Cairo University

Tmr Manga 7gr

Mohamed Tamer, Hossam ElGendi, Mohamed ElHagry

2024-11-08

Contest (1)

template.cpp 33 lines

```
#include <bits/stdc++.h>
#define pb push_back
#define F first
#define S second
#define MP make_pair
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
#define all(x) x.begin(),x.end()
#define FAST ios::sync_with_stdio(false);cout.tie(NULL);cin.tie
    (NULL);

using namespace std;
using ll = long long;
using pi = pair<int, int>;
using vi = vector<int>;
using vpi = vector <pair<int, int>>;
using vvi = vector <vector<int>>;

const int OO = 1e9 + 5;
const int N = 2e5 + 5;

void TC(){

}

int32_t main() {
    FAST
    int t = 1;
    cin >> t;
    while (t--) {
        TC();
    }
    return 0;
}
```

troubleshoot.txt 1 lines

```
3ala allah
```

Mathematics (2)

2.1 Equations

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

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

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

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

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

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

2.2 Recurrences

If $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, and $r_1, \ldots, r_k$ are distinct roots of $x^k - c_1 x^{k-1} - \cdots - c_k$, there are $d_1, \ldots, d_k$ s.t.

$$a_n = d_1 r_1^n + \cdots + d_k r_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

2.3 Trigonometry

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

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

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

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

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

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

2.4 Geometry

2.4.1 Triangles

Side lengths: a, b, c

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

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

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

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

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

Length of bisector (divides angles in two):

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

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

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

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

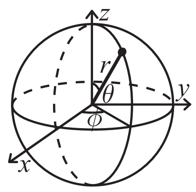
2.4.2 Quadrilaterals

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

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

2.4.3 Spherical coordinates

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



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

2.5 Derivatives/Integrals

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

Integration by parts:

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

2.6 Sums

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

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

2.7 Series

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

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

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

$$\sin x = x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\dots, (-\infty < x < \infty)$$

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

2.8 Probability theory

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

Expectation is linear:

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

For independent X and Y,

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

2.8.1 Discrete distributions

Binomial distribution

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

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

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

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

First success distribution

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

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

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

Poisson distribution

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

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

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

2.8.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

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

Normal distribution

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

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

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

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

2.9 Markov chains

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

π 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. π_j/π_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, π_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). $\lim_{k\to\infty}\mathbf{P}^k = \mathbf{1}\pi$.

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

Data structures (3)

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>
using namespace __gnu_pbds;

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

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

HashMap.h

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

d77092, 7 lines

```
#include <bits/extc++.h>
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
  const uint64_t C = ll(4e18 * acos(0)) | 71;
  ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
__gnu_pbds::gp_hash_table<ll,int,chash> h({},{},{},{},{1<<16});
```

SubMatrix.h

Description: Calculate submatrix sums quickly, given upper-left and lower-right corners (half-open).
Usage: `SubMatrix<int> m(matrix);`
`m.sum(0, 0, 2, 2); // top left 4 elements`
Time: $\mathcal{O}(N^2 + Q)$

c59ada, 13 lines

```
template<class T>
struct SubMatrix {
  vector<vector<T>> p;
  SubMatrix(vector<vector<T>>& v) {
    int R = sz(v), C = sz(v[0]);
    p.assign(R+1, vector<T>(C+1));
    rep(r,0,R) rep(c,0,C)
      p[r+1][c+1] = v[r][c] + p[r][c+1] + p[r+1][c] - p[r][c];
  }
  T sum(int u, int l, int d, int r) {
    return p[d][r] - p[d][l] - p[u][r] + p[u][l];
  }
};
```

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 {
  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)
      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);
    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);
    M a, b(*this);
    rep(i,0,N) a.d[i][i] = 1;
    while (p) {
      if (p&1) a = a*b;
      b = b*b;
      p >>= 1;
    }
    return a;
  }
};
```

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

8ec1c7, 30 lines

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

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

SparseTable.cpp

Description: Nice sparse table template

686e45, 27 lines

```
struct SparseTable {
    int n, lg;
    vector<vector<int>> sparseTable;
    vector<int> bigPow;
    SparseTable(vector<int> &a) {
        n = a.size();
        lg = __lg(n) + 2;
        sparseTable.resize(n, vector<int>(lg));
        bigPow.resize(n + 1);
        for (int k = 0; k < lg; k++) {
            for (int i = 0; i + (1 << k) - 1 < n; i++) {
                if (k == 0)
                    sparseTable[i][k] = a[i];
                else
                    sparseTable[i][k] = max(sparseTable[i][k -
                        1], sparseTable[i + (1 << (k - 1))][k
                        - 1]);
            }
        }
        bigPow[1] = 0;
        for (int k = 2; k <= n; k++)
            bigPow[k] = bigPow[k / 2] + 1;
    }
    int query(int l, int r) {
        int len = r - l + 1;
        int k = bigPow[len];
        return max(sparseTable[l][k], sparseTable[r - (1 << k)
            + 1][k]);
    }
};
```

BIT.cpp

Description: Executes point update/ range queries both in O(log(N)) on arrays for invertible functions, can query prefix for all functions

7a1a6d, 35 lines

```
const int N = 1e5 + 5;
struct BIT {
    vector<ll> tree;
    BIT(int _n = N) {
        tree.resize(_n + 2);
    }
    ll get_prefix(int k) {
        ll ans = 0;
        while (k >= 1) {
            ans += tree[k];
            k -= k & -k;
        }
        return ans;
    }
    void update_point(int k, ll v) {
        while (k < tree.size()) {
            tree[k] += v;
            k += k & -k;
        }
    }
    ll query(int l, int r) {
        return get_prefix(r) - get_prefix(l - 1);
    }
    // binary search for first prefix with sum x
    int BS(ll x) {
        int pos = 0;
        for (int sz = (1 << __lg(tree.size())); sz > 0 && x; sz
            >>= 1) {
            if (pos + sz < tree.size() && tree[pos + sz] < x) {
                x -= tree[pos + sz];
                pos += sz;
            }
```

```
        }
        return pos + 1;
    }
};
```

BIT2D.cpp

Description: Executes point update/ range queries both in $O((\log(N))^2)$ on a grid of size $O(N \times N)$ for invertible functions, can query prefix for all functions

e76240, 31 lines

```
const int N = 1e3 + 5;

struct BIT2D {
    vector<vector<ll>> tree;
    BIT2D(int _n = N) {
        tree.resize(_n + 2, vector<ll>(_n + 2));
    }
    ll get_prefix(int i, int j) {
        ++i;
        ++j;
        ll sum = 0;
        for (int x = i; x >= 1; x -= x & -x) {
            for (int y = j; y >= 1; y -= y & -y) {
                sum += tree[x][y];
            }
        }
        return sum;
    }
    void update_point(int i, int j, ll v) {
        ++i;
        ++j;
        for (int x = i; x < tree.size(); x += x & -x) {
            for (int y = j; y < tree.size(); y += y & -y) {
                tree[x][y] += v;
            }
        }
    }
    ll query(int x1, int y1, int x2, int y2) {
        return get_prefix(x2, y2) - get_prefix(x1 - 1, y2) -
            get_prefix(x2, y1 - 1) + get_prefix(x1 - 1, y1 -
            1);
    }
};
```

waveletTree.cpp

Description: Allows very weird queries

c48042, 51 lines

```
struct wavelet_tree {
#define vi vector<int>
#define pb push_back
    int lo, hi;
    wavelet_tree *l, *r;
    vi b;
    //nos are in range [x,y]
    //array indices are [from, to)
    //(usually wavelet_tree(arr+1, arr+n+1, MIN, MAX))
    wavelet_tree(int *from, int *to, int x, int y) {
        lo = x, hi = y;
        if (lo == hi or from >= to) return;
        int mid = (lo + hi) / 2;
        auto f = [mid](int x) {
            return x <= mid;
        };
        b.reserve(to - from + 1);
        b.pb(0);
        for (auto it = from; it != to; it++)
            b.pb(b.back() + f(*it));
        //see how lambda function is used here
        auto pivot = stable_partition(from, to, f);
        l = new wavelet_tree(from, pivot, lo, mid);
        r = new wavelet_tree(pivot, to, mid + 1, hi);
    }
    //kth smallest element in [l, r] (1-based)
    int kth(int l, int r, int k) {
        if (l > r) return 0;
        if (lo == hi) return lo;
        int inLeft = b[r] - b[l - 1];
        int lb = b[l - 1]; //amt of nos in first (l-1) nos that
            go in left
        int rb = b[r]; //amt of nos in first (r) nos that go in
            left
        if (k <= inLeft) return this->l->kth(lb + 1, rb, k);
        return this->r->kth(l - lb, r - rb, k - inLeft);
    }
    //count of nos in [l, r] Less than or equal to k (1-based)
    int LTE(int l, int r, int k) {
        if (l > r or k < lo) return 0;
        if (hi <= k) return r - l + 1;
        int lb = b[l - 1], rb = b[r];
        return this->l->LTE(lb + 1, rb, k) + this->r->LTE(l -
            lb, r - rb, k);
    }
    //count of nos in [l, r] equal to k (1-based)
    int count(int l, int r, int k) {
        if (l > r or k < lo or k > hi) return 0;
        if (lo == hi) return r - l + 1;
        int lb = b[l - 1], rb = b[r], mid = (lo + hi) / 2;
        if (k <= mid) return this->l->count(lb + 1, rb, k);
        return this->r->count(l - lb, r - rb, k);
    }
};
```

waveletTreeFast.cpp

Description: Wavelet tree fast

c73ff0, 61 lines

```
/*
note:
- w stores the array elements of each node
- b stores the prefix sum of frequency of elements <= mid of
    each node
- lc contains the node number of the left child of a node
- rc contains the node number of the right child of a node
- nxt is used to find the new node number to assign to a node
- in is used to allot space in the w array for each node
- [l[nd],r[nd]] is the range for elements of node nd in w and b
- psz is the number of elements in the parent of a node
- pnd is the parent of a node
- f is 1 if the current node is a left child, 0 otherwise
*/
#define mxn 100005 //array size
#define mxval 100005 //max array element
#define mxt 2000005 //max number of nodes needed, approximately
    n*(log(mxval)+4)
const int from = 0, to = mxval;
int n, q, arr[mxn], w[mxt], nxt = 1, in = 0;
int lc[mxt], rc[mxt], l[mxt], r[mxt];
ll b[mxt];
// arr (1-based)
void build(int psz = -1, bool f = 1, int pnd = -1, int nd = 1,
    int s = from, int e = to) {
    l[nd] = ++in, r[nd] = in - 1;
    int midp = psz >> 1, mid = (s + e) >> 1, i1 = (nd == 1) ? n
        : r[pnd];
    for (int i = (nd == 1) ? 1 : l[pnd]; i <= i1; i++)
        if (nd == 1 || (f && w[i] <= midp) || (!f && w[i] >
            midp))
            w[in] = (nd == 1) ? arr[i] : w[i], r[nd] = in,
            b[in] = b[in - 1] + (w[in] <= mid), in++;
    if (s == e) return;
    int sz = (nd == 1) ? n : r[nd] - l[nd] + 1;
    if (b[r[nd]] - b[l[nd] - 1]) lc[nd] = ++nxt, build(s + e,
        1, nd, lc[nd], s, mid);
    if (b[r[nd]] - b[l[nd] - 1] != sz) rc[nd] = ++nxt, build(s
        + e, 0, nd, rc[nd], mid + 1, e);
}
//kth smallest element in range [l1,r1] (0-based)
int kth(int l1, int r1, int k, int nd = 1, int s = from, int e
    = to) {
    if (s == e) return s;
    int mid = (s + e) >> 1;
    int got = b[l[nd] + r1] - b[l[nd] + l1 - 1];
    if (got >= k) return kth(b[l[nd] + l1 - 1], b[l[nd] + r1] -
        1, k, lc[nd], s, mid);
    return kth(l1 - b[l[nd] + l1 - 1], r1 - b[l[nd] + r1], k -
        got, rc[nd], mid + 1, e);
}
//count of k in range [l1,r1] (0-based)
int count(int l1, int r1, int k, int nd = 1, int s = from, int
    e = to) {
    if (s == e) return b[l[nd] + r1] - b[l[nd] + l1 - 1];
    int mid = (s + e) >> 1;
    if (mid >= k) return count(b[l[nd] + l1 - 1], b[l[nd] + r1]
        - 1, k, lc[nd], s, mid);
    return count(l1 - b[l[nd] + l1 - 1], r1 - b[l[nd] + r1], k,
        rc[nd], mid + 1, e);
}
//count of numbers <= to k in range [l1,r1] (0-based)
int LTE(int l1, int r1, int k, int nd = 1, int s = from, int e
    = to) {
    if (l1 > r1 || k < s) return 0;
    if (e <= k) return r1 - l1 + 1;
    int mid = (s + e) >> 1;
    return LTE(b[l[nd] + l1 - 1], b[l[nd] + r1] - 1, k, lc[nd],
        s, mid) +
         LTE(l1 - b[l[nd] + l1 - 1], r1 - b[l[nd] + r1], k,
            rc[nd], mid + 1, e);
}
void clr() {
    in = 0;
    nxt = 1;
    memset(b, 0, sizeof b);
}
```

3.1 Segment Trees

SegTree.cpp

Description: It's a segment tree dude $O(\log(N))$ for query, $O(r-l)$ for update

17357a, 46 lines

```
struct SegTree {
    vector<ll> tree;
    int n;
    const ll IDN = OO;

    ll combine(ll a, ll b) {
        return min(a, b);
    }
    void build(int inputN, vector<ll>& a) {
        n = inputN;
        if (__builtin_popcount(n) != 1)
            n = 1 << (__lg(n) + 1);
        tree.resize(n << 1, IDN);
        for (int i = 0; i < inputN; i++)
            tree[i + n] = a[i];
        for (int i = n - 1; i >= 1; i--)
            tree[i] = combine(tree[i << 1], tree[i << 1 | 1]);
    }
    void update(int ql, int qr, ll v, int k, int sl, int sr) {
```

```
        if (qr < sl || sr < ql || ql > qr) return;
        if (ql <= sl && qr >= sr) {
            tree[k] = v;
            return;
        }

        int mid = (sl + sr) / 2;
        update(ql, qr, v, k << 1, sl, mid);
        update(ql, qr, v, (k << 1) | 1, mid + 1, sr);
        tree[k] = combine(tree[k << 1], tree[k << 1 | 1]);
    }
    ll query(int ql, int qr, int k, int sl, int sr) {
        if (qr < sl || sr < ql || ql > qr) return IDN;
        if (ql <= sl && qr >= sr) return tree[k];

        int mid = (sl + sr) / 2;
        ll left = query(ql, qr, k << 1, sl, mid);
        ll right = query(ql, qr, k << 1 | 1, mid + 1, sr);
        return combine(left, right);
    }
    void update(int ql, int qr, ll v){
        update(ql, qr, v, 1, 0, n-1);
    }
    ll query(int ql, int qr){
        return query(ql, qr, 1, 0, n-1);
    }
};
```

SegTreeLazy.cpp

Description: Lazy Segment Tree

26771a, 62 lines

```
struct SegTree {
    vector <ll> tree;
    vector <ll> lazy;
    int n;
    const ll IDN = OO;
    const ll LAZY_IDN = 0;

    ll combine(ll a, ll b) {
        return min(a, b);
    }
    void build(int inputN, const vector<ll>& a) {
        n = inputN;
        if (__builtin_popcount(n) != 1)
            n = 1 << (__lg(n) + 1);
        tree.resize(n << 1, IDN);
        lazy.resize(n << 1, LAZY_IDN);
        for (int i = 0; i < inputN; i++)
            tree[i + n] = a[i];
        for (int i = n - 1; i >= 1; i--)
            tree[i] = combine(tree[i << 1], tree[i << 1 | 1]);
    }
    void propagate(int k, int sl, int sr) {
        if (lazy[k] != LAZY_IDN) {
            tree[k] += lazy[k];
            if (sl != sr) {
                lazy[k << 1] += lazy[k];
                lazy[k << 1 | 1] += lazy[k];
            }
        }
        lazy[k] = LAZY_IDN;
    }
    void update(int ql, int qr, ll v, int k, int sl, int sr) {
        propagate(k, sl, sr);
        if (qr < sl || sr < ql || ql > qr) return;
        if (ql <= sl && qr >= sr) {
            lazy[k] = v;
            propagate(k, sl, sr);
            return;
        }

        int mid = (sl + sr) / 2;
        update(ql, qr, v, k << 1, sl, mid);
        update(ql, qr, v, (k << 1) | 1, mid + 1, sr);
        tree[k] = combine(tree[k << 1], tree[k << 1 | 1]);
    }
    ll query(int ql, int qr, int k, int sl, int sr) {
        propagate(k, sl, sr);
        if (qr < sl || sr < ql || ql > qr) return IDN;
        if (ql <= sl && qr >= sr) return tree[k];

        int mid = (sl + sr) / 2;
        ll left = query(ql, qr, k << 1, sl, mid);
        ll right = query(ql, qr, k << 1 | 1, mid + 1, sr);
        return combine(left, right);
    }
    void update(int ql, int qr, ll v){
        update(ql, qr, v, 1, 0, n-1);
    }
    ll query(int ql, int qr){
        return query(ql, qr, 1, 0, n-1);
    }
};
```

PersistentSegmentTree.cpp

Description: Dynamic Persistent Segment tree

57b340, 82 lines

```
struct Vertex {
    Vertex *l, *r;
    int sum = 0;

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

    Vertex() : l(nullptr), r(nullptr) {}

    Vertex(Vertex *l, Vertex *r) : l(l), r(r), sum(0) {
        if (l) sum += l->sum;
        if (r) sum += r->sum;
    }

    void addChild(){
        l = new Vertex();
        r = new Vertex();
    }
};

struct Seg {
    int n;

    Seg(int n) {
        this->n = n;
    }

    Vertex merge(Vertex x, Vertex y) {
        Vertex ret;
        ret.sum = x.sum + y.sum;
        return ret;
    }

    Vertex *update(Vertex *v, int i, int lx, int rx) {
        if (lx == rx)
            return new Vertex(v->sum + 1);
        int mid = (lx + rx) / 2;
        if(!v->l)v->addChild();
        if (i <= mid) {
            return new Vertex(update(v->l, i, lx, mid), v->r);
        } else {
            return new Vertex(v->l, update(v->r, i, mid + 1, rx
                ));
        }
    }

    Vertex *update(Vertex *v, int i) {
        return update(v, i, 0, n - 1);
    }

    Vertex query(Vertex *v, int l, int r, int lx, int rx) {
        if (l > rx || r < lx)
            return {};
        if (l <= lx && r >= rx)
            return *v;
        if(!v->l)v->addChild();
        int mid = (lx + rx) / 2;
        return merge(query(v->l, l, r, lx, mid), query(v->r, l,
             r, mid + 1, rx));
    }

    Vertex query(Vertex *v, int l, int r) {
        return query(v, l, r, 0, n - 1);
    }

    int getKth(Vertex *a, Vertex *b, int k, int lx, int rx) {
        if (lx == rx) {
            return lx;
        }
        if(!a->l)a->addChild();
        if(!b->l)b->addChild();
        int rem = b->l->sum - a->l->sum;
        int mid = (lx + rx) / 2;
        if (rem >= k)
            return getKth(a->l, b->l, k, lx, mid);
        else
            return getKth(a->r, b->r, k - rem, mid + 1, rx);
    }

    int getKth(Vertex *a, Vertex *b, int k) {
        return getKth(a, b, k, 0, n - 1);
    }
};
```

DynamicLi-ChaoTree.cpp

Description: Dynamic Li Chao Tree

09ca0b, 83 lines

```
const ll OO = 1e18 + 5;
const ll maxN = 1e6 + 5;

struct Line {
    ll m, c;

    Line() : m(0), c(OO) {}

    Line(ll m, ll c) : m(m), c(c) {}
};

ll sub(ll x, Line l) {
    return x * l.m + l.c;
}

// Li Chao sparse
struct node {
    // range I am responsible for
    Line line;
    node *left, *right;

    node() {
        left = right = NULL;
```

```
    }

    node(ll m, ll c) {
        line = Line(m, c);
        left = right = NULL;
    }

    void extend(int l, int r) {
        if (left == NULL && l != r) {
            left = new node();
            right = new node();
        }
    }

    void add(Line toAdd, int l, int r) {
        assert(l <= r);
        int mid = (l + r) / 2;
        if (l == r) {
            if (sub(l, toAdd) < sub(l, line))
                swap(toAdd, line);
            return;
        }
        bool lef = sub(l, toAdd) < sub(l, line);
        bool midE = sub(mid+1, toAdd) < sub(mid+1, line);
        if(midE)
            swap(line, toAdd);
        extend(l, r);
        if(lef != midE)
            left->add(toAdd, l, mid);
        else
            right->add(toAdd, mid+1, r);
    }

    void add(Line toAdd) {
        add(toAdd, 0, maxN-1);
    }

    ll query(ll x, int l, int r) {
        int mid = (l + r) / 2;
        if (l == r || left == NULL)
            return sub(x, line);
        extend(l, r);
        if (x <= mid)
            return min(sub(x, line), left->query(x, l, mid));
        else
            return min(sub(x, line), right->query(x, mid+1, r))
                ;
    }

    ll query(ll x) {
        return query(x, 0, maxN-1);
    }

    void clear() {
        if (left != NULL) {
            left->clear();
            right->clear();
        }
        delete this;
    }
};
```

DynamicPersistentLi-ChaoTree.cpp

Description: Dynamic Persistent Li Chao Tree

710ff2, 91 lines

```
const ll OO = 1e18 + 5;
const ll maxN = 1e9 + 5;
struct Line {
    ll m, c;
    Line() : m(0), c(OO) {}
    Line(ll m, ll c) : m(m), c(c) {}
};

ll sub(ll x, Line l) {
    return x * l.m + l.c;
}

// Persistent Li Chao
struct Node {
    // range I am responsible for
    Line line;
    Node *left, *right;

    Node() {
        left = right = NULL;
    }

    Node(ll m, ll c) {
        line = Line(m, c);
        left = right = NULL;
    }

    void extend(int l, int r) {
        if (left == NULL && l != r) {
            left = new Node();
            right = new Node();
        }
    }

    Node* copy(Node* node){
        Node* newNode = new Node;
        newNode->left = node->left;
        newNode->right = node->right;
        newNode->line = node->line;
        return newNode;
    }

    Node* add(Line toAdd, int l, int r) {
        assert(l <= r);
        int mid = (l + r) / 2;
        Node* cur = copy(this);
        if (l == r) {
            if (sub(l, toAdd) < sub(l, cur->line))
                swap(toAdd, cur->line);
            return cur;
        }
        bool lef = sub(l, toAdd) < sub(l, cur->line);
        bool midE = sub(mid+1, toAdd) < sub(mid+1, cur->line);
        if(midE)
            swap(cur->line, toAdd);
        cur->extend(l, r);
        if(lef != midE)
            cur->left = cur->left->add(toAdd, l, mid);
        else
            cur->right = cur->right->add(toAdd, mid+1, r);
        return cur;
    }

    Node* add(Line toAdd) {
        return add(toAdd, 0, maxN-1);
    }

    ll query(ll x, int l, int r) {
        int mid = (l + r) / 2;
        if (l == r || left == NULL)
            return sub(x, line);
        extend(l, r);
        if (x <= mid)
            return min(sub(x, line), left->query(x, l, mid));
        else
            return min(sub(x, line), right->query(x, mid+1, r))
                ;
    }

    ll query(ll x) {
        return query(x, 0, maxN-1);
    }

    void clear() {
        if (left != NULL) {
            left->clear();
            right->clear();
        }
        delete this;
    }
};
Node* tree[N];
```

LinearPolyUpdateSegTree.cpp

Description: Allows updates of the form $ax + b$ on an arbitrary range

c12ebd, 115 lines

```
const int N = 2e5 + 5;
const int MOD = 1e9 + 7;
int add(ll a, ll b) {
    a %= MOD, b %= MOD;
    a += b;
    if (a >= MOD) a -= MOD;
    return a;
}
int mul(ll a, ll b) { return (a % MOD) * (b % MOD) % MOD; }
int powmod(ll x, ll y) {
    x %= MOD;
    int ans = 1;
    while (y) {
        if (y & 1) ans = mul(ans, x);
        x = mul(x, x);
        y >>= 1;
    }
    return ans;
}
void normalize(ll &a) {
    while (a < 0)
        a += MOD;
}
struct Node {
    ll a, b;
    Node() {}
    Node(ll _a, ll _b) : a(_a), b(_b) { normalize(); }
    void normalize() {
        ::normalize(a);
        ::normalize(b);
    }
    bool operator==(const Node &other) {
        return a == other.a && b == other.b;
    }
    bool operator!=(const Node &other) {
        return a != other.a || b != other.b;
    }
};
ll sumTerms[N];
void pre(){
    for(int i =1; i <N; ++i){
        sumTerms[i] = i + sumTerms[i-1];
        if(sumTerms[i] >= MOD)
            sumTerms[i] -= MOD;
    }
}
```

```cpp
struct SegTree {
    vector<ll> tree;
    vector<Node> lazy;
    int n;
    const ll IDN = 0;
    const Node LAZY_IDN = Node(0, 0);
    ll combine(ll a, ll b) {
        return add(a, b);
    }
    Node combineNodes(Node lt, Node rt) {
        return Node(add(lt.a, rt.a), add(lt.b, rt.b));
    }
    Node shiftNode(Node node, ll shift) {
        normalize(shift);
        node.b = add(node.b, mul(shift, node.a));
        node.normalize();
        return node;
    }
    void build(int inputN) {
        n = inputN;
        if (__builtin_popcount(n) != 1)
            n = 1 << (__lg(n) + 1);
        tree.resize(n << 1, IDN);
        lazy.resize(n << 1, LAZY_IDN);
    }
    void propagate(int k, int sl, int sr) {
        if (lazy[k] != LAZY_IDN) {
            tree[k] = add(tree[k], mul(lazy[k].a, sumTerms[sr -
                 sl]));
            tree[k] = add(tree[k], mul(lazy[k].b, (sr - sl + 1)
                ));
            if (sl != sr) {
                int mid = (sl + sr) / 2;
                lazy[k << 1] = combineNodes(lazy[k << 1], lazy[
                    k]);
                lazy[k << 1 | 1] = combineNodes(lazy[k << 1 |
                    1],
                                                shiftNode(lazy[
                                                    k], mid +
                                                    1 - sl));
            }
        }
        lazy[k].a = lazy[k].b = 0;
    }
    void update(int ql, int qr, Node v, int k, int sl, int sr)
     {
        propagate(k, sl, sr);
        if (qr < sl || sr < ql || ql > qr) return;
        if (ql <= sl && qr >= sr) {
            lazy[k] = v;
            propagate(k, sl, sr);
            return;
        }
        int mid = (sl + sr) / 2;
        update(ql, qr, v, k << 1, sl, mid);
        Node shiftedNode = shiftNode(v, mid + 1 - sl);
        update(ql, qr, shiftedNode, (k << 1) | 1, mid + 1, sr);
        tree[k] = combine(tree[k << 1], tree[k << 1 | 1]);
    }
    ll query(int ql, int qr, int k, int sl, int sr) {
        propagate(k, sl, sr);
        if (qr < sl || sr < ql || ql > qr) return IDN;
        if (ql <= sl && qr >= sr) return tree[k];
        int mid = (sl + sr) / 2;
        ll left = query(ql, qr, k << 1, sl, mid);
        ll right = query(ql, qr, k << 1 | 1, mid + 1, sr);
        return combine(left, right);
    }
    void update(int ql, int qr, Node node) {
        node = shiftNode(node, -ql);
        update(ql, qr, node, 1, 0, n - 1);
    }
    ll query(int ql, int qr) {
        return query(ql, qr, 1, 0, n - 1);
    }
};
```

QuadraticPolyUpdateSegTree.cpp

Description: Allows updates of the form ax^2+bx+c on an arbitrary range

b6d7ac, 131 lines

```cpp
const ll MOD = 1e9 + 7;
void normalize(ll &a) {
    while (a < 0)
        a += MOD;
}
struct Node {
    ll a, b, c;
    Node() {}
    Node(ll _a, ll _b, ll _c) : a(_a), b(_b), c(_c) {
        normalize();
    }
    void normalize() {
        ::normalize(a);
        ::normalize(b);
        ::normalize(c);
    }
    bool operator==(const Node &other) {
        return a == other.a && b == other.b && c == other.c;
    }
    bool operator!=(const Node &other) {
        return a != other.a || b != other.b || c != other.c;
    }
};
int add(ll a, ll b) {
    assert(a >= 0);
    assert(b >= 0);
    a %= MOD, b %= MOD;
    a += b;
    if (a >= MOD) a -= MOD;
    return a;
}
int mul(ll a, ll b) {
    assert(a >= 0);
    assert(b >= 0);
    return (a % MOD) * (b % MOD) % MOD;
}
int powmod(ll x, ll y) {
    x %= MOD;
    int ans = 1;
    while (y) {
        if (y & 1) ans = mul(ans, x);
        x = mul(x, x);
        y >>= 1;
    }
    return ans;
}
int inv(ll a) { return powmod(a, MOD - 2); }
ll sumTerms(ll x) {
    return x * (x + 1) / 2 % MOD;
}
ll sumSquares(ll x) {
    return x * (x + 1) * (2 * x + 1) / 6 % MOD;
}
struct SegTree {
    vector<ll> tree;
    vector<Node> lazy;
    int n;
    const ll IDN = 0;
    const Node LAZY_IDN = Node(0, 0, 0);
    ll combine(ll a, ll b) {
        return add(a, b);
    }
    Node combineNodes(Node lt, Node rt) {
        return Node(add(lt.a, rt.a), add(lt.b, rt.b), add(lt.c,
             rt.c));
    }
    Node shiftNode(Node node, ll shift) {
        // = a * (x + s)^2 + b * (x + s) + c
        // = a * (x^2 + 2*x*s + s^2) + b * x + b * s + c
        // = a * x^2 + a*2*x*s + a*s^2 + b * x + b * s + c
        // = a* x^2 + (2*a*s + b) * x + ( a * s^2 + b * s + c)
        normalize(shift);
        Node newNode;
        newNode.a = node.a;
        newNode.b = add(node.b, mul(node.a, shift * 2));
        newNode.c = add(node.c, add(mul(node.b, shift), mul(
            node.a, mul(shift, shift))));
        newNode.normalize();
        return newNode;
    }
    void build(int inputN) {
        n = inputN;
        if (__builtin_popcount(n) != 1)
            n = 1 << (__lg(n) + 1);
        tree.resize(n << 1, IDN);
        lazy.resize(n << 1, LAZY_IDN);
    }
    void propagate(int k, int sl, int sr) {
        if (lazy[k] != LAZY_IDN) {
            tree[k] = add(tree[k], mul(lazy[k].a, sumSquares(sr
                 - sl)));
            tree[k] = add(tree[k], mul(lazy[k].b, sumTerms(sr -
                 sl)));
            tree[k] = add(tree[k], mul(lazy[k].c, (sr - sl + 1)
                ));
            if (sl != sr) {
                int mid = (sl + sr) / 2;
                lazy[k << 1] = combineNodes(lazy[k << 1], lazy[
                    k]);
                lazy[k << 1 | 1] = combineNodes(lazy[k << 1 |
                    1],
                                                shiftNode(lazy[
                                                    k], mid +
                                                    1 - sl));
            }
        }
        lazy[k].a = lazy[k].b = lazy[k].c = 0;
    }
    void update(int ql, int qr, Node v, int k, int sl, int sr)
     {
        propagate(k, sl, sr);
        if (qr < sl || sr < ql || ql > qr) return;
        if (ql <= sl && qr >= sr) {
            lazy[k] = v;
            propagate(k, sl, sr);
            return;
        }
        int mid = (sl + sr) / 2;
        update(ql, qr, v, k << 1, sl, mid);
        Node shiftedNode = shiftNode(v, mid + 1 - sl);
        update(ql, qr, shiftedNode, (k << 1) | 1, mid + 1, sr);
        tree[k] = combine(tree[k << 1], tree[k << 1 | 1]);
    }
    ll query(int ql, int qr, int k, int sl, int sr) {
        propagate(k, sl, sr);
        if (qr < sl || sr < ql || ql > qr) return IDN;
        if (ql <= sl && qr >= sr) return tree[k];
```

```
        int mid = (sl + sr) / 2;
        ll left = query(ql, qr, k << 1, sl, mid);
        ll right = query(ql, qr, k << 1 | 1, mid + 1, sr);
        return combine(left, right);
    }
    void update(int ql, int qr, Node node) {
        node = shiftNode(node, -ql);
        update(ql, qr, node, 1, 0, n - 1);
    }
    ll query(int ql, int qr) {
        return query(ql, qr, 1, 0, n - 1);
    }
};
```

3.2 DSUStuff

DSU.cpp

Description: 1-indexed DSU d01417, 30 lines

```
struct DSU {
    int n, comps;
    vector<int> sz, par;

    DSU(int n) {
        this->n = n;
        comps = n;
        sz.resize(n + 1);
        par.resize(n + 1);
        for (int i = 1; i <= n; ++i) {
            sz[i] = 1;
            par[i] = i;
        }
    }

    int find(int x) {
        if (par[x] == x) return x;
        return find(par[x]);
    }

    bool unite(int a, int b) {
        a = find(a), b = find(b);
        if (a == b) return false;
        if (sz[a] < sz[b]) swap(a, b);
        par[b] = a;
        sz[a] += sz[b];
        comps--;
        return true;
    }
};
```

DSUWithCheckpoints.cpp

Description: 1-Indexed DSU with checkpoints and rollbacks 7994d9, 59 lines

```
struct Save {
    int big, small;
    bool isCheckPoint;
};

struct DSU {
    vi par, sz;
    int comps;
    stack<Save> saves;

    DSU(int n) {
        par.resize(n + 1);
        sz.resize(n + 1);
        comps = n;
        for (int i = 1; i <= n; ++i) {
            par[i] = i;
            sz[i] = 1;
        }
        saves = stack<Save>();
    }

    int find(int x) {
        if (par[x] == x)return x;
        return find(par[x]);
    }

    bool unite(int u, int v) {
        u = find(u);
        v = find(v);
        if (u == v)return false;

        if (sz[u] < sz[v])swap(u, v);

        saves.push({u, v, false});
        par[v] = u;
        sz[u] += sz[v];
        comps--;
        return true;
    }

    void persist() {
        saves.push({-1, -1, true});
    }

    void rollback() {
        while (!saves.top().isCheckPoint) {
            auto save = saves.top();
            saves.pop();
            comps++;
            par[save.small] = save.small;
            sz[save.big] -= sz[save.small];
        }
        saves.pop();
    }

    bool same(int u, int v) {
        return find(u) == find(v);
    }
};
```

DynamicConnectivity.cpp

Description: Dynamic Connectivity Offline 616026, 79 lines

```
struct Query {
    char t;
    int u, v;
};
struct Elem {
    int u, v, szU, cnt;
};
struct DSURollback {
    int cnt, n;
    stack <Elem> st;
    vector<bool> ans;
    vector<int> sz, par;
    vector <vector<pair < int, int>>>g;
    DSURollback(int _n) {
        cnt = _n;
        n = 1;
        while (n < _n)n *= 2;
        g.resize(2 * n + 5);
        par.resize(_n + 1);
        sz.resize(_n + 1, 1);
        iota(all(par), 0);
    }
    void rollback(int x) {
        while (st.size() > x) {
            auto e = st.top();
            st.pop();
            cnt = e.cnt;
            sz[e.u] = e.szU;
            par[e.v] = e.v;
        }
    }
    int findSet(int u) {
        return par[u] == u ? u : findSet(par[u]);
    }
    void update(int u, int v) {
        st.push({u, v, sz[u], cnt});
        cnt--;
        par[v] = u;
        sz[u] += sz[v];
    }
    void unionSet(int u, int v) {
        u = findSet(u);
        v = findSet(v);
        if (u != v) {
            if (sz[u] < sz[v])
                swap(u, v);
            update(u, v);
        }
    }
    void solve(int x, int l, int r) {
        int cur = st.size();
        for (auto i: g[x])
            unionSet(i.first, i.second);
        if (l == r) {
            if (ans[l])
                cout << cnt << endl;
            rollback(cur);
            return;
        }
        int m = (l + r) >> 1;
        solve(x * 2, l, m);
        solve(x * 2 + 1, m + 1, r);
        rollback(cur);
    }
    void traverse(int x, int lX, int rX, int l, int r, int u,
         int v) {
        if (rX < l || lX > r)
            return;
        if (lX >= l && rX <= r) {
            g[x].emplace_back(u, v);
            return;
        }
        int m = (lX + rX) >> 1;
        traverse(x * 2, lX, m, l, r, u, v);
        traverse(x * 2 + 1, m + 1, rX, l, r, u, v);
    }
    void update(int u, int v, int l, int r) {
        traverse(1, 0, n - 1, l, r, u, v);
    }
};
```

Number theory (4)

4.1 Our Templates

sieve.cpp

Description: sieve f17eba, 24 lines

```
const int N = 1e6 + 5;
int SPF[N];
```

```
void sieve() {
    for (int x = 1; x < N; x++)
        SPF[x] = x;
    for (ll x = 2; x < N; x++) {
        if (SPF[x] != x)
            continue;
        for (ll i = x * x; i < N; i += x) {
            if (SPF[i] != i)
                continue;
            SPF[i] = (int) x;
        }
    }
}
map<int, int> factorize(int x) {
    map<int, int> facts;
    while (x > 1) {
        int p = SPF[x];
        facts[p]++;
        x /= p;
    }
    return facts;
}
```

SegmentedSieve.cpp

Description: factorize numbers in the range L to R by running sieve up to sqrt(R) then using those primes to factorize

91fe31, 20 lines

```
vector<char> segmentedSieve(long long L, long long R) {
// generate all primes up to sqrt(R)
    long long lim = sqrt(R);
    vector<char> mark(lim + 1, false);
    vector<long long> primes;
    for (long long i = 2; i <= lim; ++i) {
        if (!mark[i]) {
            primes.emplace_back(i);
            for (long long j = i * i; j <= lim; j += i)
                mark[j] = true;
        }
    }
    vector<char> isPrime(R - L + 1, true);
    for (long long i: primes)
        for (long long j = max(i * i, (L + i - 1) / i * i); j
             <= R; j += i)
            isPrime[j - L] = false;
    if (L == 1)
        isPrime[0] = false;
    return isPrime;
}
```

LDE.cpp

Description: Solves a*x + b*y = c where c is divisible by gcd(a,b)

6ac25b, 107 lines

```
int gcd(int a, int b, int &x, int &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}
bool find_any_solution(int a, int b, int c, int &x0, int &y0,
     int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }
    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
void shift_solution(int &x, int &y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}
int find_all_solutions(int a, int b, int c, int minx, int maxx,
      int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;
    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;
    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;
    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;
    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;
    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;
    if (lx2 > rx2)
        swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);
    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
/*
aX + bY = g
aXt + bYt = c = gt
t = c / g
x *= t, y *= t
xUnit = b / g, yUnit = a / g;
*/
// if you want to use with Y pass: (y, x, yUnit, xUnit, bar,
    orEqual)
void raiseXOverBar(ll &x, ll &y, ll &xUnit, ll &yUnit, ll bar,
     bool orEqual) {
    if (x > bar or (x == bar and orEqual))
        return;
    ll shift = (bar - x + xUnit - orEqual) / xUnit;
    x += shift * xUnit;
    y -= shift * yUnit;
}
void lowerXUnderBar(ll &x, ll &y, ll &xUnit, ll &yUnit, ll bar,
      bool orEqual) {
    if (x < bar or (x == bar and orEqual))
        return;
    ll shift = (x - bar + xUnit - orEqual) / xUnit;
    x -= shift * xUnit;
    y += shift * yUnit;
}
void minXOverBar(ll &x, ll &y, ll &xUnit, ll &yUnit, ll bar,
     bool orEqual) {
    if (x < bar or (x == bar and !orEqual)) {
        ll shift = (bar - x + xUnit - orEqual) / xUnit;
        x += shift * xUnit;
        y -= shift * yUnit;
    } else {
        ll shift = (x - bar - !orEqual) / xUnit;
        x -= shift * xUnit;
        y += shift * yUnit;
    }
}
void maxXUnderBar(ll &x, ll &y, ll &xUnit, ll &yUnit, ll bar,
     bool orEqual) {
    if (x < bar or (x == bar and orEqual)) {
        ll shift = (bar - x - !orEqual) / xUnit;
        x += shift * xUnit;
        y -= shift * yUnit;
    } else {
        ll shift = (x - bar + xUnit - orEqual) / xUnit;
        x -= shift * xUnit;
        y += shift * yUnit;
    }
}
```

CongruenceEquation.cpp

Description: finds minimum x for which ax = b (mod m)

e0e6eb, 31 lines

```
ll extended_euclid(ll a, ll b, ll &x, ll &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    ll x1, y1;
    ll d = extended_euclid(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}
ll inverse(ll a, ll m) {
    ll x, y;
    ll g = extended_euclid(a, m, x, y);
    if (g != 1) return -1;
    return (x % m + m) % m;
}
// ax = b (mod m)
vector<ll> congruence_equation(ll a, ll b, ll m) {
    vector<ll> ret;
    ll g = gcd(a, m), x;
    if (b % g != 0) return ret;
    a /= g, b /= g;
    x = inverse(a, m / g) * b;
    for (int k = 0; k < g; ++k) { // exactly g solutions
        ret.push_back((x + m / g * k) % m);
    }
    // minimum solution = (m / g - (m - x) % (m / g)) %(m / g)
    return ret;
}
```

CRT.cpp

Description: calculate each two congruences then solve with next: sol(sol(sol(1, 2), 3), 4) T = x mod N -> T = N * k + x T = y mod M -> T = M * p + y N * k + x = M * p + y -> N * k - M * p = y - x (LDE) requires writing of extended euclidian

ad7da3, 14 lines

```
ll CRT(vector<ll> &rems, vector<ll> &mods) {
    ll prevRem = rems[0], prevMod = mods[0];
    for (int i = 1; i < rems.size(); i++) {
        ll x, y, c = rems[i] - prevRem;
        if (c % __gcd(prevMod, -mods[i]))
            return -1;
        ll g = eGCD(prevMod, -mods[i], x, y);
        x *= c / g;
        prevRem += prevMod * x;
        prevMod = prevMod / g * mods[i];
        prevRem = ((prevRem % prevMod) + prevMod) % prevMod;
    }
    return prevRem;
}
```

mobius.cpp

Description: Mobius

c67f60, 22 lines

```
const int N = 1e7;
vi prime;
bool isComp[N];
int mob[N];
void sieve(int n = N) {
    fill(isComp, isComp + n, false);
    mob[1] = 1;
    for (int i = 2; i < n; ++i) {
        if (!isComp[i]) {
            prime.push_back(i);
            mob[i] = -1;
        }
        for (int j = 0; j < prime.size() && i * prime[j] < n;
         ++j) {
            isComp[i * prime[j]] = true;
            if (i % prime[j] == 0) {
                mob[i * prime[j]] = 0;
                break;
            } else
                mob[i * prime[j]] = mob[i] * mob[prime[j]];
        }
    }
}
```

PrimitiveRoot.cpp

Description: Ord(x) is the least positive number such that $x^ord(x) = 1$ Number of x with Ord(x) = y is Phi(y) all possible Ord(x) divide Phi(n) Ord(a^k) = Ord(a) / gcd(k,Ord(a))

c6d472, 28 lines

```
int powmod(int a, int b, int p) {
    int res = 1;
    while (b)
        if (b & 1)
            res = int(res * 1ll * a % p), --b;
        else
            a = int(a * 1ll * a % p), b >>= 1;
    return res;
}
int generator(int p) {
    vector<int> fact;
    int phi = p - 1, n = phi;
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0) {
            fact.push_back(i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
        fact.push_back(n);
    for (int res = 2; res <= p; ++res) {
        bool ok = true;
        for (size_t i = 0; i < fact.size() && ok; ++i)
            ok &= powmod(res, phi / fact[i], p) != 1;
        if (ok) return res;
    }
    return -1;
}
```

longDivision.cpp

Description: long division

63d222, 14 lines

```
string longDivision(string num, ll divisor) {
    string ans;
    ll idx = 0;
    ll temp = num[idx] - '0';
    while (temp < divisor)
        temp = temp * 10 + (num[++idx] - '0');
    while (num.size() > idx) {
        ans += (temp / divisor) + '0';
        temp = (temp % divisor) * 10 + num[++idx] - '0';
    }
    if (ans.length() == 0)
        return "0";
    return ans;
}
```

FloorValues.cpp

Description: code to get all differnet values of floor(n/i)

5305c7, 4 lines

```
for (ll l = 1, r = 1; (n / l); l = r + 1) {
    r = (n / (n / l));
    // q = (n/l), process the range [l, r]
}
```

DiscreteLogarithm.cpp

Description: Returns minimum x for which $a^x/modm = b/modm$ using the babystep giantstep algorithm in sqrt(m) log(m)

dcf2d0, 29 lines

```
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int k = 1, add = 0, g;
    while ((g = gcd(a, m)) > 1) {
        if (b == k)
            return add;
        if (b % g)
            return -1;
        b /= g, m /= g, ++add;
        k = (k * 1ll * a / g) % m;
    }
    int n = sqrt(m) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;
    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }
    for (int p = 1, cur = k; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur] + add;
            return ans;
        }
    }
    return -1;
}
```

4.2 Modular arithmetic

ModularArithmetic.h

Description: Operators for modular arithmetic. You need to set mod to some number first and then you can use the structure.

"euclid.h" 35bfea, 18 lines

```
const ll mod = 17; // change to something else
struct Mod {
  ll x;
  Mod(ll xx) : x(xx) {}
  Mod operator+(Mod b) { return Mod((x + b.x) % mod); }
  Mod operator-(Mod b) { return Mod((x - b.x + mod) % mod); }
  Mod operator*(Mod b) { return Mod((x * b.x) % mod); }
  Mod operator/(Mod b) { return *this * invert(b); }
  Mod invert(Mod a) {
    ll x, y, g = euclid(a.x, mod, x, y);
    assert(g == 1); return Mod((x + mod) % mod);
  }
  Mod operator^(ll e) {
    if (!e) return Mod(1);
    Mod r = *this ^ (e / 2); r = r * r;
    return e&1 ? *this * r : r;
  }
};
```

ModInverse.h

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

6f684f, 3 lines

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

ModPow.h

b83e45, 8 lines

```
const ll mod = 1000000007; // faster if const

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

ModLog.h

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

Time: $\mathcal{O}(\sqrt{m})$

c040b8, 11 lines

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

ModSum.h

Description: Sums of mod'ed arithmetic progressions.

`modsum(to, c, k, m)` $= \sum_{i=0}^{\mathrm{to}-1} (ki + c)\%m$. `divsum` is similar but for floored division.

Time: $\log(m)$, with a large constant.

5c5bc5, 16 lines

```
typedef unsigned long long ull;
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;
  if (!k) return res;
  ull to2 = (to * k + c) / m;
  return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
}

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);
}
```

ModMulLL.h

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

bbbd8f, 11 lines

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

ModSqrt.h

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

"ModPow.h" 19a793, 24 lines

```
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  assert(modpow(a, (p-1)/2, p) == 1); // else no solution
  if (p % 4 == 3) return modpow(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  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;
  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)
      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;
    b = b * g % p;
  }
}
```

4.3 Primality

FastEratosthenes.h

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

6b2912, 20 lines

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

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 $a^b \mod c$.

"ModMulLL.h" 60dcd1, 12 lines

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

Factor.h

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

"ModMulLL.h", "MillerRabin.h" d8d98d, 18 lines

```
ull pollard(ull n) {
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  auto f = [&](ull x) { return modmul(x, x, 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;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}
vector<ull> factor(ull n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), all(r));
  return l;
}
```

4.4 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) {
  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 $0 \le x < \operatorname{lcm}(m, n)$. Assumes $mn < 2^{62}$.
Time: $\log(n)$

"euclid.h" 04d93a, 7 lines

```
ll crt(ll a, ll m, ll b, ll n) {
  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;
}
```

4.4.1 Bézout's identity

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

$$ax + by = d$$

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

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

phiFunction.h

Description: *Euler's* ϕ function is defined as $\phi(n) :=$ # of positive integers $\le n$ 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 $n = p_1^{k_1} p_2^{k_2} \ldots p_r^{k_r}$ then $\phi(n) = (p_1 - 1)p_1^{k_1 - 1} \ldots (p_r - 1)p_r^{k_r - 1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$.
$\sum_{d|n} \phi(d) = n$, $\sum_{1 \le k \le n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$
Euler's thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod n$.
Fermat's little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod p$ $\forall a$.

cf7d6d, 8 lines

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

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

4.5 Fractions

ContinuedFractions.h

Description: Given N and a real number $x \ge 0$, finds the closest rational approximation p/q with $p, q \le N$. It will obey $|p/q - x| \le 1/qN$.
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 ∞; if x is the root of a degree 2 polynomial the a's eventually become cyclic.
Time: $\mathcal{O}(\log N)$

dd6c5e, 21 lines

```
typedef double d; // for N ~ 1e7; long double for N ~ 1e9
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),
       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.
      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)) > 3*N) {
      return {NP, NQ};
    }
    LP = P; P = NP;
    LQ = Q; Q = NQ;
  }
}
```

FracBinarySearch.h

Description: Given f and N, finds the smallest fraction $p/q \in [0,1]$ such that $f(p/q)$ is true, and $p, q \le N$. 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; };

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

4.6 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

4.7 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 $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

4.8 Estimates

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

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

4.9 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:

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

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

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

Combinatorial (5)

5.1 Permutations

5.1.1 Factorial

n	1 2 3 4 5 6 7 8 9 10
$n!$	1 2 6 24 120 720 5040 40320 362880 3628800

n	11	12	13	14	15	16	17	
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14	
n	20	25	30	40	50	100	150	171
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX

5.1.2 Cycles

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

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

5.1.3 Derangements

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

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

5.1.4 Burnside's lemma

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

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

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

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

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

5.2 Partitions and subsets

5.2.1 Partition function

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

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

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

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	~2e5	~2e8

5.2.2 Lucas' Theorem

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

5.2.3 Binomials

multinomial.h

Description: Computes $\binom{k_1+\cdots+k_n}{k_1, k_2, \ldots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \ldots k_n!}$. a0a312, 5 lines

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

nCr.cpp

Description: Computes bionmial coefficients $\binom{n}{r} = \frac{n!}{(n-r)!(r)!}$ for all n and $r \le N$ in O(1) after O(N) preprocessing aba77e, 36 lines

```
const int N = 1e5 + 5;
const int MOD = 1e9 + 7;
```

```
ll fact[N], modInv[N];

ll fastExp(ll x, ll n) {
    if (n == 0)
        return 1;
    ll u = fastExp(x, n / 2);
    u = u * u % MOD;
    if (n & 1)
        u = u * x % MOD;
    return u;
}

// modInv[i] = fact[i]^-1 % MOD
void preprocess() {
    fact[0] = 1;
    for (ll i = 1; i < N; i++)
        fact[i] = fact[i - 1] * i % MOD;

    modInv[N - 1] = fastExp(fact[N - 1], MOD - 2) % MOD;
    for (ll i = N - 2; i >= 0; i--)
        modInv[i] = (i + 1) * modInv[i + 1] % MOD;
}

ll modInvF(ll x) {
    return fastExp(x, MOD - 2);
}

ll nCr(int n, int r) {
    if (r > n)
        return 0;

    // return ( n! / ((n-r)! * r!) ) % MOD
    return (fact[n] * modInv[n - r] % MOD) * modInv[r] % MOD;
}
```

nCrRecursive.cpp

Description: Computes bionmial coefficients for all n and $r \leq N$ in O(1) after O(N^2) preprocessing

d044aa, 13 lines

```
ll dp[N][N];
ll nCr(int n, int r) {
    if (r > n)
        return 0;

    ll &ret = dp[n][r];
    if (~ret)
        return ret;
    if (r == 0)return ret = 1;
    if (r == 1)return ret = n;
    if (n == 1)return ret = 1;
    return ret = nCr(n - 1, r - 1) + nCr(n - 1, r);
}
```

5.3 General purpose numbers

5.3.1 Bernoulli numbers

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

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:

$$\sum_{i=m}^{\infty} f(i) = \int_m^\infty f(x)dx - \sum_{k=1}^{\infty}\frac{B_k}{k!}f^{(k-1)}(m)$$

$$\approx \int_m^\infty f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

5.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

5.3.3 Eulerian numbers

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

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

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

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

5.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

5.3.5 Bell numbers

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

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

5.3.6 Labeled unrooted trees

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

5.3.7 Catalan numbers

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

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

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

- 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 (6)

6.1 Fundamentals

BellmanFord.h

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

830a8f, 23 lines

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

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

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

FloydWarshall.h

Description: Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix m, where $m[i][j]$ = `inf` if i and j are not adjacent. As output, $m[i][j]$ is set to the shortest distance between i and j, `inf` if no path, or `-inf` if the path goes through a negative-weight cycle.
Time: $\mathcal{O}(N^3)$

531245, 12 lines

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

Dijkstra.cpp

Description: Dijkstra

5cc452, 25 lines

```
const ll OO = 1e18;
const int N = 1e5 + 5;
vector<pair<int, ll>> adj[N];
ll dist[N];
int n, m;
void dijkstra(int src) {
    for (int i = 1; i <= n; i++)
        dist[i] = OO;
    priority_queue<pair<ll, int>, vector<pair<ll, int>>,
         greater<pair<ll,int>>> pq;
    dist[src] = 0;
    pq.push({0, src});
    while(!pq.empty()){
        int u;ll w;
        tie(w, u) = pq.top();
        pq.pop();
        if(dist[u] < w)
            continue;
        for(auto e:adj[u]){
            if(dist[u] + e.S < dist[e.F]){
                dist[e.F] = dist[u] + e.S;
                pq.push({dist[e.F], e.F});
            }
        }
    }
}
```

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

d678d8, 8 lines

```
vi topoSort(const vector<vi>& gr) {
  vi indeg(sz(gr)), q;
  for (auto& li : gr) for (int x : li) indeg[x]++;
  rep(i,0,sz(gr)) if (indeg[i] == 0) q.push_back(i);
  rep(j,0,sz(q)) for (int x : gr[q[j]])
    if (--indeg[x] == 0) q.push_back(x);
  return q;
}
```

6.2 Network flow

Dinic.h

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

d7f0f1, 42 lines

```
struct Dinic {
  struct Edge {
    int to, rev;
    ll c, oc;
    ll flow() { return max(oc - c, 0LL); } // if you need flows
  };
  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});
    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++) {
      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;
        }
    }
    return 0;
  }
  ll calc(int s, int t) {
    ll flow = 0; q[0] = s;
    rep(L,0,31) 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]) {
        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;
      }
      while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
    } while (lvl[t]);
    return flow;
  }
  bool leftOfMinCut(int a) { return lvl[a] != 0; }
};
```

MCMF.cpp

Description: MCMF

aba390, 69 lines

```
struct Edge {
    int to;
    int cost;
    int cap, flow, backEdge;
};
struct MCMF {
    const int inf = 1000000010;
    int n;
    vector<vector<Edge>> g;
    MCMF(int _n) {
        n = _n + 1;
        g.resize(n);
    }
    void addEdge(int u, int v, int cap, int cost) {
        Edge e1 = {v, cost, cap, 0, (int) g[v].size()};
        Edge e2 = {u, -cost, 0, 0, (int) g[u].size()};
        g[u].push_back(e1);
        g[v].push_back(e2);
    }
    pair<int, int> minCostMaxFlow(int s, int t) {
        int flow = 0;
        int cost = 0;
        vector<int> state(n), from(n), from_edge(n);
        vector<int> d(n);
        deque<int> q;
        while (true) {
            for (int i = 0; i < n; i++)
                state[i] = 2, d[i] = inf, from[i] = -1;
            state[s] = 1;
            q.clear();
            q.push_back(s);
            d[s] = 0;
            while (!q.empty()) {
                int v = q.front();
                q.pop_front();
                state[v] = 0;
                for (int i = 0; i < (int) g[v].size();i++) {
                    Edge e = g[v][i];
                    if (e.flow >= e.cap || (d[e.to] <=d[v] + e.
                        cost))
                        continue;
                    int to = e.to;
                    d[to] = d[v] + e.cost;
                    from[to] = v;
                    from_edge[to] = i;
                    if (state[to] == 1) continue;
                    if (!state[to] || (!q.empty() &&d[q.front()
                        ] > d[to]))
                        q.push_front(to);
                    else q.push_back(to);
                    state[to] = 1;
                }
            }
            if (d[t] == inf) break;
            int it = t, addflow = inf;
            while (it != s) {
                addflow = min(addflow,g[from[it]][from_edge[it
                    ]].cap-g[from[it]][from_edge[it]].flow);
                it = from[it];
            }
            it = t;
            while (it != s) {
                g[from[it]][from_edge[it]].flow +=addflow;
                g[it][g[from[it]][from_edge[it]].backEdge].flow
                    -=addflow;
                cost += g[from[it]][from_edge[it]].cost*
                    addflow;
                it = from[it];
            }
            flow += addflow;
        }
        return {cost, flow};
    }
};
```

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.

6.3 Matching

Kuhn.cpp

Description: maximum bipartite matching in $O(n \times m)$

735968, 50 lines

```
struct Kuhn {
    int n, m;
    vector<int> leftMatch,rightMatch;
    vector<bool> vis;
    vector<vector<int>> g;
    Kuhn(int n = 101, int m = 101) : n(n), m(m) {
        vis.resize(n);
        g.resize(n + 1);
        leftMatch.assign(m,-1);
        rightMatch.assign(n,-1);
    }
    void addEdge(int u, int v) {
        g[u].push_back(v);
    }
    bool match(int u) {
        if (vis[u])
            return false;
        vis[u] = true;
        for (auto v: g[u]) {
            if (leftMatch[v] == -1 || match(leftMatch[v])) {
                leftMatch[v] = u;
                rightMatch[u] = v;
                return true;
            }
```

```
        }
        return false;
    }
    int maxMatch() {
        vector<bool> used(n);
        for (int i = 0; i < n; ++i) {
            for (auto v: g[i]) {
                if (leftMatch[v] == -1) {
                    used[i] = true;
                    rightMatch[i] = v;
                    leftMatch[v] = i;
                    break;
                }
            }
        }
        for (int i = 0; i < n; i++) {
            if (used[i])continue;
            fill(vis.begin(), vis.end(), 0);
            match(i);
        }
        int sol = 0;
        for (int i = 0; i < m; i++)
            sol += leftMatch[i] != -1;
        return sol;
    }
};
```

HopcroftKarp.cpp

Description: Gets maximum bipartite matching

fb591e, 57 lines

```
struct HopcroftKarp {
    vector<int> leftMatch, rightMatch, dist, cur;
    vector<vector<int> > a;
    int n, m;
    HopcroftKarp() {}
    HopcroftKarp(int n, int m) {
        this->n = n;
        this->m = m;
        a = vector<vector<int> >(n);
        leftMatch = vector<int>(m, -1);
        rightMatch = vector<int>(n, -1);
        dist = vector<int>(n, -1);
        cur = vector<int>(n, -1);
    }
    void addEdge(int x, int y) {
        a[x].push_back(y);
    }
    int bfs() {
        int found = 0;
        queue<int> q;
        for (int i = 0; i < n; i++)
            if (rightMatch[i] < 0)
                dist[i] = 0,q.push(i);
            else dist[i] = -1;
        while (!q.empty()) {
            int x = q.front();
            q.pop();
            for (int i = 0; i < int(a[x].size()); i++) {
                int y = a[x][i];
                if (leftMatch[y] < 0) found = 1;
                else if (dist[leftMatch[y]] < 0)
                    dist[leftMatch[y]] = dist[x] + 1,q.push(
                        leftMatch[y]);
            }
        }
        return found;
    }
    int dfs(int x) {
        for (; cur[x] < int(a[x].size()); cur[x]++) {
            int y = a[x][cur[x]];
            if (leftMatch[y] < 0 || (dist[leftMatch[y]]== dist[
                x] + 1 && dfs(leftMatch[y]))) {
                leftMatch[y] = x;
                rightMatch[x] = y;
                return 1;
            }
        }
        return 0;
    }
    int maxMatching() {
        int match = 0;
        while (bfs()) {
            for (int i = 0; i < n; i++) cur[i] = 0;
            for (int i = 0; i < n; i++)
                if (rightMatch[i] < 0) match += dfs(i);
        }
        return match;
    }
};
```

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: $\mathcal{O}(VE)$

522b98, 22 lines

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

MinimumVertexCover.h

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

"DFSMatching.h" da4196, 20 lines

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

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 $N \le M$.

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

1e0fe9, 31 lines

```
pair<int, vi> hungarian(const vector<vi> &a) {
  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;
    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;
      int i0 = p[j0], j1, 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;
      }
      rep(j,0,m) {
        if (done[j]) u[p[j]] += delta, v[j] -= delta;
        else dist[j] -= delta;
      }
      j0 = j1;
    } while (p[j0]);
    while (j0) { // update alternating path
      int j1 = pre[j0];
      p[j0] = p[j1], j0 = j1;
    }
  }
  rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
  return {-v[0], ans}; // min cost
}
```

6.4 DFS algorithms

Tarjan.cpp

Description: Finds all bridges and cutpoints in a graph in O($n+m$)

bc53a3, 38 lines

```
int n; // number of nodes
vector<int> adj[N]; // adjacency list of graph
vector<bool> visited;
vector<int> tin, low;
int timer;
void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children = 0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
```

```
            if (low[to] > tin[v]){
                //IS_BRIDGE(v, to);
            }
            if (low[to] >= tin[v] && p!=-1){
                //IS_CUTPOINT(v);
            }
            ++children;
        }
    }
//    if(p == -1 && children > 1)
//        IS_CUTPOINT(v);
}
void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

Kosaraju.cpp

Description: Finds all strongly connected components in $O(n + m)$, may have a high constant factor, zero-based

71ff42, 56 lines

```
using vb = vector<bool>;
// assuming nodes are zero based
struct SCC {
    vvi adj, adjRev, comps;
    vpi edges;
    vi revOut, compOf;
    vb vis;
    int N;
    void init(int n) {
        N = n;
        adj.resize(n);
        adjRev.resize(n);
        vis.resize(n);
        compOf.resize(n);
    }
    void addEdge(int u, int v) {
        edges.pb(make_pair(u, v));
        adj[u].pb(v);
        adjRev[v].pb(u);
    }
    void dfs1(int u) {
        vis[u] = true;
        for (auto v:adj[u])
            if (!vis[v])
                dfs1(v);
        revOut.pb(u);
    }
    void dfs2(int u) {
        vis[u] = true;
        comps.back().pb(u);
        compOf[u] = comps.size() - 1;
        for (auto v:adjRev[u])
            if (!vis[v])dfs2(v);
    }
    void gen() {
        fill(all(vis), false);
        for (int i = 0; i < N; ++i) {
            if (!vis[i])
                dfs1(i);
        }
        reverse(all(revOut));
        fill(all(vis), false);
        for (auto node:revOut) {
            if (vis[node])continue;
            comps.pb(vi());
            dfs2(node);
        }
    }
    vvi generateCondensedGraph() {
        vvi adjCon(comps.size());
        for (auto edge:edges)
            if (compOf[edge.F] != compOf[edge.S])
                adjCon[compOf[edge.F]].pb(compOf[edge.S]);
        return adjCon;
    }
};
```

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

c6b7c7, 32 lines

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

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

TwoSatSol.cpp

Description: 2 Sat using Kosaraju's Algorithm for SCC, does generate the solution

4b9204, 99 lines

```
const int N = 2e5 + 5;
vector<int> adj[N], adjR[N], revOut;
int compOf[N], sz, comp;
bool vis[N];
void dfs1(int u) {
    vis[u] = true;
    for (auto v: adj[u])
        if (!vis[v])
            dfs1(v);
    revOut.push_back(u);
}
void dfs2(int u) {
    vis[u] = true;
    compOf[u] = comp;
    for (auto v: adjR[u])
        if (!vis[v])dfs2(v);
}
void initSCC(int n) {
    sz = n;
    revOut.clear();
    comp = 0;
    for (int i = 0; i < sz; i++) {
        adj[i].clear();
        adjR[i].clear();
        vis[i] = 0;
    }
}
void gen() {
    for (int i = 0; i < sz; ++i) {
        if (!vis[i])
            dfs1(i);
    }
    reverse(all(revOut));
    for (int i = 0; i < sz; i++)
        vis[i] = false;
    for (auto node: revOut) {
        if (vis[node])continue;
        comp++;
        dfs2(node);
    }
}
struct TwoSat {
    int N;
    TwoSat(int n) {
        N = n;
        initSCC(2 * N);
    }
    int addVar() { // only if you will use in atMostOne
        adj[2 * N].clear();
        adj[2 * N + 1].clear();
        adjR[2 * N].clear();
        adjR[2 * N + 1].clear();
        vis[2 * N] = vis[2 * N + 1] = 0;
        sz += 2;
        return N++;
    }
    // x or y, edges will be refined in the end
    void either(int x, int y) {
        x = max(2 * x, -1 - 2 * x);
        y = max(2 * y, -1 - 2 * y);
        adj[x ^ 1].push_back(y);
        adj[y ^ 1].push_back(x);
        adjR[y].push_back(x ^ 1);
        adjR[x].push_back(y ^ 1);
    }
    void implies(int x, int y) {
        either(~x, y);
    }
    void must(int x) {
        x = max(2 * x, -1 - 2 * x);
        adj[x ^ 1].push_back(x);
        adjR[x].push_back(x ^ 1);
    }
```

```
    void XOR(int x, int y) {
        either(x, y);
        either(~x, ~y);
    }
    void atMostOne(const vector<int> &li) {
        if (li.size() <= 1) return;
        int last = ~li[1];
        for (int i = 2; i < li.size(); i++) {
            int next = addVar();
            implies(li[i], last);
            either(last, next);
            implies(li[i], next);
            last = ~next;
        }
        implies(li[0], last);
    }
    vector<bool> solve() {
        gen();
        for (int i = 0; i < 2 * N; ++i)
            if (compOf[i] == compOf[i ^ 1])return {};
        vector<bool> ans(N);
        for (int i = 0; i < 2 * N; i += 2)
            ans[i / 2] = compOf[i] > compOf[i + 1];
        return ans;
    }
};
```

TwoSatNoSol.cpp

Description: 2 Sat using Tarjan's Algorithm for SCC, does not generate the solution

1ff324, 84 lines

```
const ll inf = 1e18;
vector<int> adj[N];
int low[N], scc[N], comps, timer;
stack<int> st;
bool sat;
void dfs(int u) {
    low[u] = ++timer;
    st.push(u);
    int cur = low[u];
    for (int v: adj[u]) {
        if (!low[v]) dfs(v);
        low[u] = min(low[u], low[v]);
    }
    if (low[u] == cur) {
        comps++;
        while (1) {
            int v = st.top();
            st.pop();
            scc[v] = comps;
            low[v] = inf;
            if (scc[v] == scc[v ^ 1])
                sat = false;
            if (u == v) break;
        }
    }
}
void initSCC(int n) {
    for (int i = 0; i < n; i++) {
        adj[i].clear();
        scc[i] = 0, low[i] = 0;
    }
    comps = 0, timer = 0;
    sat = true;
    while (!st.empty())st.pop();
}
struct TwoSat {
    int N;
    TwoSat(int n) {
        N = n;
        initSCC(2 * N);
    }
    int addVar() { // only if you will use in atMostOne
        adj[2 * N].clear();
        adj[2 * N + 1].clear();
        scc[2 * N] = low[2 * N + 1] = 0;
        return N++;
    }
    // x or y, edges will be refined in the end
    void either(int x, int y) {
        x = max(2 * x, -1 - 2 * x);
        y = max(2 * y, -1 - 2 * y);
        adj[x ^ 1].push_back(y);
        adj[y ^ 1].push_back(x);
    }
    void implies(int x, int y) {
        either(~x, y);
    }
    void must(int x) {
        x = max(2 * x, -1 - 2 * x);
        adj[x ^ 1].push_back(x);
    }
    void XOR(int x, int y) {
        either(x, y);
        either(~x, ~y);
    }
    void atMostOne(const vector<int> &li) {
        if (li.size() <= 1) return;
        int last = ~li[1];
        for (int i = 2; i < li.size(); i++) {
            int next = addVar();
            implies(li[i], last);
            either(last, next);
            implies(li[i], next);
            last = ~next;
        }
        implies(li[0], last);
    }
    bool solve() {
        for (int i = 0; i < 2 * N; i++)
            if (!scc[i])
                dfs(i);
        return sat;
    }
};
```

6.5 Heuristics

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 MinimumVertexCover.

6.6 Math

6.6.1 Number of Spanning Trees

Create an $N \times N$ matrix `mat`, and for each edge $a \to b \in G$, do `mat[a][b]--`, `mat[b][b]++` (and `mat[b][a]--`, `mat[a][a]++` if G is undirected). Remove the ith 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).

6.6.2 Erdős–Gallai theorem

A simple graph with node degrees $d_1 \ge \cdots \ge d_n$ exists iff $d_1 + \cdots + d_n$ is even and for every $k = 1 \ldots n$,

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

Trees (7)

7.1 Fundamentals

LCASimple.cpp

Description: shorter nicer version of LCA imo

89c1f3, 40 lines

```
const int N = 2e5 + 5;
const int LG = 20;

int anc[N][20], p[N], d[N], n, q;
vi adj[N];

void dfs(int u, int par, int dep) {
    p[u] = par;
    d[u] = dep;
    for (int e: adj[u])
        if (e != par)
            dfs(e, u, dep + 1);
}

void pre() {
    for (int k = 0; k < LG; ++k) {
        for (int u = 1; u <= n; ++u) {
            if (k == 0) anc[u][k] = p[u];
            else anc[u][k] = anc[anc[u][k - 1]][k - 1];
        }
    }
}

int binLift(int u, int x) {
    for (int b = 0; b < LG; ++b)
        if ((1 << b) & x) u = anc[u][b];
    return u;
}

int LCA(int u, int v) {
    if (d[u] < d[v])swap(u, v);
    u = binLift(u, d[u] - d[v]);
    if (u == v)return u;
    for (int b = LG-1; b >= 0; --b) {
        if (anc[u][b] == anc[v][b])continue;
        u = anc[u][b];
        v = anc[v][b];
    }
    return anc[u][0];
}
```

LCA.cpp

Description: LCA and binary lifting

bee572, 53 lines

```
const int N = 2e5 + 5;
const int LOG = 19;
vector<int> adj[N];
int depth[N], up[N][LOG], n, timer, tin[N], tout[N];
void dfs(int u, int p) {
    tin[u] = timer++;
    for (auto v: adj[u]) {
```

```cpp
        if (v == p)continue;
        depth[v] = depth[u] + 1;
        up[v][0] = u;
        dfs(v, u);
    }
    tout[u] = timer - 1;
}
bool isAncestor(int u, int v) {
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}
int LCA(int u, int v) {
    if (depth[u] < depth[v])
        swap(u, v);
    int k = depth[u] - depth[v];
    for (int i = 0; i < LOG; ++i) {
        if ((1 << i) & k) {
            u = up[u][i];
        }
    }
    if (u == v)
        return u;
    for (int i = LOG - 1; i >= 0; --i) {
        if (up[u][i] != up[v][i]) {
            u = up[u][i];
            v = up[v][i];
        }
    }
    return up[u][0];
}
int Kthancestor(int u,int k){
    if(k > depth[u])return 0;
    for (int j = LOG - 1; j >= 0; --j) {
        if(k&(1<<j)){
            u = up[u][j];
        }
    }
    return u;
}
void build() {
    dfs(0, 0);
    for (int j = 1; j < LOG; ++j) {
        for (int i = 0; i < n; ++i) {
            up[i][j] = up[up[i][j - 1]][j - 1];
        }
    }
}
```

Sack.cpp

Description: Small to large on trees with global data structure

1b176d, 36 lines

```cpp
vector<int> adj[N];
int n, sz[N], big[N];
void dfsSz(int u, int par) {
    sz[u] = 1;
    for (auto &v: adj[u]) {
        if (v == par)continue;
        dfsSz(v, u);
        sz[u] += sz[v];
        if (big[u] == -1 || sz[v] > sz[big[u]])
            big[u] = v;
    }
}
void collect(int u, int par) {
    // add(u)
    for (auto v: adj[u]) {
        if (v == par)continue;
        collect(v, u);
    }
}
void dfs(int u, int par, bool keep) {
    for (auto v: adj[u]) {
        if (v == par || v == big[u])continue;
        dfs(v, u, false);
    }
    if (~big[u]) {
        dfs(big[u], u, true);
    }
    // add(u)
    for (auto v: adj[u]) {
        if (v == par || v == big[u])continue;
        collect(v, u);
    }
    if (!keep) {
        // reset(all)
    }
}
```

CentroidDecomp.cpp

Description: Centroid Decomposition

1ec98f, 62 lines

```cpp
const int N = 2e5;
const int OO = 1e9 + 5;int sz[N], n, k, freq[N];
vi adj[N];
bool rem[N];
void preSize(int i, int par) {
    sz[i] = 1;
    for (auto e: adj[i]) {
        if (e == par || rem[e])
            continue;
        preSize(e, i);
        sz[i] += sz[e];
    }
}
int getCen(int u, int p, int curSz) {
    for (auto v: adj[u]) {
        if (rem[v] || v == p)continue;
        if (sz[v] * 2 > curSz)
            return getCen(v, u, curSz);
    }
    return u;
}
ll solve(int v, int par, int d) {
    ll ans = k >= d ? freq[k - d] : 0;
    for (auto u: adj[v]) {
        if (rem[u] || u == par)
            continue;
        ans += solve(u, v, d + 1);
    }
    return ans;
}
void update(int v, int par, int d, int inc) {
    freq[d] += inc;
    for (auto u: adj[v]) {
        if (rem[u] || u == par)
            continue;
        update(u, v, d + 1, inc);
    }
}
ll getAns(int v) {
    ll ans = 0;
    for (auto u: adj[v]) {
        if (rem[u])
            continue;
        ans += solve(u, v, 1);
        update(u, v, 1, 1);
    }
    return ans;
}
ll decompose(int v) {
    preSize(v, 0);
    int cen = getCen(v, 0, sz[v]);
    freq[0]++;
    ll ans = getAns(cen);
    update(cen, 0, 0, -1);
    rem[cen] = true;
    for (auto u: adj[cen]) {
        if (rem[u])
            continue;
        ans += decompose(u);
    }
    return ans;
}
```

HLD.cpp

Description: HLD

a8bba7, 67 lines

```cpp
class HLD {
public:
    vector<int> par, sz, head, tin, tout, who, depth;
    int dfs1(int u, vector<vector<int>> &adj) {
        for (int &v: adj[u]) {
            if (v == par[u])continue;
            depth[v] = depth[u] + 1;
            par[v] = u;
            sz[u] += dfs1(v, adj);
            if (sz[v] > sz[adj[u][0]] || adj[u][0] == par[u])
                swap(v, adj[u][0]);
        }
        return sz[u];
    }
    void dfs2(int u, int &timer, const
    vector<vector<int>> &adj) {
        tin[u] = timer++;
        for (int v: adj[u]) {
            if (v == par[u])continue;
            head[v] = (timer == tin[u] + 1 ? head[u] : v);
            dfs2(v, timer, adj);
        }
        tout[u] = timer - 1;
    }
    HLD(vector<vector<int>> adj, int r = 0)
            : par(adj.size(), -1), sz(adj.size(), 1),
              head(adj.size(), r), tin(adj.size()), who(adj.
                  size()),
              tout(adj.size()),
              depth(adj.size()){
        dfs1(r, adj);
        int x = 0;
        dfs2(r, x, adj);
        for (int i = 0; i < adj.size(); ++i)
            who[tin[i]] = i;
    }
    vector<pair<int, int>> path(int u, int v) {
        vector<pair<int, int>> res;
        for (;; v = par[head[v]]) {
            if(depth[head[u]] > depth[head[v]])swap(u,v);
            if(head[u] != head[v]){
                res.emplace_back(tin[head[v]], tin[v]);
            }
            else{
                if(depth[u] > depth[v])swap(u,v);
                res.emplace_back(tin[u],tin[v]);
                return res;
            }
        }
    }
    pair<int, int> subtree(int u) {
```

```
        return {tin[u], tout[u]};
    }
    int dist(int u, int v) {
        return depth[u] + depth[v] - 2 * depth[lca(u,v)];
    }
    int lca(int u, int v) {
        for (;; v = par[head[v]]) {
            if(depth[head[u]] > depth[head[v]])swap(u,v);
            if(head[u] == head[v]){
                if(depth[u] > depth[v])swap(u,v);
                return u;
            }
        }
    }
    bool isAncestor(int u, int v) {
        return tin[u] <= tin[v] && tout[u] >= tout[v];
    }
};
```

TreeHashing.cpp

Description: very deterministic tree hashing

45018f, 13 lines

```
const int N = 1e5;
vector<int> adj[N];
map<vector<int>, int> mp;
int dfs(int u, int par) {
    vector<int> cur;
    for (auto v: adj[u]) {
        if (v == par)continue;
        cur.push_back(dfs(v, u));
    }
    sort(all(cur));
    if (!mp.count(cur))mp[cur] = mp.size();
    return mp[cur];
}
```

TreeHashing2.cpp

Description: other tree hashing

7e4bc9, 24 lines

```
const int N = 1e5;
unsigned long long pw(unsigned long long b, unsigned long long
     p) {
    if (!p) return 1ULL;
    unsigned long long ret = pw(b, p >> 1ULL);
    ret *= ret;
    if (p & 1ULL)
        ret = ret * b;
    return ret;
}
int n;
vector<int> adj[N];
unsigned long long dfs(int u, int par) {
    vector<unsigned long long> child;
    for (auto v: adj[u]) {
        if (v == par)continue;
        child.push_back(dfs(v, u));
    }
    sort(all(child));
    unsigned long long ret = 0;
    for (int i = 0; i < child.size(); ++i) {
        ret += child[i] * child[i] + child[i] * pw(31, i + 1) +
              (unsigned long long) 42;
    }
    return ret;
}
```

MoTrees.cpp

Description: MoTrees

a80b08, 98 lines

```
const int B = 350;
const int LG = 19;
struct Query {
    int l, r, ind, lca;
    Query(int _l, int _r, int _ind, int _lca = -1) : l(_l), r(
         _r), ind(_ind), lca(_lca) {}
    bool operator<(const Query &q2) {
        return (l / B < q2.l / B) || (l / B == q2.l / B && r <
             q2.r);
    }
};
struct MoTree {
    vi in, out, flat, dep, freqV;
    vvi anc;
    int n;
    MoTree(vvi &adj, int n, vi &col, int r = 1) : n(n), in(n +
         1), out(n + 1), flat((n + 1) * 2), dep(n + 1),
                                                  freqV(n + 1),
                                                       anc(n +
                                                       1, vi(
                                                     LG)) {
        int x = 0;
        flatten(r, r, x, adj);
        preLCA();
    }
    void flatten(int v, int p, int &timer, const vvi &adj) {
        anc[v][0] = p;
        dep[v] = dep[p] + 1;
        in[v] = timer, flat[timer] = v, ++timer;
        for (auto u: adj[v])
            if (u != p) {
                flatten(u, v, timer, adj);
            }
        out[v] = timer, flat[timer] = v, ++timer;
    }
    void preLCA() {
        for (int k = 1; k < LG; k++)
            for (int i = 1; i <= n; i++)
                anc[i][k] = anc[anc[i][k - 1]][k - 1];
    }
    int binaryLift(int x, int jump) {
        for (int b = 0; b < LG; b++) {
            if (jump & (1 << b))
                x = anc[x][b];
        }
        return x;
    }
    int LCA(int a, int b) {
        if (dep[a] > dep[b])
            swap(a, b);
        int diff = dep[b] - dep[a];
        b = binaryLift(b, diff);
        if (a == b)
            return a;
        for (int bit = LG - 1; bit >= 0; bit--) {
            if (anc[a][bit] == anc[b][bit])
                continue;
            a = anc[a][bit];
            b = anc[b][bit];
        }
        return anc[a][0];
    }
    void upd(int ind, int inc) {
        int v = flat[ind];
        freqV[v] += inc;
        if (freqV[v] == 1) {
            // add()
        } else {
            // remove()
        }
    }
    vi takeQueries(int q) {
        vi ans(q);
        vector<Query> queries;
        int x, y;
        for (int i = 0; i < q; i++) {
            cin >> x >> y;
            if (in[x] > in[y])
                swap(x, y);
            int lca = LCA(x, y);
            if (lca == x)
                queries.emplace_back(in[x], in[y], i);
            else
                queries.emplace_back(out[x], in[y], i, lca);
        }
        sort(all(queries));
        int l = 0, r = 0;
        upd(0, 1);
        for (auto query: queries) {
            while (r < query.r)
                upd(++r, 1);
            while (l > query.l)
                upd(--l, 1);
            while (l < query.l)
                upd(l++, -1);
            while (r > query.r)
                upd(r--, -1);
            if (~query.lca);//addLCA
            //ans[query.ind] = ;
            if (~query.lca);//removeLCA
        }
        return ans;
    }
};
```

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

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

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

  void link(int u, int v) { // add an edge (u, v)
    assert(!connected(u, v));
    makeRoot(&node[u]);
    node[u].pp = &node[v];
  }
  void cut(int u, int v) { // remove an edge (u, v)
    Node *x = &node[u], *top = &node[v];
    makeRoot(top); x->splay();
    assert(top == (x->pp ?: x->c[0]));
    if (x->pp) x->pp = 0;
    else {
      x->c[0] = top->p = 0;
      x->fix();
    }
  }
  bool connected(int u, int v) { // are u, v in the same tree?
    Node* nu = access(&node[u])->first();
    return nu == access(&node[v])->first();
  }
  void makeRoot(Node* u) {
    access(u);
    u->splay();
    if(u->c[0]) {
      u->c[0]->p = 0;
      u->c[0]->flip ^= 1;
      u->c[0]->pp = u;
      u->c[0] = 0;
      u->fix();
    }
  }
  Node* access(Node* u) {
    u->splay();
    while (Node* pp = u->pp) {
      pp->splay(); u->pp = 0;
      if (pp->c[1]) {
        pp->c[1]->p = 0; pp->c[1]->pp = pp; }
      pp->c[1] = u; pp->fix(); u = pp;
    }
    return u;
  }
};
```

Geometry (8)

8.1 Geometric primitives

Point.h

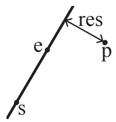
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

47ec0a, 28 lines

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

lineDistance.h

Description:
Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D, call .dist on the result of the cross product.

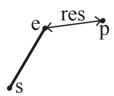


"Point.h" f6bf6b, 4 lines

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

SegmentDistance.h

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



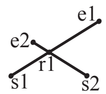
Usage: `Point<double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

"Point.h" 5c88f4, 6 lines

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

SegmentIntersection.h

Description:
If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.



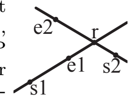
Usage: `vector<P> inter = segInter(s1,e1,s2,e2);`
`if (sz(inter)==1)`
`cout << "segments intersect at " << inter[0] << endl;`

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

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

lineIntersection.h

Description:
If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {1, point} is returned. If no intersection point exists {0, (0,0)} is returned and if infinitely many exists {-1, (0,0)} is returned. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.



Usage: `auto res = lineInter(s1,e1,s2,e2);`
`if (res.first == 1)`
`cout << "intersection point at " << res.second << endl;`

"Point.h" a01f81, 8 lines

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

sideOf.h

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

"Point.h" 3af81c, 9 lines

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

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

OnSegment.h

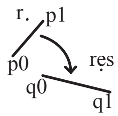
Description: Returns true iff p lies on the line segment from s to e. Use `(segDist(s,e,p)<=epsilon)` instead when using Point<double>.

"Point.h" c597e8, 3 lines

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

linearTransformation.h

Description:



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

"Point.h" 03a306, 6 lines

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

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`

0f0602, 35 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
  int half() const {
    assert(x || y);
    return y < 0 || (y == 0 && x < 0);
  }
  Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
  Angle t180() const { return {-x, -y, t + half()}; }
  Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare distances
  return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
         make_tuple(b.t, b.half(), a.x * (ll)b.y);
}

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

8.2 Circles

CircleIntersection.h

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

"Point.h" 84d6d3, 11 lines

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

CircleTangents.h

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

"Point.h" b0153d, 13 lines

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

CirclePolygonIntersection.h

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

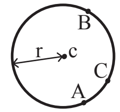
Time: $\mathcal{O}(n)$

"../../content/geometry/Point.h" a1ee63, 19 lines

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

circumcircle.h

Description:



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

MinimumEnclosingCircle.h

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

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

"circumcircle.h" 09dd0a, 17 lines

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

8.3 Polygons

InsidePolygon.h

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

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

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

"Point.h", "OnSegment.h", "SegmentDistance.h" 2bf504, 11 lines

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

PolygonArea.h

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

"Point.h" f12300, 6 lines

```
template<class T>
T polygonArea2(vector<Point<T>>& v) {
```

```
  T a = v.back().cross(v[0]);
  rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
  return a;
}
```

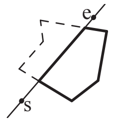
PolygonCenter.h

Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"Point.h" 9706dc, 9 lines

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

PolygonCut.h

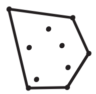


Description:
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.
Usage: `vector<P> p = ...;`
`p = polygonCut(p, P(0,0), P(1,0));`

"Point.h", "lineIntersection.h" f2b7d4, 13 lines

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

ConvexHull.h



Description:
Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.
Time: $\mathcal{O}(n \log n)$

"Point.h" 310954, 13 lines

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

HullDiameter.h

Description: Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).
Time: $\mathcal{O}(n)$

"Point.h" c571b8, 12 lines

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

PointInsideHull.h

Description: Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.
Time: $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "OnSegment.h" 71446b, 14 lines

```
typedef Point<ll> P;

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

LineHullIntersection.h

Description: Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon: • $(-1,-1)$ if no collision, • $(i,-1)$ if touching the corner i, • (i,i) if along side $(i,i+1)$, • (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]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
  int n = sz(poly), lo = 0, hi = n;
  if (extr(0)) return 0;
  while (lo + 1 < hi) {
    int m = (lo + hi) / 2;
    if (extr(m)) return m;
    int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
    (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
  }
  return lo;
}

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

8.4 Misc. Point Set Problems

ClosestPair.h

Description: Finds the closest pair of points.
Time: $\mathcal{O}(n \log n)$

"Point.h" ac41a6, 17 lines

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

kdTree.h

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

"Point.h" bac5b0, 63 lines

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

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

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

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

  Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
      x0 = min(x0, p.x); x1 = max(x1, p.x);
      y0 = min(y0, p.y); y1 = max(y1, p.y);
    }
    if (vp.size() > 1) {
      // split on x if width >= height (not ideal...)
      sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
```

```
    // divide by taking half the array for each child (not
    // best performance with many duplicates in the middle)
    int half = sz(vp)/2;
    first = new Node({vp.begin(), vp.begin() + half});
    second = new Node({vp.begin() + half, vp.end()});
  }
 }
};

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

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

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

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

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

FastDelaunay.h

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

"Point.h" eefdf5, 88 lines

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

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

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
 lll p2 = p.dist2(), A = a.dist2()-p2,
     B = b.dist2()-p2, C = c.dist2()-p2;
 return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}
Q makeEdge(P orig, P dest) {
 Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
 H = r->o; r->r()->r() = r;
 rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
 r->p = orig; r->F() = dest;
 return r;
}
void splice(Q a, Q b) {
 swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
 Q q = makeEdge(a->F(), b->p);
 splice(q, a->next());
 splice(q->r(), b);
 return q;
}

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

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

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

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

8.5 3D

PolyhedronVolume.h

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

3058c3, 6 lines

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

Point3D.h

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

8058ae, 32 lines

```
template<class T> struct Point3D {
 typedef Point3D P;
 typedef const P& R;
 T x, y, z;
 explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
 bool operator<(R p) const {
  return tie(x, y, z) < tie(p.x, p.y, p.z); }
 bool operator==(R p) const {
  return tie(x, y, z) == tie(p.x, p.y, p.z); }
 P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
 P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
 P operator*(T d) const { return P(x*d, y*d, z*d); }
 P operator/(T d) const { return P(x/d, y/d, z/d); }
 T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
 P cross(R p) const {
  return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
 }
 T dist2() const { return x*x + y*y + z*z; }
 double dist() const { return sqrt((double)dist2()); }
 //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
 double phi() const { return atan2(y, x); }
 //Zenith angle (latitude) to the z-axis in interval [0, pi]
 double theta() const { return atan2(sqrt(x*x+y*y),z); }
 P unit() const { return *this/(T)dist(); } //makes dist()=1
 //returns unit vector normal to *this and p
 P normal(P p) const { return cross(p).unit(); }
 //returns point rotated 'angle' radians ccw around axis
 P rotate(double angle, P axis) const {
  double s = sin(angle), c = cos(angle); P u = axis.unit();
  return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
 }
};
```

3dHull.h

Description: Computes all faces of the 3-dimension hull of a point set. *No four points must be coplanar*, or else random results will be returned. All faces will point outwards.
Time: $\mathcal{O}(n^2)$

"Point3D.h" 5b45fc, 49 lines

```
typedef Point3D<double> P3;

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

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

vector<F> hull3d(const vector<P3>& A) {
 assert(sz(A) >= 4);
```

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

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

sphericalDistance.h

Description: Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_2) from z axis (0 = north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. dx*radius is then the difference between the two points in the x direction and d*radius is the total distance between the points.

611f07, 8 lines

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

Strings (9)

KMP.cpp

Description: for every i, calculates the longest proper suffix of the i-th prefix that is also a prefix of the entire array

3c7d30, 27 lines

```
const int N = 1e4;
const int ALPHA = 26;
int aut[N][ALPHA];
void KMP(string &s, vi &fail) {
    int n = (int) s.size();
    for (int i = 1; i < n; i++) {
        int j = fail[i - 1];
        while (j > 0 && s[j] != s[i])
            j = fail[j - 1];
        if (s[j] == s[i])
            ++j;
        fail[i] = j;
    }
}
void constructAut(string &s, vi &fail) {
    int n = s.size();
    // for each fail function value (i is not an index)
    for (int i = 0; i < n; i++) {
    // for each possible transition
        for (int c = 0; c < ALPHA; c++) {
            if (i > 0 && s[i] != 'a' + c)
                aut[i][c] = aut[fail[i - 1]][c];
            else
                aut[i][c] = i + (s[i] == 'a' + c);
        }
    }
}
```

ZFunction.cpp

Description: for every suffix, calculates the longest prefix of that suffix that matches a prefix of the entire string

85d656, 13 lines

```
vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min(r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}
```

Manacher.cpp

Description: Calculates the maximum palindrome centered around every i, for every palindromes, i is the right index of the middle two

81799d, 35 lines

```
vi manacher_odd(string &s) {
    int n = s.size();
    string t = '^' + s + '$';
    vi p(n + 2);
    int l = 1, r = 1;
    for (int i = 1; i <= n; ++i) {
        int &len = p[i];
        int j = l + r - i;
        len = max(0, min(r - i, p[j]));
        while (t[i + len] == t[i - len])
            ++len;
        if (i + len > r) {
            r = i + len;
            l = i - len;
        }
    }
    return vi(p.begin() + 1, p.begin() + n + 1);
}
vector<pi> manacher(string &s) {
    int n = (int) s.size();
    string t;
    for (int i = 0; i < n; ++i) {
        t.pb('#');
        t.pb(s[i]);
    }
    t.pb('#');
    vi p = manacher_odd(t);
    vector<pi> ret(n);
    //odd then even
    for (int i = 0; i < n; ++i) {
        ret[i].F = (p[2 * i + 1]) / 2;
        ret[i].S = (p[2 * i] - 1) / 2;
    }
    return ret;
}
```

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.
Usage: `rotate(v.begin(), v.begin()+minRotation(v), v.end());`
Time: $\mathcal{O}(N)$

d07a42, 8 lines

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

hashing.cpp

Description: Right is the most significant digit

b5f230, 58 lines

```
const int p1 = 31, p2 = 37, MOD = 1e9 + 7;
const int N = 1e6 + 5;
int pw1[N], inv1[N], pw2[N], inv2[N];
ll powmod(ll x, ll y) {
    x %= MOD;
    ll ans = 1;
    while (y) {
        if (y & 1) ans = ans * x % MOD;
        x = x * x % MOD;
        y >>= 1;
    }
    return ans;
}
ll add(ll a, ll b) {
    a += b;
    if (a >= MOD) a -= MOD;
    return a;
}
ll sub(ll a, ll b) {
    a -= b;
    if (a < 0) a += MOD;
    return a;
}
ll mul(ll a, ll b) { return a * b % MOD; }
ll inv(ll a) { return powmod(a, MOD - 2); }
void pre() {
    pw1[0] = inv1[0] = 1;
    pw2[0] = inv2[0] = 1;
    int invV1 = inv(p1);
    int invV2 = inv(p2);
    for (int i = 1; i < N; ++i) {
        pw1[i] = mul(pw1[i - 1], p1);
        inv1[i] = mul(inv1[i - 1], invV1);
        pw2[i] = mul(pw2[i - 1], p2);
        inv2[i] = mul(inv2[i - 1], invV2);
    }
}
struct Hash {
    vector<pi> h;
    int n;
    Hash(string &s) {
```

```
        n = s.size();
        h.resize(n);
        h[0].F = h[0].S = s[0] - 'a' + 1;
        for (int i = 1; i < n; ++i) {
            h[i].F = add(h[i - 1].F, mul((s[i] - 'a' + 1),pw1[i
                ]));
            h[i].S = add(h[i - 1].S, mul((s[i] - 'a' + 1),pw2[i
                ]));
        }
    }
    pi getRange(int l, int r) {
        assert(l <= r);
        assert(r < n);
        return {
                mul(sub(h[r].F, l ? h[l - 1].F : 0),inv1[l]),
                mul(sub(h[r].S, l ? h[l - 1].S : 0),inv2[l])
        };
    }
};
```

hashingRev.cpp

Description: Left is the most significant digit

960638, 54 lines

```
const int p1 = 31, p2 = 37, MOD = 1e9 + 7;
const int N = 1e6 + 5;
int pw1[N], pw2[N];
ll powmod(ll x, ll y) {
    x %= MOD;
    ll ans = 1;
    while (y) {
        if (y & 1) ans = ans * x % MOD;
        x = x * x % MOD;
        y >>= 1;
    }
    return ans;
}
ll add(ll a, ll b) {
    a += b;
    if (a >= MOD) a -= MOD;
    return a;
}
ll sub(ll a, ll b) {
    a -= b;
    if (a < 0) a += MOD;
    return a;
}
ll mul(ll a, ll b) { return a * b % MOD; }
ll inv(ll a) { return powmod(a, MOD - 2); }
void pre() {
    pw1[0] = 1;
    pw2[0] = 1;
    for (int i = 1; i < N; ++i) {
        pw1[i] = mul(pw1[i - 1], p1);
        pw2[i] = mul(pw2[i - 1], p2);
    }
}
struct Hash {
    vector<pi> h;
    int n;
    Hash(string &s) {
        n = s.size();
        h.resize(n);
        h[0].F = h[0].S = s[0] - 'a' + 1;
        for (int i = 1; i < n; ++i) {
            h[i].F = add(mul(h[i - 1].F, p1), s[i] - 'a' + 1);
            h[i].S = add(mul(h[i - 1].S, p2), s[i] - 'a' + 1);
        }
    }
    pi getRange(int l, int r) {
        assert(l <= r);
        assert(r < n);
        return {
                sub(h[r].F, mul(l ? h[l - 1].F : 0, pw1[r - l +
                    1])),
                sub(h[r].S, mul(l ? h[l - 1].S : 0, pw2[r - l +
                    1]))
        };
    }
};
```

Trie.cpp

Description: Trie

af981f, 30 lines

```
const int K = 26;
struct Trie {
    struct Node {
        int go[K];
        int freq;

        Node() {
            fill(go, go + K, -1);
            freq = 0;
        }
    };
    vector<Node> aut;
    Trie(vector<string> &pats) {
        aut.resize(1);
        for (auto &e: pats)
            add_string(e);
    }
    void add_string(string &s) {
        int u = 0; //cur node
        for (auto ch: s) {
            int c = ch - 'a';
            if (aut[u].go[c] == -1) {
                aut[u].go[c] = (int) aut.size();
                aut.emplace_back();
            }
            u = aut[u].go[c];
            aut[u].freq++;
        }
    }
};
```

TrieForNumbers.cpp

Description: Trie for Numbers

8b4212, 47 lines

```
struct Trie {
    vector<vector<int>> trie;
    vector<int> cnt;
    // vector<int>leaves;
    int mxBit, sz;

    int addNode() {
        trie.emplace_back(2, -1);
        cnt.emplace_back();
        // leaves.emplace_back();
        sz++;
        return sz - 1;
    }

    Trie(int mx = 60) : mxBit(mx), sz(0) {
        addNode();
    };

    // insert or remove
    void insert(ll x, int type = 1) {
        int cur = 0;
        cnt[cur] += type;
        for (int i = mxBit; i >= 0; --i) {
            int t = (x >> i) & 1;
            if (trie[cur][t] == -1)
                trie[cur][t] = addNode();
            cur = trie[cur][t];
            cnt[cur] += type;
        }
        // leaves[cur] += type;
    }

    ll maxXor(ll x) {
        // no elements in trie
        int cur = 0;
        if (!cnt[cur])return -1e9;
        for (int i = mxBit; i >= 0; --i) {
            int t = (x >> i) & 1 ^ 1;
            if (trie[cur][t] == -1 ||
                !cnt[trie[cur][t]])
                t ^= 1;
            cur = trie[cur][t];
            if (t)x ^= 1ll << i;
        }
        return x;
    }
};
```

ACA.cpp

Description: ACA

b4a532, 80 lines

```
struct AhoCorasick {
    int states = 0;
    vector<int> pi;
    vector<vector<int>> trie, patterns;

    AhoCorasick(int n, int m = 26) {
        pi = vector<int>(n + 10, -1);
        patterns = vector<vector<int>>(n + 10);
        trie = vector<vector<int>>(n + 10, vector<int>(m, -1));
    }

    AhoCorasick(vector<string> &p, int n, int m = 26) {
        /*
        * MAKE SURE THAT THE STRINGS IN P ARE UNIQUE
        * N is the summation of sizes of p
        * M is the number of used alphabet
        */
        pi = vector<int>(n + 10, -1);
        patterns = vector<vector<int>>(n + 10);
        trie = vector<vector<int>>(n + 10,
                                   vector<int>(m, -1));
        for (int i = 0; i < p.size(); i++)
            insert(p[i], i);
        build();
    }

    void insert(string &s, int idx) {
        int cur = 0;
        for (auto &it: s) {
            if (trie[cur][it - 'a'] == -1)
                trie[cur][it - 'a'] = ++states;
            cur = trie[cur][it - 'a'];
        }
        patterns[cur].push_back(idx);
    }

    int nextState(int trieNode, int nxt) {
        int cur = trieNode;
        while (trie[cur][nxt] == -1)
```

```
            cur = pi[cur];
        return trie[cur][nxt];
    }

    void build() {
        queue<int> q;
        for (int i = 0; i < 26; i++) {
            if (trie[0][i] != -1)
                pi[trie[0][i]] = 0, q.push(trie[0][i]);
            else
                trie[0][i] = 0;
        }
        while (q.size()) {
            int cur = q.front();
            q.pop();
            for (int i = 0; i < 26; i++) {
                if (trie[cur][i] == -1)
                    continue;
                int f = nextState(pi[cur], i);
                pi[trie[cur][i]] = f;
                patterns[trie[cur][i]].insert(patterns[trie[cur
                    ][i]].end(), patterns[f].begin(), patterns
                    [f].end());
                q.push(trie[cur][i]);
            }
        }
    }

    vector<vector<int>> search(string &s, vector<string> &p,
         int n) {
        int cur = 0;
        vector<vector<int>> ret(n);
        for (int i = 0; i < s.length(); i++) {
            cur = nextState(cur, s[i] - 'a');
            if (cur == 0 || patterns[cur].empty())
                continue;
            // patterns vector have every pattern that is
                matched in this node
            // matched: the last index in the pattern is index
                i
            for (auto &it: patterns[cur])
                ret[it].push_back(i - p[it].length() + 1);
        }
        return ret;
    }
};
```

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

bc716b, 22 lines

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

SuffixArray.cpp

Description: Look up Suffix Array in MIT KACTL instead, much shorter, lcp[i] holds the lcp between sa[i], sa[i - 1], sa is the suffix array with the empty suffix being sa[0]

c8fdfb, 59 lines

```
struct SuffixArray {
    string S;
    vector<int> logs, sa, lcp, rank;
    vector<vector<int>> table;
    SuffixArray() {};
    SuffixArray(string &s, int lim = 256) {
        S = s;
        int n = s.size() + 1, k = 0, a, b;
        vector<int> c(s.begin(), s.end() + 1), tmp(n), frq(max(
             n, lim));
        c.back() = 0; //0 is less than any character
        sa = lcp = rank = tmp, iota(sa.begin(), sa.end(), 0);
        for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim =
             p) {
            p = j, iota(tmp.begin(), tmp.end(), n - j);
            for (int i = 0; i < n; i++) {
                if (sa[i] >= j)
                    tmp[p++] = sa[i] - j;
            }
            fill(frq.begin(), frq.end(), 0);
            for (int i = 0; i < n; i++) frq[c[i]]++;
            for (int i = 1; i < lim; i++)
                frq[i] += frq[i - 1];
            for (int i = n; i--;)
                sa[--frq[c[tmp[i]]]] = tmp[i];
            swap(c, tmp), p = 1, c[sa[0]] = 0;
            for (int i = 1; i < n; i++)
                a = sa[i - 1], b = sa[i], c[b] = (tmp[a] == tmp
                    [b] && tmp[a + j] == tmp[b + j]) ? p - 1 :
                     p++;
        }
        for (int i = 1; i < n; i++) rank[sa[i]] = i;
        for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
            for (k &&k--, j = sa[rank[i] - 1]; s[i + k] == s[j
                 + k];
        k++);
    }
    void preLcp() {
        int n = S.size() + 1;
        logs = vector<int>(n + 5);
        for (int i = 2; i < n + 5; ++i) {
            logs[i] = logs[i / 2] + 1;
        }
        table = vector<vector<int>>(n, vector<int>(20));
        for (int i = 0; i < n; ++i) {
            table[i][0] = lcp[i];
        }
        for (int j = 1; j <= logs[n]; ++j) {
            for (int i = 0; i <= n - (1 << j); ++i) {
                table[i][j] = min(table[i][j - 1], table[i + (1
                     << (j - 1))][j - 1]);
            }
        }
    }
    int queryLcp(int i, int j) {
        // if (i == j)return (int) S.size() - i;
        // i = rank[i], j = rank[j];
        if (i == j)return (int) S.size() - sa[i];
        if (i > j)
            swap(i, j);
        i++;
        int len = logs[j - i + 1];
        return min(table[i][len], table[j - (1 << len) + 1][len
            ]);
    }
};
```

PalindromicTree.cpp

Description: Palindromic Tree

1ae82a, 48 lines

```
class PalindromeTree {
public:
    int n, id, cur, tot;
    vector<array<int, 26>> go;
    vector<int> suflink, len, cnt;
    PalindromeTree() {};
    PalindromeTree(const string &s) {
        n = s.length();
        go.assign(n + 2, {});
        suflink.assign(n + 2, 0);
        len.assign(n + 2, 0);
        cnt.assign(n + 2, 0);
        suflink[0] = suflink[1] = 1;
        len[1] = -1;
        id = 2;
        cur = 0;
        tot = 0;
        for (int i = 0; i < n; i++) {
            add(s, i);
        }
    }
    int get(const string &s, int i, int v) {
        while (i - len[v] - 1 < 0 || s[i - len[v] - 1] != s[i])
             {
            v = suflink[v];
        }
        return v;
    }
    void add(const string &s, int i) {
        int ch = s[i] - 'a';
        cur = get(s, i, cur);
        if (go[cur][ch] == 0) {
            len[id] = 2 + len[cur];
            suflink[id] = go[get(s, i, suflink[cur])][ch];
            tot++;
            go[cur][ch] = id++;
        }
        cur = go[cur][ch];
        cnt[cur]++;
    }
    void countAll() {
        for (int i = id - 1; i >= 2; --i) {
            cnt[suflink[i]] += cnt[i];
        }
    }
    int cntDistinct() {
        return tot;
    }
};
```

SuffixAutomaton.cpp

Description: Suffix Automaton

9fcc42, 114 lines

```
const int M = 26, N = 1000005;
```

```
using pii = pair<int, int>;
struct suffixAutomaton {
    struct state {
        int len; // length of longest string in this class
        int link; // pointer to suffix link
        int next[M]; // adjacency list
        ll cnt; // number of times the strings in this state
            occur in the original string
        bool terminal; // by default, empty string is a suffix
        // a state is terminal if it corresponds to a suffix
        state() {
            len = 0, link = -1, cnt = 0;
            terminal = false;
            for (int i = 0; i < M; i++)
                next[i] = -1;
        }
    };
    vector<state> st;
    int sz, last, l;
    char offset = 'A'; // Careful!
    suffixAutomaton(string &s) {
        int l = s.length();
        st.resize(2 * l);
        for (int i = 0; i < 2 * l; i++)
            st[i] = state();
        sz = 1, last = 0;
        st[0].len = 0;
        st[0].link = -1;
        for (int i = 0; i < l; i++)
            addChar(s[i] - offset);
        for (int i = last; i != -1; i = st[i].link)
            st[i].terminal = true;
    }
    void addChar(int c) {
        int cur = sz++;
        assert(cur < N * 2);
        st[cur].len = st[last].len + 1;
        st[cur].cnt = 1;
        int p = last;
        while (p != -1 && st[p].next[c] == -1) {
            st[p].next[c] = cur;
            p = st[p].link;
        }
        last = cur;
        if (p == -1) {
            st[cur].link = 0;
            return;
        }
        int q = st[p].next[c];
        if (st[q].len == st[p].len + 1) {
            st[cur].link = q;
            return;
        }
        int clone = sz++;
        for (int i = 0; i < M; i++)
            st[clone].next[i] = st[q].next[i];
        st[clone].link = st[q].link;
        st[clone].len = st[p].len + 1;
        st[clone].cnt = 0; // cloned states initially have cnt
             = 0
        while (p != -1 and st[p].next[c] == q) {
            st[p].next[c] = clone;
            p = st[p].link;
        }
        st[q].link = st[cur].link = clone;
    }
    bool contains(string &t) {
        int cur = 0;
        for (int i = 0; i < t.length(); i++) {
            cur = st[cur].next[t[i] - offset];
            if (cur == -1)
                return false;
        }
        return true;
    }
    // alternatively, compute the number of paths in a DAG
    // since each substring corresponds to one unique path in
        SA
    ll numberOfSubstrings() {
        ll res = 0;
        for (int i = 1; i < sz; i++)
            res += st[i].len - st[st[i].link].len;
        return res;
    }
    void numberOfOccPreprocess() {
        vector<pii> v;
        for (int i = 1; i < sz; i++)
            v.emplace_back(st[i].len, i);
        sort(v.begin(), v.end(), greater<>());
        for (int i = 0; i < sz - 1; i++) {
            int suf = st[v[i].second].link;
            st[suf].cnt += st[v[i].second].cnt;
        }
    }
    ll numberOfOcc(string &t) {
        int cur = 0;
        for (int i = 0; i < t.length(); i++) {
            cur = st[cur].next[t[i] - offset];
            if (cur == -1)
                return 0;
        }
        return st[cur].cnt;
    }
    ll totLenSubstrings() {
        // different Substrings
        ll tot = 0;
        for (int i = 1; i < sz; i++) {
            ll shortest = st[st[i].link].len + 1;
            ll longest = st[i].len;
            ll num_strings = longest - shortest + 1;
            ll cur = num_strings * (longest + shortest) / 2;
            tot += cur;
        }
        return tot;
    }
};
```

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

edce47, 23 lines

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

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

IntervalCover.h

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

9e9d8d, 19 lines

```
template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I) {
  vi S(sz(I)), R;
  iota(all(S), 0);
  sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
  T cur = G.first;
  int at = 0;
  while (cur < G.second) { // (A)
    pair<T, int> mx = make_pair(cur, -1);
    while (at < sz(I) && I[S[at]].first <= cur) {
      mx = max(mx, make_pair(I[S[at]].second, S[at]));
      at++;
    }
    if (mx.second == -1) return {};
    cur = mx.first;
    R.push_back(mx.second);
  }
  return R;
}
```

ConstantIntervals.h

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

753a4c, 19 lines

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

10.2 Misc. algorithms

TernarySearch.h

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

9155b4, 11 lines

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

LIS.h

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

2932a0, 17 lines

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

FastKnapsack.h

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

b20ccc, 16 lines

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

10.3 Dynamic programming

KnuthDP.h

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

DivideAndConquerDP.h

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

d38d2b, 18 lines

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

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

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).

FastInput.h

Description: Read an integer from stdin. Usage requires your program to pipe in input from file.
Usage: `./a.out < input.txt`
Time: About 5x as fast as cin/scanf.

7b3c70, 17 lines

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

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

Techniques (A)

techniques.txt

159 lines

```
Recursion
Divide and conquer
  Finding interesting points in N log N
Algorithm analysis
  Master theorem
  Amortized time complexity
Greedy algorithm
  Scheduling
  Max contiguous subvector sum
  Invariants
  Huffman encoding
Graph theory
  Dynamic graphs (extra book-keeping)
  Breadth first search
  Depth first search
  * Normal trees / DFS trees
  Dijkstra's algorithm
  MST: Prim's algorithm
  Bellman-Ford
  Konig's theorem and vertex cover
  Min-cost max flow
  Lovasz toggle
  Matrix tree theorem
  Maximal matching, general graphs
  Hopcroft-Karp
  Hall's marriage theorem
  Graphical sequences
  Floyd-Warshall
  Euler cycles
  Flow networks
  * Augmenting paths
  * Edmonds-Karp
  Bipartite matching
  Min. path cover
  Topological sorting
  Strongly connected components
  2-SAT
  Cut vertices, cut-edges and biconnected components
  Edge coloring
  * Trees
  Vertex coloring
  * Bipartite graphs (=> trees)
  * 3^n (special case of set cover)
  Diameter and centroid
  K'th shortest path
  Shortest cycle
Dynamic programming
  Knapsack
  Coin change
  Longest common subsequence
  Longest increasing subsequence
  Number of paths in a dag
  Shortest path in a dag
  Dynprog over intervals
  Dynprog over subsets
  Dynprog over probabilities
  Dynprog over trees
  3^n set cover
  Divide and conquer
  Knuth optimization
  Convex hull optimizations
  RMQ (sparse table a.k.a 2^k-jumps)
  Bitonic cycle
  Log partitioning (loop over most restricted)
Combinatorics
  Computation of binomial coefficients
  Pigeon-hole principle
  Inclusion/exclusion
  Catalan number
  Pick's theorem
Number theory
  Integer parts
  Divisibility
  Euclidean algorithm
  Modular arithmetic
  * Modular multiplication
  * Modular inverses
  * Modular exponentiation by squaring
  Chinese remainder theorem
  Fermat's little theorem
  Euler's theorem
  Phi function
  Frobenius number
  Quadratic reciprocity
  Pollard-Rho
  Miller-Rabin
  Hensel lifting
  Vieta root jumping
Game theory
  Combinatorial games
  Game trees
  Mini-max
  Nim
  Games on graphs
  Games on graphs with loops
  Grundy numbers
  Bipartite games without repetition
  General games without repetition
  Alpha-beta pruning
Probability theory
Optimization
  Binary search
  Ternary search
  Unimodality and convex functions
  Binary search on derivative
Numerical methods
  Numeric integration
  Newton's method
  Root-finding with binary/ternary search
  Golden section search
Matrices
  Gaussian elimination
  Exponentiation by squaring
Sorting
  Radix sort
Geometry
  Coordinates and vectors
  * Cross product
  * Scalar product
  Convex hull
  Polygon cut
  Closest pair
  Coordinate-compression
  Quadtrees
  KD-trees
  All segment-segment intersection
Sweeping
  Discretization (convert to events and sweep)
  Angle sweeping
  Line sweeping
  Discrete second derivatives
Strings
  Longest common substring
  Palindrome subsequences
  Knuth-Morris-Pratt
  Tries
  Rolling polynomial hashes
  Suffix array
  Suffix tree
  Aho-Corasick
  Manacher's algorithm
  Letter position lists
Combinatorial search
  Meet in the middle
  Brute-force with pruning
  Best-first (A*)
  Bidirectional search
  Iterative deepening DFS / A*
Data structures
  LCA (2^k-jumps in trees in general)
  Pull/push-technique on trees
  Heavy-light decomposition
  Centroid decomposition
  Lazy propagation
  Self-balancing trees
  Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
  Monotone queues / monotone stacks / sliding queues
  Sliding queue using 2 stacks
  Persistent segment tree
```