Задача A. Вопрос встал ребром

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

После тяжёлой рабочей недели n друзей собрались вместе отдохнуть и поесть пиццы. Они решили заказать одну большую пиццу и разрезать её на несколько равных кусков. Чтобы всё было справедливо, каждому другу из компании достанется одинаковое количество кусочков. У каждого из друзей есть своё любимое число, и каждый хочет, чтобы количество доставшихся ему кусочков делилось на это число. Помогите друзьям определить, на какое минимальное количество равных кусков нужно разрезать пиццу, чтобы каждый получил равное количество кусков, которое делится на его любимое число.

Формат входных данных

В первой строке дано число n ($2 \leqslant n \leqslant 10$) – количество друзей. Во второй строке задано n чисел (каждое от 1 до 10) – любимые числа друзей.

Формат выходных данных

В ответе выведите одно число – искомое минимальное количество кусков.

Пример

стандартный ввод	стандартный вывод
2 2 3	12

Замечание

Разберём пример из условия. Если разрезать пиццу на 12 кусков, то каждый из двух друзей получит по 6 кусков. 6 делится на 2 и на 3. Можно показать, что разрезать пиццу на меньшее количество кусков так, чтобы удовлетворить все капризы друзей, не получится.

Разбор

Обозначим минимальное количество кусков, на которые нужно разрезать пиццу, за M. M в первую очередь должно делиться на n (ведь каждому другу должно достаться поровну кусочков), а $\frac{M}{n}$ должно делиться на каждое из любимых чисел друзей. Значит $\frac{M}{n}$ больше или равно, чем НОК (наименьшее общее кратное) любимых чисел друзей. Обозначим этот НОК за K. Тогда $M \geqslant nK$. Заметим, что nK кусков пиццы можно разделить поровну между друзьями, чтобы выполнялись все нужные требования, просто отдав каждому другу по K кусков (ведь K как НОК делится на каждое из любимых чисел друзей). Отсюда получаем ответ $M = nK$. Вычислить НОК всех любимых чисел друзей можно было разложив каждое число на множители или с помощью вычисления НОД алгоритмом Евклида. Отсюда получаем решение с асимптотикой $O(nA)$ или $O(n \log A)$, где A – максимальное из любимых чисел друзей.

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;

#define int ll

void solve() {
    int n; cin >> n;
    vector<int> a(n);
    for (auto &el : a) cin >> el;
    int x = a[0];
    for (int i = 1; i < n; ++i) {
        x = x / __gcd(a[i], x) * a[i]; // lcm(a[i], x)
    }
    cout << x * n;
}

signed main(signed argc, char* argv[]) {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    solve();
    return 0;
}
```

Задача B. Теория струн

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

Музыканту Иннокентию на день рождения подарили новую шестиструнную гитару. На гитаре есть 11 ладов, на которых можно зажимать каждую из шести струн. Для того чтобы получать разные ноты при игре на гитаре, нужно зажимать разные струны на разных ладах, дёргая зажатые струны (при этом можно дёрнуть струну, не зажимая её ни на каком ладу; давайте считать, что в таком случае мы "зажали струну на ладу под номером 0"и сыграли её).

Так как Иннокентий ещё и математик, то он занумеровал все элементы гитары: струны от самой тонкой (она имеет номер 1) до самой толстой (она имеет номер 6), лады от самого дальнего (он имеет номер 1) до самого ближнего (он имеет номер 11).

Для того чтобы занумеровать ещё и ноты, Иннокентий решил изучить нотную теорию для его модели гитары и узнал следующее: при зажатии струны i на ладу x получается такая же нота, как и при зажатии струны $i+1$ на ладу $x+d_i$ (если такое зажатие возможно, т.е. $x+d_i \leqslant 11$).

После изучения этой теории Иннокентий занумеровал ноты, начиная от самой низкой (она получается, если сыграть шестую струну, не зажимая никакого лада, или в наших обозначениях: "зажать струну 6 на ладу под номером 0"), которой присвоил номер 1 и заканчивая самой высокой (которая получается, если сыграть первую струну, зажатую на 11-ом ладу).

Ноты были занумерованы Иннокентием так, что одинаковым нотам соответствуют одинаковые номера, а разным нотам соответствуют разные номера, при этом более низкой ноте соответствует меньший номер, чем более высокой. При этом было использовано минимальное возможное количество чисел для такой нумерации (другими словами, никакое число от 1 до максимального не было "пропущено").

Чтобы закрепить всю эту теорию у себя в голове, Иннокентий придумал себе N вопросов для самопроверки. Вопрос под номером j (для $j = 1, \ldots, N$) звучит так: "Если я зажму струну a_j на ладу b_j и сыграю её, то ноту с каким номером я получу?". Иннокентий ответил на все вопросы в уме. Но он думает, что мог ошибиться, поэтому просит вас написать программу, которая найдёт ответ на каждый из его вопросов.

Формат входных данных

Первая строка содержит пять чисел d_1, d_2, d_3, d_4, d_5 ($1 \leqslant d_i \leqslant 11$) – "разницы"между соседними струнами из нотной теории, изученной Иннокентием. Вторая строка содержит одно целое число N ($1 \leqslant N \leqslant 12 \times 6$) – количество вопросов. В следующих N строках следует описание вопросов. В вопросе с номером j содержится два числа a_j и b_j ($1 \leqslant a_j \leqslant 6, 0 \leqslant b_j \leqslant 11$) – номер струны и номер лада из вопроса соответственно.

Формат выходных данных

В ответе выведите N строк. В каждой строке выведите одно число – ответ на соответствующий запрос.

Пример

стандартный ввод	стандартный вывод
5 4 5 5 5 7 6 0 5 0 4 0 3 0 2 0 1 0 1 11	1 6 11 16 20 25 36

Замечание

Для обычной шестиструнной гитары из реальной жизни $d_1 = d_3 = d_4 = d_5 = 5$ и $d_2 = 4$. Обратите внимание, что "зажать струну на ладу 0"означает "не зажимать струну ни на каком ладу".

Разбор

Данная задача является некоторым обобщением концепции гитарной табулатуры. Для решения достаточно было пройтись в нужном порядке двумя вложенными циклами по номерам струн и номерам ладов и задать ноту на каждую пару (струна, лад), сохранив эту ноту в двумерном массиве ответов. Для этого нужно было посмотреть, задавали ли мы до этого идентичную ноту на прошлых струнах (например, сравнив номер лада минус d_{i-1} с нулём), и если задавали то пронумеровать так же, как и идентичную ноту, а иначе присвоить оригинальный номер, который равен максимальному ранее присвоенному номеру, увеличенному на 1. Для лучшего понимания можете посмотреть код авторского решения.

Авторское решение на языке C++:

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

typedef long long ll;
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    vector<int> d = {0, 5, 4, 5, 5, 5};
    for (int i = 1; i <= 5; ++i) {
        cin >> d[i];
    }

    vector<vector<int>> answer(7, vector<int>(12));
    for (int x = 0; x <= 11; ++x) {
        answer[6][x] = x + 1;
    }

    for (int i = 5; i >= 1; --i) {
        for (int x = 0; x <= 11; ++x) {
            if (x + d[i] <= 11) {
                answer[i][x] = answer[i + 1][x + d[i]];
            } else {
                answer[i][x] = answer[i][x - 1] + 1;
            }
        }
    }

    int N; cin >> N;
    for (int it = 0; it < N; ++it) {
        int a, b;
        cin >> a >> b;
        cout << answer[a][b] << '\n';
    }
```

```
    return 0;
}
```

Задача C. Сходка музыкантов

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

Сразу после того, как Иннокентий получил в подарок на день рождения гитару, он решил организовать сходку музыкантов. Он решил позвать на сходку n своих друзей, попутно пронумеровав их от 1 до n (так как Иннокентий математик, он любит нумеровать всё вокруг). Так как все друзья Иннокентия очень важные люди, то все они не смогут прийти к началу сходки, а придут только некоторое количество минут после начала: i-ый друг обещал прийти спустя a_i минут после начала сходки. Иннокентий будет считать сходку классной, если в какой-то момент времени на сходке присутствует хотя бы k его друзей. Друзья хотят, чтобы сходка была классной, а также чтобы каждый человек пробыл на сходке равное количество минут, ведь так будет справедливее всего. Так как они очень занятые, то они хотят выбрать минимально возможное количество минут, которое каждый человек обязан провести на сходке, чтобы сходка была классной. Помогите друзьям определить, сколько минут каждый из них должен провести на сходке, чтобы Иннокентий считал её классной.

Формат входных данных

В первой строке вводится два целых числа n и k ($1 \leqslant k \leqslant n \leqslant 10^5$) – количество друзей Иннокентия, а также количество друзей Иннокентия, которые должны одновременно присутствовать на сходке, чтобы Иннокентий посчитал её крутой. Во второй строке вводится n целых чисел $a_1, a_2, \ldots, a_n$ ($1 \leqslant a_i \leqslant 10^9$) – на сколько минут каждый друг опоздает на сходку. Гарантируется, что все друзья пришли в разное время, то есть все числа в массиве a различны.

Формат выходных данных

В ответе выведите одно число – минимальное количество минут, которое каждый обязан провести на сходке, чтобы Иннокентий считал её крутой.

Система оценки

Решения, правильно работающие для $n \times max(a_i) \leqslant 10^5$, будут набирать не менее 40 баллов.

Пример

стандартный ввод	стандартный вывод
5 3 1 10 3 13 16	7

Замечание

Обратите внимание, что во момент ухода со сходки человек не находится на ней. Другими словами, если человек пришёл спустя x минут после начала и провел t минут на сходке, то он находился на сходке в полуинтервал времени $[x, x + t)$ спустя начала сходки.

Разбор

Авторское решение данной задачи использует решение за $O(n \log n \log A)$, где A - максимальный возможный ответ. Основная идея решения заключается в двоичном поиске по ответу: так мы сводим задачу нахождения ответа к задаче о проверке, является ли вечеринка классной при фиксированном ответе. Для того, чтобы проверить что ответ не превосходит определённого числа t, нам нужно проверить, что существует точка на координатной прямой, которая содержит хотя бы k полуинтервалов вида $[a_i, a_i + t)$. Понятно, что достаточно проверить все точки, которые являются началом какого-либо из этих полуинтервалов. Это можно сделать с помощью сканирующей прямой, пройдясь упорядоченно по точкам открытия и закрытия всех полуинтервалов $[a_i, a_i + t)$ и для каждой точки открытия проверяя, сколько открытых полуинтервалов содержат эту точку, а также изменяя счётчик текущих открытых полуинтервалов в зависимости от типа точки (началом или концом

полуинтервала она является). Сложность проверки составляет $O(n \log n)$, т.к. для проверки мы сортируем $O(n)$ точек, являющихся началами и концами отрезков. Сложность всего решения составляет $O(n \log n \log A)$, т.к. проверка осуществляется на каждой из $O(\log A)$ итераций двоичного поиска по ответу. Для лучшего понимания рекомендуется посмотреть код в архиве с решениями.

Авторское решение на языке C++:

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

typedef long long ll;
using namespace std;

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

    int n, k;
    cin >> n >> k;

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

    auto check = [&] (int time) {
        vector<pair<int, int>> events;
        for (int i = 0; i < n; ++i) {
            events.push_back({a[i], 1});
            events.push_back({a[i] + time, -1}); // если кто-то уходит в то же время как другой пр
        }

        sort(events.begin(), events.end());

        int cnt = 0;
        for (auto [x, type] : events) {
            cnt += type;
            if (cnt >= k) {
                return true;
            }
        }

        return false;
    };

    int l = -1, r = 1e9 + 100;
    while (r - l > 1) {
        int mid = (r + l) >> 1;
        if (check(mid)) {
            r = mid;
        } else {
            l = mid;
        }
    }

    assert(r > 0);
```

```
    cout << r;

    return 0;
}
```

Задача D. Сложность мелодии

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

Эта задача является продолжением задачи "Теория струн". Для полного понимания условия прочитайте сначала задачу "Теория струн". После того, как Кеша пронумеровал все ноты, он решил начать играть мелодии. Любая мелодия представляет собой последовательность нот. Каждая нота в последовательности описывается и задаётся своим номером (в нумерации, которую придумал Иннокентий в задаче "Теория струн"). Представим, что Иннокентий сначала сыграл струну i на ладу x, а сразу после этого сыграл струну j на ладу y. Расстоянием между этими последовательно сыгранными нотами назовём число $|i-j|+|x-y|$. Заметим, что одну и ту же ноту можно сыграть на разных струнах, поэтому расстояние между двумя одинаковыми нотами может быть разным и зависит от того, на каких струнах мы эти ноты сыграли. Для лучшего понимания смотрите секцию пояснение. Сложностью мелодии назовём суммарное расстояние между всеми соседними сыгранными нотами (заметим, что сложность мелодии может быть разной в зависимости от того, на каких струнах мы играли ноты из неё). Для заданной мелодии определите, с какой минимально возможной сложностью можно её сыграть.

Формат входных данных

Первая строка содержит пять чисел d_1, d_2, d_3, d_4, d_5 ($1 \leqslant d_i \leqslant 11$) – "разницы"между соседними струнами из нотной теории, изученной Иннокентием (означающие то же самое, что и в задаче "Теория струн"). Вторая строка содержит одно целое число n ($2 \leqslant n \leqslant 10^5$) – длину последовательности нот. Третья строка содержит n натуральных чисел – номера нот в последовательности (нумерация определяется как в задаче "Теория струн"). Гарантируется, что все номера нот корректны и любую ноту можно каким-либо образом сыграть на гитаре.

Формат выходных данных

В ответе выведите одно число – минимально возможную сложность заданной мелодии.

Система оценки

Решения, правильно работающие при $n \leqslant 5$, получат не менее 30 баллов.

Пример

стандартный ввод	стандартный вывод
5 4 5 5 5 2 1 6	1

Замечание

Для обычной шестиструнной гитары из реальной жизни $d_1 = d_3 = d_4 = d_5 = 5$ и $d_2 = 4$. В примере из условия ноты 1 и 6 обе можно сыграть на шестой струне, тогда расстояние между ними будет равно $|6-6|+|0-5| = 5$. Но более легко будет сыграть ноту 1 на шестой струне, а ноту 5 на пятой струне, тогда расстояние между нотами будет равно $|6-5|+|0-0| = 1$.

Разбор

Для того, чтобы определять по паре (струна, лад) номер ноты, воспользуемся решением задачи "Теория струн". Для тестов, где $n \leqslant 5$, можно было написать решение, которое перебирало все возможные варианты (струна, лад) для каждой из сыгранных нот и определяло бы минимальную сложность среди всех перебранных вариантов. Для получения 100 баллов можно было использовать концепцию динамического программирования: за $dp[i][j]$ обозначим минимальное расстояние, за которое мы сыграли первые i нот, если последнюю сыгранную ноту мы сыграли на струне j.

Пересчёт такой динамики достаточно тривиален: будем перебирать прошлую струну, на которой мы сыграли ноту и следующую струну, на которой собираемся сыграть ноту. Для каждой из струн найдём, на каком ладу мы должны её зажать, чтобы сыграть нужную ноту. Таким образом мы найдём расстояние между последней сыгранной нотой и нотой, которую мы собираемся сыграть. Тогда $dp[i][j]$ равно минимуму по всем $dp[i-1][k] + dist$, где dist – это расстояние между нотой a_i, которую мы собираемся на струне j и нотой a_{i-1}, которую сыграли на струне k. Итоговый ответ будет равен минимальному числу в $dp[n]$. Для лучшего понимания рекомендуем посмотреть код авторского решения.

Авторское решение на языке C++:

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

typedef long long ll;
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    vector<int> d = {0, 5, 4, 5, 5, 5};
    for (int i = 1; i <= 5; ++i) {
        cin >> d[i];
    }

    auto get_numeration = [&]() {
        vector<vector<int> > answer(7, vector<int>(12));
        for (int x = 0; x <= 11; ++x) {
            answer[6][x] = x + 1;
        }

        for (int i = 5; i >= 1; --i) {
            for (int x = 0; x <= 11; ++x) {
                if (x + d[i] <= 11) {
                    answer[i][x] = answer[i + 1][x + d[i]];
                } else {
                    answer[i][x] = answer[i][x - 1] + 1;
                }
            }
        }

        return answer;
    };


    auto numeration = get_numeration();
    const int INF = 1e9 + 100;

    auto find_note_at_string = [&](int note, int str) {
        for (int x = 0; x <= 11; ++x) {
            if (numeration[str][x] == note) {
                return x;
            }
        }
```

```
        return INF;
    };

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

    vector<vector<int> > dp(n, vector<int>(7, INF));
    // dp[i][j] - минимальная стоимость если сыграл ноты до i-ой и последнюю ноту сыграл на струне
    for (int j = 1; j <= 6; ++j) {
        int x = find_note_at_string(a[0], j);
        if (0 <= x && x <= 11) {
            dp[0][j] = 0;
        } else {
            dp[0][j] = INF;
        }
    }

    for (int i = 1; i < n; ++i) {
        for (int prev = 1; prev <= 6; ++prev) {
            int x = find_note_at_string(a[i - 1], prev);

            if (x < 0 || x > 11) {
                continue;
            }

            for (int nxt = 1; nxt <= 6; ++nxt) {
                int y = find_note_at_string(a[i], nxt);

                if (0 <= y && y <= 11) {
                    int dist = abs(x - y) + abs(prev - nxt);
                    dp[i][nxt] = min(dp[i][nxt], dp[i - 1][prev] + dist);
                }
            }
        }
    }

    cout << *min_element(dp.back().begin(), dp.back().end());

    return 0;
}
```

Задача E. Римское произведение

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

В римской системе счисления можно представить любое число от 1 до 3999 (смотрите картинку для лучшего понимания). Гай Юлий Цезарь, один из правителей Римской империи, решил оптимизировать написание чисел в римской системе счисления. Он заметил, что иногда короче записывать число как произведение нескольких римских чисел. Длиной такой записи назовём суммарную длину строки без пробелов со знаками умножения. Например, длина представления XXI*XXX равна 7. При этом сама римская запись числа тоже является представлением (например, IX имеет длину 2). Он поручил вам написать программу, которая вычислит для T чисел длину самого короткого представления каждого из них. Для лучшего понимания смотрите тесты и примечания.

Формат входных данных

В первой строке содержится число T ($1 \leqslant T \leqslant 10^5$) – количество чисел, для которых нужно найти длину самого короткого представления. В каждой из следующих T строк записано по одному числу. Каждое из чисел может принимать значения от 1 до 10^5.

Формат выходных данных

Для каждого числа в отдельной строке выведите ответ на задачу – минимальное возможное количество символов в представлении числа в виде произведения римских чисел. Если число **невозможно** представить в виде произведения римских чисел от 1 до 3999, выведите для него в ответе 100001.

Система оценки

Решения, верно работающие для $t \leqslant 10$ будут получать не менее 30 баллов. Решения, верно работающие для $t \leqslant 1000$ будут получать не менее 60 баллов.

Пример

стандартный ввод	стандартный вывод
8 1 2 3 4 12 4000 38 3977	1 2 3 2 3 4 6 9

Замечание

Например, число 12 можно представить в виде XII (12) или VI*II (6 * 2) или как IV*III (4*3). Таким образом, самым коротким представлением числа 12 является XII, оно имеет длину 3. Для числа 4000 самым коротким представлением является IV*M. Длина такого представления равна 4. Для числа 38 самым коротким представлением является XIV*II. Оно имеет длину 6.

Разбор

Обозначим искомое минимальное количество символов в представлении (в виде произведения римских чисел) числа m за ans_m. Пусть мы хотим представить некоторое число n в виде произведения чисел, записанных в римской системе, минимальной длины. Есть два варианта: либо мы можем просто записать это число в римской системе, либо это число является произведением не менее двух

чисел, которые записаны в римской системе. Давайте исходно считать ответом для числа n его длину представления в римской системе (если оно записывается в римской системе, а иначе 100001). Понятно, что от перестановки множителей произведение не будет меняться. Давайте переберём первый множитель среди делителей числа n. Тогда ans_n равен минимуму по всем возможным делителям числа n выражения $ans_d + ans_{\frac{n}{d}} + 1$, где d – это произвольный делитель числа n. Эта формула получается так: мы берём минимальную искомую запись числа d, добавляем к ней справа символ умножения и после этого приписываем справа минимальную возможную запись числа $\frac{n}{d}$. Мы не можем заранее знать, с какого делителя начинается минимальная запись, поэтому как раз перебираем все возможные делители числа n. Для того, чтобы мы знали корректные значения всех ans_i, где $i < n$, будем пересчитывать массив ans в порядке возрастания индексов. В зависимости от эффективности алгоритма поиска делителей числа решение могло набирать 30-100 баллов. На 100 баллов предполагалось решение за $O(A \log A + T)$ (где A – максимальное число в запросах) с использованием алгоритма нахождения всех делителей для каждого от 1 до A, в котором мы для каждого числа от 1 до A проходимся циклом по всем кратным ему числам и добавляем этот делитель в отдельный массив для каждого числа (таким образом мы выполним $A + \lfloor \frac{A}{2} \rfloor + \lfloor \frac{A}{3} \rfloor + \ldots + \lfloor \frac{A}{A} \rfloor = O(A \log A)$ операций.

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;

#define int ll
#define double ld

vector<int> digits = {1000, 900,
    500, 400,
    100, 90,
    50, 40,
    10, 9,
    5, 4,
    1};
vector<int> length = {1, 2,
    1, 2,
    1, 2,
    1, 2,
    1, 2,
    1, 2,
    1};


int roman_len(int x) {
    assert(len(digits) == len(length));
    int ans = 0;
    for (int i = 0; i < len(digits) && x > 0; ++i) {
        while (x >= digits[i]) {
            x -= digits[i];
```

```
            ans += length[i];
        }
    }

    return ans;
}

constexpr int maxn = 100'000 + 1;

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

    vector<int> ans(maxn, maxn);
    vector<vector<int>> dels(maxn);

    for (int i = 2; i < maxn; ++i) {
        for (int j = i; j < maxn; j += i) {
            dels[j].push_back(i);
        }
    }

    for (int i = 1; i < 4000; ++i) {
        ans[i] = min(ans[i], roman_len(i));
    }

    for (int i = 1; i < maxn; ++i) {
        for (auto del : dels[i]) {
            ans[i] = min(ans[i], ans[del] + ans[i / del] + 1);
        }
    }

    for (auto el : qs) {
        cout << ans[el] << '\n';
    }
}

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
#ifdef ONPC
    assert(freopen("input.txt", "r", stdin));
    assert(freopen("output.txt", "w", stdout));
#endif

    int TT = 1;
#ifdef MT
    cin >> TT;
#endif
    while (TT--) {
        solve();
    }
```

```
#ifdef ONPC
    cerr << "Time(ms): " << 1000 * clock() / CLOCKS_PER_SEC << endl;
#endif
    return 0;
}
```

Задача F. Красивый стол

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

Бизнесмен Владимир решил купить ресторан. В центре ресторана он решил расположить большой прямоугольный стол размерами N дециметров в длину и M дециметров в ширину. Для красоты Владимир отпилил от двух противоположных углов стола по четвертинке окружностей радиуса R дециметров (центры окружностей находятся ровно в противоположных углах стола, для лучшего понимания смотрите рисунок). Гарантируется, что стол после этого **остался связным** и не развалился на две части. В случайное место этого стола поставили круглую тарелку с деликатесом, радиусом 1 дециметр. Известно, что центр этой тарелки попал на стол или на его границы, ведь если бы центр находился вне стола, то тарелка упала бы и разбилась, что очень сильно расстроило бы Владимира. Владимир задался вопросом, с какой вероятностью случайно поставленная тарелка не выпирает за границы стола, то есть если посмотреть на стол сверху, то границы тарелки находятся строго внутри стола (возможно касаясь границ стола). Помогите Владимиру ответить на эту задачу. Ответ выведите в процентах, округлив до ближайшего целого числа (то есть ваш ответ должен быть равен целому числу от 0 до 100).

Формат входных данных

В единственной строке входных данных находятся три целых числа N, M и R ($5 \leqslant N, M, R \leqslant 100$) – размеры стола и радиус отпиленной части соответственно.

Формат выходных данных

В ответе выведите одно целое число от 0 до 100, которое означает, сколько процентов составляет вероятность того, что тарелка, поставленная на случайное место стола, не выпирает за границы стола.

Пример

стандартный ввод	стандартный вывод
10 10 5	48

Замечание

Красным на рисунке показаны выпирающие тарелки.

Синим на рисунке показана тарелка, полностью лежащая внутри стола.

Разбор

Площадь стола можно найти по формуле $N \times M - 2 \times \frac{\pi R^2}{4}$ (ведь мы из прямоугольника площади $N \times M$ вырезали две четвертинки окружностей, площадь каждой из которых равна $\frac{\pi R^2}{4}$. Несложно понять, что поставленная тарелка не будет падать со стола тогда и только тогда, когда её центр находится строго внутри стола. Стоящая на столе тарелка будет вылезать за границы стола, если расстояние от её центра до какой-либо граничной точки стола строго меньше 1. Это означает, что либо центр тарелки находится на расстоянии строго меньше $R+1$ от одного из двух углов стола, где сделан вырез радиуса R, либо центр тарелки находится на расстоянии строго меньше 1 от стороны прямоугольника, которым образован стол (до того, как из него вырезали четвертинки окружностей). Для произвольной известной точки мы можем за $O(1)$ проверить выполнение одного из этих двух условий. Для решения задачи можно было воспользоваться методом Монте-Карло: сгенерировать случайно равновероятно множество точек внутри прямоугольника и для каждой точки проверить (с помощью вышеописанных рассуждений), будет ли тарелка с центром в этой точке лежать полностью внутри стола или нет. Чтобы получить ответ, найдём отношение количества "хороших точек"(тарелки с центрами в которых будут лежать внутри стола) к количеству точек, которые мы проверяли; округлим это отношение до сотых и умножим на 100 — это и будет итоговый ответ.

Известно, что для получения абсолютной погрешности ϵ нужно проверить $O(\frac{1}{\epsilon})$ точек. В авторском решении точки генерируются с отсечением по времени (то есть в течение примерно одной секунды), что позволяет получить такую малую погрешность, которой достаточно для нахождения правильного ответа. Отметим, что ответ можно было получить и математическим способом с помощью интегрирования. Оставим эту задачу заинтересовавшемуся читателю в качестве математического упражнения.

Авторское решение на языке C++:

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

typedef long long ll;
using namespace std;

int main(int argc, char* argv[]) {
    int N, M, R;
    cin >> N >> M >> R;

    mt19937_64 mt(229);
    uniform_real_distribution<long double> dist_x = uniform_real_distribution((long double)1, (lon
    uniform_real_distribution<long double> dist_y = uniform_real_distribution((long double)1, (lon
    auto check = [&] () -> bool {
        // генерим рандомную точку [1, N - 1], [1, M - 1]
        long double x = dist_x(mt);
        long double y = dist_y(mt);

        if (x * x + y * y <= (R + 1) * (R + 1)) {
            return false;
        }

        if ((N - x) * (N - x) + (M - y) * (M - y) <= (R + 1) * (R + 1)) {
            return false;
        }

        return true;
    };

    ll good = 0, all = 0;
    while (1000 * clock() / CLOCKS_PER_SEC < 900) {
        ++all;
        good += check();
    }

    long double PI = acosl(-1);
    long double S = N * M - PI * R * R / 2;
    long double S_small = (N - 2) * (M - 2);
    long double S_need = S_small * good / all;

#define debug(x) cerr << #x << ": " << (x) << endl;

    debug(S);
    debug(S_small);
    debug(S_need);
```

```
    debug(good);
    debug(all);
    debug(S_need / (N * M));

    long double ANSWER = S_need * 100 / S;

    cerr << "ANSWER: " << fixed << setprecision(10) << ANSWER << endl;
    cout << round(ANSWER);

    return 0;
}
```