

1	Contest	1
2	Mathematics	1
2.1	Equations	1
2.2	Recurrences	2
2.3	Trigonometry	2
2.4	Geometry	2
2.5	Derivatives/Integrals	2
2.6	Sums	2
2.7	Series	2
2.8	Probability theory	2
2.9	Markov chains	3
3	Data structures	3
3.1	Matrix	3
3.2	Treap	3
3.3	FenwickTree2d	4
3.4	RMQ	4
3.5	WaveletTree	4
3.6	KDTree	4
3.7	PersistentSegTree	5
4	Numerical	6
4.1	GoldenSectionSearch	6
4.2	Polynomial	6
4.3	PolyRoots	6
4.4	PolyInterpolate	6
4.5	LinearRecurrence	6
4.6	HillClimbing	7
4.7	Determinant	7
4.8	IntDeterminant	7
4.9	Simplex	7
4.10	MatrixInverse	7
4.11	SolveLinear	8
4.12	SolveLinear2	8
4.13	SolveLinearBinary	8
4.14	Fast Fourier transforms	8
4.15	FastFourierTransformMod	8
4.16	NumberTheoreticTransform	8
4.17	FastSubsetTransform	9
5	Number theory	9
5.1	Modular arithmetic	9
5.2	Primality	9
5.3	Divisibility	10
5.4	Fractions	10
5.5	Pythagorean Triples	10

5.6	Primes	11
5.7	Estimates	11
6	Combinatorial	11
6.1	Permutations	11
6.2	Partitions and subsets	11
6.3	General purpose numbers	11
7	Graph	12
7.1	BellmanFord	12
7.2	FloydWarshall	12
7.3	TopoSort	12
7.4	Euler walk	12
7.5	Network flow	12
7.6	MinCostMaxFlow	13
7.7	EdmondsKarp	13
7.8	MinCut	13
7.9	GlobalMinCut	13
7.10	hopcroftKarp	14
7.11	DFSMatching	14
7.12	MinimumVertexCover	14
7.13	WeightedMatching	14
7.14	GeneralMatching	15
7.15	DFS algorithms	15
7.16	BiconnectedComponents	15
7.17	2sat	15
7.18	MaximalCliques	16
7.19	MaximumClique	16
7.20	MaximumIndependentSet	16
7.21	TreePower	16
7.22	LCA	16
7.23	CompressTree	16
7.24	HLD	17
7.25	LinkCutTree	17
7.26	DirectedMST	18
8	Geometry	18
8.1	Geometric primitives	18
8.2	Circles	19
8.3	Polygons	20
8.4	Misc. Point Set Problems	20
8.5	3D	21
9	Strings	22
9.1	KMP	22
9.2	Zfunc	22
9.3	Manacher	22

9.4	MinRotation	22
9.5	SuffixArray	22
9.6	SuffixTree	23
9.7	Hashing	23
9.8	AhoCorasick	23
10	Various	24
10.1	Intervals	24
10.2	TernarySearch	24
10.3	Pick's Theorem	24
10.4	Optimization tricks	24

Contest (1)

```
template.cpp 15 lines
#include <bits/stdc++.h>
using namespace std;

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

int main() {
    cin.sync_with_stdio(0); cin.tie(0);
    cin.exceptions(cin.failbit);
}
```

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

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

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

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):

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

template

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

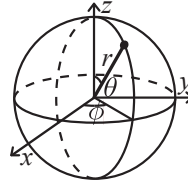
2.4.2 Quadrilaterals

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

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

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

2.4.3 Spherical coordinates



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

2.5 Derivatives/Integrals

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

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

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

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

Integration by parts:

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

2.6 Sums

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

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

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

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

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

2.7 Series

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

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

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

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

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

2.8 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x . It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$ where 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 ,

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

2.8.1 Discrete distributions

Binomial distribution

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

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

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

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

First success distribution

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

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

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

Poisson distribution

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

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

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

2.8.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

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

Normal distribution

Most real random values with mean 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). $\lim_{k \rightarrow \infty} \mathbf{P}^k = \mathbf{1P}$.

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

Data structures (3)

3.1 Matrix

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;
    }
};
```

3.2 Treap

Treap.h

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

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

9556fc, 55 lines

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

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

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

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

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

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

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

3.3 FenwickTree2d

FenwickTree2d.h

Description: Computes sums $a[i,j]$ for all $i < l, j < J$, and increases single elements $a[i,j]$. Requires that the elements to be updated are known in advance (call `fakeUpdate()` before `init()`).

Time: $\mathcal{O}(\log^2 N)$. (Use persistent segment trees for $\mathcal{O}(\log N)$.)

```
"FenwickTree.h" b28c27, 22 lines

struct FT2 {
    vector<vi> ys; vector<FT> ft;
    FT2(int limx) : ys(limx) {}
    void fakeUpdate(int x, int y) {
        for (; x < sz(ys); x |= x + 1) ys[x].push_back(y);
    }
    void init() {
        trav(v, ys) sort(all(v)), ft.emplace_back(sz(v));
    }
    int ind(int x, int y) {
        return (int)(lower_bound(all(ys[x]), y) - ys[x].begin()); }
    void update(int x, int y, ll dif) {
        for (; x < sz(ys); x |= x + 1)
```

```
        ft[x].update(ind(x, y), dif);
    }
    ll query(int x, int y) {
        ll sum = 0;
        for (; x; x &= x - 1)
            sum += ft[x-1].query(ind(x-1, y));
        return sum;
    }
};
```

3.4 RMQ

RMQ.h

Description: Range Minimum Queries on an array. Returns $\min(V[a], V[a + 1], \dots, V[b - 1])$ in constant time.

Usage: `RMQ rmq(values);`

`rmq.query(inclusive, exclusive);`

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

1f8996, 17 lines

```
template<class T>
struct RMQ {
    vector<vector<T>> jmp;
    RMQ(const vector<T>& V) {
        int N = sz(V), on = 1, depth = 1;
        while (on < N) on *= 2, depth++;
        jmp.assign(depth, V);
        rep(i, 0, depth-1) rep(j, 0, N)
            jmp[i+1][j] = min(jmp[i][j],
                               jmp[i][min(N - 1, j + (1 << i))]);
    }
    T query(int a, int b) {
        assert(a < b); // or return inf if a == b
        int dep = 31 - __builtin_clz(b - a);
        return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
    }
};
```

3.5 WaveletTree

WaveletTree.h

1cd45c, 55 lines

```
template <class T>
struct wavelet{
    struct node {
        vector<int> b;
        T lo,hi,md;
    };
    vector<node> t;

    void build(const vector<T> &c, T *A, T *B, int v, int i, int j) {
        t[v].b.resize(B-A+1);
        t[v].lo = c[i], t[v].hi = c[j], t[v].md = c[(i+j)/2];
        for(int i = 0; A+i != B; ++i)
            t[v].b[i+1] = t[v].b[i] + (A[i]<=t[v].md);
        if(i == j) return;
        T *p = stable_partition(A,B,[=](int x){return x <= t[v].md});
        build(c,A,p,v<<1,i,(i+j)/2);
        build(c,p,B,v<<1|1,(i+j)/2+1,j);
    }

    void init(T *A, int n) {
        vector<T> c(A,A+n);
        sort(c.begin(),c.end());
        c.erase(unique(c.begin(),c.end()),c.end());
        int N = c.size();
```

```
        t.resize(N<<2);
        build(c,A,A+n,1,0,N-1);
    }

    wavelet(){}
    wavelet(T *A, int n){init(A,n);}
```

//kth smallest element in [l, r]

```
T kth(int l, int r, int k, int v = 1) {
    if(t[v].lo == t[v].hi) return t[v].lo;
    int lb = t[v].b[l-1], rb = t[v].b[r], il = rb-lb;
    return (k < il) ? kth(lb+1,rb,k,v<<1) : kth(l-lb,r-rb,k-il,
        v<<1|1);
}
```

//number of elements in [l,r] <= to a

```
int leq(int l, int r, T a, int v = 1) {
    if(a < t[v].lo) return 0;
    if(t[v].hi <= a) return r-l+1;
    int lb = t[v].b[l-1], rb = t[v].b[r];
    return leq(lb+1,rb,a,v<<1) + leq(l-lb,r-rb,a,v<<1|1);
}
```

//number of elements in [l, r] equal to a

```
int count(int l, int r, T a, int v = 1) {
    if(a < t[v].lo || a > t[v].hi) return 0;
    if(t[v].lo == t[v].hi) return r-l+1;
    int lb = t[v].b[l-1], rb = t[v].b[r];
    if(a <= t[v].md) return count(lb+1,rb,a,v<<1);
    return count(l-lb,r-rb,a,v<<1|1);
}
};
```

3.6 KDTree

KDTree.h

<bits/stdc++.h> ff695b, 182 lines

```
// constructs from n points in  $\mathcal{O}(n \lg^2 n)$  time
// handles nearest-neighbor query in  $\mathcal{O}(\lg n)$  if points are well distributed
//  $\mathcal{O}(\log n)$  average nearest neighbor,  $\mathcal{O}(n)$  worst in pathological case
```

using namespace std;

```
// number type for coordinates, and its maximum value
typedef long long ntype;
const ntype sentry = numeric_limits<ntype>::max();
```

// point structure for 2D-tree, can be extended to 3D

```
struct point {
    ntype x, y;
    point(ntype xx = 0, ntype yy = 0) : x(xx), y(yy) {}
};
```

```
bool operator==(const point &a, const point &b) {
    return a.x == b.x && a.y == b.y;
}
```

// sorts points on x-coordinate

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

// sorts points on y-coordinate

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

```

}

// squared distance between points
ntype pdist2(const point &a, const point &b) {
    ntype dx = a.x-b.x, dy = a.y-b.y;
    return dx*dx + dy*dy;
}

// bounding box for a set of points
struct bbox {
    ntype x0, x1, y0, y1;

    bbox() : x0(sentry), x1(-sentry), y0(sentry), y1(-sentry) {}

    // computes bounding box from a bunch of points
    void compute(const vector<point> &v) {
        for (int i = 0; i < v.size(); ++i) {
            x0 = min(x0, v[i].x);    x1 = max(x1, v[i].x);
            y0 = min(y0, v[i].y);    y1 = max(y1, v[i].y);
        }
    }

    // squared distance between a point and this bbox, 0 if
    // inside
    ntype distance(const point &p) {
        if (p.x < x0) {
            if (p.y < y0)        return pdist2(point(x0, y0), p);
            ;
            else if (p.y > y1)    return pdist2(point(x0, y1), p);
            ;
            else                  return pdist2(point(x0, p.y), p);
        }
        else if (p.x > x1) {
            if (p.y < y0)        return pdist2(point(x1, y0), p);
            ;
            else if (p.y > y1)    return pdist2(point(x1, y1), p);
            ;
            else                  return pdist2(point(x1, p.y), p);
        }
        else {
            if (p.y < y0)        return pdist2(point(p.x, y0), p);
            ;
            else if (p.y > y1)    return pdist2(point(p.x, y1), p);
            ;
            else                  return 0;
        }
    }
};

// stores a single node of the kd-tree, either internal or leaf
struct kndnode {
    bool leaf;        // true if this is a leaf node (has one
                      // point)
    point pt;         // the single point of this is a leaf
    bbox bound;       // bounding box for set of points in
                      // children

    kndnode *first, *second; // two children of this kd-node

    kndnode() : leaf(false), first(0), second(0) {}
    ~kndnode() { if (first) delete first; if (second) delete
                  second; }

    // intersect a point with this node (returns squared
    // distance)
    ntype intersect(const point &p) {

```

```

        return bound.distance(p);
    }

    // recursively builds a kd-tree from a given cloud of
    // points
    void construct(vector<point> &vp) {
        // compute bounding box for points at this node
        bound.compute(vp);

        // if we're down to one point, then we're a leaf node
        if (vp.size() == 1) {
            leaf = true;
            pt = vp[0];
        }
        else {
            // split on x if the bbox is wider than high (not
            // best heuristic...)
            if (bound.x1-bound.x0 >= bound.y1-bound.y0)
                sort(vp.begin(), vp.end(), on_x);
            // otherwise split on y-coordinate
            else
                sort(vp.begin(), vp.end(), on_y);

            // divide by taking half the array for each child
            // (not best performance if many duplicates in the
            // middle)
            int half = vp.size()/2;
            vector<point> vl(vp.begin(), vp.begin()+half);
            vector<point> vr(vp.begin()+half, vp.end());
            first = new kndnode();    first->construct(vl);
            second = new kndnode();   second->construct(vr);
        }
    }
};

// simple kd-tree class to hold the tree and handle queries
struct kdtree {
    kndnode *root;

    // constructs a kd-tree from a points (copied here, as it
    // sorts them)
    kdtree(const vector<point> &vp) {
        vector<point> v(vp.begin(), vp.end());
        root = new kndnode();
        root->construct(v);
    }
    ~kdtree() { delete root; }

    // recursive search method returns squared distance to
    // nearest point
    ntype search(kndnode *node, const point &p) {
        if (node->leaf) {
            // commented special case tells a point not to find
            // itself
            if (p == node->pt) return sentry;
            else
                return pdist2(p, node->pt);
        }

        ntype bfirst = node->first->intersect(p);
        ntype bsecond = node->second->intersect(p);

        // choose the side with the closest bounding box to
        // search first
        // (note that the other side is also searched if needed)
        if (bfirst < bsecond) {
            ntype best = search(node->first, p);
            if (bsecond < best)

```

```

                best = min(best, search(node->second, p));
            return best;
        }
        else {
            ntype best = search(node->second, p);
            if (bfirst < best)
                best = min(best, search(node->first, p));
            return best;
        }
    }

    // squared distance to the nearest
    ntype nearest(const point &p) {
        return search(root, p);
    }
};

//
//
// some basic test code here

int main() {
    // generate some random points for a kd-tree
    vector<point> vp;
    for (int i = 0; i < 100000; ++i)
        vp.push_back(point(rand()%100000, rand()%100000));
    kdtree tree(vp);

    // query some points
    for (int i = 0; i < 10; ++i) {
        point q(rand()%100000, rand()%100000);
        cout << "Closest squared distance to (" << q.x << ", "
              << q.y << ") "
              << " is " << tree.nearest(q) << endl;
    }

    return 0;
}

//

```

3.7 PersistentSegTree

PersistentSegTree.h

<iostream>, <vector> 44b362, 82 lines

```

using namespace std;

/* persistent segment tree w/ sum query */
struct per_seg {
    vector<int> lc,rc,lx,rx,val,roots = {0};

    //CHANGE ME
    inline static int combine(int a, int b) {
        return a+b;
    }

    //build PST on indices i ... j of array a
    int build(int i, int j, int* a) {
        int v = lc.size();
        lc.push_back(-1);
        rc.push_back(-1);
        lx.push_back(i);
        rx.push_back(j);
        val.push_back(a[i]);
    }

```



```
    if(i != j) {
        int l = build(i, (i+j)/2, a), r = build((i+j)/2+1, j, a);
        lc[v] = l;
        rc[v] = r;
        val[v] = combine(val[l], val[r]);
    }
    return v;
}

int q(int v, int i, int j) {
    if(j < lx[v] || rx[v] < i)
        return 0;
    if(i <= lx[v] && rx[v] <= j)
        return val[v];
    return combine(q(lc[v], i, j), q(rc[v], i, j));
}

int u(int v, int i, int a) {
    if(i < lx[v] || rx[v] < i)
        return v;
    int w = lc.size();
    lc.push_back(lc[v]);
    rc.push_back(rc[v]);
    lx.push_back(lx[v]);
    rx.push_back(rx[v]);
    //CHANGE ME
    val.push_back(val[v]+a);
    if(lx[v] != rx[v]) {
        int l = u(lc[v], i, a), r = u(rc[v], i, a);
        lc[w] = l;
        rc[w] = r;
        val[w] = combine(val[l], val[r]);
    }
    return w;
}

//sum from i to j after t updates
int query(int i, int j, int t = -1) {
    if(t == -1) t = roots.size()-1;
    return q(roots[t], i, j);
}

//add a to position i at time t
void update(int i, int a, int t) {
    roots.push_back(u(roots[t], i, a));
}

};

/* USAGE */

int main() {
    int a[5] = {1,1,1,1,1};
    per_seg p;
    p.build(0,4,a);
    p.update(1,1);
    p.update(3,2);
    cout << p.query(2,2,5) << "\n";
    cout << p.query(0,2,5) << "\n";
    cout << p.query(1,0,3) << "\n"
    return 0;
}
```

Numerical (4)

4.1 GoldenSectionSearch

GoldenSectionSearch.h

Description: Finds the argument minimizing the function f in the interval $[a, b]$ assuming f is unimodal on the interval, i.e. has only one local minimum. The maximum error in the result is ϵ . Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.

Usage: double func(double x) { return 4+x+.3*x*x; }

double xmin = gss(-1000,1000,func);

Time: $\mathcal{O}(\log((b-a)/\epsilon))$

31d45b, 14 lines

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

4.2 Polynomial

Polynomial.h

c9b7b0, 17 lines

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

4.3 PolyRoots

PolyRoots.h

Description: Finds the real roots to a polynomial.

Usage: poly_roots({{2,-3,1}},-1e9,1e9) // solve $x^2-3x+2=0$

Time: $\mathcal{O}(n^2 \log(1/\epsilon))$

"Polynomial.h"

2cf190, 23 lines

```
vector<double> poly_roots(Poly p, double xmin, double xmax) {
    if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
    vector<double> ret;
    Poly der = p;
    der.diff();
    auto dr = poly_roots(der, xmin, xmax);
    dr.push_back(xmin-1);
    dr.push_back(xmax+1);
}
```

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

4.4 PolyInterpolate

PolyInterpolate.h

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

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

08bf48, 13 lines

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

4.5 LinearRecurrence

LinearRecurrence.h

Description: Generates the k 'th term of an n -order linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$, given $S[0 \dots n-1]$ and $tr[0 \dots n-1]$. Faster than matrix multiplication. Useful together with Berlekamp-Massey.

Usage: linearRec({0, 1}, {1, 1}, k) // k 'th Fibonacci number

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

f4e444, 26 lines

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

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

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

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



```
    }

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

4.6 HillClimbing

HillClimbing.h
Description: Poor man's optimization for unimodal functions. f40e55, 16 lines

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

double func(P p);

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

4.7 Determinant

Determinant.h
Description: Calculates determinant of a matrix. Destroys the matrix.
Time: $\mathcal{O}(N^3)$ bd5cec, 15 lines

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

4.8 IntDeterminant

IntDeterminant.h
Description: Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.
Time: $\mathcal{O}(N^3)$ 3313dc, 18 lines

```
const ll mod = 12345;
ll det(vector<vector<ll>>& a) {
    int n = sz(a); ll ans = 1;
    rep(i,0,n) {
        rep(j,i+1,n) {
            while (a[j][i] != 0) { // gcd step
                ll t = a[i][i] / a[j][i];
                if (t) rep(k,i,n)
```

```
                a[i][k] = (a[i][k] - a[j][k] * t) % mod;
                swap(a[i], a[j]);
                ans *= -1;
            }
        }
        ans = ans * a[i][i] % mod;
        if (!ans) return 0;
    }
    return (ans + mod) % mod;
}
```

4.9 Simplex

Simplex.h
Description: Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \leq b$, $x \geq 0$. Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of $c^T x$ otherwise. The input vector is set to an optimal x (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that $x = 0$ is viable.
Usage: vvd A = {{1,-1}, {-1,1}, {-1,-2}};
vvd b = {1,1,-4}, c = {-1,-1}, x;
T val = LPSolver(A, b, c).solve(x);
Time: $\mathcal{O}(NM * \#pivots)$, where a pivot may be e.g. an edge relaxation.
 $\mathcal{O}(2^n)$ in the general case. aa8530, 68 lines

```
typedef double T; // long double, Rational, double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;
```

```
const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j
```

```
struct LPSolver {
    int m, n;
    vi N, B;
    vvd D;

    LPSolver(const vvd& A, const vd& b, const vd& c) :
        m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
        rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
        rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
        rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
        N[n] = -1; D[m+1][n] = 1;
    }
}
```

```
void pivot(int r, int s) {
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
        T *b = D[i].data(), inv2 = b[s] * inv;
        rep(j,0,n+2) b[j] -= a[j] * inv2;
        b[s] = a[s] * inv2;
    }
    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
}
```

```
bool simplex(int phase) {
    int x = m + phase - 1;
    for (;;) {
        int s = -1;
        rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
        if (D[x][s] >= -eps) return true;
        int r = -1;
        rep(i,0,m) {
```

```
            if (D[i][s] <= eps) continue;
            if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                < MP(D[r][n+1] / D[r][s], B[r])) r = i;
        }
        if (r == -1) return false;
        pivot(r, s);
    }
}
```

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

4.10 MatrixInverse

MatrixInverse.h
Description: Invert matrix A . Returns rank; result is stored in A unless singular (rank < n). Can easily be extended to prime moduli; for prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ where A^{-1} starts as the inverse of $A \bmod p$, and k is doubled in each step.
Time: $\mathcal{O}(n^3)$ ebfff6, 35 lines

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

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

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


4.17 FastSubsetTransform

Description: Transform to a basis with fast convolutions of the form $c[z] = \sum_{z=x \oplus y} a[x] \cdot b[y]$, where \oplus is one of AND, OR, XOR. The size of a must be a power of two.

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

3de473, 16 lines

Number theory (5)

5.1 Modular arithmetic

```
"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); }
}
```

```
ll modsum(ull to, ll c, ll k, ll m) {
    c = ((c % m) + m) % m;
```

5.2 Primalty

```
const int MAX_PR = 5'000'000;
bitset<MAX_PR> isprime;
vi eratosthenes_sieve(int lim) {
    isprime.set(); isprime[0] = isprime[1] = 0;
    for (int i = 4; i < lim; i += 2) isprime[i] = 0;
    for (int i = 3; i*i < lim; i += 2) if (isprime[i])
        for (int j = i*i; j < lim; j += i*2) isprime[j] = 0;
    vi pr;
}
```

```
rep(i,2,lim) if (isprime[i]) pr.push_back(i);
return pr;
}
```

MillerRabin.h

Description: Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to 2^{64} ; for larger numbers, extend A randomly.
Time: 7 times the complexity of $a^b \bmod c$.

"ModMuLL.h"	6ab8e1, 12 lines
<pre>bool isPrime(ull n) { if (n < 2 n % 6 % 4 != 1) return n - 2 < 2; ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022}, s = __builtin_ctzll(n-1), d = n >> s; trav(a, A) { // ^ count trailing zeroes ull p = mod_pow(a, d, n), i = s; while (p != 1 && p != n - 1 && a % n && i--) p = mod_mul(p, p, n); if (p != n-1 && i != s) return 0; } return 1; }</pre>	

Factor.h

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

"ModMuLL.h", "MillerRabin.h"	f5adaa, 18 lines
<pre>ull pollard(ull n) { auto f = [n](ull x) { return (mod_mul(x, x, n) + 1) % n; }; if (!(n & 1)) return 2; for (ull i = 2;; i++) { ull x = i, y = f(x), p; while ((p = __gcd(n + y - x, n)) == 1) x = f(x), y = f(f(y)); if (p != n) return p; } } 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; }</pre>	

5.3 Divisibility

euclid.h

Description: Finds two integers x and y , such that $ax + by = \gcd(a, b)$. If you just need gcd, use the built in `_gcd` instead. If a and b are coprime, then x is the inverse of $a \pmod b$.

ee6239, 5 lines
<pre>ll euclid(ll a, ll b, ll &x, ll &y) { if (b) { ll d = euclid(b, a % b, y, x); return y -= a/b * x, d; } return x = 1, y = 0, a; }</pre>

Euclid.java

Description: Finds {x, y, d} s.t. $ax + by = d = \gcd(a, b)$.

6aba01, 11 lines
<pre>static BigInteger[] euclid(BigInteger a, BigInteger b) { BigInteger x = BigInteger.ONE, yy = x; BigInteger y = BigInteger.ZERO, xx = y; while (b.signum() != 0) {</pre>

	<pre>BigInteger q = a.divide(b), t = b; b = a.mod(b); a = t; t = xx; xx = x.subtract(q.multiply(xx)); x = t; t = yy; yy = y.subtract(q.multiply(yy)); y = t; } return new BigInteger[]{x, y, a; }</pre>
--	---

CRT.h

Description: Chinese Remainder Theorem.

`crt(a, m, b, n)` computes x such that $x \equiv a \pmod m$, $x \equiv b \pmod n$. If $|a| < m$ and $|b| < n$, x will obey $0 \leq x < \text{lcm}(m, n)$. Assumes $mn < 2^{62}$.

"euclid.h"	04d93a, 7 lines
<pre>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; }</pre>	

5.3.1 Bézout’s identity

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

$$ax + by = d$$

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

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

phiFunction.h

Description: *Euler’s* ϕ function is defined as $\phi(n) := \#$ of positive integers $\leq n$ that are coprime with n . $\phi(1) = 1$, p prime $\Rightarrow \phi(p^k) = (p - 1)p^{k-1}$, m, n coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$. If $n = p_1^{k_1}p_2^{k_2} \dots p_r^{k_r}$ then $\phi(n) = (p_1 - 1)p_1^{k_1-1} \dots (p_r - 1)p_r^{k_r-1}$. $\phi(n) = n \cdot \prod_{p|n} (1 - 1/p)$.

$\sum_{d|n} \phi(d) = n$, $\sum_{1 \leq k \leq n, \gcd(k, n) = 1} k = n\phi(n)/2, n > 1$

Euler’s thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod n$.

Fermat’s little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod p \ \forall a$.

cf7d6d, 8 lines
<pre>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; }</pre>

5.4 Fractions

ContinuedFractions.h

Description: Given N and a real number $x \geq 0$, finds the closest rational approximation p/q with $p, q \leq N$. It will obey $|p/q - x| \leq 1/qN$.

For consecutive convergents, $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$. (p_k/q_k) alternates between $> x$ and $< x$.) If x is rational, y eventually becomes ∞ ; if x is the root of a degree 2 polynomial the a ’s eventually become cyclic.

Time: $\mathcal{O}(\log N)$	dd6c5e, 21 lines
<pre>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; } }</pre>	

FracBinarySearch.h

Description: Given f and N , finds the smallest fraction $p/q \in [0, 1]$ such that $f(p/q)$ is true, and $p, q \leq N$. You may want to throw an exception from f if it finds an exact solution, in which case N can be removed.

Usage: `fracBS({}(Frac f) { return f.p>=3*f.q; }, 10);` // {1,3}

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

27ab3e, 25 lines
<pre>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; }</pre>

5.5 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

5.6 Primes

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

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

5.7 Estimates

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

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

Combinatorial (6)

6.1 Permutations

6.1.1 Factorial

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

IntPerm.h

Description: Permutation -> integer conversion. (Not order preserving.)

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

e1b8ea, 6 lines

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

6.1.2 Cycles

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

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

6.1.3 Derangements

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

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

6.1.4 Burnside’s lemma

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

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

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

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

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

6.2 Partitions and subsets

6.2.1 Partition function

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

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

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

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

6.2.2 Binomials

binomialModPrime.h

Description: Lucas’ thm: Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + \dots + n_1 p + n_0$ and $m = m_k p^k + \dots + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^k \binom{n_i}{m_i} \pmod{p}$. fact and invfact must hold pre-computed factorials / inverse factorials, e.g. from ModInverse.h.

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

s1845f, 10 lines

```
ll chooseModP(ll n, ll m, int p, vi& fact, vi& invfact) {
    ll c = 1;
    while (n || m) {
        ll a = n % p, b = m % p;
        if (a < b) return 0;

```

```
    c = c * fact[a] % p * invfact[b] % p * invfact[a - b] % p;
    n /= p; m /= p;
}
return c;
}
```

multinomial.h

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

a0a312, 6 lines

```
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;
}
```

6.3 General purpose numbers

6.3.1 Bernoulli numbers

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

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

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

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

6.3.3 Eulerian numbers

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

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

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

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

6.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.3.5 Bell numbers

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

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

6.3.6 Labeled unrooted trees

on n vertices: n^{n-2}

on k existing trees of size n_i : $n_1 n_2 \cdots n_k n^{k-2}$

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

6.3.7 Catalan numbers

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

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

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

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

Graph (7)

7.1 BellmanFord

BellmanFord.h

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

```
53f20b, 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) trav(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) trav(e, eds) {
        if (nodes[e.a].dist == -inf)
            nodes[e.b].dist = -inf;
    }
}
```

7.2 FloydWarshall

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 $m[i][j] = \text{inf}$ 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 $-\text{inf}$ 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;
}
```

7.3 TopoSort

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

```
vi topo_sort(const vector<vi>& gr) {
    vi indeg(sz(gr)), ret;
    trav(li, gr) trav(x, li) indeg[x]++;
    queue<int> q; // use priority queue for lexic. smallest ans.
    rep(i, 0, sz(gr)) if (indeg[i] == 0) q.push(-i);
    while (!q.empty()) {
        int i = -q.front(); // top() for priority queue
        ret.push_back(i);
        q.pop();
        trav(x, gr[i])
            if (--indeg[x] == 0) q.push(-x);
    }
    return ret;
}
```

7.4 Euler walk

EulerWalk.h

Description: Eulerian undirected/directed path/cycle algorithm. Returns a list of nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, also put it->second in s (and then ret).

Time: $\mathcal{O}(E)$ where E is the number of edges.

```
f8bd47, 27 lines
struct V {
    vector<pii> outs; // (dest, edge index)
    int nins = 0;
};

vi euler_walk(vector<V>& nodes, int nedges, int src=0) {
    int c = 0;
    trav(n, nodes) c += abs(n.nins - sz(n.outs));
    if (c > 2) return {};
    vector<vector<pii>::iterator> its;
    trav(n, nodes)
        its.push_back(n.outs.begin());
    vector<bool> eu(nedges);
    vi ret, s = {src};
    while(!s.empty()) {
        int x = s.back();
        auto& it = its[x], end = nodes[x].outs.end();
        while(it != end && eu[it->second]) ++it;
        if(it == end) { ret.push_back(x); s.pop_back(); }
        else { s.push_back(it->first); eu[it->second] = true; }
    }
    if(sz(ret) != nedges+1)
        ret.clear(); // No Eulerian cycles/paths.
    // else, non-cycle if ret.front() != ret.back()
    reverse(all(ret));
    return ret;
}
```

7.5 Network flow

PushRelabel.h

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

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

```
6c4045, 49 lines
typedef ll Flow;
struct Edge {
    int dest, back;
    Flow f, c;
};
```

```

struct PushRelabel {
    vector<vector<Edge>> g;
    vector<Flow> ec;
    vector<Edge*> cur;
    vector<vi> hs; vi H;
    PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}

    void add_edge(int s, int t, Flow cap, Flow rcap=0) {
        if (s == t) return;
        g[s].push_back({t, sz(g[t]), 0, cap});
        g[t].push_back({s, sz(g[s])-1, 0, rcap});
    }

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

    Flow maxflow(int s, int t) {
        int v = sz(g); H[s] = v; ec[t] = 1;
        vi co(2*v); co[0] = v-1;
        rep(i,0,v) cur[i] = g[i].data();
        trav(e, g[s]) add_flow(e, e.c);

        for (int hi = 0;;) {
            while (hs[hi].empty()) if (!hi--) return -ec[s];
            int u = hs[hi].back(); hs[hi].pop_back();
            while (ec[u] > 0) // discharge u
                if (cur[u] == g[u].data() + sz(g[u])) {
                    H[u] = le9;
                    trav(e, g[u]) if (e.c && H[u] > H[e.dest]+1)
                        H[u] = H[e.dest]+1, cur[u] = &e;
                    if (++co[H[u]],!--co[hi] && hi < v)
                        rep(i,0,v) if (hi < H[i] && H[i] < v)
                            --co[H[i]], H[i] = v + 1;
                    hi = H[u];
                } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
                    add_flow(*cur[u], min(ec[u], cur[u]->c));
                else ++cur[u];
        }
    }
};

```

7.6 MinCostMaxFlow

MinCostMaxFlow.h

Description: Min-cost max-flow. $\text{cap}[i][j] \neq \text{cap}[j][i]$ is allowed; double edges are not. If costs can be negative, call `setpi` before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.

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

6915ce, 81 lines

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

```
const ll INF = numeric_limits<ll>::max() / 4;
typedef vector<ll> VL;
```

```

struct MCMF {
    int N;
    vector<vi> ed, red;
    vector<VL> cap, flow, cost;
    vi seen;
    VL dist, pi;
    vector<pii> par;
```

```

    MCMF(int N) :
        N(N), ed(N), red(N), cap(N, VL(N)), flow(cap), cost(cap),
```

```
    seen(N), dist(N), pi(N), par(N) {}
```

```

    void addEdge(int from, int to, ll cap, ll cost) {
        this->cap[from][to] = cap;
        this->cost[from][to] = cost;
        ed[from].push_back(to);
        red[to].push_back(from);
    }

```

```

    void path(int s) {
        fill(all(seen), 0);
        fill(all(dist), INF);
        dist[s] = 0; ll di;
```

```

        __gnu_pbds::priority_queue<pair<ll, int>> q;
        vector<decltype(q)::point_iterator> its(N);
        q.push({0, s});
```

```

        auto relax = [&](int i, ll cap, ll cost, int dir) {
            ll val = di - pi[i] + cost;
            if (cap && val < dist[i]) {
                dist[i] = val;
                par[i] = {s, dir};
                if (its[i] == q.end()) its[i] = q.push({-dist[i], i});
                else q.modify(its[i], {-dist[i], i});
            }
        };

```

```

        while (!q.empty()) {
            s = q.top().second; q.pop();
            seen[s] = 1; di = dist[s] + pi[s];
            trav(i, ed[s]) if (!seen[i])
                relax(i, cap[s][i] - flow[s][i], cost[s][i], 1);
            trav(i, red[s]) if (!seen[i])
                relax(i, flow[i][s], -cost[i][s], 0);
        }
        rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
    }

```

```

pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
        ll fl = INF;
        for (int p,r,x = t; tie(p,r) = par[x], x != s; x = p)
            fl = min(fl, r ? cap[p][x] - flow[p][x] : flow[x][p]);
        totflow += fl;
        for (int p,r,x = t; tie(p,r) = par[x], x != s; x = p)
            if (r) flow[p][x] += fl;
            else flow[x][p] -= fl;
    }
    rep(i,0,N) rep(j,0,N) totcost += cost[i][j] * flow[i][j];
    return {totflow, totcost};
}

```

```

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

```

};

7.7 EdmondsKarp

EdmondsKarp.h

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

979bb9, 35 lines

```

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

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

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

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

```

7.8 MinCut

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.

7.9 GlobalMinCut

GlobalMinCut.h

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

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

03261f, 31 lines

```

pair<int, vi> GetMinCut(vector<vi>& weights) {
    int N = sz(weights);
    vi used(N), cut, best_cut;
    int best_weight = -1;

    for (int phase = N-1; phase >= 0; phase--) {
        vi w = weights[0], added = used;
```




```
int prev, k = 0;
rep(i,0,phase){
    prev = k;
    k = -1;
    rep(j,1,N)
        if (!added[j] && (k == -1 || w[j] > w[k])) k = j;
    if (i == phase-1) {
        rep(j,0,N) weights[prev][j] += weights[k][j];
        rep(j,0,N) weights[j][prev] = weights[prev][j];
        used[k] = true;
        cut.push_back(k);
        if (best_weight == -1 || w[k] < best_weight) {
            best_cut = cut;
            best_weight = w[k];
        }
    } else {
        rep(j,0,N)
            w[j] += weights[k][j];
        added[k] = true;
    }
}
return {best_weight, best_cut};
}
```

7.10 hopcroftKarp

hopcroftKarp.h

Description: Fast bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and $btoa$ should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched.

Usage: vi btoa(m, -1); hopcroftKarp(g, btoa);

Time: $\mathcal{O}(\sqrt{VE})$ 536939, 42 lines

```
bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi& B) {
    if (A[a] != L) return 0;
    A[a] = -1;
    trav(b, g[a]) if (B[b] == L + 1) {
        B[b] = 0;
        if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
            return btoa[b] = a, 1;
    }
    return 0;
}
```

```
int hopcroftKarp(vector<vi>& g, vi& btoa) {
    int res = 0;
    vi A(g.size()), B(btoa.size()), cur, next;
    for (;;) {
        fill(all(A), 0);
        fill(all(B), 0);
        cur.clear();
        trav(a, btoa) if (a != -1) A[a] = -1;
        rep(a,0,sz(g)) if (A[a] == 0) cur.push_back(a);
        for (int lay = 1; lay++ ) {
            bool islast = 0;
            next.clear();
            trav(a, cur) trav(b, g[a]) {
                if (btoa[b] == -1) {
                    B[b] = lay;
                    islast = 1;
                }
            }
            else if (btoa[b] != a && !B[b]) {
                B[b] = lay;
                next.push_back(btoa[b]);
            }
        }
    }
}
```

```
if (islast) break;
if (next.empty()) return res;
trav(a, next) A[a] = lay;
cur.swap(next);
}
rep(a,0,sz(g))
    res += dfs(a, 0, g, btoa, A, B);
}
```

7.11 DFSMatching

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: $\mathcal{O}(VE)$ 6a3472, 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];
    trav(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);
        trav(j,g[i])
            if (find(j, g, btoa, vis)) {
                btoa[j] = i;
                break;
            }
    }
    return sz(btoa) - (int)count(all(btoa), -1);
}
```

7.12 MinimumVertexCover

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" d0b3f2, 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);
    trav(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;
        trav(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;
}
```

7.13 WeightedMatching

WeightedMatching.h

Description: Min cost bipartite matching. Negate costs for max cost.

Time: $\mathcal{O}(N^3)$ 055ca9, 75 lines

```
typedef vector<double> vd;
bool zero(double x) { return fabs(x) < 1e-10; }
double MinCostMatching(const vector<vd>& cost, vi& L, vi& R) {
    int n = sz(cost), mated = 0;
    vd dist(n), u(n), v(n);
    vi dad(n), seen(n);
```

```
rep(i,0,n) {
    u[i] = cost[i][0];
    rep(j,1,n) u[i] = min(u[i], cost[i][j]);
}
rep(j,0,n) {
    v[j] = cost[0][j] - u[0];
    rep(i,1,n) v[j] = min(v[j], cost[i][j] - u[i]);
}
```

```
L = R = vi(n, -1);
rep(i,0,n) rep(j,0,n) {
    if (R[j] != -1) continue;
    if (zero(cost[i][j] - u[i] - v[j])) {
        L[i] = j;
        R[j] = i;
        mated++;
        break;
    }
}
```

```
for (; mated < n; mated++) { // until solution is feasible
    int s = 0;
    while (L[s] != -1) s++;
    fill(all(dad), -1);
    fill(all(seen), 0);
    rep(k,0,n)
        dist[k] = cost[s][k] - u[s] - v[k];
```

```
int j = 0;
for (;;) {
    j = -1;
    rep(k,0,n) {
        if (seen[k]) continue;
        if (j == -1 || dist[k] < dist[j]) j = k;
    }
    seen[j] = 1;
    int i = R[j];
    if (i == -1) break;
    rep(k,0,n) {
        if (seen[k]) continue;
        auto new_dist = dist[j] + cost[i][k] - u[i] - v[k];
        if (dist[k] > new_dist) {
            dist[k] = new_dist;
            dad[k] = j;
        }
    }
}
```

```
rep(k,0,n) {
    if (k == j || !seen[k]) continue;
```



```

    auto w = dist[k] - dist[j];
    v[k] += w, u[R[k]] -= w;
}
u[s] += dist[j];

while (dad[j] >= 0) {
    int d = dad[j];
    R[j] = R[d];
    L[R[j]] = j;
    j = d;
}
R[j] = s;
L[s] = j;
}
auto value = vd(1)[0];
rep(i,0,n) value += cost[i][L[i]];
return value;
}

```

7.14 GeneralMatching

GeneralMatching.h

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

```

"../numerical/MatrixInverse-mod.h" bb8be4, 40 lines
vector<pii> generalMatching(int N, vector<pii>& ed) {
    vector<vector<ll>> mat(N, vector<ll>(N)), A;
    trav(pa, ed) {
        int a = pa.first, b = pa.second, r = rand() % mod;
        mat[a][b] = r, mat[b][a] = (mod - r) % mod;
    }

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

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

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

```

7.15 DFS algorithms

SCC.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.

Usage: `scc(graph, [&](vi& v) { ... })` visits all components in reverse topological order. `comp[i]` holds the component index of a node (a component only has edges to components with lower index). `ncomps` will contain the number of components.

Time: $\mathcal{O}(E + V)$ bb2963, 24 lines

```

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

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

```

7.16 BiconnectedComponents

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: $\mathcal{O}(E + V)$ cca7e6, 33 lines

```

vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
int dfs(int at, int par, F& f) {
    int me = num[at] = ++Time, e, y, top = me;
    trav(pa, ed[at]) if (pa.second != par) {
        tie(y, e) = pa;
        if (num[y]) {
            top = min(top, num[y]);
            if (num[y] < me)
                st.push_back(e);
        } else {
            int si = sz(st);
            int up = dfs(y, e, f);
            top = min(top, up);

```

```

        if (up == me) {
            st.push_back(e);
            f(vi(st.begin() + si, st.end()));
            st.resize(si);
        }
        else if (up < me) st.push_back(e);
        else { /* e is a bridge */ }
    }
    return top;
}

```

```

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

```

7.17 2sat

2sat.h

Description: Calculates a valid assignment to boolean variables a, b, c, \dots to a 2-SAT problem, so that an expression of the type $(a \vee b) \wedge (a \vee c) \wedge (d \vee b) \wedge \dots$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions ($\sim x$).

Usage: `TwoSat ts(number of boolean variables);`
`ts.either(0, ~3);` // Var 0 is true or var 3 is false
`ts.set_value(2);` // Var 2 is true
`ts.at_most_one({0, ~1, 2});` // ≤ 1 of vars 0, ~1 and 2 are true
`ts.solve();` // Returns true iff it is solvable
`ts.values[0..N-1]` holds the assigned values to the vars
Time: $\mathcal{O}(N + E)$, where N is the number of boolean variables, and E is the number of clauses. 0911c1, 56 lines

```

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

```

`TwoSat(int n = 0) : N(n), gr(2*n) {}`

```

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

```

```

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

```

`void set_value(int x) { either(x, x); }`

```

    void at_most_one(const vi& li) { // (optional)
        if (sz(li) <= 1) return;
        int cur = ~li[0];
        rep(i,2,sz(li)) {
            int next = add_var();
            either(cur, ~li[i]);
            either(cur, next);
            either(~li[i], next);
            cur = ~next;
        }
        either(cur, ~li[1]);
    }
}

```

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

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

7.18 MaximalCliques

MaximalCliques.h
Description: Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Callback is given a bitset representing the maximal clique.
Time: $\mathcal{O}\left(3^{n/3}\right)$, much faster for sparse graphs

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

7.19 MaximumClique

MaximumClique.h
Description: Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). Can be used to find a maximum independent set by finding a clique of the complement graph.
Time: Runs in about 1s for n=155 and worst case random graphs (p=.90). Runs faster for sparse graphs.

```
typedef vector<bitset<200>> vb;
struct MaxClique {
    double limit=0.025, pk=0;
    struct Vertex { int i, d=0; };
    typedef vector<Vertex> vv;
    vb e;
    vv V;
    vector<vi> C;
    vi qmax, q, S, old;
    void init(vv& r) {
        trav(v,r) v.d = 0;
        trav(v, r) trav(j, r) v.d += e[v.i][j.i];
        sort(all(r), [](auto a, auto b) { return a.d > b.d; });
    }
};
```

```
int mxD = r[0].d;
rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
}

void expand(vv& R, int lev = 1) {
    S[lev] += S[lev - 1] - old[lev];
    old[lev] = S[lev - 1];
    while (sz(R)) {
        if (sz(q) + R.back().d <= sz(qmax)) return;
        q.push_back(R.back().i);
        vv T;
        trav(v,R) if (e[R.back().i][v.i]) T.push_back({v.i});
        if (sz(T)) {
            if (S[lev]++ / ++pk < limit) init(T);
            int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1, 1);
            C[1].clear(), C[2].clear();
            trav(v, T) {
                int k = 1;
                auto f = [&](int i) { return e[v.i][i]; };
                while (any_of(all(C[k]), f)) k++;
                if (k > mxk) mxk = k, C[mxk + 1].clear();
                if (k < mnk) T[j++] .i = v.i;
                C[k].push_back(v.i);
            }
            if (j > 0) T[j - 1].d = 0;
            rep(k,mnk,mxk + 1) trav(i, C[k])
                T[j].i = i, T[j++] .d = k;
            expand(T, lev + 1);
        } else if (sz(q) > sz(qmax)) qmax = q;
        q.pop_back(), R.pop_back();
    }
}

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

7.20 MaximumIndependentSet

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.

7.21 TreePower

TreePower.h
Description: Calculate power of two jumps in a tree, to support fast upward jumps and LCAs. Assumes the root node points to itself.
Time: construction $\mathcal{O}(N \log N)$, queries $\mathcal{O}(\log N)$

```
vector<vi> treeJump(vi& P) {
    int on = 1, d = 1;
    while(on < sz(P)) on *= 2, d++;
    vector<vi> jmp(d, P);
    rep(i,1,d) rep(j,0,sz(P))
        jmp[i][j] = jmp[i-1][jmp[i-1][j]];
    return jmp;
}

int jmp(vector<vi>& tbl, int nod, int steps){
    rep(i,0,sz(tbl))
        if(steps&(1<<i)) nod = tbl[i][nod];
    return nod;
}

int lca(vector<vi>& tbl, vi& depth, int a, int b) {
```

```
    if (depth[a] < depth[b]) swap(a, b);
    a = jmp(tbl, a, depth[a] - depth[b]);
    if (a == b) return a;
    for (int i = sz(tbl); i--;) {
        int c = tbl[i][a], d = tbl[i][b];
        if (c != d) a = c, b = d;
    }
    return tbl[0][a];
}
```

7.22 LCA

LCA.h
Description: Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected. Can also find the distance between two nodes.
Usage: LCA lca(undirGraph);
lca.query(firstNode, secondNode);
lca.distance(firstNode, secondNode);
Time: $\mathcal{O}(N \log N + Q)$

```
"/>
```

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

```
"LCA.h" dabd75, 20 lines
vpi compressTree(LCA& lca, const vi& subset) {
    static vi rev; rev.resize(sz(lca.dist));
    vi li = subset, &T = lca.time;
    auto cmp = [&](int a, int b) { return T[a] < T[b]; };
    sort(all(li), cmp);
    int m = sz(li)-1;
    rep(i,0,m) {
        int a = li[i], b = li[i+1];
        li.push_back(lca.query(a, b));
    }
    sort(all(li), cmp);
    li.erase(unique(all(li)), li.end());
    rep(i,0,sz(li)) rev[li[i]] = i;
    vpi ret = {pii(0, li[0])};
    rep(i,0,sz(li)-1) {
        int a = li[i], b = li[i+1];
        ret.emplace_back(rev[lca.query(a, b)], b);
    }
    return ret;
}
```

7.24 HLD

HLD.h

Description: Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most $\log(n)$ light edges. The function of the HLD can be changed by modifying T, LOW and f. f is assumed to be associative and commutative.

Usage: HLD hld(G);
hld.update(index, value);
tie(value, lca) = hld.query(n1, n2);
"../data-structures/SegmentTree.h" d952a9, 93 lines

```
typedef vector<pii> vpi;

struct Node {
    int d, par, val, chain = -1, pos = -1;
};

struct Chain {
    int par, val;
    vector<int> nodes;
    Tree tree;
};

struct HLD {
    typedef int T;
    const T LOW = -(1<<29);
    void f(T& a, T b) { a = max(a, b); }

    vector<Node> V;
    vector<Chain> C;

    HLD(vector<vpi>& g) : V(sz(g)) {
        dfs(0, -1, g, 0);
        trav(C, C) {
            c.tree = {sz(c.nodes), 0};
            for (int ni : c.nodes)
                c.tree.update(V[ni].pos, V[ni].val);
        }
    }

    void update(int node, T val) {
        Node& n = V[node]; n.val = val;
        if (n.chain != -1) C[n.chain].tree.update(n.pos, val);
    }
}
```

```
int pard(Node& nod) {
    if (nod.par == -1) return -1;
    return V[nod.chain == -1 ? nod.par : C[nod.chain].par].d;
}
```

```
// query all *edges* between n1, n2
pair<T, int> query(int i1, int i2) {
    T ans = LOW;
    while(i1 != i2) {
        Node n1 = V[i1], n2 = V[i2];
        if (n1.chain != -1 && n1.chain == n2.chain) {
            int lo = n1.pos, hi = n2.pos;
            if (lo > hi) swap(lo, hi);
            f(ans, C[n1.chain].tree.query(lo, hi));
            i1 = i2 = C[n1.chain].nodes[hi];
        } else {
            if (pard(n1) < pard(n2))
                n1 = n2, swap(i1, i2);
            if (n1.chain == -1)
                f(ans, n1.val), i1 = n1.par;
            else {
                Chain& c = C[n1.chain];
                f(ans, n1.pos ? c.tree.query(n1.pos, sz(c.nodes))
                    : c.tree.s[1]);
                i1 = c.par;
            }
        }
    }
    return make_pair(ans, i1);
}
```

```
// query all *nodes* between n1, n2
pair<T, int> query2(int i1, int i2) {
    pair<T, int> ans = query(i1, i2);
    f(ans.first, V[ans.second].val);
    return ans;
}
```

```
pii dfs(int at, int par, vector<vpi>& g, int d) {
    V[at].d = d; V[at].par = par;
    int sum = 1, ch, nod, sz;
    tuple<int,int,int> mx(-1,-1,-1);
    trav(e, g[at]){
        if (e.first == par) continue;
        tie(sz, ch) = dfs(e.first, at, g, d+1);
        V[e.first].val = e.second;
        sum += sz;
        mx = max(mx, make_tuple(sz, e.first, ch));
    }
    tie(sz, nod, ch) = mx;
    if (2*sz < sum) return pii(sum, -1);
    if (ch == -1) { ch = sz(C); C.emplace_back(); }
    V[nod].pos = sz(C[ch].nodes);
    V[nod].chain = ch;
    C[ch].par = at;
    C[ch].nodes.push_back(nod);
    return pii(sum, ch);
}
```

7.25 LinkCutTree

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

693483, 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 push_flip() {
        if (!flip) return;
        flip = 0; swap(c[0], c[1]);
        if (c[0]) c[0]->flip ^= 1;
        if (c[1]) c[1]->flip ^= 1;
    }
    int up() { return p ? p->c[1] == this : -1; }
    void rot(int i, int b) {
        int h = i ^ b;
        Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
        if ((y->p = p)) p->c[up()] = y;
        c[i] = z->c[i ^ 1];
        if (b < 2) {
            x->c[h] = y->c[h ^ 1];
            z->c[h ^ 1] = b ? x : this;
        }
        y->c[i ^ 1] = b ? this : x;
        fix(); x->fix(); y->fix();
        if (p) p->fix();
        swap(pp, y->pp);
    }
    void splay() {
        for (push_flip(); p; ) {
            if (p->p) p->p->push_flip();
            p->push_flip(); push_flip();
            int c1 = up(), c2 = p->up();
            if (c2 == -1) p->rot(c1, 2);
            else p->p->rot(c2, c1 != c2);
        }
    }
    Node* first() {
        push_flip();
        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));
        make_root(&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];
        make_root(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 make_root(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;
}
};

```

7.26 DirectedMST

DirectedMST.h

Description: Edmonds' algorithm for finding the weight of the minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.

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

"../data-structures/UnionFind.h" a69883, 48 lines

```

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

```

```

ll dmst(int n, int r, vector<Edge>& g) {
    UF uf(n);
    vector<Node*> heap(n);
    trav(e, g) heap[e.b] = merge(heap[e.b], new Node{e});
    ll res = 0;
    vi seen(n, -1), path(n);
    seen[r] = r;
    rep(s, 0, n) {
        int u = s, qi = 0, w;
        while (seen[u] < 0) {
            path[qi++] = u, seen[u] = s;
            if (!heap[u]) return -1;
            Edge e = heap[u]->top();

```

```

        heap[u]->delta -= e.w, pop(heap[u]);
        res += e.w, u = uf.find(e.a);
        if (seen[u] == s) {
            Node* cyc = 0;
            do cyc = merge(cyc, heap[w = path[--qi]]);
            while (uf.join(u, w));
            u = uf.find(u);
            heap[u] = cyc, seen[u] = -1;
        }
    }
    return res;
}

```

Geometry (8)

8.1 Geometric primitives

Point.h

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

47ec0a, 28 lines

```

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

```

lineDistance.h

Description:

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

"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);

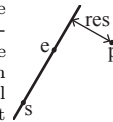
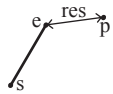
bool onSegment = segDist(a,b,p) < 1e-10;

"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;
}

```

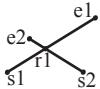


SegmentIntersection.h

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

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

```
bool operator<(Angle a, Angle b) {
    // add a.dist2() and b.dist2() to also compare distances
    return make_tuple(a.t, a.half(), a.y * (1l)b.x) <
        make_tuple(b.t, b.half(), a.x * (1l)b.y);
}
```

```
// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
    if (b < a) swap(a, b);
    return (b < a.t180() ?
        make_pair(a, b) : make_pair(b, a.t360()));
}
```

```
Angle operator+(Angle a, Angle b) { // point a + vector b
    Angle r(a.x + b.x, a.y + b.y, a.t);
    if (a.t180() < r) r.t--;
    return r.t180() < a ? r.t360() : r;
}
```

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

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;
}
```

circumcircle.h

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

```
"Point.h"
1caa3a, 9 lines
```

```
typedef Point<double> P;
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 $O(n)$

```
"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

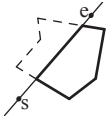
template<class T>
T polygonArea2(vector<Point<T>>& v) {
    T a = v.back().cross(v[0]);
    rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
    return a;
}
```

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" 26a0a9, 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)))
        trav(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/colinear points).

```
"Point.h", "sideOf.h", "OnSegment.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 colinear 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 colinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon: $\bullet(-1, -1)$ if no collision, $\bullet(i, -1)$ if touching the corner i , $\bullet(i, i)$ if along side $(i, i + 1)$, $\bullet(i, j)$ if crossing sides $(i, i + 1)$ and $(j, j + 1)$. In the last case, if a corner i is crossed, this is treated as happening on side $(i, i + 1)$. The points are returned in the same order as the line hits the polygon. extrVertex returns the point of a hull with the max projection onto a line.
Time: $\mathcal{O}(N + Q \log n)$

```
"Point.h" 758f22, 39 lines

typedef array<P, 2> Line;
#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
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(line[0].cross(poly[i], line[1]))
array<int, 2> lineHull(Line line, vector<P> poly) {
    int endA = extrVertex(poly, (line[0] - line[1]).perp());
    int endB = extrVertex(poly, (line[1] - line[0]).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" d31bbf, 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;
    trav(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, {(p - *lo).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 the box is wider than high (not best heuristic...)
        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: $O(n \log n)$

"Point.h" bf87ec, 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 {
    bool mark; Q o, rot; P p;
    P F() { return r()->p; }
    Q r() { return rot->rot; }
    Q prev() { return rot->o->rot; }
    Q next() { return r()->prev(); }
};
```

```
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 q[] = {new Quad{0,0,0,orig}, new Quad{0,0,0,arb},
             new Quad{0,0,0,dest}, new Quad{0,0,0,arb}};
    rep(i,0,4)
        q[i]->o = q[-i & 3], q[i]->rot = q[(i+1) & 3];
    return *q;
}
```

```
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 = 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.

1ec4d3, 6 lines

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

Point3D.h

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

8058ae, 32 lines

```
template<class T> struct Point3D {
    typedef Point3D P;
    typedef const P& R;
    T x, y, z;
    explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
}
```

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

3dHull.h

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

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

"Point3D.h" c172e9, 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);
            }
            trav(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. 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)

9.1 KMP

KMP.h

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

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

d4375c, 16 lines

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

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

9.2 Zfunc

Zfunc.h

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

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

3ae526, 12 lines

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

9.3 Manacher

Manacher.h

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

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

e7ad79, 13 lines

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

9.4 MinRotation

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.

Usage: rotate(v.begin(), v.begin()+min.rotation(v), v.end());

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

358164, 8 lines

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

9.5 SuffixArray

SuffixArray.h

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

38db9f, 23 lines

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

9.6 SuffixTree

SuffixTree.h

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

aae0b8, 50 lines

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

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

    SuffixTree(string a) : a(a) {
        fill(r, r+N, sz(a));
        memset(s, 0, sizeof s);
    }
};
```

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

// example: find longest common substring (uses ALPHA = 28)
pii best;
int lcs(int node, int i1, int i2, int olen) {
    if (l[node] <= i1 && i1 < r[node]) return 1;
    if (l[node] <= i2 && i2 < r[node]) return 2;
    int mask = 0, len = node ? olen + (r[node] - l[node]) : 0;
    rep(c, 0, ALPHA) if (t[node][c] != -1)
        mask |= lcs(t[node][c], i1, i2, len);
    if (mask == 3)
        best = max(best, {len, r[node] - len});
    return mask;
}

static pii LCS(string s, string t) {
    SuffixTree st(s + (char)('z' + 1) + t + (char)('z' + 2));
    st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
    return st.best;
}
};
```

9.7 Hashing

Hashing.h

Description: Self-explanatory methods for string hashing. acb5db, 44 lines

```
// Arithmetic mod 2^64-1. 2x slower than mod 2^64 and more
// code, but works on evil test data (e.g. Thue-Morse, where
// ABBA... and BAAB... of length 2^10 hash the same mod 2^64).
// "typedef ull H;" instead if you think test data is random,
// or work mod 10^9+7 if the Birthday paradox is not a problem.
struct H {
    typedef uint64_t ull;
    ull x; H(ull x=0) : x(x) {}
#define OP(O,A,B) H operator O(H o) { ull r = x; asm \
    ("addq %%rdx, %0\n adcq $0,%0" : "+a"(r) : B); return r; }
    OP(+, "d"(o.x)) OP(*, "mul %1\n", "r"(o.x) : "rdx")
    H operator-(H o) { return *this + ~o.x; }
    ull get() const { return x + !~x; }
    bool operator==(H o) const { return get() == o.get(); }
    bool operator<(H o) const { return get() < o.get(); }
};
static const H C = (1l)1e11+3; // (order ~ 3e9; random also ok)
```

```
struct HashInterval {
    vector<H> ha, pw;
    HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
        pw[0] = 1;
        rep(i, 0, sz(str))
            ha[i+1] = ha[i] * C + str[i],
            pw[i+1] = pw[i] * C;
    }
    H hashInterval(int a, int b) { // hash [a, b)
        return ha[b] - ha[a] * pw[b - a];
    }
};

vector<H> getHashes(string& str, int length) {
    if (sz(str) < length) return {};
    H h = 0, pw = 1;
    rep(i, 0, length)
        h = h * C + str[i], pw = pw * C;
    vector<H> ret = {h};
    rep(i, length, sz(str)) {
```

```
        ret.push_back(h = h * C + str[i] - pw * str[i-length]);
    }
    return ret;
}

H hashString(string& s) { H h{}; trav(c, s) h=h*C+c; return h; }

9.8 AhoCorasick

AhoCorasick.h
Description: Aho-Corasick tree is used for multiple pattern matching. Initialize the tree with create(patterns). find(word) returns for each position the index of the longest word that ends there, or -1 if none. findAll(., word) finds all words (up to  $N\sqrt{N}$  many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input. Time: create is  $\mathcal{O}(26N)$  where  $N$  is the sum of length of patterns. find is  $\mathcal{O}(M)$  where  $M$  is the length of the word. findAll is  $\mathcal{O}(NM)$ . 716ac4, 67 lines

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

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

```
        res.push_back(N[n].end());
        // count += N[n].nmatches;
    }
    return res;
}

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

Various (10)

10.1 Intervals

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

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

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

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

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

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

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

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

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

10.2 TernarySearch

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

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

10.3 Pick’s Theorem

- Pick’s Theorem is a useful method for determining the area of any polygon whose vertices are points on a lattice, a regularly spaced array of points. While lattices may have points in different arrangements, this essay uses a square lattice to examine Pick’s Theorem.

- Boundary Point (B): a lattice point on the polygon (including vertices)

- Interior Point (I): a lattice point in the polygon’s interior region

- Area = B / 2 + I - 1

10.4 Optimization tricks

10.4.1 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.

10.4.2 Pragmas

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

FastMod.h
Description: Compute $a\%b$ about 4 times faster than usual, where b is constant but not known at compile time. Fails for b = 1.

```
typedef unsigned long long ull;
typedef __uint128_t L;
struct FastMod {
    ull b, m;
    FastMod(ull b) : b(b), m(ull)((L(1) << 64) / b) {}
    ull reduce(ull a) {
        ull q = (ull)((L(m) * a) >> 64), r = a - q * b;
```

```
        return r >= b ? r - b : r;
    }
};

745db2, 8 lines
```

BumpAllocator.h

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

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

SmallPtr.h

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

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

BumpAllocatorSTL.h

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

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

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

Unrolling.h

```
#define F {...; ++i;}
int i = from;
while (i&3 && i < to) F // for alignment, if needed
while (i + 4 <= to) { F F F F }
while (i < to) F
```

SIMD.h

Description: Cheat sheet of SSE/AVX intrinsics, for doing arithmetic on several numbers at once. Can provide a constant factor improvement of about 4, orthogonal to loop unrolling. Operations follow the pattern "_mm(256)?_name_(si(128|256)|epi(8|16|32|64)|pd|ps)". Not all are described here; grep for _mm_ in /usr/lib/gcc/*/4.9/include/ for more. If AVX is unsupported, try 128-bit operations, "emmintrin.h" and #define __SSE__ and __MMX__ before including it. For aligned memory use _mm_malloc(size, 32) or int buf[N] alignas(32), but prefer loadu/storeu.

```
#pragma GCC target ("avx2") // or sse4.1
#include "emmintrin.h"

typedef __m256i mi;
#define L(x) _mm256_loadu_si256((mi*)&(x))
```

```
// High-level/specific methods:
// load(u)?_si256, store(u)?_si256, setzero_si256, _mm_malloc
// blendv_(epi8|ps|pd)(z?y:x), movemask_epi8 (hibits of bytes)
// i32gather_epi32(addr, x, 4): map addr[] over 32-b parts of x
// sad_epu8: sum of absolute differences of u8, outputs 4xi64
// maddubs_epi16: dot product of unsigned i7's, outputs 16xi15
// madd_epi16: dot product of signed i16's, outputs 8xi32
// extractf128_si256(, i) (256->128), cvtsi128_si32 (128->lo32)
// permute2f128_si256(x,x,1) swaps 128-bit lanes
// shuffle_epi32(x, 3*64+2*16+1*4+0) == x for each lane
// shuffle_epi8(x, y) takes a vector instead of an imm

// Methods that work with most data types (append e.g. _epi32):
// set1, blend (i8?x:y), add, adds (sat.), mullo, sub, and/or,
// andnot, abs, min, max, sign(i,x), cmp(gt|eq), unpack(lo|hi)
```

```
int sumi32(mi m) { union {int v[8]; mi m;} u; u.m = m;
    int ret = 0; rep(i,0,8) ret += u.v[i]; return ret; }
mi zero() { return _mm256_setzero_si256(); }
mi one() { return _mm256_set1_epi32(-1); }
bool all_zero(mi m) { return _mm256_testz_si256(m, m); }
bool all_one(mi m) { return _mm256_testc_si256(m, one()); }
```

```
ll example_filteredDotProduct(int n, short* a, short* b) {
    int i = 0; ll r = 0;
    mi zero = _mm256_setzero_si256(), acc = zero;
    while (i + 16 <= n) {
        mi va = L(a[i]), vb = L(b[i]); i += 16;
        va = _mm256_and_si256(_mm256_cmpgt_epi16(vb, va), va);
        mi vp = _mm256_madd_epi16(va, vb);
        acc = _mm256_add_epi64(_mm256_unpacklo_epi32(vp, zero),
            _mm256_add_epi64(acc, _mm256_unpackhi_epi32(vp, zero)));
    }
    union {ll v[4]; mi m;} u; u.m = acc; rep(i,0,4) r += u.v[i];
    for (;i<n;++i) if (a[i] < b[i]) r += a[i]*b[i]; //<- equiv
    return r;
}
```