# Задача 1. Помогите математику

| | |
|---|---|
| Имя входного файла: | `стандартный ввод` |
| Имя выходного файла: | `стандартный вывод` |
| Ограничение по времени: | 1 секунда |
| Ограничение по памяти: | 256 мегабайт |

Британский математик Джон Литлвуд однажды высказался о великом индийском математике Сринивасе Рамануджане: «каждое натуральное число было его личным другом».

Однажды Рамануджану для решения одной сложной математической задачи понадобилось ответить на $t$ вопросов. Каждый вопрос имел вид: сколько существует целых положительных чисел, которые делятся на $x$, но не делятся на $y$, и при этом не превосходят $n$.

Рамануджан смог ответить на все вопросы в уме, ведь он дружил со всеми натуральными числами. Однако он просит вас написать программу, которая ответит на каждый вопрос, чтобы проверить, что он нигде не ошибся.

## Формат входных данных

Первая строка содержит число $t$ ($1 \leqslant t \leqslant 10^5$) — количество вопросов. Далее следует описание вопросов. Каждый вопрос описывается числами $n$, $x$, $y$ ($1 \leqslant n, x, y \leqslant 10^{18}$), записанными через пробел в отдельной строке для каждого вопроса.

## Формат выходных данных

Вывод должен содержать ответы на вопросы, каждый ответ в отдельной строке.

## Система оценки

Решения, верно работающие при $t \times n \leqslant 10^5$, будут получать не менее 50 баллов.

## Пример

| Входные данные | Результат работы программы |
|---|---|
| 3<br>16 4 6<br>9 1 2<br>15 2 5 | 3<br>5<br>6 |

# Решение задачи

На 50 баллов данную задачу можно решить полным перебором: для каждого запроса перебрать все числа от 1 до $n$, посчитав среди этих чисел количество таких, которые делятся на $a$, но не делятся на $b$. К полному решению приводят следующие наблюдения:

1) количество чисел от 1 до $n$, которые делятся на $a$ равно $\lfloor \frac{n}{a} \rfloor$.

2) количество чисел от 1 до $n$, которые делятся и на $a$, и на $b$, равно $\lfloor \frac{n}{[a,b]} \rfloor$, где $[a, b]$ — НОК чисел $a$ и $b$.

3) Из пунктов 1 и 2 следует, что количество чисел от 1 до $n$, которые делятся на $a$, но не делятся на $b$, равно $\lfloor \frac{n}{a} \rfloor - \lfloor \frac{n}{[a,b]} \rfloor$. Осталось вспомнить, что НОК двух чисел равен отношению их произведения и их НОД, а НОД двух чисел можно находить алгоритмом Евклида или встроенными библиотечными функциями за $O(\max(\log a, \log b))$ Таким образом получили решение за $O(\max(\log a, \log b))$ на запрос. Итоговая асимптотика полного решения: $O(t \max(\log a, \log b))$

**Авторское решение на языке C++:**

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

typedef long long ll;
```

```
using namespace std;

#define int ll

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int t = 1;
    cin >> t;

    auto solve_stupid = [&](ll n, ll x, ll y) -> ll {
        ll ans = 0;
        for (ll i = 1; i <= n; ++i) {
            if (i % x == 0 && i % y != 0) {
                ++ans;
            }
        }
        return ans;
    };

    auto solve_smart = [&](ll n, ll x, ll y) -> ll {
        auto get = [] (ll n, ll x) {
            // сколько чисел от 1 до n делятся на x
            return n / x;
        };

        auto lcm = [] (ll a, ll b) -> ll {
            return a / __gcd(a, b) * b;
        };

        return get(n, x) - get(n, lcm(x, y));
    };

    while (t--) {
        ll n, x, y;
        cin >> n >> x >> y;
        cout << solve_smart(n, x, y) << '\n';
    }

    return 0;
}
```

# Задача 2. Волшебное слово

| | |
|---|---|
| Имя входного файла: | `стандартный ввод` |
| Имя выходного файла: | `стандартный вывод` |
| Ограничение по времени: | 1 секунда |
| Ограничение по памяти: | 256 мегабайт |

Уровнем волшебства строки назовём максимальное количество подряд идущих слов "tiim"(без кавычек), которые могут получиться удалением некоторых символов строки. Буратино решил волшебный лес из деревьев, на которых будут расти деньги. От кота Базилио он узнал, что если посадить в землю строку длиной $n$, то она вырастет в дерево, на котором будет висеть количество монет, равное уровню волшебства строки. У Буратино есть $t$ строк. Помогите ему узнать, сколько монет даст каждое выращенное им дерево.

## Формат входных данных

В первой строке написано натуральное число $t$ ($1 \leqslant t \leqslant 100$) — количество строк у Буратино. В следующих $2t$ строках следует описание всех имеющихся у него строк. Для каждого слова сначала записана его длина, а на следующей строке записана сама строка. (Для лучшего понимания смотрите тесты)

## Формат выходных данных

Выведите $t$ чисел, каждое в отдельной строке — ответы для каждой из строк, имеющихся у Буратино.

## Пример

| Входные данные | Результат работы программы |
|---|---|
| 3<br>6<br>tabiim<br>4<br>timi<br>16<br>abtmivmimabtioim | 1<br>0<br>2 |

# Решение задачи

**Разбор**

Давайте вместо того, чтобы удалять буквы из исходной строки, будем идти по этой строке и добавлять буквы в итоговый ответ. Можно заметить, что для добавления букв выгоднее всего использовать жадную стратегию: идти по буквам строки слева направо и жадно набирать из букв слово "tiim". Такое решение будет иметь асимптотику $O(n)$.

**Авторское решение на языке C++:**

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

using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int T;
    cin >> T;
```

```
    auto solve_test = [] () {
        int n; cin >> n;
        string s; cin >> s;

        int ans = 0, ost = 0;
        for (int i = 0; i < n; ++i) {
            if (ost == 0 && s[i] == 't') {
                ++ost;
            } else if (ost == 1 && s[i] == 'i') {
                 ++ost;
            } else if (ost == 2 && s[i] == 'i') {
                ++ost;
            } else if (ost == 3 && s[i] == 'm') {
                ++ans;
                ost = 0;
            }
        }

        return ans;
    };

    while (T --> 0) {
        cout << solve_test() << '\n';
    }

    return 0;
}
```

# Задача 3. Полотенце

| | |
|---|---|
| Имя входного файла: | `стандартный ввод` |
| Имя выходного файла: | `стандартный вывод` |
| Ограничение по времени: | 1 секунда |
| Ограничение по памяти: | 256 мегабайт |

*Мы строили-строили, и наконец построили. Ура!*

— Чебурашка

Скоро у Крокодила Гены день рождения, поэтому Чебурашка решил подарить ему *полотенце* прямоугольной формы. Чебурашка хочет, чтобы длины сторон были целыми положительными числами. Он знает, что Гена любит число $x$ ($x$ **чётное**), поэтому он собирается подобрать полотенце так, чтобы его периметр (сумма длин сторон полотенца) был равен $x$. Чтобы подчеркнуть значимость подарка, он хочет подарить полотенце самой большой возможной площади.

Помогите Чебурашке определить максимальную возможную площадь полотенца с фиксированным периметром $x$.

## Формат входных данных

Каждый тест содержит несколько наборов входных данных. Первая строка содержит число $t$ ($1 \leqslant t \leqslant 10$) — количество наборов входных данных. Каждый набор описывается в отдельной строке одним числом $x$ ($4 \leqslant x \leqslant 10^9$) — периметром полотенца.

## Формат выходных данных

Для каждого набора входных данных в отдельной строке выведите максимально возможную площадь полотенца с целочисленными сторонами и периметром $x$.

## Система оценки

Решения, правильно работающие на тестах, где $x \leqslant 10^5$, будут набирать не менее 30 баллов.

## Пример

| Входные данные | Результат работы программы |
|---|---|
| 3<br>4<br>10<br>12 | 1<br>6<br>9 |

## Замечание

Для периметра 4 существует единственное полотенце 1x1, которое имеет площадь 1.

Для периметра 10 существуют полотенца размера 2x3, 1x4. Из них максимальную площадь 6 имеет полотенце 2x3.

Для периметра 12 существуют полотенца размера 1x5, 2x4, 3x3. Из них максимальную площадь 9 имеет полотенце 3x3.

# Решение задачи

**Разбор**

Заметим, что стороны искомого прямоугольника являются решениями уравнения $x^2 - ax + b = 0$, где $a$ — сумма длин сторон (полупериметр), $b$ — искомая максимальная площадь. Посмотрим на дискриминант этого уравнения $D = a^2 - 4b$. Так как стороны являются целыми числами, то дискриминант должен быть точным квадратом. Теперь имеем уравнение $n^2 = a^2 - 4b$, из которого следует, что $b = \frac{a^2-b^2}{4}$. Мы хотим максимизировать $b$, а значит минимизировать $n$. Заметим, что нам достаточно, чтобы $a$ и $n$ были одной чётности, чтобы $b$ было целым числом. Значит оптимальное $n$ равно либо 0, либо 1. Таким образом, за $O(1)$ можно найти оптимальное $n$, из которого мы узнаем оптимальное $b$. Так как $b > 0$ и $a > 0$, то решения уравнения точно будут неотрицательными (убедиться в этом оставим упражнением для заинтересованного читателя). Получается что на один запрос можно отвечать за $O(1)$. Итоговая асимптотика решения $O(t)$.

**Авторское решение на языке C++:**

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

typedef long long ll;
using namespace std;

#define int ll

void solve() {
    int a; cin >> a;
    assert(a % 2 == 0);
    a /= 2;

    auto solve_stupid = [&] () {
        int ans = 0;
        for (int x = 1; x < a; ++x) {
            int y = a - x;
            int S = x * y;
            ans = max(ans, S);
        }
        return ans;
    };

    auto solve_smart = [&] () {
        cerr << a << endl;
        // x^2 - ax + b = 0
        // D = a^2 - 4b = n^2
        // a > n, т.к. a^2 = n^2 + 4b, где b > 0
        // (x, y) = (a +- n) / 2
        // т.е. a и n должны быть одной четности
        // 4b = a^2 - n^2
        // b = (a^2 - n^2) / 4
        // хотим максимизировать b
        // значит хотим минимизировать n

        for (int n = 0; n <= a; ++n) {
            // утверждается что цикл отработает быстро
            int quadro_b = (a * a - n * n);
```

```
            if (quadro_b % 4 != 0) {
                continue;
            }
            return quadro_b / 4;
        }
        return -1LL; // такого быть просто не может
    };

    cout << solve_smart() << '\n';
}

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int t = 1;
    cin >> t;
    while (t--) {
        solve();
    }
    return 0;
}
```

# Задача 4. Мечты сбываются

| | |
|---|---|
| Имя входного файла: | `стандартный ввод` |
| Имя выходного файла: | `стандартный вывод` |
| Ограничение по времени: | 1 секунда |
| Ограничение по памяти: | 256 мегабайт |

Уровнем волшебства строки назовём максимальное количество подряд идущих слов "tiim"(без кавычек), которые могут получиться удалением некоторых символов строки.

Буратино ждёт, пока вырастут его денежные деревья. Он уже начал представлять, как они взойдут и подарят ему несметные сокровища. Теперь у него в голове появилось $t$ вопросов. Каждый вопрос звучит так: сколько существует строк длины $n$ из строчных символов латинского алфавита с уровнем волшебства $k$? Помогите ему ответить на каждый вопрос. Так как ответы могут быть очень большими, выводите их по модулю $10^9 + 7$.

## Формат входных данных

В первой строке находится число $t$ ($1 \leqslant t \leqslant 10$) — количество вопросов, которые возникли у Буратино. В следующих $t$ строках описываются вопросы: в каждой строке через пробел вводятся числа $n$ и $k$ ($1 \leqslant n \times k \leqslant 10^6$) — параметры вопроса.

## Формат выходных данных

Для каждого вопроса в отдельной строке выведите ответ, вычисленный по модулю $10^9+7$.

## Пример

| Входные данные | Результат работы программы |
|---|---|
| 3<br>4 1<br>5 1<br>30 4 | 1<br>126<br>82414355 |

## Замечание

Для $n = 4$ и $k = 1$ существует единственная строка: "tiim".

# Решение задачи

**Разбор**

В авторском решении используется идея динамического программирования: обозначим за $dp[i][j][k]$ - количество строчек длины $i$ из латинских букв, у которых (с помощью удаления букв) можно выделить $j$ строк "tiim"и префикс ещё одного слова "tiim"длины ровно $k$ в конце. При пересчёте динамики нужно учитывать, что одна из 26 добавляемых в конец строки букв будет увеличивать длину последнего префикса (параметр $k$) на 1 (при этом если параметр $k$ был равен 3, то увеличится параметр $j$, а параметр $k$ станет равен 0, т.к. появилось ещё одно слово "tiim"в самом конце). Остальные же 25 букв не будут менять параметр $k$ и параметр $j$. Изначально можно инициализировать $dp[0][0][0] = 1$. Ответ после посчёта динамики будет равен $dp[n][k][0]+dp[n][k][1]+dp[n][k][2]+dp[n][k][3]$. Такая динамика хранит $O(n^2)$ состояний и пересчитывается за $O(n^2\Sigma)$, где $\Sigma$ — количество букв в алфавите (в нашем случае 26). Для лучшего понимания решения можно ознакомиться с кодом в архиве авторских решений.

**Авторское решение на языке C++:**

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

```
typedef long long ll;
using namespace std;

#define int ll

const int mod = 1'000'000'000 + 7;

int add(int a, int b) {
    a += b;
    if (a >= mod) {
        a -= mod;
    }
    return a;
}

int sub(int a, int b) {
    a -= b;
    if (a < 0) {
        a += mod;
    }
    return a;
}

int mul(int a, int b) {
    return int(1LL * a * b % mod);
}

const int alpha = 26;

void solve() {
    int n, k;
    cin >> n >> k;

    auto find_ans = [] (int n, int k) {
        vector dp(n + 1, vector<array<int, 4>>(k + 1));
        // dp[i][j][c] - сколько строк длины i, у которых алгоритм
        // выберет j строчек тиим и будет префикс длины c от нового тиима
        dp[0][0][0] = 1;
        for (int i = 1; i <= n; ++i) {
            for (int j = 0; j <= k; ++j) {
                dp[i][j][0] = mul(dp[i - 1][j][0], 25);
                if (j) {
                    (dp[i][j][0] += dp[i - 1][j - 1][3]) %= mod;
                }

                for (int c = 1; c < 4; ++c) {
                    dp[i][j][c] = (dp[i - 1][j][c] * 25LL + dp[i - 1][j][c - 1] * 1LL) % mod;
                }
            }
        }
        return ((ll)dp[n][k][0] + dp[n][k][1] + dp[n][k][2] + dp[n][k][3]) % mod;
    };
```

```
    cout << find_ans(n, k) << '\n';
}

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int t = 1;
    cin >> t;
    while (t--) {
        solve();
    }
    return 0;
}
```

# Задача 5. Большой шалаш

| | |
|---|---|
| Имя входного файла: | `стандартный ввод` |
| Имя выходного файла: | `стандартный вывод` |
| Ограничение по времени: | 1 секунда |
| Ограничение по памяти: | 256 мегабайт |

Иннокентий с друзьями решил построить шалаш. Для этого он направился в лес, в котором расположено $n$ деревьев. Каждое дерево представлено, как точка на двумерной плоскости с целочисленными координатами. Для того, чтобы построить шалаш, Иннокентий выбирает три дерева и строит между ними попарно стены из веток и листьев.

Помогите Кеше понять, какой максимальной площади он может построить шалаш.

Гарантируется, что существует три дерева, не лежащие на одной прямой.

Более формально - найдите, какой максимальной площади можно построить невырожденный треугольник на каких-либо трёх точках из заданных.

## Формат входных данных

В первой строке содержится число $n$ ($1 \leqslant n \leqslant 1000$) - количество деревьев в лесу.

В следующих $n$ строках находятся координаты каждого дерева - целые неотрицательные числа, не превосходящие $10^6$ (по два числа в каждой строке - $x$ и $y$ координата очередного дерева соответственно).

## Формат выходных данных

В единственной строке выведите ответ, округлённый до 1 знака после запятой. В качестве разделителя целой и дробной части используйте точку.

## Пример

| Входные данные | Результат работы программы |
|---|---|
| 5<br>4 0<br>8 3<br>0 3<br>3 3<br>4 1 | 12.0 |

## Замечание

Решения, верно работающие при $n \leqslant 500$, будут получать не менее 70 баллов.

# Решение задачи

**Разбор**

На 70 баллов можно было сдать задачу с помощью полного перебора всех троек точек с помощью трёх вложенных циклов с асимптотикой $O(n^3)$. Для этого нужно уметь находить площадь треугольника по координатам его вершин за $O(1)$. Это можно делать либо с помощью формулы Герона, либо с помощью косого произведения двух векторов-сторон. Опишем далее решение с асимптотикой $O(n^2 \log n)$. Построим выпуклую оболочку исходного множества точек (это можно сделать примитивным алгоритмом с итеративным добавлением за $O(n^2)$, информацию можете найти в интернете). Можно доказать, что искомый треугольник максимальной площади будет образован тремя точками, которые являются вершинами выпуклой оболочки. Оставим это упражнением для заинтересованного читателя (подсказка: попробуйте предположить, что какая-либо вершина оптимального треугольника не лежит

на выпуклой оболочке, а дальше зафиксируйте две остальные вершины и попробуйте "подвигать"третью). Давайте перебирать две вершины искомого треугольника во вложенных циклах. Теперь имеем зафиксированное основание перебираемого треугольника, нам нужно найти максимально удалённую от этого основания вершину, чтобы максимизировать высоту в перебираемом треугольнике, так как при этом и будет максимизироваться его площадь. Как за $O(\log n)$ определить третью вершину, которая будет наиболее удалённой от этой стороны? Для этого заметим, что расстояние от зафиксированной стороны до другой точки на выпуклой оболочке является выпуклой функцией в силу определения выпуклой оболочки (то есть нам выгодно брать вершинку где-то "примерно в серединке"между двумя зафиксированными вершинками, но не выгодно брать "близко"к одной из зафиксированных). Так как функция выпуклая, то можно применить тернарный поиск для нахождения максимума. Таким образом для каждой зафиксированной потенциальной стороны треугольника мы перебрали самую удалённую вершину от этой стороны. Так как искомый треугольник с максимальной площадью содержит хотя бы одну из сторон, которую мы перебрали (просто потому что мы перебрали все возможные стороны), то он был найден где-то в процессе нашего алгоритма. Данное решение получало 100 баллов. Можно было также реализовать решение, основанное на похожей идее, которое имеет асимптотику $O(n^2)$ и использует технику двух указателей. Оставим это как упражнение заинтересованному читателю.

**Авторское решение на языке C++:**

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

#define all(x) (x).begin(),(x).end()
#define len(x) (int)(x).size()

typedef long long ll;
typedef long double ld;
using namespace std;

using ptype = ld;

struct point {
    ptype x, y;

    point operator+(point oth) const {
        return {x + oth.x, y + oth.y};
    }

    point operator-(point oth) const {
        return {x - oth.x, y - oth.y};
    }

    point operator+=(point oth) {
        return *this = *this + oth;
    }

    point operator-=(point oth) {
        return *this = *this - oth;
    }
```

```
    point operator-() const {
        return {-x, -y};
    }

    ptype operator*(point oth) const {
        return x * oth.x + y * oth.y;
    }

    ptype operator%(point oth) const {
        return x * oth.y - oth.x * y;
    }

    ld getLen() const {
        return sqrtl(x * x + y * y);
    }

    ld operator~() const {
        return getLen();
    }

    bool operator==(point oth) const {
        return x == oth.x && y == oth.y;
    }
};


const ld eps = 1e-10;

bool equal(ld source, ld target) {
    return target - eps <= source && source <= target + eps;
}

struct Line {
    ptype a, b, c;

    Line(point p1, point p2) {
        ptype x1 = p1.x, y1 = p1.y;
        ptype x2 = p2.x, y2 = p2.y;

        a = (y1 - y2);
        b = (x2 - x1);
        c = -(a * x1 + b * y1);
    }

    static bool isParallel(Line first, Line second) {
        ptype A1 = first.a, B1 = first.b;
        ptype A2 = second.a, B2 = second.b;
        return equal(B1 * A2 - A1 * B2, (ld) 0);
    }
```

```
    static point getIntersec(Line first, Line second) {
        //a1x + b1y + c1 = 0
        //a2x + b2y + c2 = 0
        //A1*A2*x + B1*A2*y + C1*A2 = 0
        //A1*A2*x + A1*B2*y + A1*C2 = 0
        //(B1*A2 - A1*B2)y + (C1*A2 - A1*C2) = 0
        // y = (A1*C2 - C1*A2) / (B1*A2 - A1*B2)

        ptype A1 = first.a, B1 = first.b, C1 = first.c;
        ptype A2 = second.a, B2 = second.b, C2 = second.c;
        assert(!equal(B1 * A2 - A1 * B2, (ld) 0));

        ptype y = (A1 * C2 - C1 * A2) / (B1*A2 - A1*B2);
        ptype x;
        if (equal(A1, (ld) 0)) {
            assert(!equal(A2, ld(0)));
            x = (-C2 - B2*y) / A2;
        } else {
            x = (-C1 - B1*y) / A1;
        }
        return point{x, y};
    }

    ptype getDist(point p) {
        return (ld) (a * p.x + b * p.y + c) / sqrtl(a * a + b * b);
    }
};

typedef pair<point, point> Cut;

ld getDistCut(point p, Cut c) {
    if ((p - c.first)*(c.second - c.first) <= 0) {
        return (p - c.first).getLen();
    } else if ((p - c.second) * (c.first - c.second) <= 0) {
        return (p - c.second).getLen();
    }

    return abs(Line{c.first, c.second}.getDist(p));
}

ld getArea(point a, point b, point c) {
ld doubled_area = (b - a) % (c - a);
return abs(doubled_area) / 2.0;
}

void solve() {
int n; cin >> n;
vector<point> ps;

for (int i = 0; i < n; ++i) {
```

```
ld x, y;
cin >> x >> y;
ps.push_back(point{x, y});
}

auto get_convex_hull = [] (vector<point> ps) {
int n = len(ps);
point down = *min_element(all(ps), [&](const point p1, const point p2) {
        if (p1.y == p2.y) {
            return p1.x < p2.x;
        }
        return p1.y < p2.y;
    });

    sort(all(ps), [&](const point p1, const point p2) {
        if (p1 == down) {
            return true;
        }

        if (p2 == down) {
            return false;
        }

        if ((p1 - down) % (p2 - down) == 0) {
            return ~(p1 - down) < ~(p2 - down);
        }

        return (p1 - down) % (p2 - down) > 0; // значит p1 лежит раньше p2
    });

    assert(ps[0] == down);
    vector<point> convex;
    convex.push_back(ps[0]);
    convex.push_back(ps[1]);
    for (int i = 2; i < n; ++i) {
        while (len(convex) >= 2) {
            auto prev = convex[len(convex) - 2];
            auto cur = convex[len(convex) - 1];
            auto next = ps[i];
            if ((cur - prev) % (next - cur) > 0) {
                break;
            }
            convex.pop_back();
        }
        convex.push_back(ps[i]);
    }

    return convex;
};
```

```
auto hull = get_convex_hull(ps);
auto hull_doubled = hull;
for (auto &el : hull) {
hull_doubled.push_back(el);
}

ld ans = 0;
for (int i = 0; i < len(hull_doubled); ++i) {
for (int j = i + 1; j < i + len(hull) && j < len(hull_doubled); ++j) {
// можно искать между i и j
auto cut = make_pair(hull_doubled[i], hull_doubled[j]);
auto search = [&] () {
ld res = 0;
int left = i, right = j;
while (right - left > 2) {
int lm = (2 * left + right) / 3;
int rm = (left + 2 * right) / 3;

if (getDistCut(hull_doubled[lm], cut) < getDistCut(hull_doubled[rm], cut)) {
left = lm;
} else {
right = rm;
}
}

for (int k = max<int>(0, left - 10); k < min<int>(len(hull_doubled), right + 10); ++k) {
res = max<ld>(res, getArea(hull_doubled[i], hull_doubled[j], hull_doubled[k]));
}

return res;
};

ans = max<ld>(ans, search());
}
}

cout << fixed << setprecision(1) << ans;
}

int main() {
ios::sync_with_stdio(false);
cin.tie(nullptr);
solve();
return 0;
}
```

# Задача 6. Тайна переписки

| | |
|---|---|
| Имя входного файла: | `Стандартный ввод` |
| Имя выходного файла: | `Стандартный вывод` |
| Ограничение по времени: | 1 секунда |
| Ограничение по памяти: | 256 Мб |

Восьмиклассницы Нина и Зина хотят защитить свою переписку от посторонних, поэтому при каждой отправке сообщения шифруют его с использованием нового ключевого слова из англо-русского словаря. Порядок сообщений им известен и словари у них одинаковые, поэтому с ключом для расшифровки никаких проблем не возникает. Однако, школьницам быстро надоело делать это вручную и они захотели создать программу, которая будет шифровать сообщения за них.

В придуманном шифре повторения букв в ключевом слове удаляются, а алфавит переупорядочивается таким образом, что буквы в ключевом слове появляются последними, а перед ними идут остальные буквы алфавита в их обычном порядке.

Например, для заглавного латинского алфавита с ключевым словом «KEYWORD»:

`A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z`

становится:

`A|B|C|F|G|H|I|J|L|M|N|P|Q|S|T|U|V|X|Z|K|E|Y|W|O|R|D`

Таким образом `abc` преобразуется в `abc`; `xyz` преобразуется в `ord`.

Все сообщения написаны маленькими буквами.

Пробелы и знаки препинания остаются без изменений.

Напишите программу для шифрования сообщения

## Формат входных данных

В первой строке записан ключ шифра длиной $0 \leqslant n < 20$

Во второй строке записано сообщение для расшифровки $0 \leqslant n < 2000$

## Формат выходных данных

Расшифрованная строка

## Примеры

| Входные данные | Результат работы программы |
|---|---|
| keyword<br>abc | abc |
| keyword<br>xyz | ord |

# Решение задачи

**Авторское решение задачи на языке Python:**

```
abc = "abcdefghijklmnopqrstuvwxyz"
keyword =  input()
word =  input()
alpha=''
for letter in  abc:
    alpha += '' if letter in keyword else letter
alpha += ''.join(dict.fromkeys(keyword).keys())


s = word.translate(str.maketrans(abc, alpha))
print(s)
```