МИНОБРНАУКИ РОССИИ
ФГБОУ ВО «СГУ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

ЦЕПНЫЕ ДРОБИ И КВАДРАТНЫЕ СРАВНЕНИЯ

ЛАБОРАТОРНАЯ РАБОТА

студента 5 курса 531 группы
направления 100501 — Компьютерная безопасность
факультета КНиИТ
Улитина Ивана Владимировича

Проверил
профессор ____________ В. А. Молчанов

Саратов 2023

1 Постановка задачи

Цель работы - изучение основных свойств цепных дробей и квадратных сравнений.

Порядок выполнения работы:

1. Разобрать алгоритм разложения чисел в цепную дробь и привести его программную реализацию;
2. Разобрать алгоритмы приложений цепных дробей и привести их программную реализацию;
3. Разобрать алгоритмы вычисления символов Лежандра и Якоби и привести их программную реализацию;
4. Рассмотреть алгоритмы извлечения квадратного корня в кольце вычетов.

2 Теоретические сведения

2.1 Цепные дроби

Рассмотрим рациональное число r, представленное в виде несократимой дроби $r = \frac{a_0}{a_1}$. Так как НОД$(a_0, a_1) = 1$, то результат вычисления этого наибольшего общего делителя по алгоритму Евклида имеет вид

$$a_0 = a_1 q_1 + a_2, 0 \leq a_2 < a_1,$$

$$a_1 = a_2 q_2 + a_3, 0 \leq a_3 < a_2,$$

$$\ldots$$

$$a_{k-2} = a_{k-1} q_{k-1} + a_k, 0 \leq a_k < a_{k-1},$$

$$a_{k-1} = a_k q_k, \text{где } a_k = \text{НОД}(a_0, a_1) = 1.$$

Эти равенства можно переписать в виде:

$$\frac{a_0}{a_1} = q_1 + \frac{a_2}{a_1}, \frac{a_1}{a_2} = q_2 + \frac{a_3}{a_2}, \ldots, \frac{a_{k-2}}{a_{k-1}} = q_{k-1} + \frac{a_k}{a_{k-1}} = q_{k-1} + \frac{1}{q_k}.$$

Тогда рациональное число r можно представить следующим образом:

$$r = \frac{a_0}{a_1} = q_1 + \frac{a_2}{a_1} = q_1 + \frac{1}{\frac{a_1}{a_2}} = \cdots = q_1 + \cfrac{1}{q_2 + \cfrac{1}{\ddots + \cfrac{1}{q_{k-1} + \cfrac{1}{q_k}}}},$$

где q_1 — целое число и $q_2, \ldots, q_k$ — целые положительные числа.

Определение: Выражение вида

$$q_1 + \cfrac{1}{q_2 + \cfrac{1}{\ddots + \cfrac{1}{q_{k-1} + \cfrac{1}{q_k}}}}$$

принято называть цепной (или непрерывной) дробью с неполными частными $q_1, q_2, \ldots, q_k$ и обозначать символом $(q_1; q_2, \ldots, q_k)$.

Таким образом, любая несократимая рациональная дробь $\frac{a_0}{a_1}$ может быть представлена в виде цепной дроби $\frac{a_0}{a_1} = (q_1; q_2, \ldots, q_k)$ с неполными частными $q_1, q_2, \ldots, q_k$, полученными в результате вычисления по алгоритму Евклида наибольшего общего делителя взаимно простых чисел a_0, a_1.

2.2 Подходящие дроби и их свойства

Для цепной дроби $\frac{a_0}{a_1} = (q_1; q_2, \dots, q_k)$ выражения $\delta_1 = q_1$, $\delta_2 = q_1 + \frac{1}{q_2}, \dots,$ $\delta_k = q_1 + \frac{1}{q_2 + \frac{1}{\ddots + \frac{1}{q_{k-1} + \frac{1}{q_k}}}}$ называются подходящими дробями конечной цепной дроби $(q_1; q_2, \dots, q_k)$ и обозначаются символами $\delta_i = (q_1; q_2, \dots, q_i)$, где $1 \leq i \leq k$.

Аналогично определяются подходящие дроби $\delta_i = (q_1; q_2, \dots, q_i)$ для бесконечной цепной дроби $(q_1; q_2, \dots, q_k, \dots)$.

Индукцией по номеру $i = \overline{1, k}$ можно доказать следующие важные свойства таких подходящих дробей:

1. каждая подходящая дробь δ_i $(i = \overline{1, k})$ является несократимой рациональной дробью $\delta_i = \frac{P_i}{Q_i}$ с числителем P_i и знаменателем Q_i, которые вычисляются по следующим рекуррентным формулам:

$$P_i = q_i P_{i-1} + P_{i-2}, Q_i = q_i Q_{i-1} + Q_{i-2}$$

   с начальными условиями $P_{-1} = 0, P_0 = 1, Q_{-1} = 1, Q_0 = 0$;

2. числители и знаменатели двух последовательных подходящих дробей удовлетворяют равенству $P_i Q_{i-1} - P_{i-1} Q_i = (-1)^i$ для всех $i = \overline{1, k}$;
3. P_i, Q_i взаимно просты;
4. $$\frac{P_i}{Q_i} - \frac{P_{i-1}}{Q_{i-1}} = \frac{(-1)^i}{Q_i Q_{i-1}};$$
5. $$\delta_{2n} > \delta_{2n-1};$$
6. $$P_i Q_{i-2} - P_{i-2} Q_i = q_i \cdot (-1)^{i-1};$$
7. $$\frac{P_i}{Q_i} - \frac{P_{i-2}}{Q_{i-2}} = \frac{q_i \cdot (-1)^{i-1}}{Q_i Q_{i-2}};$$
8. $$\delta_{2n} = \frac{P_{2n}}{Q_{2n}} \text{— убывающая последовательность,}$$

$$\delta_{2n-1} = \frac{P_{2n-1}}{Q_{2n-1}} \text{— возрастающая последовательность,}$$

$$\delta_1 < \delta_3 < \dots < \delta_{2n-1} < \dots \le \frac{a}{b} \le \dots < \delta_{2n} < \dots < \delta_4 < \delta_2;$$

9.
$$\frac{P_n}{Q_n} = q_1 + \sum_{i=1}^{n} \frac{(-1)^i}{Q_i Q_{i-1}} \text{ (сходится по признаку Лейбница);}$$

10.
$$Q_n = q_n \cdot Q_{n-1} + Q_{n-2} \ge Q_{n-1} + Q_{n-2} \ge 2Q_{n-2} \ge 2^{\frac{n-2}{2}}$$

11.
$$|\alpha - \delta_i| \le |\delta_i - \delta_{i-1}| = \frac{1}{Q_i Q_{i-1}}.$$

12. Если все $q_i > 0$, то $\lim_{n\to\infty} \delta_n = \alpha$; в частности, любое число $\alpha \in \mathbf{R}_+$ представляется цепной дробью.

2.3 Приложения цепных дробей

В качестве приложений цепных дробей выделяют:

1. Решение линейных диофантовых уравнений $ax + by = c$.
2. Вычисление обратных элементов в кольце вычетов $\mathbb{Z}_m$.
3. Решение линейных сравнений $ax \equiv b \pmod{m}$.

2.3.1 Диофантовые уравнения

Определение: Диофантовыми уравнениями называются алгебраические уравнения с целочисленными коэффициентами, решение которых отыскивается в целых числах.

Например, диофантовым уравнением является уравнение вида

$$ax - by = 1$$

с целыми неотрицательными коэффициентами a, b. Если коэффициенты a, b удовлетворяют условию НОД$(a, b) = 1$ и $\frac{P_{k-1}}{Q_{k-1}}$ — предпоследняя подходящая дробь представления числа $\frac{a}{b}$ в виде цепной дроби, то из равенств $P_k Q_{k-1} - P_{k-1} Q_k = (-1)^k$, $\frac{a}{b} = \delta_k = \frac{P_k}{Q_k}$ следует, что

$$a(-1)^k Q_{k-1} - b(-1)^k P_{k-1} = 1,$$

т.е. значения $x = (-1)^k Q_{k-1}, y = (-1)^k P_{k-1}$ являются целочисленными решениями уравнения $ax - by = 1$. Легко видеть, что все целые решения исходного диофантова уравнения $ax - by = 1$ находятся по формулам:

$$x = (-1)^k Q_{k-1} + bt, y = (-1)^k P_{k-1} + at,$$

где t — произвольное целое число.

Нетрудно убедиться, что все решения диофантова уравнения $ax - by = c$ с взаимно простыми коэффициентами a, b находятся по формулам:

$$x = (-1)^k c Q_{k-1} + bt, y = (-1)^k c P_{k-1} + at,$$

где t — произвольное целое число.

2.3.2 Вычисление обратных элементов

Определение: Обратным элементом к числу a по модулю m называется такое число b, что $ab \equiv 1 \pmod m$. Обратный элемент обозначают как a^{-1}.

Для нуля обратного элемента не существует никогда, для остальных же элементов обратный элемент может как существовать, так и нет. Утверждается, что обратный элемент существует только для тех элементов a, которые взаимно просты с модулем m.

Для нахождения обратного элемента по модулю можно использовать расширенный алгоритм Евклида. Чтобы показать это, можно рассмотреть следующее уравнение:

$$ax + my = 1.$$

Это уравнение является линейным диофантовым уравнением с двумя переменными. Поскольку единица может делиться только на единицу, то уравнение имеет решение только если НОД$(a, m) = 1$.

Как уже ранее утверждалось, решение можно найти с помощью расширенного алгоритма Евклида. При этом, если мы возьмём от обеих частей уравнения остаток по модулю m, то получим:

$$ax = 1 \pmod m,$$

откуда видно, что найденное x является обратным элементом к a.

2.3.3 Решение линейных сравнений

Сравнение двух целых чисел по модулю натурального числа m — математическая операция, позволяющая ответить на вопрос о том, дают ли два выбранных целых числа при делении на m один и тот же остаток.

Сравнимость чисел a и b по модулю сравнения m записывается как:

$$a \equiv b \pmod{m}.$$

Определение: Выражение

$$a \cdot x \equiv b \pmod{m}$$

называется сравнением первой степени или линейным сравнением по модулю m.

Для проверки существования решений сравнения сначала вычисляется НОД(a, m). Если b не кратно полученному НОД, то у сравнения нет решений. Если кратно, то количество решений по модулю m равно полученному НОД.

Существует несколько алгоритмов нахождения всех решений сравнения, но в рамках рассмотрения приложений цепных дробей применяется алгоритм решения линейных диофантовых уравнений с двумя переменными. В самом деле, сравнение эквивалентно следующему линейному диофантовому уравнению:

$$a \cdot x + m \cdot y = b \pmod{m},$$

исходя из которого можно получить общую формулу решения, после чего выбрать все частные решения в диапазоне от 0 до m.

2.4 Алгоритмы вычисления символов Лежандра и Якоби

Пусть $p > 2$ — простое число.

Определение: Число $a \in \mathbb{Z}_p$ называется квадратичным вычетом по модулю p, если

$$(\exists x \in \mathbb{Z})\; x^2 \equiv a \pmod{p}.$$

В противном случае число a называется квадратичным невычетом по модулю p.

Определение: Для нечетного простого числа p символом Лежандра числа $a \in \mathbb{Z}$ называется выражение

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{если } a \text{ — квадратичный вычет по модулю } p \\ -1 & \text{если } a \text{ — квадратичный невычет по модулю } p \\ 0 & \text{если } a \equiv 0 \pmod{p}. \end{cases}$$

Свойства символа Лежандра:

1. $a \equiv b \pmod{p} \Longrightarrow \left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$,
2. $\left(\frac{ac^2}{p}\right) = \left(\frac{a}{p}\right)$ для любого $c \in \mathbb{Z}$, НОД$(c, p) = 1$.
3. Критерий Эйлера $\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} \pmod{p}$ для НОД$(a, p) = 1$.
4. $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right) \cdot \left(\frac{b}{p}\right)$,
5. $$\left(\frac{1}{p}\right) = 1, \left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}} = \begin{cases} 1 & \text{если } p \equiv 1 \pmod{4}, \\ -1 & \text{если } p \equiv 3 \pmod{4}. \end{cases}$$
6. $$\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}} = \begin{cases} 1 & \text{если } p \equiv \pm 1 \pmod{8}, \\ -1 & \text{если } p \equiv \pm 3 \pmod{8}. \end{cases}$$
7. Квадратичный закон взаимности Гаусса:

$$\left(\frac{p}{q}\right) = \left(\frac{q}{p}\right) \cdot (-1)^{\frac{p-1}{2} \cdot \frac{q-1}{2}}$$

для любых нечетных простых чисел p, q.

Определение:

Пусть натуральное число $n = p_1^{\alpha_1} \cdot \dots \cdot p_k^{\alpha_k}$. Символом Якоби числа $a \in \mathbb{Z}$ называется выражение

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \cdot \dots \cdot \left(\frac{a}{p_k}\right)^{\alpha_k}.$$

Символ Якоби для простого числа n совпадает с символом Лежандра и удовлетворяет почти всем свойствам символа Лежандра (хотя в общем случае символ Якоби не связан с квадратичными вычетами).

Символ Якоби позволяет упростить вычисление символа Лежандра $\left(\frac{a}{p}\right)$ (без разложения числа на множители).

3 Результаты работы

3.1 Описание алгоритма разложения чисел в цепную дробь и алгоритмов приложений цепных дробей

Алгоритм 1 - алгоритм разложения чисел в цепную дробь

Вход: целые числа a, b.

Выход: массив r из чисел $r_1, r_2, \ldots, r_k$ — цепная дробь.

Шаг 1. Создать пустой массив r.

Шаг 2. Взять целую часть от деления a на b и добавить в массив.

Шаг 3. Определить $c = a$. После этого переопределить $a = b$, $b = c\%b$ (то есть остаток от деления c на b).

Шаг 4. Если b не равно нулю, перейти к шагу 2, иначе к шагу 5.

Шаг 5. Результат: массив $r = r_1, \ldots, r_k$, который представляет собой разложение чисел в цепную дробь.

Псевдокод:

```
функция Непрерывная_дробь(a, b):
дробь = пустой_список

пока b не равно 0:
    дробь.добавить(целая_часть(a / b))
    c = a
    a = b
    b = c % b

вернуть дробь
```

Трудоемкость алгоритма $O(\log(\max\{a, b\}))$.

Алгоритм 2 - алгоритм решения линейных диофантовых уравнений

Вход: целые числа a, b, c — коэффициенты уравнения.

Выход: целые числа x, y — решение уравнения вида $ax - by = c$.

Шаг 1. Определить массив r — разложение чисел a, b в цепную дробь с помощью алгоритма 1.

Шаг 2. Определить массивы $p = \{0, 1\}$ и $p = \{1, 0\}$.

Шаг 3. Положить $i = 2$.

Шаг 4. Пусть $c = r_{i-2}$. Добавить в массив p значение $p_{i-1} \cdot c + p_{i-2}$. Добавить в

массив q значение $q_{i-1} \cdot c + q_{i-2}$.
Шаг 5. Пусть k — длина массива r. Если $i < k+2$, то $i = i+1$ и перейти к шагу 4. Иначе перейти к шагу 6.
Шаг 6. Определить $x = q_{k-1} \cdot (-1)^k \cdot c + bt$, $y = p_{k-1} \cdot (-1)^k \cdot c + at$.
Шаг 7. Результат: x, y.

Псевдокод:

```
функция Диофантово_уравнение(a, b, c):
дробь = Непрерывная_дробь(a, b)
p = [0, 1]
q = [1, 0]

для i от 2 до длина(дробь) + 2:
    коэф = дробь[i - 2]
    p.добавить(p[i - 1] * коэф + p[i - 2])
    q.добавить(q[i - 1] * коэф + q[i - 2])

k = длина(дробь)
x = q[k] * степень(-1, k) * c
y = p[k] * степень(-1, k) * c

вернуть (x, y, a, b)
```

Трудоемкость алгоритма $O(k)$, где k — количество элементов в разложении чисел a, b на непрерывную дробь.

Алгоритм 3 - алгоритм вычисления обратных элементов в кольце вычетов

Вход: целые число a и модуль m.
Выход: r — обратный элемент числа a в кольце вычетов $\mathbb{Z}_m$.
Шаг 1. Положить $b_1 = 1$.
Шаг 2. С помощью расширенного алгоритма Евклида от a и m получить значения d, p, q — НОД и коэффициенты перед большим и меньшим числом соответственно.
Шаг 3. Определить $b_2 = \frac{b}{d}$, $n = \frac{m}{d}$, $k = 1$.
Шаг 4. Если остаток от деления $d - 1$ на 2 не равен 0, то $k = k - 1$.
Шаг 5. Определить $r = k \cdot p \cdot b_2 \pmod{n}$.
Шаг 6. Результат: r.

Псевдокод:

```
функция Получить_обратный_элемент(a, m):
b = 1
(_, d, p, q) = Расширенный_алгоритм_Евклида(a, m, 0, 0)
новый_b = b / d
новый_m = m / d
k = 1
если (d - 1) % 2 != 0 то:
    k = -1

ответ = ((k * p * новый_b) % новый_m).остаток_по_модулю(m)

вернуть ответ
```

Трудоемкость алгоритма $O(\log(m)^2)$.

Алгоритм 4 - алгоритм решения линейных сравнений $ax \equiv b \pmod m$

Вход: целые числа a, b, m.

Выход: массив $s = \{s_1, s_2, \ldots, s_k\}$, содержащий решения линейных сравнений 1-й степени.

Шаг 1. С помощью расширенного алгоритма Евклида от a и m получить значения d, p, q — НОД и коэффициенты перед большим и меньшим числом соответственно.

Шаг 2. Определить $b_2 = \frac{b}{d}$, $n = \frac{m}{d}$, $k = 1$.

Шаг 3. Если остаток от деления $d - 1$ на 2 не равен 0, то $k = k - 1$.

Шаг 4. Определить $s_1 = k \cdot p \cdot b_2 \pmod n$. Определить массив s и добавить в него s_1.

Шаг 5. Определить $i = 1$.

Шаг 6. Опредилить $s_{i+1} = s_1 + n \cdot i \pmod m$ и добавить его в массив s.

Шаг 7. Если $i < d$, то $i = i + 1$ и перейти к шагу 6, иначе перейти к шагу 8.

Шаг 8. Результат: массив s.

Псевдокод:

```
функция Решение_линейного_сравнения(a, b, m):
    (_, d, p, q) = Расширенный_алгоритм_Евклида(a, m, 0, 0)
```

```
новый_b = b / d
новый_m = m / d

k = 1
если (d - 1) % 2 != 0 то:
    k = -1

первый_ответ = ((k * p * новый_b) %
 ↪  новый_m).остаток_по_модулю(m)
решения = [первый_ответ]

для i от 1 до d:
    решения.добавить((решения[0] + новый_m *
     ↪  i).остаток_по_модулю(m))

вывести_массив(решения)
```

Трудоемкость алгоритма $O(\log(\max\{a, m\}))$.

3.2 Описание алгоритмов вычисления символов Лежандра и Якоби и алгоритма извлечения квадратного корня в кольце вычетов

Алгоритм 5 - алгоритм вычисления символа Лежандра

Вход: целые числа a, p.

Выход: целое число, равное значению символа Лежандра $\left(\frac{a}{p}\right)$.

Шаг 1. Если $a < 0$, то по свойству 4 выделить множитель $\left(\frac{-1}{p}\right)$.

Шаг 2. Замена a на остаток от деления на p.

Шаг 3. Представить $a = p_1^{\alpha_1} \cdot \dots \cdot p_k^{\alpha_k}$ и вычислить по свойству 4

$$\left(\frac{a}{p}\right) = \left(\frac{p_1}{p}\right)^{\alpha_1} \cdot \dots \cdot \left(\frac{p_k}{p}\right)^{\alpha_k}.$$

Множители с четными степенями α_i опускаются, а вместо множителей $\left(\frac{p_i}{p}\right)^{\alpha_i}$ с нечетными степенями α_i оставить $\left(\frac{p_i}{p}\right)$.

Шаг 4. Если $p_i = 2$, то вычислить согласно свойству 6: $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$.

Шаг 5. К остальным символам $\left(\frac{p_i}{p}\right)$ применяется квадратичный закон Гаусса.

Шаг 6. Если a не равно нулю — вернуться к шагу 2. Иначе перейти к шагу 7.

Шаг 7. Результат: целое число, равное значению символа Лежандра $\left(\frac{a}{p}\right)$.

Псевдокод:

```
функция Символ_Лежандра(a, p):
если Является_простым(p) то:
    вернуть Символ_Якоби(a, p)

a = a % p
если a < 0 то:
    a = a + p

результат = 1
пока a не равно 0:
    пока a делится на 2:
        a = a / 2
        если p % 8 равно 3 или p % 8 равно 5 то:
            результат = -результат

    обменять_местами(a, p)

    если a % 4 равно 3 и p % 4 равно 3 то:
        результат = -результат

    a = a % p

если p равно 1 то:
    вернуть результат
иначе:
    вернуть 0
```

Трудоемкость алгоритма $O(\log(a) \cdot \log(p))$.

Алгоритм 6 - алгоритм вычисления символа Якоби

Вход: целые числа a, p.

Выход: целое число, равное значению символа Якоби $\left(\frac{a}{p}\right)$.

Шаг 1. Заменить a на такое b, что $a \equiv b \pmod{p}$ и $|b| < \frac{p}{2}$.

Шаг 2. Если $b < 0$, то по свойству 4 выделить множитель $\left(\frac{-1}{p}\right)$.

Шаг 3. Если b — четное, то представить $b = 2^t \cdot a_1$ и при нечетном t вычислить $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$.

Шаг 4. К символу $\left(\frac{a_1}{p}\right)$ применяется квадратичный закон взаимности Гаусса.
Шаг 5. При необходимости перейти к шагу 1. Иначе перейти к шагу 6.
Шаг 6. Результат: целое число, равное значению символа Якоби $\left(\frac{a}{p}\right)$.

Псевдокод:

```
функция Символ_Якоби(a, n):
t = 1
пока a не равно 0:
    пока a делится на 2:
        a = a / 2
        n_mod_8 = n % 8
        если n_mod_8 равно 3 или n_mod_8 равно 5 то:
            t = -t

    обменять_местами(a, n)
    если a % 4 равно 3 и n % 4 равно 3 то:
        t = -t

    a = a % n

если n равно 1 то:
    вернуть t
иначе:
    вернуть 0
```

Трудоемкость алгоритма $O(\log(p))$.

Алгоритм 7 - алгоритм извлечения квадратного корня в кольце вычетов

Вход: Число a и модуль p (где $p - 1 = 2^m q$).
Выход: Число x — квадратный корень числа a в кольце вычетов $\mathbb{Z}_m$.
Шаг 1. Случайным образом выбрать такое b, что $\left(\frac{b}{p}\right) = -1$.
Шаг 2. Вычислить последовательность $a_1, \ldots, a_n$ элементов поля $\mathbb{Z}_p$ и последовательность чисел $k_1, \ldots, k_n$ по правилу:

1. $a_1 = a$, $a_{i+1} = a_i \cdot b^{2^{m-k_i}} \pmod p$, $i \geq 1$;
2. k_i — наименьшее $k \geq 0$, при котором $a_i^{2^k q} \equiv 1 \pmod p$

Шаг 3. Если равенство $k_n = 0$ не выполняется, перейти к шагу 2.
Шаг 4. Вычислить последовательность элементов $r_n, \ldots, r_1$ элементов поля $\mathbb{Z}_p$ по правилу:

$$r_n = a_n^{\frac{q+1}{2}} \pmod p,\ r_i = r_{i+1}(b^{2^{m-k_i-1}})^{-1} \pmod p,\ i \geq 1.$$

Шаг 5. Положить $x = r_1$ — искомое решение.

Шаг 6. Результат: число x — квадратный корень числа a в кольце вычетов $\mathbb{Z}_m$.

Псевдокод:

```
функция Квадратный_корень_в_кольце_вычетов(a, p):
если символ_Лежандра(a, p) не равно 1 то:
    вернуть -1

q = p - 1
s = 0

пока q делится на 2:
    q = q / 2
    s = s + 1

если s равно 1 то:
    x = a.в_степень((p + 1) / 4) % p
    вернуть x

z = 2

пока символ_Лежандра(z, p) не равно -1:
    z = z + 1

c = z.в_степень(q) % p
r = a.в_степень((q + 1) / 2) % p
t = a.в_степень(q) % p
m = s

пока t не равно 1:
    i = 0
    zz = t

    пока zz не равно 1:
        zz = (zz * zz) % p
        i = i + 1
```

```
            b = c.в_степень(2 в степени (m - i - 1)) % p
            r = (r * b) % p
            t = (t * b.в_степень(2)) % p
            c = b.в_степень(2) % p
            m = i

    вернуть r
```

Трудоемкость алгоритма $O(\log(p)^4)$

3.3 Код программы, реализующей рассмотренные алгоритмы

```rust
use std::io;
use rand::Rng;
// use quadratic::jacobi;
use rand::distributions::Uniform;

fn read_integer() -> i32 {
    let mut n = String::new();
    io::stdin()
        .read_line(&mut n)
        .expect("failed to read input.");
    let n: i32 = n.trim().parse().expect("invalid input");
    n
}


fn euclid_gcd(a: i32, b: i32) -> i32 {
    if b == 0 {
        return a;
    };
    let r = a % b;
    return euclid_gcd(b, r);
}


fn euclid_gcd_extended(a: i32, b: i32, _x: i32, _y: i32) -> (i32, i32, i32,
↪ i32) {
    if a == 0 {
        return (a, b, 0, 1);
```

```
    }
    else {
        let (_, d, x1, y1) = euclid_gcd_extended(b % a, a, 0, 0);
        let division = b / a;
        let x = y1 - division * x1;
        let y = x1;
        return (0, d, x, y)
    }
}


fn continued_fraction(mut a: i32, mut b: i32) -> Vec<i32> {
    let mut fraction: Vec<i32> = Vec::new();
    while b != 0 {
        fraction.push(a / b);
        let c = a;
        a = b;
        b = c % b;
    }
    fraction
}


fn get_p_n_q(fraction: Vec<i32>) -> (Vec<i32>, Vec<i32>) {
    let mut p: Vec<i32> = vec![0, 1];
    let mut q: Vec<i32> = vec![1, 0];
    for i in 2..fraction.len() + 2 {
        let coef = fraction[i - 2 as usize];
        p.push(p[(i - 1) as usize] * coef + p[(i - 2) as usize]);
        q.push(q[(i - 1) as usize] * coef + q[(i - 2) as usize]);
    }
    return (p, q)
}


fn print_array(arr: Vec<i32>) {
    print!("[");
    for i in 0..arr.len() {
        print!("{}", arr[i]);
        if i < arr.len() - 1 {
            print!(", ");
```

```
        }
    }
    println!("]");
}


fn check_coprime(modules: Vec<i32>, n: i32) -> bool {
    for i in 0..n {
        for j in 0..n {
            let m_i = modules[i as usize];
            let m_j = modules[j as usize];
            if euclid_gcd(m_i, m_j) != 1 && i != j {
                return false;
            }
        }
    }
    return true;
}


fn mod_inverse(a: i32, n: i32) -> i32 {
    let fraction: Vec<i32> = Vec::new();
    let (_, g, x, _) = euclid_gcd_extended(a, n, 0, 0);
    if g != 1 {
        return -1;
    } else {
        let result = (x % n + n) % n;
        return result;
    }
}


fn is_prime(n: i32) -> bool {
    if n <= 1 {
        return false;
    }
    if n == 2 || n == 3 {
        return true;
    }
    if n % 2 == 0 || n % 3 == 0 {
        return false;
```

```
    }

    let mut i = 5;
    while i * i <= n {
        if n % i == 0 || n % (i + 2) == 0 {
            return false;
        }
        i += 6;
    }

    true
}


fn solve_continued_fraction() -> () {
    println!("Введите рациональное число r = a_0/a_1.");
    println!("Введите a_0:");
    let a0 = read_integer();
    println!("Введите a_1:");
    let a1 = read_integer();

    let fraction: Vec<i32> = Vec::new();
    let mut fraction = continued_fraction(a0, a1);
    println!("Цепная дробь:");
    print_array(fraction);
}


fn diophantine_solution(a: i32, b: i32, c: i32) -> (i32, i32, i32, i32) {
    let fraction: Vec<i32> = Vec::new();
    let mut fraction = continued_fraction(a, b);

    let p: Vec<i32> = Vec::new();
    let q: Vec<i32> = Vec::new();
    let (p, q) = get_p_n_q(fraction.clone());

    let k = fraction.len();

    let x = q[k as usize] * i32::pow(-1, k as u32) * c;
    let y = p[k as usize] * i32::pow(-1, k as u32) * c;

```

```
    (x, y, a, b)
}


fn solve_diophantine() -> () {
    println!("Введите a:");
    let a = read_integer();
    println!("Введите b:");
    let b = read_integer();
    println!("Введите c:");
    let c = read_integer();

    let (x, y, a, b) = diophantine_solution(a, b, c);

    println!("Решениями уравнения являются:");
    println!("x = {} + {}t", x, b);
    println!("y = {} + {}t", y, a);
}


fn linear_comparison_solution() -> () {
    println!("Введите a:");
    let a = read_integer();
    println!("Введите b:");
    let b = read_integer();
    println!("Введите m:");
    let m = read_integer();

    let (_, d, p, q) = euclid_gcd_extended(a, m, 0, 0);

    if (b % d) == 0 {
        println!("Сравнение имеет {} решений.", d);
    }
    else {
        println!("Сравнение не имеет решений.");
        return ();
    }

    let new_b = b / d;
    let new_m = m / d;

```

```
    let mut k = 1;
    if (d - 1) % 2 != 0 {
        k = -1;
    }

    let first_ans = ((k * p * new_b) % new_m).rem_euclid(m);
    let mut solutions: Vec<i32> = vec![first_ans];

    for i in 1..d {
        solutions.push((solutions[0 as usize] + new_m * i).rem_euclid(m));
    }

    print_array(solutions.clone());
}


fn power_mod(base: i32, exponent: i32, modulus: i32) -> i32 {
    if modulus == 1 {
        return 0;
    }
    let mut result = 1;
    let mut base = base.rem_euclid(modulus);
    let mut exp = exponent;

    while exp > 0 {
        if exp % 2 == 1 {
            result = (result * base).rem_euclid(modulus);
        }
        exp = exp >> 1;
        base = (base * base).rem_euclid(modulus);
    }

    result.rem_euclid(modulus)
}


fn get_inverse_elem() -> () {
    println!("Введите элемент a, для которого хотите найти обратный:");
    let a = read_integer();
    println!("Введите модуль m:");
    let m = read_integer();
```

```
    let b = 1;

    let (_, d, p, q) = euclid_gcd_extended(a, m, 0, 0);
    let new_b = b / d;
    let new_m = m / d;
    let mut k = 1;
    if (d - 1) % 2 != 0 {
        k = -1;
    }
    let first_ans = ((k * p * new_b) % new_m).rem_euclid(m);

    println!("Обратный элемент: {}", first_ans);
}


fn continued_fraction_application() -> () {
    println!("Выберите приложение:");
    println!("1 - решение линейных диофантовых уравнений;");
    println!("2 - вычисление обратных элементов в кольце вычетов Z_m;");
    println!("3 - решение линейных сравнений ax = b (mod m);");

    let n = read_integer();

    match n {
        1 => solve_diophantine(),
        2 => get_inverse_elem(),
        3 => linear_comparison_solution(),
        _ => println!("Введено неверное число!"),
    }
}


fn legendre_symbol(a: i32, mut p: i32) -> i32 {
    if is_prime(p) {
        return jacobi_symbol(a, p);
    }

    let mut a = a % p;
    if a < 0 {
        a += p;
    }
```

```
    let mut result = 1;
    while a != 0 {
        while a % 2 == 0 {
            a /= 2;
            if p % 8 == 3 || p % 8 == 5 {
                result = -result;
            }
        }

        std::mem::swap(&mut a, &mut p);

        if a % 4 == 3 && p % 4 == 3 {
            result = -result;
        }

        a %= p;
    }

    if p == 1 {
        result
    } else {
        0
    }
}


fn jacobi_symbol(mut a: i32, mut n: i32) -> i32 {
    let mut t = 1;
    while a != 0 {
        while a % 2 == 0 {
            a /= 2;
            let n_mod_8 = n % 8;
            if n_mod_8 == 3 || n_mod_8 == 5 {
                t = -t;
            }
        }

        std::mem::swap(&mut a, &mut n);
        if a % 4 == 3 && n % 4 == 3 {
            t = -t;
```

```
        }

        a %= n;
    }

    if n == 1 {
        return t;
    } else {
        return 0;
    }
}


fn eval_quadratic_residues() -> () {
    println!("Введите a:");
    let a = read_integer();
    println!("Введите p:");
    let p = read_integer();

    let (r1, r2) = find_square_roots(a, p);
    if r1 == r2 && r1 == 0 {
        println!("Для заданных a и p квадратных корней найти не удалось.");
        return ();
    }
    if r1 != 0 {
        println!("Найденный корень: {}", r1);
    }
    if r2 != 0 {
        println!("Найденный корень: {}", r2);
    }
}


fn sqrt_mod(a: i32, p: i32) -> i32 {
    if legendre_symbol(a, p) != 1 {
        return -1;
    }

    let mut q = p - 1;
    let mut s = 0;

```

```
    while q % 2 == 0 {
        q /= 2;
        s += 1;
    }

    if s == 1 {
        let x = a.pow(((p + 1) / 4) as u32) % p;
        return x;
    }

    let mut z = 2;

    while legendre_symbol(z, p) != -1 {
        z += 1;
    }

    let mut c = z.pow(q as u32) % p;
    let mut r = a.pow(((q + 1) / 2) as u32) % p;
    let mut t = a.pow(q as u32) % p;
    let mut m = s;

    while t != 1 {
        let mut i = 0;
        let mut zz = t;

        while zz != 1 {
            zz = (zz * zz) % p;
            i += 1;
        }

        let b = c.pow(2u32.pow((m - i - 1) as u32) as u32) % p;
        r = (r * b) % p;
        t = (t * b.pow(2)) % p;
        c = b.pow(2) % p;
        m = i;
    }

    r
}
```

```
fn find_square_roots(a: i32, p: i32) -> (i32, i32) {
    let mut root1 = 0;
    let mut root2 = 0;

    for i in 0..p {
        if (i*i % p) == a {
            if root1 == 0 {
                root1 = i;
            } else {
                root2 = i;
                break;
            }
        }
    }

    (root1, root2)
}


fn eval_symbols() -> () {
    println!("Введите a:");
    let a = read_integer();
    println!("Введите p:");
    let p = read_integer();

    println!("Символ Лежандра: {}", legendre_symbol(a, p));
    println!("Символ Якоби: {}", jacobi_symbol(a, p));
}


fn main() {
    println!("Выберите опцию:");
    println!("1 - разложение числа в цепную дробь;");
    println!("2 - алгоритм приложения цепных дробей;");
    println!("3 - алгоритмы вычисления символов Лежандра и Якоби;");
    println!("4 - алгоритм извлечения квадратного корня в кольце вычетов;");

    let n = read_integer();

    match n {
        1 => solve_continued_fraction(),
```

```
            2 => continued_fraction_application(),
            3 => eval_symbols(),
            4 => eval_quadratic_residues(),
            _ => println!("Введено неверное число!"),
        }
    }
```

3.4 Результаты тестирования программ

```
Выберите опцию:
1 - разложение числа в цепную дробь;
2 - алгоритм приложения цепных дробей;
3 - алгоритмы вычисления символов Лежандра и Якоби;
4 - алгоритм извлечения квадратного корня в кольце вычетов;
1
Введите рациональное число r = a_0/a_1.
Введите a_0:
415
Введите a_1:
93
Цепная дробь:
[4, 2, 6, 7]
```

Рисунок 1 – Тест алгоритма разложения в цепную дробь

```
Выберите опцию:
1 - разложение числа в цепную дробь;
2 - алгоритм приложения цепных дробей;
3 - алгоритмы вычисления символов Лежандра и Якоби;
4 - алгоритм извлечения квадратного корня в кольце вычетов;
2
Выберите приложение:
1 - решение линейных диофантовых уравнений;
2 - вычисление обратных элементов в кольце вычетов Z_m;
3 - решение линейных сравнений ax = b (mod m);
1
Введите a:
19
Введите b:
15
Введите c:
11
Решениями уравнения являются:
x = 44 + 15t
y = 55 + 19t
```

Рисунок 2 – Тест приложения цепной дроби, решения линейных диофантовых уравнений

```
Выберите опцию:
1 - разложение числа в цепную дробь;
2 - алгоритм приложения цепных дробей;
3 - алгоритмы вычисления символов Лежандра и Якоби;
4 - алгоритм извлечения квадратного корня в кольце вычетов;
2
Выберите приложение:
1 - решение линейных диофантовых уравнений;
2 - вычисление обратных элементов в кольце вычетов Z_m;
3 - решение линейных сравнений ax = b (mod m);
2
Введите элемент a, для которого хотите найти обратный:
3
Введите модуль m:
221
Обратный элемент: 74
```

Рисунок 3 – Тест приложения цепной дроби, вычисления обратных элементов в кольце вычетов

```
Выберите опцию:
1 - разложение числа в цепную дробь;
2 - алгоритм приложения цепных дробей;
3 - алгоритмы вычисления символов Лежандра и Якоби;
4 - алгоритм извлечения квадратного корня в кольце вычетов;
2
Выберите приложение:
1 - решение линейных диофантовых уравнений;
2 - вычисление обратных элементов в кольце вычетов Z_m;
3 - решение линейных сравнений ax = b (mod m);
3
Введите a:
45
Введите b:
21
Введите m:
132
Сравнение имеет 3 решений.
[21, 65, 109]
```

Рисунок 4 – Тест приложения цепной дроби, решения линейных сравнений 1-ой степени

```
Выберите опцию:
1 - разложение числа в цепную дробь;
2 - алгоритм приложения цепных дробей;
3 - алгоритмы вычисления символов Лежандра и Якоби;
4 - алгоритм извлечения квадратного корня в кольце вычетов;
3
Введите a:
3
Введите p:
7
Символ Лежандра: -1
Символ Якоби: -1
```

Рисунок 5 – Тест алгоритмов вычисления символов Лежандра и Якоби

```
Выберите опцию:
1 - разложение числа в цепную дробь;
2 - алгоритм приложения цепных дробей;
3 - алгоритмы вычисления символов Лежандра и Якоби;
4 - алгоритм извлечения квадратного корня в кольце вычетов;
4
Введите a:
131
Введите p:
2897
Найденный корень: 1238
Найденный корень: 1659
```

Рисунок 6 – Тест алгоритма вычисления квадратного корня в кольце вычетов

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе были рассмотрены теоретические сведения об алгоритме разложения чисел в цепную дробь, алгоритмы приложений цепных дробей, а также алгоритмы вычисления символов Лежандра и Якоби и извлечения квадратного корня в кольце вычетов. На их основе были составлены соответствующие алгоритмы. Была произведена оценка сложности созданных алгоритмов. Они послужили фундаментом для программной реализации, которая впоследствии успешно прошла тестирование, результаты которого были прикреплены к отчету вместе с листингом программы, написанной на языке Rust с использованием стандартных библиотек языка.