



Universidade Federal de Pernambuco

# las4s e pelados

Icaro Copo Papel Nunes, Joao Pou Grangeiro, Pedro Grisi

2026-02-22

## 1 Contest

## 2 Theoretical

## 3 Data structures

## 4 Numerical

## 5 Number theory

## 6 Combinatorial

## 7 Graph

## 8 Geometry

## 9 Strings

## 10 Various

# Contest (1)

### template.cpp

9 lines

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

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

### .bashrc

2 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++17 \
-fsanitize=undefined,address'
```

### hash.sh

2 lines

```
# bash hash.sh file.cpp l1 l2
sed -n $2'','$3' p' $1 | sed '/^#w/d' | cpp -D -P -
fpreprocessed | tr -d '[:space:]' | md5sum | cut -c-6
```

### stressTest.sh

20 lines

```
P=code  #nude pro filename do codigo
Q=brute #nude pro filename do brute [correto]
g++ ${P}.cpp -o sol -O2 || exit 1
g++ ${Q}.cpp -o brt -O2 || exit 1
g++ gen.cpp -o gen -O2 || exit 1
for ((i = 1; ; i++)) do
    echo $i
    ./gen $i > in
    ./sol < in > out
    ./brt < in > out2
    if (! cmp -s out out2) then
        echo "--> entrada:"
        cat in
        echo "--> saida code:"
        cat out
```

```
1     echo "--> saida brute:"
1     cat out2
1     break;
1   fi
done
5
paperStress.py
26 lines
7
927 import random
a1a import subprocess
5c9 MAX_N = 100
b5d def gen_case() -> str:
c7e     return f"1\n"
11
94a random.seed((1 << 9) | 31)
12
a22 for i in range(100):
d19     print(), print()
a3f     case = gen_case()
266     print(f"Test #{i+1}: ")
ce5     print(case)
23
d41     # test bruteforce
f60     bf = subprocess.run(['out/b'], input=case, encoding='
ascii', capture_output=True)
d41     # test solution
37c     sol = subprocess.run(['out/m'], input=case, encoding='
ascii', capture_output=True)
d55     bf_res = bf.stdout
af9     sol_res = sol.stdout
6b6     print(f"bruteforce {bf_res}, solution {sol_res}")
508     if bf_res == sol_res:
dd4         print("accepted")
f68     else:
ef2         print("WA")
1cb     break
```

### troubleshoot.txt

52 lines

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

Wrong answer:  
Print your solution! Print debug output, as well.  
Are you clearing all data structures between test cases?  
Can your algorithm handle the whole range of input?  
Read the full problem statement again.  
Do you handle all corner cases correctly?  
Have you understood the problem correctly?  
Any uninitialized variables?  
Any overflows?  
Confusing N and M, i and j, etc.?  
Are you sure your algorithm works?  
What special cases have you not thought of?  
Are you sure the STL functions you use work as you think?  
Add some assertions, maybe resubmit.  
Create some testcases to run your algorithm on.  
Go through the algorithm for a simple case.  
Go through this list again.  
Explain your algorithm to a teammate.  
Ask the teammate to look at your code.  
Go for a small walk, e.g. to the toilet.  
Is your output format correct? (including whitespace)  
Rewrite your solution from the start or let a teammate do it.

Runtime error:

Have you tested all corner cases locally?  
Any uninitialized variables?  
Are you reading or writing outside the range of any vector?  
Any assertions that might fail?  
Any possible division by 0? (mod 0 for example)  
Any possible infinite recursion?  
Invalidated pointers or iterators?  
Are you using too much memory?  
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:  
Do you have any possible infinite loops?  
What is the complexity of your algorithm?  
Are you copying a lot of unnecessary data? (References)  
How big is the input and output? (consider scanf)  
Avoid vector, map. (use arrays/unordered\_map)  
What do your teammates think about your algorithm?

Memory limit exceeded:  
What is the max amount of memory your algorithm should need?  
Are you clearing all data structures between test cases?

# Theoretical (2)

## 2.1 Mathematics

### 2.1.1 Recurrences

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

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

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

### 2.1.2 Trigonometry

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

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

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

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

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

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

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

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

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

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

### 2.1.3 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):

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

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

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

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

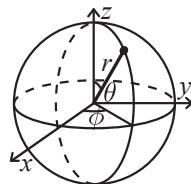
#### Quadrilaterals

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

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

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

#### Spherical coordinates



$$x = r \sin \theta \cos \phi$$

$$y = r \sin \theta \sin \phi \quad \theta = \arccos(z/\sqrt{x^2 + y^2 + z^2})$$

$$z = r \cos \theta$$

#### Pick's Theorem

The area of a simple polygon whose vertices have integer coordinates is:

$$A = I + \frac{B}{2} - 1$$

### template .bashrc hash stressTest paperStress troubleshoot

where  $I$  is the number of interior integer points, and  $B$  is the number of integer points in the border of the polygon.

#### Two Ears Theorem

Every simple polygon with more than 3 vertices has at least two non-overlapping ears (a ear is a vertex whose diagonal induced by its neighbors which lies strictly inside the polygon). Equivalently, every simple polygon can be triangulated.

#### 2.1.4 Derivatives/Integrals

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

Integration by parts:

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

#### 2.1.5 Sums

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

$$\begin{aligned} 1^2 + 2^2 + \cdots + n^2 &= \frac{n(2n+1)(n+1)}{6} \\ 1^3 + 2^3 + \cdots + n^3 &= \frac{n^2(n+1)^2}{4} \\ 1^4 + 2^4 + \cdots + n^4 &= \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} \\ \sum_{i=0}^n ic^i &= \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \neq 1 \end{aligned}$$

$$g_k(n) = \sum_{i=1}^n i^k = \frac{1}{k+1} \left( n^{k+1} + \sum_{j=1}^k \binom{k+1}{j+1} (-1)^{j+1} g_{k-j}(n) \right)$$

#### 2.1.6 Series

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

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

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

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

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

$$\sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad |c| < 1$$

$$(1+x)^n = \sum_{i=0}^n \binom{n}{i} x^i$$

$$\frac{1}{1-x} = \sum_{i=0}^{\infty} x^i, \quad (-1 < x < 1)$$

$$\frac{1}{(1-x)^n} = \sum_{i=0}^{\infty} \binom{n+i-1}{n-1} x^i, \quad (-1 < x < 1)$$

#### 2.1.7 Probability theory

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

$\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$  where  $\sigma$  is the standard deviation. If  $X$  is instead continuous it will have a probability density function  $f_X(x)$  and the sums above will instead be integrals with  $p_X(x)$  replaced by  $f_X(x)$ .

Expectation is linear:

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

For independent  $X$  and  $Y$ ,

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

#### Binomial distribution

The number of successes in  $n$  independent yes/no experiments, each which yields success with probability  $p$  is

$\text{Bin}(n, p)$ ,  $n = 1, 2, \dots$ ,  $0 \leq p \leq 1$ .

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

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

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

#### First success distribution

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

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

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

## Poisson distribution

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

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

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

## 2.2 Combinatorial

### 2.2.1 Binomial Identities

$$\begin{aligned} \binom{n-1}{k} - \binom{n-1}{k-1} &= \frac{n-2k}{k} \binom{n}{k} & \binom{n}{h} \binom{n-h}{k} &= \binom{n}{k} \binom{n-k}{h} \\ \sum_{k=0}^n k \binom{n}{k} &= n 2^{n-1} & \sum_{k=0}^n k^2 \binom{n}{k} &= (n+n^2) 2^{n-2} \\ \sum_{j=0}^k \binom{m}{j} \binom{n-m}{k-j} &= \binom{n}{k} & \sum_{j=0}^m \binom{m}{j}^2 &= \binom{2m}{m} \\ \sum_{m=0}^n \binom{m}{j} \binom{n-m}{k-j} &= \binom{n+1}{k+1} & \sum_{m=0}^n \binom{m}{k} &= \binom{n+1}{k+1} \\ \sum_{r=0}^m \binom{n+r}{r} &= \binom{n+m+1}{m} & \sum_{k=0}^n \binom{n-k}{k} &= \text{Fib}(n+1) \\ \sum_{k=0}^n \binom{r}{k} \binom{s}{n-k} &= \binom{r+s}{n} \end{aligned}$$

### 2.2.2 Permutations

#### Factorial

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

#### Cycles

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

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

#### Derangements

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

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

## Burnside's lemma

Counts the number of distinct colorings of an object under symmetry.

$$\frac{1}{|G|} \sum_{g \in G} k^{\text{cyc}(g)},$$

where  $G$  is the symmetry group,  $k$  the number of colors, and  $\text{cyc}(g)$  the number of cycles induced by  $g$ .

Example: number of ways to color a necklace with  $n$  beads using  $k$  colors (rotations only):

$$g(n) = \frac{1}{n} \sum_{i=0}^{n-1} k^{\text{gcd}(n, i)}$$

where rotation  $i$  shifts the necklace by  $i$  positions.

### 2.2.3 Partitions and subsets

#### Partition function

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

$$\begin{aligned} p(0) &= 1, \quad p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k-1)/2) \\ p(n) &\sim 0.145/n \cdot \exp(2.56\sqrt{n}) \\ \begin{array}{c|cccccccccc} n & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 20 & 50 & 100 \\ \hline p(n) & 1 & 1 & 2 & 3 & 5 & 7 & 11 & 15 & 22 & 30 & 627 & \sim 2e5 & \sim 2e8 \end{array} \end{aligned}$$

#### Lucas' Theorem

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

### 2.2.4 Sum of Binomials (FFT)

Goal: Given freq. array  $C$ , compute  $\text{Ans}[k] = \sum_i C[i] \binom{i}{k}$  for all  $k$ . Rewrite:  $\text{Ans}[k] = \frac{1}{k!} \sum_i (C[i] \cdot i!) \frac{1}{(i-k)!}$ .

- Construct  $P$  where  $P[i] = C[i] \cdot i!$
- Construct  $Q$  where  $Q[i] = (i!)^{-1}$
- Reverse  $Q$  (to handle the  $i - k$  subtraction).
- Multiply  $R = NTT(P, Q)$ .
- Result:  $\text{Ans}[k] = R[k + |Q| - 1] \cdot \frac{1}{k!}$ .

### 2.2.5 General purpose numbers

#### Bernoulli numbers

EGF of Bernoulli numbers is  $B(t) = \frac{t}{e^t - 1}$  (FFT-able).

$$B[0, \dots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \dots]$$

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

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

#### Stirling numbers of the first kind

Number of permutations on  $n$  items with  $k$  cycles.

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

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

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

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

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

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

#### 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.  $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \dots$ . For  $p$  prime,

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

#### Labeled unrooted trees

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

## Catalan numbers

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

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

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

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

## 2.3 Number Theory

### 2.3.1 Bézout's identity

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

$$ax + by = d$$

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

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

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

### 2.3.3 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  $n < 1e10$ , 6700 for  $n < 1e12$ , 200 000 for  $n < 1e19$ .

### 2.3.4 Möbius Function

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

Möbius Inversion:

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

Other useful formulas/forms:

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

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

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

### 2.3.5 Theorems

**Goldbach's conjecture:** Every even integer  $n > 2$  can be written as  $n = a + b$  with  $a, b$  prime.

**Legendre's conjecture:** There is always at least one prime between  $n^2$  and  $(n+1)^2$ .

**Lagrange's four-square theorem:** Every positive integer can be written as

$$n = a^2 + b^2 + c^2 + d^2.$$

**Zeckendorf's theorem:** Every integer  $n \geq 1$  has a unique representation as a sum of non-consecutive Fibonacci numbers:

$$n = F_{i_1} + F_{i_2} + \dots + F_{i_k}, \quad i_j - i_{j+1} \geq 2.$$

**Euclid's formula (primitive Pythagorean triples):** The Pythagorean triples are uniquely generated by

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

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

**Wilson's theorem:**  $n$  is prime iff

$$(n-1)! \equiv -1 \pmod{n}.$$

**Chicken McNugget theorem:** For coprime  $n, m$ , the largest integer not representable as  $an + bm$  (with  $a, b \geq 0$ ) is

$$nm - n - m.$$

There are  $\frac{(n-1)(m-1)}{2}$  non-representable integers, and for each pair  $(k, nm - n - m - k)$  exactly one is representable.

## 2.4 Graphs

### 2.4.1 Flows and Matching

#### Hall's Theorem

In bipartite graphs, there exists a perfect matching covering the entire side  $X$  if and only if for every subset  $Y \subseteq X$ ,

$$|Y| \leq |N(Y)|,$$

where  $N(Y)$  denotes the set of neighbors of  $Y$ .

## König's Theorem

In a bipartite graph, the size of a Minimum Vertex Cover is equal to the size of a Maximum Matching. A Minimum Vertex Cover is a minimum set of vertices such that every edge of the graph has at least one endpoint in the set.

As a consequence,

$$n - \text{Maximum Matching} = \text{Maximum Independent Set},$$

where a Maximum Independent Set is the largest set of vertices with no edges between them.

**Recovering the Minimum Vertex Cover** Given a maximum matching in a bipartite graph  $(X, Y)$ :

- Construct the residual graph by orienting:
  - non-matching edges from  $X$  to  $Y$ ;
  - matching edges from  $Y$  to  $X$ .
- Perform a BFS or DFS starting from all free (unmatched) vertices in  $X$ .
- Let  $Z_X$  be the set of reachable vertices in  $X$ , and  $Z_Y$  the set of reachable vertices in  $Y$ .

The Minimum Vertex Cover is given by:

$$(X \setminus Z_X) \cup Z_Y.$$

#### Node-Disjoint Path Cover

A node-disjoint path cover is a set of paths such that each vertex belongs to exactly one path.

In a directed acyclic graph (DAG),

$$\text{Minimum Node-Disjoint Path Cover} = n - \text{Maximum Matching}.$$

The construction is as follows: for each vertex  $u$ , create a copy  $u'$ . Add an edge  $u \rightarrow v'$  if there exists an edge  $u \rightarrow v$  in the original graph.

#### Recovering the Paths

- Vertices that do not appear as destinations in the matching are starting points of paths.
- Each matching edge  $u \rightarrow v'$  corresponds to an edge  $u \rightarrow v$  in the original DAG.
- Following these edges reconstructs all paths of the path cover.

## General Path Cover

A general path cover is a path cover where a vertex may belong to more than one path.

In a DAG, the construction is similar to the node-disjoint case, but an edge  $u \rightarrow v'$  exists if there is a path from  $u$  to  $v$  in the original graph.

**Recovering the Cover** The vertices can be grouped according to the edges used in the matching to form the path cover.

## Dilworth's Theorem

An antichain is a set of vertices such that there is no path between any pair of vertices in the set.

In a directed acyclic graph,

Minimum General Path Cover = Maximum Antichain.

**Recovering a Maximum Antichain** Given a minimum general path cover, selecting one vertex from each path produces a maximum antichain.

## 2.4.2 Number of Spanning Trees

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

## 2.4.3 Erdős–Gallai theorem

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

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

## 2.4.4 Planar Graphs

If  $G$  has  $k$  connected components, then  $n - m + f = k + 1$ .

## 2.5 Optimization tricks

### 2.5.1 Bit hacks

- `for (int x = m; x; x = (x - 1) &m) { ... }`  
loops over all subset masks of  $m$  (except 0).
- $c = x \& -x$ ,  $r = x + c$ ;  $((r \wedge x) \gg 2) / c$  |  $r$  is the next number after  $x$  with the same number of bits set.
- `rep(b, 0, K) rep(i, 0, (1 << K))`  
`if (i & 1 << b) D[i] += D[i ^ (1 << b)];`  
computes all sums of subsets.

## Bit Bit2d LineContainer

### 2.5.2 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize loops and optimizes floating points better.
- `#pragma GCC target ("avx2")` can double performance of vectorized code, but causes crashes on old machines.
- `#pragma GCC target ("bmi,bmi2,popcnt,lzcnt")` improve bit operations.
- `#pragma GCC optimize("unroll-loops")` self explanatory.

## 2.6 Various

### 2.6.1 Master Theorem (Simple)

$T(n) = aT(n/b) + O(n^d)$ . Compare  $a$  vs  $b^d$ :

- $a > b^d \Rightarrow O(n^{\log_b a})$  (Work at leaves dominates)
- $a = b^d \Rightarrow O(n^d \log n)$  (Work is uniform)
- $a < b^d \Rightarrow O(n^d)$  (Work at root dominates)

## Data structures (3)

### Bit.h

Description: `lower_bound` works the same as on vectors

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

```
8eb struct Bit {
406     vector<ll> bit;
1dd     Bit(int n) : bit(n + 1) {}
265     void update(int i, ll v) {
c38         for (i++; i < sz(bit); i += i & -i) bit[i] += v;
f21     }
74a     ll query(int i) {
b73         ll ret = 0;
71c         for (i++; i > 0; i -= i & -i) ret += bit[i];
edf         return ret;
e40     }
dc8     int lower_bound(ll v){ // min pos st sum[0, pos] >= v
bec         int pos = 0;
a40         for (int j=(1 << 23); j >= 1; j/=2){
3b1             if(pos+j < sz(bit) && bit[pos + j] < v){
b4e                 pos += j;
18d                 v -= bit[pos];
f6c             }
156         }
d75         return pos;
37b     }
589 };
```

### Bit2d.h

Description: Points called on the update function NEED to be on the `pts` vector parameter on build.

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

```
"Bit2d.h"
9c0 struct Bit2d {
a37     vector<vector<int>> ys;
fe8     vector<Bit> bit;
543     vector<int> cmp_x;
425     Bit2d(){}
521     void put(int x, int y) {
005         for (x++; x < sz(ys); x += x & -x) ys[x].push_back(y);
f3c     }
```

```
ce0     int id(const vector<int> &v, int y) {
1e9         return (upper_bound(all(v), y) - v.begin()) - 1;
19a     }
7ff     void build(vector<pii> pts) {
3cb         sort(all(pts));
f99         for(auto p : pts) cmp_x.push_back(p.first);
9a7         cmp_x.erase(unique(all(cmp_x)), cmp_x.end());
f82         ys.resize(cmp_x.size() + 1);
94d         for(auto p : pts) put(id(cmp_x, p.first), p.second);
310         for(auto &v:ys)sort(all(v)), bit.emplace_back(sz(v));
a01     }
767         void update(int x, int y, int val){
f3f             x = id(cmp_x, x);
681             for(x++; x < sz(ys); x+= x&-x)
507                 bit[x].update(id(ys[x], y), val);
c88         }
d95         int query(int x, int y){
f3f             x = id(cmp_x, x);
7c9             int ret = 0;
f32             for(x++; x > 0; x-= x&-x)
ea8                 ret += bit[x].query(id(ys[x], y));
edf             return ret;
8f7         }
251         int query(int x1, int y1, int x2, int y2){
e4d             int a = query(x2, y2)-query(x2, y1-1);
7d1             return a-query(x1-1, y2)+query(x1-1, y1-1);
c33     }
5a9 };
```

### LineContainer.h

Description: Container where you can add lines of the form  $kx+m$ , and query maximum values at points  $x$ . Useful for dynamic programming (“convex hull trick”).

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

```
8ec1c7, 32 lines
72c struct Line {
3e2     mutable ll k, m, p;
ca5     bool operator<(const Line& o) const { return k < o.k; }
abf     bool operator<=(ll x) const { return p < x; }
7e3     };

781 struct LineContainer : multiset<Line, less<> {
// (for doubles, use inf = 1/.0, div(a,b) = a/b)
fd2     static const ll inf = LLONG_MAX;
33a     ll div(ll a, ll b) { // floored division
10f         return a / b - ((a ^ b) < 0 && a % b); }
a1c     bool isect(iterator x, iterator y) {
a95         if (y == end()) return x->p = inf, 0;
9cb         if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
591         else x->p = div(y->m - x->m, x->k - y->k);
870         return x->p >= y->p;
2fa     }
a0c     void add(ll k, ll m) {
116         auto z = insert({k, m, 0}), y = z++, x = y;
7b1         while (isect(y, z)) z = erase(z);
d94         if (x != begin() && isect(--x, y))
c07             isect(x, y = erase(y));
57d         while ((y = x) != begin() && (--x)->p >= y->p)
774             isect(x, erase(y));
086     }
11 query(ll x) {
229         assert(!empty());
7d1         auto l = *lower_bound(x);
96a         return l.k * x + l.m;
d21     }
577 };
```

## Mo.h

**Description:** For subtree queries, perform an Euler tour and map each node u to the interval  $[tin[u], tin[u] + subtree\_size[u] - 1]$ . A subtree query becomes a range query over this interval.  
 For path queries between nodes U and V, Let U be the closest to the root. If V lies in U's subtree, the path corresponds to the interval  $[tin[U], tin[V]]$ . Otherwise, the path corresponds to the interval  $[min(tout[U], tout[V]), max(tin[U], tin[V])]$ .

In both cases, nodes on the U-V path appear exactly once in the interval, while all other nodes appear either 0 or 2 times.

**Usage:** `queries.push(Query(l, r, index of query))`, intervals are  $[l, r]$

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

fb7161, 44 lines

```
626 inline int64_t hilOrd(int x, int y, int pow, int rot) {
51a   if (pow == 0) return 0;
a6e   int hpow = 1 << (pow - 1);
01f   int seg = (x < hpow) ? ((y < hpow) ? 0 : 3) : ((y < hpow)
    ) ? 1 : 2;
e08   seg = (seg + rot) & 3;
669   const int rotDelta[4] = { 3, 0, 0, 1 };
d0b   int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
115   int nrot = (rot + rotDelta[seg]) & 3;
fba   int64_t sub = int64_t(1) << (2 * pow - 2);
65b   int64_t ans = seg * sub;
1ae   int64_t add = hilOrd(nx, ny, pow - 1, nrot);
ff7   ans += (seg == 1 || seg == 2) ? add : (sub - add - 1);
ba7   return ans;
ec4 }

670 struct Query {
738   int l, r, idx;
ce8   int64_t ord;
36f   Query(int l, int r, int idx) : l(l), r(r), idx(idx) {
6c4     ord = hilOrd(l, r, 21, 0);
926   }
847   bool operator < (const Query& other) const {
328     return ord < other.ord;
e05   }
315 };

240 vector<Query> queries;
4d5 int ans[m];
566 void put(int x) {} // F
c29 void remove(int x) {} // F
64b int getAns() {}

1c1 void Mo() {
3d9   int l = 0, r = -1;
bfa   sort(queries.begin(), queries.end());
275   for (Query q : queries) {
482     while (l > q.l) put(--l);
fec     while (r < q.r) put(++r);
5b8     while (l < q.l) remove(l++);
9b5     while (r > q.r) remove(r--);
745     ans[q.idx] = getAns();
5a4   }
2a4 }
```

## MoUpdate.h

**Description:** Block size should be around  $(2 * N * N)^{\frac{1}{3}}$

**Usage:** intervals are  $[l, r]$ , `addQuery(l, r, number of updates happened before this query, index of query)`, `addUpdate(index of updated position, value before update, value after update)`

**Time:**  $\mathcal{O}(Q * (2 * N * N)^{\frac{1}{3}} * F)$

f8eda8, 55 lines

496 const int B = 2700;

```
247 struct MoUpdate {
670   struct Query {
fd6     int l, r, t, idx;
fc8     Query(int l, int r, int t, int idx)
      : l(l), r(r), t(t), idx(idx) {}
f51     bool operator < (const Query& p) const {
f06       if (l / B != p.l / B) return l < p.l;
e80       if (r / B != p.r / B) return r < p.r;
        return t < p.t;
      }
bc2   };
f2f   struct Upd {
f25     int i, old, now;
      Upd(int i, int old, int now) : i(i), old(old), now(now) {}
c12   };

240   vector<Query> queries;
e2b   vector<Upd> updates;

ac5   void addQuery(int l, int r, int t, int idx) {
fc9     queries.push_back(Query(l, r, t, idx));
968   void addUpdate(int i, int old, int now) {
936     updates.push_back(Upd(i, old, now));
      }

1aa   void add(int x) {} // F
598   void rem(int x) {} // F
64b   int getAns() {}
0d2   void update(int novo, int idx, int l, int r) {
2b9     if (l <= idx && idx <= r) rem(idx);
      arr[idx] = novo;
      if (l <= idx && idx <= r) add(idx);
100   }

63d   void solve() {
cb1     int l = 0, r = -1, t = 0;
bfa     sort(queries.begin(), queries.end());
275     for (Query q : queries) {
a95       while (l > q.l) add(--l);
        while (r < q.r) add(++r);
875       while (l < q.l) rem(l++);
        while (r > q.r) rem(r--);
a38       while (t < q.t) {
fda         auto u = updates[t++];
        update(u.now, u.i, l, r);
        }
        while (t > q.t) {
d53         auto u = updates[--t];
        update(u.old, u.i, l, r);
        }
      }
      ans[q.idx] = getAns();
f06   }
b09   }
d3e }
```

## MinQueue.h

40df8d, 19 lines

```
925 struct MQueue {
fdd   int tin, tout;
375   deque<pair<int, int>> dq;
1ce   MQueue() : tin(0), tout(0) {}
619   void push(int val) {
f0d     while (!dq.empty() && min(dq.back().first, val) ==
val) dq.pop_back();
        dq.push_back(pair(val, tin++));
      }
42d   void pop() {
        // assert(!dq.empty());
        if (dq.front().second == tout) dq.pop_front();
        tout++;
      }
48c
470 }
```

```
b0e   }
f46   int front() {
      // assert(!dq.empty());
      return dq.front().first;
651   }
fa2   }
40d }
```

## SegmentTree.h

**Description:** Zero-indexed max-tree. Bounds are inclusive to the left and inclusive to the right. Can be changed by modifying T, f and unit.

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

f609d9, 21 lines

```
5ae struct Tree {
ef4   typedef int T;
cbe   static constexpr T unit = INT_MIN;
e54   T f(T a, T b) { return max(a, b); } // (any associative
fn)
6cd   vector<T> s; int n;
3d2   Tree(int n = 0, T def = unit) : s(2*n, def), n(n) {}
6a3   void update(int pos, T val) {
56a     for (s[pos += n] = val; pos /= 2; )
326       s[pos] = f(s[pos * 2], s[pos * 2 + 1]);
0e9   }
b4c   T query(int b, int e) { // query [b, e]
1a3     e++;
0f9     T ra = unit, rb = unit;
fbb   for (b += n, e += n; b < e; b /= 2, e /= 2) {
e83     if (b % 2) ra = f(ra, s[b++]);
064     if (e % 2) rb = f(s[--e], rb);
561   }
cb2   return f(ra, rb);
707   }
f60 }
```

## OrderStatisticTree.h

**Description:** A set (not multiset!) with support for finding the n'th element, and finding the index of an element. To get a map, change null-type.

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

782797, 17 lines

```
c4d #include <bits/extc++.h>
0d7 using namespace __gnu_pbds;

4fc template<class T>
c20 using Tree = tree<T, null_type, less<T>, rb_tree_tag,
3a1   tree_order_statistics_node_update>;

ad0 void example() {
c6f   Tree<int> t, t2; t.insert(8);
559   auto it = t.insert(10).first;
d28   assert(it == t.lower_bound(9));
969   assert(t.order_of_key(10) == 1);
d39   assert(t.order_of_key(11) == 2);
1b7   assert(*t.find_by_order(0) == 8);
a60   t.join(t2); // merge t2 into t
9ad }
```

## PersistentSegTree.h

**Usage:** `SegP(size of the segtree, number of updates)`

roots = {0}, newRoot = update(roots.back(), ...),  
 roots.push(newRoot)

58842f, 42 lines

```
b17 struct SegP {
709   static constexpr ll neut = 0;
bf2   struct Node {
aa3     ll v; // start with neutral value
74f     int l, r;
9ef     Node(ll v=neut, int l=0, int r=0) : v(v), l(l), r(r) {}
945   }
```

```

38f    vector<Node> seg;
068    int n, CNT;
9ea    SegB(int _n, int upd): seg(20*(upd+_n)), n(_n), CNT(1){}
2ce    ll merge(ll a, ll b) { return a + b; }
c97    int update(int root, int pos, int val, int l, int r) {
ec9        int p = CNT++;
77a        seg[p] = seg[root];
893        if (l == r) {
00f            seg[p].v += val;
74e            return p;
3d7        }
ae0        int mid = (l + r) / 2;
8a3        if (pos <= mid) {
aa8            seg[p].l = update(seg[p].l, pos, val, l, mid);
583        } else seg[p].r = update(seg[p].r, pos, val, mid+1, r);

85a        seg[p].v=merge(seg[seg[p].l].v, seg[seg[p].r].v);
74e        return p;
a90    }
6a4    int query(int p, int L, int R, int l, int r) {
3c7        if (l > R || r < L) return neut;
c26        if (L <= l && r <= R) return seg[p].v;
ae0        int mid = (l + r) / 2;
864        int left = query(seg[p].l, L, R, l, mid);
195        int right = query(seg[p].r, L, R, mid + 1, r);
90a        return merge(left, right);
e77    }
304    int update(int root, int pos, int val) {
c68        return update(root, pos, val, 0, n - 1);
84e    }
7cc    int query(int root, int L, int R) {
a53        return query(root, L, R, 0, n - 1);
2f9    }
588 };

```

## SegBeats.h

**Description:** In Segment Tree Beats, ‘lazy’ does NOT mean “updates still missing here”. The node already reflects all previous updates. Instead, ‘lazy’ stores what must be propagated to the children before recursing. Always call ‘apply(l,r,p)’ before descending. This node layout supports range add, range chmin and range chmax operations. Beats conditions:

break: MIN x: mx1 <= x ; MAX x: mi1 >= x

tag: MIN x: x > mx2 ; MAX x: x < mi2

Time: amortized  $\mathcal{O}(\log^2 N)$ , without range add  $\mathcal{O}(\log N)$

fa8527, 47 lines

```

3c9    struct node{
45e        ll mx1, mx2, sum, lazy;
9e5        ll mi1, mi2;
faa        int cMax, cMin, tam;
db3        node(int x=0) : mx1(x),mx2(-inf),mi1(x),mi2(inf),
744                cMax(1),cMin(1),tam(1),sum(x),lazy(0){}
b67        node(node a, node b){
4f5            sum = a.sum+b.sum, tam = a.tam+b.tam;
c60            lazy = 0;
15b            mx1 = max(a.mx1, b.mx1);
9ae            mx2 = max(a.mx2, b.mx2);
f62            if(a.mx1 != b.mx1) mx2 = max(mx2, min(a.mx1, b.mx1));
b60            cMax=(a.mx1==mx1 ? a.cMax:0)+(b.mx1==mx1 ? b.cMax:0);

09f            mi1 = min(a.mi1, b.mi1);
143            mi2 = min(a.mi2, b.mi2);
3bf            if(a.mi1 != b.mi1) mi2=min(mi2, max(a.mi1, b.mi1));
c18            cMin=(a.mi1==mi1 ? a.cMin:0)+(b.mi1==mi1 ? b.cMin:0);
23d        }
38d        void apply_sum(ll x){
2a1            mx1 += x, mx2 += x, mi1 += x, mi2 += x;
99b            sum += tam*x, lazy += x;
b5e        }
cf4        void apply_min(ll x){
```

```

e07        if(x >= mx1) return;
c44        sum -= (mx1 - x)*cMax;
be0        if(mi1 == mx1) mi1 = x;
8ef        if(mi2 == mx1) mi2 = x;
ea2        mx1 = x;
908    }
0c8        void apply_max(ll x){
e25        if(x <= mi1) return;
59e        sum -= (mi1 - x)*cMin;
4b1        if(mx1 == mi1) mx1 = x;
d69        if(mx2 == mi1) mx2 = x;
1ff        mi1 = x;
0e4    }
554    }
fdc    void apply(int l, int r, int p){
c8e        for(int i=2*p+1; i<=2*p+2; i++){
dbf            seg[i].apply_sum(st[p].lazy);
c90            seg[i].apply_min(st[p].mx1);
61a            seg[i].apply_max(st[p].mi1);
4b8        }
431        seg[p].lazy = 0;
dd0    }
```

## RMQ.h

Usage: RMQ rmq(values);  
rmq.query(inclusive, inclusive);  
Time:  $\mathcal{O}(|V|\log|V| + Q)$

bca062, 17 lines

```

76a    struct RMQ {
8ac        vector<vector<int>> dp;
dd1        RMQ(const vector<int>& a) : dp(1, a) {
71c            for (int i = 1, pw = 1; pw*2 <= sz(a); pw*=2, i++) {
394                dp.emplace_back(sz(a) - pw*2 + 1);
d17                for (int j = 0; j < sz(dp[i]); j++) {
dcc                    dp[i][j] = min(dp[i-1][j], dp[i-1][j+pw]);
75a                }
b68            }
3e9        }
9e3        int query(int l, int r) {
658            assert(l <= r);
884            int k = 31 - __builtin_clz(r - l + 1);
1f9            return min(dp[k][l], dp[k][r - (1 << k) + 1]);
e21        }
bca    }
```

## UnionFind.h

Description: Disjoint-set data structure with bipartite check

```

146    struct Uf{
b54        vector<int> tam, ds, bi, c;
d2c        Uf(int n) : tam(n, 1), ds(n), bi(n, 1), c(n){
244            iota(all(ds), 0);
233        }
001        int find(int i){ return (i==ds[i] ? i : find(ds[i]));}
e5a        int color(int i){
300            return (i==ds[i] ? 0 : (c[i]^color(ds[i])));
c3b        void merge(int a, int b){
8d0            int ca = color(a), cb = color(b);
605            a = find(a), b = find(b);
a89            if(a == b){
686                if(ca == cb) bi[a] = false;
505                return;
c08            }
226            if(tam[a] < tam[b]) swap(a, b);
1ac            ds[b] = a, tam[a] += tam[b];
27c            bi[a] = (bi[a] && bi[b]);
834            c[b] = (ca ^ cb ^ 1);
a70        }
6d2    };
```

## UnionFindRollback.h

Description: Disjoint-set data structure with undo. If undo is not needed, skip st, time() and rollback().

Usage: int t = uf.time(); ...; uf.rollback(t);

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

d4405e, 23 lines

```

47a    struct RollbackUF {
f80        vector<int> e;
919        vector<pii> st;
f6f        RollbackUF(int n) : e(n, -1) {}
84b        int size(int x) { return -e[find(x)]; }
626        int find(int x) { return e[x] < 0 ? x : find(e[x]); }
49f        int time() { return sz(st); }
4db        void rollback(int t) {
314            for (int i = time(); i --> t;) {
8d2                e[st[i].first] = st[i].second;
b04                st.resize(t);
30b            }
cf0            bool join(int a, int b) {
605                a = find(a), b = find(b);
5c2                if (a == b) return false;
745                if (e[a] > e[b]) swap(a, b);
bac                st.push_back({a, e[a]});
e6e                st.push_back({b, e[b]});
708                e[a] += e[b]; e[b] = a;
8a6                return true;
6c7            }
d44    };
```

## Numerical (4)

## 4.1 Polynomials and recurrences

## Polynomial.h

c9b7b0, 19 lines

```

213    struct Poly {
3a1        vector<double> a;
9a5        double operator()(double x) const {
e3c            double val = 0;
d5c            for (int i = sz(a); i--;) (val *= x) += a[i];
d94            return val;
ae7        }
0ac        void diff() {
7b6            rep(i,1,sz(a)) a[i-1] = i*a[i];
468            a.pop_back();
afc        }
087        void divroot(double x0) {
898            double b = a.back(), c; a.back() = 0;
9cf            for(int i=sz(a)-1; i--;) {
406                c = a[i], a[i] = a[i+1]*x0+b, b=c;
468                a.pop_back();
3f8            }
c9b    };
```

## PolyRoots.h

Description: Finds the real roots to a polynomial.

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

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

"Polynomial.h"

b00bfe, 24 lines

```

64a    vector<double> polyRoots(Poly p, double xmin, double xmax)
{
853        if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
539        vector<double> ret;
f55        Poly der = p;
c06        der.diff();
617        auto dr = polyRoots(der, xmin, xmax);
d85        dr.push_back(xmin-1);
12c        dr.push_back(xmax+1);
```

```

423 sort(all(dr));
b98 rep(i,0,sz(dr)-1) {
d85     double l = dr[i], h = dr[i+1];
ad1     bool sign = p(l) > 0;
b41     if (sign ^ (p(h) > 0)) {
03d         rep(it,0,60) { // while (h - l > 1e-8)
761             double m = (l + h) / 2, f = p(m);
04c             if ((f <= 0) ^ sign) l = m;
193             else h = m;
b69         }
ff5         ret.push_back((l + h) / 2);
fc2     }
d15 }
edf     return ret;
b00 }

```

PolyInverse.h

2745a7, 18 lines

```

747 vector<ll> get_inverse(vector<ll> a) {
e4d     if (a.empty()) return {};
044     int Y = sz(a) - 1, n = 32 - __builtin_clz(Y);
ba5     n = (1 << n);
711     a.resize(n);
e3e     vector<ll> inv = { modpow(a[0], mod - 2), f, c;
a2b     inv.reserve(n);
599     for (int tam = 2; tam <= n; tam *= 2) {
d29         while (sz(f) < tam) f.push_back(a[sz(f)]);
fec         c = conv(f, inv);
757         rep(i, 0, tam) c[i] = (c[i] == 0 ? 0 : mod - c[i]);
df6         c[0] += (c[0] + 2 >= mod ? 2 - mod : 2);
f8b         inv = conv(inv, c);
118         inv.resize(tam);
9f4     }
531     return inv;
274 }

```

BerlekampMassey.h

**Description:** Recovers any  $n$ -order linear recurrence relation from the first  $n$  terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for floats is not guaranteed. Output will have size  $\leq n$ .

**Usage:** berlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}

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

96548b, 21 lines

```

c10    vector<ll> berlekampMassey(vector<ll> s) {
ea1        int n = sz(s), L = 0, m = 0;
2a2        vector<ll> C(n), B(n), T;
2b3        C[0] = B[0] = 1;

d6f        ll b = 1;
3d6        rep(i,0,n) { ++m;
b7f            ll d = s[i] % mod;
45a            rep(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
53a            if (!d) continue;
169            T = C; ll coef = d * modpow(b, mod-2) % mod;
2d1            rep(j,m,n) C[j] = (C[j] - coef * B[j - m]) % mod;
b6c            if (2 * L > i) continue;
dc3            L = i + 1 - L; B = T; b = d; m = 0;
8c2        }

51b        C.resize(L + 1); C.erase(C.begin());
e98        for (ll& x : C) x = (mod - x) % mod;
a91        return C;
965    }

```

LinearRecurrence.h

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

**Usage:** linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number  
**Time:**  $\mathcal{O}(n^2 \log k)$

547b93, 27 lines

```

437     using Poly = vector<ll>;
2ef     ll linearRec(Poly S, Poly tr, ll k) {
327         int n = sz(tr);

0e9         auto combine = [&](Poly a, Poly b) {
b1c             Poly res(n * 2 + 1);
5f7             rep(i,0,n+1) rep(j,0,n+1)
389                 res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
bdc                 for (int i = 2 * n; i > n; --i) rep(j,0,n)
fc3                     res[i-1-j] = (res[i-1-j] + res[i] * tr[j]) % mod;
b76                     res.resize(n + 1);
b50                     return res;
55c                 };

bf8         Poly pol(n + 1), e(pol);
997         pol[0] = e[1] = 1;

e96         for (++k; k; k /= 2) {
491             if (k % 2) pol = combine(pol, e);
0d9                 e = combine(e, e);
813             }

cd2         ll res = 0;
e8d         rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
b50         return res;
594     }

```

LagrangeInterpolation.h

**Description:** Returns  $P(x)$ , where  $P$  is the unique polynomial of degree  $\leq |pts| - 1$  such that  $\forall i, P(i) = pts[i]$ .

5cbc52, 16 lines

```

1d7     ll lagrangeiro(const vector<ll> &pts, ll x)
f95     {
c10         int n = sz(pts);
e52         vector<ll> pref(n, 1), suf(n, 1);
a7c         for (int i=1; i<n; i++) pref[i] = mul(pref[i-1], x-(i-1))
;
71b         for (int i=n-2; i>=0; i--) suf[i] = mul(suf[i+1], x-(i+1))
);
a3e         ll y = 0;
603         for (int i=0; i<n; i++) {
591             ll oi = pts[i];
08c             oi = mul(oi, mul(pref[i], mul(suf[i], mul(ifat[i], ifat
[n-i-1]))));
b67             if ((n-i-1) % 2 == 1) oi *= -1;
ded             y = (y + oi) % mod;
b2d         }
6d1         return (y + mod) % mod;
f9e     }

```

## 4.2 Optimization

GoldenSectionSearch.h

**Description:** Finds the argument minimizing the function  $f$  in the interval  $[a, b]$  assuming  $f$  is unimodal on the interval, i.e. has only one local minimum and no local maximum. The maximum error in the result is  $\text{eps}$ . Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.

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

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

```

e4d     if (f1 < f2) { //change to > to find maximum
da5         b = x2; x2 = x1; f2 = f1;
dfb         x1 = b - r*(b-a); f1 = f(x1);
451     } else {
d6e         a = x1; x1 = x2; f1 = f2;
815         x2 = a + r*(b-a); f2 = f(x2);
2fe     }
3f5     return a;
31d }

```

Simplex.h

**Description:** Solves a general linear maximization problem: maximize  $c^T x$  subject to  $Ax \leq b$ ,  $x \geq 0$ . Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of  $c^T x$  otherwise. The input vector is set to an optimal  $x$  (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that  $x = 0$  is viable.

**Usage:** vvd A = {{1,-1}, {-1,1}, {-1,-2}};

vd b = {1,1,-4}, c = {-1,-1}, x;

T val = LPSolver(A, b, c).solve(x);

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

aa8530, 69 lines

```

943     typedef double T; // long double, Rational, double + modP
>...
487     typedef vector<T> vd;
840     typedef vector<vd> vvd;

8cb     const T eps = 1e-8, inf = 1./0;
85f     #define MP make_pair
90c     #define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])
) s=j

34b     struct LPSolver {
b5c         int m, n;
14e         vi N, B;
d5f         vvd D;
9b8         LPSolver(const vvd& A, const vd& b, const vd& c) :
f40             m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
27d                 rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
f03                 rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
59d                 rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
24a                 N[n] = -1; D[m+1][n] = 1;
6ff             }

333     void pivot(int r, int s) {
3cd         T *a = D[r].data(), inv = 1 / a[s];
12b         rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
449             T *b = D[i].data(), inv2 = b[s] * inv;
e07                 rep(j,0,n+2) b[j] -= a[j] * inv2;
e78                 b[s] = a[s] * inv2;
ca4             }
485             rep(j,0,n+2) if (j != s) D[r][j] *= inv;
3b7             rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
dbd             D[r][s] = inv;
c97             swap(B[r], N[s]);
9cd             }

24e     bool simplex(int phase) {
8b8         int x = m + phase - 1;
1de         for (;;) {
7a6             int s = -1;
aea             rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
4dc             if (D[x][s] >= -eps) return true;
56d             int r = -1;
670             rep(i,0,m) {
776                 if (D[i][s] <= eps) continue;

```

UFPE - las4s e pelados

Determinant IntDeterminant SolveLinear SolveLinearBinary XorGauss FastFourierTransform

```

c95     if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
133         < MP(D[r][n+1] / D[r][s], B[r])) r = i
;
468     }
fbf     if (r == -1) return false;
dba     pivot(r, s);
7d8     }
f15     }

859 T solve(vd &x) {
898     int r = 0;
435     rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
32d     if (D[r][n+1] < -eps) {
f65         pivot(r, n);
fda         if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
939         rep(i,0,m) if (B[i] == -1) {
37f             int s = 0;
c66             rep(j,1,n+1) ltj(D[i]);
b98             pivot(i, s);
683         }
b65     }
203     bool ok = simplex(1); x = vd(n);
972     rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
d3a     return ok ? D[m][n+1] : inf;
396     }
c57 };

```

## 4.3 Matrices

Determinant.h

**Description:** Calculates determinant of a matrix. Destroys the matrix.

**Time:**  $\mathcal{O}(N^3)$       bd5cec, 16 lines

```
e36 double det(vector<vector<double>>
70e int n = sz(a); double res = 1;
fea rep(i,0,n) {
281     int b = i;
b0b     rep(j,i+1,n) if (fabs(a[j][i]
311         if (i != b) swap(a[i], a[b]),
9b1         res *= a[i][i];
d5c         if (res == 0) return 0;
3e3         rep(j,i+1,n) {
f15             double v = a[j][i] / a[i][i];
353             if (v != 0) rep(k,i+1,n) a[k]
4ec             }
ee1     }
b50     return res;
bd5 }
```

IntDeterminant.h

**Description:** Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.

**Time:**  $\mathcal{O}(N^3)$  3313dc, 19 lines

```
031 const ll mod = 12345;
38e ll det(vector<vector<ll>>& a) {
da9     int n = sz(a); ll ans = 1;
fea     rep(i,0,n) {
3e3         rep(j,i+1,n) {
f36             while (a[j][i] != 0) { // gcd step
479                 ll t = a[i][i] / a[j][i];
b87                 if (t) rep(k,i,n)
e5b                     a[i][k] = (a[i][k] - a[j][k] * t) % mod;
332                     swap(a[i], a[j]);
17c                     ans *= -1;
e81                 }
30d             }
a97             ans = ans * a[i][i] % mod;
f4e             if (!ans) return 0;
f39 }
```

```
38     return (ans + mod) % mod;  
e8 }
```

## SolveLinear.h

**Description:** If  $\text{inv} = 1$ , finds the inverse of the matrix  $\text{eq}$  and returns it as a flat vector  
**Time:**  $\mathcal{O}(\min(n, m) nm)$       2.134 – 52.4

c134e, 52 lines

```

20 struct Gauss {
6d   const double eps = 1e-9;
3d   vector<vector<double>> eq;
54   void addEquation(const vector<double>& e) {
03     eq.push_back(e);
4f   pair<int, vector<double>> solve(int inv=0) {
14     int n = sz(eq), m = sz(eq[0]) - 1 + inv;
9c     if(inv) {
33       rep(i, 0, n) eq[i].resize(2*n), eq[i][n+i] = 1;
e2     }
cb   vector<int> where(m, -1);
73   for (int col = 0, row = 0; col < m && row < n; col++)
{
05     int sel = row;
3c     rep(i, row, n) {
64       if (abs(eq[i][col]) > abs(eq[sel][col])) sel = i;
04     }
8b     if (abs(eq[sel][col]) < eps) continue;
ad     rep(i, col, sz(eq[0])) swap(eq[sel][i], eq[row][i]);
c3     where[col] = row;
ff     rep(i, 0, n) if (i != row) {
84       double c = eq[i][col] / eq[row][col];
f1       rep(j, col, sz(eq[0])) eq[i][j] -= eq[row][j] * c;
7d     }
ef     ++row;
b8   }
9c   if(inv) {
ea     vector<double> res;
rep(i, 0, n) {
20       if (where[i] == -1) return {0, {}}; // Singular
af       rep(j, n, 2*n)
89       res.push_back(eq[where[i]][j] / eq[where[i]][i]);
;
81     }
b1     return {1, res};
00   }

```

```
33     vector<double> ans(m, 0)
```

```

70 rep(i, 0, m) {
71     if (where[i] != -1)
72         ans[i] = eq[where[i]][m] / eq[where[i]][i];
73 }
74 ea
75 rep(i, 0, n) {
76     double sum = 0;
77     rep(j, 0, m) {
78         sum = sum + ans[j] * eq[i][j];
79     }
80     if (abs(sum - eq[i][m]) > eps) return {0, {}};
81 }
82 f2
83 rep(i, 0, m) if (where[i] == -1) return {2, ans};
84 a
85 return {1, ans};
86
87 }

```

## SolveLinearBinary.h

**Time:**  $\mathcal{O}\left(\frac{\min(n,m) nm}{64}\right)$

8c946. 32 lines

```
81 pair<int, bitset<M>> gauss(vector<bitset<M>> eq) {
82     int n = eq.size(), m = M - 1;
83     vector<int> where(m, -1);
84     for(int col = 0, row = 0; col < m && row < n; col++) {
```

```
    rep(i, row, n)
        if (eq[i][col]) {
            swap(eq[i], eq[row]);
            break;
        }
        if (!eq[row][col]) continue;
    where[col] = row;
```

```

fea      rep(i, 0, n) {
b60          if (i != row && eq[i][col]) eq[i] ^= eq[row];
981      }
4ef          ++row;
c74      }
7eb      bitset<M> ans;
670      rep(i, 0, m) {
713          if (where[i] != -1) ans[i] = eq[where[i]][m];
691      }
fea      rep(i, 0, n) {
e5c          int sum = (ans & eq[i]).count();
53f          sum %= 2;
36a          if (sum != eq[i][m]) return pair(0, bitset<M>());
29e      }
670      rep(i, 0, m) {
be2          if (where[i] == -1) return pair(INF, ans);
958      }
280      return pair(1, ans);
28c  }

```

XorGauss.h

5a1937, 30 fine

```
b94 struct XorGauss {
060     int N;
471     vector<ll> basis, who, mask;
47b     XorGauss(int N) : N(N), basis(N), who(N), mask(N) {}
221     // if(ans & (1ll << j)) who[j] was used to form x
04b     bool belong(ll x) {
042         ll ans = 0;
e13         for(int i=N-1; i>=0; i--) {
4ec             if((x ^ basis[i]) < x) {
6b0                 ans ^= mask[i];
x25                 x ^= basis[i];
254             }
2ad         }
069         return (x == 0);
c26     }
397     void add(ll v, int idx) {
a4d         ll msk = 0;
042         for (int i = N - 1; i >= 0; i--) {
80f             if (!(v & (1ll << i))) continue;
bf3             if (basis[i] == 0) {
1c7                 basis[i] = v, who[i] = idx;
940                 mask[i] = (msk | (1ll << i));
505                 return;
bc8             }
00e             msk ^= mask[i];
647             v ^= basis[i];
25b         }
fcc     }
5a1 }
```

#### 4.4 Fourier transforms

## EastFourierTransform.h

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

**Time:**  $\mathcal{O}(N \log N)$  with  $N = |A| + |B|$  ( $\sim 1s$  for  $N = 2^{22}$ )      773fed, 44 lines

```
bcc typedef complex<double> C;

7c0 void fft(vector<C>& a) {
a5b     int n = a.size(), L = 31 - __builtin_clz(n);
f82     static vector<complex<long double>> R(2, 1); // 10%
faster if double
991     static vector<C> rt(2, 1);
ad8     for (static int k = 2; k < n; k *= 2) {
9d9         R.resize(n);
335         rt.resize(n);
411         auto x = polar(1.0L, acos(-1.0L) / k);
cdb         rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
a8a     }
e66     vector<ll> rev(n);
dcb     rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
47b     rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);

d3f     for (int k = 1; k < n; k *= 2) {
cda         for (int i = 0; i < n; i += 2 * k) {
0c2             for (int j = 0; j < k; j++) {
30c                 auto x = (double)&rt[j + k];
ebe                 auto y = (double)&a[i + j + k];
15c                 C z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);
20a                 a[i + j + k] = a[i + j] - z;
1b0                 a[i + j] += z;
b5b             }
1fe         }
fa0     }
b33 }
```

ccc **vector<ll>** conv(**const** **vector<ll>**& a, **const** **vector<ll>**& b) {
f88 **if** (a.empty() || b.empty()) **return** {};
920 **vector<ll>** res(sz(a) + sz(b) - 1);
441 **int** L = 32 - \_\_builtin\_clz(sz(res)), n = 1 << L;
060 **vector<C>** in(n), out(n);
bla copy(all(a), in.begin());
fef **rep**(i,0,sz(b)) in[i].imag(b[i]);
21a fft(in);
6fb **for** (C& x : in) x \*= x;
4d7 **rep**(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
3d7 fft(out);
aa3 **rep**(i,0,sz(res)) res[i] = round(imag(out[i]) / (4 \* n));
b50 **return** res;
7f4 }

**FastFourierTransformMod.h**

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

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

"FastFourierTransform.h"      b82773, 23 lines

```
192 typedef vector<ll> vl;
3fe template<int M> vl convMod(const vl &a, const vl &b) {
f88     if (a.empty() || b.empty()) return {};
19d     vl res(sz(a) + sz(b) - 1);
a6f     int B=32-__builtin_clz(sz(res)), n=1<<B,cut=int(sqrt(M));
3dd     vector<C> L(n), R(n), outs(n), outl(n);
ald     rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
97d     rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
5d5     fft(L), fft(R);
fea     rep(i,0,n) {
39d         int j = -i & (n - 1);
65e         outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
91a         outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / li;
cb3     }
d08     fft(outl), fft(outs);
35e     rep(i,0,sz(res)) {
```

```
351     ll av = ll(real(outl[i]) + .5), cv = ll(imag(outs[i]) + .5);
988     ll bv = ll(imag(outl[i]) + .5) + ll(real(outs[i]) + .5);
6a3     res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
58f     }
b50     return res;
c1f }
```

**NumberTheoreticTransform.h**

**Description:** ntt(a) computes  $\hat{f}(k) = \sum_x a[x]g^{xk}$  for all  $k$ , where  $g = \text{root}^{(mod-1)/N}$ .  $N$  must be a power of 2. Useful for convolution modulo specific nice primes of the form  $2^a b + 1$ , where the convolution result has size at most  $2^a$ . For arbitrary modulo, see FFTMod. conv(a, b) = c, where  $c[x] = \sum a[i]b[x - i]$ . For manual convolution: NTT the inputs, multiply pointwise, divide by n, reverse(start+1, end), NTT back. Inputs must be in  $[0, \text{mod}]$ .

**Time:**  $\mathcal{O}(N \log N)$       84c11e, 34 lines

```
376     const int mod = 998244353, root = 62;
192     typedef vector<ll> vl;
8ec     void ntt(vl &a) {
6ae         int n = sz(a), L = 31 - __builtin_clz(n);
7c9         static vl rt(2, 1);
8ee         for (static int k = 2, s = 2; k < n; k *= 2, s++) {
335             rt.resize(n);
d43             ll z[] = {1, modpow(root, mod >> s)};
8e7             rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
f39         }
808         vector<int> rev(n);
dcb         rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
47b         rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
657         for (int k = 1; k < n; k *= 2)
2cb             for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
86e                 ll z = rt[j+k] * a[i+j+k] % mod, &ai = a[i+j];
598                 a[i + j + k] = ai - z + (z > ai ? mod : 0);
589                 ai += (ai + z >= mod ? z - mod : z);
9a8             }
de9         }
v1 conv(const vl &a, const vl &b) {
f88         if (a.empty() || b.empty()) return {};
f51         int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),
570             n = 1 << B;
9ef         int inv = modpow(n, mod - 2);
e4c         vl L(a), R(b), out(n);
6b4         L.resize(n), R.resize(n);
d9e         ntt(L), ntt(R);
dfc         rep(i,0,n)
0db             out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
ec9             ntt(out);
c20             return out.begin(), out.begin() + s;
387 }
```

**FWHT.h**

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

**Time:**  $\mathcal{O}(N \log N)$       124c14, 20 lines

```
5ad     void FST(vector<ll>& a, bool inv) {
a9d         for (int n = sz(a), step = 1; step < n; step *= 2) {
5bd             for (int i = 0; i < n; i += 2 * step) {
4ee                 for (int j = i; j < i + step; j++) {
2fe                     ll& u = a[j], &v = a[j + step];
c6f                     tie(u, v) =
2d3                         inv ? pair(v - u, u) : pair(v, u + v); // AND
aba                         inv ? pair(v, u - v) : pair(u + v, u); // OR
a5a                         pair(u + v, u - v); // XOR
0b4                     }
fb4     }
```

```
cd3     }
c9b     if(inv) for(ll& x : a) x /= sz(a); // XOR only
075     }
eb2     vector<ll> conv(vector<ll> a, vector<ll> b) {
595         FST(a, 0); FST(b, 0);
2dd         for (int i = 0; i < sz(a); i++) a[i] *= b[i];
062         FST(a, 1); return a;
7bf     }
```

**Number theory (5)****5.1 Modular arithmetic****ModInverse.h**

**Description:** Pre-computation of modular inverses. Assumes  $\text{LIM} \leq \text{mod}$  and that  $\text{mod}$  is a prime.      c375f5, 5 lines

```
88a     const ll mod = 1000000007, LIM = 200000;
0f2     inv[1] = 1;
379     for(int i=2; i<LIM; i++)
86c         inv[i] = mod - (mod / i) * inv[mod % i] % mod;
```

**ModMulLL.h**

**Description:** Calculate  $a \cdot b \bmod c$  (or  $a^b \bmod c$ ) for  $0 \leq a, b \leq c \leq 7.2 \cdot 10^{18}$ .      bbbdb8f, 12 lines

```
f4c     typedef unsigned long long ull;
f85     ull modmul(ull a, ull b, ull M) {
2dd     ll ret = a * b - M * ull(1.L / M * a * b);
964     return ret + M * (ret < 0) - M * (ret >= (ll)M);
e93     }
4f6     ull modpow(ull b, ull e, ull mod) {
c1a     ull ans = 1;
a18     for (; e; b = modmul(b, b, mod), e /= 2)
9e8         if (e & 1) ans = modmul(ans, b, mod);
ba7     return ans;
100 }
```

**ModPow.h**

     b83e45, 9 lines

```
e2e     const ll mod = 1000000007; // faster if const
9d8     ll modpow(ll b, ll e) {
d54     ll ans = 1;
36e     for (; e; b = b * b % mod, e /= 2)
b46         if (e & 1) ans = ans * b % mod;
ba7     return ans;
d1e }
```

**ModSqrt.h**

**Description:** Tonelli-Shanks algorithm for modular square roots. Finds  $x$  s.t.  $x^2 \equiv a \pmod p$  ( $-x$  gives the other solution).

**Time:**  $\mathcal{O}(\log^2 p)$  worst case,  $\mathcal{O}(\log p)$  for most  $p$

```
"ModSqrt.h"      19a793, 25 lines
a77     ll sqrt(ll a, ll p) {
5de     a % p; if (a < 0) a += p;
b47     if (a == 0) return 0;
5c6     assert(modpow(a, (p-1)/2, p) == 1); // else no solution
a75     if (p % 4 == 3) return modpow(a, (p+1)/4, p);
// a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
b94     ll s = p - 1, n = 2;
ee5     int r = 0, m;
084     while (s % 2 == 0)
082         +r, s /= 2;
eaa     while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
0c3     ll x = modpow(a, (s + 1) / 2, p);
b74     ll b = modpow(a, s, p), g = modpow(n, s, p);
1af     for (;;) r = m {
```

```
4fd    11 t = b;
713    for (m = 0; m < r && t != 1; ++m)
c58      t = t * t % p;
ae0    if (m == 0) return x;
20e    11 gs = modpow(g, 1LL << (r - m - 1), p);
fba    g = gs * gs % p;
4fb    x = x * gs % p;
c5c    b = b * g % p;
e3a  }
19a }
```

## DiscreteLog.h

**Description:** Returns the smallest  $x$  such that  $a^x \bmod m = b \bmod m$ . If no such  $x$  exists, returns  $-1$ .

**Time:**  $O(\sqrt{m}) * \log(\sqrt{m})$

2f126b, 32 lines

```
758 int solve(int a, int b, int m) {
a6e  a %= m, b %= m;
ec4  if (a == 0) return (b ? -1 : 1);
// caso gcd(a, m) > 1
6af  int k = 1, add = 0, g;
553  while ((g = gcd(a, m)) > 1) {
d90    if (b == k) return add;
642    if (b % g) return -1;
92a    b /= g, m /= g, ++add;
803    k = (k * 111 * a / g) % m;
8a0  }

16c  int sq = sqrt(m) + 1;
b51  int big = 1;
4e1  for (int i = 0; i < sq; i++) big = (111 * big * a) % m
;

053  vector<pii> vals;
3c2  for (int q = 0, cur = b; q <= sq; q++) {
b53    vals.push_back({cur, q});
b50    cur = (111 * cur * a) % m;
837  }
62b  sort(all(vals));
90c  for (int p = 1, cur = k; p <= sq; p++) {
5d3    cur = (111 * cur * big) % m;
958    auto it = lower_bound(all(vals), pair(cur, INF));
721    if (it != vals.begin() && (--it)->first == cur) {
a30      return sq * p - it->second + add;
6fe  }
f22  }
daa  return -1;
2f1 }
```

## DiscreteRoot.h

**Description:** Returns  $x$  such that  $x^k \bmod m = a \bmod m$ . If no such  $x$  exists, returns  $-1$ .

**Time:**  $O(\sqrt{m}) * \log(\sqrt{m})$

```
"PrimitiveRoot.h", "DiscreteLog.h"
1d582e, 11 lines
// Discrete Root

27c  11 discreteRoot(11 k, 11 a, 11 m) {
738    11 g = primitiveRoot(m);
58b    11 y = discreteLog(fexp(g, k, m), a, m);
f31    if (y == -1) return y;
a58    return fexp(g, y, m);
1d5 }
```

## 5.2 Primality

## MillerRabin.h

**Description:** Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to  $7 \cdot 10^{18}$ ; for larger numbers, use Python and extend A randomly.

**Time:** 7 times the complexity of  $a^b \bmod c$ .

66fe73, 13 lines

```
"ModMullL.h"
da4  bool isPrime(ull n) {
c16  if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
062  ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 17952650
22};
ae0  ull s = __builtin_ctzll(n-1), d = n >> s;
e80  for (ull a : A) { // count trailing zeroes
6b4    ull p = modpow(a%n, d, n), i = s;
274    while (p != 1 && p != n - 1 && a % n && i--)
c77    p = modmul(p, p, n);
e28    if (p != n-1 && i != s) return 0;
edf  }
6a5  return 1;
66f  }
```

## Factor.h

**Description:** Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).

**Time:**  $\mathcal{O}(n^{1/4})$ , less for numbers with small factors.

da0c7c, 19 lines

```
"ModMullL.h", "MillerRabin.h"
7eb  ull pollard(ull n) {
222  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
5f5  auto f = [&](ull x) { return modmul(x, x, n) + i; };
f51  while (t++ % 40 || gcd(prd, n) == 1) {
be9  if (x == y) x = ++i, y = f(x);
70f  if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
b78  x = f(x), y = f(f(y));
bF8  }
002  return gcd(prd, n);
d1b  }
591  vector<ull> factor(ull n) {
1b9  if (n == 1) return {};
6b5  if (isPrime(n)) return {n};
bc6  ull x = pollard(n);
52a  auto l = factor(x), r = factor(n / x);
7af  l.insert(l.end(), all(r));
792  return l;
d54  }
```

## PrimitiveRoot.h

18a01e, 15 lines

```
// is n primitive root of p ?
ad0  bool test(11 x, 11 p) {
a56    11 m = p - 1;
845    for (11 i = 2; i * i <= m; ++i) if (!(m % i)) {
e64      if (modpow(x, i, p) == 1) return false;
599      if (modpow(x, m / i, p) == 1) return false;
53a    }
8a6    return true;
c4e  }
// find the smallest primitive root for p
220  11 search(11 p) {
1bf    for (11 i = 2; i < p; i++) if (test(i, p)) return i;
daa  return -1;
a3c  }
```

## 5.3 Divisibility

## Euclid.h

**Description:** Find  $x, y$  such that  $Ax + By = \gcd(A, B)$ . If  $\gcd(A, B) = 1$ , then  $x = A^{-1} \pmod{B}$  and  $y = B^{-1} \pmod{A}$ .

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

33ba8f, 6 lines

```
33b  }
CRT.h
ba1a4a, 25 lines
bc9  11 modinverse(11 a, 11 b, 11 s0 = 1, 11 s1 = 0) {
a76  return !b ? s0 : modinverse(b, a % b, s1, s0 - s1 * (a / b));
d8b  11 mul(11 a, 11 b, 11 m) {
a6f  return (((__int128_t)a*b)%m + m)%m;
0bc  }

28d  struct Equation {
4c5  11 mod, ans;
08f  bool valid;
145  Equation(11 a, 11 m) { mod = m, ans = a, valid = true; }
0fc  Equation() { valid = false; }
4d3  Equation(Equation a, Equation b) {
515  valid = false;
1a0  if (!a.valid || !b.valid) return;
85c  11 g = gcd(a.mod, b.mod);
44d  if ((a.ans - b.ans) % g != 0) return;
af0  valid = true;
b98  mod = a.mod * (b.mod / g);
81a  11 x = mul(a.mod, modinverse(a.mod, b.mod), mod);
38a  ans = a.ans + mul(x, (b.ans - a.ans) / g, mod);
c4c  ans = (ans % mod + mod) % mod;
6f5  }
f48  };

DivisionTrick.h
02aebb, 15 lines
7f1  void floor_ranges(int n) {
79c  for (int l = 1, r; l <= n; l = r + 1) {
746    r = n / (n / l);
// floor(n/y) has the same value for y in [l..r]
5bf  }
eee  }
678  void ceil_ranges(int n) {
79c  for (int l = 1, r; l <= n; l = r + 1) {
d47    int x = (n + l - 1) / l;
374    if (x == 1) r = n;
21b    else r = (n - 1) / (x - 1);
// ceil(n/y) has the same value for y in [l..r]
06c  }
57c  }
```

## Phi.h

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

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

**Euler's thm (generalized):**  $a, m$  arbitrary,  $n \geq \log_2 m \Rightarrow a^n \equiv a^{\phi(m)+(n \bmod \phi(m))} \pmod{m}$ .

e58bf0, 6 lines

```
d08  void calculatePhi() {
265  for(int i=0; i<LIM; i++) phi[i] = i&1 ? i : i/2;
c83  for (int i = 3; i < LIM; i += 2) if(phi[i] == i)
dc2    for (int j = i; j < LIM; j += i) phi[j] -= phi[j] / i;
e58  }
```

## Combinatorial (6)

## PartitionSolver.h

e50fb7, 61 lines

```

d38 template<const int N>
182 struct PartitionSolver {
4ce   vector<vector<int>> part, to, from;
621   PartitionSolver() {
a9d     vector<int> a;
1ed     part.push_back(a);
77f     gen(1, N, a);
796     sort(all(part));
ed4     to.assign(sz(part), vector<int>(N + 1, -1));
9a5     from = to;
ddd     for (int i = 0; i < sz(part); i++) {
a93       int sum = 0;
87f       auto arr = part[i];
bca       for (auto x : arr) sum += x;
4fa       to[i][0] = i;
615       from[i][0] = i;
afc       for (int j = 1; j + sum <= N; j++) {
123         arr = part[i];
9d6         arr.push_back(j);
ceb         sort(all(arr));
d02         to[i][j] = getIndex(arr);
942         from[to[i][j]][j] = i;
20d       }
bef     }
283   }

810   int size() const { return sz(part); }
9ee   int getIndex(const vector<int>& arr) const {
168     return lower_bound(all(part), arr) - part.begin(); }
b49   int add(int id, int num) const { return to[id][num]; }
944   int rem(int id, int num) const { return from[id][num]; }
168   vector<int> getPartition(int id) const {
37b     return part[id]; }

1ba   void gen(int i, int sum, vector<int>& a) {
a05     if (i > sum) { return; }
726     a.push_back(i);
1ed     part.push_back(a);
278     gen(i, sum - i, a);
468     a.pop_back();
48f     gen(i + 1, sum, a);
537   }
f4f };

// Number of partitions for all integers <= n
75c   vector<ll> partitionNumber(int n) {
d9c     vector<ll> ans(n + 1, 0);
82f     ans[0] = 1;
78a     for (int i = 1; i <= n; i++) {
87f       for (int j = 1; j * (3 * j + 1) / 2 <= i; j++) {
b6b         ll here = ans[i - j * (3 * j + 1) / 2];
c91         ans[i] = (ans[i] + (j & 1 ? here : -here));
365       }
7c6       for (int j = 1; j * (3 * j - 1) / 2 <= i; j++) {
ala         ll here = ans[i - j * (3 * j - 1) / 2];
c91         ans[i] = (ans[i] + (j & 1 ? here : -here));
162     }
4a3   }
ba7   return ans;
08b }

```

## Graph (7)

## 7.1 Fundamentals

## BellmanFord.h

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

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

529834, 24 lines

```

f5e   const ll inf = LLONG_MAX;
83a   struct Ed { int a, b, w, s() { return a < b ? a : -a; } };
9ac   struct Node { ll dist = inf; int prev = -1; };

6fc   void bell(vector<Node>& nodes, vector<Ed>& eds, int s) {
97b     nodes[s].dist = 0;
eb9     sort(all(eds), [] (Ed a, Ed b) { return a.s() < b.s(); });

74e     int lim = sz(nodes) / 2 + 2; // 3+100 with shuffled
vertices
c5a     rep(i, 0, lim) for (Ed ed : eds) {
905       Node cur = nodes[ed.a], &dest = nodes[ed.b];
d7d       if (abs(cur.dist) == inf) continue;
6ab       ll d = cur.dist + ed.w;
6ec       if (d < dest.dist) {
956         dest.prev = ed.a;
4c2         dest.dist = (i < lim-1 ? d : -inf);
452       }
75a     }
ced     rep(i, 0, lim) for (Ed e : eds) {
3ab       if (nodes[e.a].dist == -inf)
5ff         nodes[e.b].dist = -inf;
1d7     }
166   }

```

## FloydWarshall.h

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

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

531245, 13 lines

```

964   const ll inf = 1LL << 62;
914   void floydWarshall(vector<vector<ll>>& m) {
e9d     int n = sz(m);
831     rep(i, 0, n) m[i][i] = min(m[i][i], 0LL);
99d     rep(k, 0, n) rep(i, 0, n) rep(j, 0, n)
19b       if (m[i][k] != inf && m[k][j] != inf) {
668         auto newDist = max(m[i][k] + m[k][j], -inf);
e89         m[i][j] = min(m[i][j], newDist);
f38       }
a69     rep(k, 0, n) if (m[k][k] < 0) rep(i, 0, n) rep(j, 0, n)
ffd       if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;
f12   }

```

## 7.2 Network flow and Matching

## Dinic.h

**Time:**  $- \mathcal{O}(\min(m \cdot \text{max\_flow}, n^2 m))$ .

- For graphs with unit capacities:  $\mathcal{O}(\min(m\sqrt{m}, mn^{2/3}))$ .

- If every vertex has in-degree 1 or out-degree 1:  $\mathcal{O}(m\sqrt{n})$ .

- With capacity scaling:  $\mathcal{O}(nm \log(\text{MAXCAP}))$  with high constant factor.

```

14d   struct Dinic {
61f     const bool scaling = false;
206     int lim;
670     struct edge {
c63       int to, rev;

```

```

a14     ll cap, flow;
7f9     bool res;
6dd     edge(int to_, ll cap_, int rev_, bool res_)
a94       : to(to_), cap(cap_), rev(rev_), flow(0), res(res_) {}
477     };

002     vector<vector<edge>> g;
216     vector<int> lev, beg;
a71     ll F;
63f     Dinic(int n) : g(n), lev(n), beg(n), F(0) {}

0c5     void add(int a, int b, ll c, ll other = 0) {
de2       g[a].emplace_back(b, c, sz(g[b]), false);
fa5       g[b].emplace_back(a, other, sz(g[a])-1, true);
14f     }
123     bool bfs(int s, int t) {
e59       fill(all(lev), -1);
4e7       fill(all(beg), 0);
0a4       lev[s] = 0;
8b2       queue<int> q; q.push(s);
647       while (sz(q)) {
be1         int u = q.front(); q.pop();
bd9         for (auto& i : g[u]) {
dbc           if (lev[i.to] != -1 || (i.flow == i.cap)) continue;
b4f           if (scaling and i.cap - i.flow < lim) continue;
185           lev[i.to] = lev[u] + 1;
8ca           q.push(i.to);
f97         }
b1b       }
0de     return lev[t] != -1;
310   }
1dc   ll dfs(int v, int s, ll f = INF) {
50b     if (!f or v == s) return f;
84d     for (int& i = beg[v]; i < sz(g[v]); i++) {
027       auto& e = g[v][i];
206       if (lev[e.to] != lev[v] + 1) continue;
a30       ll foi = dfs(e.to, s, min(f, e.cap - e.flow));
749       if (!foi) continue;
3c5       e.flow += foi, g[e.to][e.rev].flow -= foi;
45c       return foi;
e08     }
bb3     return 0;
b98   }
2b4   ll maxFlow(int s, int t) {
a86     for (lim = scaling ? (1<<30) : 1; lim; lim /= 2)
69c       while (bfs(s, t)) while (ll ff = dfs(s, t)) F += ff;
4ff     return F;
6c8   }
0fe   bool incut(int u) { return lev[u] != -1; }
892 }

LowerBoundFlow.h
Description: Calculates maximum flow with lower/upper bounds on edges. Returns -1 if no feasible flow exists. add(a, b, l, r) adds edge a → b where flow f must satisfy  $l \leq f \leq r$ . add(a, b, c) adds edge a → b with capacity c (implies  $0 \leq f \leq c$ ). Same complexity as Dinic.
```

756539, 36 lines

```

" Dinic.h"
0ca   struct lb_max_flow : Dinic {
96f     vector<ll> d;
be9     lb_max_flow(int n) : Dinic(n + 2), d(n, 0) {}
b12     void add(int a, int b, int l, int r) {
c97       d[a] -= 1;
f1b       d[b] += 1;
cb6       Dinic::add(a, b, r - 1);
989     }
087     void add(int a, int b, int c) {
610       Dinic::add(a, b, c);
330     }
7a1     bool has_circulation() {

```

## UFPE - las4s e pelados

```

ac0    int n = sz(d);
854    ll cost = 0;
fea    rep(i, 0, n) {
c69    if (d[i] > 0) {
f56    cost += d[i];
4f6    Dinic::add(n, i, d[i]);
551    } else if (d[i] < 0) {
bd2    Dinic::add(i, n+1, -d[i]);
bd9    }
a13 }

9f2    return (Dinic::maxFlow(n, n+1) == cost);
cc6 }

7bd    bool has_flow(int src, int snk) {
eda    Dinic::add(snk, src, INF);
e40    return has_circulation();
4aa }
4eb    ll max_flow(int src, int snk) {
ee8    if (!has_flow(src, snk)) return -1;
99c    Dinic::F = 0;
703    return Dinic::maxFlow(src, snk);
0bb }
756 };

```

## MinCost.h

**Description:** Min-cost max-flow. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only. If graph is a DAG pi can be calculated with DP instead of Bellman ford.

**Time:**  $\mathcal{O}(FE \log(V))$  where F is max flow.  $\mathcal{O}(VE)$  for setpi.

6f4fae, 95 lines

```

c4d #include <bits/extc++.h>

9f4 const ll INF = numeric_limits<ll>::max() / 4;

6f3 struct MCMF {
670     struct edge {
ede        int from, to, rev;
e20        ll cap, cost, flow;
092    };
060    int N;
091    vector<vector<edge>> ed;
a83    vector<int> seen, vis;
0ec    vector<ll> dist, pi;
c45    vector<edge*> par;

2cc    MCMF(int N) : N(N), ed(N), seen(N), vis(N),
dc7        dist(N), pi(N), par(N) {}

6f3 void addEdge(int from, int to, ll cap, ll cost) {
ad8    if (from == to || cap == 0) return;
1af    ed[from].push_back(edge{from,to,sz(ed[to]),cap,cost,0
});}
700    ed[to].push_back(edge{to,from,sz(ed[from])-1,0,-cost,0
});}
dad    }

975    void path(int s) {
7d4        fill(all(seen), 0);
04e        fill(all(dist), INF);
a93        dist[s] = 0;
841        ll di;
937        __gnu_pbds::priority_queue<pair<ll, int>> q;
9fb        vector<decaytype(g)::point_iterator> its(N);
23b        q.push({0, s});

14d        while (!q.empty()) {
eda            s = q.top().second; q.pop();
2af            seen[s] = 1; di = dist[s] + pi[s];

```

## MinCost PushRelabel Blossom

```

6bd    for (edge& e : ed[s]) {
d20        if (!seen[e.to]) {
f1f            ll val = di - pi[e.to] + e.cost;
f3c            if (e.cap - e.flow > 0 && val < dist[e.to]){
0c7                dist[e.to] = val;
fb6                par[e.to] = &e;
22d                if (its[e.to] == q.end()) {
aac                    its[e.to] = q.push({-dist[e.to], e.to});
388                } else q.modify(its[e.to], {-dist[e.to], e.to});
80b                }
fce            }
013        }
e16    }
faa    for (int i = 0; i < N; i++) {
0ef        pi[i] = min(pi[i] + dist[i], INF);
ded    }

310    pair<ll, ll> maxflow(int s, int t) {
923        setpi(s, t);
3d3        ll totflow = 0, totcost = 0;
8dd        while (path(s), seen[t]) {
535            ll fl = INF;
733            for (edge* x = par[t]; x; x = par[x->from]) {
8ed                fl = min(fl, x->cap - x->flow);
ddf                }
f9f                totflow += fl;
733                for (edge* x = par[t]; x; x = par[x->from]) {
10b                    x->flow += fl;
e58                    ed[x->to][x->rev].flow -= fl;
3bf                }
219            }
faa            for (int i = 0; i < N; i++) {
a18                for (edge& e : ed[i]) {
7a0                    totcost += e.cost * e.flow;
774                }
a06            }
411            return {totflow, totcost / 2};
}

// If some costs can be negative, call this before
// maxflow:
eda    void setpi(int s, int t) {
3ef        fill(all(pi), INF);
156        pi[s] = 0;
45c        int it = N, ch = 1;
aa3        ll v;
5e8        while (ch-- && it--) {
faa            for (int i = 0; i < N; i++) {
c9b                if (pi[i] != INF)
fb0                    for (edge& e : ed[i]) if (e.cap)
257                        if ((v = pi[i] + e.cost) < pi[e.to])
a43                            pi[e.to] = v, ch = 1;
d0b                }
250            }
38b            assert(it >= 0); // negative cost cycle
545        }
f1d    }

PushRelabel.h

```

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

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

```

a7bbd5, 55 lines

49f struct PushRelabel {
e9b     struct Edge {
548         int dest, back;
e00         ll f, c;
571     };
ed3     vector<vector<Edge>> g;
51c     vector<ll> ec;
658     vector<Edge*> cur;
b08     vector<vector<int>> hs;
4d4     vector<int> H;
4e1     PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}

b1c     void addEdge(int s, int t, ll cap, ll rcap=0) {
50b        if (s == t) return;
cc8        g[s].push_back({t, sz(g[t]), 0, cap});
2aa        g[t].push_back({s, sz(g[s])-1, 0, rcap});
817    }

359    void addFlow(Edge& e, ll f) {
759        Edge &back = g[e.dest][e.back];
f7e        if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
d2e        e.f += f; e.c -= f; ec[e.dest] += f;
c47        back.f -= f; back.c += f; ec[back.dest] -= f;
340    }

0e0    ll calc(int s, int t) {
f00        int v = sz(g); H[s] = v; ec[t] = 1;
fbb        vector<int> co(2*v); co[0] = v-1;
e20        for (int i=0; i<v; i++) cur[i] = g[i].data();
8c2        for (Edge& e : g[s]) addFlow(e, e.c);

604    for (int hi = 0;;) {
ae9        while (hs[hi].empty()) if (!hi--) return -ec[s];
c6f        int u = hs[hi].back(); hs[hi].pop_back();
a3e        while (ec[u] > 0) // discharge u
457            if (cur[u] == g[u].data() + sz(g[u])) {
e94                H[u] = 1e9;
5fa                for (Edge& e : g[u]) {
256                    if (e.c && H[u] > H[e.dest]+1)
740                        H[u] = H[e.dest]+1, cur[u] = &e;
88f                }
f04                if (++co[H[u]], !--co[hi] && hi < v){
10d                    for (int i=0; i<v; i++){
4be                        if (hi < H[i] && H[i] < v)
021                            --co[H[i]], H[i] = v + 1;
a21                    }
cc1                }
3a2                hi = H[u];
b6b                } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1){
779                    addFlow(*cur[u], min(ec[u], cur[u]->c));
e91                }else ++cur[u];
4d7            }
b65        }
385    bool inCut(int a) { return H[a] >= sz(g); }
a7b    };

Blossom.h

```

**Description:** Max matching on general Graph.  $mate[i]$  = match of  $i$

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

21cc7b, 56 lines

```

40f vector<int> Blossom(vector<vector<int>>& g) {
10a    int n = sz(g), timer = -1;
f55    vector<int> mate(n, -1), label(n), par(n), orig(n), aux(n,
-1), q;

060    auto lca = [&](int x, int y) {
7b8        for (timer++; ; swap(x, y)) {
583            if (x == -1) continue;
4be            if (aux[x] == timer) return x;
90d            aux[x] = timer;
fb4            x=(mate[x] == -1 ? -1 : orig[par[mate[x]]]);
f6a        }
aba    };

```

```

be4    auto blossom = [&](int v, int w, int a) {
509        while (orig[v] != a) {
721            par[v] = w; w = mate[v];
1e2            if(label[w] == 1) label[w] = 0, q.push_back(w);
8c7            orig[v] = orig[w] = a;
3d0            v = par[w];
eae        }
068    };
a0f    auto aug = [&](int v) {
8c8        while (v != -1) {
86a            int pv = par[v], nv = mate[pv];
941            mate[v] = pv; mate[pv] = v; v = nv;
ba8        }
54c    };
9f9    auto bfs = [&](int root) {
be5        fill(all(label), -1);
652        iota(all(orig), 0);
4b6        q.clear();
594        label[root] = 0; q.push_back(root);
a43        rep(i, 0, sz(q)) {
4c1            int v = q[i];
5aa            for (auto x : g[v]) {
464                if (label[x] == -1) {
73a                    label[x] = 1; par[x] = v;
1bd                    if (mate[x] == -1) return aug(x), 1;
8d9                    label[mate[x]] = 0;
de3                    q.push_back(mate[x]);
641                }
018                else if (!label[x] && orig[v] != orig[x]) {
37f                    int a = lca(orig[v], orig[x]);
f12                    blossom(x, v, a);
183                    blossom(v, x, a);
405                }
ab5            }
9e2        }
bb3        return 0;
139    };
// Time halves if you start with (any) maximal
// matching.
fea    rep(i, 0, n) {
698        if (mate[i] == -1) bfs(i);
7b5    }
568    return mate;
21c }

```

## HopcroftKarp.h

Description: ans is the size of the max matching.

The match of x is  $l[x]$

Usage: HopcroftKarp(|X|, |Y|, edges(x, y))

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

c4f2f2, 46 lines

```

725    struct HopcroftKarp {
e40        vector<int> g, l, r;
959        int ans;
b82        HopcroftKarp(int n, int m, vector<pii> e)
            : g(sz(e)), l(n, -1), r(m, -1), ans(0) {
aa0            shuffle(all(e), rng);
322            vector<int> deg(n + 1);
235            for (auto& [x, y] : e) deg[x]++;
b4a            rep(i, 1, n+1) deg[i] += deg[i - 1];
85a            for (auto& [x, y] : e) g[--deg[x]] = y;

5ae            vector<int> q(n);
667            while (true) {
661                vector<int> a(n, -1), p(n, -1);
6bb                int t = 0;
fea                rep(i, 0, n) {
4b1                    if (l[i] == -1) {
b53                        q[t++] = a[i] = p[i] = i;

```

```

4b6                    }
62e                    }
a15                    }
edb                    }
912                    }
08c                    }
0ba                    }
360                    }
89a                    }
d3b                    }
ee7                    }
dbb                    }
2a5                    }
ebf                    }
6aa                    }
c2b                    }
b54                    }
f06                    }
a74                    }
d11                    }
9ef                    }
e8a                    }
0ab                    }
984                    }
bc5                    }
6ec                    }
c4f                    }

f55                    }
193                    }
b84                    for (int j = 1; j < m; j++) {
eb3                        if (p[j]) ans[p[j] - 1] = j - 1;
c9a                    }
def                    return { -v[0], ans }; // min cost
4a7    }

```

## GlobalMinCut.h

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

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

8b0e19, 22 lines

```

192    pair<int, vi> globalMinCut(vector<vi> mat) {
afa        pair<int, vi> best = {INT_MAX, {}};
755        int n = sz(mat);
91d        vector<vi> co(n);
d0f        rep(i, 0, n) co[i] = {i};
488        rep(ph, 1, n) {
2e9            vi w = mat[0];
e44            size_t s = 0, t = 0;
694            rep(it, 0, n-ph) { // O(V^2) -> O(E log V) with prio.
queue
d6e                w[t] = INT_MIN;
a5f                s = t, t = max_element(all(w)) - w.begin();
d39                rep(i, 0, n) w[i] += mat[t][i];
ec9            }
3df                best = min(best, {w[t] - mat[t][t], co[t]}));
096                co[s].insert(co[s].end(), all(co[t]));
959                rep(i, 0, n) mat[s][i] += mat[t][i];
984                rep(i, 0, n) mat[i][s] = mat[s][i];
5dd                mat[0][t] = INT_MIN;
ca0            }
f26            return best;
8b0    }

```

## 7.3 DFS algorithms

### Bridges.h

1fa56b, 24 lines

```

cd9    vector<int> g[ms];
9e4    int low[ms], tin[ms], vis[ms], t;

403    void dfs(int u = 0, int p = -1) {
b9c        vis[u] = true;
b4a        low[u] = tin[u] = t++;
7b9        for (auto v : g[u]) {
530            if (v == p) continue;
c84            if (vis[v]) {
34f                low[u] = min(low[u], tin[v]);
728            }
4e6            else {
95e                dfs(v, u);
ab6                low[u] = min(low[u], low[v]);
795                // if (low[v] >= tin[u] && p != -1), U is an
// articulation point
4b8                if (low[v] > tin[u]) {
308                    // edge from U to V is a bridge
4b8                    // children++
862                }
677                // if(children > 1 && p == -1) root is an articulation
// point
30c    }

```

### BridgeOnline.h

**Description:** Maintains bridges and 2-edge-connected components (2-ECC) incrementally.  $\text{ds}[0]$  tracks Connected Components (CC).  $\text{ds}[1]$  tracks 2-ECCs. Nodes  $u, v$  are in the same 2-ECC iff  $\text{dsfind}(u, 1) == \text{dsfind}(v, 1)$ .  $g$  stores the spanning forest edges (edges that were bridges when added). An edge  $(u, v) \in g$  is a current bridge iff  $\text{dsfind}(u, 1) != \text{dsfind}(v, 1)$ .  $\text{bridges}$  tracks the total count of active bridges. Use  $\text{init}()$  before starting.

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

```

4dd int bridges;
801 int ds[2][ms], sz[2][ms];
87b int h[ms], pai[ms], old[ms];
cd9 vector<int> g[ms];

ca2 void init() {
786     bridges = 0;
f0d     rep(i, 0, ms) {
a4e         g[i].clear(), h[i] = 0;
606         ds[0][i] = ds[1][i] = i;
8f3         sz[0][i] = sz[1][i] = 1;
4a6     }
c1e }

243 int dsfind(int j, int i) {
7fa     if(j == ds[i][j]) return ds[i][j];
db7     return ds[i][j] = dsfind(ds[i][j], i);
4a4 }

b55 void dfs(int u, int p, int l) {
40d     h[u] = l;
49e     pai[u] = p;
a32     old[u] = dsfind(u, 1);
4d5     for (int v : g[u]) {
730         if (v == p) continue;
0c5         dfs(v, u, l + 1);
11d     }
f2e }

94c void updateNodes(int u, int p) {
840     if (old[u] == old[p]) {
dc4         ds[1][u] = ds[1][p];
574     }
e79     else ds[1][u] = u;
4d5     for (int v : g[u]) {
730         if (v == p) continue;
0c1         updateNodes(v, u);
42a     }
329 }

814 void mergeTrees(int a, int b) {
cbf     bridges++;
5cb     int iniA = a, iniB = b;
19d     a = dsfind(a, 0), b = dsfind(b, 0);
834     if (sz[0][a] < sz[0][b]) swap(a, b), swap(iniA, iniB);
e14     dfs(iniB, iniA, h[iniA] + 1);
376     old[iniA] = -1;
ee0     updateNodes(iniB, iniA);
86b     ds[0][b] = a;
013     sz[0][a] += sz[0][b];
c9a }

416 void removeBridges(int a, int b) {
532     a = dsfind(a, 1), b = dsfind(b, 1);
984     while (a != b) {
e7a         bridges--;
54b         if (h[a] < h[b]) swap(a, b);
// ponte entre (a, pai[a]) deixou de existir
9f6         ds[1][a] = dsfind(pai[a], 1);
e40         a = ds[1][a];
cda         }

```

## BlockCutTree DominatorTree

```

02b void addEdge(int a, int b) {
7b9     if (dsfind(a, 0) == dsfind(b, 0)) { 69d
69d         removeBridges(a, b);
221     }
4e6     else {
447         // nova ponte entre (a, b)
025         g[a].push_back(b);
3e9         g[b].push_back(a);
f8e         mergeTrees(a, b);
447     }
e57 }

```

## BlockCutTree.h

**Description:** Constructs the Block-Cut Tree, which is a bipartite graph with blocks (maximal 2-vertex-connected components) on one side and articulation points on the other. Works for disconnected graphs. Tree size is  $\leq 2N$ . Be careful with self loops and multi edges. art[i]: number of new components created by removing  $i$  (AP if  $\geq 1$ ). blocks[i]: vertices/edges of block  $i$ . tree[i]: the tree node index corresponding to block  $i$ . pos[i]: the tree node index corresponding to vertex  $i$ .

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

e55ab0, 66 lines

```

d10 struct block_cut_tree {
d8e     vector<vector<int>> g, blocks, tree;
43b     vector<vector<pair<int, int>>> edgblocks;
4ce     stack<int> s;
6c0     stack<pair<int, int>> s2;
2bb     vector<int> id, art, pos;

763     block_cut_tree(vector<vector<int>> g_) : g(g_) {
625         int n = sz(g);
37a         id.resize(n, -1), art.resize(n), pos.resize(n);
6f2         build();
246     }

df6     int dfs(int i, int& t, int p = -1) {
cf0         int lo = id[i] = t++;
18e         s.push(i);

827         if (p != -1) s2.emplace(i, p);
43f         for (int j : g[i])
6bf             if (j != p and id[j] != -1) s2.emplace(i, j);

cac         for (int j : g[i]) if (j != p) {
9a3             if (id[j] == -1) {
121                 int val = dfs(j, t, i);
0c3                 lo = min(lo, val);

588                 if (val >= id[i]) {
66a                     art[i]++;
483                     blocks.emplace_back(1, i);
110                     while (blocks.back().back() != j)
138                         blocks.back().push_back(s.top()), s.pop();

128                     edgblocks.emplace_back(1, s2.top()), s2.pop();
904                     while (edgblocks.back().back() != pii(j, i))
bce                         edgblocks.back().push_back(s2.top()), s2.pop();
041                 }
38c             }
328         else lo = min(lo, id[j]);
5b6     }
924     if (p == -1) {
2db         if (art[i]) art[i]--;
4e6         else{
483             blocks.emplace_back(1, i);
433             edgblocks.emplace_back();
333         }

```

```
384         }
253     return lo;
6d7 }

0a8 void build() {
6bb     int t = 0;
c80     rep(i, 0, sz(g)) if(id[i] == -1) dfs(i, t, -1);
d0e tree.resize(sz(blocks));
008     rep(i, 0, sz(g)) if (art[i])
b9a         pos[i] = sz(tree), tree.emplace_back();
05c
403     rep(i, 0, sz(blocks)) for (int j : blocks[i]) {
4e6         if (!art[j]) pos[j] = i;
4e6         else {
49d             tree[i].push_back(pos[j]);
9a7             tree[pos[j]].push_back(i);
01e         }
27c     }
5a7 }
e55 }
```

## DominatorTree.h

**Description:** Builds the Dominator Tree of a directed graph rooted at `root`. Node  $u$  dominates  $v$  if every path from `root` to  $v$  passes through  $u$ . The immediate dominator of  $v$  is the unique dominator closest to  $v$  (excluding  $v$ ). Returns a vector `par` where `par[u]` is the parent of  $u$  in the tree. Roots and unreachable nodes satisfy `par[u] = u`.

Time:  $\mathcal{O}(M \log N)$  8c4613, 55 lines

```

3db struct dominator_tree {
577     int n, t;
324     vector<vector<int>> g, rg, bucket;
7f3     vector<int> arr, par, rev, sdom, dom, ds, lbl;
226     dominator_tree(int n) : n(n), t(0), g(n), rg(n), bucket(n),
7a1         arr(n, -1), par(n), rev(n), sdom(n), dom(n), ds(n), lbl(n) {}
c2b     void add_edge(int u, int v) { g[u].push_back(v); }

315     void dfs(int u) {
12e         arr[u] = t;
64f         rev[t] = u;
bad         lbl[t] = sdom[t] = ds[t] = t;
c82         t++;
6f1         for (int w : g[u]) {
0c2             if (arr[w] == -1) {
8c6                 dfs(w);
81a                 par[arr[w]] = arr[u];
869                 }
f8e                 rg[arr[w]].push_back(arr[u]);
93a             }
b04         }
792         int find(int u, int x=0) {
9fe             if (u == ds[u]) return x ? -1 : u;
41f             int v = find(ds[u], x+1);
388             if (v < 0) return u;
b30             if (sdm[lbl[ds[u]]] < sdom[lbl[u]]) lbl[u] = lbl[ds[u]];
300             ds[u] = v;
784             return x ? v : lbl[u];
a59         }

46f     vector<int> run(int root) {
14e         dfs(root);
b81         iota(all(dom), 0);
da8         for (int i=t-1; i>=0; i--) {
76c             for (int w : rg[i]) sdom[i] = min(sdom[i], sdom[find(w)]);
1);
c94             if (i) bucket[sdom[i]].push_back(i);
3b2             for (int w : bucket[i]) {

```

```

46a     int v = find(w);
ae4     if (sdom[v] == sdom[w]) dom[w] = sdom[w];
41c     else dom[w] = v;
1e6 }
fd8     if (i > 1) ds[i] = par[i];
b9e }
e8f     rep(i, 1, t) {
7d7     if (dom[i] != sdom[i]) dom[i] = dom[dom[i]];
32d }
af8     vector<int> par(n);
2c2     iota(all(par), 0);
533     rep(i, 0, t) par[rev[i]] = rev[dom[i]];
148     return par;
900 }
8c4 };

```

## EulerPath.h

**Description:** Receives as input graph(node, edge index), number of edges and source. Returns list of node, index of edge he came from, if path/circuit does not exists returns empty list.

a3ed13, 27 lines

```

b4a     vector<pii> eulerPath(const vector<vector<pii>>& g, int
    nedges, int src) {
625     int n = sz(g);
b47     vector<int> deg(n, 0), its(n, 0), used(nedges + 1, 0);
a42     vector<pii> s = { {src, -1} };
//deg[src]++; //to allow paths, not only circuits
a5f     vector<pii> ret;
980     while (!s.empty()) {
        int u = s.back().first, &it = its[u];
c45     if (it == sz(g[u])) {
            ret.push_back(s.back());
            s.pop_back();
            continue;
        }
        auto& [nxt, id] = g[u][it++];
        if (!used[id]) {
            deg[u]--, deg[nxt]++;
            used[id] = 1;
            s.push_back({ nxt, id });
777        }
    }
    for (int x : deg) {
518        if (x < 0 || sz(ret) != (nedges + 1)) return {};
26e    }
969    reverse(ret.begin(), ret.end());
edf    return ret;
a3e }

```

## SCC.h

**Description:** Kosaraju algorithm for calculating strongly connected components. Components are ordered in topological order.

008ff2, 36 lines

```

bf0     struct SCC {
dab     int n, ncomp;
0e3     vector<vector<int>> g, inv;
829     vector<int> comp, vis, stk;
8b6     SCC(){}
471     SCC(int n)
        : n(n), ncomp(0), g(n), inv(n), comp(n, -1), vis(n){}
315     void dfs(int u) {
150         vis[u] = 1;
        for (int v : g[u]) if (!vis[v]) dfs(v);
967         stk.push_back(u);
37b     }
f20     void dfs_inv(int u) {
62c         comp[u] = ncomp;
3a5         for (int v : inv[u]) {

```

```

df4             if (comp[v] == -1) dfs_inv(v);
0a0         }
984     }
63d     void solve() {
603         for (int i = 0; i < n; i++) {
b65             if (!vis[i]) dfs(i);
358         }
340         reverse(all(stk));
49b         for (int u : stk) {
9ef             if (comp[u] != -1) continue;
672             dfs_inv(u);
a8f             ncomp++;
ecb         }
ef8     }
010     void add_edge(int a, int b) {
025         g[a].push_back(b);
a6a         inv[b].push_back(a);
1ec     }
008 };

```

## TwoSat.h

**Usage:** not A = ~A

c8b989, 37 lines

```

d9d     struct TwoSat{
1a8     int n;
3c9     SCC scc;
7c7     vector<int> value;
425     vector<pii> e;
e2c     TwoSat(int n) : n(n){}
6c0     bool solve(){
b36         value.resize(n);
8cc         scc = SCC(2*n);
1f3         for(auto &x : e) scc.add_edge(x.first, x.second);
7f9         scc.solve();
3df         for(int i=0; i<2*n; i++)
f83             if(scc.comp[i] == scc.comp[i^1]) return false;
830         for(int i=0; i<n; i++)
733             value[i] = scc.comp[id(i)] > scc.comp[id(~i)];
8a6         return true;
949     }
a0a     void atMostOne(vector<int> &li){
615         if(sz(li) <= 1) return;
da9         int cur = ~li[0];
b25         for(int i = 2; i < sz(li); i++) {
abb             int next = n+i;
e0a             addOr(cur, ~li[i]);
f26             addOr(cur, next);
7ba             addOr(~li[i], next);
072             cur = ~next;
e3d         }
921         addOr(cur, ~li[1]);
bbb     }
41b     int id(int v) { return v < 0 ? (~v) * 2 ^ 1 : v * 2; }
276     void add(int a, int b) { e.push_back({id(a), id(b)}); }
bc7     void addOr(int a, int b) { add(~a, b); add(~b, a); }
671     void addImp(int a, int b) { addOr(~a, b); }
d9d     void addEqual(int a, int b){ addOr(a, ~b); addOr(~a, b);
}
ec3     void isFalse(int a) { addImp(a, ~a); }
c8b     };

```

## 7.4 Coloring

## EdgeColoring.h

**Description:** Given a simple, undirected graph with max degree  $D$ , computes a  $(D + 1)$ -coloring of the edges such that no neighboring edges share a color. ( $D$ -coloring is NP-hard, but can be done for bipartite graphs by repeated matchings of max-degree nodes.)

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

```

e210e2, 32 lines
f41     vi edgeColoring(int N, vector<pii> eds) {
727         vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
10d         for (pii e : eds) ++cc[e.first], ++cc[e.second];
e2f         int u, v, ncols = *max_element(all(cc)) + 1;
fda         vector<vi> adj(N, vi(ncols, -1));
6ec         for (pii e : eds) {
119             tie(u, v) = e;
e51             fan[0] = v;
loc.assign(ncols, 0);
696             int at = u, end = u, d, c = free[u], ind = 0, i = 0;
3b2             while (d = free[v], !loc[d] && (v = adj[u][d]) != -1) {
3e1                 loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
01e                 cc[loc[d]] = c;
997                 for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at][cd]
}) {
4ff                     swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
79f                     while (adj[fan[i]][d] != -1) {
a9f                         int left = fan[i], right = fan[+i], e = cc[i];
99b                         adj[u][e] = left;
ccb                         adj[left][e] = u;
f7e                         adj[right][e] = -1;
d99                         free[right] = e;
316                     }
dfd                         adj[u][d] = fan[i];
c45                         adj[fan[i]][d] = u;
0e1                         for (int y : {fan[0], u, end}) {
3fa                             for (int z = free[y] = 0; adj[y][z] != -1; z++);
fdc                         }
29d                         rep(i, 0, sz(eds))
961                             for (tie(u, v) = eds[i]; adj[u][ret[i]] != v; ++ret[i])
];
edf                         return ret;
e21     }

```

## 7.5 Heuristics

## MaxClique.h

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

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

2eeaf4, 53 lines

```

db9     using vb = vector<bitset<200>>;
c7d     struct Maxclique {
24e     double limit=0.025, pk=0;
c04     struct Vertex { int i, d=0; };
547     using vv = vector<Vertex>;
d44     vb e;
df7     vv V;
e5c     vector<vector<int>> C;
497     vector<int> qmax, q, S, old;
fe3     void init(vv& r) {
fd3         for (auto& v : r) v.d = 0;
583         for (auto v : r) for (auto j : r) v.d += e[v.i][j.i];
0f1         sort(all(r), [] (auto a, auto b) { return a.d > b.d; });
c43         int mxd = r[0].d;
3f8         for(int i=0; i<sz(r); i++) r[i].d = min(i, mxd) + 1;
526     }
bc8     void expand(vv& R, int lev = 1) {
ac1         S[lev] += S[lev - 1] - old[lev];
92c         old[lev] = S[lev - 1];
d18         while (sz(R)) {
3fd             if (sz(q) + R.back().d <= sz(qmax)) return;
d62             q.push_back(R.back().i);
f28             vv T;
7fb             for(auto v : R) {
if (e[R.back().i][v.i]) T.push_back({v.i});

```

```
d21     if (sz(T)) {
eea       if (S[lev]++ / ++pk < limit) init(T);
457       int j = 0, mxk = 1, mnk = max(sz(qmax)-sz(q)+1, 1);
9bc       C[1].clear(), C[2].clear();
969       for (auto v : T) {
bfe         int k = 1;
8f5         auto f = [&](int i) { return e[v.i][i]; };
5c6         while (any_of(all(C[k]), f)) k++;
782         if (k > mxk) mxk = k, C[mxk + 1].clear();
18a         if (k < mnk) T[j++].i = v.i;
0e6           C[k].push_back(v.i);
322       }
238       if (j > 0) T[j - 1].d = 0;
d2f       for (int k=mnk; k<mxk + 1; k++) {
5bf         for (int i : C[k])
361           T[j].i = i, T[j++].d = k;
9dc       }
22d       expand(T, lev + 1);
61f     } else if (sz(q) > sz(qmax)) qmax = q;
c81     q.pop_back(), R.pop_back();
3e0   }
81d }
b2d vector<int> maxClique(){ init(V), expand(V); return qmax; }
b40 Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S) {
0d1   for (int i=0; i<sz(e); i++) V.push_back({i});
b60 }
534 }
```

## MaximalCliques.h

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

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

b0d5b1, 13 lines

```
753 typedef bitset<128> B;
044 template<class F>
6a9 void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B R ={} ) {
9bb   if (!P.any()) { if (!X.any()) f(R); return; }
a8e   auto q = (P | X).Find_first();
cd1   auto cands = P & ~eds[q];
3d7   rep(i,0,sz(eds)) if (cands[i]) {
a75     R[i] = 1;
e78     cliques(eds, f, P & eds[i], X & eds[i], R);
bb6     R[i] = P[i] = 0; X[i] = 1;
181   }
c9d }
```

## 7.6 Trees

### Centroid.h

**Description:** Call decomp(0) to solve, marked array should be initially set to zero.

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

b73755, 27 lines

```
6b6 int tam[ms], marked[ms];
2a1 int calc_tam(int u, int p) {
5d1   tam[u] = 1;
4d5   for (int v : g[u]) {
5ea     if (v != p && !marked[v]) tam[u] += calc_tam(v, u);
d09   }
f95   return tam[u];
d5d }
5fb int get_centroid(int u, int p, int tot) {
4d5   for (int v : g[u]) {
38c     if (v != p && !marked[v] && (tam[v] > (tot / 2)))
32c       return get_centroid(v, u, tot);
```

```
b6c     }
03f     return u;
0c7   }
// Cent is a child of P in the centroid tree
179   void decomp(int u, int p = -1) {
308     calc_tam(u, -1);
bd4     int cent = get_centroid(u, -1, tam[u]);
83d     marked[cent] = 1;
9f1     for (int v : g[cent]) {
c6e       if (!marked[v]) decomp(v, cent);
194     }
dc1   }
```

## HLD.h

**Description:** If values are stored on edges, set EDGE = true and store each edge's value at the endpoint farther from the root (the deeper node). rp[i] is the representative (head) of the heavy path containing node i: it is the node in that chain that is closest to the root.

a129d6, 51 lines

```
5f2   template<bool EDGE> struct HLD {
577     int n, t;
789     vector<vector<int>> g;
003     vector<int> pai, rp, tam, pos, val, arr;
f1e     Seg seg;
bcf     HLD(int n, vector<vector<int>>& g, vector<int>& val)
      : n(n), t(0), g(g), pai(n), rp(n), tam(n, 1),
616       pos(n), val(val), arr(n) {
f80       calc_tam(0, -1);
c91       dfs(0, -1);
d14       seg.build(arr);
a43     }
```

```
2a1     int calc_tam(int u, int p) {
49e       pai[u] = p;
704       for (int& v : g[u]) {
730         if (v == p) continue;
2e4         tam[u] += calc_tam(v, u);
2d5         if (tam[v] > tam[g[u][0]] || g[u][0] == p)
a7f           swap(g[u][0], v);
0a3       }
f95       return tam[u];
c19     }
```

```
fb6     void dfs(int u, int p) {
4c8       pos[u] = t++;
d7b       arr[pos[u]] = val[u];
4d5       for (int v : g[u]) {
730         if (v == p) continue;
84d         rp[v] = (v == g[u][0] ? rp[u] : v);
95e         dfs(v, u);
42d       }
de1     }
```

```
4ea     int query(int a, int b) { // query on the path from a
to b
1a4       int ans = 0; // neutral value
34d       while (rp[a] != rp[b]) {
a11         if (pos[a] < pos[b]) swap(a, b);
9a5         ans = max(ans, seg.query(pos[rp[a]], pos[a]));
677         a = pai[rp[a]];
}
9bc         if (pos[a] > pos[b]) swap(a, b);
0f8         ans = max(ans, seg.query(pos[a] + EDGE, pos[b]));
ba7         return ans;
e8a     }
534     void update(int a, int x) {
e5e       seg.update(pos[a], x);
5db     }
```

a12 ;

## LCA.h

**Description:** LCA algorithm using binary lifting,  $is\_ancestor(a, b)$  returns true if  $a$  is an ancestral of  $b$  and false otherwise.

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

db7791, 26 lines

```
67e   int tin[MAXN], tout[MAXN], timer=0;
768   int up[MAXN][BITS];
fb6   void dfs(int u, int p) {
545     tin[u] = timer++;
532     for (int i=1; i<BITS; i++) {
88a       up[u][i] = up[up[u][i-1]][i-1];
4a0     }
712     for (int v : g[u]) if (v != p) dfs(v, u);
4f8     tout[u] = timer;
4a1   }
```

```
f31   bool is_ancestor(int u, int v) {
d34     return (tin[u] <= tin[v] && tout[u] >= tout[v]);
f9f   }
```

```
310   int lca(int u, int v){
bd5     if (is_ancestor(u, v)) return u;
6fc     if (is_ancestor(v, u)) return v;
3c3     for (int i=BITS-1; i>=0; i--) {
3a3       if (up[u][i] && !is_ancestor(up[u][i], v)) {
c3f         u = up[u][i];
49e       }
dc4     }
c15     return up[u][0];
001   }
```

## VirtualTree.h

**Description:** Given a rooted tree and a subset  $S$  of nodes, compute the minimal subtree that contains all the nodes by adding all (at most  $|S| - 1$ ) pairwise LCA's and compressing edges. virt[u] is the adjacency list of the virtual tree: it stores pairs (v, dist), where v is a neighbor of u in the virtual tree and dist is the distance between u and v in the original tree.

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

"LCA.h"

0b1 vector<pair<int, int>> virt[ms];

d0c void build\_virt(vector<int>& v) {
078 auto cmp = [=](int i, int j){ return tin[i] < tin[j]; };
b84 sort(all(v), cmp);
1ee for (int i = 0, n = sz(v); i + 1 < n; i++) {
4cf v.push\_back(lca(v[i], v[i + 1]));
b84 sort(all(v), cmp);
64f v.erase(unique(all(v)), v.end());
7b4 stack<int> st;
3a7 for (auto u : v) {
c53 if (st.empty()) {
4a6 st.push(u);
e82 }
4e6 else {
7eb while (sz(st) && !is\_ancestor(st.top(), u)) st.pop();
88b int p = st.top();
bfa virt[p].emplace\_back(u, abs(lvl[u] - lvl[p]));
0a5 virt[u].emplace\_back(p, abs(lvl[u] - lvl[p]));
4a6 st.push(u);
92c }
f46 }
c83 }

## DirectedMST.h

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

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

```
.../data-structures/UnionFindRollback.h"
39e620, 61 lines
030 struct Edge { int a, b; ll w; };
bf2 struct Node {
25f   Edge key;
c17   Node *l, *r;
981   ll delta;
a9c   void prop() {
6f9     key.w += delta;
d2d     if (l) l->delta += delta;
d86     if (r) r->delta += delta;
978     delta = 0;
}d3;
866   Edge top() { prop(); return key; }
ab4 };
3eb   Node *merge(Node *a, Node *b) {
b9f     if (!a || !b) return a ?: b;
626     a->prop(), b->prop();
dc2     if (a->key.w > b->key.w) swap(a, b);
485     swap(a->l, (a->r = merge(b, a->r)));
3f5     return a;
c51 }
7bb   void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }

002 pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
8df   RollbackUF uf(n);
3f8   vector<Node*> heap(n);
563   for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{});
});d2;
11 res = 0;
517   vi seen(n, -1), path(n), par(n);
559   seen[r] = r;
dd6   vector<Edge> Q(n), in(n, {-1, -1}), comp;
111   deque<tuple<int, int, vector<Edge>>> cycs;
328   rep(s, 0, n) {
3cb     int u = s, qi = 0, w;
a0a     while (seen[u] < 0) {
572       if (!heap[u]) return {-1, {}};
ebe     Edge e = heap[u]->top();
5ed     heap[u]->delta -= e.w, pop(heap[u]);
952     Q[qi] = e, path[qi++] = u, seen[u] = s;
d56     res += e.w, u = uf.find(e.a);
9e2     if (seen[u] == s) {
28d       Node* cyc = 0;
cab       int end = qi, time = uf.time();
f38       do cyc = merge(cyc, heap[w = path[--qi]]);
4f9       while (uf.join(u, w));
562       u = uf.find(u), heap[u] = cyc, seen[u] = -1;
c06       cycs.push_front({u, time, {&Q[qi], &Q[end]}});
00a     }
c8f   }
068   rep(i, 0, qi) in[uf.find(Q[i].b)] = Q[i];
fa3 }

e41   for (auto& [u, t, comp] : cycs) { // restore sol (optional)
36c     uf.rollback(t);
1d0     Edge inEdge = in[u];
251     for (auto& e : comp) in[uf.find(e.b)] = e;
56d     in[uf.find(inEdge.b)] = inEdge;
4f9   }
427   rep(i, 0, n) par[i] = in[i].a;
efb   return {res, par};
efa }
```

# Geometry (8)

## 8.1 Geometric primitives

### Point.h

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

```
47ec0a, 29 lines
48b   template <class T> int sgn(T x) { return (x > 0) - (x < 0) }
; }
4fc   template<class T>
f26   struct Point {
ea4     typedef Point P;
645     T x, y;
ea6     explicit Point(T x=0, T y=0) : x(x), y(y) {}
0d0     bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y) }
; }
ec7     bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y) }
; }
279     P operator+(P p) const { return P(x+p.x, y+p.y); }
40d     P operator-(P p) const { return P(x-p.x, y-p.y); }
e03     P operator*(T d) const { return P(x*d, y*d); }
0b9     P operator/(T d) const { return P(x/d, y/d); }
57b     T dot(P p) const { return x*p.x + y*p.y; }
460     T cross(P p) const { return x*p.y - y*p.x; }
b3d     T cross(P a, P b) const { return (a-*this).cross(b-*this) }
; }
f68     T dist2() const { return x*x + y*y; }
18b     double dist() const { return sqrt((double)dist2()); }
// angle to x-axis in interval [-pi, pi]
907     double angle() const { return atan2(y, x); }
d06     P unit() const { return *this/dist(); } // makes dist()=1
200     P perp() const { return P(-y, x); } // rotates +90
degrees
852     P normal() const { return perp().unit(); }
// returns point rotated 'a' radians ccw around the
origin
f23     P rotate(double a) const {
482       return P(x*cos(a)-y*sin(a), x*sin(a)+y*cos(a)); }
902     friend ostream& operator<<(ostream& os, P p) {
9a9       return os << "(" << p.x << ", " << p.y << ")";
d2d     }
};
```

### lineDistance.h

#### Description:

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

"Point.h"

f6bf6b, 5 lines

```
7dc   template<class P>
0bf   double lineDist(const P& a, const P& b, const P& p) {
14f     return (double)(b-a).cross(p-a)/(b-a).dist();
e07   }
008 }
```

### SegmentDistance.h

#### Description:

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

**Usage:** Point<double> a, b(2,2), p(1,1);

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

"Point.h"

5c88f4, 7 lines

```
626   typedef Point<double> P;
929   double segDist(P& s, P& e, P& p) {
a44     if (s==e) return (p-s).dist();
```

s e res p

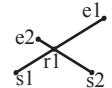
5c88f4, 7 lines

```
f81   auto d = (e-s).dist2(), t = min(d,max(.0, (p-s).dot(e-s)))
; }
2c1   return ((p-s)*d-(e-s)*t).dist()/d;
ae7 }
```

### SegmentIntersection.h

#### Description:

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



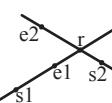
**Usage:** vector<P> inter = segInter(s1,e1,s2,e2);  
if (sz(inter)==1)  
cout << "segments intersect at " << inter[0] << endl;  
"Point.h", "OnSegment.h"

```
9d57f2, 14 lines
dae template<class P> vector<P> segInter(P a, P b, P c, P d) {
0b6   auto oa = a.cross(d, a), ob = a.cross(d, b),
318     oc = a.cross(b, c), od = a.cross(b, d);
// Checks if intersection is single non-endpoint point.
914   if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
e5b   return {(a * ob - b * oa) / (ob - oa)};
4c1   set<P> s;
ccb   if (onSegment(c, d, a)) s.insert(a);
0ad   if (onSegment(c, d, b)) s.insert(b);
3d8   if (onSegment(a, b, c)) s.insert(c);
2fa   if (onSegment(a, b, d)) s.insert(d);
a35   return {all(s)};
9d5 }
```

### lineIntersection.h

#### Description:

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



**Usage:** auto res = lineInter(s1,e1,s2,e2);

if (res.first == 1)  
cout << "intersection point at " << res.second << endl;  
"Point.h"

```
7dc template<class P>
0bf pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
14f   auto d = (e1 - s1).cross(e2 - s2);
8cc   if (d == 0) // if parallel
d99   return {-(s1.cross(e1, s2) == 0), P(0, 0)};
f6b   auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
9b8   return {1, (s1 * p + e1 * q) / d};
472 }
```

### sideOf.h

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

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

"Point.h"

```
7dc template<class P>
70b   int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }
7dc template<class P>
```

s e res p

3af81c, 10 lines

```
b5e int sideOf(const P& s, const P& e, const P& p, double eps)
{  
79e auto a = (e-s).cross(p-s);
653 double l = (e-s).dist()*eps;
c32 return (a > l) - (a < -l);
33f }
```

**OnSegment.h**

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

`"Point.h"` c597e8, 4 lines

```
514 template<class P> bool onSegment(P s, P e, P p) {
5fb return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
c59 }
```

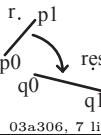
**linearTransformation.h**

**Description:**

Apply the linear transformation (translation, rotation and scaling) which takes line  $p_0-p_1$  to line  $q_0-q_1$  to point r.

`"Point.h"` 03a306, 7 lines

```
626 typedef Point<double> P;
644 P linearTransformation(const P& p0, const P& p1,
f06 const P& q0, const P& q1, const P& r) {
99f P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
0aa return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist
2());
45e }
```

**LineProjectionReflection.h**

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

`"Point.h"` b5562d, 6 lines

```
7dc template<class P>
981 P lineProj(P a, P b, P p, bool refl=false) {
de3 P v = b - a;
3fc return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
4b7 }
```

**Angle.h**

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

**Usage:** `vector<Angle> v = {w[0], w[0].t360() ...}; // sorted`  
`int j = 0; rep(i, 0, n) { while (v[j] < v[i].t180()) ++j; }`  
`// sweeps j such that (j-i) represents the number of positively oriented triangles with vertices at 0 and i` 0f0602, 36 lines

```
755 struct Angle {
e91 int x, y;
8bd int t;
5ac Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
de8 Angle operator-(Angle b) const { return {x-b.x, y-b.y, t} };
3cd int half() const {
840 assert(x || y);
aa4 return y < 0 || (y == 0 && x < 0);
c93 }
dfc Angle t90() const { return {-y, x, t + (half() && x >= 0)} };
726 Angle t180() const { return {-x, -y, t + half()} };
925 Angle t360() const { return {x, y, t + 1}; }
e25 };
a92 bool operator<(Angle a, Angle b) {
// add a.dist2() and b.dist2() to also compare distances
```

```
ea7 return make_tuple(a.t, a.half(), a.y * (11)b.x) <
05f make_tuple(b.t, b.half(), a.x * (11)b.y);
ce5 }

// Given two points, this calculates the smallest angle
// between them, i.e., the angle that covers the defined line
// segment.
908 pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
ee4 if (b < a) swap(a, b);
423 return (b < a.t180()) ?
c35 make_pair(a, b) : make_pair(b, a.t360());
}
5ea Angle operator+(Angle a, Angle b) { // point a + vector b
eb1 Angle r(a.x + b.x, a.y + b.y, a.t);
8ca if (a.t180() < r) r.t--;
d9f return r.t180() < a ? r.t360() : r;
3d8 }
106 Angle angleDiff(Angle a, Angle b) { // angle b - angle a
125 int tu = b.t - a.t; a.t = b.t;
e63 return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
ba3 }
```

**HalfPlane.h**

**Description:** Computes the intersection of a set of half-planes. Half-planes are sorted by angle and processed with a deque, removing redundant or conflicting constraints. Parallel half-planes are handled explicitly. Returns the convex polygon of the intersection, or empty if infeasible.

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

`"Point.h"` cf24a8, 72 lines

```
984 using ld = long double;
207 using P = Point<ld>;
533 struct Hp { // Half plane struct
// 'p' is a passing point of the line and 'pq' is the
// direction vector of the line.
812 P p, pq;
d29 ld angle;
b93 Hp() {}
65d Hp(const P& a, const P& b) : p(a), pq(b - a) {
0e3 angle = atan2(pq.y, pq.x);
}
2ff bool out(const P& r) { return pq.cross(r - p) < -eps; }
d36 bool operator < (const Hp& e) const {
1dd return angle < e.angle;
}
44e friend P inter(const Hp& s, const Hp& t) {
e99 020 ld alpha = (t.p - s.p).cross(t.pq) / s.pq.cross(t.pq);
93b return s.p + (s.pq * alpha);
825 }
b46 };

fa5 vector<P> hp_intersect(vector<Hp>& H) {
12f P box[4] = { P(inf, inf), P(-inf, inf),
968 P(-inf, -inf), P(inf, -inf) };

1cd for(int i = 0; i<4; i++) {
1a8 Hp aux(box[i], box[(i+1) % 4]);
d82 H.push_back(aux);
}
560 sort(all(H));
6c5 deque<Hp> dq;
486 int len = 0;
908 for(int i = 0; i < sz(H); i++) {
3fb while(len>1 && H[i].out(inter(dq[len-1], dq[len-2]))) {
c70 dq.pop_back();
}
c0b --len;
654 }
```

```
a31 }
757 while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
c68 dq.pop_front();
654 --len;
}
1eb a5a if(len && fabsl(H[i].pq.cross(dq[len-1].pq)) < eps) {
25f if (H[i].pq.dot(dq[len-1].pq) < 0.0)
282 return vector<P>();
e7b if (H[i].out(dq[len-1].p)) {
c70 dq.pop_back();
654 --len;
}
2dc else continue;
9a0 }
fc2 dq.push_back(H[i]);
250 ++len;
8ed }
```

```
337 while(len> 2 && dq[0].out(inter(dq[len-1], dq[len-2]))) {
c70 dq.pop_back();
654 --len;
}
faa }
81e while (len > 2 && dq[len-1].out(inter(dq[0], dq[1]))) {
c68 dq.pop_front();
654 --len;
}
694 }
1a3 if (len < 3) return vector<P>();
7e7 vector<P> ret(len);
cc7 for(int i = 0; i+1 < len; i++) {
01e ret[i] = inter(dq[i], dq[i+1]);
00f }
4fd ret.back() = inter(dq[len-1], dq[0]);
edf return ret;
deb }
```

**8.2 Circles****CircleIntersection.h**

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

`"Point.h"` ba7267, 12 lines

```
626 typedef Point<double> P;
27f bool circleInter(P a,P b,double r1,double r2,pair<P, P>*&
out) {
b48 if (a == b) { assert(r1 != r2); return false; }
f30 P vec = b - a;
6c8 double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2;
c28 double p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*
d2;
5b0 if (sum*sum < d2 || dif*dif > d2) return false;
84d P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) /
d2);
21e *out = {mid + per, mid - per};
8a6 return true;
170 }
```

**CircleTangents.h**

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

```
"Point.h" b0153d, 14 lines
7dc template<class P>
3a5 vector<pair<P, P>> tangents(P c1, double r1, P c2, double
r2) {
c0b P d = c2 - c1;
```

```

432     double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
018     if (d2 == 0 || h2 < 0) return {};
c14     vector<pair<P, P>> out;
092     for (double sign : {-1, 1}) {
2ad         P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
2e3         out.push_back({cl + v * r1, c2 + v * r2});
e25     }
b21     if (h2 == 0) out.pop_back();
fe8     return out;
483 }

```

## CircleLine.h

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

"Point.h" e0cfba, 10 lines

```

7dc template<class P>
195 vector<P> circleLine(P c, double r, P a, P b) {
33b     P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
55a     double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
3e4     if (h2 < 0) return {};
071     if (h2 == 0) return {p};
7cd     P h = ab.unit() * sqrt(h2);
d65     return {p - h, p + h};
59a }

```

## CirclePolygonIntersection.h

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

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

"../../../../content/geometry/Point.h" 19add1, 20 lines

```

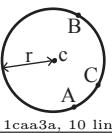
626 typedef Point<double> P;
361 #define arg(p, q) atan2(p.cross(q), p.dot(q))
bb9     double circlePoly(P c, double r, vector<P> ps) {
6d1     auto tri = [&](P p, P q) {
c9c         auto r2 = r * r / 2;
291         P d = q - p;
127         auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist
2();
eea         auto det = a * a - b;
691         if (det <= 0) return arg(p, q) * r2;
f43         auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det
));
aba         if (t < 0 || 1 <= s) return arg(p, q) * r2;
57f         P u = p + d * s, v = q + d * (t-1);
8c0         return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
a52     };
bef     auto sum = 0.0;
8f4     rep(i,0,sz(ps))
3b7         sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
e66     return sum;
f08 }

```

## circumcircle.h

**Description:**

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



"Point.h" 1caa3a, 10 lines

```

626 typedef Point<double> P;
510     double ccRadius(const P& A, const P& B, const P& C) {
14b         return (B-A).dist()*(C-B).dist()*(A-C).dist() /
f73             abs((B-A).cross(C-A))/2;
607 }
c0d     P ccCenter(const P& A, const P& B, const P& C) {
28a         P b = C-A, c = B-A;

```

```

680     return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
793 }

```

## MinimumEnclosingCircle.h

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

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

"circumcircle.h" 09dd0a, 18 lines

```

a28     pair<P, double> mec(vector<P> ps) {
4da         shuffle(all(ps), mt19937(time(0)));
f6a         P o = ps[0];
328         double r = 0, EPS = 1 + 1e-8;
2be         rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
5cc             o = ps[i], r = 0;
4da             rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
a30                 o = (ps[i] + ps[j]) / 2;
6f7                 r = (o - ps[i]).dist();
102                 rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
fa9                     o = ccCenter(ps[i], ps[j], ps[k]);
6f7                     r = (o - ps[i]).dist();
648                 }
7b0             }
dcf         }
645         return {o, r};
09d }

```

## 8.3 Polygons

### InsidePolygon.h

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

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

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

"Point.h", "OnSegment.h", "SegmentDistance.h" 2bf504, 12 lines

```

7dc template<class P>
0cc     bool inPolygon(vector<P> &p, P a, bool strict = true) {
8b7         int cnt = 0, n = sz(p);
fea         rep(i,0,n) {
444             P q = p[(i + 1) % n];
cbd             if (onSegment(p[i], q, a)) return !strict;
//or: if (segDist(p[i], q, a) <= eps) return !strict;
007             cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) >
0;
1b9         }
70a         return cnt;
c72     }

```

## PolygonArea.h

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

"Point.h" f12300, 7 lines

```

4fc     template<class T>
a51     T polygonArea2(vector<Point<T>> &v) {
2f8         T a = v.back().cross(v[0]);
06e         rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
3f5         return a;
693     }

```

## PolygonCenter.h

**Description:** Returns the center of mass for a polygon.

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

"Point.h" 9706dc, 10 lines

```

626     typedef Point<double> P;
6d9     P polygonCenter(const vector<P>& v) {
f9f     P res(0, 0); double A = 0;
70b     for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {

```

```

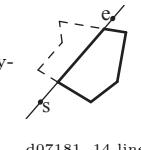
346         res = res + (v[i] + v[j]) * v[j].cross(v[i]);
3ea         A += v[j].cross(v[i]);
307     }
33c     return res / A / 3;
0d0 }

```

## PolygonCut.h

**Description:**

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



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

"Point.h" d07181, 14 lines

```

626     typedef Point<double> P;
37d     vector<P> polygonCut(const vector<P>& poly, P s, P e) {
fe2         vector<P> res;
d48         rep(i,0,sz(poly)) {
21c             P cur = poly[i], prev = i ? poly[i-1] : poly.back();
c5f             auto a = s.cross(e, cur), b = s.cross(e, prev);
2dc             if ((a < 0) != (b < 0))
380                 res.push_back(cur + (prev - cur) * (a / (a - b)));
c5c             if (a < 0)
a5f                 res.push_back(cur);
757         }
b50         return res;
42c }

```

## PolygonUnion.h

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

**Time:**  $\mathcal{O}(N^2)$ , where  $N$  is the total number of points

"Point.h", "sideOf.h" 3931c6, 34 lines

```

626     typedef Point<double> P;
142     double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y
; }
61d     double polyUnion(vector<vector<P>> &poly) {
499         double ret = 0;
9af         rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
9c8             P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
05c             vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
cbd             rep(j,0,sz(poly)) if (i != j) {
cc1                 rep(u,0,sz(poly[j])) {
418                     P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])
];
688                     int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
68b                     if (sc != sd) {
295                         double sa = C.cross(D, A), sb = C.cross(D, B);
e90                         if (min(sc, sd) < 0)
dac                             segs.emplace_back(sa / (sa - sb), sgn(sc - sd))
;
cf7                     } else if (!sc && !sd && j < i && sgn((B-A).dot(D-C))
>0) {
5b4                         segs.emplace_back(rat(C - A, B - A), 1);
e96                         segs.emplace_back(rat(D - A, B - A), -1);
313                     }
0d1                 }
fdc             }
861             sort(all(segs));
153             for (auto& s : segs) s.first = min(max(s.first, 0.0), 1
.0);
68c             double sum = 0;
723             int cnt = segs[0].second;
067             rep(j,1,sz(segs)) {
081                 if (!cnt) sum += segs[j].first - segs[j - 1].first;
6e9                 cnt += segs[j].second;
f58             }

```

```
320     ret += A.cross(B) * sum;
191 }
ad6 return ret / 2;
6e8 }
```

**ConvexHull.h****Description:**

Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull. If you want to keep the collinear points in the convex hull, change the comparison to  $h[t-2].cross(h[t-1], p) < 0$  and the size of the vector  $h$  to  $2 * sz(pts) + 1$ .

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

"Point.h"  

```
2c0 typedef Point<11> P;
f16 vector<P> convexHull(vector<P> pts) {
f78 if (sz(pts) <= 1) return pts;
3cb sort(all(pts));
abf vector<P> h(sz(pts)+1);
573 int s = 0, t = 0;
628 for (int it = 2; it--; s = --t, reverse(all(pts)))
4eb     for (P p : pts) {
3da         while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
f39         h[t++] = p;
bf0     }
036 return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
ec8 }
```



310954, 14 lines

**HullDiameter.h**

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

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

"Point.h"  

```
2c0 typedef Point<11> P;
d31 array<P, 2> hullDiameter(vector<P> S) {
e79 int n = sz(S), j = n < 2 ? 0 : 1;
354 pair<11, array<P, 2>> res({0, {S[0], S[0]}});
e4d rep(i, 0, j)
42e     for (; j = (j + 1) % n) {
ca1         res = max(res, {{S[i] - S[j]}.dist2(), {S[i], S[j]}})
;         if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >=
0)
c2b             break;
56c     }
3f2 return res.second;
5f7 }
```

**PointInsideHull.h**

**Description:** Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.

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

"Point.h", "sideOf.h", "OnSegment.h"  

```
71446b, 15 lines
2c0 typedef Point<11> P;

2d4 bool inHull(const vector<P>& l, P p, bool strict = true) {
d44     int a = 1, b = sz(l) - 1, r = !strict;
5cc     if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
6bc     if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
456     if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p) <=
-r)
d1f         return false;
48a     while (abs(a - b) > 1) {
4f7         int c = (a + b) / 2;
```

```
ac8     (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
b26 }
06f return sgn(l[a].cross(l[b], p)) < r;
c74 }
```

**LineHullIntersection.h**

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

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

"Point.h"  

```
7cf45b, 40 lines
530 #define cmp(i, j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
f84 #define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
e7e template <class P> int extrVertex(vector<P>& poly, P dir)
{
    int n = sz(poly), lo = 0, hi = n;
    if (extr(0)) return 0;
    while (lo + 1 < hi) {
        int m = (lo + hi) / 2;
        if (extr(m)) return m;
        int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
        f0c     if (ls < ms || (ls == ms && ls == cmp(lo, m))) ? hi : lo) =
m;
        68a }
        253     return lo;
    7f0 }

8e0 #define cmpL(i) sgn(a.cross(poly[i], b))
7dc template <class P>
ec4 array<int, 2> lineHull(P a, P b, vector<P>& poly) {
409     int endA = extrVertex(poly, (a - b).perp());
761     int endB = extrVertex(poly, (b - a).perp());
1a8     if (cmpL(endA) < 0 || cmpL(endB) > 0)
423         rep(i, 0, j)
234         int lo = endB, hi = endA, n = sz(poly);
c2d         while ((lo + 1) % n != hi) {
57e             int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
7f6             if (cmpL(m) == cmpL(endB)) lo = hi = m;
525         }
7dd         res[i] = (lo + !cmpL(hi)) % n;
356         swap(endA, endB);
c05     }
e00     if (res[0] == res[1]) return {res[0], -1};
3d1     if (!cmpL(res[0]) && !cmpL(res[1]))
959         switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
3f3             case 0: return {res[0], res[0]};
223             case 2: return {res[1], res[1]};
8fa         }
b50         return res;
36f }
```

**Minkowski.h**

**Description:** Computes the Minkowski sum of two convex polygons. Polygons must be convex and given in CCW order. Returns the vertices of the Minkowski sum polygon in CCW order.

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

"Point.h"  

```
664d67, 24 lines
780 using P = Point<11>;
```

```
89f vector<P> minkowski(vector<P> p, vector<P> q) {
a8e     auto fix = [](vector<P>& A) {
bec         int pos = 0;
2bb         for (int i = 1; i < sz(A); i++) {
609             if (A[i].y < A[pos].y || (A[i].y == A[pos].y && A[i].x < A[pos].x))
e4c                 pos = i;
f76             }
703             rotate(A.begin(), A.begin() + pos, A.end());
9e5             A.push_back(A[0]), A.push_back(A[1]);
236         };
889         fix(p), fix(q);
db6         vector<P> result;
692         int i = 0, j = 0;
98a         while (i < sz(p) - 2 || j < sz(q) - 2) {
942             result.push_back(p[i] + q[j]);
3bd             auto cross = (p[i + 1] - p[i]).cross(q[j + 1] - q[j]);
c3c             if (cross >= 0 && i < sz(p) - 2) i++;
f33             if (cross <= 0 && j < sz(q) - 2) j++;
801         }
dc8         return result;
2f9 }
```

**Extreme.h**

**Description:** Finds an extreme vertex of a convex polygon according to a unimodal comparator. The comparator defines a total order along the polygon (given in CCW order).

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

"Point.h"  

```
70b181, 26 lines
780 using P = Point<11>;
c88 int extreme(vector<P> &pol, const function<bool(P, P)> &cmp) {
b1c     int n = pol.size();
4a2     auto extr = [&](int i, bool& cur_dir) {
22a         cur_dir = cmp(pol[(i+1)%n], pol[i]);
61a         return !cur_dir and !cmp(pol[(i+n-1)%n], pol[i]);
364     };
63d     bool last_dir, cur_dir;
a0d     if (extr(0, last_dir)) return 0;
993     int l = 0, r = n;
ead     while (l+1 < r) {
ee4         int m = (l+r)/2;
f29         if (extr(m, cur_dir)) return m;
44a         bool rel_dir = cmp(pol[m], pol[l]);
b18         if (!last_dir and cur_dir or
261             (last_dir == cur_dir and rel_dir == cur_dir)) {
8a6             l = m;
1f1             last_dir = cur_dir;
94a             } else r = m;
606         }
792         return l;
985     }
cad     int max_dot(vector<P> &pol, P v) {
a98         return extreme([&](P p, P q) { return p.dot(v) > q.dot(v); });
27e }
```

**Tangents.h**

**Description:** Finds the left and right tangent points from an external point p to a convex polygon given in CCW order. A tangent point is a vertex where the segment p->v touches the polygon without intersecting its interior, defining the limits of visibility from p. Returns the indices of the left and right tangent vertices.

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

"Point.h", "Extreme.h"  

```
dfc85f, 11 lines
780 using P = Point<11>;
08d bool ccw(P p, P q, P r) {
```

```

274     return (q-p).cross(r-q) > 0;
0f3 }
826 pair<int, int> tangents(vector<P> &pol, P p) {
ae2     auto L = [&](P q, P r) { return ccw(p, r, q); };
98c     auto R = [&](P q, P r) { return ccw(p, q, r); };
861     return {extreme(pol, L), extreme(pol, R)};
3dc }

```

## 8.4 Misc. Point Set Problems

### ClosestPair.h

**Description:** Finds the closest pair of points.

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

`"Point.h"`

ac41a6, 18 lines

```

2c0 typedef Point<ll> P;
24b pair<P, P> closest(vector<P> v) {
7f9     assert(sz(v) > 1);
7f7     set<P> S;
879     sort(all(v), [](P a, P b) { return a.y < b.y; });
571     pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
eec     int j = 0;
813     for (P p : v) {
3fb         P d{1 + (ll)sqrt(ret.first), 0};
8be         while (v[j].y <= p.y - d.x) S.erase(v[j++]);
a5a         auto lo = S.lower_bound(p - d), hi = S.upper_bound(p +
d);
c77         for (; lo != hi; ++lo)
113             ret = min(ret, {*(lo - p).dist2(), {*lo, p}});
8aa         S.insert(p);
5b0     }
70d     return ret.second;
bf2 }

```

### ManhattanMST.h

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

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

`"Point.h"`

df6f59, 24 lines

```

bde typedef Point<int> P;
ea9 vector<array<int, 3>> manhattanMST(vector<P> ps) {
850     vi id{sz(ps)};
27c     iota(all(id), 0);
8c1     vector<array<int, 3>> edges;
8de     rep(k, 0, 4) {
1dd         sort(all(id), [&](int i, int j) {
02b             return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y});
702         map<int, int> sweep;
1e2         for (int i : id) {
84d             for (auto it = sweep.lower_bound(-ps[i].y);
904                 it != sweep.end(); sweep.erase(it++)) {
61d                 int j = it->second;
6f3                 P d = ps[i] - ps[j];
d18                 if (d.y > d.x) break;
537                 edges.push_back({d.y + d.x, i, j});
271             }
923             sweep[-ps[i].y] = i;
e69         }
4eb         for (P & p : ps) if (k & 1) p.x = -p.x; else swap(p.x, p
.y);
a11     }
da2     return edges;
a11 }

```

### kdTree.h

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

`"Point.h"`

## ClosestPair ManhattanMST kdTree FastDelaunay

```

9a6     typedef long long T;
293     typedef Point<T> P;
305     const T INF = numeric_limits<T>::max();

173     bool on_x(const P & a, const P & b) { return a.x < b.x; }
0bd     bool on_y(const P & a, const P & b) { return a.y < b.y; }

bf2     struct Node {
975         P pt; // if this is a leaf, the single point in it
877         T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
a23         Node *first = 0, *second = 0;

86a         T distance(const P & p) { // min squared distance to a
point
28b             T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
88e             T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
d98             return (P(x,y) - p).dist2();
ca4         }

d97         Node(vector<P>&& vp) : pt(vp[0]) {
741             for (P p : vp) {
ad3                 x0 = min(x0, p.x); x1 = max(x1, p.x);
e5d                 y0 = min(y0, p.y); y1 = max(y1, p.y);
310             }
994             if (vp.size() > 1) {
// split on x if width >= height (not ideal...)
9b7                 sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
// divide by taking half the array for each child
not
// best performance with many duplicates in the
middle)
0f9                 int half = sz(vp)/2;
48e                 first = new Node({vp.begin(), vp.begin() + half});
902                 second = new Node({vp.begin() + half, vp.end()});
66e             }
204         }
a77     };

dad     struct KDTree {
95f         Node* root;
c30         KDTree(const vector<P>& vp) : root(new Node(all(vp))) {
}

113         pair<T, P> search(Node *node, const P & p) {
ec4             if (!node->first) {
// uncomment if we should not find the point itself:
// if (p == node->pt) return {INF, P()};
47e                 return make_pair((p - node->pt).dist2(), node->pt);
119             }

ea4             Node *f = node->first, *s = node->second;
d40             T bfirst = f->distance(p), bsec = s->distance(p);
a16             if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

// search closest side first, other side if needed
86c             auto best = search(f, p);
314             if (bsec < best.first)
509                 best = min(best, search(s, p));
f26             return best;
74c         }

// find nearest point to a point, and its squared
distance
// (requires an arbitrary operator< for Point)
9b6             pair<T, P> nearest(const P & p) {
195                 return search(root, p);
94c             }
6f5         };

```

## FastDelaunay.h

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

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

`"Point.h"`

```

2c0     typedef Point<ll> P;
806     typedef struct Quad* Q;
449     typedef __int128_t ll1; // (can be ll if coords are < 2e4)
59b     P arb(LLONG_MAX, LLONG_MAX); // not equal to any other
point

070     struct Quad {
461         Q rot, o; P p = arb; bool mark;
b38         P & F() { return r()>p; }
23a         Q & r() { return rot->rot; }
f4f         Q prev() { return rot->o->rot; }
57e         Q next() { return r()>prev(); }
180     } *H;

d15     bool circ(P p, P a, P b, P c) { // is p in the
circumcircle?
4b4         ll1 p2 = p.dist2(), A = a.dist2() - p2,
ffa         B = b.dist2() - p2, C = c.dist2() - p2;
59a         return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B >
0;
6af     }

00a     Q makeEdge(P orig, P dest) {
bdf     Q r = H ? H : new Quad{new Quad{new Quad{new Quad{}}}};
516     H = r->o; r->r()->r() = r;
2c3     rep(i, 0, 4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r-
r();
ed2     r->p = orig; r->F() = dest;
4c1     return r;
b3b     }

d8d     void splice(Q a, Q b) {
686         swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
86c     }

e92     Q connect(Q a, Q b) {
fc2     Q q = makeEdge(a->F(), b->p);
6e6         splice(q, a->next());
642         splice(q->r(), b);
bef         return q;
4a4     }

196     pair<Q, Q> rec(const vector<P>& s) {
e63         if (sz(s) <= 3) {
4a0             Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
if (sz(s) == 2) return {a, a->r()};
19e             splice(a->r(), b);
5f8             auto side = s[0].cross(s[1], s[2]);
b9f             Q c = side ? connect(b, a) : 0;
3d8             return {side < 0 ? c->r() : a, side < 0 ? c : b->r()};
c9e     }

5ef     #define H(e) e->F(), e->p
c98     #define valid(e) (e->F().cross(H(base)) > 0)
a3e     Q A, B, ra, rb;
f5e     int half = sz(s) / 2;
391     tie(ra, A) = rec(all(s) - half);
d9b     tie(B, rb) = rec({sz(s) - half + all(s)});
f80     while ((B->p).cross(H(A)) < 0 && (A = A->next()) ||
b08         (A->p).cross(H(B)) > 0 && (B = B->r()->o)));
76d     Q base = connect(B->r(), A);
87f     if (A->p == ra->p) ra = base->r();
b58     if (B->p == rb->p) rb = base;

```

```

3e6 #define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
f02     while (circ(e->dir->F(), H(base), e->F())) { \
936         Q t = e->dir; \
6d3         splice(e, e->prev()); \
16e         splice(e->r(), e->r()->prev()); \
d47         e->o = H; H = e; e = t; \
a2e     }
1de for (;;) {
eaa     DEL(LC, base->r(), o); DEL(RC, base, prev());
6fa     if (!valid(LC) && !valid(RC)) break;
e09     if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC)))) \
b74         base = connect(RC, base->r());
295     else
271         base = connect(base->r(), LC->r());
fcf }
345     return { ra, rb };
7cf }

dal vector<P> triangulate(vector<P> pts) {
af6     sort(all(pts)); assert(unique(all(pts)) == pts.end());
e00     if (sz(pts) < 2) return {};
235     Q e = rec(pts).first;
50c     vector<Q> q = {e};
6c1     int qi = 0;
7a5     while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
806     #define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
43e         c.push_back(c->r()); c = c->next(); } while (c != e); } \
9d6     ADD; pts.clear();
b58     while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
a42     return pts;
a02 }

```

## 8.5 3D

### PolyhedronVolume.h

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

3058c3, 7 lines

```

f9c template<class V, class L>
cb3 double signedPolyVolume(const V& p, const L& trilist) {
9e8     double v = 0;
b72     for (auto i : trilist) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
fb8     return v / 6;
fca }

```

### Point3D.h

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

8058ae, 33 lines

```

f10 template<class T> struct Point3D {
f07     typedef Point3D P;
d0e     typedef const P& R;
329     T x, y, z;
cf2     explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
803     bool operator<(R p) const {
8ee         return tie(x, y, z) < tie(p.x, p.y, p.z); }
236     bool operator==(R p) const {
bd6         return tie(x, y, z) == tie(p.x, p.y, p.z); }
9ae     P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
54a     P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
743     P operator*(T d) const { return P(x*d, y*d, z*d); }
17b     P operator/(T d) const { return P(x/d, y/d, z/d); }
e49     T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
270     P cross(R p) const {
923         return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
a77     }

```

```

b70     T dist2() const { return x*x + y*y + z*z; }
18b     double dist() const { return sqrt((double)dist2()); }
//Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
3d6     double phi() const { return atan2(y, x); }
//Zenith angle (latitude) to the z-axis in interval [0, pi]
0fa     double theta() const { return atan2(sqrt(x*x+y*y), z); }
55e     P unit() const { return *this/(T)dist(); } //makes dist()
=1
//returns unit vector normal to *this and p
685     P normal(P p) const { return cross(p).unit(); }
//returns point rotated 'angle' radians ccw around axis
c67     P rotate(double angle, P axis) const {
7cd         double s = sin(angle), c = cos(angle); P u = axis.unit();
() ;
6b7         return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
73a     }
805     };

```

### 3dHull.h

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

**Time:**  $O(n^2)$

"Point3D.h" 5b45fc, 50 lines

```

b8e     typedef Point3D<double> P3;
9ce     struct PR {
1fc         void ins(int x) { (a == -1 ? a : b) = x; }
82f         void rem(int x) { (a == x ? a : b) = -1; }
2ad         int cnt() { return (a != -1) + (b != -1); }
ba2         int a, b;
cf7     };
5e4     struct F { P3 q; int a, b, c; };
b6d     vector<F> hull3d(const vector<P3>& A) {
cd9     assert(sz(A) >= 4);
ec1     vector<vector<PR>> E(sz(A)), vector<PR>(sz(A), {-1, -1});
#define E(x,y) E[f.x][f.y]
afe     vector<F> FS;
9e0     auto inf = [&](int i, int j, int k, int l) {
2ce         P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
fai         if (q.dot(A[l]) > q.dot(A[i])) {
eaa             q = q * -1;
f22             F f{q, i, j, k};
ee5             E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
471             FS.push_back(f);
d73         };
30c         rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
047             mf(i, j, k, 6 - i - j - k);
            };
3ef         rep(i,4,sz(A)) {
3b5             rep(j,0,sz(FS)) {
068                 F f = FS[j];
04f                 if (f.q.dot(A[i]) > f.q.dot(A[f.a])) {
412                     E(a,b).rem(f.c);
b61                     E(a,c).rem(f.b);
e5c                     E(b,c).rem(f.a);
8d5                     swap(FS[j--], FS.back());
eef                     FS.pop_back();
5cd                 }
220             };
97f             int nw = sz(FS);
c63             rep(j,0,nw) {
068                 F f = FS[j];
561                 #define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
}

```

```

3da         C(a, b, c); C(a, c, b); C(b, c, a);
248     }
472     }
864     for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
770         A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
311     return FS;
be2     };

```

### sphericalDistance.h

**Description:** Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude)  $f_1(\phi_1)$  and  $f_2(\phi_2)$  from x axis and zenith angles (latitude)  $t_1(\theta_1)$  and  $t_2(\theta_2)$  from z axis ( $0 =$  north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so that is what you have you can use only the two last rows.  $dx*radius$  is then the difference between the two points in the x direction and  $d*radius$  is the total distance between the points.

611f07, 9 lines

```

c5f     double sphericalDistance(double f1, double t1,
3e8         double f2, double t2, double radius) {
284         double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
277         double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
c7e         double dz = cos(t2) - cos(t1);
c09         double d = sqrt(dx*dx + dy*dy + dz*dz);
154         return radius*2*asin(d/2);
4fa     };

```

## Strings (9)

### AhoCorasick.h

95b3e7, 46 lines

```

c2e     int trie[ms][sigma], fail[ms], terminal[ms], superfail[ms];
1e1     bool present[ms];
965     int z = 1;
ca3     int val(char c) { return c - 'a'; }
f97     void add(string& p) {
b3d         int cur = 0;
b4b         for (int i = 0; i < (int)p.size(); i++) {
9e4             int& nxt = trie[cur][val(p[i])];
b6e             if (nxt == 0) nxt = z++;
1bc             cur = nxt;
a92         }
c0e         present[cur] = true;
b07         terminal[cur]++;
6aa     }
0a8     void build() {
26a         queue<int> q;
f47         for (q.push(0); !q.empty(); q.pop()) {
fb5             int on = q.front();
0b2             for (int i = 0; i < sigma; i++) {
df1                 int& to = trie[on][i];
279                 int f = (on == 0 ? 0 : trie[fail[on]][i]);
de7                 int sf = (present[f] ? f : superfail[f]);
24d                 if (!to) {
c4e                     to = f;
6fd                 }
4e6             }
3ef             else {
b86                 fail[to] = f;
superfail[to] = sf;
// merge infos (ex: terminal[to] += terminal[f])
91b                 q.push(to);
692             }
bff     };

```

```
e61     }
b89 }

54e void search(string& s) {
b3d     int cur = 0;
b4f     for (char c : s) {
3ba         cur = trie[cur][val(c)];
        // process infos on current node (ex: occurrences
           += terminal[cur])
5ac     }
d1b }
```

## Hash.h

**Description:** C can also be random, operator is [l, r]

79e7f5, 28 lines

```
541 using ull = uint64_t;
542 struct H {
858     ull x; H(ull x = 0) : x(x) {}
c9b     H operator+(H o) { return x + o.x + (x + o.x < x); }
5cd     H operator-(H o) { return *this + ~o.x; }
167     H operator*(H o) {
2f3         auto m = (_uint128_t)x * o.x;
540         return H((ull)m) + (ull)(m >> 64);
681     }
bf2     ull get() const { return x + !~x; }
03c     bool operator==(H o) const{ return get() == o.get(); }
0ab     bool operator<(H o) const{ return get() < o.get(); }
bf6 };
862 static const H C = (11)1e11 + 3;
61c struct Hash {
2f2     vector<H> h, pw;
1df     Hash(string& str) : h(str.size()), pw(str.size()) {
9bc         pw[0] = 1, h[0] = str[0];
1c5         for (int i = 1; i < str.size(); i++) {
90a             h[i] = h[i - 1] * C + str[i];
b3c             pw[i] = pw[i - 1] * C;
57e         }
f1b     }
75e     H operator()(int l, int r) {
91f         return h[r] - (l ? h[l - 1] * pw[r - l + 1] : 0);
9cf     }
c36 };
```

## KMP.h

**Description:** pi[x] computes the length of the longest prefix of s that ends at x, other than s[0..x] itself (abacaba -> 0010123).

c7ef15, 10 lines

```
a56 vector<int> pi(const string& s) {
627     vector<int> p(sz(s));
edb     for(int i = 1; i < sz(s); i++) {
052         int g = p[i-1];
6c0         while (g && s[i] != s[g]) g = p[g-1];
7cf         p[i] = g + (s[i] == s[g]);
a2e     }
74e     return p;
c7c }
```

## KmpAutomaton.h

**Description:** go[i][j] = length of the longest prefix of s that is a suffix of s[0..i] followed by the letter j (i.e., the next matched prefix length if, at state i, we read letter j).

8833cb, 17 lines

```
ab6     int go[ms][sigma];
ca3     int val(char c) { return c - 'a'; }
8cf     void automaton(string& s) {
3cc         for (int i = 0; i < sigma; i++)
48d             go[0][i] = (i == val(s[0]));
8cc         for (int i = 1, bdr = 0; i <= (int)s.size(); i++) {
```

```
782         for (int j = 0; j < sigma; j++) {
6ef             go[i][j] = go[bdr][j];
87c         }
f8d         if (i < (int)s.size()) {
02f             go[i][val(s[i])] = i + 1;
364             bdr = go[bdr][val(s[i])];
63b         }
d7e     }
0c5 }
```

## Manacher.h

**Description:** p[0][i+1] is the length of matches of even length palindrome,

starting from [i, i+1].

p[1][i] is the length of matches of odd length palindrome, starting from [i, i].

(abaxx -&gt; p[0] = 00001)

(abaxx -&gt; p[1] = 01000)

e7ad79, 14 lines

```
fcl     array<vi, 2> manacher(const string& s) {
f89     int n = sz(s);
f77     array<vi,2> p = {vi(n+1), vi(n)};
c9a     rep(z,0,2) for (int i=0,l=0,r=0; i<n; i++) {
24e         int t = r-i+1;
e70         if (i<r) p[z][i] = min(t, p[z][l+t]);
fff         int L = i-p[z][i], R = i+p[z][i]-!z;
649         while (L>=1 && R+1<n && s[L-1] == s[R+1])
895             p[z][i]++, L--, R++;
f28         if (R>r) l=L, r=R;
a84     }
74e     return p;
e7a }
```

## MinRotation.h

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

d07a42, 10 lines

```
5fa     int minRotation(string s) {
c6c     int a=0, N=sz(s), s += s;
840     rep(b,0,N) rep(k,0,N) {
32f         if (a+k == b || s[a+k] < s[b+k]) {
873             b += max(0, k-1); break;
068             if (s[a+k] > s[b+k]) { a = b; break; }
937         }
3f5     return a;
d07 }
```

## SuffixArray.h

**Description:** lcp[i] is the length of the longest common prefix between the suffixes s[sa[i]..n-1] and s[sa[i-1]..n-1].

If we concatenate multiple strings using separator characters, the separator that appears furthest to the right must be the smallest character in the alphabet.

048424, 31 lines

```
3f4     struct SuffixArray {
716     vector<int> sa, lcp;
d91     SuffixArray(string s, int lim=256) {
59b         s.push_back('$');
323         int n = sz(s), k = 0, a, b;
9f1         vector<int> x(all(s)), y(n), ws(max(n, lim));
a4f         sa = lcp = y, iota(all(sa), 0);
25d         for(int j = 0, p = 0; p < n; j = max(1, j*2), lim = p) {
3cd             p = j, iota(all(y), n - j);
603             for(int i=0; i<n; i++){
071                 if (sa[i] >= j) y[p++] = sa[i] - j;
cb4             }
911             fill(all(ws), 0);
483             for(int i=0; i<n; i++) ws[x[i]]++;
5d9             for(int i=1; i<lim; i++) ws[i] += ws[i - 1];
a9e             for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
}
```

```
c7d     swap(x, y), p = 1, x[sa[0]] = 0;
6f5     for(int i=1; i<n; i++){
93f         a = sa[i - 1], b = sa[i];
ddb         x[b] = p-1;
a32         if(y[a] != y[b] || y[a+j] != y[b+j]) x[b] = p++;
1ba     }
c36     }
65b     for (int i = 0, j; i < n - 1; lcp[x[i++]] = k)
904         for (k && k--, j = sa[x[i - 1]];
262             s[i + k] == s[j + k]; k++);
68a     sa = vector<int>(sa.begin() + 1, sa.end());
5d4     lcp = vector<int>(lcp.begin() + 1, lcp.end());
4db     }
048 }
```

```
c7d     swap(x, y), p = 1, x[sa[0]] = 0;
6f5     for(int i=1; i<n; i++){
93f         a = sa[i - 1], b = sa[i];
ddb         x[b] = p-1;
a32         if(y[a] != y[b] || y[a+j] != y[b+j]) x[b] = p++;
1ba     }
c36     }
65b     for (int i = 0, j; i < n - 1; lcp[x[i++]] = k)
904         for (k && k--, j = sa[x[i - 1]];
262             s[i + k] == s[j + k]; k++);
68a     sa = vector<int>(sa.begin() + 1, sa.end());
5d4     lcp = vector<int>(lcp.begin() + 1, lcp.end());
4db     }
048 }
```

## Zfunc.h

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

495392, 13 lines

```
572     vector<int> ZFunc(const string& s) {
d6b         int n = sz(s), a = 0, b = 0;
2b1         vector<int> z(n, 0);
29a         if (!z.empty()) z[0] = 0;
6f5         for (int i = 1; i < n; i++) {
fe0             int end = i;
98f             if (i < b) end = min(i + z[i - a], b);
65f             while (end < n && s[end] == s[end - i]) ++end;
816             z[i] = end - i; if (end > b) a = i, b = end;
253         }
070     return z;
495 }
```

## Various (10)

## 10.1 Misc. algorithms

## Dates.h

**Description:** dateToInt converts Gregorian date to integer (Julian day number). intToDate converts integer (Julian day number) to Gregorian date: month/day/year. intToDay converts Julian day number to day of the week

688e5b, 23 lines

```
37c     string day[] = { "Mon", "Tue", "Wed", "Thu", "Fri", "Sat",
                      "Sun" };
fb9     int dateToInt(int m, int d, int y) {
e70         return
773         1461 * (y + 4800 + (m - 14) / 12) / 4 +
649         367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
fa0         3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
3aa         d - 32075;
a73     }
3fe     void intToDate(int jd, int& m, int& d, int& y) {
ee1         int x, n, i, j;
33a         x = jd + 68569;
403         n = 4 * x / 146097;
33e         x -= (146097 * n + 3) / 4;
6fc         i = (4000 * (x + 1)) / 1461001;
b1d         x -= 1461 * i / 4 - 31;
fc9         j = 80 * x / 2447;
c8d         d = x - 2447 * j / 80;
179         x = j / 11;
335         m = j + 2 - 12 * x;
23d         y = 100 * (n - 49) + i + x;
ccb     }
04e     string intToDay(int jd) { return day[jd % 7]; }
```

## MultisetHash.h

5648da, 8 lines

cdc ull hashify(ull sum) {

```
7b8     sum += FIXED_RANDOM;
6ec     sum += 0x9e3779b97f4a7c15;
dc6     sum = (sum ^ (sum >> 30)) * 0xbff58476d1ce4e5b9;
005     sum = (sum ^ (sum >> 27)) * 0x94d049bb133111eb;
358     return sum ^ (sum >> 31);
564 }
```

## Rand.h

2de3f8, 8 lines

```
c8a mt19937 rng(chrono::steady_clock::now().time_since_epoch()
    .count());
// -64

463 int uniform(int l, int r) { // [l, r]
a7f     uniform_int_distribution<int> uid(l, r);
f54     return uid(rng);
d9e }
```

## IntervalContainer.h

**Description:** Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).

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

edce47, 24 lines

```
d91 set<pii>::iterator addInterval(set<pii>& is, int L, int R)
{
    if (L == R) return is.end();
d4c    auto it = is.lower_bound({L, R}), before = it;
dc6    while (it != is.end() && it->first <= R) {
164        R = max(R, it->second);
1a5        before = it = is.erase(it);
fe9    }
1af    if (it != is.begin() && (--it)->second >= L) {
3ca        L = min(L, it->first);
164        R = max(R, it->second);
861        is.erase(it);
0de    }
aa0    return is.insert(before, {L,R});
d57 }

675 void removeInterval(set<pii>& is, int L, int R) {
17b    if (L == R) return;
bef    auto it = addInterval(is, L, R);
e14    auto r2 = it->second;
655    if (it->first == L) is.erase(it);
016    else (int&)it->second = L;
ee9    if (R != r2) is.emplace(R, r2);
059 }
```

## IntervalCover.h

**Description:** Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add || R.empty(). Returns empty set on failure (or if G is empty).

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

9e9d8d, 20 lines

```
4fc template<class T>
dbe vi cover(pair<T, T> G, vector<pair<T, T>> I) {
3d5    vi S(sz(I)), R;
d00    iota(all(S), 0);
591    sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
d10    T cur = G.first;
05e    int at = 0;
336    while (cur < G.second) { // (A)
438        pair<T, int> mx = make_pair(cur, -1);
f07        while (at < sz(I) && I[S[at]].first <= cur) {
032            mx = max(mx, make_pair(I[S[at]].second, S[at]));
69a            at++;
c42        }
```

```
c54     if (mx.second == -1) return {};
953     cur = mx.first;
fbf     R.push_back(mx.second);
dd1    }
b1a    return R;
b8d }
```

## TernarySearch.h

**Description:** Find the smallest  $i$  in  $[a, b]$  that maximizes  $f(i)$ , assuming that  $f(a) < \dots < f(i) \geq \dots \geq f(b)$ . To reverse which of the sides allows non-strict inequalities, change the  $<$  marked with (A) to  $\leq$ , and reverse the loop at (B). To minimize  $f$ , change it to  $>$ , also at (B).

**Usage:**  $\text{int } \text{ind} = \text{ternSearch}(0, n-1);$

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

a995fb, 11 lines

```
53a int ternSearch(int a, int b) {
25b     assert(a <= b);
329     while (b - a >= 5) {
924         int mid = (a + b) / 2;
c9e         if (f(mid) < f(mid+1)) a = mid; // (A)
ceb         else b = mid+1;
ce7     }
95e     rep(i, a+1, b+1) if (f(a) < f(i)) a = i; // (B)
355     return a;
a99 }
```

## 10.2 Dynamic programming

## KnuthDP.h

**Description:** When doing DP on intervals:  $dp[i][j] = \min_{i < k < j} (dp[i][k] + dp[k][j]) + f(i, j)$ , where the (minimal) optimal  $k$  increases with both  $i$  and  $j$ . 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  $a \leq b \leq c \leq d$ . Another sufficient criteria is:  $opt[i][j-1] \leq opt[i][j] \leq opt[i+1][j]$

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

fea016, 22 lines

```
7cc 11 knuth(){
6a7      memset(opt, -1, sizeof opt);
45b      for(int i=n-1; i>=0; i--) {
8e7          dp[i][i] = 0; // base case
b28          opt[i][i] = i;
94f          for(int j=i+1; j<n; j++) {
2e2              int optL = (!j ? 0 : opt[i][j-1]);
dc4              int optR = (^opt[i+1][j] ? opt[i+1][j] : n-1);
554              ll cst = cost(i, j);
f12              dp[i][j] = INF;
3bb              optL = max(i, optL), optR = min(j-1, optR);
349              for(int k=optL; k<=optR; k++) {
f8b                  ll now = dp[i][k] + dp[k+1][j] + cst;
e83                  if(now <= dp[i][j]){
960                      dp[i][j] = now;
14d                      opt[i][j] = k;
5fc                  }
114             }
4ce         }
96c     }
fea }
```

## DivideAndConquerDP.h

**Description:** Divide and Conquer DP maintaining cost, can be used when  $opt[i][j] \leq opt[i][j+1]$ . In this code everything is 1-based. Memory can be optimized by keeping only the last row

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

c7cb38, 42 lines

```
129 void add(int idx) {}
404 void rem(int idx) {}

749 void deC(int i, int l, int r, int optL, int optR) {
de6     if (l > r) return;
995     int j = (l + r) / 2;
```

```
d9a     for (int k = r; k > j; k--) rem(k);
c45     int opt = optL;
364     for (int k = optL; k <= min(optR, j); k++) {
597         // cost = cost[k, j]
532         int val = dp[i - 1][k - 1] + cost;
482         if (val < dp[i][j]) {
513             dp[i][j] = val;
446             opt = k;
178         }
183     }
93f     rem(k);
5d9     for (int k = min(optR, j); k >= optL; k--) add(k);
rem(j);
deC(i, l, j - 1, optL, opt);

ebd     for (int k = j; k <= r; k++) add(k);
0b6     for (int k = optL; k < opt; k++) rem(k);
deC(i, j + 1, r, opt, optR);

9bb     for (int k = optL; k < opt; k++) add(k);
460 }

d57 int solve(int N, int M) { // 1-based
d9f     for (int i = 0; i <= M; i++) {
138         for (int j = 0; j <= N; j++) {
3db             dp[i][j] = inf; // base case
a26         }
e0f     }
c21     cost = 0; // neutral value
c62     for (int i = 1; i <= N; i++) add(i);
143     for (int i = 1; i <= M; i++) {
156         deC(i, 1, N, 1, N);
c97     }
01a     return dp[M][N];
3ab }
```