Алгоритм Евклида

Миличевич Александра

15 Февраля, 2025, Москва, Россия

Российский Университет Дружбы Народов

Цель лабораторной работы

Познакомиться с алгоритмами вычисления наибольшего общего делителя (Алгоритмами Евклида

1. Алгоритм Евклида
2. Расширенный алгоритм Евклида.
3. Бинарный алгоритм Евклида.
4. Расширенный бинарный алгоритм Евклида

Выполнение лабораторной работы

Объяснение алгоритмов Евклида для нахождения НОД)

Этот документ описывает три функции, реализующие различные алгоритмы Евклида для вычисления наибольшего общего делителя (НОД) двух чисел.

`euclidean_simply(first_number, second_number)`

Как работает:

1. Функция использует цикл `while`, который выполняется до тех пор, пока оба числа (`first_number` и `second_number`) не станут равны нулю.
2. Внутри цикла, если `first_number` больше или равно `second_number`, то `first_number` заменяется на остаток от деления `first_number` на `second_number` (`first_number %= second_number`).
3. Иначе, `second_number` заменяется на остаток от деления `second_number` на `first_number` (`second_number %= first_number`).

Код

```python
# Функция уменьшает числа до тех пор, пока одно из них не станет нулем
# Практически, для этого используется цикл
def euclidean_simply(first_number, second_number):
    """Вычисляет наибольший общий делитель (НОД) двух чисел,
    используя простой алгоритм Евклида."""
    while first_number != 0 and second_number != 0:
        if first_number >= second_number:
            first_number %= second_number
        else:
            second_number %= first_number
    return first_number or second_number
```

Рис. 1: code (simply eucleadian)

Расширенный алгоритм Евклида

Как работает:

- 1. Функция использует рекурсию.
- 2. Базовый случай: если `first_number` равно 0, то возвращается кортеж `(second_number, 0, 1).
- 3. В противном случае, функция вызывает себя рекурсивно с параметрами `second_number % first_number` и `first_number`, получает результаты `div, x, y`.
- 4. Затем вычисляет и возвращает новый кортеж `(div, y - (second_number // first_number) * x, x)`. Этот кортеж содержит НОД и коэффициенты Безу.

Код

```python
# Функция для расширенного алгоритма Евклида
# first_number * x + second_number * y = gcd(first_number, second_number)
# Алгоритм находит НОД и его линейное представление
def euclidean_extended(first_number, second_number):
    """Вычисляет НОД двух чисел и коэффициенты Безу.
       Возвращает кортеж (gcd, x, y), где gcd - НОД, а
       x и y удовлетворяют уравнению first_number * x + second_number * y = gcd."""
    if first_number == 0:
        return (second_number, 0, 1) # Базовый случай: НОД(0, second_number) = second_number
    else:
        div, x, y = euclidean_extended(second_number % first_number, first_number)
        return (div, y - (second_number // first_number) * x, x)
```

Рис. 2: extended

Бинарный алгоритм Евклида

Как работает:

1. Используется переменная `common_power_of_2` для отслеживания общих степеней двойки.
2. Сначала числа делятся на 2, пока оба четные, при этом `common_power_of_2` умножается на 2.
3. Далее, используются `u` и `v` для хранения текущих значений.
4. Пока `u` не равно нулю, выполняется цикл:
    * Если `u` четное, то `u` делится на 2.
    * Иначе, если `v` четное, то `v` делится на 2.
    * Иначе, если `u` больше или равно `v`, то `u` вычитается `v`.
    * Иначе, `v` вычитается `u`.
5. Функция возвращает результат, умноженный на накопленную степень двойки.

Код

```python
# Функция для бинарного алгоритма Евклида
def binary_euclidean(first_number, second_number):
    """Вычисляет НОД двух чисел, используя бинарный алгоритм Евклида."""
    common_power_of_2 = 1  # Переменная для отслеживания общих степеней 2
    while (first_number % 2 == 0 and second_number % 2 == 0):
        first_number //= 2
        second_number //= 2
        common_power_of_2 *= 2
    u, v = first_number, second_number
    while u != 0:
        if u % 2 == 0:
            u //= 2
        elif v % 2 == 0:
            v //= 2
        elif u >= v:
            u -= v
        else:
            v -= u
    return common_power_of_2 * v
```

Рис. 3: binary

Расширенный бинарный алгоритм Евклида

Как работает:

1. Так же как и в `binary_euclidean`, выделяется общая степень двойки.
2. Используются `u` и `v` для хранения текущих значений. `A, B, C, D` — это коэффициенты для расширенного алгоритма.
3. Пока `u` не равно нулю, выполняется цикл:
    * Если `u` четное, то `u` делится на 2. Если `A` и `B` четные, то они тоже делятся на 2. Иначе, `A` и `B` обновляются с учетом четности и делителя.
    * Аналогично для `v` и `C, D`.
    * Если `u >= v`, то `u` и `A, B` уменьшаются.
    * Иначе, `v` и `C, D` уменьшаются.
4. Функция возвращает НОД и коэффициенты Безу, умноженные на общую степень двойки.

Код

```python
# Функция для расширенного бинарного алгоритма Евклида
def binary_euclidean_extended(first_number, second_number):
    """Вычисляет НОД двух чисел и коэффициенты Безу, используя
    расширенный бинарный алгоритм Евклида."""
    common_power_of_2 = 1
    while (first_number % 2 == 0 and second_number % 2 == 0):
        first_number //= 2
        second_number //= 2
        common_power_of_2 *= 2
    u, v = first_number, second_number
    A, B, C, D = 1, 0, 0, 1
    while u != 0:
        if u % 2 == 0:
            u //= 2
            if A % 2 == 0 and B % 2 == 0:
                A //= 2
                B //= 2
            else:
                A = (A + second_number) // 2
                B = (B - first_number) // 2
        elif v % 2 == 0:
            v //= 2
            if C % 2 == 0 and D % 2 == 0:
                C //= 2
                D //= 2
            else:
                C = (C + second_number) // 2
                D = (D - first_number) // 2
        elif u >= v:
            u -= v
            A -= C
            B -= D
        else:
            v -= u
            C -= A
            D -= B
    return common_power_of_2 * v, C, D


# Ввод чисел
first_number = int(input("Введите первое число: "))
second_number = int(input("Введите второе число: "))

if first_number >= 0 and 0 <= second_number <= first_number:
    print("НОД (простой Евклид):", euclidean_simply(first_number, second_number))
    print("НОД и коэффициенты Безу (расширенный Евклид):", euclidean_extended(first_number, second_number))
    print("НОД (бинарный Евклид):", binary_euclidean(first_number, second_number))
    print("НОД и коэффициенты Безу (расширенный бинарный Евклид):", binary_euclidean_extended(first_number, second_number))
else:
    print("Введенные значения не соответствуют условиям.")
```

Рис. 4: extended binary

Выводы

Результаты выполнения лабораторной работы

Программно реализованы алгоритми Евклида для нахождения НОД.