1 Template

1.1 Template

```cpp
// !!!!!!
// rename to template.cpp instead of main.cpp
#include <bits/stdc++.h>

#define fr first
#define sc second
#define all(a) (a).begin(), (a).end()

using namespace std;

#ifdef ONPC
mt19937 rnd(223);
#else
mt19937 rnd(chrono::high_resolution_clock::now()
    .time_since_epoch().count());
#endif

#define TIME (clock() * 1.0 / CLOCKS_PER_SEC)

#ifdef ONPC
#define show(x) cerr << "LINE " << __LINE__ << ": " << #x << "=" << x
#else
#define show(x) 42
#endif

using ll = long long;
using ld = double;

void solve() {
}

int32_t main() {
#ifdef ONPC
  freopen("../a.in", "r", stdin);
//  freopen("../a.out", "w", stdout);
#endif
  ios::sync_with_stdio(0); cin.tie(0);
  cout << fixed << setprecision(20);

  solve();
  fflush(stdout);

  cerr << "\n\nConsumed " << TIME << endl;
}

//CMAKE
cmake_minimum_required(VERSION 3.15)
project(proj)

set(CMAKE_CXX_STANDARD 17)

set(CMAKE_CXX_FLAGS "-DONPC -Wall -Wextra -Wshadow -Wfatal-errors -Wconversion
↪  -Wno-sign-compare -Wno-sign-conversion")
set(CMAKE_CXX_FLAGS_RELEASE "-O2")

set(CMAKE_CXX_FLAGS_DEBUG "-g -D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC
↪  -fsanitize=address,undefined")

add_executable(proj a.cpp)

// изменение шрифта: editor -> general -> allow to change font size with
↪  mouse+wheel -> all editors
// дальше ctrl+мышка
// норм подсветка: editor -> inspections -> убрать галочку с C++ и General
// languages -> C++ -> Clangd -> заменить флаги компиляции
// gdb run a.exe
// then where
// measure memory: alias mem='command time -f '\''%Mkb\n%es'\'''
// на пробном туре: узнать max source limit
```

2 Заметки

2.1 Количества деревьев

- Деревьев на n различных вершинах: n^{n-2}
- Остовных деревьев в полном двудольном графе: $m^{n-1}n^{m-1}$
- Лесов на n различных вершинах, в которых m деревьев, причем i-я вершина лежит в i-м дереве ($i \leqslant m$): mn^{n-m-1}
- Количество деревьев, если даны степени вершин: $\frac{(n-2)!}{(d[0]-1)!\cdot(d[1]-1)!\cdot\ldots\cdot(d[n-1]-1)!}$
- Количество деревьев, если уже есть связные компоненты: $n^{k-2}\prod a_i$

2.2 Разное

- Лемма Бернсайда: $|X/G| = \frac{1}{|G|}\sum\limits_{g\in G}|X^g|$. Строим группу действий G, для каждого действия считаем количество неподвижных элементов X (не забыть e), все складываем и делим на $|G|$.
- Теорема Пойа: $|[k]^X/G| = \frac{1}{|G|}\sum\limits_{\pi\in G}k^{C(\pi)}$, $C(\pi)$ - число циклов в перестановке π, k - число цветов у одного элемента перестановки (красим каждый элемент X в один из k цветов). G группа перестановок.
- Формула Пика: $S = I + \frac{B}{2} - 1$. Обобщение: $S = V - \frac{E}{2} + H - 1$, V - число целочисленных точек внутри и на границе, E - число ребер (соседних целых точек) на границе (считается дважды для

вырожденных ребер), H - число дыр в многоугольнике. В случае хорошего многоугольника $E = B$, $H = 0$.

- Обобщенные числа Каталана (количество строк из n X-ов и k Y-ов таких, что на каждом префиксе $cnt(Y) < cnt(X) + m$):
$C_m(n,k) = \begin{cases} \binom{n+k}{k}, & 0 \leqslant k < m \\ \binom{n+k}{k} - \binom{n+k}{k-m}, & m \leqslant k \leqslant n+m-1 \\ 0, & k > n+m-1 \end{cases}$
Работает при $m \geqslant 1$, при $m = 1$ получаются обычные числа Каталана
- BEST Theorem: число эйлеровых циклов в связном ор. графе равно $t_v \cdot \prod_i (deg_i - 1)!$, t_v - число остовов направленных в v, считается через теорему Кирхгофа. t_v равны для всех v в эйлеровом графе
- Lindström–Gessel–Viennot lemma: пусть есть взвешенный ориентированный граф, два множества вершин $a[n]$ и $b[n]$ и $e(a,b)$ — сумма величин всех простых путей из a в b, где величина пути $w(P)$ это произведение весов ребер. Тогда построим матрицу M со значениями $e(a_i, b_j)$. Будем считать такие $(P_1, \ldots, P_n)$, что P_i — путь из a_i в $b_{\sigma_i(P)}$ (σ — перестановка). При этом никакие два пути не пересекаются по вершинам. Тогда

$$\det M = \sum_{(P_1,\ldots,P_n)} sign(\sigma(P)) \prod_{i=1}^{n} w(P_i)$$

В частности, если все ребра веса 1 и единственная перестановка это Id (пути на сетке, например), то $\det M$ равно количеству всех возможных множеств путей.

- $\binom{n}{m}$ по модулю p: раскладываем в p-ичную систему счисления n и m, считаем

$$\binom{n}{m} = \prod_{i=0}^{k} \binom{n_i}{m_i} \mod p$$

- d_1, d_2 периоды, n длина, если $n \geq d_1 + d_2 - \gcd(d_1, d_2)$, то $\gcd(d_1, d_2)$ это тоже период
- Strings facts:
  - $X^\infty = Y^\infty \iff XY = YX$
  - Если $X^\infty \neq Y^\infty$, то $lcp(X^\infty, Y^\infty) = lcp(XY, YX)$
- Surreal numbers:
  - $G = \{L|R\}$: L — множество игр, в которые может пойти левый, R — правый
  - $x \leqslant y \Leftrightarrow X_L \not\geqslant y \wedge x \not\geqslant Y_R$, где $X_L \not\geqslant y \Leftrightarrow \forall x \in X_L : x \not\geqslant y$, остальные операторы тривиально определяются через $\leqslant$
  - Например $\{|\} = 0, \{|0\} = -1, \{0|\} = 1, \{0|1\} = \frac{1}{2}, \{|8\} = \{|\} = 0$
  - Сумма игр: $G_1 + G_2 = \{L_1 + G_2, G_1 + L_2 | R_1 + G_2, G_1 + R_2\}$
  - Исход: $G < 0$ — правый выигрывает, > 0 — левый, $= 0$ — второй, несравнимы — первый
- Два эйлеровых обхода: обычный (при входе записываем вершину), необычный (при выходе записываем вершину).

  Вертикальный путь до v: [префикс обычного кончающийся в v] минус [префикс необычного длины при входе в v]
- $A = \begin{pmatrix} a & b \\ \frac{1-a^2}{b} & -a \end{pmatrix}$, $A = I_n - QP$, $PQ = 2I_s$, $A = 2P - I, P^2 = P$
- $\det(A + uv^\top) = (1 + v^\top A^{-1} u) \det(A)$
  $\det(A + UWV^\top) = \det(W^{-1} + V^\top A^{-1} U) \det(W) \det(A)$
- $(A + uv^\top)^{-1} = A^{-1} - \frac{A^{-1} uv^\top A^{-1}}{1 + v^\top A^{-1} u}$
  $(A + UCV)^{-1} = A^{-1} - A^{-1} U (C^{-1} + VA^{-1}U)^{-1} VA^{-1}$
- $n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \times \exp \frac{1}{12n + 0{,}7509}$
- Т. Эрдеша-Галлаи: $d_1 \geq d_2 \geq \ldots \geq d_n$ степени прост неор графа, если $\sum_{i=1}^{n} d_i$ четно и $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)\ \forall 1 \leq k \leq n$.

  Для орграфа (вход и исход) $(a_1, b_1), \ldots, (a_n, b_n)$, т.ч. $a_1 \geq \ldots \geq a_n$, если $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ и $\sum_{i=1}^{k} a_i \leq \sum_{i=1}^{k} \min(b_i, k-1) + \sum_{i=k+1}^{n} \min(b_i, k)$.

2.3 Матрицы

- Матрица Татта: для любого ребра (i,j): $A_{i,j} = -x_{i,j}$, $A_{j,i} = x_{i,j}$. Инициализируем x случайно. Ранг $= 2 \cdot max$ парсоч. Вероятность ошибки равна n/mod
- По теореме Шварца-Зиппеля, вероятность зануления ненулевого многочлена при подстановке чисел равна n/mod
- Матрица Кирхгофа: дан неор. граф. $x_{i,i} = deg_i$; $x_{i,j} = -1, x_{j,i} = -1$ если есть ребро (i,j). Алг. дополнение матрицы равно числу остовных деревьев. Можно удалить последнюю строку и столбец и посчитать определитель.

  Обобщение: дан ор граф. без петель. $x_{i,i} = in_i$, $x_{i,j} = -count(i,j)$. Алг. дополнение без i строки и столбца равно числу ор. остовов с корнем в i

2.4 Математические формулы

- Расстояние между точками по сфере: $L = R \cdot \arccos(\cos\theta_1 \cdot \cos\theta_2 + \sin\theta_1 \cdot \sin\theta_2 \cdot \cos(\varphi_1 - \varphi_2))$, где θ – широты (от $-\frac{\pi}{2}$ до $\frac{\pi}{2}$), φ – долготы (от $-\pi$ до π)
- Объем шарового сегмента: $V = \pi h^2 (R - \frac{1}{3}h)$.
  Площадь поверхности шарового сегмента: $S = 2\pi Rh$.
  Здесь h – высота от вершины сектора до секущей плоскости.
- Объем усеченного конуса: $\frac{1}{3}\pi H(R^2 + Rr + r^2)$.
  Площадь боковой поверхности конуса: $S = \pi R\sqrt{R^2 + H^2}$.
- Формула Грина для вычисления площади по параметризации контура: $S = \frac{1}{2}\int (x\,dy - y\,dx) = \int_t (x(t)y'(t) - x'(t)y(t))dt$. Положительна, если обходить против часовой стрелки.
- Центр масс: $G_x = \frac{1}{6A} \sum_{i=0}^{n-1} (x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$, для y: $y_i + y_{i+1}$
- Объем многомерного шара $V_{2k}(R) = \frac{\pi^k}{k!} R^k$, $V_{2k+1}(R) = \frac{2(k!)(4\pi)^k}{(2k+1)!} R^k$
- Код Грея: $g_n = n \oplus \frac{n}{2}$
- AND соседних 1 бит: $A_{n+1} = 2A_n + 2rev(A_n)$, затем ко всем добавляем всем последний бит $i \bmod 2$, затем в конец A_{n+1} добавляем число 1.
- Числа Фибоначчи: $F_0 = 0$, $F_1 = 1$, $F_n = \dfrac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$
  - $\gcd(F_n, F_m) = F_{\gcd(n,m)}$
  - $\sum_{k=1}^{n} F_k^2 = F_n F_{n+1}$
  - $F_n^2 - F_{n-1} F_{n+1} = (-1)^{n-1}$
  - $F_n^2 + F_{n+1}^2 = F_{2n+1}$
- Числа Стирлинга: $s(n,k)$ — количество перестановок на n элементах, в которых ровно k циклов. $S(n,k)$ – это количество способов разбить n-элементное множество на k непустых подмножеств.
  - † $s(n,k) = (n-1) \cdot s(n-1,k) + s(n-1,k-1)$
  - † $S(n,k) = k \cdot S(n-1,k) + S(n-1,k-1)$
  - † $x^{\underline{n}} = x \cdot (x-1) \cdot \ldots \cdot (x-n+1) = \sum_{i=1}^{n} (-1)^{n-i} \cdot s(n,\ i) \cdot x^i$
  - † $x^n = \sum_{i=0}^{n} S(n,i) \cdot x^{\underline{i}}$
  - † $S(n,k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k+j} \binom{k}{j} j^n$, $S(n,k) = \sum_{j=1}^{k} (-1)^{k-j} \frac{j^{n-1}}{(j-1)!(k-j)!}$
  - † $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$, $B_n = \sum_{k=1}^{n} S(n,k)$
  - † $B(x) = \sum_{n=0}^{\infty} \frac{B_n}{n!} x^n = e^{e^x - 1}$
  - † $B_{n+m} = \sum_{k=0}^{n} \sum_{j=0}^{m} S(m,j) \binom{n}{k} j^{n-k} B_k$
- Gambler's ruin. У первого игрока есть n_1 монет, у второго n_2. На каждом шаге с вероятностью p второй отдает одну монету первому, а с вероятностью $q = 1 - p$ первый отдает одну монету второму. Игра заканчивается, когда у кого-нибудь не остается монет. Тогда первый выигрывает с вероятностью

$$\frac{1 - \left(\frac{p}{q}\right)^{n_1}}{1 - \left(\frac{p}{q}\right)^{n_1+n_2}}.$$

- Random walk. Пусть r — вероятность когда-нибудь попасть в правую от старта клетку. Тогда $r = p + q \cdot r^2$, отсюда выводится r (бинпоиск или $r = \frac{p}{q}$). По аналогии, вероятность когда-нибудь попасть вправо на n равна r^n.

- пентагональная теорема Эйлера: $\prod_{k=1}^{\infty}(1-x^k) = \sum_{q=-\infty}^{\infty}(-1)^q x^{\frac{3q^2+q}{2}}$.
- $\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$, $\sum_{k=1}^{n} k^3 = \frac{n^2(n+1)^2}{4}$
- $\mu(n)$ — функция Мёбиуса, равна 0 для чисел, кратных квадрату простого, для остальных -1 в степени количества простых в разложении.

$$\sum_{d|n}\mu(d) = 1 \iff n = 1$$

$$g(n) = \sum_{d|n} f(d) \iff f(n) = \sum_{d|n}\mu(d)g\left(\frac{n}{d}\right)$$

$$M(n) = \sum_{k=1}^{n}\mu(k) \Rightarrow \sum_{k=1}^{n} M\left(\frac{n}{k}\right) = 1$$

$$\varphi(n) - \text{Euler's totient function} \Rightarrow \sum_{m=1}^{n}\varphi(m) = \sum_{k=1}^{n} M(n/k)k$$

$$s_f(n) = \frac{s_{f*g}(n) - \sum_{d=2}^{n} s_f(\lfloor\frac{n}{d}\rfloor)g(d)}{g(1)}$$

$$d = 1*1,\ \sigma = id*1,\ \chi_1 = 1*\mu,\ 1 = d*\mu,$$
$$id = \sigma*\mu,\ id = \varphi*1,\ \sigma = \varphi*d$$

- $x^p - x = x(x-1)(x-2)\dots(x-(p-1)) \mod p$; Р/м $(x+a)^{\frac{p-1}{2}} - 1$

2.5 Ряды

- $\log(1-x) = \sum_{n=1}^{\infty} -\frac{1}{n}x^n$, $sin(x) = \sum_{n=1}^{\infty} -\frac{1}{(2n-1)!}x^{2n-1}$
- $cos(x) = \sum_{n=0}^{\infty} -\frac{1}{(2n)!}x^{2n}$, $\frac{1}{(1-x)^m} = \sum_{n=0}^{\infty}\binom{n+m-1}{n}x^n$
- $\log(\prod(1-ax))$ позволяет получить сумму k-х степеней
- C_n количество связных графов с n вершинами:
  $C_n = 2^{\binom{n}{2}} - \frac{1}{n}\sum_{k=1}^{n-1} k\binom{n}{k}2^{\binom{n-k}{2}}C_k$
  $\sum_{k=0}^{\infty} C_k\frac{x^k}{k!} = 1 + \log(\sum_{k=0}^{\infty} 2^{\binom{k}{2}}\frac{x^k}{k!})$
- $Catalan(x) = \frac{1-\sqrt{1-4x}}{2x}$; $\sum_k \binom{n}{k}\binom{m}{s-k} = \binom{n+m}{s}$
- $\sum_{n=k}^{\infty} S(n,k)x^{n-k} = \prod_{r=1}^{k}\frac{1}{1-rx}$, $\sum_{n=k}^{\infty}(-1)^{n-k}s(n,k)\frac{x^n}{n!} = \frac{(\log(1+x))^k}{k!}$
- Префиксные суммы: деление на $1-x$, брать производную, первообразную и тп.

2.6 Двойственная задача ЛП

Кратко: максимизировать c^Tx при $Ax \le b, x \ge 0$ эквивалентно минимизировать b^Ty при $A^Ty \ge c, y \ge 0$

Важный факт: оптимальный ответ всегда в вершине политопа, поэтому некоторое множество неравенств на самом деле равенства, из которых однозначно определяются x (как из слау).

Восстановить ответ:

- если $y_i > 0$, то $A_ix = b_i$
- если $A_ix < b_i$, то $y_i = 0$
- если $x_j > 0$, то $A_j^Ty = c_j$
- если $A_j^Ty > c_j$, то $x_j = 0$

Пусть прямая задача определена как:

- Дан набор из n переменных: $x_1, \dots, x_n$.
- Для каждой переменной i определено ограничение на знак — она должна быть либо неотрицательной ($x_i \geqslant 0$), либо неположительной ($x_i \leqslant 0$), либо ограничение не задано ($x_i \in \mathbb{R}$).
- Задана целевая функция: Максимизировать $c_1x_1 + \dots + c_nx_n$
- Задан список из m ограничений. Каждое ограничение j равно: $a_{j1}x_1 + \dots + a_{jn}x_n \lesseqgtr b_j$, где символ перед b_j может быть одним из трёх — $\geqslant$, $\leqslant$ или $=$.

Двойственная задача строится следующим образом:

- Каждое ограничение прямой задачи становится двойственной переменной. Таким образом, получаем m переменных: $y_1, \dots, y_m$.
- Знак ограничения каждой двойственной переменной "противоположен" знаку ограничения в прямой задаче. Таким образом, "$\geqslant b_j$" становится "$y_j \leqslant 0$", "$\leqslant b_j$" превращается в "$y_j \geqslant 0$", а "$= b_j$" превращается в "$y_j \in \mathbb{R}$".
- Целевая функция двойственной задачи равна (минимизировать) $b_1y_1 + \dots + b_my_m$
- Каждая переменная прямой задачи становится двойственным ограничением. Таким образом, получаем n ограничений. Коэффициент двойственной переменной в двойственных ограничениях равен коэффициенту переменной из ограничения прямой задачи. Таким образом, каждое ограничение i есть: $a_{1i}y_1 + \dots + a_{mi}y_m \lesseqgtr c_i$, где символ перед c_i аналогичен ограничению на переменную i в прямой задаче. Так, $x_i \leqslant 0$ превращается в "$\leqslant c_i$", "$x_i \geqslant 0$" превращается в "$\geqslant c_i$", а $x_i \in \mathbb{R}$ превращается в "$= c_i$".

Можно также расставлять знаки используя термин естественности:

- Естественные неравенства: (соответствует dual переменной ≥ 0) задача max, неравенство $\leq b_i$, задача min, неравенство $\geq b_i$;
- Неестественные неравенства: (соответствует переменной ≤ 0) задача max, неравенство $\geq b_i$, задача min, неравенство $\leq b_i$;

2.7 Матроиды

- J слева, $X \setminus J$ справа
- $y \in J$, $z \in X \setminus J$:
  - $J - y + z \in I_1$: edge $y \to z$
  - $J - y + z \in I_2$: edge $z \to y$
- shortest path from $\{z | J + z \in I_1\}$ to $\{z | J + z \in I_2\}$
- $\sum_{x\in J} w(x) \to max$:
  - $l(v) = w(v)$, if $v \in J$; $l(v) = -w(v)$ otherwise
  - calculate potentials $p(v)$ - кратчайшее расстояние, равное сумме $l(v)$ по всем вершинам пути с текущей итерации:
    - $w_1(v) = p(v), w_2(v) = w(v) - p(v)$, если $v \in J$
    - $w_1(v) = w(v) + p(v), w_2(v) = -p(v)$, если $v \in X \setminus J$
  - Далее меняем J на $J \oplus P$
  - Получаем новые потенциалы:
    - $p(v) = w_1(v)$, если $v \in J$
    - $p(v) = -w_2(v)$, если $v \in X \setminus J$

2.8 Segment tree beats

1) Операции +=, max= (хранишь 2 минимума на отрезке и колво первого, брейк по первому)

2) Есть массив A, на нем операции +=

(ист максимум/минимум) Есть массив B, что каждый раз $B_i = \max(B_i, A_i)$

(ист сумма) Есть массив C, что каждый раз C_i += A_i Просят считать сумму B, C

- ист макс: $D_i = B_i - A_i$. A_i += $x \to D_i = \max(D_i - x, 0)$
- ист мин: $D_i = B_i - A_i$. A_i += $x \to D_i = \min(D_i - x, 0)$
- ист сумма: $D_i = C_i - t \cdot A_i$ (t время). A_i += $x \to D_i = D_i - (t-1)\cdot x$

3) Если просят ист максимум с += и max=, хранишь таг $\max(a + x, b)$ для A и для B (2 тага в вершине)

2.9 Мусор

- Покрасить ребра двудольного графа в d цветов: найти парсоч для вершин степени d
- `c = x&-x, r = x+c; (((r^x) >> 2)/c) | r` is the next number after `x` with the same number of bits set.
- `feenableexcept(29);` kills the program on NaNs (1), 0-divs (4), infinities (8) and denormals (16).
- pragma GCC optimize ("trapv") kills the program on integer overflows (but is really slow).
- `signal(SIGSEGV, [](int) { _Exit(0); });`
- `throw 42;` чтобы выходить из программы в мультитесте.

3 Числки

3.1 Много делителей

Под разложением подразумевается набор степеней первых простых чисел. Например, $60 = 2^2 \cdot 3^1 \cdot 5^1$.

Используется максимум 12 первых простых чисел: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37.

	n	$d(n)$	Разложение
$\leqslant 10^1$	6	4	1, 1
$\leqslant 10^2$	60	12	2, 1, 1
$\leqslant 10^3$	840	32	3, 1, 1, 1
$\leqslant 10^4$	7560	64	3, 3, 1, 1
$\leqslant 10^5$	83,160	128	3, 3, 1, 1, 1
$\leqslant 10^6$	720,720	240	4, 2, 1, 1, 1, 1
$\leqslant 10^7$	8,648,640	448	6, 3, 1, 1, 1, 1
$\leqslant 10^8$	73,513,440	768	5, 3, 1, 1, 1, 1, 1
$\leqslant 10^9$	735,134,400	1344	6, 3, 2, 1, 1, 1, 1
$\leqslant 10^{10}$	6,983,776,800	2304	5, 3, 2, 1, 1, 1, 1, 1
$\leqslant 10^{11}$	97,772,875,200	4032	6, 3, 2, 2, 1, 1, 1, 1
$\leqslant 10^{12}$	963,761,198,400	6720	6, 4, 2, 1, 1, 1, 1, 1, 1
$\leqslant 10^{13}$	9,316,358,251,200	10,752	6, 3, 2, 1, 1, 1, 1, 1, 1, 1
$\leqslant 10^{14}$	97,821,761,637,600	17,280	5, 4, 2, 2, 1, 1, 1, 1, 1, 1
$\leqslant 10^{15}$	866,421,317,361,600	26,880	6, 4, 2, 1, 1, 1, 1, 1, 1, 1, 1
$\leqslant 10^{16}$	8,086,598,962,041,600	41,472	8, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1
$\leqslant 10^{17}$	74,801,040,398,884,800	64,512	6, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1
$\leqslant 10^{18}$	897,612,484,786,617,600	103,680	8, 4, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1

3.2 Количество разбиений числа на слагаемые

n	$p(n)$	n	$p(n)$	n	$p(n)$	n	$p(n)$
1	1	26	2436	51	239,943	76	9,289,091
2	2	27	3010	52	281,589	77	10,619,863
3	3	28	3718	53	329,931	78	12,132,164
4	5	29	4565	54	386,155	79	13,848,650
5	7	30	5604	55	451,276	80	15,796,476
6	11	31	6842	56	526,823	81	18,004,327
7	15	32	8349	57	614,154	82	20,506,255
8	22	33	10,143	58	715,220	83	23,338,469
9	30	34	12,310	59	831,820	84	26,543,660
10	42	35	14,883	60	966,467	85	30,167,357
11	56	36	17,977	61	1,121,505	86	34,262,962
12	77	37	21,637	62	1,300,156	87	38,887,673
13	101	38	26,015	63	1,505,499	88	44,108,109
14	135	39	31,185	64	1,741,630	89	49,995,925
15	176	40	37,338	65	2,012,558	90	56,634,173
16	231	41	44,583	66	2,323,520	91	64,112,359
17	297	42	53,174	67	2,679,689	92	72,533,807
18	385	43	63,261	68	3,087,735	93	82,010,177
19	490	44	75,175	69	3,554,345	94	92,669,720
20	627	45	89,134	70	4,087,968	95	104,651,419
21	792	46	105,558	71	4,697,205	96	118,114,304
22	1002	47	124,754	72	5,392,783	97	133,230,930
23	1255	48	147,273	73	6,185,689	98	150,198,136
24	1575	49	173,525	74	7,089,500	99	169,229,875
25	1958	50	204,226	75	8,118,264	100	190,569,292

3.3 Количество простых чисел

n	$\pi(n)$	n	$\pi(n)$	n	$\pi(n)$
2	1	$2 \cdot 10^3$	303	$2 \cdot 10^6$	148,933
4	2	$4 \cdot 10^3$	550	$4 \cdot 10^6$	283,146
6	3	$6 \cdot 10^3$	783	$6 \cdot 10^6$	412,849
8	4	$8 \cdot 10^3$	1007	$8 \cdot 10^6$	539,777
10^1	4	10^4	1229	10^7	664,579
$2 \cdot 10^1$	8	$2 \cdot 10^4$	2262	$2 \cdot 10^7$	1,270,607
$4 \cdot 10^1$	12	$4 \cdot 10^4$	4203	$4 \cdot 10^7$	2,433,654
$6 \cdot 10^1$	17	$6 \cdot 10^4$	6057	$6 \cdot 10^7$	3,562,115
$8 \cdot 10^1$	22	$8 \cdot 10^4$	7837	$8 \cdot 10^7$	4,669,382
10^2	25	10^5	9592	10^8	5,761,455
$2 \cdot 10^2$	46	$2 \cdot 10^5$	17,984	$2 \cdot 10^8$	11,078,937
$4 \cdot 10^2$	78	$4 \cdot 10^5$	33,860	$4 \cdot 10^8$	21,336,326
$6 \cdot 10^2$	109	$6 \cdot 10^5$	49,098	$6 \cdot 10^8$	31,324,703
$8 \cdot 10^2$	139	$8 \cdot 10^5$	63,951	$8 \cdot 10^8$	41,146,179
10^3	168	10^6	78,498	10^9	50,847,534

3.4 Простые числа

По два рандомных простых числа на каждый диапазон.

$\leqslant 10^1$	5	7
$\leqslant 10^2$	47	67
$\leqslant 10^3$	727	839
$\leqslant 10^4$	5827	9257
$\leqslant 10^5$	63,841	69,859
$\leqslant 10^6$	393,209	418,799
$\leqslant 10^7$	2,189,321	7,914,601
$\leqslant 10^8$	55,531,009	84,106,573
$\leqslant 10^9$	331,034,513	842,384,651
$\leqslant 10^{10}$	5,181,109,469	6,231,902,767
$\leqslant 10^{11}$	74,188,428,887	91,056,196,939
$\leqslant 10^{12}$	511,051,062,853	921,633,177,131
$\leqslant 10^{13}$	2,957,720,312,387	9,866,869,529,287
$\leqslant 10^{14}$	16,117,262,959,349	40,665,615,401,219
$\leqslant 10^{15}$	417,488,822,150,503	960,736,110,002,941
$\leqslant 10^{16}$	7,844,092,896,161,623	9,131,554,486,645,081
$\leqslant 10^{17}$	10,306,459,369,469,099	91,179,983,651,335,507
$\leqslant 10^{18}$	664,481,138,468,163,557	834,248,894,892,599,033
$\leqslant 2^{63}$	7,696,662,867,157,535,449	8,011,293,081,038,232,049

3.5 Числа Белла

Количество способов разбить n элементов на подмножества. Например, 3 можно разбить 5-ю способами: $[[1,2,3]]$, $[[1],[2],[3]]$, $[[1,2],[3]]$, $[[1,3],[2]]$, $[[2,3],[1]]$.

i	B_i	i	B_i
1	1	14	190,899,322
2	2	15	1,382,958,545
3	5	16	10,480,142,147
4	15	17	82,864,869,804
5	52	18	682,076,806,159
6	203	19	5,832,742,205,057
7	877	20	51,724,158,235,372
8	4140	21	474,869,816,156,751
9	21,147	22	4,506,715,738,447,323
10	115,975	23	44,152,005,855,084,346
11	678,570	24	445,958,869,294,805,289
12	4,213,597	25	4,638,590,332,229,999,353
13	27,644,437	26	49,631,246,523,618,756,274

4 Рандомный код

4.1 Быстрый аллокатор

```
const int MAXMEM = 4e8;
int mpos = 0;
char mem[MAXMEM];
inline void* operator new(size_t n) {
  if (mpos + n >= MAXMEM)
    mpos = MEMSIZE / 3;
  char* ret = mem + mpos;
  mpos += n;
  return (void*)ret;
}
inline void operator delete (void *) {}
```

4.2 Фенвик

```
template<typename T>
struct fenwick {
  vector<T> tree;
  int n, K;
  fenwick(int n = 0) : n(n) {
    tree.assign(n, 0);
    K = 0;
    while ((1 << K) <= n) ++K;
  }

  void add(int i, T k) {
    for (; i < n; i = (i | (i + 1))) tree[i] += k;
  }

  T ask(int r) {
    T res = 0;
    for (; r >= 0; r = (r & (r + 1)) - 1) res += tree[r];
    return res;
  }
  T ask(int l, int r) {
    if (l > r) return 0;
    return ask(r) - ask(l - 1);
  }

  int lower_bound(T x) { // find first i such that sum[0..i] >= x
    int cur = 0;
    T cur_sum = 0;
    for (int k = K - 1; k >= 0; --k) {
      int ind = cur | ((1 << k) - 1);
      if (ind >= n) continue;
      if (cur_sum + tree[ind] >= x) continue;
      cur_sum += tree[ind]; cur |= (1 << k);
    }
    return cur;
  }
};
```

4.3 PBDS

```
#pragma GCC optimize("Ofast")
#pragma GCC optimize("fast-math")
#pragma GCC optimize("unroll-loops")
//builtin_popcount,clz...
#pragma GCC target("popcnt,lzcnt")
//general bit operations x&(x-1)...
#pragma GCC target("abm,bmi,bmi2")
// ne pomogaet
#pragma GCC target("sse,sse2,sse3,sse4,sse4.1,sse4.2")

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;
template <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
↪  tree_order_statistics_node_update>;
template <typename K, typename V>
using ordered_map = tree<K, V, less<K>, rb_tree_tag,
↪  tree_order_statistics_node_update>;
// ordered_set<T> q;
// insert(z)
// each element is inserted only if not present

// order_of_key(z)
// index(0-based) of lower_bound(z)

// find_by_order(z)
// iterator of z-th element

gp_hash_table<int, int> table;

bitset<N> a;
for (int i = a._Find_first(); i != a.size(); i = a._Find_next(i))
```

4.4 Hilbert order

```
// работает за лог, в компараторе не вызывать, pow = ceil(log(n))
inline ll hilbertOrder(int x, int y, int pow, int rotate = 0) {
  if (pow == 0) {
    return 0;
  }
  int hpow = 1 << (pow-1);
  int seg = (x < hpow) ? (
    (y < hpow) ? 0 : 3
  ) : (
    (y < hpow) ? 1 : 2
  );
  seg = (seg + rotate) & 3;
  const int rotateDelta[4] = {3, 0, 0, 1};
  int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
  int nrot = (rotate + rotateDelta[seg]) & 3;
  ll subSquareSize = ll(1) << (2*pow - 2);
  ll ans = seg * subSquareSize;
  ll add = hilbertOrder(nx, ny, pow-1, nrot);
  ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add - 1);
  return ans;
}
```

4.5 Дерево отрезков

```
//lichao

// insert: add line kx+b to all tree
// query: max(kx+b)
// can change to add line kx+b on segment
// segment tree [1;r), root 0
// can be extended if you need (e.g. add kx+b to all x in range) пример если
↪  запрос на всем дереве: пушим корень вниз (как будто запрос
↪  добавления прямой), потом лениво прибавляем как обычно
struct Line {
  ld k, b;
  ld operator()(ld x) { return k * x + b; }
} a[maxn * 4];
void insert(int l, int r, Line s, int v=0) {
  if(l + 1 == r) {
    if(s(l) > a[v](l)) a[v] = s;
    return;
  }
  int m = (l + r) / 2;
  if(a[v].k > s.k) swap(a[v], s);
  if(a[v](m) < s(m)) {
    swap(a[v], s);
    insert(l, m, s, 2 * v + 1);
  }
  else insert(m, r, s, 2 * v + 2);
}
ld query(int l, int r, int x, int v=0) {
  if(l + 1 == r) return a[v](x);
  int m = (l + r) / 2;
  if(x < m) return max(a[v](x), query(l, m, x, 2 * v + 1));
  else return max(a[v](x), query(m, r, x, 2 * v + 2));
}

//hld збаня
namespace hld {
  const int N = 1 << 17;
  int par[N], heavy[N], h[N];
  int root[N], pos[N];
  int n;
  vector<vector<int>> e;
  segtree tree;

  int dfs(int v) {
    int sz = 1, mx = 0;
    for (int to : e[v]) {
      if (to == par[v]) continue;
      par[to] = v;
      h[to] = h[v] + 1;
      int cur = dfs(to);
      if (cur > mx) heavy[v] = to, mx = cur;
      sz += cur;
    }
    return sz;
  }

  template <typename T>
  void path(int u, int v, T op) {
    for (; root[u] != root[v]; v = par[root[v]]) {
      if (h[root[u]] > h[root[v]]) swap(u, v);
      op(pos[root[v]], pos[v]);
    }
    if (h[u] > h[v]) swap(u, v);
    op(pos[u], pos[v]);
  }

  // обычный список смежности дерева
  void init(vector<vector<int>> _e) {
    e = _e;
    n = e.size();
    tree = segtree(n);
    memset(heavy, -1, sizeof(heavy[0]) * n);
    par[0] = -1;
    h[0] = 0;
    dfs(0);
    for (int i = 0, cpos = 0; i < n; i++) {
      if (par[i] == -1 || heavy[par[i]] != i) {
        for (int j = i; j != -1; j = heavy[j]) {
          root[j] = i;
          pos[j] = cpos++;
        }
      }
    }
  }

  void add(int v, int x) {
    tree.add(pos[v], x);
  }

  int get(int u, int v) {
    int res = 0;
    path(u, v, [&](int l, int r) {
      // обновлять ответ отрезком ДОшки [l, r] включительно
      res = max(res, tree.get(l, r));
    });
    return res;
  }
}

//catdog
struct Node {
  // item
  void update(Node &A, Node &B) {}
};

struct Str {
  vector<Node> q;
  int s;
  void update(int v) {
    q[v].update(q[2 * v], q[2 * v + 1]);
  }
  void upd(int v, Node const &w) {
    assert(v < s);
    v += s;
    q[v] = w;
    v /= 2;
    while (v) {
      update(v);
      v /= 2;
    }
  }

  void init(int n) {
    s = 1;
    while (s < n)
      s *= 2;
    q.resize(2 * s);
    for (int i = s - 1; i > 0; i--)
      update(i);
  }

  Node get() {
    return q[1];
  }
};

int n;

vector<int> e[maxn];

int p[maxn], sz[maxn];

void prec(int v, int par) {
  p[v] = par;
  sz[v] = 1;
  // если нет обратного ребра, удалить строчку
  if (v)
    e[v].erase(find(all(e[v]), par));
  if (e[v].empty())
    return;
  for (int i : e[v])
    if (i != par) {
      prec(i, v);
      sz[v] += sz[i];
    }
  int id = 0;
  for (int i = 1; i < e[v].size(); i++)
    if (sz[e[v][id]] < sz[e[v][i]])
      id = i;
```

```
  swap(e[v][0], e[v][id]);
}

Str tree[maxn];

int hid[maxn], hpar[maxn], hsz[maxn];

int who[maxn], mas[maxn][2];

void upd(int v) {
  Node s;
  // посчитать Node для v из массивов who и mas (для примера)
  tree[hpar[v]].upd(hid[v], s);
}

void dfs(int v) {
  hid[v] = hsz[hpar[v]]++;
  if (e[v].empty()) {
    tree[hpar[v]].init(hsz[hpar[v]]);
    return;
  }
  hpar[e[v][0]] = hpar[v];
  dfs(e[v][0]);
  for (int i = 1; i < e[v].size(); i++) {
    hpar[e[v][i]] = e[v][i];
    dfs(e[v][i]);
  }
}

void solve() {
  cin >> n;
  for (int i = 1; i < n; i++) {
    int v, u;
    cin >> v >> u;
    v--;
    u--;
    e[v].push_back(u);
    e[u].push_back(v);
  }
  prec(0, 0);
  dfs(0);
  int zaps;
  cin >> zaps;
  while (zaps--) {
    int tp, v;
    cin >> tp >> v;
    v--;
    // обновили значение для v
    who[v] = tp - 1;
    while (1) {
      int u = hpar[v];
      auto old = tree[u].get();
      upd(v);
      if (u == 0)
        break;
      auto res = tree[u].get();
      u = p[u];
      // обновляем u через сына hpar[v]
      mas[u][0] += res.fr - old.fr;
      mas[u][1] += res.sc - old.sc;
      v = u;
    }
    auto res = tree[0].get();
    // ответ
  }
}
```

5 Строки

5.1 Ахо-Корасик

```
//fail - суффиксные ссылки
//ребер автомата нету, можно их строить по ходу копируя
↪ отсутствующие переходы из суффссылки
//время построения такое же как у p-функции(линия)
const int MAXN = 100005, MAXC = 26;
int trie[MAXN][MAXC], fail[MAXN], term[MAXN], piv;
void init(vector<string> &v){
  memset(trie, 0, sizeof(trie));
  memset(fail, 0, sizeof(fail));
  memset(term, 0, sizeof(term));
  piv = 0;
  for(auto &i : v) {
    int p = 0;
    for(auto &j : i) {
      if(!trie[p][j]) trie[p][j] = ++piv;
      p = trie[p][j];
    }
    term[p] = 1;
  }
  queue<int> que;
  for (int i = 0; i < MAXC; i++) {
    if (trie[0][i]) que.push(trie[0][i]);
  }
  while (!que.empty()) {
    int x = que.front();
    que.pop();
    for (int i = 0; i < MAXC; i++) {
      if (trie[x][i]) {
        int p = fail[x];
        while (p && !trie[p][i]) p = fail[p];
        p = trie[p][i];
        fail[trie[x][i]] = p;
        // if (term[p]) term[trie[x][i]] = 1; // !!! push in suflink tree
        que.push(trie[x][i]);
      }
    }
  }
}
bool query(string &s) {
  int p = 0;
  for (auto &i : s) {
    while (p && !trie[p][i]) p = fail[p];
    p = trie[p][i];
    if (term[p]) return 1;
  }
  return 0;
}
```

5.2 Алгоритм Манакера

```
template <typename T>
vector<int> manacher(int n, const T &s) {
  if (n == 0) {
    return vector<int>();
  }
  vector<int> res(2 * n - 1, 0);
  int l = -1, r = -1;
  for (int z = 0; z < 2 * n - 1; z++) {
    int i = (z + 1) >> 1;
    int j = z >> 1;
    int p = (i >= r ? 0 : min(r - i, res[2 * (l + r) - z]));
    while (j + p + 1 < n && i - p - 1 >= 0) {
      if (!(s[j + p + 1] == s[i - p - 1])) {
        break;
      }
      p++;
    }
    if (j + p > r) {
      l = i - p;
      r = j + p;
    }
    res[z] = p;
  }
  return res;
  // res[2 * i] = odd radius in position i
  // res[2 * i + 1] = even radius between positions i and i + 1
  // s = "abaa" -> res = {0, 0, 1, 0, 0, 1, 0}
  // центры идут в порядке слева направо, сначала 0, потом между 01
↪ итд

}
template <typename T>
vector<int> manacher(const T &s) {
  return manacher((int) s.size(), s);
}
```

5.3 Суффиксный массив

```
// O(nlogn), 170ms for n=500k
void build(const vector<int>& str, vector<int>& p) {
  vector<int> s(str);
  for (int& i : s) i++;
  s.emplace_back(0);
  int mx = *max_element(s.begin(), s.end()) + 1;
  int n = s.size();
  vector<int> cnt(max(mx, n));
  for (int i = 0; i < mx; i++) cnt[i] = 0;
  for (int i = 0; i < n; i++) cnt[s[i]]++;
  for (int i = 1; i < mx; i++) cnt[i] += cnt[i - 1];
  p.resize(n);
  for (int i = n - 1; i >= 0; i--) p[--cnt[(int)s[i]]] = i;
  int cl = 1;
  vector<int> c(n);
  c[p[0]] = 0;
  for (int i = 1; i < n; i++) {
    cl += s[p[i]] != s[p[i - 1]];
    c[p[i]] = cl - 1;
  }
  vector<int> pn(n), cn(n);
  for (int len = 1; len < n; len <<= 1) {
    for (int i = 0; i < cl; i++) cnt[i] = 0;
    for (int i = 0; i < n; i++) cnt[c[i]]++;
    for (int i = 1; i < cl; i++) cnt[i] += cnt[i - 1];
    for (int i = 0; i < n; i++) pn[i] = (p[i] - len + n) % n;
    for (int i = n - 1; i >= 0; i--) p[--cnt[c[pn[i]]]] = pn[i];
    cl = 1;
    cn[p[0]] = 0;
    for (int i = 1; i < n; i++) {
      cl += c[p[i]] != c[p[i - 1]] || c[(p[i] + len) % n] != c[(p[i - 1] + len)
↪ % n];
      cn[p[i]] = cl - 1;
    }
    for (int i = 0; i < n; i++) c[i] = cn[i];
  }
  for (int i = 0; i < n; i++) {
    if (p[i] == n - 1) {
      for (int j = i; j < n - 1; j++)
        p[j] = p[j + 1];
      break;
    }
  }
  n--;
  p.resize(n);
  for (int i = 0; i < n; i++) c[p[i]] = i;
}

void kasai(const vector<int>& s, const vector<int>& p, vector<int>& lcp) {
  int n = s.size();
  vector<int> c(n);
  lcp.resize(n);
  int z = 0;
  for (int i = 0; i < n; i++) {
    int j = c[i];
    if (j == n - 1) {
      z = 0;
    } else {
      while (s[i + z] == s[p[j + 1] + z]) z++;
    }
    lcp[j] = z;
```

```
    z -= !!z;
  }
}

// _____________________________________________________________________________
// O(n), 50ms for n=500k
void sais_core(int n, int m, const int* s, int* sa, char* type, int* lms, int*
↪  cnt) {
  int n1 = 0, ch = -1;
  type[n - 1] = 1;
  for (int i = n - 2; i >= 0; --i)
    type[i] = s[i] == s[i + 1] ? type[i + 1] : s[i] < s[i + 1];
  fill(cnt, cnt + m, 0);
  for (int i = 0; i < n; ++i) ++cnt[s[i]];
  partial_sum(cnt, cnt + m, cnt);
  auto induced_sort = [&](const int v[]) {
    fill(sa, sa + n, 0);
    int *cur = cnt + m;
    auto push_S = [&](const int x) { sa[--cur[s[x]]] = x; };
    auto push_L = [&](const int x) { sa[cur[s[x]]++] = x; };
    copy(cnt, cnt + m, cur);
    for (int i = n1 - 1; i >= 0; --i) push_S(v[i]);
    copy(cnt, cnt + m - 1, cur + 1);
    for (int i = 0; i < n; ++i)
      if (sa[i] > 0 && type[sa[i] - 1] == 0)
        push_L(sa[i] - 1);
    copy(cnt, cnt + m, cur);
    for (int i = n - 1; i >= 0; --i)
      if (sa[i] > 0 && type[sa[i] - 1])
        push_S(sa[i] - 1);
  };
  for (int i = 1; i < n; ++i)
    if (type[i - 1] == 0 && type[i] == 1)
      type[i] = 2, lms[n1++] = i;
  induced_sort(lms);
  auto lms_equal = [&](int x, int y) {
    if (s[x] == s[y])
      while (s[++x] == s[++y] && type[x] == type[y])
        if (type[x] == 2 || type[y] == 2)
          return true;
    return false;
  };
  int* s1 = remove_if(sa, sa + n, [&](const int x) { return type[x] != 2; });
  for (int i = 0; i < n1; ++i)
    s1[sa[i] >> 1] = ch += ch <= 0 || !lms_equal(sa[i], sa[i - 1]);
  for (int i = 0; i < n1; ++i)
    s1[i] = s1[lms[i] >> 1];
  if (ch + 1 < n1)
    sais_core(n1, ch + 1, s1, sa, type + n, lms + n1, cnt + m);
  else
    for (int i = 0; i < n1; ++i)
      sa[s1[i]] = i;
  for (int i = 0; i < n1; ++i)
    lms[n1 + i] = lms[sa[i]];
  induced_sort(lms + n1);
}

void calc(const vector<int>& str, vector<int>& sa) {
  vector<int> s(str);
  for (int& i : s) i++;
  s.emplace_back(0);
  int n = s.size();
  int m = *max_element(s.begin(), s.end()) + 1;
  vector<int> lms(n), cnt(2 * max(n, m));
  vector<char> type(2 * n);
  sa.resize(n);
  sais_core(n, m, s.data(), sa.data(), type.data(), lms.data(), cnt.data());
  n--;
  for (int i = 0; i < n; i++) {
    sa[i] = sa[i + 1];
  }
  sa.resize(n);
}
```

5.4 Суффиксный автомат

```
// [len[suf[v]]+1,len[v]] are lengths of suffixes of strings coming to v
// right context of s: the set of w, s.t. sw is suffix of the string
// vertex of SA <--> right context
const int A = 26;
const int MAX_N = 1e5;
const int NC = 2 * MAX_N + 5;

int len[NC], suf[NC], go[NC][A], cnt, root, all;
bool term[NC];

int new_node() {
  len[cnt] = 0;
  suf[cnt] = -1;
  term[cnt] = false;
  for (int i = 0; i < A; i++) go[cnt][i] = -1;
  return cnt++;
}

void add_symbol(char x) {
  int v = new_node();
  len[v] = len[all] + 1;
  swap(all, v);
  assert(v != all);
  while (v != -1 && go[v][x - 'a'] == -1) {
    go[v][x - 'a'] = all;
    v = suf[v];
  }
  if (v == -1) { suf[all] = root; return; }
  int pr = go[v][x - 'a'];
  if (len[v] + 1 == len[pr])
    suf[all] = pr;
  else {
    int clone = new_node();
    len[clone] = len[v] + 1;
    for (int i = 0; i < A; i++) go[clone][i] = go[pr][i];
    suf[clone] = suf[pr];
    suf[pr] = suf[all] = clone;
    while (v != -1 && go[v][x - 'a'] == pr) {
      go[v][x - 'a'] = clone;
      v = suf[v];
    }
  }
}

bool used[NC]; // dfs and this stuff only if needed
int cnt_sub[NC]; // number of substrings for path ending in v
ll cnt_pref[NC]; // number of DIFFERENT substrings: prefix is path ending in v
ll cnt_pref_all[NC]; // the number of ALL substrings: prefix is path ending in v
int max_sub[NC];

void dfs(int p) {
  used[p] = true;
  for (int i = 0; i < A; i++)
    if (go[p][i] != -1 && !used[go[p][i]])
      dfs(go[p][i]);
  cnt_sub[p] = term[p];
  cnt_pref[p] = 1;
  for (int i = 0; i < A; i++) {
    if (go[p][i] != -1) {
      cnt_sub[p] += cnt_sub[go[p][i]];
      cnt_pref[p] += cnt_pref[go[p][i]];
    }
  }
  cnt_pref_all[p] = cnt_sub[p];
  for (int i = 0; i < A; i++) {
    if (go[p][i] != -1) {
      cnt_pref_all[p] += cnt_pref_all[go[p][i]];
    }
  }
}

void build(const string& s) {
  cnt = 0;
  root = new_node();
  all = root;
  for (char x : s) add_symbol(x);
  int v = all;
  while (v != -1) { term[v] = true; v = suf[v]; }
  dfs(root);
}
```

5.5 Суффиксное дерево

```
const int MAX_N = 500'000;
const int VN = 2 * MAX_N + 5;
const int A = 26 + 1;

// ts - the number of vertices, tv - vertex of the full string
// p[v] - parent of v, s[v] - suflink of v, t[v][x] == 0 - no edge
// edge p[v] --> v contains string a[l[v]..r[v]]
// 0 (root); 2, 3, ..., ts-1 [inner vertices]
// add fictive 'z'+1 to the end to make leafs for suffixes
string a;
int t[VN][A], l[VN], r[VN], p[VN], s[VN];
int tv, tp, ts, la;

void ukkadd(int c) {
  a += (char)('a' + c);
  suff:;
  if (r[tv] < tp) {
    if (t[tv][c] == -1) {
      t[tv][c] = ts;
      l[ts] = la;
      p[ts++] = tv;
      tv = s[tv];
      tp = r[tv] + 1;
      goto suff;
    }
    tv = t[tv][c];
    tp = l[tv];
  }
  if (tp == -1 || c == a[tp] - 'a') {
    tp++;
  } else {
    l[ts + 1] = la;
    p[ts + 1] = ts;
    l[ts] = l[tv];
    r[ts] = tp - 1;
    p[ts] = p[tv];
    t[ts][c] = ts + 1;
    t[ts][a[tp] - 'a'] = tv;
    l[tv] = tp;
    p[tv] = ts;
    t[p[ts]][a[l[ts]] - 'a'] = ts;
    ts += 2;
    tv = s[p[ts - 2]];
    tp = l[ts - 2];
    while (tp <= r[ts - 2]) { tv = t[tv][a[tp] - 'a']; tp += r[tv] - l[tv] + 1; }
    if (tp == r[ts - 2] + 1) s[ts - 2] = tv; else s[ts - 2] = ts;
    tp = r[tv] - tp + r[ts - 2] + 2;
    goto suff;
  }
}

int cnt_sub[VN]; // the number of substrings for path ending in v

void dfs(int i) {
  int cnt = 0;
  for (int x = 0; x < A; x++) {
    if (t[i][x] > 0) {
      cnt++;
      dfs(t[i][x]);
      cnt_sub[i] += cnt_sub[t[i][x]];
    }
```

```
  }
  cnt_sub[i] += (cnt == 0);
}

void build(string str) {
  str += (char)('z' + 1);
  a.clear();
  ts = 2; tv = 0; tp = 0;
  fill(r, r + VN, (int)str.size() - 1);
  s[0] = 1;
  l[0] = -1; r[0] = -1;
  l[1] = -1; r[1] = -1;
  memset(t, -1, sizeof(t));
  fill(t[1],t[1] + 26,0);
  for (la = 0; la < (int)str.size(); la++) ukkadd(str[la] - 'a');
  dfs(0);
}

int cnt_substr(const string& str) {
  int v = 0;
  int i = 0;
  while (i < (int)str.size()) {
    v = t[v][str[i] - 'a'];
    if (v == 0) return 0;
    int st = l[v];
    int cnt = min(r[v] - l[v] + 1, (int)str.size() - i);
    for (int x = 0; x < cnt; x++) {
      if (a[st + x] != str[i + x]) {
        return 0;
      }
    }
    i += cnt;
  }
  return cnt_sub[v];
}
```

5.6 Z-функция, префикс-функция

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

vector<int> prefix_function (string s) {
  int n = (int) s.length();
  vector<int> pi (n);
  for (int i=1; i<n; ++i) {
    int j = pi[i-1];
    while (j > 0 && s[i] != s[j])
      j = pi[j-1];
    if (s[i] == s[j])  ++j;
    pi[i] = j;
  }
  return pi;
}
```

5.7 Линдон, минимальный циклический сдвиг

```
// Lyndon:
// String is simple, if it is smaller than all its suffixes
// s = s_1 s_2 ... s_k, s_1 >= s_2 >= ... >= s_k, s_i is simple
string min_cyclic_shift (string s) {
  s += s;
  int n = (int) s.length();
  int i=0, ans=0;
  while (i < n/2) {
    ans = i;
    int j=i+1, k=i;
    while (j < n && s[k] <= s[j]) {
      if (s[k] < s[j])
        k = i;
      else
        ++k;
      ++j;
    }
    while (i <= k)  i += j - k;
  }
  return s.substr (ans, n/2);
}

template <typename T>
vector<int> duval_prefixes(int n, const T &s) {
  vector<int> z = z_function(n, s);
  vector<int> ans(n, 0);
  int i = 0, pos = 0;
  while (i < n) {
    int j = i, k = i;
    while (j < n) {
      j++;
      if (j > pos) {
        if (z[k] <= pos - k && s[z[k]] < s[k + z[k]]) {
          int shift = (pos - i) / (j - k) * (j - k);
          ans[pos] = ans[pos - shift] + shift;
        } else {
          ans[pos] = i;
        }
        pos++;
      }
      if (s[k] < s[j]) k = i; else
      if (!(s[j] < s[k])) k++; else
      else break;
    }
    while (i <= k) {
      i += j - k;
    }
  }
  return ans;
  // returns 0-indexed positions of the least cyclic shifts of all prefixes
}
template <typename T>
vector<int> duval_prefixes(const T &s) {
  return duval_prefixes((int) s.size(), s);
}
```

5.8 Руны

```
// their number <= n, sum (r-l+1)/(2period) <= n, sum of (r-l+1) <= O(n log(n))
struct Run {
  int period; // period is minimal period of [l, r], r - l >= 2period
  int l, r; // for [l-1, r] and [l, r+1] cond is not satisfied

  Run() = default;
  Run(int period, int l, int r) : period(period), l(l), r(r) {}

  friend bool operator<(const Run& r1, const Run& r2) {
    if (r1.period != r2.period) {
      return r1.period < r2.period;
    }
    return r1.l != r2.l ? r1.l < r2.l : r1.r < r2.r;
  }

  friend bool operator==(const Run& r1, const Run& r2) { return r1.period ==
↪  r2.period && r1.l == r2.l && r1.r == r2.r; }
};

// nlogn, was ~900ms for n=10^6
vector<Run> run_enumerate(string& s) {
  char sentinel = '#';
  for (auto& e : s) assert(e != sentinel);

  vector<Run> glob_result;

  auto div_conq = [&](auto div_conq, int l, int r) -> vector<Run> {
    if (r - l <= 1) return {};

    const int m = (l + r) >> 1;
    vector<Run> run_l = div_conq(div_conq, l, m);
    vector<Run> run_r = div_conq(div_conq, m, r);

    string rl = s.substr(m, r - m);
    rl += sentinel;
    rl += s.substr(l, m - l);
    vector<int> z_rl = z_function(rl);

    reverse(rl.begin(), rl.end());
    vector<int> z_rl_rev = z_function(rl);

    const int siz = rl.size();

    vector<Run> result;

    auto add_ans = [&](Run &&run) {
      if (run.l == l || run.r == r) {
        result.emplace_back(move(run));
      } else {
        glob_result.emplace_back(move(run));
      }
    };

    const int len_l = m - l, len_r = r - m;
    vector<Run> run_m(len_r / 2 + 1);
    for (auto &run: run_r) {
      if (run.l != m) {
        add_ans(move(run));
        continue;
      }
      run_m[run.period] = move(run);
    }
    for (auto &run: run_l) {
      if (run.r != m) {
        add_ans(move(run));
        continue;
      }
      const int period = run.period;
      if (z_rl[siz - period] == period) {
        if (run_m[period].period) {
          run.r = run_m[period].r;
          run_m[period] = Run{};
          add_ans(move(run));
        } else {
          run.r = m + period + z_rl[period];
          add_ans(move(run));
        }
      } else {
        run.r = m + z_rl[siz - period];
        add_ans(move(run));
      }
    }
    for (auto &run: run_m) {
      if (run.period) {
        const int period = run.period;
        if (z_rl[siz - period] == period) {
          if (2 * period <= len_l && z_rl[siz - 2 * period] >= period) continue;
          run.l = m - period - z_rl_rev[period];
          add_ans(move(run));
        } else {
          run.l = m - z_rl_rev[siz - period];
          add_ans(move(run));
        }
      }
    }
  }
```

```
    for (int period = 1; period <= len_l; ++period) {
      bool skip_r = 2 * period <= len_r && z_rl[period] >= period;
      bool skip_l = 2 * period <= len_l && z_rl[siz - 2 * period] >= period;
      if (z_rl[siz - period] == period) {
        if (skip_l || skip_r) continue;
        const int beg_pos = m - period - z_rl_rev[period];
        const int end_pos = m + period + z_rl[period];
        add_ans(Run{ period, beg_pos, end_pos });
      } else {
        if (!skip_r) {
          const int beg_pos = m - z_rl_rev[siz - period];
          const int end_pos = m + period + z_rl[period];
          if (end_pos - beg_pos >= 2 * period) {
            add_ans(Run{ period, beg_pos, end_pos });
          }
        }
        if (!skip_l) {
          const int beg_pos = m - period - z_rl_rev[period];
          const int end_pos = m + z_rl[siz - period];
          if (end_pos - beg_pos >= 2 * period) {
            add_ans(Run{ period, beg_pos, end_pos });
          }
        }
      }
    }
    return result;
  };
  const int n = s.size();
  vector<tuple<int, int, int>> runs;
  for (Run& run : div_conq(div_conq, 0, n)) {
    runs.emplace_back(run.l, run.r, run.period);
  }
  for (Run& run : glob_result) {
    runs.emplace_back(run.l, run.r, run.period);
  }
  sort(begin(runs), end(runs));
  runs.erase(
      unique(
          begin(runs), end(runs),
          [](auto& r1, auto& r2) {
            return get<0>(r1) == get<0>(r2) && get<1>(r1) == get<1>(r2);
          }
      ), end(runs)
  );
  vector<Run> res;
  for (auto& [l, r, t] : runs) res.emplace_back(t, l, r);
  return res;
}
```

6 Numeric

6.1 Симплекс

```
//#define ld double

const ld eps = 1e-9;

bool eq(ld x, ld y) {
  return abs(x - y) < eps;
}

bool ls(ld x, ld y) {
  return x + eps < y;
}

vector<ld> simplex(vector<vector<ld>> a) {
  int n = a.size() - 1;
  int m = a[0].size() - 1;
  vector<int> left(n + 1), up(m + 1);
  iota(up.begin(), up.end(), 0);
  iota(left.begin(), left.end(), m);
  auto pivot = [&](int x, int y) {
    swap(left[x], up[y]);
    ld k = a[x][y];
    a[x][y] = 1;
    vector<int> vct;
    for (int j = 0; j <= m; j++) {
      a[x][j] /= k;
      if (!eq(a[x][j], 0)) vct.push_back(j);
    }
    for (int i = 0; i <= n; i++) {
      if (eq(a[i][y], 0) || i == x) continue;
      k = a[i][y];
      a[i][y] = 0;
      for (int j : vct) a[i][j] -= k * a[x][j];
    }
  };
  while (1) {
    int x = -1;
    for (int i = 1; i <= n; i++) if (ls(a[i][0], 0) && (x == -1 || a[i][0] <
↪  a[x][0])) x = i;
    if (x == -1) break;
    int y = -1;
    for (int j = 1; j <= m; j++) if (ls(a[x][j], 0) && (y == -1 || a[x][j] <
↪  a[x][y])) y = j;
    if (y == -1) assert(0); // infeasible
    pivot(x, y);
  }
  while (1) {
    int y = -1;
    for (int j = 1; j <= m; j++) if (ls(0, a[0][j]) && (y == -1 || a[0][j] >
↪  a[0][y])) y = j;
    if (y == -1) break;
    int x = -1;
    for (int i = 1; i <= n; i++) if (ls(0, a[i][y]) && (x == -1 || a[i][0] /
↪  a[i][y] < a[x][0] / a[x][y])) x = i;
    if (x == -1) assert(0); // unbounded
    pivot(x, y);
  }
  vector<ld> ans(m + 1);
  for (int i = 1; i <= n; i++) if (left[i] <= m) ans[left[i]] = a[i][0];
  ans[0] = -a[0][0];
  return ans;
}
// j=1..m: x[j]>=0
// i=1..n: sum(j=1..m) A[i][j]*x[j] <= A[i][0]
// max sum(j=1..m) A[0][j]*x[j]
// res[0] is answer
// res[1..m] is certificate
```

6.2 FFT

```
namespace fft {
using dbl = double;  // works for max value (max(a)*max(b)*n) up to 1e14
↪  (multiply_mod with n up to 1e5)
// using dbl = long double;  // works for max value (max(a)*max(b)*n) up to 1e17
const dbl PI = acosl(-1.01);

struct Complex {
  dbl x, y;
  Complex(dbl x = 0, dbl y = 0) : x(x), y(y) {}
  Complex conj() const {
    return Complex(x, -y);
  }
};

Complex  operator *  (const Complex &a, const Complex &b) { return Complex(a.x
↪  * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
void  operator /= (  Complex &a, int n)  { a.x /= n; a.y /= n; }
Complex  operator /  (const Complex &a, int n)  { return Complex(a.x / n,
↪  a.y / n); }
Complex  operator +  (const Complex &a, const Complex &b) { return Complex(a.x
↪  + b.x, a.y + b.y); }
Complex  operator -  (const Complex &a, const Complex &b) { return Complex(a.x
↪  - b.x, a.y - b.y); }

string  to_string  (  const Complex &x) { return (string)"(" +
↪  std::to_string(x.x) + ", " + std::to_string(x.y) + ")"; };
ostream& operator << (ostream &o, const Complex &x) { return o << to_string(x);
↪  }

vector<Complex> buf1;
vector<Complex> buf2;

vector<Complex> w = {1, 1};
vector<int> reversed = {0};

void update_n(int n) {
  assert((n & (n - 1)) == 0);
  int cur = reversed.size();
  if (n <= cur) return;
  reversed.resize(n);
  w.resize(n + 1);
  while (cur < n) {
    for (int i = 0; i < cur; ++i)
      reversed[i] <<= 1;
    for (int i = cur; i < (cur << 1); ++i)
      reversed[i] = reversed[i - cur] ^ 1;
    for (int i = (cur << 1) - 2; i > 0; i -= 2)
      w[i] = w[i / 2];
    for (int i = 1; i < (cur << 1); i += 2)
      w[i] = Complex(cos(PI * i / cur), sin(PI * i / cur));
    cur *= 2;
  }
  w.back() = 1;
}

void fft_internal(vector<Complex> &v, int from, int n, bool inv) {
  update_n(n);
  int N = reversed.size();

  int d = __lg(N) - __lg(n);

  for (int i = 1; i < n; ++i)
    if (i < (reversed[i] >> d))
      swap(v[from + i], v[from + (reversed[i] >> d)]);

  for (int ln = 1; ln < n; ln <<= 1) {
    int step = (inv ? -N : N) / (ln * 2);
    for (int i = 0; i < n; i += (ln << 1)) {
      int ind = (inv ? N : 0);
      for (int j = 0; j < ln; ++j) {
        Complex y = v[from + i + j + ln] * w[ind];
        ind += step;
        v[from + i + j + ln] = v[from + i + j] - y;
        v[from + i + j]    = v[from + i + j] + y;
      }
    }
  }

  if (inv)
    for (int i = 0; i < n; ++i)
      v[from + i] /= n;
}

vector<Complex> fft(const vector<int> &v, int n = -1) { // only if you need this
  if (n == -1) {
    n = 1;
    while (n < v.size()) n <<= 1;
  }
  assert(v.size() <= n);
  buf1.assign(n, {0, 0});
  for (int i = 0; i < v.size(); ++i)
    buf1[i].x = v[i];
  fft_internal(buf1, 0, n, false);
  return vector<Complex>(buf1.begin(), buf1.end());
}
vector<long long> fft(const vector<Complex> &v) { // only if you need this
  assert(!v.empty() && (v.size() & (v.size() - 1)) == 0);
  buf1.resize(v.size());
```

```
  for (int i = 0; i < v.size(); ++i)
    buf1[i] = v[i];
  fft_internal(buf1, 0, buf1.size(), true);
  vector<long long> result(v.size());
  for (int i = 0; i < result.size(); ++i)
    result[i] = llround(buf1[i].x);
  return result;
}

vector<long long> multiply(const vector<int> &a, const vector<int> &b) {
  if (a.empty() || b.empty()) return {};
  int n = 2;
  while (n < a.size() + b.size() - 1) n <<= 1;

  buf1.assign(n, {0, 0});

  for (int i = 0; i < a.size(); ++i)
    buf1[i].x = a[i];
  for (int i = 0; i < b.size(); ++i)
    buf1[i].y = b[i];

  fft_internal(buf1, 0, n, false);

  for (int i = 0; i <= (n >> 1); ++i) {
    // a --fft--> a1 + a2*i
    // b --fft--> b1 + b2*i
    // fact: FFT(a)[k] = FFT(a)[n - k].conj()
    // using this we can get formulas for FFT(a) and FFT(b) from FFT(a+bi)

    int j = (n - i) & (n - 1);

    auto v = (buf1[i] + buf1[j].conj()) * (buf1[i] - buf1[j].conj()) / 4;
    swap(v.x, v.y);

    buf1[i] = v.conj();
    buf1[j] = v;
  }

  fft_internal(buf1, 0, n, true);

  vector<long long> result(a.size() + b.size() - 1);
  for (int i = 0; i < result.size(); ++i)
    result[i] = llround(buf1[i].x);
  return result;
}

vector<int> multiply_mod(const vector<int> &a, const vector<int> &b, int mod) {
  if (a.empty() || b.empty()) return {};
  int n = 2;
  while (n < a.size() + b.size() - 1) n <<= 1;

  buf1.assign(n * 2, {0, 0});
  for (int i = 0; i < a.size(); ++i) {
    buf1[i].x = a[i] & ((1 << 15) - 1);
    buf1[i].y = a[i] >> 15;
  }
  buf2.assign(n * 2, {0, 0});
  for (int i = 0; i < b.size(); ++i) {
    buf2[i].x = b[i] & ((1 << 15) - 1);
    buf2[i].y = b[i] >> 15;
  }

  fft_internal(buf1, 0, n, false);
  fft_internal(buf2, 0, n, false);

  for (int i = 0; i <= (n >> 1); ++i) {
    int j = (n - i) & (n - 1);

    Complex as = (buf1[i] + buf1[j].conj()) / 2;
    Complex bs = (buf2[i] + buf2[j].conj()) / 2;
    Complex al = (buf1[i] - buf1[j].conj()) / 2;
    Complex bl = (buf2[i] - buf2[j].conj()) / 2;

    Complex asbs = as * bs;
    Complex albs = al * bs;
    Complex asbl = as * bl; swap(asbl.x, asbl.y);
    Complex albl = al * bl; swap(albl.x, albl.y);

    buf1[i] = asbs + albl.conj();
    buf1[j] = asbs.conj() - albl;

    buf2[i] = asbl.conj() + albs;
    buf2[j] = asbl - albs.conj();
  }

  fft_internal(buf1, 0, n, true);
  fft_internal(buf2, 0, n, true);

  vector<int> result(a.size() + b.size() - 1);
  for (int i = 0; i < result.size(); ++i) {
    long long asbs = llround(buf1[i].x);
    long long albl = llround(buf1[i].y);
    long long asbl = llround(buf2[i].x);
    long long albs = llround(buf2[i].y);
    result[i] = (((albl % mod) << 30) + (((asbl + albs) % mod) << 15) + asbs) %
↪   mod;
  }
  return result;
}
} // fft
```

6.3 NTT

```
#define V vector
void nft(bool type, V<Mint>& a) {
  Mint G = 3;
  int n = int(a.size()), s = 0;
  while ((1 << s) < n) s++;
  assert(1 << s == n);

  static V<Mint> ep, iep;
  while (int(ep.size()) <= s) {
    ep.push_back(pow(G, Mint(-1).v / (1 << ep.size())));
    iep.push_back(ep.back().inv());
  }
  V<Mint> b(n);
  for (int i = 1; i <= s; i++) {
    int w = 1 << (s - i);
    Mint base = type ? iep[i] : ep[i], now = 1;
    for (int y = 0; y < n / 2; y += w) {
      for (int x = 0; x < w; x++) {
        auto l = a[y << 1 | x];
        auto r = now * a[y << 1 | x | w];
        b[y | x] = l + r;
        b[y | x | n >> 1] = l - r;
      }
      now *= base;
    }
    swap(a, b);
  }
}

// 200ms for z=2^20
V<Mint> multiply_nft(const V<Mint>& a, const V<Mint>& b) {
  int n = int(a.size()), m = int(b.size());
  if (!n || !m) return {};
  if (min(n, m) <= 50) { // maybe not needed (?)
    V<Mint> ans(n + m - 1); // MOD^2 opt is possible
    for (int i = 0; i < n; i++)
      for (int j = 0; j < m; j++) ans[i + j] += a[i] * b[j];
    return ans;
  }
  int lg = 0;
  while ((1 << lg) < n + m - 1) lg++;
  int z = 1 << lg;
  auto a2 = a, b2 = b;
  a2.resize(z);
  b2.resize(z);
  nft(false, a2);
  nft(false, b2);
  for (int i = 0; i < z; i++) a2[i] *= b2[i];
  nft(true, a2);
  a2.resize(n + m - 1);
  Mint iz = Mint(z).inv();
  for (int i = 0; i < n + m - 1; i++) a2[i] *= iz;
  return a2;
}
```

6.4 Hadamard

```
// careful with overflow, max value is max(res)*len(res), because of division
↪  in the end
// n = 2^k (адамар не определен иначе, код не работает)

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
    }
  }
  // precalc inverse and replace with multiplication?
  if (inv) for (auto& x : a) x /= a.size(); // XOR only
}
vi conv(vi a, vi b) {
  FST(a, 0); FST(b, 0);
  for (int i = 0; i < (int)a.size(); i++) a[i] *= b[i];
  FST(a, 1); return a;
}
```

6.5 Subset convolution

```
// res[i] = sum_{j = submask(i)} a[i] * b[i ^ j]
template<typename T>
vector<T> subset_convolution(const vector<T> &a, const vector<T> &b) {
  int n = a.size();
  assert((n & (n - 1)) == 0);
  assert(a.size() == b.size());
  static vector<vector<T>> buf1;
  static vector<vector<T>> buf2;
  static vector<T> buf3;

  buf1.assign(__lg(n) + 1, vector<T>(n, 0));
  buf2.assign(__lg(n) + 1, vector<T>(n, 0));

  for (int i = 0; i < n; ++i) {
    int cnt = __builtin_popcount(i);
    buf1[cnt][i] = a[i];
    buf2[cnt][i] = b[i];
  }

  auto submask_sum = [&](vector<T> &v) {
    for (int b = 0; (1 << b) < n; ++b)
      for (int i = 0; i < n; ++i)
        if (((i >> b) & 1) == 0)
          v[i ^ (1 << b)] += v[i];
  };

  for (int i = 0; i < buf1.size(); ++i) {
    submask_sum(buf1[i]);
    submask_sum(buf2[i]);
```

```
  }

  vector<T> res(n, 0);

  for (int i = 0; i < buf1.size(); ++i) {
    buf3.assign(n, 0);

    for (int j = 0; j <= i; ++j)
      for (int k = 0; k < n; ++k)
        buf3[k] += buf1[j][k] * buf2[i - j][k];

    for (int b = 0; (1 << b) < n; ++b)
      for (int i = 0; i < n; ++i)
        if (((i >> b) & 1) == 0)
          buf3[i ^ (1 << b)] -= buf3[i];

    for (int j = 0; j < n; ++j)
      if (__builtin_popcount(j) == i)
        res[j] = buf3[j];
  }

  return res;
}
```

6.6 Берлекэмп

```
//at least 2k elements required

//f(a, m) - m-th element of a, m is 1-based
//BM(a) - smallest recurrence
//if q = BM(a), then
//sum(q_i * a_i) = 0 true for any shift of a
//works in n^2(recurrenta) + k^2 * log(n)(n-th element)

//template<typename T>
//use T=Mint
vector<T> BM(vector<T> a) {
  vector<T> p = {1};
  vector<T> q = {1};
  int l = 0;
  for (int r = 1; r <= (int) a.size(); r++) {
    T delta = 0;
    for (int j = 0; j <= l; j++) {
      delta += a[r - 1 - j] * p[j];
    }
    q.insert(q.begin(), 0);
    if (delta != 0) {
      vector<T> t = p;
      if (q.size() > t.size()) {
        t.resize(q.size());
      }
      for (int i = 0; i < (int) q.size(); i++) {
        t[i] -= delta * q[i];
      }
      if (2 * l <= r - 1) {
        q = p;
        T od = 1 / delta;
        for (T& x : q) {
          x *= od;
        }
        l = r - l;
      }
      swap(p, t);
    }
  }
  assert((int) p.size() == l + 1);
//  assert(l * 2 + 30 < (int) a.size());
  reverse(p.begin(), p.end());
  return p;
}
template<typename T>
vector<T> mul(vector<T> a, vector<T> b) {
  vector<T> c(a.size() + b.size() - 1);
  for (int i = 0; i < (int)a.size(); i++) {
    for (int j = 0; j < (int)b.size(); j++) {
      c[i + j] = (c[i + j] + a[i] * b[j]);
    }
  }
  return c;
}
template<typename T>
vector<T> mem(vector<T> a, vector<T> b) {
  if (a.size() < b.size()) a.resize(b.size() - 1);

  T o = 1 / T(b.back());
  for (int i = (int)a.size() - 1; i >= (int)b.size() - 1; i--) {
    if (a[i] == 0) continue;
    T coef = o * (-a[i]);
    for (int j = 0; j < (int)b.size(); j++) {
      a[i - (int)b.size() + 1 + j] = a[i - (int)b.size() + 1 + j] + coef * b[j];
    }
  }
  while (a.size() >= b.size()) {
    assert(a.back() == 0);
    a.pop_back();
  }
  return a;
}
template<typename T>
vector<T> bin(ll n, vector<T> p) {
  vector<T> res(1, 1);
  vector<T> a(2); a[1] = 1;
  while (n) {
    if (n & 1) res = mem(mul(res, a), p);
    a = mem(mul(a, a), p);
    n >>= 1;
  }
  return res;
}
//m is 1-based
template<typename T>
T f(vector<T> t, ll m) {
  vector<T> v = BM(t);
  vector<T> o = bin(m - 1, v);
  T res = 0;
  for (int i = 0; i < (int)o.size(); i++) res = res + o[i] * t[i];
  return res;
}
```

6.7 Mint

```
// если модуль подается на вход, убрать все <> и раскомментировать нужные строки
using uint = unsigned int;
using ull = unsigned long long;
template <uint MD> struct ModInt {
  using M = ModInt;
  // static int MD;
  uint v;
  ModInt(ll _v = 0) { set_v(uint(_v % MD + MD)); }
  M& set_v(uint _v) {
    v = (_v < MD) ? _v : _v - MD;
    return *this;
  }
  explicit operator bool() const { return v != 0; }
  M operator-() const { return M() - *this; }
  M operator+(const M& r) const { return M().set_v(v + r.v); }
  M operator-(const M& r) const { return M().set_v(v + MD - r.v); }
  M operator*(const M& r) const { return M().set_v(uint((ull)v * r.v % MD)); }
  M operator/(const M& r) const { return *this * r.inv(); }
  M& operator+=(const M& r) { return *this = *this + r; }
  M& operator-=(const M& r) { return *this = *this - r; }
  M& operator*=(const M& r) { return *this = *this * r; }
  M& operator/=(const M& r) { return *this = *this / r; }
  bool operator==(const M& r) const { return v == r.v; }
  bool operator!=(const M& r) const { return v != r.v; }
  M inv() const;
  friend istream& operator>>(istream& is, M& r) { ll x; is >> x; r = M(x); return is; }
  friend ostream& operator<<(ostream& os, const M& r) { return os << r.v; }
};

template<uint MD>
ModInt<MD> pow(ModInt<MD> x, ll n) {
  ModInt<MD> r = 1;
  while (n) {
    if (n & 1) r *= x;
    x *= x;
    n >>= 1;
  }
  return r;
}

template<uint MD>
ModInt<MD> ModInt<MD>::inv() const { return pow(*this, MD - 2); }
// or copy egcd and {return egcd(MD, v, 1).second;}

// if MD is from input
// this line is necessary, read later as you wish
// int ModInt::MD;

using Mint = ModInt<998244353>;
// using Mint = double;

//fail for b = 1
typedef unsigned long long ul; typedef __uint128_t L;
namespace MF {
  int b;
  ul m;

  void init(int b_) {
    b = b_;
    m = (ul((L(1) << 64) / b));
  }

  int reduce(ul a) {
    ul q = (ul)((L(m)*a)>>64);
    int r = a-q*b;
    return r>=b?r-b:r;
  }
};
```

6.8 Гаусс вещественный

```
// Ax = b
// 0 -> no solution, 1 -> single solution, 2 -> infinite solutions (if 1,2: x contains any)
// operations: (n^2 m)/2 если у системы есть решение, убрать эпсы
int gauss(vector<vector<ld>> A, vector<ld> b, vector<ld>& x, const ld eps = 1e-9) {
  vector<int> in(A.size(), -1);
  for (int i = 0; i < (int)A.size(); i++) {
    int best = 0;
    for (int j = 0; j < (int)A[i].size(); j++) {
      if (abs(A[i][j]) > abs(A[i][best])) { // any non-zero if Mint
        best = j;
      }
    }
    if (abs(A[i][best]) <= eps) { // == 0
      if (abs(b[i]) > eps) { // != 0
        return 0;
      }
      continue;
    }
    in[i] = best;
    for (int l = i + 1; l < A.size(); l++) {
      if (l != i) {
        ld cf = -A[l][in[i]] / A[i][in[i]];
        b[l] += cf * b[i];
        for (int j = 0; j < A[i].size(); j++) {
```

```
            A[l][j] += cf * A[i][j];
          }
        }
      }
    }
    x.assign(A[0].size(), 0);
    int cnt = 0;
    for (int i = (int)A.size() - 1; i >= 0; i--) {
      if (in[i] != -1) {
        cnt++;
        ld cur = b[i];
        for (int j = 0; j < (int)A[i].size(); j++) {
          cur -= A[i][j] * x[j];
        }
        x[in[i]] = cur / A[i][in[i]];
      }
    }
    return (cnt != (int)A[0].size()) ? 2 : 1;
}
```

6.9 Линал

```
Mint determinant(vector<vector<Mint>>& a) {
    int n = int(a.size());
    Mint ans = 1;
    for (int row = 0; row < n; row++) {
      for (int col = row; col < n; col++) {
        if (a[row][col] != 0) {
          if (col != row) {
            for (int i = row; i < n; i++) {
              swap(a[i][row], a[i][col]);
            }
            ans = -ans;
          }
          break;
        }
      }
      if (a[row][row] == 0) {
        return 0;
      }
      ans *= a[row][row];
      Mint cf = Mint(1) / a[row][row];
      for (int i = row + 1; i < n; i++) {
        if (a[i][row] != 0) {
          Mint cur = a[i][row] * cf;
          for (int col = row; col < n; col++) {
            a[i][col] -= a[row][col] * cur;
          }
        }
      }
    }
    return ans;
}

// n <= m, O(n^2 m)
int get_rank(vector<vector<Mint>>& a) {
    int n = int(a.size()), m = int(a[0].size());
    int rank = n;
    for (int row = 0; row < n; row++) {
      for (int col = row; col < m; col++) {
        if (a[row][col] != 0) {
          if (col != row) {
            for (int i = row; i < n; i++) {
              swap(a[i][row], a[i][col]);
            }
          }
          break;
        }
      }
      if (a[row][row] == 0) {
        rank--;
        continue;
      }
      Mint cf = Mint(1) / a[row][row];
      for (int i = row + 1; i < n; i++) {
        if (a[i][row] != 0) {
          Mint cur = a[i][row] * cf;
          for (int col = row; col < m; col++) {
            a[i][col] -= a[row][col] * cur;
          }
        }
      }
    }
    return rank;
}

vector<Mint> characteristic_polynomial(vector<vector<Mint>>& mat){
    int n = int(mat.size());
    if (n == 0) return {Mint(1)};
    vector<Mint> T(n);
    for (int y = 1; y < n; y++) {
      int y1 = y;
      while (y1 < n && mat[y1][y - 1] == 0) y1++;
      if (y1 == n) continue;
      if (y != y1) {
        for (int col = 0; col < n; col++) swap(mat[y][col], mat[y1][col]);
        for (int row = 0; row < n; row++) swap(mat[row][y], mat[row][y1]);
      }
      T[y] = Mint(1) / mat[y][y - 1];
      for (int y2 = y + 1; y2 < n; y2++) T[y2] = T[y] * mat[y2][y - 1];
      for (int y2 = y + 1; y2 < n; y2++) for (int x = y - 1; x < n; x++)
↪       mat[y2][x] -= mat[y][x] * T[y2];
      for (int y2 = 0; y2 < n; y2++) for (int x = y + 1; x < n; x++) mat[y2][y] +=
↪       mat[y2][x] * T[x];
    }
    for (int y = 0; y < n; y++) {
      Mint tmp = 1;
      for (int x = y + 1; x < n; x++) {
        mat[y][x] *= (tmp *= -mat[x][x - 1]);
      }
    }
    vector<vector<Mint>> dp(n+1, vector<Mint>(n + 1, 0));
    dp[0][0] = 1;
    for (int y = 0; y < n; y++) {
      for (int x = 0; x <= y; x++) dp[y + 1][x + 1] -= dp[y][x];
      for (int x = 0; x <= y; x++) dp[y + 1][x] += dp[y][x] * mat[y][y];
      for (int y2 = 0; y2 < y; y2++) for (int x = 0; x <= y2; x++) dp[y + 1][x] +=
↪       dp[y2][x] * mat[y2][y];
    }
    vector<Mint> res(n + 1);
    for (int i = 0; i <= n; i++) res[i] = ((n % 2 == 1) ? -dp[n][i] : dp[n][i]);
    return res;
}

bool inverse(vector<vector<Mint>>& a, vector<vector<Mint>>& res) {
    int n = int(a.size());
    res.assign(n, vector<Mint>(n, 0));
    for (int i = 0; i < n; i++) {
      res[i][i] = 1;
    }
    for (int row = 0; row < n; row++) {
      for (int row2 = row; row2 < n; row2++) {
        if (a[row2][row] != 0) {
          if (row2 != row) {
            for (int i = 0; i < n; i++) {
              swap(a[row][i], a[row2][i]);
              swap(res[row][i], res[row2][i]);
            }
          }
          break;
        }
      }
      if (a[row][row] == 0) {
        return false;
      }
      Mint cf = Mint(1) / a[row][row];
      for (int col = 0; col < n; col++) {
        a[row][col] *= cf;
        res[row][col] *= cf;
      }
      for (int i = 0; i < n; i++) {
        if (i != row && a[i][row] != 0) {
          Mint cur = a[i][row];
          for (int col = row; col < n; col++) {
            a[i][col] -= a[row][col] * cur;
          }
          for (int col = 0; col < n; col++) {
            res[i][col] -= res[row][col] * cur;
          }
        }
      }
    }
    return true;
}

// Ax = b, O(n^2 m)
// -1 -> no solution, otherwise dimension of solution space
int gauss(vector<vector<Mint>>& A, vector<Mint>& b, vector<Mint>& x,
↪   vector<vector<Mint>>& basis) {
    int n = int(A.size()), m = int(A[0].size());
    vector<int> in(n, -1);
    for (int i = 0; i < n; i++) {
      int best = 0;
      for (int j = 0; j < m; j++) {
        if (A[i][j] != 0) {
          best = j;
          break;
        }
      }
      if (A[i][best] == 0) {
        if (b[i] != 0) {
          return -1;
        }
        continue;
      }
      in[i] = best;
      Mint mul = Mint(1) / A[i][in[i]];
      for (int l = i + 1; l < n; l++) {
        if (l != i) {
          Mint cf = A[l][in[i]] * mul;
          b[l] -= cf * b[i];
          for (int j = 0; j < m; j++) {
            A[l][j] -= cf * A[i][j];
          }
        }
      }
    }
    x.assign(m, 0);
    int cnt = 0;
    for (int i = n - 1; i >= 0; i--) {
      if (in[i] != -1) {
        cnt++;
        Mint cur = b[i];
        for (int j = 0; j < (int)A[i].size(); j++) {
          cur -= A[i][j] * x[j];
        }
        x[in[i]] = cur / A[i][in[i]];
      }
    }
    if (m > cnt) {
      basis.assign(m - cnt, vector<Mint>(m));
      vector<bool> empty(m, true);
      for (int i : in) if (i != -1) empty[i] = false;
      int pos = 0;
      for (int i = 0; i < m; i++) if (empty[i]) basis[pos++][i] = 1;
      vector<tuple<int, int, Mint>> history;
      for (int i = 0; i < n; i++) {
        if (in[i] != -1) {
          Mint mul = Mint(1) / A[i][in[i]];
```

```
        for (int col = 0; col < m; col++) {
          if (col != in[i] && A[i][col] != 0) {
            Mint cf = A[i][col] * mul;
            history.emplace_back(in[i], col, cf);
            for (int row = 0; row <= i; row++) {
              A[row][col] -= A[row][in[i]] * cf;
            }
          }
        }
      }
    }
    reverse(history.begin(), history.end());
    for (auto [i, j, cf] : history) {
      for (int row = 0; row < (int)basis.size(); row++) {
        basis[row][i] -= basis[row][j] * cf;
      }
    }
  }
  return m - cnt;
}
```

7 Графы

7.1 Дерево доминаторов

```
vector<vector<int>> dominator_tree(vector<vector<int>> g, int root) {
  int n = g.size();
  vector<int> p(n);
  for (int i = 0; i < n; ++i) {
    p[i] = i;
  }
  swap(p[root], p[0]);
  swap(g[0], g[root]);
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < g[i].size(); ++j) {
      g[i][j] = p[g[i][j]];
    }
  }
  vector<vector<int>> tree(n);
  vector<vector<int>> bucket(n);
  vector<int> sdom(n), dom(n), par(n), label(n), dsu(n);
  vector<vector<int>> gi(n);
  vector<int> arr(n, -1), rev(n);
  int tm = 0;

  function<int(int, int)> ask = [&](int u, int x) {
    if (u == dsu[u]) return x ? -1 : u;
    int v = ask(dsu[u], x + 1);
    if (v < 0) return u;
    if (sdom[label[dsu[u]]] < sdom[label[u]])
      label[u] = label[dsu[u]];
    dsu[u] = v;
    return x ? v : label[u];
  };
  auto un = [&](int u, int v) {
    dsu[v] = u;
  };

  function<void(int)> dfs = [&](int v) {
    arr[v] = tm;
    rev[tm] = v;
    label[tm] = sdom[tm] = dsu[tm] = tm;
    ++tm;
    for (int k : g[v]) {
      if (arr[k] == -1) {
        dfs(k);
        par[arr[k]] = arr[v];
      }
      gi[arr[k]].pb(arr[v]);
    }
  };
  dfs(0);
  assert(tm == n);  // connected

  for (int i = n - 1; i >= 0; --i) {
    for (int j : gi[i]) {
      sdom[i] = min(sdom[i], sdom[ask(j, 0)]);
    }
    if (i != 0) bucket[sdom[i]].pb(i);
    for (int w : bucket[i]) {
      int v = ask(w, 0);
      if (sdom[v] == sdom[w]) dom[w] = sdom[w];
      else dom[w] = v;
    }
    if (i != 0) un(par[i], i);
  }
  for (int i = 1; i < n; ++i) {
    if (dom[i] != sdom[i])
      dom[i] = dom[dom[i]];
    tree[rev[dom[i]]].pb(rev[i]);
    tree[rev[i]].pb(rev[dom[i]]);
  }

  swap(tree[root], tree[0]);
  for (int i = 0; i < tree.size(); ++i) {
    for (int j = 0; j < tree[i].size(); ++j) {
      tree[i][j] = p[tree[i][j]];
    }
  }

  return tree;
}
```

7.2 Пересечение матроидов

```
//try swaping oracle1 and oracle2 in case of tl
//works really slow, speedup possible
//use dijkstra in case of costs (minimize pair (weight, edges_num))

bool oracle1(vector<int>& v) {}
bool oracle2(vector<int>& v) {}

vector<int> inter() {
  vector<int> s;
  vector<bool> f(k, false);
  while (s.size() < k) {
    vector<bool> vy(k, false);
    queue<int> q;
    vector<int> dist(k, (int)(1e9));
    vector<int> pr(k);
    bool fnd = false;
    for (int i = 0; i < k; i++)
      if (!f[i]) {
        s.push_back(i);
        if (oracle2(s)) // Y - set of (i + s) in I_2
          vy[i] = true;
        if (oracle1(s)) { // X - set of (i + s) in I_1
          if (vy[i]) {
            f[i] = true;
            fnd = true;
            break;
          }
          dist[i] = 0;
          pr[i] = -1;
          q.push(i);
        }
        s.pop_back();
      } // we should find any shortest path from X to Y
    if (fnd) // (s - v + u) in I_2 --> from right to left
      continue; // (s - v + u) in I_1 --> from left to right
    int l = -1;
    while (!q.empty()) {
      int p = q.front();
      q.pop();
      if (vy[p]) {
        l = p;
        break;
      }
      if (f[p]) {
        vector<int> cur = s;
        for (int i = 0; i < s.size(); i++)
          if (cur[i] == p) {
            swap(cur[i], cur[(int)s.size() - 1]);
            cur.pop_back();
            break;
          }
        for (int i = 0; i < k; i++)
          if (!f[i]) {
            cur.push_back(i);
            if (oracle1(cur))
              if (dist[i] > dist[p] + 1) {
                dist[i] = dist[p] + 1;
                pr[i] = p;
                q.push(i);
              }
            cur.pop_back();
          }
      } else {
        vector<int> cur = s;
        for (int i = 0; i < s.size(); i++) {
          swap(cur[i], cur[(int)s.size() - 1]);
          cur.pop_back();
          cur.push_back(p);
          if (oracle2(cur))
            if (dist[s[i]] > dist[p] + 1) {
              dist[s[i]] = dist[p] + 1;
              pr[s[i]] = p;
              q.push(s[i]);
            }
          cur.pop_back();
          cur.push_back(s[i]);
          swap(cur[i], cur[(int)s.size() - 1]);
        }
      }
    }
    if (l == -1)
      return s;
    while (l != -1) {
      f[l] = !f[l];
      l = pr[l];
    }
    s.clear();
    for (int i = 0; i < k; i++)
      if (f[i])
        s.push_back(i);
  }
  return s;
}
```

7.3 Два китайца

```
// minimum directed tree
// O(m log m)
// weights must be non-negative, otherwise just increase all of them by const

const int SZ = 2e5 + 7;
int n, m, root, from[SZ], to[SZ], val[SZ]; // size m
int rt[SZ], ch[SZ][2], h[SZ], mrk[SZ]; // size bigger >= 2m
int dfn[SZ], sz[SZ], out[SZ], cnode, ts;
vector<int> sml[SZ], ansseq;

void mark(int x, int v) {
  mrk[x] += v;
  val[x] += v;
}

void down(int x) {
  if (mrk[x]) {
    mark(ch[x][0], mrk[x]);
    mark(ch[x][1], mrk[x]);
    mrk[x] = 0;
```

```
  }
}

int merge(int p, int q) {
  if (!p || !q) return p + q;
  down(p); down(q);
  if (val[p] > val[q]) swap(p, q);
  ch[p][1] = merge(ch[p][1], q);
  if (h[ch[p][1]] > h[ch[p][0]]) swap(ch[p][0], ch[p][1]);
  h[p] = h[ch[p][1]] + 1;
  return p;
}

void del(int& x) {
  down(x);
  x = merge(ch[x][0], ch[x][1]);
}

struct dsu {
  int fa[SZ];
  dsu() {
    iota(fa, fa + SZ, 0);
  }
  int find(int x) {
    return fa[x] == x ? x : (fa[x] = find(fa[x]));
  }
  void merge(int p, int q) {
    fa[find(q)] = find(p);
  }
} connect, extnode;

void dfs(int x) {
  dfn[x] = ++ts;
  sz[x] = 1;
  for (auto t : sml[x]) {
    dfs(t);
    sz[x] += sz[t];
  }
}

void getans(int x, int idx) {
  if (x <= n) {
    ansseq.push_back(idx);
    return;
  }
  for (auto t : sml[x]) {
    getans(t, dfn[to[idx]] >= dfn[t] && dfn[to[idx]] < dfn[t] + sz[t] ? idx :
↪  out[t]);
  }
}

/*
 * numeration from 1 !!! rooted in root (1 <= root <= n)
 * m edges (from[i], to[i], val[i]) (1 <= i <= m, 1 <= from, to <= n, val >= 0)
 * -1: NOANS; ansseq -> edges indices list
 */
ll directed_mst() {
  ll ans = 0;
  cnode = n;
  for (int i = 1; i <= m; ++i) {
    rt[to[i]] = merge(rt[to[i]], i);
  }
  int beg = 1;
  while (beg <= cnode) {
    int t = beg++;
    if (t == root) continue;
    while (rt[t] && extnode.find(from[rt[t]]) == extnode.find(t)) del(rt[t]);
    if (!rt[t]) {
      return -1;
    }
    out[t] = rt[t];
    ans += val[out[t]];
    mark(rt[t] , -val[out[t]]);
    if (connect.find(t) == connect.find(from[out[t]])) {
      ++cnode;
      connect.merge(t, cnode);
      while (t != cnode) {
        extnode.merge(cnode, t);
        sml[cnode].push_back(t);
        rt[cnode] = merge(rt[cnode], rt[t]);
        t = extnode.find(from[out[t]]);
      }
    } else {
      connect.merge(t, from[out[t]]);
    }
  }
  for (int i = cnode; i; --i) {
    if (!dfn[i] && i != root) {
      dfs(i);
      getans(i, out[i]);
    }
  }
  return ans;
}
```

7.4 Двусвязность

```
// блоки точек сочленения, красит ребра
namespace bicon {
  // петли красятся в -1, остальные в 0..maxcol-1
  bool vis[maxn];
  int fup[maxn], tin[maxn];
  int tnow;
  vector<pair<int, int>> e[maxn];
  vector<int> col;
  int maxcol;

  // v точка сочленения если есть сын fup[u] >= tin[v]
  // отдельно корень точка сочленения если хотя бы 2 сына
  void pfs(int v, int pid) {
    vis[v] = 1;
    tin[v] = fup[v] = tnow++;
    for (auto [u, id] : e[v])
      if (id != pid) {
        if (!vis[u]) {
          pfs(u, id);
          fup[v] = min(fup[v], fup[u]);
        } else {
          fup[v] = min(fup[v], tin[u]);
        }
      }
  }

  // если надо дерево блоков точек сочленения: соединяем точку
↪  сочленения с цветом инцидентных ребер
  void dfs(int v, int color, int pid) {
    vis[v] = 1;
    for (auto [u, id] : e[v])
      if (id != pid) {
        if (!vis[u]) {
          int c = color;
          if (fup[u] >= tin[v])
            c = maxcol++;
          col[id] = c;
          dfs(u, c, id);
        } else if (tin[u] < tin[v])
          col[id] = color;
      }
  }

  vector<int> get(int n, vector<pair<int, int>> ed) {
    tnow = 0;
    col.assign(ed.size(), -1);
    for (int it = 0; it < ed.size(); it++) {
      auto [v, u] = ed[it];
      e[v].emplace_back(u, it);
      e[u].emplace_back(v, it);
    }
    for (int i = 0; i < n; i++)
      if (!vis[i])
        pfs(i, -1);
    fill(vis, vis + n, 0);
    maxcol = 0;
    for (int i = 0; i < n; i++)
      if (!vis[i]) {
        // maxcol++ случится в корне
        dfs(i, maxcol, -1);
      }
    return col;
  }
}
```

7.5 Эйлеров цикл

```
// в e лежат пары вершин {u, v} (с нуля), в g[u] индексы ребер,
↪  инцидентных u
// если компонента одна, то go(0), иначе надо ставить пометки на
↪  вершины и запускать несколько раз, как дфс
// возвращает !развернутый! вершинный путь в path, включая оба конца
// в edges !развернутый! реберный путь (убрать его или вершинный,
↪  если не надо)

vector<pair<int, int>> e(m);
vector<vector<int>> g(n);  // g[i] contains indices of edges from i in e
vector<bool> ue(m, false);
vector<int> path;
vector<int> ind(n, -1);

void go(int v, int eid = -1) {
  while (ind[v] + 1 < g[v].size()) {
    if (!ue[g[v][++ind[v]]]) {
      ue[g[v][ind[v]]] = true;
      go(e[g[v][ind[v]]].first ^ e[g[v][ind[v]]].second ^ v, g[v][ind[v]]);
    }
  }
  if (eid != -1)
    edges.push_back(eid);
  path.push_back(v);
}
```

7.6 Link Cut

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

```
    swap(pp, y->pp);
  }
  void splay() { /// Splay this up to the root. Always finishes without flip
↪  set.
    for (push_flip(); p; ) {
      if (p->p) p->p->push_flip();
      p->push_flip(); push_flip();
      int c1 = up(), c2 = p->up();
      if (c2 == -1) p->rot(c1, 2);
      else p->p->rot(c2, c1 != c2);
    }
  }
  Node* first() { /// Return the min element of the subtree rooted at this,
↪  splayed to the top.
    push_flip();
    return c[0] ? c[0]->first() : (splay(), this);
  }
};

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

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

7.7 Хордальный граф

```
// Properties:
// - Union of cliques is graph G
// - (K1 inter K2) is subset of all cliques on path between K1 and K2
// - all cliques in which vertex v lies is a subtree
// - K1, K2, ... - all cliques in rooted tree topsort order:
//   K_j inter (K_1 union K_2 union ... K_{j-1}) subset of K[par[j]]
void chordaltree(const vector<vector<int>>& e) {
  int n = e.size();
  vector<int> mark(n);
  set<pair<int, int>> st;
  for (int i = 0; i < n; i++) st.insert({-mark[i], i});
  vector<int> vct(n); // will be the perfect order, all vertices after v
↪  (connected with it) are clique
  vector<pair<int, int>> ted; // edges of the rooted clique tree (from --> to).
↪  0 is root (and fictive)
  vector<vector<int>> who(n);
  vector<vector<int>> verts(1); // list of vertices of each clique in the tree
  vector<int> cliq(n, -1); // for each vertex id of the clique in which this
↪  vertex lies
  cliq.emplace_back(0);
  vector<int> last(n + 1, n);
  int prev = n + 1;
  for (int i = n - 1; i >= 0; i--) {
    int x = st.begin()->second;
    st.erase(st.begin());
    if (mark[x] <= prev) {
      vector<int> cur = who[x];
      cur.emplace_back(x);
      verts.emplace_back(cur);
      ted.emplace_back(cliq[last[x]], (int)verts.size() - 1);
    } else {
      verts.back().emplace_back(x);
    }
    for (int y : e[x]) {
      if (cliq[y] != -1) continue;
      who[y].emplace_back(x);
      st.erase({-mark[y], y});
      mark[y]++;
      st.insert({-mark[y], y});
      last[y] = x;
    }
    prev = mark[x];
    vct[i] = x;
    cliq[x] = (int)verts.size() - 1;
  }
}
```

7.8 K-ый путь

```
//find k smallest paths from s to t in digraph O(nlogn+klogk)
// if weights>=0, then dijkstra will do the work
// otherwise provide topsort order
struct X {
  ll w;
  int to;
};

bool operator<(X a, X b) {
  return a.w < b.w;
}

struct heap {
  X w;
  int h;
  heap *l, *r;
  heap(X w_) {
    w = w_;
    l = r = 0;
    h = 1;
  }
};

typedef heap* ptr;

ptr create(X w) {
  return new heap(w);
}

ptr merge(ptr t1, ptr t2) {
  if (t1 == 0 && t2 == 0) return 0;
  if (t1 == 0) return t2;
  if (t2 == 0) return t1;
  if (t2->w < t1->w) swap(t1, t2);
  ptr z = new heap(*t1);
  z->r = merge(t1->r, t2);
  if (z->l == 0 || z->l->h < z->r->h)
    swap(z->l, z->r);
  z->h = 1 + (z->r == 0 ? 0 : z->r->h);
  return z;
}

ptr pop(ptr t) {
  if (t != 0) return merge(t->l, t->r);
  return 0;
}

ptr push(ptr t, X w) {
  return merge(t, create(w));
}

struct edge {
  int to, w, i;
};

vector<ll> solve(vector<vector<pair<int, int>>> e_, int k, int s = 0, int t =
↪  -1, vector<int> order = {}) {
  // (to, w)
  int n = e_.size();
  if (t == -1) t = n - 1;
  vector<vector<edge>> e(n);
  {
    int cnt = 0;
    for (int i = 0; i < n; i++)
      for (auto j : e_[i])
        e[i].push_back({j.first, j.second, cnt++});
  }
  vector<vector<edge>> eo(n);
  for (int i = 0; i < n; i++)
    for (auto j : e[i])
      eo[j.to].push_back({i, j.w, j.i});
  const ll llinf = 1e18;
  vector<ll> ds(n, llinf);
  vector<int> par(n, -1);
  vector<int> prr(n, -1);
  {
    ds[t] = 0;
    if (order.empty()) {
      for (int i = 0; i < n; i++)
        for (auto s : e[i])
          assert(s.w >= 0);
      priority_queue<pair<ll, int>, vector<pair<ll, int>>, greater<pair<ll,
↪  int>>> que;
      que.push({0, t});
      while (!que.empty()) {
        auto [w, v] = que.top();
        que.pop();
        if (w != ds[v])
          continue;
        for (auto i: eo[v])
          if (ds[i.to] > ds[v] + i.w) {
            ds[i.to] = ds[v] + i.w;
            par[i.to] = i.i;
            prr[i.to] = v;
            que.push({ds[i.to], i.to});
          }
      }
    } else {
      assert(order.size() == n);
      vector<int> ob(n);
      for (int i = 0; i < n; i++)
        ob[order[i]] = i;
      for (int i = 0; i < n; i++)
```

```
        for (auto s : e[i])
          assert(ob[i] < ob[s.to]);
      reverse(order.begin(), order.end());
      for (int v : order)
        if (ds[v] != llinf) {
          for (auto i: eo[v])
            if (ds[i.to] > ds[v] + i.w) {
              ds[i.to] = ds[v] + i.w;
              par[i.to] = i.i;
              prr[i.to] = v;
            }
        }
    }
    vector<vector<int>> ces(n);
    for (int i = 0; i < n; i++)
      if (prr[i] != -1)
        ces[prr[i]].push_back(i);
    vector<ptr> g(n);
    {
      vector<int> que;
      que.push_back(t);
      for (int it = 0; it < que.size(); it++) {
        int v = que[it];
        for (int i : ces[v])
          que.push_back(i);
        if (ds[v] != llinf) {
          g[v] = 0;
          for (auto i : e[v])
            if (ds[i.to] != llinf) {
              if (i.i == par[v])
                g[v] = merge(g[v], g[i.to]);
              else
                g[v] = push(g[v], X{i.w - ds[v] + ds[i.to], i.to});
            }
        }
      }
    }
    vector<ll> ans;
    priority_queue<pair<ll, ptr>, vector<pair<ll, ptr>>, greater<pair<ll, ptr>>>
    ↪ que;
    if (ds[s] != llinf)
      que.push({ds[s], create({ds[s], s})});
    while (!que.empty() && ans.size() < k) {
      auto [w, h] = que.top();
      que.pop();
      ans.push_back(w);
      int v = h->w.to;
      if (g[v] != 0)
        que.push({w + g[v]->w.w, g[v]});
      if (h->l != 0)
        que.push({w + h->l->w.w - h->w.w, h->l});
      if (h->r != 0)
        que.push({w + h->r->w.w - h->w.w, h->r});
    }
    while (ans.size() < k)
      ans.push_back(-1);
    return ans;
}
```

8 Потоки

8.1 Диниц

```
// поток с ограничениями: создаем s', t' новые исток сток, добавляем
↪  (важно) ребро t->s [0;inf]
// ребро a->b [l;r] заменяем на s'->b [0;l], a->t' [0;l], a->b [0;r-l]
// поток как бы течет s'->b->t->s->a->t'
// проверяем насыщенность ребер s'
// если надо найти мин поток, делаем бинпоиск по cap на ребре t->s

//useful to remove scaling for huge/special graphs
//works in (dinic iterations) n * nm (at max m augmenting paths, each finds in
↪  n)
//с масштабированием работает за nm log(C)
//e sqrt(e) для единичной сети, e sqrt(v) для потока
template<typename T = int>
struct Dinic {
  struct Edge {
    int v, u;
    T f, c;
  };
  vector<Edge> ed;
  vector<vector<int>> e;
  int n, st, fin;
  T flow;

  vector<int> ptr, ds, que;

  Dinic(int n_, int st_, int fin_) : n(n_), st(st_), fin(fin_), flow(0) {
    e.resize(n);
    ptr.resize(n);
    ds.resize(n);
    que.resize(n);
  }

  void add(int v, int u, T c, T bc = 0, T f = 0) {
    int w = int(ed.size());
    ed.push_back({v, u, f, c});
    ed.push_back({u, v, -f, bc});
    e[v].push_back(w);
    e[u].push_back(w + 1);
  }

  bool bfs(T scale) {
    fill(ds.begin(), ds.end(), -1);
    ds[st] = 0;
    int sz = 1;
    que[0] = st;
    for (int it = 0; it < sz; it++) {
      int v = que[it];
      for (int i : e[v])
        if (ed[i].f + scale <= ed[i].c && ds[ed[i].u] == -1) {
          ds[ed[i].u] = ds[v] + 1;
          if (ed[i].u == fin)
            return 1;
          que[sz++] = ed[i].u;
        }
    }
    return 0;
  }

  T dfs(int v, T w, T cs) {
    if (v == fin)
      return w;
    int &i = ptr[v];
    while (i >= 0) {
      auto &t = ed[e[v][i]];
      if (ds[t.u] == ds[v] + 1 && t.f + cs <= t.c) {
        T o = dfs(t.u, min(w, t.c - t.f), cs);
        if (o) {
          t.f += o;
          ed[e[v][i] ^ 1].f -= o;
          return o;
        }
      }
      i--;
    }
    return 0;
  }

  T get(int scale = 30) { // scale=0 for no scale
    for (T i = ((T(1)) << scale); i > 0; i >>= 1)
      while (bfs(i)) {
        for (int j = 0; j < n; j++)
          ptr[j] = int(e[j].size()) - 1;
        while (T w = dfs(st, numeric_limits<T>::max(), i))
          flow += w;
      }
    return flow;
  }

  T go(int v, T w) { // не нужно если просто поток
    if (v == fin)
      return w;
    int &i = ptr[v];
    while (i >= 0) {
      auto &t = ed[e[v][i]];
      if (t.f > 0) {
        T o = go(t.u, min(w, t.f));
        if (o > 0) {
          // edge t has flow o in this path
          t.f -= o;
          ed[e[v][i] ^ 1].f += o;
          return o;
        }
      }
      i--;
    }
    return 0;
  }

  void decompose() { // разбить поток на пути
    for (int j = 0; j < n; j++)
      ptr[j] = int(e[j].size()) - 1;
    while (1) {
      T w = go(st, numeric_limits<T>::max());
      if (w == 0)
        break;
    }
  }

  // для разреза пускаешь функцию, аналогичную go, с условием t.f <
  ↪  t.c
};
```

8.2 Min-cost-max-flow

```
// time complexity - Ford-Bellman(EV) + Dijka(ElogV or V^2) * |f|
// use_que for using priority_queue in dijka, otherwise n^2
// primal dual: instead of augmenting along one path, use max flow algorithm to
↪  augment through all paths, then change potentials (shortest distance) array
↪  as usual with Dijka
// works in max_cost * Dinic (each time d[t] increases by 1)
template<typename F = int, typename C = int>
struct mcmf {
  struct Edge {
    int v, u;
    F f, c;
    C cost;
  };

  vector<Edge> ed;
  vector<vector<int>> e;
  int n, st, fin;
  F flow;
  C cost;

  vector<C> pot, npot;
  vector<int> par, tak;

  priority_queue<pair<C, int>> que;

  mcmf(int n_, int st_, int fin_) : n(n_), st(st_), fin(fin_), flow(0), cost(0)
  ↪  {
    e.resize(n);
    pot.resize(n);
    npot.resize(n);
    par.resize(n);
    tak.resize(n);
```

```
}

void add(int v, int u, F c, C cost_, F bc = 0, F f = 0) {
  int w = ed.size();
  ed.push_back({v, u, f, c, cost_});
  ed.push_back({u, v, -f, bc, -cost_});
  e[v].push_back(w);
  e[u].push_back(w + 1);
}

bool ford_bellman() {
  const C INF = numeric_limits<C>::max();
  fill(pot.begin(), pot.end(), INF);
  pot[st] = 0;
  bool upd;
  for (int i = 0; i < n; i++) {
    upd = 0;
    for (int j1 = 0; j1 < int(ed.size()); j1++) {
      auto &j = ed[j1];
      if (pot[j.v] != INF && j.f < j.c && pot[j.u] > pot[j.v] + j.cost) {
        pot[j.u] = pot[j.v] + j.cost;
        par[j.u] = j1;
        upd = 1;
      }
    }
    if (!upd) break;
  }
  assert(!upd);
  return pot[fin] != INF;
}

bool dijka(bool use_que) {
  const C INF = numeric_limits<C>::max();
  // if you don't want to use priority_queue, don't write first bracket
  // since use_que always true
  if (!use_que) {
    fill(tak.begin(), tak.end(), 0);
    fill(npot.begin(), npot.end(), INF);
    npot[st] = 0;
    for (int it = 0; it < n; it++) {
      int v = -1;
      C vw = INF;
      for (int i = 0; i < n; i++)
        if (!tak[i] && vw > npot[i])
          v = i, vw = npot[i];
      if (v == -1) break;
      tak[v] = 1;
      for (int i : e[v])
      if (ed[i].f < ed[i].c) {
        auto &w = ed[i];
        assert(pot[w.u] != INF);
        assert(pot[v] != INF);
        C uw = w.cost + pot[v] - pot[w.u];
        assert(uw >= 0);
        uw += npot[v];
        if (npot[w.u] > uw) {
          npot[w.u] = uw;
          par[w.u] = i;
        }
      }
    }
    for (int i = 0; i < n; i++)
      pot[i] = (npot[i] == INF ? INF : pot[i] + npot[i]);
  } else {
    fill(tak.begin(), tak.end(), 0);
    fill(npot.begin(), npot.end(), INF);
    npot[st] = 0;
    que.push({0, st});
    while (!que.empty()) {
      int v = que.top().sc;
      que.pop();
      if (tak[v]) continue;
      tak[v] = 1;
      for (int i : e[v])
      if (ed[i].f < ed[i].c) {
        auto &w = ed[i];
        assert(pot[w.u] != INF);
        assert(pot[v] != INF);
        C uw = w.cost + pot[v] - pot[w.u];
        assert(uw >= 0);
        uw += npot[v];
        if (npot[w.u] > uw) {
          npot[w.u] = uw;
          que.push({-uw, w.u});
          par[w.u] = i;
        }
      }
    }
    for (int i = 0; i < n; i++)
      pot[i] = (npot[i] == INF ? INF : pot[i] + npot[i]);
  }
  return pot[fin] != INF;
}

pair<F, C> get(bool use_que = 0) {
  if (!ford_bellman())
    return {0, 0};
  while (1) {
    vector<int> q;
    int v = fin;
    F fl = numeric_limits<F>::max();
    C cs = 0;
    while (v != st) {
      q.push_back(par[v]);
      auto &w = ed[par[v]];
      fl = min(fl, w.c - w.f);
      cs += w.cost;
      v = w.v;
    }
    flow += fl;
    cost += fl * cs;
    for (int &i : q)
      ed[i].f += fl, ed[i ^ 1].f -= fl;
    if (!dijka(use_que))
      return {flow, cost};
  }
}
};
```

8.3 Венгерка

```
// works in n^2 * m
// set values to a[1..n][1..m] (n <= m)
// run calc(n, m) to find MINIMUM weighted matching
// permutation in ans[1..n]
// -v[0] is answer
// если ответ не влезает в инт, vector<ll> v, u, return type -> ll
// -v[0] и u[0] переполнится, остальное не больше макс ребра
// остальное можно инт
// works with negative values
// u, v are potentials, s.t. u[i] + v[j] <= a[i][j] and u[i] + v[j] equals to
↪  matching
const int N = 210, inf = 1e9 + 100;
int a[N][N];
int ans[N];
int calc(int n, int m) {
  ++n, ++m;
  vector<int> u(n), v(m), p(m), prev(m);
  for (int i = 1; i < n; ++i) {
    p[0] = i;
    int x = 0;
    vector<int> mn(m, inf);
    vector<int> was(m, 0);
    while (p[x]) {
      was[x] = 1;
      int ii = p[x], dd = inf, y = 0;
      for (int j = 1; j < m; ++j) if (!was[j]) {
        int cur = a[ii][j] - u[ii] - v[j];
        if (cur < mn[j]) mn[j] = cur, prev[j] = x;
        if (mn[j] < dd) dd = mn[j], y = j;
      }
      for (int j = 0; j < m; j++) {
        if (was[j]) u[p[j]] += dd, v[j] -= dd;
        else mn[j] -= dd;
      }
      x = y;
    }
    while (x) {
      int y = prev[x];
      p[x] = p[y];
      x = y;
    }
  }
  for (int j = 1; j < m; ++j)
    ans[p[j]] = j;
  return -v[0];
}
```

8.4 Глобальный разрез

```
//global mincut in undirected graph, works in n^3, can be nm logn + n^2
//ищем пару вершин, находим минкат, стягиваем пару
//итеративно добавляем в множество A по вершине, последние 2
↪  добавленные стягиваем
// в g числа становятся суммой исходных, может переполниться
// best_cost - answer
// best_cut - cut
// copy g before running (it will be modified)
const int MAXN = 500;
int n, g[MAXN][MAXN];
int best_cost = 1e9;
vector<int> best_cut;
void mincut() {
  vector<int> v[MAXN];
  for (int i=0; i<n; ++i)
    v[i].assign (1, i);
  int w[MAXN];
  bool exist[MAXN], in_a[MAXN];
  memset (exist, true, sizeof exist);
  for (int ph=0; ph<n-1; ++ph) {
    memset (in_a, false, sizeof in_a);
    memset (w, 0, sizeof w);
    for (int it=0, prev; it<n-ph; ++it) {
      int sel = -1;
      for (int i=0; i<n; ++i)
        if (exist[i] && !in_a[i] && (sel == -1 || w[i] > w[sel]))
          sel = i;
      if (it == n-ph-1) {
        if (w[sel] < best_cost)
          best_cost = w[sel],  best_cut = v[sel];
        v[prev].insert (v[prev].end(), v[sel].begin(), v[sel].end());
        for (int i=0; i<n; ++i)
          g[prev][i] = g[i][prev] += g[sel][i];
        exist[sel] = false;
      }
      else {
        in_a[sel] = true;
        for (int i=0; i<n; ++i)
          w[i] += g[sel][i];
        prev = sel;
      }
    }
  }
}
```

8.5 Гомори-Ху

```
// input: undirected graph
// output: tree in form [p_i, w_i], res[0] is empty
// runs dinic n-1 times
// mincut in this tree <=> mincut in original graph
```

```cpp
// g - (u, v, w)
vector<pair<int, int>> gomory_hu(int n, vector<tuple<int, int, int>> const &g) {
  Dinic f(n, 0, 0);
  for (auto [v, u, w] : g)
    f.add(v, u, w, w);
  vector<pair<int, int>> res(n);
  vector<int> pr(n, 0);
  for (int i = 1; i < n; i++) {
    for (auto &j : f.ed)
      j.f = 0;
    f.st = i;
    f.fin = pr[i];
    auto flow = f.get();
    vector<bool> cut(n);
    for (int j = 0; j < n; j++)
      cut[j] = (f.ds[j] != -1);
    for (int j = i + 1; j < n; j++) {
      if (cut[j] == cut[i] && pr[j] == pr[i])
        pr[j] = i;
    }
    res[i] = {pr[i], flow};
  }
  return res;
}
```

9 Паросочетание и рядом

9.1 Кун

```cpp
// в g обычный граф, в a множество вершин левой доли (для куна
↪  лучше брать меньшую долю)
// в списке смежности можно хранить только ребра направо

bool dfs_mt(int v, vector<vector<int>>& g, vector<int>& f, vector<bool>& u) {
  u[v] = true;
  for (auto k : g[v]) {
    if (f[k] == -1 || (!u[f[k]] && dfs_mt(f[k], g, f, u))) {
      f[k] = v;
      return true;
    }
  }
  return false;
}

vector<pair<int, int>> find_matching(vector<vector<int>>& g, vector<int> &a) {
  vector<bool> u(g.size(), false);
  vector<int> f(g.size(), -1);
  for (auto v : a)
    if (dfs_mt(v, g, f, u))
      u.assign(g.size(), false);
  vector<pair<int, int>> res;
  for (int v = 0; v < f.size(); ++v)
    if (f[v] != -1)
      res.emplace_back(f[v], v);
  return res;
}
```

9.2 Доминирующее множество

```cpp
// описание в куне

void dfs_ds(int v, vector<vector<int>>& g, vector<int>& to, vector<bool>& u,
↪  vector<bool>& take) {
  u[v] = true;
  for (auto k : g[v]) {
    take[k] = true;
    if (!u[to[k]])
      dfs_ds(to[k], g, to, u, take);
  }
}

vector<int> find_dominating_set(vector<vector<int>>& g, vector<int>& a) {
  auto mt = find_matching(g, a);
  vector<int> to(g.size(), -1);
  for (auto edge : mt) {
    to[edge.first] = edge.second;
    to[edge.second] = edge.first;
  }
  vector<bool> u(g.size(), false);
  vector<bool> take(g.size(), false);
  for (auto v : a)
    if (to[v] == -1)
      dfs_ds(v, g, to, u, take);
  for (auto e : mt)
    if (!take[e.second])
      take[e.first] = true;
  vector<int> res(mt.size());
  for (int i = 0; i < mt.size(); ++i) {
    if (take[mt[i].second]) res[i] = mt[i].second;
    else res[i] = mt[i].first;
  }
  return res;
}

vector<int> find_independent_set(vector<vector<int>>& g, vector<int>& a) {
  vector<int> res;
  vector<bool> u(g.size(), false);
  for (auto v : find_dominating_set(g, a))
    u[v] = true;
  for (int i = 0; i < g.size(); ++i)
    if (!u[i])
      res.push_back(i);
  return res;
}
```

9.3 Blossom

```cpp
//runs in N^3
//mate - matching
class Blossom {
public:
  const static int N = 2e4 + 10;
  int mate[N], n, ret;
  vector <int> G[N];
  void init() {
    for (int i = 0; i < N; i++)
      G[i].clear();
  }
  int run(int n) {
    this->n = n;
    memset(mate, -1, sizeof(mate[0]) * n);
    {
      // greedy matching
      vector<int> ord(n);
      iota(all(ord), 0);
      shuffle(ord);
      for (int i : ord) {
        shuffle(G[i]);
        for (int j : G[i]) {
          if (mate[i] == -1 && mate[j] == -1) {
            mate[i] = mate[j];
            mate[j] = mate[i];
          }
        }
      }
    }
    for (int i = 0; i < n; i++) if (mate[i] == -1) aug(i, G);
    for (int i = ret = 0; i < n; i++) ret += (mate[i] > i); // i -- mate[i]
↪  connected
    return ret;
  }
  void add_edge(int a, int b) {
    G[a].push_back(b);
    G[b].push_back(a);
  }
private:
  int next[N], dsu[N], mark[N], vis[N];
  // queue is slow, consider using vector
  queue<int> Q;
  int get(int x) {
    return (x == dsu[x]) ? x : (dsu[x] = get(dsu[x]));
  }
  void merge(int a, int b) {
    dsu[get(a)] = get(b);
  }
  int lca(int x, int y) {
    static int t = 0;
    ++t;
    for (; ; swap(x, y)) if (x != -1) {
      if (vis[x = get(x)] == t) return x;
      vis[x] = t;
      x = (mate[x] != -1) ? next[mate[x]] : -1;
    }
  }
  void group(int a, int p) {
    for (int b, c; a != p; merge(a, b), merge(b, c), a = c) {
      b = mate[a], c = next[b];
      if (get(c) != p) next[c] = b;
      if (mark[b] == 2) mark[b] = 1, Q.push(b);
      if (mark[c] == 2) mark[c] = 1, Q.push(c);
    }
  }
  void aug(int s, const vector<int> G[]) {
    for (int i = 0; i < n; i++) next[i] = vis[i] = -1, dsu[i] = i, mark[i] = 0;
    while (!Q.empty()) Q.pop();
    Q.push(s);
    mark[s] = 1;
    while (mate[s] == -1 && !Q.empty()) {
      int x = Q.front();
      Q.pop();
      for (int y : G[x]) {
        if (y != mate[x] && get(x) != get(y) && mark[y] != 2) {
          if (mark[y] == 1) {
            int p = lca(x, y);
            if (get(x) != p) next[x] = y;
            if (get(y) != p) next[y] = x;
            group(x, p);
            group(y, p);
          } else if (mate[y] == -1) {
            next[y] = x;
            for (int j = y, k, l; j != -1; j = l) {
              k = next[j];
              l = mate[k];
              mate[j] = k;
              mate[k] = j;
            }
            break;
          } else {
            next[y] = x;
            Q.push(mate[y]);
            mark[mate[y]] = 1;
            mark[y] = 2;
          }
        }
      }
    }
  }
} T; // T.init() at the begin
```

9.4 Хопкрофт

```
//E sqrt(V) bipartite matching
#define vi vector<int>
#define sz(x) (x).size()
#define trav(x, y) for (auto x : y)
#define rep(x, l, r) for (int x = l; x < r; x++)
bool dfs(int a, int layer, const vector<vi>& g, vi& btoa,
    vi& A, vi& B) {
  if (A[a] != layer) return 0;
  A[a] = -1;
  trav(b, g[a]) if (B[b] == layer + 1) {
    B[b] = -1;
    if (btoa[b] == -1 || dfs(btoa[b], layer+2, g, btoa, A, B))
      return btoa[b] = a, 1;
  }
  return 0;
}
// доли размеров n и m
// в g ребра из левой в правую долю
// сертификат в btoa
//btoa - vi(m, -1)
//g.size() - n
int hopcroftKarp(const 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), -1);
    /// Find the starting nodes for BFS (i.e. layer 0).
    cur.clear();
    trav(a, btoa) if(a != -1) A[a] = -1;
    rep(a,0,sz(g)) if(A[a] == 0) cur.push_back(a);
    /// Find all layers using bfs.
    for (int lay = 1;; lay += 2) {
      bool islast = 0;
      next.clear();
      trav(a, cur) trav(b, g[a]) {
        if (btoa[b] == -1) {
          B[b] = lay;
          islast = 1;
        }
        else if (btoa[b] != a && B[b] == -1) {
          B[b] = lay;
          next.push_back(btoa[b]);
        }
      }
      if (islast) break;
      if (next.empty()) return res;
      trav(a, next) A[a] = lay+1;
      cur.swap(next);
    }
    /// Use DFS to scan for augmenting paths.
    rep(a,0,sz(g)) {
      if(dfs(a, 0, g, btoa, A, B))
        ++res;
    }
  }
}
```

10 Математика

10.1 Ряды

```
struct poly : vector<Mint> {
  poly() {}
  poly(const vector<Mint> &a) : vector<Mint>(a) {}
  poly(int n) : vector<Mint>(n, 0) {}
  template <class It>
  poly(It left, It right) : vector<Mint>(left, right) {}
  inline void normalize() {
    while (size() > 1 && back() == 0) pop_back();
  }
  inline Mint& operator[](int pos) {
    if (pos >= (int)size()) resize(pos + 1);
    return vector<Mint>::operator[](pos);
  }
  inline Mint operator[](int pos) const {
    if (pos < (int)size()) return vector<Mint>::operator[](pos);
    return 0;
  }
  inline poly operator+(const poly &b) const {
    const poly &a = *this;
    poly ans(max((int)a.size(), (int)b.size()));
    for (int i = 0; i < ans.size(); i++) ans[i] = a[i] + b[i];
    return ans;
  }
  inline poly operator+=(const poly &p) {
    return *this = *this + p;
  }
  inline poly operator-(const poly &b) const {
    const poly &a = *this;
    poly ans(max((int)a.size(), (int)b.size()));
    for (int i = 0; i < ans.size(); i++) ans[i] = (a[i] - b[i]);
    return ans;
  }
  inline poly operator-=(const poly &p) {
    return *this = *this - p;
  }
  inline poly operator*(Mint d) const {
    poly ans = *this;
    for (int i = 0; i < (int)ans.size(); i++) ans[i] = ans[i] * d;
    return ans;
  }
  inline poly operator*=(Mint d) {
    return *this = *this * d;
  }
  inline poly operator/(Mint d) const {
    d = d.inv();
    poly ans = *this;
    for (int i = 0; i < (int)ans.size(); i++) ans[i] = ans[i] * d;
    return ans;
  }
  inline poly operator/=(Mint d) {
    return *this = *this / d;
  }
  inline poly operator*(const poly &p) const {
    return poly(multiply_nft(*this, p));    // here any multiply with fft
↪ should be used
  }
  inline poly operator*=(const poly &p) {
    return *this = *this * p;
  }
  inline poly cut(int n) const {
    poly a = (*this); a.resize(n); return a;
  }
  inline poly invp() const {
    poly a(1);
    a[0] = 1;
    if ((*this)[0] != 1) a[0] = (*this)[0].inv();
    for (int n = 1; n < (int)size(); n <<= 1) {
      poly ca = cut(n + n);
      poly ra = (a * a);
      ra.resize(n + n);
      ra = (ra * ca);
      ra.resize(n + n);
      a += a;
      a -= ra;
      a = a.cut(n + n);
    }
    a.resize(size());
    return a;
  }
  inline poly rev() const {
    poly a = (*this);
    reverse(a.begin(), a.end());
    return a;
  }
  inline poly getdiv(poly b) const { // use it only to divide polynomials
    poly a = (*this);
    a.normalize();
    b.normalize();
    if (a.size() < b.size()) return poly(1);
    int k = (int)a.size() - (int)b.size() + 1;
    if (b.back() != 1) b /= b.back();
    poly rb = b.rev().cut(k);
    return (rb.invp() * a.rev()).cut(k).rev();
  }
  inline poly operator/(const poly &b) const { // use it only to divide
↪ polynomials
    return getdiv(b);
  }
  inline poly operator/=(const poly &b) { // use it only to divide polynomials
    return *this = *this / b;
  }
  inline poly operator%(const poly &b) const { // use it only to divide
↪ polynomials
    poly a = (*this);
    poly d = (a / b);
    if (d.size() == 1 && d[0] == 0) return a;
    poly r = (a - d * b);
    r.normalize();
    return r;
  }
  inline poly operator%=(const poly &b) { return *this = *this % b; }
  inline poly der() const {
    const poly& a = (*this);
    vector<Mint> ans;
    for (int i = 1; i < size(); i++) {
      ans.emplace_back(a[i] * i);
    }
    return ans;
  }
  inline poly integral() const {
    const poly& a = (*this);
    vector<Mint> ans;
    ans.emplace_back(0);
    for (int i = 0; i < size(); i++) {
      ans.push_back(a[i] / (i + 1));
    }
    return ans;
  }
};
// n=500k, inv: 300ms, log: 500ms, exp: 1100ms; n=2^17, multipoint: 1500ms
// Newton: solving f(x) = 0, make iterations x = x - f(x)/f'(x)
// be careful with length of a, make 0 at the end to get the longer result

inline poly log(poly a) { // should be a[0] = 1
  poly b = a.der();
  b *= a.invp();
  b.resize(a.size() - 1);
  return b.integral();
}

inline poly exp(poly f) { // should be f[0] = 0
  poly a(1);
  a[0] = 1;
  for (int n = 1; n < (int)f.size(); n <<= 1) {
    a.resize(2 * n);
    poly ca = f.cut(2 * n);
    a += (ca - log(a)) * a;
    a = a.cut(2 * n);
  }
  a.resize(f.size());
  return a;
}

inline poly good_sqrt(poly f) { // should be f[0] = 1
  poly a(1);
  a[0] = 1;
```

```
    auto inv2 = Mint(2).inv();
    for (int n = 1; n < (int)f.size(); n <<= 1) {
        a.resize(2 * n);
        a += (f.cut(2 * n) * a.invp());
        a = (a * inv2).cut(2 * n);
    }
    a.resize(f.size());
    return a;
}

inline poly sqrt(poly f) {
    for (int i = 0; i < f.size(); i++) {
        if (f[i] != 0) {
            int t = mod_sqrt(f[i]);
            if (i % 2 == 1 || t == -1) return poly();
            poly ct = poly(f.begin() + i, f.end()) / f[i];
            ct.resize(f.size() - (i / 2));
            poly res = good_sqrt(ct);
            vector<Mint> ans(i / 2, 0);
            for (const auto& x : res) {
                ans.emplace_back(x);
            }
            return poly(ans) * Mint(t);
        }
    }
    return poly(f.size());
}

vector<Mint> multipoint(poly p, const vector<Mint>& a) {
    vector<poly> all(4 * a.size());
    function<void(int, int, int)> calc;
    calc = [&](int i, int l, int r) {
        if (r - l == 1) {
            all[i] = poly({-a[l], 1});
            return;
        }
        int mid = (l + r) / 2;
        calc(2 * i + 1, l, mid);
        calc(2 * i + 2, mid, r);
        all[i] = all[2 * i + 1] * all[2 * i + 2];
    };
    calc(0, 0, a.size());
    vector<Mint> ans(a.size());
    function<void(poly, int, int, int)> solve;
    solve = [&](poly p, int i, int l, int r) {
        if (r - l == 1) {
            ans[l] = p[0];
            return;
        }
        int mid = (l + r) / 2;
        solve(p % all[2 * i + 1], 2 * i + 1, l, mid);
        solve(p % all[2 * i + 2], 2 * i + 2, mid, r);
    };
    solve(p % all[0], 0, 0, a.size());
    return ans;
}
```

10.2 Полезные функции

```
ll rev(ll a, ll m) {
    if (a == 1) {
        return 1;
    }
    return (1LL - rev(m % a, a) * m) / a + m;
}

ll mult(ll a, ll b, ll m) {
    ll x = trunc((ld)a * (ld)b / (ld)m);
    x = max(0LL, x - 2);
    return (a * b - x * m) % m;
}

inline int power(int a, int k) {
    int res = 1;
    while (b > 0) {
        if (b & 1) res = 1LL * res * a % MOD;
        a = 1LL * a * a % MOD;
        b >>= 1;
    }
    return res;
}

// Дробь с минимальным знаменателем между двумя дробями
pair<ll, ll> find_best(pair<ll, ll> l, pair<ll, ll> r) { // (first/second)
    if (l.first >= l.second) {
        ll d = l.first / l.second;
        pair<ll, ll> res = find_best(
            make_pair(l.first - d * l.second, l.second), make_pair(r.first - d *
↪  r.second, r.second)
        );
        res.first += res.second * d;
        return res;
    }
    if (r.first > r.second) return make_pair(1, 1);
    pair<ll, ll> res = find_best(make_pair(r.second, r.first),
↪  make_pair(l.second, l.first));
    return make_pair(res.second, res.first);
}

// Сумма линейных функций по модулю
ll solve(ll n, ll a, ll b, ll m) { // sum(i=0..n-1) (a + b * i) div m
    if (b == 0) return n * (a / m);
    if (a >= m) return n * (a / m) + solve(n, a % m, b, m);
    if (b >= m) return n * (n - 1) / 2 * (b / m) + solve(n, a, b % m, m);
    return solve((a + b * n) / m, (a + b * n) % m, m, b);
}

// Другая странная сумма
ll floordiv(ll a, ll b) {
    if (a > 0) return a / b;
    else if (a % b == 0) return a / b;
    else return a / b - 1;
}

using LLL = Mint;
// or __int128, or ll if answer is <1e18
// for Mint maybe precalculate inv(2)

LLL ksum(ll a, ll b, ll c, ll n) {
    LLL ans = 0;
    ans += LLL(b / c) * LLL(n);
    b %= c;
    ans += LLL(a / c) * LLL(n) * LLL(n - 1) / LLL(2);
    a %= c;
    // here and later no need for __int128 if numbers are small
    if ((__int128)a * n + b < c) return ans;
    else return ans + ksum(c, ((__int128)a * n + b) % c, a, ((__int128)a * n + b)
↪  / c);
}

LLL rangesum(ll l1, ll l2, ll a, ll b, ll c) {// sum of [(ax+b)/c] for x =
↪  l1...l2
    if (l1 > l2) return 0;
    ll d = floordiv(b, c);
    b = b - d * c;
    return ksum(a, b, c, l2 + 1) - ksum(a, b, c, l1) + LLL(d) * LLL(l2 - l1 + 1);
}

// min дробь >= и x/y; max дробь <= и x/y со знаменателем <= n
namespace RationalApprox {
using ftype = int64_t;
using point = complex<ftype>;
#define x real
#define y imag

ftype cross(point a, point b) { return (conj(a) * b).y(); }
bool cmp(point a, point b) { return cross(a, b) < 0; }

const int inf = 1e9;

pair<array<ftype, 2>, array<ftype, 2>> solve(int N, int x, int y) {
    static vector<point> r = {1i, 1};
    r.resize(2);
    while (y && max(r.back().x(), r.back().y()) <= N) {
        ftype t = x / y;
        r.push_back(*(end(r) - 2) + *(end(r) - 1) * t);
        tie(x, y) = pair{y, x % y};
    }
    if (max(r.back().x(), r.back().y()) <= N) {
        auto A = r.back();
        auto B = r.back();
        return make_pair(array<ftype, 2>{A.x(), A.y()}, array<ftype, 2>{B.x(),
↪  B.y()});
    }
    auto A = *(end(r) - 2);
    ftype t = inf;
    if (A.x()) {
        t = min(t, (N - (end(r) - 3)->x()) / A.x());
    }
    if (A.y()) {
        t = min(t, (N - (end(r) - 3)->y()) / A.y());
    }
    auto B = *(end(r) - 3) + t * A;
    if (cmp(B, A))
        swap(B, A);
    return make_pair(array<ftype, 2>{A.x(), A.y()}, array<ftype, 2>{B.x(),
↪  B.y()});
}
} // namespace RationalApprox

// min_of_mod_of_linear(n, m, a, b, 1, 1, 1) is min (ax+b)%m for 0<=x<n
int min_of_mod_of_linear(int n, int m, int a, int b, int cnt, int p, int q) {
    if (a == 0)
        return b;
    if (cnt & 1) {
        if (b >= a) {
            int t = (m - b + a - 1) / a;
            int c = (t - 1) * p + q;
            if (n <= c)
                return b;
            n -= c;
            b += a * t - m;
        }
        b = a - 1 - b;
    } else {
        if (b < m - a) {
            int t = (m - b - 1) / a;
            int c = t * p;
            if (n <= c)
                return a * ((n - 1) / p) + b;
            n -= c;
            b += a * t;
        }
        b = m - 1 - b;
    }
    cnt++;
    int d = m / a;
    int c = min_of_mod_of_linear(n, a, m % a, b, cnt, (d - 1) * p + q, d * p + q);
    return cnt & 1 ? m - 1 - c : a - 1 - c;
}
```

10.3 Интегрирование

```
ld integrate(ld L, ld R) { // g(x) should be declared
    const int ITERS = 5e5;
    ld ans = 0;
    ld step = (R - L) / ITERS;
    for (int it = 0; it < ITERS; it++) {
        ld xl = L + step * it;
        ld xr = L + step * (it + 1);
```

```
    ld x1 = (xl + xr) / 2;
    ld x0 = x1 - (x1 - xl) * sqrt(3.0 / 5);
    ld x2 = x1 + (x1 - xl) * sqrt(3.0 / 5);
    ans += (5 * g(x0) + 8 * g(x1) + 5 * g(x2)) / 18 * step;
  }
  return ans;
};
```

10.4 Поллард и Миллер-Рабин

```
struct Factorizer {
  vector<int> mnp, pr;
  int precn = 100, spb = 100;
  mt19937_64 rng;

  Factorizer() : rng(123) {
    mnp.assign(precn + 1, -1);
    for (int i = 2; i <= precn; ++i) {
      if (mnp[i] == -1) {
        mnp[i] = i;
        pr.push_back(i);
      }
      int k = mnp[i];
      for (int j : pr) {
        if (j * i > precn) break;
        mnp[i * j] = j;
        if (j == k) break;
      }
    }
  }

  bool is_prime(ll n, bool cs = true) {
    if (n <= precn) return mnp[n] == n;

    if (cs) {
      for (int p : pr) {
        if (p > spb || (ll)p * p > n) break;
        if (n % p == 0) return false;
      }
    }

    int s = 0;
    ll d = n - 1;
    while (d % 2 == 0) {
      ++s;
      d >>= 1;
    }
    for (ll a : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) {
      if (a >= n) break;
      ll x = mpow_long(a, d, n);
      if (x == 1 || x == n - 1) continue;
      bool comp = true;
      for (int i = 0; i < s - 1; ++i) {
        x = mul_mod(x, x, n);
        if (x == 1) return false;
        if (x == n - 1) {
          comp = false;
          break;
        }
      }
      if (comp) return false;
    }
    return true;
  }

  vector<pair<ll, int>> factorize(ll n, bool cs = true) {
    vector<pair<ll, int>> res;
    if (cs) {
      for (int p : pr) {
        if (p > spb) break;
        if ((ll)p * p > n) break;
        if (n % p == 0) {
          res.emplace_back(p, 0);
          while (n % p == 0) {
            n /= p;
            res.back().second++;
          }
        }
      }
    }
    if (n == 1) return res;
    if (is_prime(n, false)) {
      res.emplace_back(n, 1);
      return res;
    }

    ll d = get_divisor(n);
    auto a = factorize(d, false);
    for (auto &[d, c] : a) {
      c = 0;
      while (n % d == 0) {
        n /= d;
        ++c;
      }
    }

    auto b = factorize(n, false);

    for (auto p : a) res.push_back(p);
    for (auto p : b) res.push_back(p);

    return res;
  }

  ll rnd(ll l, ll r) {
    return uniform_int_distribution<ll>(l, r)(rng);
  }

  ll mpow_long(ll a, ll p, ll mod) {
    ll res = 1;
    while (p) {
      if (p & 1) res = mul_mod(res, a, mod);
      p >>= 1;
      a = mul_mod(a, a, mod);
    }
    return res;
  }

  ll mul_mod(ll a, ll b, ll mod) {
    ll res = a * b - mod * (ll)((long double)1 / mod * a * b);
    if (res < 0) res += mod;
    if (res >= mod) res -= mod;
    return res;
  }

  ll get_divisor(ll n) {
    auto f = [&](ll x) -> ll {
      ll res = mul_mod(x, x, n) + 1;
      if (res == n) res = 0;
      return res;
    };

    while (true) {
      ll x = rnd(1, n - 1);
      ll y = f(x);
      while (x != y) {
        ll d = gcd(n, abs(x - y));
        if (d == 0) break;
        else if (d != 1) return d;
        x = f(x);
        y = f(f(y));
      }
    }
  }
};
```

10.5 Число простых

```
// works in <1s for 1e11, 4s for 1e12
// depending on f, can count number of primes up to n inclusive, or sum, or sum
↪  of squares, ...
ll count_primes(ll n) {
  auto f = [&](ll n) {
    // should be multiplicative (f(ab) = f(a)f(b)),
    // the result will be sum f(p) over all primes
    return 1;
  };
  auto pref = [&](ll n) {
    // should return sum_{i=1..n} f(i)
    return n;
  };

  vector<ll> v;
  v.reserve((int)sqrt(n) * 2 + 20);
  ll sq;
  {
    ll k = 1;
    for (; k * k <= n; ++k) {
      v.pb(k);
    }
    --k;
    sq = k;
    if (k * k == n) --k;
    for (; k >= 1; --k) {
      v.pb(n / k);
    }
  }
  vector<ll> s(v.size());
  for (int i = 0; i < s.size(); ++i)
    s[i] = pref(v[i]) - 1;
  auto geti = [&](ll x) {
    if (x <= sq) return (int)x - 1;
    else         return (int)(v.size() - (n / x));
  };
  for (ll p = 2; p * p <= n; ++p) {
    if (s[p - 1] != s[p - 2]) {
      ll sp = s[p - 2];
      ll p2 = p * p;
      for (int i = (int)v.size() - 1; i >= 0; --i) {
        if (v[i] < p2) {
          break;
        }
        s[i] -= (s[geti(v[i] / p)] - sp) * f(p);
      }
    }
  }
  return s.back();
}
```

10.6 Расширенный Евклид и КТО

```
// finds x and y such that a * x + b * y = c
pair<ll, ll> egcd(ll a, ll b, ll c) {
  if (a == 0) {
    // b * y = c
    assert(c % b == 0);
    return {0, c / b};
  }
  auto [y0, x0] = egcd(b % a, a, c);
  ll y = y0;
  ll x = x0 - (b / a) * y;
  return {x, y};
}

ll crt(ll m1, ll r1, ll m2, ll r2) {
  // ll d = gcd(m1, m2); assert(r1 % d == r2 % d)
  ll x = egcd(m1, -m2, r2 - r1).first;
  x = (x % m2 + m2) % m2;
  x = x * m1 + r1;
  return x % lcm(m1, m2);
}
```

10.7 Все пифагоровы тройки

```
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  if (modpow(a, (p-1)/2, p) != 1) return -1;
  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;
  /// find a non-square mod p
  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 = modmul(t, t, p);
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = modmul(gs, gs, p);
    x = modmul(x, gs, p);
    b = modmul(b, g, p);
  }
}

array<pair<ll, ll>, 4> getabprime(ll p) {
  if (p == 2 || p % 4 == 3) {
    return {{{0, p}, {p, 0}, {-1, -1}, {-1, -1}}};
  }
  ll res = sqrt(p - 1, p);
  assert(res != -1);
  ll a = res, b = p;
  while (b != 0) {
    a %= b;
    swap(a, b);
    if ((__int128)a * a + (__int128)b * b == p) {
      ll x = abs(a * a - b * b);
      ll y = 2 * a * b;
      return {{{x, y}, {y, x}, {0, p}, {p, 0}}};
    }
  }
  assert(false);
}

vector<pair<ll, ll>> getpairs(ll n) {
  vector<pair<ll, ll>> res;
  vector<pair<ll, ll>> next;
  auto pr = factorizer.factorize(n);
  res.emplace_back(0, 1);
  res.emplace_back(1, 0);
  for (auto [p, c] : pr) {
    auto prs = getabprime(p);
    for (int x = 0; x < c; ++x) {
      next.clear();
      for (auto [a1, b1] : res) {
        for (auto [a2, b2] : prs) {
          if (a2 == -1) continue;
          next.emplace_back(a1 * a2 + b1 * b2, abs(a1 * b2 - a2 * b1));
          next.emplace_back(abs(a1 * b2 - a2 * b1), a1 * a2 + b1 * b2);
        }
      }
      sort(next.begin(), next.end());
      next.resize(unique(next.begin(), next.end()) - next.begin());
      swap(next, res);
    }
  }
  return res;
}
```

10.8 Произведение нимберов

```
// a + b = mex([a' + b for a'<a] + [a + b' for b'<b])
// a * b = mex([a'*b + a'*b' + a*b' for a'<a, b'<b])
constexpr struct bit_prod_t {
  array<array<uint64_t, 64>, 64> v = {};
  constexpr bit_prod_t() {
    for (int i = 0; i < 64; ++i) {
      for (int j = 0; j < 64; ++j) {
        if (!(i & j)) {
          v[i][j] = uint64_t(1) << (i | j);
        } else {
          // square(2^2^a) = 2^2^a ^ 2^(2^a-1)
          int a = (i & j) & -(i & j);
          v[i][j] = v[i ^ a][j] ^ v[(i ^ a) | (a - 1)][(j ^ a) | (i & (a - 1))];
        }
      }
    }
  }

  const array<uint64_t, 64> &operator [] (int i) const {
    return v[i];
  }
} bit_prod;

uint64_t nim_prod(uint64_t a, uint64_t b) {
  uint64_t res = 0;
  for (int i = 0; (i < 64) && (a >> i); ++i)
    if ((a >> i) & 1)
      for (int j = 0; (j < 64) && (b >> j); ++j)
        if ((b >> j) & 1)
          res ^= bit_prod[i][j];
  return res;
}
```

10.9 Различные подпоследовательности

```
int n;
scanf("%d", &n);
vector<long long> a(n);
for (int i = 0; i < n; i++) scanf("%lld", &a[i]);
vector<long long> dp(n + 1, 0);
dp[0] = 1ll;
map<long long, long long> count;
for (int i = 0; i < n; i++) {
  dp[i + 1] += 2ll * dp[i] - count[a[i]] + mod;
  dp[i + 1] %= mod;
  count[a[i]] += dp[i] - count[a[i]] + mod;
  count[a[i]] %= mod;
}
printf("%lld\n", (dp[n] - 1 + mod) % mod);
```

11 Геометрия

11.1 Ближайшая пара точек

```
ll ClosestPair(vector<pair<int, int>> a) {
  int n = a.size();
  sort(all(a));
  set<pair<int, int>> s;

  ll best = 1e18;
  int j = 0;
  for (int i = 0; i < n; ++i) {
    int d = ceil(sqrt(best));
    while (a[i].fr - a[j].fr >= d) {
      s.erase({a[j].sc, a[j].fr});
      j += 1;
    }

    auto it1 = s.lower_bound({a[i].sc - d, a[i].fr});
    auto it2 = s.upper_bound({a[i].sc + d, a[i].fr});

    for (auto it = it1; it != it2; ++it) {
      ll dx = a[i].fr - it->sc;
      ll dy = a[i].sc - it->fr;
      best = min(best, dx * dx + dy * dy);
    }
    s.insert({a[i].sc, a[i].fr});
  }
  return best;
}
```

11.2 Калиперы

```
void convexHull(vector<vect<ll>>& v) {
  if (v.empty()) return;
  sort(v.begin(), v.end(), [&](const auto& v1, const auto& v2) {
    return tie(v1.x, v1.y) < tie(v2.x, v2.y);
  });
  vector<vect<ll>> st;
  st.reserve(v.size() + 1);
  for (int order = 0; order < 2; order++) {
    int sz = order == 1 ? st.size() : 1;
    for (const auto &vc: v) {
      while (st.size() > sz) {
        if (((st.back() - st.end()[-2]) ^ (vc - st.back())) > 0) break;
        st.pop_back();
      }
      st.emplace_back(vc);
    }
    reverse(v.begin(), v.end());
  }
  if (st.size() > 1) st.pop_back();
  swap(v, st);
}

ll diameter(vector<vect<ll>> v) {
  convexHull(v);
  int m = v.size();
  if (m == 1) return 0.0;
  int k = 1;
  while (((v[0] - v[m - 1]) ^ (v[(k + 1) % m] - v[k])) > 0) ++k;
  ll res = 0;
  for (int i = 0, j = k; i <= k && j < m; i++) {
    res = max(res, (v[i] - v[j]).len2());
    while (j < m && ((v[(i + 1) % m] - v[i]) ^ (v[(j + 1) % m] - v[j])) > 0) {
      j = (j + 1) % m;
      res = max(res, (v[i] - v[j]).len2());
    }
  }
  return res;
}
```

11.3 Пересечение полуплоскостей

```
namespace hpi {
  const ld eps = 1e-8;
  typedef pair<ld, ld> pi;

  bool z(ld x) { return fabs(x) < eps; }

  ld ccw(pi a, pi b, pi c) {
    return (b.fr - a.fr) * (c.sc - a.sc) - (b.sc - a.sc) * (c.fr - a.fr);
  }

  struct line {
    ld a, b, c;

    bool operator<(const line &l) const {
      bool f1 = pi(a, b) > pi(0, 0);
      bool f2 = pi(l.a, l.b) > pi(0, 0);
      if (f1 != f2) return f1 > f2;
      ld t = ccw(pi(0, 0), pi(a, b), pi(l.a, l.b));
      return z(t) ? c * hypot(l.a, l.b) < l.c * hypot(a, b) : t > 0;
    }

    pi sl() { return pi(a, b); }
  };

  pi cross(line a, line b) {
    ld det = a.a * b.b - b.a * a.b;
```

```
    return pi((a.c * b.b - a.b * b.c) / det, (a.a * b.c - a.c * b.a) / det);
  }

  bool bad(line a, line b, line c) {
    if (ccw(pi(0, 0), a.sl(), b.sl()) <= 0) return false;
    pi cs = cross(a, b);
    return cs.first * c.a + cs.second * c.b >= c.c;
  }

  bool solve(vector<line>& v, vector<pi>* solution) { // ax + by <= c;
    sort(v.begin(), v.end());
    deque<line> d;
    for (auto &i: v) {
      if (!d.empty() && z(ccw(pi(0, 0), d.back().sl(), i.sl()))) continue;
      while (d.size() >= 2 && bad(d[d.size() - 2], d.back(), i)) d.pop_back();
      while (d.size() >= 2 && bad(i, d[0], d[1])) d.pop_front();
      d.push_back(i);
    }
    while (d.size() > 2 && bad(d[d.size() - 2], d.back(), d[0])) d.pop_back();
    while (d.size() > 2 && bad(d.back(), d[0], d[1])) d.pop_front();
    if (solution != nullptr) solution->clear();
    for (int i = 0; i < d.size(); i++) {
      line cur = d[i], nxt = d[(i + 1) % d.size()];
      if (ccw(pi(0, 0), cur.sl(), nxt.sl()) <= eps) return false;
      if (solution != nullptr) solution->emplace_back(cross(cur, nxt));
    }
    v = vector<line>(d.begin(), d.end());
    return true;
  }
}

// halfplane of d(p, v1) <= d(p, v2), ax + by <= c
hpi::line nearest(const vect<ll>& v1, const vect<ll>& v2) {
  return hpi::line{
    (ld)((v2.x - v1.x) * 2), (ld)((v2.y - v1.y) * 2),
    (ld)(v2.x * v2.x + v2.y * v2.y - v1.y * v1.y - v1.x * v1.x)
  };
}

// halfplane on the left side of v1v2, ax + by <= c
hpi::line left_side(const vect<ll>& v1, const vect<ll>& v2) {
  return hpi::line{
    (ld)(v2.y - v1.y), (ld)(v1.x - v2.x),
    (ld)(v1.x * v2.y - v2.x * v1.y)
  };
}

const int X = 1e9;

// points should be different !!!
vector<vector<hpi::line>> voronoi(const vector<vect<ll>>& v) {
  vector<vector<hpi::line>> res;
  res.reserve(v.size());
  for (int i = 0; i < (int)v.size(); i++) {
    vector<hpi::line> lines;
    lines.reserve((int)v.size() + 3);
    lines.emplace_back(hpi::line{-1, 0, X});
    lines.emplace_back(hpi::line{1, 0, X});
    lines.emplace_back(hpi::line{0, -1, X});
    lines.emplace_back(hpi::line{0, 1, X});
    for (int j = 0; j < (int)v.size(); j++) {
      if (j != i) {
        lines.emplace_back(nearest(v[i], v[j]));
      }
    }
    hpi::solve(lines, nullptr);
    res.emplace_back(lines);
  }
  return res;
}
```

11.4 Dynamic Convex Hull

```
//query for maximum, lines kx + m

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;
  }
};
```

11.5 Триангуляция

```
// polygon must be given counterclockwise (can be checked with signed area),
↪  points must be different !!!
vector<tuple<int, int, int>> triangulation(const vector<vect<ll>> &v) {
  vector<tuple<int, int, int>> ans;
  int n = v.size();
  vector<bool> used(n, false);
  queue<int> all;
  for (int i = 0; i < n; i++) all.push(i);
  for (int iter = 0; iter < n - 2; iter++) {
    while (true) {
      if (all.empty()) return ans;
      int p = all.front();
      all.pop();
      if (used[p]) continue;
      int x = (p + n - 1) % n;
      while (used[x]) x = (x + n - 1) % n;
      int y = p;
      int z = (p + 1) % n;
      while (used[z]) z = (z + 1) % n;
      if (((v[x] - v[y]) ^ (v[z] - v[y])) >= 0) continue;
      bool bad = false;
      for (int i = 0; i < n; i++) {
        if (!used[i] && i != x && i != y && i != z) {
          if (area(v[x], v[y], v[z]) ==
              area(v[x], v[y], v[i]) + area(v[x], v[z], v[i]) + area(v[z], v[y],
↪  v[i])) {
            bad = true;
            break;
          }
        }
      }
      if (bad) continue;
      ans.emplace_back(x, y, z);
      used[y] = true;
      all.push(x);
      all.push(z);
      break;
    }
  }
  return ans;
}
```

11.6 Касательные к многоугольнику

```
int tangent(const vector<vect>& v, const vect& p, int cf) {
  int step = 1;
  for (; step < (int)v.size(); step *= 2);
  int pos = 0;
  int n = (int)v.size();
  for (; step > 0; step /= 2) {
    int best = pos;
    for (int dx = -1; dx <= 1; dx += 2) {
      int id = ((pos + step * dx) % n + n) % n;
      if (((v[id] - p) ^ (v[best] - p)) * cf > 0)
        best = id;
    }
    pos = best;
  }
  return pos;
}

pair<int, int> tangents(const vector<vect>& v, const vect& p) {
  return make_pair(tangent(v, p, 1), tangent(v, p, -1));
}
```

11.7 Касательные к двум окружностям

```
vector<line<ld>> commonTangents(const vect<ld>& A, ld rA, const vect<ld>& B, ld
↪  rB) {
  vector<line<ld>> res;
  auto C = B - A;
  ld z = C.len2(); // ~M^2
  for (int i = -1; i <= 1; i += 2) {
    for (int j = -1; j <= 1; j += 2)  {
      ld r = rB * j - rA * i; // ~M
      ld d = z - r * r; // ~M^2
      if (d < -1e-10) continue;
      d = sqrt(max(0.01, d)); // ~M
      auto magic = vect(r, d);
      vect v((magic * C) / z, (magic ^ C) / z); // ~(M, M)
      ld CC = (rA * i - (v * A)) / v.len2(); // ~1
      vect O = v * -CC;
      // point + directional vector
      res.emplace_back(O, v.rotate());
    }
  }
  return res;
}

// HOW TO USE ::
// --      *D*---------------*F*
// --     *...*-            -*...*
// --    *.....* -        - *.....*
// --   *.......*  -    -  *.......*
// --   *...A...*   --     *...B...*
// --   *.......*  -    -  *.......*
// --    *.....* -        - *.....*
// --     *...*-            -*...*
// --      *C*---------------*E*
// --  res = {CE, CF, DE, DF}
```

11.8 Проверка пересечения полуплоскостей

```
template <typename T>
bool intersection(const line<T>& l1, const line<T>& l2, vect<ld>& p) {
  auto pr = l1.a * l2.b - l1.b * l2.a;
  if (abs(pr) == 0) { return false; }
  auto prx = l1.b * l2.c - l1.c * l2.b;
  auto pry = l1.c * l2.a - l1.a * l2.c;
  p.x = (ld)prx / pr;
```

```
    p.y = (ld)pry / pr;
    return true;
}

// ax + by + c >= 0
template <typename T>
bool checkPlaneInt(vector<line<T>> l, vect<ld>& A) {
    shuffle(l.begin(), l.end(), rnd);
    auto f = [&](int i, const vect<ld>& a) {
        return a.x * l[i].a + a.y * l[i].b + l[i].c;
    };
    auto some_point = [&](int i) {
        if (abs(l[i].a) > abs(l[i].b)) {
            return vect<ld>(-(ld)l[i].c / l[i].a, 0.0);
        } else {
            return vect<ld>(0.0, -(ld)l[i].c / l[i].b);
        }
    };
    A = some_point(0);
    for (int i = 1; i < (int)l.size(); i++) {
        if (f(i, A) < -eps) {
            bool has_mn = false;
            bool has_mx = false;
            vect<ld> mn, mx;
            A = some_point(i);
            for (int j = 0; j < i; j++) {
                auto vj = l[j].normal();
                auto vi = l[i].normal();
                auto vec = (vj ^ vi);
                if (abs(vec) < eps) {
                    auto p = some_point(i);
                    if ((vj * vi) < -eps && f(j, p) < -eps) {
                        return false;
                    }
                } else {
                    vect<ld> cur;
                    intersection(l[i], l[j], cur);
                    if (vec < 0) {
                        if (!has_mx || f(j, mx) < 0) {
                            mx = cur;
                        }
                        has_mx = true;
                        if (has_mn && f(j, mn) < -eps) {
                            return false;
                        }
                    } else {
                        if (!has_mn || f(j, mn) < 0) {
                            mn = cur;
                        }
                        has_mn = true;
                        if (has_mx && f(j, mx) < -eps) {
                            return false;
                        }
                    }
                }
            }
            if (has_mx && has_mn) {
                if (make_pair(mx.y, mx.x) > make_pair(mn.y, mn.x)) {
                    A = mx;
                } else {
                    A = mn;
                }
            } else if (has_mx) {
                A = mx;
            } else if (has_mn) {
                A = mn;
            }
        }
    }
    return true;
}
```

11.9 Диагонали невыпуклого многоугольника

```
// c lies on segment ab
bool on_seg(const vect& a, const vect& b, const vect& c) {
    return ((a - c) ^ (b - c)) == 0 && ((a - c) * (b - c)) <= 0;
}

// c lies on ray ab
bool on_ray(const vect& a, const vect& b, const vect& c) {
    return ((c - a) ^ (b - a)) == 0 && ((c - a) * (b - a)) >= 0;
}

// ray ab intersects segment cd
bool intersect_ray_seg(const vect& a, const vect& b, const vect& c, const vect&
↪  d) {
    if (on_ray(a, b, c) || on_ray(a, b, d)) {
        return true;
    }
    int pr1 = sign((c - a) ^ (b - a));
    int pr2 = sign((b - a) ^ (d - a));
    if (pr1 == 0 || pr2 == 0 || pr1 != pr2) {
        return false;
    }
    int pr3 = sign((c - a) ^ (d - a));
    return pr3 == 0 || pr1 == pr3;
}

// segments ab, cd intersects in their interior point
bool intersect_seg(const vect& a, const vect& b, const vect& c, const vect& d) {
    return abs(sign((d - a) ^ (c - a)) + sign((c - b) ^ (d - b))) == 2 &&
           abs(sign((a - c) ^ (b - c)) + sign((b - d) ^ (a - d))) == 2;
}

// signed distance from point a to point of intersection of lines ab and cd
ld intersection_distance(const vect& a, const vect& b, const vect& c, const
↪  vect& d) {
    ll pr = (d - c) ^ (b - a);
    if (pr == 0) { // ab and cd are parallel
        return min((c - a) * (b - a), (d - a) * (b - a)) / (ld)(b - a).len();
    }
    return (b - a).len() * (ld)((d - a) ^ (c - a)) / (ld)pr;
}

// a is base vector, is b in lower halfplane?
bool halfplane(const vect& a, const vect& b) {
    ll pr = a ^ b;
    if (pr == 0) return a * b < 0;
    return pr < 0;
}

// a is base vector, compare b and c starting from it
bool cmp_from(const vect& a, const vect& b, const vect& c) {
    bool hb = halfplane(a, b);
    bool hc = halfplane(a, c);
    return (hb != hc ? hb < hc : (b ^ c) > 0);
}

// pieces of intersection of line ab and polygon v
ld solve(const vector<vect>& v, const vect& a, const vect& b) {
    int n = (int)v.size();
    vector<pair<ld, int>> events;
    for (int i = 0; i < n; i++) {
        int x = sign((b - a) ^ (v[i] - a));
        int y = sign((b - a) ^ (v[(i + 1) % n] - a));
        if (x == y) {
            continue;
        }
        events.emplace_back(intersection_distance(a, b, v[i], v[(i + 1) % n]), (x <
↪  y ? 1 : -1) * (x != 0 && y != 0 ? 2 : 1));
    }
    // coordinates a --> 0, b --> ||b-a||, balance != 0 --> inner
    sort(events.begin(), events.end());
    ld ans = 0;
    ld cur = 0;
    int bal = 0;
    for (int i = 0; i < (int)events.size(); i++) {
        if (bal != 0) {
            cur += events[i].first - events[i - 1].first;
            ans = max(ans, cur);
        } else {
            cur = 0;
        }
        bal += events[i].second;
    }
    return ans;
}
```

11.10 Выпуклая оболочка (3D)

```
// point on plane, normal vector
// normal is going to other side to polyhedron
template <typename T>
struct Plane {
    vect3d<T> 0, v;
    vector<int> id;
};

template <typename T>
vector<Plane<T>> convexHull3(const vector<vect3d<T>>& pts) {
    int n = pts.size();
    vector<pair<vect3d<T>, int>> p;
    for (int i = 0; i < n; i++) {
        p.emplace_back(pts[i], i);
    }
    vector<Plane<T>> res;
    for (int i = 0; i < 4; i++) {
        vector<pair<vect3d<T>, int>> tmp;
        for (int j = 0; j < 4; j++) {
            if (i != j) {
                tmp.emplace_back(p[j]);
            }
        }
        res.emplace_back(Plane<T>{tmp[0].first, (tmp[1].first - tmp[0].first) ^
↪  (tmp[2].first - tmp[0].first),
                                  {tmp[0].second, tmp[1].second, tmp[2].second}});
        if ((p[i].first - res.back().0) * res.back().v > 0) {
            res.back().v = res.back().v * T(-1);
            swap(res.back().id[0], res.back().id[1]);
        }
    }
    vector<vector<int>> use(n, vector<int>(n, 0));
    int cnt_used = 0;
    for (int i = 4; i < n; i++) {
        int cur = 0;
        cnt_used++;
        vector<pair<int, int>> curEdge;
        for (int j = 0; j < res.size(); j++) {
            if ((p[i].first - res[j].0) * res[j].v > 0) {
                for (int t = 0; t < 3; t++) {
                    int v = res[j].id[t];
                    int u = res[j].id[(t + 1) % 3];
                    use[v][u] = cnt_used;
                    curEdge.emplace_back(v, u);
                }
            }
            else {
                res[cur++] = res[j];
            }
        }
        res.resize(cur);
        for (auto x: curEdge) {
            if (use[x.second][x.first] == cnt_used) continue;
            res.emplace_back(Plane<T>{p[i].first, (p[x.first].first - p[i].first) ^
↪  (p[x.second].first - p[i].first), {x.first, x.second, i}});
        }
    }
    return res;
}
```

11.11 Минимальная накрывающая окружность

```
struct circle {
  vect<ld> c;
  ld r;

  bool inside(const vect<ld>& p) {
    return (p - c).len() <= r + 1e-10;
  }
};

circle make_circle(const vect<ld>& a, const vect<ld>& b) {
  return circle{(a + b) * (ld)0.5, (b - a).len() / 2};
}

template <typename T>
vect<ld> intersection(const line<T>& l1, const line<T>& l2) {
  auto pr = l1.a * l2.b - l1.b * l2.a;
  assert(abs(pr) > 1e-10);
  auto prx = l1.b * l2.c - l1.c * l2.b;
  auto pry = l1.c * l2.a - l1.a * l2.c;
  return vect<ld>((ld)prx / pr, (ld)pry / pr);
}

circle make_circle(const vect<ld>& a, const vect<ld>& b, const vect<ld>& c) {
  line<ld> l1(2 * (b.x - a.x), 2 * (b.y - a.y), (a.x * a.x + a.y * a.y - b.x *
↪  b.x - b.y * b.y));
  line<ld> l2(2 * (c.x - a.x), 2 * (c.y - a.y), (a.x * a.x + a.y * a.y - c.x *
↪  c.x - c.y * c.y));
  vect<ld> p = intersection(l1, l2);
  return circle{p, (a - p).len()};
}

circle minCoveringCircle(vector<vect<ld>> v) {
  mt19937 rnd(239);
  shuffle(v.begin(), v.end(), rnd);
  circle ans{v[0], 0};
  for (int i = 1; i < (int)v.size(); i++) {
    if (!ans.inside(v[i])) {
      ans = circle{v[i], 0};
      for (int j = 0; j < i; j++) {
        if (!ans.inside(v[j])) {
          ans = make_circle(v[i], v[j]);
          for (int k = 0; k < j; k++) {
            if (!ans.inside(v[k])) {
              ans = make_circle(v[i], v[j], v[k]);
            }
          }
        }
      }
    }
  }
  return ans;
}
```

11.12 Пересечение многоугольника и круга

```
template <typename T>
ld arg(const vect<T>& a, const vect<T>& b) {
  return atan2l(a ^ b, a * b);
}

// <0 (clockwise), >0 (counterclockwise)
ld circlePoly(const vect<ld>& c, ld r, const vector<vect<ld>>& ps) {
  auto tri = [&](const vect<ld>& p, const vect<ld>& q) {
    auto r2 = r * r / 2;
    auto d = q - p;
    auto a = (d * p) / d.len2(), b = (p.len2() - r * r) / d.len2();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max((ld)0, -a - sqrt(det)), t = min((ld)1, -a + sqrt(det));
    if (t < 0 || 1 <= s) return arg(p, q) * r2;
    auto u = p + d * s, v = p + d * t;
    return arg(p, u) * r2 + (u ^ v) / 2 + arg(v, q) * r2;
  };
  auto sum = 0.0;
  for (int i = 0; i < (int)ps.size(); i++) {
    sum += tri(ps[i] - c, ps[(i + 1) % (int)ps.size()] - c);
  }
  return sum;
}
```

11.13 Внутри многоугольника

```
// 0 - outside, 1 - on borders, 2 - strictly inside
template <typename T>
int inPolygon(const vector<vect<T>> &p, const vect<T>& a) {
  int cnt = 0, n = p.size();
  for (int i = 0; i < n; i++) {
    auto q = p[(i + 1) % n];
    if (on_seg(p[i], q, a)) return 1;
    cnt ^= ((int)(a.y < p[i].y) - (int)(a.y < q.y)) * ((p[i] - a) ^ (q - a)) > 0;
  }
  return cnt ? 2 : 0;
}
```

11.14 Еще геом. функции

```
struct Meta {
  int type; // 0 - seg, 1 - circle
  pt O;
  dbl R;
};

const Meta SEG = {0, pt(0, 0), 0};

vector<pair<pt, Meta>> cut(vector<pair<pt, Meta>> p, Line l) {
  vector<pair<pt,Meta>> res;
  int n = p.size();
  for (int i = 0; i < n; i++) {
    pt A = p[i].F;
    pt B = p[(i + 1) % n].F;
    if (le(0, l.v * (A - l.O))) {
      if (eq(0, l.v * (A - l.O)) && p[i].S.type == 1 && ls(0, l.v % (p[i].S.O -
↪  A)))
        res.pb({A, SEG});
      else
        res.pb(p[i]);
    }
    if (p[i].S.type == 0) {
      if (sign(l.v * (A - l.O)) * sign(l.v * (B - l.O)) == -1) {
        pt FF = Line(A, B) * l;
        res.pb(make_pair(FF, SEG));
      }
    }
    else {
      pt E, F;
      if (intCL(p[i].S.O, p[i].S.R, l, E, F)) { // intersect circle with line;
↪  output E, F
        if (onArc(p[i].S.O, A, E, B)) // if E lies between A and B
          res.pb({E, SEG});
        if (onArc(p[i].S.O, A, F, B)) // if F lies between A and B
          res.pb({F, p[i].S});
      }
    }
  }
  return res;
}

bool checkPoint(vector<Line> l, pt& ret) {
  random_shuffle(all(l));
  pt A = l[0].O;
  for (int i = 1; i < sz(l); i++) {
    if (!le(0, l[i].v * (A - l[i].O))) {
      dbl mn = -INF;
      dbl mx = INF;
      for (int j = 0; j < i; j++) {
        if (eq(l[j].v * l[i].v, 0)) {
          if (l[j].v % l[i].v < 0 && (l[j].O - l[i].O) % l[i].v.rotate() <= 0) {
            return false;
          }
        }
        else {
          pt u = l[j].v.rotate();
          dbl proj = (l[j].O - l[i].O) % u / (l[i].v % u);
          if (l[i].v * l[j].v > 0) {
            mx = min(mx, proj);
          }
          else {
            mn = max(mn, proj);
          }
        }
      }
      if (mn <= mx) {
        A = l[i].O + l[i].v * mn;
      }
      else {
        return false;
      }
    }
  }
  ret = A;
  return true;
}
```

11.15 CHT static

```
// max, first * x + second
vector<pair<ld, pair<int, int>>> cht(vector<pair<int, int>>& ln) {
  sort(ln.begin(), ln.end(), [&](const auto& t1, const auto& t2) {
    return make_pair(t1.first, -t1.second) < make_pair(t2.first, -t2.second);
  });
  vector<pair<ld, pair<int, int>>> ch;
  for (const auto& line : ln) {
    while (!ch.empty()) {
      auto lst = ch.back().second;
      if (lst.first == line.first) break;
      ld x = (ld)(lst.second - line.second) / (line.first - lst.first);
      if (x > ch.back().first) { ch.emplace_back(x, line); break; }
      ch.pop_back();
    }
    if (ch.empty()) { ch.emplace_back(make_pair(-INF, line)); continue; }
  }
  return ch;
}
```