Задача 1. 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 6 tabiim 4 timi 16 abtmivmimabtioim	1 0 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 2. Экзамен

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

В одном известном вузе есть n студентов первого курса, которые изучают дисциплину «Алгоритмы и структуры данных». Оценка за этот предмет складывается из нескольких факторов:

- оценка за теоретические домашние работы
- оценка за контрольную работу
- оценка за семинарские занятия
- бонусная оценка
- оценка за экзамен

Каждая из оценок является целым числом от 0 до 10 включительно. Итоговая оценка за курс считается по следующей формуле: $O_{itog} = 0.25 \times O_{teor} + 0.25 \times O_{kr} + 0.2 \times O_{sem} + 0.3 \times O_{exam} + 0.1 \times O_{bon}$, где O_{itog} — итоговая оценка за курс, O_{teor} — оценка за теоретические домашние работы, O_{sem} — оценка за семинарские занятия и O_{bon} — бонусная оценка, O_{exam} — оценка за экзамен.

Обратите внимание, что оценка **округляется** до ближайшего целого числа, то есть по правилам арифметического округления (оценка с дробной частью $\geqslant 0.5$ округляется вверх).

Завтра пройдёт экзамен по упомянутой дисциплине. У каждого ученика есть желаемая итоговая оценка, которую он хочет получить за курс. Также каждый ученик знает все свои оценки, кроме оценки за экзамен. Посчитайте для каждого ученика, какую минимальную целую оценку от 0 до 10 включительно ему достаточно получить на экзамене, чтобы его итоговая оценка за курс была не ниже желаемой, либо выведите -1, если он не сможет добиться желаемой итоговой оценки независимо от оценки за экзамен.

Формат входных данных

В первой строке входных данных записано число n ($1 \leqslant n \leqslant 10^5$) — количество учеников, которые изучают дисциплину.

В каждой из следующих n строк записано по 5 целых чисел от 0 до 10 включительно — желаемая итоговая оценка соответствующего ученика, его оценка за теоретические домашние работы (O_{teor}), оценка за контрольную работу (O_{kr}), оценка за семинарские занятия(O_{sem}), бонусная оценка (O_{bon}).

Формат выходных данных

Выведите n строк: в каждой строке напечатайте одно целое число от 0 до 10 включительно — минимальную целую оценку за экзамен, которую нужно получить соответствующему ученику, чтобы его итоговая оценка была не ниже желаемой, либо -1, если независимо от оценки ученика за экзамен его итоговая оценка будет ниже желаемой.

Пример

Входные данные	Результат работы программы
3 10 4 5 6 4 8 9 9 9 9 4 10 10 10 10	-1 1 0

Решение задачи

Разбор

Для решения задачи достаточно для каждого ученика перебрать оценку от 0 до 10 за экзамен и посчитать по формуле из условия, достигается ли для какой-то из этих оценок желаемая итоговая оценка за курс, и если да, то среди всех подходящих выбрать минимальную. Для того, чтобы избежать ошибок округления, можно домножить все оценки на 100 и считать итоговую оценку в целых числах. Итоговая асимптотика решения $O(n)$.

Авторское решение на языке C++:

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

using namespace std;

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

    int n; cin >> n;
    auto solve_test = [&] (int need, int teor, int kr, int sem, int bon) -> int {
        for (int exam = 0; exam <= 10; ++exam) {
            int itog = teor * 25 + kr * 25 + sem * 20 + exam * 30 + bon * 10;
            if (itog >= need * 100 - 50) {
                return exam;
            }
        }
        return -1;
    };

    for (int i = 0; i < n; ++i) {
        int need, teor, kr, sem, bon;
        cin >> need >> teor >> kr >> sem >> bon;

        cout << solve_test(need, teor, kr, sem, bon) << '\n';
    }

    return 0;
}
```

Задача 3. Тестировщик

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

Тестировщику электрооборудования попали на проверку семисегментные индикаторы

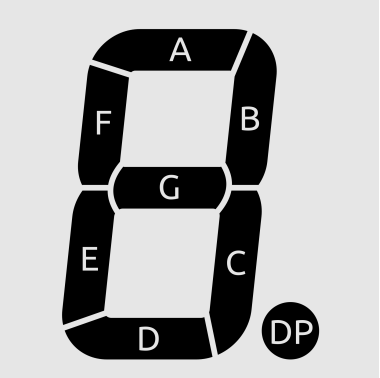

Рис. 1: Изображение индикатора

На каждом индикаторе поочередно выводится произвольный набор чисел. Верное отображение чисел - следующее:

```
 _          _    _          _    _    _    _    _
| |    |   _|   _|  |_|  |_   |_     |  |_|  |_|
|_|    |  |_    _|    |   _|  |_|    |  |_|   _|
```

Все выведенные числа выглядят корректно, при этом есть вероятность, что один из сегментов "сломан т.е. всегда светится или всегда погашен.

Найти сломанный сегмент. Если он есть - вывести его код (A-G), см. рис. 1, если все сегменты работают верно или данных недостаточно, чтобы найти сломанный - вывести 'None'

Формат входных данных

Три строки с выводом индикаторов в ASCII формате, каждый вывод индикатора представляет собой 3 строки и 3 столбца.

Количество индикаторов

$$n \leqslant 100$$

Формат выходных данных

Один символ (A-G), если есть битый сегмент или 'None'

Примеры

Входные данные	Результат работы программы
`` _ _ | | | _| |_| | _| ```	b

Решение задачи

Решение задачи на языке Python

```
items_tmpl = [' _ \n| |\n|_|', '   \n  |\n  |', ' _ \n _|\n|_ ',
' _ \n _|\n _|', '   \n|_|\n  |', ' _ \n|_ \n _|', ' _ \n|_ \n|_|',
' _ \n  |\n  |', ' _ \n|_|\n|_|', ' _ \n|_|\n _|']


def dead_segment(d):
    return (h:=[None]+[e[0]for e in zip(' a \nfgb\nedc',*d)if len({*e})==2])[len(h)==2]

line1 =  input()
line2 =  input()
line3 = input()

items = []



for i in range(len(line1)//3):
    item = line1[i*3]+line1[i*3+1]+line1[i*3+2]+'\n'+\
            line2[i*3]+line2[i*3+1]+line2[i*3+2]+'\n'+\
            line3[i*3]+line3[i*3+1]+line3[i*3+2];

    items.append(item);

print(dead_segment(items))
```

Задача 4. Круговой расклад

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

Профессор Трелони открыла новый способ гадания: она берёт двумерную плоскость и чертит на ней окружность радиуса r с центром в точке (0, 0). После этого она просит человека, которому она гадает, провести n прямых на этой же плоскости. Далее она считает количество треугольников, образованных этими прямыми, которые лежат полностью внутри окружности (если какая-то точка лежит на границе окружности, то считается что она тоже лежит внутри).

Помогите профессору Трелони понять, сколько будет треугольников, образованных данными прямыми и лежащих внутри окружности.

Формат входных данных

В первой строке содержится два числа n и r ($1 \leqslant n, r \leqslant 100$) - количество прямых и радиус окружности соответственно.

В следующих n строках следует описание всех прямых.

Каждая прямая описывается в отдельной строке тремя числами a, b, c ($-100 \leqslant a, b, c \leqslant 100$) - коэффициенты уравнения, задающего прямую (уравнением $ax + by + c = 0$).

Гарантируется, что во входных данных нет двух совпадающих прямых.

Формат выходных данных

В ответе запишите одно число - искомое количество треугольников.

Пример

Входные данные	Результат работы программы
4 5 -2 -3 1 -1 1 2 -3 -1 -3 1 0 -2	2

Замечание

Треугольник, образованный тремя прямыми - это треугольник, составленный из их точек пересечения (если они попарно пересекаются).

Треугольники, образованные набором прямых, - это множество треугольников, образованных всевозможными тройками из данных прямых.

Решение задачи

Разбор

Заметим, что треугольник лежит внутри окружности тогда и только тогда, когда все три его вершины лежат внутри этой окружности. Проверить, что вершина лежит внутри окружности, можно с помощью уравнения окружности. Чтобы найти все возможные треугольники, нужно уметь пересекать две прямые. Это можно сделать, если решить систему из двух уравнений прямых, выразив x и y координаты точек пересечения. Для уточнения деталей рекомендуется посмотреть код авторского решения.

Авторское решение на языке C++:

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

typedef long long ll;
using namespace std;

#define int ll

const ll INF = 1e18 + 100;

struct line {
    int a, b, c;

    bool operator<(line oth) const {
        return a < oth.a || (a == oth.a && b < oth.b) ||
        (a == oth.a && b == oth.b && c < oth.c);
    }
};

bool IsIntersecInside(line l1, line l2, ll r) {
    auto [a1, b1, c1] = l1;
    auto [a2, b2, c2] = l2;

    ll znam = a1 * b2 - a2 * b1;

    if (znam == 0) {
        return false;
    }

    // x^2 + y^2 <= r^2
    //

    ll x = (c1 * b2 - c2 * b1);
    ll y = (a1 * c2 - a2 * c1);

    return x * x + y * y <= r * r * znam * znam;
}

void solve() {
    int n, r; cin >> n >> r;
    vector<line> ls;

    for (int i = 0; i < n; ++i) {
        int a, b, c;
        cin >> a >> b >> c;
        ls.push_back(line{a, b, c});
    }
    int ans = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = i + 1; j < n; ++j) {
            for (int k = j + 1; k < n; ++k) {
```

```
                bool res = IsIntersecInside(ls[i], ls[j], r);
                res = res && IsIntersecInside(ls[i], ls[k], r);
                res = res && IsIntersecInside(ls[j], ls[k], r);
                ans += res;
            }
        }
    }

    cout << ans;
}

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

Задача 5. 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 4 1 5 1 30 4	1 126 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 Σ — 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 6. Громкие шкафы

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

Чем больше шкаф, тем громче он падает

— Народная мудрость

В магазине мебели есть отдел, в котором продаются шкафы. В этом отделе в ряд расположены n шкафов с высотами $a_1, a_2, \ldots, a_n$ соответственно. Известно, что расстояние между любыми двумя соседними шкафами составляет 1 метр. Хулиган Петя захотел наделать шума в этом магазине. Поэтому он решил толкнуть один из шкафов влево или вправо. При этом, если шкаф падает, то он задевает другие шкафы, которые тоже начинают от этого падать в ту же сторону. Шкаф высотой x заденет все шкафы, которые находятся на расстоянии строго меньше x от него. Также, когда шкаф высотой x падает, то он издаёт x децибел шума. Это значит, что суммарный шум, наделанный падением каких-то шкафов, равен сумме их высот. Петя задался вопросом, какой максимальный суммарный шум он сможет наделать в магазине, если толкнёт ровно один шкаф так, чтобы он начал падать влево или вправо.

Формат входных данных

В первой строке содержится число t ($1 \leqslant t \leqslant 10$) — количество наборов входных данных, для которых необходимо найти решение. Каждый набор входных данных начинается с числа n ($1 \leqslant n \leqslant 10^5$) — количества шкафов в магазине. В следующей строке каждого набора находится массив a из n чисел ($1 \leqslant a_i \leqslant n$), задающий высоты для шкафов.

Формат выходных данных

Для каждого набора входных данных выведите одно число в отдельной строке — максимальный шум, который может наделать Петя в магазине, толкнув ровно один любой шкаф влево или вправо.

Система оценки

Решения, работающие корректно на тестах, где $n \leqslant 1000$, будут получать не менее 60 баллов.

Примеры

Входные данные	Результат работы программы
3 6 3 1 5 1 5 3 11 1 3 4 3 1 9 11 9 8 1 1 9 6 8 6 5 1 2 1 4 1	18 50 34
1 10 1 1 1 1 1 1 1 1 1 1	1

Решение задачи

Разбор

Представим, что мы толкнули какой-либо шкаф в какую-либо сторону (без ограничения общности считаем, что толкнули его влево). Это означает, что какой-то подотрезок шкафов левее упал. При этом первый шкаф левее который не упал находится достаточно далеко от каждого из свалившихся шкафов, поскольку они его не задели. Как найти первый шкаф левее, который не будет задет? За $O(n)$ можно просто идти влево по шкафам и поддерживать самое левое место, куда свалился какой-либо из упавших шкафов. Если следующий шкаф слева находится правее этого места, то он тоже свалится влево и возможно сделает самое левое место ещё левее. Таким образом, научились решать задачу за $O(n^2)$: просто симулируем падение каждого шкафа влево и вправо за $O(n)$. Так как всего шкафов n, то суммарное количество действий для всех симуляций будет равно $O(n^2)$. Такое решение набирало 60 баллов. Далее опишем решение на 100 баллов. Посчитаем $dp[i]$ - самый левый шкаф, который упадёт, если мы толкнём i-й шкаф. Будем пересчитывать dp слева направо. Изначально считаем, что $dp[i] = i$. Заметим, что $dp[i]$ можно пересчитать как $\min(dp[j])$ по всем j таким что $i - j < a_i$, что равносильно $j > i - a_i$. С помощью вышеописанного способа эту динамику можно посчитать за $O(n^2)$. Научимся считать её за $O(n \log n)$. Для этого нужно использовать структуру данных, называемую "дерево отрезков": она умеет находить минимум на подотрезке массива за $O(\log n)$ и обновлять значение по какому-то индексу массива за $O(\log n)$. Будем хранить все насчитанные состояния dp в дереве отрезков. Дальше для пересчёта $dp[i]$ используем это дерево отрезков, чтобы найти искомый минимум $\min(dp[i - a_i + 1], dp[i - a_i + 2], \ldots, dp[i])$ за $O(\log n)$. После этого обновим через найденное число значение $dp[i]$ и пойдём вычислять следующие значения dp. Таким образом ответ будет равен $\max(i - dp[i] + 1)$ по всем возможным i. Для того чтобы учесть случай, когда мы толкнули шкаф вправо, можно применить известный для такого типа задач трюк: перевернём изначальный массив и будем решать такую же задачу (толкать шкаф влево) уже на нём. Из двух полученных ответов для исходного и перевернутого массива выберем максимальный. В решении мы сделали $O(n)$ запросов в дерево отрезков, каждый из которых отработал за $O(\log n)$, значит итоговая асимптотика решения $O(n \log n)$.

Авторское решение на языке C++:

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

#define all(x) (x).begin(),(x).end()

typedef long long ll;
using namespace std;

#define int ll

const int INF = 1e9 + 100;
struct segtree {
    int size = 1;
    vector<pair<int, int>> tree;

    segtree(int n) {
        while (size < n) size *= 2;
```

```
        tree.assign(size * 2 - 1, {INF, INF});
    }

    void upd(int i, ll v) {
        upd(i, v, 0, 0, size);
    }

    pair<ll, int> get(int l, int r) {
        return get(l, r, 0, 0, size);
    }

private:
    void upd(int i, ll v, int x, int lx, int rx) {
        if (rx - lx == 1) {
            tree[x] = {v, i};
        } else {
            int m = (rx + lx) >> 1;
            if (i < m) {
                upd(i, v, 2 * x + 1, lx, m);
            } else {
                upd(i, v, 2 * x + 2, m, rx);
            }
            tree[x] = min(tree[2 * x + 1], tree[2 * x + 2]);
        }
    }

    pair<ll, int> get(int l, int r, int x, int lx, int rx) {
        if (lx >= r || rx <= l) {
            return {INF, INF};
        }

        if (l <= lx && rx <= r) {
            return tree[x];
        }

        int m = (rx + lx) >> 1;
        return min(get(l, r, 2 * x + 1, lx, m), get(l, r, 2 * x + 2, m, rx));
    }
};

void solve() {
    int n;
    cin >> n;
    vector<int> a(n);
    for (auto &el: a) cin >> el;

    auto solve_60 = [](int n, vector<int> a) {
        auto solve_left = [&]() -> ll {
            // что если толкнем шкаф влево
            ll ans = 0;
```

```
        for (int i = 0; i < n; ++i) {
            int left = i - a[i] + 1;
            ll cur = 0;
            for (int j = i; j >= 0 && j >= left; --j) {
                cur += a[i];
                left = min(left, j - a[j] + 1);
            }

            ans = max(ans, cur);
        }
        return ans;
    };

    ll ans = solve_left();
    reverse(all(a));
    ans = max<ll>(ans, solve_left());
    return ans;
};

auto solve_100 = [](int n, vector<int> a) {
    auto solve_left = [&]() -> ll {
        vector<ll> pref(n + 1);
        for (int i = 0; i < n; ++i) pref[i + 1] = pref[i] + a[i];
        auto get_sum = [&pref](int left, int right) {
            return pref[right + 1] - pref[left];
        };

        segtree st(n);
        vector<ll> dp(n);

        dp[0] = a[0];
        st.upd(0, 0);

        for (int i = 1; i < n; ++i) {
            auto [mn_val, ind] = st.get(max<int>(0, i - a[i] + 1), i);
            if (ind == INF) {
                assert(a[i] == 1);
                dp[i] = a[i];
            } else {
                if (mn_val >= i - a[i] + 1) {
                    dp[i] = get_sum(max<int>(0, i - a[i] + 1), i);
                } else {
                    dp[i] = dp[ind] + get_sum(ind + 1, i);
                }
            }
            st.upd(i, max<int>(0, i - a[i] + 1));
        }

        return *max_element(all(dp));
    };
```

```
        ll ans = solve_left();
        reverse(all(a));
        ans = max<ll>(ans, solve_left());
        return ans;
    };

    cout << solve_100(n, a) << '\n';
}

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