

The University of Austin at Texas

# UT Orange

Mark Wen, Caleb Hu, Aaryan Prakash

ICPC World Finals 2024

September 19, 2024

|    |                 |
|----|-----------------|
| 1  | Contest         |
| 2  | Mathematics     |
| 3  | Data structures |
| 4  | Numerical       |
| 5  | Number theory   |
| 6  | Combinatorial   |
| 7  | Graph           |
| 8  | Geometry        |
| 9  | Strings         |
| 10 | Various         |
| 11 | Extra Stuff     |

Contest (1)

template.cpp 14 lines

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

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

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

.bashrc 10 lines

```
run () {
    ok=1
    if [[ ! -f $1 || $1 -ot $1.cpp ]]
    then
        g++ $1.cpp -O2 -o $1 -std=c++17 -Wall -Wextra -Wshadow
            -Wconversion -fsanitize=undefined,address || ok=0
    fi
    [[ $ok -eq 1 ]] && ./$1
}

xmodmap -e 'clear Lock' -e 'keycode 0x42 = Escape'
```

.vimrc 6 lines

```
set cin aw ai is ts=4 sw=4 tm=50 rnu noeb bg=dark ru cul
mouse=a
sy on | no ; :
" Select region and then type :Hash to hash your selection.
" Useful for verifying that there aren't mistypes.
ca Hash w !cpp -dD -P -fpreprocessed \|| tr -d '[:space:]' \
```

1 \|| md5sum \|| cut -c-6

1 brute.sh 12 lines

```
#!/bin/zsh

sz=100
for ((i=1;;i++)); do
    echo "$i"
    ./gen "$i" "$sz" > input
    ./sol < input > output1
    ./brute < input > output2
    if (! diff output1 output2); then
        break
    fi
done
```

Mathematics (2)

2.1 Equations

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

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

2.2 Recurrences

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

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

Non-distinct roots  $r$  become polynomial factors, e.g.  $a_n = (d_1n + d_2)r^n$ .

2.3 Trigonometry

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

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

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

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

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

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

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

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

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

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

2.4 Geometry

2.4.1 Triangles

Circumradius:  $R = abc/4A$

Inradius:  $r = A/p$

Length of median (divides triangle into two equal-area triangles):  $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

Length of bisector (divides angles in two):

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

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

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

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

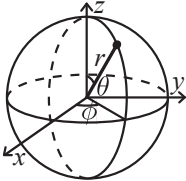
2.4.2 Quadrilaterals

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

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

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

2.4.3 Spherical coordinates



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

2.5 Derivatives/Integrals

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

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

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

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

Integration by parts:

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

## 2.6 Sums

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

$$\begin{aligned}1+2+3+\cdots+n&=\frac{n(n+1)}{2}\\1^2+2^2+3^2+\cdots+n^2&=\frac{n(2n+1)(n+1)}{6}\\1^3+2^3+3^3+\cdots+n^3&=\frac{n^2(n+1)^2}{4}\\1^4+2^4+3^4+\cdots+n^4&=\frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}\end{aligned}$$

## 2.7 Series

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

## 2.8 Probability theory

$$\begin{aligned}\sigma^2&=V(X)=\mathbb{E}(X^2)-(\mathbb{E}(X))^2 \\ \mathbb{E}(aX+bY)&=a\mathbb{E}(X)+b\mathbb{E}(Y) \\ \text{ind. } X,Y,V(aX+bY)&=a^2V(X)+b^2V(Y).\end{aligned}$$

### 2.8.1 Discrete distributions

#### Binomial distribution

$$\begin{aligned}p(k)&=\binom{n}{k}p^k(1-p)^{n-k} \\ \mu&=np, \, \sigma^2=np(1-p)\end{aligned}$$

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

#### Geometric distribution

$$\begin{aligned}p(k)&=p(1-p)^{k-1}, \, k=1,2,\ldots \\ \mu&=\frac{1}{p}, \, \sigma^2=\frac{1-p}{p^2}\end{aligned}$$

## OrderStatisticTree HashMap Matrix

### Poisson distribution

$$\begin{aligned}p(k)&=e^{-\lambda}\frac{\lambda^k}{k!}, k=0,1,2,\ldots \\ \mu&=\lambda, \, \sigma^2=\lambda\end{aligned}$$

### 2.8.2 Continuous distributions

#### Uniform distribution

$$\begin{aligned}f(x)&=\begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases} \\ \mu&=\frac{a+b}{2}, \, \sigma^2=\frac{(b-a)^2}{12}\end{aligned}$$

### Exponential distribution

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

### Normal distribution

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

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

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

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

## 2.9 Markov chains

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

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

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

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

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

## Data structures (3)

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

782797, 16 lines

```
#include <bits/extc++.h> //893
using namespace __gnu_pbds;

template<class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>; //988

void example() {
    Tree<int> t, t2; t.insert(8);
    auto it = t.insert(10).first;
    assert(it == t.lower_bound(9)); //6bd
    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
} //cbb
```

### HashMap.h

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

d77092, 7 lines

```
#include <bits/extc++.h> //1e4
// 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)
        ; }
}; //198
__gnu_pbds::gp_hash_table<ll,int,chash> h({},{},{},{},{
    1<<16});
```

### Matrix.h

**Description:** Basic operations on square matrices.  
**Usage:** Matrix<int, 3> A;  
A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};  
vector<int> vec = {1,2,3};  
vec = (A^N) \* vec;

c43c7d, 26 lines

```
template<class T, int N> struct Matrix { //1aa
    typedef Matrix M;
    array<array<T, N>, N> d{};
    M operator*(const M& m) const {
        M a;
        rep(i,0,N) rep(j,0,N) //683
            rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k][j];
    }
};
```

```
        return a;
    }
    vector<T> operator*(const vector<T>& vec) const {
        vector<T> ret(N);//9bd
        rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];
        return ret;
    }
    M operator^(ll p) const {
        assert(p >= 0);//358
        M a, b(*this);
        rep(i,0,N) a.d[i][i] = 1;
        while (p) {
            if (p&1) a = a*b;
            b = b*b;//1d8
            p >>= 1;
        }
        return a;
    }
};//214
```

### LineContainer.h

**Description:** Container where you can add lines of the form  $kx+m$ , and query maximum values at points  $x$ . Useful for dynamic programming (“convex hull trick”).  
**Time:**  $\mathcal{O}(\log N)$

Sec1c7, 30 lines

```
struct Line {//7e3
    mutable ll k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};//d77
struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }//66e
    bool isect(iterator x, iterator y) {
        if (y == end()) return x->p = inf, 0;
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;//bec
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y)
        );//890
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());//b07
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};
```

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

30f532, 55 lines

```
struct Node {//e9f
    Node *l = 0, *r = 0;
    int val, y, c = 1;
    Node(int v) : val(v), y(rand()) {}
    void recalc();
};//3ef
```

```
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) //5d5
    if (n) { each(n->l, f); f(n->val); each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
    if (!n) return {};//ca5
    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};//b54
    } else {
        auto pa = split(n->r, k - cnt(n->l) - 1); // and just "
            k"
        n->r = pa.first;
        n->recalc();
        return {n, pa.second};//86d
    }
}
```

```
Node* merge(Node* l, Node* r) {
    if (!l) return r;//fbf
    if (!r) return l;
    if (l->y > r->y) {
        l->r = merge(l->r, r);
        l->recalc();
        return l;//780
    } else {
        r->l = merge(l, r->l);
        r->recalc();
        return r;
    }//96d
}
```

```
Node* ins(Node* t, Node* n, int pos) {
    auto pa = split(t, pos);
    return merge(merge(pa.first, n), pa.second);//99b
}

// Example application: move the range [l, r) to index k
void move(Node*& t, int l, int r, int k) {
    Node *a, *b, *c;//99c
    tie(a,b) = split(t, l); tie(b,c) = split(b, r - 1);
    if (k <= l) t = merge(ins(a, b, k), c);
    else t = merge(a, ins(c, b, k - r));
}
```

### FenwickTree.h

**Description:** Computes partial sums  $a[0] + a[1] + \dots + a[\text{pos} - 1]$ , and updates single elements  $a[i]$ , taking the difference between the old and new value.  
**Time:** Both operations are  $\mathcal{O}(\log N)$ .

e62fac, 22 lines

```
struct FT {//711
    vector<ll> s;
    FT(int n) : s(n) {}
    void update(int pos, ll dif) { // a[pos] += dif
        for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
    }//cc4
    ll query(int pos) { // sum of values in [0, pos)
        ll res = 0;
        for (; pos > 0; pos &= pos - 1) res += s[pos-1];
        return res;
    }//477
    int lower_bound(ll sum) {// min pos st sum of [0, pos] >=
        sum
        // Returns n if no sum is >= sum, or -1 if empty sum is
        .
    }
```

```
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1) {//fc5
        if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
            pos += pw, sum -= s[pos-1];
    }
    return pos;
}//e03
};
```

### FenwickTree2d.h

**Description:** Computes sums  $a[i,j]$  for all  $i < I, j < J$ , and increases single elements  $a[i,j]$ . Requires that the elements to be updated are known in advance (call `fakeUpdate()` before `init()`).  
**Time:**  $\mathcal{O}(\log^2 N)$ . (Use persistent segment trees for  $\mathcal{O}(\log N)$ .)

"FenwickTree.h" 157f07, 22 lines

```
struct FT2 {//e22
    vector<vi> ys; vector<FT> ft;
    FT2(int limx) : ys(limx) {}
    void fakeUpdate(int x, int y) {
        for (; x < sz(ys); x |= x + 1) ys[x].push_back(y);
    }//57f
    void init() {
        for (vi& v : ys) sort(all(v)), ft.emplace_back(sz(v));
    }
    int ind(int x, int y) {
        return (int)(lower_bound(all(ys[x]), y) - ys[x].begin()
        ); }//358
    void update(int x, int y, ll dif) {
        for (; x < sz(ys); x |= x + 1)
            ft[x].update(ind(x, y), dif);
    }
    ll query(int x, int y) {//688
        ll sum = 0;
        for (; x; x &= x - 1)
            sum += ft[x-1].query(ind(x-1, y));
        return sum;
    }//e03
};
```

### 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> //722
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);//f6c
        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) {//a3d
        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)]);
    }
};//214
```

### MoQueries.h

**Description:** Answer interval or tree path queries by finding an approximate TSP through the queries, and moving from one query to the next by adding/removing points at the ends. If values are on tree edges, change step to add/remove the edge (a,c) and remove the initial add call (but keep in).

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

```

a12ef4, 49 lines
void add(int ind, int end) { ... } // add a[ind] (end = 0
    or 1)//342
void del(int ind, int end) { ... } // remove a[ind]
int calc() { ... } // compute current answer
```

```

vi mo(vector<pii> Q) {
    int L = 0, R = 0, blk = 350; // ~N/sqrt(Q)//cb0
    vi s(sz(Q)), res = s;
#define K(x) pii(x.first/blk, x.second ^ -(x.first/blk & 1)
    )
    iota(all(s), 0);
    sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]);
        });
    for (int qi : s) { //623
        pii q = Q[qi];
        while (L > q.first) add(--L, 0);
        while (R < q.second) add(R++, 1);
        while (L < q.first) del(L++, 0);
        while (R > q.second) del(--R, 1); //d22
        res[qi] = calc();
    }
    return res;
}
//842
vi moTree(vector<array<int, 2>> Q, vector<vi>& ed, int root
    =0){
    int N = sz(ed), pos[2] = {}, blk = 350; // ~N/sqrt(Q)
    vi s(sz(Q)), res = s, I(N), L(N), R(N), in(N), par(N);
    add(0, 0), in[0] = 1;
    auto dfs = [&](int x, int p, int dep, auto& f) -> void {
        //263
        par[x] = p;
        L[x] = N;
        if (dep) I[x] = N++;
        for (int y : ed[x]) if (y != p) f(y, x, !dep, f);
        if (!dep) I[x] = N++; //23e
        R[x] = N;
    };
    dfs(root, -1, 0, dfs);
#define K(x) pii(I[x[0]] / blk, I[x[1]] ^ -(I[x[0]] / blk &
    1))
    iota(all(s), 0); //064
    sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]);
        });
    for (int qi : s) rep(end,0,2) {
        int &a = pos[end], b = Q[qi][end], i = 0;
#define step(c) { if (in[c]) { del(a, end); in[a] = 0; } \
            else { add(c, end); in[c] = 1; } a = c; } \
        //440
        while (!(L[b] <= L[a] && R[a] <= R[b]))
            I[i++] = b, b = par[b];
        while (a != b) step(par[a]);
        while (i--) step(I[i]);
        if (end) res[qi] = calc(); //695
    }
    return res;
}
```

### NoamQueue.h

**Description:** Online queue-like deletion from a data structure supporting stack-like deletion

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

```

753397, 22 lines
struct update { //46a
    bool type;
```

```

    update() { type = 0; }
};
template<typename D, typename U>
struct noam : public D { //816
    vector<U> s;
    void push(const U &u) { D::push(u); s.push_back(u); }
    void pop() {
        auto i = s.end(); int c = 0;
        do { //31c
            c += (--i)->type ? 1 : -1; D::pop();
        } while (c < 0 && i != begin(s));
        auto j = stable_partition(i, s.end(), [](auto &x) {
            return !x.type; });
        if (i == begin(s)) {
            reverse(i, j); //cdd
            for_each(i, j, [](auto &x) { x.type = 1; });
        }
        s.pop_back();
        while (i != s.end()) D::push(*i), i++;
    } //e03
};
```

### OfflineDeletion.h

**Description:** Delete from a data structure given insertions and roll-backs ds needs: void push(U u), void pop(), Q query()

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

```

9b8b74, 32 lines
template<typename D, typename U, typename Q> //bf5
struct offline_deletion : public D {
    vector<Q> ans;
    vector<vector<U>> updates;
    int q;
    offline_deletion(int queries) : q(queries), ans(q) { //
        e0a
        int lg = 0; while((1 << lg) < q) lg++;
        updates.resize(1 << (lg + 1));
    }
    void update(int i, int l, int r, int L, int R, U u) {
        if (r < L || R < l) return; //4e2
        if (L <= l && r <= R) {
            updates[i].push_back(u); return;
        }
        int m = (l + r) / 2;
        update(2 * i + 1, l, m, L, R, u); //087
        update(2 * i + 2, m + 1, r, L, R, u);
    }
    void insert(U u, int l, int r) { update(0, 0, q - 1, l,
        r, u); }
    void insert(U u, int l) { insert(u, l, q - 1); }
    void solve(int i, int l, int r) { //fa5
        for (auto &u : updates[i]) D::push(u);
        if (l == r) ans[l] = D::query();
        else {
            int m = (l + r) / 2;
            solve(2 * i + 1, l, m); //eb8
            solve(2 * i + 2, m + 1, r);
        }
        for (int j = 0; j < (int) updates[i].size(); j++) D
            ::pop();
    }
    void solve() { solve(0, 0, q - 1); } //dbe
};
```

## Numerical (4)

### 4.1 Polynomials and recurrences

#### Polynomial.h

```

c9b7b0, 17 lines
struct Poly { //1b7
```

```

vector<double> a;
double operator()(double x) const {
    double val = 0;
    for (int i = sz(a); i--;) (val += x) += a[i];
    return val; //06d
}
void diff() {
    rep(i,1,sz(a)) a[i-1] = i*a[i];
    a.pop_back();
} //b82
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();
} //e03
};
```

### 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)
    { //840
    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); //9c1
    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]; //189
        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; //a7f
                else h = m;
            }
            ret.push_back((l + h) / 2);
        }
    } //808
    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; //159
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; //746
    rep(k,0,n) rep(i,0,n) {
        res[i] += y[k] * temp[i];
        swap(last, temp[i]);
        temp[i] -= last * x[k];
    } //0e1
    return res;
}
```

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:**  $O(N^2)$

```
"/..number-theory/ModPow.h" 96548b, 20 lines
vector<ll> berlekampMassey(vector<ll> s) { //b21
    int n = sz(s), L = 0, m = 0;
    vector<ll> C(n), B(n), T;
    C[0] = B[0] = 1;

    ll b = 1; //4c7
    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; //1b2
        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;
    }
    //255
    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:**  $O(n^2 \log k)$

```
f4e444, 26 lines
typedef vector<ll> Poly; //bb1
ll linearRec(Poly S, Poly tr, ll k) {
    int n = sz(tr);

    auto combine = [&](Poly a, Poly b) {
        Poly res(n * 2 + 1); //251
        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); //12f
        return res;
    };

    Poly pol(n + 1), e(pol);
    pol[0] = e[1] = 1; //df7

    for (++k; k; k /= 2) {
        if (k % 2) pol = combine(pol, e);
        e = combine(e, e);
    } //c0e

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

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 and no local maximum. The maximum error in the result is  $\epsilon$ . 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:**  $O(\log((b - a)/\epsilon))$

```
31d45b, 14 lines
double gss(double a, double b, double (*f)(double)) { //40b
    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 //707
            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); //ec9
        }
    return a;
}
```

HillClimbing.h

**Description:** Poor man's optimization for unimodal functions.

```
8ccca1, 14 lines
typedef array<double, 2> P; //68a

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) { //2dc
            P p = cur.second;
            p[0] += dx*jmp;
            p[1] += dy*jmp;
            cur = min(cur, make_pair(f(p), p));
        } //a63
    }
    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> //e93
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; //2d2
}
```

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; //e70
#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; //b17
    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);
} //836
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}};  
vd b = {1,1,-4}, c = {-1,-1}, x;  
T val = LPSolver(A, b, c).solve(x);  
**Time:**  $O(NM * \text{\#pivots})$ , where a pivot may be e.g. an edge relaxation.  $O(2^n)$  in the general case.

```
aa5530, 68 lines
typedef double T; // long double, Rational, double + modP
>... //629
typedef vector<T> vd;
typedef vector<vd> vvd;

const T eps = 1e-8, inf = 1/.0;
#define MP make_pair //94e
#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; //282
    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]; //108
            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;
        }
    //9c3
    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; //d0d
            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; //aa5
        swap(B[r], N[s]);
    }
}
```

```
bool simplex(int phase) {
    int x = m + phase - 1; //c51
    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; //bc0
        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
        ;
    } //00c
    if (r == -1) return false;
    pivot(r, s);
}
} //d2f
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); //f81
        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); //866
        }
    }
    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; //401
}
};
```

### 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) { //309
    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; //454
        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]; //07b
        }
    }
    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; //cab
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 //c65
                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; //bc6
            }
        }
        ans = ans * a[i][i] % mod;
        if (!ans) return 0;
    } //b19
    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; //2cf
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); //940
    vi col(m); iota(all(col), 0);

    rep(i,0,n) {
        double v, bv = 0;
        rep(r,i,n) rep(c,i,m) //ddb
            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; //de0
        }
        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]); //328
        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]; //af1
        }
        rank++;
    }

    x.assign(m, 0); //3c5
    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];
    } //807
    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) //22b
// ... 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]; //4e3
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; //d90

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); //2c9
    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; //13e
    }
    int bc = (int)A[br]._Find_next(i-1);
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]); //b88
    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]; //76c
        A[j] ^= A[i];
    }
    rank++;
} //7a7
x = bs();
for (int i = rank; i--;) {
    if (!b[i]) continue;
    x[col[i]] = 1;
    rep(j,0,i) b[j] ^= A[j][i]; //df7
}
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) { //9a9
    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) { //214
        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; //e5b
        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]; //afc
        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]; //c80
        }
        rep(j,i+1,n) A[i][j] /= v;
        rep(j,0,n) tmp[i][j] /= v;
        A[i][i] = 1;
    } //bfb

    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];
    } //e74

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

MatrixInverse-mod.h

**Description:** Invert matrix  $A$  modulo a prime. Returns rank; result is stored in  $A$  unless singular (rank < n). 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:  $\mathcal{O}(n^3)$ 

"../number-theory/ModPow.h"
0b7b13, 37 lines

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

    rep(i,0,n) { //79d
        int r = i, c = i;
        rep(j,i,n) rep(k,i,n) if (A[j][k]) {
            r = j; c = k; goto found;
        }
        return i; //4e3
    }
found:
    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]); //416
    ll v = modpow(A[i][i], mod - 2);
    rep(j,i+1,n) {
        ll f = A[j][i] * v % mod;
        A[j][i] = 0;
        rep(k,i+1,n) A[j][k] = (A[j][k] - f*A[i][k]) % mod; //9f7
        rep(k,0,n) tmp[j][k] = (tmp[j][k] - f*tmp[i][k]) % mod;
    }
    rep(j,i+1,n) A[i][j] = A[i][j] * v % mod;
    rep(j,0,n) tmp[i][j] = tmp[i][j] * v % mod;
    A[i][i] = 1; //e3d
}

for (int i = n-1; i > 0; --i) rep(j,0,i) {
    ll v = A[j][i];
    rep(k,0,n) tmp[j][k] = (tmp[j][k] - v*tmp[i][k]) % mod; //4b2
}

rep(i,0,n) rep(j,0,n)
    A[col[i]][col[j]] = tmp[i][j] % mod + (tmp[i][j] < 0) * mod;
return n; //400
}

```

Tridiagonal.h

**Description:**  $x = \text{tridiagonal}(d, p, q, b)$  solves the equation system

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} d_0 & p_0 & 0 & 0 & \cdots & 0 \\ 0 & d_1 & p_1 & 0 & \cdots & 0 \\ 0 & q_1 & d_2 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & q_{n-3} & d_{n-2} & p_{n-2} \\ 0 & 0 & \cdots & 0 & q_{n-2} & d_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{pmatrix}$$

This is useful for solving problems on the type

$$a_i = b_i a_{i-1} + c_i a_{i+1} + d_i, 1 \leq i \leq n,$$

where  $a_0, a_{n+1}, b_i, c_i$  and  $d_i$  are known.  $a$  can then be obtained from

$$\{a_i\} = \text{tridiagonal}(\{1, -1, -1, \dots, -1, 1\}, \{0, c_1, c_2, \dots, c_n\}, \{b_1, b_2, \dots, b_n, 0\}, \{a_0, d_1, d_2, \dots, d_n, a_{n+1}\}).$$

Fails if the solution is not unique.

If  $|d_i| > |p_i| + |q_{i-1}|$  for all  $i$ , or  $|d_i| > |p_{i-1}| + |q_i|$ , or the matrix is positive definite, the algorithm is numerically stable and neither `tr` nor the check for `diag[i] == 0` is needed.

```
Time:  $\mathcal{O}(N)$ 
8f9fa8, 26 lines

typedef double T; //399
vector<T> tridiagonal(vector<T> diag, const vector<T>& super,
    const vector<T>& sub, vector<T> b) {
    int n = sz(b); vi tr(n);
    rep(i,0,n-1) {
        if (abs(diag[i]) < 1e-9 * abs(super[i])) { // diag[i]
            == 0 //464
            b[i+1] -= b[i] * diag[i+1] / super[i];
            if (i+2 < n) b[i+2] -= b[i] * sub[i+1] / super[i];
            diag[i+1] = sub[i]; tr[++i] = 1;
        } else {
            diag[i+1] -= super[i]*sub[i]/diag[i]; //d50
            b[i+1] -= b[i]*sub[i]/diag[i];
        }
    }
    for (int i = n; i--;) {
        if (tr[i]) { //054
            swap(b[i], b[i-1]);
            diag[i-1] = diag[i];
            b[i] /= super[i-1];
        } else {
            b[i] /= diag[i]; //20b
            if (i) b[i-1] -= b[i]*super[i-1];
        }
    }
    return b;
} //cbb

```

4.4 Fourier transforms

FastFourierTransform.h

**Description:** `fft(a)` computes  $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$  for all  $k$ .  $N$  must be a power of 2. Useful for convolution: `conv(a, b) = c`, where  $c[x] = \sum a[i]b[x-i]$ . For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by `n`, `reverse(start+1, end)`, FFT back. Rounding is safe if  $(\sum a_i^2 + \sum b_i^2) \log_2 N < 9 \cdot 10^{14}$  (in practice  $10^{16}$ ; higher for random inputs). Otherwise, use NTT/FFT-Mod.

**Time:**  $\mathcal{O}(N \log N)$  with  $N = |A| + |B|$  ( $\sim 1s$  for  $N = 2^{22}$ )

```
00ced6, 35 lines

typedef complex<double> C; //1ec
typedef vector<double> vd;
void fft(vector<C>& a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1); // (^ 10% faster if double) //c50
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n); rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
    } //292
    vi rev(n);
    rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i,0,n) 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) rep(j,0,k) { //577
            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;
        }
    } //15f
    vd conv(const vd& a, const vd& b) {
        if (a.empty() || b.empty()) return {};
        vd res(sz(a) + sz(b) - 1);
        int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
        vector<C> in(n), out(n); //d93
        copy(all(a), begin(in));

```

```

        rep(i,0,sz(b)) in[i].imag(b[i]);
        fft(in);
        for (C& x : in) x *= x;
        rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]); //36e
        fft(out);
        rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
        return res;
    }

```

FastFourierTransformMod.h

**Description:** Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as  $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$  (in practice  $10^{16}$  or higher). Inputs must be in  $[0, \text{mod})$ .

**Time:**  $\mathcal{O}(N \log N)$ , where  $N = |A| + |B|$  (twice as slow as NTT or FFT)

```
"FastFourierTransform.h"
b82773, 22 lines

typedef vector<ll> vl; //2c4
template<int M> vl convMod(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    vl res(sz(a) + sz(b) - 1);
    int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M))
        ;
    vector<C> L(n), R(n), outs(n), outl(n); //c4f
    rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut)
        ;
    rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut)
        ;
    fft(L), fft(R);
    rep(i,0,n) {
        int j = -i & (n - 1); //3eb
        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);
    rep(i,0,sz(res)) { //58f
        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; //510
}

```

NumberTheoreticTransform.h

**Description:** `ntt(a)` computes  $\hat{f}(k) = \sum_x a[x]g^{xk}$  for all  $k$ , where  $g = \text{root}^{(\text{mod}-1)/N}$ .  $N$  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  $[0, \text{mod})$ .

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

```
"../number-theory/ModPow.h"
ced03d, 35 lines

const ll mod = (119 << 23) + 1, root = 62; // =
    998244353 //0ca
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
typedef vector<ll> vl;
void ntt(vl &a) {
    int n = sz(a), L = 31 - __builtin_clz(n); //cc5
    static vl rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
        rt.resize(n);
        ll z[] = {1, modpow(root, mod >> s)};
        rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod; //4a0
    }
    vi rev(n);
    rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);

```



```
    for (int k = 1; k < n; k *= 2)//ed7
    for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
        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);
    }//dfc
}
vl conv(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),
        n = 1 << B;//d58
    int inv = modpow(n, mod - 2);
    vl L(a), R(b), out(n);
    L.resize(n), R.resize(n);
    ntt(L), ntt(R);
    rep(i,0,n)//f18
        out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
    ntt(out);
    return {out.begin(), out.begin() + s};
}
```

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

```
void FST(vi& a, bool inv) {//ae8
    for (int n = sz(a), step = 1; step < n; step *= 2) {
        for (int i = 0; i < n; i += 2 * step) rep(j,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//0af
                pii(u + v, u - v); // XOR
        }
        if (inv) for (int& x : a) x /= sz(a); // XOR only
    }//dc4
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;
} //cbb
```

## Number theory (5)

### 5.1 Modular arithmetic

#### ModInverse.h

**Description:** Pre-computation of modular inverses. Assumes LIM  $\leq$  mod and that mod is a prime.

```
ll* inv = new ll[LIM] - 1; inv[1] = 1;//b4a
rep(i,2,LIM) inv[i] = mod - (mod / i) * inv[mod % i] % mod;
```

#### ModPow.h

```
const int mod = 1000000007; // faster if const//dce
```

```
ll modpow(ll b, ll e) {
    ll ans = 1;
    for (; e; b = b * b % mod, e /= 2)
        if (e & 1) ans = ans * b % mod;//7e5
    return ans;
}
```

#### ModLog.h

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

```
Time:  $\mathcal{O}(\sqrt{m})$ 
c040b8, 11 lines
ll modLog(ll a, ll b, ll m) {//260
    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;//d16
    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;
} //cbb
```

#### ModSum.h

**Description:** Sums of mod'ed arithmetic progressions. 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.

```
typedef unsigned long long ull; //df3
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }

ull divsum(ull to, ull c, ull k, ull m) {
    ull res = k / m * sumsq(to) + c / m * to;
    k %= m; c %= m; //e1a
    if (!k) return res;
    ull to2 = (to * k + c) / m;
    return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
} //1ae
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);
} //cbb
```

#### 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; //a9c
ull modmul(ull a, ull b, ull M) {
    ll ret = a * b - M * ull(1.L / M * a * b);
    return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {//51d
    ull ans = 1;
    for (; e; b = modmul(b, b, mod), e /= 2)
        if (e & 1) ans = modmul(ans, b, mod);
    return ans;
} //cbb
```

#### 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
ll sqrt(ll a, ll p) {//473
    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//a48
    ll 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; //c4b
ll x = modpow(a, (s + 1) / 2, p);
ll b = modpow(a, s, p), g = modpow(n, s, p);
for (; r = m) {
    ll t = b;
    for (m = 0; m < r && t != 1; ++m) //faf
        t = t * t % p;
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p; //a28
    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

```
const int LIM = 1e6; //058
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; //083
    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) {//62d
        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); //c68
    }
    for (int i : pr) isPrime[i] = 1;
    return pr;
}
```

#### 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) {//60a
    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; //81c
        while (p != 1 && p != n - 1 && a % n && i--)
            p = modmul(p, p, n);
        if (p != n-1 && i != s) return 0;
    }
    return 1; //84a
}
```

#### 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"
21a173, 18 lines

ull pollard(ull n) { //c81
    ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
    auto f = [&](ull a) { return modmul(a, a, n) + i; };
    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;
        //049
        x = f(x), y = f(f(y));
    }
    return __gcd(prd, n);
}

vector<ull> factor(ull n) { //c19
    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)); //363
    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

ll euclid(ll a, ll b, ll &x, ll &y) { //33b
    if (!b) return x = 1, y = 0, a;
    ll 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

ll crt(ll a, ll m, ll b, ll n) { //eae
    if (n > m) swap(a, b), swap(m, n);
    ll 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; //6ac
}
```

#### 5.3.1 Bézout’s identity

For  $a \neq 0, b \neq 0$ , 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  $\phi$  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; //70b
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) //103
        for (int j = i; j < LIM; j += i) phi[j] -= phi[j] / i;
}
```

### 5.4 Fractions

**ContinuedFractions.h**  
**Description:** Given  $N$  and a real number  $x \geq 0$ , finds the closest rational approximation  $p/q$  with  $p, q \leq N$ . It will obey  $|p/q - x| \leq 1/qN$ . For consecutive convergents,  $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$ . ( $p_k/q_k$  alternates between  $> x$  and  $< x$ .) If  $x$  is rational,  $y$  eventually becomes  $\infty$ ; if  $x$  is the root of a degree 2 polynomial the  $a$ ’s eventually become cyclic.  
**Time:**  $\mathcal{O}(\log N)$

```
ea42e2, 21 lines

typedef double d; // for N ~ 1e7; long double for N ~ 1e9
//32b
pair<ll, ll> approximate(d x, ll N) {
    ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y = x
    ;
    for (;) {
        ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q : inf
        ),
        a = (ll)floor(y), b = min(a, lim), //5ad
        NP = b*P + LP, NQ = b*Q + LQ;
        if (a > b) {
            // If b > a/2, we have a semi-convergent that gives
            us a
            // better approximation; if b = a/2, we *may* have
            one.
            // Return {P, Q} here for a more canonical
            approximation. //fcb
            return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (d)Q)
            ) ?
                make_pair(NP, NQ) : make_pair(P, Q);
        }
        if (abs(y = 1/(y - (d)a)) > (d)N*3) {
            return {NP, NQ}; //5c7
        }
        LP = P; P = NP;
        LQ = Q; Q = NQ;
    }
} //cbb
```

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

```
27ab3e, 25 lines

struct Frac { ll p, q; }; //386

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
    //262
    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) { //7e2
```

```
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;
    } //bf0
}
hi.p += lo.p * adv;
hi.q += lo.q * adv;
dir = !dir;
swap(lo, hi); //f58
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

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

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:

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

Other useful formulas/forms:

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

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

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

## 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 preserv-ing.) Integer -> permutation can use a lookup table.  
**Time:**  $\mathcal{O}(n)$

```
044568, 6 lines
int permToInt(vi& v) { //cf9
    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;
} //cbb
```

#### 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

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

### 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

multinomial.h  
**Description:** Computes  $\binom{k_1 + \dots + k_n}{k_1, k_2, \dots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \dots k_n!}$ . a0a312, 6 lines

```
11 multinomial(vi& v) { //efe
    11 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;
} //cbb
```

### 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.

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

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

#### 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.

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

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

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

#### 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,

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

#### 6.3.6 Labeled unrooted trees

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

#### 6.3.7 Catalan numbers

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

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

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

- 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  $\text{dist} = \text{inf}$ ; nodes reachable through negative-weight cycles get  $\text{dist} = -\text{inf}$ . Assumes  $V^2 \max |w_i| < \sim 2^{63}$ .  
**Time:**  $\mathcal{O}(VE)$

```
830a8f, 23 lines

const ll inf = LLONG_MAX; //019
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; //3a0
    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]; //e21
        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); //69b
        }
    }
    rep(i,0,lim) for (Ed e : eds) {
        if (nodes[e.a].dist == -inf)
            nodes[e.b].dist = -inf; //943
    }
}
```

**TopoSort.h**  
**Description:** Topological sorting. Given is an oriented graph. Output is an ordering of vertices, such that there are edges only from left to right. If there are cycles, the returned list will have size smaller than  $n$  – nodes reachable from cycles will not be returned.  
**Time:**  $\mathcal{O}(|V| + |E|)$

```
66a137, 14 lines

vi topoSort(const vector<vi>& gr) { //3ae
    vi indeg(sz(gr)), ret;
    for (auto& li : gr) for (int x : li) indeg[x]++;
    queue<int> q; // use priority_queue for lexic. largest ans.
    rep(i,0,sz(gr)) if (indeg[i] == 0) q.push(i);
    while (!q.empty()) { //ce0
        int i = q.front(); // top() for priority queue
        ret.push_back(i);
        q.pop();
        for (int x : gr[i])
            if (--indeg[x] == 0) q.push(x); //3dc
    }
    return ret;
}
```

## 7.2 Network flow

**PushRelabel.h**  
**Description:** Push-relabel using the highest label selection rule and the gap heuristic. Quite fast in practice. To obtain the actual flow, look at positive values only.

**Time:**  $\mathcal{O}(V^2\sqrt{E})$

```
0ae1d4, 48 lines

struct PushRelabel { //d82
    struct Edge {
        int dest, back;
        ll f, c;
    };
    vector<vector<Edge>> g; //bef
    vector<ll> ec;
    vector<Edge*> cur;
```

```
vector<vi> hs; vi H;
PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}
}
//07d
void addEdge(int s, int t, ll cap, ll rcap=0) {
    if (s == t) return;
    g[s].push_back({t, sz(g[t]), 0, cap});
    g[t].push_back({s, sz(g[s])-1, 0, rcap});
} //a02

void addFlow(Edge& e, ll f) {
    Edge &back = g[e.dest][e.back];
    if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
    e.f += f; e.c -= f; ec[e.dest] += f; //124
    back.f -= f; back.c += f; ec[back.dest] -= f;
}

ll calc(int s, int t) {
    int v = sz(g); H[s] = v; ec[t] = 1;
    vi co(2*v); co[0] = v-1; //a96
    rep(i,0,v) cur[i] = g[i].data();
    for (Edge& e : g[s]) addFlow(e, e.c);

    for (int hi = 0;;) {
        while (hs[hi].empty()) if (!hi--) return -ec[s]; //e2e
        int u = hs[hi].back(); hs[hi].pop_back();
        while (ec[u] > 0) // discharge u
            if (cur[u] == g[u].data() + sz(g[u])) {
                H[u] = 1e9;
                for (Edge& e : g[u]) if (e.c && H[u] > H[e.dest] + 1) //9ff
                    H[u] = H[e.dest] + 1, cur[u] = &e;
                if (++co[H[u]], !--co[hi] && hi < v)
                    rep(i,0,v) if (hi < H[i] && H[i] < v)
                        --co[H[i]], H[i] = v + 1;
                hi = H[u]; //7ed
            } else if (cur[u]->c && H[u] == H[cur[u]->dest] + 1)
                addFlow(*cur[u], min(ec[u], cur[u]->c));
            else ++cur[u];
        }
    } //a5b
    bool leftOfMinCut(int a) { return H[a] >= sz(g); }
};
```

**MinCostMaxFlow.h**  
**Description:** Min-cost max-flow. If costs can be negative, call `setpi` before `maxflow`, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.  
**Time:**  $\mathcal{O}(FE \log(V))$  where  $F$  is max flow.  $\mathcal{O}(VE)$  for `setpi`.

```
<bits/extc++.h> 0676ba, 77 lines

const ll INF = numeric_limits<ll>::max() / 4; //d4e

struct MCMF {
    struct edge {
        int from, to, rev;
        ll cap, cost, flow; //309
    };
    int N;
    vector<vector<edge>> ed;
    vi seen;
    vector<ll> dist, pi; //16a
    vector<edge*> par;

    MCMF(int _N) : N(_N), ed(N), seen(N), dist(N), pi(N), par(N) {}

    void addEdge(int from, int to, ll cap, ll cost) { //c71
        if (from == to) return;
        ed[from].push_back(edge{ from,to,sz(ed[to]),cap,cost,0 });
        ed[to].push_back(edge{ to,from,sz(ed[from])-1,0,-cost,0 });
    }
```

```
}
} //635
void path(int s) {
    fill(all(seen), 0);
    fill(all(dist), INF);
    dist[s] = 0; ll di;
} //17c
__gnu_pbds::priority_queue<pair<ll, int>> q;
vector<decltype(q)::point_iterator> its(N);
q.push({ 0, s });

while (!q.empty()) { //95a
    s = q.top().second; q.pop();
    seen[s] = 1; di = dist[s] + pi[s];
    for (edge& e : ed[s]) if (!seen[e.to]) {
        ll val = di - pi[e.to] + e.cost;
        if (e.cap - e.flow > 0 && val < dist[e.to]) { //c63
            dist[e.to] = val;
            par[e.to] = &e;
            if (its[e.to] == q.end())
                its[e.to] = q.push({ -dist[e.to], e.to });
            else //dbc
                q.modify(its[e.to], { -dist[e.to], e.to });
        }
    }
    rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF); //02d
}

pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) { //aa6
        ll fl = INF;
        for (edge* x = par[t]; x; x = par[x->from])
            fl = min(fl, x->cap - x->flow);

        totflow += fl; //21b
        for (edge* x = par[t]; x; x = par[x->from]) {
            x->flow += fl;
            ed[x->to][x->rev].flow -= fl;
        }
    } //cd4
    rep(i,0,N) for (edge& e : ed[i]) totcost += e.cost * e.flow;
    return {totflow, totcost/2};
}

// If some costs can be negative, call this before maxflow: //7c7
void setpi(int s) { // (otherwise, leave this out)
    fill(all(pi), INF); pi[s] = 0;
    int it = N, ch = 1; ll v;
    while (ch-- && it--)
        rep(i,0,N) if (pi[i] != INF) //42d
            for (edge& e : ed[i]) if (e.cap)
                if ((v = pi[i] + e.cost) < pi[e.to])
                    pi[e.to] = v, ch = 1;
    assert(it >= 0); // negative cost cycle
} //e03
};
```

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

```
482fe0, 36 lines

template<class T> T edmondsKarp(vector<unordered_map<int, T>>& //324
    graph, int source, int sink) {
    assert(source != sink);
    T flow = 0;
```

```
vi par(sz(graph)), q = par;
//cf9
for (;;) {
    fill(all(par), -1);
    par[source] = 0;
    int ptr = 1;
    q[0] = source;//623

    rep(i,0,ptr) {
        int x = q[i];
        for (auto e : graph[x]) {
            if (par[e.first] == -1 && e.second > 0) { //3a4
                par[e.first] = x;
                q[ptr++] = e.first;
                if (e.first == sink) goto out;
            }
        } //3cd
    }
    return flow;
}
out:
T inc = numeric_limits<T>::max();
for (int y = sink; y != source; y = par[y]) //d19
    inc = min(inc, graph[par[y]][y]);

flow += inc;
for (int y = sink; y != source; y = par[y]) {
    int p = par[y]; //b79
    if ((graph[p][y] -= inc) <= 0) graph[p].erase(y);
    graph[y][p] += inc;
}
} //cbb
```

**Dinic.h**  
**Description:** Flow algorithm with complexity  $O(VE \log U)$  where  $U = \max|cap|$ .  $O(\min(E^{1/2}, V^{2/3})E)$  if  $U = 1$ ;  $O(\sqrt{VE})$  for bipartite matching.

```
struct Dinic { //299
    struct Edge {
        int to, rev;
        ll c, oc;
        ll flow() { return max(oc - c, 0LL); } // if you need flows
    }; //8ec
    vi lvl, ptr, q;
    vector<vector<Edge>> adj;
    Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
    void addEdge(int a, int b, ll c, ll rcap = 0) {
        adj[a].push_back({b, sz(adj[b]), c, c}); //ed0
        adj[b].push_back({a, sz(adj[a]) - 1, rcap, rcap});
    }
    ll dfs(int v, int t, ll f) {
        if (v == t || !f) return f;
        for (int& i = ptr[v]; i < sz(adj[v]); i++) { //b2a
            Edge& e = adj[v][i];
            if (lvl[e.to] == lvl[v] + 1)
                if (ll p = dfs(e.to, t, min(f, e.c))) {
                    e.c -= p, adj[e.to][e.rev].c += p;
                    return p; //f3e
                }
        }
        return 0;
    }
    ll calc(int s, int t) { //b4c
        ll flow = 0; q[0] = s;
        rep(L,0,3l) do { // 'int L=30' maybe faster for random data
            lvl = ptr = vi(sz(q));
            int qi = 0, qe = lvl[s] = 1;
            while (qi < qe && !lvl[t]) { //796
```

```
int v = q[qi++];
for (Edge e : adj[v])
    if (!lvl[e.to] && e.c >> (30 - L))
        q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
} //4ca
while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
} while (lvl[t]);
return flow;
}
bool leftOfMinCut(int a) { return lvl[a] != 0; } //b90
};
```

**MinCut.h**  
**Description:** After running max-flow, the left side of a min-cut from  $s$  to  $t$  is given by all vertices reachable from  $s$ , only traversing edges with positive residual capacity.

```
d41d8c, 1 lines
//d41

GlobalMinCut.h
Description: Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.
Time:  $O(V^3)$ 
8b0e19, 21 lines
```

```
pair<int, vi> globalMinCut(vector<vi> mat) { //f64
    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) { //c8f
        vi w = mat[0];
        size_t s = 0, t = 0;
        rep(it,0,n-ph) { //  $O(V^2) \rightarrow O(E \log V)$  with prio.
            queue
            w[t] = INT_MIN;
            s = t, t = max_element(all(w)) - w.begin(); //0bb
            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]; //a2c
        rep(i,0,n) mat[i][s] = mat[s][i];
        mat[0][t] = INT_MIN;
    }
    return best;
} //cbb
```

**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:**  $O(V)$  Flow Computations

```
"PushRelabel.h"
085993, 13 lines

struct Edge { int from, to; ll cap; }; //81a
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
    vector<Edge> tree;
    vi par(N);
    rep(i,1,N) {
        PushRelabel D(N); // Dinic also works //146
        for (Edge t : ed) D.addEdge(t.from, t.to, t.cap, t.cap)
            ;
        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
            ;
    } //eec
    return tree;
}
```

**7.3 Matching**  
**hopcroftKarp.h**  
**Description:** Fast bipartite matching algorithm. Graph  $g$  should be a list of neighbors of the left partition, and  $btoa$  should be a vector full of -1's of the same size as the right partition. Returns the size of the matching.  $btoa[i]$  will be the match for vertex  $i$  on the right side, or -1 if it's not matched.  
**Usage:** vi btoa(m, -1); hopcroftKarp(g, btoa);  
**Time:**  $O(\sqrt{VE})$

```
f612e4, 42 lines

bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi& B) { //d9e
    if (A[a] != L) return 0;
    A[a] = -1;
    for (int b : g[a]) if (B[b] == L + 1) {
        B[b] = 0;
        if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B)) //613
            return btoa[b] = a, 1;
    }
    return 0;
}
//ad4
int hopcroftKarp(vector<vi>& g, vi& btoa) {
    int res = 0;
    vi A(g.size()), B(btoa.size()), cur, next;
    for (;;) {
        fill(all(A), 0); //db3
        fill(all(B), 0);
        cur.clear();
        for (int a : btoa) if (a != -1) A[a] = -1;
        rep(a,0,sz(g)) if (A[a] == 0) cur.push_back(a);
        for (int lay = 1; ; lay++) { //559
            bool islast = 0;
            next.clear();
            for (int a : cur) for (int b : g[a]) {
                if (btoa[b] == -1) {
                    B[b] = lay; //1ca
                    islast = 1;
                }
            }
            else if (btoa[b] != a && !B[b]) {
                B[b] = lay;
                next.push_back(btoa[b]); //1eb
            }
        }
        if (islast) break;
        if (next.empty()) return res;
        for (int a : next) A[a] = lay; //4f3
        cur.swap(next);
    }
    rep(a,0,sz(g))
        res += dfs(a, 0, g, btoa, A, B);
} //67c
```

**DFSMatching.h**  
**Description:** Simple bipartite matching algorithm. Graph  $g$  should be a list of neighbors of the left partition, and  $btoa$  should be a vector full of -1's of the same size as the right partition. Returns the size of the matching.  $btoa[i]$  will be the match for vertex  $i$  on the right side, or -1 if it's not matched.  
**Usage:** vi btoa(m, -1); dfsMatching(g, btoa);  
**Time:**  $O(VE)$

```
522b98, 22 lines

bool find(int j, vector<vi>& g, vi& btoa, vi& vis) { //400
    if (btoa[j] == -1) return 1;
    vis[j] = 1; int di = btoa[j];
    for (int e : g[di])
        if (!vis[e] && find(e, g, btoa, vis)) {
            btoa[e] = di; //a0e
            return 1;
        }
```

```
    }
    return 0;
}
int dfsMatching(vector<vi>& g, vi& btoa) { //52f
    vi vis;
    rep(i,0,sz(g)) {
        vis.assign(sz(btoa), 0);
        for (int j : g[i])
            if (find(j, g, btoa, vis)) { //e5b
                btoa[j] = i;
                break;
            }
    }
    return sz(btoa) - (int)count(all(btoa), -1); //ff5
}
```

### 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) { //60f
    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; //0db
    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) { //
            dc5
            seen[e] = true;
            q.push_back(match[e]);
        }
    }
    rep(i,0,n) if (!lfound[i]) cover.push_back(i); //849
    rep(i,0,m) if (seen[i]) cover.push_back(n+i);
    assert(sz(cover) == res);
    return cover;
}
```

### WeightedMatching.h

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

|                              |                  |
|------------------------------|------------------|
| "Time: $\mathcal{O}(N^2M)$ " | df0677, 31 lines |
|------------------------------|------------------|

```
pair<int, vi> hungarian(const vector<vi> &a) { //64f
    if (a.empty()) return {0, {}};
    int n = sz(a) + 1, m = sz(a[0]) + 1;
    vi u(n), v(m), p(m), ans(n - 1);
    rep(i,1,n) {
        p[0] = i; //0b5
        int j0 = 0; // add "dummy" worker 0
        vi dist(m, INT_MAX), pre(m, -1);
        vector<bool> done(m + 1);
        do { // dijkstra
            done[j0] = true; //bd1
            int i0 = p[j0], j1 = -1, delta = INT_MAX;
            rep(j,1,m) if (!done[j]) {
                auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
                if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
                if (dist[j] < delta) delta = dist[j], j1 = j; //865
            }
            rep(j,0,m) {
                if (done[j]) u[p[j]] += delta, v[j] -= delta;
```

```
                else dist[j] -= delta;
            } //aa1
            j0 = j1;
        } while (p[j0]);
        while (j0) { // update alternating path
            int j1 = pre[j0];
            p[j0] = p[j1], j0 = j1; //88f
        }
    }
    rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
    return {-v[0], ans}; // min cost
} //cbb
```

### GeneralMatching.h

**Description:** Matching for general graphs. Fails with probability  $N/\text{mod}$ .

|                             |                                    |                  |
|-----------------------------|------------------------------------|------------------|
| "Time: $\mathcal{O}(N^3)$ " | "../numerical/MatrixInverse-mod.h" | 1e40dd, 40 lines |
|-----------------------------|------------------------------------|------------------|

```
vector<pii> generalMatching(int N, vector<pii>& ed) { //19e
    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;
    } //063

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

    if (M != N) do { //f88
        mat.resize(M, vector<ll>(M));
        rep(i,0,N) {
            mat[i].resize(M);
            rep(j,N,M) {
                int rr = rand() % mod; //b47
                mat[i][j] = rr, mat[j][i] = (mod - rr) % mod;
            }
        }
        while (matInv(A = mat) != M);
    } //92b
    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; //e0a
            }
        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); //b7f
            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); //3c7
        }
    }
    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: $\mathcal{O}(E + V)$ " | 76b5c9, 24 lines |
|-------------------------------|------------------|

```
vi val, comp, z, cont; //ed2
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)); //b9e

    if (low == val[j]) {
        do {
            x = z.back(); z.pop_back();
            comp[x] = ncomps; //f1f
            cont.push_back(x);
        } while (x != j);
        f(cont); cont.clear();
        ncomps++;
    } //658
    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); //5bc
    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: $\mathcal{O}(E + V)$ " | 037821, 28 lines |
|-------------------------------|------------------|

```
template<class F> //c2c
void bicomps(vector<vector<pii>> &ed, F f) {
    vi num(sz(ed)), st;
    int t=0;
    auto dfs = [&](auto &&self, int at, int par) -> int {
        int me = num[at] = ++t, top = me; //b12
        for (auto [y, e] : ed[at]) if (e != par) {
            if (num[y]) {
                top = min(top, num[y]);
                if (num[y] < me)
                    st.push_back(e); //630
            } else {
                int si = sz(st);
                int up = self(self, y, e);
                top = min(top, up);
                if (up == me) { //c92
                    st.push_back(e);
                    f(vi(st.begin() + si, st.end()));
                    st.resize(si);
                }
                else if (up < me) st.push_back(e); //1a1
                else { /* e is a bridge */ }
            }
        }
        return top;
    }
}
```

```
    };//835
    rep(i,0,sz(ed)) if (!num[i]) dfs(dfs, i, -1);
}
```

### Articulation.h

**Description:** Finds articulation points (removal separates graph)  
**Time:**  $\mathcal{O}(n+m)$

a7b0ba, 25 lines

```
vector<bool> cutpoints(const vector<vi> &adj) { //259
    int timer=0, n=sz(adj);
    vi tin(n,-1), low(n,-1);
    vector<bool> vis(n);
    vector<bool> iscut(n);
    auto dfs = [&](auto &&self, int v, int p) -> void { //7a1
        vis[v] = true;
        tin[v] = low[v] = timer++;
        int ch = 0;
        for (int to : adj[v]) {
            if (to == p) continue; //b9d
            if (vis[to])
                low[v] = min(low[v], tin[to]);
            else {
                self(self, to, v);
                low[v] = min(low[v], low[to]); //914
                if (low[to] >= tin[v] && p!=-1) iscut[v]=1;
                ++ch;
            }
        }
        if(p == -1 && ch > 1) iscut[v]=1; //4e2
    };
    rep(i,0,n) if (!vis[i]) dfs(dfs, i, -1);
    return iscut;
}
```

### 2sat.h

**Description:** Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type  $(a||b)\&\&(!a||c)\&\&(d||!b)\&\&...$  becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions (~x).  
**Usage:** TwoSat ts(number of boolean variables);  
ts.either(0, ~3); // Var 0 is true or var 3 is false  
ts.setValue(2); // Var 2 is true  
ts.atMostOne({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true  
ts.solve(); // Returns true iff it is solvable  
ts.values[0..N-1] holds the assigned values to the vars  
**Time:**  $\mathcal{O}(N+E)$ , where N is the number of boolean variables, and E is the number of clauses.

5f9706, 56 lines

```
struct TwoSat { //7c0
    int N;
    vector<vi> gr;
    vi values; // 0 = false, 1 = true

    TwoSat(int n = 0) : N(n), gr(2*n) {} //54e

    int addVar() { // (optional)
        gr.emplace_back();
        gr.emplace_back();
        return N++; //662
    }

    void either(int f, int j) {
        f = max(2*f, -1-2*f);
        j = max(2*j, -1-2*j); //3b0
        gr[f].push_back(j^1);
        gr[j].push_back(f^1);
    }

    void setValue(int x) { either(x, x); }
} //41c
void atMostOne(const vi& li){ // (optional)
```

```
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i,2,sz(li)) {
        int next = addVar(); //f5e
        either(cur, ~li[i]);
        either(cur, next);
        either(~li[i], next);
        cur = ~next;
    } //276
    either(cur, ~li[1]);
}

vi val, comp, z; int time = 0;
int dfs(int i) { //7e3
    int low = val[i] = ++time, x; z.push_back(i);
    for(int e : gr[i]) if (!comp[e])
        low = min(low, val[e] ?: dfs(e));
    if (low == val[i]) do {
        x = z.back(); z.pop_back(); //0c0
        comp[x] = low;
        if (values[x>>1] == -1)
            values[x>>1] = x&1;
    } while (x != i);
    return val[i] = low; //749
}

bool solve() {
    values.assign(N, -1);
    val.assign(2*N, 0); comp = val; //4fa
    rep(i,0,2*N) if (!comp[i]) dfs(i);
    rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
    return 1;
}
} //214
```

### 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) { //fda
    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]);
        //e35
        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); //8f2
        }
    }
    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.)

ca07a0, 31 lines

```
Time:  $\mathcal{O}(NM)$ 

vi edgeColoring(int N, vector<pii> eds) { //d26
    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) { //945
        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) //665
            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) { //f7e
            int left = fan[i], right = fan[++i], x = cc[i];
            adj[u][x] = left;
            adj[left][x] = u;
            adj[right][x] = -1;
            free[right] = x; //e59
        }
        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++) //b06
                ;
    }
    rep(i,0,sz(eds))
        for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i
            ];
    return ret;
} //cbb
```

## 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.

b0d5b1, 12 lines

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

typedef bitset<128> B; //abb
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]; //7d8
    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;
    } //67c
}
```

### 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.

f7c0bc, 49 lines

```
typedef vector<bitset<200>> vb; //b92
struct Maxclique {
    double limit=0.025, pk=0;
    struct Vertex { int i, d=0; };
    typedef vector<Vertex> vv;
```

```
vb e;//5b2
vv V;
vector<vi> C;
vi qmax, q, S, old;
void init(vv& r) {
    for (auto& v : r) v.d = 0;//dab
    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;
} //a6a
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;//6b0
        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);//feb
            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]; };//547
                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);
            } //08b
            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;//15f
        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)
) {} //02b
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 MiniumVertexCover.

d41d8c, 1 lines

//d41

## 7.7 Trees

### 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;//386
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);//a92
    int m = sz(li)-1;
```

```
    rep(i,0,m) {
        int a = li[i], b = li[i+1];
        li.push_back(lca.lca(a, b));
    } //c76
    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) { //ff8
        int a = li[i], b = li[i+1];
        ret.emplace_back(rev[lca.lca(a, b)], b);
    }
    return ret;
} //cbb
```

### 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. Takes as input the full adjacency list. being true means that values are stored in the edges, as opposed to the nodes.

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

cfe575, 55 lines

```
template<bool op_edges = false> //001
struct hld {
    vector<vi> adj;
    vi par, size, in, head, d
    hld(int n) : adj(n), par(n), size(n), in(n), head(n), d
        (n) {}
    //8ac
    void add_edge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    //9ee
    void dfs_size(int v = 0, int p = 0) {
        size[v] = 1;
        for (int &e : adj[v]) {
            if (e != p) {
                d[e] = d[v] + 1; //a74
                par[e] = v;
                dfs_size(e, v);
                size[v] += size[e];
                if (size[e] > size[adj[v][0]] || adj[v][0]
                    == p) swap(e, adj[v][0]);
            } //96d
        }
    }
    void dfs_hld(int v = 0, int p = 0) {
        static int t = 0; //978
        in[v] = t++;
        for (int e : adj[v]) {
            if (e != p) {
                if (e == adj[v][0]) {
                    head[e] = head[v]; //c3e
                } else {
                    head[e] = e;
                }
                dfs_hld(e, v);
            } //47a
        }
    }
}
```

```
template<typename F>
void op_path(int x, int y, F op) { //b81
    while (head[x] != head[y]) {
        while (d[head[x]] > d[head[y]]) swap(x, y);
        op(in[head[y]], in[y] + 1);
        y = par[head[y]];
    } //387
    if (d[x] > d[y]) swap(x, y);
```

```
        op(in[x] + (op_edges ? 1 : 0), in[y] + 1);
    }

    template<typename F> //d25
    void op_subtree(int x, F op) {
        op(in[x] + (op_edges ? 1 : 0), in[x] + size[x]);
    }
};
```

### 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  $\mathcal{O}(\log N)$ .

0fb462, 90 lines

```
struct Node { // Splay tree. Root's pp contains tree's
    parent. //a4e
    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; //b8f
        if (c[1]) c[1]->p = this;
        // (+ update sum of subtree elements etc. if wanted)
    }
    void pushFlip() {
        if (!flip) return; //dfd
        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; } //3a9
    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]; //eb7
        if (b < 2) {
            x->c[h] = y->c[h ^ 1];
            y->c[h ^ 1] = x;
        }
        z->c[i ^ 1] = this; //430
        fix(); x->fix(); y->fix();
        if (p) p->fix();
        swap(pp, y->pp);
    }
    void splay() { //4c8
        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); //9e8
            else p->p->rot(c2, c1 != c2);
        }
    }
    Node* first() {
        pushFlip(); //828
        return c[0] ? c[0]->first() : (splay(), this);
    }
};

struct LinkCut { //d99
    vector<Node> node;
    LinkCut(int N) : node(N) {}

    void link(int u, int v) { // add an edge (u, v)
        assert(!connected(u, v)); //166
        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]; //0b9
    makeRoot(top); x->splay();
    assert(top == (x->pp ?: x->c[0]));
    if (x->pp) x->pp = 0;
    else {
        x->c[0] = top->p = 0; //158
        x->fix();
    }
}

bool connected(int u, int v) { // are u, v in the same tree?
    Node* nu = access(&node[u])->first(); //781
    return nu == access(&node[v])->first();
}

void makeRoot(Node* u) {
    access(u);
    u->splay(); //09d
    if(u->c[0]) {
        u->c[0]->p = 0;
        u->c[0]->flip ^= 1;
        u->c[0]->pp = u;
        u->c[0] = 0; //41e
        u->fix();
    }
}

Node* access(Node* u) {
    u->splay(); //4e7
    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; //f4d
        }
    }
    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" 057d96, 60 lines

struct Edge { int a, b; ll w{}; }; //4d9
struct Node {
    Edge key;
    Node *l=0, *r=0;
    ll delta{};
    void prop() { //936
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    } //5dc
    Edge top() { prop(); return key; }
};

Node *merge(Node *a, Node *b) {
    if (!a || !b) return a ?: b;
    a->prop(), b->prop(); //72a
    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); } //8e9

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});
}; //0f3
```

```
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; //4c6
rep(s, 0, n) {
    int u = s, qi = 0, w;
    while (seen[u] < 0) {
        if (!heap[u]) return {-1, {}};
        Edge e = heap[u]->top(); //2b0
        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; //fff
            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]}}); //984
        }
    }
    rep(i, 0, qi) in[uf.find(Q[i].b)] = Q[i];
}
//eba
for (auto& [u, t, cc] : cycs) { // restore sol (optional)
    uf.rollback(t);
    Edge inEdge = in[u];
    for (auto& e : cc) in[uf.find(e.b)] = e;
    in[uf.find(inEdge.b)] = inEdge; //ffd
}
rep(i, 0, n) par[i] = in[i].a;
return {res, par};
}
```

7.8 Math

7.8.1 Number of Spanning Trees

Create an  $N \times N$  matrix  $\text{mat}$ , 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.)
.../634da7, 29 lines

template <class T> int sgn(T x) { return (x > 0) - (x < 0); } //fa7
template<class T>
struct Point {
    typedef Point P;
```

```
T x, y;
explicit Point(T _x=0, T _y=0) : x(_x), y(_y) {} //a5f
bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
P operator+(P p) const { return P(x+p.x, y+p.y); }
P operator-(P p) const { return P(x-p.x, y-p.y); }
P operator*(T d) const { return P(x*d, y*d); } //e11
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; } // + => p on right
T cross(P a, P b) const { return (a-*this).cross(b-*this); }
bool half() const { return y < 0 || (y == 0 && x < 0); } //053
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//8da
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)); } //ad4
friend ostream& operator<<(ostream& os, P p) {
    return os << "(" << p.x << ", " << p.y << ")"; }
};
```

AngleSort.h

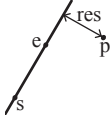
**Description:** Sorts points radially across the origin. To sort around a point, sort a-p and b-p.

```
"Point.h" c10d46, 7 lines

template<class P> //159
void anglesort(vector<P> &v, P p=P(0, 0)) {
    sort(all(v), [p](P a, P b) {
        a = a - p, b = b - p;
        return a.half() == b.half() ? a.cross(b) > 0 : a.half() < b.half();
    }); //b97
}
```

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.

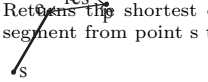


```
"Point.h" f6bf6b, 4 lines

template<class P> //f6b
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;

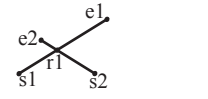
"Point.h"

5c88f4, 6 lines

```
typedef Point<double> P; //b95
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;
} //cbb
```

### 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;

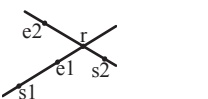
"Point.h", "OnSegment.h"

9d57f2, 13 lines

```
template<class P> vector<P> segInter(P a, P b, P c, P d) {
    //dec
    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)}; //8a0
    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); //814
    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> //47e
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); //16d
    return {1, (s1 * p + e1 * q) / d};
}
```

### sideOf.h

**Description:** Returns where p is as seen from s towards e. 1/0/-1 ⇔ left/on line/right. If the optional argument eps is given 0 is returned if p is within distance eps from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.  
**Usage:** bool left = sideOf(p1,p2,q)==1;

"Point.h"

3af81c, 9 lines

```
template<class P> //059
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); //7c7
    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>.

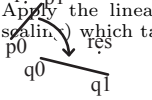
"Point.h"

c597e8, 3 lines

```
template<class P> bool onSegment(P s, P e, P p) { //c59
    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; //d52
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());
} //cbb
```

### 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"

28b5e9, 35 lines

```
struct Angle { //717
    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 { //a5b
        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()}; } //de0
    Angle t360() const { return {x, y, t + 1}; }
};

bool operator<(Angle a, Angle b) {
    // add a.dist2() and b.dist2() to also compare distances
    return make_tuple(a.t, a.half(), a.y * (ll)b.x) < //41b
        make_tuple(b.t, b.half(), a.x * (ll)b.y);
}
```

// Given two points, this calculates the smallest angle between them, i.e., the angle that covers the defined line segment. //f86

```
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()));
} //b11
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;
} //073
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
    int tu = b.t - a.t; a.t = b.t;
    return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a
        )};
}
```

## 8.2 Circles

### CircleIntersection.h

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

"Point.h"

84d6d3, 11 lines

```
typedef Point<double> P; //deb
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; //367
    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;
} //cbb
```

### 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>//c18
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;//446
    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();//918
    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>//64a
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};//fd3
    P h = ab.unit() * sqrt(h2);
    return {p - h, p + h};
}
```

### CirclePolygonIntersection.h

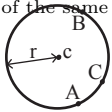
**Description:** Returns the area of the intersection of a circle with a ccw polygon.

```
"Time: O(n)
"../content/geometry/Point.h" alee63, 19 lines
typedef Point<double> P;//a6c
#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;//eda
        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;//174
        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))//a61
        sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
    return sum;
}
```

### circumcircle.h

#### Description:

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



```
"Point.h" 1caa3a, 9 lines
typedef Point<double> P;//032
double ccRadius(const P& A, const P& B, const P& C) {
    return (B-A).dist()*(C-B).dist()*(A-C).dist()/
        abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {//793
    P b = C-A, c = B-A;
    return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

### MinimumEnclosingCircle.h

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

```
"Time: expected O(n)
"circumcircle.h" 09dd0a, 17 lines
pair<P, double> mec(vector<P> ps) {//b50
    shuffle(all(ps), mt19937(time(0)));
    P o = ps[0];
    double r = 0, EPS = 1 + 1e-8;
    rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
        o = ps[i], r = 0;//d54
        rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
            o = (ps[i] + ps[j]) / 2;
            r = (o - ps[i]).dist();
            rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
                o = ccCenter(ps[i], ps[j], ps[k]);//4ec
                r = (o - ps[i]).dist();
            }
        }
    }
    return {o, r};//2ac
}
```

## 8.3 Polygons

### InsidePolygon.h

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

**Usage:** vector<P> v = {P{4,4}, P{1,2}, P{2,1}}; bool in = inPolygon(v, P{3, 3}, false);

```
"Time: O(n)
"Point.h", "OnSegment.h", "SegmentDistance.h" 2bf504, 11 lines
template<class P>//1c1
bool inPolygon(vector<P> &p, P a, bool strict = true) {
    int cnt = 0, n = sz(p);
    rep(i,0,n) {
        P q = p[(i + 1) % n];
        if (onSegment(p[i], q, a) return !strict;//fa7
        //or: if (segDist(p[i], q, a) <= eps) return !strict;
        cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
    }
    return cnt;
} //cbb
```

### PolygonArea.h

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

```
"Point.h" f12300, 6 lines
template<class T>//b19
T polygonArea2(vector<Point<T>>& v) {
    T a = v.back().cross(v[0]);
    rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
    return a;
} //cbb
```

### PolygonCenter.h

**Description:** Returns the center of mass for a polygon. **Time:**  $O(n)$

```
"Point.h" 9706dc, 9 lines
typedef Point<double> P;//082
P polygonCenter(const vector<P>& v) {
    P res(0, 0); double A = 0;
    for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
        res = res + (v[i] + v[j]) * v[j].cross(v[i]);
        A += v[j].cross(v[i]);//168
    }
    return res / A / 3;
}
```

### PolygonCut.h

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



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

```
"Point.h", "LineIntersection.h" f2b7d4, 13 lines
typedef Point<double> P;//366
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
    vector<P> res;
    rep(i,0,sz(poly)) {
        P cur = poly[i], prev = i ? poly[i-1] : poly.back();
        bool side = s.cross(e, cur) < 0;//44d
        if (side != (s.cross(e, prev) < 0))
            res.push_back(lineInter(s, e, cur, prev).second);
        if (side)
            res.push_back(cur);
    } //0e1
    return res;
}
```

### PolygonUnion.h

**Description:** Calculates the area of the union of n polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be needed.)

```
"Time: O(N^2), where N is the total number of points
"Point.h", "sideOf.h" 3931c6, 33 lines
typedef Point<double> P;//49c
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y;
}
double polyUnion(vector<vector<P>>& poly) {
    double ret = 0;
    rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
        P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];//896
        vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
        rep(j,0,sz(poly)) if (i != j) {
            rep(u,0,sz(poly[j])) {
                P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])
                    ];
            }
        }
    }
}
```

```
int sc = sideOf(A, B, C), sd = sideOf(A, B, D);//407
if (sc != sd) {
    double sa = C.cross(D, A), sb = C.cross(D, B);
    if (min(sc, sd) < 0)
        segs.emplace_back(sa / (sa - sb), sgn(sc - sd))
        ;
} else if (!sc && !sd && j<i && sgn((B-A).dot(D-C)) > 0)//8be
    segs.emplace_back(rat(C - A, B - A), 1);
    segs.emplace_back(rat(D - A, B - A), -1);
}
}//155
sort(all(segs));
for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
double sum = 0;
int cnt = segs[0].second;
rep(j,1,sz(segs)) //88e
    if (!cnt) sum += segs[j].first - segs[j - 1].first;
    cnt += segs[j].second;
}
ret += A.cross(B) * sum;
}//f48
return ret / 2;
}
```

### ConvexHull.h

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



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

```
"Point.h" 310954, 13 lines
typedef Point<ll> P;//3e3
vector<P> convexHull(vector<P> pts) {
    if (sz(pts) <= 1) return pts;
    sort(all(pts));
    vector<P> h(sz(pts)+1);
    int s = 0, t = 0;//f18
    for (int it = 2; it--; s = --t, reverse(all(pts)))
        for (P p : pts) {
            while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t
            --;
            h[t++] = p;
        }//aa0
    return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```

### HullDiameter.h

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

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

```
"Point.h" c571b8, 12 lines
typedef Point<ll> P;//5c7
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) //56c
            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;//52a
}
```

### 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
typedef Point<ll> P;//7a3

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);//4a6
    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;//0da
    }
    return sgn(l[a].cross(l[b], p)) < r;
}
```

### LineHullIntersection.h

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

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

```
"Point.h" 7cf45b, 39 lines
#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))//b9d
#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) //51a
        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;
    }//e8c
    return lo;
}

#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>//7fd
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};//04b
    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;//ec0
            (cmpL(m) == cmpL(endB) ? lo : hi) = m;
        }
        res[i] = (lo + !cmpL(hi)) % n;
        swap(endA, endB);
    }//6ab
    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]};//08a
    }
return res;
}
```

### HullTangents.h

**Description:** Finds the two tangent vertices on the convex hull to some point. Point must be outside. Appears to be left then right.

```
"Point.h" 0bdfcf, 22 lines
template<typename P, typename F>//134
int extremeVertex(const P& poly, F direction) {
    int n = sz(poly), l = 0, ls;
    auto vertexCmp = [&](int i, int j) {
        return sgn(direction(poly[j]).cross(poly[j] - poly[i]))
        ; };
    auto isExtreme = [&](int i, int& is) //d3d
        return (is = vertexCmp((i+1)%n, i)) >= 0 && vertexCmp(i, (i+n-1)%n) < 0; };
    for (int r = isExtreme(0, ls) ? 1 : n; l + 1 < r;) {
        int m = (l + r) / 2, ms;
        if (isExtreme(m, ms)) return m;
        if (ls != ms ? ls < ms : ls == vertexCmp(l, m)) r = m;
        //beb
        else l = m, ls = ms;
    }
    return l;
}//d22
template<typename P>
pair<int, int> tangentsConvex(const P &point, const vector<P>& poly) {
    return {
        extremeVertex(poly, [&](const P& q) { return q - point;
        }),
        extremeVertex(poly, [&](const P& q) { return point - q;
        })};//fa7
}
```

### MinkowskiSum.h

**Description:** Returns the set of all sums of points of two convex polygons.

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

```
"Point.h" 01bc35, 29 lines
typedef Point<ll> P;//9c1
void reorder_polygon(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;//bf2
    }
    rotate(p.begin(), p.begin() + pos, p.end());
}

vector<P> minkowski(vector<P> p, vector<P> q) //be7
    reorder_polygon(p);
    reorder_polygon(q);

    p.push_back(p[0]);
    p.push_back(p[1]);//f50
    q.push_back(q[0]);
    q.push_back(q[1]);

    vector<P> result;
    int i = 0, j = 0;//98e
    while (i < sz(p) - 2 || j < sz(q) - 2) {
        result.push_back(p[i] + q[j]);
    }
```

```
        auto cross = (p[i + 1] - p[i]).cross(q[j + 1] - q[j]);
        if (cross >= 0 && i < sz(p) - 2) ++i;
        if (cross <= 0 && j < sz(q) - 2) ++j;//19e
    }
    return result;
}
```

### 8.4 Misc. Point Set Problems

**ClosestPair.h**  
**Description:** Finds the closest pair of points.  
**Time:**  $O(n \log n)$

|           |                  |
|-----------|------------------|
| "Point.h" | ac41a6, 17 lines |
|-----------|------------------|

```
typedef Point<ll> P;//9e7
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()}};//e83
    int j = 0;
    for (P p : v) {
        P d{1 + (ll)sqrt(ret.first), 0};
        while (v[j].y <= p.y - d.x) S.erase(v[j++]);
        auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);//cb2
        for (; lo != hi; ++lo)
            ret = min(ret, {( *lo - p).dist2(), { *lo, p } });
        S.insert(p);
    }
    return ret.second;//982
}
```

**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:**  $O(N \log N)$

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

```
typedef Point<int> P;//bde
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) { //9bd
        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); //0bb
                 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}); //5b9
            }
            sweep[-ps[i].y] = i;
        }
        for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x, p.y);
    } //aa4
    return edges;
}
```

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

|           |                  |
|-----------|------------------|
| "Point.h" | bac5b0, 63 lines |
|-----------|------------------|

```
typedef long long T;//632
```

```
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; } //c56

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; //5b4

    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(); //a82
    }

    Node(vector<P>&& vp) : pt(vp[0]) {
        for (P p : vp) {
            x0 = min(x0, p.x); x1 = max(x1, p.x); //151
            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); //1d2
            // 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()}); //ace
        }
    }
};

struct KDTree { //72b
    Node* root;
    KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}

    pair<T, P> search(Node *node, const P& p) {
        if (!node->first) { //119
            // 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);
        }
        //a89
        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 //bfa
        auto best = search(f, p);
        if (bsec < best.first)
            best = min(best, search(s, p));
        return best;
    } //13a

    // 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); //213
    }
};
```

**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:**  $O(n \log n)$

|           |                 |
|-----------|-----------------|
| "Point.h" | cefd5, 88 lines |
|-----------|-----------------|

```
typedef Point<ll> P;//503
typedef struct Quad* Q;
typedef __int128_t ll1; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX, LLONG_MAX); // not equal to any other point

struct Quad { //8bb
    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(); } //0bd
} *H;

bool circ(P p, P a, P b, P c) { // is p in the circumcircle ?
    ll1 p2 = p.dist2(), A = a.dist2()-p2,
        B = b.dist2()-p2, C = c.dist2()-p2; //520
    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; //60f
    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) { //5b1
    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()); //3cc
    splice(q->r(), b);
    return q;
}

pair<Q,Q> rec(const vector<P>& s) { //a03
    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]); //d54
        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//f35
#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)}); //c17
    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; //a99

#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()); \///475
    splice(e->r(), e->r()->prev()); \
    e->o = H; H = e; e = t; \
}
for (;;) {
    DEL(LC, base->r(), o); DEL(RC, base, prev());///031
    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());///907
}
return { ra, rb };
}

vector<P> triangulate(vector<P> pts) {///e5d
    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;///02b
    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;///24a
    return pts;
}
```

## 8.5 3D

### PolyhedronVolume.h

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

30583c, 6 lines

```
template<class V, class L>///27c
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;
}///cbb
```

### Point3D.h

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

6eb43e, 32 lines

```
template<class T> struct Point3D {///811
    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 {///5e8
        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); }
    ///9b1
    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);
        ///58a
    }
    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); }///a2c
    ///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(); }///e88
    ///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;
    }///e03
};
```

### 3dHull.h

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

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

"Point3D.h" 5b45fc, 49 lines

typedef Point3D<double> P3;///*e28*

```
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); }///c34
    int a, b;
};
```

```
struct F { P3 q; int a, b, c; };
///36b
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;///de0
    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};///923
        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);///e21
```

```
    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])) {///b63
                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();///0df
            }
            int nw = sz(FS);
            rep(j,0,nw) {
                F f = FS[j];///945
#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(///ab3
```

```

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

### sphericalDistance.h

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

611f07, 8 lines

```
double sphericalDistance(double f1, double t1,///6da
    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);///65e
    return radius*2*asin(d/2);
}
```

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

d4375c, 16 lines

```
vi pi(const string& s) {///f6d
    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]);///0ff
    }
    return p;
}
```

```
vi match(const string& s, const string& pat) {///752
    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;
}///cbb
```

### Zfunc.h

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

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

ee09e2, 12 lines

```
vi Z(const string& S) {///fc3
    vi z(sz(S));
    int l = -1, r = -1;
    rep(i,1,sz(S)) {
        z[i] = i >= r ? 0 : min(r - i, z[i - l]);
        while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])///8ec
            z[i]++;
        if (i + z[i] > r)
            l = i, r = i + z[i];
    }
    return z;///939
}
```

### 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).



|  |                  |
|--|------------------|
| <b>Time:</b> $\mathcal{O}(N)$  | e7ad79, 13 lines |
| <pre>array&lt;vi, 2&gt; manacher(const string&amp; s) { //510     int n = sz(s);     array&lt;vi, 2&gt; p = {vi(n+1), vi(n)};     rep(z, 0, 2) for (int i=0, l=0, r=0; i &lt; n; i++) {         int t = r-i+!z;         if (i&lt;r) p[z][i] = min(t, p[z][l+t]); //f50         int L = i-p[z][i], R = i+p[z][i]-!z;         while (L&gt;=1 &amp;&amp; R+1&lt;n &amp;&amp; s[L-1] == s[R+1])             p[z][i]++, L--, R++;         if (R&gt;r) l=L, r=R;     } //291     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:**  $\mathcal{O}(N)$

|   |                 |
|---|-----------------|
| <pre>int minRotation(string s) { //20f     int a=0, N=sz(s); s += s;     rep(b, 0, N) rep(k, 0, N) {         if (a+k == b    s[a+k] &lt; s[b+k]) { b += max(0, k-1);             break; }         if (s[a+k] &gt; s[b+k]) { a = b; break; }     } //3a8     return a; }</pre> | d07a42, 8 lines |
|---|-----------------|

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

|  |                  |
|--|------------------|
| <pre>struct SuffixArray { //58c     vi sa, lcp;     SuffixArray(string&amp; s, int lim=256) { // or basic-string&lt;int&gt;         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); //032         for (int j = 0, p = 0; p &lt; n; j = max(1, j * 2), lim = p) {             p = j, iota(all(y), n - j);             rep(i, 0, n) if (sa[i] &gt;= j) y[p++] = sa[i] - j;             fill(all(ws), 0);             rep(i, 0, n) ws[x[i]]++; //f08             rep(i, 1, lim) ws[i] += ws[i - 1];             for (int i = n; i--;) sa[--ws[x[y[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] &amp;&amp; y[a + j] == y[b + j]) ? p - 1 : p                 ++; //f9f         }         rep(i, 1, n) rank[sa[i]] = i;         for (int i = 0, j; i &lt; n - 1; lcp[rank[i++]] = k)             for (k &amp;&amp; k--, j = sa[rank[i] - 1];                 s[i + k] == s[j + k]; k++); //31d     } };</pre> | 38db9f, 23 lines |
|--|------------------|

### 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:**  $\mathcal{O}(26N)$

|  |                  |
|--|------------------|
| <pre>struct SuffixTree { //b1f     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;     //b11     void ukkadd(int i, int c) { suff:         if (r[v]&lt;=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]; //99f         }         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; //604             v=s[p[m]]; q=l[m];             while (q&lt;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;         } //478     }      SuffixTree(string a) : a(a) {         fill(r, r+N, sz(a));         memset(s, 0, sizeof s); //f11         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]));     } //d1a      // example: find longest common substring (uses ALPHA = 28)     pii best;     int lcs(int node, int i1, int i2, int olen) {         if (l[node] &lt;= i1 &amp;&amp; i1 &lt; r[node]) return 1; //636         if (l[node] &lt;= i2 &amp;&amp; i2 &lt; 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) //a3a             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)             ); //78c         st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);         return st.best;     } };</pre> | aae0b8, 50 lines |
|--|------------------|

### Hashing.h

**Description:** Self-explanatory methods for string hashing.  
 $\text{Arithmetic mod } 2^{64}-1$ .  $2x$  slower than mod  $2^{64}$  and more  
code, but works on evil test data (e.g. Thue–Morse,  
where  
ABBA... and BAAB... of length  $2^{10}$  hash the same mod  $2^{64}$ ).

|   |  |
|---|--|
| <pre>// "typedef ull H;" instead if you think test data is random, // or work mod 10^9+7 if the Birthday paradox is not a problem. typedef uint64_t ull; //98c struct H {     ull x; H(ull x=0) : x(x) {}     H operator+(H o) { return x + o.x + (x + o.x &lt; x); }     H operator-(H o) { return *this + ~o.x; }     H operator*(H o) { auto m = (__uint128_t)x * o.x; //884         return H((ull)m) + (ull)(m &gt;&gt; 64); }     ull get() const { return x + !~x; }     bool operator==(H o) const { return get() == o.get(); }     bool operator&lt;(H o) const { return get() &lt; o.get(); } }; //7dd static const H C = (1ll)1e11+3; // (order ~ 3e9; random also ok)  struct HashInterval {     vector&lt;H&gt; ha, pw;     HashInterval(string&amp; str) : ha(sz(str)+1), pw(ha) { //c1e         pw[0] = 1;         rep(i, 0, sz(str))             ha[i+1] = ha[i] * C + str[i],             pw[i+1] = pw[i] * C;     } //b8f     H hashInterval(int a, int b) { // hash [a, b)         return ha[b] - ha[a] * pw[b - a];     } }; //4b7 vector&lt;H&gt; getHashes(string&amp; str, int length) {     if (sz(str) &lt; length) return {};     H h = 0, pw = 1;     rep(i, 0, length)         h = h * C + str[i], pw = pw * C; //7ab     vector&lt;H&gt; ret = {h};     rep(i, length, sz(str)) {         ret.push_back(h = h * C + str[i] - pw * str[i-length]);     }     return ret; //413 }</pre> |  |
|---|--|

H hashString(string& s){H h{}; for(char c:s) h=h\*C+c;return h;}

### 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  $\mathcal{O}(NM)$ .

|  |                  |
|--|------------------|
| <pre>struct AhoCorasick { //724     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)); } //cc2     };     vector&lt;Node&gt; N;     vi backp;     void insert(string&amp; s, int j) {         assert(!s.empty()); //757</pre> | 135677, 66 lines |
|--|------------------|

```
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; //20b
}
if (N[n].end == -1) N[n].start = j;
backp.push_back(N[n].end);
N[n].end = j;
N[n].nmatches++; //77c
}
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); //12a

    queue<int> q;
    for (q.push(0); !q.empty(); q.pop()) {
        int n = q.front(), prev = N[n].back;
        rep(i, 0, alpha) { //57b
            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
                ]) //338
                = N[y].end;
                N[ed].nmatches += N[y].nmatches;
                q.push(ed);
            }
        } //c05
    }
}
vi find(string word) {
    int n = 0;
    vi res; // 11 count = 0; //a68
    for (char c : word) {
        n = N[n].next[c - first];
        res.push_back(N[n].end);
        // count += N[n].nmatches;
    } //bb1
    return res;
}
vector<vi> findAll(vector<string>& pat, string word) {
    vi r = find(word);
    vector<vi> res(sz(word)); //008
    rep(i, 0, sz(word)) {
        int ind = r[i];
        while (ind != -1) {
            res[i - sz(pat[ind]) + 1].push_back(ind);
            ind = backp[ind]; //8f0
        }
    }
    return res;
}
} //214
```

## 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)
{ //ba1
    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); //ea6
}
if (it != is.begin() && (--it)->second >= L) {
    L = min(L, it->first);
    R = max(R, it->second);
    is.erase(it); //05d
}
return is.insert(before, {L, R});
}
```

```
void removeInterval(set<pii>& is, int L, int R) { //858
    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; //61f
    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> //0e2
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; //ed8
    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])); //607
            at++;
        }
        if (mx.second == -1) return {};
        cur = mx.first;
        R.push_back(mx.second); //26b
    }
    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> //570
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; //05f
    } else {
        int mid = (from + to) >> 1;
        rec(from, mid, f, g, i, p, f(mid));
        rec(mid+1, to, f, g, i, p, q);
    } //729
}
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); //a6c
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> //7d4
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) //ec4
        else b = mid+1;
    }
    rep(i, a+1, b+1) if (f(a) < f(i)) a = i; // (B)
    return a;
} //cbb
```

#### 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) { //47f
    if (S.empty()) return {};
    vi prev(sz(S));
    typedef pair<I, int> p;
    vector<p> res;
    rep(i, 0, sz(S)) { //a50
        // 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; //476
    }
    int L = sz(res), cur = res.back().second;
    vi ans(L);
    while (L--) ans[L] = cur, cur = prev[cur];
    return ans; //342
}
```

#### 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) { //e2b
    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); //14a
    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]) //45b
            v[x-w[j]] = max(v[x-w[j]], j);
    }
    for (a = t; v[a+m-t] < 0; a--);
```



```
    return a;
} //cbb
```

### 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:**  $\mathcal{O}(N^2)$

441d8c, 1 lines

//d41

**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:**  $\mathcal{O}((N + (hi - lo)) \log N)$

d38d2b, 18 lines

```
struct DP { // Modify at will: //ff9
    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); }
} //ec8
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))) //680
        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);
} //a30
void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

### 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

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

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

#### 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  $[0, 2b)$ .

751a02, 8 lines

```
typedef unsigned long long ull; //010
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; //430
    }
};
```

**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() //c51
    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); //818
    }
    return buf[bc++]; // returns 0 on EOF
}

int readInt() { //f26
    int a, c;
    while ((a = gc()) < 40);
    if (a == '-') return -readInt();
    while ((c = gc()) >= 48) a = a * 10 + c - 48;
    return a - 48; //d34
}
```

**BumpAllocator.h**  
**Description:** When you need to dynamically allocate many objects and don't care about freeing them. "new X" otherwise has an overhead of something like 0.05us + 16 bytes per allocation.

745db2, 8 lines

```
// Either globally or in a single class: //c17
static char buf[450 << 20];
void* operator new(size_t s) {
    static size_t i = sizeof(buf);
    assert(s < i);
    return (void*)&buf[i -= s]; //ef5
}
```

```
}
void operator delete(void*) {}

SmallPtr.h
Description: A 32-bit pointer that points into BumpAllocator memory.
"SmallPtr.h" 2dd6c9, 10 lines
template<class T> struct ptr { //bda
    unsigned ind;
    ptr(T* p = 0) : ind(p ? unsigned((char*)p - buf) : 0) {
        assert(ind < sizeof(buf));
    }
    T& operator*() const { return *(T*)(buf + ind); } //95f
    T* operator->() const { return &*this; }
    T& operator[](int a) const { return (&this)[a]; }
    explicit operator bool() const { return ind; }
};
```

**BumpAllocatorSTL.h**  
**Description:** BumpAllocator for STL containers.  
**Usage:** `vector<vector<int, small<int>>> ed(N);` `bb66d4, 14 lines`

```
char buf[450 << 20] alignas(16); //2c8
size_t buf_ind = sizeof(buf);
```

```
template<class T> struct small {
    typedef T value_type;
    small() {} //8ec
    template<class U> small(const U&) {}
    T* allocate(size_t n) {
        buf_ind -= n * sizeof(T);
        buf_ind &= 0 - alignof(T);
        return (T*)(buf + buf_ind); //ad1
    }
    void deallocate(T*, size_t) {}
};
```

## Extra Stuff (11)

**CentroidDecomposition.h**  
**Description:** Centroid decomposition on tree  
**Time:**  $\mathcal{O}(n \log n)$

918f4f, 38 lines

```
struct CD { //e81
    vector<vector<int>> adj;
    vector<int> size, vis;

    int dfs_size(int v, int p) {
        size[v] = 1; //43a
        for (int e : adj[v]) {
            if (e != p && !vis[e]) {
                size[v] += dfs_size(e, v);
            }
        } //ef7
        return size[v];
    }

    int dfs_root(int v, int p, int n) {
        for (int e : adj[v]) { //14b
            if (e != p && !vis[e] && 2 * size[e] > n) {
                return dfs_root(e, v, n);
            }
        }
        return v; //4ef
    }

    void centroid(int v, int p) {
        dfs_size(v, p);
        int c = dfs_root(v, p, size[v]); //44f
        vis[c] = true;
    }
};
```

```

    // do processing here
    // make sure to ignore visited nodes
//260
    for (int e : adj[c]) {
        if (!vis[e]) {
            centroid(e, c);
        }
    } //29b
}
};

```

## Eertree.h

**Time:** "Tree" of all palindromic substrings (there are two roots). Also has suffix links. 91804b, 56 lines

```

struct eertree { //076
    struct node {
        array<int, 26> nxt;
        int sufflink, len, cnt;
        vector<int> edges;
    }; //a3e

    string s;
    vector<node> tree;
    int suff, num;
//0a2
    eertree(const string &s) : s(s), tree(sz(s)+2), suff(2), num(2) {
        tree[1].len = -1, tree[1].sufflink = 1;
        tree[2].len = 0, tree[2].sufflink = 1;
        tree[1].edges.push_back(2);
        rep(i, 0, sz(s)) add(i); //736
    }

    void add(int pos) {
        int cur = suff, cur_len = 0;
        char c = s[pos]; //989

        while (true) {
            cur_len = tree[cur].len;
            if (pos - 1 - cur_len > -1 && s[pos - 1 - cur_len] == s[pos]) break;
            cur = tree[cur].sufflink; //b02
        }

        if (tree[cur].nxt[c]) {
            suff = tree[cur].nxt[c];
            tree[suff].cnt++; //be0
            return;
        }

        suff = ++num;
        tree[num].len = tree[cur].len + 2; //d5d
        tree[num].cnt = 1;
        tree[cur].nxt[c] = num;

        if (tree[num].len == 1) {
            tree[num].sufflink = 2; //bac
            tree[2].edges.push_back(num);
            return;
        }

        while (true) { //4bc
            cur = tree[cur].sufflink;
            cur_len = tree[cur].len;
            if (pos - 1 - cur_len > -1 && s[pos - 1 - cur_len] == s[pos]) {
                tree[num].sufflink = tree[cur].nxt[c];
                tree[tree[cur].nxt[c]].edges.push_back(num);
                //049
                break;
            }
        }
    }
};

```

```

    }
}
}; //214

```

## Knuth.h

**Description:** DP must be in the form  $dp(i, j) = \min[dp(i, k) + dp(k + 1, j) + C(i, j)]$  such that  $opt(i, j - 1) \leq opt(i, j) \leq opt(i + 1, j)$ . True if for  $a \leq b \leq c \leq d$ , then  $C(b, c) \leq C(a, d)$  and  $C(a, c) + C(b, d) \leq C(a, d) + C(b, c)$ .  
**Time:**  $\mathcal{O}(n^2)$  776312, 20 lines

```

template<class C> //e79
int solve(int n, C c) {
    vector dp(n, vi(n));
    auto opt = dp;
    rep(i, 0, n) opt[i][i] = i; // initialize dp[i][i] here
    for (int i = n-2; i >= 0; i--) { //157
        rep(j, i+1, n) {
            int mn = INT_MAX;
            int cost = C(i, j);
            for (int k = opt[i][j-1]; k <= min(j-1, opt[i+1][j]); k++) {
                if (mn >= dp[i][k] + dp[k+1][j] + cost) { //
                    201
                    opt[i][j] = k;
                    mn = dp[i][k] + dp[k+1][j] + cost;
                }
            }
            dp[i][j] = mn; //b2a
        }
    }
    return dp[0][n-1];
}

```

## ModInt.h

**Description:** all operations just work, MOD should fit in int  
**Time:** constant for operations,  $\mathcal{O}(\log e)$  for pow 209e30, 32 lines

```

constexpr int MOD = 1e9+7; //d87

struct mi {
    int v;
    mi() : mi(0) {}
    mi(int _v) : v(_v) { //715
        if (v >= MOD) v -= MOD;
        if (v < 0) v += MOD;
    }
    mi(ll _v) : mi((int)(_v % MOD)) {}
    mi operator+(const mi &m2) const { return mi(v + m2.v); } //89d
    mi operator-(const mi &m2) const { return mi(v - m2.v); }
    mi operator*(const mi &m2) const { return mi((ll) v * m2.v); }
    mi operator/(const mi &m2) const { return mi((ll) v * m2.inv().v); }
    mi &operator+=(const mi &m2) { return *this = *this + m2; }
    mi &operator-=(const mi &m2) { return *this = *this - m2; } //e99
    mi &operator*=(const mi &m2) { return *this = *this * m2; }
    mi &operator/=(const mi &m2) { return *this = *this / m2; }
    mi pow(ll e) const {
        mi res = 1;
        mi n = *this; //148
        while (e > 0) {
            if (e & 1) res *= n;
            n *= n;
        }
    }
};

```

```

        e >= 1;
    } //c3c
    return res;
}
mi inv() const {
    return pow(MOD - 2);
} //e03
};

```

## Pruefer.h

**Description:** Helps construct random tree Choose random n-2 length array, values [0, n-1]  
**Time:**  $\mathcal{O}(n)$  295d68, 23 lines

```

vector<pii> pruefer_decode(const vi &code) { //865
    int n = sz(code) + 2;
    vi degree(n, 1);
    for (int i : code)
        degree[i]++;
//c77
    set<int> leaves;
    rep(i, 0, n)
        if (degree[i] == 1)
            leaves.insert(i);
//f6a
    vector<pii> edges;
    for (int v : code) {
        int leaf = *leaves.begin();
        leaves.erase(leaves.begin());
//1b3
        edges.emplace_back(leaf, v);
        if (--degree[v] == 1)
            leaves.insert(v);
    }
    edges.emplace_back(*leaves.begin(), n-1); //062
    return edges;
}

```

## SuffixAutomaton.h

**Description:** Builds suffix automaton for a string. Each node corresponds to a class of substrings which end at the same indices.

**Time:**  $\mathcal{O}(n)$  bebd7b, 55 lines

```

struct SuffixAutomaton { //3d3
    struct Node {
        int len = 0, lnk = 0;
        int nxt[26];
    };
    string s; //c66
    vector<Node> t; int last = 0;
    SuffixAutomaton(string _s = "") {
        t.pb({0, -1, {}});
        for (char c : _s) add(c);
    } //c4c
    void add(int c) { s += (char) c; c -= 'a';
        int u = last; int v = last = sz(t);
        t.pb({t[u].len + 1, 0, {}});
        while (u >= 0 && !t[u].nxt[c])
            t[u].nxt[c] = v, u = t[u].lnk; //b21
        if (u == -1) return;
        int q = t[u].nxt[c];
        if (t[u].len + 1 == t[q].len)
            { t[v].lnk = q; return; }
        int cpy = sz(t); t.pb(t[q]); //aa4
        t[cpy].len = t[u].len + 1;
        while (u >= 0 && t[u].nxt[c] == q)
            t[u].nxt[c] = cpy, u = t[u].lnk;
        t[v].lnk = t[q].lnk = cpy;
    } //c74
    vi cnt() {
        vi res(sz(t), 0);
    }
};

```

```

    int cur = 0;
    for (char c : s)
        res[cur = t[cur].nxt[c - 'a']]++;//e36
    vector<pii> srt;
    rep(i, 1, sz(t))
        srt.pb({-t[i].len, i});
    sort(all(srt));
    for (auto &p : srt)//df9
        res[t[p.second].lnk] += res[p.second];
    return res;
}

vi first() {
    vi res(sz(t), sz(s));//b72
    int cur = 0;
    for (int i = 0; i < sz(s); i++) {
        cur = t[cur].nxt[s[i] - 'a'];
        res[cur] = min(res[cur], i);
    }//18b
    vector<pii> srt;
    rep(i, 1, sz(t))
        srt.pb({-t[i].len, i});
    sort(all(srt));
    for (auto &p : srt)//23d
        res[t[p.second].lnk] = min(res[t[p.second].lnk]
                                     , res[p.second]);
    return res;
}
};

```