York University Team Notebook C++ (2021-2022)

York University ACM

March 3, 2022

Contents

1 Bits

1.1 iterate_submasks.cpp

```
for (int sub = mask; ; sub = (sub - 1) & mask) {
    printf("%3d: ", sub);
    if (sub == 0) break; // move this to loop condition if you don't want 0
}
```

1.2 iterate_supermasks.cpp

```
for (int super = mask; super < 1 << n; super = (super + 1) | mask)
```

1.3 xor_basis.cpp

```
#include <limits>
template<typename T>
struct xor_basis {
    static constexpr int B = 8 * sizeof(T);
    T basis[B];
    int sz = 0;

    void insert(T x) {
        for (int i=B; i>=0; i--) {
            if (x>>i==0) continue;
            if (!basis[i]) {
                basis[i]=x;
                break;
            }
            x^=basis[i];
        }
    }

    T max_value(T start = 0) {
        for (int i=B; i>=0; i--) {
            if (basis[i]) {
                start = max(start, start^basis[i]);
            }
        }
        return start;
    }
};
```

2 DataStructure

2.1 2d_diff_array.cpp

```
#include <bits/stdc++.h>
using namespace std;
template <typename T> struct diff_2d {
    int n, m;
    vector<vector<T>> dif;
    diff_2d(int n_, int m_)
        : n(n_), m(m_), dif(n + 2, vector<T>(m + 2)) {}
    void add(int x1, int y1, int x2, int y2, T c) {
        x1++, x2++, y1++, y2++;
        dif[x1][y1] += c;
        dif[x2 + 1][y1] -= c;
        dif[x1][y2 + 1] -= c;
        dif[x2 + 1][y2 + 1] += c;
    }

    vector<vector<T>> build() {
        vector res(n, vector<T>(m));
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                dif[i][j] += dif[i - 1][j] + dif[i][j - 1] - dif[i - 1][j - 1];
                res[i - 1][j - 1] = dif[i][j];
            }
        }
        return res;
    }
};
```

2.2 2d_pref_sum.cpp

```
#include <bits/stdc++.h>
using namespace std;
template<typename T>
struct pref_sum_2d {
    int n, m;
    vector<vector<T>> sum;
    template<typename U>
    pref_sum_2d(const vector<vector<U>>& a)
        : n((int)a.size()), m((int)a[0].size()), sum(n+1, vector<T>(m+1)) {
            for (int i = 0; i < n; i++)
                for (int j = 0; j < m; j++) {
                    sum[i+1][j+1]=a[i][j] + sum[i][j+1] + sum[i+1][j] - sum[i][j];
                }
        }
    T query(int x1, int y1, int x2, int y2) {
        return sum[x2+1][y2+1] - sum[x2+1][y1] - sum[x1][y2+1] + sum[x1][y1];
    }
};
```

2.3 fenwick.cpp

```
template <typename T> struct fenwick {
    int n; vector<T> t;
    fenwick(int n_) : n(n_), t(n + 1) {}
    fenwick(const vector<T> &v) : fenwick((int)v.size()) {
        for (int i = 1; i <= n; i++) {
            t[i] += v[i - 1];
            int j = i + (i & -i);
            if (j <= n) t[j] += t[i];
        }
    }
    void add(int i, T x) {
        assert(i >= 0 && i < n);
        for (i++; i <= n; i += i & -i) {
            t[i] += x;
        }
    }
    template <typename U = T> U query(int i) {
        assert(i >= 0 && i < n);
        U res{};
        for (i++; i > 0; i -= i & -i)
            res += t[i];
        return res;
    }
    template <typename U = T> U query(int l, int r) {
        assert(l >= 0 && l <= r && r < n);
        return query<U>(r) - (l ? query<U>(l - 1) : U{});
    }
    int search(T prefix) { // finds first pos s.t. sum(0, pos)>=prefix
        int pos = 0;
        T sum = 0;
        for (int i = __lg(n); i >= 0; i--) {
            // could change < to <= to make it find upper bound
            if (pos + (1 << i) <= n && (sum + t[pos + (1 << i)] < prefix)) {
                pos += (1 << i);
                sum += t[pos];
            }
        }
        return pos;
    }
};
// fenwick tree with range update and range sum query
struct fenwick_rg {
    int n;
    vector<int64_t> sum1, sum2;
    fenwick_rg(int n_) : n(n_), sum1(n + 1), sum2(n + 1) {}
  private:
    void add(int i, int x) {
        assert(i >= 0 && i < n);
        i++;
        int64_t v = (int64_t)i * x;
        for (; i <= n; i += i & -i)
            sum1[i] += x, sum2[i] += v;
    }
  public:
    void add(int l, int r, int x) {
        assert(l >= 0 && l <= r && r < n);
        add(l, x);
        if (r + 1 < n) add(r + 1, -x);
    }
    int64_t query(int p) {
        assert(p >= 0 && p < n);
        p++;
        int64_t res{};
        for (int i = p; i; i -= i & -i)
            res += (p + 1) * sum1[i] - sum2[i];
        return res;
    }
    int64_t query(int l, int r) {
        assert(l >= 0 && l <= r && r < n);
        return query(r) - (l ? query(l - 1) : 0);
    }
};
```

2.4 indexed-set.cpp

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
template <class T, class V=null_type> using Tree = tree<T, V, std::less<T>,
        rb_tree_tag, tree_order_statistics_node_update>;
```

2.5 lazy_segtree.cpp

```
// lazy propagation
#include<bits/stdc++.h>
using namespace std;
struct SegTree {
    struct Node {
        int v=0; // don't forget to set default value (used for leaves), not
             necessarily zero element
        int lazy=0;

        Node() = default;
        explicit Node(int val) : v(val) {}

        void apply(int l, int r, int x) {
            lazy += x;
            v += x;
        }
        // used to check if need to propagate
        bool has_lazy() { return lazy!=0; }
        void clear_lazy() { lazy=0; }

        static Node merge(const Node& lhs, const Node& rhs) {
            Node res;
            res.v=min(lhs.v,rhs.v);
            return res;
        }
    };
    int n;
    vector<Node> t;
    SegTree(int n_) : n(n_), t(4 * n) {}
    SegTree(int n_, int x) : SegTree(n_) {
        build(1, 0, n - 1, [&](int i) { return x; });
    }
    SegTree(int n_, function<int(int)> f) : SegTree(n_) {
        build(1, 0, n-1, f);
    }
    SegTree(const vector<int> &v) : SegTree((int)v.size()) {
        build(1, 0, n - 1, [&](int i) { return v[i]; });
    }
    void pull(int node) { t[node] = Node::merge(t[node * 2], t[node * 2 + 1]); }
    void build(int node, int l, int r, function<int(int)> f) {
        if (l == r) {
            t[node]=Node{f(l)};
            return;
        }
        int mid = (l + r) / 2;
        build(node * 2, l, mid, f);
        build(node * 2 + 1, mid + 1, r, f);
        pull(node);
    }
    void push(int p, int l, int r) {
        if (t[p].has_lazy()) {
            int m = (l + r) / 2;
            t[p * 2].apply(l, m, t[p].lazy);
            t[p * 2 + 1].apply(m + 1, r, t[p].lazy);
            t[p].clear_lazy();
        }
    }
    template<typename U>
    void update(int node, int ql, int qr, int l, int r, U x) {
        if (r < ql || l > qr) return;
        if (ql <= l && qr >= r) return t[node].apply(l, r, x);
        push(node, l, r);
        int mid = (l + r) / 2;
        update(node * 2, ql, qr, l, mid, x);
        update(node * 2 + 1, ql, qr, mid + 1, r, x);
        pull(node);
    }
    Node get(int node, int ql, int qr, int l, int r) {
        if (ql <= l && qr >= r) return t[node];
        push(node, l, r);
        int mid = (l + r) / 2;
        if (qr <= mid) return get(node << 1, ql, qr, l, mid);
        if (ql > mid) return get(node << 1 | 1, ql, qr, mid+1, r);
        return Node::merge(get(node << 1, ql, qr, l, mid), get(node << 1 | 1, ql, qr,
             mid+1, r));
    }
    // wrapper
    template <typename U>
    void add(int l, int r, U x) {
        if (l==r+1) return; // empty interval, but also can be bug in code
        assert(l >= 0 && l <= r && r < n);
        update(1, l, r, 0, n-1, x);
    }
    Node get(int l, int r) {
        assert(l >= 0 && l <= r && r < n);
        return get(1, l, r, 0, n-1);
    }
};
```

2.6 line_container.cpp

```
#include <bits/stdc++.h>
using namespace std;
/**
 * Credit: https://github.com/kth-competitive-programming/kactl/blob/main/content/data-structures/LineContainer.h
 * Author: Simon Lindholm
 * Date: 2017-04-20
 * License: CC0
 * Source: own work
 * Description: Container where you can add lines of the form kx+m, and query
 * maximum values at points x. Useful for dynamic programming (``convex hull
 * trick''). Time: O(\log N) Status: stress-tested
 */


using ll = long long;

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

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b);
  }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y))
      isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

2.7 monotonic_dp_hull.cpp

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
// monotonic_dp_hull enables you to do the following two operations in amortized O(1) time:
// 1. Insert a line (k, b) into the structure. k must be non-decreasing.
// 2. For any value of x, query the maximum value of k * x + b. x must be non-decreasing.
// Note:
// 1. if slope and/or query is non-increasing, change position of operation
// 2. if slope and/or query is in arbitrary order, use line_container instead which has complexity of O(log n) per operation
struct monotonic_dp_hull {
    struct line {
        ll k, b;
        ll eval(ll x) { return k * x + b; }
    };

    bool bad(const line &a, const line &b, const line &c) {
        return (c.b - a.b) * (a.k - b.k) <= (b.b - a.b) * (a.k - c.k);
    }

    deque<line> lines;

    void insert(ll k, ll b) {
        assert(lines.empty() || k > lines.back().k); // ensure slope is monotonic
        line cur{k, b};
        while (lines.size() >= 2 && bad(*(lines.rbegin() + 1), lines.back(), cur))
            lines.pop_back();
        lines.push_back(cur);
    }

    ll query(ll x) {
        assert(!lines.empty());
        while (lines.size() >= 2 && lines[0].eval(x) <= lines[1].eval(x))
            lines.pop_front();
        return lines[0].eval(x);
    }
};
```

2.8 persistent_seg.cpp

```
//find the nth biggest number
#include<bits/stdc++.h>
using namespace std;
struct PST {
    int n, tot=0;
    struct node {
        int lc, rc, sum;
    };
    vector<node> t;
    vector<int> roots; // left child, right child
    PST(int n_) : n(n_), t(n<<5), roots(1) { // change the size to n<<6 if there are 2*n modification
        build(0, n-1, roots[0]); // the initial root node is 1!
    }
#define lc(rt) t[t[rt].lc]
#define rc(rt) t[t[rt].rc]
    void pushup(int rt) {
        t[rt].sum = lc(rt).sum + rc(rt).sum;
    }
    void build(int l, int r, int& rt) {
        rt = ++tot;
        if (l == r) return;
        int mid = (l + r) >> 1;
        build(l, mid, t[rt].lc);
        build(mid + 1, r, t[rt].rc);
        pushup(rt);
    }
    void update(int pos, int val, int l, int r, int old, int& rt) {
        rt = ++tot;
        t[rt] = t[old];
        if (l == r) {
            t[rt].sum = t[old].sum + val;
            return;
        }
        int mid = (l + r) >> 1;
        if (pos <= mid) update(pos, val, l, mid, t[old].lc, t[rt].lc);
        else update(pos, val, mid + 1, r, t[old].rc, t[rt].rc);
        pushup(rt);
    }
    int update(int pos, int val) { // return the root of the new version
        int new_root;
        update(pos, val, 0, n-1, roots.back(), new_root);
        roots.push_back(new_root);
        return new_root;
    }
    int query(int u, int v, int l, int r, int k) {
        if (l==r) return l;
        int mid=(l+r)/2, x=lc(v).sum-lc(u).sum;
        if (k<=x) return query(t[u].lc, t[v].lc, l, mid, k);
        return query(t[u].rc, t[v].rc, mid+1, r, k-x);
    }
};
int main(){
    int n, q;
    cin>>n>>q;
    vector<int> a(n);
    for (auto& x : a) cin>>x;
    auto comp=a;
    sort(comp.begin(), comp.end());
    comp.erase(unique(comp.begin(), comp.end()), comp.end());
    PST tr(comp.size());
    vector<int> roots(n+1);
    roots[0]=1;
    for (int i=0; i<n; i++) {
        int p=lower_bound(comp.begin(), comp.end(), a[i])-comp.begin();
        roots[i+1]=tr.update(p, 1);
    }
    while (q--) {
        int l, r, k;
        cin>>l>>r>>k;
        cout<<comp[tr.query(roots[l-1], roots[r], 0, comp.size()-1, k)]<<'\n';
    }
}
```

2.9 segtree.cpp

```
template <typename T>
struct SegTree {
    int n;
    vector<T> t;
    SegTree(int n_) : n(n_), t(4 * n) {
        build(1, 0, n-1, vector(n, T()));
    }
    template<typename U>
    SegTree(const vector<U> &v) : SegTree((int)v.size()) {
        build(1, 0, n - 1, v);
    }
    void pull(int node) { t[node] = t[node << 1] + t[node << 1 | 1]; }
    template<typename U>
    void build(int node, int l, int r, const vector<U> &v) {
        if (l == r) {
            t[node] = T(v[l]);
            return;
        }
        int mid = (l + r) >> 1;
        build(node << 1, l, mid, v);
        build(node << 1 | 1, mid + 1, r, v);
        pull(node);
    }
    template<typename U>
    void add(int node, int i, U x, int l, int r) {
        if (l == r) {
            t[node] += x;
            return;
        }
        int mid = (l + r) / 2;
        if (i <= mid) add(node << 1, i, x, l, mid);
        else add(node << 1 | 1, i, x, mid + 1, r);
        pull(node);
    }
    void set(int node, int i, T x, int l, int r) {
        if (l == r) {
            t[node] = x;
            return;
        }
        int mid = (l + r) / 2;
        if (i <= mid) set(node << 1, i, x, l, mid);
        else set(node << 1 | 1, i, x, mid + 1, r);
        pull(node);
    }
    T get(int node, int ql, int qr, int l, int r) {
        if (ql <= l && qr >= r) return t[node];
        int mid = (l + r) >> 1;
        if (qr <= mid) return get(node << 1, ql, qr, l, mid);
        if (ql > mid) return get(node << 1 | 1, ql, qr, mid+1, r);
        return get(node << 1, ql, qr, l, mid) + get(node << 1 | 1, ql, qr, mid+1, r);
    }
    // wrapper
    template <typename U>
    void add(int i, U x) {
        assert(i >= 0 && i < n);
        add(1, i, x, 0, n-1);
    }
    void set(int i, T x) {
        assert(i >= 0 && i < n);
        set(1, i, x, 0, n-1);
    }
    T get(int l, int r) {
        assert(l >= 0 && l <= r && r < n);
        return get(1, l, r, 0, n-1);
    }
};
struct node {
    int v=0; // value for leaves
    node() = default;
    // may need more constructor
    node operator+(const node& rhs) const { // used in get() and pull()
        return {v+rhs.v};
    }
    node& operator +=(const node& rhs) { // used in add()
        v+=rhs.v;
        return *this;
    }
};
```

2.10 segtree_none_recursive.cpp

```
struct SegTree{
    int n;
    vector<int> t;
    SegTree(int n_):n(n_),t(2*n){}
    SegTree(vector<int>& a):SegTree((int)a.size()){
        for (int i=0;i<n;i++) t[n+i]=a[i];
        for (int i = n - 1; i > 0; --i) t[i] = t[i<<1] + t[i<<1|1];
    }
    void update(int p, int value) {  // set value at position p
        t[p += n] = value;
        for (; p > 1; p >>= 1) t[p>>1] = t[p] + t[p^1];
    }
    int query(int l, int r) {  // sum on interval [l, r)
        int res = 0;
        for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
            if (l&1) res += t[l++];
            if (r&1) res += t[--r];
        }
        return res;
    }
};
```

2.11 sliding_window.cpp

```
template<typename T, typename compare = less<T>>
struct sliding_window {
    int k; // width of the window
    deque<pair<int, T>> q;
    compare cmp;
    sliding_window(int k_) : k(k_), cmp() {}
    void add(int i, T x) {
        while (!q.empty() && !cmp(q.back().second, x)) q.pop_back();
        q.emplace_back(i, x);
        while (q.front().first <= i - k) q.pop_front();
    }
    T get() { return q.front().second; }
};
```

2.12 sparse-table.cpp

```
#include <bits/stdc++.h>
using namespace std;
template <typename T> struct sparse {
    int n, logn;
    vector<vector<T>> v;
    function<T(T, T)> F;
    sparse(const vector<T> &a, function<T(T, T)> func)
        : n((int)a.size()), logn(__lg(n)), v(logn + 1, vector<T>(n + 1)), F(func) {
        v[0] = a;
        for (int i = 1; i <= logn; i++)
            for (int j = 0; j + (1 << i) - 1 < n; j++)
                v[i][j] = F(v[i - 1][j], v[i - 1][j + (1 << (i - 1))]);
    }
    T query(int x, int y) {
        assert(x<=y);
        int s = __lg(y - x + 1);
        return F(v[s][x], v[s][y - (1 << s) + 1]);
    }
};

namespace st { // 2d sparse table
    using T = int;
    int n, m, logn, logm;
    static const int N = 1e3 + 5;
    T t[13][13][N][N]; // array layout matches loop order to ensure efficiency

    template<typename U>
    void init(const vector<vector<U>>& val) {
        n = ((int)val.size()), m = ((int)val[0].size()),
        logn = (__lg(n)), logm = (__lg(m));
        for (int i = 0; i < n; i++) for (int j = 0; j < m; j++) t[0][0][i][j] = val[i][j];
        for (int i = 0; i <= logn; i++)
            for (int j = 0; j <= logm; j++) {
                if (i == 0 && j == 0) continue;
                for (int row = 0; row + (1 << i) - 1 < n; row++) {
                    for (int col = 0; col + (1 << j) - 1 < m; col++) {
                        // auto &v = t[row][col];
                        if (i == 0)
                            t[i][j][row][col] = min(t[i][j - 1][row][col], t[i][j - 1][row][col + (1 << (j - 1))]);
                        if (j == 0)
                            t[i][j][row][col] = min(t[i - 1][j][row][col], t[i - 1][j][row + (1 << (i - 1))][col]);
                        else
                            t[i][j][row][col] = min(t[i][j - 1][row][col], t[i][j - 1][row][col + (1 << (j - 1))]);
                    }
                }
            }
    }
    T query(int x1, int x2, int y1, int y2) {
        assert(n!=0 && m!=0);
        assert(x1 <= x2);
        assert(y1 <= y2);
        assert(x1 >= 0 && x1 < n);
        assert(x2 >= 0 && x2 < n);
        assert(y1 >= 0 && y1 < m);
        assert(y2 >= 0 && y2 < m);
        int kx = __lg(x2 - x1 + 1), ky = __lg(y2 - y1 + 1);
        return min(
            {t[kx][ky][x1][y1], t[kx][ky][x2 - (1 << kx) + 1][y1],
             t[kx][ky][x1][y2 - (1 << ky) + 1],
             t[kx][ky][x2 - (1 << kx) + 1][y2 - (1 << ky) + 1]});
    }

};
```

2.13 treap_rotate.cpp

```
mt19937 gen(chrono::high_resolution_clock::now().time_since_epoch().count());
struct Treap{
    struct data{
        int v;
        bool operator == (const data& d) const {
            return v==d.v;
        }
        bool operator < (const data& d) const {
            return v<d.v;
        }
    };
    struct node{
        int ch[2],sz,dup;
        unsigned k;
        data d;
        node(int z=1):sz(z),dup(z),k(gen()){
            ch[0]=ch[1]=0;
        }
    };
    vector<node> nodes;
    vector<int> recycle;
    int root,reserve_size;
    Treap(int size=0){
        nodes.clear();
        recycle.clear();
        nodes.reserve(size+1);
        nodes.push_back(node(0));
        root=0;
        reserve_size=size+1;
    }
    void reserve(){
        if(size()>=reserve_size) nodes.reserve((reserve_size*=2)+1);
    }
    int new_node(){
        int id=nodes.size();
        if(!recycle.empty()){
            id=recycle.back();
            recycle.pop_back();
            nodes[id]=node();
        }else nodes.push_back(node());
        return id;
    }
    void update(int rt){
        node& n=nodes[rt];
        n.sz=n.dup+nodes[n.ch[0]].sz+nodes[n.ch[1]].sz;
    }
    int insert(int& rt, data& d){// insert a data in bst rooted at rt
        if(rt==0){
            rt=new_node();
            nodes[rt].d=d;
            return rt;
        }
        node& cur=nodes[rt];
        cur.sz++;
        if(d==cur.d){
            cur.dup++;
            return rt;
        }
        //changed
        bool r=cur.d<d;
        int& s=cur.ch[r];
        int ret=insert(s,d);
        if(nodes[s].k<cur.k) rotate(rt,r),update(rt);
        return ret;
    }
    void rotate(int& rt,int r){
        node& cur=nodes[rt];
        int s=cur.ch[r];
        cur.ch[r]=nodes[s].ch[r^1];
        nodes[s].ch[r^1]=rt;
        update(rt);
        rt=s;
    }
    int find(int& rt,const data& d){
        if(rt==0) return 0;
        if(d==nodes[rt].d) return rt;
        //changed
        return find(nodes[rt].ch[(nodes[rt].d<d)],d);
    }
    bool erase_founded(int& rt,const data& d){//returns if founded
        if(rt==0) return false;
        if(d==nodes[rt].d){
            nodes[rt].sz--;
            if(--nodes[rt].dup<=0) remove(rt);
            return true;
        }
        //changed
        if(erase_founded(nodes[rt].ch[(nodes[rt].d<d)],d)){
            nodes[rt].sz--;
            return true;
        }
        return false;
    }
    void remove(int& rt){
        if(rt==0) return;
        if(!nodes[rt].ch[0] || !nodes[rt].ch[1]){ // if one child is empty
            recycle.push_back(rt);
            rt=nodes[rt].ch[!nodes[rt].ch[0]];
        }else{
            int r=nodes[nodes[rt].ch[0]].k<nodes[nodes[rt].ch[1]].k;
            rotate(rt,r^1);
            remove(nodes[rt].ch[r]);
            update(rt);
        }
    }
    int kth(int rt,int k){
        node& cur=nodes[rt];
        int sz=nodes[cur.ch[0]].sz;
        if(sz>=k) return kth(cur.ch[0],k);
        if(sz+cur.dup>=k) return rt;
        return kth(cur.ch[1],k-sz-cur.dup);
    }
    int rank(int rt,const data& d,bool count_dup){
        if(rt==0) return 1;
        node& cur=nodes[rt];
        if(d==cur.d) return nodes[cur.ch[0]].sz+1+count_dup*cur.dup;
        if(d<cur.d) return rank(cur.ch[0],d,count_dup);
        return rank(cur.ch[1],d,count_dup)+nodes[cur.ch[0]].sz+cur.dup;
    }
    //interface
    int get(int id){ return nodes[id].d.v; }
    int size(){ return nodes[root].sz;}
    int insert(data v){ reserve(); return insert(root,v);}
    bool erase(data v){ return erase_founded(root,v);}
    int find(data v){ return find(root,v);}//return id;
    int find_by_order(int k){ return get(kth(root,k));}
    int order_of_key(data v,bool count_dup=0){ return rank(root,v,count_dup);}
};
int main(){
    tr.find_by_order(tr.order_of_key({x})-1);//first element smaller than x
    tr.find_by_order(tr.order_of_key({x},true));//upper_bound(x)
}
```

2.14 treap_set.cpp

```
// using treap to maintain a sequence that support multiple operation, index
// starts from 0!
#include<bits/stdc++.h>
mt19937 gen(chrono::high_resolution_clock::now().time_since_epoch().count());
struct data {
    long long v;
    data(long long _v = 0) : v(_v) {}
    operator bool() const { return v != 0; }
    operator int() const { return v; }
    bool operator < (const data& d) const {
        return v < d.v;
    }
};
template <typename T> struct Treap {
    struct node {
        int ch[2], sz=0;
        unsigned k=0;
        T d, sum;
        node() = default;
        node(T d_) : sz(1), k((unsigned)gen()), d(d_), sum(d_) { ch[0] = ch[1] = 0; }
    };
    vector<node> nodes;
    int root, recyc;
    Treap(int size=2e5) {
        nodes.reserve((size = max(size, 15)) + 1);
        nodes.emplace_back();
        root = recyc = 0;
    }
    inline int &ch(int rt, int r) { return nodes[rt].ch[r]; }
    int new_node(const T &d) {
        int id = (int)nodes.size();
        if (recyc) {
            id = recyc;
            if (ch(recyc, 0) && ch(recyc, 1))
                recyc = merge(ch(recyc, 0), ch(recyc, 1));
            else
                recyc = ch(recyc, ch(recyc,0) ? 0 : 1);
            nodes[id] = node(d);
        } else nodes.push_back(node(d));
        return id;
    }
    int pull(int rt) {
        node &n = nodes[rt];
        n.sz = 1 + nodes[n.ch[0]].sz + nodes[n.ch[1]].sz;
        n.sum = n.d + nodes[n.ch[0]].sum + nodes[n.ch[1]].sum;
        return rt;
    }
    int merge(int tl, int tr) {
        if (!tl) return tr;
        if (!tr) return tl;
        if (nodes[tl].k < nodes[tr].k) {
            // pushdown(tl);
            ch(tl, 1) = merge(ch(tl, 1), tr);
            return pull(tl);
        } else {
            // pushdown(tr);
            ch(tr, 0) = merge(tl, ch(tr, 0));
            return pull(tr);
        }
    }
    void split_size(int rt, int k, int &x, int &y) { // split between k-th element and (k+1)-th element
        if (!rt) {
            x = y = 0;
            return;
        }
        // pushdown(rt);
        if (nodes[ch(rt, 0)].sz > k) {
            y = rt;
            split_size(ch(rt, 0), k, x, ch(rt, 0));
        } else {
            x = rt;
            split_size(ch(rt, 1), k - nodes[ch(rt, 0)].sz - 1, ch(rt, 1), y);
    }
    pull(rt);
}
void split_val(int rt, const T& target, int& x, int& y) {// split into two sets such that one contains <=k and other contains >k
    if (!rt) {
        x=y=0;
        return;
    }
    if (target < nodes[rt].d) {
        y = rt;
        split_val(ch(rt, 0), target, x, ch(rt, 0));
    } else {
        x = rt;
        split_val(ch(rt, 1), target, ch(rt, 1), y);
    }
    pull(rt);
}
void remove(int &rt) {
    if (recyc == 0) recyc = rt;
    else recyc = merge(recyc, rt);
    rt = 0;
}
int order_of_key(int rt, const T& d) {
    if (rt==0) return 0;
    node& cur = nodes[rt];
    if (d <= cur.d) return order_of_key(cur.ch[0], d);
    return order_of_key(cur.ch[1], d)+nodes[cur.ch[0]].sz+1;
}
// interface
int size() { return nodes[root].sz; }
void insert(const T& v) {
    int lt, rt;
    split_val(root, v, lt, rt);
    root = merge(merge(lt, new_node(v)), rt);
}
bool erase(const T& v, bool all=false) {
    int lt, mt, nt, rt;
    split_val(root, v - 1, lt, mt);
    split_val(mt, v, nt, rt);
    bool found=nodes[nt].d==v;
    if (found && (all or nt==0))
        root = merge(lt, rt);
    else
        root = merge(merge(merge(lt, ch(nt, 0)), ch(nt, 1)), rt);
    return found;
}
int order_of_key(const T& v) {
    return order_of_key(root, v);
}
T find_by_order(int x)  {// order starts from 0!
    assert(x<size() && x>=-1);
    int lt, mt, rt;
    split_size(root, x, mt, rt);
    split_size(mt, x-1, lt, mt);
    root = merge(merge(lt, mt), rt);
        return nodes[mt].d;
    }
    T predecessor(const T& v) {
        return find_by_order(order_of_key(v) - 1);
    }
    T successor(const T& v) {
        int x, y;
        split_val(root, v, x, y);
        int z=y;
        while (ch(z, 0)) z=ch(z, 0);
        root=merge(x, y);
        return nodes[z].d;
    }
};
```

2.15 treap_split.cpp

```
// using treap to maintain a sequence that support multiple operation, index
// 0-based index, change pull(), add(), pushdown() according to the problem
mt19937 gen(chrono::high_resolution_clock::now().time_since_epoch().count());
template <typename T> struct Treap {
    struct node {
        int ch[2], sz;
        unsigned k;
        T d, sum, lazy;
        node(T d_, int z = 1)
            : sz(z), k((unsigned)gen()), d(d_), sum(d), lazy() {
            ch[0] = ch[1] = 0;
        }
    };
    vector<node> nodes;
    int root=0, recyc=0;
    Treap(int size = 2e5) {
        nodes.reserve(size);
        nodes.emplace_back(0, 0);
    }
    inline int &ch(int rt, int r) { return nodes[rt].ch[r]; }
    int new_node(const T &d) {
        int id = (int)nodes.size();
        if (recyc) {
            id = recyc;
            if (ch(recyc, 0) && ch(recyc, 1))
                recyc = merge(ch(recyc, 0), ch(recyc, 1));
            else
                recyc = ch(recyc, ch(recyc, 0) ? 0 : 1);
            nodes[id] = node(d);
        } else
            nodes.push_back(node(d));
        return id;
    }
    int pull(int rt) {
        node &n = nodes[rt];
        n.sz = 1 + nodes[n.ch[0]].sz + nodes[n.ch[1]].sz;
        n.sum = n.d + nodes[n.ch[0]].sum + nodes[n.ch[1]].sum;
        return rt;
    }
    void add(int rt, const T &d) {
        node &n = nodes[rt];
        n.lazy = n.lazy + d;
        n.d = n.d + d;
        n.sum = n.sum + d * n.sz;
    }
    void pushdown(int rt) {
        node &n = nodes[rt];
        if (n.lazy) {
            add(n.ch[0], n.lazy);
            add(n.ch[1], n.lazy);
            n.lazy = T();
        }
    }
    int merge(int tl, int tr) {
        if (!tl) return tr;
        if (!tr) return tl;
        if (nodes[tl].k < nodes[tr].k) {
            pushdown(tl);
            ch(tl, 1) = merge(ch(tl, 1), tr);
            return pull(tl);
        } else {
            pushdown(tr);
            ch(tr, 0) = merge(tl, ch(tr, 0));
            return pull(tr);
        }
    }
    void split(int rt, int k, int &x, int &y) { // split out first k element
        if (!rt) {
            x = y = 0;
            return;
        }
        pushdown(rt);
        if (k <= nodes[ch(rt, 0)].sz) {
            y = rt;
            split(ch(rt, 0), k, x, ch(rt, 0));
            pull(y);
        } else {
            x = rt;
            split(ch(rt, 1), k - nodes[ch(rt, 0)].sz - 1, ch(rt, 1), y);
            pull(x);
        }
    }
    void remove(int &rt) {
        if (recyc == 0) recyc = rt;
        else recyc = merge(recyc, rt);
        rt = 0;
    }
    // interface
    int size() { return nodes[root].sz; }
    const T& operator[](int k) {
        assert(k>=0 && k<size());
        int x, y, z;
        split(root, k+1, y, z);
        split(y, k, x, y);
        root = merge(merge(x, y), z);
        return nodes[y];
    }
    void insert(int k, T v) { // insert at kth position
        assert(k>=0 && k<=size());
        int l, r;
        split(root, k, l, r);
        int rt = new_node(v);
        root = merge(merge(l, rt), r);
    }
    void erase(int l, int r) {
        assert(l>=0 && l<=r && r<size());
        int x, y, z;
        split(root, r + 1, y, z);
        split(y, l, x, y);
        remove(y);
        root = merge(x, z);
    }
    void range_add(int l, int r, T v) {
        assert(l>=0 && l<=r && r<size());
        int x, y, z;
        split(root, r + 1, y, z);
        split(y, l, x, y);
        add(y, v);
        root = merge(merge(x, y), z);
    }
    T getsum(int l, int r) {
        assert(l>=0 && l<=r && r<size());
        int x, y, z;
        split(root, r + 1, y, z);
        split(y, l, x, y);
        T ret = nodes[y].sum;
        root = merge(merge(x, y), z);
        return ret;
    }
};
```

2.16 trie.cpp

```
struct Trie {
  Trie * child[26];
  int nums=0;
  Trie() {
    for(int i=0;i<26;i++) child[i]=NULL;
    nums=0;
  }
};
void insert(Trie *root, const string &s) {
  Trie* r=root;
  for(int i=0;i<s.size();i++){
    int key=s[i]-'a';
    if(r->child[key]==NULL) p->child[key]=new Trie();
    r=r->child[key];
  }
  ++r->nums;
}
bool find(Trie *root, const string &s) {
  Tree* r=root;
  for(int i=0;i<s.size();++i){
    int key=s[i]-'a';
    if(r->child[key]==NULL) return false;
    r=r->child[key];
  }
  return r->nums>0;
}
```

2.17 union_find.cpp

```
struct UF {
    int n;
    vector<int> pa; // parent or size, positive number means parent, negative number means size
    explicit UF(int _n) : n(_n), pa(n, -1) {}
    int find(int x) {
        assert(0 <= x && x < n);
        return pa[x] < 0 ? x : pa[x]=find(pa[x]);
    }
    bool join(int x, int y) {
        assert(0 <= x && x < n && 0 <= y && y < n);
        x=find(x), y=find(y);
        if (x==y) return false;
        if (-pa[x] < -pa[y]) swap(x, y); // size of x is smaller than size of y
        pa[x]+=pa[y];
        pa[y]=x;
        return true;
    }
    int size(int x) {
        assert(0 <= x && x < n);
        return -pa[x];
    }
    vector<vector<int>> groups() {
        vector<int> leader(n);
        for (int i=0; i<n; i++) leader[i]=find(i);
        vector<vector<int>> res(n);
        for (int i=0; i<n; i++) {
            res[leader[i]].push_back(i);
        }
        res.erase(remove_if(res.begin(), res.end(),
                    [](const vector<int>& v) { return v.empty(); }), res.end());
        return res;
    }
};
```

3 Geometry

3.1 angle.h

```
double DEG_to_RAD(double d) { return d*M_PI/180.0; }
double RAD_to_DEG(double r) { return r*180.0/M_PI; }
double rad(P p1,P p2){
        return atan2l(p1.det(p2),p1.dot(p2));
}
bool inAngle(P a, P b, P c, P p) {
  assert(crossOp(a,b,c) != 0);
  if (crossOp(a,b,c) < 0) swap(b,c);
  return crossOp(a,b,p) >= 0 && crossOp(a,c,p) <= 0;
}
double angle(P v, P w) {
  return acos(clamp(v.dot(w) / v.abs() / w.abs(), -1.0, 1.0));
}
double orientedAngle(P a, P b, P c) { // BAC
  if (crossOp(a,b,c) >= 0) return angle(b-a, c-a);
  else return 2*M_PI - angle(b-a, c-a);
}
```

3.2 circle.h

```
// double chord(double r, double ang) return sqrt(2*r*r*(1-cos(ang))); // or 2*r*sin(ang/2)
// double secarea(double r, double ang) {return (ang/2)*(r*r);} // rad
// double segarea(double r, double ang) {return secarea(r, ang) - r*r*sin(ang)/2;}
int type(P o1,double r1,P o2,double r2){
        double d = o1.distTo(o2);
        if(cmp(d,r1+r2) == 1) return 4; // outside each other
        if(cmp(d,r1+r2) == 0) return 3; // touch outside
        if(cmp(d,abs(r1-r2)) == 1) return 2; // one inside another
        if(cmp(d,abs(r1-r2)) == 0) return 1; // touch inside
        return 0;
}
vector<P> isCL(P o,double r,P p1,P p2){
        if (cmp(abs((o-p1).det(p2-p1)/p1.distTo(p2)),r)>0) return {};
        double x = (p1-o).dot(p2-p1), y = (p2-p1).abs2(), d = x * x - y * ((p1-o).abs2() - r*r);
        d = max(d,0.0); P m = p1 - (p2-p1)*(x/y), dr = (p2-p1)*(sqrt(d)/y);
        return {m-dr,m+dr}; //along dir: p1->p2
}
vector<P> isCC(P o1, double r1, P o2, double r2) { //need to check whether two circles are the same
        double d = o1.distTo(o2);
        if (cmp(d, r1 + r2) == 1) return {};
        if (cmp(d,abs(r1-r2))==-1) return {};
        d = min(d, r1 + r2);
        double y = (r1 * r1 + d * d - r2 * r2) / (2 * d), x = sqrt(r1 * r1 - y * y);
        P dr = (o2 - o1).unit();
        P q1 = o1 + dr * y, q2 = dr.rot90() * x;
        return {q1-q2,q1+q2};//along circle 1
}
vector<P> tanCP(P o, double r, P p) {
        double x = (p - o).abs2(), d = x - r * r;
        if (sign(d) <= 0) return {}; // on circle => no tangent
        P q1 = o + (p - o) * (r * r / x);
        P q2 = (p - o).rot90() * (r * sqrt(d) / x);
        return {q1-q2,q1+q2}; //counter clock-wise
}
vector<L> extanCC(P o1, double r1, P o2, double r2) {
        vector<L> ret;
        if (cmp(r1, r2) == 0) {
                P dr = (o2 - o1).unit().rot90() * r1;
                ret.push_back(L(o1 + dr, o2 + dr)), ret.push_back(L(o1 - dr, o2 - dr));
        } else {
                P p = (o2 * r1 - o1 * r2) / (r1 - r2);
                vector<P> ps = tanCP(o1, r1, p), qs = tanCP(o2, r2, p);
                for(int i = 0; i < min(ps.size(),qs.size());i++) ret.push_back(L(ps[i
                    ], qs[i])); //c1 counter-clock wise
        }
        return ret;
}
vector<L> intanCC(P o1, double r1, P o2, double r2) {
        vector<L> ret;
        P p = (o1 * r2 + o2 * r1) / (r1 + r2);
        vector<P> ps = tanCP(o1,r1,p), qs = tanCP(o2,r2,p);
        for(int i = 0; i < min(ps.size(),qs.size()); i++) ret.push_back(L(ps[i], qs[i
            ])); //c1 counter-clock wise
        return ret;
}
double areaCT(double r, P p1, P p2){
        vector<P> is = isCL(P(0,0),r,p1,p2);
        if(is.empty()) return r*r*rad(p1,p2)/2;
        bool b1 = cmp(p1.abs2(),r*r) == 1, b2 = cmp(p2.abs2(), r*r) == 1;
        if(b1 && b2){
                if(sign((p1-is[0]).dot(p2-is[0])) <= 0 &&
                        sign((p1-is[0]).dot(p2-is[0])) <= 0)
                return r*r*(rad(p1,is[0]) + rad(is[1],p2))/2 + is[0].det(is[1])/2;
                else return r*r*rad(p1,p2)/2;
        }
        if(b1) return (r*r*rad(p1,is[0]) + is[0].det(p2))/2;
        if(b2) return (p1.det(is[1]) + r*r*rad(is[1],p2))/2;
        return p1.det(p2)/2;
}
P inCenter(P A, P B, P C) {
        double a = (B - C).abs(), b = (C - A).abs(), c = (A - B).abs();
        return (A * a + B * b + C * c) / (a + b + c);
}
P circumCenter(P a, P b, P c) {
        P bb = b - a, cc = c - a;
        double db = bb.abs2(), dc = cc.abs2(), d = 2 * bb.det(cc);
        return a - P(bb.y * dc - cc.y * db, cc.x * db - bb.x * dc) / d;
}
P othroCenter(P a, P b, P c) {
        P ba = b - a, ca = c - a, bc = b - c;
        double Y = ba.y * ca.y * bc.y,
        A = ca.x * ba.y - ba.x * ca.y,
        x0 = (Y + ca.x * ba.y * b.x - ba.x * ca.y * c.x) / A,
        y0 = -ba.x * (x0 - c.x) / ba.y + ca.y;
        return {x0, y0};
}
```

3.3 geometry.h

```
typedef double T;
const double EPS = 1e-9;
inline int sign(double a) { return a < -EPS ? -1 : a > EPS; }
inline int cmp(double a, double b){ return sign(a-b); }
struct P {
  T x,y;
  P() {}
        P(T _x, T _y) : x(_x), y(_y) {}
  P operator+(P p) {return {x+p.x, y+p.y};}
  P operator-(P p) {return {x-p.x, y-p.y};}
  P operator*(T d) {return {x*d, y*d};}
  P operator/(T d) {return {x/d, y/d};} // only for floatingpoint
  bool operator<(P p) const {
    int c = cmp(x, p.x);
    if (c) return c == -1;
    return cmp(y, p.y) == -1;
  }
  bool operator==(P o) const{
                return cmp(x,o.x) == 0 && cmp(y,o.y) == 0;
        }
  double dot(P p) { return x * p.x + y * p.y; }
  double det(P p) { return x * p.y - y * p.x; }
        double distTo(P p) { return (*this-p).abs(); }
        double alpha() { return atan2(y, x); }
  void read() { cin>>x>>y; }
  void write() {cout<<"("<<x<<","<<y<<")"<<endl;}
  double abs() { return sqrt(abs2());}
        double abs2() { return x * x + y * y; }
        P rot90() { return P(-y,x);}
        P unit() { return *this/abs(); }
  int quad() const { return sign(y) == 1 || (sign(y) == 0 && sign(x) >= 0); }
        P rot(double an){ return {x*cos(an)-y*sin(an),x*sin(an) + y*cos(an)}; }
};
#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))
#define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))
bool isConvex(vector<P> p) {
  bool hasPos=false, hasNeg=false;
  for (int i=0, n=p.size(); i<n; i++) {
    int o = cross(p[i], p[(i+1)%n], p[(i+2)%n]);
    if (o > 0) hasPos = true;
    if (o < 0) hasNeg = true;
  }
  return !(hasPos && hasNeg);
}
bool half(P p) {
  assert(p.x != 0 || p.y != 0); // (0, 0) is not covered
  return p.y > 0 || (p.y == 0 && p.x < 0);
}
void polarSortAround(P o, vector<P> &v) {
  sort(v.begin(), v.end(), [&o](P v, P w) {
      return make_tuple(half(v-o), 0) <
        make_tuple(half(w-o), cross(o, v, w));
  });
}
P proj(P p1, P p2, P q) {
        P dir = p2 - p1;
        return p1 + dir * (dir.dot(q - p1) / dir.abs2());
}
P reflect(P p1, P p2, P q){
        return proj(p1,p2,q) * 2 - q;
}
// tested with https://open.kattis.com/problems/closestpair2
pair<P, P> closest(vector<P> v) {
  assert(sz(v) > 1);
  set <P> S;
  sort(v.begin(), v.end(), [](P a, P b) { return a.y < b.y; });
  pair<T, pair<P, P>> ret{(T)1e18, {P(), P()}};
  int j = 0;
  for(P p : v) {
    P d { 1 + (T) sqrt(ret.first), 0 };
    while(p.y - v[j].y >= d.x) S.erase(v[j++]);
    auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
    for(; lo != hi; ++lo) {
      ret = min(ret, {(p - (*lo)).abs2(), {*lo, p}});
    }
    S.insert(p);
  }
  return ret.second;
}
struct L {
  P ps[2]; P v; T c;
  L() {}
  P& operator[](int i) { return ps[i]; }
  // From direction vector v and offset c
  L(P v, T c) : v(v), c(c) {}
  // From equation ax+by=c
  L(T a, T b, T c) : v({b,-a}), c(c) {}
  // From points P and Q
  L(P p, P q) : v(q-p), c(cross(P(0, 0), v,p)) {
    ps[0] = p;
    ps[1] = q;
  }
  P dir() { return ps[1] - ps[0]; }
  bool include(P p) { return sign((ps[1] - ps[0]).det(p - ps[0])) > 0; }
  T side(P p) {return cross(P(0, 0), v,p)-c;}
  T dist(P p) {return abs(side(p)) / v.abs();}
  T sqDist(P p) {return side(p)*side(p) / (double)v.abs();}
  L perpThrough(P p) {return L(p, p + v.rot90());}
  bool cmpProj(P p, P q) {
    return v.dot(p) < v.dot(q);
  }
  L translate(P t) {return L(v, c + cross(P(0,0), v,t));}
  L shiftLeft(double dist) {return L(v, c + dist*v.abs());}
  L shiftRight(double dist) {return L(v, c - dist*v.abs());}
};
bool chkLL(P p1, P p2, P q1, P q2) {
        double a1 = cross(q1, q2, p1), a2 = -cross(q1, q2, p2);
        return sign(a1+a2) != 0;
}
P isLL(P p1, P p2, P q1, P q2) {
        double a1 = cross(q1, q2, p1), a2 = -cross(q1, q2, p2);
        return (p1 * a2 + p2 * a1) / (a1 + a2);
}
P isLL(L l1,L l2){ return isLL(l1[0],l1[1],l2[0],l2[1]); }
bool parallel(L l0, L l1) { return sign( l0.dir().det( l1.dir() ) ) == 0; }
bool sameDir(L l0, L l1) { return parallel(l0, l1) && sign(l0.dir().dot(l1.dir()) )
    == 1; }
bool cmp (P a,  P b) {
        if (a.quad() != b.quad()) {
                return a.quad() < b.quad();
        } else {
                return sign( a.det(b) ) > 0;
        }
}
bool operator < (L l0, L l1) {
        if (sameDir(l0, l1)) {
                return l1.include(l0[0]);
        } else {
                return cmp( l0.dir(), l1.dir() );
        }
}
bool check(L u, L v, L w) {
        return w.include(isLL(u,v));
}
vector<P> halfPlaneIS(vector<L> &l) {
        sort(l.begin(), l.end());
        deque<L> q;
        for (int i = 0; i < (int)l.size(); ++i) {
                if (i && sameDir(l[i], l[i - 1])) continue;
                while (q.size() > 1 && !check(q[q.size() - 2], q[q.size() - 1], l[i])
                     ) q.pop_back();
                while (q.size() > 1 && !check(q[1], q[0], l[i])) q.pop_front();
                q.push_back(l[i]);
        }
        while (q.size() > 2 && !check(q[q.size() - 2], q[q.size() - 1], q[0])) q.
             pop_back();
        while (q.size() > 2 && !check(q[1], q[0], q[q.size() - 1])) q.pop_front();
        vector<P> ret;
        for (int i = 0; i < (int)q.size(); ++i) ret.push_back(isLL(q[i], q[(i + 1) %
             q.size()]));
        return ret;
}
struct cmpX {
  bool operator()(P a, P b) const {
    return make_pair(a.x, a.y) < make_pair(b.x, b.y);
  }
};
bool intersect(double l1,double r1,double l2,double r2){
        if(l1>r1) swap(l1,r1); if(l2>r2) swap(l2,r2);
        return !( cmp(r1,l2) == -1 || cmp(r2,l1) == -1 );
}
bool isSS(P p1, P p2, P q1, P q2){
        return intersect(p1.x,p2.x,q1.x,q2.x) && intersect(p1.y,p2.y,q1.y,q2.y) &&
        crossOp(p1,p2,q1) * crossOp(p1,p2,q2) <= 0 && crossOp(q1,q2,p1)
                        * crossOp(q1,q2,p2) <= 0;
}
bool isSS_strict(P p1, P p2, P q1, P q2){
        return crossOp(p1,p2,q1) * crossOp(p1,p2,q2) < 0 && crossOp(q1,q2,p1)
                        * crossOp(q1,q2,p2) < 0;
}
bool isMiddle(double a, double m, double b) {
        return sign(a - m) == 0 || sign(b - m) == 0 || (a < m != b < m);
}
bool isMiddle(P a, P m, P b) {
        return isMiddle(a.x, m.x, b.x) && isMiddle(a.y, m.y, b.y);
```

```
}
bool onSeg(P p1, P p2, P q){
        return crossOp(p1,p2,q) == 0 && isMiddle(p1, q, p2);
}
bool onSeg_strict(P p1, P p2, P q){
        return crossOp(p1,p2,q) == 0 && sign((q-p1).dot(p1-p2)) * sign((q-p2).dot(p1-p2)) < 0;
}
double nearest(P p1,P p2,P q){
        P h = proj(p1,p2,q);
        if(isMiddle(p1,h,p2))
                return q.distTo(h);
        return min(p1.distTo(q),p2.distTo(q));
}
double disSS(P p1, P p2, P q1, P q2){
        if(isSS(p1,p2,q1,q2)) return 0;
        return min(min(nearest(p1,p2,q1),nearest(p1,p2,q2)), min(nearest(q1,q2,p1),nearest(q1,q2,p2)));
}
double DEG_to_RAD(double d) { return d*M_PI/180.0; }
double RAD_to_DEG(double r) { return r*180.0/M_PI; }
double rad(P p1,P p2){
        return atan2l(p1.det(p2),p1.dot(p2));
}
bool inAngle(P a, P b, P c, P p) {
  assert(crossOp(a,b,c) != 0);
  if (crossOp(a,b,c) < 0) swap(b,c);
  return crossOp(a,b,p) >= 0 && crossOp(a,c,p) <= 0;
}
double angle(P v, P w) {
  return acos(clamp(v.dot(w) / v.abs() / w.abs(), -1.0, 1.0));
}
double orientedAngle(P a, P b, P c) { // BAC
  if (crossOp(a,b,c) >= 0) return angle(b-a, c-a);
  else return 2*M_PI - angle(b-a, c-a);
}
// double chord(double r, double ang) return sqrt(2*r*r*(1-cos(ang))); // or 2*r*sin(ang/2)
// double secarea(double r, double ang) {return (ang/2)*(r*r);} // rad
// double segarea(double r, double ang) {return secarea(r, ang) - r*r*sin(ang)/2;}
int type(P o1,double r1,P o2,double r2){
        double d = o1.distTo(o2);
        if(cmp(d,r1+r2) == 1) return 4; // outside each other
        if(cmp(d,r1+r2) == 0) return 3; // touch outside
        if(cmp(d,abs(r1-r2)) == 1) return 2; // one inside another
        if(cmp(d,abs(r1-r2)) == 0) return 1; // touch inside
        return 0;
}
vector<P> isCL(P o,double r,P p1,P p2){
        if (cmp(abs((o-p1).det(p2-p1)/p1.distTo(p2)),r)>0) return {};
        double x = (p1-o).dot(p2-p1), y = (p2-p1).abs2(), d = x * x - y * ((p1-o).abs2() - r*r);
        d = max(d,0.0); P m = p1 - (p2-p1)*(x/y), dr = (p2-p1)*(sqrt(d)/y);
        return {m-dr,m+dr}; //along dir: p1->p2
}
vector<P> isCC(P o1, double r1, P o2, double r2) { //need to check whether two circles are the same
        double d = o1.distTo(o2);
        if (cmp(d, r1 + r2) == 1) return {};
        if (cmp(d,abs(r1-r2))==-1) return {};
        d = min(d, r1 + r2);
        double y = (r1 * r1 + d * d - r2 * r2) / (2 * d), x = sqrt(r1 * r1 - y * y);
        P dr = (o2 - o1).unit();
        P q1 = o1 + dr * y, q2 = dr.rot90() * x;
        return {q1-q2,q1+q2};//along circle 1
}
vector<P> tanCP(P o, double r, P p) {
        double x = (p - o).abs2(), d = x - r * r;
        if (sign(d) <= 0) return {}; // on circle => no tangent
        P q1 = o + (p - o) * (r * r / x);
        P q2 = (p - o).rot90() * (r * sqrt(d) / x);
        return {q1-q2,q1+q2}; //counter clock-wise
}
vector<L> extanCC(P o1, double r1, P o2, double r2) {
        vector<L> ret;
        if (cmp(r1, r2) == 0) {
                P dr = (o2 - o1).unit().rot90() * r1;
                ret.push_back(L(o1 + dr, o2 + dr)), ret.push_back(L(o1 - dr, o2 - dr));
        } else {
                P p = (o2 * r1 - o1 * r2) / (r1 - r2);
                vector<P> ps = tanCP(o1, r1, p), qs = tanCP(o2, r2, p);
                for(int i = 0; i < min(ps.size(),qs.size());i++) ret.push_back(L(ps[i], qs[i])); //c1 counter-clock wise
        }
        return ret;
}
vector<L> intanCC(P o1, double r1, P o2, double r2) {
        vector<L> ret;
        P p = (o1 * r2 + o2 * r1) / (r1 + r2);
        vector<P> ps = tanCP(o1,r1,p), qs = tanCP(o2,r2,p);
        for(int i = 0; i < min(ps.size(),qs.size()); i++) ret.push_back(L(ps[i], qs[i])); //c1 counter-clock wise
        return ret;
}
double areaCT(double r, P p1, P p2){
        vector<P> is = isCL(P(0,0),r,p1,p2);
        if(is.empty()) return r*r*rad(p1,p2)/2;
        bool b1 = cmp(p1.abs2(),r*r) == 1, b2 = cmp(p2.abs2(), r*r) == 1;
        if(b1 && b2){
                if(sign((p1-is[0]).dot(p2-is[0])) <= 0 &&
                        sign((p1-is[0]).dot(p2-is[0])) <= 0)
                return r*r*(rad(p1,is[0]) + rad(is[1],p2))/2 + is[0].det(is[1])/2;
                else return r*r*rad(p1,p2)/2;
        }
        if(b1) return (r*r*rad(p1,is[0]) + is[0].det(p2))/2;
        if(b2) return (p1.det(is[1]) + r*r*rad(is[1],p2))/2;
        return p1.det(p2)/2;
}
P inCenter(P A, P B, P C) {
        double a = (B - C).abs(), b = (C - A).abs(), c = (A - B).abs();
        return (A * a + B * b + C * c) / (a + b + c);
}
P circumCenter(P a, P b, P c) {
        P bb = b - a, cc = c - a;
        double db = bb.abs2(), dc = cc.abs2(), d = 2 * bb.det(cc);
        return a - P(bb.y * dc - cc.y * db, cc.x * db - bb.x * dc) / d;
}
P othroCenter(P a, P b, P c) {
        P ba = b - a, ca = c - a, bc = b - c;
        double Y = ba.y * ca.y * bc.y,
        A = ca.x * ba.y - ba.x * ca.y,
        x0 = (Y + ca.x * ba.y * b.x - ba.x * ca.y * c.x) / A,
        y0 = -ba.x * (x0 - c.x) / ba.y + ca.y;
        return {x0, y0};
}
//polygon
double area(vector<P> ps){
        double ret = 0;
  for(int i=0; i< ps.size(); i++) ret += ps[i].det(ps[(i+1)%ps.size()]);
        return ret/2;
}
int contain(vector<P> ps, P p){ //2:inside,1:on_seg,0:outside
        int n = ps.size(), ret = 0;
        for(int i = 0; i < n; i++) {
                P u=ps[i],v=ps[(i+1)%n];
                if(onSeg(u,v,p)) return 1;
                if(cmp(u.y,v.y)<=0) swap(u,v);
                if(cmp(p.y,u.y) >0 || cmp(p.y,v.y) <= 0) continue;
                ret ^= crossOp(p,u,v) > 0;
        }
        return ret*2;
}
vector<P> convexHull(vector<P> ps) {
        int n = ps.size(); if(n <= 1) return ps;
        sort(ps.begin(), ps.end());
        vector<P> qs(n * 2); int k = 0;
        for (int i = 0; i < n; qs[k++] = ps[i++])
                while (k > 1 && crossOp(qs[k - 2], qs[k - 1], ps[i]) <= 0) --k;
        for (int i = n - 2, t = k; i >= 0; qs[k++] = ps[i--])
                while (k > t && crossOp(qs[k - 2], qs[k - 1], ps[i]) <= 0) --k;
        qs.resize(k - 1);
        return qs;
}
vector<P> convexHullNonStrict(vector<P> ps) {
        //caution: need to unique the Ps first
        int n = ps.size(); if(n <= 1) return ps;
        sort(ps.begin(), ps.end());
        vector<P> qs(n * 2); int k = 0;
        for (int i = 0; i < n; qs[k++] = ps[i++])
                while (k > 1 && crossOp(qs[k - 2], qs[k - 1], ps[i]) < 0) --k;
        for (int i = n - 2, t = k; i >= 0; qs[k++] = ps[i--])
                while (k > t && crossOp(qs[k - 2], qs[k - 1], ps[i]) < 0) --k;
        qs.resize(k - 1);
        return qs;
}
double convexDiameter(vector<P> ps){
        int n = ps.size(); if(n <= 1) return 0;
        int is = 0, js = 0; for(int k = 1; k < n; k++) is = ps[k]<ps[is]?k:is, js = ps[js] < ps[k]?k:js;
        int i = is, j = js;
        double ret = ps[i].distTo(ps[j]);
        do{
                if((ps[(i+1)%n]-ps[i]).det(ps[(j+1)%n]-ps[j]) >= 0)
                        (++j)%=n;
                else
                        (++i)%=n;
                ret = max(ret,ps[i].distTo(ps[j]));
        }while(i!=is || j!=js);
        return ret;
}
vector<P> convexCut(const vector<P>&ps, P q1, P q2) {
        vector<P> qs;
        int n = ps.size();
        for(int i = 0; i<n; i++) {
                P p1 = ps[i], p2 = ps[(i+1)%n];
                int d1 = crossOp(q1,q2,p1), d2 = crossOp(q1,q2,p2);
                if(d1 >= 0) qs.push_back(p1);
                if(d1 * d2 < 0) qs.push_back(isLL(p1,p2,q1,q2));
        }
        return qs;
}
```

3.4 line.h

```
struct L {
  P ps[2]; P v; T c;
  L() {}
  P& operator[](int i) { return ps[i]; }
  // From direction vector v and offset c
  L(P v, T c) : v(v), c(c) {}
  // From equation ax+by=c
  L(T a, T b, T c) : v({b,-a}), c(c) {}
  // From points P and Q
  L(P p, P q) : v(q-p), c(cross(P(0, 0), v,p)) {
    ps[0] = p;
    ps[1] = q;
  }
  P dir() { return ps[1] - ps[0]; }
  bool include(P p) { return sign((ps[1] - ps[0]).det(p - ps[0])) > 0; }
  T side(P p) {return cross(P(0, 0), v,p)-c;}
  T dist(P p) {return abs(side(p)) / v.abs();}
  T sqDist(P p) {return side(p)*side(p) / (double)v.abs();}
  L perpThrough(P p) {return L(p, p + v.rot90());}
  bool cmpProj(P p, P q) {
    return v.dot(p) < v.dot(q);
  }
  L translate(P t) {return L(v, c + cross(P(0,0), v,t));}
  L shiftLeft(double dist) {return L(v, c + dist*v.abs());}
  L shiftRight(double dist) {return L(v, c - dist*v.abs());}
};
bool chkLL(P p1, P p2, P q1, P q2) {
        double a1 = cross(q1, q2, p1), a2 = -cross(q1, q2, p2);
        return sign(a1+a2) != 0;
}
P isLL(P p1, P p2, P q1, P q2) {
        double a1 = cross(q1, q2, p1), a2 = -cross(q1, q2, p2);
        return (p1 * a2 + p2 * a1) / (a1 + a2);
}
P isLL(L l1,L l2){ return isLL(l1[0],l1[1],l2[0],l2[1]); }
bool parallel(L l0, L l1) { return sign( l0.dir().det( l1.dir() ) ) == 0; }
bool sameDir(L l0, L l1) { return parallel(l0, l1) && sign(l0.dir().dot(l1.dir()) )
    == 1; }
bool cmp (P a,  P b) {
        if (a.quad() != b.quad()) {
                return a.quad() < b.quad();
        } else {
                return sign( a.det(b) ) > 0;
        }
}
bool operator < (L l0, L l1) {
        if (sameDir(l0, l1)) {
                return l1.include(l0[0]);
        } else {
                return cmp( l0.dir(), l1.dir() );
        }
}
bool check(L u, L v, L w) {
        return w.include(isLL(u,v));
}
vector<P> halfPlaneIS(vector<L> &l) {
        sort(l.begin(), l.end());
        deque<L> q;
        for (int i = 0; i < (int)l.size(); ++i) {
                if (i && sameDir(l[i], l[i - 1])) continue;
                while (q.size() > 1 && !check(q[q.size() - 2], q[q.size() - 1], l[i])
                    ) q.pop_back();
                while (q.size() > 1 && !check(q[1], q[0], l[i])) q.pop_front();
                q.push_back(l[i]);
        }
        while (q.size() > 2 && !check(q[q.size() - 2], q[q.size() - 1], q[0])) q.
            pop_back();
        while (q.size() > 2 && !check(q[1], q[0], q[q.size() - 1])) q.pop_front();
        vector<P> ret;
        for (int i = 0; i < (int)q.size(); ++i) ret.push_back(isLL(q[i], q[(i + 1) %
            q.size()]));
        return ret;
}
```

3.5 point.h

```
typedef double T;
const double EPS = 1e-9;
inline int sign(double a) { return a < -EPS ? -1 : a > EPS; }
inline int cmp(double a, double b){ return sign(a-b); }
struct P {
  T x,y;
  P() {}
        P(T _x, T _y) : x(_x), y(_y) {}
  P operator+(P p) {return {x+p.x, y+p.y};}
  P operator-(P p) {return {x-p.x, y-p.y};}
  P operator*(T d) {return {x*d, y*d};}
  P operator/(T d) {return {x/d, y/d};} // only for floatingpoint
  bool operator<(P p) const {
    int c = cmp(x, p.x);
    if (c) return c == -1;
    return cmp(y, p.y) == -1;
  }
  bool operator==(P o) const{
                return cmp(x,o.x) == 0 && cmp(y,o.y) == 0;
        }
  double dot(P p) { return x * p.x + y * p.y; }
  double det(P p) { return x * p.y - y * p.x; }
        double distTo(P p) { return (*this-p).abs(); }
        double alpha() { return atan2(y, x); }
  void read() { cin>>x>>y; }
  void write() {cout<<"("<<x<<","<<y<<")"<<endl;}
  double abs() { return sqrt(abs2());}
        double abs2() { return x * x + y * y; }
        P rot90() { return P(-y,x);}
        P unit() { return *this/abs(); }
  int quad() const { return sign(y) == 1 || (sign(y) == 0 && sign(x) >= 0); }
        P rot(double an){ return {x*cos(an)-y*sin(an),x*sin(an) + y*cos(an)}; }
};
#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))
#define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))
bool isConvex(vector<P> p) {
  bool hasPos=false, hasNeg=false;
  for (int i=0, n=p.size(); i<n; i++) {
    int o = cross(p[i], p[(i+1)%n], p[(i+2)%n]);
    if (o > 0) hasPos = true;
    if (o < 0) hasNeg = true;
  }
  return !(hasPos && hasNeg);
}
bool half(P p) {
  assert(p.x != 0 || p.y != 0); // (0, 0) is not covered
  return p.y > 0 || (p.y == 0 && p.x < 0);
}
void polarSortAround(P o, vector<P> &v) {
  sort(v.begin(), v.end(), [&o](P v, P w) {
      return make_tuple(half(v-o), 0) <
        make_tuple(half(w-o), cross(o, v, w));
  });
}
P proj(P p1, P p2, P q) {
        P dir = p2 - p1;
        return p1 + dir * (dir.dot(q - p1) / dir.abs2());
}
P reflect(P p1, P p2, P q){
        return proj(p1,p2,q) * 2 - q;
}
// tested with https://open.kattis.com/problems/closestpair2
pair<P, P> closest(vector<P> v) {
  assert(sz(v) > 1);
  set <P> S;
  sort(v.begin(), v.end(), [](P a, P b) { return a.y < b.y; });
  pair<T, pair<P, P>> ret{(T)1e18, {P(), P()}};
  int j = 0;
  for(P p : v) {
    P d { 1 + (T) sqrt(ret.first), 0 };
    while(p.y - v[j].y >= d.x) S.erase(v[j++]);
    auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
    for(; lo != hi; ++lo) {
      ret = min(ret, {(p - (*lo)).abs2(), {*lo, p}});
    }
    S.insert(p);
  }
  return ret.second;
}
```

3.6 polygon.h

```
//polygon
double area(vector<P> ps){
        double ret = 0;
  for(int i=0; i< ps.size(); i++) ret += ps[i].det(ps[(i+1)%ps.size()]);
        return ret/2;
}
int contain(vector<P> ps, P p){ //2:inside,1:on_seg,0:outside
        int n = ps.size(), ret = 0;
        for(int i = 0; i < n; i++) {
                P u=ps[i],v=ps[(i+1)%n];
                if(onSeg(u,v,p)) return 1;
                if(cmp(u.y,v.y)<=0) swap(u,v);
                if(cmp(p.y,u.y) >0 || cmp(p.y,v.y) <= 0) continue;
                ret ^= crossOp(p,u,v) > 0;
        }
        return ret*2;
}
vector<P> convexHull(vector<P> ps) {
        int n = ps.size(); if(n <= 1) return ps;
        sort(ps.begin(), ps.end());
        vector<P> qs(n * 2); int k = 0;
        for (int i = 0; i < n; qs[k++] = ps[i++])
                while (k > 1 && crossOp(qs[k - 2], qs[k - 1], ps[i]) <= 0) --k;
        for (int i = n - 2, t = k; i >= 0; qs[k++] = ps[i--])
                while (k > t && crossOp(qs[k - 2], qs[k - 1], ps[i]) <= 0) --k;
        qs.resize(k - 1);
        return qs;
}
vector<P> convexHullNonStrict(vector<P> ps) {
        //caution: need to unique the Ps first
        int n = ps.size(); if(n <= 1) return ps;
        sort(ps.begin(), ps.end());
        vector<P> qs(n * 2); int k = 0;
        for (int i = 0; i < n; qs[k++] = ps[i++])
                while (k > 1 && crossOp(qs[k - 2], qs[k - 1], ps[i]) < 0) --k;
        for (int i = n - 2, t = k; i >= 0; qs[k++] = ps[i--])
                while (k > t && crossOp(qs[k - 2], qs[k - 1], ps[i]) < 0) --k;
        qs.resize(k - 1);
        return qs;
}
double convexDiameter(vector<P> ps){
        int n = ps.size(); if(n <= 1) return 0;
        int is = 0, js = 0; for(int k = 1; k < n; k++) is = ps[k]<ps[is]?k:is, js =
            ps[js] < ps[k]?k:js;
        int i = is, j = js;
        double ret = ps[i].distTo(ps[j]);
        do{
                if((ps[(i+1)%n]-ps[i]).det(ps[(j+1)%n]-ps[j]) >= 0)
                        (++j)%=n;
                else
                        (++i)%=n;
                ret = max(ret,ps[i].distTo(ps[j]));
        }while(i!=is || j!=js);
        return ret;
}
vector<P> convexCut(const vector<P>&ps, P q1, P q2) {
        vector<P> qs;
        int n = ps.size();
        for(int i = 0; i<n; i++) {
                P p1 = ps[i], p2 = ps[(i+1)%n];
                int d1 = crossOp(q1,q2,p1), d2 = crossOp(q1,q2,p2);
                if(d1 >= 0) qs.push_back(p1);
                if(d1 * d2 < 0) qs.push_back(isLL(p1,p2,q1,q2));
        }
        return qs;
}
```

3.7 segment.h

```
struct cmpX {
  bool operator()(P a, P b) const {
    return make_pair(a.x, a.y) < make_pair(b.x, b.y);
  }
};
bool intersect(double l1,double r1,double l2,double r2){
        if(l1>r1) swap(l1,r1); if(l2>r2) swap(l2,r2);
        return !( cmp(r1,l2) == -1 || cmp(r2,l1) == -1 );
}
bool isSS(P p1, P p2, P q1, P q2){
        return intersect(p1.x,p2.x,q1.x,q2.x) && intersect(p1.y,p2.y,q1.y,q2.y) &&
        crossOp(p1,p2,q1) * crossOp(p1,p2,q2) <= 0 && crossOp(q1,q2,p1)
                        * crossOp(q1,q2,p2) <= 0;
}
bool isSS_strict(P p1, P p2, P q1, P q2){
        return crossOp(p1,p2,q1) * crossOp(p1,p2,q2) < 0 && crossOp(q1,q2,p1)
                        * crossOp(q1,q2,p2) < 0;
}
bool isMiddle(double a, double m, double b) {
        return sign(a - m) == 0 || sign(b - m) == 0 || (a < m != b < m);
}
bool isMiddle(P a, P m, P b) {
        return isMiddle(a.x, m.x, b.x) && isMiddle(a.y, m.y, b.y);
}
bool onSeg(P p1, P p2, P q){
        return crossOp(p1,p2,q) == 0 && isMiddle(p1, q, p2);
}
bool onSeg_strict(P p1, P p2, P q){
        return crossOp(p1,p2,q) == 0 && sign((q-p1).dot(p1-p2)) * sign((q-p2).dot(p1-
            p2)) < 0;
}
double nearest(P p1,P p2,P q){
        P h = proj(p1,p2,q);
        if(isMiddle(p1,h,p2))
                return q.distTo(h);
        return min(p1.distTo(q),p2.distTo(q));
}
double disSS(P p1, P p2, P q1, P q2){
        if(isSS(p1,p2,q1,q2)) return 0;
        return min(min(nearest(p1,p2,q1),nearest(p1,p2,q2)), min(nearest(q1,q2,p1),
            nearest(q1,q2,p2)));
}
```

4 Graph

4.1 2-sat.cpp

```
// suppose you have some boolean variables a, b, c, d...
// assign each variable true or false such that the expression like
// the following is true:
// (a or not b) and (not a or b) and (not a or not b) and (a or not c)
// the expression is a conjunction of multiple clauses, where each clause
// is a disjunction of exactly two literals

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

struct two_SAT {
    int n;
    SCC g;
    two_SAT(int n) : n(n), g(n*2) {} // n is the number of literals
    void add(int u, bool neg_u, int v, bool neg_v) { // neg_u is if u is negated,
         same for v
        g.add_edge(2*u+neg_u, 2*v+!neg_v);
        g.add_edge(2*v+neg_v, 2*u+!neg_u);
    }

    vector<bool> solve() {
        g.solve();
        de(g.color);
        vector<bool> res(n);
        for (int i=0; i<n; i++) {
            if (g.color[2*i]==g.color[2*i+1]) return {};
            res[i]=g.color[2*i]>g.color[2*i+1];
        }
        return res;
    }
};
```

4.2 BellmanFord.cpp

```
struct BellmanFord {
    static constexpr long long INF=1e18;
    int n, last_relaxed=-1;
    vector<tuple<int, int, int>> edges;
    vector<bool> bad; //has negative cycle on the path
    vector<int> pre;
    vector<ll> dis;
    BellmanFord(int _n) : n(_n), bad(n), pre(n), dis(n, INF) {}
    void add_edge(int u, int v, int w) {
        edges.emplace_back(u, v, w);
    }
    void run(int start) {
        dis[start]=0;
        for (int i=0; i<n-1; i++) {
            for (auto [u, v, w] : edges) {
                if (dis[u]<INF && dis[v]>dis[u]+w) {
                    dis[v]=dis[u]+w;
                    pre[v]=u;
                }
            }
        }
        for (auto [u, v, w] : edges) {
            if (dis[u]<INF && dis[v]>dis[u]+w) {
                dis[v]=dis[u]+w;
                bad[v]=true;
                last_relaxed=v;
                pre[v]=u;
            }
        }
        for (int i=0; i<n; i++) {
            for (auto [u, v, w] : edges) {
                if (bad[u]) bad[v]=true;
            }
        }
    }
    vector<int> find_cycle() {
        dis.assign(n, 0); // without this, only cycle reachable from 0 will be
             counted
        run(0);
        if (last_relaxed==-1) return {};
        int x=last_relaxed;
        for (int i=0; i<n; i++) x=pre[x];
        vector<int> cycle;
        for (int cur=x; ; cur=pre[cur]) {
            cycle.push_back(cur);
            if (cur==x && cycle.size()>1) break;
        }
        reverse(cycle.begin(), cycle.end());
        return cycle;
    }
    long long get_dis(int x) {
        return bad[x] ? -INF : dis[x];
    }
};
```

4.3 Hopcroft-Karp.cpp

```
struct HopcroftKarp {
    int n, m;
    Dinic flow;
    vector<int> l, r;
    HopcroftKarp(int n, int m) : n(n), m(m), flow(n+m+2), l(n, -1), r(m, -1) {}
    void add_edge(int u, int v) {
        flow.addEdge(u, n+v, 1);
    }
    int solve() {
        for (int i=0; i<n; i++)
            flow.addEdge(n+m, i, 1);
        for (int i=0; i<m; i++)
            flow.addEdge(n+i, n+m+1, 1);
        int res = flow.maxFlow(n+m, n+m+1);
        for (int i=0; i<n; i++) {
            if (flow.match[i]!=-1) {
                l[i]=flow.match[i]-n;
                r[flow.match[i]-n]=i;
            }
        }
        return res;
    }
};
int main() {
    ios::sync_with_stdio(false);
    int l, r, m;
    cin>>l>>r>>m;
    HopcroftKarp g(l, r);
    while (m--) {
        int u, v;
        cin>>u>>v;
        g.add_edge(u, v);
    }
    cout<<g.solve()<<'\n';
    for (int i=0; i<l; i++) {
        if (g.l[i]!=-1) cout<<i<<' '<<g.l[i]<<'\n';
    }
}
```

4.4 MCMF.cpp

```
struct Flow {
  static inline constexpr ll INF = INT64_MAX >> 1;
  int n;
  vector<tuple<int, int, int>> e;
  vector<vector<int>> g;
  vector<int> prev;
  vector<ll> h; // distance, also potential
  Flow(int n) : n(n), g(n), h(n), prev(n) {}
  void addEdge(int u, int v, int w, int c) {
    if (u == v) return;
    g[u].emplace_back(e.size());
    e.emplace_back(v, w, c);
    g[v].emplace_back(e.size());
    e.emplace_back(u, 0, -c);
  }
  bool dijkstra(int s, int t) {
    priority_queue<pair<ll, int>> q;
    fill(prev.begin(), prev.end(), -1);
    vector<ll> d(n, INF);
    d[s] = 0;
    q.push({0, s});
    while (!q.empty()) {
      auto [du, u] = q.top();
      q.pop();
      if (d[u] != -du) continue;
      for (auto i : g[u]) {
        auto [v, w, c] = e[i];
        c += h[u] - h[v];
        if (w > 0 && d[v] > d[u] + c) {
          d[v] = d[u] + c;
          prev[v] = i;
          q.push({-d[v], v});
        }
      }
    }
    for (int i = 0; i < n; ++i) {
      if ((h[i] += d[i]) > INF) h[i] = INF;
    }
    return h[t] != INF;
  }
  pair<ll, ll> maxFlow(int s, int t) {
    ll flow = 0, cost = 0;
    while (dijkstra(s, t)) {
      int f = INT_MAX, now = t;
      vector<int> r;
      while (now != s) {
        r.emplace_back(prev[now]);
        f = min(f, get<1>(e[prev[now]]));
        now = get<0>(e[prev[now] ^ 1]);
      }
      for (auto i : r) {
        get<1>(e[i]) -= f;
        get<1>(e[i ^ 1]) += f;
      }
      flow += f;
      cost += ll(f) * h[t];
    }
    return {flow, cost};
  }
};
```

4.5 augmented_path_BPM.cpp

```
// augmented path algorithm for maximum-caredinality bipartite matching
// Worst time complexity: O(nm), but very hard to hack (since we can shuffle),
// usually runs extremely fast, 2e5 vertices and edges in 60 ms.
mt19937 rng(1);
struct aug_path {
    vector<vector<int>> g;
    vector<int> L, R, vis;
    aug_path(int n, int m) : g(n), L(n, -1), R(m, -1), vis(n) {}
    void add_edge(int a, int b) { g[a].push_back(b); }
    bool match(int u) {
        if (vis[u]) return false;
        vis[u] = true;
        for (auto v : g[u]) {
            if (R[v] == -1) {
                L[u] = v;
                R[v] = u;
                return true;
            }
        }
        for (auto vec : g[u]) {
            if (match(R[vec])) {
                L[u] = vec;
                R[vec] = u;
                return true;
            }
        }
        return false;
    }
    int solve() {
        // shuffle to avoid counter test case, but may be slightly slower
        // for (auto& v : g)
        //     shuffle(v.begin(), v.end(), rng);
        // vector<int> order(L.size());
        // iota(order.begin(), order.end(), 0);
        // shuffle(order.begin(), order.end(), rng);
        bool ok = true;
        while (ok) {
            ok=false;
            fill(vis.begin(), vis.end(), 0);
            // for (auto i : order)
            for (int i = 0; i < (int)L.size(); ++i)
                if (L[i] == -1) ok |= match(i);
        }
        int ret = 0;
        for (int i = 0; i < L.size(); ++i)
            ret += (L[i] != -1);
        return ret;
    }
};
int main() {
    ios::sync_with_stdio(false);
    int l, r, m;
    cin>>l>>r>>m;
    aug_path g(l, r);
    while (m--) {
        int u, v;
        cin>>u>>v;
        g.add_edge(u, v);
    }
    cout<<g.solve()<<'\n';
    for (int i=0; i<l; i++) {
        if (g.L[i]!=-1) cout<<i<<' '<<g.L[i]<<'\n';
    }
}
```

4.6 biconnected_components.cpp

```
#include <vector>
using namespace std;
struct BCC {
    int n, pos = 0;
    vector<vector<int>> g;
    vector<int> ord, low, cuts, stk;
    vector<vector<int>> comps; // components
    BCC(int n_) : n(n_), g(n), ord(n, -1), low(n) {}
    void add_edge(int u, int v) {
        g[u].push_back(v);
        g[v].push_back(u);
    }
    void dfs(int u, int pa) {
        low[u] = ord[u] = pos++;
        stk.push_back(u);
        int cnt=0;
        bool is_cut = false;
        for (auto v : g[u]) {
            if (v == pa) continue;
            if (ord[v] == -1) {
                cnt++;
                dfs(v, u);
                low[u] = min(low[u], low[v]);
                if (low[v] >= ord[u]) {
                    if (u != pa || cnt > 1) is_cut = true;
                    // the subtree will be disconnected if we remove vertex u,
                    // do something if needed
                    comps.emplace_back();
                    while (true) {
                        int back = stk.back();
                        stk.pop_back();
                        comps.back().push_back(back);
                        if (back == v) break;
                    }
                    comps.back().push_back(u);
                }
            } else low[u]=min(low[u], ord[v]);
        }
        if (is_cut) cuts.push_back(u);
    }
    void solve() {
        for (int i = 0; i < n; i++) {
            if (ord[i] == -1) dfs(i, i);
        }
    }
};
```

4.7 binary_lifting.cpp

```
struct Binary_lifting {
    const int sz, level;
    const vector<vector<int>>& g;
    vector<vector<int>> pa;
    vector<int> dep;
    Binary_lifting(const vector<vector<int>>& g_) :
        sz((int)g_.size()),
        level(__lg(sz)+2),
        g(g_),
        pa(sz, vector<int>(level)),
        dep(g.size()) {}
    void dfs(int u, int p) {
        pa[u][0] = p;
        dep[u] = dep[p] + 1;
        for (int i = 1; i < level; i++) {
            pa[u][i] = pa[pa[u][i - 1]][i - 1];
        }
        for (auto v : g[u]) {
            if (v == p) continue;
            dfs(v, u);
        }
    };
    int jump(int u, int step) {
        for (int i=0; i<level; i++) {
            if (step>>i&1) u=pa[u][i];
        }
        return u;
    }
    int lca(int x, int y) {
        if (dep[x] > dep[y]) swap(x, y);
        y=jump(y, dep[y] - dep[x]);
        if (x == y) return x;
        for (int i=level-1; i>=0; i--) {
            if (pa[x][i] != pa[y][i]) {
                x = pa[x][i];
                y = pa[y][i];
            }
        }
        return pa[x][0];
    }
};
```

4.8 blossom.cpp

```
// https://codeforces.com/blog/entry/92339
// another faster algorithm https://judge.yosupo.jp/submission/51928
#include <bits/stdc++.h>

using namespace std;

struct blossom {
    int n, m;
    vector<int> mate;
    vector<vector<int>> b;
    vector<int> p, d, bl;
    vector<vector<int>> g;
    blossom(int n) : n(n) {
        m = n + n / 2;
        mate.assign(n, -1);
        b.resize(m);
        p.resize(m);
        d.resize(m);
        bl.resize(m);
        g.assign(m, vector<int>(m, -1));
    }
    void add_edge(int u, int v) {
        g[u][v] = u;
        g[v][u] = v;
    }
```

```
void match(int u, int v) {
    g[u][v] = g[v][u] = -1;
    mate[u] = v;
    mate[v] = u;
}
vector<int> trace(int x) {
    vector<int> vx;
    while(true) {
        while(bl[x] != x) x = bl[x];
        if(!vx.empty() && vx.back() == x) break;
        vx.push_back(x);
        x = p[x];
    }
    return vx;
}
void contract(int c, int x, int y, vector<int> &vx, vector<int> &vy) {
    b[c].clear();
    int r = vx.back();
    while(!vx.empty() && !vy.empty() && vx.back() == vy.back()) {
        r = vx.back();
        vx.pop_back();
        vy.pop_back();
    }
    b[c].push_back(r);
    b[c].insert(b[c].end(), vx.rbegin(), vx.rend());
    b[c].insert(b[c].end(), vy.begin(), vy.end());
    for(int i = 0; i <= c; i++) {
        g[c][i] = g[i][c] = -1;
    }
    for(int z : b[c]) {
        bl[z] = c;
        for(int i = 0; i < c; i++) {
            if(g[z][i] != -1) {
                g[c][i] = z;
                g[i][c] = g[i][z];
            }
        }
    }
}
vector<int> lift(vector<int> &vx) {
    vector<int> A;
    while(vx.size() >= 2) {
        int z = vx.back(); vx.pop_back();
        if(z < n) {
            A.push_back(z);
            continue;
        }
        int w = vx.back();
        int i = (A.size() % 2 == 0 ? find(b[z].begin(), b[z].end(), g[z][w]) - b[
             z].begin() : 0);
        int j = (A.size() % 2 == 1 ? find(b[z].begin(), b[z].end(), g[z][A.back()
             ]) - b[z].begin() : 0);
        int k = b[z].size();
        int dif = (A.size() % 2 == 0 ? i % 2 == 1 : j % 2 == 0) ? 1 : k - 1;
        while(i != j) {
            vx.push_back(b[z][i]);
            i = (i + dif) % k;
        }
        vx.push_back(b[z][i]);
    }
    return A;
}
int solve() {
    for(int ans = 0; ; ans++) {
        fill(d.begin(), d.end(), 0);
        queue<int> Q;
        for(int i = 0; i < m; i++) bl[i] = i;
        for(int i = 0; i < n; i++) {
            if(mate[i] == -1) {
                Q.push(i);
                p[i] = i;
                d[i] = 1;
            }
        }
        int c = n;
        bool aug = false;
        while(!Q.empty() && !aug) {
            int x = Q.front(); Q.pop();
            if(bl[x] != x) continue;
            for(int y = 0; y < c; y++) {
                if(bl[y] == y && g[x][y] != -1) {
                    if(d[y] == 0) {
                        p[y] = x;
                        d[y] = 2;
                        p[mate[y]] = y;
                        d[mate[y]] = 1;
                        Q.push(mate[y]);
                    }else if(d[y] == 1) {
                        vector<int> vx = trace(x);
                        vector<int> vy = trace(y);
                        if(vx.back() == vy.back()) {
                            contract(c, x, y, vx, vy);
                            Q.push(c);
                            p[c] = p[b[c][0]];
                            d[c] = 1;
                            c++;
                        }else {
                            aug = true;
                            vx.insert(vx.begin(), y);
                            vy.insert(vy.begin(), x);
                            vector<int> A = lift(vx);
                            vector<int> B = lift(vy);
                            A.insert(A.end(), B.rbegin(), B.rend());
                            for(int i = 0; i < (int) A.size(); i += 2) {
                                match(A[i], A[i + 1]);
                                if(i + 2 < (int) A.size()) add_edge(A[i + 1], A[i
                                      + 2]);
                            }
                        }
                        break;
                    }
                }
            }
            if(!aug) return ans;
        }
    }
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n, m;
    cin >> n >> m;
    blossom B(n);
    for(int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        B.add_edge(u, v);
    }
    cout << B.solve() << '\n';
    for(int i = 0; i < n; i++) {
        if(i < B.mate[i]) {
            cout << i << ' ' << B.mate[i] << '\n';
        }
    }
};
```

4.9 bridges.cpp

```
struct Bridge {
    int n, pos=0;
    vector<vector<pair<int, int>>> g; // graph, component
    vector<int> ord, low, bridges; // order, low link, belong to which component
    Bridge(int n) : n(n), g(n), ord(n, -1), low(n) {}
    void add_edge(int u, int v, int i) {
        g[u].emplace_back(v, i);
        g[v].emplace_back(u, i);
    }
    void dfs(int u, int p) {
        ord[u] = low[u] = pos++;
        int cnt = 0;
        for (auto [v, i] : g[u]) {
            // in case there're repeated edges, only skip the first one
            if (v == p && cnt == 0) {
                cnt++;
                continue;
            }
            if (ord[v] == -1) dfs(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] > ord[u]) bridges.push_back(i);
        }
    }
    void solve() {
        for (int i = 0; i < n; i++)
            if (ord[i] == -1) dfs(i, i);
    }
};
```

4.10 dijkstra.cpp

```
constexpr long long INF=1e18;
template<typename G>
vector<long long> dijkstra(const G& g, int start) {
    vector dis(g.size(), INF);
    // vector<pii> pre[N];
    using node=pair<long long, int>;
    priority_queue<node, vector<node>, greater<>> q;
    dis[start] = 0;
    q.emplace(0, start);
    while (!q.empty()) {
        auto [d, u] = q.top();
        q.pop();
        if (d != dis[u]) continue;
        for (auto [v, cost] : g[u]) {
            if (dis[v] > dis[u] + cost) {
                dis[v] = dis[u] + cost;
                // pre[v].clear();
                // pre[v].pb({cost,u});
                q.emplace(dis[v], v);
            }
            // else if(dis[v]==dis[u]+cost)
            // pre[v].pb({cost,u});
        }
    }
    return dis;
}
// dijkstra for small edge weight (less than 10) aka 1-k bfs
vector<int> SmallDijkstra(const vector<vector<pair<int, int>>>& g, int src, int lim)
     {
    vector<vector<int>> qs(lim);
    vector<int> dis(g.size(), -1);
    dis[src] = 0;
    qs[0].push_back(src);
    for (int d = 0, maxd = 0; d <= maxd; ++d) {
        for (auto& q = qs[d % lim]; q.size(); ) {
            int u = q.back();
            q.pop_back();
            if (dis[u] != d) continue;
            for (auto [v, c] : g[u]) {
                if (dis[v] != -1 && dis[v] <= d + c) continue;
                dis[v] = d + c;
                qs[(d + c) % lim].push_back(v);
                maxd = max(maxd, d + c);
            }
        }
    }
    return dis;
}
```

4.11 dinic.cpp

```
// indexed from 0!
struct Dinic {
    static constexpr int INF = 1e9;
    int n;
    struct Edge {
        int to, cap;
        Edge(int to, int cap) : to(to), cap(cap) {}
};
vector<Edge> e;
vector<std::vector<int>> g;
vector<int> cur, h; // h = shortest distance from source, calculated in bfs
// after computing flow, edge (u, v) such that h[u]!=-1 and h[v]==-1 are part of
     min cut
Dinic(int n) : n(n), g(n) {}
bool bfs(int s, int t) {
    h.assign(n, -1);
    std::queue<int> que;
    h[s] = 0;
    que.push(s);
    while (!que.empty()) {
        int u = que.front();
        que.pop();
        for (int i : g[u]) {
            auto [v, c] = e[i];
            if (c > 0 && h[v] == -1) {
                h[v] = h[u] + 1;
                if (v == t) return true;
                que.push(v);
            }
        }
    }
    return false;
}
int dfs(int u, int t, int f) {
    if (u == t) return f;
    int r = f;
    for (int &i = cur[u]; i < int(g[u].size()); ++i) {
        int j = g[u][i];
        auto [v, c] = e[j];
        if (c > 0 && h[v] == h[u] + 1) {
            int a = dfs(v, t, std::min(r, c));
            e[j].cap -= a;
            e[j ^ 1].cap += a;
            r -= a;
            if (r == 0) return f;
        }
    }
    return f - r;
}
void addEdge(int u, int v, int c) {
    g[u].push_back((int)e.size());
    e.emplace_back(v, c);
    g[v].push_back((int)e.size());
    e.emplace_back(u, 0);
}
int maxFlow(int s, int t) {
    int ans = 0;
    while (bfs(s, t)) {
        cur.assign(n, 0);
        ans += dfs(s, t, INF);
    }
```

```
        return ans;
    }
};
```

4.12 divide_and_conquer_on_trees.cpp

```
vector<vector<pair<int, int>>> g;
vector<int> query, subtreeSize, parent;
vector<bool> blocked;
// calculate substree size
void calSize(int u, int p) {
    parent[u] = p;
    subtreeSize[u] = 1;
    for (auto [v, w] : g[u]) {
        if (v == p || blocked[v]) continue;
        calSize(v, u);
        subtreeSize[u] += subtreeSize[v];
    }
}
// if needed solveTree can return value
void solveTree(int root) {
    queue<pii> cur; // store the result for current subtree
    for (auto [v, w] : g[root]) {
        if (blocked[v]) continue;
        queue<pair<int, int>> q; // change if type of element if needed
        q.push({v, w});
        while (!q.empty()) {
            auto [u, dis] = q.front();
            q.pop();
            // do ... to update answer
            cur.push({dis, len});
            for (auto [to, wei] : g[u]) {
                if (to == parent[u] || blocked[to]) continue;
                q.push({to, dis + wei});
            }
        }
        while (!cur.empty()) {
            auto [dis, len] = cur.front();
            // do ... to update the result for the current tree
            cur.pop();
        }
    }
}
// return some value if needed
void go(int entry) {
    calSize(entry, entry);
    int centroid = entry;
    int bestSize = subtreeSize[entry];
    queue<int> q;
    q.push(entry);
    while (!q.empty()) {
        int u = q.front();
        q.pop();

        int size = subtreeSize[entry] - subtreeSize[u];
        for (auto [v, w] : g[u]) {
            if (v == parent[u] || blocked[v]) continue;
            size = max(size, subtreeSize[v]);
            q.push(v);
        }
        if (size < bestSize) centroid = u, bestSize = size;
    }
    calSize(centroid, centroid);
    blocked[centroid] = true;
    // do ... to clear the previous result
    solveTree(centroid);
    for (auto [v, w] : g[centroid]) {
        if (!blocked[v]) go(v);
    }
}
```

4.13 dsu_on_tree.cpp

```
int main() {
    vector<int> bch(n, -1);
    int cur_big = -1;
    auto get_big = [&](auto &dfs, int u, int p) -> int {
        int sz = 1, mx = 0;
        for (auto v : g[u]) {
            if (v == p) continue;
            int csz = dfs(dfs, v, u);
            if (csz > mx) mx = csz, bch[u] = v;
            sz += csz;
        }
        return sz;
    };
    auto add = [&](auto &slf, int u, int p, int x) -> void {
        // update info of u here
        for (auto v : g[u]) {
            if (v == p || v == cur_big) continue;
            slf(slf, v, u, x);
        }
    };
    auto dfs = [&](auto &dfs, int u, int pa, bool keep) -> void {
        int big = bch[u];
        for (auto v : g[u])
            if (v != pa && v != big) dfs(dfs, v, u, 0);
        if (big != -1) {
            dfs(dfs, big, u, 1);
            cur_big = big;
        }
        add(add, u, pa, 1);
        // now you get all the info of subtree of u, answer queries about u
        // here.
        cur_big = -1;
        if (!keep) add(add, u, pa, -1);
    };
}
```

4.14 eulerian-path-directed.cpp

```
struct Eulerian_path {
    int n, edge_cnt = 0;
    vector<vector<pair<int, int>>> g;
    vector<int> path, indeg, outdeg;
    vector<bool> used;
    Eulerian_path(int _n) : n(_n), g(n), indeg(n), outdeg(n) {}
    void add_edge(int u, int v) {
        g[u].emplace_back(v, edge_cnt);
        indeg[v]++, outdeg[u]++;
        edge_cnt++;
    }
    void dfs(int u) {
        while (!g[u].empty()) {
            auto [v, edge] = g[u].back();
            g[u].pop_back();
            if (used[edge]) continue;
            used[edge] = true;
            dfs(v);
        }
        path.push_back(u);
    }
    vector<int> solve(int start) {
        for (int i = 0; i < n; i++)
            if (indeg[i] != outdeg[i]) return {};
        used.resize(edge_cnt);
        dfs(start);
        if ((int)path.size() != edge_cnt + 1)
            return {}; // the graph is not connected
        reverse(path.begin(), path.end());
        return path;
    }

    vector<int> solve(int start, int end) {
        add_edge(start, end);
        auto res = solve(end);
        if (!empty(res))
            res.erase(res.begin()); // the first edge has to be the newly
                                // added edge
        return res;
    }
};
```

4.15 eulerian-path.cpp

```
struct Eulerian_path {
    int n, edge_cnt = 0;
    vector<vector<pair<int, int>>> g;
    vector<int> path, deg;
    vector<bool> used;
    Eulerian_path(int _n) : n(_n), g(n), deg(n) {}
    void add_edge(int u, int v) {
        g[u].emplace_back(v, edge_cnt);
        g[v].emplace_back(u, edge_cnt);
        deg[u]++, deg[v]++;
        edge_cnt++;
    }
    void dfs(int u) {
        while (!g[u].empty()) {
            auto [v, edge] = g[u].back();
            g[u].pop_back();
            if (used[edge]) continue;
            used[edge] = true;
            dfs(v);
        }
        path.push_back(u);
    }
    vector<int> solve(int start) {
        for (auto x : deg)
            if (x % 2) return {};
        used.resize(edge_cnt);
        dfs(start);
        if ((int)path.size() != edge_cnt + 1)
            return {}; // the graph is not connected
        reverse(path.begin(), path.end());
        return path;
    }

    vector<int> solve(int start, int end) {
        add_edge(start, end);
        auto res = solve(end);
        if (!empty(res))
            res.erase(res.begin()); // the first edge has to be the newly added edge
        return res;
    }
};
```

4.16 heavy-light_decomp.cpp

```
#include <vector>

#include "../DataStructure/fenwick.cpp"
using namespace std;
struct HLD {
    vector<vector<int>> g;
    vector<int> pa, dep, heavy, head, pos, posr; // initialize heavy with -1
    int cnt=0;
    fenwick<long long> tr;
    HLD(int n) : g(n), pa(n), dep(n), heavy(n, -1), head(n), pos(n), posr(n), tr(n)
         {}
    void add_edge(int u, int v) {
        g[u].push_back(v);
        g[v].push_back(u);
    }
    int dfs(int u) {
        int size = 1;
        int mx = 0;
        for (int v : g[u]) {
            if (v != pa[u]) {
                pa[v] = u, dep[v] = dep[u] + 1;
                int csize = dfs(v);
                size += csize;
                if (csize > mx) mx = csize, heavy[u] = v;
            }
        }
        return size;
    }
    void dfs2(int u, int h) {
        head[u] = h, pos[u] = cnt++; //0-based index
        if (heavy[u] != -1) dfs2(heavy[u], h);
        for (int v : g[u]) {
            if (v != pa[u] && v != heavy[u])
                dfs2(v, v);
        }
        posr[u] = cnt;
    }
    long long pathsum(int u, int v) {
        long long res = 0;
        while (head[u] != head[v]) {
            if (dep[head[u]] < dep[head[v]]) swap(u, v);
            res += tr.query(pos[head[u]], pos[u]);
            u = pa[head[u]];
        }
        if (pos[u] > pos[v]) swap(u, v);
        res += tr.query(pos[u], pos[v]);
        return res;
    }
    int lca(int u, int v) {
        while (head[u] != head[v]) {
            if (dep[head[u]] > dep[head[v]]) u = pa[head[u]];
            else v = pa[head[v]];
        }
        return dep[u] > dep[v] ? v : u;
    }
};
```

4.17 hungarian.cpp

```
// credits: https://github.com/the-tourist/algo/blob/master/flows/hungarian.cpp
// hungarian algorithm for bipartite graph matching, matches every node on the
// left with a node on the right and the sum of the weights is minimal.
// a[i][j] is the cost for i in L to be matched with j in R. (0-indexed)
// pa[i] is the node in R matched with i
// pb[j] is the node in L matched with j
// Negate the cost for max cost.
// Time: O(n^2M)
template<typename T>
struct Hungarian {
    int n, m;
    vector< vector<T> > a;
    vector<T> u, v;
    vector<int> pa, pb, way;
    vector<T> minv;
    vector<bool> used;
    T inf;
    Hungarian(int _n, int _m) : n(_n), m(_m), a(n, vector<T>(m)), u(n+1), v(m+1), pa(n+1, -1), pb(m+1, -1), way(m, -1), minv(m), used(m+1) {
        assert(n <= m);
        inf = numeric_limits<T>::max();
    }
    inline void add_row(int i) {
        fill(minv.begin(), minv.end(), inf);
        fill(used.begin(), used.end(), false);
        pb[m] = i;
        pa[i] = m;
        int j0 = m;
        do {
            used[j0] = true;
            int i0 = pb[j0];
            T delta = inf;
            int j1 = -1;
            for (int j = 0; j < m; j++) {
                if (!used[j]) {
                    T cur = a[i0][j] - u[i0] - v[j];
                    if (cur < minv[j]) {
                        minv[j] = cur;
                        way[j] = j0;
                    }
                    if (minv[j] < delta) {
                        delta = minv[j];
                        j1 = j;
                    }
                }
            }
            for (int j = 0; j <= m; j++) {
                if (used[j]) {
                    u[pb[j]] += delta;
                    v[j] -= delta;
                } else {
                    minv[j] -= delta;
                }
            }
            j0 = j1;
        } while (pb[j0] != -1);
        do {
            int j1 = way[j0];
            pb[j0] = pb[j1];
            pa[pb[j0]] = j0;
            j0 = j1;
        } while (j0 != m);
    }
    inline T current_score() {
        return -v[m];
    }
    inline T solve() {
        for (int i = 0; i < n; i++) {
            add_row(i);
        }
        return current_score();
    }
};
```

4.18 kosaraju_SCC.cpp

```
struct kosaraju {
    int n;
    vector<bool> vis;
    vector<int> color, order;
    vector<vector<int>> g, g2, comp;
    kosaraju(int n_) : n(n_), vis(n), color(n, -1), g(n), g2(n) {}
    void add_edge(int u, int v) {
        g[u].push_back(v);
        g2[v].push_back(u);
    }
    void dfs1(int u) {
        vis[u] = true;
        for (int v : g[u])
            if (!vis[v]) dfs1(v);
        order.push_back(u);
    }
    void dfs2(int u) {
        comp.back().push_back(u);
        for (int v : g2[u])
            if (color[v] == -1)
                dfs2(v);
    }
    void solve() {
        for (int i = 0; i < n; ++i)
            if (!vis[i]) dfs1(i);
        for (int i = n - 1; i >= 0; --i)
            if (color[order[i]] == -1) {
                comp.emplace_back();
                dfs2(order[i]);
            }
        // reverse(comp.begin(), comp.end()); to sort components in topological
        // order
        for (int i = 0; i < (int)comp.size(); i++) {
            for (int x : comp[i])
                color[x] = i;
        }
    }
};
```

4.19 push-relabel.cpp

```
/*
        Push Relabel O(n^3) implimentation using FIFO method to chose push
   vertex. This uses gapRelabel heuristic to fasten the process even further. If
   only the maxFlow value is required then the algo can be stopped as soon as
   the gap relabel method is called. However, to get the actual flow values in
   the edges, we need to let the algo terminate itself.
        This implimentation assumes zero based vertex indexing. Edges to the
   graph can be added using the addEdge method only. capacity for residual edges
   is set to be zero. To get the actual flow values iterate through the edges
   and check for flow for an edge with cap > 0.
        This implimentaion is superior over dinic's for graphs where graph is
   dense locally at some places and mostly sparse. For randomly generated
   graphs, this implimentation gives results within seconds for n = 10000 nodes,
   m = 1000000 edges.
        Code Tested on : SPOJ FASTFLOW
        @author : triveni
*/
typedef int fType;
struct edge {
    int from, to;
    fType cap, flow;
    edge(int from, int to, fType cap, fType flow = 0)
        : from(from), to(to), cap(cap), flow(flow) {}
};
struct PushRelabel {
    int N;
    vector<edge> edges;
    vector<vector<int>> G;
    vector<int> h, inQ, count;
    vector<fType> excess;
    queue<int> Q;
    PushRelabel(int N) : N(N), count(N << 1), G(N), h(N), inQ(N), excess(N) {}
    void addEdge(int from, int to, int cap) {
        G[from].push_back(edges.size());
        edges.push_back(edge(from, to, cap));
        G[to].push_back(edges.size());
        edges.push_back(edge(to, from, 0));
    }
    void enQueue(int u) {
        if (!inQ[u] && excess[u] > 0) Q.push(u), inQ[u] = true;
    }
    void Push(int edgeIdx) {
        edge &e = edges[edgeIdx];
        int toPush = min<fType>(e.cap - e.flow, excess[e.from]);
        if (toPush > 0 && h[e.from] > h[e.to]) {
            e.flow += toPush;
            excess[e.to] += toPush;
            excess[e.from] -= toPush;
            edges[edgeIdx ^ 1].flow -= toPush;
            enQueue(e.to);
        }
    }
    void Relabel(int u) {
        count[h[u]] -= 1;
        h[u] = 2 * N - 2;
        for (int i = 0; i < G[u].size(); ++i) {
            edge &e = edges[G[u][i]];
            if (e.cap > e.flow) h[u] = min(h[u], h[e.to]);
        }
        count[++h[u]] += 1;
    }
    void gapRelabel(int height) {
        for (int u = 0; u < N; ++u)
            if (h[u] >= height && h[u] < N) {
                count[h[u]] -= 1;
                count[h[u] = N] += 1;
                enQueue(u);
            }
    }
    void Discharge(int u) {
        for (int i = 0; excess[u] > 0 && i < G[u].size(); ++i) {
            Push(G[u][i]);
        }
        if (excess[u] > 0) {
            if (h[u] < N && count[h[u]] < 2) gapRelabel(h[u]);
            else
                Relabel(u);
        } else if (!Q.empty()) { // dequeue
            Q.pop();
            inQ[u] = false;
        }
    }
    fType getFlow(int src, int snk) {
        h[src] = N;
        inQ[src] = inQ[snk] = true;
        count[0] = N - (count[N] = 1);
        for (int i = 0; i < G[src].size(); ++i) {
            excess[src] += edges[G[src][i]].cap;
            Push(G[src][i]);
        }
        while (!Q.empty()) {
            Discharge(Q.front());
        }
        return excess[snk];
    }
};
int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    PushRelabel df(n);
    while (m--) {
        int x, y, c;
        // cin >> x >> y >> c; // 0- based index
        scanf("%d%d%d", &x, &y, &c);
        --x, --y;
        if (x != y) {
            df.addEdge(x, y, c);
            df.addEdge(y, x, c);
        }
    }
    cout << df.getFlow(0, n - 1) << "\n";
    return 0;
}
```

4.20 tarjan_SCC.cpp

```
// Note that strictly speaking this is not the original tarjan's algorithm
// because we use a slightly different definition for lowlink. However this
// algorithm is still correctly and easier to code.
// See: https://cs.stackexchange.com/questions/96635/tarjans-scc-example-showing-
     necessity-of-lowlink-definition-and-calculation-r?rq=1

#include <vector>
using namespace std;
struct SCC {
    int n, pos = 0;
    vector<vector<int>> g;
    vector<bool> on_stk;
    vector<int> low, ord, stk, color;
    vector<vector<int>> comp;
    SCC(int _n) : n(_n), g(n), on_stk(n), low(n), ord(n, -1), color(n) {}
    void add_edge(int u, int v) { g[u].push_back(v); }
    void dfs(int u) {
        low[u] = ord[u] = pos++;
        stk.push_back(u);
        on_stk[u] = true;
        for (auto v : g[u]) {
            if (ord[v] == -1) dfs(v);
            if (on_stk[v]) low[u] = min(low[u], low[v]);
        }
        if (low[u] == ord[u]) {
            comp.emplace_back();
            while (true) {
                int v = stk.back();
                stk.pop_back();
                on_stk[v] = false;
                comp.back().push_back(v);
                if (u == v) break;
            }
        }
    }
    void solve() {
        for (int i = 0; i < n; i++)
            if (ord[i] == -1) dfs(i);
        // reverse(comp.begin(), comp.end()); to sort components in topological
        // order
        for (int i = 0; i < (int)comp.size(); i++) {
            for (int x : comp[i])
                color[x] = i;
        }
    }
};
```

4.21 two_edge_connected_components.cpp

```
struct TECC {
    int n, pos=0;
    vector<int> ord, low, color; // order, low link, belong to which component
    vector<vector<int>> g, comp; // graph, component
    TECC(int n) : n(n), ord(n, -1), low(n), color(n, -1), g(n) {}
    void add_edge(int u, int v) {
        g[u].emplace_back(v);
        g[v].emplace_back(u);
    }
    bool is_bridge(int u, int v) {
        if (ord[u] > ord[v]) swap(u, v);
        return ord[u] < low[v];
    }
    void dfs(int u, int p) {
        ord[u] = low[u] = pos++;
        int cnt = 0;
        for (int v : g[u]) {
            // in case there're repeated edges, only skip the first one
            if (v == p && cnt == 0) {
                cnt++;
                continue;
            }
            if (ord[v] == -1) dfs(v, u);
            low[u] = min(low[u], low[v]);
        }
    }
    void fill_component(int u) {
        comp.back().emplace_back(u);
        for (int v : g[u]) {
            if (color[v] != -1 || is_bridge(v, u)) continue;
            color[v] = color[u];
            fill_component(v);
        }
    }
    int build() {
        for (int i = 0; i < n; i++)
            if (ord[i] == -1) dfs(i, i);
        int k = 0;
        for (int i = 0; i < n; i++) {
            if (color[i] != -1) continue;
            color[i] = k++;
            comp.emplace_back();
            fill_component(i);
        }
        return k;
    }
};
int main() {
    int n, m;
    cin >> n >> m;
    TECC g(n);
    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        g.add_edge(a, b);
    }
    int k = g.build();
    cout << k << '\n';
    for (int i = 0; i < k; i++) {
        cout << g.comp[i].size() << ' ';
        for (int v : g.comp[i])
            cout << v << ' ';
    }
    return 0;
}
```

5 Math

5.1 BSGS.cpp

```
// solve a^x=b(mod n), 0<= x <n
#define MOD 76543
int hs[MOD], head[MOD], next[MOD], id[MOD], top;
void insert(int x, int y) {
    int k = x % MOD;
    hs[top] = x, id[top] = y, next[top] = head[k], head[k] = top++;
}
int find(int x) {
    int k = x % MOD;
    for (int i = head[k]; i != -1; i = next[i])
        if (hs[i] == x) return id[i];
    return -1;
}
int BSGS(int a, int b, int n) {
    memset(head,-1, sizeof(head));
    top = 1;
    if (b == 1) return 0;
    int m = sqrt(n * 1.0), j;
    long long x = 1, p = 1;
    for (int i = 0; i < m; ++i, p = p * a % n)
        insert(p * b % n, i);
    for (long long i = m;; i += m) {
        if ((j = find(x = x * p % n)) != -1) return i-j;
        if (i > n) break;
    }
    return -1;
}
```

5.2 ChineseRT.cpp

```
// a x + b y = gcd(a, b)
ll extgcd(ll a, ll b, ll &x, ll &y) {
    ll g = a; x = 1; y = 0;
    if (b != 0) g = extgcd(b, a % b, y, x), y -= (a / b) * x;
    return g;
}
// Solve linear congruences equation:
// a[i] * x = b[i] MOD m[i] (mi don't need to be co-prime)
// M - lcm, x - smalleset integer solution
bool chinese(const vector<ll> &a, const vector<ll> &b, const vector<ll> &m, ll &x, ll
      &M) {
    ll n = a.size();
    x = 0; M = 1;
    for(int i = 0; i < n; i++) {
        ll a_ = a[i] * M, b_ = b[i] - a[i] * x, m_ = m[i];
        ll y, t, g = extgcd(a_, m_, y, t);
        if (b_ % g) return false;
        b_ /= g; m_ /= g;
        x += M * (y * b_ % m_);
        M *= m_;
    }
    x = (x + M) % M;
    return true;
}
```

5.3 binomial.cpp

```
#include <vector>
using namespace std;
inline namespace binomial {
    using T = mint;
    // using T = long long;
    vector<vector<T>> binom;
    void init(int n) {
        binom.resize(n+1, vector<T>(n+1));
        binom[0][0]=1;
        for (int i=1; i<=n; i++) {
            binom[i][0]=binom[i][i]=1;
            for (int j=1; j<i; j++)
                binom[i][j]=binom[i-1][j]+binom[i-1][j-1];
        }
    }

    T C(int n, int m) { // n choose m
        if (m<0 || m>n) return T{};
        return binom[n][m];
    }
}
```

5.4 euclid.h

```
ll euclid(ll a, ll b, ll &x, ll &y) {
        if (!b) return x = 1, y = 0, a;
        ll d = euclid(b, a % b, y, x);
        return y -= a/b * x, d;
}
```

5.5 euler.h

```
#define NEGPOW(e) ((e) % 2 ? -1 : 1)

int jacobi(int a, int m) {
    if (a == 0) return m == 1 ? 1 : 0;
    if (a % 2)  return NEGPOW((a-1)*(m-1)/4)*jacobi(m%a, a);
    else return NEGPOW((m*m-1)/8)*jacobi(a/2, m);
}

int invMod(int a, int m) {
    int x, y;
    if (extgcd(a, m, x, y) == 1) return (x + m) % m;
    else return 0; // unsolvable
}

// No solution when: n(p-1)/2 = -1 mod p
int sqrtMod(int n, int p) {
  int S, Q, W, i, m = invMod(n, p);
  for (Q = p - 1, S = 0; Q % 2 == 0; Q /= 2, ++S);
  do { W = rand() % p; } while (W == 0 || jacobi(W, p) != -1);
  for (int R = powMod(n, (Q+1)/2, p), V = powMod(W, Q, p); ;) {
    int z = R * R * m % p;
    for (i = 0; i < S && z % p != 1; z *= z, ++i);
    if (i == 0) return R;
    R = (R * powMod(V, 1 << (S-i-1), p)) % p;
  }
}

bool eulercriterion(int n, int p) {
  if(powMod(n, (p-1)/2, p) == 1) return true;
  return false;
}

int powMod(int a, int b, int p) {
  int res=1;
  while(b) {
    if(b&1) res=int( res * 1ll * a % p), --b;
    else a=int (a * 1ll * a%p), b>>=1;
  }
  return res;
}
```

5.6 exGCD.cpp

```
#include<bits/stdc++.h>
using ll=long long;
//   {g, x, y}: ax+by=gcd(a,b)
tuple<ll, ll, ll> exgcd(ll a, ll b) {
    if (b==0) return {a, 1, 0};
    auto [g, x, y]=exgcd(b, a%b);
    return {g, y, x-a/b*y};
}
/*
solve ax+by=c, equivalently ax=c (mod b)
all solutions: x=x0+b/g*t, y=y0-a/g*t
smallest positive x=(x0%t+t)%t, where t=b/g
*/
bool liEu(ll a, ll b, ll c, ll& x, ll& y) {
    ll g;
    tie(g, x, y)=exgcd(a, b);
    if (c % g != 0) return false;
    ll k = c / g;
    x *= k;
    y *= k;
    // smallest positive x:
    // b/=g;
    // x=(x%b+b)%b;
    return true;
}
```

5.7 factorial.hpp

```
inline namespace Factorial {
vector<mint> fac, invfac;
void init(int n) {
    fac.resize(n + 1);
    invfac.resize(n + 1);
    fac[0] = 1;
    for (int i = 1; i <= n; i++)
        fac[i] = fac[i - 1] * i;
    invfac[n] = fac[n].inv();
    for (int i = n - 1; i >= 0; i--)
        invfac[i] = invfac[i + 1] * (i + 1);
}
mint C(int n, int k) { // n choose m
    if (k < 0 || k > n) return 0;
    assert((int)size(fac) > n);
    return fac[n] * invfac[n - k] * invfac[k];
}
mint P(int n, int m) { // n choose m with permutation
    assert(!fac.empty());
    return fac[n] * invfac[n - m];
}
} // namespace Factorial
```

5.8 factorization.cpp

```
namespace Fractorization {
    using u64 = uint64_t;
    using u128 = __uint128_t;
    using ull = unsigned long long;
    mt19937 rand(chrono::steady_clock::now().time_since_epoch().count());

    u64 binPow(u64 a, u64 b, u64 mod){
        if(b == 0) return 1;
        if(b&1) return (u128)a * binPow(a, b^1, mod) % mod;
    return binPow((u128)a * a % mod, b>>1, mod);
}
bool checkComp(u64 n, u64 a, u64 d, int s){
    u64 x = binPow(a, d, n);
    if(x == 1 || x == n-1) return false;
    for (int r=1; r<s; r++) {
        x = (u128)x * x % n;
        if(x == n-1) return false;
    }
    return true;
}
bool RabinMiller(u64 n){
    if(n < 2) return false;
    int r = 0;
    u64 d = n-1;
    while(!(d & 1))
        d >>= 1, r++;
    for(int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}){
        if(n == a) return true;
        if(checkComp(n, a, d, r)) return false;
    }
    return true;
}
ull mult(ull a, ull b, ull mod){
    return (u128)a * b % mod;
}
ull rho(ull n) { // wiull find a factor < n, but not necessarily prime
    if (~n & 1) return 2;
    ull c = rand() % n, x = rand() % n, y = x, d = 1;
    while (d == 1) {
        x = (mult(x, x, n) + c) % n;
        y = (mult(y, y, n) + c) % n;
        y = (mult(y, y, n) + c) % n;
        d = gcd(max(x, y)-min(x, y), n);
    }
    return d == n ? rho(n) : d;
}
vector<pair<ull, int>> factorRho(ull n) {
    map <ull, int> fact;
    function<void(ull)> factRho=[&](ull n){
        if(n == 1) return;
        if(RabinMiller(n)){
            fact[n]++;
            return;
        }
        ull factor = rho(n);
        factRho(factor);
        factRho(n/factor);
    };
    factRho(n);
    vector<pair<ull, int>> facts;
    for (auto& p : fact) facts.push_back(p);
    return facts;
}
vector<pair<int, int>> factor(int n) {
    vector<pair<int, int>> facts;
        for (int f=2; f*f<=n; f++) {
            if (n%f==0) {
                int c=0;
                while (n%f==0) {
                    n/=f;
                    c++;
                }
                facts.emplace_back(f, c);
            }
        }
        return facts;
    }
}
using namespace Fractorization;
```

5.9 fft.cpp

```
#include <bits/stdc++.h>

using namespace std;

using cd = complex<double>;
const double PI = acos(-1);
void fft(vector<cd>& a, bool invert) {
    int n = a.size();
    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1) j ^= bit;
        j ^= bit;
        if (i < j) swap(a[i], a[j]);
    }
    for (int len = 2; len <= n; len <<= 1) {
        double ang = 2 * PI / len * (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2; j++) {
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }
    if (invert) {
        for (cd & x : a) x /= n;
    }
}

vector<int> multiply(vector<int> const& a, vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++) fa[i] *= fb[i];
    fft(fa, true);

    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

5.10 gauss.h

```
const double EPS = 1e-9;
const int INF = 2;

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

  vector<int> where (m, -1);
  for (int col=0, row=0; col<m && row<n; ++col) {
    int sel = row;
    for (int i=row; i<n; ++i)
      if (abs (a[i][col]) > abs (a[sel][col]))
        sel = i;
    if (abs (a[sel][col]) < EPS)
      continue;
    for (int i=col; i<=m; ++i)
      swap (a[sel][i], a[row][i]);
    where[col] = row;

    for (int i=0; i<n; ++i)
      if (i != row) {
        double c = a[i][col] / a[row][col];
        for (int j=col; j<=m; ++j)
          a[i][j] -= a[row][j] * c;
      }
    ++row;
  }

  ans.assign (m, 0);
  for (int i=0; i<m; ++i)
    if (where[i] != -1)
      ans[i] = a[where[i]][m] / a[where[i]][i];
  for (int i=0; i<n; ++i) {
    double sum = 0;
    for (int j=0; j<m; ++j)
      sum += ans[j] * a[i][j];
    if (abs (sum - a[i][m]) > EPS)
      return 0;
  }

  for (int i=0; i<m; ++i)
    if (where[i] == -1)
      return INF;
  return 1;
}
```

5.11 inverse.h

```
const ll MOD = 998244353;
vector<ll> inv(n+1);
inv[1]=1;
for(int i = 2; i < n + 1; ++i) inv[i] = MOD - (MOD/i) * inv[MOD % i] % MOD;
```

5.12 lucas.h

```
// when n and m are big but p is small
ll Lucas(ll n, ll m, ll p) {
  if (m == 0) return 1;
  return (C(n % p, m % p, p) * Lucas(n / p, m / p, p)) % p;
}
```

5.13 mod_int.hpp

```
template <int MOD>
struct ModInt {
    int val;
    ModInt(int v = 0) : val(v % MOD) { if (val < 0) val += MOD; };
    ModInt operator+() const { return ModInt(val); }
    ModInt operator-() const { return ModInt(MOD - val); }
    ModInt inv() const {
        auto a = val, m = MOD, u = 0, v = 1;
        while (a != 0) { auto t = m / a; m -= t * a; swap(a, m); u -= t * v; swap(u,
             v); }
        assert(m == 1);
        return u;
    }
    friend ModInt operator+ (ModInt lhs, const ModInt& rhs) { return lhs += rhs; }
    friend ModInt operator- (ModInt lhs, const ModInt& rhs) { return lhs -= rhs; }
    friend ModInt operator* (ModInt lhs, const ModInt& rhs) { return lhs *= rhs; }
    friend ModInt operator/ (ModInt lhs, const ModInt& rhs) { return lhs /= rhs; }
    ModInt& operator+=(const ModInt& x) { if ((val += x.val) >= MOD) val -= MOD;
         return *this; }
    ModInt& operator-=(const ModInt& x) { if ((val -= x.val) < 0) val += MOD; return
         *this; }
    ModInt& operator*=(const ModInt& x) { val = int64_t(val) * x.val % MOD; return *
         this; }
    ModInt& operator/=(const ModInt& x) { return *this *= x.inv(); }
    bool operator==(const ModInt& b) const { return val == b.val; }
    bool operator!=(const ModInt& b) const { return val != b.val; }
    friend std::istream& operator>>(std::istream& is, ModInt& x) noexcept { return is
          >> x.val; }
    friend std::ostream& operator<<(std::ostream& os, const ModInt& x) noexcept {
         return os << x.val; }
};
using mint = ModInt<1'000'000'007>;
```

5.14 nfft.h

```
using i64 = long long;
using u64 = unsigned long long;
using u32 = unsigned;
constexpr int P = 998244353;
std::vector<int> rev, roots{0, 1};
int power(int a, int b) {
    int res = 1;
    for (; b; b >>= 1, a = 1ll * a * a % P)
        if (b & 1)
```

```
            res = 1ll * res * a % P;
    return res;
}
void dft(std::vector<int> &a) {
    int n = a.size();
    if (int(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; ++i)
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
    }
    for (int i = 0; i < n; ++i)
        if (rev[i] < i)
            std::swap(a[i], a[rev[i]]);
    if (int(roots.size()) < n) {
        int k = __builtin_ctz(roots.size());
        roots.resize(n);
        while ((1 << k) < n) {
            int e = power(3, (P - 1) >> (k + 1));
            for (int i = 1 << (k - 1); i < (1 << k); ++i) {
                roots[2 * i] = roots[i];
                roots[2 * i + 1] = 1ll * roots[i] * e % P;
            }
            ++k;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; ++j) {
                int u = a[i + j];
                int v = 1ll * a[i + j + k] * roots[k + j] % P;
                int x = u + v;
                if (x >= P)
                    x -= P;
                a[i + j] = x;
                x = u - v;
                if (x < 0)
                    x += P;
                a[i + j + k] = x;
            }
        }
    }
}
void idft(std::vector<int> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    int inv = power(n, P - 2);
    for (int i = 0; i < n; ++i)
        a[i] = 1ll * a[i] * inv % P;
}
struct Poly {
    std::vector<int> a;
    Poly() {}
    Poly(int a0) {
        if (a0)
```

```
        a = {a0};
}
Poly(const std::vector<int> &a1) : a(a1) {
    while (!a.empty() && !a.back())
        a.pop_back();
}
int size() const {
    return a.size();
}
int operator[](int idx) const {
    if (idx < 0 || idx >= size())
        return 0;
    return a[idx];
}
Poly mulxk(int k) const {
    auto b = a;
    b.insert(b.begin(), k, 0);
    return Poly(b);
}
Poly modxk(int k) const {
    k = std::min(k, size());
    return Poly(std::vector<int>(a.begin(), a.begin() + k));
}
Poly divxk(int k) const {
    if (size() <= k)
        return Poly();
    return Poly(std::vector<int>(a.begin() + k, a.end()));
}
friend Poly operator+(const Poly a, const Poly &b) {
    std::vector<int> res(std::max(a.size(), b.size()));
    for (int i = 0; i < int(res.size()); ++i) {
        res[i] = a[i] + b[i];
        if (res[i] >= P)
            res[i] -= P;
    }
    return Poly(res);
}
friend Poly operator-(const Poly a, const Poly &b) {
    std::vector<int> res(std::max(a.size(), b.size()));
    for (int i = 0; i < int(res.size()); ++i) {
        res[i] = a[i] - b[i];
        if (res[i] < 0)
            res[i] += P;
    }
    return Poly(res);
}
friend Poly operator*(Poly a, Poly b) {
    int sz = 1, tot = a.size() + b.size() - 1;
    while (sz < tot)
        sz *= 2;
    a.a.resize(sz);
    b.a.resize(sz);
    dft(a.a);
    dft(b.a);
    for (int i = 0; i < sz; ++i)
        a.a[i] = 1ll * a[i] * b[i] % P;
        idft(a.a);
        return Poly(a.a);
    }
    Poly &operator+=(Poly b) {
        return (*this) = (*this) + b;
    }
    Poly &operator-=(Poly b) {
        return (*this) = (*this) - b;
    }
    Poly &operator*=(Poly b) {
        return (*this) = (*this) * b;
    }
    Poly deriv() const {
        if (a.empty())
            return Poly();
        std::vector<int> res(size() - 1);
        for (int i = 0; i < size() - 1; ++i)
            res[i] = 1ll * (i + 1) * a[i + 1] % P;
        return Poly(res);
    }
    Poly integr() const {
        if (a.empty())
            return Poly();
        std::vector<int> res(size() + 1);
        for (int i = 0; i < size(); ++i)
            res[i + 1] = 1ll * a[i] * power(i + 1, P - 2) % P;
        return Poly(res);
    }
    Poly inv(int m) const {
        Poly x(power(a[0], P - 2));
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (2 - modxk(k) * x)).modxk(k);
        }
        return x.modxk(m);
    }
    Poly log(int m) const {
        return (deriv() * inv(m)).integr().modxk(m);
    }
    Poly exp(int m) const {
        Poly x(1);
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (1 - x.log(k) + modxk(k))).modxk(k);
        }
        return x.modxk(m);
    }
    Poly sqrt(int m) const {
        Poly x(1);
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2);
        }
        return x.modxk(m);
    }
    Poly mulT(Poly b) const {
        if (b.size() == 0)
            return Poly();
        int n = b.size();
        std::reverse(b.a.begin(), b.a.end());
        return ((*this) * b).divxk(n - 1);
    }
    std::vector<int> eval(std::vector<int> x) const {
        if (size() == 0)
            return std::vector<int>(x.size(), 0);
        const int n = std::max(int(x.size()), size());
        std::vector<Poly> q(4 * n);
        std::vector<int> ans(x.size());
        x.resize(n);
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                q[p] = std::vector<int>{1, (P - x[l]) % P};
            } else {
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m, r);
                q[p] = q[2 * p] * q[2 * p + 1];
            }
        };
        build(1, 0, n);
        std::function<void(int, int, int, const Poly &)> work = [&](int p, int l, int
           r, const Poly &num) {
            if (r - l == 1) {
                if (l < int(ans.size()))
                    ans[l] = num[0];
            } else {
                int m = (l + r) / 2;
                work(2 * p, l, m, num.mulT(q[2 * p + 1]).modxk(m - l));
                work(2 * p + 1, m, r, num.mulT(q[2 * p]).modxk(r - m));
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
    }
};
```

5.15 power.h

```
ll qpow(ll a, ll b) {
  ll res = 1;
  for(; b; b >>= 1, a = 1ll* a* a % MOD) if(b&1) res = 1ll * res * a % MOD;
  return res;
}
```

5.16 sieve.cpp

```
namespace Sieve {
    vector<int> primes;
    vector<int> mn_factor;
    void get_primes(int N) {
        mn_factor.resize(N+1);
        for (int i = 2; i <= N; ++i) {
            if (mn_factor[i]==0) {
                primes.push_back(i);
                mn_factor[i]=i;
            }
            for (auto p : primes){
                if ((long long)i * p > N) break;
                mn_factor[i * p] = p;
                if (i % p == 0) break;
            }
        }
    }
    bool is_prime(int n) {
        return mn_factor[n]==0;
    }
    vector<pair<int, int>> factor(int n) {
        vector<pair<int, int>> factors;
        while (n > 1) {
            int fac=mn_factor[n], cnt=0;
            while (n%fac==0) {
                cnt++;
                n/=fac;
            }
            factors.emplace_back(fac, cnt);
        }
        return factors;
    };
    vector<int> phi;
    void get_euler(int n) {
        phi.resize(n+1);
        phi[1] = 1;
        for (int i = 2; i <= n; i++) {
            if (phi[i]) continue;
            for (int j = i; j <= n; j += i) {
                if (!phi[j]) phi[j] = j;
                phi[j] = phi[j] / i * (i - 1);
            }
        }
    }
}
using namespace Sieve;
```

5.17 simplex.h

```
/**
 * Author: Stanford
 * Source: Stanford Notebook
 * License: MIT
 * Description: Solves a general linear maximization problem: maximize $c^T x$
      subject to $Ax \le b$, $x \ge 0$.
 * Returns -inf if there is no solution, inf if there are arbitrarily good solutions,
       or the maximum value of $c^T x$ otherwise.
 * The input vector is set to an optimal $x$ (or in the unbounded case, an arbitrary
      solution fulfilling the constraints).
 * Numerical stability is not guaranteed. For better performance, define variables
      such that $x = 0$ is viable.
 * Usage:
 * vvd A = {{1,-1}, {-1,1}, {-1,-2}};
 * vd b = {1,1,-4}, c = {-1,-1}, x;
 * T val = LPSolver(A, b, c).solve(x);
 * Time: O(NM * \#pivots), where a pivot may be e.g. an edge relaxation. O(2^n) in
      the general case.
 * Status: seems to work?
 */
typedef double T; // long double, Rational, double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;

const T eps = 1e-8, inf = 1/.0;
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j

struct LPSolver {
        int m, n;
        vi N, B;
        vvd D;

        LPSolver(const vvd& A, const vd& b, const vd& c) :
                m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
                        FOR(i,0,m) FOR(j,0,n) D[i][j] = A[i][j];
                        FOR(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i];}
                        FOR(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
                        N[n] = -1; D[m+1][n] = 1;
                }

        void pivot(int r, int s) {
                T *a = D[r].data(), inv = 1 / a[s];
                FOR(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
                        T *b = D[i].data(), inv2 = b[s] * inv;
                        FOR(j,0,n+2) b[j] -= a[j] * inv2;
                        b[s] = a[s] * inv2;
                }
                FOR(j,0,n+2) if (j != s) D[r][j] *= inv;
                FOR(i,0,m+2) if (i != r) D[i][s] *= -inv;
                D[r][s] = inv;
                swap(B[r], N[s]);
        }

        bool simplex(int phase) {
                int x = m + phase - 1;
                for (;;) {
                        int s = -1;
                        FOR(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
                        if (D[x][s] >= -eps) return true;
                        int r = -1;
                        FOR(i,0,m) {
                                if (D[i][s] <= eps) continue;
                                if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                                             < MP(D[r][n+1] / D[r][s], B[r])) r = i;
                        }
                        if (r == -1) return false;
                        pivot(r, s);
                }
        }
        T solve(vd &x) {
                int r = 0;
                FOR(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
                if (D[r][n+1] < -eps) {
                        pivot(r, n);
                        if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
                        FOR(i,0,m) if (B[i] == -1) {
                                int s = 0;
                                FOR(j,1,n+1) ltj(D[i]);
                                pivot(i, s);
                        }
                }
                bool ok = simplex(1); x = vd(n);
                FOR(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
                return ok ? D[m][n+1] : inf;
        }
};
```

6 Misc

6.1 Mo’s_algorithm.cpp

```
// Mo's algorithm, solve m offline queries on array of length n in O(n sqrt(m))
struct MO {
    int n, m=0;
    struct node {
        int l, r, id;
    };
    vector<node> query;
    MO(int _n) : n(_n) {}
    void add_query(int l, int r) {
        query.push_back({l, r, m++});
    }
    template<typename F>
    vector<int> solve(F&& move) {
        const int BLOCK_SIZE = (n<=m ? ceil(sqrt(n)) : n/ceil(sqrt(m)));
        sort(query.begin(), query.end(), [&](const node& lhs, const node& rhs) {
            if (lhs.l / BLOCK_SIZE != rhs.l / BLOCK_SIZE) return lhs.l < rhs.l;
            return ((lhs.l / BLOCK_SIZE) & 1) ? lhs.r < rhs.r : lhs.r > rhs.r;
        });
        vector<int> ans(m);
        int l=0, r=-1, cur=0;
        for (const auto& [ql, qr, id] : query) {
            while (l > ql) move(--l, 1, cur);
            while (r < qr) move(++r, 1, cur);
            while (l < ql) move(l++, -1, cur);
            while (r > qr) move(r--, -1, cur);
            ans[id]=cur;
        }
        return ans;
    }
};
// example: find the most occurrence in ranges
int main() {
    int n, q;
    MO mo(n);
    vector<int> a(n), counter(n+1), freq(3e5+1);
    auto ans=mo.solve([&](int i, int dir, int& cur) {
        int val=a[i];
        int c=freq[val];
        counter[c]--;
        if (dir==1) {
            freq[val]++;
            counter[freq[val]]++;
            cur=max(cur, freq[val]);
        } else {
            freq[val]--;
            counter[freq[val]]++;
            if (counter[cur]==0) cur--;
        }
    });
}
```

7 String

7.1 ac-automaton.cpp

```
/** Modified from:
 * https://github.com/kth-competitive-programming/kactl/blob/master/content/strings/
      AhoCorasick.h
 * Try to handdle duplicated patterns beforehand, otherwise change 'end' to
 * vector; empty patterns are not allowed. Time: construction takes $O(26N)$,
 * where $N =$ sum of length of patterns. find(x) is $O(N)$, where N = length of
 * x. findAll is $O(N+M)$ where M is number of occurrence of all pattern (up to N*
      sqrt(N)) */
struct AhoCorasick {
    enum { alpha = 26, first = 'a' }; // change this!
    struct Node {
        // back: failure link, points to longest suffix that is in the trie.
        // end: longest pattern that ends here, is -1 if no patten ends here.
        // nmatches: number of (patterns that is a suffix of current
        // node)/(duplicated patterns), depends on needs.
        // output: output link, points to the longest pattern that is a suffix
        // of current node
        int back, end = -1, nmatches = 0, output = -1;
        array<int, alpha> next;
        Node(int v = -1) { fill(next.begin(), next.end(), v); }
    };
    vector<Node> N;
    AhoCorasick() : N(1) {}
    void insert(string &s, int j) { // j: id of string s
        assert(!s.empty());
        int n = 0;
        for (char c : s) {
            int &m = N[n].next[c - first];
            if (m == -1) {
                m = (int)N.size();
                N.emplace_back();
            }
            n = m;
        }
        N[n].end = j;
        N[n].nmatches++;
    }
    void build() {
        N[0].back = (int)N.size();
        N.emplace_back(0);
        queue<int> q;
        q.push(0);
        while (!q.empty()) {
            int n = q.front();
            q.pop();
            for (int i = 0; i < alpha; i++) {
                int pnx = N[N[n].back].next[i];
                auto &nxt = N[N[n].next[i]];
                if (N[n].next[i] == -1) N[n].next[i] = pnx;
                else {
                    nxt.back = pnx;
                    // if prev is an end node, then set output to prev node,
                    // otherwise set to output link of prev node
                    nxt.output = N[pnx].end == -1 ? N[pnx].output : pnx;
                    // if we don't want to distinguish info of patterns that is
                    // a suffix of current node, we can add info to the next
                    // node like this: nxt.nmatches+=N[pnx].nmatches;
                    q.push(N[n].next[i]);
                }
            }
        }
    }
    // for each position, finds the longest pattern that ends here
    vector<int> find(const string &text) {
        int len = (int)text.size();
        vector<int> res(len);
        int n = 0;
        for (int i = 0; i < len; i++) {
            n = N[n].next[text[i] - first];
            res[i] = N[n].end;
        }
        return res;
    }
    // for each position, finds the all that ends here
    vector<vector<int>> find_all(const string &text) {
        int len = (int)text.size();
        vector<vector<int>> res(len);
        int n = 0;
        for (int i = 0; i < len; i++) {
            n = N[n].next[text[i] - first];
            res[i].push_back(N[n].end);
            for (int ind = N[n].output; ind != -1; ind = N[ind].output) {
                assert(N[ind].end != -1);
                res[i].push_back(N[ind].end);
            }
        }
        return res;
    }
};
```

7.2 kmp.cpp

```
vector<int> prefix_function(const string& s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j]) j = pi[j - 1];
        if (s[i] == s[j]) j++;
        pi[i] = j;
    }
    return pi;
}
```

7.3 manacher.cpp

```
vector<int> manacher(const string& ss){
    string s;
    for(auto ch:ss) s+="#",s+=ch;
    s+="#";
    int n=(int)s.size();
    vector<int> d1(n);
    for (int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 1 : min(d1[l + r - i], r - i);
        while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) k++;
        d1[i] = k--;
        if (i + k > r) l = i - k, r = i + k;
    }
    return d1;
}
```

7.4 polyhash.cpp

```
#include<bits/stdc++.h>
using ll = long long;
struct PolyHash {
    static constexpr int mod = (int)1e9 + 123;
    static vector<int> pow;
    static constexpr int base = 233;
    vector<int> pref;
    PolyHash(const string &s) : pref(s.size() + 1) {
        assert(base < mod);
        int n = (int)s.size();
        while ((int)pow.size() <= n) {
            pow.push_back((ll)pow.back() * base % mod);
        }
        for (int i = 0; i < n; i++) {
            pref[i + 1] = ((ll)pref[i] * base + s[i]) % mod;
        }
    }
    int get_hash() {
        return pref.back();
    }
    int substr(int pos, int len) {
        return (pref[pos + len] - (ll)pref[pos] * pow[len] % mod + mod) % mod;
    }
};
vector<int> PolyHash::pow{1};
```

7.5 suffix array.cpp

```
#include<bits/stdc++.h>
//O(n log(n)),actually calculates cyclic shifts
vector<int> suffix_array(string s) {
    s+="#";
    int n = (int)s.size(), N = n + 256;
    vector<int> sa(n), ra(n);
    for(int i = 0; i < n; i++) sa[i] = i, ra[i] = s[i];
    for(int k = 0; k < n; k ? k *= 2 : k++) {
        vector<int> nsa(sa), nra(n), cnt(N);
        for(int i = 0; i < n; i++) nsa[i] = (nsa[i] - k + n) % n;
        for(int i = 0; i < n; i++) cnt[ra[i]]++;
        for(int i = 1; i < N; i++) cnt[i] += cnt[i - 1];
        for(int i = n - 1; i >= 0; i--) sa[--cnt[ra[nsa[i]]]] = nsa[i];

        int r = 0;
        for(int i = 1; i < n; i++) {
            if(ra[sa[i]] != ra[sa[i - 1]]) r++;
            else if(ra[(sa[i] + k) % n] != ra[(sa[i - 1] + k) % n]) r++;
            nra[sa[i]] = r;
        }
        ra = nra;
    }
    sa.erase(sa.begin());
    return sa;
}
vector<int> build_lcp(const string& s, const vector<int>& sa) { // lcp of sa[i] and sa[i-1]
    int n = (int)s.size();
    vector<int> pos(n);
    for (int i = 0; i < n; i++) pos[sa[i]] = i;
    vector<int> lcp(n);
    for (int i = 0, k = 0; i < n; i++) {
        if (pos[i] == 0) continue;
        if (k) k--;
        while (s[i+k] == s[sa[pos[i]-1]+k]) k++;
        lcp[pos[i]] = k;
    }
    return lcp;
}
```

7.6 suffix automaton.cpp

```
// source: https://cp-algorithms.com/string/suffix-automaton.html
struct SAM {
    struct state {
        int len = 0, link = -1;
        unordered_map<char, int> next;
    };
    int last = 0; // the index of the equivalence class of the whole string
    vector<state> st;
    void extend(char c) {
        int cur = (int)st.size();
        st.emplace_back();
        st[cur].len = st[last].len + 1;
        int p = last;
        while (p != -1 && !st[p].next.count(c)) {
            st[p].next[c] = cur;
            p = st[p].link;
        }
        if (p == -1) st[cur].link = 0;
        else {
            int q = st[p].next[c];
            if (st[p].len + 1 == st[q].len) {
                st[cur].link = q;
            } else {
                int clone = (int)st.size();
                st.push_back(st[q]);
                st[clone].len = st[p].len + 1;
                while (p != -1 && st[p].next[c] == q) {
                    st[p].next[c] = clone;
                    p = st[p].link;
                }
                st[q].link = st[cur].link = clone;
            }
        }
        last = cur;
    }
    SAM() { st.emplace_back(); }
    SAM(const string &s) : SAM() {
        for (auto c : s)
            extend(c);
    }
};
```

7.7 suffix_array_linear.cpp

```
//O(n)
vector<int> suffix_array(const string& s, int char_bound) {
    int n=s.size();
    vector<int> a(n);
    if (n == 0) return a;
    if (char_bound != -1) {
        vector<int> aux(char_bound, 0);
        for (int i = 0; i < n; i++) aux[s[i]]++;
        int sum = 0;
        for (int i = 0; i < char_bound; i++) {
            int add = aux[i];
            aux[i] = sum;
            sum += add;
        }
        for (int i = 0; i < n; i++) {
            a[aux[s[i]]++] = i;
        }
    } else {
        iota(a.begin(), a.end(), 0);
        sort(a.begin(), a.end(), [&s](int i, int j) { return s[i] < s[j]; });
    }
    vector<int> sorted_by_second(n);
    vector<int> ptr_group(n);
    vector<int> new_group(n);
    vector<int> group(n);
    group[a[0]] = 0;
    for (int i = 1; i < n; i++) {
        group[a[i]] = group[a[i - 1]] + (!(s[a[i]] == s[a[i - 1]]));
    }
    int cnt = group[a[n - 1]] + 1;
    int step = 1;
    while (cnt < n) {
        int at = 0;
        for (int i = n - step; i < n; i++) {
            sorted_by_second[at++] = i;
        }
        for (int i = 0; i < n; i++) {
            if (a[i] - step >= 0) {
                sorted_by_second[at++] = a[i] - step;
            }
        }
        for (int i = n - 1; i >= 0; i--) {
            ptr_group[group[a[i]]] = i;
        }
        for (int i = 0; i < n; i++) {
            int x = sorted_by_second[i];
            a[ptr_group[group[x]]++] = x;
        }
        new_group[a[0]] = 0;
        for (int i = 1; i < n; i++) {
            if (group[a[i]] != group[a[i - 1]]) {
                new_group[a[i]] = new_group[a[i - 1]] + 1;
            } else {
                int pre = (a[i - 1] + step >= n ? -1 : group[a[i - 1] + step]);
                int cur = (a[i] + step >= n ? -1 : group[a[i] + step]);
                new_group[a[i]] = new_group[a[i - 1]] + (pre != cur);
            }
        }
        swap(group, new_group);
        cnt = group[a[n - 1]] + 1;
        step <<= 1;
    }
    return a;
}
```

7.8 trie.cpp

```
#include <bits/stdc++.h>
using namespace  std;
template<typename T>
struct Trie {
    struct node {
        map<T, int> ch;
        bool is_leaf;
    };
    vector<node> t;
    Trie() { new_node(); }
    int new_node() {
        t.emplace_back();
        return (int)t.size()-1;
    }
    template<typename S> void insert(const S& s) {
        int p=0;
        for (int i=0; i<(int)s.size(); i++) {
            auto ch=s[i];
            if (!t[p].ch.count(ch)) {
                t[p].ch[ch]=new_node();
            }
            p=t[p].ch[ch];
        }
        t[p].is_leaf=true;
    }
    template<typename S> bool find(const S& s) {
        int p=0;
        for (auto ch : s) {
            if (!t[p].ch.count(ch)) return false;
            p=t[p].ch[ch];
        }
        return t[p].is_leaf;
    }
};
```

7.9 z-function.cpp

```
// In other words, z[i] is the length of the longest common prefix between s and the suffix of s starting at i.
vector<int> z_function(const string& s) {
    int n = (int)s.size();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r) z[i] = min(r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }
    return z;
}
```