



Universidade Estadual de Campinas

Inimigos do Tiago

Yan Silva, Luiz Oda, Bernardo Archegas

ICPC World Finals 2024

Sep 13, 2024

- 1 Contest
- 2 Mathematics
- 3 Data structures
- 4 Numerical
- 5 Number theory
- 6 Combinatorial
- 7 Graph
- 8 Geometry
- 9 Strings
- 10 Various

Contest (1)

template.cpp37 lines

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

#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define per(i, a, b) for(int i = b-1; i>=a ; i--)
#define trav(a, x) for(auto& a : x)
#define allin(a , x) for(auto a : x)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;
mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count());

#define Unique(v) sort(all(v));v.erase(unique(all(v)),v.end());
// g++ -g a.cpp -std=c++17 -Wall -Wextra -Wno-unused-result -Wconversion -Wfatal-errors -o a.out
// -fsanitize=undefined,address (dont abuse, slow compiling)
// ulimit -s unlimited (change stack size)

// debug:
void dbg_out() { cerr << endl; }
template<typename Head, typename... Tail> void dbg_out(Head H, Tail... T){
    cerr << ' ' << H;
    dbg_out(T...);
}

#ifdef LOCAL
#define dbg(...) cerr<< "(" << #__VA_ARGS__<< "):" , dbg_out(
    __VA_ARGS__ ) , cerr << endl
#else
#define dbg(...) 42
#endif
// usage: dbg(x,y,z); works for any printable thing
// compile: g++ name.cpp -DLOCAL && ./a.out

int main() {
    cin.tie(0)->sync_with_stdio(0);
```

31 lines

```
cin.exceptions(cin.failbit);
}

run.sh

g++ -std=c++17 -o generator generator.cpp
g++ -std=c++17 -o main main.cpp
g++ -std=c++17 -o naive naive.cpp

value=1

while [ $value -le $1 ]
do
    ./generator $value > in.txt
    ./main < in.txt > out_main.txt
    ./naive < in.txt > out_naive.txt

    result=$(diff out_main.txt out_naive.txt)

    if [ $? -eq 0 ]
    then
        printf "Test $value: Accepted\n"
    else
        printf "Test $value: Wrong Answer\n"
        printf "\nInput:\n"
        cat in.txt
        printf "\nOutput:\n"
        cat out_main.txt
        printf "\nAnswer:\n"
        cat out_naive.txt

        break
    fi

    value=$((value+1))
done
```

Hash.sh3 lines

```
# Para usar (hash das linhas [11, 12]):
# bash hash.sh arquivo.cpp 11 12
sed -n $2', '$3' p' $1 | sed '/^#w/d' | cpp -dD -P -
    fpreprocessed | tr -d '[:space:]' | md5sum | cut -c-6
```

troubleshoot.txt52 lines

```
Pre-submit:
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all data structures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
```

Go through this list again.

Explain your algorithm to a teammate.

Ask the teammate to look at your code.

Go for a small walk, e.g. to the toilet.

Is your output format correct? (including whitespace)

Rewrite your solution from the start or let a teammate do it.

Runtime error:

Have you tested all corner cases locally?

Any uninitialized variables?

Are you reading or writing outside the range of any vector?

Any assertions that might fail?

Any possible division by 0? (mod 0 for example)

Any possible infinite recursion?

Invalidated pointers or iterators?

Are you using too much memory?

Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:

Do you have any possible infinite loops?

What is the complexity of your algorithm?

Are you copying a lot of unnecessary data? (References)

How big is the input and output? (consider scanf)

Avoid vector, map. (use arrays/unordered_map)

What do your teammates think about your algorithm?

Memory limit exceeded:

What is the max amount of memory your algorithm should need?

Are you clearing all data structures between test cases?

TemplateBer.cpp63 lines

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

using namespace std;
using ll = long long;
using pii = pair<int, int>;
using pll = pair<ll, ll>;

mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count());

const int MOD = 1e9 + 7;
const int MAXN = 2e5 + 5;
const ll INF = 2e18;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);

    return 0;
}

// bitset
//
// deve ser inicializado com um valor constante
const int N = 50;
bitset<N> bit;
// pode receber uma string binaria ou um int
bit = "1010", bit = 13;
bit.count() // quantas posicoes acesas
bit.set() // acende tudo
bit.set(3) // acende a posicao 3
bit.reset() // apaga tudo
bit.reset(4) // apaga a posicao 4
bit.flip() // flipa tudo
bit.flip(2) // flipa a posicao 2
string a = bit.to_string() // transforma em string
bit &= bit
```

```
bit |= bit
bit ^= bit

// __int128_t
//
// nao pode ler nem printar
// pra ler use a to128
// pra printar use a tostring

__int128_t to128(string s) {
    __int128_t a = 0;
    __int128_t pot = 1;
    int n = s.size();
    for (int i = n - 1; i >= 0; i--, pot *= 10) {
        a += (s[i] - '0') * pot;
    }
    return a;
}

string tostring(__int128_t a) {
    string s;
    while (a) {
        s += (a % 10) + '0';
        a /= 10;
    }
    reverse(s.begin(), s.end());
    return (s.empty() ? "0" : s);
}
```

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 $x=-b/2a$.

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

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,\ldots,r_k are distinct roots of $x^k+c_1x^{k-1}+\cdots+c_k$, there are d_1,\ldots,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

$$\begin{aligned}\sin(v+w)&=\sin v\cos w+\cos v\sin w \\ \cos(v+w)&=\cos v\cos w-\sin v\sin w\end{aligned}$$

$$\begin{aligned}\tan(v+w)&=\frac{\tan v+\tan w}{1-\tan v\tan w} \\ \sin v+\sin w&=2\sin\frac{v+w}{2}\cos\frac{v-w}{2} \\ \cos v+\cos w&=2\cos\frac{v+w}{2}\cos\frac{v-w}{2}\end{aligned}$$

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

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

$$\begin{aligned}a\cos x+b\sin x&=r\cos(x-\phi) \\ a\sin x+b\cos x&=r\sin(x+\phi)\end{aligned}$$

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

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):

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

Length of bisector (divides angles in two):

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

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

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

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

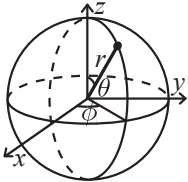
2.4.2 Quadrilaterals

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

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

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

2.4.3 Spherical coordinates



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

2.5 Derivatives/Integrals

$$\frac{d}{dx}\arcsin x=\frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{dx}\arccos x=-\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx}\tan x=1+\tan^2x \qquad \frac{d}{dx}\arctan x=\frac{1}{1+x^2}$$

$$\int\tan ax=-\frac{\ln|\cos ax|}{a} \qquad \int x\sin ax=\frac{\sin ax-ax\cos ax}{a^2}$$

$$\int e^{-x^2}=\frac{\sqrt{\pi}}{2}\text{erf}(x) \qquad \int xe^{ax}dx=\frac{e^{ax}}{a^2}(ax-1)$$

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

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

$$1+2+3+\cdots+n=\frac{n(n+1)}{2}$$

$$1^2+2^2+3^2+\cdots+n^2=\frac{n(2n+1)(n+1)}{6}$$

$$1^3+2^3+3^3+\cdots+n^3=\frac{n^2(n+1)^2}{4}$$

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

2.7 Series

$$e^x=1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots,(-\infty<x<\infty)$$

$$\ln(1+x)=x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\ldots,(-1<x\leq 1)$$

$$\sqrt{1+x}=1+\frac{x}{2}-\frac{x^2}{8}+\frac{2x^3}{32}-\frac{5x^4}{128}+\ldots,(-1\leq x\leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots, \quad (-\infty < x < \infty)$$
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots, \quad (-\infty < x < \infty)$$

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 σ 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 wich yields success with probability p is $\text{Fs}(p)$, $0 \leq p \leq 1$.

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

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

Poisson distribution

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

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

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

Data structures (3)

OrderStatisticTree.h

Description: A set (not multiset!) with support for finding the n'th element, and finding the index of an element. To get a map, change null_type. **Time:** $\mathcal{O}(\log N)$

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

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

void example() {
    Tree<int> t, t2; t.insert(8);
    auto it = t.insert(10).first;
    assert(it == t.lower_bound(9));
    assert(t.order_of_key(10) == 1);
    assert(t.order_of_key(11) == 2);
    assert(*t.find_by_order(0) == 8);
    t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

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).

```
#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});
```

SegTreeBeats.cpp

Description: query(a, b) - min(v[a..b]), max(v[a..b]), sum(v[a..b]) updatemin(a, b, x) faz com que v[i] <- min(v[i], x), para i em [a, b] updatemax faz o mesmo com max, e updatesum soma x em todo mundo do intervalo [a, b]

Time: build - $\mathcal{O}(n)$ query - $\mathcal{O}(\log(n))$ update - $\mathcal{O}(\log^2(n))$ amortizado (se nao usar updatesum, fica log(n) amortizado)

```
#define f first
#define s second

namespace beats {
    struct node {
        int tam;
        ll sum, lazy; // lazy pra soma
        ll mil, mi2, mi; // mi = #mi1
        ll mal, ma2, ma; // ma = #ma1

        node(ll x = 0) {
            sum = mil = mal = x;
            mi2 = LINF, ma2 = -LINF;
            mi = ma = tam = 1;
            lazy = 0;
        }
        node(const node& l, const node& r) {
            sum = l.sum + r.sum, tam = l.tam + r.tam;
            lazy = 0;
            if (l.mil > r.mil) {
                mil = r.mil, mi = r.mi;
                mi2 = min(l.mi2, r.mi2);
            }
        }
    };
}
```

```
} else if (l.mil < r.mil) {
    mil = l.mil, mi = l.mi;
    mi2 = min(r.mil, l.mi2);
} else {
    mil = l.mil, mi = l.mi+r.mi;
    mi2 = min(l.mi2, r.mi2);
}

if (l.mal < r.mal) {
    mal = r.mal, ma = r.ma;
    ma2 = max(l.mal, r.ma2);
} else if (l.mal > r.mal) {
    mal = l.mal, ma = l.ma;
    ma2 = max(r.mal, l.ma2);
} else {
    mal = l.mal, ma = l.ma+r.ma;
    ma2 = max(l.ma2, r.ma2);
}
}

void setmin(ll x) {
    if (x >= mal) return;
    sum += (x - mal)*ma;
    if (mil == mal) mil = x;
    if (mi2 == mal) mi2 = x;
    mal = x;
}

void setmax(ll x) {
    if (x <= mil) return;
    sum += (x - mil)*mi;
    if (mal == mil) mal = x;
    if (ma2 == mil) ma2 = x;
    mil = x;
}

void setsum(ll x) {
    mil += x, mi2 += x, mal += x, ma2 += x;
    sum += x*tam;
    lazy += x;
}

};

node seg[4*MAX];
int n, *v;

node build(int p=1, int l=0, int r=n-1) {
    if (l == r) return seg[p] = {v[l]};
    int m = (l+r)/2;
    return seg[p] = {build(2*p, l, m), build(2*p+1, m+1, r)};
}

void build(int n2, int* v2) {
    n = n2, v = v2;
    build();
}

void prop(int p, int l, int r) {
    if (l == r) return;
    for (int k = 0; k < 2; k++) {
        if (seg[p].lazy) seg[2*p+k].setsum(seg[p].lazy);
        seg[2*p+k].setmin(seg[p].mal);
        seg[2*p+k].setmax(seg[p].mil);
    }
    seg[p].lazy = 0;
}

pair<pair<ll, ll>, ll> query(int a, int b, int p=1, int l=0,
    int r=n-1) {
    if (b < l or r < a) return {{LINF, -LINF}, 0};
    if (a <= l and r <= b) return {{seg[p].mil, seg[p].mal},
        seg[p].sum};
    prop(p, l, r);
    int m = (l+r)/2;
    auto L = query(a, b, 2*p, l, m), R = query(a, b, 2*p+1, m
        +1, r);
}
```

```
        return {{min(L.f.f, R.f.f), max(L.f.s, R.f.s)}, L.s+R.s};
    }
    node updatemin(int a, int b, ll x, int p=1, int l=0, int r=n-1) {
        if (b < l or r < a or seg[p].mal <= x) return seg[p];
        if (a <= l and r <= b and seg[p].ma2 < x) {
            seg[p].setmin(x);
            return seg[p];
        }
        prop(p, l, r);
        int m = (l+r)/2;
        return seg[p] = {updatemin(a, b, x, 2*p, l, m),
            updatemin(a, b, x, 2*p+1, m+1, r)};
    }
    node updatemax(int a, int b, ll x, int p=1, int l=0, int r=n-1) {
        if (b < l or r < a or seg[p].mil >= x) return seg[p];
        if (a <= l and r <= b and seg[p].mi2 > x) {
            seg[p].setmax(x);
            return seg[p];
        }
        prop(p, l, r);
        int m = (l+r)/2;
        return seg[p] = {updatemax(a, b, x, 2*p, l, m),
            updatemax(a, b, x, 2*p+1, m+1, r)};
    }
    node updatesum(int a, int b, ll x, int p=1, int l=0, int r=n-1) {
        if (b < l or r < a) return seg[p];
        if (a <= l and r <= b) {
            seg[p].setsum(x);
            return seg[p];
        }
        prop(p, l, r);
        int m = (l+r)/2;
        return seg[p] = {updatesum(a, b, x, 2*p, l, m),
            updatesum(a, b, x, 2*p+1, m+1, r)};
    }
};
```

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

9556fc, 55 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 pa = split(n->l, k);
        n->l = pa.second;
        n->recalc();
        return {pa.first, n};
    } else {
        auto pa = split(n->r, k - cnt(n->l) - 1); // and just "k"
        n->r = pa.first;
```

```
        n->recalc();
        return {n, pa.second};
    }
}

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);
        l->recalc();
        return l;
    } else {
        r->l = merge(l, r->l);
        r->recalc();
        return r;
    }
}

Node* ins(Node* t, Node* n, int pos) {
    auto pa = split(t, pos);
    return merge(merge(pa.first, n), pa.second);
}

// 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));
}
```

TreapImplicitaUFMG.cpp
Description: Todas as operacoes custam $\mathcal{O}(\log(n))$ com alta probabilidade

83ba4d, 89 lines

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

template<typename T> struct treap {
    struct node {
        node *l, *r;
        int p, sz;
        T val, sub, lazy;
        bool rev;
        node(T v) : l(NULL), r(NULL), p(rng()), sz(1), val(v), sub(v), lazy(0), rev(0) {}
        void prop() {
            if (lazy) {
                val += lazy, sub += lazy*sz;
                if (l) l->lazy += lazy;
                if (r) r->lazy += lazy;
            }
            if (rev) {
                swap(l, r);
                if (l) l->rev ^= 1;
                if (r) r->rev ^= 1;
            }
            lazy = 0, rev = 0;
        }
        void update() {
            sz = 1, sub = val;
            if (l) l->prop(), sz += l->sz, sub += l->sub;
            if (r) r->prop(), sz += r->sz, sub += r->sub;
        }
    };

    node* root;

    treap() { root = NULL; }
```

```
    treap(const treap& t) {
        throw logic_error("Nao copiar a treap!");
    }
    ~treap() {}
    vector<node*> q = {root};
    while (q.size()) {
        node* x = q.back(); q.pop_back();
        if (!x) continue;
        q.push_back(x->l), q.push_back(x->r);
        delete x;
    }

    int size(node* x) { return x ? x->sz : 0; }
    int size() { return size(root); }
    void join(node* l, node* r, node& i) { // assume que l < r
        if (!l or !r) return void(i = l ? l : r);
        l->prop(), r->prop();
        if (l->p > r->p) join(l->r, r, l->r), i = l;
        else join(l, r->l, r->l), i = r;
        i->update();
    }
    void split(node* i, node& l, node& r, int v, int key = 0) {
        if (!i) return void(r = l = NULL);
        i->prop();
        if (key + size(i->l) < v) split(i->r, i->r, r, v, key+size(i->l)+1), l = i;
        else split(i->l, l, i->l, v, key), r = i;
        i->update();
    }
    void push_back(T v) {
        node* i = new node(v);
        join(root, i, root);
    }
    T query(int l, int r) {
        node *L, *M, *R;
        split(root, M, R, r+1), split(M, L, M, l);
        T ans = M->sub;
        join(L, M, M), join(M, R, root);
        return ans;
    }
    void update(int l, int r, T s) {
        node *L, *M, *R;
        split(root, M, R, r+1), split(M, L, M, l);
        M->lazy += s;
        join(L, M, M), join(M, R, root);
    }
    void reverse(int l, int r) {
        node *L, *M, *R;
        split(root, M, R, r+1), split(M, L, M, l);
        M->rev ^= 1;
        join(L, M, M), join(M, R, root);
    }
};
```

RMQ.h
Description: Range Minimum Queries on an array. Returns $\min(V[a], V[a + 1], \dots V[b - 1])$ in constant time.
Usage: RMQ rmq(values);
rmq.query(inclusive, exclusive);
Time: $\mathcal{O}(|V| \log |V| + Q)$

510c32, 16 lines

```
template<class T>
struct RMQ {
    vector<vector<T>> jmp;
    RMQ(const vector<T>& V) : jmp(1, V) {
        for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2, ++k) {
            jmp.emplace_back(sz(V) - pw * 2 + 1);
            rep(j, 0, sz(jmp[k]))
```

```
        jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw]);
    }
}
T query(int a, int b) {
    assert(a < b); // or return inf if a == b
    int dep = 31 - __builtin_clz(b - a);
    return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
}
};
```

ChtDinamico.cpp

Description: para double, use LINF = 1/.0, div(a, b) = a/b update(x) atualiza o ponto de intersecao da reta x overlap(x) verifica se a reta x sobrepoe a proxima add(a, b) adiciona reta da forma ax + b query(x) computa maximo de ax + b para entre as retas O(log(n)) amortizado por insercao O(log(n)) por query

03d6ec, 39 lines

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

struct dynamic_hull : multiset<Line, less<>> {
    ll div(ll a, ll b) {
        return a / b - ((a ^ b) < 0 and a % b);
    }

    void update(iterator x) {
        if (next(x) == end()) x->p = LINF;
        else if (x->a == next(x)->a) x->p = x->b >= next(x)->b ?
            LINF : -LINF;
        else x->p = div(next(x)->b - x->b, x->a - next(x)->a);
    }

    bool overlap(iterator x) {
        update(x);
        if (next(x) == end()) return 0;
        if (x->a == next(x)->a) return x->b >= next(x)->b;
        return x->p >= next(x)->p;
    }

    void add(ll a, ll b) {
        auto x = insert({a, b, 0});
        while (overlap(x)) erase(next(x)), update(x);
        if (x != begin() and !overlap(prev(x))) x = prev(x), update
            (x);
        while (x != begin() and overlap(prev(x)))
            x = prev(x), erase(next(x)), update(x);
    }

    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        #warning cuidado com overflow!
        return l.a * x + l.b;
    }
};
```

Numerical (4)

4.1 Polynomials and recurrences

Polynomial.h

c9b7b0, 17 lines

```
struct Poly {
    vector<double> a;
    double operator()(double x) const {
```

```
        double val = 0;
        for (int i = sz(a); i--;) (val *= x) += a[i];
        return val;
    }
    void diff() {
        rep(i,1,sz(a)) a[i-1] = i*a[i];
        a.pop_back();
    }
    void divroot(double x0) {
        double b = a.back(), c; a.back() = 0;
        for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
        a.pop_back();
    }
};
```

PolyRoots.h

Description: Finds the real roots to a polynomial.
Usage: polyRoots({{2,-3,1}},-1e9,1e9) // solve x^2-3x+2 = 0
Time: $\mathcal{O}(n^2 \log(1/\epsilon))$

"Polynomial.h" b00bfe, 23 lines

```
vector<double> polyRoots(Poly p, double xmin, double xmax) {
    if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
    vector<double> ret;
    Poly der = p;
    der.diff();
    auto dr = polyRoots(der, xmin, xmax);
    dr.push_back(xmin-1);
    dr.push_back(xmax+1);
    sort(all(dr));
    rep(i,0,sz(dr)-1) {
        double l = dr[i], h = dr[i+1];
        bool sign = p(l) > 0;
        if (sign ^ (p(h) > 0)) {
            rep(it,0,60) { // while (h - l > 1e-8)
                double m = (l + h) / 2, f = p(m);
                if ((f <= 0) ^ sign) l = m;
                else h = m;
            }
            ret.push_back((l + h) / 2);
        }
    }
    return ret;
}
```

PolyInterpolate.h

Description: Given n points $(x[i], y[i])$, computes an $n-1$ -degree polynomial p that passes through them: $p(x) = a[0] * x^0 + \dots + a[n-1] * x^{n-1}$. For numerical precision, pick $x[k] = c * \cos(k/(n-1) * \pi)$, $k = 0 \dots n-1$.
Time: $\mathcal{O}(n^2)$

08bf48, 13 lines

```
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
    vd res(n), temp(n);
    rep(k,0,n-1) rep(i,k+1,n)
        y[i] = (y[i] - y[k]) / (x[i] - x[k]);
    double last = 0; temp[0] = 1;
    rep(k,0,n) rep(i,0,n) {
        res[i] += y[k] * temp[i];
        swap(last, temp[i]);
        temp[i] -= last * x[k];
    }
    return res;
}
```

EvalInterpol.h

Description: Dado 'n' pontos (i, y[i]), i in [0, n), avalia o polinomio de grau n-1 que passa por esses pontos em 'x'. Tudo modular, precisa do mint
Time: $\mathcal{O}(n)$

4fe929, 20 lines

```
mint evaluate_interpolation(int x, vector<mint> y) {
    int n = y.size();

    vector<mint> sulf(n+1, 1), fat(n, 1), ifat(n);
    for (int i = n-1; i >= 0; i--) sulf[i] = sulf[i+1] * (x - i);
    for (int i = 1; i < n; i++) fat[i] = fat[i-1] * i;
    ifat[n-1] = 1/fat[n-1];
    for (int i = n-2; i >= 0; i--) ifat[i] = ifat[i+1] * (i + 1);

    mint pref = 1, ans = 0;
    for (int i = 0; i < n; pref *= (x - i++)) {
        mint num = pref * sulf[i+1];

        mint den = ifat[i] * ifat[n-1 - i];
        if ((n-1 - i)%2) den *= -1;

        ans += y[i] * num * den;
    }
    return ans;
}
```

BerlekampMassey.h

Description: Recovers any n -order linear recurrence relation from the first $2n$ terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for floats is not guaranteed. Output will have size $\leq n$.
Usage: berlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}
Time: $\mathcal{O}(N^2)$

"../number-theory/ModPow.h" 96548b, 20 lines

```
vector<ll> berlekampMassey(vector<ll> s) {
    int n = sz(s), L = 0, m = 0;
    vector<ll> C(n), B(n), T;
    C[0] = B[0] = 1;

    ll b = 1;
    rep(i,0,n) { ++m;
        ll d = s[i] % mod;
        rep(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
        if (!d) continue;
        T = C; ll coef = d * modpow(b, mod-2) % mod;
        rep(j,m,n) C[j] = (C[j] - coef * B[j - m]) % mod;
        if (2 * L > i) continue;
        L = i + 1 - L; B = T; b = d; m = 0;
    }

    C.resize(L + 1); C.erase(C.begin());
    for (ll& x : C) x = (mod - x) % mod;
    return C;
}
```

LinearRecurrence.h

Description: Generates the k 'th term of an n -order linear recurrence $S[i] = \sum_j S[i - j - 1]tr[j]$, given $S[0 \dots \geq n - 1]$ and $tr[0 \dots n - 1]$. Faster than matrix multiplication. Useful together with Berlekamp-Massey.
Usage: linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number
Time: $\mathcal{O}(n^2 \log k)$

f4e444, 26 lines

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
    int n = sz(tr);

    auto combine = [&](Poly a, Poly b) {
        Poly res(n * 2 + 1);
        rep(i,0,n+1) rep(j,0,n+1)
            res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
        for (int i = 2 * n; i > n; --i) rep(j,0,n)
            res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
        res.resize(n + 1);
    }
```

```
    return res;
};

Poly pol(n + 1), e(pol);
pol[0] = e[1] = 1;

for (++k; k; k /= 2) {
    if (k % 2) pol = combine(pol, e);
    e = combine(e, e);
}

ll res = 0;
rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
return res;
}
```

4.2 Optimization

GoldenSectionSearch.h

Description: Finds the argument minimizing the function f in the interval $[a, b]$ assuming f is unimodal on the interval, i.e. has only one local minimum. The maximum error in the result is ϵ *eps*. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.
Usage: double func(double x) { return 4+x+.3*x*x; }
double xmin = gss(-1000,1000,func);
Time: $\mathcal{O}(\log((b-a)/\epsilon))$

31d45b, 14 lines

```
double gss(double a, double b, double (*f)(double)) {
    double r = (sqrt(5)-1)/2, eps = 1e-7;
    double x1 = b - r*(b-a), x2 = a + r*(b-a);
    double f1 = f(x1), f2 = f(x2);
    while (b-a > eps)
        if (f1 < f2) { //change to > to find maximum
            b = x2; x2 = x1; f2 = f1;
            x1 = b - r*(b-a); f1 = f(x1);
        } else {
            a = x1; x1 = x2; f1 = f2;
            x2 = a + r*(b-a); f2 = f(x2);
        }
    return a;
}
```

HillClimbing.h

Description: Poor man's optimization for unimodal functions. Seeef, 14 lines

```
typedef array<double, 2> P;
```

```
template<class F> pair<double, P> hillClimb(P start, F f) {
    pair<double, P> cur(f(start), start);
    for (double jmp = 1e9; jmp > 1e-20; jmp /= 2) {
        rep(j,0,100) rep(dx,-1,2) rep(dy,-1,2) {
            P p = cur.second;
            p[0] += dx*jmp;
            p[1] += dy*jmp;
            cur = min(cur, make_pair(f(p), p));
        }
    }
    return cur;
}
```

Integrate.h

Description: Simple integration of a function over an interval using Simpson's rule. The error should be proportional to h^4 , although in practice you will want to verify that the result is stable to desired precision when epsilon changes.

4756fc, 7 lines

```
template<class F>
double quad(double a, double b, F f, const int n = 1000) {
    double h = (b - a) / 2 / n, v = f(a) + f(b);
```

```
    rep(i,1,n*2)
        v += f(a + i*h) * (i&1 ? 4 : 2);
    return v * h / 3;
}
```

IntegrateAdaptive.h

Description: Fast integration using an adaptive Simpson's rule.
Usage: double sphereVolume = quad(-1, 1, [](double x) { return quad(-1, 1, [&](double y) { return quad(-1, 1, [&](double z) { return x*x + y*y + z*z < 1; }}});});

92dd79, 15 lines

```
typedef double d;
#define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6

template <class F>
d rec(F& f, d a, d b, d eps, d S) {
    d c = (a + b) / 2;
    d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
    if (abs(T - S) <= 15 * eps || b - a < 1e-10)
        return T + (T - S) / 15;
    return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S2);
}

template<class F>
d quad(d a, d b, F f, d eps = 1e-8) {
    return rec(f, a, b, eps, S(a, b));
}
```

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}};
vvd b = {1,1,-4}, c = {-1,-1}, x;
T val = LPSolver(A, b, c).solve(x);
Time: $\mathcal{O}(NM * \#pivots)$, 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 + modP>...
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;
        }
    }
}
```

```
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;
}
};
```

4.3 Matrices

Determinant.h

Description: Calculates determinant of a matrix. Destroys the matrix.
Time: $\mathcal{O}(N^3)$

bd5cec, 15 lines

```
double det(vector<vector<double>>& a) {
    int n = sz(a); double res = 1;
    rep(i,0,n) {
        int b = i;
        rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
        if (i != b) swap(a[i], a[b]), res *= -1;
        res *= a[i][i];
        if (res == 0) return 0;
        rep(j,i+1,n) {
            double v = a[j][i] / a[i][i];
            if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
        }
    }
    return res;
}
```

IntDeterminant.h

Description: Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.
Time: $\mathcal{O}(N^3)$

3313dc, 18 lines

```
const ll mod = 12345;
ll det(vector<vector<ll>>& a) {
```

```
int n = sz(a); ll ans = 1;
rep(i,0,n) {
    rep(j,i+1,n) {
        while (a[j][i] != 0) { // gcd step
            ll t = a[i][i] / a[j][i];
            if (t) rep(k,i,n)
                a[i][k] = (a[i][k] - a[j][k] * t) % mod;
            swap(a[i], a[j]);
            ans *= -1;
        }
    }
    ans = ans * a[i][i] % mod;
    if (!ans) return 0;
}
return (ans + mod) % mod;
}
```

SolveLinear.h
Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.
Time: $\mathcal{O}(n^2m)$

44c9ab, 38 lines

```
typedef vector<double> vd;
const double eps = 1e-12;

int solveLinear(vector<vd>& A, vd& b, vd& x) {
    int n = sz(A), m = sz(x), rank = 0, br, bc;
    if (n) assert(sz(A[0]) == m);
    vi col(m); iota(all(col), 0);

    rep(i,0,n) {
        double v, bv = 0;
        rep(r,i,n) rep(c,i,m)
            if ((v = fabs(A[r][c])) > bv)
                br = r, bc = c, bv = v;
        if (bv <= eps) {
            rep(j,i,n) if (fabs(b[j]) > eps) return -1;
            break;
        }
        swap(A[i], A[br]);
        swap(b[i], b[br]);
        swap(col[i], col[bc]);
        rep(j,0,n) swap(A[j][i], A[j][bc]);
        bv = 1/A[i][i];
        rep(j,i+1,n) {
            double fac = A[j][i] * bv;
            b[j] -= fac * b[i];
            rep(k,i+1,m) A[j][k] -= fac*A[i][k];
        }
        rank++;
    }

    x.assign(m, 0);
    for (int i = rank; i--;) {
        b[i] /= A[i][i];
        x[col[i]] = b[i];
        rep(j,0,i) b[j] -= A[j][i] * b[i];
    }
    return rank; // (multiple solutions if rank < m)
}
```

SolveLinear2.h
Description: To get all uniquely determined values of x back from SolveLinear, make the following changes:

"SolveLinear.h"08e495, 7 lines

```
rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
```

```
    rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
    x[col[i]] = b[i] / A[i][i];
fail:; }
```

SolveLinearBinary.h
Description: Solves $Ax = b$ over \mathbb{F}_2 . If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b .
Time: $\mathcal{O}(n^2m)$

fa2d7a, 34 lines

```
typedef bitset<1000> bs;

int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
    int n = sz(A), rank = 0, br;
    assert(m <= sz(x));
    vi col(m); iota(all(col), 0);
    rep(i,0,n) {
        for (br=i; br<n; ++br) if (A[br].any()) break;
        if (br == n) {
            rep(j,i,n) if(b[j]) return -1;
            break;
        }
        int bc = (int)A[br]._Find_next(i-1);
        swap(A[i], A[br]);
        swap(b[i], b[br]);
        swap(col[i], col[bc]);
        rep(j,0,n) if (A[j][i] != A[j][bc]) {
            A[j].flip(i); A[j].flip(bc);
        }
        rep(j,i+1,n) if (A[j][i]) {
            b[j] ^= b[i];
            A[j] ^= A[i];
        }
        rank++;
    }

    x = bs();
    for (int i = rank; i--;) {
        if (!b[i]) continue;
        x[col[i]] = 1;
        rep(j,0,i) b[j] ^= A[j][i];
    }
    return rank; // (multiple solutions if rank < m)
}
```

MatrixInverse.h
Description: Invert matrix A . Returns rank; result is stored in A unless singular ($\text{rank} < n$). Can easily be extended to prime moduli; for prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ where A^{-1} starts as the inverse of $A \pmod p$, and k is doubled in each step.
Time: $\mathcal{O}(n^3)$

ebfff6, 35 lines

```
int matInv(vector<vector<double>>& A) {
    int n = sz(A); vi col(n);
    vector<vector<double>> tmp(n, vector<double>(n));
    rep(i,0,n) tmp[i][i] = 1, col[i] = i;

    rep(i,0,n) {
        int r = i, c = i;
        rep(j,i,n) rep(k,i,n)
            if (fabs(A[j][k]) > fabs(A[r][c]))
                r = j, c = k;
        if (fabs(A[r][c]) < 1e-12) return i;
        A[i].swap(A[r]); tmp[i].swap(tmp[r]);
        rep(j,0,n)
            swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
        swap(col[i], col[c]);
        double v = A[i][i];
        rep(j,i+1,n) {
            double f = A[j][i] / v;
```

```
        A[j][i] = 0;
        rep(k,i+1,n) A[j][k] -= f*A[i][k];
        rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
    }
    rep(j,i+1,n) A[i][j] /= v;
    rep(j,0,n) tmp[i][j] /= v;
    A[i][i] = 1;
}

for (int i = n-1; i > 0; --i) rep(j,0,i) {
    double v = A[j][i];
    rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
}

rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
return n;
}
```

4.4 Fourier transforms

FastFourierTransformBer.cpp

Description: FFT5902c7, 38 lines

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

vector<ll> conv(const vector<ll>& a, const vector<ll>& b) {
    if (a.empty() || b.empty()) return {};
    vector<ll> res(a.size() + b.size() - 1);
    int L = 32 - __builtin_clz(res.size()), n = 1 << L;
    vector<C> in(n), out(n);
    copy(a.begin(), a.end(), begin(in));
    for (int i = 0; i < b.size(); i++) in[i].imag(b[i]);
    fft(in);
    for (C &x : in) x *= x;
    for (int i = 0; i < n; i++) out[i] = in[-i & (n - 1)] -
        conj(in[i]);
    fft(out);
    for (int i = 0; i < res.size(); i++) res[i] = round(1.0*
        imag(out[i]) / (4 * n));
    return res;
}
```


FastFourierTransformModBer.cpp

Description: FFTMOD	e1f846, 47 lines
<pre>typedef complex<double> C; void fft(vector<C>& a) { int n = a.size(), L = 31 - __builtin_clz(n); static vector<complex<long double>> R(2, 1); static vector<C> rt(2, 1); // (^ 10% faster if double) for (static int k = 2; k < n; k *= 2) { R.resize(n); rt.resize(n); auto x = polar(1.0L, acos(-1.0L) / k); for (int i = k; i < 2 * k; i++) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2]; } vector<ll> rev(n); for (int i = 0; i < n; i++) rev[i] = (rev[i / 2] (i & 1) << L) / 2; for (int i = 0; i < n; i++) if (i < rev[i]) swap(a[i], a[rev[i]]); for (int k = 1; k < n; k *= 2) { for (int i = 0; i < n; i += 2 * k) { for (int j = 0; j < k; j++) { C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled) a[i + j + k] = a[i + j] - z; a[i + j] += z; } } } } template<int M> vector<ll> convMod(const vector<ll> &a, const vector<ll> &b) { if (a.empty() b.empty()) return {}; vector<ll> res(a.size() + b.size() - 1); int B = 32 - __builtin_clz(res.size()), n = 1<<B, cut = int(sqrt(M)); vector<C> L(n), R(n), outs(n), outl(n); for (int i = 0; i < (int)a.size(); i++) L[i] = C((int)a[i] / cut, (int)a[i] % cut); for (int i = 0; i < (int)b.size(); i++) R[i] = C((int)b[i] / cut, (int)b[i] % cut); fft(L), fft(R); for (int i = 0; i < n; i++) { int j = -i & (n - 1); outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n); outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i; } fft(outl), fft(outs); for (int i = 0; i < (int)res.size(); i++) { ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5); ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5); res[i] = ((av % M * cut + bv) % M * cut + cv) % M; } return res; }</pre>	

NumberTheoreticTransformBer.cpp

Description: ntt(a) computes $\hat{f}(k) = \sum_x a[x]g^{xk}$ for all k , where $g = \text{root}^{(MOD-1)/N}$. N must be a power of 2. Useful for convolution modulo specific nice primes of the form $2^a b + 1$, where the convolution result has size at most 2^a . For arbitrary modulo, see FFTMod. conv(a, b) = c, where $c[x] = \sum a[i]b[x - i]$. For manual convolution: NTT the inputs, multiply pointwise, divide by n, reverse(start+1, end), NTT back. Inputs must be in [0, MOD).	
Time: $\mathcal{O}(N \log N)$	8c0f20, 36 lines
<pre>const ll MOD = (119 << 23) + 1, root = 62; // = 998244353 // For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21</pre>	

// and 483 << 21 (same root). The last two are > 10^9.

<pre>void ntt(vector<ll> &a) { int n = a.size(), L = 31 - __builtin_clz(n); static vector<ll> rt(2, 1); for (static int k = 2, s = 2; k < n; k *= 2, s++) { rt.resize(n); ll z[] = {1, fexp(root, MOD >> s)}; for (int i = k; i < 2 * k; i++) rt[i] = rt[i / 2] * z[i & 1] % MOD; } vector<int> rev(n); for (int i = 0; i < n; i++) rev[i] = (rev[i / 2] (i & 1) << L) / 2; for (int i = 0; i < n; i++) if (i < rev[i]) swap(a[i], a[rev[i]]]); for (int k = 1; k < n; k *= 2) { for (int i = 0; i < n; i += 2 * k) { for (int j = 0; j < k; j++) { ll z = rt[j + k] * a[i + j + k] % MOD, &ai = a[i + j]; a[i + j + k] = ai - z + (z > ai ? MOD : 0); ai += (ai + z >= MOD ? z - MOD : z); } } } } vector<ll> conv(const vector<ll> &a, const vector<ll> &b) { if (a.empty() b.empty()) return {}; int s = a.size() + b.size() - 1, B = 32 - __builtin_clz(s), n = 1 << B; int inv = fexp(n, MOD - 2); vector<ll> L(a), R(b), out(n); L.resize(n), R.resize(n); ntt(L), ntt(R); for (int i = 0; i < n; i++) out[-i & (n - 1)] = (ll)L[i] * R[i] % MOD * inv % MOD; ntt(out); return {out.begin(), out.begin() + s}; }</pre>	
--	--

FastSubsetTransform.h

Description: Transform to a basis with fast convolutions of the form $c[z] = \sum_{z=x\oplus y} a[x] \cdot b[y]$, where \oplus is one of AND, OR, XOR. The size of a must be a power of two.	
Time: $\mathcal{O}(N \log N)$	464cf3, 16 lines

<pre>void FST(vi& a, bool inv) { for (int n = sz(a), step = 1; step < n; step *= 2) { for (int i = 0; i < n; i += 2 * step) rep(j,i,i+step) { int &u = a[j], &v = a[j + step]; tie(u, v) = inv ? pii(v - u, u) : pii(v, u + v); // AND inv ? pii(v, u - v) : pii(u + v, u); // OR pii(u + v, u - v); // XOR } } if (inv) for (int& x : a) x /= sz(a); // XOR only } vi conv(vi a, vi b) { FST(a, 0); FST(b, 0); rep(i,0,sz(a)) a[i] *= b[i]; FST(a, 1); return a; }</pre>	
--	--

Number theory (5)

5.1 Modular arithmetic

ModInverse.h

Description: Pre-computation of modular inverses. Assumes LIM ≤ mod and that mod is a prime.

const ll mod = 1000000007, LIM = 200000; ll* inv = new ll[LIM] - 1; inv[1] = 1; rep(i,2,LIM) inv[i] = mod - (mod / i) * inv[mod % i] % mod;	6f684f, 3 lines
---	-----------------

ModPow.h

const ll mod = 1000000007; // faster if const	b83e45, 8 lines
---	-----------------

<pre>ll modpow(ll b, ll e) { ll ans = 1; for (; e; b = b * b % mod, e /= 2) if (e & 1) ans = ans * b % mod; return ans; }</pre>	
---	--

ModLog.h

Description: Returns the smallest $x > 0$ s.t. $a^x = b \pmod m$, or -1 if no such x exists. modLog(a,1,m) can be used to calculate the order of a .

Time: $\mathcal{O}(\sqrt{m})$	c040b8, 11 lines
-------------------------------	------------------

<pre>ll modLog(ll a, ll b, ll m) { ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1; unordered_map<ll, ll> A; while (j <= n && (e = f = e * a % m) != b % m) A[e * b % m] = j++; if (e == b % m) return j; if (__gcd(m, e) == __gcd(m, b)) rep(i,2,n+2) if (A.count(e = e * f % m)) return n * i - A[e]; return -1; }</pre>	
--	--

ModSum.h

Description: Sums of mod'ed arithmetic progressions. $\text{modsum}(to, c, k, m) = \sum_{i=0}^{to-1} (ki + c) \% m$. divsum is similar but for floored division.

Time: log(m), with a large constant.	5c5bc5, 16 lines
--------------------------------------	------------------

<pre>typedef unsigned long long ull; ull sumsq(ull to) { return to / 2 * ((to-1) 1); }</pre>	
--	--

<pre>ull divsum(ull to, ull c, ull k, ull m) { ull res = k / m * sumsq(to) + c / m * to; k %= m; c %= m; if (!k) return res; ull to2 = (to * k + c) / m; return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k); }</pre>	
--	--

<pre>ll modsum(ull to, ll c, ll k, ll m) { c = ((c % m) + m) % m; k = ((k % m) + m) % m; return to * c + k * sumsq(to) - m * divsum(to, c, k, m); }</pre>	
---	--

ModMulLL.h

Description: Calculate $a \cdot b \pmod c$ (or $a^b \pmod c$) for $0 \leq a, b \leq c \leq 7.2 \cdot 10^{18}$. Time: $\mathcal{O}(1)$ for modmul, $\mathcal{O}(\log b)$ for modpow

typedef unsigned long long ull; ull modmul(ull a, ull b, ull M) { ll ret = a * b - M * ull(1.L / M * a * b);	bbbd8f, 11 lines
--	------------------

```
    return ret + M * (ret < 0) - M * (ret >= (1l)M);
}

ull modpow(ull b, ull e, ull mod) {
    ull ans = 1;
    for (; e; b = modmul(b, b, mod), e /= 2)
        if (e & 1) ans = modmul(ans, b, mod);
    return ans;
}
```

ModSqrt.h
Description: Tonelli-Shanks algorithm for modular square roots. Finds x s.t. $x^2 = a \pmod{p}$ ($-x$ gives the other solution).
Time: $\mathcal{O}(\log^2 p)$ worst case, $\mathcal{O}(\log p)$ for most p

"ModPow.h"	19a793, 24 lines
------------	------------------

```
1l sqrt(1l a, 1l p) {
    a %= p; if (a < 0) a += p;
    if (a == 0) return 0;
    assert(modpow(a, (p-1)/2, p) == 1); // else no solution
    if (p % 4 == 3) return modpow(a, (p+1)/4, p);
    // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
    1l s = p - 1, n = 2;
    int r = 0, m;
    while (s % 2 == 0)
        ++r, s /= 2;
    while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
    1l x = modpow(a, (s + 1) / 2, p);
    1l b = modpow(a, s, p), g = modpow(n, s, p);
    for (; r = m) {
        1l t = b;
        for (m = 0; m < r && t != 1; ++m)
            t = t * t % p;
        if (m == 0) return x;
        1l gs = modpow(g, 1LL << (r - m - 1), p);
        g = gs * gs % p;
        x = x * gs % p;
        b = b * g % p;
    }
}
```

5.2 Primality

FastEratosthenes.h
Description: Prime sieve for generating all primes smaller than LIM.
Time: LIM=1e9 \approx 1.5s

	6b2912, 20 lines
--	------------------

```
const int LIM = 1e6;
bitset<LIM> isPrime;
vi eratosthenes() {
    const int S = (int)round(sqrt(LIM)), R = LIM / 2;
    vi pr = {2}, sieve(S+1); pr.reserve((int)(LIM/log(LIM)*1.1));
    vector<pii> cp;
    for (int i = 3; i <= S; i += 2) if (!sieve[i]) {
        cp.push_back({i, i * i / 2});
        for (int j = i * i; j <= S; j += 2 * i) sieve[j] = 1;
    }
    for (int L = 1; L <= R; L += S) {
        array<bool, S> block{};
        for (auto &[p, idx] : cp)
            for (int i=idx; i < S+L; idx = (i+=p)) block[i-L] = 1;
        rep(i,0,min(S, R - L))
            if (!block[i]) pr.push_back((L + i) * 2 + 1);
    }
    for (int i : pr) isPrime[i] = 1;
    return pr;
}
```

CrivoSegmentado.h

Description: Crivo para decompor numeros no range [L,R] , $R \leq 10^{12}$
Precisa precomputar os primos menores que sqrtR com outro crivo Apos o crivo, mark[i-l] tem 1 se i nao for primo, caso contrario, mark[i-l] eh um primo.
Time: $\mathcal{O}(R - L + \sqrt{RT}(R))$

	2773dc, 22 lines
--	------------------

```
const int N = 1e6 + 100; // N>=sqrt(MaxR)
bool mark[N];
vi pr;
void crivo_segmentado(1l l,1l r){
    //crivo(); jogando primos em pr
    for(1l i = 1;i<=r;i++){
        mark2[i-1] = i;
        ans[i-1] = 1;
    }
    for(1l p : pr)
        for(1l i = (1 + p -1)/p * p ;i<=r;i+=p){
            1l cnt =0;
            while(mark2[i-1]%p==0){
                mark2[i-1]/=p;
                cnt++;
            }
            ans[i-1] = ans[i-1]*(cnt + 1)%M; // exemplo -> qtd de
                divisores de i
        }
    for(1l i = 1;i<=r;i++)
        if(mark2[i-1]!=1)//eh um primo > Sqrt
            ans[i-1] = ans[i-1] * (1 + 1) %M;
}
```

MillerRabin.h
Description: Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to $7 \cdot 10^{18}$; for larger numbers, use Python and extend A randomly.
Time: 7 times the complexity of $a^b \pmod{c}$.

"ModMulLL.h"	60dcd1, 12 lines
--------------	------------------

```
bool isPrime(ull n) {
    if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
    ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
        s = __builtin_ctzll(n-1), d = n >> s;
    for (ull a : A) { // ^ count trailing zeroes
        ull p = modpow(a%n, d, n), i = s;
        while (p != 1 && p != n - 1 && a % n && i--)
            p = modmul(p, p, n);
        if (p != n-1 && i != s) return 0;
    }
    return 1;
}
```

Factor.h
Description: Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).
Time: $\mathcal{O}\left(n^{1/4}\right)$, less for numbers with small factors.

"ModMulLL.h", "MillerRabin.h"	a33cf6, 18 lines
-------------------------------	------------------

```
ull pollard(ull n) {
    auto f = [n](ull x) { return modmul(x, x, n) + 1; };
    ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
    while (t++ % 40 || __gcd(prd, n) == 1) {
        if (x == y) x = ++i, y = f(x);
        if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
        x = f(x), y = f(f(y));
    }
    return __gcd(prd, n);
}

vector<ull> factor(ull n) {
    if (n == 1) return {};
    if (isPrime(n)) return {n};
    ull x = pollard(n);
```

```
    auto l = factor(x), r = factor(n / x);
    l.insert(l.end(), all(r));
    return l;
}
```

5.3 Divisibility

euclid.h
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 $a \pmod{b}$.

	33ba8f, 5 lines
--	-----------------

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

CRT.h
Description: Chinese Remainder Theorem.
`crt(a, m, b, n)` computes x such that $x \equiv a \pmod{m}, x \equiv b \pmod{n}$. If $|a| < m$ and $|b| < n$, x will obey $0 \leq x < \text{lcm}(m, n)$. Assumes $mn < 2^{62}$.
Time: $\log(n)$

"euclid.h"	04d93a, 7 lines
------------	-----------------

```
1l crt(1l a, 1l m, 1l b, 1l n) {
    if (n > m) swap(a, b), swap(m, n);
    1l x, y, g = euclid(m, n, x, y);
    assert((a - b) % g == 0); // else no solution
    x = (b - a) % n * x % n / g * m + a;
    return x < 0 ? x + m*n/g : x;
}
```

5.3.1 Bézout’s identity

For $a \neq, b \neq 0$, then $d = \gcd(a, b)$ is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If (x, y) is one solution, then all solutions are given by

$$\left(x + \frac{kb}{\gcd(a,b)}, y - \frac{ka}{\gcd(a,b)}\right), \quad k \in \mathbb{Z}$$

phiFunction.h
Description: Euler’s ϕ function is defined as $\phi(n) := \#$ of positive integers $\leq n$ that are coprime with n . $\phi(1) = 1, p$ prime $\Rightarrow \phi(p^k) = (p - 1)p^{k-1}$, m, n coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$. If $n = p_1^{k_1}p_2^{k_2}...p_r^{k_r}$ then $\phi(n) = (p_1 - 1)p_1^{k_1-1}...(p_r - 1)p_r^{k_r-1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$. $\sum_{d|n} \phi(d) = n, \sum_{1 \leq k \leq n, \gcd(k, n) = 1} k = n\phi(n)/2, n > 1$.
Euler’s thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod{n}$.
Fermat’s little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod{p} \forall a$.

	cf7d6d, 8 lines
--	-----------------

```
const int LIM = 5000000;
int phi[LIM];

void calculatePhi() {
    rep(i,0,LIM) phi[i] = i&1 ? i : i/2;
    for (int i = 3; i < LIM; i += 2) if(phi[i] == i)
        for (int j = i; j < LIM; j += i) phi[j] -= phi[j] / i;
}
```

5.4 Fractions

FracBinarySearch.h

Description: Given f and N , finds the smallest fraction $p/q \in [0,1]$ such that $f(p/q)$ is true, and $p, q \leq N$. You may want to throw an exception from f if it finds an exact solution, in which case N can be removed.
Usage: `fracBS([])(Frac f) { return f.p>=3*f.q; }, 10); // {1,3}`
Time: $\mathcal{O}(\log(N))$

`struct` `Frac` { `ll` `p`, `q`; };

```
template<class F>
Frac fracBS(F f, ll N) {
    bool dir = 1, A = 1, B = 1;
    Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N]
    if (f(lo)) return lo;
    assert(f(hi));
    while (A || B) {
        ll adv = 0, step = 1; // move hi if dir, else lo
        for (int si = 0; step; (step *= 2) >= si) {
            adv += step;
            Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
            if (abs(mid.p) > N || mid.q > N || dir == !f(mid)) {
                adv -= step; si = 2;
            }
        }
        hi.p += lo.p * adv;
        hi.q += lo.q * adv;
        dir = !dir;
        swap(lo, hi);
        A = B; B = !!adv;
    }
    return dir ? hi : lo;
}
```

5.5 Pythagorean Triples

The Pythagorean triples are uniquely generated by

$$a = k \cdot (m^2 - n^2), \quad b = k \cdot (2mn), \quad c = k \cdot (m^2 + n^2),$$

with $m > n > 0$, $k > 0$, $m \perp n$, and either m or n even.

5.6 Primes

$p = 962592769$ is such that $2^{21} \mid p - 1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power p^a , except for $p = 2, a > 2$, and there are $\phi(\phi(p^a))$ many. For $p = 2, a > 2$, the group $\mathbb{Z}_{2^a}^\times$ is instead isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

5.7 Estimates

$$\sum_{d \mid n} d = O(n \log \log n).$$

The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200 000 for $n < 1e19$.

5.8 Mobius Function

$$\mu(n) = \begin{cases} 0 & n \text{ is not square free} \\ 1 & n \text{ has even number of prime factors} \\ -1 & n \text{ has odd number of prime factors} \end{cases}$$

Mobius Inversion:

$$g(n) = \sum_{d \mid n} f(d) \Leftrightarrow f(n) = \sum_{d \mid n} \mu(d) g(n/d)$$

IntPerm multinomial

Other useful formulas/forms:

$$\sum_{d \mid n} \mu(d) = [n = 1] \text{ (very useful)}$$

$$g(n) = \sum_{n \mid d} f(d) \Leftrightarrow f(n) = \sum_{n \mid d} \mu(d/n) g(d)$$

$$g(n) = \sum_{1 \leq m \leq n} f(\lfloor \frac{n}{m} \rfloor) \Leftrightarrow f(n) = \sum_{1 \leq m \leq n} \mu(m) g(\lfloor \frac{n}{m} \rfloor)$$

Combinatorial (6)

6.1 Permutations

6.1.1 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		

IntPerm.h

Description: Permutation -> integer conversion. (Not order preserving.)
Integer -> permutation can use a lookup table.
Time: $\mathcal{O}(n)$

```
int permToInt(vi& v) {
    int use = 0, i = 0, r = 0;
    for(int x:v) r = r * ++i + __builtin_popcount(use & -(1<<x)),
        use |= 1 << x; // (note: minus, not ~!)
    return r;
}
```

6.1.2 Cycles

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

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

6.1.3 Derangements

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

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

6.1.4 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

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

6.2 Partitions and subsets

6.2.1 Partition function

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

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

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	$\sim 2e5$	$\sim 2e8$

6.2.2 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 $\binom{n}{m} \equiv \prod_{i=0}^k \binom{n_i}{m_i} \pmod{p}$.

6.2.3 Binomials

$$\sum_{k \leq n} (-1)^k \binom{r}{k} = (-1)^n \binom{r-1}{n}$$

$$\sum_{j=0}^m \binom{n+j}{n} = \binom{n+m+1}{m}$$

$$\sum_{j=0}^n \binom{j}{m} = \binom{n+1}{m+1}$$

$$\sum_{j=0}^n j \binom{n}{j} = n 2^{n-1}$$

$$\sum_k \binom{r}{k} \binom{s}{n-k} = \binom{r+s}{n}$$

$$\sum_{i=0}^n \binom{n}{i}^2 = \binom{2n}{n}$$

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

$$E[X^c] = \sum_{k=0}^c \left\{ \begin{smallmatrix} c \\ k \end{smallmatrix} \right\} n^{\underline{k}} p^k \text{ where } n^{\underline{k}} = n(n-1) \cdots (n-k+1).$$

multinomial.h

Description: Computes $\binom{k_1 + \cdots + k_n}{k_1, k_2, \dots, k_n} = \frac{(\sum k_i)!}{k_1!k_2! \dots k_n!}$.

```
11 multinomial(vi& v) {
    ll c = 1, m = v.empty() ? 1 : v[0];
    rep(i,1,sz(v)) rep(j,0,v[i])
        c = c * ++m / (j+1);
    return c;
}
```

6.3 General purpose numbers

6.3.1 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:

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

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

$$c(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$$

$$c(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \dots$$

6.3.3 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. $\pi(j) > j$.

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

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

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

6.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.3.5 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,

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

6.3.6 Labeled unrooted trees

on n vertices: n^{n-2}

on k existing trees of size n_i : $n_1n_2 \cdots n_k n^{k-2}$

with degrees d_i : $(n-2)!/((d_1-1)! \cdots (d_n-1)!)$

6.3.7 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.

Graph (7)

7.1 Fundamentals

BellmanFord.h

Description: Calculates shortest paths from s in a graph that might have negative edge weights. Unreachable nodes get dist = inf; nodes reachable through negative-weight cycles get dist = -inf. Assumes $V^2 \max |w_i| < \sim 2^{63}$.
Time: $\mathcal{O}(VE)$

```
const ll inf = LLONG_MAX;
struct Ed { int a, b, w, s() { return a < b ? a : -a; } };
struct Node { ll dist = inf; int prev = -1; };

void bellmanFord(vector<Node>& nodes, vector<Ed>& eds, int s) {
    nodes[s].dist = 0;
    sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s(); });

    int lim = sz(nodes) / 2 + 2; // /3+100 with shuffled vertices
    rep(i,0,lim) for (Ed ed : eds) {
        Node cur = nodes[ed.a], &dest = nodes[ed.b];
        if (abs(cur.dist) == inf) continue;
        ll d = cur.dist + ed.w;
        if (d < dest.dist) {
            dest.prev = ed.a;
            dest.dist = (i < lim-1 ? d : -inf);
        }
    }
    rep(i,0,lim) for (Ed e : eds) {
        if (nodes[e.a].dist == -inf)
            nodes[e.b].dist = -inf;
    }
}
```

TrianglesSquare.cpp

Description: adj[i] -> todos os vizinhos (grafo nao direcionado) g[i] -> vizinhos de i tal que deg[i]>deg[j] ou deg[i] == deg[j] e i<j deg[i] -> grau int ok(int i,int j)return deg[i]==deg[j]?i<j:deg[i]>deg[j]; if(ok(i,j))g[i].pb(j);else g[j].pb(i);
Time: $\mathcal{O}(M\sqrt{tM})$

```
rep(i,0,n){
    // Quadrados:
    for(int to1 : g[i]){
        cnt[to1]=0;
        for(int to2 : adj[to1])
            cnt[to2]=0;
    }
    for(int to1 : g[i]){
        for(int to2 : adj[to1])if(ok(i,to2)){
            res+=cnt[to2]; // qtd de quadrados
            cnt[to2]++;
            if(res>=M)res-=M;
        }
    }
    // triangulo:
    for(int to : adj[i])vis[to]=1;
    for(int to : g[i])for(int to2 : g[to])if(vis[to2])res++;
    for(int to : adj[i])vis[to]=0;
}
```

FloydWarshall.h

Description: Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix m , where $m[i][j]$ = inf if i and j are not adjacent. As output, $m[i][j]$ is set to the shortest distance between i and j , inf if no path, or -inf if the path goes through a negative-weight cycle.

Time: $\mathcal{O}(N^3)$

```
const ll inf = 1LL << 62;
void floydWarshall(vector<vector<ll>>& m) {
    int n = sz(m);
    rep(i,0,n) m[i][i] = min(m[i][i], 0LL);
    rep(k,0,n) rep(i,0,n) rep(j,0,n)
        if (m[i][k] != inf && m[k][j] != inf) {
            auto newDist = max(m[i][k] + m[k][j], -inf);
            m[i][j] = min(m[i][j], newDist);
        }
    rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
        if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;
}
```

7.2 Network flow

MinCostMaxFlowBer.cpp

Description: declara mcmf<int ou ll> nome(qtd de nodes)

```
697b4c, 124 lines
/*
 * min_cost_flow(s, t, f) computa o par (fluxo, custo)
 * com max(fluxo) menor ou igual a f que tenha min(custo)
 * min_cost_flow(s, t) eh o Fluxo maximo de custo minimo de s
   pra t
 * Se for um dag, da pra substituir o SPFA por uma DP pra nao
 * para O(nm) no comeco
 * Se nao tiver aresta com custo negativo, nao precisa do SPFA
 * Time: O(nm + fm log n)
 */
template<typename T> struct mcmf {
    struct edge {
        int to, rev, flow, cap; // para, id da reversa, fluxo,
            capacidade
        bool res; // se eh reversa
        T cost; // custo da unidade de fluxo
    };
};
```

```
edge() : to(0), rev(0), flow(0), cap(0), cost(0), res(false) {}
edge(int to_, int rev_, int flow_, int cap_, T cost_, bool res_) : to(to_), rev(rev_), flow(flow_), cap(cap_), res(res_), cost(cost_) {}
};

vector<vector<edge>> g;
vector<int> par_idx, par;
T inf;
vector<T> dist;

mcmf(int n) : g(n), par_idx(n), par(n), inf(numeric_limits<T>::max()/3) {}

void add(int u, int v, int w, T cost) { // de u pra v com cap w e custo cost
    edge a = edge(v, g[v].size(), 0, w, cost, false);
    edge b = edge(u, g[u].size(), 0, 0, -cost, true);

    g[u].push_back(a);
    g[v].push_back(b);
}

vector<T> spfa(int s) { // nao precisa se nao tiver custo negativo
    deque<int> q;
    vector<bool> is_inside(g.size(), 0);
    dist = vector<T>(g.size(), inf);

    dist[s] = 0;
    q.push_back(s);
    is_inside[s] = true;

    while (!q.empty()) {
        int v = q.front();
        q.pop_front();
        is_inside[v] = false;

        for (int i = 0; i < g[v].size(); i++) {
            auto [to, rev, flow, cap, res, cost] = g[v][i];
            if (flow < cap and dist[v] + cost < dist[to]) {
                dist[to] = dist[v] + cost;

                if (is_inside[to]) continue;
                if (!q.empty() and dist[to] > dist[q.front()]) q.push_back(to);
                else q.push_front(to);
                is_inside[to] = true;
            }
        }
    }
    return dist;
}

bool dijkstra(int s, int t, vector<T>& pot) {
    priority_queue<pair<T, int>, vector<pair<T, int>>, greater<>> q;
    dist = vector<T>(g.size(), inf);
    dist[s] = 0;
    q.emplace(0, s);
    while (q.size()) {
        auto [d, v] = q.top();
        q.pop();
        if (dist[v] < d) continue;
        for (int i = 0; i < g[v].size(); i++) {
            auto [to, rev, flow, cap, res, cost] = g[v][i];
            cost += pot[v] - pot[to];
            if (flow < cap and dist[v] + cost < dist[to]) {
```

```
                dist[to] = dist[v] + cost;
                q.emplace(dist[to], to);
                par_idx[to] = i, par[to] = v;
            }
        }
    }
    return dist[t] < inf;
}

pair<int, T> min_cost_flow(int s, int t, int flow = INF) {
    vector<T> pot(g.size(), 0);
    pot = spfa(s); // mudar algoritmo de caminho minimo aqui

    int f = 0;
    T ret = 0;
    while (f < flow and dijkstra(s, t, pot)) {
        for (int i = 0; i < g.size(); i++)
            if (dist[i] < inf) pot[i] += dist[i];

        int mn_flow = flow - f, u = t;
        while (u != s){
            mn_flow = min(mn_flow, g[par[u]][par_idx[u]].cap - g[par[u]][par_idx[u]].flow);
            u = par[u];
        }

        ret += pot[t] * mn_flow;

        u = t;
        while (u != s) {
            g[par[u]][par_idx[u]].flow += mn_flow;
            g[u][g[par[u]][par_idx[u]].rev].flow -= mn_flow;
            u = par[u];
        }

        f += mn_flow;
    }

    return make_pair(f, ret);
}

// Opcional: retorna as arestas originais por onde passa flow = cap
vector<pair<int,int>> recover() {
    vector<pair<int,int>> used;
    for (int i = 0; i < g.size(); i++) for (edge e : g[i])
        if(e.flow == e.cap && !e.res) used.push_back({i, e.to});
    return used;
}

};
```

EdmondsKarp.h

Description: Flow algorithm with guaranteed complexity $O(VE^2)$. To get edge flow values, compare capacities before and after, and take the positive values only.

482fe0, 35 lines

```
template<class T> T edmondsKarp(vector<unordered_map<int, T>>& graph, int source, int sink) {
    assert(source != sink);
    T flow = 0;
    vi par(sz(graph)), q = par;

    for (;;) {
        fill(all(par), -1);
        par[source] = 0;
        int ptr = 1;
        q[0] = source;
```

```
        rep(i,0,ptr) {
            int x = q[i];
            for (auto e : graph[x]) {
                if (par[e.first] == -1 && e.second > 0) {
                    par[e.first] = x;
                    q[ptr++] = e.first;
                    if (e.first == sink) goto out;
                }
            }
        }
        return flow;
    out:
    T inc = numeric_limits<T>::max();
    for (int y = sink; y != source; y = par[y])
        inc = min(inc, graph[par[y]][y]);

    flow += inc;
    for (int y = sink; y != source; y = par[y]) {
        int p = par[y];
        if ((graph[p][y] -= inc) <= 0) graph[p].erase(y);
        graph[y][p] += inc;
    }
}

DinicBer.cpp
Description: Grafo com capacidades 1 o tempo é  $O(\sqrt{n} \cdot m)$  INF ta pra int
Time:  $\mathcal{O}(\min(m * max_{flow}, n^2 m))$ 
f7fa46, 82 lines

struct dinic {
    const bool scaling = false;
    int lim;
    struct edge {
        int to, cap, rev, flow;
        bool res;
        edge(int to_, int cap_, int rev_, bool res_) : to(to_), cap(cap_), rev(rev_), flow(0), res(res_) {}
    };

    vector<vector<edge>> g;
    vector<int> lev, beg;
    ll F;
    dinic(int n) : g(n), F(0) {}

    void add(int a, int b, int c) {
        g[a].emplace_back(b, c, g[b].size(), false);
        g[b].emplace_back(a, 0, g[a].size()-1, true);
    }

    bool bfs(int s, int t) {
        lev = vector<int>(g.size(), -1); lev[s] = 0;
        beg = vector<int>(g.size(), 0);
        queue<int> q; q.push(s);
        while (q.size()) {
            int u = q.front(); q.pop();
            for (auto& i : g[u]) {
                if (lev[i.to] != -1 or (i.flow == i.cap)) continue;
                if (scaling and i.cap - i.flow < lim) continue;
                lev[i.to] = lev[u] + 1;
                q.push(i.to);
            }
        }
        return lev[t] != -1;
    }

    int dfs(int v, int s, int f = INF) {
        if (!f or v == s) return f;
        for (int& i = beg[v]; i < g[v].size(); i++) {
            auto& e = g[v][i];
            if (lev[e.to] != lev[v] + 1) continue;
```

```
int foi = dfs(e.to, s, min(f, e.cap - e.flow));
if (!foi) continue;
e.flow += foi, g[e.to][e.rev].flow -= foi;
return foi;
}
return 0;
}
ll max_flow(int s, int t) {
for (lim = scaling ? (1<<30) : 1; lim; lim /= 2)
while (bfs(s, t)) while (int ff = dfs(s, t)) F += ff;
return F;
}

// arestas com fluxo
vector<pii> flow_edges(int s, int t) {
max_flow(s, t);
vector<pii> ans;
int n = g.size();
for (int i = 0; i < n; i++) {
for (auto edge : g[i]) {
if (!edge.res && edge.flow)
ans.emplace_back(i, edge.to);
}
}
return ans;
}

// arestas no min cut
vector<pii> cut_edges(int s, int t) {
max_flow(s, t);
vector<pii> cut;
vector<int> vis(g.size(), 0), st = {s};
vis[s] = 1;
while (st.size()) {
int u = st.back(); st.pop_back();
for (auto e : g[u]) if (!vis[e.to] and e.flow < e.cap)
vis[e.to] = 1, st.push_back(e.to);
}
for (int i = 0; i < g.size(); i++) for (auto e : g[i])
if (vis[i] and !vis[e.to] and !e.res) cut.emplace_back(i, e.to);
return cut;
}
};
```

LowerBoundMaxFlow.cpp

Description: Max flow com lower bound nas arestas add(a, b, l, r): adiciona aresta de a pra b, onde precisa passar f de fluxo, l <= f <= r add(a, b, c): adiciona aresta de a pra b com capacidade c Mesma complexidade do Dinic INF tem que ser int, de preferencia 1e9

```
<bits/stdc++.h>
using namespace std;
using ll = long long;
using pii = pair<int, int>;
using pll = pair<ll, ll>;

mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count());

const int MOD = 1e9 + 7;
const int MAXN = 2e5 + 5;
const int INF = 1e9;

struct dinic {
const bool scaling = false;
int lim;
struct edge {
int to, cap, rev, flow;
bool res;
};
};
```

```
edge(int to_, int cap_, int rev_, bool res_)
: to(to_), cap(cap_), rev(rev_), flow(0), res(res_) {}
};

vector<vector<edge>> g;
vector<int> lev, beg;
ll F;
dinic(int n) : g(n), F(0) {}

void add(int a, int b, int c) {
g[a].emplace_back(b, c, g[b].size(), false);
g[b].emplace_back(a, 0, g[a].size()-1, true);
}

bool bfs(int s, int t) {
lev = vector<int>(g.size(), -1); lev[s] = 0;
beg = vector<int>(g.size(), 0);
queue<int> q; q.push(s);
while (q.size()) {
int u = q.front(); q.pop();
for (auto& i : g[u]) {
if (lev[i.to] != -1 or (i.flow == i.cap)) continue;
if (scaling and i.cap - i.flow < lim) continue;
lev[i.to] = lev[u] + 1;
q.push(i.to);
}
}
return lev[t] != -1;
}

int dfs(int v, int s, int f = INF) {
if (!f or v == s) return f;
for (int& i = beg[v]; i < g[v].size(); i++) {
auto& e = g[v][i];
if (lev[e.to] != lev[v] + 1) continue;
int foi = dfs(e.to, s, min(f, e.cap - e.flow));
if (!foi) continue;
e.flow += foi, g[e.to][e.rev].flow -= foi;
return foi;
}
return 0;
}

ll max_flow(int s, int t) {
for (lim = scaling ? (1<<30) : 1; lim; lim /= 2)
while (bfs(s, t)) while (int ff = dfs(s, t)) F += ff;
return F;
}
};

struct lb_max_flow : dinic {
vector<int> d;
lb_max_flow(int n) : dinic(n + 2), d(n, 0) {}
void add(int a, int b, int l, int r) {
d[a] -= l;
d[b] += l;
dinic::add(a, b, r - l);
}

void add(int a, int b, int c) {
dinic::add(a, b, c);
}

bool has_circulation() {
int n = d.size();

ll cost = 0;
for (int i = 0; i < n; i++) {
if (d[i] > 0) {
cost += d[i];
dinic::add(n, i, d[i]);
} else if (d[i] < 0) {
dinic::add(i, n+1, -d[i]);
}
}
};
```

```
}

return (dinic::max_flow(n, n+1) == cost);
}

bool has_flow(int src, int snk) {
dinic::add(snk, src, INF);
return has_circulation();
}

ll max_flow(int src, int snk) {
if (!has_flow(src, snk)) return -1;
dinic::F = 0;
return dinic::max_flow(src, snk);
}
};

int main() {
ios::sync_with_stdio(false);
cin.tie(0);

// recebo o numero de vertices e m arestas do tipo
// vai de a pra b e fluxo tem que estar entre [c, d]
// note que nao preciso adicionar uma source nem uma sink
// o algoritmo ja faz isso por mim
int n, m;
cin >> n >> m;
lb_max_flow ber(n);
map<pii, int> id;
vector<int> ans(m);
for (int i = 0; i < m; i++) {
int a, b, c, d;
cin >> a >> b >> c >> d;
a--, b--;
id[{a, b}] = i;
ans[i] = c;
ber.add(a, b, c, d);
}

// Se tem um fluxo que atende todas as exigencias
if (ber.has_circulation()) {
cout << "YES" << endl;

// Importante: n eh a qtd de vertices
for (int i = 0; i < n; i++) {
for (auto edge : ber.g[i]) {
if (edge.res) {
int j = edge.to;
if (j >= n) continue;
ans[id[{j, i}]] -= edge.flow;
}
}
}

// imprime fluxo em cada aresta
for (int x : ans) cout << x << '\n';
}
else {
cout << "NO\n";
}
return 0;
}
};
```

GlobalMinCut.h

Description: Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.

```
Time: O(V^3)

pair<int, vi> globalMinCut(vector<vi> mat) {
pair<int, vi> best = {INT_MAX, {}};
int n = sz(mat);
};
```

```
vector<vi> co(n);
rep(i,0,n) co[i] = {i};
rep(ph,1,n) {
    vi w = mat[0];
    size_t s = 0, t = 0;
    rep(it,0,n-ph) { // O(V^2) -> O(E log V) with prio. queue
        w[t] = INT_MIN;
        s = t, t = max_element(all(w)) - w.begin();
        rep(i,0,n) w[i] += mat[t][i];
    }
    best = min(best, {w[t] - mat[t][t], co[t]});
    co[s].insert(co[s].end(), all(co[t]));
    rep(i,0,n) mat[s][i] += mat[t][i];
    rep(i,0,n) mat[i][s] = mat[s][i];
    mat[0][t] = INT_MIN;
}
return best;
}
```

GomoryHu.h

Description: Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.
Time: $\mathcal{O}(V)$ Flow Computations

```
"PushRelabel.h" 0418b3, 13 lines
typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
    vector<Edge> tree;
    vi par(N);
    rep(i,1,N) {
        PushRelabel D(N); // Dinic also works
        for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
        tree.push_back({i, par[i], D.calc(i, par[i])});
        rep(j,i+1,N)
            if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
    }
    return tree;
}
```

7.3 Matching

BipartiteEdgeCover.cpp

Description: Edge Cover. Requires to use fast kuhn beforehand Returns the set of edges such that every vertex of the graph the graph is incident to at least one edge of the set size = number of not isolated vertices - maximum matching
Time: $\mathcal{O}(VE)$ but pretty fast actually. random shuffle?

```
vector<pii> bipartiteEdgeCover(){
    solve();
    vi aux = match;
    vector<pii> res;
    rep(i,1,n+1){
        if(aux[i]){
            for(auto w : g[i]){
                if(aux[w]==i) res.pb(pii(i,w));
                else if(aux[w]==0) res.pb(pii(i,w)), aux[w]=-1;
            }
        }
    }
    rep(i,n+1,n+m+1){
        if(aux[i]){
            for(auto w : g[i]){
                if(!aux[w]) res.pb(pii(w,i)), aux[w]=1;
            }
        }
    }
    return res;
}
```

MinimumVertexCover.h

Description: Finds a minimum vertex cover in a bipartite graph. The size is the same as the size of a maximum matching, and the complement is a maximum independent set.

```
"DFSMatching.h" da4196, 20 lines
vi cover(vector<vi>& g, int n, int m) {
    vi match(m, -1);
    int res = dfsMatching(g, match);
    vector<bool> lfound(n, true), seen(m);
    for (int it : match) if (it != -1) lfound[it] = false;
    vi q, cover;
    rep(i,0,n) if (lfound[i]) q.push_back(i);
    while (!q.empty()) {
        int i = q.back(); q.pop_back();
        lfound[i] = 1;
        for (int e : g[i]) if (!seen[e] && match[e] != -1) {
            seen[e] = true;
            q.push_back(match[e]);
        }
    }
    rep(i,0,n) if (!lfound[i]) cover.push_back(i);
    rep(i,0,m) if (seen[i]) cover.push_back(n+i);
    assert(sz(cover) == res);
    return cover;
}
```

Hungarian.h

Description: Resolve o problema de assignment (matriz n x n) colocar os valores da matriz em 'a' (pode < 0) assignment() retorna um par com o valor do assignment minimo, e a coluna escolhida por cada linha
Time: $\mathcal{O}(n^3)$

```
template<typename T> struct hungarian {
    int n;
    vector<vector<T>>> a;
    vector<T> u, v;
    vector<int> p, way;
    T inf;

    hungarian(int n_) : n(n_), u(n+1), v(n+1), p(n+1), way(n+1) {
        a = vector<vector<T>>>(n, vector<T>(n));
        inf = numeric_limits<T>::max();
    }
    pair<T, vector<int>>> assignment() {
        for (int i = 1; i <= n; i++) {
            p[0] = i;
            int j0 = 0;
            vector<T> minv(n+1, inf);
            vector<int> used(n+1, 0);
            do {
                used[j0] = true;
                int i0 = p[j0], j1 = -1;
                T delta = inf;
                for (int j = 1; j <= n; j++) if (!used[j]) {
                    T cur = a[i0-1][j-1] - u[i0] - v[j];
                    if (cur < minv[j]) minv[j] = cur, way[j] = j0;
                    if (minv[j] < delta) delta = minv[j], j1 = j;
                }
                for (int j = 0; j <= n; j++)
                    if (used[j]) u[p[j]] += delta, v[j] -= delta;
                else minv[j] -= delta;
                j0 = j1;
            } while (p[j0] != 0);
            do {
                int j1 = way[j0];
                p[j0] = p[j1];
                j0 = j1;
            } while (j0);
        }
    }
}
```

```
vector<int> ans(n);
for (int j = 1; j <= n; j++) ans[p[j]-1] = j-1;
return make_pair(-v[0], ans);
}
};
```

GeneralMatching.h

Description: Matching for general graphs. Fails with probability N/mod .
Time: $\mathcal{O}(N^3)$

```
"../numerical/MatrixInverse-mod.h" cb1912, 40 lines
vector<pii> generalMatching(int N, vector<pii>& ed) {
    vector<vector<ll>>> mat(N, vector<ll>(N)), A;
    for (pii pa : ed) {
        int a = pa.first, b = pa.second, r = rand() % mod;
        mat[a][b] = r, mat[b][a] = (mod - r) % mod;
    }

    int r = matInv(A = mat), M = 2*N - r, fi, fj;
    assert(r % 2 == 0);

    if (M != N) do {
        mat.resize(M, vector<ll>(M));
        rep(i,0,N) {
            mat[i].resize(M);
            rep(j,N,M) {
                int r = rand() % mod;
                mat[i][j] = r, mat[j][i] = (mod - r) % mod;
            }
        }
    } while (matInv(A = mat) != M);

    vi has(M, 1); vector<pii> ret;
    rep(it,0,M/2) {
        rep(i,0,M) if (has[i])
            rep(j,i+1,M) if (A[i][j] && mat[i][j]) {
                fi = i; fj = j; goto done;
            }
        assert(0); done:
        if (fj < N) ret.emplace_back(fi, fj);
        has[fi] = has[fj] = 0;
        rep(sw,0,2) {
            ll a = modpow(A[fi][fj], mod-2);
            rep(i,0,M) if (has[i] && A[i][fj]) {
                ll b = A[i][fj] * a % mod;
                rep(j,0,M) A[i][j] = (A[i][j] - A[fi][j] * b) % mod;
            }
            swap(fi,fj);
        }
    }
    return ret;
}
```

7.4 DFS algorithms

SCC.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.
Usage: scc(graph, [&](vi& v) { ... }) visits all components in reverse topological order. comp[i] holds the component index of a node (a component only has edges to components with lower index). ncomps will contain the number of components.
Time: $\mathcal{O}(E + V)$

```
vi val, comp, z, cont;
int Time, ncomps;
template<class G, class F> int dfs(int j, G& g, F& f) {
    int low = val[j] = ++Time, x; z.push_back(j);
    for (auto e : g[j]) if (comp[e] < 0)
        low = min(low, val[e] ? dfs(e,g,f));
```

```
if (low == val[j]) {
    do {
        x = z.back(); z.pop_back();
        comp[x] = ncomps;
        cont.push_back(x);
    } while (x != j);
    f(cont); cont.clear();
    ncomps++;
}
return val[j] = low;
}
template<class G, class F> void scc(G& g, F f) {
    int n = sz(g);
    val.assign(n, 0); comp.assign(n, -1);
    Time = ncomps = 0;
    rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
}
```

BiconnectedComponents.h

Description: Finds all biconnected components in an undirected graph, and runs a callback for the edges in each. In a biconnected component there are at least two distinct paths between any two nodes. Note that a node can be in several components. An edge which is not in a component is a bridge, i.e., not part of any cycle.

Usage: int eid = 0; ed.resize(N);
for each edge (a,b) {
ed[a].emplace_back(b, eid);
ed[b].emplace_back(a, eid++); }
bicomps([&](const vi& edgelist) {...});

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

2965e5, 33 lines

```
vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
int dfs(int at, int par, F& f) {
    int me = num[at] = ++Time, e, y, top = me;
    for (auto pa : ed[at]) if (pa.second != par) {
        tie(y, e) = pa;
        if (num[y]) {
            top = min(top, num[y]);
            if (num[y] < me)
                st.push_back(e);
        } else {
            int si = sz(st);
            int up = dfs(y, e, f);
            top = min(top, up);
            if (up == me) {
                st.push_back(e);
                f(vi(st.begin() + si, st.end()));
                st.resize(si);
            }
            else if (up < me) st.push_back(e);
            else { /* e is a bridge */ }
        }
    }
    return top;
}

template<class F>
void bicomps(F f) {
    num.assign(sz(ed), 0);
    rep(i,0,sz(ed)) if (!num[i]) dfs(i, -1, f);
}
```

2SAT.cpp

Description: solve() retorna um par, o first fala se eh possivel atribuir, o second fala se cada variavel eh verdadeira $O(\neg V \neg + \neg E \neg) = O(\text{variaveis} + \text{restricoes})$

ef6b3b, 65 lines

```
struct sat {
    int n, tot;
    vector<vector<int>> g;
    vector<int> vis, comp, id, ans;
    stack<int> s;

    sat() {}
    sat(int n_ : n(n_), tot(n), g(2*n) {}

    int dfs(int i, int& t) {
        int lo = id[i] = t++;
        s.push(i), vis[i] = 2;
        for (int j : g[i]) {
            if (!vis[j]) lo = min(lo, dfs(j, t));
            else if (vis[j] == 2) lo = min(lo, id[j]);
        }
        if (lo == id[i]) while (1) {
            int u = s.top(); s.pop();
            vis[u] = 1, comp[u] = i;
            if ((u>>1) < n and ans[u>>1] == -1) ans[u>>1] = ~u&1;
            if (u == i) break;
        }
        return lo;
    }

    void add_impl(int x, int y) { // x-> y = !x ou y
        x = x >= 0 ? 2*x : -2*x-1;
        y = y >= 0 ? 2*y : -2*y-1;
        g[x].push_back(y);
        g[y^1].push_back(x^1);
    }

    void add_cl(int x, int y) { // x ou y
        add_impl(~x, y);
    }

    void add_xor(int x, int y) { // x xor y
        add_cl(x, y), add_cl(~x, ~y);
    }

    void add_eq(int x, int y) { // x = y
        add_xor(~x, y);
    }

    void add_true(int x) { // x = T
        add_impl(~x, x);
    }

    void at_most_one(vector<int> v) { // no max um verdadeiro
        g.resize(2*(tot+v.size()));
        for (int i = 0; i < v.size(); i++) {
            add_impl(tot+i, ~v[i]);
            if (i) {
                add_impl(tot+i, tot+i-1);
                add_impl(v[i], tot+i-1);
            }
        }
        tot += v.size();
    }

    pair<bool, vector<int>> solve() {
        ans = vector<int>(n, -1);
        int t = 0;
        vis = comp = id = vector<int>(2*tot, 0);
        for (int i = 0; i < 2*tot; i++) if (!vis[i]) dfs(i, t);
        for (int i = 0; i < tot; i++)
            if (comp[2*i] == comp[2*i+1]) return {false, {}};
        return {true, ans};
    }
}
```

```
};

EulerWalk.h
Description: Eulerian undirected/directed path/cycle algorithm. Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns a list of nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, add .second to s and ret.
Time:  $\mathcal{O}(V + E)$ 
780b64, 15 lines

vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src=0) {
    int n = sz(gr);
    vi D(n), its(n), eu(nedges), ret, s = {src};
    D[src]++; // to allow Euler paths, not just cycles
    while (!s.empty()) {
        int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
        if (it == end){ ret.push_back(x); s.pop_back(); continue; }
        tie(y, e) = gr[x][it++];
        if (!eu[e]) {
            D[x]--, D[y]++;
            eu[e] = 1; s.push_back(y);
        }
    }
    for (int x : D) if (x < 0 || sz(ret) != nedges+1) return {};
    return {ret.rbegin(), ret.rend()};
}

7.5 Coloring
EdgeColoring.h
Description: Given a simple, undirected graph with max degree D, computes a (D + 1)-coloring of the edges such that no neighboring edges share a color. (D-coloring is NP-hard, but can be done for bipartite graphs by repeated matchings of max-degree nodes.)
Time:  $\mathcal{O}(NM)$ 
e210e2, 31 lines

vi edgeColoring(int N, vector<pii> eds) {
    vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
    for (pii e : eds) ++cc[e.first], ++cc[e.second];
    int u, v, ncols = *max_element(all(cc)) + 1;
    vector<vi> adj(N, vi(ncols, -1));
    for (pii e : eds) {
        tie(u, v) = e;
        fan[0] = v;
        loc.assign(ncols, 0);
        int at = u, end = u, d, c = free[u], ind = 0, i = 0;
        while (d = free[v], !loc[d] && (v = adj[u][d]) != -1)
            loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
        cc[loc[d]] = c;
        for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at][cd])
            swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
        while (adj[fan[i]][d] != -1) {
            int left = fan[i], right = fan[++i], e = cc[i];
            adj[u][e] = left;
            adj[left][e] = u;
            adj[right][e] = -1;
            free[right] = e;
        }
        adj[u][d] = fan[i];
        adj[fan[i]][d] = u;
        for (int y : {fan[0], u, end})
            for (int& z = free[y] = 0; adj[y][z] != -1; z++);
    }
    rep(i,0,sz(eds))
        for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i];
    return ret;
}
```


7.6 Heuristics

MaximalCliques.h

Description: Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Callback is given a bitset representing the maximal clique.

Time: $\mathcal{O}\left(3^{n/3}\right)$, much faster for sparse graphs

```
typedef bitset<128> B;
template<class F>
void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B R={}) {
    if (!P.any()) { if (!X.any()) f(R); return; }
    auto q = (P | X).Find_first();
    auto cands = P & ~eds[q];
    rep(i,0,sz(eds)) if (cands[i]) {
        R[i] = 1;
        cliques(eds, f, P & eds[i], X & eds[i], R);
        R[i] = P[i] = 0; X[i] = 1;
    }
}
```

MaximumClique.h

Description: Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). Can be used to find a maximum independent set by finding a clique of the complement graph.

Time: Runs in about 1s for n=155 and worst case random graphs (p=.90). Runs faster for sparse graphs.

```
typedef vector<bitset<200>> vb;
struct Maxclique {
    double limit=0.025, pk=0;
    struct Vertex { int i, d=0; };
    typedef vector<Vertex> vv;
    vb e;
    vv V;
    vector<vi> C;
    vi qmax, q, S, old;
    void init(vv& r) {
        for (auto& v : r) v.d = 0;
        for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
        sort(all(r), [](auto a, auto b) { return a.d > b.d; });
        int mxD = r[0].d;
        rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
    }
    void expand(vv& R, int lev = 1) {
        S[lev] += S[lev - 1] - old[lev];
        old[lev] = S[lev - 1];
        while (sz(R)) {
            if (sz(q) + R.back().d <= sz(qmax)) return;
            q.push_back(R.back().i);
            vv T;
            for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i});
            if (sz(T)) {
                if (S[lev]++ / ++pk < limit) init(T);
                int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1, 1);
                C[1].clear(), C[2].clear();
                for (auto v : T) {
                    int k = 1;
                    auto f = [&](int i) { return e[v.i][i]; };
                    while (any_of(all(C[k]), f)) k++;
                    if (k > mxk) mxk = k, C[mxk + 1].clear();
                    if (k < mnk) T[j++].i = v.i;
                    C[k].push_back(v.i);
                }
                if (j > 0) T[j - 1].d = 0;
                rep(k,mnk,mxk + 1) for (int i : C[k])
                    T[j].i = i, T[j++].d = k;
                expand(T, lev + 1);
            } else if (sz(q) > sz(qmax)) qmax = q;
            q.pop_back(), R.pop_back();
        }
    }
};
```

```
    }
}
vi maxClique() { init(V), expand(V); return qmax; }
Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S) {
    rep(i,0,sz(e)) V.push_back({i});
}
};
```

MaximumIndependentSet.h

Description: To obtain a maximum independent set of a graph, find a max clique of the complement. If the graph is bipartite, see MinimumVertex-Cover.

TortoiseAndHare.cpp

Description: Tortoise and Hare Acha um ciclo com O(1) de memoria em O(n) Util para grafos funcionais implicitos em que vc calcula a resposta localmente e depois envia a resposta em O(1)

```
ll m = (111<<40);

ll prox(ll a) {
    return (a + (a>>20) + 12345) % m;
}

array<ll, 3> floyd() {

    // retornar a distancia ate o ciclo, inicio do ciclo e tamanho do ciclo
    array<ll, 3> ans = array<ll, 3>();

    // ponto de inicio
    ll slow = 1611516670, fast = 1611516670;

    // primeiro andamos ate que eles se encontrem
    do {
        slow = prox(slow);
        fast = prox(prox(fast));
    } while (slow != fast);

    // depois voltamos um deles para o comeco e andamos ate que eles se encontrem dnv
    slow = 1611516670;
    while (slow != fast) {
        ans[0]++;
        slow = prox(slow);
        fast = prox(fast);
    }

    // calcula o tamanho do ciclo
    ans[1] = slow;
    do {
        ans[2]++;
        slow = prox(slow);
    } while (slow != fast);
    return ans;
}
```

7.7 Trees

LCA.h

Description: Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.

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

```
"/data-structures/RMQ.h"
0f62fb, 21 lines

struct LCA {
    int T = 0;
```

```
vi time, path, ret;
RMQ<int> rmq;

LCA(vector<vi>& C) : time(sz(C)), rmq((dfs(C,0,-1), ret)) {}
void dfs(vector<vi>& C, int v, int par) {
    time[v] = T++;
    for (int y : C[v]) if (y != par) {
        path.push_back(v), ret.push_back(time[v]);
        dfs(C, y, v);
    }
}

int lca(int a, int b) {
    if (a == b) return a;
    tie(a, b) = minmax(time[a], time[b]);
    return path[rmq.query(a, b)];
}

//dist(a,b){return depth[a] + depth[b] - 2*depth[lca(a,b)];}
```

CompressTree.h

Description: Given a rooted tree and a subset S of nodes, compute the minimal subtree that contains all the nodes by adding all (at most |S| - 1) pairwise LCA's and compressing edges. Returns a list of (par, orig_index) representing a tree rooted at 0. The root points to itself.

Time: $\mathcal{O}(|S| \log |S|)$

```
"LCA.h"
9775a0, 21 lines

typedef vector<pair<int, int>> vpi;
vpi compressTree(LCA& lca, const vi& subset) {
    static vi rev; rev.resize(sz(lca.time));
    vi li = subset, &T = lca.time;
    auto cmp = [&](int a, int b) { return T[a] < T[b]; };
    sort(all(li), cmp);
    int m = sz(li)-1;
    rep(i,0,m) {
        int a = li[i], b = li[i+1];
        li.push_back(lca.lca(a, b));
    }
    sort(all(li), cmp);
    li.erase(unique(all(li)), li.end());
    rep(i,0,sz(li)) rev[li[i]] = i;
    vpi ret = {pii(0, li[0])};
    rep(i,0,sz(li)-1) {
        int a = li[i], b = li[i+1];
        ret.emplace_back(rev[lca.lca(a, b)], b);
    }
    return ret;
}
```

HLD.h

Description: Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most log(n) light edges. Code does additive modifications and max queries, but can support commutative segtree modifications/queries on paths and subtrees. Takes as input the full adjacency list. VALS_EDGES being true means that values are stored in the edges, as opposed to the nodes. All values initialized to the segtree default. Root must be 0.

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

```
"/data-structures/LazySegmentTree.h"
6f34db, 46 lines

template <bool VALS_EDGES> struct HLD {
    int N, tim = 0;
    vector<vi> adj;
    vi par, siz, depth, rt, pos;
    Node *tree;
    HLD(vector<vi> adj_)
        : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1), depth(N),
          rt(N), pos(N), tree(new Node(0, N)){ dfsSz(0); dfsHld(0); }
    void dfsSz(int v) {
```

```

    if (par[v] != -1) adj[v].erase(find(all(adj[v]), par[v]));
    for (int& u : adj[v]) {
        par[u] = v, depth[u] = depth[v] + 1;
        dfsSz(u);
        siz[v] += siz[u];
        if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
    }
}
void dfsHld(int v) {
    pos[v] = tim++;
    for (int u : adj[v]) {
        rt[u] = (u == adj[v][0] ? rt[v] : u);
        dfsHld(u);
    }
}
template <class B> void process(int u, int v, B op) {
    for (; rt[u] != rt[v]; v = par[rt[v]]) {
        if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
        op(pos[rt[v]], pos[v] + 1);
    }
    if (depth[u] > depth[v]) swap(u, v);
    op(pos[u] + VALS_EDGES, pos[v] + 1);
}
void modifyPath(int u, int v, int val) {
    process(u, v, [&](int l, int r) { tree->add(l, r, val); });
}
int queryPath(int u, int v) { // Modify depending on problem
    int res = -1e9;
    process(u, v, [&](int l, int r) {
        res = max(res, tree->query(l, r));
    });
    return res;
}
int querySubtree(int v) { // modifySubtree is similar
    return tree->query(pos[v] + VALS_EDGES, pos[v] + siz[v]);
}
};

```

LinkCutTree.h

Description: Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree.

Time: All operations take amortized $\mathcal{O}(\log N)$.

5909e2, 90 lines

```

struct Node { // Splay tree. Root's pp contains tree's parent.
    Node *p = 0, *pp = 0, *c[2];
    bool flip = 0;
    Node() { c[0] = c[1] = 0; fix(); }
    void fix() {
        if (c[0]) c[0]->p = this;
        if (c[1]) c[1]->p = this;
        // (+ update sum of subtree elements etc. if wanted)
    }
    void pushFlip() {
        if (!flip) return;
        flip = 0; swap(c[0], c[1]);
        if (c[0]) c[0]->flip ^= 1;
        if (c[1]) c[1]->flip ^= 1;
    }
    int up() { return p ? p->c[1] == this : -1; }
    void rot(int i, int b) {
        int h = i ^ b;
        Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
        if ((y->p = p)) p->c[up()] = y;
        c[i] = z->c[i ^ 1];
        if (b < 2) {
            x->c[h] = y->c[h ^ 1];
            z->c[h ^ 1] = b ? x : this;
        }
    }
};

```

```

y->c[i ^ 1] = b ? this : x;
fix(); x->fix(); y->fix();
if (p) p->fix();
swap(pp, y->pp);
}
void splay() {
    for (pushFlip(); p; ) {
        if (p->p) p->p->pushFlip();
        p->pushFlip(); pushFlip();
        int c1 = up(), c2 = p->up();
        if (c2 == -1) p->rot(c1, 2);
        else p->p->rot(c2, c1 != c2);
    }
}
Node* first() {
    pushFlip();
    return c[0] ? c[0]->first() : (splay(), this);
}
};

struct LinkCut {
    vector<Node> node;
    LinkCut(int N) : node(N) {}

    void link(int u, int v) { // add an edge (u, v)
        assert(!connected(u, v));
        makeRoot(&node[u]);
        node[u].pp = &node[v];
    }

    void cut(int u, int v) { // remove an edge (u, v)
        Node *x = &node[u], *top = &node[v];
        makeRoot(top); x->splay();
        assert(top == (x->pp ? x->c[0]));
        if (x->pp) x->pp = 0;
        else {
            x->c[0] = top->p = 0;
            x->fix();
        }
    }

    bool connected(int u, int v) { // are u, v in the same tree?
        Node* nu = access(&node[u])->first();
        return nu == access(&node[v])->first();
    }

    void makeRoot(Node* u) {
        access(u);
        u->splay();
        if (u->c[0]) {
            u->c[0]->p = 0;
            u->c[0]->flip ^= 1;
            u->c[0]->pp = u;
            u->c[0] = 0;
            u->fix();
        }
    }

    Node* access(Node* u) {
        u->splay();
        while (Node* pp = u->pp) {
            pp->splay(); u->pp = 0;
            if (pp->c[1]) {
                pp->c[1]->p = 0; pp->c[1]->pp = pp; }
            pp->c[1] = u; pp->fix(); u = pp;
        }
        return u;
    }
};

```

DirectedMST.h

Description: Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.

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

...data-structures/UnionFindRollback.h

39e620, 60 lines

```

struct Edge { int a, b; ll w; };
struct Node {
    Edge key;
    Node *l, *r;
    ll delta;
    void prop() {
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    }
    Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
    if (!a || !b) return a ? b;
    a->prop(), b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
}
void pop(Node& a) { a->prop(); a = merge(a->l, a->r); }

pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
    RollbackUF uf(n);
    vector<Node*> heap(n);
    for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node(e));
    ll res = 0;
    vi seen(n, -1), path(n), par(n);
    seen[r] = r;
    vector<Edge> Q(n), in(n, {-1,-1}), comp;
    deque<tuple<int, int, vector<Edge>>> cycs;
    rep(s, 0, n) {
        int u = s, qi = 0, w;
        while (seen[u] < 0) {
            if (!heap[u]) return {-1, {}};
            Edge e = heap[u]->top();
            heap[u]->delta -= e.w, pop(heap[u]);
            Q[qi] = e, path[qi++] = u, seen[u] = s;
            res += e.w, u = uf.find(e.a);
            if (seen[u] == s) {
                Node* cyc = 0;
                int end = qi, time = uf.time();
                do cyc = merge(cyc, heap[w = path[--qi]]);
                while (uf.join(u, w));
                u = uf.find(u), heap[u] = cyc, seen[u] = -1;
                cycs.push_front({u, time, {Q[qi], &Q[end]}});
            }
        }
        rep(i, 0, qi) in[uf.find(Q[i].b)] = Q[i];
    }

    for (auto& [u, t, comp] : cycs) { // restore sol (optional)
        uf.rollback(t);
        Edge inEdge = in[u];
        for (auto& e : comp) in[uf.find(e.b)] = e;
        in[uf.find(inEdge.b)] = inEdge;
    }
    rep(i, 0, n) par[i] = in[i].a;
    return {res, par};
}

```

TreeIsomorphism.cpp

Description: Acha se duas arvores enraizadas sao isomorficas Usa hashing Se nao forem enraizadas basta achar os centroides ai tenta ver se uma arvore enraizada em um centroide eh igual a outra enraizada em um dos 2 centroides

<bits/stdc++.h>5090f3, 53 lines

```
using namespace std;
using ll = long long;
using pii = pair<int, int>;
using pll = pair<ll, ll>;
```

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

```
const int MOD = 1e9 + 7;
const int MAXN = 1e5 + 5;
const ll INF = 2e18;
```

```
vector<int> v[2][MAXN];
int id = 0;
map<vector<int>, int> mp;
```

```
int dfs(int node, int p, bool at) {
    vector<int> val;
    for (int x : v[at][node]) {
        if (x != p) {
            val.push_back(dfs(x, node, at));
        }
    }
    sort(val.begin(), val.end());
    if (!mp[val]) mp[val] = ++id;
    return mp[val];
}
```

```
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int t;
    cin >> t;
    while (t--) {
        mp.clear();
        id = 0;
        int n, a, b;
        cin >> n;
        for (int i = 0; i < 2; i++) {
            for (int j = 1; j <= n; j++) {
                v[i][j].clear();
            }
            for (int j = 0; j < n - 1; j++) {
                cin >> a >> b;
                v[i][a].push_back(b);
                v[i][b].push_back(a);
            }
        }
        int s0 = dfs(1, -1, 0);
        int s1 = dfs(1, -1, 1);
        cout << (s0 == s1 ? "YES\n" : "NO\n");
    }
    return 0;
}
```

TreeIsomorphism

7.8 Math

7.8.1 Number of Spanning Trees

Create an $N \times N$ matrix mat , and for each edge $a \rightarrow b \in G$, do $\text{mat}[a][b]--$, $\text{mat}[b][b]++$ (and $\text{mat}[b][a]--$, $\text{mat}[a][a]++$ if G is undirected). Remove the i th row and column and take the determinant; this yields the number of directed spanning trees rooted at i (if G is undirected, remove any row/column).

7.8.2 Erdős–Gallai theorem
A simple graph with node degrees $d_1 \geq \dots \geq d_n$ exists iff $d_1 + \dots + d_n$ is even and for every $k = 1 \dots n$,

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k).$$

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); }
    // 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-(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 << ")"; }
};
```

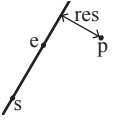
lineDistance.h

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.

"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();
}
```



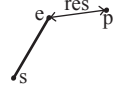
SegmentDistance.h

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;

"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;
}
```



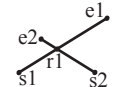
SegmentIntersection.h

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;

"Point.h", "OnSegment.h"9d57f2, 13 lines

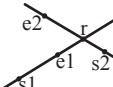
```
template<class P> vector<P> segInter(P a, P b, P c, P d) {
    auto oa = c.cross(d, a), ob = c.cross(d, b),
        oc = a.cross(b, c), od = a.cross(b, d);
    // Checks if intersection is single non-endpoint point.
    if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
        return {(a * ob - 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;
"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 . $1/0/-1 \Leftrightarrow$ 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;
"Point.h" 3af81c, 9 lines

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

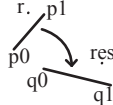
template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
    auto a = (e-s).cross(p-s);
    double l = (e-s).dist()*eps;
    return (a > l) - (a < -l);
}
```

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>.
"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.

"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();
}
```

LineProjectionReflection.h
Description: Projects point p onto line ab. Set refl=true to get reflection of point p across line ab insted. The wrong point will be returned if P is an integer point and the desired point doesn't have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow.
"Point.h" b5562d, 5 lines

```
template<class P>
P lineProj(P a, P b, P p, bool refl=false) {
    P v = b - a;
    return p - v.perp()* (1+refl)*v.cross(p-a)/v.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
"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 * (1l)b.x) <
        make_tuple(b.t, b.half(), a.x * (1l)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)};
}
```

```
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)};
}
```

SortByAngle.cpp
Description: Sort points by angle around a chosen pivot, untied by distance
Time: $\mathcal{O}(n \log n)$
"Point.h" fe7ecb, 7 lines

```
sort(all(pts) , [&](P a , P b){
    a = a - pivot , b = b - pivot;
    int hp1 = a < P(0,0) , hp2 = b < P(0,0);
    if(hp1 != hp2) return hp1 < hp2;
    if(a.cross(b) != 0) return a.cross(b) > 0;
    return a.dist2() < b.dist2();
});
```

8.2 Circles

CircleIntersection.h
Description: Computes the pair of points at which two circles intersect. Returns false in case of no intersection.
"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.
"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;
}
```

CircleLine.h
Description: Finds the intersection between a circle and a line. Returns a vector of either 0, 1, or 2 intersection points. P is intended to be Point<double>.
"Point.h" e0cfba, 9 lines

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

CirclePolygonIntersection.h
Description: Returns the area of the intersection of a circle with a ccw polygon.
Time: $\mathcal{O}(n)$
"../../../../content/geometry/Point.h" aleec63, 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 = p + d * t;
        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;
}
```



```
}

```

HullDiameter.h

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

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

"Point.h"	c571b8, 12 lines
<pre>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; }</pre>	

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

"Point.h", "sideOf.h", "OnSegment.h"	71446b, 14 lines
<pre>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; }</pre>	

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

"Point.h"	7cf45b, 39 lines
<pre>#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))</pre>	

<pre>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; }</pre>	
---	--

Minkowski.cpp

Description: Minkowski sum	b0e98b, 22 lines
<pre>void reorderPolygon(vector<P>& p){ int pos = 0; for(int i=1;i<sz(p);i++){ if(p[i].y < p[pos].y (p[i].y == p[pos].y && p[i].x < p[pos].x)){ pos = i; } } rotate(p.begin(),p.begin()+pos,p.end()); } vector<P> minkowski(vector<P> p, vector<P> q){ reorderPolygon(p), reorderPolygon(q); p.pb(p[0]),p.pb(p[1]),q.pb(q[0]),q.pb(q[1]); vector<P> result; int i=0,j=0; while(i<sz(p)-2 j<sz(q)-2){ result.pb(p[i] + q[j]); auto cross = (p[i+1] - p[i]).cross(q[j+1] - q[j]); if(cross >= 0)++i; if(cross <= 0)++j; } return result; }</pre>	

halfPlaneIntersec.cpp

Description: Finds the half plane intersection. If can be unbounded/non-convex May be useful to add a bounding box.

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

"Point.h", "lineIntersection.h"	27aab1, 56 lines
<pre>typedef Point<double> P; struct L { P a, b; double ang; L(){} L(P a, P b) : a(a), b(b) {} // reta para esquerda de a->b nessa ordem. }; double angle (L la) { return atan2(-(la.a.y - la.b.y), la.b.x - la.a.x); } const double inf = 1e100, eps = 1e-9;</pre>	

<pre>const double PI = acos(-1.0L); int cmp (double a, double b = 0) { if (abs(a-b) < eps) return 0; return (a < b) ? -1 : +1; } pair<int, P> lineInter(L la,L lb){ return lineInter(la.a,la.b,lb.a,lb.b); } bool check (L la, L lb, L lc) { P p = lineInter(lb, lc).ss; double det = (la.b - la.a).cross(p - la.a); return cmp(det) < 0; } const int N = 500100; int dq[2*N]; vector<P> hpi (vector<L> line) { rep(i,0,sz(line))line[i].ang = angle(line[i]); sort(line.begin(), line.end(), [&](L la,L lb){ if (cmp(la.ang, lb.ang) == 0) return (lb.b - lb.a). cross(la.b - lb.a) > eps; return cmp(la.ang,lb.ang) < 0; }); vector<L> pl(1, line[0]); rep(i,0,sz(line)) if (cmp(angle(line[i]), angle(pl.back())) != 0) pl.push_back(line[i]); int idl = N,idr = N; dq[idr++]=0; dq[idr++]=1; rep(i,2,sz(pl)) { while (idr - idl > 1 && check(pl[i], pl[dq[idr-1]], pl[dq[idr - 2]])) idr--; while (idr - idl > 1 && check(pl[i], pl[dq[idl]], pl[dq [idl+1]])) idl++; dq[idr++]=i; } while (idr - idl > 1 && check(pl[dq[idl]], pl[dq[idr-1]], pl[dq[idr - 2]])) idr--; while (idr - idl > 1 && check(pl[dq[idr-1]], pl[dq[idl]], pl[dq[idl + 1]])) idl++; vector<P> res; rep(i,idl,idr){ int nxt = (i + 1 == idr ? idl : i+1); auto inter = lineInter(pl[dq[i]], pl[dq[nxt]]); if (inter.ff!=1){//non-convex res.clear(); return res; } res.pb(inter.ss); } return res; }</pre>	
--	--

8.4 Misc. Point Set Problems

ClosestPair.h

Description: Finds the closest pair of points.

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

"Point.h"	ac41a6, 17 lines
<pre>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 + (1ll)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);</pre>	

```
    for (; lo != hi; ++lo)
        ret = min(ret, {( *lo - p).dist2(), { *lo, p } });
    S.insert(p);
}
return ret.second;
}
```

ManhattanMST.h

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

"Point.h"	df6f59, 23 lines
-----------	------------------

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

kdTree.h

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

"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);
    }
};
```

DelaunayTriangulation.h

Description: Computes the Delaunay triangulation of a set of points. Each circumcircle contains none of the input points. If any three points are collinear or any four are on the same circle, behavior is undefined.
Time: $\mathcal{O}(n^2)$

"Point.h", "3dHull.h"	c0e7bc, 10 lines
-----------------------	------------------

```
template<class P, class F>
void delaunay(vector<P>& ps, F trifu) {
    if (sz(ps) == 3) { int d = (ps[0].cross(ps[1], ps[2]) < 0);
        trifu(0,1+d,2-d); }
    vector<P3> p3;
    for (P p : ps) p3.emplace_back(p.x, p.y, p.dist2());
    if (sz(ps) > 3) for(auto t:hull3d(p3)) if ((p3[t.b]-p3[t.a]).
        cross(p3[t.c]-p3[t.a]).dot(P3(0,0,1)) < 0)
        trifu(t.a, t.c, t.b);
}
```

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], ... }, all counter-clockwise.
Time: $\mathcal{O}(n \log n)$

"Point.h"	eefdf5, 88 lines
-----------	------------------

```
typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t l1l; // (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?
    l1l 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.

<pre>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; }</pre>	3058c3, 6 lines
---	-----------------

Point3D.h

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

<pre>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; } };</pre>	8058ae, 32 lines
--	------------------

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: $O(n^2)$	5b45fc, 49 lines
-----------------------	------------------

<pre>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; }; };</pre>	611f07, 8 lines
---	-----------------

sphericalDistance.h

Description: Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_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.

<pre>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); }</pre>	
--	--

Strings (9)

KMP.h

Description: pi[x] computes the length of the longest prefix of s that ends at x, other than s[0...x] itself (abacaba -> 0010123). Can be used to find all occurrences of a string.

Time: $O(n)$	d4375c, 16 lines
---------------------	------------------

<pre>vi pi(const string& s) { vi p(sz(s)); rep(i,1,sz(s)) { int g = p[i-1]; while (g && s[i] != s[g]) g = p[g-1]; p[i] = g + (s[i] == s[g]); } return p; }</pre>	
--	--

<pre>vi match(const string& s, const string& pat) { vi p = pi(pat + '\0' + s), res; rep(i,sz(p)-sz(s),sz(p)) if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat)); return res; }</pre>	
---	--

Zfunc.h

Description: z[x] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)

Time: $O(n)$	ee09e2, 12 lines
---------------------	------------------

<pre>vi Z(const string& S) { vi z(sz(S)); int l = -1, r = -1; rep(l,1,sz(S)) { z[l] = i >= r ? 0 : min(r - i, z[i - l]); while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]]) z[i]++; if (i + z[i] > r) l = i, r = i + z[i]; } return z; }</pre>	
---	--

Manacher.h

Description: For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest odd (half rounded down).

Time: $O(N)$	e7ad79, 13 lines
---------------------	------------------

<pre>array<vi, 2> manacher(const string& s) { int n = sz(s); array<vi,2> p = {vi(n+1), vi(n)}; rep(z,0,2) for (int i=0,l=0,r=0; i < n; i++) { int t = r-i!+z; if (i<r) p[z][i] = min(t, p[z][l+t]); int L = i-p[z][i], R = i+p[z][i]-!z; while (L>=1 && R+1<n && s[L-1] == s[R+1]) p[z][i]++, L--, R++; if (R>r) l=L, r=R; } return p; }</pre>	
---	--

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string. **Usage:** rotate(v.begin(), v.begin()+minRotation(v), v.end());

Time: $O(N)$	d07a42, 8 lines
---------------------	-----------------

```
int minRotation(string s) {
```



```
int a=0, N=sz(s); s += s;
rep(b,0,N) rep(k,0,N) {
    if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1); break;}
    if (s[a+k] > s[b+k]) { a = b; break; }
}
return a;
}
```

SuffixArray.h

Description: Builds suffix array for a string. sa[i] is the starting index of the suffix which is i'th in the sorted suffix array. The returned vector is of size $n + 1$, and sa[0] = n. The lcp array contains longest common prefixes for neighbouring strings in the suffix array: lcp[i] = lcp(sa[i], sa[i-1]), lcp[0] = 0. The input string must not contain any zero bytes.
Time: $\mathcal{O}(n \log n)$

38db9f, 23 lines

```
struct SuffixArray {
    vi sa, lcp;
    SuffixArray(string& s, int lim=256) { // or basic_string<int>
        int n = sz(s) + 1, k = 0, a, b;
        vi x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
        sa = lcp = y, iota(all(sa), 0);
        for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
            p = j, iota(all(y), n - j);
            rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
            fill(all(ws), 0);
            rep(i,0,n) ws[x[i]]++;
            rep(i,1,lim) ws[i] += ws[i - 1];
            for (int i = n; i--;) sa[--ws[x[i]]] = y[i];
            swap(x, y), p = 1, x[sa[0]] = 0;
            rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
                (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
        }
        rep(i,1,n) rank[sa[i]] = i;
        for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
            for (k && k--, j = sa[rank[i] - 1];
                s[i + k] == s[j + k]; k++);
    }
};
```

Tandem.cpp

Description: Find all (i, p) such that $s.substr(i, p) == s.substr(i + p, p)$. No two intervals with the same period intersect or touch. getLCP must deal with cases $i == j$ (n-i) and $\max(i, j) >= n$ (return 0) Equivalent to normal lcp queries besides that
Time: $\mathcal{O}(N \log N)$

0f3453, 13 lines

```
vector<array<int,3>> solve(str s) {
    int N = sz(s); SuffixArray A,B;
    A.init(s); reverse(all(s)); B.init(s);
    vector<array<int,3>> runs;
    for (int p = 1; 2*p <= N; ++p) { // do in O(N/p) for period p
        for (int i = 0, lst = -1; i+p <= N; i += p) {
            int l = i-B.getLCP(N-i-p,N-i), r = i-p+A.getLCP(i,i+p);
            if (l > r || l == lst) continue;
            runs.pb({lst = l,r,p}); // for each i in [l,r],
        } // s.substr(i,p) == s.substr(i+p,p)
    }
    return runs;
}
```

SuffixTree.h

Description: Ukkonen's algorithm for online suffix tree construction. Each node contains indices [l, r] into the string, and a list of child nodes. Suffixes are given by traversals of this tree, joining [l, r] substrings. The root is 0 (has l = -1, r = 0), non-existent children are -1. To get a complete tree, append a dummy symbol – otherwise it may contain an incomplete path (still useful for substring matching, though).
Time: $\mathcal{O}(26N)$

aae0b8, 50 lines

```
struct SuffixTree {
    enum { N = 200010, ALPHA = 26 }; // N ~ 2*maxlen+10
    int toi(char c) { return c - 'a'; }
    string a; // v = cur node, q = cur position
    int t[N][ALPHA], l[N], r[N], p[N], s[N], v=0, q=0, m=2;

    void ukkadd(int i, int c) { suff:
        if (r[v] <= q) {
            if (t[v][c] == -1) { t[v][c] = m; l[m] = i;
                p[m++] = v; v = s[v]; q = r[v]; goto suff; }
            v = t[v][c]; q = l[v];
        }
        if (q == -1 || c == toi(a[q])) q++; else {
            l[m+1] = i; p[m+1] = m; l[m] = l[v]; r[m] = q;
            p[m] = p[v]; t[m][c] = m+1; t[m][toi(a[q])] = v;
            l[v] = q; p[v] = m; t[p[m]][toi(a[l[m]])] = m;
            v = s[p[m]]; q = l[m];
            while (q < r[m]) { v = t[v][toi(a[q])]; q += r[v] - l[v]; }
            if (q == r[m]) s[m] = v; else s[m] = m+2;
            q = r[v] - (q - r[m]); m += 2; goto suff;
        }
    }

    SuffixTree(string a) : a(a) {
        fill(r, r+N, sz(a));
        memset(s, 0, sizeof s);
        memset(t, -1, sizeof t);
        fill(t[1], t[1]+ALPHA, 0);
        s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p[1] = 0;
        rep(i,0,sz(a)) ukkadd(i, toi(a[i]));
    }

    // example: find longest common substring (uses ALPHA = 28)
    pii best;
    int lcs(int node, int i1, int i2, int olen) {
        if (l[node] <= i1 && i1 < r[node]) return 1;
        if (l[node] <= i2 && i2 < r[node]) return 2;
        int mask = 0, len = node ? olen + (r[node] - l[node]) : 0;
        rep(c,0,ALPHA) if (t[node][c] != -1)
            mask |= lcs(t[node][c], i1, i2, len);
        if (mask == 3)
            best = max(best, {len, r[node] - len});
        return mask;
    }
    static pii LCS(string s, string t) {
        SuffixTree st(s + (char)('z' + 1) + t + (char)('z' + 2));
        st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
        return st.best;
    }
};
```

Hashing.h

Description: Self-explanatory methods for string hashing. Useful primes: 1e9 + 7, 1e9 + 9, 998244353, 139, 137, 1e18 + 9

b9525a, 20 lines

```
struct hash_interval {
    ll c, mod;
    vector<ll> h, p;
    hash_interval(const string &s, ll c, ll mod)
        : c(c), mod(mod), h(sz(s) + 1), p(sz(s) + 1)
    {
        p[0] = 1;
        h[0] = 0;
        for (int i = 0; i < sz(s); i++)
        {
            h[i + 1] = (c * h[i] + s[i]) % mod;
            p[i + 1] = (c * p[i]) % mod;
        }
    }
};
```

```
}
// Returns hash of interval s[a...b] (where 0 <= a <= b < sz(s))
ll get(int a, int b)
{
    return (h[b + 1] - ((h[a] * p[b - a + 1]) % mod) + mod) %
        mod;
}
};
```

AhoCorasick.h

Description: Aho-Corasick automaton, used for multiple pattern matching. Initialize with AhoCorasick ac(patterns); the automaton start node will be at index 0. find(word) returns for each position the index of the longest word that ends there, or -1 if none. findAll(–, word) finds all words (up to $N\sqrt{N}$ many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input. For large alphabets, split each symbol into chunks, with sentinel bits for symbol boundaries.
Time: construction takes $\mathcal{O}(26N)$, where N = sum of length of patterns. find(x) is $\mathcal{O}(N)$, where N = length of x. findAll is $\mathcal{O}(NM)$.

f35677, 66 lines

```
struct AhoCorasick {
    enum {alpha = 26, first = 'A'}; // change this!
    struct Node {
        // (nmatches is optional)
        int back, next[alpha], start = -1, end = -1, nmatches = 0;
        Node(int v) { memset(next, v, sizeof(next)); }
    };
    vector<Node> N;
    vi backp;
    void insert(string& s, int j) {
        assert(!s.empty());
        int n = 0;
        for (char c : s) {
            int& m = N[n].next[c - first];
            if (m == -1) { n = m = sz(N); N.emplace_back(-1); }
            else n = m;
        }
        if (N[n].end == -1) N[n].start = j;
        backp.push_back(N[n].end);
        N[n].end = j;
        N[n].nmatches++;
    }
    AhoCorasick(vector<string>& pat) : N(1, -1) {
        rep(i,0,sz(pat)) insert(pat[i], i);
        N[0].back = sz(N);
        N.emplace_back(0);

        queue<int> q;
        for (q.push(0); !q.empty(); q.pop()) {
            int n = q.front(), prev = N[n].back;
            rep(i,0,alpha) {
                int &ed = N[n].next[i], y = N[prev].next[i];
                if (ed == -1) ed = y;
                else {
                    N[ed].back = y;
                    (N[ed].end == -1 ? N[ed].end : backp[N[ed].start])
                        = N[y].end;
                    N[ed].nmatches += N[y].nmatches;
                    q.push(ed);
                }
            }
        }
    }
    vi find(string word) {
        int n = 0;
        vi res; // ll count = 0;
        for (char c : word) {
```

```
n = N[n].next[c - first];
res.push_back(N[n].end());
// count += N[n].nmatches;
}
return res;
}
vector<vi> findAll(vector<string>& pat, string word) {
    vi r = find(word);
    vector<vi> res(sz(word));
    rep(i,0,sz(word)) {
        int ind = r[i];
        while (ind != -1) {
            res[i - sz(pat[ind]) + 1].push_back(ind);
            ind = backp[ind];
        }
    }
    return res;
}
};
```

Various (10)

10.1 Intervals

IntervalContainer.h
Description: Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).
Time: $\mathcal{O}(\log N)$

```
set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
    if (L == R) return is.end();
    auto it = is.lower_bound({L, R}), before = it;
    while (it != is.end() && it->first <= R) {
        R = max(R, it->second);
        before = it = is.erase(it);
    }
    if (it != is.begin() && (--it)->second >= L) {
        L = min(L, it->first);
        R = max(R, it->second);
        is.erase(it);
    }
    return is.insert(before, {L,R});
}
```

```
void removeInterval(set<pii>& is, int L, int R) {
    if (L == R) return;
    auto it = addInterval(is, L, R);
    auto r2 = it->second;
    if (it->first == L) is.erase(it);
    else (int&)it->second = L;
    if (R != r2) is.emplace(R, r2);
}
```

IntervalCover.h

Description: Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add || R.empty(). Returns empty set on failure (or if G is empty).
Time: $\mathcal{O}(N \log N)$

```
template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I) {
    vi S(sz(I)), R;
    iota(all(S), 0);
    sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
    T cur = G.first;
    int at = 0;
```

```
while (cur < G.second) { // (A)
    pair<T, int> mx = make_pair(cur, -1);
    while (at < sz(I) && I[S[at]].first <= cur) {
        mx = max(mx, make_pair(I[S[at]].second, S[at]));
        at++;
    }
    if (mx.second == -1) return {};
    cur = mx.first;
    R.push_back(mx.second);
}
return R;
}
```

ConstantIntervals.h

Description: Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.
Usage: constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});
Time: $\mathcal{O}(k \log \frac{n}{k})$

```
template<class F, class G, class T>
void rec(int from, int to, F& f, G& g, int& i, T& p, T q) {
    if (p == q) return;
    if (from == to) {
        g(i, to, p);
        i = to; p = q;
    } else {
        int mid = (from + to) >> 1;
        rec(from, mid, f, g, i, p, f(mid));
        rec(mid+1, to, f, g, i, p, q);
    }
}
template<class F, class G>
void constantIntervals(int from, int to, F f, G g) {
    if (to <= from) return;
    int i = from; auto p = f(i), q = f(to-1);
    rec(from, to-1, f, g, i, p, q);
    g(i, to, q);
}
```

10.2 Misc. algorithms

TernarySearch.h

Description: Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \dots < f(i) \geq \dots \geq f(b)$. To reverse which of the sides allows non-strict inequalities, change the $<$ marked with (A) to $<=$, and reverse the loop at (B). To minimize f , change it to $>$, also at (B).
Usage: int ind = ternSearch(0,n-1,[&](int i){return a[i];});
Time: $\mathcal{O}(\log(b-a))$

```
template<class F>
int ternSearch(int a, int b, F f) {
    assert(a <= b);
    while (b - a >= 5) {
        int mid = (a + b) / 2;
        if (f(mid) < f(mid+1)) a = mid; // (A)
        else b = mid+1;
    }
    rep(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
    return a;
}
```

LIS.h

Description: Compute indices for the longest increasing subsequence.
Time: $\mathcal{O}(N \log N)$

```
template<class I> vi lis(const vector<I>& S) {
    if (S.empty()) return {};
    vi prev(sz(S));
```

```
typedef pair<I, int> p;
vector<p> res;
rep(i,0,sz(S)) {
    // change 0 -> i for longest non-decreasing subsequence
    auto it = lower_bound(all(res), p{S[i], 0});
    if (it == res.end()) res.emplace_back(), it = res.end()-1;
    *it = {S[i], i};
    prev[i] = it == res.begin() ? 0 : (it-1)->second;
}
int L = sz(res), cur = res.back().second;
vi ans(L);
while (L--) ans[L] = cur, cur = prev[cur];
return ans;
}
```

FastKnapsack.h

Description: Given N non-negative integer weights w and a non-negative target t , computes the maximum $S \leq t$ such that S is the sum of some subset of the weights.
Time: $\mathcal{O}(N \max(w_i))$

```
int knapsack(vi w, int t) {
    int a = 0, b = 0, x;
    while (b < sz(w) && a + w[b] <= t) a += w[b++];
    if (b == sz(w)) return a;
    int m = *max_element(all(w));
    vi u, v(2*m, -1);
    v[a+m-t] = b;
    rep(i,b,sz(w)) {
        u = v;
        rep(x,0,m) v[x+w[i]] = max(v[x+w[i]], u[x]);
        for (x = 2*m; --x > m;) rep(j, max(0,u[x]), v[x])
            v[x-w[j]] = max(v[x-w[j]], j);
    }
    for (a = t; v[a+m-t] < 0; a--);
    return a;
}
```

StableMariage.h

Description: Match every element from A to B so that there is no pair x in A and y in B and x not paired with y s.t. y prefers x and x prefers y . $a[i]$ has the elements of B sorted by preference of i , $b[j]$ elements of A sorted by preference of j . —A— $<=$ —B—. Retorna um vetor v de tamanho —A— onde $v[i]$ guarda o match de i
Time: $\mathcal{O}(AB)$

```
vector<int> stable_marriage(vector<vector<int>>& a, vector<
    vector<int>>& b) {
    int n = a.size(), m = b.size();
    assert(a[0].size() == m and b[0].size() == n and n <= m);
    vector<int> match(m, -1), it(n, 0);
    vector inv_b(m, vector<int>(n));
    for (int i = 0; i < m; i++) for (int j = 0; j < n; j++)
        inv_b[i][b[i][j]] = j;
```

```
queue<int> q;
for (int i = 0; i < n; i++) q.push(i);
while (q.size()) {
    int i = q.front(); q.pop();
    int j = a[i][it[i]];

    if (match[j] == -1) match[j] = i;
    else if (inv_b[j][i] < inv_b[j][match[j]]) {
        q.emplace(match[j]);
        it[match[j]]++;
        match[j] = i;
    } else q.emplace(i), it[i]++;
}
```

```
vector<int> ret(n);
for (int i = 0; i < m; i++) if (match[i] != -1) ret[match[i]]
    = i;
return ret;
}
```

10.3 Dynamic programming

KnuthDP.h
Description: When doing DP on intervals: $a[i][j] = \min_{i < k < j} (a[i][k] + a[k][j]) + f(i, j)$, where the (minimal) optimal k increases with both i and j , one can solve intervals in increasing order of length, and search $k = p[i][j]$ for $a[i][j]$ only between $p[i][j - 1]$ and $p[i + 1][j]$. This is known as Knuth DP. Sufficient criteria for this are if $f(b, c) \leq f(a, d)$ and $f(a, c) + f(b, d) \leq f(a, d) + f(b, c)$ for all $a \leq b \leq c \leq d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.
Time: $\mathcal{O}(N^2)$

DivideAndConquerDP.h
Description: Given $a[i] = \min_{lo(i) \leq k < hi(i)} (f(i, k))$ where the (minimal) optimal k increases with i , computes $a[i]$ for $i = L..R - 1$.
Time: $\mathcal{O}((N + (hi - lo)) \log N)$

```
struct DP { // Modify at will:
    int lo(int ind) { return 0; }
    int hi(int ind) { return ind; }
    ll f(int ind, int k) { return dp[ind][k]; }
    void store(int ind, int k, ll v) { res[ind] = pii(k, v); }

    void rec(int L, int R, int LO, int HI) {
        if (L >= R) return;
        int mid = (L + R) >> 1;
        pair<ll, int> best(LLONG_MAX, LO);
        rep(k, max(LO, lo(mid)), min(HI, hi(mid)))
            best = min(best, make_pair(f(mid, k), k));
        store(mid, best.second, best.first);
        rec(L, mid, LO, best.second+1);
        rec(mid+1, R, best.second, HI);
    }
    void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

1D1D.h
Description: Let $w(i, j)$ be the cost function. The required condition is the quadrangle inequality: for all $i \leq j$, $w(i, j) + w(i + 1, j + 1) \leq w(i + 1, j) + w(i, j + 1)$ its easier to just list out a few values of w and check then prove. $dp[x] = \min(dp[k] + w(k, x))$ for all k from 0 to $x - 1$. Let $k(x)$ be the optimal index for $dp(x)$. Then for all $i \leq j$, $k(i) \leq k(j)$ because of the quadrangle inequality If we can't use CHT for this kind of DP optimization, May try this
Time: $\mathcal{O}(N \log N)$

```
dp[0] = 0;
vector<pair<int, int> > v; // (start pos, best k)
v.push_back(make_pair(0, 0));
for (int x = 1; x <= n; x++) {
int k = (--lower_bound(v.begin(), v.end(), make_pair(x + 1, 0))
    )->second;
dp[x] = dp[k] + w(k, x);
for (int i = (int)v.size() - 1; i >= 0; i--) {
    int y = v[i].first, oldk = v[i].second;
    if (y > x && dp[x] + w(x, y) < dp[oldk] + w(oldk, y)) v.
        pop_back();
    else {
        int l = y + 1, r = n + 1;
        while (l < r) {
            int mid = (l + r) / 2;
            if (dp[x] + w(x, mid) < dp[oldk] + w(oldk, mid)) r =
                mid;
```

```
        else l = mid + 1;
    }
    if (r != n + 1) v.push_back(make_pair(r, x));
    break;
}
}
if (v.size() == 0) v.push_back(make_pair(0, x));
}
cout << dp[n] << '\n';
```

10.4 Debugging tricks

- `signal(SIGSEGV, [](int) { _Exit(0); });`
converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). `_GLIBCXX_DEBUG` failures generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).
- `feenableexcept(29);` kills the program on NaNs (1), 0-divs (4), infinities (8) and denormals (16).

10.5 Optimization tricks
`__builtin_ia32_ldmxcsr(40896);` disables denormals (which make floats 20x slower near their minimum value).

- 10.5.1 Bit hacks**
- `c = x&-x, r = x+c; ((r^x) >> 2)/c` | r is the next number after x with the same number of bits set.
 - `rep(b, 0, K) rep(i, 0, (1 << K))`
if $(i \& 1 << b)$ $D[i] += D[i \wedge (1 << b)]$;
computes all sums of subsets.

- 10.5.2 Pragmas**
- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize loops and optimizes floating points better.
 - `#pragma GCC target ("avx2")` can double performance of vectorized code, but causes crashes on old machines.
 - `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

FastMod.h
Description: Compute $a \% b$ about 5 times faster than usual, where b is constant but not known at compile time. Returns a value congruent to $a \pmod b$ in the range $[0, 2b)$.

```
typedef unsigned long long ull;
struct FastMod {
    ull b, m;
    FastMod(ull b) : b(b), m(-1ULL / b) {}
    ull reduce(ull a) { // a % b + (0 or b)
        return a - (ull)((__uint128_t(m) * a) >> 64) * b;
    }
};
```

FastInput.h
Description: Read an integer from stdin. Usage requires your program to pipe in input from file.
Usage: `./a.out < input.txt`

Time: About 5x as fast as `cin/scanf`.
7b3c70, 17 lines

```
inline char gc() { // like getchar()
    static char buf[1 << 16];
    static size_t bc, be;
    if (bc >= be) {
        buf[0] = 0, bc = 0;
        be = fread(buf, 1, sizeof(buf), stdin);
    }
    return buf[bc++]; // returns 0 on EOF
}
```

```
int readInt() {
    int a, c;
    while ((a = gc()) < 40);
    if (a == '-') return -readInt();
    while ((c = gc()) >= 48) a = a * 10 + c - 480;
    return a - 48;
}
```

SmallPtr.h
Description: A 32-bit pointer that points into BumpAllocator memory.
2dd6c9, 10 lines

```
template<class T> struct ptr {
    unsigned ind;
    ptr(T* p = 0) : ind(p ? unsigned((char*)p - buf) : 0) {
        assert(ind < sizeof buf);
    }
    T& operator*() const { return *(T*)(buf + ind); }
    T* operator->() const { return &*this; }
    T& operator[](int a) const { return (&*this)[a]; }
    explicit operator bool() const { return ind; }
};
```