



Chulalongkorn University

# Here for the Free Snacks

Teetat Thamronglak  
Thawin Tengamnuay  
Thitirin Sastarasadhit

2025-10-31

Contest	1
Mathematics	1
Data structures	2
Number Theory	4
Combinatorial	4
Tree	5
Flows	7
Geometry	10
String	14
Convolutions	15
Polynomials	17
Modular Arithmetic	18
Linear Programming	19
Miscellaneous	19

## Contest (1)

template.hpp27 lines

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

```
using namespace std;
using namespace __gnu_pbds;
```

```
using ll = long long;
using db = long double;
using vi = vector<int>;
using vl = vector<ll>;
using vd = vector<db>;
using pii = pair<int,int>;
using pll = pair<ll,ll>;
using pdd = pair<db,db>;
const int INF=0x3fffffff;
const ll LINF=0x1fffffffffffffff;
const db DINF=numeric_limits<db>::infinity();
const db EPS=1e-9;
const db PI=acos(db(-1));
```

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

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
count());
mt19937_64 rng64(chrono::steady_clock::now().time_since_epoch()
.count());
```

.bashrc3 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++17 \
-fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps = <
```

hash.sh3 lines

```
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]' | md5sum |cut -c-6
```

## Mathematics (2)

### 2.1 Equations

$$ax^2+bx+c=0\Rightarrow x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

The extremum is given by  $x=-b/2a$ .

$$\begin{aligned}ax+by&=e & x&=\frac{ed-bf}{ad-bc}\\cx+dy&=f & y&=\frac{af-ec}{ad-bc}\end{aligned}$$

In general, given an equation  $Ax=b$ , the solution to a variable  $x_i$  is given by

$$x_i=\frac{\det A'_i}{\det A}$$

where  $A'_i$  is  $A$  with the  $i$ 'th column replaced by  $b$ .

### 2.2 Recurrences

If  $a_n=c_1a_{n-1}+\cdots+c_ka_{n-k}$ , and  $r_1,\dots,r_k$  are distinct roots of  $x^k-c_1x^{k-1}-\cdots-c_k$ , there are  $d_1,\dots,d_k$  s.t.

$$a_n=d_1r_1^n+\cdots+d_kr_k^n.$$

Non-distinct roots  $r$  become polynomial factors, e.g.

$$a_n=(d_1n+d_2)r^n.$$

### 2.3 Trigonometry

$$\sin(v+w)=\sin v\cos w+\cos v\sin w$$

$$\cos(v+w)=\cos v\cos w-\sin v\sin w$$

$$\tan(v+w)=\frac{\tan v+\tan w}{1-\tan v\tan w}$$

$$\sin v+\sin w=2\sin\frac{v+w}{2}\cos\frac{v-w}{2}$$

$$\cos v+\cos w=2\cos\frac{v+w}{2}\cos\frac{v-w}{2}$$

$$(V+W)\tan(v-w)/2=(V-W)\tan(v+w)/2$$

where  $V,W$  are lengths of sides opposite angles  $v,w$ .

$$a\cos x+b\sin x=r\cos(x-\phi)$$

$$a\sin x+b\cos x=r\sin(x+\phi)$$

where  $r=\sqrt{a^2+b^2},\phi=\operatorname{atan2}(b,a)$ .

### 2.4 Geometry

#### 2.4.1 Triangles

Side lengths:  $a,b,c$

$$\text{Semiperimeter: }p=\frac{a+b+c}{2}$$

$$\text{Area: }A=\sqrt{p(p-a)(p-b)(p-c)}$$

$$\text{Circumradius: }R=\frac{abc}{4A}$$

$$\text{Inradius: }r=\frac{A}{p}$$

Length of median (divides triangle into two equal-area triangles):

$$m_a=\frac{1}{2}\sqrt{2b^2+2c^2-a^2}$$

Length of bisector (divides angles in two):

$$s_a=\sqrt{bc\left[1-\left(\frac{a}{b+c}\right)^2\right]}$$

$$\text{Law of sines: }\frac{\sin\alpha}{a}=\frac{\sin\beta}{b}=\frac{\sin\gamma}{c}=\frac{1}{2R}$$

$$\text{Law of cosines: }a^2=b^2+c^2-2bc\cos\alpha$$

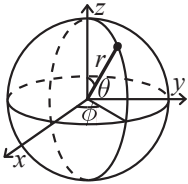
$$\text{Law of tangents: }\tan\frac{\alpha+\beta}{2}=\frac{a+b}{a-b}\tan\frac{\alpha-\beta}{2}$$

With side lengths  $a,b,c$ , diagonals  $e,f$ , diagonals angle  $\theta$ , area  $A$  and magic flux  $F=b^2+\tan^2\theta-a^2-c^2$ :

$$4A=2ef\cdot\sin\theta=F\tan\theta=\sqrt{4e^2f^2-F^2}$$

#### 2.4.3 Spherical coordinates

For cyclic quadrilaterals the sum of opposite angles is  $180^\circ$ ,  $ef=ac+bd$ , and  $A=\sqrt{(p-a)(p-b)(p-c)(p-d)}$ .



$$\begin{aligned}x&=r\sin\theta\cos\phi & r&=\sqrt{x^2+y^2+z^2}\\y&=r\sin\theta\sin\phi & \theta&=\operatorname{acos}(z/\sqrt{x^2+y^2+z^2})\\z&=r\cos\theta & \phi&=\operatorname{atan2}(y,x)\end{aligned}$$

2.5 Derivatives/Integrals

$$\frac{d}{dx} \arcsin x = \frac{1}{\sqrt{1-x^2}}$$
$$\frac{d}{dx} \tan x = 1 + \tan^2 x$$
$$\int \tan ax = -\frac{\ln|\cos ax|}{a}$$
$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2} \operatorname{erf}(x)$$

$$\frac{d}{dx} \arccos x = -\frac{1}{\sqrt{1-x^2}}$$
$$\frac{d}{dx} \arctan x = \frac{1}{1+x^2}$$
$$\int x \sin ax = \frac{\sin ax - ax \cos ax}{a^2}$$
$$\int x e^{ax} dx = \frac{e^{ax}}{a^2} (ax - 1)$$

Integration by parts:

$$\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$$

2.6 Sums

$$c^a + c^{a+1} + \dots + c^b = \frac{c^{b+1} - c^a}{c - 1}, c \neq 1$$

$$1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$$
$$1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(2n+1)(n+1)}{6}$$
$$1^3 + 2^3 + 3^3 + \dots + n^3 = \frac{n^2(n+1)^2}{4}$$
$$1^4 + 2^4 + 3^4 + \dots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

2.7 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots, (-\infty < x < \infty)$$
$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots, (-1 < x \leq 1)$$
$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \dots, (-1 \leq x \leq 1)$$
$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots, (-\infty < x < \infty)$$
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots, (-\infty < x < \infty)$$

2.8 Probability theory

Let  $X$  be a discrete random variable with probability  $p_X(x)$  of assuming the value  $x$ . It will then have an expected value (mean)  $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$  and variance  $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$  where  $\sigma$  is the standard deviation. If  $X$  is instead continuous it will have a probability density function  $f_X(x)$  and the sums above will instead be integrals with  $p_X(x)$  replaced by  $f_X(x)$ .

Expectation is linear:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent  $X$  and  $Y$ ,

$$V(aX + bY) = a^2V(X) + b^2V(Y).$$

2.8.1 Discrete distributions

Binomial distribution

The number of successes in  $n$  independent yes/no experiments, each which yields success with probability  $p$  is  $\text{Bin}(n, p)$ ,  $n = 1, 2, \dots$ ,  $0 \leq p \leq 1$ .

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\mu = np, \sigma^2 = np(1-p)$$

$\text{Bin}(n, p)$  is approximately  $\text{Po}(np)$  for small  $p$ .

First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each which yields success with probability  $p$  is  $\text{Fs}(p)$ ,  $0 \leq p \leq 1$ .

$$p(k) = p(1-p)^{k-1}, k = 1, 2, \dots$$

$$\mu = \frac{1}{p}, \sigma^2 = \frac{1-p}{p^2}$$

Poisson distribution

The number of events occurring in a fixed period of time  $t$  if these events occur with a known average rate  $\kappa$  and independently of the time since the last event is  $\text{Po}(\lambda)$ ,  $\lambda = t\kappa$ .

$$p(k) = e^{-\lambda} \frac{\lambda^k}{k!}, k = 0, 1, 2, \dots$$

$$\mu = \lambda, \sigma^2 = \lambda$$

2.8.2 Continuous distributions

Uniform distribution

If the probability density function is constant between  $a$  and  $b$  and 0 elsewhere it is  $\text{U}(a, b)$ ,  $a < b$ .

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2}, \sigma^2 = \frac{(b-a)^2}{12}$$

Exponential distribution

The time between events in a Poisson process is  $\text{Exp}(\lambda)$ ,  $\lambda > 0$ .

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$
$$\mu = \frac{1}{\lambda}, \sigma^2 = \frac{1}{\lambda^2}$$

Normal distribution

Most real random values with mean  $\mu$  and variance  $\sigma^2$  are well described by  $\mathcal{N}(\mu, \sigma^2)$ ,  $\sigma > 0$ .

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If  $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$  and  $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$  then

$$aX_1 + bX_2 + c \sim \mathcal{N}(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

2.9 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let  $X_1, X_2, \dots$  be a sequence of random variables generated by the Markov process. Then there is a transition matrix  $\mathbf{P} = (p_{ij})$ , with  $p_{ij} = \Pr(X_n = i | X_{n-1} = j)$ , and  $\mathbf{p}^{(n)} = \mathbf{P}^n \mathbf{p}^{(0)}$  is the probability distribution for  $X_n$  (i.e.,  $p_i^{(n)} = \Pr(X_n = i)$ ), where  $\mathbf{p}^{(0)}$  is the initial distribution.

$\pi$  is a stationary distribution if  $\pi = \pi \mathbf{P}$ . If the Markov chain is *irreducible* (it is possible to get to any state from any state), then  $\pi_i = \frac{1}{\mathbb{E}(T_i)}$  where  $\mathbb{E}(T_i)$  is the expected time between two visits in state  $i$ .  $\pi_j/\pi_i$  is the expected number of visits in state  $j$  between two visits in state  $i$ .

For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors,  $\pi_i$  is proportional to node  $i$ 's degree.

A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1).  $\lim_{k \rightarrow \infty} \mathbf{P}^k = \mathbf{1}\pi$ .

A Markov chain is an A-chain if the states can be partitioned into two sets **A** and **G**, such that all states in **A** are absorbing ( $p_{ii} = 1$ ), and all states in **G** leads to an absorbing state in **A**. The probability for absorption in state  $i \in \mathbf{A}$ , when the initial state is  $j$ , is  $a_{ij} = p_{ij} + \sum_{k \in \mathbf{G}} a_{ik} p_{kj}$ . The expected time until absorption, when the initial state is  $i$ , is  $t_i = 1 + \sum_{k \in \mathbf{G}} p_{ki} t_k$ .

## Data structures (3)

### HashMap.h

**Description:** Hash map with mostly the same API as unordered\_map, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

```

d77092, 7 lines
#include <bits/extc++.h>
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
    const uint64_t C = 114e18 * acos(0) | 71;
    ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
__gnu_pbds::gp_hash_table<ll,int, chash> h({},{},{},{},{1<16});
```

### line-container.hpp

**Description:** Line Container (Minimize).

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

```

24a5c4, 37 lines
struct Line{
    static bool querymode;
    ll m,c;
    mutable ll p;
    Line(ll m,ll c):m(m),c(c),p(0){}
    Line(ll p):m(0),c(0),p(p){}
    bool operator<(const Line &o) const{
        return querymode?p<o.p:m>o.m;
    }
};

bool Line::querymode=false;

struct LineContainer:multiset<Line>{
    ll div(ll a,ll b){
        return a/b-((a^b)<0&&a%b);
    }
    bool isect(iterator x,iterator y){
        if(y==end()) return x->p=LINF, false;
        if(x->m==y->m) x->p=x->c<=y->c?LINF:-LINF;
        else x->p=div(x->c-y->c,y->m-x->m);
        return x->p>y->p;
    }
    void add(ll m,ll c){
        auto x=insert(Line(m,c)),y=next(x);
        while(isect(x,y)) y=erase(y);
        if((y=x)!=begin() && isect(--x,y)) isect(x,erase(y));
        while((y=x)!=begin() && (--x->p>y->p) isect(x,erase(y)));
    }
    ll get(ll x){
        if(empty()) return LINF;
        Line::querymode=true;
        auto l=lower_bound(Line(x));
        Line::querymode=false;
        return l->m*x+l->c;
    }
};
```

## HashMap line-container Treap segment-tree-beats

### Treap.h

**Description:** A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.

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

```

1754b4, 53 lines
struct Node {
    Node *l = 0, *r = 0;
    int val, y, c = 1;
    Node(int val) : val(val), y(rand()) {}
    void recalc();
};

int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }

template<class F> void each(Node* n, F f) {
    if (n) { each(n->l, f); f(n->val); each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
    if (!n) return {};
    if (cnt(n->l) >= k) { // "n->val >= k" for lower_bound(k)
        auto [L,R] = split(n->l, k);
        n->l = R;
        n->recalc();
        return {L, n};
    } else {
        auto [L,R] = split(n->r, k - cnt(n->l) - 1); // and just "k"
        n->r = L;
        n->recalc();
        return {n, R};
    }
}

Node* merge(Node* l, Node* r) {
    if (!l) return r;
    if (!r) return l;
    if (l->y > r->y) {
        l->r = merge(l->r, r);
        return l->recalc(), l;
    } else {
        r->l = merge(l, r->l);
        return r->recalc(), r;
    }
}

Node* ins(Node* t, Node* n, int pos) {
    auto [l,r] = split(t, pos);
    return merge(merge(l, n), r);
}
```

// Example application: move the range [l, r) to index k

```

void move(Node& t, int l, int r, int k) {
    Node *a, *b, *c;
    tie(a,b) = split(t, l); tie(b,c) = split(b, r - l);
    if (k <= l) t = merge(ins(a, b, k), c);
    else t = merge(a, ins(c, b, k - r));
}
```

### segment-tree-beats.hpp

**Description:** Segment Tree Beats

```

bc2a26, 134 lines
struct SegmentTreeBeats{
    struct Node{
        ll sum,add;
        ll mn,mn2,fn;
        ll mx,mx2,fx;
        Node() {
            sum=add=fn=fx=0, mn=mn2=LINF, mx=mx2=-LINF;
        }
    };
};
```

```

Node(ll v){
    sum=mn=mx=v, add=0, mn2=LINF, mx2=-LINF, fn=fx=1;
}

friend Node operator+(const Node &l, const Node &r){
    Node res;
    res.sum=l.sum+r.sum;
    res.add=0;
    if(l.mx>r.mx){
        res.mx=l.mx, res.fx=l.fx;
        res.mx2=max(l.mx2, r.mx);
    } else if(r.mx>l.mx){
        res.mx=r.mx, res.fx=r.fx;
        res.mx2=max(r.mx2, l.mx);
    } else{
        res.mx=l.mx, res.fx=l.fx+r.fx;
        res.mx2=max(l.mx2, r.mx2);
    }
    if(l.mn<r.mn){
        res.mn=l.mn, res.fn=l.fn;
        res.mn2=min(l.mn2, r.mn);
    } else if(r.mn<l.mn){
        res.mn=r.mn, res.fn=r.fn;
        res.mn2=min(r.mn2, l.mn);
    } else{
        res.mn=l.mn, res.fn=l.fn+r.fn;
        res.mn2=min(l.mn2, r.mn2);
    }
    return res;
}

void apply(int l, int r, ll v){
    sum+=(r-l+1)*v;
    mx+=v, mx2+=v;
    mn+=v, mn2+=v;
    add+=v;
}

void chmin(ll v){
    if(v>=mx) return;
    sum+=(v-mx)*fx;
    if(mn==mx) mn=v;
    if(mn2==mx) mn2=v;
    mx=v;
}

void chmax(ll v){
    if(v<=mn) return;
    sum+=(v-mn)*fn;
    if(mx==mn) mx=v;
    if(mx2==mn) mx2=v;
    mn=v;
}

};

int n;
vector<Node> t;
SegmentTreeBeats(){}
SegmentTreeBeats(int n){init(n, [&](int){return 0;});}
template<class F>
SegmentTreeBeats(int n, const F &f){init(n, f);}
template<class F>
void init(int _n, const F &f){
    n=_n;
    int s=1;
    while(s<n*2) s<<=1;
    t.assign(s, Node());
    build(f);
}

template<class F>
void build(int l, int r, int i, const F &f){
    if(l==r) return void(t[i]=f(l));
    int m=(l+r)/2;
    build(l, m, i*2, f);
```

```
        build(m+1,r,i*2+1,f);
        pull(i);
    }
    void pull(int i){
        t[i]=t[i*2]+t[i*2+1];
    }
    void push(int l,int r,int i){
        int m=(l+r)/2;
        t[i*2].apply(l,m,t[i].add);
        t[i*2+1].apply(m+1,r,t[i].add);
        t[i*2].chmin(t[i].mx);
        t[i*2+1].chmin(t[i].mx);
        t[i*2].chmax(t[i].mn);
        t[i*2+1].chmax(t[i].mn);
        t[i].add=0;
    }
    void range_add(int l,int r,int i,int x,int y,ll v){
        if(y<l||r<x) return;
        if(x<=l&&r<=y) return t[i].apply(l,r,v);
        int m=(l+r)/2;
        push(l,r,i);
        range_add(l,m,i*2,x,y,v);
        range_add(m+1,r,i*2+1,x,y,v);
        pull(i);
    }
    void range_chmin(int l,int r,int i,int x,int y,ll v){
        if(y<l||r<x||t[i].mx<=v) return;
        if(x<=l&&r<=y&&t[i].mx2<v) return t[i].chmin(v);
        int m=(l+r)/2;
        push(l,r,i);
        range_chmin(l,m,i*2,x,y,v);
        range_chmin(m+1,r,i*2+1,x,y,v);
        pull(i);
    }
    void range_chmax(int l,int r,int i,int x,int y,ll v){
        if(y<l||r<x||t[i].mn>=v) return;
        if(x<=l&&r<=y&&t[i].mn2>v) return t[i].chmax(v);
        int m=(l+r)/2;
        push(l,r,i);
        range_chmax(l,m,i*2,x,y,v);
        range_chmax(m+1,r,i*2+1,x,y,v);
        pull(i);
    }
    ll query(int l,int r,int i,int x,int y){
        if(y<l||r<x) return 0;
        if(x<=l&&r<=y) return t[i].sum;
        int m=(l+r)/2;
        push(l,r,i);
        return query(l,m,i*2,x,y)+query(m+1,r,i*2+1,x,y);
    }
    template<class F>
    void build(const F &f){build(0,n-1,1,f);}
    void range_add(int x,int y,ll v){range_add(0,n-1,1,x,y,v);}
    void range_chmin(int x,int y,ll v){range_chmin(0,n-1,1,x,y,v);}
    void range_chmax(int x,int y,ll v){range_chmax(0,n-1,1,x,y,v);}
    ll query(int x,int y){return query(0,n-1,1,x,y);}
};
```

## Number Theory (4)

```
euclid.hpp
Description: Finds two integers x and y,such that ax+by=gcd(a,b). If
you just need gcd, use the built in _gcd instead. If a and b are coprime, then
x is the inverse of a (mod b). x=x0+k*(b/g) y=y0-k*(a/g)
36BBSf, 5 lines
ll euclid(ll a,ll b,ll &x,ll &y){
```

```
euclid crt floor-sum

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

crt.hpp
Description: Chinese Remainder Theorem.
crt(a, m, b, n) computes x such that x≡a (mod m), x≡b (mod n). If
|a|<m and |b|<n, x will obey 0≤x<lcm(m,n). Assumes mn<2^62. If
x0 and y0 is one of the solutions of ax+by=g, then the general solution
is x=x0+k*(b/g) and y=y0-k*(a/g).
Time: log(n)
"src/number-theory/euclid.hpp" cfd447, 7 lines
ll crt(ll a,ll m,ll b,ll n){
    if(n>m) swap(a,b),swap(m,n);
    ll x,y,g=euclid(m,n,x,y);
    if((a-b)%g!=0) return -1LL; // no solution
    x=(b-a)%n*x%n/g*m+a;
    return x<0?x+m*n/g:x;
}
```

```
floor-sum.hpp
Description: Floor sum function. f(a,b,c,n)=∑_{x=0}^n⌊(ax+b)/c⌋ becareful
when a,b,c are negative (use custom floor division and mod instead)
Time: O(log a) d088d2, 7 lines
ll floor_sum(ll a,ll b,ll c,ll n){
    ll res=n*(n+1)/2*(a/c)+(n+1)*(b/c);
    a%=c,b%=c;
    if(a==0) return res;
    ll m=(a*n+b)/c;
    return res+n*m-floor_sum(c,c-b-1,a,m-1);
}
```

## Combinatorial (5)

### 5.1 Permutations

#### 5.1.1 Factorial

$n$	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800
$n$	11	12	13	14	15	16	17			
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14			
$n$	20	25	30	40	50	100	150	171		
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX		

#### 5.1.2 Cycles

Let  $g_S(n)$  be the number of  $n$ -permutations whose cycle lengths all belong to the set  $S$ . Then

$$\sum_{n=0}^{\infty} g_S(n) \frac{x^n}{n!} = \exp\left(\sum_{n \in S} \frac{x^n}{n}\right)$$

#### 5.1.3 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1)+D(n-2)) = nD(n-1) + (-1)^n = \left\lfloor \frac{n!}{e} \right\rfloor$$

#### 5.1.4 Burnside’s lemma

Given a group  $G$  of symmetries and a set  $X$ , the number of elements of  $X$  up to symmetry equals

$$\frac{1}{|G|} \sum_{g \in G} |X^g|,$$

where  $X^g$  are the elements fixed by  $g$  ( $g.x = x$ ).

If  $f(n)$  counts “configurations” (of some sort) of length  $n$ , we can ignore rotational symmetry using  $G = \mathbb{Z}_n$  to get

$$g(n) = \frac{1}{n} \sum_{k=0}^{n-1} f(\gcd(n,k)) = \frac{1}{n} \sum_{k|n} f(k) \phi(n/k).$$

### 5.2 Partitions and subsets

#### 5.2.1 Partition function

Number of ways of writing  $n$  as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1, \quad p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k-1)/2)$$

	$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$														
<i>n</i>	0	1	2	3	4	5	6	7	8	9	20	50	100		
<i>p</i> ( <i>n</i> )	1	1	2	3	5	7	11	15	22	30	627	~2e5	~2e8		

#### 5.2.2 Lucas’ Theorem

Let  $n, m$  be non-negative integers and  $p$  a prime. Write  $n = n_k p^k + \dots + n_1 p + n_0$  and  $m = m_k p^k + \dots + m_1 p + m_0$ . Then  $\binom{n}{m} \equiv \prod_{i=0}^k \binom{n_i}{m_i} \pmod{p}$ .

### 5.3 General purpose numbers

#### 5.3.1 Bernoulli numbers

EGF of Bernoulli numbers is  $B(t) = \frac{t}{e^t-1}$  (FFT-able).  $B[0, \dots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \dots]$

Sums of powers:

$$\sum_{i=1}^n n^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k \cdot (n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\begin{aligned} \sum_{i=m}^{\infty} f(i) &= \int_m^{\infty} f(x) dx - \sum_{k=1}^{\infty} \frac{B_k}{k!} f^{(k-1)}(m) \\ &\approx \int_m^{\infty} f(x) dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m)) \end{aligned}$$

#### 5.3.2 Stirling numbers of the first kind

Number of permutations on  $n$  items with  $k$  cycles.

$$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k), \quad c(0,0) = 1$$

$$\sum_{k=0}^n c(n,k) x^k = x(x+1) \dots (x+n-1)$$

$c(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$   
 $c(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \dots$

5.3.3 Eulerian numbers

Number of permutations  $\pi \in S_n$  in which exactly  $k$  elements are greater than the previous element.  $k \ j$ :s s.t.  $\pi(j) > \pi(j+1)$ ,  $k+1 \ j$ :s s.t.  $\pi(j) \geq j$ ,  $k \ j$ :s s.t.  $\pi(j) > j$ .

$$E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$$

$$E(n,0) = E(n,n-1) = 1$$

$$E(n,k) = \sum_{j=0}^k (-1)^j \binom{n+1}{j} (k+1-j)^n$$

5.3.4 Stirling numbers of the second kind

Partitions of  $n$  distinct elements into exactly  $k$  groups.

$$S(n,k) = S(n-1,k-1) + kS(n-1,k)$$

$$S(n,1) = S(n,n) = 1$$

$$S(n,k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n$$

5.3.5 Bell numbers

Total number of partitions of  $n$  distinct elements.  $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \dots$  For  $p$  prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod{p}$$

5.3.6 Labeled unrooted trees

# on  $n$  vertices:  $n^{n-2}$   
# on  $k$  existing trees of size  $n_i$ :  $n_1 n_2 \dots n_k n^{k-2}$   
# with degrees  $d_i$ :  $(n-2)! / ((d_1-1)! \dots (d_n-1)!)$

5.3.7 Catalan numbers

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

$$C_0 = 1, \ C_{n+1} = \frac{2(2n+1)}{n+2} C_n, \ C_{n+1} = \sum C_i C_{n-i}$$

$$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \dots$$

- sub-diagonal monotone paths in an  $n \times n$  grid.
- strings with  $n$  pairs of parenthesis, correctly nested.
- binary trees with with  $n+1$  leaves (0 or 2 children).
- ordered trees with  $n+1$  vertices.
- ways a convex polygon with  $n+2$  sides can be cut into triangles by connecting vertices with straight lines.
- permutations of  $[n]$  with no 3-term increasing subseq.

Tree (6)

link-cut-tree static-top-tree

link-cut-tree.hpp

**Description:** Link Cut Tree (1-indexed)

38324f, 78 lines

```
template<int N,class T>
struct LinkCutTree{
    int ch[N][2],par[N],lz[N],rev[N];
    T val[N],sum[N],rsum[N];
    void toggle(int v){
        if(!v) return;
        swap(ch[v][0],ch[v][1]);
        swap(sum[v],rsum[v]);
        rev[v]^=1;
    }
    void push(int v){
        if(!v||!rev[v]) return;
        toggle(ch[v][0]);
        toggle(ch[v][1]);
        rev[v]=0;
    }
    void pull(int v){
        if(!v) return;
        sum[v]=sum[ch[v][0]]+val[v]+sum[ch[v][1]];
        rsum[v]=rsum[ch[v][0]]+val[v]+rsum[ch[v][1]];
    }
    bool is_root(int v){
        return ch[par[v]][0]!=v&&ch[par[v]][1]!=v;
    }
    bool pos(int v){
        return ch[par[v]][1]==v;
    }
    void rotate(int v){
        int u=par[v],g=par[u];
        bool x=pos(v);
        if(!is_root(u)) ch[g][pos(u)]=v;
        ch[u][x]=ch[v][!x];
        if(ch[u][x]) par[ch[u][x]]=u;
        ch[v][!x]=u,par[u]=v,par[v]=g;
        pull(u),pull(v);
    }
    void splay(int v){
        if(!v) return;
        for(push(v);!is_root(v);rotate(v)){
            int u=par[v];
            if(is_root(u)) push(u),push(v);
            else push(par[u]),push(u),push(v),rotate(pos(u)==pos(v)?u:v);
        }
    }
    void access(int v){
        for(int u=v,c=0;u;u=par[u]){
            splay(u);
            ch[u][1]=c;
            pull(c=u);
        }
        splay(v);
    }
    void evert(int v){
        access(v),toggle(v);
    }
    void link(int u,int v){
        evert(u);
        access(v);
        par[u]=v;
    }
    void cut(int u,int v){
        evert(u);
        access(v);
        assert(par[u]==v);
        ch[v][0]=par[u]=0;
```

```
        pull(v);
    }
    T aggregate(int u,int v){
        evert(u);
        access(v);
        return sum[v];
    }
    void set(int u,T v){
        evert(u);
        val[u]=v;
        pull(u);
    }
};
```

static-top-tree.hpp

**Description:** Static Top Tree.

d0731a, 198 lines

```
"src/contest/template.hpp"

template<class G>
struct StaticTopTree{
    using P = pair<int,int>;
    enum Type{Compress,Rake,AddEdge,AddVertex,Vertex};
    int n,root;
    G &adj;
    vector<int> hv,fa,lch,rch,par;
    vector<Type> type;
    StaticTopTree(G &adj):adj(adj){build();}
    int dfs(int u){
        int s=1,mx=0;
        for(auto v:adj[u]){
            if(v==fa[u]) continue;
            fa[v]=u;
            int t=dfs(v);
            if(t>mx)mx=t,hv[u]=v;
            s+=t;
        }
        return s;
    }
    void build(){
        n=adj.size();
        hv=fa=lch=rch=par=vector<int>(n,-1);
        type.assign(n,Compress);
        dfs(0,-1);
        root=compress(0).second;
    }
    int add(int i,int l,int r,Type t){
        if(i==-1){
            i=n++;
            lch.emplace_back(l);
            rch.emplace_back(r);
            par.emplace_back(-1);
            type.emplace_back(t);
        }else{
            lch[i]=l,rch[i]=r,type[i]=t;
        }
        if(l!=-1)par[l]=i;
        if(r!=-1)par[r]=i;
        return i;
    }
    /*
    pair<int,int> merge(vector<pair<int,int>> a,Type t){
        if(a.size()==1)return a[0];
        int tot=0;
        vector<pair<int,int>> l,r;
        for(auto [i,s]:a)tot+=s;
        for(auto [i,s]:a){
            (tot>s?l:r).emplace_back(i,s);
            tot-=s*2;
        }
        auto [i,si]=merge(l,t);
```

```

    auto [j, sj]=merge(r, t);
    return {add(-1, i, j, t), si+sj};
}
*/
P compress(int i){
    vector<P> a{add_vertex(i)};
    auto work=[&]() {
        auto [sj, j]=a.back();
        a.pop_back();
        auto [si, i]=a.back();
        a.back()={max(si, sj)+1, add(-1, i, j, Compress)};
    };
    while(hv[i]!=-1){
        a.emplace_back(add_vertex(i=hv[i]));
        while(true){
            if(a.size()>=3&&(a.end()[-3].first==a.end()[-2].first||a.end()[-3].first<=a.back().first)){
                P tmp=a.back();
                a.pop_back();
                work();
                a.emplace_back(tmp);
            }else if(a.size()>=2&&a.end()[-2].first<=a.back().first){
                work();
            }else break;
        }
    }
    while(a.size()>=2)work();
    return a[0];
}
P rake(int i){
    priority_queue<P, vector<P>, greater<P>> pq;
    for(int j:adj[i])if(j!=fa[i]&&j!=hv[i])pq.emplace(add_edge(j));
    while(pq.size()>=2){
        auto [si, i]=pq.top();pq.pop();
        auto [sj, j]=pq.top();pq.pop();
        pq.emplace(max(si, sj)+1, add(-1, i, j, Rake));
    }
    return pq.empty()?make_pair(0, -1):pq.top();
}
P add_edge(int i){
    auto [sj, j]=compress(i);
    return {sj+1, add(-1, j, -1, AddEdge)};
}
P add_vertex(int i){
    auto [sj, j]=rake(i);
    return {sj+1, add(i, j, -1, j==-1?Vertex::AddVertex)};
}
};

/*
struct TreeDP{
    struct Path{
        static Path unit();
    };
    struct Point{
        static Point unit();
    };
    static Path compress(Path l, Path r);
    static Point rake(Point l, Point r);
    static Point add_edge(Path p);
    static Path add_vertex(Point p, int u);
    static Path vertex(int u);
};
*/

template<class G, class TreeDP>

```

```

struct StaticTopTreeRerootingDP{
    using Path = typename TreeDP::Path;
    using Point = typename TreeDP::Point;
    StaticTopTree<G> stt;
    vector<Path> path, rpath;
    vector<Point> point;
    StaticTopTreeRerootingDP(G &adj):stt(adj){
        int n=stt.n;
        path.resize(n);
        point.resize(n);
        rpath.resize(n);
        dfs(stt.root);
    }
    void _update(int u){
        if(stt.type[u]==stt.Vertex){
            path[u]=rpath[u]=TreeDP::vertex(u);
        }else if(stt.type[u]==stt.Compress){
            path[u]=TreeDP::compress(path[stt.lch[u]], path[stt.rch[u]]);
            rpath[u]=TreeDP::compress(rpath[stt.rch[u]], rpath[stt.lch[u]]);
        }else if(stt.type[u]==stt.Rake){
            point[u]=TreeDP::rake(point[stt.lch[u]], point[stt.rch[u]]);
        }else if(stt.type[u]==stt.AddEdge){
            point[u]=TreeDP::add_edge(path[stt.lch[u]]);
        }else{
            path[u]=rpath[u]=TreeDP::add_vertex(point[stt.lch[u]], u);
        }
    }
    void dfs(int u){
        if(u==-1)return;
        dfs(stt.lch[u]);
        dfs(stt.rch[u]);
        _update(u);
    }
    void update(int u){
        for(;u!=-1;u=stt.par[u])_update(u);
    }
    Path query_all(){
        return path[stt.root];
    }
    Path query_subtree(int u){
        Path res=path[u];
        while(true){
            int p=stt.par[u];
            if(p==-1||stt.type[p]!=stt.Compress)break;
            if(stt.lch[p]==u)res=TreeDP::compress(path[stt.rch[p]], res);
        }
        return res;
    }
    Path query_reroot(int u){
        auto rec=[&](auto &&rec, int u)->Point {
            int p=stt.par[u];
            Path below=Path::unit(), above=Path::unit();
            while(p!=-1&&stt.type[p]==stt.Compress){
                int l=stt.lch[p], r=stt.rch[p];
                if(l==u)below=TreeDP::compress(below, path[r]);
                else above=TreeDP::compress(above, rpath[l]);
                u=p;
                p=stt.par[u];
            }
            if(p!=-1){
                u=p;
                p=stt.par[u];
                Point sum=Point::unit();
                while(stt.type[p]==stt.Rake){

```

```

                    int l=stt.lch[p], r=stt.rch[p];
                    sum=TreeDP::rake(sum, u=r?point[l]:point[r]);
                }
                u=p;
                p=stt.par[u];
            }
            sum=TreeDP::rake(sum, rec(rec, p));
            above=TreeDP::compress(above, TreeDP::add_vertex(sum, p));
        }
        return TreeDP::rake(TreeDP::add_edge(below), TreeDP::add_edge(above));
    };
    Point res=rec(rec, u);
    if(stt.type[u]==stt.AddVertex){
        res=TreeDP::rake(res, point[stt.lch[u]]);
    }
    return TreeDP::add_vertex(res, u);
}
};

```

## 6.1 Dominator Tree

Time:  $O((N + M) \log N)$

Source: USACO

- A node  $u$  **strictly dominates** another node  $v$  if  $u$  dominates  $v$  and  $u \neq v$ .
- Let the **immediate dominator** of  $u$ , or  $\text{idom}(u)$ , be the unique node  $v$  such that it strictly dominates node  $u$  and every other dominator of node  $u$  strictly dominates node  $v$ .
- Let  $e(u)$  be the entry time in node  $u$  during a Euler tour.
- Let the **semi-dominator** of  $u$ , or  $\text{sdom}(u)$ , be a node  $v$  such that there exists a path from  $v$  to  $u$  and  $e(i) \geq e(u)$  for every intermediate node  $i$  along the path from  $u$  to  $v$ , excluding the ends. If there are multiple nodes that satisfy this requirement, we take the node  $v$  with the smallest  $e(v)$ .
- Let the **relative dominator** of  $u$ , or  $\text{rdom}(u)$ , be the vertex  $x \neq \text{sdom}(u)$  on the path from  $\text{sdom}(u)$  to  $u$  in the Euler tour tree with the lowest  $\text{sdom}$  node number. Unlike with the  $\text{sdom}$ , ties in this function can be broken arbitrarily.

### Important properties

Proofs of these properties are located later in the module. For all of these properties, let  $u$  be a node that isn't the starting node.

1.  $\text{sdom}(u)$  is a proper ancestor of  $u$  in the DFS tree.
2. If  $\text{rdom}(u) = u$ , then  $\text{idom}(u) = \text{sdom}(u)$ .
3. If not, then  $\text{idom}(u) = \text{idom}(\text{rdom}(u))$ .

dominator-tree.cpp

192 lines

```

#include <algorithm>
#include <cassert>
#include <cmath>
#include <cstdint>

```

```

#include <functional>
#include <iostream>
#include <vector>

using std::cout;
using std::endl;
using std::vector;

// BeginCodeSnip{Tree Class for rdom}
template <typename T> class Tree {
private:
    const int root = 0;
    const vector<int> &parents;
    const vector<T> &vals;
    const int log2dist;
    vector<vector<int>>> pow2ends;
    vector<vector<T>>> pow2mins;
    vector<int> depth;

public:
    Tree(const vector<int> &parents, const vector<T> &vals)
        : parents(parents), vals(vals),
          log2dist(std::ceil(std::log2(parents.size() + 1))),
          pow2ends(parents.size(), vector<int>(log2dist + 1)),
          pow2mins(parents.size(), vector<T>(log2dist + 1)) {
        assert(parents[root] == -1);

        vector<vector<int>>> children(parents.size());
        for (int n = 0; n < parents.size(); n++) {
            if (parents[n] != -1) { children[parents[n]].push_back(n); }
        }
        depth = vector<int>(parents.size());
        vector<int> frontier{root};
        while (!frontier.empty()) {
            int curr = frontier.back();
            frontier.pop_back();
            for (int n : children[curr]) {
                depth[n] = depth[curr] + 1;
                frontier.push_back(n);
            }
        }

        for (int n = 0; n < parents.size(); n++) {
            pow2mins[n][0] = vals[n];
            pow2ends[n][0] = parents[n];
        }

        for (int p = 1; p <= log2dist; p++) {
            for (int n = 0; n < parents.size(); n++) {
                int halfway = pow2ends[n][p - 1];
                if (halfway == -1) {
                    pow2ends[n][p] = -1;
                    pow2mins[n][p] = pow2mins[n][p - 1];
                } else {
                    pow2ends[n][p] = pow2ends[halfway][p - 1];
                    pow2mins[n][p] =
                        std::min(pow2mins[n][p - 1], pow2mins[halfway][p - 1]);
                }
            }
        }
    }

    /**
     * @return the min value on the path from the ancestor to its
     *         descendant,
     *         not including the ancestor.
     */
    T min_val(int ancestor, int desc) {

```

```

        int dist = depth[desc] - depth[ancestor];
        T ret = vals[desc];
        int at = desc;
        for (int pow = 0; pow <= log2dist; pow++) {
            if ((dist & (1 << pow)) != 0) {
                ret = std::min(ret, pow2mins[at][pow]);
                at = pow2ends[at][pow];
            }
        }
        assert(at == ancestor);
        return ret;
    }
};
// EndCodeSnip

int main() {
    int city_num;
    int flight_num;
    std::cin >> city_num >> flight_num;
    vector<vector<int>>> adj(city_num);
    vector<vector<int>>> rev_adj(city_num);
    for (int f = 0; f < flight_num; f++) {
        int a, b;
        std::cin >> a >> b;
        adj[--a].push_back(--b);
        rev_adj[b].push_back(a);
    }

    // Variables used for the Euler Tour
    vector<int> visit_order;
    // visit_time is initialized with INF so nodes that aren't
    // reachable
    // during the DFS won't mess up our sdом calculation
    vector<int> visit_time(city_num, INT32_MAX);
    vector<int> visit_parent(city_num, -1);
    vector<bool> visited(city_num, false);
    std::function<void(int, int)> dfs;
    dfs = [&](int at, int prev) {
        if (visited[at]) { return; }
        visited[at] = true;

        visit_time[at] = visit_order.size();
        visit_order.push_back(at);
        visit_parent[at] = prev;
        for (int next : adj[at]) { dfs(next, at); }
    };
    dfs(0, -1);

    // We use a function-based interface instead of a class-based
    // one due to
    // heavy reliance on previously calculated values
    vector<int> dsu_parent(city_num, -1);
    vector<int> dsu_min(city_num, INT32_MAX);
    std::function<int(int)> find;
    find = [&](int x) {
        if (dsu_parent[x] == -1) { return x; }
        if (dsu_parent[dsu_parent[x]] != -1) {
            int parent_res = find(dsu_parent[x]);
            if (visit_time[parent_res] < visit_time[dsu_min[x]]) {
                dsu_min[x] = parent_res;
            }
            dsu_parent[x] = dsu_parent[dsu_parent[x]];
        }
        assert(dsu_min[x] != INT32_MAX);
        return dsu_min[x];
    };

    vector<int> sdом(city_num, -1);
    for (int i = visit_order.size() - 1; i > 0; i--) {

```

```

        int c = visit_order[i];
        // Iterate over all nodes with a directed edge to c
        for (int from : rev_adj[c]) {
            int find_res = find(from);
            // Take the node with the earliest visit time in the
            // traversal
            if (sdом[c] == -1 || visit_time[find_res] < visit_time[
                sdом[c]]) {
                sdом[c] = find_res;
            }
        }

        // Link c to its parent in the DFS tree
        dsu_parent[c] = visit_parent[c];
        dsu_min[c] = sdом[c];
    }

    // Initialize the values in the tree
    vector<std::pair<int, int>> sdом_times(city_num, {INT32_MAX,
        -1});
    for (int c : visit_order) {
        if (c != 0) {
            // The tree takes the minimum of these pairs, but what we
            // really want is the node itself— therefore, we store
            // both
            sdом_times[c] = {visit_time[sdom[c]], c};
        }
    }
    Tree tree(visit_parent, sdом_times);

    vector<int> rdom(city_num, -1);
    for (int c : visit_order) {
        if (c != 0) {
            // Use the definition of the rdom
            rdom[c] = tree.min_val(sdom[c], c).second;
        }
    }

    // Use the properties previously given to compute the idom
    vector<int> idом(city_num, -1);
    for (int c : visit_order) {
        if (c != 0) { idом[c] = rdom[c] == c ? sdом[c] : idом[rdom[
            c]]; }
    }

    // Trace the idом tree from the finish node to the start node
    // ,
    // finding all dominators along the way
    vector<int> critical{0};
    int at = city_num - 1;
    while (at != 0) {
        critical.push_back(at);
        at = idом[at];
    }

    std::sort(critical.begin(), critical.end());

    cout << critical.size() << '\n';
    for (int i = 0; i < critical.size() - 1; i++) { cout <<
        critical[i] + 1 << ' '; }
    cout << critical.back() + 1 << endl;
}

```



## dinic.hpp

**Description:** Dinic's Algorithm for finding the maximum flow.**Time:**  $\mathcal{O}(VE \log U)$  where  $U$  is the maximum flow.

2b9ab1, 88 lines

```

template<class T,bool directed=true,bool scaling=true>
struct Dinic{
    static constexpr T INF=numeric_limits<T>::max()/2;
    struct Edge{
        int to;
        T flow,cap;
        Edge(int _to,T _cap):to(_to),flow(0),cap(_cap){}
        T remain(){return cap-flow;}
    };
    int n,s,t;
    T U;
    vector<Edge> e;
    vector<vector<int>> g;
    vector<int> ptr,lv;
    bool calculated;
    T max_flow;
    Dinic(){
        Dinic(int n,int s,int t){init(n,s,t);}
        void init(int _n,int _s,int _t){
            n=_n,s=_s,t=_t;
            U=0;
            e.clear();
            g.assign(n,{});
            calculated=false;
        }
        void add_edge(int from,int to,T cap){
            assert(0<=from&&from<n&&0<=to&&to<n);
            g[from].emplace_back(e.size());
            e.emplace_back(to,cap);
            g[to].emplace_back(e.size());
            e.emplace_back(from,directed?0:cap);
            U=max(U,cap);
        }
        bool bfs(T scale){
            lv.assign(n,-1);
            vector<int> q{s};
            lv[s]=0;
            for(int i=0;i<(int)q.size();i++){
                int u=q[i];
                for(int j:g[u]){
                    int v=e[j].to;
                    if(lv[v]==-1&&e[j].remain()>=scale){
                        q.emplace_back(v);
                        lv[v]=lv[u]+1;
                    }
                }
            }
            return lv[t]!=-1;
        }
        T dfs(int u,int t,T f){
            if(u==t||f==0)return f;
            for(int &i=ptr[u];i<(int)g[u].size();i++){
                int j=g[u][i];
                int v=e[j].to;
                if(lv[v]==lv[u]+1){
                    T res=dfs(v,t,min(f,e[j].remain()));
                    if(res>0){
                        e[j].flow+=res;
                        e[j^1].flow-=res;
                        return res;
                    }
                }
            }
            return 0;
        }
    };
};

```

## binary-optimization k-ary-optimization

```

T flow(){
    if(calculated)return max_flow;
    calculated=true;
    max_flow=0;
    for(T scale=scaling?1LL<<(63-__builtin_clzll(U)):1LL;
        scale>0;scale>>=1){
        while(bfs(scale)){
            ptr.assign(n,0);
            while(true){
                T f=dfs(s,t,INF);
                if(f==0)break;
                max_flow+=f;
            }
        }
        return max_flow;
    }
    pair<T,vector<int>> cut(){
        flow();
        vector<int> res(n);
        for(int i=0;i<n;i++)res[i]=(lv[i]==-1);
        return {max_flow,res};
    }
};

```

## binary-optimization.hpp

**Description:** Binary Optimization. minimize  $\kappa + \sum_i \theta_i(x_i) + \sum_{i<j} \phi_{ij}(x_i, x_j) + \sum_{i<j<k} \psi_{ijk}(x_i, x_j, x_k)$  where  $x_i \in \{0, 1\}$  and  $\phi_{ij}, \psi_{ijk}$  are submodular functions. a set function  $f$  is submodular if  $f(S) + f(T) \geq f(S \cap T) + f(S \cup T)$  for all  $S, T$ .  $\phi_{ij}(0, 1) + \phi_{ij}(1, 0) \geq \phi_{ij}(1, 1) + \phi_{ij}(0, 0)$ .

"src/flows/dinic.hpp"

0d9c72, 95 lines

```

template<class T,bool minimize=true>
struct BinaryOptimization{
    static constexpr T INF=numeric_limits<T>::max()/2;
    int n,s,t,buf;
    T base;
    map<pair<int,int>,T> edges;
    BinaryOptimization(int _n):n(_n),s(n),t(n+1),buf(n+2),base(0){}
    void add_edge(int u,int v,T w){
        assert(w>=0);
        if(u==v||w==0)return;
        auto &e=edges[{u,v}];
        e=min(e+w,INF);
    }
    void add0(T w){
        base+=w;
    }
    void _add1(int i,T a,T b){
        if(a<=b){
            add0(a);
            add_edge(s,i,b-a);
        }else{
            add0(b);
            add_edge(i,t,a-b);
        }
    }
    void add1(int i,T x0,T x1){
        assert(0<=i&&i<n);
        if(!minimize)x0=-x0,x1=-x1;
        _add1(i,x0,x1);
    }
    void _add2(int i,int j,T a,T b,T c,T d){
        assert(b+c>=a+d);
        add0(a);
        _add1(i,0,c-a);
        _add1(j,0,d-c);
        add_edge(i,j,b+c-a-d);
    }
};

```

```

}
void add2(int i,int j,T x00,T x01,T x10,T x11){
    assert(i!=j&&0<=i&&i<n&&0<=j&&j<n);
    if(!minimize)x00=-x00,x01=-x01,x10=-x10,x11=-x11;
    _add2(i,j,x00,x01,x10,x11);
}
void _add3(int i,int j,int k,T a,T b,T c,T d,T e,T f,T g,T h){
    T p=a+d+f+g-b-c-e-h;
    if(p>=0){
        add0(a);
        _add1(i,0,f-b);
        _add1(j,0,g-e);
        _add1(k,0,d-c);
        _add2(i,j,0,c+e-a-g,0,0);
        _add2(i,k,0,0,b+e-a-f,0);
        _add2(j,k,0,b+c-a-d,0,0);
        int u=buf++;
        add0(-p);
        add_edge(i,u,p);
        add_edge(j,u,p);
        add_edge(k,u,p);
        add_edge(u,t,p);
    }else{
        add0(h);
        _add1(i,c-g,0);
        _add1(j,b-d,0);
        _add1(k,e-f,0);
        _add2(i,j,0,0,d+f-b-h,0);
        _add2(i,k,0,d+g-c-h,0,0);
        _add2(j,k,0,0,f+g-e-h,0);
        int u=buf++;
        add0(p);
        add_edge(s,u,-p);
        add_edge(u,i,-p);
        add_edge(u,j,-p);
        add_edge(u,k,-p);
    }
}
void add3(int i,int j,int k,T x000,T x001,T x010,T x011,T x100,T x101,T x110,T x111){
    assert(i!=j&&j!=k&&k!=i&&0<=i&&i<n&&0<=j&&j<n&&0<=k&&k<n);
    if(!minimize){
        x000=-x000,x001=-x001,x010=-x010,x011=-x011;
        x100=-x100,x101=-x101,x110=-x110,x111=-x111;
    }
    _add3(i,j,k,x000,x001,x010,x011,x100,x101,x110,x111);
}
pair<T,vector<int>> solve(){
    Dinic<T> dinic(buf,s,t);
    for(auto &[p,w]:edges){
        auto [u,v]=p;
        dinic.add_edge(u,v,w);
    }
    auto [ans,cut]=dinic.cut();
    ans+=base;
    ans=min(ans,INF);
    cut.resize(n);
    return {minimize?ans:-ans,cut};
}
};

```

## k-ary-optimization.hpp

**Description:** k-ary Optimization. minimize  $\kappa + \sum_i \theta_i(x_i) + \sum_{i < j} \phi_{ij}(x_i, x_j)$  where  $x_i \in \{0, 1, \dots, k-1\}$  and  $\phi_{i,j}$  is monge. A function  $f$  is monge if  $f(a, c) + f(b, d) \leq f(a, d) + f(b, c)$  for all  $a < b$  and  $c < d$ .  $\phi_{ij}(x-1, y) + \phi_{ij}(x, y+1) \leq \phi_{ij}(x-1, y+1) + \phi_{ij}(x, y)$ .  $\phi_{ij}(x, y) + \phi_{ij}(x-1, y+1) - \phi_{ij}(x-1, y) - \phi_{ij}(x, y+1) \geq 0$ .

"src/flows/dinic.hpp" 5139ce, 88 lines

```
template<class T, bool minimize=true>
struct K_aryOptimization{
    static constexpr T INF=numeric_limits<T>::max()/2;
    int n,s,t,buf;
    T base;
    vector<int> ks;
    vector<vector<int>> id;
    map<pair<int,int>,T> edges;
    K_aryOptimization(int n,int k){init(vector<int>(n,k));}
    K_aryOptimization(const vector<int> &_ks){init(_ks);}
    void init(const vector<int> &_ks){
        ks=_ks;
        n=ks.size();
        s=0,t=1,buf=2;
        base=0;
        id.clear();
        edges.clear();
        for(auto &k:ks){
            assert(k>=1);
            vector<int> a(k+1);
            a[0]=s,a[k]=t;
            for(int i=1;i<k;i++)a[i]=buf++;
            id.emplace_back(a);
            for(int i=2;i<k;i++)add_edge(a[i],a[i-1],INF);
        }
    }
    void add_edge(int u,int v,T w){
        assert(w>=0);
        if(u==v||w==0)return;
        auto &e=edges[{u,v}];
        e=min(e+w,INF);
    }
    void add0(T w){
        base+=w;
    }
    void _addl(int i,vector<T> cost){
        add0(cost[0]);
        for(int j=1;j<ks[i];j++){
            T x=cost[j]-cost[j-1];
            if(x>0)add_edge(id[i][j],t,x);
            if(x<0)add0(x),add_edge(s,id[i][j],-x);
        }
    }
    void addl(int i,vector<T> cost){
        assert(0<=i&&i<n&&(int)cost.size()==ks[i]);
        if(!minimize)for(auto &x:cost)x=-x;
        _addl(i,cost);
    }
    void _add2(int i,int j,vector<vector<T>> cost){
        int h=ks[i],w=ks[j];
        _addl(j,cost[0]);
        for(int x=h-1;x>=0;x--)for(int y=0;y<w;y++)cost[x][y]-=cost[0][y];
        vector<T> a(h);
        for(int x=0;x<h;x++)a[x]=cost[x][w-1];
        _addl(i,a);
        for(int x=0;x<h;x++)for(int y=0;y<w;y++)cost[x][y]-=a[x];
        for(int x=1;x<h;x++){
            for(int y=0;y<w-1;y++){
                T w=cost[x][y]+cost[x-1][y+1]-cost[x-1][y]-cost[x][y+1];
                assert(w>=0); // monge
            }
        }
    }
};
```

min-cost-flow hopcroft-karp

```
add_edge(id[i][x],id[j][y+1],w);
}
}
}
void add2(int i,int j,vector<vector<T>> cost){
    assert(0<=i&&i<n&&0<=j&&j<n&&i!=j);
    assert((int)cost.size()==ks[i]);
    for(auto &v:cost)assert((int)v.size()==ks[j]);
    if(!minimize)for(auto &v:cost)for(auto &x:v)x=-x;
    _add2(i,j,cost);
}
pair<T,vector<int>> solve(){
    Dinic<T> dinic(buf,s,t);
    for(auto &[p,w]:edges){
        auto [u,v]=p;
        dinic.add_edge(u,v,w);
    }
    auto [val,cut]=dinic.cut();
    val+=base;
    if(!minimize)val=-val;
    vector<int> ans(n);
    for(int i=0;i<n;i++){
        ans[i]=ks[i]-1;
        for(int j=1;j<ks[i];j++)ans[i]-=cut[id[i][j]];
    }
    return {val,ans};
}
};
```

min-cost-flow.hpp

**Description:** minimum-cost flow algorithm.

**Time:**  $\mathcal{O}(FE \log V)$  where  $F$  is max flow.

ca28ef, 83 lines

```
template<class F,class C>
struct MinCostFlow{
    struct Edge{
        int to;
        F flow,cap;
        C cost;
        Edge(int _to,F _cap,C _cost):to(_to),flow(0),cap(_cap),cost(_cost){}
        F getcap(){return cap-flow;}
    }
};
int n;
vector<Edge> e;
vector<vector<int>> adj;
vector<C> pot,dist;
vector<int> pre;
bool neg;
const F FINF=numeric_limits<F>::max()/2;
const C CINF=numeric_limits<C>::max()/2;
MinCostFlow(){
    MinCostFlow(int _n){
        init(_n);
    }
    void init(int _n){
        n=_n;
        e.clear();
        adj.assign(n,{});
        neg=false;
    }
    void addEdge(int u,int v,F cap,C cost){
        adj[u].emplace_back(e.size());
        e.emplace_back(v,cap,cost);
        adj[v].emplace_back(e.size());
        e.emplace_back(u,0,-cost);
        if(cost<0)neg=true;
    }
};
```

```
}
bool dijkstra(int s,int t){
    using P = pair<C,int>;
    dist.assign(n,CINF);
    pre.assign(n,-1);
    priority_queue<P,vector<P>,greater<P>> pq;
    dist[s]=0;
    pq.emplace(0,s);
    while(!pq.empty()){
        auto [d,u]=pq.top();
        pq.pop();
        if(dist[u]<d)continue;
        for(int i:adj[u]){
            int v=e[i].to;
            C ndist=d+pot[u]-pot[v]+e[i].cost;
            if(e[i].getcap()>0&&dist[v]>ndist){
                pre[v]=i;
                dist[v]=ndist;
                pq.emplace(ndist,v);
            }
        }
    }
    return dist[t]<CINF;
}
pair<F,C> flow(int s,int t){
    F flow=0;
    C cost=0;
    pot.assign(n,0);
    if(neg)for(int t=0;t<n;t++)for(int i=0;i<e.size();i++){
        if(e[i].getcap()>0){
            int u=e[i^1].to,v=e[i].to;
            pot[v]=min(pot[v],pot[u]+e[i].cost);
        } // Bellman-Ford
    }
    while(dijkstra(s,t)){
        for(int i=0;i<n;i++)pot[i]+=dist[i];
        F aug=FINF;
        for(int u=t;u!=s;u=pre[u]^1).to){
            aug=min(aug,e[pre[u]].getcap());
        } // find bottleneck
        for(int u=t;u!=s;u=pre[u]^1).to){
            e[pre[u]].flow+=aug;
            e[pre[u]^1].flow-=aug;
        } // push flow
        flow+=aug;
        cost+=aug*pot[t];
    }
    return {flow,cost};
}
};
```

hopcroft-karp.hpp

**Description:** Fast bipartite matching algorithm.

**Time:**  $\mathcal{O}(E\sqrt{V})$

456024, 52 lines

```
struct HopcroftKarp{
    int n,m;
    vector<int> match,lv,ptr;
    vector<vector<int>> adj;
    HopcroftKarp(){
        HopcroftKarp(int _n,int _m){init(_n,_m);}
        void init(int _n,int _m){
            n=_n,m=_m;
            adj.assign(n+m,vector<int>{});
        }
        void add_edge(int u,int v){
            adj[u].emplace_back(v+n);
        }
        void bfs(){
            lv.assign(n,-1);
        }
    }
};
```

```
queue<int> q;
for(int i=0;i<n;i++)if(match[i]==-1){
    lv[i]=0;
    q.emplace(i);
}
while(!q.empty()){
    int u=q.front();
    q.pop();
    for(int v:adj[u])if(match[v]!=-1&&lv[match[v]]==-1){
        lv[match[v]]=lv[u]+1;
        q.emplace(match[v]);
    }
}

bool dfs(int u){
    for(int &i=ptr[u];i<adj[u].size();i++){
        int v=adj[u][i];
        if(match[v]==-1|| (lv[match[v]]==lv[u]+1&&dfs(match[v]))){
            match[u]=v,match[v]=u;
            return true;
        }
    }
    return false;
}

int max_matching(){
    int ans=0,cnt=0;
    match.assign(n+m,-1);
    do{
        ptr.assign(n,0);
        bfs();
        cnt=0;
        for(int i=0;i<n;i++)if(match[i]==-1&&dfs(i))cnt++;
        ans+=cnt;
    }while(cnt);
    return ans;
}

};
```

hungarian-algorithm.hpp  
Description: Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes cost[N][M], where cost[i][j] = cost for L[i] to be matched with R[j] and returns (min cost, match), where L[i] is matched with R[match[i]]. Negate costs for max cost. Requires  $N \leq M$ .  
Time:  $\mathcal{O}(N^2M)$

2540b8, 34 lines

```
pair<ll, vector<int>>> hungarian(const vector<vector<ll>>& a) {
    if (a.empty()) return {0, {}};
    int n = a.size() + 1, m = a[0].size() + 1;
    vector<ll> u(n), v(m);
    vector<int> p(m), ans(n - 1);
    for(int i=1;i<n;i++) {
        p[0] = i;
        int j0 = 0; // add "dummy" worker 0
        vector<ll> dist(m, LLONG_MAX);
        vector<int> pre(m, -1);
        vector<bool> done(m + 1);
        do { // dijkstra
            done[j0] = true;
            int i0 = p[j0], j1;
            ll delta = LLONG_MAX;
            for(int j=1;j<m;j++) if (!done[j]) {
                auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
                if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
                if (dist[j] < delta) delta = dist[j], j1 = j;
            }
            for(int j=0;j<m;j++) {
```

```
        if (done[j]) u[p[j]] += delta, v[j] -= delta;
        else dist[j] -= delta;
    }
    j0 = j1;
} while (p[j0]);
while (j0) { // update alternating path
    int j1 = pre[j0];
    p[j0] = p[j1], j0 = j1;
}
}

for(int j=1;j<m;j++) if (p[j]) ans[p[j] - 1] = j - 1;
return {-v[0], ans}; // min cost
}
```

general-matching.hpp  
Description: General matching using blossom algorithm.  
Time:  $\mathcal{O}\left(E\sqrt{V}\log_{\max(2,1+E/V)}\right)$ .

965487, 90 lines

```
struct GeneralMatching{
    int n;
    vector<vector<int>>> adj;
    vector<int> match,vis,link,fa,dep;
    queue<int> q;
    GeneralMatching(int n):n(n),adj(n),match(n,-1),vis(n),link(n),fa(n),dep(n){}
    void add_edge(int u,int v){
        adj[u].emplace_back(v);
        adj[v].emplace_back(u);
    }
    int fp(int u){
        while(fa[u]!=u)u=fa[u]=fa[fa[u]];
        return u;
    }
    int lca(int u,int v){
        u=fp(u),v=fp(v);
        while(u!=v){
            if(dep[u]<dep[v])swap(u,v);
            u=fp(link[match[u]]);
        }
        return u;
    }
    void blossom(int u,int v,int p){
        while(fp(u)!=p){
            link[u]=v;
            v=match[u];
            if(vis[v]==0){
                vis[v]=1;
                q.emplace(v);
            }
            fa[u]=fa[v]=p;
            u=link[v];
        }
    }
    void augment(int u){
        while(!q.empty())q.pop();
        iota(fa.begin(),fa.end(),0);
        fill(vis.begin(),vis.end(),-1);
        q.emplace(u);
        vis[u]=1;
        dep[u]=0;
        while(!q.empty()){
            int u=q.front();
            q.pop();
            for(auto v:adj[u]){
                if(vis[v]==-1){
                    vis[v]=0;
                    link[v]=u;
                    dep[v]=dep[u]+1;
                    if(match[v]==-1){
```

```
                        for(int x=v,y=u,z;y!=-1;x=z,y=x==-1?-1:link[x]){
                            link[x]={
                                z=match[y];
                                match[x]=y;
                                match[y]=x;
                            }
                            return 1;
                        }
                        vis[match[v]]=1;
                        dep[match[v]]=dep[u]+2;
                        q.emplace(match[v]);
                    }else if(vis[v]==1&&fp(u)!=fp(v)){
                        int p=lca(u,v);
                        blossom(u,v,p);
                        blossom(v,u,p);
                    }
                }
            }
        }
        return 0;
    }
    int max_matching(){
        int res=0;
        for(int u=0;u<n;u++){
            if(match[u]==-1)continue;
            for(auto v:adj[u]){
                if(match[v]==-1){
                    match[u]=v;
                    match[v]=u;
                    res++;
                    break;
                }
            }
        }
        for(int u=0;u<n;u++){
            if(match[u]==-1){
                res+=augment(u);
            }
        }
        return res;
    }
};
```

## Geometry (8)

### 8.1 Geometric primitives

Point.h  
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

47ec0a, 28 lines

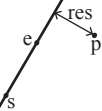
```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
    typedef Point P;
    T x, y;
    explicit Point(T x=0, T y=0) : x(x), y(y) {}
    bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
    bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
    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)); }
friend ostream& operator<<(ostream& os, P p) {
    return os << "(" << p.x << ", " << p.y << ")"; }
};
```

lineDistance.h

Description:

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. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D, call .dist on the result of the cross product.



"src/geometry/Point.h"

f6bf6b, 4 lines

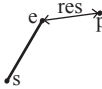
```
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
    return (double) (b-a).cross(p-a)/(b-a).dist();
}
```

SegmentDistance.h

Description:

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

Usage: Point<double> a, b(2,2), p(1,1);  
bool onSegment = segDist(a,b,p) < 1e-10;



"src/geometry/Point.h"

5c88f4, 6 lines

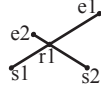
```
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
    if (s==e) return (p-s).dist();
    auto d = (e-s).dist2(), t = min(d,max(.0, (p-s).dot(e-s)));
    return ((p-s)*d-(e-s)*t).dist()/d;
}
```

SegmentIntersection.h

Description:

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. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.

Usage: vector<P> inter = segInter(s1,e1,s2,e2);  
if (sz(inter)==1)  
cout << "segments intersect at " << inter[0] << endl;



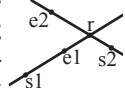
"src/geometry/Point.h", "src/geometry/OnSegment.h"

9d57f2, 13 lines

```
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 a 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)};
```

```
}

lineIntersection.h
Description:
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. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.
Usage: auto res = lineInter(s1,e1,s2,e2);
if (res.first == 1)
cout << "intersection point at " << res.second << endl;
```



"src/geometry/Point.h"

a01f81, 8 lines

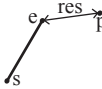
```
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
    auto d = (e1 - s1).cross(e2 - s2);
    if (d == 0) // if parallel
        return {-(s1.cross(e1, s2) == 0), P(0, 0)};
    auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
    return {1, (s1 * p + e1 * q) / d};
}
```

sideOf.h

Description:

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. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.

Usage: bool left = sideOf(p1,p2,q)==1;



"src/geometry/Point.h"

3af81c, 9 lines

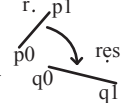
```
template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

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);
}
```

OnSegment.h

Description:

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



"src/geometry/Point.h"

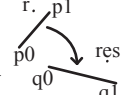
c597e8, 3 lines

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

linearTransformation.h

Description:

Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.



"src/geometry/Point.h"

03a306, 6 lines

```
typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
    const P& q0, const P& q1, const P& r) {
    P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
    return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
}
```

Angle.h

Description:

A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.

```
Usage: vector<Angle> v = {w[0], w[0].t360() ...}; // sorted
int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }
// sweeps j such that (j-i) represents the number of positively
oriented triangles with vertices at 0 and i
```

"src/geometry/Point.h"

0f0602, 35 lines

```
struct Angle {
    int x, y;
    int t;
    Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
    Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
    int half() const {
        assert(x || y);
        return y < 0 || (y == 0 && x < 0);
    }
    Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
    Angle t180() const { return {-x, -y, t + half()}; }
    Angle t360() const { return {x, y, t + 1}; }
};
```

```
bool operator<(Angle a, Angle b) {
    // add a.dist2() and b.dist2() to also compare distances
    return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
        make_tuple(b.t, b.half(), a.x * (ll)b.y);
}
```

```
// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
    if (b < a) swap(a, b);
    return (b < a.t180() ?
        make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
    Angle r(a.x + b.x, a.y + b.y, a.t);
    if (a.t180() < r) r.t--;
    return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
    int tu = b.t - a.t; a.t = b.t;
    return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

## 8.2 Circles

CircleIntersection.h

Description:

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

```
"src/geometry/Point.h" 84d6d3, 11 lines

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;
}
```

CircleTangents.h

Description:

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.

```
"src/geometry/Point.h" b0153d, 13 lines
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;
}
```

### CirclePolygonIntersection.h

**Description:** Returns the area of the intersection of a circle with a ccw polygon.

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

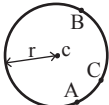
"src/geometry/Point.h"	19add1, 19 lines
------------------------	------------------

```
typedef Point<double> P;
#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;
}
```

### circumcircle.h

**Description:**

The circumcircle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.



"src/geometry/Point.h"	1caa3a, 9 lines
------------------------	-----------------

```
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
    return (B-A).dist()*(C-B).dist()*(A-C).dist()/
        abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
    P b = C-A, c = B-A;
    return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

### MinimumEnclosingCircle.h

**Description:** Computes the minimum circle that encloses a set of points.

**Time:** expected  $\mathcal{O}(n)$

"circumcircle.h"	09dd0a, 17 lines
------------------	------------------

```
pair<P, double> mec(vector<P> ps) {
    shuffle(all(ps), mt19937(time(0)));
    P o = ps[0];
    double r = 0, EPS = 1 + 1e-8;
    rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
        o = ps[i], r = 0;
        rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
            o = (ps[i] + ps[j]) / 2;

```

```
        r = (o - ps[i]).dist();
        rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
            o = ccCenter(ps[i], ps[j], ps[k]);
            r = (o - ps[i]).dist();
        }
    }
    return {o, r};
}
```

## 8.3 Polygons

### InsidePolygon.h

**Description:** Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.

**Usage:** vector<P> v = {P{4,4}, P{1,2}, P{2,1}};  
bool in = inPolygon(v, P{3, 3}, false);  
**Time:**  $\mathcal{O}(n)$

"src/geometry/Point.h", "src/geometry/OnSegment.h", "src/geometry/SegmentDistance.h"	2bf504, 11 lines
--------------------------------------------------------------------------------------	------------------

```
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;
}
```

### PolygonArea.h

**Description:** Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

"src/geometry/Point.h"	f12300, 6 lines
------------------------	-----------------

```
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;
}
```

### PolygonCenter.h

**Description:** Returns the center of mass for a polygon.

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

"src/geometry/Point.h"	9706dc, 9 lines
------------------------	-----------------

```
typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
    P res(0, 0); double A = 0;
    for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
        res = res + (v[i] + v[j]) * v[j].cross(v[i]);
        A += v[j].cross(v[i]);
    }
    return res / A / 3;
}
```

### PolygonCut.h

**Description:**

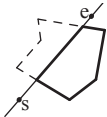
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.

**Usage:** vector<P> p = ...;  
p = polygonCut(p, P(0,0), P(1,0));

"src/geometry/Point.h"	d07181, 13 lines
------------------------	------------------

```
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;
}
```

### ConvexHull.h

**Description:**

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:**  $\mathcal{O}(n \log n)$

"src/geometry/Point.h"	310954, 13 lines
------------------------	------------------

```
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])};
}
```



### HullDiameter.h

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

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

"src/geometry/Point.h"	c571b8, 12 lines
------------------------	------------------

```
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;
}
```

### PointInsideHull.h

**Description:** 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.

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

"src/geometry/Point.h", "src/geometry/sideOf.h", "src/geometry/OnSegment.h"	71446b, 14 lines
-----------------------------------------------------------------------------	------------------

```
typedef Point<ll> P;

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;
}
```

LineHullIntersection.h

**Description:** Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon:  $\bullet(-1, -1)$  if no collision,  $\bullet(i, -1)$  if touching the corner  $i$ ,  $\bullet(i, i)$  if along side  $(i, i+1)$ ,  $\bullet(i, j)$  if crossing sides  $(i, i+1)$  and  $(j, j+1)$ . In the last case, if a corner  $i$  is crossed, this is treated as happening on side  $(i, i+1)$ . The points are returned in the same order as the line hits the polygon. extrVertex returns the point of a hull with the max projection onto a line.

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

"src/geometry/Point.h"	7cf45b, 39 lines
------------------------	------------------

```
#define cmp(i, j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
    int n = sz(poly), lo = 0, hi = n;
    if (extr(0)) return 0;
    while (lo + 1 < hi) {
        int m = (lo + hi) / 2;
        if (extr(m)) return m;
        int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
        (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
    }
    return lo;
}

#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
    int endA = extrVertex(poly, (a - b).perp());
    int endB = extrVertex(poly, (b - a).perp());
    if (cmpL(endA) < 0 || cmpL(endB) > 0)
        return {-1, -1};
    array<int, 2> res;
    rep(i, 0, 2) {
        int lo = endB, hi = endA, n = sz(poly);
        while ((lo + 1) % n != hi) {
            int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
            (cmpL(m) == cmpL(endB) ? lo : hi) = m;
        }
        res[i] = (lo + !cmpL(hi)) % n;
        swap(endA, endB);
    }
    if (res[0] == res[1]) return {res[0], -1};
    if (!cmpL(res[0]) && !cmpL(res[1]))
        switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
            case 0: return {res[0], res[0]};
            case 2: return {res[1], res[1]};
        }
    return res;
}
```

8.4 Misc. Point Set Problems

ClosestPair.h

**Description:** Finds the closest pair of points.

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

"src/geometry/Point.h"	ac41a6, 17 lines
------------------------	------------------

```
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;
}
```

kdTree.h

**Description:** KD-tree (2d, can be extended to 3d)

"src/geometry/Point.h"	bac5b0, 63 lines
------------------------	------------------

```
typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();
```

```
bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }
```

```
struct Node {
    P pt; // if this is a leaf, the single point in it
    T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
    Node *first = 0, *second = 0;

    T distance(const P& p) { // min squared distance to a point
        T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
        T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
        return (P(x, y) - p).dist2();
    }
}
```

```
Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
        x0 = min(x0, p.x); x1 = max(x1, p.x);
        y0 = min(y0, p.y); y1 = max(y1, p.y);
    }
    if (vp.size() > 1) {
        // split on x if width >= height (not ideal...)
        sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
        // divide by taking half the array for each child (not
        // best performance with many duplicates in the middle)
        int half = sz(vp)/2;
        first = new Node({vp.begin(), vp.begin() + half});
        second = new Node({vp.begin() + half, vp.end()});
    }
}
```

```
struct KDTree {
    Node* root;
    KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}
}
```

```
pair<T, P> search(Node *node, const P& p) {
    if (!node->first) {
        // uncomment if we should not find the point itself:
        // if (p == node->pt) return {INF, P()};
        return make_pair((p - node->pt).dist2(), node->pt);
    }
}
```

```
Node *f = node->first, *s = node->second;
T bfirst = f->distance(p), bsec = s->distance(p);
if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

// search closest side first, other side if needed
auto best = search(f, p);
if (bsec < best.first)
    best = min(best, search(s, p));
return best;
```

```
}

// find nearest point to a point, and its squared distance
// (requires an arbitrary operator< for Point)
pair<T, P> nearest(const P& p) {
    return search(root, p);
}
};
```

FastDelaunay.h

**Description:** Fast Delaunay triangulation. Each circumcircle contains none of the input points. There must be no duplicate points. If all points are on a line, no triangles will be returned. Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in order  $\{t[0][0], t[0][1], t[0][2], t[1][0], \dots\}$ , all counter-clockwise.

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

"src/geometry/Point.h"	eefdf5, 88 lines
------------------------	------------------

```
typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t ll1; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX, LLONG_MAX); // not equal to any other point
```

```
struct Quad {
    Q rot, o; P p = arb; bool mark;
    P& F() { return r()->p; }
    Q& r() { return rot->rot; }
    Q prev() { return rot->o->rot; }
    Q next() { return r()->prev(); }
} *H;
```

```
bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
    ll1 p2 = p.dist2(), A = a.dist2()-p2,
        B = b.dist2()-p2, C = c.dist2()-p2;
    return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}
```

```
Q makeEdge(P orig, P dest) {
    Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
    H = r->o; r->r()->r() = r;
    rep(i, 0, 4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
    r->p = orig; r->F() = dest;
    return r;
}
```

```
void splice(Q a, Q b) {
    swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
```

```
Q connect(Q a, Q b) {
    Q q = makeEdge(a->F(), b->p);
    splice(q, a->next());
    splice(q->r(), b);
    return q;
}
```

```
pair<Q, Q> rec(const vector<P>& s) {
    if (sz(s) <= 3) {
        Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
        if (sz(s) == 2) return {a, a->r()};
        splice(a->r(), b);
        auto side = s[0].cross(s[1], s[2]);
        Q c = side ? connect(b, a) : 0;
        return {side < 0 ? c->r() : a, side < 0 ? c : b->r()};
    }
}
```

```
#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
Q A, B, ra, rb;
int half = sz(s) / 2;
tie(ra, A) = rec({all(s) - half});
tie(B, rb) = rec({sz(s) - half + all(s)});
while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
```

```
(A->p.cross(H(B)) > 0 && (B = B->r()->o)));
Q base = connect(B->r(), A);
if (A->p == ra->p) ra = base->r();
if (B->p == rb->p) rb = base;

#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
while (circ(e->dir->F(), H(base), e->F())) { \
    Q t = e->dir; \
    splice(e, e->prev()); \
    splice(e->r(), e->r()->prev()); \
    e->o = H; H = e; e = t; \
}
for (;;) {
    DEL(LC, base->r(), o); DEL(RC, base, prev());
    if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
        base = connect(RC, base->r());
    else
        base = connect(base->r(), LC->r());
}
return { ra, rb };
}
```

```
vector<P> triangulate(vector<P> pts) {
    sort(all(pts)); assert(unique(all(pts)) == pts.end());
    if (sz(pts) < 2) return {};
    Q e = rec(pts).first;
    vector<Q> q = {e};
    int qi = 0;
    while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
q.push_back(c->r()); c = c->next(); } while (c != e); }
    ADD; pts.clear();
    while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
    return pts;
}
```

8.5 3D

PolyhedronVolume.h

Description: Magic formula for the volume of a polyhedron. Faces should point outwards.

3058c3, 6 lines

```
template<class V, class L>
double signedPolyVolume(const V& p, const L& trilst) {
    double v = 0;
    for (auto i : trilst) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
    return v / 6;
}
```

Point3D.h

Description: Class to handle points in 3D space. T can be e.g. double or long long.

8058ae, 32 lines

```
template<class T> struct Point3D {
    typedef Point3D P;
    typedef const P& R;
    T x, y, z;
    explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
    bool operator<(R p) const {
        return tie(x, y, z) < tie(p.x, p.y, p.z); }
    bool operator==(R p) const {
        return tie(x, y, z) == tie(p.x, p.y, p.z); }
    P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
    P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
    P operator*(T d) const { return P(x*d, y*d, z*d); }
    P operator/(T d) const { return P(x/d, y/d, z/d); }
    T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
    P cross(R p) const {
        return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
    }
};
```

```
}
T dist2() const { return x*x + y*y + z*z; }
double dist() const { return sqrt((double)dist2()); }
//Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
double phi() const { return atan2(y, x); }
//Zenith angle (latitude) to the z-axis in interval [0, pi]
double theta() const { return atan2(sqrt(x*x+y*y), z); }
P unit() const { return *this/(T)dist(); } //makes dist()==1
//returns unit vector normal to *this and p
P normal(P p) const { return cross(p).unit(); }
//returns point rotated 'angle' radians ccw around axis
P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle); P u = axis.unit();
    return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
}
}
```

3dHull.h

Description: Computes all faces of the 3-dimension hull of a point set. \*No four points must be coplanar\*, or else random results will be returned. All faces will point outwards.

Time:  $\mathcal{O}(n^2)$

"Point3D.h" 5b45fc, 49 lines

```
typedef Point3D<double> P3;

struct PR {
    void ins(int x) { (a == -1 ? a : b) = x; }
    void rem(int x) { (a == x ? a : b) = -1; }
    int cnt() { return (a != -1) + (b != -1); }
    int a, b;
};
```

struct F { P3 q; int a, b, c; };

```
vector<F> hull3d(const vector<P3>& A) {
    assert(sz(A) >= 4);
    vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
    vector<F> FS;
    auto mf = [&](int i, int j, int k, int l) {
        P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
        if (q.dot(A[l]) > q.dot(A[i]))
            q = q * -1;
        F f{q, i, j, k};
        E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
        FS.push_back(f);
    };
    rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
        mf(i, j, k, 6 - i - j - k);
```

```
    rep(i,4,sz(A)) {
        rep(j,0,sz(FS)) {
            F f = FS[j];
            if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
                E(a,b).rem(f.c);
                E(a,c).rem(f.b);
                E(b,c).rem(f.a);
                swap(FS[j--], FS.back());
                FS.pop_back();
            }
        }
        int nw = sz(FS);
        rep(j,0,nw) {
            F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
            C(a, b, c); C(a, c, b); C(b, c, a);
        }
        for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
```

```
    A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
    return FS;
};
```

sphericalDistance.h

Description: Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 ( $\phi_1$ ) and f2 ( $\phi_2$ ) from x axis and zenith angles (latitude) t1 ( $\theta_1$ ) and t2 ( $\theta_2$ ) from z axis (0 = north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. dx\*radius is then the difference between the two points in the x direction and d\*radius is the total distance between the points.

611f07, 8 lines

```
double sphericalDistance(double f1, double t1,
    double f2, double t2, double radius) {
    double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
    double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
    double dz = cos(t2) - cos(t1);
    double d = sqrt(dx*dx + dy*dy + dz*dz);
    return radius*2*asin(d/2);
}
```

String (9)

suffix-array.hpp

Description: Suffix Array.

58c0a5, 39 lines

```
template<class STR>
struct SuffixArray{
    int n;
    vector<int> sa,isa,lcp;
    // SparseTable<MinMonoid<int>> st;
    SuffixArray(){}
    SuffixArray(const STR &s){init(s);}
    void init(const STR &s){
        n=(int)s.size();
        sa=isa=lcp=vector<int>(n+1);
        sa[0]=n;
        iota(sa.begin()+1,sa.end(),0);
        sort(sa.begin()+1,sa.end(),[&](int i,int j){return s[i]<s[j];});
        for(int i=1;i<=n;i++){
            int x=sa[i-1],y=sa[i];
            isa[y]=i>1&&s[x]==s[y]?isa[x]:i;
        }
        for(int len=1;len<=n;len<=1){
            vector<int> ps(sa),pi(isa),pos(n+1);
            iota(pos.begin(),pos.end(),0);
            for(auto i:ps)if((i-=len)>=0)sa[pos[isa[i]]++]=i;
            for(int i=1;i<=n;i++){
                int x=sa[i-1],y=sa[i];
                isa[y]=pi[x]==pi[y]&&pi[x+len]==pi[y+len]?isa[x]:i;
            }
        }
        for(int i=0,k=0;i<n;i++){
            for(int j=sa[isa[i]-1];j+k<n&&s[j+k]==s[i+k];k++){
                lcp[isa[i]]=k;
                if(k)k--;
            }
            // st.init(lcp);
        }
        // int get_lcp(int i,int j){
        //     if(i==j)return n-i;
        //     auto [l,r]=minmax(isa[i],isa[j]);
        //     return st.query(l+1,r);
        // }
```

```
};

suffix-automaton.hpp
Description: Suffix Automaton.
37a4fa, 47 lines

template<class STR>
struct SuffixAutomaton{
    using T = typename STR::value_type;
    struct Node{
        map<T,int> nxt;
        int link,len;
        Node(int link,int len):link(link),len(len){}
    };
    vector<Node> nodes;
    int last;
    SuffixAutomaton():nodes{Node(-1,0)},last(0){}
    SuffixAutomaton(const STR &s):SuffixAutomaton(){
        for(auto c:s)extend(c);
    }
    int new_node(int link,int len){
        nodes.emplace_back(Node(link,len));
        return (int)nodes.size()-1;
    }
    void extend(T c){
        int cur=new_node(0,nodes[last].len+1);
        int p=last;
        while(p!=-1&&!nodes[p].nxt.count(c)){
            nodes[p].nxt[c]=cur;
            p=nodes[p].link;
        }
        if(p!=-1){
            int q=nodes[p].nxt[c];
            if(nodes[p].len+1==nodes[q].len){
                nodes[cur].link=q;
            }else{
                int r=new_node(nodes[q].link,nodes[p].len+1);
                nodes[r].nxt=nodes[q].nxt;
                while(p!=-1&&nodes[p].nxt[c]==q){
                    nodes[p].nxt[c]=r;
                    p=nodes[p].link;
                }
                nodes[q].link=nodes[cur].link=r;
            }
        }
        last=cur;
    }
    ll distinct_substrings(){
        ll res=0;
        for(int i=1;i<(int)nodes.size();i++)res+=nodes[i].len-
            nodes[nodes[i].link].len;
        return res;
    }
};
```

```
aho-corasick.hpp
Description: Aho-Corasick.
f2b759, 54 lines

template<class T>
struct AhoCorasick{
    struct Node{
        array<int,26> ch;
        int fail;
        T val;
        Node(){
            fill(ch.begin(),ch.end(),-1);
            fail=-1;
            val=0;
        }
    };
};
```

```
vector<Node> nodes;
AhoCorasick(){new_node();}
int new_node(){
    nodes.emplace_back(Node());
    return nodes.size()-1;
}
void insert(const string &s,const T &val){
    int u=0;
    for(auto x:s){
        int c=x-'a';
        if(nodes[u].ch[c]==-1)nodes[u].ch[c]=new_node();
        u=nodes[u].ch[c];
    }
    nodes[u].val+=val;
}
void build(){
    vector<int> q{0};
    for(int i=0;i<q.size();i++){
        int u=q[i];
        int v;
        for(int c=0;c<26;c++){
            if((v=nodes[u].ch[c])!=-1){
                int p=nodes[u].fail;
                while(p!=-1&&nodes[p].ch[c]==-1)p=nodes[p].fail;
                p=p!=-1?nodes[p].ch[c]:0;
                nodes[v].fail=p;
                nodes[v].val+=nodes[p].val;
                q.emplace_back(v);
            }
        }
    }
    for(auto u:q){
        for(int c=0;c<26;c++){
            if(nodes[u].ch[c]==-1){
                int p=nodes[u].fail;
                while(p!=-1&&nodes[p].ch[c]==-1)p=nodes[p].fail;
                nodes[u].ch[c]=p!=-1?nodes[p].ch[c]:0;
            }
        }
    }
};
```

```
z-algorithm.hpp
Description: Z Algorithm. z[i] := the length of the longest common prefix
between s and s[i].
b93726, 12 lines
```

```
template<class STR>
vector<int> z_algorithm(const STR &s){
    int n=(int)s.size();
    vector<int> z(n);
    z[0]=n;
    for(int i=1,l=0,r=1;i<n;i++){
        if(i<r)z[i]=min(r-i,z[i-1]);
        while(i+z[i]<n&&s[z[i]]==s[i+z[i]])z[i]++;
        if(i+z[i]>r)l=i,r=i+z[i];
    }
    return z;
}
```

```
prefix-function.hpp
Description: Prefix function. pi[i] := the length of the longest proper prefix
of s[0:i] which is also a suffix of s[0:i].
3d65fe, 11 lines
```

```
template<class STR>
vector<int> prefix_function(const STR &s){
    int n=(int)s.size();
```

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

```
manacher.hpp
Description: Manacher's Algorithm. pal[i] := the length of the longest
palindrome centered at i/2.
53856e, 15 lines
```

```
template<class STR>
vector<int> manacher(const STR &s){
    int n=(int)s.size();
    if(n==0)return {};
    vector<int> pal(2*n-1);
    for(int p=0,l=-1,r=-1;p<2*n-1;p++){
        int i=(p+1)>>1,j=p>>1;
        int k=(i>=r?0:min(r-i,pal[2*(l+r)-p]));
        while(j+k+1<n&&i-k-1>=0&&s[j+k+1]==s[i-k-1])k++;
        pal[p]=k;
        if(j+k>r)l=i-k,r=j+k;
    }
    for(int i=0;i<2*n-1;i++)pal[i]=pal[i]<<1|(i&1^1);
    return pal;
}
```

Convolutions (10)

```
and-convolution.hpp
Description: Bitwise AND Convolution. Superset Zeta Transform: A'[S] =
Σ_{T⊇S} A[T]. Superset Mobius Transform: A[T] = Σ_{S⊇T} (-1)^{|S-T|} A'[S].
Time: O(N log N).
7916f8, 34 lines
```

```
template<class T>
void superset_zeta(vector<T> &a){
    int n=(int)a.size();
    assert(n==(n&-n));
    for(int i=1;i<n;i<=(1){
        for(int j=0;j<n;j++){
            if(j&i){
                a[j^i]+=a[j];
            }
        }
    }
}
```

```
template<class T>
void superset_mobius(vector<T> &a){
    int n=(int)a.size();
    assert(n==(n&-n));
    for(int i=n;i>=1;i){
        for(int j=0;j<n;j++){
            if(j&i){
                a[j^i]-=a[j];
            }
        }
    }
}
```

```
template<class T>
vector<T> and_convolution(vector<T> a,vector<T> b){
    superset_zeta(a);
    superset_zeta(b);
    for(int i=0;i<(int)a.size();i++)a[i]*=b[i];
    superset_mobius(a);
```



```
    return a;
}
```

### or-convolution.hpp

**Description:** Bitwise OR Convolution. Subset Zeta Transform:  $A'[S] = \sum_{T \subseteq S} A[T]$ . Subset Mobius Transform:  $A[T] = \sum_{S \subseteq T} (-1)^{|T-S|} A'[S]$ .  
**Time:**  $\mathcal{O}(N \log N)$ .

c58b77, 34 lines

```
template<class T>
void subset_zeta(vector<T> &a) {
    int n=(int)a.size();
    assert(n==(n&-n));
    for(int i=1; i<n; i<=<1) {
        for(int j=0; j<n; j++) {
            if(j&i) {
                a[j]+=a[j^i];
            }
        }
    }
}

template<class T>
void subset_mobius(vector<T> &a) {
    int n=(int)a.size();
    assert(n==(n&-n));
    for(int i=n; i>=>=1) {
        for(int j=0; j<n; j++) {
            if(j&i) {
                a[j]-=a[j^i];
            }
        }
    }
}
```

```
template<class T>
vector<T> or_convolution(vector<T> a, vector<T> b) {
    subset_zeta(a);
    subset_zeta(b);
    for(int i=0; i<(int)a.size(); i++) a[i]*=b[i];
    subset_mobius(a);
    return a;
}
```

### xor-convolution.hpp

**Description:** Bitwise XOR Convolution. Fast Walsh-Hadamard Transform:  $A'[S] = \sum_{T} (-1)^{|S \& T|} A[T]$ .  
**Time:**  $\mathcal{O}(N \log N)$ .

05848d, 29 lines

```
template<class T>
void fwht(vector<T> &a) {
    int n=(int)a.size();
    assert(n==(n&-n));
    for(int i=1; i<n; i<=<1) {
        for(int j=0; j<n; j++) {
            if(j&i) {
                T &u=a[j^i], &v=a[j];
                tie(u, v)=make_pair(u+v, u-v);
            }
        }
    }
}

template<class T>
vector<T> xor_convolution(vector<T> a, vector<T> b) {
    int n=(int)a.size();
    fwht(a);
    fwht(b);
    for(int i=0; i<n; i++) a[i]*=b[i];
    fwht(a);
}
```

```
T div=T(1)/T(n);
if(div==T(0)) {
    for(auto &x:a) x/=n;
}else{
    for(auto &x:a) x*=div;
}
return a;
}
```

### gcd-convolution.hpp

**Description:** GCD Convolution. Multiple Zeta Transform:  $A'[n] = \sum_{n|m} A[m]$ . Multiple Mobius Transform:  $A[n] = \sum_{n|m} \mu(m/n) A'[m]$ .  
**Time:**  $\mathcal{O}(N \log \log N)$ .

7f6c2d, 34 lines

```
template<class T>
void multiple_zeta(vector<T> &a) {
    int n=(int)a.size();
    vector<bool> is_prime(n, true);
    for(int p=2; p<n; p++) {
        if(!is_prime[p]) continue;
        for(int i=(n-1)/p; i>=1; i--) {
            is_prime[i*p]=false;
            a[i]+=a[i*p];
        }
    }
}

template<class T>
void multiple_mobius(vector<T> &a) {
    int n=(int)a.size();
    vector<bool> is_prime(n, true);
    for(int p=2; p<n; p++) {
        if(!is_prime[p]) continue;
        for(int i=1; i*p<n; i++) {
            is_prime[i*p]=false;
            a[i]-=a[i*p];
        }
    }
}
```

```
template<class T>
vector<T> gcd_convolution(vector<T> a, vector<T> b) {
    multiple_zeta(a);
    multiple_zeta(b);
    for(int i=0; i<(int)a.size(); i++) a[i]*=b[i];
    multiple_mobius(a);
    return a;
}
```

### lcm-convolution.hpp

**Description:** LCM Convolution. Divisor Zeta Transform:  $A'[n] = \sum_{d|n} A[d]$ . Divisor Mobius Transform:  $A[n] = \sum_{d|n} \mu(n/d) A'[d]$ .  
**Time:**  $\mathcal{O}(N \log \log N)$ .

41fe9d, 34 lines

```
template<class T>
void divisor_zeta(vector<T> &a) {
    int n=(int)a.size();
    vector<bool> is_prime(n, true);
    for(int p=2; p<n; p++) {
        if(!is_prime[p]) continue;
        for(int i=1; i*p<n; i++) {
            is_prime[i*p]=false;
            a[i*p]+=a[i];
        }
    }
}

template<class T>
void divisor_mobius(vector<T> &a) {
}
```

```
int n=(int)a.size();
vector<bool> is_prime(n, true);
for(int p=2; p<n; p++) {
    if(!is_prime[p]) continue;
    for(int i=(n-1)/p; i>=1; i--) {
        is_prime[i*p]=false;
        a[i*p]-=a[i];
    }
}
```

```
template<class T>
vector<T> lcm_convolution(vector<T> a, vector<T> b) {
    divisor_zeta(a);
    divisor_zeta(b);
    for(int i=0; i<(int)a.size(); i++) a[i]*=b[i];
    divisor_mobius(a);
    return a;
}
```

### max-plus-convolution.hpp

**Description:** Max Plus Convolution. Find  $C[k] = \max_{i+j=k} \{A[i] + B[j]\}$ .  
**Time:**  $\mathcal{O}(N)$ .

7176a2, 94 lines

```
// SMAWCK algorithm for finding row-wise maxima.
// f(i, j, k) checks if M[i][j] <= M[i][k].
// f(i, j, k) checks if M[i][k] is at least as good as M[i][j].
// higher is better.
template<class F>
vector<int> smawck(const F &f, const vector<int> &rows, const
    vector<int> &cols) {
    int n=(int)rows.size(), m=(int)cols.size();
    if(max(n, m)<=2) {
        vector<int> ans(n, -1);
        for(int i=0; i<n; i++) {
            for(int j:cols) {
                if(ans[i]==-1 || f(rows[i], ans[i], j)) {
                    ans[i]=j;
                }
            }
        }
        return ans;
    }
    if(n<m) {
        // reduce
        vector<int> st;
        for(int j:cols) {
            while(true) {
                if(st.empty()) {
                    st.emplace_back(j);
                    break;
                } else if(f(rows[(int)st.size()-1], st.back(), j)) {
                    st.pop_back();
                } else if(st.size()<n) {
                    st.emplace_back(j);
                    break;
                } else {
                    break;
                }
            }
        }
        return smawck(f, rows, st);
    }
    vector<int> ans(n, -1);
    vector<int> new_rows;
    for(int i=1; i<n; i+=2) {
        new_rows.emplace_back(rows[i]);
    }
}
```

```
auto res=smawck(f,new_rows,cols);
for(int i=0;i<new_rows.size();i++){
    ans[2*i+1]=res[i];
}
for(int i=0,l=0,r=0;i<n;i+=2){
    if(i+1==n)r=m;
    while(r<m&&cols[r]<=ans[i+1])r++;
    ans[i]=cols[l++];
    for(;l<r;l++){
        if(f(rows[i],ans[i],cols[l])){
            ans[i]=cols[l];
        }
    }
    l--;
}
return ans;
}
```

```
template<class F>
vector<int> smawck(const F &f,int n,int m){
    vector<int> rows(n),cols(m);
    iota(rows.begin(),rows.end(),0);
    iota(cols.begin(),cols.end(),0);
    return smawck(f,rows,cols);
}

// Max Plus Convolution.
// b must be convex, i.e. b[i]-b[i-1]>=b[i+1]-b[i].
template<class T>
vector<T> max_plus_convolution_arbitrary_convex(vector<T> a,
const vector<T> &b){
    if(a.empty()||b.empty())return {};
    if((int)b.size()==1){
        for(auto &x:a)x+=b[0];
        return a;
    }
    int n=(int)a.size(),m=(int)b.size();
    auto f=[&](int i,int j){
        return a[j]+b[i-j];
    };
    auto cmp=[&](int i,int j,int k){
        if(i<k)return false;
        if(i-j>=m)return true;
        return f(i,j)<=f(i,k);
    };
    auto best=smawck(cmp,n+m-1,n);
    vector<T> ans(n+m-1);
    for(int i=0;i<n+m-1;i++){
        ans[i]=f(i,best[i]);
    }
    return ans;
}
```

Polynomials (11)

11.1 Generating Functions

11.1.1 Common Series

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \cdots = \sum_{n=0}^{\infty} x^n$$
$$-\ln(1-x) = x + \frac{x^2}{2} + \frac{x^3}{3} + \cdots = \sum_{n=1}^{\infty} \frac{x^n}{n}$$

formal-power-series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$
$$(1-x)^{-k} = \sum_{n=0}^{\infty} \binom{n+k-1}{k-1} x^n$$

11.1.2 Famous Polynomials

Bell numbers  $B_n$  are the number of partitions of  $n$  distinct objects. let  $B(x)$  be EGF of Bell numbers, then

$$B(x) = e^{e^x-1} = \sum_{n=0}^{\infty} B_n \frac{x^n}{n!}$$

11.2 Newton’s Method

if  $F(Q) = 0$ , then  $Q_{2n} \equiv Q_n - \frac{F(Q_n)}{F'(Q_n)} \pmod{x^{2n}}$

$$Q = P^{-1} : Q_{2n} \equiv Q_n \cdot (2 - P \cdot Q_n^2) \pmod{x^{2n}}$$
$$Q = \ln P = \int \frac{P'}{P} dx$$
$$Q = e^P : Q_{2n} \equiv Q_n(1 + P - \ln Q_n) \pmod{x^{2n}}$$
$$Q = \sqrt{P} : Q_{2n} \equiv \frac{1}{2}(Q_n + P \cdot Q_n^{-1}) \pmod{x^{2n}}$$
$$Q = P^k = \alpha^k x^{kt} e^{k \ln T}; P = \alpha \cdot x^t \cdot T, T(0) = 1$$

```
formal-power-series.hpp
Description: basic operations of formal power series
"src/polynomials/ntt.hpp" 8679c4, 135 lines

template<class mint>
struct FormalPowerSeries:vector<mint>{
    using vector<mint>::vector;
    using FPS = FormalPowerSeries;

    FPS &operator+=(const FPS &rhs){
        if(rhs.size()>this->size())this->resize(rhs.size());
        for(int i=0;i<rhs.size();i++)(*this)[i]+=rhs[i];
        return *this;
    }
    FPS &operator+=(const mint &rhs){
        if(this->empty())this->resize(1);
        (*this)[0]+=rhs;
        return *this;
    }
    FPS &operator-=(const FPS &rhs){
        if(rhs.size()>this->size())this->resize(rhs.size());
        for(int i=0;i<rhs.size();i++)(*this)[i]-=rhs[i];
        return *this;
    }
    FPS &operator-=(const mint &rhs){
        if(this->empty())this->resize(1);
        (*this)[0]-=rhs;
        return *this;
    }
    FPS &operator*=(const FPS &rhs){
        auto res=NTT<mint>()(*this, rhs);
        return *this=FPS(res.begin(), res.end());
    }
    FPS &operator*=(const mint &rhs){
```

```
    for(auto &a:*this)a*=rhs;
    return *this;
}
friend FPS operator+(FPS lhs,const FPS &rhs){return lhs+=
    rhs;}
friend FPS operator+(FPS lhs,const mint &rhs){return lhs+=
    rhs;}
friend FPS operator+(const mint &lhs,FPS &rhs){return rhs+=
    lhs;}
friend FPS operator-(FPS lhs,const FPS &rhs){return lhs-=
    rhs;}
friend FPS operator-(FPS lhs,const mint &rhs){return lhs-=
    rhs;}
friend FPS operator-(const mint &lhs,FPS rhs){return -(rhs-
    lhs);}
friend FPS operator*(FPS lhs,const FPS &rhs){return lhs*=
    rhs;}
friend FPS operator*(FPS lhs,const mint &rhs){return lhs*=
    rhs;}
friend FPS operator*(const mint &lhs,FPS rhs){return rhs*=
    lhs;}

FPS operator-() {return (*this)*-1;}

FPS rev(){
    FPS res(*this);
    reverse(res.begin(),res.end());
    return res;
}
FPS pre(int sz){
    FPS res(this->begin(),this->begin()+min((int)this->size()
        ,sz));
    if(res.size()<sz)res.resize(sz);
    return res;
}
FPS shrink(){
    FPS res(*this);
    while(!res.empty()&&res.back()==mint{})res.pop_back();
    return res;
}
FPS operator>>(int sz){
    if(this->size()<=sz)return {};
    FPS res(*this);
    res.erase(res.begin(),res.begin()+sz);
    return res;
}
FPS operator<<(int sz){
    FPS res(*this);
    res.insert(res.begin(),sz,mint{});
    return res;
}
FPS diff(){
    const int n=this->size();
    FPS res(max(0,n-1));
    for(int i=1;i<n;i++)res[i-1]=(*this)[i]*mint(i);
    return res;
}
FPS integral(){
    const int n=this->size();
    FPS res(n+1);
    res[0]=0;
    if(n>0)res[1]=1;
    ll mod=mint::get_mod();
    for(int i=2;i<=n;i++)res[i]=(-res[mod%i])*(mod/i);
    for(int i=0;i<n;i++)res[i+1]=(*this)[i];
    return res;
}
mint eval(const mint &x){
    mint res=0,w=1;
```

```
        for(auto &a:*this) res+=a*w,w*=x;
        return res;
    }

FPS inv(int deg=-1){
    assert(!this->empty() && (*this)[0]!=mint(0));
    if(deg==-1) deg=this->size();
    FPS res{mint(1)/(*this)[0]};
    for(int i=2;i>>1<deg;i<=1){
        res=(res*(mint(2)-res*pre(i))).pre(i);
    }
    return res.pre(deg);
}

FPS log(int deg=-1){
    assert(!this->empty() && (*this)[0]==mint(1));
    if(deg==-1) deg=this->size();
    return (pre(deg).diff()*inv(deg)).pre(deg-1).integral();
}

FPS exp(int deg=-1){
    assert(this->empty() || (*this)[0]==mint(0));
    if(deg==-1) deg=this->size();
    FPS res{mint(1)};
    for(int i=2;i>>1<deg;i<=1){
        res=(res*(pre(i)-res.log(i)+mint(1))).pre(i);
    }
    return res.pre(deg);
}

FPS pow(ll k,int deg=-1){
    const int n=this->size();
    if(deg==-1) deg=n;
    if(k==0){
        FPS res(deg);
        if(deg) res[0]=mint(1);
        return res;
    }
    for(int i=0;i<n;i++){
        if(__int128_t(i)*k>deg) return FPS(deg,mint(0));
        if((*this)[i]==mint(0)) continue;
        mint rev=mint(1)/(*this)[i];
        FPS res=(((*this*rev)>>i).log(deg)*k).exp(deg);
        res=(res*binpow((*this)[i],k))<<(i*k)).pre(deg);
        return res;
    }
    return FPS(deg,mint(0));
}
};
```

fft.hpp  
Description: Fast Fourier Transform  
Time:  $\mathcal{O}(N \log N)$   
3b931c, 73 lines

```
template<class T=ll,int mod=0>
struct FFT{
    using vt = vector<T>;
    using cd = complex<db>;
    using vc = vector<cd>;

    static const bool INT=true;

    static void fft(vc &a){
        int n=a.size(),L=31-__builtin_clz(n);
        vc rt(n);
        rt[1]=1;
        for(int k=2;k<n;k*=2){
            cd z=polar(db(1),PI/k);
            for(int i=k;i<2*k;i++) rt[i]=i&1?rt[i/2]*z:rt[i/2];
        }
        vector<int> rev(n);
```

```
        for(int i=1;i<n;i++) rev[i]=(rev[i/2] | (i&1)<<L)/2;
        for(int i=1;i<n;i++) if(i<rev[i]) swap(a[i],a[rev[i]]);
        for(int k=1;k<n;k*=2) for(int i=0;i<n;i+=2*k) for(int j=0;j<k;j++){
            cd z=rt[j+k]*a[i+j+k];
            a[i+j+k]=a[i+j]-z;
            a[i+j]+=z;
        }
    }
    template<class U>
    static db norm(const U &x){
        return INT?round(x):x;
    }
    static vt conv(const vt &a,const vt &b){
        if(a.empty() || b.empty()) return {};
        vt res(a.size()+b.size()-1);
        int L=32-__builtin_clz(res.size()),n=1<<L;
        vc in(n),out(n);
        copy(a.begin(),a.end(),in.begin());
        for(int i=0;i<b.size();i++) in[i].imag(b[i]);
        fft(in);
        for(auto &x:in) x*=x;
        for(int i=0;i<n;i++) out[i]=in[-i&(n-1)]-conj(in[i]);
        fft(out);
        for(int i=0;i<res.size();i++) res[i]=norm(imag(out[i])/(4*n));
        return res;
    }
    static vl convMod(const vl &a,const vl &b){
        assert(mod>0);
        if(a.empty() || b.empty()) return {};
        vl res(a.size()+b.size()-1);
        int L=32-__builtin_clz(res.size()),n=1<<L;
        ll cut=int(sqrt(mod));
        vc in1(n),in2(n),out1(n),out2(n);
        for(int i=0;i<a.size();i++) in1[i]=cd(ll(a[i])/cut,ll(a[i])%cut); // a1 + i * a2
        for(int i=0;i<b.size();i++) in2[i]=cd(ll(b[i])/cut,ll(b[i])%cut); // b1 + i * b2
        fft(in1),fft(in2);
        for(int i=0;i<n;i++){
            int j=-i&(n-1);
            out1[j]=(in1[i]+conj(in1[j]))*in2[i]/(2.1*n); // f1 * (g1 + i * g2) = f1 * g1 + i f1 * g2
            out2[j]=(in1[i]-conj(in1[j]))*in2[i]/cd(0.1,2.1*n); // f2 * (g1 + i * g2) = f2 * g1 + i f2 * g2
        }
        fft(out1),fft(out2);
        for(int i=0;i<res.size();i++){
            ll x=round(real(out1[i])),y=round(imag(out1[i]))+round(real(out2[i])),z=round(imag(out2[i]));
            res[i]=((x%mod*cut+y)%mod*cut+z)%mod; // a1 * b1 * cut^2 + (a1 * b2 + a2 * b1) * cut + a2 * b2
        }
        return res;
    }
    vt operator()(const vt &a,const vt &b){
        return mod>0?conv(a,b):convMod(a,b);
    }
};
template<>
struct FFT<db>{
    static const bool INT=false;
};
```

ntt.hpp  
Description: Number Theoretic Transform

```
Time:  $\mathcal{O}(N \log N)$ 
"src/modular-arithmetic/binpow.hpp", "src/modular-arithmetic/montgomery-modint.hpp"
fe4963, 39 lines
template<class mint>
struct NTT{
    using vm = vector<mint>;

    static constexpr mint root=mint::get_root();
    static_assert(root!=0);

    static void ntt(vm &a){
        int n=a.size(),L=31-__builtin_clz(n);
        vm rt(n);
        rt[1]=1;
        for(int k=2,s=2;k<n;k*=2,s++){
            mint z[]={1,binpow(root,MOD>>s)};
            for(int i=k;i<2*k;i++) rt[i]=rt[i/2]*z[i&1];
        }
        vector<int> rev(n);
        for(int i=1;i<n;i++) rev[i]=(rev[i/2] | (i&1)<<L)/2;
        for(int i=1;i<n;i++) if(i<rev[i]) swap(a[i],a[rev[i]]);
        for(int k=1;k<n;k*=2) for(int i=0;i<n;i+=2*k) for(int j=0;j<k;j++){
            mint z=rt[j+k]*a[i+j+k];
            a[i+j+k]=a[i+j]-z;
            a[i+j]+=z;
        }
    }
    static vm conv(const vm &a,const vm &b){
        if(a.empty() || b.empty()) return {};
        int s=a.size()+b.size()-1,n=1<<(32-__builtin_clz(s));
        mint inv=mint(n).inv();
        vm in1(a),in2(b),out(n);
        in1.resize(n),in2.resize(n);
        ntt(in1),ntt(in2);
        for(int i=0;i<n;i++) out[-i&(n-1)]=in1[i]*in2[i]*inv;
        ntt(out);
        return vm(out.begin(),out.begin()+s);
    }
    vm operator()(const vm &a,const vm &b){
        return conv(a,b);
    }
};
```

lagrange-interpolate.hpp  
Description: Lagrange interpolation. Given f(0)...f(n) return a polynomial with degree n.  
Time:  $\mathcal{O}(N)$   
c06459, 16 lines

```
template<class mint>
mint lagrange_interpolate(vector<mint> &f,mint c){
    int n=f.size();
    if(c.val()<n) return f[c.val()];
    vector<mint> l(n+1),r(n+1);
    l[0]=r[n]=1;
    for(int i=0;i<n;i++) l[i+1]=l[i]*(c-i);
    for(int i=n-1;i>=0;i--) r[i]=r[i+1]*(c-i);
    mint ans=0;
    for(int i=0;i<n;i++){
        mint cur=f[i]*comb.ifac(i)*comb.ifac(n-i-1);
        if((n-i-1)&1) cur*=-1;
        ans+=cur*l[i]*r[i+1];
    }
    return ans;
}
```

binpow.hpp

Description: n-th power using divide and conquer

Time:  $\mathcal{O}(\log b)$

8a3f76, 6 lines

```
template<class T>
constexpr T binpow(T a,ll b){
    T res=1;
    for(;b>0;b>>=1,a*=a) if (b&1) res*=a;
    return res;
}
```

montgomery-modint.hpp

Description: modular arithmetic operators using Montgomery space

c4c9af, 105 lines

```
template<uint32_t mod,uint32_t root=0>
struct MontgomeryModInt{
    using mint = MontgomeryModInt;
    using i32 = int32_t;
    using u32 = uint32_t;
    using u64 = uint64_t;

    static constexpr u32 get_r(){
        u32 res=1;
        for(i32 i=0;i<5;i++) res*=2-mod*res;
        return res;
    }

    static const u32 r=get_r();
    static const u32 n2=-u64(mod)%mod;
    static_assert(mod<(1<<30));
    static_assert((mod&1)==1);
    static_assert(r*mod==1);

    u32 x;

    constexpr MontgomeryModInt():x(0){}
    constexpr MontgomeryModInt(const int64_t &v):x(reduce(u64(v
        %mod+mod)*n2)){}

    static constexpr u32 get_mod(){return mod;}
    static constexpr mint get_root(){return mint(root);}
    explicit constexpr operator int64_t()(const{return val();})

    static constexpr u32 reduce(const u64 &v){
        return (v+u64(u32(v)*u32(-r))*mod)>>32;
    }

    constexpr u32 val()const{
        u32 res=reduce(x);
        return res>=mod?res-mod:res;
    }

    constexpr mint inv()const{
        int a=val(),b=mod,u=1,v=0,q=0;
        while(b>0){
            q=a/b;
            a-=q*b;
            u-=q*v;
            swap(a,b);
            swap(u,v);
        }
        return mint(u);
    }

    constexpr mint &operator+=(const mint &rhs){
        if(i32(x+=rhs.x-2*mod)<0)x+=2*mod;
        return *this;
    }

    constexpr mint &operator-=(const mint &rhs){
        if(i32(x-=rhs.x)<0)x+=2*mod;
```

binpow montgomery-modint Simplex

```
        return *this;
    }
    constexpr mint &operator*=(const mint &rhs){
        x=reduce(u64(x)*rhs.x);
        return *this;
    }
    constexpr mint &operator/=(const mint &rhs){
        return *this*=rhs.inv();
    }

    constexpr mint &operator++(){return *this+=mint(1);}
    constexpr mint &operator--(){return *this-=mint(1);}
    constexpr mint operator++(int){
        mint res=*this;
        return *this+=mint(1),res;
    }
    constexpr mint operator--(int){
        mint res=*this;
        return *this-=mint(1),res;
    }

    constexpr mint operator-()const{return mint()-mint(*this);}
    ;
    constexpr mint operator+()const{return mint(*this);};

    friend constexpr mint operator+(const mint &lhs,const mint
        &rhs){return mint(lhs)+=rhs;}
    friend constexpr mint operator-(const mint &lhs,const mint
        &rhs){return mint(lhs)-=rhs;}
    friend constexpr mint operator*(const mint &lhs,const mint
        &rhs){return mint(lhs)*=rhs;}
    friend constexpr mint operator/(const mint &lhs,const mint
        &rhs){return mint(lhs)/=rhs;}
    friend constexpr bool operator==(const mint &lhs,const mint
        &rhs){
        return (lhs.x>=mod?lhs.x-mod:lhs.x)==(rhs.x>=mod?rhs.x-
            mod:rhs.x);
    }
    friend constexpr bool operator!=(const mint &lhs,const mint
        &rhs){
        return (lhs.x>=mod?lhs.x-mod:lhs.x)!=(rhs.x>=mod?rhs.x-
            mod:rhs.x);
    }
    friend constexpr bool operator<(const mint &lhs,const mint
        &rhs){
        return (lhs.x>=mod?lhs.x-mod:lhs.x)<(rhs.x>=mod?rhs.x-
            mod:rhs.x); // for std::map
    }

    friend istream &operator>>(istream &is,mint &o){
        int64_t v;
        is >> v;
        o=mint(v);
        return is;
    }
    friend ostream &operator<<(ostream &os,const mint &o){
        return os << o.val();
    }
};
using mint998 = MontgomeryModInt<998244353,3>;
using mint107 = MontgomeryModInt<1000000007>;
```

# Linear Programming (13)

## 13.1 LP Duality

primal problem:

maximize

$c^T x$

subject to

$Ax \leq b$

$x \geq 0$

dual problem:

minimize

$b^T y$

subject to

$A^T y \geq c$

$y \geq 0$

Maximization	Minimization
Inequality constraint $\leq$	Nonnegative variable $\geq$
Inequality constraint $\geq$	Nonpositive variable $\leq$
Equality constraint $=$	Free variable
Nonnegative variable $\geq$	Inequality constraint $\geq$
Nonpositive variable $\leq$	Inequality constraint $\leq$
Free variable	Equality constraint $=$

### Simplex.h

**Description:** Solves a general linear maximization problem: maximize  $c^T x$  subject to  $Ax \leq b, x \geq 0$ . 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:**  $\mathcal{O}(NM * \text{\#pivots})$ , where a pivot may be e.g. an edge relaxation.  $\mathcal{O}(2^n)$  in the general case.

aa8530, 68 lines

```
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 MP make_pair
#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)) {
            rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
            rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
            rep(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];
        rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
            T *b = D[i].data(), inv2 = b[s] * inv;
            rep(j,0,n+2) b[j] -= a[j] * inv2;
            b[s] = a[s] * inv2;
        }
    }
};
```

```
rep(j,0,n+2) if (j != s) D[r][j] *= inv;
rep(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;
        rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
        if (D[x][s] >= -eps) return true;
        int r = -1;
        rep(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;
    rep(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;
        rep(i,0,m) if (B[i] == -1) {
            int s = 0;
            rep(j,1,n+1) ltj(D[i]);
            pivot(i, s);
        }
    }
    bool ok = simplex(1); x = vd(n);
    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
}
};
```

Miscellaneous (14)

integrate.hpp  
Description: find definite integral with Simpson’s method, error proportional to  $dx^4$ .

197721, 14 lines

```
template<class F>
db quad(db a,db b,const F &f,int n){
    db res=0;
    db dx=(b-a)/n;
    db fl=0,fr=f(a);
    for(int i=0;i<n;i++){
        db l=a+dx*i,r=l+dx;
        fl=fr;
        fr=f(r);
        db fm=f((l+r)/2);
        res+=fl+4*fm+fr;
    }
    return res*dx/6;
}
```

range-xor.hpp  
Description: find all range of x such that  $l \leq x \text{ xor } p < r$ .

"src/miscellaneous/range-xor.hpp" cc7fb9, 18 lines

```
template<class F>
void range_xor(ll p,ll l,ll r,const F &query){
    for(int i=0;i<60;i++){
```

```
        if(l==r)break;
        ll b=1LL<<i;
        if(l&b){
            query(l^p,(l^p)+b);
            l+=b;
        }
        if(r&b){
            r-=b;
            query(r^p,(r^p)+b);
        }
        if(p&b){
            p^=b;
        }
    }
}
```