool operator ==(H o) const { return get() == o.get(); }
    bool operator <(H o) const { return get() < o.get(); }
};
static const H BASE = (ll)1e11+3; // (order ~ 3e9; random also
ok)

struct HashInterval {
    vector<H> ha, pw;
    HashInterval(string& str) : ha(str.size()+1), pw(ha) {
        pw[0] = 1;
        for (int i = 0; i < str.size(); i ++) {
            ha[i + 1] = ha[i] * BASE + str[i];
            pw[i + 1] = pw[i] * BASE;
        }
    }
    H hashInterval(int a, int b) { // hash [a, b]
        return ha[b + 1] - ha[a] * pw[b - a + 1];
    }
};
H hashString(string& s){ H h;
    for(char c : s) h = h * BASE + c;
    return h;
}
// struct H for two hashes mod 2 primes
// use ull() to get values of hash in unsigned long long
template<int M, class B>
struct HA {
    int x; B b; HA(int x=0) : x(x), b(x) {}
    HA(int x, B b) : x(x), b(b) {}
    HA operator+(HA o){int y = x+o.x; return{y - (y>=M)*M,
b+o.b};}
    HA operator-(HA o){int y = x-o.x; return{y + (y< 0)*M,
b-o.b};}
    HA operator*(HA o) { return {(int)(1LL*x*o.x % M), b*o.b};
}
    explicit operator ull() { return x ^ (ull) b << 21; }
    bool operator==(HA o) const { return (ull)*this == (ull)o;
}
    bool operator<(HA o) const { return (ull)*this < (ull)o; }
};
typedef HA<1000000007, HA<1000000009, unsigned>> H;
static const H BASE(311, HA<1000000009, unsigned>(20003));
```

7.3 KMP

```
// pref[i] = the longest prefix of s that is a suffix of
s[0...i]
vector<int> prefix_function(const string &p) {
    int n = p.length();
    vector<int> pref(n + 1);
    for (int i = 0, j = pref[0] = -1; i < n; pref[++i] = ++j)
        while (j >= 0 && p[i] != p[j]) j = pref[j];
    return pref;
}
vector<int> knuth_morris_pratt(const string &s, const string
&p) {
    int n = s.length(), m = p.length();
    vector<int> pref = prefix_function(p), matches;
    for (int i = 0, j = 0; i < n; ++i) {
        while (j >= 0 && s[i] != p[j]) j = pref[j];
        if (++j == m)
            matches.push_back(i - m + 1), j = pref[j];
    }
    return matches;
}
```

7.4 Manacher

```
/** manacher[i] = length of the longest palindrome:
 of odd size and center i/2 if i is even
 of even size and centers i/2 and (i+1)/2 if i is odd*/
struct Manacher {
    vector<int> rad; vector<pair<int,int>> pal;
    Manacher(const T &s) {
        int N = 2*s.size(); rad.resize(N);
        for(int i = 0, j = 0, k ; i < N ; i += k, j =
max(j-k,0ll)) {
            for(; i >= j && i+j+1 < N && s[(i-j)/2] ==
s[(i+j+1)/2] ; j++);
            rad[i] = j;
            for(k = 1 ; i >= k && rad[i] >= k && rad[i-k] !=
rad[i]-k ; k++)
                rad[i+k] = min(rad[i-k],rad[i]-k);
        }
        for(int i = 0 ; i < N ; i++)

if(rad[i])pal.push_back({(i-rad[i]+1)/2,(i+rad[i]-1)/2});
    }
    bool is_pal(int b, int e) {
        if(b > e)swap(b,e);
        int n = rad.size()/2;
        return b >= 0 && e < n && rad[b+e] >= e-b+1;
    }
};
```

7.5 Suffix Array

```
/* Suffix array + lcp O(n log n) 1-indexed with trash at 0
 * To compute SA of a vector, put -INFINITY at the end
 * and change string to vector
 * lcp[i] = lcp(s[sa[i-1]...], s[sa[i]...]) */
struct SA {
    int n;
    vector<int> sa, rank, lcp;
    SA(const string &s) :
            n(s.size() + 1), sa(n), rank(n), lcp(n) {
        vector<int> tmpSa(n), bucket(n);
        iota(sa.rbegin(), sa.rend(), 0);
        sort(next(sa.begin()), sa.end(),
             [&](int i,int j) { return s[i] < s[j]; });
        for (int i = 1, j = 0; i < n; i ++) {
            rank[sa[i]] = rank[sa[i - 1]] +
                          (i == 1 || s[sa[i - 1]] < s[sa[i]]);
            if (rank[sa[i]] != rank[sa[i - 1]])
                bucket[++j] = i;
        }
        for (int len = 1; len <= n; len += len) {
            for (int i = 0; i < n; i ++) {
                int j = sa[i] - len;
                if (j < 0) j += n;
                tmpSa[bucket[rank[j]] ++] = j;
            }
            bucket[0] = sa[tmpSa[0]] = 0;
            for (int i = 1, j = 0; i < n; i ++) {
                if (rank[tmpSa[i]] != rank[tmpSa[i - 1]] ||
                    rank[tmpSa[i] + len] != rank[tmpSa[i - 1]
+ len])
                    bucket[++j] = i;
                sa[tmpSa[i]] = j;
            }
            copy(sa.begin(), sa.end(), rank.begin());
            sa.swap(tmpSa);
            if (rank[sa[n - 1]] == n - 1) break;
        }
        for (int i = 0, j = rank[0], k = 0; i < n - 1; i ++, k
++) {
            while (k >= 0 && s[i] != s[sa[j - 1] + k])
                lcp[j] = k--, j = rank[sa[j] + 1];
        }
    }
};
```

7.6 Z-Function

```
// z[i] = length of the longest common prefix of s and s[i..n]
vector<int> zfunction(const string &s) {
    int n = s.length();
    vector<int> z(n, n);
    for (int i = 1, g = 0, f; i < n; ++i)
        if (i < g && z[i - f] != g - i)
            z[i] = min(z[i - f], g - i);
        else {
            for (g = max(g, f = i); g < n && s[g] == s[g - f];
++g);
            z[i] = g - f;
        }
    return z;
}
```

8 Geometry

8.1 Basics

```
typedef complex<double> point;
typedef vector <point> polygon;
#define NEXT(i) (((i) + 1) % n)
struct circle { point p; double r; };
struct line { point p, q; };
using segment = line;
const double eps = 1e-9;
// fix comparations on doubles with this two functions
int sign(double x) { return x < -eps ? -1 : x > eps; }
int dblcmp(double x, double y) { return sign(x - y); }
double dot(point a, point b) { return real(conj(a) * b); }
double cross(point a, point b) { return imag(conj(a) * b); }
double area2(point a, point b, point c) {
    return cross(b - a, c - a);
}
int ccw(point a, point b, point c) {
    b -= a; c -= a;
    if (cross(b, c) > 0) return +1; // counter clockwise
    if (cross(b, c) < 0) return -1; // clockwise
    if (dot(b, c) < 0) return +2; // c--a--b on line
    if (dot(b, b) < dot(c, c)) return -2; // a--b--c on line
    return 0;
}
namespace std {
    bool operator<(point a, point b) {
        if (a.real() != b.real())
            return a.real() < b.real();
        return a.imag() < b.imag();
    }
}
```

8.2 Antipodal Points

```
vector<pair<int, int>> antipodal(const polygon &P) {
    vector<pair<int, int>> ans; int n = P.size();
    if (P.size() == 2) ans.push_back({ 0, 1 });
    if (P.size() < 3) return ans;
    int q0 = 0;
    while (abs(area2(P[n - 1], P[0], P[NEXT(q0)]))
           > abs(area2(P[n - 1], P[0], P[q0])))
        ++q0;
    for (int q = q0, p = 0; q != 0 && p <= q0; ++p) {
        ans.push_back({p, q});
        while (abs(area2(P[p], P[NEXT(p)], P[NEXT(q)])) >
            abs(area2(P[p], P[NEXT(p)], P[q]))) {
            q = NEXT(q);
            if (p != q0 || q != 0) ans.push_back({p, q});
            else return ans;
        }
        if (abs(area2(P[p], P[NEXT(p)], P[NEXT(q)])) ==
            abs(area2(P[p], P[NEXT(p)], P[q]))) {
            if (p != q0 || q != n - 1)
                ans.push_back({p, NEXT(q)});
            else
                ans.push_back({NEXT(p), q});
        }
    }
    return ans;
}
```

8.3 Polygon Centroid

```
point centroid(const polygon &P) {
    point c(0, 0);
    double scale = 3.0 * area2(P);//area2 = 2 * polygon_area
    for (int i = 0, n = P.size(); i < n; ++i) {
        int j = NEXT(i);
        c = c + (P[i] + P[j]) * (cross(P[i], P[j]));
    }
    return c / scale;
}
```

8.4 Circle

```
// circle-circle intersection
vector <point> intersect(circle C, circle D) {
    double d = abs(C.p - D.p);
    if (sign(d - C.r - D.r) > 0) return {};      // too far
    if (sign(d - abs(C.r - D.r)) < 0) return {}; // too close
    double a = (C.r * C.r - D.r * D.r + d * d) / (2 * d);
    double h = sqrt(C.r * C.r - a * a);
    point v = (D.p - C.p) / d;
    if (sign(h) == 0) return {C.p + v * a}; // touch
    return {C.p + v * a + point(0, 1) * v * h, // intersect
            C.p + v * a - point(0, 1) * v * h};
}
// circle-line intersection
vector <point> intersect(line L, circle C) {
    point u = L.p - L.q, v = L.p - C.p;
    double a = dot(u, u), b = dot(u, v),
        c = dot(v, v) - C.r * C.r;
    double det = b * b - a * c;
    if (sign(det) < 0) return {};           // no solution
    if (sign(det) == 0) return {L.p - b / a * u}; // touch
    return {L.p + (-b + sqrt(det)) / a * u,
            L.p + (-b - sqrt(det)) / a * u};
}
// circle tangents through point
vector <point> tangent(point p, circle C) {
    // not tested enough
    double D = abs(p - C.p);
    if (D + eps < C.r) return {};
    point t = C.p - p;
    double theta = asin(C.r / D);
    double d = cos(theta) * D;
    t = t / abs(t) * d;
    if (abs(D - C.r) < eps) return {p + t};
    point rot(cos(theta), sin(theta));
    return {p + t * rot, p + t * conj(rot)};
}
bool incircle(point a, point b, point c, point p) {
    a -= p; b -= p; c -= p;
    return norm(a) * cross(b, c) + norm(b) * cross(c, a) +
        norm(c) * cross(a, b) >= 0;
    // < : inside, = cocircular, > outside
}
point three_point_circle(point a, point b, point c) {
    point x = 1.0 / conj(b - a), y = 1.0 / conj(c - a);
    return (y - x) / (conj(x) * y - x * conj(y)) + a;
}
/*  Area of the intersection of a circle with a polygon
  Circle's center lies in (0, 0)
  Polygon must be given counterclockwise*/
#define x(_t) (xa + (_t) * a)
#define y(_t) (ya + (_t) * b)
double radian(double xa, double ya, double xb, double yb) {
    return atan2(xa * yb - xb * ya, xa * xb + ya * yb);
}
double part
    (double xa, double ya, double xb, double yb, double r) {
```

```
    double l = sqrt((xa - xb) * (xa - xb) +
            (ya - yb) * (ya - yb));
    double a = (xb - xa) / l, b = (yb - ya) / l,
        c = a * xa + b * ya;
    double d = 4.0 * (c * c - xa * xa - ya * ya + r * r);
    if (d < eps) return radian(xa, ya, xb, yb) * r * r * 0.5;
    else {
        d = sqrt(d) * 0.5; double s = -c - d, t = -c + d;
        if (s < 0.0) s = 0.0;
        else if (s > l) s = l;
        if (t < 0.0) t = 0.0;
        else if (t > l) t = l;
        return (x(s) * y(t) - x(t) * y(s) +
            (radian(xa, ya, x(s), y(s)) +
            radian(x(t), y(t), xb, yb)) * r * r) * 0.5;
    }
}
double intersection_circle_polygon(const polygon &P, double r)
{
    double s = 0.0;
    int n = P.size();;
    for (int i = 0; i < n; i++)
        s += part(P[i].real(), P[i].imag(),
                  P[NEXT(i)].real(), P[NEXT(i)].imag(), r);
    return fabs(s);
}
```

8.5 Closest Pair Points

```
double closest_pair_points(vector <point> &P) {
    auto cmp = [](point a, point b) {
        return make_pair(a.imag(), a.real()) <
            make_pair(b.imag(), b.real());
    };
    int n = P.size(); sort(P.begin(), P.end());
    set < point, decltype(cmp) > S(cmp);
    const double oo = 1e9; /*adjust*/ double ans = oo;
    for (int i = 0, ptr = 0; i < n; ++i) {
        while(ptr < i && abs(P[i].real()-P[ptr].real()) >= ans)
            S.erase(P[ptr++]);
        auto lo=S.lower_bound(point(-oo, P[i].imag() -
ans-eps));
        auto hi=S.upper_bound(point(-oo, P[i].imag() +
ans+eps));
        for (decltype(lo) it = lo; it != hi; ++it)
            ans = min(ans, abs(P[i] - *it));
        S.insert(P[i]);
    }
    return ans;
}
```

8.6 Convex Cut

```
/*  Cut a convex polygon by a line and
  return the part to the left of the line */
polygon convex_cut(const polygon &P, const line &l) {
    polygon Q;
    for (int i = 0, n = P.size(); i < n; ++i) {
        point A = P[i], B = P[(i + 1) % n];
        if (ccw(l.p, l.q, A) != -1) Q.push_back(A);
        if (ccw(l.p, l.q, A) * ccw(l.p, l.q, B) < 0)
            Q.push_back(crosspoint((line) {A, B}, l));
    }
    return Q;
}
```

8.7 Convex Hull

```
polygon convex_hull(vector <point> &P) {
    int n = P.size(), k = 0;
    vector <point> h(2 * n);
    sort(P.begin(), P.end());
    for (int i = 0; i < n; h[k++] = P[i++])
        while(k >= 2 && area2(h[k - 2], h[k - 1], P[i]) <=
0)--k;
    for (int i = n - 2, t = k + 1; i >= 0; h[k++] = P[i--])
        while(k >= t && area2(h[k - 2], h[k - 1], P[i]) <=
0)--k;
    return polygon(h.begin(), h.begin() + k - (k > 1));
}
```

8.8 Line Operations

```
bool intersectLL(const line &l, const line &m) {
    return abs(cross(l.q - l.p, m.q - m.p)) > eps//non-parallel
        || abs(cross(l.q - l.p, m.p - l.p)) < eps;// same line
}
bool intersectLS(const line &l, const segment &s) {
    return cross(l.q - l.p, s.p - l.p) * // s[0] is left of l
        cross(l.q - l.p, s.q - l.p) < eps;//s[1] is right of l
}
bool intersectLP(const line &l, const point &p) {
    return abs(cross(l.q - p, l.p - p)) < eps;
}
bool intersectSS(const segment &s, const segment &t) {
    return ccw(s.p, s.q, t.p) * ccw(s.p, s.q, t.q) <= 0 &&
        ccw(t.p, t.q, s.p) * ccw(t.p, t.q, s.q) <= 0;
}
bool intersectSP(const segment &s, const point &p) {
    return abs(s.p - p) + abs(s.q - p) - abs(s.q - s.p) < eps;
    // triangle inequality
    return min(real(s.p), real(s.q)) <= real(p) &&
        real(p) <= max(real(s.p), real(s.q)) &&
        min(imag(s.p), imag(s.q)) <= imag(p) &&
        imag(p) <= max(imag(s.p), imag(s.q)) &&
        cross(s.p - p, s.q - p) == 0;
}
point projection(const line &l, const point &p) {
    double t = dot(p - l.p, l.p - l.q) / norm(l.p - l.q);
    return l.p + t * (l.p - l.q);
}
point reflection(const line &l, const point &p) {
    return p + 2.0 * (projection(l, p) - p);
}
double distanceLP(const line &l, const point &p) {
    return abs(p - projection(l, p));
}
double distanceLL(const line &l, const line &m) {
    return intersectLL(l, m) ? 0 : distanceLP(l, m.p);
}
double distanceLS(const line &l, const line &s) {
    if (intersectLS(l, s)) return 0;
    return min(distanceLP(l, s.p), distanceLP(l, s.q));
}
double distanceSP(const segment &s, const point &p) {
    const point r = projection(s, p);
    if (intersectSP(s, r)) return abs(r - p);
    return min(abs(s.p - p), abs(s.q - p));
}
double distanceSS(const segment &s, const segment &t) {
    if (intersectSS(s, t)) return 0;
    return min(min(distanceSP(s, t.p), distanceSP(s, t.q)),
               min(distanceSP(t, s.p), distanceSP(t, s.q)));
}
point crosspoint(const line &l, const line &m) {
    double A = cross(l.q - l.p, m.q - m.p);
    double B = cross(l.q - l.p, l.q - m.p);
    if (abs(A) < eps && abs(B) < eps)
        return m.p; // same line
    if (abs(A) < eps)
        assert(false); // !!!PRECONDITION NOT SATISFIED!!!
    return m.p + B / A * (m.q - m.p);
}
```

8.9 Pick's theorem

```
typedef long long ll; typedef complex <ll> point;
struct segment {point p, q;};
ll points_on_segment(const segment &s) {
    point p = s.p - s.q;
    return __gcd(abs(p.real()), abs(p.imag()));
}
// <Lattice points (not in boundary), Lattice points on
boundary>
pair <ll, ll> pick_theorem(polygon &P) {
    ll A = area2(P), B = 0, I = 0;
    for (int i = 0, n = P.size(); i < n; ++i)
        B += points_on_segment({P[i], P[NEXT(i)]});
    A = abs(A); I = (A - B) / 2 + 1; return {I, B};
}
```

8.10 Point 3D

```
const double pi = acos(-1.0);
//Construct a point on a sphere with center in origin and rad
R
struct point3d {
    double x, y, z;
    point3d(double x = 0, double y = 0, double z = 0) :
        x(x), y(y), z(z) {}
    double operator*(const point3d &p) const {
        return x * p.x + y * p.y + z * p.z;
    }
    point3d operator-(const point3d &p) const {
        return point3d(x - p.x, y - p.y, z - p.z);
    }
};
double abs(point3d p) {
    return sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
}
point3d from_polar(double lat, double lon, double R) {
    lat = lat / 180.0 * pi;
    lon = lon / 180.0 * pi;
    return point3d(R * cos(lat) * sin(lon),
                   R * cos(lat) * cos(lon), R * sin(lat));
}
struct plane { double A, B, C, D; };
double euclideanDistance(point3d p, point3d q) {
    return abs(p - q);
}
/* Geodisic distance between points in a sphere
   R is the radius of the sphere */
double geodesic_distance(point3d p, point3d q, double r) {
    return r * acos(p * q / r / r);
}
const double eps = 1e-9;
// Find the rect of intersection of two planes on the space
// The rect is given parametrical
void planePlaneIntersection(plane p, plane q) {
    if (abs(p.C * q.B - q.C * p.B) < eps)
        return; // Planes are parallel
    double mz=(q.A * p.B - p.A * q.B) / (p.C * q.B - q.C *
p.B);
    double nz=(q.D * p.B - p.D * q.B) / (p.C * q.B - q.C *
p.B);
    double my=(q.A * p.C - p.A * q.C) / (p.B * q.C - p.C *
q.B);
    double ny=(q.D * p.C - p.D * q.C) / (p.B * q.C - p.C *
q.B);
    // parametric rect: (x, my * x + ny, mz * x * nz)
}
```

8.11 Point in Polygon

```
enum { OUT, ON, IN };
int contains(const polygon &P, const point &p) {
    bool in = false;
    for (int i = 0, n = P.size(); i < n; ++i) {
        point a = P[i] - p, b = P[NEXT(i)] - p;
        if (imag(a) > imag(b)) swap(a, b);
        if (imag(a) <= 0 && 0 < imag(b))
            if (cross(a, b) < 0) in = !in;
        if (cross(a, b) == 0 && dot(a, b) <= 0)
            return ON;
    }
    return in ? IN : OUT;
}
```

8.12 Polar Sort

```
int quad(point x) {
    if (x.imag() >= 0 && x.real() > 0) return 0;
    if (x.imag() > 0 && x.real() <= 0) return 1;
    if (x.imag() <= 0 && x.real() < 0) return 2;
    return 3;
}
vector <point> angular_sort(vector <point> &v) {
    sort(all(v), [&](point &a, point &b) {
        if (quad(a) != quad(b)) return quad(a) < quad(b);
        return cross(a, b) > 0;
    });
    return v;
}
```

8.13 Polygon Area

```
double area2(const polygon &P) {
    double A = 0;
    for (int i = 0, n = P.size(); i < n; ++i)
        A += cross(P[i], P[NEXT(i)]);
    return A;
}
```

8.14 Polygon Width

```
// Compute the width of a convex polygon
const int oo = 1e9; // adjust
double check(int a, int b, int c, int d, const polygon &P) {
    for (int i = 0; i < 4 && a != c; ++i) {
        if (i == 1) swap(a, b);
        else swap(c, d);
    }
    if (a == c) { // a admits a support line parallel to bd
        double A = abs(area2(P[a], P[b], P[d]));
        double base = abs(P[b] - P[d]);
        return A / base;
    }
    return oo;
}
double polygon_width(const polygon &P) {
    if (P.size() < 3)
        return 0;
    auto pairs = antipodal(P);
    double best = oo;
    int n = pairs.size();
    for (int i = 0; i < n; ++i) {
        double tmp = check(pairs[i].first, pairs[i].second,
           pairs[NEXT(i)].first, pairs[NEXT(i)].second, P);
        best = min(best, tmp);
    }
    return best;
}
```

8.15 Rectilinear MST

```
typedef long long ll; typedef complex<ll> point;
ll rectilinear_mst(vector<point> ps) {
    vector<int> id(ps.size()); iota(id.begin(), id.end(), 0);
    struct edge { int src, dst; ll weight; };
    vector<edge> edges;
    for (int s = 0; s < 2; ++s) {
        for (int t = 0; t < 2; ++t) {
            sort(id.begin(), id.end(), [&](int i, int j) {
                return real(ps[i] - ps[j]) < imag(ps[j] -
ps[i]);
            });
            map<ll, int> sweep;
            for (int i: id) {
                for (auto it = sweep.lower_bound(-imag(ps[i]));
                    it != sweep.end(); sweep.erase(it++)) {
                    int j = it->second;
                    if (imag(ps[j] - ps[i]) < real(ps[j] -
ps[i]))
                        break;
                    ll d = abs(real(ps[i] - ps[j])) +
                            abs(imag(ps[i] - ps[j]));
                    edges.push_back({i, j, d});
                }
                sweep[-imag(ps[i])] = i;
            }
            for (auto &p: ps)
                p = point(imag(p), real(p));
        }
        for (auto &p: ps)
            p = point(-real(p), imag(p));
    }
    ll cost = 0;
    sort(edges.begin(), edges.end(), [](edge a, edge b) {
        return a.weight < b.weight;
    });
    union_find uf(ps.size());
    for (edge e: edges)
        if (uf.join(e.src, e.dst))
            cost += e.weight;
    return cost;
}
```

9 Various

9.1 Divide and Conquer

```
void DIVIDE_AND_CONQUER(int k, int L, int R, int optL, int
optR) {
    if(L > R)return;
    int m = (L+R)/2, opt = -1;
    dp[m][k] = oo;
    for(int i = optL ; i <= min(m,optR) ; i++) {
        ll t = dp[i-1][k-1] + w(i,m);
        if(dp[m][k] > t)dp[m][k] = t, opt = i;
    }
    DIVIDE_AND_CONQUER(k,L,m-1,optL,opt);
    DIVIDE_AND_CONQUER(k,m+1,R,opt,optR);
}
```

9.2 Integer Division

```
inline ll ceil(ll a, ll b) {
    return a/b + ((a>0)^(b>0) ? 0 : a%b != 0);
}
inline ll floor(ll a, ll b) {
    return a/b - ((a>0)^(b>0) ? a%b != 0 : 0);
}
```

9.3 Bit hacks

- `x & -x` is the least bit in `x`.

- `for (int x = m; x; ) { --x &= m; ... }` loops over all subset masks of `m` (except `m` itself).

- `c = x&-x, r = x+c; (((r^x) >> 2)/c) | r` is the next number after `x` with the same number of bits set.

- `rep(b,0,K) rep(i,0,(1 << K))`
  `if (i & 1 << b) D[i] += D[i^(1 << b)];` computes all sums of subsets.

9.4 Formulas

- Pick's Theorem: $A = i + \frac{b}{2} - 1$

- Euler's formula: $vertices - edges + faces = 2$ (In connected planar graphs)

- $Catalan_n = \frac{1}{n+1} \cdot \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{2n!}{n!(n+1)!}$

- $Catalan_n^{(k)} = \frac{k+1}{n+k+1} \cdot \binom{2n+k}{n}$ Count of balanced parentheses sequences consisting of n+k pairs of parentheses where the first K symbols are open brackets.

- Stirling Numbers of the First Kind:

  - unsigned $s(n,k)$ counts the number of permutations of size n with k cycles.
  - $s(n+1,k) = -n \cdot s(n,k) + s(n,k-1)$ (for unsigned just n instead of $-n$)
  - $s(0,0) = 1$ and $s(0,n) = s(n,0) = 0$ for $n > 0$.
  - $(x)_n = \sum_{k=0}^{n} s(n,k) \cdot x^k$ where $(x)_n = x(x-1)(x-2)...(x-n+1)$, so the Stirling numbers are the coefficients of the rising factorials polynomial.

- Stirling Numbers of the Second Kind:

  - Number of ways to partition a set of n objects into k non-empty subsets.
  - $S(n+1,k) = k \cdot S(n,k) + S(n,k-1)$
  - $S(n,k) = \frac{1}{k!} \sum_{i=0}^{k} (-1)^i \binom{k}{i} (k-i)^n$

- Euclides' Pythagorean triple: $a^2 + b^2 = c^2 \iff a = 2mn, b = m^2 - n^2, c = m^2 + n^2$

- $\sum_{d|n} \phi(d) = n$

- inv[i] = (MOD-MOD/i)*inv[MOD%i]

9.5 Hash table

```
#include <bits/extc++.h>
using namespace __gnu_pbds;

struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
chrono::steady_clock::nmow().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};

gp_hash_table<long long, int, custom_hash> safe_hash_table;
```

9.6 Convex Hull Trick

```
typedef long long ll; typedef complex<ll> point;
ll cross(point a, point b) { return imag(conj(a) * b); }
///a_x*b_y-b_x*a_y
ll dot(point a, point b) { return real(conj(a) * b); } ///
a_x*b_x+a_y*b_y
ll area2(point a, point b, point c) { return cross(b - a, c -
a); }
namespace std {
    bool operator<(const point &a, const point &b) {
        return real(a) < real(b) || (real(a) == real(b) &&
imag(a) < imag(b));
    }
}
const ll oo = 0x3f3f3f3f3f3f3f3f;
struct dynamic_hull {
    dynamic_hull() : hulls() {}
    void add_point(point p) {
        hull h; h.add_point(p);
        for (hull &_h : hulls)
            if (_h.empty()) {
                h.swap(_h);
                break;
            }
            else h = merge(h, _h), _h.clear();
        if (!h.empty()) hulls.emplace_back(h);
    }
    ll max_dot(point p) {
        ll best = -oo;
        for (hull &h : hulls)
            if (!h.empty()) best = max(best, h.max_dot(p));
        return best;
    }
private:
    struct hull : vector<point> {
        void add_point(point p) {
            for (int s = size(); s > 1; --s)
                if (area2(at(s - 2), at(s - 1), p) < 0) break;
                else pop_back();
            push_back(p);
        }
        ll max_dot(point p) {
            int lo = 0, hi = (int) size() - 1, mid;
            while (lo < hi) {
                mid = (lo + hi) / 2;
                if (dot(at(mid), p) <= dot(at(mid + 1), p))
                    lo = mid + 1;
                else hi = mid;
            }
            return dot(at(lo), p);
        }
    };
    static hull merge(const hull &a, const hull &b) {
        hull h; size_t i = 0, j = 0;
        while (i < a.size() && j < b.size())
            if (a[i] < b[j]) h.add_point(a[i++]);
            else h.add_point(b[j++]);
        while (i < a.size()) h.add_point(a[i++]);
        while (j < b.size()) h.add_point(b[j++]);
        return h;
    }
    vector<hull> hulls;
};
```

9.7 SOS DP

```
for(int i = 0; i<(1<<N); ++i)
    F[i] = A[i];
for(int i = 0;i < N; ++i)
    for(int mask = 0; mask < (1<<N); ++mask)
        if(mask & (1<<i))
            F[mask] += F[mask^(1<<i)];
```

9.8 Gray code

```
int g (int n) {
    return n ^ (n >> 1);
}
int rev_g (int g) {
  int n = 0;
  for (; g; g >>= 1)
    n ^= g;
  return n;
}
```

9.9 CMake

```
cmake_minimum_required(VERSION 3.24)
project(icpc)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -O2
-Wl,-stack_size,10000000")

set(filename)
```

```
foreach(number RANGE 97 122 1)
    string(ASCII ${number} filename)
    file(WRITE codes/${filename}.cpp)
    add_executable(${filename} codes/${filename}.cpp)
endforeach()
```