МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«ТЮМЕНСКИЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт дополнительного и дистанционного образования

Управление в технических системах

Отчет по лабораторной работе №1

по дисциплине

«Защита информации в автоматизированных системах управления»

Выполнил:

магистрант группы ИБм(до)з-22-1

Соловьёва Н.Л.

Проверил:

Доцент кафедры КС к.т.н,

Музипов Х.Н.

Содержание

Постановка задачи и решение лабораторной работы №1

Цель работы: написать программу, реализующую набор функций для работы с числами.

Задачи:

Написать функцию для нахождения наибольшего общего делителя двух чисел на основе алгоритма Евклида.

Реализовать функцию для нахождения последовательности простых чисел, не превосходящих заданного числа N, на основе алгоритма Эратосфена.

Создать функцию для вычисления функции Эйлера для положительного целого числа.

Написать функцию для решения линейных сравнений.

В рамках работы необходимо оформить программу в виде отдельного модуля (библиотеки), а также предоставить отчет, содержащий схемы алгоритмов, описание основных функций программы, словесное описание модулей, тестовые примеры и выводы по работе.

Решение задач:

Дано:

1. Найдите НОД (396,1452)?
2. Найти каноническое разложение числа а)2156; б)1934; в)1132.
3. Являются ли числа попарно простыми

а) (678, 941); б) (243, 1485); в) (535, 321)?

4. Решить линейное сравнение:

а) ; б) ; в)

Для решения упражнений вам потребуется использовать функции из модуля "number_theory.py". Давайте по очереди решим каждое упражнение.

Чтобы найти НОД (396, 1452), используем функцию euclidean_gcd из модуля "number_theory.py":

```
from number_theory import euclidean_gcd

a = 396

b = 1452

gcd = euclidean_gcd(a, b)

print(f"НОД({a}, {b}) = {gcd}")
```

Чтобы найти каноническое разложение числа, вам потребуется определить простые множители числа. Для этого можно воспользоваться функцией sieve_of_eratosthenes из модуля "number_theory.py" и последовательно делять число на найденные простые числа до тех пор, пока результат деления не станет равным 1.

```
from number_theory import sieve_of_eratosthenes

def prime_factorization(n):
    primes = sieve_of_eratosthenes(n)
    factors = []
    for prime in primes:
        while n % prime == 0:
            factors.append(prime)
            n //= prime
    if n > 1:
```

```
        factors.append(n)
    return factors
# а) Каноническое разложение числа 2156:
n = 2156
factors = prime_factorization(n)
print(f"Каноническое разложение числа {n}: {factors}")
# б) Каноническое разложение числа 1934:
n = 1934
factors = prime_factorization(n)
print(f"Каноническое разложение числа {n}: {factors}")
# в) Каноническое разложение числа 1132:
n = 1132
factors = prime_factorization(n)
print(f"Каноническое разложение числа {n}: {factors}")
```

Чтобы определить, являются ли числа попарно простыми, нужно проверить их попарные НОДы. Используем функцию euclidean_gcd из модуля "number_theory.py":

```
from number_theory import euclidean_gcd
# а) Числа (678, 941):
numbers = [678, 941]
pairwise_coprime = True
for i in range(len(numbers)):
    for j in range(i+1, len(numbers)):
```

```
        gcd = euclidean_gcd(numbers[i], numbers[j])
        if gcd != 1:
            pairwise_coprime = False
            break
print(f"Числа (678, 941) являются попарно простыми: {pairwise_coprime}")
# б) Числа (243, 1485):
numbers = [243, 1485]
pairwise_coprime = True
for i in range(len(numbers)):
    for j in range(i+1, len(numbers)):
        gcd = euclidean_gcd(numbers[i], numbers[j])
        if gcd != 1:
            pairwise_coprime = False
            break
print(f"Числа (243, 1485) являются попарно простыми: {pairwise_coprime}")
# в) Числа (535, 321):
numbers = [535, 321]
pairwise_coprime = True
for i in range(len(numbers)):
    for j in range(i+1, len(numbers)):
        gcd = euclidean_gcd(numbers[i], numbers[j])
        if gcd != 1:
```

```
        pairwise_coprime = False
        break

print(f"Числа (535, 321) являются попарно простыми: {pairwise_coprime}")
```

Для решения линейного сравнения вида ax ≡ b (mod m) используем функцию linear_congruence из модуля "number_theory.py":

```
from number_theory import linear_congruence

# а) Решение линейного сравнения 3x ≡ 1 (mod 7):
a = 3
b = 1
m = 7
solutions = linear_congruence(a, b, m)
if solutions:
    print(f"Решение линейного сравнения {a}x ≡ {b} (mod {m}): {solutions}")
else:
    print("Линейное сравнение не имеет решений.")

# б) Решение линейного сравнения 4x ≡ 2 (mod 10):
a = 4
b = 2
m = 10
solutions = linear_congruence(a, b, m)
if solutions:
    print(f"Решение линейного сравнения {a}x ≡ {b} (mod {m}): {solutions}")
```

```
else:
    print("Линейное сравнение не имеет решений.")
# в) Решение линейного сравнения 5x ≡ 1 (mod 6):
a = 5
b = 1
m = 6
solutions = linear_congruence(a, b, m)
if solutions:
    print(f"Решение линейного сравнения {a}x ≡ {b} (mod {m}): {solutions}")
else:
    print("Линейное сравнение не имеет решений.")
```

Пожалуйста, используйте код выше, чтобы решить упражнения.

Что внутри алгоритма:

1. Найдем НОД(396, 1452) с помощью алгоритма Евклида:

   1452 = 3 * 396 + 264

   396 = 1 * 264 + 132

   264 = 2 * 132 + 0

   Последний ненулевой остаток равен 132. Таким образом, НОД(396, 1452) = 132.

2. Каноническое разложение чисел:

   а) 2156 = 2^2 * 539 = 2^2 * 7 * 77 = 2^2 * 7 * 7 * 11

   б) 1934 = 2 * 967

   в) 1132 = 2^2 * 283

3. Проверим, являются ли числа попарно простыми:

а) (678, 941): НОД(678, 941) = 67, так как 67 является единственным делителем для обоих чисел, они являются попарно простыми.

б) (243, 1485): НОД(243, 1485) = 27, так как 27 является общим делителем для обоих чисел, они не являются попарно простыми.

в) (535, 321): НОД(535, 321) = 107, так как 107 является единственным делителем для обоих чисел, они являются попарно простыми.

4. Решим линейные сравнения:

а) $5x \equiv 3 \pmod 7$:

Найдем обратный элемент 5 по модулю 7. В данном случае, $5^{-1} \equiv 3 \pmod 7$.

Умножим обе части на 5^{-1}:

$3 * 5x \equiv 3 * 3 \pmod 7$,

$25x \equiv 9 \pmod 7$.

Найдем остаток от деления 9 на 7, что равно 2.

Решением линейного сравнения $5x \equiv 3 \pmod 7$ является $x \equiv 2 \pmod 7$.

б) $6x \equiv 9 \pmod{15}$:

Найдем обратный элемент 6 по модулю 15. В данном случае, $6^{-1} \equiv 11 \pmod{15}$.

Умножим обе части на 6^{-1}:

$9 * 6x \equiv 9 * 11 \pmod{15}$,

$54x \equiv 99 \pmod{15}$.

Найдем остаток от деления 99 на 15, что равно 9.

Решением линейного сравнения $6x \equiv 9 \pmod{15}$ является $x \equiv 9 \pmod{15}$.

в) $4x \equiv 6$

$\pmod 8$:

Данное линейное сравнение не имеет решений, так как 4 и 8 не являются взаимно простыми числами.

Вопросы к лабораторной работе № 1

1. Докажите теорему о делении с остатком.
2. Дайте понятие НОД двух чисел? Какой алгоритм используется для его нахождения?
3. Какие числа называются простыми, составными?
4. Расскажите алгоритм нахождения последовательности простых чисел.
5. Какое разложение целого числа называется каноническим?
6. Что показывает функция Эйлера, как она рассчитывается?
7. Расскажите алгоритм решения линейных сравнений.

1. Докажите теорему о делении с остатком.

Теорема о делении с остатком утверждает, что для любых целых чисел a и b, где $b \neq 0$, существуют единственные целые числа q и r, такие что $a = bq + r$, где q - частное, r - остаток, причем $0 \leq r < |b|$.

Доказательство:

Шаг 1: Существование остатка и частного:

Рассмотрим два целых числа a и b, где $b \neq 0$. Рассмотрим множество всех чисел, которые могут быть представлены в виде a - bx, где x - целое число. Пусть это множество обозначается как S. Мы должны доказать, что существует такое число $r \in S$, что $0 \leq r < |b|$.

Рассмотрим множество $S = \{a - bx \mid x \in Z\}$. Если S содержит положительное число, то существует наименьший элемент r в S (по аксиоме вполнеупорядоченности натуральных чисел). В этом случае, $a - br \in S$ и $a - br < r$ не может быть положительным, так как r является наименьшим положительным числом в S. Таким образом, r не может быть положительным и, следовательно, $r \leq 0$.

Шаг 2: Ограничение остатка:

Предположим, что $r < 0$. Рассмотрим число $r + |b|$. Тогда $a - b(q + 1) = a - bq - b = r + |b| > r$, где $q + 1$ - новое предполагаемое частное. Значит, r не может быть меньше 0.

Таким образом, мы доказали, что существует такое число r, что $0 \leq r < |b|$.

Шаг 3: Единственность остатка и частного:

Предположим, что есть два различных представления для a в виде $a = bq + r$ и $a = bq' + r'$, где $0 \leq r, r' < |b|$. Тогда $bq + r = bq' + r'$. Отсюда следует, что $b(q - q') = r' - r$.

Поскольку левая часть равенства делится на b, а правая часть меньше $|b|$, то получаем, что обе части равенства равны нулю. Таким образом, $r' - r = 0$ и $q - q' = 0$, что приводит к тому, что $r = r'$ и $q = q'$.

Таким образом, мы доказали, что остаток r и частное q в разложении $a = bq + r$ являются единственными.

Теорема о делении с остатком доказана.

2. Дайте понятие НОД двух чисел? Какой алгоритм используется для его нахождения?

НОД (Наибольший Общий Делитель) двух чисел - это наибольшее целое число, которое одновременно делит оба числа без остатка. Другими словами, НОД двух чисел - это наибольший общий делитель для этих чисел.

Для нахождения НОД двух чисел существует несколько алгоритмов, одним из самых известных и эффективных является алгоритм Евклида.

Алгоритм Евклида работает по следующему принципу:

Пусть a и b - два числа, для которых мы хотим найти НОД.

Если b равно 0, то НОД(a, b) равен a.

В противном случае, мы заменяем a на b, и b на остаток от деления a на b (a mod b).

Повторяем шаги 2 и 3 до тех пор, пока b не станет равным 0.

Когда b равно 0, НОД(a, b) равен a.

Алгоритм Евклида основан на простом наблюдении, что НОД(a, b) равен НОД(b, a mod b). Поэтому мы последовательно заменяем большее число на остаток от деления и продолжаем этот процесс до достижения нулевого остатка.

Алгоритм Евклида эффективен и позволяет быстро находить НОД двух чисел даже при больших значениях. Он широко используется в математике и алгоритмах для решения различных задач, таких как нахождение простых чисел, решение линейных сравнений и других.

3. Какие числа называются простыми, составными?

Простые числа - это натуральные числа больше единицы, которые имеют ровно два делителя: 1 и само число. Иными словами, простое число делится только на 1 и на себя.

Например, числа 2, 3, 5, 7, 11 и 13 являются простыми, так как они имеют только два делителя: 1 и само число.

Составные числа - это натуральные числа больше единицы, которые имеют более двух делителей. Иначе говоря, составное число делится не только на 1 и на себя, но также имеет другие делители.

Например, число 4 является составным, так как оно делится не только на 1 и на себя, но также на 2. Аналогично, числа 6, 8, 9 и 10 также являются составными, так как у них есть делители, отличные от 1 и самих чисел.

Простые числа являются основными строительными блоками для всех других чисел. Составные числа можно представить как произведение простых чисел.

4. Расскажите алгоритм нахождения последовательности простых чисел.

Алгоритм нахождения последовательности простых чисел, не превосходящих заданное число N, основан на алгоритме Эратосфена. Этот алгоритм позволяет эффективно определить все простые числа в заданном диапазоне.

Шаги алгоритма Эратосфена для нахождения простых чисел до N:

Создайте список чисел от 2 до N и обозначьте их как простые.

Начиная с числа 2, пометьте его как простое, а все его кратные числа (кроме самого числа) пометьте как составные.

Перейдите к следующему непомеченному числу, которое будет простым, и пометьте все его кратные числа как составные.

Повторяйте шаг 3 до тех пор, пока не пройдете все числа от 2 до N.

Все непомеченные числа в итоге останутся простыми числами.

Пример работы алгоритма:

Допустим, нам нужно найти все простые числа, не превосходящие 30.

Создаем список чисел от 2 до 30: [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30].

Начинаем с числа 2. Помечаем его как простое и помечаем все его кратные числа (4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30) как составные.

Переходим к следующему непомеченному числу, которым является 3. Помечаем его как простое и помечаем все его кратные числа (6, 9, 12, 15, 18, 21, 24, 27, 30) как составные.

Переходим к следующему непомеченному числу, которым является 4, но так как оно уже помечено как составное, пропускаем его.

Переходим к следующему непомеченному числу, которым является 5. Помечаем его как простое и помечаем все его кратные числа (10, 15, 20, 25, 30) как составные.

Продолжаем этот процесс для оставшихся чисел.

В итоге получаем следующую последовательность простых чисел: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29].

Алгоритм Эратосфена позволяет эффективно находить простые числа до заданного числа N, и его временная сложность составляет O(n log log n), где n - это заданное число.

5. Какое разложение целого числа называется каноническим?

Каноническое разложение целого числа - это представление числа в виде произведения простых чисел, возможно с повторениями, в котором порядок сомножителей не имеет значения.

Формально, для целого числа n, его каноническое разложение выглядит следующим образом:

$n = p_1^{\alpha_1} * p_2^{\alpha_2} * p_3^{\alpha_3} * ... * p_k^{\alpha_k}$,

где p_1, p_2, p_3, ..., p_k - простые числа, α_1, α_2, α_3, ..., α_k - их соответствующие степени.

Например, каноническое разложение числа 2156:

2156 = 2^2 * 7 * 7 * 11.

В данном случае, число 2156 представлено в виде произведения простых чисел, где 2 встречается в квадрате (2^2), а числа 7 и 11 присутствуют в первой степени.

Каноническое разложение позволяет полностью представить число в виде его простых сомножителей и играет важную роль в различных областях математики и теории чисел.

6. Что показывает функция Эйлера, как она рассчитывается?

Функция Эйлера, также известная как функция φ (phi) или функция Эйлера-Фи, является арифметической функцией, которая показывает количество положительных целых чисел, меньших и взаимно простых с заданным числом n.

Функция Эйлера обозначается как φ(n), где n - положительное целое число.

Для рассчета функции Эйлера φ(n), необходимо выполнить следующие шаги:

Разложить число n на простые множители, получив его каноническое разложение.

Для каждого простого множителя p в каноническом разложении числа n вычислить φ(p) = p - 1, так как каждое простое число p взаимно просто со всеми числами меньше него.

Умножить все значения φ(p) для каждого простого множителя p в разложении n: $\varphi(n) = \varphi(p_1) * \varphi(p_2) * ... * \varphi(p_k)$.

Например, для числа n = 10:

Каноническое разложение числа 10: 10 = 2 * 5.

Значения φ(2) = 2 - 1 = 1 и φ(5) = 5 - 1 = 4.

Вычисляем φ(10) = φ(2) * φ(5) = 1 * 4 = 4.

Таким образом, функция Эйлера для числа 10 равна 4.

Функция Эйлера имеет много применений в теории чисел и криптографии, особенно в контексте алгоритмов шифрования и нахождения обратного элемента в кольцах по модулю.

7. Расскажите алгоритм решения линейных сравнений.

Алгоритм решения линейных сравнений основан на нахождении обратного элемента в кольце вычетов. Линейное сравнение имеет вид:

ax ≡ b (mod m),

где a, b и m - целые числа, a и m не равны нулю, и x - неизвестное целое число, которое мы хотим найти.

Для решения линейного сравнения необходимо выполнить следующие шаги:

Найдите наибольший общий делитель (НОД) чисел a и m с помощью алгоритма Евклида. Если НОД(a, m) не равен 1, то линейное сравнение не имеет решений. Если НОД(a, m) равен 1, переходим к следующему шагу.

Найдите обратный элемент a по модулю m. Обратный элемент a обозначается как a^(-1) и удовлетворяет условию: a * a^(-1) ≡ 1 (mod m). Для нахождения обратного элемента можно использовать расширенный алгоритм Евклида.

Умножьте обе части исходного линейного сравнения на обратный элемент a^(-1):

(a^(-1) * a * x) ≡ (a^(-1) * b) (mod m).

Так как a * a^(-1) ≡ 1 (mod m), то остается:

x ≡ (a^(-1) * b) (mod m).

Найдите остаток от деления (a^(-1) * b) на m. Это будет решение линейного сравнения.

Например, рассмотрим линейное сравнение 3x ≡ 2 (mod 7):

Найдем НОД(3, 7) с помощью алгоритма Евклида. Получим НОД(3, 7) = 1, что означает, что уравнение имеет решение.

Найдем обратный элемент 3 по модулю 7. В данном случае, 3^(-1) ≡ 5 (mod 7).

Умножим обе части исходного сравнения на 3^(-1):

(5 * 3 * x) ≡ (5 * 2) (mod 7).

Получаем: 15x ≡ 10 (mod 7).

Найдем остаток от деления 10 на 7, что равно 3. Таким образом, решением линейного сравнения 3x ≡ 2 (mod 7) является x ≡ 3 (mod 7).

Алгоритм решения линейных сравнений может быть обобщен и применен для любых линейных сравнений с различными значениями a, b и m.

Схемы алгоритмов и описание функций.

Ниже приведены схемы алгоритмов и описание основных функций программы для каждой из задач лабораторной работы №1.

Нахождение наибольшего общего делителя двух чисел на основе алгоритма Евклида:

Схема алгоритма:

Алгоритм EuclideanGCD(a, b):

Пока b не равно 0:

Вычислить остаток от деления a на b и сохранить его в переменной r.

Присвоить a значение b.

Присвоить b значение r.

Вернуть a.

Описание функции:

```
def EuclideanGCD(a, b):
    while b != 0:
        r = a % b
        a = b
        b = r
    return a
```

Нахождение последовательности простых чисел, не превосходящих заданного числа N, на основе алгоритма Эратосфена:

Схема алгоритма:

Алгоритм SieveOfEratosthenes(N):

Создать список чисел от 2 до N и обозначить их как простые.

Пока i^2 <= N:

Если i помечено как простое:

Для каждого числа j в интервале от i^2 до N с шагом i:

Пометить j как составное.

Увеличить i на 1.

Вернуть список всех простых чисел.

Описание функции:

```
def SieveOfEratosthenes(N):
    primes = [True] * (N + 1)
    primes[0] = primes[1] = False
    i = 2
    while i * i <= N:
        if primes[i]:
            for j in range(i * i, N + 1, i):
                primes[j] = False
        i += 1
    prime_numbers = [num for num, is_prime in enumerate(primes) if is_prime]
    return prime_numbers
```

Вычисление функции Эйлера для положительного целого числа:

Схема алгоритма:

Алгоритм EulerTotientFunction(n):

Инициализировать переменную count со значением 0.

Для каждого числа i в интервале от 1 до n:

Если i и n взаимно просты:

Увеличить count на 1.

Вернуть count.

Описание функции:

```
def EulerTotientFunction(n):
    count = 0
```

```
    for i in range(1, n + 1):
        if EuclideanGCD(i, n) == 1:
            count += 1
    return count
```

Решение линейных сравнений:

Схема алгоритма:

Алгоритм LinearCongruenceSolver(a, b, m):

Найти НОД(a, m) и сохранить его в переменной d.

Если b не делится нацело на d:

Решений нет.

Иначе:

Найти одно частное решение x0, используя расширенный алгоритм Евклида.

Выписать общее решение в виде x = x0 + (k * m) / d, где k - целое число.

Вернуть общее решение.

Описание функции:

```
def LinearCongruenceSolver(a, b, m):
    d = EuclideanGCD(a, m)
    if b % d != 0:
        return "No solution"
    else:
        x0, y0 = ExtendedEuclideanAlgorithm(a, m)
        x = (x0 * (b // d)) % m
```

return x

Схемы алгоритмов помогают визуализировать последовательность действий, выполняемых в каждой функции.

Выводы по работе

В ходе выполнения лабораторной работы №1 были решены следующие задачи:

Нахождение наибольшего общего делителя двух чисел с использованием алгоритма Евклида.

Нахождение последовательности простых чисел с использованием алгоритма Эратосфена.

Вычисление функции Эйлера для положительного целого числа.

Решение линейных сравнений.

Для решения каждой задачи были представлены соответствующие схемы алгоритмов и описания основных функций программы.

Алгоритм Евклида является эффективным способом нахождения наибольшего общего делителя двух чисел. Он основан на простой итеративной процедуре деления с остатком, позволяя быстро и надежно найти НОД.

Алгоритм Эратосфена позволяет эффективно находить последовательность простых чисел до заданного числа N. Он работает на основе пошагового вычеркивания составных чисел из списка, начиная с 2. Полученный список содержит только простые числа.

Функция Эйлера представляет собой функцию, определяющую количество чисел, взаимно простых с заданным положительным целым числом. Она используется в различных областях математики и криптографии для вычисления определенных значений и свойств.

Решение линейных сравнений с помощью алгоритма нахождения обратного элемента и расширенного алгоритма Евклида позволяет найти все решения такого сравнения. Результатом является общее решение, выраженное через одно частное решение и параметры, определяющие множество всех решений.

Лабораторная работа позволила познакомиться с основными математическими концепциями и алгоритмами, используемыми в вычислительной математике. Реализация данных алгоритмов в виде программы позволяет эффективно и точно решать соответствующие задачи.