FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Team Reference Document

ACM-ICPC World Finals, 2018 April 15–20, Beijing

Contestants:
Filip Bialas
Richard Hladík
Václav Volhejn

Coach:
doc. Mgr. Zdeněk Dvořák, Ph.D.

Prologue

Many algorithms in this notebook were taken from the KACTL notebook (with minor modifications). Its original version is maintained at `https://github.com/kth-competitive-programming/kactl`.
The sources of this notebook are maintained at `https://github.com/trinerdi/icpc-notebook`. Authorship information for each source file and much more can be found there.

trinerdi/template.cpp

```
84  #include <bits/stdc++.h>
07  using namespace std;
e7  typedef long long ll;
5e  typedef long double ld;
3e  #define rep(i, a, n) for (int i = (a); i < (n); i++)
b4  #define per(i, a, n) for (int i = (n) - 1; i >= (a); i--)
7c  int main(void) {
6f      ios_base::sync_with_stdio(0);
7d  }
```

trinerdi/sc.sh

```
88  # Usage: e [-d] test1.in … testN.in
a6  # Compiles the solution (if it has changed since the last time) and runs it on test1.in … testN.in.
84  # If no tests are given, runs the solution on all files whose name contains 'in'.
8c  # If -d is given, compiles the solution with debug flags and runs it with valgrind.
3a  e(){
f7      t=$(basename `pwd`)
08      if [ "$1" = -d ]; then fl=-g; v=valgrind; d=$t-debug; shift
7c      else fl=-O2\ -fsanitize=address; v=; d=$t; fi
f7      if [ $# = 0 ]; then args=*in*
2c      else args="$@"; fi
4a      [ $t.cpp -nt $d ] && ( g++ -std=c++14 -lm -Wall -Wextra -Wno-sign-compare -Wshadow -DLOCAL $fl $t.cpp -o $d || return )
04      for i in $args; do echo $i:; $v ./$d < $i; done
7d  }
ab  create(){
73      for i in {a..z}; do  # Change z as needed
15          mkdir "$i" && cd "$i" || continue
16          cp -n ../template.cpp "$i".cpp; touch "$i".in1; cd ..
67      done
7d  }
```

trinerdi/vimrc

```
d0  syntax on
f8  filetype plugin indent on
70  set number
bb  set ts=4
b4  set mouse=a
5b  set sw=4
```

trinerdi/check.sh

```
8a  IFS=
bf  while read -r a; do
83      # Doesn't really work in general (multiline comments are broken, ...), but works well enough
f5      printf "%s  %s\n" "$(echo "$a" | sed -re 's/\s+|\/\/.*"//g' | md5sum | head -c2)" "$a"
67  done
```

Mathematics (text)

Equations

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

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

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

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.

Recurrences

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

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

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

Trigonometry

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

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

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

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

Geometry

Triangles

Side lengths: a,b,c
Semiperimeter: $p=\frac{a+b+c}{2}$
Area: $A=\sqrt{p(p-a)(p-b)(p-c)}$
Circumradius: $R=\frac{abc}{4A}$
Inradius: $r=\frac{A}{p}$
Length of median (divides triangle into two equal-area triangles):

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

Length of bisector (divides angles in two): $s_a=\sqrt{bc\left[1-\left(\frac{a}{b+c}\right)^2\right]}$

Law of sines: $\frac{\sin\alpha}{a}=\frac{\sin\beta}{b}=\frac{\sin\gamma}{c}=\frac{1}{2R}$
Law of cosines: $a^2=b^2+c^2-2bc\cos\alpha$
Law of tangents:

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

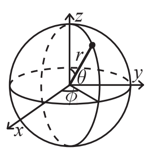
Quadrilaterals

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

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

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

Spherical coordinates



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

Derivatives/Integrals

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

Integration by parts:

$$\int_a^b f(x)g(x)\,\mathrm{d}x=[F(x)g(x)]_a^b-\int_a^b F(x)g'(x)\,\mathrm{d}x$$

Sums

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

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

Series

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

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

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

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

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

Probability theory

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

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

For independent X and Y,

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

Discrete distributions

Binomial distribution

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

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

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

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

First success distribution

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

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

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

Poisson distribution

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

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

Continuous distributions

Uniform distribution

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

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

Exponential distribution

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

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

Normal distribution

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

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

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

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

Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let $X_1, X_2, \ldots$ be a sequence of random variables generated by the Markov process. Then there is a transition matrix $\mathbf{P} = (p_{ij})$, with $p_{ij} = \Pr(X_n = i | X_{n-1} = j)$, and $\mathbf{p}^{(n)} = \mathbf{P}^n \mathbf{p}^{(0)}$ is the probability distribution for X_n (i.e., $p_i^{(n)} = \Pr(X_n = i)$), where $\mathbf{p}^{(0)}$ is the initial distribution.
π is a stationary distribution if $\pi = \pi\mathbf{P}$. If the Markov chain is *irreducible* (it is possible to get to any state from any state), then $\pi_i = \frac{1}{\mathbb{E}(T_i)}$ where $\mathbb{E}(T_i)$ is the expected time between two visits in state i. π_j/π_i is the expected number of visits in state j between two visits in state i.
For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors, π_i is proportional to node i's degree.
A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1). $\lim_{k\to\infty} \mathbf{P}^k = \mathbf{1}\pi$.
A Markov chain is an A-chain if the states can be partitioned into two sets $\mathbf{A}$ 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 $t_i = 1 + \sum_{k\in\mathbf{G}} p_{ki}t_k$.

Number-theoretical

Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

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

Estimates

$\sum_{d|n} d = O(n \log\log n)$.
The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200 000 for $n < 1e19$.

Combinatorial (text)

Permutations

Cycles

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

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

Involutions

An involution is a permutation with maximum cycle length 2, and it is its own inverse.

$$a(n) = a(n-1) + (n-1)a(n-2)$$
$$a(0) = a(1) = 1$$

1, 1, 2, 4, 10, 26, 76, 232, 764, 2620, 9496, 35696, 140152

Stirling numbers of the first kind

$$s(n, k) = (-1)^{n-k}c(n, k)$$

$c(n, k)$ is the unsigned Stirling numbers of the first kind, and they count the number of permutations on n items with k cycles.

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

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. $\pi(j) > j$.

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

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 $G = \mathbb{Z}_{\mathbb{n}}$ to get

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

Partitions and subsets

Partition function

Partitions of n with exactly k parts, $p(n, k)$, i.e., writing n as a sum of k positive integers, disregarding the order of the summands.

$$p(n, k) = p(n-1, k-1) + p(n-k, k)$$
$$p(0, 0) = p(1, n) = p(n, n) = p(n, n-1) = 1$$

For partitions with any number of parts, $p(n)$ obeys

$$p(0) = 1, \; p(n) = \sum_{k\in\mathbb{Z}\setminus\{\mathbb{0}\}}(-1)^{k+1}p(n - k(3k-1)/2)$$
$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

Bell numbers

Total number of partitions of n distinct elements.

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

The first are 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597. For a prime p

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

Triangles

Given rods of length $1, \dots, n$,

$$T(n) = \frac{1}{24}\begin{cases} n(n-2)(2n-5) & n \text{ even} \\ (n-1)(n-3)(2n-1) & n \text{ odd} \end{cases}.$$

is the number of distinct triangles (positive are) that can be constructed, i.e., the # of 3-subsets of $[n]$ s.t. $x \le y \le z$ and $z \ne x + y$.

General purpose numbers

Catalan numbers

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

$$C_{n+1} = \frac{2(2n+1)}{n+2}C_n$$

$$C_0 = 1, C_{n+1} = \sum C_i C_{n-i}$$

First few are 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900.

- # of monotonic lattice paths of a $n \times n$-grid which do not pass above the diagonal.
- # of expressions containing n pairs of parenthesis which are correctly matched.
- # of full binary trees with with $n+1$ leaves (0 or 2 children).
- # of non-isomorphic ordered trees with $n+1$ vertices.
- # of ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- # of permutations of $[n]$ with no three-term increasing subsequence.

Super Catalan numbers

The number of monotonic lattice paths of a $n \times n$-grid that do not touch the diagonal.

$$S(n) = \frac{3(2n-3)S(n-1) - (n-3)S(n-2)}{n}$$

$$S(1) = S(2) = 1$$

1, 1, 3, 11, 45, 197, 903, 4279, 20793, 103049, 518859

Motzkin numbers

Number of ways of drawing any number of nonintersecting chords among n points on a circle. Number of lattice paths from $(0,0)$ to $(n,0)$ never going below the x-axis, using only steps NE, E, SE.

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

$$M(0) = M(1) = 1$$

1, 1, 2, 4, 9, 21, 51, 127, 323, 835, 2188, 5798, 15511, 41835, 113634

Narayana numbers

Number of lattice paths from $(0,0)$ to $(2n,0)$ never going below the x-axis, using only steps NE and SE, and with k peaks.

$$N(n,k) = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}$$

$$N(n,1) = N(n,n) = 1$$

$$\sum_{k=1}^{n} N(n,k) = C_n$$

1, 1, 1, 1, 3, 1, 1, 6, 6, 1, 1, 10, 20, 10, 1, 1, 15, 50

Schröder numbers

Number of lattice paths from $(0,0)$ to (n,n) using only steps N,NE,E, never going above the diagonal. Number of lattice paths from $(0,0)$ to $(2n,0)$ using only steps NE, SE and double east EE, never going below the x-axis. Twice the Super Catalan number, except for the first term. 1, 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098

Data structures

Order statistics tree

Description: A set (not multiset!) with support for finding the k-th element, and finding the index of an element.
Time: $\mathcal{O}(\log N)$

```
f6 #include <bits/extc++.h>using namespace __gnu_pbds;
82 template <class T>
28 using Tree = tree<T, null_type, less<T>, rb_tree_tag,
4a       tree_order_statistics_node_update>;
7a void example() {
d5     Tree<int> t, t2; t.insert(8);
22     auto it = t.insert(10).first;
aa     assert(it == t.lower_bound(9));
00     assert(t.order_of_key(10) == 1);
30     assert(t.order_of_key(11) == 2);
f8     assert(*t.find_by_order(0) == 8);
27     t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
7d }
```

Lazy segment tree

Description: Lazy minimum segment tree supporting range updates and queries. Exclusive right bounds.
Time: $\mathcal{O}(\log N)$ per update/query

```
08 struct Segtree {
43     const ll INF = 1e18; // Neutral element of min
45     int l, r; // Exclusive righ
4b     ll val = 0, lazy = 0;
35     Segtree *lson = NULL, *rson = NULL;

66     Segtree (int _l, int _r) : l(_l), r(_r) {}
77     ~Segtree() { delete lson; delete rson; }

92     void unlazy() {
10         val += lazy;
65         if (l == r) return;
56         if(lson == NULL) {
27             int mid = (l+r)/2;
ef             lson = new Segtree(l, mid);
16             rson = new Segtree(mid, r);
7d         }
e9         lson->lazy += lazy; // <- Propagate
e9         rson->lazy += lazy;
37         lazy = 0;
7d     }

a6     void rangeUpdate(int fr, int to, ll x) {
45         unlazy();
47         if (fr >= r || l >= to) return;
46         if (fr <= l && to >= r) {
fd             lazy += x; // <- Add lazy value
45             unlazy();
71         } else {
b6             lson->rangeUpdate(fr, to, x);
f9             rson->rangeUpdate(fr, to, x);
36             val = min(lson->val, rson->val); // <- Combine from sons
7d         }
7d     }

88     ll rangeQuery(int fr, int to) {
40         if (fr >= r || l >= to) return INF;
45         unlazy();
46         if (fr <= l && to >= r) {
92             return val;
71         } else {
91             if (lson == NULL) return 0; // default value of `val`
49             return min(lson->rangeQuery(fr, to),  // <- Combine from sons
a1                        rson->rangeQuery(fr, to));
7d         }
7d     }
6c };
```

Persistent segment tree

Description: A segment tree whose updates do not invalidate previous versions. For N elements, call `new Segtree(0, N)`. Exclusive right bounds.
Time: $\mathcal{O}(\log N)$ per update/query

```
08 struct Segtree {
45     int l, r;
44     ll val = 0;
35     Segtree *lson = NULL, *rson = NULL;

e7     Segtree (int _l, int _r) : l(_l), r(_r) {
5a         if (r - l > 1) {
27             int mid = (l + r) / 2;
ef             lson = new Segtree(l, mid);
16             rson = new Segtree(mid, r);
7d         }
7d     }

77     ~Segtree() { delete lson; delete rson; }

96     Segtree* rangeUpdate(int fr, int to, ll x) {
68         if (fr >= r || l >= to) return this; // Range is not in the
                 segment
46         if (fr <= l && to >= r) { // Range is completely in the segment
a6             return new Segtree(l, r, val + x, lson, rson);
7d         }
7b         return new Segtree(l, r, val,
e6                            lson->rangeUpdate(fr, to, x),
7c                            rson->rangeUpdate(fr, to, x));
7d     }

a0     ll pointQuery(int i) const {
37         if (r - l == 1) return val;
27         int mid = (l + r) / 2;
0f         return val + ((i < mid) ? lson : rson)->pointQuery(i);
7d     }
32 private: // Constructor used internally, does not initialize children
27     Segtree(int _l, int _r, ll _val, Segtree* _lson, Segtree* _rson) :
ab         l(_l), r(_r), val(_val), lson(_lson), rson(_rson) {}
6c };
```

Union-find data structure

Description: Disjoint-set data structure. Can also get size of an element's component.
Time: $\mathcal{O}(\alpha(N))$

```
f1 struct UF {
a9     vector<int> e;
71     UF(int n) : e(n, -1) {}
24     bool same_set(int a, int b) { return find(a) == find(b); }
96     int size(int x) { return -e[find(x)]; }
59     int find(int x) { return e[x] < 0 ? x : e[x] = find(e[x]); }
90     void join(int a, int b) {
ed         a = find(a), b = find(b);
e0         if (a == b) return;
5d         if (e[a] > e[b]) swap(a, b);
7b         e[a] += e[b]; e[b] = a;
7d     }
6c };
```

Matrix

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;
```

```
1d template <class T, int N> struct Matrix {
4c     typedef Matrix M;
a1     array<array<T, N>, N> d{};
2e     M operator*(const M& m) const {
2e         M a;
2a         rep(i,0,N) rep(j,0,N)
71             rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k][j];
84         return a;
7d     }
82     vector<T> operator*(const vector<T>& vec) const {
f0         vector<T> ret(N);
dc         rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];
9e         return ret;
7d     }
10     M operator^(ll p) const {
6c         assert(p >= 0);
0b         M a, b(*this);
a6         rep(i,0,N) a.d[i][i] = 1;
a5         while (p) {
57             if (p&1) a = a*b;
16             b = b*b;
9d             p >>= 1;
7d         }
84         return a;
7d     }
6c };
```

Line container

Description: Container where you can add lines of the form $kx+m$, and query maximum values at points x. Useful for dynamic programming. Assumes k, m, x are positive.
Time: $\mathcal{O}(\log N)$

```
a2 bool Q;
0a struct Line {
04     mutable ll k, m, p;
c4     bool operator<(const Line& o) const {
e7         return Q ? p < o.p : k < o.k;
7d     }
6c };

09 struct LineContainer : multiset<Line> {
       // (for doubles, use inf = 1/.0, div(a,b) = a/b)
c7     const ll inf = LLONG_MAX;
87     ll div(ll a, ll b) { // floored division
1a         return a / b - ((a ^ b) < 0 && a % b); }
5d     bool isect(iterator x, iterator y) {
25         if (y == end()) { x->p = inf; return false; }
51         if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
d5         else x->p = div(y->m - x->m, x->k - y->k);
f1         return x->p >= y->p;
7d     }
a7     void add(ll k, ll m) {
70         auto z = insert({k, m, 0}), y = z++, x = y;
b9         while (isect(y, z)) z = erase(z);
39         if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
a2         while ((y = x) != begin() && (--x)->p >= y->p)
3d             isect(x, erase(y));
7d     }
da     ll query(ll x) {
b8         assert(!empty());
52         Q = 1; auto l = *lower_bound({0,0,x}); Q = 0;
a0         return l.k * x + l.m;
7d     }
6c };
```

Fast line container

Description: Container where you can add lines of the form $kx+m$, and query maximum values at points x. Assumes that the ks of the added lines are non-decreasing. Use `sorted_query()` for cases where x is non-decreasing.
Time: amortized $\mathcal{O}(1)$ per `add()` and `sorted_query()`, $\mathcal{O}(\log n)$ per `query()`

```
0a struct Line {
04     mutable ll k, m, p; // p is the position from which the line is
                                optimal
19     ll val(ll x) const { return k*x + m; }
b4     bool operator<(const Line& o) const { return p < o.p; }
6c };
99 ll floordiv (ll a, ll b) {
46     return a / b - ((a^b) < 0 && a % b);
7d }
   // queries and line intersections should be in range (-INF, INF)
a9 const ll INF = 1e17;
ab struct LineContainer : vector<Line> {
cd     ll isect(const Line& a, const Line& b) {
35         if (a.k == b.k) return a.m > b.m ? (-INF) : INF;
e6         ll res = floordiv(b.m - a.m, a.k - b.k);
48         if (a.val(res) < b.val(res)) res++;
b1         return res;
7d     }
a7     void add(ll k, ll m) {
d5         Line a = {k,m,INF};
42         while(!empty() && isect(a, back()) <= back().p) pop_back();
03         a.p = empty() ? (-INF) : isect(a, back());
24         push_back(a);
7d     }
da     ll query(ll x) {
b8         assert(!empty());
31         return (--upper_bound(begin(), end(), Line({0,0,x})))->val(x);
7d     }
4e     int qi = 0;
f8     ll sorted_query(ll x) {
b8         assert(!empty());
9e         qi = min(qi, (int)size() - 1);
42         while(qi < size()-1 && (*this)[qi+1].p <= x) qi++;
98         return (*this)[qi].val(x);
7d     }
6c };
```

Fenwick Tree

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

```
c2 struct FT {
f9     vector<ll> s;
05     FT(int n) : s(n) {}
35     void update(int pos, ll dif) { // a[pos] += dif
b5         for (; pos < s.size(); pos |= pos + 1) s[pos] += dif;
7d     }
2d     ll query(int pos) { // sum of values in [0, pos)
29         ll res = 0;
49         for (; pos > 0; pos &= pos - 1) res += s[pos-1];
b1         return res;
7d     }
25     int lower_bound(ll sum) {// min pos st sum of [0, pos] >= sum
           // Returns n if no sum is >= sum, or -1 if empty sum is.
bb         if (sum <= 0) return -1;
76         int pos = 0;
e4         for (int pw = 1 << 25; pw; pw >>= 1) {
bc             if (pos + pw <= s.size() && s[pos + pw-1] < sum)
aa                 pos += pw, sum -= s[pos-1];
7d         }
e8         return pos;
7d     }
6c };
```

2D Fenwick tree

Description: Computes sums $a[i \ldots j]$ for all $i < I$, $j < J$, and increases single elements $a[i \ldots j]$. 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 $\mathcal{O}(\log N)$.)

```
37 #include "FenwickTree.h"
12 struct FT2 {
1f     vector<vector<int>> ys; vector<FT> ft;
1f     FT2(int limx) : ys(limx) {}
f6     void fakeUpdate(int x, int y) {
4d         for (; x < ys.size(); x |= x + 1) ys[x].push_back(y);
7d     }
d3     void init() {
e5         for(auto& v : ys) sort(v.begin(), v.end()),
                               ft.emplace_back(v.size());
7d     }
18     int ind(int x, int y) {
75         return (int)(lower_bound(ys[x].begin(), ys[x].end(), y) -
                               ys[x].begin()); }
33     void update(int x, int y, ll dif) {
3e         for (; x < ys.size(); x |= x + 1)
5a             ft[x].update(ind(x, y), dif);
7d     }
```

```
89     ll query(int x, int y) {
42         ll sum = 0;
e3         for (; x; x &= x - 1)
43             sum += ft[x-1].query(ind(x-1, y));
ae         return sum;
7d     }
6c };
```

RMQ

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

```
6e #ifndef RMQ_HAVE_INFconst int inf = numeric_limits<int>::max();
8c #endif
82 template <class T>
3c struct RMQ {
5a     vector<vector<T>> jmp;
e8     RMQ(const vector<T>& V) {
c9         int N = V.size(), on = 1, depth = 1;
4f         while (on < V.size()) on *= 2, depth++;
21         jmp.assign(depth, V);
ff         rep(i,0,depth-1) rep(j,0,N)
91             jmp[i+1][j] = min(jmp[i][j],
39             jmp[i][min(N - 1, j + (1 << i))]);
7d     }
55     T query(int a, int b) {
fa         if (b <= a) return inf;
e3         int dep = 31 - __builtin_clz(b - a);
e1         return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
7d     }
6c };
```

Treap

Description: Binary search tree supporting k-th smallest element queries and getting indices of elements. Higher weights are higher in the heap.
Time: $\mathcal{O}(\log N)$ per query, with a fairly large constant.
Usage:
```
Treap* t = NULL;
insert(t, 5); insert(t, 2);
assert(getKth(t, 1) == 5);
```

```
3d #define SIZE(t) ((t) ? (t)->size : 0)
32 struct Treap {
b0     Treap *lson = NULL, *rson = NULL;
54     ll val;
b1     int weight, size = 0;
64     Treap(ll _val) : val(_val), weight(rand()), size(1) {}
b2     ~Treap() { delete lson; delete rson; }
df     void setSon(bool right, Treap *newSon) {
b6         Treap *&upd = right ? rson : lson;
dd         upd = newSon;
74         size = 1 + SIZE(lson) + SIZE(rson);
7d     }
6c };
   // Warning: Mutates l, r. All of l must be lower than r.
52 Treap* merge(Treap *l, Treap *r) {
1f     if (!l) return r;
0b     if (!r) return l;
22     if (l->weight > r->weight) {
b2         l->setSon(true, merge(l->rson, r));
70         return l;
71     } else {
57         r->setSon(false, merge(l, r->lson));
e2         return r;
7d     }
7d }
54 pair<Treap*, Treap*> split(Treap *a, ll val) { // Warning: Mutates a
50     if (!a) return {NULL, NULL};
c5     if (a->val <= val) {
0c         pair<Treap*, Treap*> res = split(a->rson, val);
99         a->setSon(true, res.first);
34         return {a, res.second};
71     } else {
76         pair<Treap*, Treap*> res = split(a->lson, val);
be         a->setSon(false, res.second);
45         return {res.first, a};
7d     }
7d }
5a void insert(Treap *&a, ll val) {
12     if (!a) {
df         a = new Treap(val);
71     } else {
1d         pair<Treap*, Treap*> spl = split(a, val);
26         a = merge(merge(spl.first, new Treap(val)), spl.second);
7d     }
7d }
9d void erase(Treap *&a, ll val) {
1d     pair <Treap *, Treap *> spl = split(a, val);
0e     pair <Treap *, Treap *> spl2 = split(spl.first, --val);
10     assert(spl2.second->size == 1);
b0     delete spl2.second;
c7     a = merge(spl2.first, spl.second);
7d }
db ll getKth(Treap *a, int k) { // zero-indexed
53     assert(k < a->size);
c7     while (true) {
05         int lsize = SIZE(a->lson);
1d         if (lsize == k)
11             return a->val;
1f         else if (lsize > k) a = a->lson;
40         else a = a->rson, k -= lsize + 1;
7d     }
7d }
```

Numerical

Polynomial

Description: A struct for operating on polynomials.

```
3b struct Polynomial {
11     int n; vector<double> a;
65     Polynomial(int n): n(n), a(n+1) {}
6f     double operator()(double x) const {
56         double val = 0;
d5         for(int i = n; i >= 0; --i) (val *= x) += a[i];
92         return val;
7d     }
7d     void derivative() {
b4         rep(i,1,n+1) a[i-1] = i*a[i];
58         a.pop_back(); --n;
7d     }
10     void divroot(double x0) {
2e         double b = a.back(), c; a.back() = 0;
39         for(int i=n--; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
b4         a.pop_back();
7d     }
6c };
```

Binary search

Description: Finds a zero point of f on the interval $[a, b]$. $f(a)$ must be less than 0 and $f(b)$ greater than 0. Useful for solving equations like $kx = \sin(x)$ as in the example below.

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

Usage:
```
double func(double x) { return .23*x-sin(x); }
double x0 = bs(0,4,func);
```

```
7c double bs(double a, double b, double (*f)(double)) {
       //for(int i = 0; i < 60; ++i){
24     while (b-a > 1e-6) {
22         double m = (a+b)/2;
d7         if (f(m) > 0) b = m;
d8         else a = m;
7d     }
84     return a;
7d }
```

Golden section search

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 `eps`. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.

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

Usage:
```
double func(double x) { return 4+x+.3*x*x; }
double xmin = gss(-1000,1000,func);
```

```
37 double gss(double a, double b, double (*f)(double)) {
b6     double r = (sqrt(5)-1)/2, eps = 1e-7;
b7     double x1 = b - r*(b-a), x2 = a + r*(b-a);
fb     double f1 = f(x1), f2 = f(x2);
44     while (b-a > eps)
3b         if (f1 < f2) { //change to > to find maximum
e5             b = x2; x2 = x1; f2 = f1;
c5             x1 = b - r*(b-a); f1 = f(x1);
71         } else {
0b             a = x1; x1 = x2; f1 = f2;
18             x2 = a + r*(b-a); f2 = f(x2);
7d         }
84     return a;
7d }
```

Polynomial roots

Description: Finds the real roots to a polynomial.

Usage:
```
vector<double> roots; Polynomial p(2);
p.a[0] = 2; p.a[1] = -3; p.a[2] = 1;
poly_roots(p,-1e10,1e10,roots); // x^2-3x+2=0
```

```
2e #include "Polynomial.h"
```

```
bf void poly_roots(const Polynomial& p, double xmin, double xmax,
                                   vector<double>& roots) {
95     if (p.n == 1) { roots.push_back(-p.a.front()/p.a.back()); }
3a     else {
6f         Polynomial d = p;
25         d.derivative();
1e         vector<double> dr;
49         poly_roots(d, xmin, xmax, dr);
0d         dr.push_back(xmin-1);
1b         dr.push_back(xmax+1);
5d         sort(dr.begin(), dr.end());
2c         for (auto i = dr.begin(), j = i++; i != dr.end(); j = i++){
10             double l = *j, h = *i, m, f;
36             bool sign = p(l) > 0;
1f             if (sign ^ (p(h) > 0)) {
                   //for(int i = 0; i < 60; ++i){
ff                 while(h - l > 1e-8) {
50                     m = (l + h) / 2, f = p(m);
d0                     if ((f <= 0) ^ sign) l = m;
ca                     else h = m;
7d                 }
50                 roots.push_back((l + h) / 2);
7d             }
7d         }
7d     }
7d }
```

Determinant

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

```
56 double det(vector<vector<double>>& a) {
f7     int n = a.size(); double res = 1;
8f     rep(i,0,n) {
d8         int b = i;
e5         rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
e8         if (i != b) swap(a[i], a[b]), res *= -1;
6b         res *= a[i][i];
c5         if (res == 0) return 0;
33         rep(j,i+1,n) {
c3             double v = a[j][i] / a[i][i];
52             if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
7d         }
7d     }
b1     return res;
7d }
```

Linear programming

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.
Time: $\mathcal{O}(NM \cdot \#pivots)$, where a pivot may be e.g. an edge relaxation. $xO(2^n)$ in the general case.
Usage:

```
vvd A = {{1,-1}, {-1,1}, {-1,-2}};
vd b = {1,1,-4}, c = {-1,-1}, x;
T val = LPSolver(A, b, c).solve(x);
```

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

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

06 struct LPSolver {
67     int m, n;
9f     vector<int> N, B;
cd     vvd D;

9e     LPSolver(const vvd& A, const vd& b, const vd& c) :
8f         m(b.size()), n(c.size()), N(n+1), B(m), D(m+2, vd(n+2)) {
3c             rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
1c             rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i];}
73             rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
36             N[n] = -1; D[m+1][n] = 1;
7d         }

cf     void pivot(int r, int s) {
7a         T *a = D[r].data(), inv = 1 / a[s];
e6         rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
90             T *b = D[i].data(), inv2 = b[s] * inv;
d2             rep(j,0,n+2) b[j] -= a[j] * inv2;
2c             b[s] = a[s] * inv2;
7d         }
f1         rep(j,0,n+2) if (j != s) D[r][j] *= inv;
3b         rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
1c         D[r][s] = inv;
a3         swap(B[r], N[s]);
7d     }

0d     bool simplex(int phase) {
47         int x = m + phase - 1;
5a         for (;;) {
b7             int s = -1;
1a             rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
8a             if (D[x][s] >= -eps) return true;
22             int r = -1;
40             rep(i,0,m) {
bc                 if (D[i][s] <= eps) continue;
e8                 if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
f8                              < MP(D[r][n+1] / D[r][s], B[r])) r = i;
7d             }
49             if (r == -1) return false;
62             pivot(r, s);
7d         }
7d     }

2c     T solve(vd &x) {
d4         int r = 0;
2b         rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
ba         if (D[r][n+1] < -eps) {
73             pivot(r, n);
61             if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
a1             rep(i,0,m) if (B[i] == -1) {
2f                 int s = 0;
1b                 rep(j,1,n+1) ltj(D[i]);
03                 pivot(i, s);
7d             }
7d         }
8f         bool ok = simplex(1); x = vd(n);
66         rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
79         return ok ? D[m][n+1] : inf;
7d     }
6c };
```

Linear equations

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

```
9a typedef vector<double> vd;
06 const double eps = 1e-12;

06 int solveLinear(vector<vd>& A, vd& b, vd& x) {
09     int n = A.size(), m = x.size(), rank = 0, br, bc;
64     if (n) assert(A[0].size() == m);
6e     vector<int> col(m); iota(col.begin(), col.end(), 0);

8f     rep(i,0,n) {
15         double v, bv = 0;
df         rep(r,i,n) rep(c,i,m)
e2             if ((v = fabs(A[r][c])) > bv)
16                 br = r, bc = c, bv = v;
8e         if (bv <= eps) {
74             rep(j,i,n) if (fabs(b[j]) > eps) return -1;
b9             break;
7d         }
f5         swap(A[i], A[br]);
fa         swap(b[i], b[br]);
8c         swap(col[i], col[bc]);
b7         rep(j,0,n) swap(A[j][i], A[j][bc]);
63         bv = 1/A[i][i];
33         rep(j,i+1,n) {
e9             double fac = A[j][i] * bv;
66             b[j] -= fac * b[i];
84             rep(k,i+1,m) A[j][k] -= fac*A[i][k];
7d         }
74         rank++;
7d     }

1d     x.assign(m, 0);
cf     for (int i = rank; i--;) {
3f         b[i] /= A[i][i];
4f         x[col[i]] = b[i];
38         rep(j,0,i) b[j] -= A[j][i] * b[i];
7d     }
e3     return rank; // (multiple solutions if rank < m)
7d }
```

Linear equations^{++}

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

```
9b #include "SolveLinear.h"

1f rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
   // ... then at the end:
28 x.assign(m, undefined);
e8 rep(i,0,rank) {
28     rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
65     x[col[i]] = b[i] / A[i][i];
1a fail:; }
```

Linear equations in $\mathbb{Z}_2$

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

```
24  typedef bitset<1000> bs;

a1  int solveLinear(vector<bs>& A, vector<int>& b, bs& x, int m) {
c1      int n = A.size(), rank = 0, br;
97      assert(m <= x.size());
6e      vector<int> col(m); iota(col.begin(), col.end(), 0);
8f      rep(i,0,n) {
c1          for (br=i; br<n; ++br) if (A[br].any()) break;
45          if (br == n) {
2f              rep(j,i,n) if(b[j]) return -1;
b9              break;
7d          }
8a          int bc = (int)A[br]._Find_next(i-1);
f5          swap(A[i], A[br]);
fa          swap(b[i], b[br]);
8c          swap(col[i], col[bc]);
45          rep(j,0,n) if (A[j][i] != A[j][bc]) {
17              A[j].flip(i); A[j].flip(bc);
7d          }
ef          rep(j,i+1,n) if (A[j][i]) {
b9              b[j] ^= b[i];
ba              A[j] ^= A[i];
7d          }
74          rank++;
7d      }

48      x = bs();
cf      for (int i = rank; i--;) {
a9          if (!b[i]) continue;
21          x[col[i]] = 1;
f5          rep(j,0,i) b[j] ^= A[j][i];
7d      }
e3      return rank; // (multiple solutions if rank < m)
7d  }
```

Matrix inversion

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 mod p, and k is doubled in each step.
Time: $\mathcal{O}(n^3)$

```
99  int matInv(vector<vector<double>>& A) {
04      int n = A.size(); vector<int> col(n);
3c      vector<vector<double>> tmp(n, vector<double>(n));
f2      rep(i,0,n) tmp[i][i] = 1, col[i] = i;

8f      rep(i,0,n) {
fd          int r = i, c = i;
96          rep(j,i,n) rep(k,i,n)
4b              if (fabs(A[j][k]) > fabs(A[r][c]))
c6                  r = j, c = k;
1c          if (fabs(A[r][c]) < 1e-12) return i;
7f          A[i].swap(A[r]); tmp[i].swap(tmp[r]);
08          rep(j,0,n)
d5              swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
ec          swap(col[i], col[c]);
97          double v = A[i][i];
33          rep(j,i+1,n) {
95              double f = A[j][i] / v;
0c              A[j][i] = 0;
49              rep(k,i+1,n) A[j][k] -= f*A[i][k];
3a              rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
7d          }
c2          rep(j,i+1,n) A[i][j] /= v;
56          rep(j,0,n) tmp[i][j] /= v;
ea          A[i][i] = 1;
7d      }

3b      for (int i = n-1; i > 0; --i) rep(j,0,i) {
bf          double v = A[j][i];
b7          rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
7d      }

46      rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
73      return n;
7d  }
```

FFT

Description: Fast Fourier transform. Also includes convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. a and b should be of roughly equal size. For convolutions of integers, rounding the results of conv works if $(|a| + |b|)\max(a, b) < \sim 10^9$ (in theory maybe 10^6); you may want to use an NTT from the Number Theory chapter instead.
Time: $\mathcal{O}(N \log N)$

```
2b  #include <valarray>

c7  typedef valarray<complex<double> > carray;
26  void fft(carray& x, carray& roots) {
39      int N = x.size();
8d      if (N <= 1) return;
e6      carray even = x[slice(0, N/2, 2)];
59      carray odd = x[slice(1, N/2, 2)];
31      carray rs = roots[slice(0, N/2, 2)];
26      fft(even, rs);
e0      fft(odd, rs);
e1      rep(k,0,N/2) {
e0          auto t = roots[k] * odd[k];
d1          x[k    ] = even[k] + t;
7f          x[k+N/2] = even[k] - t;
7d      }
7d  }

9a  typedef vector<double> vd;
49  vd conv(const vd& a, const vd& b) {
91      int s = a.size() + b.size() - 1, L = 32-__builtin_clz(s), n = 1<<L;
b9      if (s <= 0) return {};
67      carray av(n), bv(n), roots(n);
9a      rep(i,0,n) roots[i] = polar(1.0, -2 * M_PI * i / n);
f3      copy(a.begin(), a.end(), begin(av)); fft(av, roots);
93      copy(b.begin(), b.end(), begin(bv)); fft(bv, roots);
51      roots = roots.apply(conj);
c5      carray cv = av * bv; fft(cv, roots);
af      vd c(s); rep(i,0,s) c[i] = cv[i].real() / n;
5f      return c;
7d  }
```

Number theory

Fast exponentiation

Description: Returns a^p % mod.
Time: $\mathcal{O}(\log p)$

```
c7  ll fastexp(ll a, ll p, ll mod) {
23      a = ((a % mod) + mod) % mod;
53      ll res = 1 % mod;
cc      for (; p; a = (a * a) % mod, p /= 2)
e4          if (p % 2)
fa              res = (res * a) % mod;
b1      return res;
7d  }
```

Primality test

Description: Deterministic Miller-Rabin primality test, works for $p \leq 2^{32}$.
Time: $\mathcal{O}(\log p)$

```
38  #include "fastexp.cpp"
94  bool isprime(ll p) {
83      vector<ll> wit = {2, 7, 61};
        // For p < 1e18, use 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37
        // plus KACTL mod_pow (which can multiply modulo m ≤ 1e18)
2f      if (count(wit.begin(), wit.end(), p))
35          return true;
a6      if (p < 2 || !(p % 2))
fb          return false;
ae      int cnt = 0;
53      ll d = p - 1;
d2      while (d % 2 == 0)
7e          d /= 2, cnt++;
99      for (ll a: wit) {
f7          bool passed = false;
22          ll ad = fastexp(a, d, p);
6e          passed |= ad == 1;
24          for (int i = 0; i < cnt; i++, ad = (ad * ad) % p)
48              passed |= ad == p - 1;
53          if (!passed)
fb              return false;
7d      }
35      return true;
7d  }
```

Sieve of Eratosthenes

Description: Prime sieve for generating all primes up to a certain limit. isprime$[i]$ is true iff i is a prime.
Time: lim=100'000'000 $\approx$ 0.8 s. Runs 30% faster if only odd indices are stored.

```
39  const int MAX_PR = 5000000;
9e  bitset<MAX_PR> isprime;
00  vector<int> eratosthenes_sieve(int lim) {
b2      isprime.set(); isprime[0] = isprime[1] = 0;
f7      for (int i = 4; i < lim; i += 2) isprime[i] = 0;
cf      for (int i = 3; i*i < lim; i += 2) if (isprime[i])
8c          for (int j = i*i; j < lim; j += i*2) isprime[j] = 0;
c0      vector<int> pr;
e2      rep(i,2,lim) if (isprime[i]) pr.push_back(i);
83      return pr;
7d  }
```

Extended Euclid's Algorithm

Description: Finds the Greatest Common Divisor to the integers a and b. Euclid also finds two integers x and y, such that $ax + by = \gcd(a, b)$. If a and b are coprime, then x is the inverse of $a \pmod{b}$.

```
f9  ll euclid(ll a, ll b, ll &x, ll &y) {
84      if (b) { ll d = euclid(b, a % b, y, x);
c0          return y -= a/b * x, d; }
a3      return x = 1, y = 0, a;
7d  }
```

Modular arithmetic

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

```
#include "euclid.h"

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

Modular inverse (precomputation)

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

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

Modular multiplication for ll

Description: Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$) for large c.
Time: $\mathcal{O}(64/bits \cdot \log b)$, where $bits = 64 - k$, if we want to deal with k-bit numbers.

```
typedef unsigned long long ull;
const int bits = 10;
// if all numbers are less than 2^k, set bits = 64-k
const ull po = 1 << bits;
ull mod_mul(ull a, ull b, ull &c) {
    ull x = a * (b & (po - 1)) % c;
    while ((b >>= bits) > 0) {
        a = (a << bits) % c;
        x += (a * (b & (po - 1))) % c;
    }
    return x % c;
}
ull mod_pow(ull a, ull b, ull mod) {
    if (b == 0) return 1;
    ull res = mod_pow(a, b / 2, mod);
    res = mod_mul(res, res, mod);
    if (b & 1) return mod_mul(res, a, mod);
    return res;
}
```

Modular square roots

Description: Tonelli-Shanks algorithm for modular square roots.
Time: $\mathcal{O}(\log^2 p)$ worst case, often $\mathcal{O}(\log p)$

```
#include "../../trinerdi/number-theory/fastexp.cpp"

ll sqrt(ll a, ll p) {
    a %= p; if (a < 0) a += p;
    if (a == 0) return 0;
    assert(fastexp(a, (p-1)/2, p) == 1);
    if (p % 4 == 3) return fastexp(a, (p+1)/4, p);
    // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
    ll s = p - 1;
    int r = 0;
    while (s % 2 == 0)
        ++r, s /= 2;
    ll n = 2; // find a non-square mod p
    while (fastexp(n, (p - 1) / 2, p) != p - 1) ++n;
    ll x = fastexp(a, (s + 1) / 2, p);
    ll b = fastexp(a, s, p);
    ll g = fastexp(n, s, p);
    for (;;) {
        ll t = b;
        int m = 0;
        for (; m < r; ++m) {
            if (t == 1) break;
            t = t * t % p;
        }
        if (m == 0) return x;
        ll gs = fastexp(g, 1 << (r - m - 1), p);
        g = gs * gs % p;
        x = x * gs % p;
        b = b * g % p;
        r = m;
    }
}
```

Discrete logarithm

Description: Return a possible $\log_A B \bmod P$ or -1 if none exists. P must be prime and $1 < A < P < 2^{31}$.
Time: $\mathcal{O}(\sqrt{P} \log P)$

```
#include "../../base.hpp"
#include "../../number-theory/fastexp.cpp"

ll dlog(ll A, ll B, ll P) {
  ll M = (ll)ceil(sqrt(P-1.0));
  vector< pair<ll, int> > P1, P2;
  ll pom = fastexp(A,M,P);
  P1.push_back(make_pair(1,0));
  for (int i=1; i<M; i++) P1.push_back(make_pair( (P1[i-1].first * pom)%P
                                ,i));
  sort(P1.begin(), P1.end());
  ll Ainv = fastexp(A,P-2,P);
  P2.push_back(make_pair(B,0));
  for (int i=1; i<M; i++) P2.push_back(make_pair( (P2[i-1].first *
                                Ainv)%P, i));
  sort(P2.begin(), P2.end());
  int i,j;
  for (i=0, j=0; P1[i].first != P2[j].first; ) {
    if (P1[i].first < P2[j].first) i++; else j++;
    if ( i==M || j==M ) return -1;
  }
  return ( M * P1[i].second + P2[j].second ) % (P-1);
}
```

NTT

Description: Number theoretic transform. Can be used for convolutions modulo specific nice primes of the form $2^a b + 1$, where the convolution result has size at most 2^a. For other primes/integers, use two different primes and combine with CRT. May return negative values.
Time: $\mathcal{O}(N \log N)$

```
#include "../../trinerdi/number-theory/fastexp.cpp"

const ll mod = (119 << 23) + 1, root = 3; // = 998244353
// For p < 2^30 there is also e.g. (5 << 25, 3), (7 << 26, 3),
// (479 << 21, 3) and (483 << 21, 5). The last two are > 10^9.

typedef vector<ll> vl;
void ntt(ll* x, ll* temp, ll* roots, int N, int skip) {
    if (N == 1) return;
    int n2 = N/2;
    ntt(x     , temp, roots, n2, skip*2);
    ntt(x+skip, temp, roots, n2, skip*2);
    rep(i,0,N) temp[i] = x[i*skip];
    rep(i,0,n2) {
        ll s = temp[2*i], t = temp[2*i+1] * roots[skip*i];
        x[skip*i] = (s + t) % mod; x[skip*(i+n2)] = (s - t) % mod;
    }
}
void ntt(vl& x, bool inv = false) {
    ll e = fastexp(root, (mod-1) / x.size(), mod);
    if (inv) e = fastexp(e, mod-2, mod);
    vl roots(x.size(), 1), temp = roots;
    rep(i,1,x.size()) roots[i] = roots[i-1] * e % mod;
    ntt(&x[0], &temp[0], &roots[0], x.size(), 1);
}
vl conv(vl a, vl b) {
    int s = a.size() + b.size() - 1; if (s <= 0) return {};
    int L = s > 1 ? 32 - __builtin_clz(s - 1) : 0, n = 1 << L;
    if (s <= 200) { // (factor 10 optimization for |a|,|b| = 10)
        vl c(s);
        rep(i,0,a.size()) rep(j,0,b.size())
            c[i + j] = (c[i + j] + a[i] * b[j]) % mod;
        return c;
    }
    a.resize(n); ntt(a);
    b.resize(n); ntt(b);
    vl c(n); ll d = fastexp(n, mod-2, mod);
    rep(i,0,n) c[i] = a[i] * b[i] % mod * d % mod;
    ntt(c, true); c.resize(s); return c;
}
```

Factorization

Description: Pollard's rho algorithm. It is a probabilistic factorisation algorithm, whose expected time complexity is good. Before you start using it, run `init(bits)`, where bits is the length of the numbers you use. Returns factors of the input without duplicates.
Time: Expected running time should be good enough for 50-bit numbers.

```
#include "ModMulLL.h"
#include "../../trinerdi/number-theory/prime.cpp"
#include "eratosthenes.h"

vector<ull> pr;
ull f(ull a, ull n, ull &has) {
    return (mod_mul(a, a, n) + has) % n;
}
vector<ull> factor(ull d) {
    vector<ull> res;
    for (int i = 0; i < pr.size() && pr[i]*pr[i] <= d; i++)
        if (d % pr[i] == 0) {
            while (d % pr[i] == 0) d /= pr[i];
```

```
0e            res.push_back(pr[i]);
7d        }
       //d is now a product of at most 2 primes.
10     if (d > 1) {
55         if (isprime(d))
07             res.push_back(d);
97         else while (true) {
20             ull has = rand() % 2321 + 47;
ed             ull x = 2, y = 2, c = 1;
49             for (; c==1; c = __gcd((y > x ? y - x : x - y), d)) {
4d                 x = f(x, d, has);
df                 y = f(f(y, d, has), d, has);
7d             }
4f             if (c != d) {
c0                 res.push_back(c); d /= c;
a4                 if (d != c) res.push_back(d);
b9                 break;
7d             }
7d         }
7d     }
b1     return res;
7d }
b4 void init(int bits) {//how many bits do we use?
84     vector<int> p = eratosthenes_sieve(1 << ((bits + 2) / 3));
72     pr.assign(p.begin(), p.end());
7d }
```

Phi function

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

```
48 const int LIM = 5000000;
80 int phi[LIM];
01 void calculatePhi() {
59     rep(i,0,LIM) phi[i] = i&1 ? i : i/2;
3d     for(int i = 3; i < LIM; i += 2)
56         if(phi[i] == i)
8a             for(int j = i; j < LIM; j += i)
17                 (phi[j] /= i) *= i-1;
7d }
```

Chinese remainder theorem

Description: `chinese(a, m, b, n)` returns a number x, such that $x \equiv a \pmod{m}$ and $x \equiv b \pmod{n}$. For not coprime n, m, use `chinese_common`. Note that all numbers must be less than 2^{31} if you have Z = unsigned long long.
Time: $\log(m + n)$

```
ae #include "euclid.h"
e9 template <class Z> Z chinese(Z a, Z m, Z b, Z n) {
41     Z x, y; euclid(m, n, x, y);
d5     Z ret = a * (y + m) % m * n + b * (x + n) % n * m;
48     if (ret >= m * n) ret -= m * n;
9e     return ret;
7d }
6a template <class Z> Z chinese_common(Z a, Z m, Z b, Z n) {
77     Z d = gcd(m, n);
7b     if (((b -= a) %= n) < 0) b += n;
64     if (b % d) return -1; // No solution
8c     return d * chinese(Z(0), m/d, b/d, n/d) + a;
7d }
```

Combinatorics

Permutation serialization

Description: Permutations to/from integers. The bijection is order preserving.
Time: $\mathcal{O}(n^2)$

```
8a int factorial[] = {1, 1, 2, 6, 24, 120, 720, 5040}; // etc.
00 template <class Z, class It>
e5 void perm_to_int(Z& val, It begin, It end) {
bd     int x = 0, n = 0;
a2     for (It i = begin; i != end; ++i, ++n)
90         if (*i < *begin) ++x;
44     if (n > 2) perm_to_int<Z>(val, ++begin, end);
10     else val = 0;
a4     val += factorial[n-1]*x;
7d }
74 /* range [begin, end) does not have to be sorted. */
00 template <class Z, class It>
47 void int_to_perm(Z val, It begin, It end) {
bf     Z fac = factorial[end - begin - 1];
       // Note that the division result will fit in an integer!
ea     int x = val / fac;
f3     nth_element(begin, begin + x, end);
36     swap(*begin, *(begin + x));
0b     if (end - begin > 2) int_to_perm(val % fac, ++begin, end);
7d }
```

Derangements

Description: Generates the i-th derangement of S_n (in lexicographical order). (Derangement is a permutation with no fixed points.)

```
63 template <class T, int N>
e2 struct derangements {
ca     T dgen[N][N], choose[N][N], fac[N];
d7     derangements() {
ac         fac[0] = choose[0][0] = 1;
7a         memset(dgen, 0, sizeof(dgen));
94         rep(m,1,N) {
7e             fac[m] = fac[m-1] * m;
4b             choose[m][0] = choose[m][m] = 1;
86             rep(k,1,m)
21                 choose[m][k] = choose[m-1][k-1] + choose[m-1][k];
7d         }
7d     }
fa     T DGen(int n, int k) {
62         T ans = 0;
16         if (dgen[n][k]) return dgen[n][k];
1c         rep(i,0,k+1)
5c             ans += (i&1?-1:1) * choose[k][i] * fac[n-i];
d1         return dgen[n][k] = ans;
7d     }
7e     void generate(int n, T idx, int *res) {
e4         int vals[N];
62         rep(i,0,n) vals[i] = i;
8f         rep(i,0,n) {
16             int j, k = 0, m = n - i;
ad             rep(j,0,m) if (vals[j] > i) ++k;
7d             rep(j,0,m) {
3b                 T p = 0;
96                 if (vals[j] > i) p = DGen(m-1, k-1);
ad                 else if (vals[j] < i) p = DGen(m-1, k);
21                 if (idx <= p) break;
e7                 idx -= p;
7d             }
e5             res[i] = vals[j];
55             memmove(vals + j, vals + j + 1, sizeof(int)*(m-j-1));
7d         }
7d     }
6c };
```

Binomial coefficient

Description: The number of k-element subsets of an n-element set, $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
Time: $\mathcal{O}(\min(k, n - k))$

```
54 ll choose(int n, int k) {
9a     ll c = 1, to = min(k, n-k);
af     if (to < 0) return 0;
c9     rep(i,0,to) c = c * (n - i) / (i + 1);
5f     return c;
7d }
```

Binomial modulo prime

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

```
bf ll chooseModP(ll n, ll m, int p, vector<int>& fact, vector<int>& invfact)
                              {
2f     ll c = 1;
59     while (n || m) {
7a         ll a = n % p, b = m % p;
4f         if (a < b) return 0;
54         c = c * fact[a] % p * invfact[b] % p * invfact[a - b] % p;
b2         n /= p; m /= p;
7d     }
5f     return c;
7d }
```

Rolling binomial

Description: $\binom{n}{k}$ mod m in time proportional to the difference between (n, k) and the previous (n, k).

```
ca const ll mod = 1000000007;
e7 vector<ll> invs; // precomputed up to max n, inclusively
a8 struct Bin {
0d     int N = 0, K = 0;  ll r = 1;
9c     void m(ll a, ll b) { r = r * a % mod * invs[b] % mod; }
54     ll choose(int n, int k) {
96         if (k > n || k < 0) return 0;
1f         while (N < n) ++N, m(N, N-K);
10         while (K < k) ++K, m(N-K+1, K);
f8         while (K > k) m(K, N-K+1), --K;
cb         while (N > n) m(N-K, N), --N;
e2         return r;
7d     }
6c };
```

Multinomial

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

Time: $\mathcal{O}((\sum k_i) - k_1)$

```
e6 ll multinomial(vector<int>& v) {
b6     ll c = 1, m = v.empty() ? 1 : v[0];
09     rep(i,1,v.size()) rep(j,0,v[i])
3e         c = c * ++m / (j+1);
5f     return c;
7d }
```

Graphs and trees

Bellman–Ford

Description: Calculates shortest path in a graph that might have negative edge distances. Propagates negative infinity distances (sets $dist = -inf$), and returns true if there is some negative cycle. Unreachable nodes get $dist = inf$.

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

```
6d typedef ll T; // or whatever
67 struct Edge { int src, dest; T weight; };
50 struct Node { T dist; int prev; };
ab struct Graph { vector<Node> nodes; vector<Edge> edges; };

7e const T inf = numeric_limits<T>::max();
2c bool bellmanFord2(Graph& g, int start_node) {
98     for(auto& n : g.nodes) { n.dist = inf; n.prev = -1; }
27     g.nodes[start_node].dist = 0;

73     rep(i,0,g.nodes.size()) for(auto& e : g.edges) {
ca         Node& cur = g.nodes[e.src];
db         Node& dest = g.nodes[e.dest];
6c         if (cur.dist == inf) continue;
82         T ndist = cur.dist + (cur.dist == -inf ? 0 : e.weight);
a3         if (ndist < dest.dist) {
c6             dest.prev = e.src;
30             dest.dist = (i >= g.nodes.size()-1 ? -inf : ndist);
7d         }
7d     }
14     bool ret = 0;
73     rep(i,0,g.nodes.size()) for(auto& e : g.edges) {
d6         if (g.nodes[e.src].dist == -inf)
8e             g.nodes[e.dest].dist = -inf, ret = 1;
7d     }
9e     return ret;
7d }
```

Floyd–Warshall

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

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

```
5b const ll inf = 1LL << 62;
64 void floydWarshall(vector<vector<ll>>& m) {
3f     int n = m.size();
0b     rep(i,0,n) m[i][i] = min(m[i][i], {});
9b     rep(k,0,n) rep(i,0,n) rep(j,0,n)
9d         if (m[i][k] != inf && m[k][j] != inf) {
89             auto newDist = max(m[i][k] + m[k][j], -inf);
64             m[i][j] = min(m[i][j], newDist);
7d         }
a1     rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
3e         if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;
7d }
```

Topological sorting

Description: Topological sorting. Given is an oriented graph. Output is an ordering of vertices (array idx), such that there are edges only from left to right. The function returns false if there is a cycle in the graph.

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

```
55 template <class E, class I>
85 bool topo_sort(const E &edges, I &idx, int n) {
20     vector<int> indeg(n);
d4     rep(i,0,n)
ed         for(auto& e : edges[i])
07             indeg[e]++;
3b     queue<int> q; // use priority queue for lexic. smallest ans.
08     rep(i,0,n) if (indeg[i] == 0) q.push(-i);
6e     int nr = 0;
ad     while (q.size() > 0) {
57         int i = -q.front(); // top() for priority queue
09         idx[i] = nr++;
ec         q.pop();
ed         for(auto& e : edges[i])
0b             if (--indeg[e] == 0) q.push(-e);
7d     }
28     return nr == n;
7d }
```

Euler walk

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: $\mathcal{O}(E)$

```
87 struct V {
fd     vector<pair<int,int>> outs; // (dest, edge index)
ef     int nins = 0;
6c };
2f vector<int> euler_walk(vector<V>& nodes, int nedges, int src=0) {
58     int c = 0;
6c     for(auto& n : nodes) c += abs(n.nins - n.outs.size());
28     if (c > 2) return {};
02     vector<vector<pair<int,int>>::iterator> its;
86     for(auto& n : nodes)
d7         its.push_back(n.outs.begin());
d1     vector<bool> eu(nedges);
08     vector<int> ret, s = {src};
4f     while(!s.empty()) {
e2         int x = s.back();
c7         auto& it = its[x], end = nodes[x].outs.end();
26         while(it != end && eu[it->second]) ++it;
ab         if(it == end) { ret.push_back(x); s.pop_back(); }
f4         else { s.push_back(it->first); eu[it->second] = true; }
7d     }
5b     if(ret.size() != nedges+1)
45         ret.clear(); // No Eulerian cycles/paths.
       // else, non-cycle if ret.front() != ret.back()
18     reverse(ret.begin(), ret.end());
9e     return ret;
7d }
```

Goldberg's (push-relabel) algorithm

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

```
e9 typedef ll Flow;
b4 struct Edge {
1c     int dest, back;
a4     Flow f, c;
6c };
70 struct PushRelabel {
09     vector<vector<Edge>> g;
2a     vector<Flow> ec;
6e     vector<Edge*> cur;
31     vector<vector<int>> hs; vector<int> H;
e4     PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}
d3     void add_edge(int s, int t, Flow cap, Flow rcap=0) {
fa         if (s == t) return;
a5         Edge a = {t, g[t].size(), 0, cap};
ec         Edge b = {s, g[s].size(), 0, rcap};
a3         g[s].push_back(a);
e2         g[t].push_back(b);
7d     }
af     void add_flow(Edge& e, Flow f) {
16         Edge &back = g[e.dest][e.back];
45         if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
db         e.f += f; e.c -= f; ec[e.dest] += f;
85         back.f -= f; back.c += f; ec[back.dest] -= f;
7d     }
dd     Flow maxflow(int s, int t) {
42         int v = g.size(); H[s] = v; ec[t] = 1;
e3         vector<int> co(2*v); co[0] = v-1;
c8         rep(i,0,v) cur[i] = g[i].data();
d0         for(auto& e : g[s]) add_flow(e, e.c);
8e         for (int hi = 0;;) {
a9             while (hs[hi].empty()) if (!hi--) return -ec[s];
a4             int u = hs[hi].back(); hs[hi].pop_back();
ec             while (ec[u] > 0)  // discharge u
8e                 if (cur[u] == g[u].data() + g[u].size()) {
10                     H[u] = 1e9;
aa                     for(auto& e : g[u]) if (e.c && H[u] > H[e.dest]+1)
8b                         H[u] = H[e.dest]+1, cur[u] = &e;
e2                     if (++co[H[u]], !--co[hi] && hi < v)
6e                         rep(i,0,v) if (hi < H[i] && H[i] < v)
50                             --co[H[i]], H[i] = v + 1;
12                     hi = H[u];
7b                 } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
fa                     add_flow(*cur[u], min(ec[u], cur[u]->c));
25                 else ++cur[u];
7d         }
7d     }
6c };
```

Min-cost max-flow

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

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

```
46 #include <bits/extc++.h>
06 const ll INF = numeric_limits<ll>::max() / 4;
62 typedef vector<ll> VL;
```

```
struct MCMF {
    int N;
    vector<vector<int>> ed, red;
    vector<VL> cap, flow, cost;
    vector<int> seen;
    VL dist, pi;
    vector<pair<int,int>> 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(seen.begin(), seen.end(), 0);
        fill(dist.begin(), dist.end(), 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];
            for(auto& i : ed[s]) if (!seen[i])
                relax(i, cap[s][i] - flow[s][i], cost[s][i], 1);
            for(auto& 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(pi.begin(), pi.end(), INF); pi[s] = 0;
        int it = N, ch = 1; ll v;
        while (ch-- && it--)
            rep(i,0,N) if (pi[i] != INF)
                for(auto& 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
    }
};
```

Edmonds–Karp

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.

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

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

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

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

Min-cut

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.

Global min-cut

Description: Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.
Time: $\mathcal{O}(V^3)$

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

    for (int phase = N-1; phase >= 0; phase--) {
        vector<int> 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};
}
```

$\mathcal{O}(\sqrt{V}E)$ maximum matching (Hopkroft–Karp)

Description: Find a maximum matching in a bipartite graph. g must **only** contain edges from left to right.
Time: $\mathcal{O}(\sqrt{V}E)$
Usage: `vector<int> ba(m, -1); hopcroftKarp(g, ba);`

```
bool dfs(int a, int layer, const vector<vector<int>>& g, vector<int>&
                btoa,
        vector<int>& A, vector<int>& B) {
    if (A[a] != layer) return 0;
    A[a] = -1;
    for(auto& b : g[a]) if (B[b] == layer + 1) {
        B[b] = -1;
        if (btoa[b] == -1 || dfs(btoa[b], layer+2, g, btoa, A, B))
            return btoa[b] = a, 1;
    }
    return 0;
}

int hopcroftKarp(const vector<vector<int>>& g, vector<int>& btoa) {
    int res = 0;
    vector<int> A(g.size()), B(btoa.size()), cur, next;
    for (;;) {
        fill(A.begin(), A.end(), 0);
        fill(B.begin(), B.end(), -1);
        cur.clear();
        for(auto& a : btoa) if(a != -1) A[a] = -1;
        rep(a,0,g.size()) if(A[a] == 0) cur.push_back(a);
        for (int lay = 1;; lay += 2) {
            bool islast = 0;
            next.clear();
            for(auto& a : cur) for(auto& b : g[a]) {
                if (btoa[b] == -1) {
                    B[b] = lay;
                    islast = 1;
                }
                else if (btoa[b] != a && B[b] == -1) {
                    B[b] = lay;
```

```
fd                      next.push_back(btoa[b]);
7d                  }
7d              }
e5              if (islast) break;
db              if (next.empty()) return res;
76              for(auto& a : next) A[a] = lay+1;
9e              cur.swap(next);
7d          }
dd          rep(a,0,g.size()) {
5b              if(dfs(a, 0, g, btoa, A, B))
2c                  ++res;
7d          }
7d      }
7d  }
```

$\mathcal{O}(EV)$ maximum matching (DFS)

Description: This is a simple matching algorithm but should be just fine in most cases. Graph g should be a list of neighbours of the left partition, there must **not** be edges from the right partition to the left. n is the size of the left partition and m is the size of the right partition. If you want to get the matched pairs, $match[i]$ contains match for vertex i on the right side or -1 if it's not matched.
Time: $\mathcal{O}(EV)$

```
a9  vector<int> match;
16  vector<bool> seen;
2f  bool find(int j, const vector<vector<int>>& g) {
4e      if (match[j] == -1) return 1;
71      seen[j] = 1; int di = match[j];
da      for(auto& e : g[di])
6c          if (!seen[e] && find(e, g)) {
83              match[e] = di;
ed              return 1;
7d          }
29      return 0;
7d  }
61  int dfs_matching(const vector<vector<int>>& g, int n, int m) {
6b      match.assign(m, -1);
8f      rep(i,0,n) {
f9          seen.assign(m, 0);
e6          for(auto& j :g[i])
57              if (find(j, g)) {
e4                  match[j] = i;
b9                  break;
7d              }
7d      }
92      return m - (int)count(match.begin(), match.end(), -1);
7d  }
```

Min-cost matching

Description: Min cost bipartite matching. Negate costs for max cost.
Time: $\mathcal{O}(N^3)$

```
9a  typedef vector<double> vd;
56  bool zero(double x) { return fabs(x) < 1e-10; }
c4  double MinCostMatching(const vector<vd>& cost, vector<int>& L,
                                vector<int>& R) {
c8      int n = cost.size(), mated = 0;
31      vd dist(n), u(n), v(n);
4e      vector<int> dad(n), seen(n);
8f      rep(i,0,n) {
d7          u[i] = cost[i][0];
da          rep(j,1,n) u[i] = min(u[i], cost[i][j]);
7d      }
b3      rep(j,0,n) {
0c          v[j] = cost[0][j] - u[0];
09          rep(i,1,n) v[j] = min(v[j], cost[i][j] - u[i]);
7d      }
c6      L = R = vector<int>(n, -1);
60      rep(i,0,n) rep(j,0,n) {
29          if (R[j] != -1) continue;
ca          if (zero(cost[i][j] - u[i] - v[j])) {
4c              L[i] = j;
03              R[j] = i;
aa              mated++;
b9              break;
7d          }
7d      }
1a      for (; mated < n; mated++) { // until solution is feasible
2f          int s = 0;
e0          while (L[s] != -1) s++;
fa          fill(dad.begin(), dad.end(), -1);
e6          fill(seen.begin(), seen.end(), 0);
b5          rep(k,0,n)
9c              dist[k] = cost[s][k] - u[s] - v[k];
61          int j = 0;
5a          for (;;) {
7e              j = -1;
cc              rep(k,0,n){
bf                  if (seen[k]) continue;
0b                  if (j == -1 || dist[k] < dist[j]) j = k;
7d              }
e6              seen[j] = 1;
91              int i = R[j];
f7              if (i == -1) break;
cc              rep(k,0,n) {
bf                  if (seen[k]) continue;
e8                  auto new_dist = dist[j] + cost[i][k] - u[i] - v[k];
21                  if (dist[k] > new_dist) {
7b                      dist[k] = new_dist;
bf                      dad[k] = j;
7d                  }
7d              }
7d          }
cc          rep(k,0,n) {
b5              if (k == j || !seen[k]) continue;
0f              auto w = dist[k] - dist[j];
30              v[k] += w, u[R[k]] -= w;
7d          }
7e          u[s] += dist[j];
07          while (dad[j] >= 0) {
fd              int d = dad[j];
c0              R[j] = R[d];
3a              L[R[j]] = j;
0b              j = d;
7d          }
6a          R[j] = s;
b0          L[s] = j;
7d      }
81      auto value = vd(1)[0];
71      rep(i,0,n) value += cost[i][L[i]];
34      return value;
7d  }
```

General matching

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

```
a6  #include "../numerical/MatrixInverse-mod.h"
5c  vector<pair<int,int>> generalMatching(int N, vector<pair<int,int>>& ed) {
f0      vector<vector<ll>> mat(N, vector<ll>(N)), A;
e9      for(auto& pa : ed) {
39          int a = pa.first, b = pa.second, r = rand() % mod;
48          mat[a][b] = r, mat[b][a] = (mod - r) % mod;
7d      }
ca      int r = matInv(A = mat), M = 2*N - r, fi, fj;
39      assert(r % 2 == 0);
07      if (M != N) do {
08          mat.resize(M, vector<ll>(M));
90          rep(i,0,N) {
1b              mat[i].resize(M);
9c              rep(j,N,M) {
8d                  int r = rand() % mod;
c9                  mat[i][j] = r, mat[j][i] = (mod - r) % mod;
7d              }
7d          }
ff      } while (matInv(A = mat) != M);
81      vector<int> has(M, 1); vector<pair<int,int>> ret;
55      rep(it,0,M/2) {
6f          rep(i,0,M) if (has[i])
95              rep(j,i+1,M) if (A[i][j] && mat[i][j]) {
69                  fi = i; fj = j; goto done;
1f          } assert(0); done:
fc          if (fj < N) ret.emplace_back(fi, fj);
d1          has[fi] = has[fj] = 0;
80          rep(sw,0,2) {
97              ll a = modpow(A[fi][fj], mod-2);
9f              rep(i,0,M) if (has[i] && A[i][fj]) {
24                  ll b = A[i][fj] * a % mod;
26                  rep(j,0,M) A[i][j] = (A[i][j] - A[fi][j] * b) % mod;
7d              }
80              swap(fi,fj);
7d          }
7d      }
9e      return ret;
7d  }
```

Minimum vertex cover

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 an independent set.

```
93  #include "DFSMatching.h"
83  vector<int> cover(vector<vector<int>>& g, int n, int m) {
76      int res = dfs_matching(g, n, m);
31      seen.assign(m, false);
75      vector<bool> lfound(n, true);
38      for(auto& it : match) if (it != -1) lfound[it] = false;
9c      vector<int> q, cover;
9f      rep(i,0,n) if (lfound[i]) q.push_back(i);
77      while (!q.empty()) {
7e          int i = q.back(); q.pop_back();
85          lfound[i] = 1;
2d          for(auto& e : g[i]) if (!seen[e] && match[e] != -1) {
89              seen[e] = true;
c1              q.push_back(match[e]);
7d          }
7d      }
1d      rep(i,0,n) if (!lfound[i]) cover.push_back(i);
```

```
41      rep(i,0,m) if (seen[i]) cover.push_back(n+i);
d5      assert(cover.size() == res);
d8      return cover;
7d  }
```

Strongly connected components

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. `scc()` 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: $\mathcal{O}(E+V)$
Usage: `scc(graph, [\&](vector<int>\& v) { ... })`

```
c5  vector<int> val, comp, z, cont;
39  int Time, ncomps;
fc  template<class G, class F> int dfs(int j, G& g, F f) {
6e      int low = val[j] = ++Time, x; z.push_back(j);
1f      for(auto& e :g[j]) if (comp[e] < 0)
e4          low = min(low, val[e] ?: dfs(e,g,f));
8f      if (low == val[j]) {
76          do {
36              x = z.back(); z.pop_back();
4f              comp[x] = ncomps;
2b              cont.push_back(x);
09          } while (x != j);
32          f(cont); cont.clear();
4d          ncomps++;
7d      }
55      return val[j] = low;
7d  }
02  template<class G, class F> void scc(G& g, F f) {
46      int n = g.size();
f1      val.assign(n, 0); comp.assign(n, -1);
19      Time = ncomps = 0;
00      rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
7d  }
```

Biconnected components

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.
Time: $\mathcal{O}(E+V)$
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 vector<int>\& edgelist) {...});
```

```
78  vector<int> num, st;
49  vector<vector<pair<int,int>>> ed;
78  int Time;
d1  template<class F>
0b  int dfs(int at, int par, F f) {
8e      int me = num[at] = ++Time, e, y, top = me;
b4      for(auto& pa : ed[at]) if (pa.second != par) {
c8          tie(y, e) = pa;
07          if (num[y]) {
14              top = min(top, num[y]);
8b              if (num[y] < me)
1d                  st.push_back(e);
71          } else {
bf              int si = st.size();
c4              int up = dfs(y, e, f);
e0              top = min(top, up);
74              if (up == me) {
1d                  st.push_back(e);
af                  f(vector<int>(st.begin() + si, st.end()));
09                  st.resize(si);
7d              }
7f              else if (up < me)
1d                  st.push_back(e);
                // else e is a bridge
7d          }
7d      }
d8      return top;
7d  }
d1  template<class F>
cf  void bicomps(F f) {
24      num.assign(ed.size(), 0);
1c      rep(i,0,ed.size()) if (!num[i]) dfs(i, -1, f);
7d  }
```

2-SAT

Description: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a\|\|!b)\&\&(!a\|\|c)\&\&(d\|\|!b)\&\&\ldots$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions (x).
Time: $\mathcal{O}(N+E)$, where N is the number of boolean variables, and E is the number of clauses.
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
```

```
e1  struct TwoSat {
81      int N;
1b      vector<vector<int>> gr;
8d      vector<int> values; // 0 = false, 1 = true
43      TwoSat(int n = 0) : N(n), gr(2*n) {}
53      int add_var() { // (optional)
a3          gr.emplace_back();
a3          gr.emplace_back();
22          return N++;
7d      }
3d      void either(int f, int j) {
ed          f = (f >= 0 ? 2*f : -1-2*f);
51          j = (j >= 0 ? 2*j : -1-2*j);
90          gr[f^1].push_back(j);
46          gr[j^1].push_back(f);
7d      }
8c      void set_value(int x) { either(x, x); }
ab      void at_most_one(const vector<int>& li) { // (optional)
20          if (li.size() <= 1) return;
46          int cur = ~li[0];
71          rep(i,2,li.size()) {
25              int next = add_var();
f8              either(cur, ~li[i]);
68              either(cur, next);
5a              either(~li[i], next);
01              cur = ~next;
7d          }
15          either(cur, ~li[1]);
7d      }
d3      vector<int> val, comp, z; int time = 0;
b5      int dfs(int i) {
d9          int low = val[i] = ++time, x; z.push_back(i);
f3          for(auto& e : gr[i]) if (!comp[e])
59              low = min(low, val[e] ?: dfs(e));
bf          ++time;
6a          if (low == val[i]) do {
36              x = z.back(); z.pop_back();
a7              comp[x] = time;
47              if (values[x>>1] == -1)
95                  values[x>>1] = !(x&1);
d5          } while (x != i);
8e          return val[i] = low;
7d      }
49      bool solve() {
e1          values.assign(N, -1);
12          val.assign(2*N, 0); comp = val;
43          rep(i,0,2*N) if (!comp[i]) dfs(i);
10          rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
ed          return 1;
7d      }
6c  };
```

Tree jumps

Description: Calculate power of two jumps in a tree. Assumes the root node points to itself.
Time: $\mathcal{O}(|V|\log|V|)$

```
c4  vector<vector<int>> treeJump(vector<int>& P){
be      int on = 1, d = 1;
8a      while(on < P.size()) on *= 2, d++;
1f      vector<vector<int>> jmp(d, P);
36      rep(i,1,d) rep(j,0,P.size())
10          jmp[i][j] = jmp[i-1][jmp[i-1][j]];
82      return jmp;
7d  }
2b  int jmp(vector<vector<int>>& tbl, int nod, int steps){
04      rep(i,0,tbl.size())
87          if(steps&(1<<i)) nod = tbl[i][nod];
07      return nod;
7d  }
```

LCA

Description: Lowest common ancestor. Finds the lowest common ancestor 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.
Time: $\mathcal{O}(|V|\log|V|+Q)$
Usage:
```
LCA lca(undirGraph);
lca.query(firstNode, secondNode);
lca.distance(firstNode, secondNode);
```

```
b1  typedef vector<pair<int,int>> vpi;
f3  typedef vector<vpi> graph;
90  const pair<int,int> inf(1 << 29, -1);
9b  #define RMQ_HAVE_INF#include "../data-structures/RMQ.h"
46  struct LCA {
5b      vector<int> time;
bc      vector<ll> dist;
5a      RMQ<pair<int,int>> rmq;
e4      LCA(graph& C) : time(C.size(), -99), dist(C.size()), rmq(dfs(C)) {}
```

```
fc      vpi dfs(graph& C) {
d8          vector<tuple<int, int, int, ll> > q(1);
01          vpi ret;
d2          int T = 0, v, p, d; ll di;
77          while (!q.empty()) {
62              tie(v, p, d, di) = q.back();
80              q.pop_back();
69              if (d) ret.emplace_back(d, p);
c8              time[v] = T++;
65              dist[v] = di;
e2              for(auto& e : C[v]) if (e.first != p)
07                  q.emplace_back(e.first, v, d+1, di + e.second);
7d          }
9e          return ret;
7d      }

43      int query(int a, int b) {
6b          if (a == b) return a;
83          a = time[a], b = time[b];
25          return rmq.query(min(a, b), max(a, b)).second;
7d      }
31      ll distance(int a, int b) {
da          int lca = query(a, b);
8e          return dist[a] + dist[b] - 2 * dist[lca];
7d      }
6c  };
```

Tree compression

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

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

```
7e  #include "LCA.h"

ef  vpi compressTree(LCA& lca, const vector<int>& subset) {
0b      static vector<int> rev; rev.resize(lca.dist.size());
d2      vector<int> li = subset, &T = lca.time;
7b      auto cmp = [&](int a, int b) { return T[a] < T[b]; };
b0      sort(li.begin(), li.end(), cmp);
48      int m = li.size()-1;
40      rep(i,0,m) {
df          int a = li[i], b = li[i+1];
64          li.push_back(lca.query(a, b));
7d      }
b0      sort(li.begin(), li.end(), cmp);
73      li.erase(unique(li.begin(), li.end()), li.end());
13      rep(i,0,li.size()) rev[li[i]] = i;
f3      vpi ret = {pair<int,int>(0, li[0])};
bb      rep(i,0,li.size()-1) {
df          int a = li[i], b = li[i+1];
7c          ret.emplace_back(rev[lca.query(a, b)], b);
7d      }
9e      return ret;
7d  }
```

Centroid decomposition

Description: Centroid-decomposes given tree. The sample usage calculates degree of each vertex.

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

Usage: `decompose(G, 0);`

```
b6  typedef vector<vector<int>> Graph;

    // Helper - returns {subtree size, centroid} of given subtree.
97  pair<int, int> _find_centroid(const Graph &G, int v, int father, int
                                    totcnt) {
76      int ourcnt = 1;
55      int centroid = -1;
d9      int biggest = 0;
9f      for (int s: G[v]) {
a9          if (s == father)
9c              continue;
b6          int subcnt, possible_centroid;
d4          tie(subcnt, possible_centroid) = _find_centroid(G, s, v, totcnt);
1a          ourcnt += subcnt;
d7          biggest = max(subcnt, biggest);
18          if (possible_centroid != -1)
c7              centroid = possible_centroid;
7d      }

34      int above = totcnt - ourcnt;
5d      if (above <= totcnt / 2 && biggest <= totcnt / 2)
b6          centroid = v;

c6      return {ourcnt, centroid};
7d  }

    // Given a forest G and a vertex v, returns the centroid of the tree
                                    containing v.
9b  int find_centroid(const Graph &G, int v) {
c2      int n = _find_centroid(G, v, -1, 0).first;
63      return _find_centroid(G, v, -1, n).second;
7d  }

b5  vector<int> counts; // Only used for the example
    // Destroys and recreates edges, preserving their order. Replace <cut
                                    here>
    // code with desired action for each centroid
47  void decompose(Graph &G, int start) {
0f      int v = find_centroid(G, start);
        // <cut here>
3a      counts[v] += G[v].size();
3d      for (int s: G[v])
75          counts[s]++;
        // </cut here>

9f      for (int s: G[v]) {
b0          int pos = -1;
5a          while (G[s][++pos] != v) {}
99          swap(G[s][pos], G[s][G[s].size() - 1]);
34          G[s].pop_back();
ed          decompose(G, s);
02          G[s].push_back(v);
99          swap(G[s][pos], G[s][G[s].size() - 1]);
7d      }
7d  }
```

HLD + LCA

Description: Calculates the heavy-light decomposition of given tree. Data for chains is stored in one flat structure. Also usable for lowest common ancestor.

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

Usage:
```
initHLD();
// What is the value on edges on path from 1 to 2?
quepdate(1, 2, false, ZERO);
// Update path from 1 to 2 with value 30
quepdate(1, 2, true, 30);
```

```
b4  vector<vector<int>> G;
83  vector<int> et, in, out, subs, depth, top, par;
6d  typedef ll T;

    // Implement these four to customise behavior
81  T ZERO = 0; // Neutral element for operations
d5  T combine(T a, T b) { return a + b; } // How to compose two results
    // Query or update interval from a to b (0<=a,b<|V|)
f5  T flat_quepdate(int a, int b, bool upd, T val);
17  void flat_init(void); // Initialize the flat data structure

13  void dfs_counts(int v = 0) {
8d      subs[v] = 1;
55      for (auto &s: G[v]) {
86          par[s] = v, depth[s] = depth[v] + 1;
23          dfs_counts(s);
41          subs[v] += subs[s];
8c          if (subs[s] > subs[G[v][0]]) swap(s, G[v][0]);
7d      }
7d  }

49  int dfs_numbering(int v = 0, int t = -1) {
18      in[v] = ++t, et.push_back(v);
75      for (auto s: G[v])
f3          t = dfs_numbering(s, t);
28      return out[v] = t;
7d  }

9c  void buildHLD(int v = 0, int c = 0) {
a5      top[v] = c;
a7      rep(i, 0, G[v].size())
05          buildHLD(G[v][i], (i) ? G[v][i] : c);
7d  }

a6  void initHLD(void) {
bc      in = {}, in.resize(G.size());
94      out = subs = depth = top = par = in;
f5      par[0] = -1, et = {}, depth[0] = 0;
43      dfs_counts();
1c      dfs_numbering();
7b      buildHLD();
e4      flat_init();
7d  }

    // Needs dfs_counts(), buildHLD()
66  int lca(int a, int b) {
fa      for (; top[a] != top[b]; b = par[top[b]])
a0          if (depth[top[a]] > depth[top[b]])
6b              swap(a, b);
4f      return (depth[a] < depth[b]) ? a : b;
7d  }

    // a is an ancestor of b; a isn't included in the query
5f  T _quepdate(int a, int b, bool upd, T val) {
3c      T res = ZERO;
fa      for (; top[a] != top[b]; b = par[top[b]])
21          res = combine(res, flat_quepdate(in[top[b]], in[b], upd, val));
97      return combine(res, flat_quepdate(in[a] + 1, in[b], upd, val));
7d  }

    // lca(a, b) isn't included in the query, which is desired if operating on
                                    edge weights
54  T quepdate(int a, int b, bool upd, T val) {
5a      int l = lca(a, b);
fa      return combine(_quepdate(l, a, upd, val), _quepdate(l, b, upd, val));
7d  }
```

Link-cut tree

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

```
c1 struct Node { // Splay tree. Root's pp contains tree's parent.
9a     Node *p = 0, *pp = 0, *c[2];
b0     bool flip = 0;
6a     Node() { c[0] = c[1] = 0; fix(); }
0b     void fix() {
fd         if (c[0]) c[0]->p = this;
cc         if (c[1]) c[1]->p = this;
           // (+ update sum of subtree elements etc. if wanted)
7d     }
66     void push_flip() {
a9         if (!flip) return;
a5         flip = 0; swap(c[0], c[1]);
d0         if (c[0]) c[0]->flip ^= 1;
f9         if (c[1]) c[1]->flip ^= 1;
7d     }
b4     int up() { return p ? p->c[1] == this : -1; }
d4     void rot(int i, int b) {
30         int h = i ^ b;
e1         Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
75         if ((y->p = p)) p->c[up()] = y;
06         c[i] = z->c[i ^ 1];
aa         if (b < 2) {
c6             x->c[h] = y->c[h ^ 1];
23             z->c[h ^ 1] = b ? x : this;
7d         }
72         y->c[i ^ 1] = b ? this : x;
f3         fix(); x->fix(); y->fix();
05         if (p) p->fix();
0e         swap(pp, y->pp);
7d     }
46     void splay() {
2c         for (push_flip(); p; ) {
f5             if (p->p) p->p->push_flip();
b9             p->push_flip(); push_flip();
03             int c1 = up(), c2 = p->up();
84             if (c2 == -1) p->rot(c1, 2);
04             else p->p->rot(c2, c1 != c2);
7d         }
7d     }
3d     Node* first() {
a5         push_flip();
24         return c[0] ? c[0]->first() : (splay(), this);
7d     }
6c };
76 struct LinkCut {
f7     vector<Node> node;
44     LinkCut(int N) : node(N) {}
64     void link(int u, int v) { // add an edge (u, v)
bc         assert(!connected(u, v));
48         make_root(&node[u]);
c4         node[u].pp = &node[v];
7d     }
24     void cut(int u, int v) { // remove an edge (u, v)
c8         Node *x = &node[u], *top = &node[v];
dd         make_root(top); x->splay();
1a         assert(top == (x->pp ?: x->c[0]));
de         if (x->pp) x->pp = 0;
3a         else {
08             x->c[0] = top->p = 0;
bf             x->fix();
7d         }
7d     }
d1     bool connected(int u, int v) { // are u, v in the same tree?
1a         Node* nu = access(&node[u])->first();
d6         return nu == access(&node[v])->first();
7d     }
a5     void make_root(Node* u) {
13         access(u);
e9         u->splay();
ec         if(u->c[0]) {
6e             u->c[0]->p = 0;
e9             u->c[0]->flip ^= 1;
21             u->c[0]->pp = u;
26             u->c[0] = 0;
3a             u->fix();
7d         }
7d     }
4f     Node* access(Node* u) {
e9         u->splay();
d9         while (Node* pp = u->pp) {
e3             pp->splay(); u->pp = 0;
d1             if (pp->c[1]) {
82                 pp->c[1]->p = 0; pp->c[1]->pp = pp; }
ad             pp->c[1] = u; pp->fix(); u = pp;
7d         }
e4         return u;
7d     }
6c };
```

Counting the number of spanning trees

Description: To count the number of spanning trees in an undirected graph G: create an $N \times N$ matrix `mat`, and for each edge $(a, b) \in G$, do `mat[a][a]++, mat[b][b]++, mat[a][b]--, mat[b][a]--;`. Remove the last row and column, and take the determinant.

Geometry

Geometric primitives

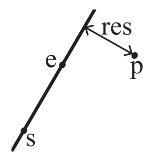
Point

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

```
82 template <class T>
b5 struct Point {
96     typedef Point P;
e7     T x, y;
70     explicit Point(T x=0, T y=0) : x(x), y(y) {}
c8     bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
6c     bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
46     P operator+(P p) const { return P(x+p.x, y+p.y); }
e8     P operator-(P p) const { return P(x-p.x, y-p.y); }
3e     P operator*(T d) const { return P(x*d, y*d); }
69     P operator/(T d) const { return P(x/d, y/d); }
9a     T dot(P p) const { return x*p.x + y*p.y; }
a2     T cross(P p) const { return x*p.y - y*p.x; }
b4     T cross(P a, P b) const { return (a-*this).cross(b-*this); }
a2     T dist2() const { return x*x + y*y; }
4b     double dist() const { return sqrt((double)dist2()); }
       // angle to x-axis in interval [-pi, pi]
e6     double angle() const { return atan2(y, x); }
28     P unit() const { return *this/dist(); } // makes dist()=1
73     P perp() const { return P(-y, x); } // rotates +90 degrees
58     P normal() const { return perp().unit(); }
       // returns point rotated 'a' radians ccw around the origin
32     P rotate(double a) const {
ba         return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
6c };
```

Line distance

Description:



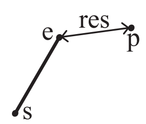
Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. $a = b$ gives NaN. `P` is supposed to be `Point<T>` or `Point3D<T>` where `T` is e.g. double or long long. It uses products in intermediate steps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

```
ec #include "Point.h"

10 template <class P>
97 double lineDist(const P& a, const P& b, const P& p) {
54     return (double)(b-a).cross(p-a)/(b-a).dist();
7d }
```

Segment distance

Description:



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

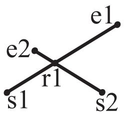
Usage: `Point<double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

```
ec #include "Point.h"

22 typedef Point<double> P;
ea double segDist(P& s, P& e, P& p) {
09     if (s==e) return (p-s).dist();
f1     auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
64     return ((p-s)*d-(e-s)*t).dist()/d;
7d }
```

Segment intersection

Description:



If a unique intersection point between the line segments s_1—e_1 and s_2—e_2 exists r_1 is set to this point and 1 is returned. If no intersection point exists 0 is returned and if infinitely many exists 2 is returned and r_1 and r_2 are set to the two ends of the common line. The wrong position will be returned if `P` is `Point<int>` 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. Use SegmentIntersectionQ to get just a true/false answer.

Usage:
```
Point<double> intersection, dummy;
if (segmentIntersection(s1,e1,s2,e2,intersection,dummy)==1)
cout << "segments intersect at " << intersection << endl;
```

```
ec #include "Point.h"

10 template <class P>
51 int segmentIntersection(const P& s1, const P& e1,
32         const P& s2, const P& e2, P& r1, P& r2) {
77     if (e1==s1) {
36         if (e2==s2) {
a6             if (e1==e2) { r1 = e1; return 1; } //all equal
4f             else return 0; //different point segments
70         } else return segmentIntersection(s2,e2,s1,e1,r1,r2);//swap
7d     }
       //segment directions and separation
69     P v1 = e1-s1, v2 = e2-s2, d = s2-s1;
67     auto a = v1.cross(v2), a1 = v1.cross(d), a2 = v2.cross(d);
b7     if (a == 0) { //if parallel
ad         auto b1=s1.dot(v1), c1=e1.dot(v1),
73              b2=s2.dot(v1), c2=e2.dot(v1);
40         if (a1 || a2 || max(b1,min(b2,c2))>min(c1,max(b2,c2)))
29             return 0;
e5         r1 = min(b2,c2)<b1 ? s1 : (b2<c2 ? s2 : e2);
71         r2 = max(b2,c2)>c1 ? e1 : (b2>c2 ? s2 : e2);
2d         return 2-(r1==r2);
7d     }
ae     if (a < 0) { a = -a; a1 = -a1; a2 = -a2; }
67     if (0<a1 || a<-a1 || 0<a2 || a<-a2)
29         return 0;
9b     r1 = s1-v1*a2/a;
ed     return 1;
7d }
```

Segment intersection (boolean version)

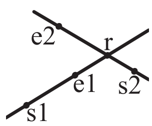
Description: Like segmentIntersection, but only returns true/false. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.

```
ec #include "Point.h"

10 template <class P>
20 bool segmentIntersectionQ(P s1, P e1, P s2, P e2) {
77     if (e1 == s1) {
6a         if (e2 == s2) return e1 == e2;
56         swap(s1,s2); swap(e1,e2);
7d     }
69     P v1 = e1-s1, v2 = e2-s2, d = s2-s1;
e4     auto a = v1.cross(v2), a1 = d.cross(v1), a2 = d.cross(v2);
b7     if (a == 0) { // parallel
ad         auto b1 = s1.dot(v1), c1 = e1.dot(v1),
73              b2 = s2.dot(v1), c2 = e2.dot(v1);
dc         return !a1 && max(b1,min(b2,c2)) <= min(c1,max(b2,c2));
7d     }
ae     if (a < 0) { a = -a; a1 = -a1; a2 = -a2; }
5b     return (0 <= a1 && a1 <= a && 0 <= a2 && a2 <= a);
7d }
```

Line intersection

Description:



If a unique intersection point of the lines going through s_1, e_1 and s_2, e_2 exists, r is set to this point and 1 is returned. If no intersection point exists 0 is returned and if infinitely many exists -1 is returned. If $s_1 = e_1$ or $s_2 = e_2$, then -1 is returned. The wrong position will be returned if `P` is `Point<int>` 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:
```
point<double> intersection;
if (1 == LineIntersection(s1,e1,s2,e2,intersection))
cout << "intersection point at " << intersection << endl;
```

```
ec #include "Point.h"

10 template <class P>
45 int lineIntersection(const P& s1, const P& e1, const P& s2,
62         const P& e2, P& r) {
df     if ((e1-s1).cross(e2-s2)) { //if not parallell
cb         r = s2-(e2-s2)*(e1-s1).cross(s2-s1)/(e1-s1).cross(e2-s2);
ed         return 1;
58     } else
96         return -((e1-s1).cross(s2-s1)==0 || s2==e2);
7d }
```

Point-line orientation

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

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

```
ec #include "Point.h"

10 template <class P>
14 int sideOf(const P& s, const P& e, const P& p) {
a8     auto a = (e-s).cross(p-s);
12     return (a > 0) - (a < 0);
7d }
10 template <class P>
96 int sideOf(const P& s, const P& e, const P& p, double eps) {
a8     auto a = (e-s).cross(p-s);
4b     double l = (e-s).dist()*eps;
ab     return (a > l) - (a < -l);
7d }
```

Point on segment

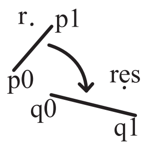
Description: Returns true iff p lies on the line segment from s to e. Intended for use with e.g. `Point<long long>` where overflow is an issue. Use `segDist(s,e,p)<=eps` instead when using `Point<double>`.

```
ec #include "Point.h"

10 template <class P>
fe bool onSegment(const P& s, const P& e, const P& p) {
ae     P ds = p-s, de = p-e;
7d     return ds.cross(de) == 0 && ds.dot(de) <= 0;
7d }
```

Linear transformation

Description:



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

```
ec #include "Point.h"

22 typedef Point<double> P;
a6 P linearTransformation(const P& p0, const P& p1,
46         const P& q0, const P& q1, const P& r) {
6a     P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
43     return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
7d }
```

Angle

Description: A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping.

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
```

```
63 struct Angle {
4d     int x, y;
76     int t;
32     Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
7b     Angle operator-(Angle a) const { return {x-a.x, y-a.y, t}; }
4d     int quad() const {
fa         assert(x || y);
c4         if (y < 0) return (x >= 0) + 2;
0d         if (y > 0) return (x <= 0);
e9         return (x <= 0) * 2;
7d     }
b8     Angle t90() const { return {-y, x, t + (quad() == 3)}; }
33     Angle t180() const { return {-x, -y, t + (quad() >= 2)}; }
c4     Angle t360() const { return {x, y, t + 1}; }
6c };
b9 bool operator<(Angle a, Angle b) {
       // add a.dist2() and b.dist2() to also compare distances
cb     return make_tuple(a.t, a.quad(), a.y * (ll)b.x) <
a4            make_tuple(b.t, b.quad(), a.x * (ll)b.y);
7d }
0c bool operator>=(Angle a, Angle b) { return !(a < b); }
ad bool operator>(Angle a, Angle b) { return b < a; }
51 bool operator<=(Angle a, Angle b) { return !(b < a); }
```

```
   // Given two points, this calculates the smallest angle between
   // them, i.e., the angle that covers the defined line segment.
98 pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
9f     if (b < a) swap(a, b);
dc     return (b < a.t180() ?
5b             make_pair(a, b) : make_pair(b, a.t360()));
7d }

1e Angle operator+(Angle a, Angle b) { // where b is a vector
24     Angle r(a.x + b.x, a.y + b.y, a.t);
63     if (r > a.t180()) r.t--;
71     return r.t180() < a ? r.t360() : r;
7d }
```

Circles

Circle intersection

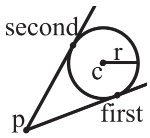
Description: Computes a pair of points at which two circles intersect. Returns false in case of no intersection.

```
ec #include "Point.h"

22 typedef Point<double> P;
fe bool circleIntersection(P a, P b, double r1, double r2,
93         pair<P, P>* out) {
99     P delta = b - a;
84     assert(delta.x || delta.y || r1 != r2);
92     if (!delta.x && !delta.y) return false;
c5     double r = r1 + r2, d2 = delta.dist2();
15     double p = (d2 + r1*r1 - r2*r2) / (2.0 * d2);
a3     double h2 = r1*r1 - p*p*d2;
06     if (d2 > r*r || h2 < 0) return false;
99     P mid = a + delta*p, per = delta.perp() * sqrt(h2 / d2);
c0     *out = {mid + per, mid - per};
35     return true;
7d }
```

Circle tangents

Description:



Returns a pair of the two points on the circle with radius r centered around c whose tangent lines intersect p. If p lies within the circle, NaN-points are returned. `P` is intended to be `Point<double>`. The first point is the one to the right as seen from the p towards c.

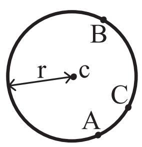
Usage: `typedef Point<double> P;`
`pair<P,P> p = circleTangents(P(100,2),P(0,0),2);`

```
ec #include "Point.h"

10 template <class P>
03 pair<P,P> circleTangents(const P &p, const P &c, double r) {
e0     P a = p-c;
87     double x = r*r/a.dist2(), y = sqrt(x-x*x);
23     return make_pair(c+a*x+a.perp()*y, c+a*x-a.perp()*y);
7d }
```

Circumcircle

Description:



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

```
ec #include "Point.h"

22 typedef Point<double> P;
7d double ccRadius(const P& A, const P& B, const P& C) {
cd     return (B-A).dist()*(C-B).dist()*(A-C).dist()/
98             abs((B-A).cross(C-A))/2;
7d }
38 P ccCenter(const P& A, const P& B, const P& C) {
f6     P b = C-A, c = B-A;
82     return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
7d }
```

Minimum enclosing circle

Description: Computes the minimum circle that encloses a set of points.
Time: Expected $\mathcal{O}(N)$

```
47 #include "circumcircle.h"

1d pair<double, P> mec2(vector<P>& S, P a, P b, int n) {
ae     double hi = INFINITY, lo = -hi;
8f     rep(i,0,n) {
b3         auto si = (b-a).cross(S[i]-a);
54         if (si == 0) continue;
83         P m = ccCenter(a, b, S[i]);
85         auto cr = (b-a).cross(m-a);
54         if (si < 0) hi = min(hi, cr);
06         else lo = max(lo, cr);
7d     }
c4     double v = (0 < lo ? lo : hi < 0 ? hi : 0);
e3     P c = (a + b) / 2 + (b - a).perp() * v / (b - a).dist2();
32     return {(a - c).dist2(), c};
7d }
a4 pair<double, P> mec(vector<P>& S, P a, int n) {
b3     random_shuffle(S.begin(), S.begin() + n);
dd     P b = S[0], c = (a + b) / 2;
e1     double r = (a - c).dist2();
b6     rep(i,1,n) if ((S[i] - c).dist2() > r * (1 + 1e-8)) {
34         tie(r,c) = (n == S.size() ?
64             mec(S, S[i], i) : mec2(S, a, S[i], i));
7d     }
37     return {r, c};
7d }
63 pair<double, P> enclosingCircle(vector<P> S) {
c5     assert(!S.empty()); auto r = mec(S, S[0], S.size());
9f     return {sqrt(r.first), r.second};
7d }
```

Polygons

Inside general polygon

Description: Returns true if p lies within the polygon described by the points between iterators *begin* and *end*. If strict, false is returned when p is on the edge of the polygon. Answer is calculated by counting the number of intersections between the polygon and a line going from p to infinity in the positive x-direction. The algorithm uses products in intermediate steps so watch out for overflow. If points within ϵ from an edge should be considered as on the edge, replace the line `if (onSegment...` with the comment below it (this will cause overflow for int and long long).
Time: $\mathcal{O}(N)$
Usage: `typedef Point<int> pi;`
`vector<pi> v; v.push_back(pi(4,4));`
`v.push_back(pi(1,2)); v.push_back(pi(2,1));`
`bool in = insidePolygon(v.begin(),v.end(), pi(3,4), false);`

```
ec #include "Point.h"
c4 #include "onSegment.h"
ff #include "SegmentDistance.h"

12 template <class It, class P>
d0 bool insidePolygon(It begin, It end, const P& p,
32         bool strict = true) {
d8     int n = 0; //number of isects with line from p to (inf,p.y)
c8     for (It i = begin, j = end-1; i != end; j = i++) {
           //if p is on edge of polygon
68         if (onSegment(*i, *j, p)) return !strict;
           //or: if (segDist(*i, *j, p) <= epsilon) return !strict;
           //increment n if segment intersects line from p
53         n += (max(i->y,j->y) > p.y && min(i->y,j->y) <= p.y &&
f4                 ((*j-*i).cross(p-*i) > 0) == (i->y <= p.y));
7d     }
9c     return n&1; //inside if odd number of intersections
7d }
```

Polygon area

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

```
ec #include "Point.h"

82 template <class T>
32 T polygonArea2(vector<Point<T>>& v) {
9c     T a = v.back().cross(v[0]);
18     rep(i,0,v.size()-1) a += v[i].cross(v[i+1]);
84     return a;
7d }
```

Polygon's center of mass

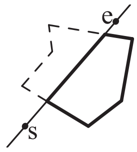
Description: Returns the center of mass for a polygon.

```
ec #include "Point.h"

22 typedef Point<double> P;
c6 Point<double> polygonCenter(vector<P>& v) {
d4     auto i = v.begin(), end = v.end(), j = end-1;
7b     Point<double> res{0,0}; double A = 0;
8b     for (; i != end; j=i++) {
55         res = res + (*i + *j) * j->cross(*i);
6d         A += j->cross(*i);
7d     }
14     return res / A / 3;
7d }
```

Polygon cut

Description:



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

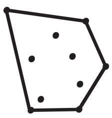
Usage:
```
vector<P> p = ...;
p = polygonCut(p, P(0,0), P(1,0));
```

```
ec #include "Point.h"
02 #include "lineIntersection.h"

22 typedef Point<double> P;
7e vector<P> polygonCut(const vector<P>& poly, P s, P e) {
dd     vector<P> res;
c3     rep(i,0,poly.size()) {
4b         P cur = poly[i], prev = i ? poly[i-1] : poly.back();
0c         bool side = s.cross(e, cur) < 0;
bf         if (side != (s.cross(e, prev) < 0)) {
ac             res.emplace_back();
31             lineIntersection(s, e, cur, prev, res.back());
7d         }
2e         if (side)
d7             res.push_back(cur);
7d     }
b1     return res;
7d }
```

Convex hull

Description:



Returns a vector of indices 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: $\mathcal{O}(N \log N)$

Usage:
```
vector<P> ps, hull;
for(auto& i : convexHull(ps)) hull.push_back(ps[i]);
```

```
ec #include "Point.h"

69 typedef Point<ll> P;
8b pair<vector<int>, vector<int>> ulHull(const vector<P>& S) {
67     vector<int> Q(S.size()), U, L;
87     iota(Q.begin(), Q.end(), 0);
57     sort(Q.begin(), Q.end(), [&S](int a, int b){ return S[a] < S[b]; });
aa     for(auto& it : Q) {
4f #define ADDP(C, cmp) while (C.size() > 1 && S[C[C.size()-2]].cross(\
2f     S[it], S[C.back()]) cmp 0) C.pop_back(); C.push_back(it);
fa         ADDP(U, <=); ADDP(L, >=);
7d     }
80     return {U, L};
7d }

2b vector<int> convexHull(const vector<P>& S) {
28     vector<int> u, l; tie(u, l) = ulHull(S);
06     if (S.size() <= 1) return u;
0a     if (S[u[0]] == S[u[1]]) return {0};
2a     l.insert(l.end(), u.rbegin()+1, u.rend()-1);
70     return l;
7d }
```

Polygon diameter

Description: Calculates the max squared distance of a set of points.

```
f8 #include "ConvexHull.h"

25 vector<pair<int,int>> antipodal(const vector<P>& S, vector<int>& U,
                                  vector<int>& L) {
fb     vector<pair<int,int>> ret;
89     int i = 0, j = L.size() - 1;
c3     while (i < U.size() - 1 || j > 0) {
01         ret.emplace_back(U[i], L[j]);
dc         if (j == 0 || (i != U.size()-1 && (S[L[j]] - S[L[j-1]])
42                     .cross(S[U[i+1]] - S[U[i]]) > 0)) ++i;
96         else --j;
7d     }
9e     return ret;
7d }

2e pair<int,int> polygonDiameter(const vector<P>& S) {
44     vector<int> U, L; tie(U, L) = ulHull(S);
14     pair<ll, pair<int,int>> ans;
45     for(auto& x : antipodal(S, U, L))
7d         ans = max(ans, {(S[x.first] - S[x.second]).dist2(), x});
9f     return ans.second;
7d }
```

Inside polygon (pseudo-convex)

Description: Determine whether a point t lies inside a given polygon (counter-clockwise order). The polygon must be such that every point on the circumference is visible from the first point in the vector. It returns 0 for points outside, 1 for points on the circumference, and 2 for points inside.

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

```
ec #include "Point.h"
d1 #include "sideOf.h"
c4 #include "onSegment.h"

69 typedef Point<ll> P;
d0 int insideHull2(const vector<P>& H, int L, int R, const P& p) {
2a     int len = R - L;
5e     if (len == 2) {
70         int sa = sideOf(H[0], H[L], p);
66         int sb = sideOf(H[L], H[L+1], p);
e4         int sc = sideOf(H[L+1], H[0], p);
8e         if (sa < 0 || sb < 0 || sc < 0) return 0;
18         if (sb==0 || (sa==0 && L == 1) || (sc == 0 && R == H.size()))
ed             return 1;
5f         return 2;
7d     }
36     int mid = L + len / 2;
4c     if (sideOf(H[0], H[mid], p) >= 0)
5a         return insideHull2(H, mid, R, p);
09     return insideHull2(H, L, mid+1, p);
7d }

56 int insideHull(const vector<P>& hull, const P& p) {
d8     if (hull.size() < 3) return onSegment(hull[0], hull.back(), p);
18     else return insideHull2(hull, 1, hull.size(), p);
7d }
```

Intersect line with convex polygon (queries)

Description: Line-convex polygon intersection. The polygon must be ccw and have no colinear points. `isct(a,b)` returns a pair describing the intersection of a line with the polygon:

- $(-1,-1)$ if no collision,
- $(i,-1)$ if touching the corner i,
- (i,i) if along side $(i,i+1)$,
- (i,j) if crossing sides $(i,i+1)$ and $(j,j+1)$. In the last case, if a corner i is crossed, this is treated as happening on side $(i,i+1)$. The points are returned in the same order as the line hits the polygon.

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

```
ec #include "Point.h"

b8 ll sgn(ll a) { return (a > 0) - (a < 0); }
69 typedef Point<ll> P;
75 struct HullIntersection {
81     int N;
d8     vector<P> p;
be     vector<pair<P, int>> a;

52     HullIntersection(const vector<P>& ps) : N(ps.size()), p(ps) {
dd         p.insert(p.end(), ps.begin(), ps.end());
56         int b = 0;
4d         rep(i,1,N) if (P{p[i].y,p[i].x} < P{p[b].y, p[b].x}) b = i;
90         rep(i,0,N) {
ad             int f = (i + b) % N;
16             a.emplace_back(p[f+1] - p[f], f);
7d         }
7d     }

03     int qd(P p) {
37         return (p.y < 0) ? (p.x >= 0) + 2
ad              : (p.x <= 0) * (1 + (p.y <= 0));
7d     }

03     int bs(P dir) {
86         int lo = -1, hi = N;
2a         while (hi - lo > 1) {
54             int mid = (lo + hi) / 2;
be             if (make_pair(qd(dir), dir.y * a[mid].first.x) <
75                 make_pair(qd(a[mid].first), dir.x * a[mid].first.y))
46                 hi = mid;
e5             else lo = mid;
7d         }
a6         return a[hi%N].second;
7d     }

1d     bool isign(P a, P b, int x, int y, int s) {
65         return sgn(a.cross(p[x], b)) * sgn(a.cross(p[y], b)) == s;
7d     }

a2     int bs2(int lo, int hi, P a, P b) {
62         int L = lo;
ff         if (hi < lo) hi += N;
2a         while (hi - lo > 1) {
54             int mid = (lo + hi) / 2;
5e             if (isign(a, b, mid, L, -1)) hi = mid;
e5             else lo = mid;
7d         }
87         return lo;
7d     }
```

```
23     pair<int,int> isct(P a, P b) {
f1         int f = bs(a - b), j = bs(b - a);
e9         if (isign(a, b, f, j, 1)) return {-1, -1};
b5         int x = bs2(f, j, a, b)%N,
97             y = bs2(j, f, a, b)%N;
c9         if (a.cross(p[x], b) == 0 &&
e8             a.cross(p[x+1], b) == 0) return {x, x};
64         if (a.cross(p[y], b) == 0 &&
8b             a.cross(p[y+1], b) == 0) return {y, y};
d8         if (a.cross(p[f], b) == 0) return {f, -1};
b3         if (a.cross(p[j], b) == 0) return {j, -1};
2e         return {x, y};
7d     }
6c };
```

Misc. Point Set Problems

Closest pair of points

Description: i_1, i_2 are the indices to the closest pair of points in the point vector p after the call. The distance is returned.
Time: $\mathcal{O}(N \log N)$

```
ec #include "Point.h"
05 template <class It>
45 bool it_less(const It& i, const It& j) { return *i < *j; }
05 template <class It>
f9 bool y_it_less(const It& i,const It& j) {return i->y < j->y;}
da template<class It, class IIt> /* IIt = vector<It>::iterator */
26 double cp_sub(IIt ya, IIt yaend, IIt xa, It &i1, It &i2) {
65     typedef typename iterator_traits<It>::value_type P;
cb     int n = yaend-ya, split = n/2;
7f     if(n <= 3) { // base case
67         double a = (*xa[1]-*xa[0]).dist(), b = 1e50, c = 1e50;
1e         if(n==3) b=(*xa[2]-*xa[0]).dist(), c=(*xa[2]-*xa[1]).dist();
a2         if(a <= b) { i1 = xa[1];
7d             if(a <= c) return i2 = xa[0], a;
38             else return i2 = xa[2], c;
f3         } else { i1 = xa[2];
fb             if(b <= c) return i2 = xa[0], b;
81             else return i2 = xa[1], c;
57     }   }
5c     vector<It> ly, ry, stripy;
4e     P splitp = *xa[split];
c9     double splitx = splitp.x;
ef     for(IIt i = ya; i != yaend; ++i) { // Divide
9c         if(*i != xa[split] && (**i-splitp).dist2() < 1e-12)
2d             return i1 = *i, i2 = xa[split], 0;// nasty special case!
a1         if (**i < splitp) ly.push_back(*i);
c0         else ry.push_back(*i);
7d     } // assert((signed)lefty.size() == split)
8c     It j1, j2; // Conquer
3e     double a = cp_sub(ly.begin(), ly.end(), xa, i1, i2);
05     double b = cp_sub(ry.begin(), ry.end(), xa+split, j1, j2);
61     if(b < a) a = b, i1 = j1, i2 = j2;
82     double a2 = a*a;
ef     for(IIt i = ya; i != yaend; ++i) { // Create strip (y-sorted)
c6         double x = (*i)->x;
57         if(x >= splitx-a && x <= splitx+a) stripy.push_back(*i);
7d     }
51     for(IIt i = stripy.begin(); i != stripy.end(); ++i) {
03         const P &p1 = **i;
1a         for(IIt j = i+1; j != stripy.end(); ++j) {
92             const P &p2 = **j;
a2             if(p2.y-p1.y > a) break;
48             double d2 = (p2-p1).dist2();
1a             if(d2 < a2) i1 = *i, i2 = *j, a2 = d2;
57     }   }
28     return sqrt(a2);
7d }
05 template<class It> // It is random access iterators of point<T>
09 double closestpair(It begin, It end, It &i1, It &i2 ) {
06     vector<It> xa, ya;
8e     assert(end-begin >= 2);
04     for (It i = begin; i != end; ++i)
62         xa.push_back(i), ya.push_back(i);
26     sort(xa.begin(), xa.end(), it_less<It>);
06     sort(ya.begin(), ya.end(), y_it_less<It>);
6a     return cp_sub(ya.begin(), ya.end(), xa.begin(), i1, i2);
7d }
```

KD tree

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

```
ec #include "Point.h"
61 typedef long long T;
a1 typedef Point<T> P;
a0 const T INF = numeric_limits<T>::max();
c5 bool on_x(const P& a, const P& b) { return a.x < b.x; }
d6 bool on_y(const P& a, const P& b) { return a.y < b.y; }
c1 struct Node {
cd     P pt; // if this is a leaf, the single point in it
1e     T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
5d     Node *first = 0, *second = 0;
64     T distance(const P& p) { // min squared distance to a point
77         T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
41         T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
31         return (P(x,y) - p).dist2();
7d     }
54     Node(vector<P>&& vp) : pt(vp[0]) {
b1         for (P p : vp) {
33             x0 = min(x0, p.x); x1 = max(x1, p.x);
a3             y0 = min(y0, p.y); y1 = max(y1, p.y);
7d         }
da         if (vp.size() > 1) {
               // split on x if the box is wider than high (not best
                   heuristic...)
74             sort(vp.begin(), vp.end(), 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)
47             int half = vp.size()/2;
84             first = new Node({vp.begin(), vp.begin() + half});
a0             second = new Node({vp.begin() + half, vp.end()});
7d         }
7d     }
6c };
fb struct KDTree {
f3     Node* root;
8c     KDTree(const vector<P>& vp) : root(new Node({vp.begin(), vp.end()}))
           {}
60     pair<T, P> search(Node *node, const P& p) {
8e         if (!node->first) {
               // uncomment if we should not find the point itself:
               // if (p == node->pt) return {INF, P()};
fd             return make_pair((p - node->pt).dist2(), node->pt);
7d         }
70         Node *f = node->first, *s = node->second;
76         T bfirst = f->distance(p), bsec = s->distance(p);
a9         if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);
           // search closest side first, other side if needed
6a         auto best = search(f, p);
2c         if (bsec < best.first)
24             best = min(best, search(s, p));
60         return best;
7d     }
       // find nearest point to a point, and its squared distance
       // (requires an arbitrary operator< for Point)
ad     pair<T, P> nearest(const P& p) {
1a         return search(root, p);
7d     }
6c };
```

Delaunay triangulation

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

```
ec #include "Point.h"
ed #include "3dHull.h"
b0 template<class P, class F>
35 void delaunay(vector<P>& ps, F trifun) {
8d     if (ps.size() == 3) { int d = (ps[0].cross(ps[1], ps[2]) < 0);
25         trifun(0,1+d,2-d); }
4e     vector<P3> p3;
90     for(auto& p : ps) p3.emplace_back(p.x, p.y, p.dist2());
01     if (ps.size() > 3) for(auto& t : hull3d(p3)) if ((p3[t.b]-p3[t.a]).
83             cross(p3[t.c]-p3[t.a]).dot(P3(0,0,1)) < 0)
da         trifun(t.a, t.c, t.b);
7d }
```

3D point

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

```
62 template <class T> struct Point3D {
25     typedef Point3D P;
dd     typedef const P& R;
e9     T x, y, z;
24     explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
70     bool operator<(R p) const {
3d         return tie(x, y, z) < tie(p.x, p.y, p.z); }
fc     bool operator==(R p) const {
c7         return tie(x, y, z) == tie(p.x, p.y, p.z); }
4f     P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
40     P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
42     P operator*(T d) const { return P(x*d, y*d, z*d); }
d8     P operator/(T d) const { return P(x/d, y/d, z/d); }
46     T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
e6     P cross(R p) const {
7d         return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
7d     }
77     T dist2() const { return x*x + y*y + z*z; }
4b     double dist() const { return sqrt((double)dist2()); }
       //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
3c     double phi() const { return atan2(y, x); }
       //Zenith angle (latitude) to the z-axis in interval [0, pi]
29     double theta() const { return atan2(sqrt(x*x+y*y),z); }
```

```
8c      P unit() const { return *this/(T)dist(); } //makes dist()=1
        //returns unit vector normal to *this and p
11      P normal(P p) const { return cross(p).unit(); }
        //returns point rotated 'angle' radians ccw around axis
d0      P rotate(double angle, P axis) const {
f8          double s = sin(angle), c = cos(angle); P u = axis.unit();
a6          return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
7d      }
6c  };
```

Polyhedron volume

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

```
9a  template <class V, class L>
bb  double signed_poly_volume(const V& p, const L& trilist) {
65    double v = 0;
0f    for(auto& i : trilist) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
b8    return v / 6;
7d  }
```

3D hull

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

```
ee  #include "Point3D.h"

aa  typedef Point3D<double> P3;

e3  struct PR {
88      void ins(int x) { (a == -1 ? a : b) = x; }
fd      void rem(int x) { (a == x ? a : b) = -1; }
c7      int cnt() { return (a != -1) + (b != -1); }
11      int a, b;
6c  };

a2  struct F { P3 q; int a, b, c; };

71  vector<F> hull3d(const vector<P3>& A) {
21      assert(A.size() >= 4);
95      vector<vector<PR>> E(A.size(), vector<PR>(A.size(), {-1, -1}));
4b  #define E(x,y) E[f.x][f.y]
a6      vector<F> FS;
2b      auto mf = [&](int i, int j, int k, int l) {
bd          P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
da          if (q.dot(A[l]) > q.dot(A[i]))
ca              q = q * -1;
04          F f{q, i, j, k};
81          E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
b4          FS.push_back(f);
6c      };
9d      rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
a9          mf(i, j, k, 6 - i - j - k);

b8      rep(i,4,A.size()) {
6c          rep(j,0,FS.size()) {
97              F f = FS[j];
04              if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
1c                  E(a,b).rem(f.c);
7e                  E(a,c).rem(f.b);
59                  E(b,c).rem(f.a);
a1                  swap(FS[j--], FS.back());
ce                  FS.pop_back();
7d              }
7d          }
cb          int nw = FS.size();
ba          rep(j,0,nw) {
97              F f = FS[j];
02  #define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
12              C(a, b, c); C(a, c, b); C(b, c, a);
7d          }
7d      }
f0      for(auto& it : FS) if ((A[it.b] - A[it.a]).cross(
b4          A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
c8      return FS;
6c  };
```

Spherical distance

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 lines. $dx \cdot radius$ is then the difference between the two points in the x direction and $d \cdot radius$ is the total distance between the points.

```
f3  double sphericalDistance(double f1, double t1,
86          double f2, double t2, double radius) {
41      double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
57      double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
65      double dz = cos(t2) - cos(t1);
a8      double d = sqrt(dx*dx + dy*dy + dz*dz);
b6      return radius*2*asin(d/2);
7d  }
```

Strings

KMP

Description: $pi[x]$ computes the length of the longest prefix of s that ends at x, other than $s[0..x]$ itself. This is used by find to find all occurences of a string.
Time: $\mathcal{O}(|pattern|)$ for pi, $\mathcal{O}(|word| + |pattern|)$ for find
Usage: `vector<int> p = pi(pattern); vector<int> occ = find(word, p);`

```
5f  vector<int> pi(const string& s) {
d4      vector<int> p(s.size());
5e      rep(i,1,s.size()) {
21          int g = p[i-1];
c4          while (g && s[i] != s[g]) g = p[g-1];
f9          p[i] = g + (s[i] == s[g]);
7d      }
5e      return p;
7d  }

45  vector<int> match(const string& s, const string& pat) {
a1      vector<int> p = pi(pat + '\0' + s), res;
f0      rep(i,p.size()-s.size(),p.size())
ae          if (p[i] == pat.size()) res.push_back(i - 2 * pat.size());
b1      return res;
7d  }
```

Longest palindrome

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: $\mathcal{O}(N)$

```
63  void manacher(const string& s) {
a8      int n = s.size();
a4      vector<int> p[2] = {vector<int>(n+1), vector<int>(n)};
b0      rep(z,0,2) for (int i=0,l=0,r=0; i < n; i++) {
d0          int t = r-i+!z;
04          if (i<r) p[z][i] = min(t, p[z][l+t]);
25          int L = i-p[z][i], R = i+p[z][i]-!z;
49          while (L>=1 && R+1<n && s[L-1] == s[R+1])
93              p[z][i]++, L--, R++;
64          if (R>r) l=L, r=R;
57  }}
```

Lexigographically smallest rotation

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

```
6c  int min_rotation(string s) {
c2      int a=0, N=s.size(); s += s;
91      rep(b,0,N) rep(i,0,N) {
73          if (a+i == b || s[a+i] < s[b+i]) {b += max(0, i-1); break;}
d9          if (s[a+i] > s[b+i]) { a = b; break; }
7d      }
84      return a;
7d  }
```

Suffix array

Description: Builds suffix array for a string. $a[i]$ 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 $a[0] = n$. The `lcp` function calculates longest common prefixes for neighbouring strings in suffix array. The returned vector is of size $n+1$, and $ret[0] = 0$.
Time: $\mathcal{O}(N \log^2 N)$ where N is the length of the string for creation of the SA. $\mathcal{O}(N)$ for longest common prefixes.
Memory: $\mathcal{O}(N)$

```
57  typedef pair<ll, int> pli;
8e  void count_sort(vector<pli> &b, int bits) { // (optional)
        //this is just 3 times faster than stl sort for N=10^6
13      int mask = (1 << bits) - 1;
10      rep(it,0,2) {
5d          int move = it * bits;
80          vector<int> q(1 << bits), w(q.size() + 1);
76          rep(i,0,b.size())
01              q[(b[i].first >> move) & mask]++;
64          partial_sum(q.begin(), q.end(), w.begin() + 1);
d7          vector<pli> res(b.size());
76          rep(i,0,b.size())
37              res[w[(b[i].first >> move) & mask]++] = b[i];
c3          swap(b, res);
7d      }
7d  }
ec  struct SuffixArray {
ee      vector<int> a;
d5      string s;
2e      SuffixArray(const string& _s) : s(_s + '\0') {
c0          int N = s.size();
a0          vector<pli> b(N);
09          a.resize(N);
90          rep(i,0,N) {
c5              b[i].first = s[i];
0e              b[i].second = i;
7d          }
a6          int q = 8;
ea          while ((1 << q) < N) q++;
f2          for (int moc = 0;; moc++) {
6a              count_sort(b, q); // sort(b.begin(), b.end()) can be used as
                                      well
33              a[b[0].second] = 0;
7d              rep(i,1,N)
bf                  a[b[i].second] = a[b[i - 1].second] +
2f                      (b[i - 1].first != b[i].first);
```

```cpp
            if ((1 << moc) >= N) break;
            rep(i,0,N) {
                b[i].first = (ll)a[i] << q;
                if (i + (1 << moc) < N)
                    b[i].first += a[i + (1 << moc)];
                b[i].second = i;
            }
        }
        rep(i,0,a.size()) a[i] = b[i].second;
    }
    vector<int> lcp() {
        // longest common prefixes: res[i] = lcp(a[i], a[i-1])
        int n = a.size(), h = 0;
        vector<int> inv(n), res(n);
        rep(i,0,n) inv[a[i]] = i;
        rep(i,0,n) if (inv[i] > 0) {
            int p0 = a[inv[i] - 1];
            while (s[i + h] == s[p0 + h]) h++;
            res[inv[i]] = h;
            if(h > 0) h--;
        }
        return res;
    }
};
```

Suffix tree

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

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

    // example: find longest common substring (uses ALPHA = 28)
    pair<int,int> 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 pair<int,int> LCS(string s, string t) {
        SuffixTree st(s + (char)('z' + 1) + t + (char)('z' + 2));
        st.lcs(0, s.size(), s.size() + 1 + t.size(), 0);
        return st.best;
    }
};
```

String hashing

Description: Various self-explanatory methods for string hashing.

```cpp
typedef unsigned long long H;
static const H C = 123891739; // arbitrary

// Arithmetic mod 2^64-1. 5x slower than mod 2^64 and more
// code, but works on evil test data (e.g. Thue-Morse).
// "typedef H K;" instead if you think test data is random.
struct K {
    typedef __uint128_t H2;
    H x; K(H x=0) : x(x) {}
    K operator+(K o){ return x + o.x + H(((H2)x + o.x)>>64); }
    K operator*(K o){ return K(x*o.x)+ H(((H2)x * o.x)>>64); }
    H operator-(K o) { K a = *this + ~o.x; return a.x + !~a.x; }
};
struct HashInterval {
    vector<K> ha, pw;
    HashInterval(string& str) : ha(str.size()+1), pw(ha) {
        pw[0] = 1;
        rep(i,0,str.size())
            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 (str.size() < length) return {};
    K h = 0, pw = 1;
    rep(i,0,length)
        h = h * C + str[i], pw = pw * C;
    vector<H> ret = {h - 0};
    rep(i,length,str.size()) {
        ret.push_back(h * C + str[i] - pw * str[i-length]);
        h = ret.back();
    }
    return ret;
}

H hashString(string& s) {
    K h = 0;
    for(auto& c : s) h = h * C + c;
    return h - 0;
}
```

Aho-Corasick

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: Function 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 $\mathcal{O}(NM)$.

```cpp
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;
        for(auto& c : s) {
            int& m = N[n].next[c - first];
            if (m == -1) { n = m = N.size(); 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,pat.size()) insert(pat[i], i);
        N[0].back = N.size();
        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);
                }
            }
        }
    }
    vector<int> find(string word) {
        int n = 0;
        vector<int> res; // ll count = 0;
        for(auto& c : word) {
            n = N[n].next[c - first];
            res.push_back(N[n].end);
            // count += N[n].nmatches;
        }
        return res;
    }
    vector<vector<int>> findAll(vector<string>& pat, string word) {
```

```
40         vector<int> r = find(word);
1d         vector<vector<int>> res(word.size());
3f         rep(i,0,word.size()) {
24             int ind = r[i];
63             while (ind != -1) {
6e                 res[i - pat[ind].size() + 1].push_back(ind);
b5                 ind = backp[ind];
7d             }
7d         }
b1         return res;
7d     }
6c };
```

Various

Bit hacks

Description: Various bit manipulation functions/snippets.

```
2a int lowestSetBit(int x) { return x & -x }

29 void forAllSubsetMasks(int m) { // Including m itself
c9     for (int x = m; x; --x &= m) { /* ... */ }
7d }

09 int nextWithSamePopcount(int x) { // 3->5->6->9->10->12->17...
8d     int c = x&-x, r = x+c;
a4     return (((r^x) >> 2)/c) | r;
7d }

   // For each mask, compute sum of values D[1<<i] of its set bits i
da void sumsOfSubsets() {
d2     int K = 3;
5c     vector<int> D(1<<K, 0);
19     D[1] = 4; D[2] = 3; D[4] = 8;
f1     rep(b,0,K) rep(i,0,(1 << K)) if (i & 1 << b) D[i] += D[i^(1 << b)];
7d }
```

Closest lower element in a set

Description: Functions to get the closest lower element in a set. Given S, k, returns $\max\{x \mid x \in S; x \leq k\}$, or $-\infty$ if there is no suitable element. Strict version replaces $\leq$ with $<$.

```
43 const ll INF = 1e18;

76 ll closestLower(set<ll>& s, ll k) {
5d     auto it = s.upper_bound(k);
a1     return (it == s.begin()) ? -INF : *(--it);
7d }

4c ll closestLowerStrict(set<ll>& s, ll k) {
a5     auto it = s.lower_bound(k);
a1     return (it == s.begin()) ? -INF : *(--it);
7d }
```

Coordinate compression

Description: Compress vector v into v' such that $v_i < v_j \iff v'_i < v'_j$ and elements are integers bounded by $0 \leq v'_i < |v|$. Mutates v.

Usage: `vector<ll> v = {6, 2, -3, 2};`
`compress(v); // v == {2, 1, 0, 1}`

```
7b void compress(vector<ll>& v) {
67     vector<ll> w = v;
68     sort(w.begin(), w.end());
30     w.erase(unique(w.begin(), w.end()), w.end());
48     for(auto& x : v)
b1         x = lower_bound(w.begin(), w.end(), x) - w.begin();
7d }
```

Interval container

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

```
82 template <class T>
0b auto addInterval(set<pair<T, T>>& is, T L, T R) {
2c     if (L == R) return is.end();
e8     auto it = is.lower_bound({L, R}), before = it;
bc     while (it != is.end() && it->first <= R) {
44         R = max(R, it->second);
e0         before = it = is.erase(it);
7d     }
3a     if (it != is.begin() && (--it)->second >= L) {
d1         L = min(L, it->first);
44         R = max(R, it->second);
64         is.erase(it);
7d     }
a1     return is.insert(before, {L,R});
6c };

82 template <class T>
9e void removeInterval(set<pair<T, T>>& is, T L, T R) {
5e     if (L == R) return;
69     auto it = addInterval(is, L, R);
32     T r2 = it->second;
e2     if (it->first == L) is.erase(it);
86     else (T&)it->second = L;
ef     if (R != r2) is.emplace(R, r2);
6c };
```

Interval cover

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

```
82 template<class T>
02 vector<int> cover(pair<T, T> G, vector<pair<T, T>> I) {
fd     vector<int> S(I.size()), R;
eb     iota(S.begin(), S.end(), 0);
5d     sort(S.begin(), S.end(), [&](int a, int b) { return I[a] < I[b]; });
6e     T cur = G.first;
9d     int at = 0;
ff     while (cur < G.second) { // (A)
85         pair<T, int> mx = make_pair(cur, -1);
b4         while (at < I.size() && I[S[at]].first <= cur) {
32             mx = max(mx, make_pair(I[S[at]].second, S[at]));
10             at++;
7d         }
38         if (mx.second == -1) return {};
67         cur = mx.first;
d1         R.push_back(mx.second);
7d     }
a7     return R;
7d }
```

Split function into constant intervals

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.

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

Usage: `constantIntervals(0, v.size(), [\&](int x){return v[x];}, [\&](int lo, int hi, T val){...});`

```
84 template<class F, class G, class T>
1d void rec(int from, int to, F f, G g, int& i, T& p, T q) {
a6     if (p == q) return;
48     if (from == to) {
d2         g(i, to, p);
ed         i = to; p = q;
71     } else {
8e         int mid = (from + to) >> 1;
0d         rec(from, mid, f, g, i, p, f(mid));
a7         rec(mid+1, to, f, g, i, p, q);
7d     }
7d }
73 template<class F, class G>
d2 void constantIntervals(int from, int to, F f, G g) {
4d     if (to <= from) return;
86     int i = from; auto p = f(i), q = f(to-1);
60     rec(from, to-1, f, g, i, p, q);
0d     g(i, to, q);
7d }
```

Divide and conquer DP

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

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

```
f6 struct DP { // Modify at will:
a3     int lo(int ind) { return 0; }
3c     int hi(int ind) { return ind; }
18     ll f(int ind, int k) { return dp[ind][k]; }
9c     void store(int ind, int k, ll v) { res[ind] = pair<int,int>(k, v); }

5e     void rec(int L, int R, int LO, int HI) {
34         if (L >= R) return;
fa         int mid = (L + R) >> 1;
ab         pair<ll, int> best(LLONG_MAX, LO);
43         rep(k, max(LO,lo(mid)), min(HI,hi(mid)))
ff             best = min(best, make_pair(f(mid, k), k));
7f         store(mid, best.second, best.first);
62         rec(L, mid, LO, best.second+1);
cf         rec(mid+1, R, best.second, HI);
7d     }
06     void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
6c };
```

Knuth DP optimization

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

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

Ternary search

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

```
d1 template<class F>
df int ternSearch(int a, int b, F f) {
f8     assert(a <= b);
eb     while (b - a >= 5) {
c1         int mid = (a + b) / 2;
1c         if (f(mid) < f(mid+1)) // (A)
e6             a = mid;
7e         else
da             b = mid+1;
7d     }
2c     rep(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
84     return a;
7d }
```

Longest common subsequence

Description: Finds the longest common subsequence.
Time: $\mathcal{O}(NM)$, where N and M are the lengths of the sequences.
Memory: $\mathcal{O}(NM)$.

```
f8 template <class T> T lcs(const T &X, const T &Y) {
b5     int a = X.size(), b = Y.size();
dd     vector<vector<int>> dp(a+1, vector<int>(b+1));
47     rep(i,1,a+1) rep(j,1,b+1)
8f         dp[i][j] = X[i-1]==Y[j-1] ? dp[i-1][j-1]+1 :
9e             max(dp[i][j-1],dp[i-1][j]);
d0     int len = dp[a][b];
a8     T ans(len,0);
e4     while(a && b)
de         if(X[a-1]==Y[b-1]) ans[--len] = X[--a], --b;
a4         else if(dp[a][b-1]>dp[a-1][b]) --b;
e0         else --a;
7d     return ans;
7d }
```

Longest increasing subsequence

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

```
a0 template<class I> vector<int> lis(vector<I> S) {
69     vector<int> prev(S.size());
c9     typedef pair<I, int> p;
70     vector<p> res;
96     rep(i,0,S.size()) {
ef         p el { S[i], i };
           //S[i]+1 for non-decreasing
52         auto it = lower_bound(res.begin(), res.end(), p { S[i], 0 });
fb         if (it == res.end()) res.push_back(el), it = --res.end();
fc         *it = el;
32         prev[i] = it==res.begin() ?0:(it-1)->second;
7d     }
34     int L = res.size(), cur = res.back().second;
57     vector<int> ans(L);
2e     while (L--) ans[L] = cur, cur = prev[cur];
7d     return ans;
7d }
```