Teamnote of 2mic1cup

swishy, lenhanbo, hungcubu

Last updated on October 11, 2025

Contents

1 Helpers

1.1 Stress Tester

Overview

Simple .bat file for stress testing.

Implementation

```
@echo off
g++ -std=c++20 -o solution test.cpp
g++ -std=c++20 -o brute brute.cpp
g++ -std=c++20 -o gen gen.cpp

for /l %%x in (1, 1, 1000) do (
    gen > input.in
    solution < input.in > output.out
    brute < input.in > output2.out
    fc output.out output2.out > nul

    if ERRORLEVEL 1 (
        echo INPUT
        type input.in
        echo.
        echo SOLUTION OUTOUT
        type output.out
        echo.
        echo CORRECT OUTPUT
        type output2.out
        echo.
    )
)
echo all tests passed
```

1.2 Random

Overview

Self explanatory.

Implementation

```
#define uid(a, b) uniform_int_distribution<long long>(a, b)(rng)
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
```

Usage

- $uid(a, b)$ returns random integer between $[a, b]$

1.3 Hungbucu's amazing crazy diabolical template

Overview

Bro only contribution

Implementation

```
#include         <bits/stdc++.h>
#include  <ext/pb_ds/assoc_container.hpp>

#define            TASK   "test"
#define              ll   long long
#define             ull   unsigned ll
#define              db   long double
#define             pLL   pair<ll, ll>
#define             pLI   pair<ll, int>
#define             pIL   pair<int, ll>
#define             pII   pair<int, int>
#define             vec   vector
#define              vL   vec<ll>
#define             vvL   vec<vL>
#define              vI   vec<int>
#define             vvI   vec<vI>
#define            vvvI   vec<vvI>
#define           vvvvI   vec<vvvI>
#define              vD   vec<db>
#define             vvD   vec<vD>
#define             vLL   vec<pLL>
#define             vLI   vec<pLI>
#define             vIL   vec<pIL>
#define             vII   vec<pII>
#define            vvII   vec<vII>
#define              vS   vec<string>
#define             vvS   vec<vS>
#define              vB   vec<bool>
#define             vvB   vec<vB>
#define            umap   unordered_map
#define          gphash   gp_hash_table
#define            mset   multiset
#define            pque   priority_queue
#define          all(a)   a.begin(), a.end()
#define         rall(a)   a.rbegin(), a.rend()
#define       stt(a, n)   a.begin(), a.begin() + n
#define       stf(a, n)   a.begin() + n, a.end()
#define              eb   emplace_back
#define              pb   push_back
#define              pf   push_front
#define            popb   pop_back
#define            popf   pop_front
#define             ins   insert
#define             asg   assign
#define             rev   reverse
#define              fi   first
#define              se   second
#define              th   third
#define              ub   upper_bound
#define              lb   lower_bound
#define             ite   iterator
#define           fs(n)   fixed << setprecision(n)

using         namespace   std;
using         namespace   __gnu_pbds;

const ll          llINF = 1e18;
const int        intINF = 1e9;
const ll            MOD = 1e9 + 7;

template<   class T   >
using       ordered_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
#define            oset   ordered_set

template<     class A,
              class B,
              class C >   struct triple {
    A fi; B se; C th;

    triple() {}
    triple(A a,B b,C c) : fi(a), se(b), th(c) {}
};
#define            tIII   triple<int, int, int>
#define            tLLL   triple<ll, ll, ll>
#define            vIII   vec<tIII>
#define           vvIII   vec<vIII>
#define            vLLL   vec<tLLL>

mt19937 rd(chrono::high_resolution_clock::now().time_since_epoch().count());
ll     Rand(ll L, ll R)   { return uniform_int_distribution<ll>(L, R)(rd); }

inline int     read() {
    char c;    while (c = getchar(), c < '0' || c > '9'); int n = c - '0';
    while           (c = getchar(), c >= '0' && c <= '9') n = 10 * n + c - '0';
    return         n; }

vI          prime, lpf;
void  primeSieve(int n)   { prime.asg(1, 2); lpf.asg(n + 1, 2); lpf[0] = lpf[1] = 1;
                            for (int i = 3; i <= n; i += 2) { if (lpf[i] == 2) { lpf[i] = i; prime.pb(i); }
                            for (int j = 0; j < prime.size() && i * prime[j] <= n && prime[j] <= lpf[i]; ++ j) lpf[i * prime[j]] = prime[j];
                          } }
vvI               dvs;
void    dvsSieve(int n)   { dvs.asg(n + 1, vI());
                            for (int i = 1; i <= n; ++ i) {
                            for (int j = i; j <= n; j += i)
                            dvs[j].pb(i);
                          } }

template<   class T   >   bool maximize(T &a, T b) { if (b > a) return a = b, 1; return 0; }
template<   class T   >   bool minimize(T &a, T b) { if (b < a) return a = b, 1; return 0; }

ll      gcd(ll a, ll b)   { return b ? gcd(b, a % b) : a; }
ll      lcm(ll a, ll b)   { return a / gcd(a, b) * b; }

ll  fastPow(ll n, ll p,
            ll m = MOD)   { ll r = 1; for (n %= m; p; p >>= 1) { if (p & 1) (r *= n) %= m; (n *= n) %= m; } return r; }
ll         invMod(ll n,
            ll m = MOD)   { return fastPow(n, m - 2, m); }

ll          mask(int i)   { return i < 0 ? 0 : 1LL << i; }
bool   bit(ll n, int i)   { return n >> i & 1LL; }
#define        popcount   __builtin_popcountll
#define             clz   __builtin_clzll
#define             ctz   __builtin_ctzll
```

2 Data Structure

2.1 Iterative Segment Tree

Overview

For-loop implementation of segment tree, faster than recursive. Note: Operation that depends on ordering is not supported (For example: Minimum prefix sum)

Time complexity: $\mathcal{O}(n)$ **for constructor,** $\mathcal{O}(\log n)$ **for query**

Implementation

```cpp
template<typename T>
struct SegmentTreeFast{
    vector<T> a;
    T defv;
    int n;

    SegmentTreeFast(int n, T defv) : n(n), defv(defv){
        a = vector<T>(2 * n, defv);
    }

    T cmb(T a, T b){ //change if needed
        return a + b;
    }

    void build(){ //array is at i + n index
        for (int i = n - 1; i > 0; --i)
            a[i] = cmb(a[i << 1], a[i << 1 | 1]);
    }

    void update(int i, T v){
        for (a[i += n] = v; i > 1; i >>= 1)
            a[i >> 1] = cmb(a[i], a[i ^ 1]);
    }

    T get(int l, int r){
        r++;
        T res = defv;
        for (l += n, r += n; l < r; l >>= 1, r >>= 1){
            if (l&1) res = cmb(res, a[l++]);
            if (r&1) res = cmb(res, a[--r]);
        }

        return res;
    }
};
```

2.2 Lazy Segment Tree

Overview

Segment tree that supports ranged update.

Time complexity: $\mathcal{O}(n)$ **for constructor,** $\mathcal{O}(\log n)$ **for query**

Implementation

```cpp
template<typename T>
class SegmentTreeLazy{
public:
    vector<T> st, lazy;
    T defv;
    int n;

    SegmentTreeLazy(int n, T defv) : n(n), defv(defv){
        st = vector<T>(n * 4, defv);
        lazy = vector<T>(n * 4, defv);
    }

    void update(int l, int r, T v){
        _update(0, n - 1, 0, l, r, v);
    }

    T get(int l, int r){
        return _get(0, n - 1, l, r, 0);
    }

private:
    T cmb(T l, T r){
        return l + r;
    }

    void push(int i, int l, int r){
        int mid = (l + r) / 2;
        lazy[i * 2 + 1] += lazy[i];
        lazy[i * 2 + 2] += lazy[i];

        st[i * 2 + 1] += (mid - l + 1) * lazy[i];
        st[i * 2 + 2] += (r - mid) * lazy[i];

        lazy[i] = 0;
    }

    void _update(int l, int r, int crr, int ql, int qr, T v){
        if (qr < l || ql > r)
            return;

        if (l >= ql && r <= qr){
            st[crr] += (r - l + 1) * v;
            lazy[crr] += v;
            return;
        }

        push(crr, l, r);
        int mid = (l + r) / 2;
        _update(l, mid, crr * 2 + 1, ql, qr, v);
        _update(mid + 1, r, crr * 2 + 2, ql, qr, v);

        st[crr] = cmb(st[crr * 2 + 1], st[crr * 2 + 2]);
    }

    T _get(int l, int r, int ql, int qr, int crr){
        if (qr < l || ql > r)
            return defv;
        if (l >= ql && r <= qr)
            return st[crr];

        push(crr, l, r);
        int mid = (l + r) / 2;
        return cmb(_get(l, mid, ql, qr, crr * 2 + 1), _get(mid + 1, r, ql, qr, crr * 2 + 2));
    }
};
```

2.3 Sparse Table

Overview

Uses binary lifting for efficient queries, offline only.

Time complexity: $\mathcal{O}(n \log n)$ **for constructor,** $\mathcal{O}(1)$ **for query**

Implementation

```cpp
template <typename T, class Combine = function<T(const T &, const T &)>>
struct SparseTable{
    vector<vector<T>> f;
    vector<int> lg;
    Combine cmb;
    int n;

    SparseTable(vector<T> &init, const Combine &cmb) : n(init.size()), cmb(cmb){
        lg = vector<int>(n + 1, 0);
        for (int i = 2; i <= n; i++)
            lg[i] = lg[i / 2] + 1;
        for (int i = 0; i < n; i++){
            f.push_back(vector<int>(lg[n] + 1, -1));
            f[i][0] = init[i];
        }
        for (int j = 1; (1 << j) <= n; j++){
            for (int i = 0; (i + (1 << j) - 1) < n; i++)
                f[i][j] = cmb(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
        }
    }

    T get(int l, int r){
        int k = lg[r - l + 1];
        return cmb(f[l][k], f[r - (1 << k) + 1][k]);
    }
};
```

Usage

- Init minimum range query and uses integer type

```cpp
SparseTable<int> rmq(a, [](int a, int b){
    return min(a, b);
});
```

2.4 Implicit Treap

Overview

Implicit treap implementation with range add update and range sum query. push() and upd() functions should be changed accordingly like lazy segment tree.

Time complexity: $\mathcal{O}(\log n)$ **on average for all operations, large constant!!**

Implementation

```cpp
typedef node* pnode;
struct ImplicitTreap{
public:
    pnode root;
    ImplicitTreap(){
        root = new node(-1, 0);
    }
    void insert(int i, ll val){
        pnode t1, t2;
        split(root, i + 1, 0, t1, t2);
        merge(t1, t1, new node(val));
        merge(root, t1, t2);
    }
    void erase(int i){
        _erase(root, i + 1, 0);
    }
    ll query(int l, int r){
        pnode t1, t2, t3;
        split(root, r + 2, 0, t2, t3);
        split(t2, l + 1, 0, t1, t2);

        ll res = t2->sum;
        merge(root, t1, t2);
        merge(root, root, t3);

        return res;
    }
    void update(int l, int r, ll val){
        pnode t1, t2, t3;
        split(root, r + 2, 0, t2, t3);
        split(t2, l + 1, 0, t1, t2);

        t2->add += val;
        merge(root, t1, t2);
        merge(root, root, t3);
    }
    void split(pnode t, int key, int add, pnode &l, pnode &r){
        if (!t){
            l = r = nullptr;
            return;
        }
        push(t);
        int impl_key = add + _cnt(t->l);
        if (key <= impl_key)
            split(t->l, key, add, l, t->l), r = t;
        else
            split(t->r, key, add + _cnt(t->l) + 1, t->r, r), l = t;
        upd(t);
    }

    void merge(pnode &t, pnode l, pnode r){
        push(l); push(r);
        if (!l || !r)
            t = l? l : r;
        else if (l->prior > r->prior)
            merge(r->l, l, r->l), t = r;
        else
            merge(l->r, l->r, r), t = l;
        upd(t);
    }
private:
    void _erase(pnode &t, int key, int add){
        push(t);
        int impl_key = add + _cnt(t->l);
        if (impl_key == key){
            pnode it = t;
            merge(t, t->l, t->r);
            delete it;
        }
        else if (key < impl_key)
            _erase(t->l, key, add);
        else
            _erase(t->r, key, add + _cnt(t->l) + 1);
        upd(t);
    }
    void push(pnode t){
        if (!t) return;
        t->sum += t->add * (ll)_cnt(t);
        t->val += t->add;
        if (t->l) t->l->add += t->add;
        if (t->r) t->r->add += t->add;

        t->add = 0;
    }
    int _cnt(pnode t){
        if (!t) return 0;
        return t->cnt;
    }
    ll _sum(pnode t){
        if (!t) return 0;
        push(t);
        return t->sum;
    }
    void upd(pnode t){
        if (!t) return;
        t->sum = t->val + _sum(t->l) + _sum(t->r);
        t->cnt = _cnt(t->l) + _cnt(t->r) + 1;
    }
};
```

2.5 Dynamic Segment Tree

Overview

Range queries and updates on larger range ($1 \leq l \leq r \leq 10^9$)

Time complexity: $\mathcal{O}(\log M)$ for every operations, where M is max range

Implementation

```cpp
struct Node{
    ll sum, tl, tr;
    Node *l = nullptr, *r = nullptr;

    Node (ll _tl, ll _tr){
        tl = _tl;
        tr = _tr;
        sum = 0;
    }

    void extend(){
        if (tl == tr) return;
        ll mid = (tl + tr) / 2;

        if (!l)
            l = new Node(tl, mid);
        if (!r)
            r = new Node(mid + 1, tr);
    }
};

class funkysegtree{
    void _upd(Node *node, ll x, ll val){
        node->sum += val;
        if (node->tl > x || node->tr < x)
            return;
        if (node->tl == node->tr)
            return;

        ll mid = (node->tl + node->tr) / 2;
        node->extend();

        if (x <= mid)
            _upd(node->l, x, val);
        else
            _upd(node->r, x, val);
    }

    ll _get(Node *node, ll ql, ll qr){
        if (qr < node->tl || ql > node->tr)
            return 0;

        else if (ql <= node->tl && qr >= node->tr)
            return node->sum;

        ll mid = (node->tl + node->tr) / 2;
        node->extend();

        if (ql > mid)
            return _get(node->r, ql, qr);
        else if (qr <= mid)
            return _get(node->l, ql, qr);
        else
            return _get(node->l, ql, mid) + _get(node->r, mid + 1, qr);
    }

public:
    Node *root = nullptr;
    ll _size;

    funkysegtree(ll __size){
        root = new Node(0, __size);
        _size = __size;
    };

    void upd(ll x, ll val){
        _upd(root, x, val);
    }

    ll get(ll l, ll r){
        return _get(root, l, r);
    }
};
```

2.6 Persistent Segment Tree

Overview

Preserving history for every segment tree updates.

Time complexity: $\mathcal{O}(\log N)$ for every operations

Implementation

```cpp
struct Vertex {
    Vertex *l, *r;
    int sum;

    Vertex(int val) : l(nullptr), r(nullptr), sum(val) {}
    Vertex(Vertex *l, Vertex *r) : l(l), r(r), sum(0) {
```

```cpp
        if (l) sum += l->sum;
        if (r) sum += r->sum;
    }
};

Vertex* build(ll a[], int tl, int tr) {
    if (tl == tr)
        return new Vertex(a[tl]);
    int tm = (tl + tr) / 2;
    return new Vertex(build(a, tl, tm), build(a, tm+1, tr));
}

int get_sum(Vertex* v, int tl, int tr, int l, int r) {
    if (l > r)
        return 0;
    if (l == tl && tr == r)
        return v->sum;
    int tm = (tl + tr) / 2;
    return get_sum(v->l, tl, tm, l, min(r, tm))
         + get_sum(v->r, tm+1, tr, max(l, tm+1), r);
}

Vertex* update(Vertex* v, int tl, int tr, int pos, int new_val) {
    if (tl == tr)
        return new Vertex(new_val);
    int tm = (tl + tr) / 2;
    if (pos <= tm)
        return new Vertex(update(v->l, tl, tm, pos, new_val), v->r);
    else
        return new Vertex(v->l, update(v->r, tm+1, tr, pos, new_val));
}
```

Usage

- Init and update segment tree with n nodes, each function returns a pointer, save if needed for later.

```cpp
vector<Vertex*> roots;
roots.push_back(build(a, 0, n - 1)); //init
(...)
roots.push_back(update(roots.back(), 0, n - 1, x, 1)); //update at the last moment
(...)
roots.push_back(update(roots[a], 0, n - 1, x, 1)); //update at some specific moment
```

- Query the segment tree at a specific moment.

```cpp
ll res = get_sum(roots[x], 0, n - 1, l, r);
```

2.7 2D Fenwick Tree

Overview

Query and update on a 2D array.

Time complexity: $\mathcal{O}(\log^2 n)$ for every operations

Implementation

```cpp
ll bit[1001][1001];
ll n, m;

void update(ll x, ll y, ll val){
        for (; y <= n; y += (y & (-y))){
                for (ll i = x; i <= m; i += (i & (-i)))
                        bit[y][i] += val;
        }
}

ll query(ll x, ll y){
        ll res = 0;
        for (ll i = y; i; i -= (i & (-i)))
                for (ll j = x; j; j -= (j & (-j)))
                        res += bit[i][j];
        return res;
}

ll query(ll x1, ll y1, ll x2, ll y2){
        ll res = query(x2, y2) - query(x1 - 1, y2) - query(x2, y1 - 1) + query(x1 - 1, y1 - 1);
        return res;
}
```

Usage

- query(x, y) returns sum of value from $(1, 1)$ to (x, y).
- query(x1, y1, x2, y2) returns sum of value from $(x1, y1)$ to $(x2, y2)$.

2.8 Disjoint Set Union

Overview

Union disjoint set lol.

Time complexity: $\mathcal{O}(\alpha(n))$

Implementation

```cpp
struct DissjointSet{
    vector<int> p;
    int cnt = 0;

    DissjointSet(){}
    DissjointSet(int n){
        cnt = n;
        p = vector<int>(n, -1);
    }

    int find(int n){
        return p[n] < 0 ? n : p[n] = find(p[n]);
    }

    void merge(int u, int v){
        if ((u = find(u)) == (v = find(v)))
            return;

        cnt--;
        if (p[v] < p[u])
            swap(u, v);

        p[u] += p[v];
        p[v] = u;
    }
};
```

2.9 Line Container

Overview

Add lines of the form $y = kx + m$, and query maximum value at point x.

Time complexity: $\mathcal{O}(\log n)$

Implementation

```cpp
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)
        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;
        }
    //add line y = kx + m
        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;
        }
};
```

2.10 Lichao Tree

Overview

Add lines of the form $y = ax + b$, and query maximum value at point x, segment tree implementation.

Time complexity: $\mathcal{O}(\log n)$

Implementation

```
struct LichaoTree{
    struct Line{
        ll a, b;
        Line() : a(0), b(-inf) {}
        Line(ll a, ll b): a(a), b(b) {}
        ll get(ll x){
            return a * x + b;
        }
    };
public:
    vector<Line> st;
    int n;
    LichaoTree(int n) : n(n){
        st.resize(4 * n);
    }
    void add_line(Line line, int indx = 1, int l = 0, int r = -1){
        if (r == -1) r = n;
        int m = (l + r) / 2;
        bool left = line.get(l) > st[indx].get(l);
        bool mid = line.get(m) > st[indx].get(m);

        if (mid)
            swap(line, st[indx]);
        if (r - l == 1) return;
        else if (left != mid)
            add_line(line, 2 * indx, l, m);
        else
            add_line(line, 2 * indx + 1, m, r);
    }
    ll query(ll x, int indx = 1, int l = 0, int r = -1){
        if (r == -1) r = n;
        if (r - l == 1) return st[indx].get(x);
        int mid = (l + r) / 2;
        if (x < mid)
            return max(st[indx].get(x), query(x, 2 * indx, l, mid));
        else
            return max(st[indx].get(x), query(x, 2 * indx + 1, mid, r));
    }
};
```

2.11 Ordered Set

Overview

A set that supports finding k-th maximum value, or getting the order of an element.

Time complexity: $\mathcal{O}(\log n)$, large constant

Implementation

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;
template<class T> using ordset = tree<T, null_type, less<T>, rb_tree_tag,
↪  tree_order_statistics_node_update>;
```

Usage

- Uses just like a normal set, but with some added functions.

```
ordset<int> s;
s.insert(1);
s.insert(2);
s.insert(4);
s.find_by_order(0) //Returns 1
s.order_of_key(4) //Returns 2
```

2.12 Minimum Stack/Deque

Overview

Maintains minimum value in a stack/deque.

Time complexity: $\mathcal{O}(\alpha(n))$, large constant

Implementation

```
struct minstack {
        stack<pair<int, int>> st;
        int getmin() {return st.top().second;}
        bool empty() {return st.empty();}
        int size() {return st.size();}
        void push(int x) {
                int mn = x;
                if (!empty()) mn = min(mn, getmin());
                st.push({x, mn});
        }
        void pop() {st.pop();}
        int top() {return st.top().first;}
        void swap(minstack &x) {st.swap(x.st);}
};

struct mindeque {
        minstack l, r, t;
        void rebalance() {
                bool f = false;
                if (r.empty()) {f = true; l.swap(r);}
                int sz = r.size() / 2;
                while (sz--) {t.push(r.top()); r.pop();}
                while (!r.empty()) {l.push(r.top()); r.pop();}
                while (!t.empty()) {r.push(t.top()); t.pop();}
                if (f) l.swap(r);
        }
        int getmin() {
                if (l.empty()) return r.getmin();
                if (r.empty()) return l.getmin();
                return min(l.getmin(), r.getmin());
        }
        bool empty() {return l.empty() && r.empty();}
        int size() {return l.size() + r.size();}
        void push_front(int x) {l.push(x);}
        void push_back(int x) {r.push(x);}
        void pop_front() {if (l.empty()) rebalance(); l.pop();}
        void pop_back() {if (r.empty()) rebalance(); r.pop();}
        int front() {if (l.empty()) rebalance(); return l.top();}
        int back() {if (r.empty()) rebalance(); return r.top();}
        void swap(mindeque &x) {l.swap(x.l); r.swap(x.r);}
};
```

2.13 Dynamic Bitset

Overview

Bitset with varied length support. NOTE: This requires relatively new version of GCC, and it might be BUGGED using the shift operator.

Time complexity: $\mathcal{O}$(n / 32)

Implementation

```
#include <tr2/dynamic_bitset>
using namespace tr2;
```

Usage

- Init a dynamic bitset with length n.

```
dynamic_bitset<> bs;
bs.resize(n);
```

3 Graph

3.1 Graph

Overview

Helper class, some implementations below will use this.

Implementation

```
struct Graph{
    vector<vector<int>> edg;
    int n;

    Graph(int n) : n(n){
        edg = vector<vector<int>>(n, vector<int>());
    }
    void add(int u, int v){
```

```cpp
        edg[u].push_back(v);
    }
    void bi_add(int u, int v){
        edg[u].push_back(v);
        edg[v].push_back(u);
    }
    void clear(){
        for (int u = 0; u < n; u++)
            edg[u].clear();
    }
    void remove_dup(){
        for (int u = 0; u < n; u++){
            sort(edg[u].begin(), edg[u].end());
            edg[u].erase(unique(edg[u].begin(), edg[u].end()), edg[u].end());
        }
    }
};
```

3.2 Strongly Connected Components

Overview

Find strongly connected components, compress the graph if needed

Time complexity: $\mathcal{O}(N)$

Implementation

```cpp
struct StronglyConnected{
    Graph &G;
    vector<vector<int>> components;
    vector<int> low, num, new_num;
    vector<bool> deleted;
    stack<int> st;
    int indx, scc, n;

    StronglyConnected(Graph &G) : G(G), n(G.n){
        low = num = new_num = vector<int>(n, 0);
        indx = scc = 0;
        deleted = vector<bool>(n, 0);

        for (int i = 0; i < n; i++){
            if (!num[i])
                dfs(i);
        }
    }

    void dfs(int u){
        low[u] = num[u] = ++indx;
        st.push(u);

        for (int v : G.edg[u]){
            if (deleted[v]) continue;
            if (!num[v]){
                dfs(v);
                low[u] = min(low[u], low[v]);
            }
            else
                low[u] = min(low[u], num[v]);
        }

        if (low[u] == num[u]){
            int crr = -1;
            vector<int> cmp;

            while (crr != u){
                crr = st.top();
                cmp.push_back(crr);
                st.pop();

                new_num[crr] = scc;
                deleted[crr] = 1;
            }

            components.push_back(cmp);
            scc++;
        }
    }

    void compress(){
        Graph _G(scc);
        for (int u = 0; u < n; u++){
            for (int v : G.edg[u]){
                int _u = new_num[u], _v = new_num[v];
                if (_u != _v)
                    _G.add(_u, _v);
            }
        }
        G = _G;
    }
};
```

3.3 Bridges and Articulations

Overview

Find bridges and articulations!!

Time complexity: $\mathcal{O}(N)$

Implementation

```cpp
struct BridgeArt{
    Graph &G;
    vector<int> low, num, arts;
    vector<bool> isart;
    vector<pair<int, int>> bridges;
    int indx, n;

    BridgeArt(Graph &G) : G(G), n(G.n){
        indx = 0;
        low = num = vector<int>(n, 0);
        isart = vector<bool>(n, 0);

        for (int i = 0; i < n; i++){
            if (!num[i])
                dfs(i, i);
        }
        for (int i = 0; i < n; i++){
            if (isart[i])
                arts.push_back(i);
        }
    }

    void dfs(int u, int pre){
        low[u] = num[u] = ++indx;
        int cnt = 0;

        for (int v : G.edg[u]){
            if (v == pre) continue;
            if (!num[v]){
                dfs(v, u);
                low[u] = min(low[u], low[v]);
                cnt++;
                if (u == pre){
                    if (cnt > 1)
                        isart[u] = 1;
                }
                else{
                    if (num[u] <= low[v])
                        isart[u] = 1;
                }
                if (num[v] == low[v])
                    bridges.push_back({u, v});
            }
            else
                low[u] = min(low[u], num[v]);
        }
    }
};
```

3.4 Two SAT

Overview

Solve a system of boolean formula, where every clause has exactly two literals.

Time complexity: $\mathcal{O}(N + M)$, M can be a slowing factor

Implementation

```cpp
struct TwoSAT{
    vector<vector<int>> edg1, edg2;
    vector<int> scc, res;
    vector<bool> b;
    stack<int> topo;
    int n;

    TwoSAT(int n) : n(n){
        edg1 = edg2 = vector<vector<int>>(2 * n);
        scc = res = vector<int>(2 * n, 0);
        b = vector<bool>(2 * n, 0);
    }

    void dfs1(ll u){
        b[u] = 1;
        for (ll v : edg1[u]){
            if (!b[v])
                dfs1(v);
        }

        topo.push(u);
    }

    void dfs2(ll u, ll root){
        scc[u] = root;
        for (ll v : edg2[u]){
            if (scc[v] == -1)
                dfs2(v, root);
        }
    }

```

```
    bool solve(){
        for (int i = 0; i < 2 * n; i++){
            scc[i] = -1;
            if (!b[i])
                dfs1(i);
        }

        int j = 0;
        while (siz(topo)){
            ll u = topo.top();
            topo.pop();

            if (scc[u] == -1)
                dfs2(u, j++);
        }

        for (int i = 0; i < n; i++){
            if (scc[i * 2] == scc[i * 2 + 1])
                return 0;
            res[i] = scc[i * 2] > scc[i * 2 + 1];
        }

        return 1;
    }

    void add(int x, bool a, int y, bool b){
        int X = x * 2 + (a & 1), Y = y * 2 + (b & 1);
        int _X = x * 2 + 1 - (a & 1), _Y = y * 2 + 1 - (b & 1);

        edg1[_X].push_back(Y);
        edg1[_Y].push_back(X);
        edg2[Y].push_back(_X);
        edg2[X].push_back(_Y);
    }
};
```

❓ Usage

- The $add(x, a, y, b)$ function add the clause (x OR y), where a, b signify whether x or y is negated or not.
- The $solve()$ function returns 1 if there exist a valid assignment, and 0 otherwise. The valid assignment will then be stored in res.

3.5 MCMF

Overview

Find a maximum flow with minimum cost, SPFA implementation.

Time complexity: $\mathcal{O}(N^3)$ with a bullshit factor

Implementation

```
struct edge{
    int v;
    ll cost, capacity;
    edge* rv;
    edge(int v, ll cost, ll capacity) : v(v), cost(cost), capacity(capacity){}
};

struct MCMF{
    vector<vector<edge*>> edg;
    vector<pair<int, edge*>> par;
    vector<ll> dis;

    MCMF(int n){
        edg = vector<vector<edge*>>(n);
    }
    void add_edge(int u, int v, ll capacity, ll cost){
        edge* e = new edge(v, cost, capacity);
        edge* re = new edge(u, -cost, 0);

        e->rv = re;
        re->rv = e;

        edg[u].push_back(e);
        edg[v].push_back(re);
    }
    void spfa(int start){
        int n = edg.size();
        auto inq = vec(n, 0);
        dis = vec(n, inf);
        par = vector<pair<int, edge*>>(n, {-1, nullptr});

        queue<int> q;
        q.push(start);
        dis[start] = 0;

        while (q.size()){
            int u = q.front(); q.pop();
            inq[u] = 0;

            for (auto e : edg[u]){
                if (e->capacity > 0 && dis[e->v] > dis[u] + e->cost){
                    dis[e->v] = dis[u] + e->cost;
                    par[e->v] = {u, e};

                    if (!inq[e->v]){
                        inq[e->v] = 1;
                        q.push(e->v);
                    }
                }
            }
        }
    }
    pl get(int start, int end, ll max_flow = inf){
        ll flow = 0, cost = 0;
        while (flow < max_flow){
            spfa(start);
            if (dis[end] == inf) break;

            ll f = max_flow - flow;
            int u = end;

            while (u != start){
                f = min(f, par[u].y->capacity);
                u = par[u].x;
            }

            flow += f;
            cost += f * dis[end];

            u = end;
            while (u != start){
                par[u].y->capacity -= f;
                par[u].y->rv->capacity += f;
                u = par[u].x;
            }
        }
        if (flow == max_flow || max_flow == inf)
            return {flow, cost};
        else
            return {-1, -1};
    }
};
```

3.6 Maximum Flow (Dinic)

Overview

Maximum flow using Dinic's algorithm.

Time complexity: $\mathcal{O}(V^2E)$ for general graphs, but in practice $\approx \mathcal{O}(E^{1.5})$

Implementation

```
template<int V, class T=long long>
class max_flow {
        static const T INF = numeric_limits<T>::max();

        struct edge {
                int t, rev;
                T cap, f;
        };

public:
        vector<edge> adj[V];
        ll dist[V];
        int ptr[V];

        bool bfs(int s, int t) {
                memset(dist, -1, sizeof dist);
                dist[s] = 0;
                queue<int> q({ s });
                while (!q.empty() && dist[t] == -1) {
                        int n = q.front();
                        q.pop();
                        for (auto& e : adj[n]) {
                                if (dist[e.t] == -1 && e.cap != e.f) {
                                        dist[e.t] = dist[n] + 1;
                                        q.push(e.t);
                                }
                        }
                }
                return dist[t] != -1;
        }

        T augment(int n, T amt, int t) {
                if (n == t) return amt;
                for (; ptr[n] < adj[n].size(); ptr[n]++) {
                        edge& e = adj[n][ptr[n]];
                        if (dist[e.t] == dist[n] + 1 && e.cap != e.f) {
                                T flow = augment(e.t, min(amt, e.cap - e.f), t);
                                if (flow != 0) {
                                        e.f += flow;
                                        adj[e.t][e.rev].f -= flow;
                                        return flow;
                                }
                        }
                }
                return 0;
        }

```

```cpp
        void add(int u, int v, T cap=1, T rcap=0) {
                adj[u].push_back({ v, (int) adj[v].size(), cap, 0 });
                adj[v].push_back({ u, (int) adj[u].size() - 1, rcap, 0 });
        }

        T calc(int s, int t) {
                T flow = 0;
                while (bfs(s, t)) {
                        memset(ptr, 0, sizeof ptr);
                        while (T df = augment(s, INF, t))
                                flow += df;
                }
                return flow;
        }

        void clear() {
                for (int n = 0; n < V; n++)
                        adj[n].clear();
        }
};
```

3.7 Maximum Matching (Hopcroft Karp)

Overview

Find maximum matching on bipartite graph.

Time complexity: $\mathcal{O}(m\sqrt{n})$ **worst case**

Implementation

```cpp
struct HopcroftKarp{
    vector<vector<int>> edg;
    vector<int> U, V;
    vector<int> pu, pv;
    vector<int> dist;

    //NOTE: This graph is 1-indexed!!!
    HopcroftKarp(int n, int m){
        edg = vector<vector<int>>(n + 1);
        for (int i = 0; i < n; i++)
            U.push_back(i + 1);
        for (int i = 0; i < m; i++)
            V.push_back(i + 1);

        pu = vector<int>(n + 1, 0);
        pv = vector<int>(m + 1, 0);
        dist = vector<int>(n + 1, inf);
    }

    void add_edge(int u, int v){
        edg[u].push_back(v);
    }

    bool bfs(){
        queue<int> q;
        for (int u : U){
            if (!pu[u]){
                q.push(u);
                dist[u] = 0;
            }

            else
                dist[u] = inf;
        }

        dist[0] = inf;
        while (q.size() > 0){
            int u = q.front();
            q.pop();

            if (dist[u] < dist[0]){
                for (int v : edg[u]){
                    if (dist[pv[v]] == inf){
                        q.push(pv[v]);
                        dist[pv[v]] = dist[u] + 1;
                    }
                }
            }
        }

        if (dist[0] == inf)
            return 0;
        return 1;
    }

    bool dfs(ll u){
        if (u == 0) return 1;
        for (int v : edg[u]){
            if (dist[pv[v]] == (dist[u] + 1)){
                if (dfs(pv[v])){
                    pu[u] = v;
                    pv[v] = u;
                    return 1;
                }
            }
        }

        dist[u] = 0;
        return 0;
    }

    int solve(){
        int res = 0;
        while (bfs()){
            for (int u : U){
                if (!pu[u])
                    if (dfs(u))
                        res++;
            }
        }

        return res;
    }
};
```

3.8 General Matching (Blossom)

Overview

Find maximum matching on general graph.

Time complexity: $\mathcal{O}(n^3)$ **worst case**

Implementation

```cpp
struct Matching {
    int n;
    vector<vector<int>> g;
    vector<int> mt;
    vector<int> is_ev, gr_buf;
    vector<pi> nx;
    int st;
    int group(int x) {
        if(gr_buf[x] == -1 || is_ev[gr_buf[x]] != st) return gr_buf[x];
        return gr_buf[x] = group(gr_buf[x]);
    }
    void match(int p, int b) {
        int d = mt[p];
        mt[p] = b;
        if(d == -1 || mt[d] != p) return;
        if(nx[p].second == -1) {
            mt[d] = nx[p].first;
            match(nx[p].first, d);
        } else {
            match(nx[p].first, nx[p].second);
            match(nx[p].second, nx[p].first);
        }
    }
    bool arg() {
        is_ev[st] = st;
        gr_buf[st] = -1;
        nx[st] = pi(-1, -1);
        queue<int> q;
        q.push(st);
        while(q.size()) {
            int a = q.front();
            q.pop();
            for(auto b : g[a]) {
                if(b == st) continue;
                if(mt[b] == -1) {
                    mt[b] = a;
                    match(a, b);
                    return true;
                }
                if(is_ev[b] == st) {
                    int x = group(a), y = group(b);
                    if(x == y) continue;
                    int z = -1;
                    while(x != -1 || y != -1) {
                        if(y != -1) swap(x, y);
                        if(nx[x] == pi(a, b)) {
                            z = x;
                            break;
                        }
                        nx[x] = pi(a, b);
                        x = group(nx[mt[x]].first);
                    }
                    for(int v : {group(a), group(b)}) {
                        while(v != z) {
                            q.push(v);
                            is_ev[v] = st;
                            gr_buf[v] = z;
                            v = group(nx[mt[v]].first);
                        }
                    }
                } else if(is_ev[mt[b]] != st) {
                    is_ev[mt[b]] = st;
                    nx[b] = pi(-1, -1);
                    nx[mt[b]] = pi(a, -1);
                    gr_buf[mt[b]] = b;
                    q.push(mt[b]);
                }
            }
        }
        return false;
    }
    Matching(const vector<vector<int>> &_g) : n(int(_g.size())), g(_g), mt(n, -1), is_ev(n, -1), gr_buf(n), nx(n) {
        for(st = 0; st < n; st++)
```

```cpp
            if(mt[st] == -1) arg();
    }
    vector<pi> max_match() {
        vector<pi> res;
        for (int i = 0; i < n; i++){
            if(i < mt[i])
                res.push_back({i, mt[i]});
        }
        return res;
    }
};
```

4 Math

4.1 Modular Int

Overview

Helper class, some implementations below will use this.

Implementation

```cpp
template<ll mod = 1000000007>
struct modu{
    ll val;
    modu(ll x){
        val = x;
        val %= mod;
        if (val < 0) val += mod;
    }
    modu(){ val = 0; }

    operator ll() const { return val; }
    modu operator+(modu const& other){ return val + other.val; }
    modu operator-(modu const& other){ return val - other.val; }
    modu operator*(modu const& other){ return val * other.val; }
    modu operator/(modu const& other){ return *this * other.inv(); }
    modu operator+=(modu const& other) { *this = *this + other; return *this; }
    modu operator-=(modu const& other) { *this = *this - other; return *this; }
    modu operator*=(modu const& other) { *this = *this * other; return *this; }
    modu operator/=(modu const& other) { *this = *this / other; return *this; }
    modu operator++(int) {modu tmp = *this; *this += 1; return tmp;}
    modu operator++() {*this += 1; return *this;}
    modu operator--(int) {modu tmp = *this; *this -= 1; return tmp;}
    modu operator--() {*this -= 1; return *this;}
    modu operator-() {return modu(-val);}
    friend ostream& operator<<(ostream& os, modu const& m) { return os << m.val; }
    friend istream& operator>>(istream& is, modu & m) { return is >> m.val; }

    modu pow(ll x) const{
        if (x == 0)
            return 1;
        if (x % 2 == 0){
            modu tmp = pow(x / 2);
            return tmp * tmp;
        }
        else
            return pow(x - 1) * *this;
    }

    modu inv() const{ return pow(mod - 2); }
};
```

4.2 Modular Square Root

Overview

Operations on field

$$\langle u, v\rangle = u + v\sqrt{k} \mod p$$

Implementation

```cpp
ll MOD = 999999893;
ll sq = 2;

class EX {
  int re, im;
  static int trim(int a) {
    if (a >= MOD) a -= MOD;
    if (a < 0) a += MOD;
    return a;
  }
  static int inv(const int a) {
    int ans = 1;
    for (int cur = a, p = MOD - 2; p; p >>= 1, cur = 1ll * cur * cur % MOD) {
      if (p&1) ans = 1ll * ans * cur % MOD;
    }
    return ans;
  };
public:
  EX(int re = 0, int im = 0) : re(re), im(im) {}
  EX& operator=(EX oth) { return re = oth.re, im = oth.im, *this; }
  int norm() const {
    return trim((1ll * re * re - 1ll * sq * im % MOD * im) % MOD);
  }
  EX conj() const {
    return EX(re, trim(MOD - im));
  }
  EX operator*(EX oth) const {
    return EX((1ll * re * oth.re + 1ll * sq * im % MOD * oth.im) % MOD,
          (1ll * re * oth.im + 1ll * im * oth.re) % MOD);
  };
  EX operator/(int n) const {
    return EX(1ll * re * inv(n) % MOD, 1ll * im * inv(n) % MOD);
  }
  EX operator/(EX oth) const { return *this * oth.conj() / oth.norm(); }
  EX operator+(EX oth) const { return EX(trim(re + oth.re), trim(im + oth.im)); }
  EX operator-(EX oth) const {
    return EX(trim(re - oth.re), trim(im - oth.im));
  }
  EX pow(long long n) const {
    EX ans(1);
    for (EX a = *this; n; n >>= 1, a = a * a) {
      if (n&1) ans = a * ans;
    }
    return ans;
  }
  bool operator==(EX oth) const { return re == oth.re and im == oth.im; }
  bool operator!=(EX oth) const { return not (*this == oth); }
  int real() const& { return re; }
  int imag() const& { return im; }
};
```

4.3 Discrete Log

Overview

Given a, b, m, find any x that satisfy

$$a^x = b \mod m$$

Time complexity: $\mathcal{O}(N \log\log N)$

Implementation

```cpp
// Returns minimum x for which a ^ x % m = b % m.
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int k = 1, add = 0, g;
    while ((g = gcd(a, m)) > 1) {
        if (b == k)
            return add;
        if (b % g)
            return -1;
        b /= g, m /= g, ++add;
        k = (k * 1ll * a / g) % m;
    }

    int n = sqrt(m) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;

    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }

    for (int p = 1, cur = k; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur] + add;
            return ans;
        }
    }
    return -1;
}
```

4.4 Primite Root

Overview

Given a, n, find g so that for any a such that $gcd(a, n) = 1$, there exists k such that

$$g^k = a \mod n$$

Time complexity: $\mathcal{O}(Ans \cdot \log \phi(n) \cdot \log n))$

Implementation

```
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;
}

int generator (int p) {
    vector<int> fact;
    int phi = p-1,  n = phi;
    for (int i=2; i*i<=n; ++i)
        if (n % i == 0) {
            fact.push_back (i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
        fact.push_back (n);

    for (int res=2; res<=p; ++res) {
        bool ok = true;
        for (size_t i=0; i<fact.size() && ok; ++i)
            ok &= powmod (res, phi / fact[i], p) != 1;
        if (ok)  return res;
    }
    return -1;
}
```

4.5 Euler's Totient Funnction

Overview

Find $\phi(i)$ for i from 1 to N.

Time complexity: $\mathcal{O}(N \log \log N)$

Implementation

```
int phi[def];
void phi(int n) {
    phi[0] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i++)
        phi[i] = i - 1;

    for (int i = 2; i <= n; i++)
        for (int j = 2 * i; j <= n; j += i)
              phi[j] -= phi[i];
}
```

4.6 Chinese Remainder Theorem

Overview

Given a system of congruences

$$a = a_1 \mod M_1, a = a_2 \mod M_2, \ldots$$

where M_i might not be pairwise coprime, find any a that satisfy it.

Time complexity: $\mathcal{O}(N \log \max(M_i))$

Implementation

```
typedef __int128_t i128;
i128 exeuclid(i128 a, i128 b, i128& x, i128& y){
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    i128 x1, y1;
    i128 d = exeuclid(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

struct CBT{
    i128 A = 0, M = 0;
    void add(i128 a, i128 m){
        a = ((a % m) + m) % m;
        i128 _M = M;
        if (M == 0){
            A = a, M = m;
            return;
        }
        if (A == -1) return;
        i128 p, q;
        i128 g = exeuclid(M, m, p, q);
        if ((a - A) % g != 0){
            A = -1, M = -1;
            return;
        }
        i128 mul = (a - A) / g;
        M = m * M / g;
        A = (((_M * p * mul + A) % M) + M) % M;
    }
};
```

Usage

- The $add(x, y)$ function add the condition $a = x \mod y$.
- If $a \neq -1$, the solution a will satisfy $a = A \mod M$.

4.7 Extended Euclidean

Overview

Given a, b, find any x, y that satisfy

$$ax + by = gcd(a, b)$$

Note that the function pass x, y by reference and returns $gcd(a, b)$.

Time complexity: $\mathcal{O}(\log n))$

Implementation

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

4.8 Linear Diophantine

Overview

Given a, b, c, find any x, y that satisfy

$$ax + by = c$$

Time complexity: $\mathcal{O}(\log n))$

Implementation

```
bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g) {
    g = extended_euclid(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
```

4.9 Matrix

Overview

Matrix helper class.

Implementation

```cpp
template <typename T>
struct Matrix{
    vector<vector<T>> m;
    Matrix (vector<vector<T>> &m) : T(m){}
    Matrix (int r, int c) {
        m = vector<vector<T>>(r, vector<T>(c));
    }

    int row() const {return m.size();}
    int col() const {return m[0].size();}

    static Matrix identity(int n){
        Matrix res = Matrix(n, n);
        for (int i = 0; i < n; i++)
            res[i][i] = 1;
        return res;
    }

    auto & operator [] (int i) { return m[i]; }
    const auto & operator[] (int i) const { return m[i]; }

    Matrix operator * (const Matrix &b){
        Matrix a = *this;
        assert(a.col() == b.row());

        Matrix c(a.row(), b.col());
        for (int i = 0; i < a.row(); i++)
            for (int j = 0; j < b.col(); j++)
                for (int k = 0; k < a.col(); k++)
                    c[i][j] += a[i][k] * b[k][j];
        return c;
    }

    Matrix pow(ll x){
        assert(row() == col());
        Matrix crr = *this, res = identity(row());
        while (x > 0){
            if (x % 2 == 1)
                res = res * crr;
            crr = crr * crr;
            x /= 2;
        }
        return res;
    }
};
```

4.10 Miller Rabin Primality Test

Overview

Deterministic implementation of Miller Rabin.

Time complexity: Should be fast

Implementation

```cpp
ll binpower(ll base, ll e, ll mod) {
    ll result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (__int128_t)result * base % mod;
        base = (__int128_t)base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_composite(ll n, ll a, ll d, int s) {
    ll x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (__int128_t)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
};

bool MillerRabin(ll n) { // returns true if n is prime, else returns false.
    if (n < 2)
        return false;

    int r = 0;
    ll d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }

    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```

4.11 Fast Fourier Transform

Overview

multiplymod(A, B, M) returns C where

$$C[u] = \sum_{i=1}^{|A|} \sum_{j=1}^{|B|} A_i \cdot B_j \mod M \ (i + j = u)$$

Time complexity: $\mathcal{O}(n \log n)$

Implementation

```cpp
using cpx = complex<double>;
const double PI = acos(-1);
vector<cpx> roots = {{0, 0}, {1, 0}};

void ensure_capacity(int min_capacity) {
    for (int len = roots.size(); len < min_capacity; len *= 2) {
        for (int i = len >> 1; i < len; i++) {
            roots.emplace_back(roots[i]);
            double angle = 2 * PI * (2 * i + 1 - len) / (len * 2);
            roots.emplace_back(cos(angle), sin(angle));
        }
    }
}

void fft(vector<cpx> &z, bool inverse) {
    int n = z.size();
    assert((n & (n - 1)) == 0);
    ensure_capacity(n);
    for (unsigned i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j >= bit; bit >>= 1)
            j -= bit;
        j += bit;
        if (i < j)
            swap(z[i], z[j]);
    }
    for (int len = 1; len < n; len <<= 1) {
        for (int i = 0; i < n; i += len * 2) {
            for (int j = 0; j < len; j++) {
                cpx root = inverse ? conj(roots[j + len]) : roots[j + len];
                cpx u = z[i + j];
                cpx v = z[i + j + len] * root;
                z[i + j] = u + v;
                z[i + j + len] = u - v;
            }
        }
    }
    if (inverse)
        for (int i = 0; i < n; i++)
            z[i] /= n;
}
vector<int> multiply_mod(const vector<int> &a, const vector<int> &b, int m) {
    int need = a.size() + b.size() - 1;
    int n = 1;
    while (n < need)
        n <<= 1;
    vector<cpx> A(n);
    for (size_t i = 0; i < a.size(); i++) {
        int x = (a[i] % m + m) % m;
        A[i] = cpx(x & ((1 << 15) - 1), x >> 15);
    }
    fft(A, false);

    vector<cpx> B(n);
    for (size_t i = 0; i < b.size(); i++) {
        int x = (b[i] % m + m) % m;
        B[i] = cpx(x & ((1 << 15) - 1), x >> 15);
    }
    fft(B, false);

    vector<cpx> fa(n);
    vector<cpx> fb(n);
    for (int i = 0, j = 0; i < n; i++, j = n - i) {
        cpx a1 = (A[i] + conj(A[j])) * cpx(0.5, 0);
        cpx a2 = (A[i] - conj(A[j])) * cpx(0, -0.5);
        cpx b1 = (B[i] + conj(B[j])) * cpx(0.5, 0);
        cpx b2 = (B[i] - conj(B[j])) * cpx(0, -0.5);
        fa[i] = a1 * b1 + a2 * b2 * cpx(0, 1);
        fb[i] = a1 * b2 + a2 * b1;
    }

    fft(fa, true);
    fft(fb, true);
    vector<int> res(need);
    for (int i = 0; i < need; i++) {
        long long aa = (long long)(fa[i].real() + 0.5);
        long long bb = (long long)(fb[i].real() + 0.5);
        long long cc = (long long)(fa[i].imag() + 0.5);
        res[i] = (aa % m + (bb % m << 15) + (cc % m << 30)) % m;
    }
    return res;
}
```

4.12 OR Convolution

Overview

convoluteor(A, B) returns C where

$$C[u] = \sum_{i=1}^{|A|}\sum_{j=1}^{|B|} A_i \cdot B_j \mod M \ (i|j = u)$$

Time complexity: $\mathcal{O}(2^N \cdot N)$

Implementation

```cpp
vector<int> convolute_or(vector<int> &a, vector<int> &b){
    int n = a.size();
    for (int i = 0; i < n; i++) for (int j = 0; j < (1 << n); j++){
        if ((j >> i) & 1){
            a[j] += a[j - (1 << i)];
            b[j] += b[j - (1 << i)];
        }
    }
    for (int i = n - 1; i >= 0; i--){
        for (int j = (1 << n) - 1; j >= 0; j--){
            if ((j >> i) & 1)
                a[j] -= a[j - (1 << i)];
        }
    }
    auto c = vector<int>(n, 0);
    for (int i = n - 1; i < (1 << n); i++)
        c[i] = a[i] * b[i];
    for (int i = n - 1; i >= 0; i--){
        for (int j = (1 << n) - 1; j >= 0; j--){
            if ((j >> i) & 1)
                c[j] -= c[j - (1 << i)];
        }
    }
}
```

4.13 XOR Convolution

Overview

idk lol.

Implementation

```cpp
void xorconv(vector<int> &a,int modul){ // chuyen tu dang binh thuong sang dang dac biet, xong cu lay a[i] = b[i] * c[i] ...
    int n = a.size();
    for(int m = n/2; m; m/=2){
        for(int i = 0 ; i < n; i+= 2 * m){
            for(int j = 0;j <m;++ j){
                int x = a[i + j];
                int y = a[i + j + m];
                a[i + j] = (x + y)%modul;
                a[i + j + m] = (x-y+modul) % modul;
            }
        }
    }
}
void xorconv2(vector<int> &a,int modul){ // chuyen tu dang dac biet ve dang binh thuong => dap an sau khi fft
    int n = a.size();
    for(int m = 1; m<n; m*=2){
        for(int i = 0 ; i < n; i+= 2 * m){
            for(int j = 0;j <m;++ j){
                int x = a[i + j];
                int y = a[i + j + m];
                a[i + j] = (x + y)%modul;
                a[i + j + m] = (x-y+modul) % modul;
            }
        }
    }
    for(int i = 0;i<n;++i){
        a[i] = 1LL * (ll)a[i] * binpow(n,modul - 2, modul) %modul;
    }
}
```

5 String

5.1 Rolling Hash

Overview

Rolling hash implementation, use multiple mod if necessary.

Time complexity: $\mathcal{O}(N)$

Implementation

```cpp
struct hashu{
        ll n;
        vector<dd> p, h;

        hashu(string s){
                n = s.size();
                p = vector<dd>(n + 1);
                h = vector<dd>(n + 1);

                p[0] = {1, 1};
                for (int i = 1; i <= n; i++)
                        p[i] = p[i - 1] * base;
                for (int i = 1; i <= n; i++)
                        h[i] = (h[i - 1] * base) + (s[i - 1] - '0');
        }

        dd get(ll l, ll r){
                return h[r + 1] - (h[l] * p[r - l + 1]);
        }
};
```

5.2 Z-Function

Overview

Return an array where the i-th element corresponds to the longest substring starting from i that matches the prefix of s.

Time complexity: $\mathcal{O}(N)$

Implementation

```cpp
vector<int> z_func(string s){
        int n = s.size();
        vector<int> v(n);

        int l = 0, r = 0;
        for (int i = 1; i < n; i++){
                if (i < r)
                        v[i] = min(r - i, v[i - l]);
                while ((v[i] + i) < n && s[v[i]] == s[v[i] + i])
                        v[i]++;
                if ((v[i] + i) > r)
                        l = i, r = v[i] + i;
        }

        return v;
}
```

5.3 Prefix Function

Overview

Return an array where the i-th element corresponds to the longest substring ending at i that matches the prefix of s.

Time complexity: $\mathcal{O}(N)$

Implementation

```cpp
vector<int> pref_func(string s){
    int n = siz(s);
    vector<int> v(n);

    for (int i = 1; i < n; i++){
        ll j = v[i - 1];
        while (j > 0 && s[j] != s[i])
            j = v[j - 1];
        if (s[j] == s[i])
            j++;
        v[i] = j;
    }

    return v;
}
```

5.4 Manacher's Algorithm

Overview

Return an array where the i-th element corresponds to the longest palindrome that has i as the center, note that the algorithm only works for odd length palindrome, even can also be easily handled by inserting a dummy character in every even indicies.

Time complexity: $\mathcal{O}(N)$

Implementation

```
vector<int> manacher(string s) {
    int n = s.size();
    s = "$" + s + "^";
    vector<int> p(n + 2);
    int l = 1, r = 1;
    for(int i = 1; i <= n; i++) {
        p[i] = max(0, min(r - i, p[l + (r - i)]));
        while(s[i - p[i]] == s[i + p[i]]) {
            p[i]++;
        }
        if(i + p[i] > r) {
            l = i - p[i], r = i + p[i];
        }
    }
    return vector<int>(begin(p) + 1, end(p) - 1);
}
```

5.5 Aho-Corasick

Overview

Construct an automaton of `Trie` nodes, where $dp[i][c]$ is the next state of i when adding character c. If no state exists, we repeatedly go through the next longest available suffix j of i, and try to get $dp[j][c]$.

Time complexity: $\mathcal{O}(M * K)$, where M is the number of nodes in the `Trie`, and K is the alphabet size

Implementation

```
struct node{
    int p[26];
    int link;

    node(){
        for (int i = 0; i < 26; i++)
            p[i] = -1;
    }
};

struct Trie{
    int indx = 1;
    int dp[def][26];
    vector<node> p;

    Trie(){
        p.push_back(node());
    }

    int add(string s){
        ll crr = 0;
        for (int i = 0; i < s.size(); i++){
            int c = s[i] - 'a';
            if (p[crr].p[c] == -1){
                p[crr].p[c] = indx++;
                p.push_back(node());
            }

            crr = p[crr].p[c];
        }

        return crr;
    }

    void buildsuffix(){
        int n = p.size();

        queue<int> q;
        q.push(0);

        p[0].link = 0;
        for (int i = 0; i < n; i++) for (int j = 0; j < 26; j++)
                dp[i][j] = 0;

        while (q.size()){
            int u = q.front();
            q.pop();

            for (int i = 0; i < 26; i++){
                int v = p[u].p[i];
                if (v != -1){
                    dp[u][i] = v;
                    p[v].link = (u == 0)? 0 : dp[p[u].link][i];
                    q.push(v);
                }

                else
                    dp[u][i] = dp[p[u].link][i];
            }
        }
    }
};
```

6 Tree

6.1 Tree

Overview

Helper class, some implementations below will use this.

Implementation

```
struct Tree{
    vector<vector<int>> edg;
    vector<int> par, depth;
    int n, root;

    Tree(int n, int root) : n(n), root(root){
        edg = vector<vector<int>>(n, vector<int>());
    }
    void add(int u, int v){
        edg[u].push_back(v);
        edg[v].push_back(u);
    }
    void clear(){
        for (int u = 0; u < n; u++)
            edg[u].clear();
    }
    void remove_dup(){
        for (int u = 0; u < n; u++){
            sort(edg[u].begin(), edg[u].end());
            edg[u].erase(unique(edg[u].begin(), edg[u].end()), edg[u].end());
        }
    }
    void get_info(){
        par = depth = vector<int>(n, 0);
        par[root] = -1;
        dfs(root, -1);
    }
    void dfs(int u, int pre){
        for (int v : edg[u]){
            if (v == pre) continue;
            par[v] = u; depth[v] = depth[u] + 1;
            dfs(v, u);
        }
    }
};
```

6.2 Lowest Common Ancestor

Overview

Uses binary lifting to find the k-th parent of a node.

Time complexity: $\mathcal{O}(n \log n)$ for build, $\mathcal{O}(\log n)$ for query

Implementation

```
struct LCA{
    vector<vector<int>> f;
    Tree T;
    int n, k;

    LCA(Tree &_T) : T(_T){
        n = T.n; k = log2(n) + 2;
        for (int i = 0; i < n; i++)
            f.push_back(vector<int>(k, -1));
        T.get_info();

        for (int i = 0; i < n; i++)
            f[i][0] = T.par[i];
        for (int j = 1; j < k; j++) for (int i = 0; i < n; i++){
            int p = f[i][j - 1];
            if (p != -1)
                f[i][j] = f[p][j - 1];
        }
    }

    int get(int u, int v){
        if (T.depth[u] < T.depth[v])
            swap(u, v);
```

```
        for (int i = k - 1; i >= 0; i--){
            if (f[u][i] != -1 && T.depth[f[u][i]] >= T.depth[v])
                u = f[u][i];
        }
        if (u == v) return u;
        for (int i = k - 1; i >= 0; i--){
            if (f[u][i] != -1 && f[u][i] != f[v][i])
                u = f[u][i], v = f[v][i];
        }
        return T.par[u];
    }
};
```

6.3 Heavy Light Decomposition

Overview

Clean implementation of HLD, only uses 1 segment, $pos[u]$ is the position of u on the segment. Change the query function if needed, for now it's just max query using a segment tree

Time complexity: $\mathcal{O}(n \log n)$ for build, $\mathcal{O}(\log^2 n)$ for query

Implementation

```
struct HLD{
    vector<int> head, par, h, pos, big;
    int n, indx = 0;
    Tree T;

    HLD(Tree &_T) : T(_T){
        n = T.n;
        head = par = h = pos = big = vector<int>(n, 0);
        dfs(0, -1);
        decompose(0, 0, -1);
    }
    int dfs(int u, int pre){
        ll res = 1;
        big[u] = -1;

        int crr_size = 0;
        for (int v : T.edg[u]){
            if (v == pre)
                continue;

            par[v] = u; h[v] = h[u] + 1;
            int child_size = dfs(v, u);

            if (child_size > crr_size)
                big[u] = v, crr_size = child_size;
            res += child_size;
        }

        return res;
    }
    void decompose(int u, int root, int pre){
        head[u] = root, pos[u] = indx++;
        if (big[u] != -1)
            decompose(big[u], root, u);
        for (int v : T.edg[u]){
            if (v == pre || v == big[u])
                continue;
            decompose(v, v, u);
        }
    }
    ll query(int u, int v){
        ll res = -inf;
        while (head[u] != head[v]){
            if (h[head[u]] < h[head[v]])
                swap(u, v);
            maxi(res, st.get(pos[head[u]], pos[u]));
            u = par[head[u]];
        }

        if (h[u] < h[v])
            swap(u, v);
        maxi(res, st.get(pos[v], pos[u]));

        return res;
    }
};
```

6.4 Centroid Decomposition

Overview

Uses the centroid of a tree to decompose into smaller subtrees, each node will be recursively decomposed in $\mathcal{O}(log)$ times.

Time complexity: $\mathcal{O}(n \log n)$

Implementation

```
vector<ll> edg[def];
bool dead[def];
ll cnt[def];

void dfs(ll u, ll pre){
        cnt[u] = 1;
        for (ll v : edg[u]){
                if (v == pre || dead[v])
                        continue;
                dfs(v, u);
                cnt[u] += cnt[v];
        }
}

ll centroid(ll u, ll pre, ll n){
        for (ll v : edg[u]){
                if (v == pre || dead[v])
                        continue;
                if (cnt[v] > (n / 2))
                        return centroid(v, u, n);
        }
        return u;
}
long long get(ll u){
        dfs(u, -1);
        ll root = centroid(u, -1, cnt[u]);
        dead[root] = 1;

        for (ll v : edg[root]){
                if (!dead[v])
                        get(v);
        }
        return res;
}
```

7 Geometry (Kactl)

7.1 Kactl template

Overview

Kactl implementation sometimes use their own template, reference this for clarity.

Implementation

```
#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;
```

7.2 Point

Overview

Helper class, some implementations below will use this.

Implementation

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
        typedef Point P;
        T x, y;
        explicit Point(T x=0, T y=0) : x(x), y(y) {}
        bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
        bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
        P operator+(P p) const { return P(x+p.x, y+p.y); }
        P operator-(P p) const { return P(x-p.x, y-p.y); }
        P operator*(T d) const { return P(x*d, y*d); }
        P operator/(T d) const { return P(x/d, y/d); }
        T dot(P p) const { return x*p.x + y*p.y; }
        T cross(P p) const { return x*p.y - y*p.x; }
        T cross(P a, P b) const { return (a-*this).cross(b-*this); }
        T dist2() const { return x*x + y*y; }
        double dist() const { return sqrt((double)dist2()); }
        // angle to x-axis in interval [-pi, pi]
        double angle() const { return atan2(y, x); }
        P unit() const { return *this/dist(); } // makes dist()=1
        P perp() const { return P(-y, x); } // rotates +90 degrees
        P normal() const { return perp().unit(); }
        // returns point rotated 'a' radians ccw around the origin
        P rotate(double a) const {
                return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
```

```cpp
	friend ostream& operator<<(ostream& os, P p) {
		return os << "(" << p.x << "," << p.y << ")"; }
};
```

7.3 CCW

Overview

- Returns where p is as seen from s towards e. 1/0/-1 ⇔ left/on line/right.
- If the optional argument *eps* is given 0 is returned if p is within distance *eps* from the line.

Implementation

```cpp
template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
	auto a = (e-s).cross(p-s);
	double l = (e-s).dist()*eps;
	return (a > l) - (a < -l);
}
```

7.4 Circle Intersection

Overview

Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

Time complexity: $\mathcal{O}(1)$

Implementation

```cpp
typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out) {
	if (a == b) { assert(r1 != r2); return false; }
	P vec = b - a;
	double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
		p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
	if (sum*sum < d2 || dif*dif > d2) return false;
	P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
	*out = {mid + per, mid - per};
	return true;
}
```

7.5 Circle Line

Overview

Finds the intersection between a circle and a line. Returns a vector of either 0, 1, or 2 intersection points.

Time complexity: $\mathcal{O}(1)$

Implementation

```cpp
template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
	P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
	double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
	if (h2 < 0) return {};
	if (h2 == 0) return {p};
	P h = ab.unit() * sqrt(h2);
	return {p - h, p + h};
}
```

7.6 Circle Polygon

Overview

Returns the area of the intersection of a circle with a ccw polygon.

Time complexity: $\mathcal{O}(n)$

Implementation

```cpp
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
	auto tri = [&](P p, P q) {
		auto r2 = r * r / 2;
		P d = q - p;
		auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
		auto det = a * a - b;
		if (det <= 0) return arg(p, q) * r2;
		auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
		if (t < 0 || 1 <= s) return arg(p, q) * r2;
		P u = p + d * s, v = q + d * (t-1);
		return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
	};
	auto sum = 0.0;
	rep(i,0,sz(ps))
		sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
	return sum;
}
```

7.7 Circle Tagents

Overview

- Finds the external tangents of two circles, or internal if r2 is negated.
- Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers).
- .first and .second give the tangency points at circle 1 and 2 respectively.
- To find the tangents of a circle with a point set r2 to 0.

Time complexity: $\mathcal{O}(1)$

Implementation

```cpp
template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2) {
	P d = c2 - c1;
	double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
	if (d2 == 0 || h2 < 0)  return {};
	vector<pair<P, P>> out;
	for (double sign : {-1, 1}) {
		P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
		out.push_back({c1 + v * r1, c2 + v * r2});
	}
	if (h2 == 0) out.pop_back();
	return out;
}
```

7.8 Closest pair of points

Overview

Finds the closest pair of points.

Time complexity: $\mathcal{O}(n \log n)$

Implementation

```cpp
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
	assert(sz(v) > 1);
	set<P> S;
	sort(all(v), [](P a, P b) { return a.y < b.y; });
	pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
	int j = 0;
	for (P p : v) {
		P d{1 + (ll)sqrt(ret.first), 0};
		while (v[j].y <= p.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, {(*lo - p).dist2(), {*lo, p}});
		S.insert(p);
	}
	return ret.second;
}
```

7.9 Convex Hull

Overview

Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.

Time complexity: $\mathcal{O}(n \log n)$

Implementation

```
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
	if (sz(pts) <= 1) return pts;
	sort(all(pts));
	vector<P> h(sz(pts)+1);
	int s = 0, t = 0;
	for (int it = 2; it--; s = --t, reverse(all(pts)))
		for (P p : pts) {
			while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
			h[t++] = p;
		}
	return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```

7.10 Hull Diameter

Overview

Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).

Time complexity: $\mathcal{O}(n)$

Implementation

```
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
	int n = sz(S), j = n < 2 ? 0 : 1;
	pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
	rep(i,0,j)
		for (;; j = (j + 1) % n) {
			res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
			if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
				break;
		}
	return res.second;
}
```

7.11 Point inside Hull

Overview

- Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.
- **NOTE:** Requires **7.12** and **7.2**.

Time complexity: $\mathcal{O}(\log n)$

Implementation

```
bool inHull(const vector<P>& l, P p, bool strict = true) {
	int a = 1, b = sz(l) - 1, r = !strict;
	if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
	if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
	if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p)<= -r)
		return false;
	while (abs(a - b) > 1) {
		int c = (a + b) / 2;
		(sideOf(l[0], l[c], p) > 0 ? b : a) = c;
	}
	return sgn(l[a].cross(l[b], p)) < r;
}
```

7.12 Point on Segment

Overview

Returns true iff p lies on the line segment from s to e. Use $segDist(s, e, p) <= epsilon$ instead when using $Point < double >$.

Time complexity: $\mathcal{O}(1)$

Implementation

```
template<class P> bool onSegment(P s, P e, P p) {
	return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

7.13 Segment Distance

Overview

Returns the shortest distance between point p and the line segment from point s to e.

Time complexity: $\mathcal{O}(1)$

Implementation

```
template<class P> bool onSegment(P s, P e, P p) {
	return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

7.14 Segment Intersection

Overview

- If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned.
- If no intersection point exists an empty vector is returned.
- If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment.
- **NOTE:** Requires **7.12**.

Time complexity: $\mathcal{O}(1)$

Implementation

```
template<class P> vector<P> segInter(P a, P b, P c, P d) {
	auto oa = c.cross(d, a), ob = c.cross(d, b),
	     oc = a.cross(b, c), od = a.cross(b, d);
	// Checks if intersection is single non-endpoint point.
	if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
		return {(a * ob - b * oa) / (ob - oa)};
	set<P> s;
	if (onSegment(c, d, a)) s.insert(a);
	if (onSegment(c, d, b)) s.insert(b);
	if (onSegment(a, b, c)) s.insert(c);
	if (onSegment(a, b, d)) s.insert(d);
	return {all(s)};
}
```

7.15 Line Distance

Overview

- Returns the signed distance between point p and the line containing points a and b.
- Positive value on left side and negative on right as seen from a towards b. a==b gives nan.

Time complexity: $\mathcal{O}(1)$

Implementation

```
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
	return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

7.16 Line Intersection

Overview

- If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {1, point} is returned.
- If no intersection point exists {0, (0,0)} is returned and if infinitely many exists {-1, (0,0)} is returned.

Time complexity: $\mathcal{O}(1)$

Implementation

```
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
	auto d = (e1 - s1).cross(e2 - s2);
	if (d == 0) // if parallel
		return {-(s1.cross(e1, s2) == 0), P(0, 0)};
	auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
	return {1, (s1 * p + e1 * q) / d};
}
```

7.17 Polygon Area

Overview

Returns twice the signed area of a polygon.

Time complexity: $\mathcal{O}(n)$

Implementation

```
template<class T>
T polygonArea2(vector<Point<T>>& v) {
	T a = v.back().cross(v[0]);
	rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
	return a;
}
```

7.18 Polygon Cut

Overview

Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.

Time complexity: $\mathcal{O}(n)$

Implementation

```
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
	vector<P> res;
	rep(i,0,sz(poly)) {
		P cur = poly[i], prev = i ? poly[i-1] : poly.back();
		auto a = s.cross(e, cur), b = s.cross(e, prev);
		if ((a < 0) != (b < 0))
			res.push_back(cur + (prev - cur) * (a / (a - b)));
		if (a < 0)
			res.push_back(cur);
	}
	return res;
}
```

7.19 Point inside Polygon

Overview

- Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary.
- **NOTE:** Requires **7.12** and **7.2**.

Time complexity: $\mathcal{O}(n)$

Implementation

```
template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
	int cnt = 0, n = sz(p);
	rep(i,0,n) {
		P q = p[(i + 1) % n];
		if (onSegment(p[i], q, a)) return !strict;
		//or: if (segDist(p[i], q, a) <= eps) return !strict;
		cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
	}
	return cnt;
}
```

7.20 Manhattan MST

Overview

Given N points, returns up to $4 * N$ edges, which are guaranteed to contain a minimum spanning tree for the graph with edge weights $w(p, q) = |p.x - q.x| + |p.y - q.y|$. Edges are in the form (distance, src, dst). Use a standard MST algorithm on the result to find the final MST.

Time complexity: $\mathcal{O}(n)$

Implementation

```
typedef Point<int> P;
vector<array<int, 3>> manhattanMST(vector<P> ps) {
	vi id(sz(ps));
	iota(all(id), 0);
	vector<array<int, 3>> edges;
	rep(k,0,4) {
		sort(all(id), [&](int i, int j) {
			return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
		map<int, int> sweep;
		for (int i : id) {
			for (auto it = sweep.lower_bound(-ps[i].y);
				it != sweep.end(); sweep.erase(it++)) {
				int j = it->second;
				P d = ps[i] - ps[j];
				if (d.y > d.x) break;
				edges.push_back({d.y + d.x, i, j});
			}
			sweep[-ps[i].y] = i;
		}
		for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x, p.y);
	}
	return edges;
}
```

8 Notes

8.1 Finding min cut

To build a min cut, once you have finished finding the max flow, bfs from source one more time. Edges that connect reached vertex and unreached vertex is considered a cut.

8.2 Finding minimum vertex cover on bipartite graph (Kőnig's theorem)

- Size of maximum matching = Size of minimum vertex cover.
- To build, use flow to find the maximum matching, and bfs from source one more time. The minimum vertex cover is the set of all vertices in the left partition that were not visited, combined with all vertices in the right partition that were visited.
- The weighted version is the same, except the capacity of the edge from source/sink to a vertex is that vertex weight.