MIT

Copypaste

Anton Trygub, Mingyang Deng, Ziqian Zhong

For the 46th ACM-ICPC World Finals

2024-04-09

MIT Copypaste Team Notebook

Anton Trygub, Mingyang Deng, Ziqian Zhong 2024-04-09

Contents

Contest (1)

template.cpp

14 lines

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

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

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

.bashrc

2 lines

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

hash.sh

3 lines

```
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]'| dos2unix |
    md5sum |cut -c-6
```

Mathematics (2)

2.1 Equations

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

Generally for $Ax=b$,

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

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

2.2 Recurrences

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

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

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

2.3 Trigonometry

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

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

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

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

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

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

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

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

2.4 Geometry

2.4.1 Triangles

Side lengths: a,b,c

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

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

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

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

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

Length of bisector (divides angles in two):

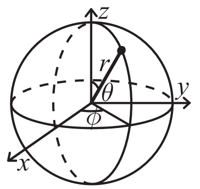
$$s_a=\sqrt{bc\left[1-\left(\frac{a}{b+c}\right)^2\right]}$$

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

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

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

2.4.2 Spherical coordinates



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

2.4.3 Quadrilaterals

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

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

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

2.5 Derivatives/Integrals

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

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

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

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

Integration by parts:

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

$$\int x^n dx = \frac{1}{n+1}x^{n+1} \tag{2.1}$$

$$\int \frac{1}{x}dx = \ln|x| \tag{2.2}$$

$$\int udv = uv - \int vdu \tag{2.3}$$

$$\int \frac{1}{ax+b}dx = \frac{1}{a}\ln|ax+b| \tag{2.4}$$

$$\int \frac{1}{(x+a)^2}dx = -\frac{1}{x+a} \tag{2.5}$$

$$\int (x+a)^n dx = \frac{(x+a)^{n+1}}{n+1}, n \neq -1 \tag{2.6}$$

$$\int x(x+a)^n dx = \frac{(x+a)^{n+1}((n+1)x-a)}{(n+1)(n+2)} \tag{2.7}$$

$$\int \frac{1}{1+x^2}dx = \tan^{-1}x \tag{2.8}$$

$$\int \frac{1}{a^2+x^2}dx = \frac{1}{a}\tan^{-1}\frac{x}{a} \tag{2.9}$$

$$\int \frac{x}{a^2+x^2}dx = \frac{1}{2}\ln|a^2+x^2| \tag{2.10}$$

$$\int \frac{x^2}{a^2+x^2}dx = x - a\tan^{-1}\frac{x}{a} \tag{2.11}$$

$$\int \frac{x^3}{a^2+x^2}dx = \frac{1}{2}x^2 - \frac{1}{2}a^2\ln|a^2+x^2| \tag{2.12}$$

$$\int \frac{1}{ax^2+bx+c}dx = \frac{2}{\sqrt{4ac-b^2}}\tan^{-1}\frac{2ax+b}{\sqrt{4ac-b^2}} \tag{2.13}$$

$$\int \frac{1}{(x+a)(x+b)}dx = \frac{1}{b-a}\ln\frac{a+x}{b+x},\ a \neq b \tag{2.14}$$

$$\int \frac{x}{(x+a)^2}dx = \frac{a}{a+x} + \ln|a+x| \tag{2.15}$$

$$\begin{aligned}\int \frac{x}{ax^2+bx+c}dx &= \frac{1}{2a}\ln|ax^2+bx+c| \\ &\quad - \frac{b}{a\sqrt{4ac-b^2}}\tan^{-1}\frac{2ax+b}{\sqrt{4ac-b^2}}\end{aligned} \tag{2.16}$$

$$\int \sqrt{x-a}dx = \frac{2}{3}(x-a)^{3/2} \tag{2.17}$$

$$\int \frac{1}{\sqrt{x\pm a}}dx = 2\sqrt{x\pm a} \tag{2.18}$$

$$\int \frac{1}{\sqrt{a-x}}dx = -2\sqrt{a-x} \tag{2.19}$$

$$\int x\sqrt{x-a}dx = \frac{2}{3}a(x-a)^{3/2} + \frac{2}{5}(x-a)^{5/2} \tag{2.20}$$

$$\int \sqrt{ax+b}dx = \left(\frac{2b}{3a} + \frac{2x}{3}\right)\sqrt{ax+b} \tag{2.21}$$

$$\int (ax+b)^{3/2}dx = \frac{2}{5a}(ax+b)^{5/2} \tag{2.22}$$

$$\int \frac{x}{\sqrt{x\pm a}}dx = \frac{2}{3}(x\mp 2a)\sqrt{x\pm a} \tag{2.23}$$

$$\int \sqrt{\frac{x}{a-x}}dx = -\sqrt{x(a-x)} - a\tan^{-1}\frac{\sqrt{x(a-x)}}{x-a} \tag{2.24}$$

$$\int \sqrt{\frac{x}{a+x}}dx = \sqrt{x(a+x)} - a\ln\left[\sqrt{x}+\sqrt{x+a}\right] \tag{2.25}$$

$$\int x\sqrt{ax+b}dx = \frac{2}{15a^2}(-2b^2+abx+3a^2x^2)\sqrt{ax+b} \tag{2.26}$$

$$\begin{aligned}\int \sqrt{x(ax+b)}dx &= \frac{1}{4a^{3/2}}\Big[(2ax+b)\sqrt{ax(ax+b)} \\ &\quad -b^2\ln\left|a\sqrt{x}+\sqrt{a(ax+b)}\right|\Big]\end{aligned} \tag{2.27}$$

$$\begin{aligned}\int \sqrt{x^3(ax+b)}dx &= \left[\frac{b}{12a} - \frac{b^2}{8a^2x} + \frac{x}{3}\right]\sqrt{x^3(ax+b)} \\ &\quad + \frac{b^3}{8a^{5/2}}\ln\left|a\sqrt{x}+\sqrt{a(ax+b)}\right|\end{aligned} \tag{2.28}$$

$$\int \sqrt{x^2\pm a^2}dx = \frac{1}{2}x\sqrt{x^2\pm a^2} \pm \frac{1}{2}a^2\ln\left|x+\sqrt{x^2\pm a^2}\right| \tag{2.29}$$

$$\int \sqrt{a^2-x^2}dx = \frac{1}{2}x\sqrt{a^2-x^2} + \frac{1}{2}a^2\tan^{-1}\frac{x}{\sqrt{a^2-x^2}} \tag{2.30}$$

$$\int x\sqrt{x^2\pm a^2}dx = \frac{1}{3}\left(x^2\pm a^2\right)^{3/2} \tag{2.31}$$

$$\int \frac{1}{\sqrt{x^2\pm a^2}}dx = \ln\left|x+\sqrt{x^2\pm a^2}\right| \tag{2.32}$$

$$\int \frac{1}{\sqrt{a^2-x^2}}dx = \sin^{-1}\frac{x}{a} \tag{2.33}$$

$$\int \frac{x}{\sqrt{x^2\pm a^2}}dx = \sqrt{x^2\pm a^2} \tag{2.34}$$

$$\int \frac{x}{\sqrt{a^2-x^2}}dx = -\sqrt{a^2-x^2} \tag{2.35}$$

$$\int \frac{x^2}{\sqrt{x^2\pm a^2}}dx = \frac{1}{2}x\sqrt{x^2\pm a^2} \mp \frac{1}{2}a^2\ln\left|x+\sqrt{x^2\pm a^2}\right| \tag{2.36}$$

$$\begin{aligned}\int \sqrt{ax^2+bx+c}dx &= \frac{b+2ax}{4a}\sqrt{ax^2+bx+c} \\ &\quad + \frac{4ac-b^2}{8a^{3/2}}\ln\left|2ax+b+2\sqrt{a(ax^2+bx^{+}c)}\right|\end{aligned} \tag{2.37}$$

$$\begin{aligned}\int x\sqrt{ax^2+bx+c} &= \frac{1}{48a^{5/2}}\Big(2\sqrt{a}\sqrt{ax^2+bx+c} \\ &\quad \times\left(-3b^2+2abx+8a(c+ax^2)\right) \\ &\quad +3(b^3-4abc)\ln\left|b+2ax+2\sqrt{a}\sqrt{ax^2+bx+c}\right|\Big)\end{aligned} \tag{2.38}$$

$$\int \frac{1}{\sqrt{ax^2+bx+c}}dx = \frac{1}{\sqrt{a}}\ln\left|2ax+b+2\sqrt{a(ax^2+bx+c)}\right| \tag{2.39}$$

$$\begin{aligned}\int \frac{x}{\sqrt{ax^2+bx+c}}dx &= \frac{1}{a}\sqrt{ax^2+bx+c} \\ &\quad - \frac{b}{2a^{3/2}}\ln\left|2ax+b+2\sqrt{a(ax^2+bx+c)}\right|\end{aligned} \tag{2.40}$$

$$\int \frac{dx}{(a^2+x^2)^{3/2}} = \frac{x}{a^2\sqrt{a^2+x^2}} \tag{2.41}$$

$$\int \ln ax dx = x\ln ax - x \tag{2.42}$$

$$\int \frac{\ln ax}{x}dx = \frac{1}{2}(\ln ax)^2 \tag{2.43}$$

$$\int \ln(ax+b)dx = \left(x+\frac{b}{a}\right)\ln(ax+b) - x, a \neq 0 \tag{2.44}$$

$$\int \ln(x^2+a^2)\,\mathrm{dx} = x\ln(x^2+a^2) + 2a\tan^{-1}\frac{x}{a} - 2x \tag{2.45}$$

$$\int \ln(x^2-a^2)\,\mathrm{dx} = x\ln(x^2-a^2) + a\ln\frac{x+a}{x-a} - 2x \tag{2.46}$$

$$\begin{aligned}\int \ln\left(ax^2+bx+c\right)dx &= \frac{1}{a}\sqrt{4ac-b^2}\tan^{-1}\frac{2ax+b}{\sqrt{4ac-b^2}} \\ &\quad - 2x + \left(\frac{b}{2a}+x\right)\ln\left(ax^2+bx+c\right)\end{aligned} \tag{2.47}$$

$$\begin{aligned}\int x\ln(ax+b)dx &= \frac{bx}{2a} - \frac{1}{4}x^2 \\ &\quad + \frac{1}{2}\left(x^2-\frac{b^2}{a^2}\right)\ln(ax+b)\end{aligned} \tag{2.48}$$

$$\begin{aligned}\int x\ln\left(a^2-b^2x^2\right)dx &= -\frac{1}{2}x^2+ \\ &\quad \frac{1}{2}\left(x^2-\frac{a^2}{b^2}\right)\ln\left(a^2-b^2x^2\right)\end{aligned} \tag{2.49}$$

$$\int e^{ax}dx = \frac{1}{a}e^{ax} \tag{2.50}$$

$$\int \sqrt{x}e^{ax}dx = \frac{1}{a}\sqrt{x}e^{ax} + \frac{i\sqrt{\pi}}{2a^{3/2}}\operatorname{erf}\left(i\sqrt{ax}\right), \quad \text{where } \operatorname{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt \tag{2.51}$$

$$\int xe^x dx = (x-1)e^x \tag{2.52}$$

$$\int xe^{ax}dx = \left(\frac{x}{a} - \frac{1}{a^2}\right)e^{ax} \tag{2.53}$$

$$\int x^2e^x dx = \left(x^2 - 2x + 2\right)e^x \tag{2.54}$$

$$\int x^2e^{ax}dx = \left(\frac{x^2}{a} - \frac{2x}{a^2} + \frac{2}{a^3}\right)e^{ax} \tag{2.55}$$

$$\int x^3e^x dx = \left(x^3 - 3x^2 + 6x - 6\right)e^x \tag{2.56}$$

$$\int x^ne^{ax}\,\mathrm{d}x = \frac{x^ne^{ax}}{a} - \frac{n}{a}\int x^{n-1}e^{ax}\,\mathrm{d}x \tag{2.57}$$

$$\int x^ne^{ax}\,\mathrm{d}x = \frac{(-1)^n}{a^{n+1}}\Gamma[1+n,-ax], \quad \text{where } \Gamma(a,x) = \int_x^\infty t^{a-1}e^{-t}\,\mathrm{d}t \tag{2.58}$$

$$\int e^{ax^2}\,\mathrm{d}x = -\frac{i\sqrt{\pi}}{2\sqrt{a}}\operatorname{erf}\left(ix\sqrt{a}\right) \tag{2.59}$$

$$\int e^{-ax^2}\,\mathrm{d}x = \frac{\sqrt{\pi}}{2\sqrt{a}}\operatorname{erf}\left(x\sqrt{a}\right) \tag{2.60}$$

$$\int xe^{-ax^2}\,\mathrm{d}x = -\frac{1}{2a}e^{-ax^2} \tag{2.61}$$

$$\int x^2e^{-ax^2}\,\mathrm{d}x = \frac{1}{4}\sqrt{\frac{\pi}{a^3}}\operatorname{erf}(x\sqrt{a}) - \frac{x}{2a}e^{-ax^2} \tag{2.62}$$

$$\int \sin ax dx = -\frac{1}{a}\cos ax \tag{2.63}$$

$$\int \sin^2 ax dx = \frac{x}{2} - \frac{\sin 2ax}{4a} \tag{2.64}$$

$$\int \sin^3 ax dx = -\frac{3\cos ax}{4a} + \frac{\cos 3ax}{12a} \tag{2.65}$$

$$\int \cos ax dx = \frac{1}{a}\sin ax \tag{2.66}$$

$$\int \cos^2 ax dx = \frac{x}{2} + \frac{\sin 2ax}{4a} \tag{2.67}$$

$$\int \cos^3 ax dx = \frac{3\sin ax}{4a} + \frac{\sin 3ax}{12a} \tag{2.68}$$

$$\int \cos ax \sin bx dx = \frac{\cos[(a-b)x]}{2(a-b)} - \frac{\cos[(a+b)x]}{2(a+b)}, a \neq b \tag{2.69}$$

$$\int \sin^2 ax \cos bx dx = -\frac{\sin[(2a-b)x]}{4(2a-b)} + \frac{\sin bx}{2b} - \frac{\sin[(2a+b)x]}{4(2a+b)} \tag{2.70}$$

$$\int \sin^2 x \cos x dx = \frac{1}{3}\sin^3 x \tag{2.71}$$

$$\int \cos^2 ax \sin bx dx = \frac{\cos[(2a-b)x]}{4(2a-b)} - \frac{\cos bx}{2b} - \frac{\cos[(2a+b)x]}{4(2a+b)} \tag{2.72}$$

$$\int \cos^2 ax \sin ax dx = -\frac{1}{3a}\cos^3 ax \tag{2.73}$$

$$\int \sin^2 ax \cos^2 bx dx = \frac{x}{4} - \frac{\sin 2ax}{8a} - \frac{\sin[2(a-b)x]}{16(a-b)} + \frac{\sin 2bx}{8b} - \frac{\sin[2(a+b)x]}{16(a+b)} \tag{2.74}$$

$$\int \sin^2 ax \cos^2 ax dx = \frac{x}{8} - \frac{\sin 4ax}{32a} \tag{2.75}$$

$$\int \tan ax dx = -\frac{1}{a}\ln\cos ax \tag{2.76}$$

$$\int \tan^2 ax dx = -x + \frac{1}{a}\tan ax \tag{2.77}$$

$$\int \tan^3 ax dx = \frac{1}{a}\ln\cos ax + \frac{1}{2a}\sec^2 ax \tag{2.78}$$

$$\int \sec x dx = \ln|\sec x + \tan x| = 2\tanh^{-1}\left(\tan\frac{x}{2}\right) \tag{2.79}$$

$$\int \sec^2 ax dx = \frac{1}{a}\tan ax \tag{2.80}$$

$$\int \sec^3 x\,\mathrm{d}x = \frac{1}{2}\sec x\tan x + \frac{1}{2}\ln|\sec x + \tan x| \tag{2.81}$$

$$\int \sec x \tan x dx = \sec x \tag{2.82}$$

$$\int \sec^2 x \tan x dx = \frac{1}{2}\sec^2 x \tag{2.83}$$

$$\int \sec^n x \tan x dx = \frac{1}{n}\sec^n x, n \neq 0 \tag{2.84}$$

$$\int \csc x dx = \ln\left|\tan\frac{x}{2}\right| = \ln|\csc x - \cot x| + C \tag{2.85}$$

$$\int \csc^2 ax dx = -\frac{1}{a}\cot ax \tag{2.86}$$

$$\int \csc^3 x dx = -\frac{1}{2}\cot x\csc x + \frac{1}{2}\ln|\csc x - \cot x| \tag{2.87}$$

$$\int \csc^n x \cot x dx = -\frac{1}{n}\csc^n x, n \neq 0 \tag{2.88}$$

$$\int \sec x \csc x dx = \ln|\tan x| \tag{2.89}$$

$$\int x\cos x dx = \cos x + x\sin x \tag{2.90}$$

$$\int x\cos ax dx = \frac{1}{a^2}\cos ax + \frac{x}{a}\sin ax \tag{2.91}$$

$$\int x^2\cos x dx = 2x\cos x + \left(x^2 - 2\right)\sin x \tag{2.92}$$

$$\int x^2\cos ax dx = \frac{2x\cos ax}{a^2} + \frac{a^2x^2 - 2}{a^3}\sin ax \tag{2.93}$$

$$\int x^n cosx dx = -\frac{1}{2}(i)^{n+1}\left[\Gamma(n+1,-ix) + (-1)^n\Gamma(n+1,ix)\right] \tag{2.94}$$

$$\int x^n cosax dx = \frac{1}{2}(ia)^{1-n}\left[(-1)^n\Gamma(n+1,-iax) - \Gamma(n+1,ixa)\right] \tag{2.95}$$

$$\int x\sin x dx = -x\cos x + \sin x \tag{2.96}$$

$$\int x\sin ax dx = -\frac{x\cos ax}{a} + \frac{\sin ax}{a^2} \tag{2.97}$$

$$\int x^2\sin x dx = \left(2 - x^2\right)\cos x + 2x\sin x \tag{2.98}$$

$$\int x^2\sin ax dx = \frac{2 - a^2x^2}{a^3}\cos ax + \frac{2x\sin ax}{a^2} \tag{2.99}$$

$$\int x^n\sin x dx = -\frac{1}{2}(i)^n\left[\Gamma(n+1,-ix) - (-1)^n\Gamma(n+1,-ix)\right] \tag{2.100}$$

$$\int e^x\sin x dx = \frac{1}{2}e^x(\sin x - \cos x) \tag{2.101}$$

$$\int e^{bx}\sin ax dx = \frac{1}{a^2+b^2}e^{bx}(b\sin ax - a\cos ax) \tag{2.102}$$

$$\int e^x\cos x dx = \frac{1}{2}e^x(\sin x + \cos x) \tag{2.103}$$

$$\int e^{bx}\cos ax dx = \frac{1}{a^2+b^2}e^{bx}(a\sin ax + b\cos ax) \tag{2.104}$$

$$\int xe^x \sin x dx = \frac{1}{2}e^x(\cos x - x\cos x + x\sin x) \tag{2.105}$$

$$\int xe^x \cos x dx = \frac{1}{2}e^x(x\cos x - \sin x + x\sin x) \tag{2.106}$$

2.6 Sums

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

$$\begin{aligned}
\sum_{i=1}^{n} i^1 &= \tfrac{n(n+1)}{2} = \tfrac{1}{2}n^2 + \tfrac{1}{2}n \\
\sum_{i=1}^{n} i^2 &= \tfrac{n(n+1)(2n+1)}{6} = \tfrac{1}{3}n^3 + \tfrac{1}{2}n^2 + \tfrac{1}{6}n \\
\sum_{i=1}^{n} i^3 &= \left[\tfrac{n(n+1)}{2}\right]^2 = \tfrac{1}{4}n^4 + \tfrac{1}{2}n^3 + \tfrac{1}{4}n^2 \\
\sum_{i=1}^{n} i^4 &= \tfrac{n(n+1)(2n+1)(3n^2+3n-1)}{30} \\
&= \tfrac{1}{5}n^5 + \tfrac{1}{2}n^4 + \tfrac{1}{3}n^3 - \tfrac{1}{30}n \\
\sum_{i=1}^{n} i^5 &= \tfrac{n^2(n+1)^2(2n^2+2n-1)}{12} \\
&= \tfrac{1}{6}n^6 + \tfrac{1}{2}n^5 + \tfrac{5}{12}n^4 - \tfrac{1}{12}n^2 \\
\sum_{i=1}^{n} i^6 &= \tfrac{n(n+1)(2n+1)(3n^4+6n^3-3n+1)}{42} \\
&= \tfrac{1}{7}n^7 + \tfrac{1}{2}n^6 + \tfrac{1}{2}n^5 - \tfrac{1}{6}n^3 + \tfrac{1}{42}n
\end{aligned}$$

2.7 Series

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

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

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

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

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

Polynomial Tricks

Newton's Method: $F_{t+1}(x) \equiv F_t(x) - \frac{G(F_t(x))}{G'(F_t(x))}$

Lagrange inversion: $n[x^n]F(x)^k = k[x^{-k}]G(x)^{-n}$ (where $F(G(x)) = x$) or $[x^n]F(x)^k = [x^{-k-1}]G(x)^{-n-1}G'$

With H: $[x^n]H(F(x)) = 1/n[x^{n-1}]H'(x)(x/G(x))^n$ or $[x^n]H(F(x)) = [x^n]H(x)(x/G(x))^{n+1}G'(x)$

Bernoulli

Let $S_k(x) = \sum_{i=0}^{x-1} i^k$.

$S_k(x) = \frac{1}{k+1}\sum_{g=0}^{k} C_{k+1}^{g} B_g x^{k+1-g} = k!\sum_{g=1}^{k+1} x^g/g! \times B_{k+1-g}/(k+1-g)!$.

$\sum_{i=0}^{\infty} \frac{B_i}{i!}x^i = \frac{1}{\sum_{i=0}^{\infty} \frac{x^i}{(i+1)!}}$

If you need $\sum_{i=0}^{x} i^k$ add one to B_1.

$$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\sum_{k=0}^{m} C(m+1,k)B_k n^{m+1-k} \quad (m > 0)$$

2.8 Probability theory

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

2.8.1 Discrete distributions

Binomial distribution

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

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

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

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

First success distribution

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

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

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

Poisson distribution

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

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

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

2.8.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

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

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

Normal distribution

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

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

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

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

2.9 Markov chains

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

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

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

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

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

2.10 Misc

Surreal Number

For real $x < y$: if contains integer, integer closest to 0. Otherwise the one with smallest denominator which is a power of 2.

L/R	0	$*$	$\uparrow$	$\downarrow$
0	$* = \{0 \mid 0\}$	$\uparrow = \{0 \mid *\}$	$\Uparrow * = \{0 \mid \uparrow\}$	$* = \{0 \mid 0\}$
$*$	$\downarrow = \{* \mid 0\}$	$0 = \{\mid\}$	$0 = \{\mid\}$	$\{* \mid \downarrow\}$
$\uparrow$	$* = \{0 \mid 0\}$	$\{\uparrow \mid *\}$	$\Uparrow * = \{0 \mid \uparrow\}$	$* = \{0 \mid 0\}$
$\downarrow$	$\Downarrow * = \{\downarrow \mid 0\}$	$0 = \{\mid\}$	$0 = \{\mid\}$	$\Downarrow * = \{\downarrow \mid 0\}$

Data structures (3)

Rope.cpp

Description: "ext/rope" contains a data structure that is a generalization of a string.

<bits/stdc++.h>, <ext/rope>, std::, __gnu_cxx::　59d0df, 47 lines

```cpp
// rope: push_back(x), insert(pos,x), erase(pos,x),
// replace(pos,x), substr(pos,x), copy(x), at[x], +=, mutable
const int maxn=1e5+10;
rope<char> *his[maxn];
int n,d[maxn];
inline void updata(int x){
  while(x<=n) d[x]++, x+=x&-x;
}
inline int get(int x){
  int res=0;
  while(x) res+=d[x], x-=x&-x;
  return res;
}
void deb(rope<char> s){
  for(int i=0;i<s.length();i++)
  cout<<s[i];puts("");
}
int main(){
  cin>>n;
  his[0]=new rope<char>();
  for(int i=1;i<=n;i++){
    his[i]=new rope<char>(*his[i-1]);
//    deb(*his[i]);
    char opt; cin>>opt;
    if(opt=='T'){
      cin>>opt;
      his[i]->push_back(opt);
      updata(i);
    }else if(opt=='U'){
      updata(i);
      int x,l=1,r=i,mid,now=get(i);
      cin>>x;
      while(l<r){
        mid=(l+r)>>1;
        if(now-get(mid)>x) l=mid+1;
        else r=mid;
      }
      his[i]=his[l-1];

    }else if(opt=='Q'){
      int x; cin>>x; --x;
      putchar(his[i]->at(x));
      putchar('\n');
    }
  }
  return 0;
}
```

OrderStatisticTree.h

Description: A set (not multiset!) with support for finding the n'th element, and finding the index of an element (number of smaller elements). To get a map, change null_type.

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

__gnu_pbds::　4dcf35, 15 lines

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

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

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

DSU.cpp

Description: Union find set.

0782da, 27 lines

```cpp
struct DSU {
    vector<int> sz, parent;
    void make_set(int v) {
        parent[v] = v;
        sz[v] = 1;
    }
    int find_set(int v) {
        if (v == parent[v])
            return v;
        return find_set(parent[v]);
    }
    void union_sets(int a, int b) {
        a = find_set(a);
        b = find_set(b);
        if (a != b) {
            if (sz[a] < sz[b])
                swap(a, b);
            parent[b] = a;
            sz[a] += sz[b];
        }
    }
    DSU(int n) {
        parent.resize(n);
        sz.resize(n);
        for (int i = 0; i<n; i++) make_set(i);
    }
};
```

HashTable.cpp

Description: A hash table. TODO: test if it's faster than e.g. cuckoo.

69bb04, 16 lines

```cpp
#define MOD 1000007
struct my_hash_table{
  int h[MOD],nxt[SZ],vv[SZ],ec; ll fl[SZ];
  int &operator[](ll v){
    for(int e=h[v%MOD];e;e=nxt[e])if(fl[e]==v)return vv[e];
    ++ec,fl[ec]=v,nxt[ec]=h[v%MOD],h[v%MOD]=ec;
    vv[ec]=0; return vv[ec];
  }
  bool count(int v){
    for(int e=h[v%MOD];e;e=nxt[e])if(fl[e]==v)return true;
    return false;
  }
  void clear() {
    for(int i=1;i<=ec;++i) h[fl[i]%MOD]=0;ec=0;
  }
};
```

LinkCutTree.cpp

Description: Link cut tree.

6d9141, 57 lines

```cpp
int ch[SZ][2],fa[SZ],sum[SZ],vv[SZ],mx[SZ];
bool rev[SZ];
bool top(int x) {return !(ch[fa[x]][0]==x||ch[fa[x]][1]==x);}
void pd(int x) {
    if(!rev[x]) return;
    rev[x]=0;
    rev[ch[x][0]]^=1;
    rev[ch[x][1]]^=1;
    swap(ch[x][0],ch[x][1]);
}
void upd(int x) {
    sum[x]=sum[ch[x][0]]+sum[ch[x][1]]+vv[x];
    if(vv[mx[ch[x][0]]]>vv[mx[ch[x][1]]]
  &&vv[mx[ch[x][0]]]>vv[x]) mx[x]=mx[ch[x][0]];
    else if(vv[mx[ch[x][1]]]>vv[x]) mx[x]=mx[ch[x][1]];
    else mx[x]=x;
}
void rot(int x) {
    if(top(x)) return;
    int y=fa[x],c=ch[y][0]==x;
    int f=fa[y];
    if(!top(y)) ch[f][ch[f][1]==y]=x;
    if(ch[x][c]) fa[ch[x][c]]=y;
    ch[y][!c]=ch[x][c];
    ch[x][c]=y; fa[x]=f; fa[y]=x;
    upd(y); upd(x);
}
int ss[SZ],sn;
void splay(int x) {
    sn=0;
    for(int c=x;;c=fa[c]) {
        ss[++sn]=c;
        if(top(c)) break;
    }
    while(sn) pd(ss[sn--]);
    while(!top(x)) {
        int y=fa[x];
        if(!top(y)) {
            if(ch[fa[y]][0]==y^ch[y][0]==x) rot(x);
            else rot(y);
        }
        rot(x);
    }
}
void access(int x) {
    for(int c=0;x;c=x,x=fa[x]) splay(x), ch[x][1]=c, upd(x);
}
```

```
void makeroot(int x) {access(x); splay(x); rev[x]^=1;}
void link(int a,int b) {makeroot(a); fa[a]=b;}
void cut(int a,int b) {makeroot(a); access(b); splay(b); ch[b
    ][0]=fa[a]=0;}
int findroot(int x) {
    access(x); splay(x);
    int lc=x;
    while(ch[lc][0]) lc=ch[lc][0];
    splay(lc); return lc;
}
int getrd(int a,int b) {makeroot(a); access(b); splay(b);
    return b;}
```

Treap.cpp

Description: Treap.

<bits/stdc++.h>, std::   58a367, 72 lines

```
#define ll long long
#define SZ 233333
mt19937 mt1(time(0));
int ch[SZ][2],sz[SZ],an=0,root;
ll tag[SZ],val[SZ],sum[SZ];
void addnode(int& ad,int x) {
  ad=++an; sz[ad]=1; sum[ad]=val[ad]=x;
}
void pd(int x) {
  if(!x||!tag[x]) return;
  val[x]+=tag[x];
  if(ch[x][0]) tag[ch[x][0]]+=tag[x], sum[ch[x][0]]+=tag[x]*sz[
      ch[x][0]];
  if(ch[x][1]) tag[ch[x][1]]+=tag[x], sum[ch[x][1]]+=tag[x]*sz[
      ch[x][1]];
  tag[x]=0;
}
void upd(int x) {
  if(!x) return;
  sz[x]=1+sz[ch[x][0]]+sz[ch[x][1]];
  sum[x]=val[x]+sum[ch[x][0]]+sum[ch[x][1]];
}
void split(int x,int& a,int& b,int s) {
  if(sz[x]<=s) a=x, b=0;
  else if(s==0) a=0, b=x;
  else {
    pd(x);
    if(sz[ch[x][0]]>=s)
      b=x, split(ch[x][0],a,ch[x][0],s), upd(x);
    else
      a=x, split(ch[x][1],ch[x][1],b,s-sz[ch[x][0]]-1), upd(x);
  }
}
void merge(int& ad,int a,int b) {
  if(a&&b) {
    if((int)mt1()%(sz[a]+sz[b])<sz[a])
      pd(ad=a), merge(ch[a][1],ch[a][1],b);
    else
      pd(ad=b), merge(ch[b][0],a,ch[b][0]);
    upd(ad);
  }
  else ad=a|b;
}
void edit(int l,int r,ll v) {
  int a,b,c;
  split(root,a,b,l-1);
  split(b,b,c,r-l+1);
  tag[b]+=v;
  sum[b]+=v*sz[b];
  merge(a,a,b);
  merge(root,a,c);
}
ll query(int l,int r) {
  int a,b,c;
  split(root,a,b,l-1);
  split(b,b,c,r-l+1);
  ll ans=sum[b];
  merge(a,a,b);
  merge(root,a,c);
  return ans;
}
int n,ns[SZ];
int ins(int l,int r) {
  if(l>r) return 0;
  if(l==r) {
    int ad; addnode(ad,ns[l]); return ad;
  }
  int mid=(l+r)>>1,lc=ins(l,mid),rc=ins(mid+1,r);
  merge(lc,lc,rc); return lc;
}
void init() {
  addnode(root,0);
  merge(root,ins(1,n),root);
}
```

LineContainer.h

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

Time: $\mathcal{O}(\log N)$ 8ec1c7, 30 lines

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

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

RMQ.h

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

Usage: `RMQ rmq(values);`
`rmq.query(inclusive, exclusive);`

Time: $\mathcal{O}(|V|\log|V| + Q)$ 510c32, 16 lines

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

Numerical (4)

4.1 Polynomials and recurrences

PolyInterpolate.h

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

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

vd = vector<double>   285367, 12 lines

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

LinearRecurrence.cpp

Description: Linear recurrence toolkit.

ll = long long, std::   420684, 60 lines

```
const int MOD=1e9+7;
ll qp(ll a,ll b) {
  ll x=1; a%=MOD;
  while(b) {
    if(b&1) x=x*a%MOD;
    a=a*a%MOD; b>>=1;
  }
  return x;
}
namespace linear_seq {
inline vector<int> BM(vector<int> x) {
  vector<int> ls,cur;
  int pn=0,lf,ld;
  for(int i=0;i<int(x.size());++i) {
    ll t=-x[i]%MOD;
    for(int j=0;j<int(cur.size());++j)
      t=(t+x[i-j-1]*(ll)cur[j])%MOD;
    if(!t) continue;
    if(!cur.size())
    {cur.resize(i+1); lf=i; ld=t; continue;}
    ll k=-t*qp(ld,MOD-2)%MOD;
    vector<int> c(i-lf-1); c.pb(-k);
    for(int j=0;j<int(ls.size());++j) c.pb(ls[j]*k%MOD);
    if(c.size()<cur.size()) c.resize(cur.size());
    for(int j=0;j<int(cur.size());++j)
      c[j]=(c[j]+cur[j])%MOD;
    if(i-lf+(int)ls.size()>=(int)cur.size())
```

```
      ls=cur,lf=i,ld=t;
    cur=c;
  }
  vector<int>&o=cur;
  for(int i=0;i<int(o.size());++i)
    o[i]=(o[i]%MOD+MOD)%MOD;
  return o;
}
int N; ll a[SZ],h[SZ],t_[SZ],s[SZ],t[SZ];
inline void mull(ll*p,ll*q) {
  for(int i=0;i<N+N;++i) t_[i]=0;
  for(int i=0;i<N;++i) if(p[i])
    for(int j=0;j<N;++j)
      t_[i+j]=(t_[i+j]+p[i]*q[j])%MOD;
  for(int i=N+N-1;i>=N;--i) if(t_[i])
    for(int j=N-1;~j;--j)
      t_[i-j-1]=(t_[i-j-1]+t_[i]*h[j])%MOD;
  for(int i=0;i<N;++i) p[i]=t_[i];
}
inline ll calc(ll K) {
  for(int i=N;~i;--i) s[i]=t[i]=0;
  s[0]=1; if(N!=1) t[1]=1; else t[0]=h[0];
  for(;K;mull(t,t),K>>=1) if(K&1) mull(s,t); ll su=0;
  for(int i=0;i<N;++i) su=(su+s[i]*a[i])%MOD;
  return (su%MOD+MOD)%MOD;
}
inline int gao(vector<int> x,ll n) {
  if(n<int(x.size())) return x[n];
  vector<int> v=BM(x); N=v.size(); if(!N) return 0;
  for(int i=0;i<N;++i) h[i]=v[i],a[i]=x[i];
  return calc(n);
}
}
```

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 *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; }`
`double xmin = gss(-1000,1000,func);`
Time: $\mathcal{O}(\log((b-a)/\epsilon))$

31d45b, 14 lines

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

IntegrateAdaptive.h

Description: Fast integration using an adaptive Simpson's rule.
Usage: `double sphereVolume = quad(-1, 1, [](double x) {`
`return quad(-1, 1, [&](double y) {`
`return quad(-1, 1, [&](double z) {`
`return x*x + y*y + z*z < 1; });});});`

d = double 50bf66, 14 lines

```
#define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6

template <class F>
d rec(F& f, d a, d b, d eps, d S) {
  d c = (a + b) / 2;
  d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
  if (abs(T - S) <= 15 * eps || b - a < 1e-10)
    return T + (T - S) / 15;
  return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S2);
}
template<class F>
d quad(d a, d b, F f, d eps = 1e-8) {
  return rec(f, a, b, eps, S(a, b));
}
```

Simplex.cpp

Description: Simplex. Maximize $\sum_j c_j x_j$ subject to $\sum_j a_{i,j} x_j \leq b_i$ and $x_j \geq 0$.

f6ff59, 27 lines

```
const int N=25;const double eps=1e-7;
int n,m,op,lid[N],cid[N];double a[N][N],b[N];
void pivot(int l,int e){
  int i,j;double r=a[l][e];a[l][e]=1;for(i=0;i<=m;i++)a[l][i]/=
      r;swap(lid[l],cid[e]);
  for(i=0;i<=n;i++)if(i!=l)for(r=a[i][e],a[i][e]=j=0;j<=m;j++)a
      [i][j]-=r*a[l][j];
}
int main(){
  int i,j,l,e;double t,tt;scanf("%d%d%d",&m,&n,&op);
  for(i=1;i<=m;i++)scanf("%lf",&a[0][i]),cid[i]=i;
  for(i=1;i<=n;scanf("%lf",&a[i++][0]))for(j=1;j<=m;j++)scanf("
      %lf",&a[i][j]);
  while(true){
    for(l=0,i=1,t=-eps;i<=n;i++)if(a[i][0]<t)t=a[l=i][0];if(!l)
        break;
    for(e=0,i=1,t=-eps;i<=m;i++)if(a[l][i]<t&&(!e||(rand()&1)))
        t=a[l][e=i];
    if(!e)puts("Infeasible"),exit(0);pivot(l,e);
  }
  while(true){
    for(e=0,i=1,t=eps;i<=m;i++)if(a[0][i]>t)t=a[0][e=i];if(!e)
        break;
    for(l=0,i=1;i<=n;i++)if(a[i][e]>eps&&((tt=a[i][0]/a[i][e])<
        t||!l))t=tt,l=i;
    if(!l)puts("Unbounded"),exit(0);pivot(l,e);
  }
  printf("%.10lf\n",-a[0][0]);
  if(op){
    for(i=1;i<=n;i++)if(lid[i])b[lid[i]]=a[i][0];
    for(i=1;i<=m;i++)printf("%.10lf%c",b[i]," \n"[i==m]);
  }
  return 0;
}
```

4.3 Matrices

Determinant.h

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

bd5cec, 15 lines

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

SolveLinear.h

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

vd = vector<double> daa98b, 37 lines

```
const double eps = 1e-12;

int solveLinear(vector<vd>& A, vd& b, vd& x) {
  int n = sz(A), m = sz(x), rank = 0, br, bc;
  if (n) assert(sz(A[0]) == m);
  vi col(m); iota(all(col), 0);

  rep(i,0,n) {
    double v, bv = 0;
    rep(r,i,n) rep(c,i,m)
      if ((v = fabs(A[r][c])) > bv)
        br = r, bc = c, bv = v;
    if (bv <= eps) {
      rep(j,i,n) if (fabs(b[j]) > eps) return -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    rep(j,i+1,n) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      rep(k,i+1,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
  }

  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

SolveLinear2.h

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

"SolveLinear.h" 08e495, 7 lines

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

SolveLinearBinary.h

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

bs = bitset<1000> 26d73e, 32 lines

```
int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
  int n = sz(A), rank = 0, br;
  assert(m <= sz(x));
  vi col(m); iota(all(col), 0);
  rep(i,0,n) {
    for (br=i; br<n; ++br) if (A[br].any()) break;
    if (br == n) {
      rep(j,i,n) if(b[j]) return -1;
      break;
    }
    int bc = (int)A[br]._Find_next(i-1);
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) if (A[j][i] != A[j][bc]) {
      A[j].flip(i); A[j].flip(bc);
    }
    rep(j,i+1,n) if (A[j][i]) {
      b[j] ^= b[i];
      A[j] ^= A[i];
    }
    rank++;
  }

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

MatrixInverse.h

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

ebfff6, 35 lines

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

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

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

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

Tridiagonal.h

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

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} d_0 & p_0 & 0 & 0 & \cdots & 0 \\ q_0 & d_1 & p_1 & 0 & \cdots & 0 \\ 0 & q_1 & d_2 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & q_{n-3} & d_{n-2} & p_{n-2} \\ 0 & 0 & \cdots & 0 & q_{n-2} & d_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{pmatrix}.$$

This is useful for solving problems on the type

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

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

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

Fails if the solution is not unique.
If $|d_i| > |p_i| + |q_{i-1}|$ for all i, or $|d_i| > |p_{i-1}| + |q_i|$, or the matrix is positive definite, the algorithm is numerically stable and neither `tr` nor the check for `diag[i] == 0` is needed.
Time: $\mathcal{O}(N)$

T = double d284f8, 25 lines

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

4.4 Fourier transforms

FastFourierTransform.h

Description: fft(a) computes $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$ for all k. N must be a power of 2. Useful for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by n, reverse(start+1, end), FFT back. Rounding is safe if $(\sum a_i^2 + \sum b_i^2)\log_2 N < 9 \cdot 10^{14}$ (in practice 10^{16}; higher for random inputs). Otherwise, use NTT/FFTMod.
Time: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ ($\sim 1s$ for $N = 2^{22}$)

C = complex<double>, vd = vector<double> 22c834, 33 lines

```
void fft(vector<C>& a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vector<complex<long double>> R(2, 1);
  static vector<C> rt(2, 1);  // (^ 10% faster if double)
  for (static int k = 2; k < n; k *= 2) {
    R.resize(n); rt.resize(n);
    auto x = polar(1.0L, acos(-1.0L) / k);
    rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
  }
  vi rev(n);
  rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
  rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
      C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled)
      a[i + j + k] = a[i + j] - z;
      a[i + j] += z;
    }
}
vd conv(const vd& a, const vd& b) {
  if (a.empty() || b.empty()) return {};
  vd res(sz(a) + sz(b) - 1);
  int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
  vector<C> in(n), out(n);
  copy(all(a), begin(in));
  rep(i,0,sz(b)) in[i].imag(b[i]);
  fft(in);
  for (C& x : in) x *= x;
  rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
  fft(out);
  rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
  return res;
}
```

FastFourierTransformMod.h

Description: Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$ (in practice 10^{16} or higher). Inputs must be in $[0, \text{mod})$.
Time: $\mathcal{O}(N \log N)$, where $N = |A| + |B|$ (twice as slow as NTT or FFT)

"FastFourierTransform.h", vl = vector<ll> c1f2f1, 21 lines

```
template<int M> vl convMod(const vl &a, const vl &b) {
  if (a.empty() || b.empty()) return {};
  vl res(sz(a) + sz(b) - 1);
  int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
  vector<C> L(n), R(n), outs(n), outl(n);
  rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
  rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
  fft(L), fft(R);
  rep(i,0,n) {
    int j = -i & (n - 1);
    outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
    outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
  }
  fft(outl), fft(outs);
  rep(i,0,sz(res)) {
    ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
    ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
    res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
  }
  return res;
}
```

FastSubsetTransform.h

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

464cf3, 16 lines

```
void FST(vi& a, bool inv) {
  for (int n = sz(a), step = 1; step < n; step *= 2) {
    for (int i = 0; i < n; i += 2 * step) rep(j,i,i+step) {
      int &u = a[j], &v = a[j + step]; tie(u, v) =
        inv ? pii(v - u, u) : pii(v, u + v); // AND
        inv ? pii(v, u - v) : pii(u + v, u); // OR
        pii(u + v, u - v);                   // XOR
    }
  }
  if (inv) for (int& x : a) x /= sz(a); // XOR only
}
vi conv(vi a, vi b) {
  FST(a, 0); FST(b, 0);
  rep(i,0,sz(a)) a[i] *= b[i];
  FST(a, 1); return a;
}
```

Number theory (5)

5.1 Modular arithmetic

ModularArithmetic.h

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

"euclid.h" 35bfea, 18 lines

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

ModInverse.h

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

6f684f, 3 lines

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

ModPow.h

b83e45, 7 lines

```
const ll mod = 1000000007; // faster if const
ll modpow(ll b, ll e) {
  ll ans = 1;
  for (; e; b = b * b % mod, e /= 2)
    if (e & 1) ans = ans * b % mod;
  return ans;
}
```

ModLog.h

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

c040b8, 11 lines

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

ModMulLL.h

Description: Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$) for $0 \le a, b \le c \le 7.2 \cdot 10^{18}$. Requires x87 80-bit floats (e.g. GCC). Only valid for $< 2^{52} \approx 4.5 \cdot 10^{15}$ if long double is 64-bit.
Time: $\mathcal{O}(1)$ for modmul, $\mathcal{O}(\log b)$ for modpow

ull = unsigned long long fabd71, 10 lines

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

ModSqrt.h

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

"ModPow.h" 19a793, 24 lines

```
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  assert(modpow(a, (p-1)/2, p) == 1); // else no solution
  if (p % 4 == 3) return modpow(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1, n = 2;
  int r = 0, m;
  while (s % 2 == 0)
    ++r, s /= 2;
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p), g = modpow(n, s, p);
  for (;; r = m) {
    ll t = b;
    for (m = 0; m < r && t != 1; ++m)
      t = t * t % p;
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

Euclidean.cpp

Description: Floor sum.

ll = long long 9d1ecd, 20 lines

```
ll flr(ll x, ll y) { // floor(x / y). need y > 0
  ll res = x / y;
  if (res * y > x) res -= 1;
  return res;
}
ll f(ll a, ll b, ll c, ll n) { // sum_{0<=i<=n} flr((ai+b)/c).
    need c > 0
  ll pls = (a / c) * n * (n + 1) / 2;
  a %= c;
  if (a < 0) pls -= n * (n + 1) / 2, a += c;
  pls += (n + 1) * (b / c);
  b %= c;
  if (b < 0) pls -= (n + 1), b += c;
  if (a == 0) {
    ll ans = flr(b, c) * (n + 1);
    return ans + pls;
  }
  ll m = flr(a * n + b, c);
  ll res = n * m - f(c, c - b - 1, a, m - 1);
  return res + pls;
}
```

5.2 Primality

FastEratosthenes.h

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

6b2912, 20 lines

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

MillerRabin.h

Description: Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to $7 \cdot 10^{18}$; for larger numbers, use Python and extend A randomly.
Time: 7 times the complexity of $a^b \bmod c$.

"ModMulLL.h" 60dcd1, 12 lines

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

Factor.h

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

"ModMulLL.h", "MillerRabin.h" a33cf6, 18 lines

```
ull pollard(ull n) {
  auto f = [n](ull x) { return modmul(x, x, n) + 1; };
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  while (t++ % 40 || __gcd(prd, n) == 1) {
    if (x == y) x = ++i, y = f(x);
    if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}
vector<ull> factor(ull n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), all(r));
  return l;
}
```

Min25SieveDFS.cpp

Description: Min25 sieve with DFS.

<bits/stdc++.h>, std::, us = unsigned, ull = unsigned long long, ll = long long 83c4ef, 105 lines

```
const us MOD=1e9+7;
#define SS 2333333
ll n,c0[SS],c1[SS],b0[SS],b1[SS];
#define MU 6000000
int S,ps[MU/10],pn=0;
int U=MU;
#define V 40
unsigned fc[V][MU];
inline ull F(ull x,us g) {
  if(g<V&&x<U) return fc[g][x];
    if(x<=1||ps[g]>x) return 0;
    ull ans=((x>S)?b1[n/x]:c1[x])-c1[ps[g-1]]+MOD;
    for(us j=g;j<=pn&&ps[j]*(ll)ps[j]<=x;++j) {
        ull cn=x/ps[j],ce=ps[j]*(ll)ps[j];
        for(us e=1;cn>=ps[j];++e,cn/=ps[j],ce*=ps[j])
            ans+=F(cn,j+1)*(ps[j]^e)+(ps[j]^(e+1)),ans%=MOD;
    }
    return ans%MOD;
}
int mf[MU];
unsigned f[MU];
int main() {
//   n=1e11;
    cin>>n;
  S=sqrtl(n);
  U=min(U,int(S*30));
  for(int i=1;i<U;++i) f[i]=1;
  for(int i=2;i<U;++i) {
    if(!mf[i]) {
      f[i]=i^1; ps[++pn]=i; mf[i]=pn;
    }
    for(int j=1;j<=pn&&i*ps[j]<U;++j) {
      int w=i*ps[j]; mf[w]=j;
      if(i%ps[j]==0) {
        int p=ps[j];
        int o=w,c=0;
        while(o%p==0) o/=p,++c;
        f[w]=f[o]*(p^c);
        break;
      }
      else f[w]=f[i]*(ps[j]^1);
    }
  }
  for(int j=0;j<V;++j) {
    unsigned*fcj=fc[j];
    for(int i=2;i<U;++i) {
      if(mf[i]>=j) {
        fcj[i]=fcj[i-1]+f[i];
        (fcj[i]>=MOD)?(fcj[i]-=MOD):0;
      }
      else fcj[i]=fcj[i-1];
    }
  }
  pn=0;
    for(int i=1;i<=S;++i) {
        ll t=(n/i)%MOD; b0[i]=t;
        b1[i]=t*(t+1)/2%MOD; c0[i]=i;
        c1[i]=i*(ll)(i+1)/2%MOD;
    }
    for(int i=2;i<=S;++i)
    {
        if(c0[i]==c0[i-1]) continue; //not a prime
        ll x0=c0[i-1],x1=c1[i-1]%MOD,r=(ll)i*i; ps[++pn]=i;
        int u=min((ll)S,n/(i*(ll)i)),uu=min(u,S/i);
        for(int j=1;j<=uu;++j)
            (b1[j]-=(b1[j*i]-x1)*i)%=MOD,
            b0[j]-=b0[j*i]-x0;
        ll t=n/i;
        if(t<=2147483647) {
        int tt=t;
        for(int j=uu+1;j<=u;++j)
            b1[j]-=(c1[tt/j]-x1)*i,
            b0[j]-=c0[tt/j]-x0;
        }
        else {
        for(int j=uu+1;j<=u;++j)
            (b1[j]-=(c1[t/j]-x1)*i)%=MOD,
            b0[j]-=c0[t/j]-x0;
        }
        for(int j=S;j>=r;--j)
            c1[j]-=(c1[j/i]-x1)*i,
            c0[j]-=c0[j/i]-x0;
    }
    for(int i=1;i<=S;++i) {
        c1[i]%=MOD;
        b1[i]%=MOD;
        c1[i]-=c0[i];
        b1[i]-=b0[i];
        if(i>=2) c1[i]+=2;
        if(n>=2LL*i) b1[i]+=2;
        c1[i]=(c1[i]%MOD+MOD)%MOD;
        b1[i]=(b1[i]%MOD+MOD)%MOD;
    }
    ps[pn+1]=S+1;
    vector<ll> u;
    for(ll i=1;i<=n;) {
      ll d=n/i,r=n/d; i=r+1;
      u.pb(((1+F(d,1))%MOD+MOD)%MOD);
  }
  sort(u.begin(),u.end());
  u.erase(unique(u.begin(),u.end()),u.end());
  ll ans=0;
  for(auto c:u) ans^=c;
  cout<<ans<<"\n";
}
```

5.3 Divisibility

euclid.h

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

33ba8f, 5 lines

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

MagicEuclid.cpp

Description: Magic euclid. Calculate $\sum_{x=1}^{L} A^x B^{\lfloor (Px+R)/Q \rfloor}$.

<bits/stdc++.h>, std::, ll = __int128 73f8b3, 58 lines

```
const int mod=998244353;
long long p,q,r,l; int n;
inline int add(int x,int y){return x+y<mod?x+y:x+y-mod;}
struct matrix{
  int a[21][21];
  matrix(){memset(a,0,sizeof(a));}
  matrix operator +(const matrix &b){
    matrix c;
    for(int i=1;i<=n;i++)
      for(int j=1;j<=n;j++)
        c.a[i][j]=add(a[i][j],b.a[i][j]);
    return c;
  }
  matrix operator *(const matrix &b){
    matrix (c);
    for(int i=1;i<=n;i++){
      for(int k=1;k<=n;k++){
        if(!a[i][k]) continue;
        for(int j=1;j<=n;j++)
        c.a[i][j]=add(c.a[i][j],1ll*a[i][k]*b.a[k][j]%mod);
      }
    }
    return c;
  }
}a,b,I,ey;
struct node{
  matrix u,r,s;
  node operator +(const node &k){
    return node{u*k.u,r*k.r,s+r*k.s*u};
  }
  node operator *(ll n){
    node x=node{I,I,ey},y=node{u,r,s};
    while(n){
      if(n&1) x=x+y;
      y=y+y,n>>=1;
    }
    return x;
  }
}ans;
inline node solve(ll p,ll q,ll b,ll l,node u,node r){
  if(!l) return node{I,I,ey};
  if(p>=q) return solve(p%q,q,b,l,u,u*(p/q)+r);
  ll cnt=(l*p+b)/q; if(!cnt) return r*l;
  return r*((q-b-1)/p)+u+solve(q,p,(q-b-1)%p,cnt-1,r,u)+r*(l-(
      cnt*q-b-1)/p);
}
int main(){
  cin>>p>>q>>r>>l>>n;
  for(int i=1;i<=n;i++)
    for(int j=1;j<=n;j++) cin>>a.a[i][j];
  for(int i=1;i<=n;i++)
    for(int j=1;j<=n;j++) cin>>b.a[i][j];
  for(int i=1;i<=n;i++) I.a[i][i]=1;
  ans=node{b,I,ey}*(r/q)+solve(p,q,r%q,l,node{b,I,ey},node{I,a,
      a});
  for(int i=1;i<=n;i++){
    for(int j=1;j<=n;j++) cout<<ans.s.a[i][j]<<' ';
    cout<<endl;
  }
}
```

CRT.h

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

Time: log(n)

"euclid.h" 04d93a, 7 lines

```
ll crt(ll a, ll m, ll b, ll n) {
  if (n > m) swap(a, b), swap(m, n);
  ll x, y, g = euclid(m, n, x, y);
  assert((a - b) % g == 0); // else no solution
  x = (b - a) % n * x % n / g * m + a;
  return x < 0 ? x + m*n/g : x;
}
```

5.4 Fractions

ContinuedFractions.h

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

dd6c5e, 21 lines

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

FracBinarySearch.h

Description: Given f and N, finds the smallest fraction $p/q \in [0, 1]$ such that $f(p/q)$ is true, and $p, q \leq N$. You may want to throw an exception from f if it finds an exact solution, in which case N can be removed.
Usage: `fracBS([](Frac f) { return f.p>=3*f.q; }, 10); // {1,3}`
Time: $\mathcal{O}(\log(N))$

27ab3e, 25 lines

```
struct Frac { ll p, q; };

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

FarreySequence.cpp

Description: Use the Farrey sequence to find fraction between a/b and c/d with minimum denominator.

<cstdio>, <algorithm>, std::, ll = long long, pair<ll,ll>P = d18c2a, 24 lines

```
ll a,b,c,d,t;
ll gcd(ll a,ll b){return b?gcd(b,a%b):a;}
P cal(ll a,ll b,ll c,ll d){
  ll x=a/b+1;
  if(x*d<c)return P(x,1);
  if(!a)return P(1,d/c+1);
  if(a<=b&&c<=d){
    P t=cal(d,c,b,a);
    swap(t.first,t.second);
    return t;
  }
  x=a/b;
  P t=cal(a-b*x,b,c-d*x,d);
  t.first+=t.second*x;
  return t;
}
int main(){
  while(~scanf("%lld%lld%lld%lld",&a,&b,&c,&d)){
    t=gcd(a,b),a/=t,b/=t;
    t=gcd(c,d),c/=t,d/=t;
    P p=cal(a,b,c,d);
    printf("%lld/%lld\n",p.first,p.second);
  }
}
```

PellEquation.py

21 lines

```
# Pell equation solver, solves x^2 - n*y^2 = 1
# x[k] + y[k] sqrt(n) = (x[1] + y[1] sqrt(n))^k
def solve(n):
    j=1
    while j*j<n: j+=1
    if j*j==n:
        return (-1,-1)
    p,q,a,g,h=[[0 for _ in range(1001)]for _ in range(5)]
    p[1]=q[0]=h[1]=1
    p[0]=q[1]=g[1]=0
    a[2]=j-1
    i=2
    while 1:
        g[i]=-g[i-1]+a[i]*h[i-1]
        h[i]=(n-g[i]*g[i])//h[i-1]
        a[i+1]=(g[i]+a[2])//h[i]
        p[i]=a[i]*p[i-1]+p[i-2]
        q[i]=a[i]*q[i-1]+q[i-2]
        if(p[i]*p[i]-n*q[i]*q[i]==1):
          return (p[i],q[i])
        i+=1
```

5.5 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

Combinatorial (6)

6.1 Permutations

6.1.1 Factorial

IntPerm.h

Description: Permutation -> integer conversion. (Not order preserving.) Integer -> permutation can use a lookup table.
Time: $\mathcal{O}(n)$

044568, 6 lines

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

6.1.2 Cycles

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

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

6.1.3 Derangements

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

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

6.2 Partitions and subsets

6.2.1 Partition function

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

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

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

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

6.2.2 Lucas' Theorem

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

6.3 General purpose numbers

6.3.1 Bernoulli numbers

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

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

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

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

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

6.3.3 Eulerian numbers

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

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

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

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

6.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.3.5 Bell numbers

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

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

6.3.6 Catalan numbers

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

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

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

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

Graph (7)

7.1 Network flow

PushRelabel.h

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

Time: $\mathcal{O}\left(V^2\sqrt{E}\right)$ 0ae1d4, 46 lines

```
struct PushRelabel {
  struct Edge {
    int dest, back;
    ll f, c;
  };
  vector<vector<Edge>> g;
  vector<ll> ec;
  vector<Edge*> cur;
  vector<vi> hs; vi H;
  PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}
  void addEdge(int s, int t, ll cap, ll rcap=0) {
    if (s == t) return;
    g[s].push_back({t, sz(g[t]), 0, cap});
    g[t].push_back({s, sz(g[s])-1, 0, rcap});
  }
  void addFlow(Edge& e, ll f) {
    Edge &back = g[e.dest][e.back];
    if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
    e.f += f; e.c -= f; ec[e.dest] += f;
    back.f -= f; back.c += f; ec[back.dest] -= f;
  }
  ll calc(int s, int t) {
    int v = sz(g); H[s] = v; ec[t] = 1;
    vi co(2*v); co[0] = v-1;
    rep(i,0,v) cur[i] = g[i].data();
    for (Edge& e : g[s]) addFlow(e, e.c);

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

MinCostFlowAnton.cpp

Description: Simple min cost flow. 9e97e2, 67 lines

```
struct Edge {
    int from, to, capacity, cost;
};
vector<vector<int>> adj, cost, capacity;
const int INF = 1e9;
void shortest_paths(int n, int v0, vector<int>& d, vector<int>&
     p) {
    d.assign(n, INF);
    d[v0] = 0;
    vector<bool> inq(n, false);
    queue<int> q;
    q.push(v0);
    p.assign(n, -1);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (int v : adj[u]) {
            if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v])
                 {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (!inq[v]) {
                    inq[v] = true;
                    q.push(v);
                }
            }
        }
    }
}
int min_cost_flow(int N, vector<Edge> edges, int K, int s, int
     t) {
    adj.assign(N, vector<int>());
    cost.assign(N, vector<int>(N, 0));
    capacity.assign(N, vector<int>(N, 0));
    for (Edge e : edges) {
        adj[e.from].push_back(e.to);
        adj[e.to].push_back(e.from);
        cost[e.from][e.to] = e.cost;
        cost[e.to][e.from] = -e.cost;
        capacity[e.from][e.to] = e.capacity;
    }
    int flow = 0;
    int cost = 0;
    vector<int> d, p;
    while (flow < K) {
        shortest_paths(N, s, d, p);
        if (d[t] == INF) break;
        // find max flow on that path
        int f = K - flow;
        int cur = t;
        while (cur != s) {
            f = min(f, capacity[p[cur]][cur]);
            cur = p[cur];
        }
        // apply flow
        flow += f;
        cost += f * d[t];
        cur = t;
        while (cur != s) {
            capacity[p[cur]][cur] -= f;
            capacity[cur][p[cur]] += f;
            cur = p[cur];
        }
    }
    if (flow < K)
```

```
        return -1;
    else
        return cost;
}
```

Dinic.cpp

Description: Dinic.

14ea98, 40 lines

```
int N,S,T,M=1; //M=1 is important!!
int fst[SZ],vb[SZ],nxt[SZ],vc[SZ];
void ad_de_(int a,int b,int c)
{++M;nxt[M]=fst[a];fst[a]=M;vb[M]=b;vc[M]=c;}
void ad_de(int a,int b,int c){ad_de_(a,b,c);ad_de_(b,a,0);}
int d[SZ],ff[SZ];
bool bfs() {
  static int qs[SZ]; int h=0,t=0;
  memset(d,-1,sizeof(int)*(N+1));
  qs[t++]=S; d[S]=0;
  while(h^t) {
    int x=qs[h++];
    for esb(x,e,b) {
      if((~d[b])||!vc[e]) continue;
      d[b]=d[x]+1; qs[t++]=b;
    }
  }
  return d[T]!=-1;
}
ll dfs(int x,ll u) {
  if(x==T||!u) return u;
  ll f=0;
  for(int&e=fst[x];e;e=nxt[e]) {
    int b=vb[e];
    if(d[b]!=d[x]+1) continue;
    ll s=dfs(vb[e],min(u-f,(ll)vc[e]));
    f+=s; vc[e]-=s; vc[e^1]+=s;
    if(f==u) break;
  }
  if(!f) d[x]=-1;
  return f;
}
ll dinic() {
  ll ans=0;
  memcpy(ff,fst,sizeof(int)*(N+1));
  while(bfs())
    ans+=dfs(S,1e18),
    memcpy(fst,ff,sizeof(int)*(N+1));
  return ans;
}
```

DinicAnton.cpp

Description: Dinic.

b8d428, 75 lines

```
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap
        ) {}
};
struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;
    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }
    void add_edge(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }
    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }
    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid
            ++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u] || edges[id].cap -
                edges[id].flow < 1)
                continue;
            long long tr = dfs(u, min(pushed, edges[id].cap -
                edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }
    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

MinCut.h

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

GlobalMinCut.h

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

8b0e19, 21 lines

```
pair<int, vi> globalMinCut(vector<vi> mat) {
  pair<int, vi> best = {INT_MAX, {}};
  int n = sz(mat);
  vector<vi> co(n);
  rep(i,0,n) co[i] = {i};
  rep(ph,1,n) {
    vi w = mat[0];
    size_t s = 0, t = 0;
    rep(it,0,n-ph) { // O(V^2) -> O(E log V) with prio. queue
      w[t] = INT_MIN;
      s = t, t = max_element(all(w)) - w.begin();
      rep(i,0,n) w[i] += mat[t][i];
    }
    best = min(best, {w[t] - mat[t][t], co[t]});
    co[s].insert(co[s].end(), all(co[t]));
    rep(i,0,n) mat[s][i] += mat[t][i];
    rep(i,0,n) mat[i][s] = mat[s][i];
    mat[0][t] = INT_MIN;
  }
  return best;
}
```

GomoryHu.h

Description: Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.
Time: $\mathcal{O}(V)$ Flow Computations

"PushRelabel.h", Edge = array<ll, 3> 65c0c2, 12 lines

```
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
  vector<Edge> tree;
  vi par(N);
  rep(i,1,N) {
    PushRelabel D(N); // Dinic also works
    for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
    tree.push_back({i, par[i], D.calc(i, par[i])});
    rep(j,i+1,N)
      if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
  }
  return tree;
}
```

7.2 Matching

Blossom.cpp

Description: Maximum matching with blossom.

1b2a6f, 52 lines

```
vector<int> Blossom(vector<vector<int>>& graph) {
  int n = graph.size(), timer = -1;
  vector<int> mate(n, -1), label(n), parent(n),
              orig(n), aux(n, -1), q;
  auto lca = [&](int x, int y) {
    for (timer++; ; swap(x, y)) {
      if (x == -1) continue;
      if (aux[x] == timer) return x;
      aux[x] = timer;
      x = (mate[x] == -1 ? -1 : orig[parent[mate[x]]]);
    }
  };
  auto blossom = [&](int v, int w, int a) {
    while (orig[v] != a) {
      parent[v] = w; w = mate[v];
      if (label[w] == 1) label[w] = 0, q.push_back(w);
      orig[v] = orig[w] = a; v = parent[w];
    }
  };
  auto augment = [&](int v) {
    while (v != -1) {
      int pv = parent[v], nv = mate[pv];
      mate[v] = pv; mate[pv] = v; v = nv;
    }
  };
  auto bfs = [&](int root) {
    fill(label.begin(), label.end(), -1);
    iota(orig.begin(), orig.end(), 0);
    q.clear();
    label[root] = 0; q.push_back(root);
    for (int i = 0; i < (int)q.size(); ++i) {
      int v = q[i];
      for (auto x : graph[v]) {
        if (label[x] == -1) {
          label[x] = 1; parent[x] = v;
          if (mate[x] == -1)
            return augment(x), 1;
          label[mate[x]] = 0; q.push_back(mate[x]);
        } else if (label[x] == 0 && orig[v] != orig[x]) {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a); blossom(v, x, a);
        }
      }
    }
    return 0;
  };
  // Time halves if you start with (any) maximal matching.
  for (int i = 0; i < n; i++)
    if (mate[i] == -1)
      bfs(i);
  return mate;
}
```

MaximumMatchingRandom.cpp

Description: Randomized maximum matching.

<bits/stdc++.h>, std:: c68909, 19 lines

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());
int n,m,ct,a[505],mt[505],vs[505],lim=5;vector<int> v[505];
bool dfs(int x){
  vs[x]=1;shuffle(v[x].begin(),v[x].end(),rng);
  for(int i=0;i<v[x].size();i++){
    int y=v[x][i],z=mt[y];if(vs[z])continue;
    mt[x]=y,mt[y]=x,mt[z]=0;if(!z||dfs(z))return 1;mt[x]=0,mt[y
        ]=z,mt[z]=y;
  }return 0;
}
int main(){
  scanf("%d%d",&n,&m);
  while(m--){int x,y;scanf("%d%d",&x,&y);v[x].push_back(y);v[y
      ].push_back(x);}
  for(int t=1;t<=lim;t++){
    for(int i=1;i<=n;i++){if(!mt[i]){for(int j=1;j<=n;j++)vs[j
        ]=0;dfs(i);}}
    int tp=0;for(int i=1;i<=n;i++)tp+=bool(mt[i]);
    if(tp>ct){ct=tp;for(int i=1;i<=n;i++)a[i]=mt[i];}
  }printf("%d\n",ct/2);for(int i=1;i<=n;i++)printf("%d ",a[i]);
  return 0;
}
```

MaximumWeightMatching.cpp

Description: Blossom for weighted maximum matching.

<bits/stdc++.h>, std::, ll = long long 323ac4, 201 lines

```
//accept only complete graphs
#define cin kin
#define DIST(e) (lab[e.u]+lab[e.v]-g[e.u][e.v].w*2)
const int N=1023,INF=1e9;
struct Edge{
  int u,v,w;
} g[N][N];
int n,m,n_x,lab[N],match[N],slack[N],st[N],pa[N],flower_from[N
    ][N],S[N],vis[N];
vector<int> flower[N];
deque<int> q;
void update_slack(int u,int x){
  if(!slack[x]||DIST(g[u][x])<DIST(g[slack[x]][x]))slack[x]=u;
}
void set_slack(int x){
  slack[x]=0;
  for(int u=1; u<=n; ++u)
    if(g[u][x].w>0&&st[u]!=x&&S[st[u]]==0)update_slack(u,x);
}
void q_push(int x){
  if(x<=n)return q.push_back(x);
  for(int i=0; i<flower[x].size(); i++)q_push(flower[x][i]);
}
void set_st(int x,int b){
  st[x]=b;
  if(x<=n)return;
  for(int i=0; i<flower[x].size(); ++i)set_st(flower[x][i],b);
}
int get_pr(int b,int xr){
  int pr=find(flower[b].begin(),flower[b].end(),xr)-flower[b].
      begin();
  if(pr%2==1){
    reverse(flower[b].begin()+1,flower[b].end());
    return (int)flower[b].size()-pr;
  }
  else return pr;
}
void set_match(int u,int v){
  match[u]=g[u][v].v;
  if(u<=n)return;
  Edge e=g[u][v];
  int xr=flower_from[u][e.u],pr=get_pr(u,xr);
  for(int i=0; i<pr; ++i)set_match(flower[u][i],flower[u][i^1])
      ;
  set_match(xr,v);
  rotate(flower[u].begin(),flower[u].begin()+pr,flower[u].end()
      );
}
void augment(int u,int v){
  int xnv=st[match[u]];
  set_match(u,v);
  if(!xnv)return;
  set_match(xnv,st[pa[xnv]]);
  augment(st[pa[xnv]],xnv);
}
int get_lca(int u,int v){
  static int t=0;
  for(++t; u||v; swap(u,v)){
    if(u==0)continue;
    if(vis[u]==t)return u;
    vis[u]=t;
    u=st[match[u]];
    if(u)u=st[pa[u]];
  }
  return 0;
}
void add_blossom(int u,int lca,int v){
  int b=n+1;
  while(b<=n_x&&st[b])++b;
  if(b>n_x)++n_x;
  lab[b]=0,S[b]=0;
  match[b]=match[lca];
  flower[b].clear();
  flower[b].push_back(lca);
  for(int x=u,y; x!=lca; x=st[pa[y]])
    flower[b].push_back(x),flower[b].push_back(y=st[match[x]]),
        q_push(y);
  reverse(flower[b].begin()+1,flower[b].end());
  for(int x=v,y; x!=lca; x=st[pa[y]])
    flower[b].push_back(x),flower[b].push_back(y=st[match[x]]),
        q_push(y);
  set_st(b,b);
  for(int x=1; x<=n_x; ++x)g[b][x].w=g[x][b].w=0;
  for(int x=1; x<=n; ++x)flower_from[b][x]=0;
  for(int i=0; i<flower[b].size(); ++i){
    int xs=flower[b][i];
    for(int x=1; x<=n_x; ++x)
      if(g[b][x].w==0||DIST(g[xs][x])<DIST(g[b][x]))
        g[b][x]=g[xs][x],g[x][b]=g[x][xs];
    for(int x=1; x<=n; ++x)
      if(flower_from[xs][x])flower_from[b][x]=xs;
  }
  set_slack(b);
}
void expand_blossom(int b)  // S[b] == 1
{
  for(int i=0; i<flower[b].size(); ++i)
    set_st(flower[b][i],flower[b][i]);
  int xr=flower_from[b][g[b][pa[b]].u],pr=get_pr(b,xr);
  for(int i=0; i<pr; i+=2){
    int xs=flower[b][i],xns=flower[b][i+1];
    pa[xs]=g[xns][xs].u;
    S[xs]=1,S[xns]=0;
    slack[xs]=0,set_slack(xns);
    q_push(xns);
  }
  S[xr]=1,pa[xr]=pa[b];
  for(int i=pr+1; i<flower[b].size(); ++i){
    int xs=flower[b][i];
    S[xs]=-1,set_slack(xs);
  }
  st[b]=0;
}
bool on_found_Edge(const Edge &e){
  int u=st[e.u],v=st[e.v];
  if(S[v]==-1){
    pa[v]=e.u,S[v]=1;
    int nu=st[match[v]];
    slack[v]=slack[nu]=0;
    S[nu]=0,q_push(nu);
  }
  else if(S[v]==0){
    int lca=get_lca(u,v);
    if(!lca)return augment(u,v),augment(v,u),1;
    else add_blossom(u,lca,v);
  }
  return 0;
}
bool matching(){
  fill(S,S+n_x+1,-1),fill(slack,slack+n_x+1,0);
  q.clear();
  for(int x=1; x<=n_x; ++x)
    if(st[x]==x&&!match[x])pa[x]=0,S[x]=0,q_push(x);
  if(q.empty())return 0;
  for(;;){
    while(q.size()){
      int u=q.front();
      q.pop_front();
      if(S[st[u]]==1)continue;
      for(int v=1; v<=n; ++v)
        if(g[u][v].w>0&&st[u]!=st[v]){
          if(DIST(g[u][v])==0){
            if(on_found_Edge(g[u][v]))return 1;
          }
          else update_slack(u,st[v]);
        }
    }
    int d=INF;
    for(int b=n+1; b<=n_x; ++b)
      if(st[b]==b&&S[b]==1)d=min(d,lab[b]/2);
    for(int x=1; x<=n_x; ++x)
      if(st[x]==x&&slack[x]){
        if(S[x]==-1)d=min(d,DIST(g[slack[x]][x]));
        else if(S[x]==0)d=min(d,DIST(g[slack[x]][x])/2);
      }
    for(int u=1; u<=n; ++u){
      if(S[st[u]]==0){
        if(lab[u]<=d)return 0;
        lab[u]-=d;
      }
      else if(S[st[u]]==1)lab[u]+=d;
    }
    for(int b=n+1; b<=n_x; ++b)
      if(st[b]==b){
        if(S[st[b]]==0)lab[b]+=d*2;
        else if(S[st[b]]==1)lab[b]-=d*2;
      }
    q.clear();
    for(int x=1; x<=n_x; ++x)
      if(st[x]==x&&slack[x]&&st[slack[x]]!=x&&DIST(g[slack[x]][
          x])==0)
        if(on_found_Edge(g[slack[x]][x]))return 1;
    for(int b=n+1; b<=n_x; ++b)
      if(st[b]==b&&S[b]==1&&lab[b]==0)expand_blossom(b);
  }
  return 0;
}
pair<ll,int> weight_blossom(){
  fill(match,match+n+1,0);
  n_x=n;
  int n_matches=0;
  ll tot_weight=0;
  for(int u=0; u<=n; ++u)st[u]=u,flower[u].clear();
  int w_max=0;
  for(int u=1; u<=n; ++u)
    for(int v=1; v<=n; ++v){
      flower_from[u][v]=(u==v?u:0);
      w_max=max(w_max,g[u][v].w);
    }
  for(int u=1; u<=n; ++u)lab[u]=w_max;
  while(matching())++n_matches;
  for(int u=1; u<=n; ++u)
    if(match[u]&&match[u]<u)
      tot_weight+=g[u][match[u]].w;
  return make_pair(tot_weight,n_matches);
}
int main(){
  cin>>n>>m;
  for(int u=1; u<=n; ++u)
    for(int v=1; v<=n; ++v)
      g[u][v]=Edge {u,v,0};
  for(int i=0,u,v,w; i<m; ++i){
    cin>>u>>v>>w;
    g[u][v].w=g[v][u].w=w;
  }
  cout<<weight_blossom().first<<'\n';
  for(int u=1; u<=n; ++u)cout<<match[u]<<' ';
}
```

hopcroftKarp.h

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

f612e4, 42 lines

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

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

DFSMatching.h

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

522b98, 22 lines

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

MinimumVertexCover.h

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

"DFSMatching.h" da4196, 20 lines

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

WeightedMatching.h

Description: Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes cost[N][M], where cost[i][j] = cost for L[i] to be matched with R[j] and returns (min cost, match), where L[i] is matched with R[match[i]]. Negate costs for max cost. Requires $N \leq M$.
Time: $\mathcal{O}\left(N^2M\right)$

1e0fe9, 31 lines

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

GeneralMatching.h

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

"../numerical/MatrixInverse-mod.h" cb1912, 40 lines

```
vector<pii> generalMatching(int N, vector<pii>& ed) {
  vector<vector<ll>> mat(N, vector<ll>(N)), A;
  for (pii pa : ed) {
    int a = pa.first, b = pa.second, r = rand() % mod;
    mat[a][b] = r, mat[b][a] = (mod - r) % mod;
  }

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

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

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

MinimumAverageCycle.cpp

Description: Find the directed cycle with minimum average. d343b9, 19 lines

```
const int N=3005,M=10005;
const double inf=1e18;
int n,m,i,j,u[M],v[M];double w[M],f[N][N],ans=1e9,now,tmp;
inline void up(double&a,double b){a>b?(a=b):0;}
int main(){
  scanf("%d%d",&n,&m);
  for(i=1;i<=m;i++)scanf("%d%d%lf",&u[i],&v[i],&w[i]);
  for(i=1;i<=n;i++)for(j=1;j<=n;j++)f[i][j]=inf;
  for(i=0;i<n;i++)for(j=1;j<=m;j++)up(f[i+1][v[j]],f[i][u[j]]+w
      [j]);
  for(i=1;i<=n;i++)if(f[n][i]<inf/2){
    now=-1e9;
    for(j=0;j<n;j++)if(f[j][i]<inf/2){
      tmp=1.0*(f[n][i]-f[j][i])/(n-j);
      if(now<tmp)now=tmp;
    }
    up(ans,now);
  }
  printf("%.8f",ans);
}
```

7.3 DFS algorithms

DominatorTree.cpp

Description: Building dominator tree.

<cstdio> 9b5715, 41 lines

```
// Assuming S can reach all graph (flowgraph), dom tree with
     source S
// dfn[x]: x's DFS order
// id[x]: x-th in DFS order
// gd[x]: list of childs for x-th in DFS order
// idom[x]: parent in dom tree for x-th in DFS order
// sd[x]: (semi-)dominator for x-th in DFS order
// id[idom[dfn[x]]]: closest dominator of x
const int N=5010,M=200010;
int n,m,i,x,y,q[N],ans;
int g1[N],g2[N],gd[N],v[M*3+N],nxt[M*3+N],ed;
int cnt,dfn[N],id[N],fa[N],f[N],mn[N],sd[N],idom[N];
void add(int*g,int x,int y){v[++ed]=y;nxt[ed]=g[x];g[x]=ed;}
int F(int x){
  if(f[x]==x)return x;
  int y=F(f[x]);
  if(sd[mn[x]]>sd[mn[f[x]]])mn[x]=mn[f[x]];
  return f[x]=y;
}
void dfs(int x){
  id[dfn[x]=++cnt]=x;
  for(int i=g1[x];i;i=nxt[i])if(!dfn[v[i]])dfs(v[i]),fa[dfn[v[i
      ]]]=dfn[x];
}
void tarjan(int S){
  int i,j,k,x;
  for(cnt=0,i=1;i<=n;i++)gd[i]=dfn[i]=id[i]=fa[i]=idom[i]=0,f[i
      ]=sd[i]=mn[i]=i;
  dfs(S);
  for(i=n;i>1;i--){
    for(j=g2[id[i]];j;j=nxt[j])F(k=dfn[v[j]]),sd[i]=sd[i]<sd[mn
        [k]]?sd[i]:sd[mn[k]];
    add(gd,sd[i],i);
    for(j=gd[f[i]=x=fa[i]];j;j=nxt[j])F(k=v[j]),idom[k]=sd[mn[k
        ]]<x?mn[k]:x;
    gd[x]=0;
  }
  for(i=2;i<=n;add(gd,idom[i],i),i++)if(idom[i]!=sd[i])idom[i]=
      idom[idom[i]];
}
int main(){
  while(~scanf("%d%d",&n,&m)){
    for(ed=0,i=1;i<=n;i++)g1[i]=g2[i]=0;
    while(m--)scanf("%d%d",&x,&y),add(g1,x,y),add(g2,y,x);
    tarjan(1);
  }
}
```

SCC.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.
Usage: `scc(graph, [&](vi& v) { ... })` visits all components in reverse topological order. `comp[i]` holds the component index of a node (a component only has edges to components with lower index). `ncomps` will contain the number of components.
Time: $\mathcal{O}(E+V)$

76b5c9, 24 lines

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

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

BiconnectedComponents.h

Description: Finds all biconnected components in an undirected graph, and runs a callback for the edges in each. In a biconnected component there are at least two distinct paths between any two nodes. Note that a node can be in several components. An edge which is not in a component is a bridge, i.e., not part of any cycle.
Usage:
```
int eid = 0; ed.resize(N);
for each edge (a,b) {
ed[a].emplace_back(b, eid);
ed[b].emplace_back(a, eid++); }
bicomps([&](const vi& edgelist) {...});
```
Time: $\mathcal{O}(E+V)$

2965e5, 33 lines

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

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

EulerWalk.h

Description: Eulerian undirected/directed path/cycle algorithm. Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns a list of nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, add .second to s and ret.
Time: $\mathcal{O}(V+E)$

780b64, 15 lines

```
vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src=0) {
  int n = sz(gr);
  vi D(n), its(n), eu(nedges), ret, s = {src};
  D[src]++; // to allow Euler paths, not just cycles
  while (!s.empty()) {
    int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
    if (it == end){ ret.push_back(x); s.pop_back(); continue; }
    tie(y, e) = gr[x][it++];
    if (!eu[e]) {
      D[x]--, D[y]++;
      eu[e] = 1; s.push_back(y);
    }}
  for (int x : D) if (x < 0 || sz(ret) != nedges+1) return {};
  return {ret.rbegin(), ret.rend()};
}
```

7.4 Coloring

ChordalGraphOrder.cpp

Description: Ordering for chordal graph.

66d260, 8 lines

```
int i,j,k,col[N],ans;bool vis[N];
int main(){
  for(i=n;i;i--){
    for(k=0,j=1;j<=n;j++)if(!vis[j]&&col[j]>=col[k])k=j; //
        optimize this
    for(vis[k]=1,j=g[k];j;j=nxt[j])if(!vis[v[j]])if(++col[v[j
        ]]>ans)ans=col[v[j]];
  }
  printf("%d",ans+1);
}
```

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, 31 lines

```
vi edgeColoring(int N, vector<pii> eds) {
  vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
  for (pii e : eds) ++cc[e.first], ++cc[e.second];
  int u, v, ncols = *max_element(all(cc)) + 1;
  vector<vi> adj(N, vi(ncols, -1));
  for (pii e : eds) {
    tie(u, v) = e;
    fan[0] = v;
    loc.assign(ncols, 0);
    int at = u, end = u, d, c = free[u], ind = 0, i = 0;
    while (d = free[v], !loc[d] && (v = adj[u][d]) != -1)
      loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
    cc[loc[d]] = c;
    for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at][cd])
      swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
    while (adj[fan[i]][d] != -1) {
      int left = fan[i], right = fan[++i], e = cc[i];
      adj[u][e] = left;
      adj[left][e] = u;
      adj[right][e] = -1;
      free[right] = e;
    }
    adj[u][d] = fan[i];
    adj[fan[i]][d] = u;
    for (int y : {fan[0], u, end})
      for (int& z = free[y] = 0; adj[y][z] != -1; z++);
  }
  rep(i,0,sz(eds))
    for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i];
  return ret;
}
```

7.5 Heuristics

MaximalCliques.h

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

B = bitset<128> b63b43, 11 lines

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

7.6 Trees

CompressTree.h

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

"LCA.h", vpi = vector<pair<int, int>> 83c9a2, 20 lines

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

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, 60 lines

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

pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
  RollbackUF uf(n);
  vector<Node*> heap(n);
  for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e});
  ll res = 0;
  vi seen(n, -1), path(n), par(n);
  seen[r] = r;
  vector<Edge> Q(n), in(n, {-1,-1}), comp;
  deque<tuple<int, int, vector<Edge>>> cycs;
  rep(s,0,n) {
    int u = s, qi = 0, w;
    while (seen[u] < 0) {
      if (!heap[u]) return {-1,{}};
      Edge e = heap[u]->top();
      heap[u]->delta -= e.w, pop(heap[u]);
      Q[qi] = e, path[qi++] = u, seen[u] = s;
      res += e.w, u = uf.find(e.a);
      if (seen[u] == s) {
        Node* cyc = 0;
        int end = qi, time = uf.time();
        do cyc = merge(cyc, heap[w = path[--qi]]);
        while (uf.join(u, w));
        u = uf.find(u), heap[u] = cyc, seen[u] = -1;
        cycs.push_front({u, time, {&Q[qi], &Q[end]}});
      }
    }
    rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
  }

  for (auto& [u,t,comp] : cycs) { // restore sol (optional)
    uf.rollback(t);
    Edge inEdge = in[u];
    for (auto& e : comp) in[uf.find(e.b)] = e;
    in[uf.find(inEdge.b)] = inEdge;
  }
  rep(i,0,n) par[i] = in[i].a;
  return {res, par};
}
```

7.7 Math

7.7.1 Number of Spanning Trees

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

7.7.2 Erdős–Gallai theorem

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

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

Geometry (8)

2DGeometry.cpp

Description: Basics of 2D geometry

<bits/stdc++.h>, std::, ld = long double 83be95, 128 lines

```
const ld PI = acos(-1);
const ld EPS = 1e-12;
int sgn(ld x) {
  if (x > EPS) return 1;
  else if (x < -EPS) return -1;
  return 0;
}
struct point {
  ld x, y;
  point(ld x = 0, ld y = 0) : x(x), y(y) {}
  point operator+(const point& o) const {
    return point(x + o.x, y + o.y);
  }
  point operator-(const point& o) const {
    return point(x - o.x, y - o.y);
  }
  point operator*(const ld& o) const {
    return point(x * o, y * o);
  }
  point operator/(const ld& o) const {
    return point(x / o, y / o);
  }
  ld operator*(const point& o) const {
    return x * o.y - y * o.x;
  }
  ld operator^(const point& o) const {
    return x * o.x + y * o.y;
  }
  point rotate() const {
    return point(-y, x);
  }
  point unit() const {
    ld w = abs();
    return point(x / w, y / w);
  }
  int get() const {
    return sgn(y) == 1 || (sgn(y) == 0 && sgn(x) == -1);
  }
  ld abs() const {
    return sqrt(x * x + y * y);
  }
};
struct line {
  point p0, p1;
  line(point p0, point p1) : p0(p0), p1(p1) {
  }
  bool contain(point p) const {
    return sgn((p1 - p0) * (p - p0)) == 1;
  }
  line move(ld d) const {
    point delta = (p1 - p0).rotate().unit() * d;
    return line(p0 + delta, p1 + delta);
  }
};
struct pointID {
  point p;
  int id;
  pointID(point p, int id) : p(p), id(id) {
  }
  bool operator<(const pointID& o) const {
    return make_pair(p.x, p.y) < make_pair(o.p.x, o.p.y);
  }
};
bool compare_angle(point p0, point p1) {
  return p0.get() < p1.get() || (p0.get() == p1.get() && sgn(p0
       * p1) > 0);
}
point line_intersection(point p0, point p1, point p2, point p3)
     {
  ld foo = (p0 - p2) * (p3 - p2);
  ld bar = (p3 - p2) * (p1 - p2);
  return (p0 * bar + p1 * foo) / (foo + bar);
}
point line_intersection(line l0, line l1) {
  return line_intersection(l0.p0, l0.p1, l1.p0, l1.p1);
}
bool parallel(line l0, line l1) {
  return sgn((l0.p1 - l0.p0) * (l1.p1 - l1.p0)) == 0;
}
bool same_direction(line l0, line l1) {
  return parallel(l0, l1) && sgn((l0.p1 - l0.p0) ^ (l1.p1 - l1.
      p0)) == 1;
}
bool operator<(const line& l0, const line& l1) {
  if (same_direction(l0, l1))
    return l1.contain(l0.p0);
  else
    return compare_angle(l0.p1 - l0.p0, l1.p1 - l1.p0);
}
bool check(line l0, line l1, line l2) {
  return l2.contain(line_intersection(l0, l1));
}
vector<line> halfplane_intersection(vector<line> l) {
  sort(l.begin(), l.end());
  deque<line> q;
  for (int i = 0; i < (int) l.size(); ++i) {
    if (i && same_direction(l[i], l[i - 1]))
      continue;
    while ((int) q.size() > 1 && !check(q[q.size() - 2], q[q.
        size() - 1], l[i]))
      q.pop_back();
    while ((int) q.size() > 1 && !check(q[1], q[0], l[i]))
      q.pop_front();
    q.push_back(l[i]);
  }
  while ((int) q.size() > 2 && !check(q[q.size() - 2], q[q.size
      () - 1], q[0]))
    q.pop_back();
  while ((int) q.size() > 2 && !check(q[1], q[0], q[q.size() -
      1]))
    q.pop_front();
  if ((int) q.size() == 2)
    return vector<line>();
  vector<line> res;
  for (int i = 0; i < (int) q.size(); ++i)
    res.push_back(q[i]);
  return res;
}
vector<pointID> convex_hull(vector<pointID> pts) {
  sort(pts.begin(), pts.end());
  if(pts.size()<=1) return pts;
  vector<pointID> ch;
  for (int rot = 0; rot < 2; ++rot) {
    int lim = ch.size();
    for (pointID p : pts) {
      while ((int) ch.size() > lim + 1 && (ch[ch.size() - 1].p
          - ch[ch.size() - 2].p) * (p.p - ch[ch.size() - 2].p)
           <= 0)
        ch.pop_back();
      ch.push_back(p);
    }
    ch.pop_back();
    reverse(pts.begin(), pts.end());
  }
  return ch;
}
```

8.1 Geometric primitives

Point.h

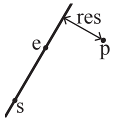
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

47ec0a, 28 lines

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

lineDistance.h

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

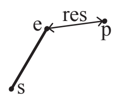


"Point.h" f6bf6b, 4 lines

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

SegmentDistance.h

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



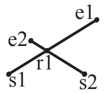
Usage: Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;

"Point.h", P = Point<double> ae751a, 5 lines

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

SegmentIntersection.h

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



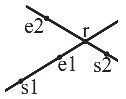
Usage: vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
cout << "segments intersect at " << inter[0] << endl;

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

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

lineIntersection.h

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



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

"Point.h" a01f81, 8 lines

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

sideOf.h

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

"Point.h" 3af81c, 9 lines

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

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

OnSegment.h

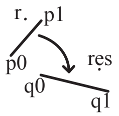
Description: Returns true iff p lies on the line segment from s to e. Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.

"Point.h" c597e8, 3 lines

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

linearTransformation.h

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



"Point.h", P = Point<double> 096877, 5 lines

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

Angle.h

Description: A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.
Usage: vector<Angle> v = {w[0], w[0].t360() ...}; // sorted
int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }
// sweeps j such that (j-i) represents the number of positively oriented triangles with vertices at 0 and i

0f0602, 35 lines

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

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

SVG.raw

11 lines

```
<?xml version="1.0" standalone="no"?>
<svg width="200" height="250" version="1.1" xmlns="http://www.
    w3.org/2000/svg">
  <rect x="10" y="10" width="30" height="30" stroke="black"
      fill="transparent" stroke-width="5"/>
  <circle cx="25" cy="75" r="1" stroke="red" fill="transparent"
       stroke-width="5"/>
  <ellipse cx="75" cy="75" rx="20" ry="5" stroke="red" fill="
      transparent" stroke-width="5"/>
  <line x1="10" x2="50" y1="110" y2="150" stroke="orange"
      stroke-width="5"/>
  <polyline points="60 110 65 120 70 115 75 130 80 125 85 140
      90 135 95 150 100 145"
      stroke="orange" fill="transparent" stroke-width="5"/>
  <polygon points="50 160 55 180 70 180 60 190 65 205 50 195 35
      205 40 190 30 180 45 180"
      stroke="green" fill="transparent" stroke-width="5"/>
</svg>
```

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", P = Point<double> c64785, 10 lines

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

CircleTangents.h

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

"Point.h" b0153d, 13 lines

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

CirclePolygonIntersection.h

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

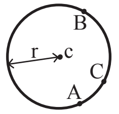
Time: $\mathcal{O}(n)$

"../../content/geometry/Point.h", P = Point<double> 11e475, 18 lines

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

circumcircle.h

Description:



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

"Point.h", P = Point<double> c98102, 8 lines

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

MinimumEnclosingCircle.h

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

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

"circumcircle.h" 09dd0a, 17 lines

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

8.3 Polygons

InsidePolygon.h

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

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

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

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

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

PolygonArea.h

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

"Point.h" f12300, 6 lines

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

PolygonCenter.h

Description: Returns the center of mass for a polygon.

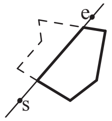
Time: $\mathcal{O}(n)$

"Point.h", P = Point<double> 0d0d84, 8 lines

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

PolygonCut.h

Description:



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

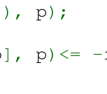
Usage: `vector<P> p = ...;`
`p = polygonCut(p, P(0,0), P(1,0));`

"Point.h", "lineIntersection.h", P = Point<double> af7b53, 12 lines

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

ConvexHull.h

Description:



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

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

"Point.h", P = Point<ll> ec85f8, 12 lines

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

HullDiameter.h

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

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

"Point.h", P = Point<ll> 5f726b, 11 lines

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

PointInsideHull.h

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

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

"Point.h", "sideOf.h", "OnSegment.h", P = Point<ll> c74639, 12 lines

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

LineHullIntersection.h

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

"Point.h" 7cf45b, 39 lines

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

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

8.4 Misc. Point Set Problems

DelaunayTriangulation.h

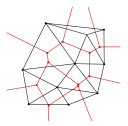
Description: Computes the Delaunay triangulation of a set of points. Each circumcircle contains none of the input points. If any three points are collinear or any four are on the same circle, behavior is undefined.
Time: $\mathcal{O}(n^2)$

"Point.h", "3dHull.h" c0e7bc, 10 lines

```
template<class P, class F>
void delaunay(vector<P>& ps, F trifun) {
  if (sz(ps) == 3) { int d = (ps[0].cross(ps[1], ps[2]) < 0);
    trifun(0,1+d,2-d); }
  vector<P3> p3;
  for (P p : ps) p3.emplace_back(p.x, p.y, p.dist2());
  if (sz(ps) > 3) for(auto t:hull3d(p3)) if ((p3[t.b]-p3[t.a]).
      cross(p3[t.c]-p3[t.a]).dot(P3(0,0,1)) < 0)
    trifun(t.a, t.c, t.b);
}
```

FastDelaunay.h

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



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

"Point.h", P = Point<ll>, Q = struct Quad* 416ecd, 81 lines

```
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point
struct Quad {
  Q rot, o; P p = arb; bool mark;
  P& F() { return r()->p; }
  Q& r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return r()->prev(); }
  Quad(Q s=0) {rot=s; o=0; p=arb; mark=0;}
} *H;
bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
  lll p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}
Q makeEdge(P orig, P dest) {
  Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
  H = r->o; r->r()->r() = r;
  rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
  r->p = orig; r->F() = dest;
  return r;
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}
pair<Q,Q> rec(const vector<P>& s) {
  if (sz(s) <= 3) {
    Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
    if (sz(s) == 2) return { a, a->r() };
    splice(a->r(), b);
    auto side = s[0].cross(s[1], s[2]);
    Q c = side ? connect(b, a) : 0;
    return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
  }
#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
  Q A, B, ra, rb;
  int half = sz(s) / 2;
  tie(ra, A) = rec({all(s) - half});
  tie(B, rb) = rec({sz(s) - half + all(s)});
  while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
         (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
  Q base = connect(B->r(), A);
  if (A->p == ra->p) ra = base->r();
  if (B->p == rb->p) rb = base;
#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
    while (circ(e->dir->F(), H(base), e->F())) { \
      Q t = e->dir; \
      splice(e, e->prev()); \
      splice(e->r(), e->r()->prev()); \
      e->o = H; H = e; e = t; \
    }
  for (;;) {
    DEL(LC, base->r(), o);  DEL(RC, base, prev());
    if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
      base = connect(RC, base->r());
    else
      base = connect(base->r(), LC->r());
  }
  return { ra, rb };
}
vector<P> triangulate(vector<P> pts) {
  sort(all(pts));  assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};
  Q e = rec(pts).first;
  vector<Q> q = {e};
  int qi = 0;
  while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
  q.push_back(c->r()); c = c->next(); } while (c != e); }
  ADD; pts.clear();
  while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
  return pts;
}
```

MinkowskiSum.cpp

Description: A random(bad) minkowski sum template found online.

<bits/stdc++.h>, std::, ld = long double, ll = long long, co = complex<ll> abbc5b, 69 lines

```
#define X real()
#define Y imag()
#define F first
#define S second
ll ccw(co a, co b, co c) {
  return ((c-a)*conj(b-a)).Y;
}
int ar(co x) {
  if (x.Y>=0&&x.X<0) return 1;
  if (x.X>=0&&x.Y>0) return 2;
  if (x.Y<=0&&x.X>0) return 3;
  return 4;
}
bool cp(pair<co, pair<int, int> > p1, pair<co, pair<int, int> >
    p2) {
  if (ar(p1.F)!=ar(p2.F))
    return ar(p1.F)<ar(p2.F);
  assert((ccw({0, 0}, p1.F, p2.F)==0)==(ccw({0, 0}, p2.F, p1.F)
      ==0));
  if (ccw({0, 0}, p1.F, p2.F)==0)
    return p1.S>p2.S;
  return ccw({0, 0}, p2.F, p1.F)>0;
}
vector<co> minkowski(vector<co>& a, vector<co>& b) {
  int n=a.size();
  int m=b.size();
  if (n==0) return b;
  if (m==0) return a;
  if (n==1) {
    vector<co> ret(m);
    for (int i=0;i<m;i++) ret[i]=b[i]+a[0];
    return ret;
  }
  if (m==1) {
    vector<co> ret(n);
    for (int i=0;i<n;i++) ret[i]=a[i]+b[0];
    return ret;
  }
  vector<pair<co, pair<int, int> > > pp;
  int f1=0,f2=0;
  for (int i=0;i<n;i++) {
    if (ccw(a[(i-1+n)%n], a[i], a[(i+1)%n])!=0) {
      f1=i; break;
    }
  }
  for (int i=0;i<n;i++)
    pp.push_back({a[(i+1+f1)%n]-a[(i+f1)%n], {1, i}});
  for (int i=0;i<m;i++) {
    if (ccw(b[(i-1+m)%m], b[i], b[(i+1)%m])!=0) {
      f2=i;break;
    }
  }
  for (int i=0;i<m;i++)
    pp.push_back({b[(i+1+f2)%m]-b[(i+f2)%m], {2, i}});
  sort(pp.rbegin(), pp.rend(), cp);
  co s={0, 0},ad={0, 0};
  for (int i=0;i<(int)pp.size();i++) {
    s+=pp[i].F;
    if (pp[i].S.F!=pp[i+1].S.F) {
      if (pp[i].S.F==1) ad=a[(pp[i].S.S+1+f1)%n]+b[(pp[i+1].S.S
          +f2)%m];
      else ad=b[(pp[i].S.S+1+f2)%m]+a[(pp[i+1].S.S+f1)%n];
      ad-=s;break;
    }
  }
  s=ad;
  vector<co> ret(pp.size());
  for (int i=0;i<(int)pp.size();i++) {
    ret[i]=s;s+=pp[i].F;
  }
  return ret;
}
```

8.5 3D

PolyhedronVolume.h

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

3058c3, 6 lines

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

Point3D.h

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

8058ae, 32 lines

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

3dHull.h

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

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

"Point3D.h", P3 = Point3D<double> b8fb93, 47 lines

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

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

vector<F> hull3d(const vector<P3>& A) {
  assert(sz(A) >= 4);
  vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
  vector<F> FS;
  auto mf = [&](int i, int j, int k, int l) {
    P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
    if (q.dot(A[l]) > q.dot(A[i]))
      q = q * -1;
    F f{q, i, j, k};
    E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
    FS.push_back(f);
  };
  rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
    mf(i, j, k, 6 - i - j - k);

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

sphericalDistance.h

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

611f07, 8 lines

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

3DConvexHull.cpp

Description: 3d convex hull in $O(n^2 logn)$. First 4 points should not be coplanar.

f6e03c, 56 lines

```
struct v3 {
  ld x,y,z;
  v3() : x(0), y(0), z(0) {}
  v3(ld xx, ld yy, ld zz) : x(xx), y(yy), z(zz) {};
};
v3 operator+(v3 a, v3 b) {
  return v3(a.x+b.x, a.y+b.y, a.z+b.z);
}
v3 operator-(v3 a, v3 b) {
  return v3(a.x-b.x, a.y-b.y, a.z-b.z);
}
v3 operator*(v3 a, v3 b) {
  return v3(a.y*b.z-a.z*b.y,a.z*b.x-a.x*b.z,a.x*b.y-a.y*b.x);
}
ld operator^(v3 a, v3 b) {
  return a.x*b.x+a.y*b.y+a.z*b.z;
}
bool onSameHalfSpace(v3 a, v3 b, v3 p1, v3 p2, v3 p3) {
  v3 hlp = (p2-p1)*(p3-p1);
  return (hlp^(a-p1))*(hlp^(b-p1))>=0;
}
int n;
v3 pts[1010];
map<pair<pair<int,int>,int>,int> convHull;
void toggleHull(int i1, int i2, int i3, int ref) {
  if (convHull.count({{i1,i2},i3})) {
    convHull.erase({{i1,i2},i3});
  } else convHull[{{i1,i2},i3}]=ref;
}
void makeHull() {
  convHull[{{0,1},2}]=3;
  convHull[{{0,1},3}]=2;
  convHull[{{0,2},3}]=1;
  convHull[{{1,2},3}]=0;
  for (int i=4;i<n;i++) {
    vector<pair<pair<int,int>,pair<int,int>>> toChange;
    for (auto hullFace : convHull) {
      int i1=hullFace.F.F.F;
      int i2=hullFace.F.F.S;
      int i3=hullFace.F.S;
      v3 pt1=pts[i1];
      v3 pt2=pts[i2];
      v3 pt3=pts[i3];
      v3 ref=pts[hullFace.S];
      if (!onSameHalfSpace(pts[i],ref,pt1,pt2,pt3)) {
        toChange.push_back({{i1,i2},{i,i3}});
        toChange.push_back({{i1,i3},{i,i2}});
        toChange.push_back({{i2,i3},{i,i1}});
        toChange.push_back({{i1,i2},{i3,i}});
      }
    }
    for (auto diff : toChange) {
      toggleHull(diff.F.F,diff.F.S,diff.S.F, diff.S.S);
    }
  }
}
```

Strings (9)

Zfunc.h

Description: z[x] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)
Time: $\mathcal{O}(n)$

ee09e2, 12 lines

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

Manacher.h

Description: For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest odd (half rounded down).
Time: $\mathcal{O}(N)$

e7ad79, 13 lines

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

MinRotation.h

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

d07a42, 8 lines

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

LyndonRuns.cpp

Description: A string is called *simple* (or a Lyndon word), if it is strictly smaller than any of its own nontrivial suffixes. It can be shown that a string is simple, if and only if it is strictly smaller than all its nontrivial cyclic shifts. The Lyndon factorization of the string s is a factorization $s = w_1 w_2 \dots w_k$, where all strings w_i are simple, and they are in non-increasing order $w_1 \geq w_2 \geq \dots \geq w_k$. Gives lyndon for all the suffixes (op=1, $[i, ly_i]$).

<bits/stdc++.h>, std::, ull = uint64_t 3cb092, 70 lines

```
const ull base = 131;
const int N = 1e6 + 5, mod = 1e9 + 7;
ull pw[N], h[N], pw2[N], h2[N];
char s[N];
int n, ly[N], stk[N], tp, tot;
void init_hash(const char *s, int n) {
    pw[0] = 1;
    for(int i=1; i<=n; i++) {
        h[i] = (h[i-1]*base+s[i])%mod;
        pw[i] = pw[i-1]*base%mod;
    }
}
ull get(int l, int r) { return (h[r] - h[l-1]*pw[r-l+1]%mod+mod
    )%mod; }
int lcp(int x, int y) {
    int l = 1, r = n - max(x, y) + 1, ans = 0;
    while(l<=r) {
        int mid = (l+r)>>1;
        if(get(x, x+mid-1)==get(y, y+mid-1)) l = mid + 1, ans =
             mid;
        else r = mid - 1;
    }
    return ans;
}
int lcs(int x, int y) {
    int l = 1, r = min(x, y), ans = 0;
    while(l<=r) {
        int mid = (l+r)>>1;
        if(get(x-mid+1, x)==get(y-mid+1, y)) l = mid + 1, ans =
             mid;
        else r = mid - 1;
    }
    return ans;
}
struct runs {
    int l, r, p;
    bool operator < (const runs &oth) {
        return l==oth.l ? (r==oth.r ? p<oth.p : r<oth.r) : l<
            oth.l;
    }
}r[N<<1];
void check(int x, int y) {
    int L = lcs(x-1, y-1), R = lcp(x, y);
    if(L+R>=y-x) r[++tot] = {x-L, y+R-1, y-x};
}
bool cmp(int x, int y) {
    int len = lcp(x, y);
    return s[x+len]<s[y+len];
}
void lyndon(bool op) {
    ly[n] = n, stk[0] = n + 1, stk[tp=1] = n;
    for(int i=n-1; i>=1; i--) {
        while(tp && cmp(i, stk[tp])==op) --tp;
        stk[++tp] = i;
        ly[i] = stk[tp-1] - 1;
    }
}
int main() {
    scanf("%s", s+1);
    n = strlen(s+1);
    init_hash(s, n);
    for(int op=0; op<=1; op++) {
        lyndon(op);
        for(int i=1; i<n; i++) check(i, ly[i]+1);
    }
    sort(r+1, r+tot+1);
    int cnt = 0;
    for(int i=1; i<=tot; i++) {
        if(r[i].l!=r[i-1].l||r[i].r!=r[i-1].r) r[++cnt] = r[i];
    }
    printf("%d\n", cnt);
    for(int i=1; i<=cnt; i++) printf("%d %d %d\n", r[i].l, r[i
        ].r, r[i].p);
    return 0;
}
```

PalindromeTree.cpp

Description: Palindromic tree. Odd and even roots etc.

653931, 27 lines

```
struct PTree {
int ch[SZ][26],len[SZ],fail[SZ],cnt[SZ],s[SZ],cl,an,lst;
int addn(int l) {len[an]=l; return an++;}
PTree() {
  cl=an=lst=0;
  memset(ch,0,sizeof(ch));
  addn(0); addn(-1);
  fail[0]=1; s[0]=-233;
}
int gfail(int x,int l) {
  while(s[l-len[x]-1]!=s[l]) x=fail[x];
  return x;
}
void add(int c) {
  s[++cl]=c;
  int cp=gfail(lst,cl);
  if(!ch[cp][c]) {
    int nn=addn(len[cp]+2);
    fail[nn]=ch[gfail(fail[cp],cl)][c];
    ch[cp][c]=nn;
  }
  cnt[lst=ch[cp][c]]++;
}
void getcnt() {
  for(int i=an-1;i>=2;i--) cnt[fail[i]]+=cnt[i];
}
}pt;
```

SuffixAM.cpp

Description: Suffix automaton. To build upon a Trie use BFS.

35c15b, 37 lines

```
int N,ch[SZ][26],fa[SZ],ml[SZ];
void init() {N=0; fa[0]=-1; ml[0]=0;}
int ins(int p,int c) {
  int x=ch[p][c],y=0;
  if(!x) {
    x=++N;ml[x]=ml[p]+1;y=1;
    for(;p!=-1&&!ch[p][c];p=fa[p]) ch[p][c]=x;
  }
  if(p==-1) fa[x]=0;
  else if(ml[ch[p][c]]==ml[p]+1) y?(fa[x]=ch[p][c]):0;
  else {
    int q=++N,cc=ch[p][c];ml[q]=ml[p]+1;
    for(int i=0;i<26;i++) ch[q][i]=ch[cc][i];
    fa[q]=fa[cc],fa[cc]=fa[x]=q;
    for(;p!=-1&&ch[p][c]==cc;p=fa[p]) ch[p][c]=q;
    if(!y) return q;
  }
  return x;
}
```

```
int qs[SZ],od[SZ]; //od[0]..od[N]
void topo() {
  for(int i=0;i<=N;i++) qs[i]=0;
  for(int i=0;i<=N;i++) qs[ml[i]]++;
  for(int i=1;i<=N;++i) qs[i]+=qs[i-1];
  for(int i=N;i>=0;i--) od[--qs[ml[i]]]=i;
}
char s[SZ]; int ri[SZ];
int main(){
  init(); scanf("%s",s+1); int w=0;
  for(int j=1;s[j];j++) ++ri[w=ins(w,s[j]-'a')];
  topo(); ll ans=0;
  for(int i=N;i;--i) {
    int x=od[i]; ri[fa[x]]+=ri[x];
    if(ri[x]!=1) ans=max(ans,(ll)ri[x]*ml[x]);
  }
  printf("%lld\n",ans);
}
```

SuffixArray.cpp

Description: Suffix array, 0-indexed. lcp(a,b) equals minimum h between rk[a] and rk[b].

ebc8b2, 41 lines

```
int sa[SZ],rk[SZ],su[SZ],ork[SZ];
char s[SZ]; int n,h[SZ];
bool diff(int a,int b,int g)
{return ork[a]!=ork[b]||ork[a+g]!=ork[b+g];}
void build_sa() {
  int G=max(n,200);
  for(int i=0;i<=G;++i) su[i]=0;
  for(int i=0;i<n;++i) ++su[rk[i]=s[i]];
  for(int i=1;i<=G;++i) su[i]+=su[i-1];
  for(int i=0;i<n;++i) sa[--su[rk[i]]]=i;
  for(int g=1;g<=n;g<<=1) {
    for(int i=0;i<=G;++i) su[i]=0;
    static int ts[SZ]; int tn=0;
    for(int i=n-1;i>=0;--i) {
      if(sa[i]>=g) ts[++tn]=sa[i]-g;
      ork[i]=rk[i];
    }
    for(int i=n-g;i<n;++i) ts[++tn]=i;
    for(int i=1;i<=tn;++i) ++su[rk[ts[i]]];
    for(int i=1;i<=G;++i) su[i]+=su[i-1];
    for(int i=1;i<=tn;++i)
      sa[--su[rk[ts[i]]]]=ts[i];
    int t=0;
    for(int i=0;i<n;++i) {
      if(i&&diff(sa[i-1],sa[i],g)) ++t;
      rk[sa[i]]=t;
    }
  }
  for(int i=0;i<n;++i) rk[sa[i]]=i;
  int g=0;
  for(int i=0;i<n;++i) {
    g=max(g-1,0);
    if(rk[i]==n-1) {
      h[rk[i]]=0;
      continue;
    }
    int A=i,B=sa[rk[i]+1];
    while(s[A+g]==s[B+g]) ++g;
    h[rk[i]]=g;
  }
}
```

BasicSubstringStructure.cpp

Description: Basic substring structure. The triangle is partitioned into substairs and each group is of several identical substairs. Total perimeter of substairs is linear.

b57c91, 50 lines

```
template<int NN,int Z>
class SAM{
    static constexpr int N=NN*2;
public:
    int OO,tot,n,Lst,ch[N][Z],fa[N],len[N],pre[N],posl[N],posr[
         N],occ[N],lis[N],tcnt[N],tmp[N],_ffa[__lg(N)+1][N],ffa
         [N][__lg(N)+1]; vector<int> son[N];
    SAM(int OOO=0) : OO(OOO) { tot=Lst=1; }
    inline void push_back(int c){
        int np=++tot,p=Lst; Lst=np; occ[np]=1; len[np]=++n; pre
             [n]=np; posr[np]=n;
        while(p&&!ch[p][c]) ch[p][c]=np,p=fa[p];; if(!p) return
             fa[np]=1,void();
        int v=ch[p][c]; if(len[v]==len[p]+1) return fa[np]=v,
             void();
        int nv=++tot; memcpy(ch[nv],ch[v],sizeof(ch[nv])); fa[
             nv]=fa[v]; posr[nv]=posr[v]; len[nv]=len[p]+1; fa[
             v]=fa[np]=nv;
        while(p&&ch[p][c]==v) ch[p][c]=nv,p=fa[p];
    }
    inline void build(int oo=0){
        For(u,2,tot) _ffa[0][u]=fa[u];; For(i,1,__lg(n)) For(u
             ,2,tot) _ffa[i][u]=_ffa[i-1][_ffa[i-1][u]];
        For(u,1,tot) For(i,0,__lg(n)) ffa[u][i]=_ffa[i][u];
        For(u,2,tot) son[fa[u]].push_back(u);;
        function<void(int)> dfs; dfs = [&](int u) { for(int v:
             son[u]) dfs(v),posr[u]=max(posr[u],posr[v]); };
             if(oo) dfs(1);
        For(u,2,tot) posl[u]=posr[u]-len[u]+1;
        For(u,1,tot) tcnt[posl[u]]++;; For(i,1,n) tcnt[i]+=tcnt
             [i-1];; For(u,1,tot) tmp[tcnt[posl[u]]--]=u;; if(
             oo) reverse(tmp+1,tmp+1+tot);
        For(i,1,n) tcnt[i]=0;; For(i,1,tot) tcnt[len[tmp[i
             ]]]++;; For(i,1,n) tcnt[i]+=tcnt[i-1];; Rev(i,tot
             ,1) lis[tcnt[len[tmp[i]]]--]=tmp[i];
        Rev(i,tot,1) { int u=lis[i]; for(int v:son[u]) occ[u]+=
             occ[v]; }
    }
    inline int find(int l,int r) const { int u=pre[r]; Rev(i,
         __lg(n),0) if(len[ffa[u][i]]>=r-l+1) u=ffa[u][i];;
         return u; }
};
template<int NN,int Z>
class BasicStringStructure{
public:
    static constexpr int N=NN*2; struct Node { int u,r; };
         using pii = pair<int,int>; struct Stair { int posl,
         posr; vector<Node> X,Y; } st[N];
    int n,stcnt,num0[N],num1[N]; SAM<NN,Z> S0,S1;
    BasicStringStructure<NN,Z>() : S0(0), S1(1) {}
    inline void build(int _n,int* s){
        n=_n; For(i,1,n) S0.push_back(s[i]);; Rev(i,n,1) S1.
             push_back(s[i]);; fprintf(stderr,"T1:%ld\n",clock
             ()); S0.build(0); S1.build(1);
        stcnt=0; Rev(i,S0.tot,2){
            int u=S0.lis[i],l=S0.posl[u],r=S0.posr[u],l2=l+S0.
                 len[u]-S0.len[S0.fa[u]]-1,v=0; For(c,0,Z-1) if
                 (S0.ch[u][c]&&S0.occ[S0.ch[u][c]]==S0.occ[u])
                 { v=S0.ch[u][c]; break; }
            if(!v) {  num0[u]=++stcnt; st[stcnt].posl=l; st[
                 stcnt].posr=r; st[stcnt].X.push_back(Node{u,l2
                 }); }
            else { st[num0[u]=num0[v]].X.push_back(Node{u,l2});
                  }
        }
        stcnt=0; Rev(i,S1.tot,2){
            int u=S1.lis[i],l=n-S1.posr[u]+1,r=n-S1.posl[u]+1,
                 r2=r-(S1.len[u]-S1.len[S1.fa[u]])+1,v=0; For(c
                 ,0,Z-1) if(S1.ch[u][c]&&S1.occ[S1.ch[u][c]]==
                 S1.occ[u]) { v=S1.ch[u][c]; break; }
            if(!v) { num1[u]=++stcnt; assume(st[stcnt].posl==l
                 &&st[stcnt].posr==r); st[stcnt].Y.push_back(
                 Node{u,r2}); }
            else { st[num1[u]=num1[v]].Y.push_back(Node{u,r2});
                  }
        }
        // For(i,1,stcnt){
        //     printf("Stair #%d (%d,%d): X = ",i,st[i].posl,st
             [i].posr);
        //     for(auto x:st[i].X) printf("%d[%d,%d] ",x.u,x.l,
             x.r);
        //     printf(" Y = "); for(auto y:st[i].Y) printf("%d
             [%d,%d] ",y.u,y.l,y.r);; puts("");
        // }
    }
};
```

Various (10)

10.1 Intervals

IntervalContainer.h

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

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

edce47, 23 lines

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

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

IntervalCover.h

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

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

9e9d8d, 19 lines

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

10.2 Misc. algorithms

ExpressionEvaluation.cpp

Description: Expression evaluation sample.

2b0f18, 78 lines

```
bool delim(char c) {return c == ' ';}
bool is_op(char c) {
  return c == '+' || c == '-' || c == '*' || c == '/';
}
bool is_unary(char c) {
  return c == '+' || c == '-';
}
int priority (char op) {
  if (op < 0) // unary operator
    return 3;
  if (op == '+' || op == '-')
    return 1;
  if (op == '*' || op == '/')
    return 2;
  return -1;
}
void process_op(stack<int>& st, char op) {
  if (op < 0) {
    int l = st.top(); st.pop();
    switch (-op) {
      case '+': st.push(l); break;
      case '-': st.push(-l); break;
    }
  } else {
    int r = st.top(); st.pop();
    int l = st.top(); st.pop();
    switch (op) {
      case '+': st.push(l + r); break;
      case '-': st.push(l - r); break;
      case '*': st.push(l * r); break;
      case '/': st.push(l / r); break;
    }
  }
}
int evaluate(string s) {
  stack<int> st;
  stack<char> op;
  bool may_be_unary = true;
  for (int i = 0; i < (int)s.size(); i++) {
    if (delim(s[i])) continue;
    if (s[i] == '(') {
      op.push('('); may_be_unary = true;
    } else if (s[i] == ')') {
      while (op.top() != '(') {
        process_op(st, op.top());
        op.pop();
      }
      op.pop(); may_be_unary = false;
    } else if (is_op(s[i])) {
      char cur_op = s[i];
      if (may_be_unary && is_unary(cur_op))
        cur_op = -cur_op;
      while (!op.empty() && (
            (cur_op >= 0 && priority(op.top()) >= priority(
                cur_op)) ||
            (cur_op < 0 && priority(op.top()) > priority(cur_op
                ))
          )) {
        process_op(st, op.top()); op.pop();
      }
      op.push(cur_op);
      may_be_unary = true;
    } else {
      int number = 0;
      while (i < (int)s.size() && isalnum(s[i]))
        number = number * 10 + s[i++] - '0';
      --i;
      st.push(number);
      may_be_unary = false;
    }
  }
  while (!op.empty()) {
    process_op(st, op.top()); op.pop();
  }
  return st.top();
}
int32_t main() {
  cout << evaluate("12-3-(3 - 4)");
  return 0;
}
```

FastKnapsack.h

Description: Given N non-negative integer weights w and a non-negative target t, computes the maximum S <= t such that S is the sum of some subset of the weights.

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

b20ccc, 16 lines

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

Datetime.py

12 lines

```
# python datetime example
import datetime
d = datetime.datetime(2022, 12, 25)
print((d.year, d.month, d.day, d.hour, d.minute, d.second, d.
    microsecond))
d2 = datetime.datetime.fromtimestamp(1326244364)
dt = datetime.timedelta(days = 3, hours = 34, minutes = 56,
    seconds = 234, microseconds=1000001)
d = d + dt
print((d.year, d.month, d.day, d.hour, d.minute, d.second, d.
    microsecond))
print(d-d2) #<class 'datetime.timedelta'>, 4005 days,
    14:47:11.000001
print(d.strftime("%Y/%m/%d, %H:%M:%S"))
d3 = datetime.datetime.strptime("25 December, 2022", "%d %B, %Y
    ")
print(["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
    "Friday", "Saturday"][d3.weekday()])
```

ZellerCongruence

$$w = (\lfloor \frac{c}{4} \rfloor - 2c + y + \lfloor \frac{y}{4} \rfloor + \lfloor \frac{13(m+1)}{5} \rfloor + d - 1) \bmod 7$$

w: 0 sunday, 1 monday, 2 tuesday, 6 saturday. c: first two digits of year. y: last two digits of year. m: month where $3 \le m \le 14$: change mon $1, 2$ to mon $13, 14$ in the previous year! d: day. Modular result may be negative.

10.3 Debugging tricks

- `signal(SIGSEGV, [](int) { _Exit(0); });` converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). `_GLIBCXX_DEBUG` failures generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).
- `feenableexcept(29);` kills the program on NaNs (`1`), 0-divs (`4`), infinities (`8`) and denormals (`16`).

10.4 Optimization tricks

`__builtin_ia32_ldmxcsr(40896);` disables denormals (which make floats 20x slower near their minimum value).

10.4.1 Bit hacks

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

10.4.2 Pragmas

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