МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра дискретной математики и алгоритмики

ИСПОЛЬЗОВАНИЕ АЛГОРИТМА ЕВКЛИДА В ЗАДАЧАХ С ЦЕЛОЧИСЛЕННЫМИ ПАРАМЕТРАМИ

Курсовая работа

Якубовского Владислава
Александровича
студента 3 курса,
специальность «информатика»
Научный руководитель:
старший преподаватель кафедры ДМА
Ким Д.В.

Минск, 2023

ОГЛАВЛЕНИЕ

Введение **3**

1 Алгоритм Евклида **4**
1.1 Алгоритм Евклида 4

2 Задача дискретной оптимизации **6**
2.1 Постановка задачи 6
2.2 Необходимые определения 6
2.3 Решение 7
2.4 Реализация 10

3 Количество целых точек под и на прямой **14**
3.1 Постановка задачи 14
3.2 Количество целых точек на прямой 14
3.3 Количество целых точек строго под прямой 15
3.4 Решение при L = 0 15
3.5 Реализация при L = 0 19
3.6 Решение исходной задачи 20
3.7 Реализация 20

Заключение **22**
Приложение А. Алгоритм Евклида 23

ВВЕДЕНИЕ

Алгоритм Евклида - эффективный алгоритм для нахождения наибольшего общего делителя двух целых чисел, описанный Евклидом в книге «Начала» (около 300 г. до н.э.). На этом алгоритме строятся эффективные решения большого количества задач с целочисленными параметрами. Примерами таких задач могут быть нахождение решений линейных Диофантовых уравнений, нахождение обратного числа по простому модулю и т.д. В этой работе рассматриваются нетривиальные применения алгоритма Евклида для разработки алгоритмов решения задач.

ГЛАВА 1
АЛГОРИТМ ЕВКЛИДА

1.1 Алгоритм Евклида

Определение. Наибольшим общим делителем 2 натуральных чисел a и b называется наибольшее положительное число, которое является делителем a и b одновременно, то есть:

$$\gcd(a, b) = \max_{k=1...\infty \,:\; k \mid a \text{ и } k \mid b} k \tag{1.1}$$

Для удобства положим:

$$\gcd(0, 0) = 0 \tag{1.2}$$
$$\gcd(0, a) = a, a \neq 0$$

так как 0 не бывает делителем.

Формулировка. Даны два целых неотрицательных числа a и b. Требуется найти их наибольший общий делитель ($\gcd(a, b)$). Утверждается, что:

$$\gcd(a, b) = \begin{cases} a & b = 0, \\ \gcd(b, a \bmod b) & \text{иначе.} \end{cases} \tag{1.3}$$

Доказательство корректности. Сначала заметим, что при каждой итерации алгоритма Евклида его второй аргумент строго убывает, следовательно, поскольку он неотрицательный, то алгоритм Евклида всегда завершается.

Для доказательства корректности нам необходимо показать, что:

$$\gcd(a, b) = \gcd(b, a \bmod b), \forall a \geqslant 0, b > 0 \tag{1.4}$$

Покажем, что величина, стоящая в левой части равенства, делится на стоящую в правой, а стоящая в правой — делится на стоящую в левой. Очевидно, это будет означать, что левая и правая части совпадают, что и докажет корректность алгоритма Евклида.

Обозначим $d = \gcd(a, b)$. Тогда, по определению, $d|a$ и $d|b$.

$$a \bmod b = a - b \left\lfloor \frac{a}{b} \right\rfloor \tag{1.5}$$

Но тогда отсюда следует, что $d \mid (a \bmod b)$.

Итак, вспоминая утверждение $d \mid b$, получаем систему:

$$\begin{cases} d \mid b, \\ d \mid (a \bmod b) \end{cases} \tag{1.6}$$

Воспользуемся теперь следующим простым фактом: если для каких-то трёх чисел p, q, r выполнено: $p \mid q$ и $p \mid r$, то выполняется и: $p \mid \gcd(q, r)$. В нашей ситуации получаем:

$$d \mid \gcd(b, a \bmod b) \tag{1.7}$$

Или, подставляя вместо d его определение как $\gcd(a, b)$, получаем:

$$\gcd(a, b) \mid \gcd(b, a \bmod b) \tag{1.8}$$

Итак, мы провели половину доказательства: показали, что левая часть делит правую. Вторая половина доказательства производится аналогично [1].

Т.к. a и b убывают, то и $(a \bmod b)$ тоже убывает. Из этого следует, что алгоритм придет в ситуацию $(a^* \bmod b^*) = 0$, то есть b^* делит a^*, и следующим шагом вернет b^*, а в силу эквивалентности переходов получаем:

$$\gcd(a, b) \ = \gcd(a^*, b^*) \ = \gcd(b^*, a^* \bmod b^*) = b^* \tag{1.9}$$

■

Оценим временную сложность алгоритма Евклида. Для этого покажем, что:

$$a \bmod b \leqslant \frac{a}{2}, \forall a \geqslant b \tag{1.10}$$

Рассмотрим 2 случая:
1) $b \leqslant \frac{a}{2}$, тогда очевидно (1.10) выполняется.
2) $b > \frac{a}{2}$, тогда a представим в виде:

$$a = b + (a \bmod b) \tag{1.11}$$

Из этого получаем:

$$a \bmod b = a - b \leqslant a - \frac{a}{2} = \frac{a}{2} \tag{1.12}$$

Что и требовалось показать.

Из (1.10) следует, что наибольшее число в алгоритме Евклида с каждым шагом уменьшается минимум в 2 раза. Тогда, в силу того, что на каждый переход требуется $\mathcal{O}(1)$ операций, временная сложность алгоритма Евклида $\mathcal{O}\left(\log \max\{a, b\}\right)$.

ГЛАВА 2
ЗАДАЧА ДИСКРЕТНОЙ ОПТИМИЗАЦИИ

2.1 Постановка задачи

В связи с использованием модульной арифметики при решении частных комбинаторных задач часто встречается, что дроби $\frac{a}{b}$ закодированы следующим образом:

1) Заранее фиксируется простой модуль p, одинаковый для всех дробей и преобразований в рамках задачи.

2) Находят обратное число к b по модулю p, то есть такое число b^{-1}, что $bb^{-1} = 1 \pmod p$.

3) Находят такое число $q < p$, что $ab^{-1}(\text{mod p}) = q$ и говорят, что дробь $\frac{a}{b}$ по модулю p имеет остаток q.

Задача.

Есть дробь $\dfrac{a}{b}$, которая по простому модулю p имеет остаток $q < p$, то есть:

$$a = (bq) \bmod p \tag{2.1}$$

Заданы p и q, найти такие целые $a \geqslant 0$, $b > 0$ удовлетворяющие условию (2.1), что сумма $a + b$ — минимальная.

2.2 Необходимые определения

Определение 1. Функция, которая в соответствие каждому действительному числу x ставит наименьшее целое число y, не меньшее чем x, называется $ceil$:

$$ceil(x) - 1 < x \leqslant ceil(x) \tag{2.2}$$
$$ceil(x) \in \mathbb{Z}$$

Определение 2. Функция, которая в соответствие каждому действительному числу x ставит наименьшее целое число y, строго большее чем x, называется $strongCeil$:

$$strongCeil(x) - 1 \leqslant x < strongCeil(x) \tag{2.3}$$
$$strongCeil(x) \in \mathbb{Z}$$

Определение 3. Функция, которая в соответствие каждому действительному числу x ставит наибольшее целое число y, не большее чем x, называется $floor$:

$$floor(x) \leqslant x < floor(x) + 1 \tag{2.4}$$
$$floor(x) \in \mathbb{Z}$$

Определение 4. Функция, которая в соответствие каждому действительному числу x ставит наибольшее целое число y, строго меньшее чем x, называется $strongFloor$:

$$strongFloor(x) < x \leqslant strongFloor(x) + 1 \tag{2.5}$$
$$strongFloor(x) \in \mathbb{Z}$$

2.3 Решение

Преобразуем (2.1):

$$a = (bq) \bmod p = bq - floor\left(\frac{bq}{p}\right) p \tag{2.6}$$

Получается, что теперь нужно минимизировать по b функцию

$$a + b = (q+1)b - floor\left(\frac{bq}{p}\right) p \tag{2.7}$$

Вообще говоря, задачи нахождения минимума/максимума, где целевая функция имеет вид:

$$f(x) = Ax + Bg\left(\frac{P}{Q}x\right) \tag{2.8}$$
$$x \in \mathbb{Z} \cap [l, r], A, B \in \mathbb{R},$$
$$P, Q \in \mathbb{N}, l, r \in \mathbb{N}^0,$$
$$g \text{ — } floor, strongFloor, ceil \text{ или } strongCeil$$

можно решить за $\mathcal{O}(\log \max\{P, Q\})$

Опишем такое рекурсивное решение и оценим его время работы:

Если $l = r$ вернем l.

а) $g = floor$, тогда имеем:

$$f(x) = Ax + B \cdot floor\left(\frac{P}{Q}x\right) \tag{2.9}$$

Если P делится на Q — посчитаем значение $floor$, получим:

$$f(x) = Ax + B\frac{P}{Q}x = A'x \tag{2.10}$$

Тогда просто возвращаем решение: максимальный или минимальный x, в зависимости от знака A' и задачи, которую мы решаем(минимум или максимум).

Пусть P не делится на Q, тогда возьмем $k = floor\left(\frac{P}{Q}\right)$, и вынесем kx из-под $floor$:

$$Ax + B \cdot floor\left(\frac{P}{Q}x\right) = (A + Bk)x + B \cdot floor\left((\frac{P}{Q} - k)x\right) =$$

$$= A^{'}x + B \cdot floor\left(\frac{P^{'}}{Q}x\right) \tag{2.11}$$

Положим $y = floor\left(\frac{P^{'}}{Q}x\right)$. Поменяем порядок суммирования, для этого попробуем выразить иксы через игреки. Из (2.4) получается:

$$y \leqslant \frac{P^{'}}{Q}x < y + 1 \Rightarrow y\frac{Q}{P^{'}} \leqslant x < (y + 1)\frac{Q}{P^{'}} \tag{2.12}$$

Теперь для каждого игрека выберем оптимальный икс. Опять же это зависит от задачи(минимум или максимум) и от знака $A^{'}$. Пусть мы решаем задачу на минимум:

1) $A^{'} < 0$ — нужно взять наибольший икс, в соответствие которому ставится этот игрек — $x = strongFloor\left((y + 1)\frac{Q}{P^{'}}\right)$. Получается новая задача на минимум вида (2.8):

$$By + A^{'} \cdot strongFloor\left(\frac{Q}{P^{'}}(y + 1)\right) \tag{2.13}$$

$$l_{new} = floor\left(\frac{P^{'}}{Q}l\right) + 1, r_{new} = floor\left(\frac{P^{'}}{Q}r\right) + 1,$$

$$A_{new} = B, B_{new} = A^{'} = A + Bk,$$

$$P_{new} = Q, Q_{new} = P^{'} = P \bmod Q, g_{new} = ceil$$

Замечание. $y+1$ нам не мешает: в этом случае можно, например, «сдвинуть» все индексы игреков влево.

Здесь важно заметить, что в соответствие самому большому игреку мы хотели бы поставить:

$$x = strongFloor\left((y_{max} + 1)\frac{Q}{P^{'}}\right) \tag{2.14}$$

и чаще всего у нас удается это сделать, кроме случаев, когда оказыается, что значение в (2.14) больше, чем r. В таком случае нужно для этого игрека взять $x = r$.

2) $A^{'} \geqslant 0$ — нужно взять наименьший икс, в соответствие которому ставится этот игрек $x = ceil\left(\frac{Q}{P^{'}}y\right)$. Получается новая задача на минимум вида (2.8):

$$By + A^{'} \cdot ceil\left(\frac{Q}{P^{'}}y\right) \tag{2.15}$$

$$l_{new} = floor(\frac{P^{'}}{Q}l), r_{new} = floor(\frac{P^{'}}{Q}r),$$

$$A_{new} = B, B_{new} = A^{'} = A + Bk,$$

$$P_{new} = Q, Q_{new} = P^{'} = P \bmod Q, g_{new} = ceil$$

Здесь важно заметить, что в соответствие самому маленькому игреку мы хотели бы поставить:

$$x = ceil\left(\frac{Q}{P'}y_{min}\right) \tag{2.16}$$

и чаще всего у нас удается это сделать, кроме случаев, когда оказыается, что значение в (2.16) меньше, чем l. В таком случае нужно для этого игрека взять $x = l$. Дальше подобные замечания будут опускаться.

Для задачи на максимум все показывается точно так же.

б) $g = ceil$, тогда вместо (2.20) получаем:

$$y - 1 < \frac{P^{'}}{Q}x \leqslant y \Rightarrow (y-1)\frac{Q}{P'} < x \leqslant y\frac{Q}{P'} \tag{2.17}$$

Пусть задача на максимум:

1) $A^{'} < 0$ — нужно взять наименьший икс, в соответствие которому ставится этот игрек — $x = strongCeil\left((y-1)\frac{Q}{P'}\right)$. Получается новая задача на максимум вида (2.8):

$$By + A^{'} \cdot strongCeil\left(\frac{Q}{P'}(y-1)\right) \tag{2.18}$$

$$l_{new} = ceil\left(\frac{P^{'}}{Q}l\right), r_{new} = ceil\left(\frac{P^{'}}{Q}r\right),$$

$$A_{new} = B, B_{new} = A^{'},$$

$$P_{new} = Q, Q_{new} = P^{'} = P \bmod Q, g_{new} = strongCeil$$

2) $A \geqslant 0$ — нужно взять наибольший икс, в соответствие которому ставится этот игрек — $x = floor\left(\frac{Q}{P'}y\right)$. Получается новая задача на максимум вида (2.8):

$$By + A^{'} \cdot floor\left(\frac{Q}{P'}y\right) \tag{2.19}$$

$$l_{new} = ceil\left(\frac{P^{'}}{Q}l\right), r_{new} = ceil\left(\frac{P^{'}}{Q}r\right),$$

$$A_{new} = B, B_{new} = A^{'},$$

$$P_{new} = Q, Q_{new} = P^{'} = P \bmod Q, g_{new} = floor$$

в) g = strongCeil, вместо (2.20), получим:

$$y - 1 \leqslant \frac{P'}{Q}x < y \Rightarrow (y-1)\frac{Q}{P'} \leqslant x < y\frac{Q}{P'} \tag{2.20}$$

Аналогично в зависимости от задачи и от знака A нужно положить

$$x = ceil\left((y-1)\frac{Q}{P'}\right) \tag{2.21}$$

или

$$x = strongFloor\left(y\frac{Q}{P'}\right) \tag{2.22}$$

г) $g = strongFloor$, вместо (2.20), получим:

$$y < \frac{P'}{Q}x \leqslant y + 1 \Rightarrow y\frac{Q}{P'} < x \leqslant (y+1)\frac{Q}{P'} \tag{2.23}$$

Точно так же в зависимости от задачи и от знака A нужно положить

$$x = strongCeil\left(y\frac{Q}{P'}\right) \tag{2.24}$$

или

$$x = floor\left((y+1)\frac{Q}{P'}\right) \tag{2.25}$$

Заметим, что с каждым шагом между P и Q происходят те же действия, что и происходили бы в алгоритме Евклида ($Q_{new} = P \bmod Q, P_{new} = Q$). Из этого следует, что за $\mathcal{O}\left(\log\left(\max\left(P, Q\right)\right)\right)$ шагов(если алгоритм еще не завершится на проверке $l = r$), в аргументе функции g будет целое число, и наша функция сведется к функции, подобной (2.10), после чего завершится.

Из этого следует, что сложность описанного выше алгоритма — $\mathcal{O}\left(\log(max(P, Q))\right)$.

Возвращаясь к (2.7), положив в алгоритм, описанный выше

$$l = 1, r = p - 1, A = q + 1, B = p, P = q, Q = p, g = floor$$

сначала получим оптимальное b, потом, подставив b в (2.6), получим a.

2.4 Реализация

Реализация алгоритма состоит из двух частей:
- построение по p и q функции (2.7);
- нахождение ее точки минимума, это и будет наше b;
- определение значения a по полученному b.

Первый и третий шаги происходят непосредственно в функции ниже, которая возвращает пару, первый элемент которой — a, второй — b:

```
std::pair<int64_t, int64_t> MinAPlusB(int64_t p, int64_t q) {
    assert(q < p);
    if (q == 0) {
        return {0, 1};
    }
    int64_t b = FMinVal(q + 1, -p, q, p, 1, p-1, RoundingFunction::kFloor);
    return {b * q % p, b};
}
```

Листинг 2.1 — Алгоритм нахождения таких a и b

Алгоритм выше использует функцию, находящую точку минимума функции вида (2.8), ниже часть ее реализации, где $typeToDivFunc[g](x, y) = g(\frac{x}{y})$:

```
int64_t FMinVal(int64_t A, int64_t B, int64_t P, int64_t Q,
                int64_t l, int64_t r, RoundingFunction g) {
    if (l > r) {
        return bad_pos;
    }

    if (l == r) {
        return l;
    }

    if (B == 0) {
        if (A > 0) {
            return l;
        }

        return r;
    }

    if (P % Q == 0) {
        const int64_t l_result = A * l + B * typeToDivFunc[g](P * l, Q),
                      r_result = A * r + B * typeToDivFunc[g](P * r, Q);

        if (l_result < r_result) {
            return l;
        }

        return r;
    }

    const int64_t k = P / Q;

    const int64_t A_new = B, B_new = A + B * k,
                  Q_new = P % Q, P_new = Q;
    int64_t l_new = typeToDivFunc[g](Q_new * l, Q),
            r_new = typeToDivFunc[g](Q_new * r, Q);

    const int64_t l_val = B_new * l + B * l_new, r_val = B_new * r + B * r_new;
    int64_t m_val = 1e18;
    int64_t mid;

    switch (g) {
        case RoundingFunction::kFloor: {
            if (B_new < 0) {
                {
```

```
                    int64_t first = math::StrongFloorQuotient(
                            P_new * (l_new + 1),Q_new),
                            second = math::StrongFloorQuotient(
                                    P_new * (r_new + 1),Q_new);
                    if (first < l) {
                        l_new++;
                    }
                    if (second > r) {
                        r_new--;
                    }
                }
                auto pos = FMinVal(A_new, B_new, P_new, Q_new,
                                   l_new + 1, r_new + 1,
                                   RoundingFunction::kStrongFloor);
                if (pos != bad_pos) {
                    mid = math::StrongFloorQuotient(pos * P_new, Q_new);
                    m_val = B_new * mid + B * typeToDivFunc[g](Q_new * mid, Q);
                }
            } else {
                {
                    int64_t first = math::CeilQuotient(
                            P_new * l_new, Q_new),
                            second = math::CeilQuotient(
                                    P_new * r_new, Q_new);
                    if (first < l) {
                        l_new++;
                    }
                    if (second > r) {
                        r_new--;
                    }
                }
                auto pos = FMinVal(A_new, B_new, P_new, Q_new,
                                   l_new, r_new,
                                   RoundingFunction::kCeil);
                if (pos != bad_pos) {
                    mid = math::CeilQuotient(pos * P_new, Q_new);
                    m_val = B_new * mid + B * typeToDivFunc[g](Q_new * mid, Q);
                }
            }
        }
        break;
        case RoundingFunction::kCeil: {
           /* ... */
        }
        break;
        case RoundingFunction::kStrongCeil: {
            /* ... */
        }
        break;
        case RoundingFunction::kStrongFloor: {
            /* ... */
        }
    }

    auto min = std::min(l_val, std::min(m_val, r_val));
    if (l_val == min) {
        return l;
    }
    if (m_val == min) {
        return mid;
```

```
105     }
106
107     return r;
108 }
```

Листинг 2.2 — Алгоритм нахождения точки минимума

ГЛАВА 3
КОЛИЧЕСТВО ЦЕЛЫХ ТОЧЕК ПОД И НА ПРЯМОЙ

3.1 Постановка задачи

Дана прямая $y = \frac{P}{Q}x$ на плоскости, $P \in \mathbb{N}^0, Q \in \mathbb{N}$.

Сколько целых точек плоскости с $L \leqslant x \leqslant R$, где $L, R \in \mathbb{Z}$ суммарно находится на и под/над этой прямой?

То есть количество различных точек $(x, y) \in \mathbb{Z}^2$, у которых:

$$L \leqslant x \leqslant R$$

$$\begin{cases} (0 \leqslant y \leqslant \dfrac{P}{Q}x), & x \geqslant 0 \\ (\dfrac{P}{Q}x \leqslant y \leqslant 0), & x < 0 \end{cases}$$

3.2 Количество целых точек на прямой

Для начала научимся считать количество целых точек на прямой $y = \frac{P}{Q}x$, $0 \leqslant x \leqslant R$. Это сделать несложно:

1) Если $P = 0$, то таких точек ровно $R + 1$.

2) Пусть $P \neq 0$, сократим P и Q на $\gcd(P, Q)$:

$$P^* = \frac{P}{\gcd(P,Q)}, Q^* = \frac{Q}{\gcd(P,Q)} \tag{3.1}$$

Чтобы в (3.1) y был целым при каком-то целом x, необходимо и достаточно, чтобы $x = 0$ или:

$$\gcd(P^* \cdot x, Q^*) = Q^* \tag{3.2}$$

Т.к. $\gcd(P^*, Q^*) = 1$, то:

$$\gcd(x, Q^*) = Q^* \tag{3.3}$$

Отсюда получаем:

$$x = kQ^*, k \in \mathbb{N}^0 \tag{3.4}$$

Тогда:

$$k_{max} = \left\lfloor \frac{R}{Q^*} \right\rfloor = \left\lfloor \frac{R \cdot \gcd(P,Q)}{Q} \right\rfloor \tag{3.5}$$

Тогда количество целых точек на прямой это:

$$1 + \left\lfloor \frac{R \cdot \gcd(P,Q)}{Q} \right\rfloor \tag{3.6}$$

3.3 Количество целых точек строго под прямой

Дальше научимся считать количество целых точек, находящихся строго под прямой $y = kx, k \in \mathbb{N}^0, 0 \leqslant x \leqslant R$.

Переберем иксы:

0) при $x = 0$ число точек, лежащих строго под прямой, равно 0.

1) при $x = 1$ нам подходят все целые точки $(1, y)$, в которых $0 \leqslant y < k \cdot 1$. Таких точек ровно k штук.

...

j) при $x = j$ нам подходят все целые точки (j, y), в которых $0 \leqslant y < k \cdot j$. Таких точек ровно $k \cdot j$ штук.

...

r) при $x = R$ нам подходят все целые точки (R, y), в которых $0 \leqslant y < k \cdot R$. Таких точек ровно $r \cdot j$ штук.

Получается сумма:

$$\sum_{i=0}^{R} ki = \frac{kR(R+1)}{2} \tag{3.7}$$

3.4 Решение при L = 0

Имеем

$$y = \frac{P}{Q}x, P \in \mathbb{N}^0, Q \in \mathbb{N}, 0 \leqslant x \leqslant R, R \in \mathbb{Z} \tag{3.8}$$

Проведем прямую

$$y = \frac{(P - P \bmod Q)}{Q}x \tag{3.9}$$

Посчитаем число целых точек, лежащих строго под этой прямой. Т.к. $\frac{(P-P \bmod Q)}{Q} \in \mathbb{N}^0$, это число будет в точности (3.7), где

$$k = \frac{(P - P \bmod Q)}{Q} \tag{3.10}$$

Нам осталось посчитать количество точек на исходной прямой,на проведенной прямой (3.9), и между двумя этими прямыми.

Дальше нужно «передвинуть» все точки (x, y) на вектор $(0, -\frac{(P-P \bmod Q)}{Q}x)$:

$$y^* + \frac{(P - P \bmod Q)}{Q}x^* = \frac{P}{Q}x^* \tag{3.11}$$

Перенесем икс вправо и раскроем скобки:

$$y^* = \frac{P \bmod Q}{Q}x^* \tag{3.12}$$

Заметим, что после преобразования целые точки с прямой (3.9) перейдут на прямую $y = 0$, а все целые точки с прямой (3.8) перейдут на прямую (3.12). Соответственно целые точки между этими прямыми, перейдут на плоскость, находящуюся выше чем $y = 0$, и ниже, чем $y = \frac{(P - P \bmod Q)}{Q} x$.

Преобразование имеет такой вид: мы от y отнимаем какое-то целое число, поэтому все целые точки до преобразования, перейдут в другие целые точки, а все точки с нецелыми игреками перейдут в другие точки с нецелыми игреками.

Так же стоит отметить, что все точки, которые мы уже посчитали в (3.7)-(3.10) перейдут строго под прямую $y = 0$.

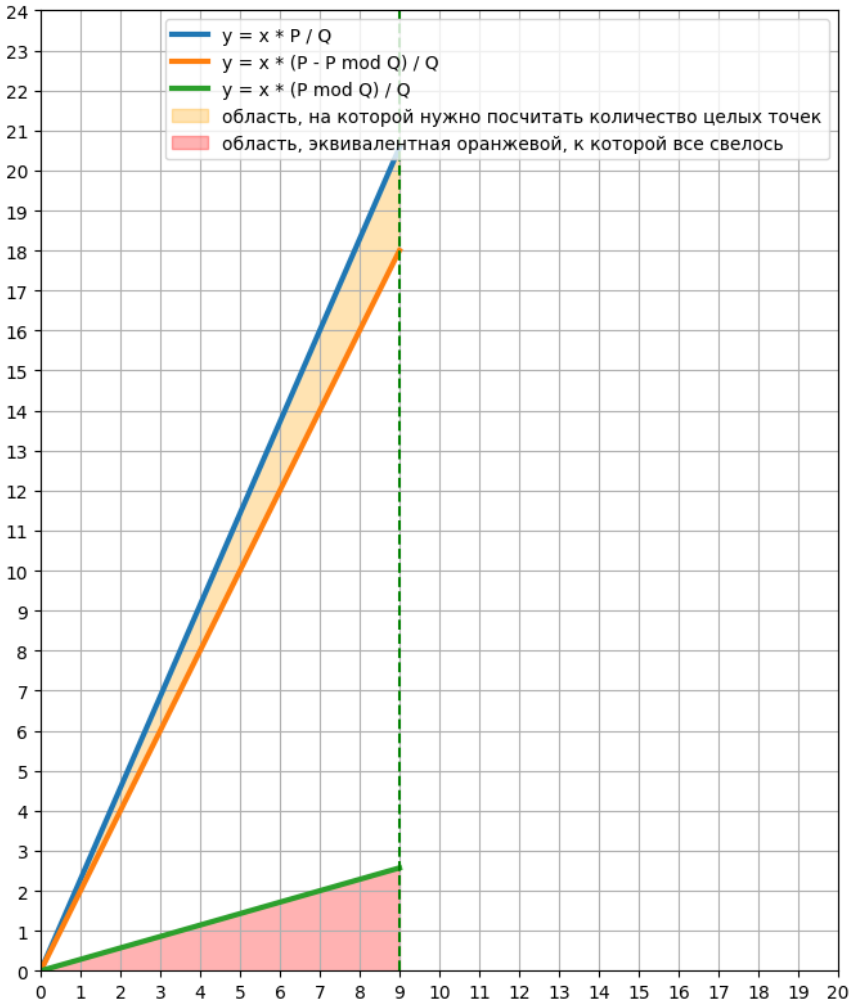

Рисунок 3.1 — Пример такого преобразования для $P = 16$, $Q = 7$, $R = 9$

Получается, нам осталось решить такую же задачу, но с функцией (3.12). Проведем прямую:

$$y = \frac{Q}{P \bmod Q} x, \tag{3.13}$$

$$0 \leqslant x \leqslant \left\lfloor \frac{P \bmod Q}{Q} r \right\rfloor$$

Посмотрим, что можно с ней сделать:

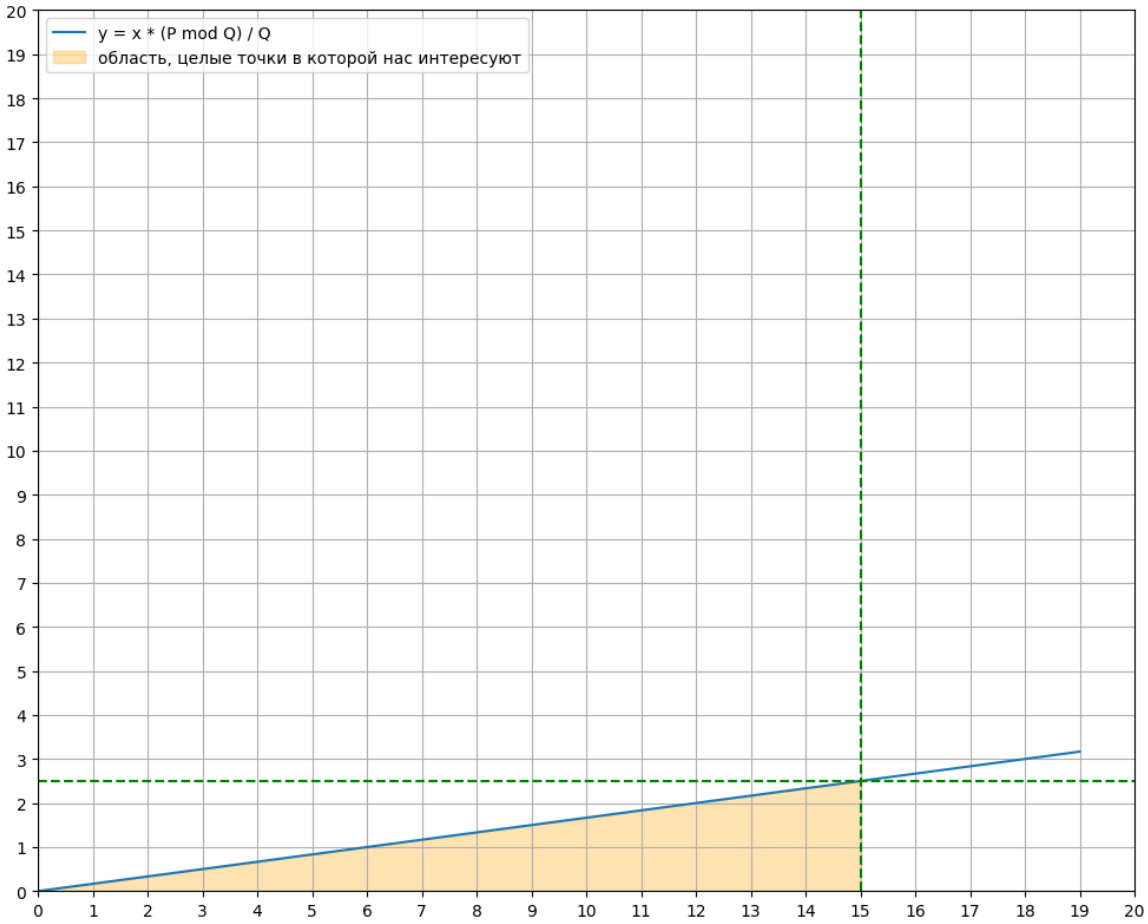

Рисунок 3.2 — Пример для $P = 1, Q = 6, R = 15$

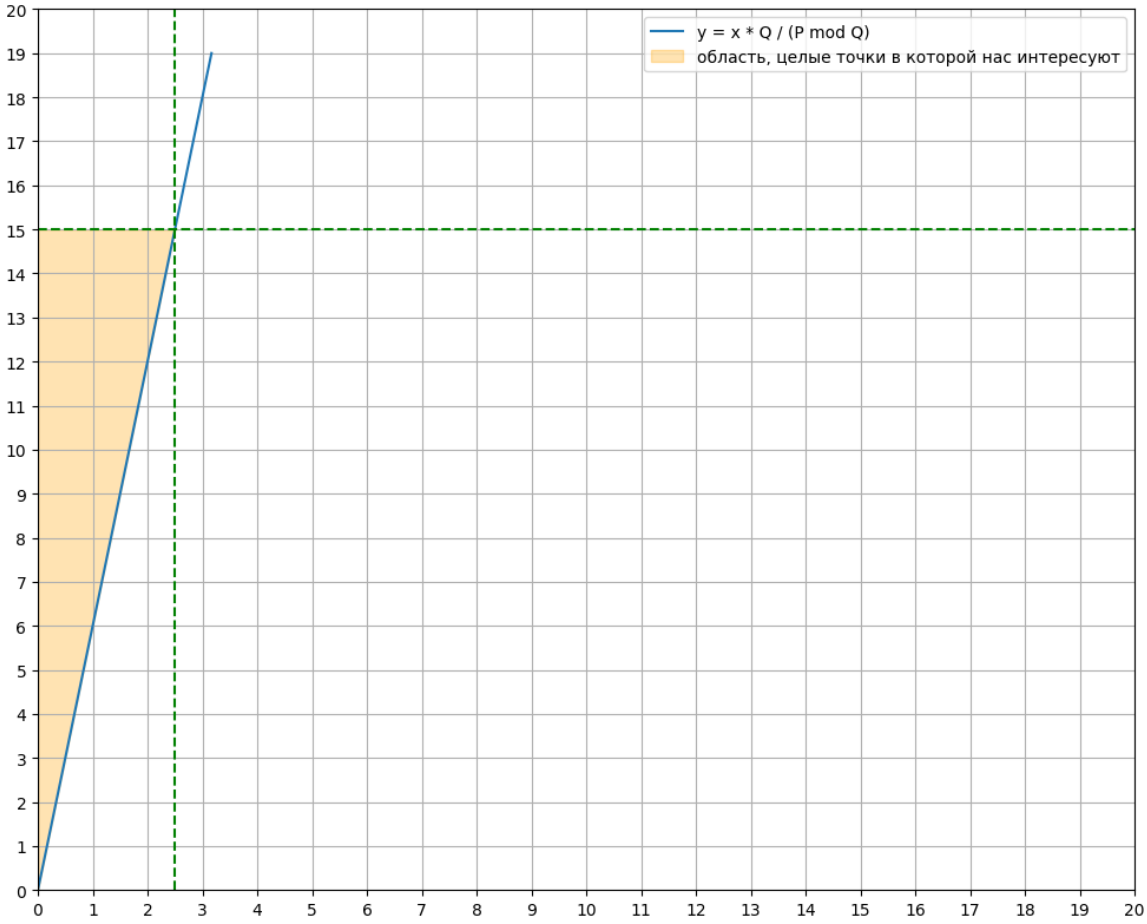

Рисунок 3.3 — Пример для $P = 1, Q = 6, R = \frac{15}{6}$

Так как такое преобразование — просто отражение прямой относительно $y = x$, это можно решить следующим образом:

1) Посчитать количество целых точек внутри такого прямоугольника:

$$\begin{cases} 0 \leqslant x \leqslant \lfloor \frac{P \bmod Q}{Q} R \rfloor, \\ 0 \leqslant y \leqslant rR \end{cases} \tag{3.14}$$

2) Рекурсивно посчитать суммарное количество точек на и под прямой (3.13)

3) Вычесть значение, посчитанное в шаге 2, из количества всех целых точек прямоугольника.

4) Так как мы в 3 шаге вычли и целые точки на прямой, которая входит в интересующее нас множество, нужно их вернуть. Просто прибавить количество целых точек на прямой, как в (3.6).

Как и писалось выше, мы получаем новую такую задачу, но с другой функцией:

$$y = \frac{P_{new}}{Q_{new}} x \tag{3.15}$$

$$P_{new} = Q, Q_{new} = P \bmod Q$$

Отношение точно такое же, как и в алгоритме Евклида. Из этого следует, что за $\mathcal{O}(\log(\max(P, Q)))$ вызовов функция сведется к $y = 0$, после чего рекурсия завершится. Для (3.6) можно посчитать gcd один раз. Дальше, из-за соотношения в (3.15), каждый раз этот gcd будет одинаковым.

Из этого следует, что итоговая временная сложность алгоритма — $\mathcal{O}(\log \max\{P, Q\})$

3.5 Реализация при L = 0

Функция, вычисляющая количество целых точек строго под прямой $y = kx, 0 \leqslant x \leqslant r$:

```
uint64_t GetUnderLine(int64_t k, int64_t r) {
    uint64_t points_under_line = static_cast<uint64_t >(r) * (r + 1) * k / 2;
    return points_under_line;
}
```

Листинг 3.1 — Количество целых точек строго под прямой

Функция, вычисляющая количество целых точек на прямой $y = \frac{xQ}{P}, 0 \leqslant x \leqslant r$:

```
uint64_t GetOnLine(int64_t P, int64_t Q, int64_t r, int64_t gcd) {
    if (P == 0) {
        return static_cast<uint64_t>(r) + 1;
    } else {
        return 1 + std::floor(r * gcd / Q);
    }
}
```

Листинг 3.2 — Количество целых точек на прямой

Итоговая функция, вычисляющая количество целых точек на и под прямой $y = \frac{xQ}{P}, 0 \leqslant x \leqslant r$:

```
uint64_t GetOnAndUnderLine(int64_t P, int64_t Q, int64_t r,
                           std::optional<int64_t> gcd = std::nullopt) {
    if (gcd == std::nullopt) {
        gcd = GCD(P, Q);
    }
    uint64_t points_on_line = GetOnLine(P, Q, r, *gcd);
    if (P % Q == 0) {
        int64_t k = P / Q;
        uint64_t points_under_line = GetUnderLine(k, r);
        return points_on_line + points_under_line;
    }

    uint64_t points_1_part = GetUnderLine((P - P % Q) / Q, r);

    int64_t new_r = std::floor(static_cast<double>(P % Q) * r / Q);
    uint64_t points_inverted = GetOnAndUnderLine(Q, P % Q,new_r, gcd);
    uint64_t area = static_cast<uint64_t>(new_r + 1) * (r + 1);

    uint64_t points_2_part = area - points_inverted;

```

```
    return points_1_part + points_2_part + points_on_line;
}
```

Листинг 3.3 — Количество целых точек на и под прямой

3.6 Решение исходной задачи

Имеем

$$y = \frac{P}{Q}x, \tag{3.16}$$

$$P \in \mathbb{N}^0, Q \in \mathbb{N}, L \leqslant x \leqslant R, L, R \in \mathbb{Z}$$

Сведем (3.16) к (3.8), для этого рассмотрим все ситуации, связанные с расположением L и R относительно нуля:

1) $L < 0$ и $R = 0$ — так как наша прямая симметрична относительно центра координат, то решение текущей задачи будет эквивалентно решению задачи (3.8) с $R = -L$

2) $L < 0$ и $R > 0$ — задачу можно разбить на 2 подзадачи:

- посчитать количество целых точек при $L \leqslant x \leqslant 0$;
- посчитать количество целых точек при $0 \leqslant x \leqslant R$.

Обе подзадачи мы умеем решать. Осталось решить их по отдельности, сложить результаты, и вычесть количество точек, которе мы посчитали 2 раза. При таком разбиении 2 раза мы считаем только одну целую точку — (0, 0).

3) $L = 0, R >= 0$ — в точности (3.8)

4) $L > 0, R > 0$ — посчитаем количество целых точек при $x \in [0, R]$, и вычтем количество целых точек под и на прямой, у которых $x < l$. Очевидно, что это (3.8) с $x \in [0, L - 1]$.

5) $L < 0, R < 0$ — так как наша прямая симметрична относительно центра координат, очевидно, что данная задача эквивалента такой же задаче с $L = -R$, $R = -L$, а это в точности пункт 4.

3.7 Реализация

```
uint64_t GetOnAndUnderLine(int64_t P, int64_t Q, int64_t l, int64_t r) {
    assert (l <= r);
    if (l < 0 && r < 0) {
        int64_t new_l = -r, new_r = -l;
        return GetOnAndUnderLine(P, Q, new_l, new_r);
    }
    if (l < 0 && r == 0) {
        return GetOnAndUnderLine(P, Q, -l);
    }
    if (l < 0 && r > 0) {
        return GetOnAndUnderLine(P, Q, r) + GetOnAndUnderLine(P, Q, -l) - 1;
    }
    if (l == 0) {
        return GetOnAndUnderLine(P, Q, r);
    }
```

```
    if (l > 0 && r > 0) {
        return GetOnAndUnderLine(P, Q, r) - GetOnAndUnderLine(P, Q, l - 1);
    }
    assert(false);
}
```

Листинг 3.4 — Алгоритм нахождения количества целых точек на и под прямой

ЗАКЛЮЧЕНИЕ

В данной работе был придуман и реализован алгоритм решения задачи (2.1), доказана его корректность и оценена время работы.

Также было придумано решение, доказана его корректность, оценено время работы и реализован алгоритм для задачи про количество целых точек над и на прямой.

Исходные коды решений можно посмотреть по ссылке `https://github.com/deadboysdontcry/course-work`

Алгоритм Евклида

```cpp
#include <iostream>

int gcd (int a, int b) {
    while (b) {
        a %= b;
        swap (a, b);
    }
    return a;
}

int main() {
    int a, b;
    std::cin >> a >> b;
    std::cout << gcd(a, b) << '\n';
    return 0;
}
```

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Алгоритм Евклида. Доказательство коректности и времени работы алгоритма Евклида. Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн - Алгоритмы. Построение и анализ. Издание 3-е, 2013, 978-980 с.