

Отчёт по лабораторной работе №8

Математические основы защиты информации и информационной безопасности

Целочисленная арифметика многократной точности

**Выполнил: Мануэл Марсия Педру,
НФИмд-02-25, 1032255503**

Содержание

1 Цель работы	5
2 Выполнение лабораторной работы	6
2.1 Реализация алгоритма 1 (сложение неотрицательных целых чисел)	6
2.2 Реализация алгоритма 2 (вычитание неотрицательных целых чисел)	7
2.3 Реализация алгоритма 3 (умножение неотрицательных целых чисел столбиком)	8
2.4 Реализация алгоритма 5 (деление многоразрядных целых чисел) .	8
3 Список литературы. Библиография	11

Список иллюстраций

2.1	Реализация алгоритма 1	6
2.2	Реализация алгоритма 1	7
2.3	Реализация алгоритма 2	7
2.4	Реализация алгоритма 3	8
2.5	Реализация алгоритма 5	9
2.6	Проверка	9
2.7	Проверка	10
2.8	Проверка	10

Список таблиц

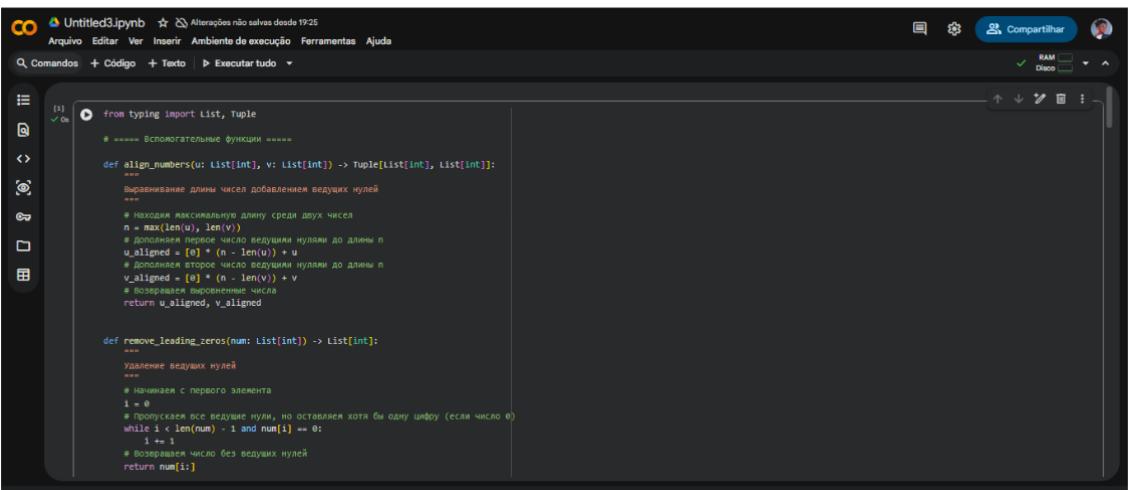
1 Цель работы

Изучить алгоритмы для выполнения арифметических операций с большими целыми числами и научиться их реализовывать.

2 Выполнение лабораторной работы

2.1 Реализация алгоритма 1 (сложение неотрицательных целых чисел)

Выполним реализацию этого алгоритма на языке python (рис. 2.1) и (рис. 2.2):



```
from typing import List, Tuple

# ===== Вспомогательные функции =====

def align_numbers(u: List[int], v: List[int]) -> Tuple[List[int], List[int]]:
    """
    Выравнивание длины чисел добавлением ведущих нулей
    """
    # Находим максимальную длину среди двух чисел
    n = max(len(u), len(v))
    # Дополняем первое число ведущими нулями до длины n
    u_aligned = [0] * (n - len(u)) + u
    # Дополняем второе число ведущими нулями до длины n
    v_aligned = [0] * (n - len(v)) + v
    # Возвращаем выровненные числа
    return u_aligned, v_aligned

def remove_leading_zeros(num: List[int]) -> List[int]:
    """
    Удаление ведущих нулей
    """
    # Начинаем с первого элемента
    i = 0
    # Пропускаем все ведущие нули, но оставляем хотя бы одну цифру (если число 0)
    while i < len(num) - 1 and num[i] == 0:
        i += 1
    # Возвращаем число без ведущих нулей
    return num[i:]
```

Рис. 2.1: Реализация алгоритма 1

```

# ===== Алгоритм 1: Сложение неотрицательных целых чисел =====
def big_add(u: List[int], v: List[int], b: int) -> List[int]:
    """
    Алгоритм 1: Сложение неотрицательных целых чисел
    """
    # Выравниваем числа по длине
    u_aligned, v_aligned = align_numbers(u, v)
    # Получаем длину выровненных чисел
    n = len(u_aligned)
    # Создаем массив для результата (на 1 элемент больше для возможного переноса)
    w = [0] * (n + 1)
    # Инициализируем перенос
    k = 0

    # Проходим по разрядам справа налево
    for j in range(n - 1, -1, -1):
        # Вычисляем сумму разрядов с учетом переноса
        total = u_aligned[j] + v_aligned[j] + k
        # Определяем текущий разряд результата (остаток от деления на основание)
        w[j + 1] = total % b
        # Вычисляем новый перенос (целая часть от деления)
        k = total // b

    # Записываем оставшийся перенос в старший разряд
    w[0] = k
    # Удаляем ведущие нули и возвращаем результат
    return remove_leading_zeros(w)

# ===== Алгоритм 2: Вычитание неотрицательных целых чисел =====

```

Рис. 2.2: Реализация алгоритма 1

2.2 Реализация алгоритма 2 (вычитание неотрицательных целых чисел)

Выполним реализацию этого алгоритма на языке python (рис. 2.3):

```

# ===== Алгоритм 2: Вычитание неотрицательных целых чисел =====
def big_subtract(u: List[int], v: List[int], b: int) -> List[int]:
    """
    Алгоритм 2: Вычитание неотрицательных целых чисел
    """
    # Выравниваем числа по длине
    u_aligned, v_aligned = align_numbers(u, v)
    # Получаем длину выровненных чисел
    n = len(u_aligned)
    # Создаем массив для результата
    w = [0] * n
    # Инициализируем заем
    k = 0

    # Проходим по разрядам справа налево
    for j in range(n - 1, -1, -1):
        # Вычисляем разность с учетом заема
        diff = u_aligned[j] - v_aligned[j] + k
        if diff < 0:
            # Если результат отрицательный, занимаем из старшего разряда
            w[j] = diff + b # добавляем основание системы
            k = -1 # устанавливаем флаг заема
        else:
            w[j] = diff # записываем разность
            k = 0 # сбрасываем флаг заема

    # Удаляем ведущие нули и возвращаем результат
    return remove_leading_zeros(w)

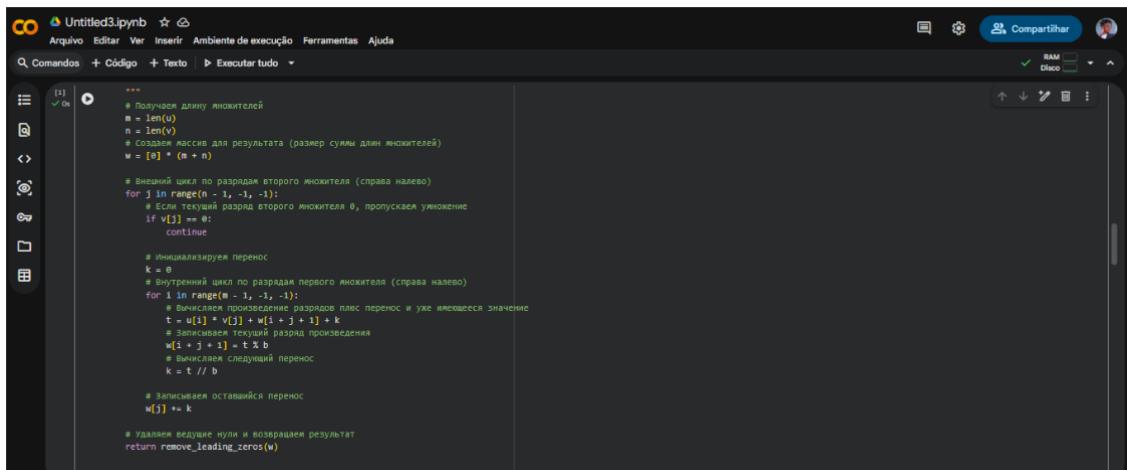
# ===== Алгоритм 3: Умножение неотрицательных целых чисел столбиком =====

```

Рис. 2.3: Реализация алгоритма 2

2.3 Реализация алгоритма 3 (умножение неотрицательных целых чисел столбиком)

Выполним реализацию этого алгоритма на языке python (рис. 2.4):



```
# Получаем длину множителей
m = len(u)
n = len(v)
# Создаем массив для результата (размер суммы длин множителей)
w = [0] * (m + n)

# Внешний цикл по разрядам второго множителя (справа налево)
for j in range(n - 1, -1, -1):
    # Если текущий разряд второго множителя 0, пропускаем умножение
    if v[j] == 0:
        continue

    # Инициализируем перенос
    k = 0
    # Внутренний цикл по разрядам первого множителя (справа налево)
    for i in range(m - 1, -1, -1):
        # Вычисляем произведение разрядов плюс перенос и уже имеющееся значение
        t = u[i] * v[j] + w[i + j + 1] + k
        # Записываем текущий разряд произведения
        w[i + j + 1] = t % b
        # Вычисляем следующий перенос
        k = t // b

    # Записываем оставшийся перенос
    w[0] += k

# Удаляем недлинные нули и возвращаем результат
return remove_leading_zeros(w)
```

Рис. 2.4: Реализация алгоритма 3

2.4 Реализация алгоритма 5 (деление многоразрядных целых чисел)

Выполним реализацию этого алгоритма на языке python (рис. 2.5):

```

def big_divide(u: List[int], v: List[int], b: int) -> Tuple[List[int], List[int]]:
    """
    Алгоритм 5: деление многоразрядных целых чисел (упрощенная версия)
    """
    # Преобразуем массивы цифр в числа
    num_u = int(''.join(str(digit) for digit in u), b)
    num_v = int(''.join(str(digit) for digit in v), b)

    # Вычисляем частное и остаток с помощью встроенных операций
    q_num = num_u // num_v  # Целочисленное деление
    r_num = num_u % num_v  # Остаток от деления

    # Преобразуем частное обратно в список цифр в заданной системе счисления
    if q_num == 0:
        q = [0]
    else:
        q_digits = []
        while q_num > 0:
            q_digits.insert(0, q_num % b)
            q_num //= b
        q = q_digits

    # Преобразуем остаток обратно в список цифр в заданной системе счисления
    if r_num == 0:
        r = [0]
    else:
        r_digits = []
        while r_num > 0:
            r_digits.insert(0, r_num % b)
            r_num //= b
        r = r_digits

```

Рис. 2.5: Реализация алгоритма 5

Проверим работу алгоритмов (рис. 2.6):

```

def test_arithmetic():
    """
    Расширенное тестирование арифметики больших чисел
    """
    print("Тест 1 (небольшие числа в десятичной системе):")
    u = [1, 2, 3]
    v = [4, 5]
    print(f"u = {u}, v = {v}")

    result_add = big_add(u, v, 10)
    print(f"Сложение: Результат = {result_add}")

    result_sub = big_subtract(u, v, 10)
    print(f"Вычитание: Результат = {result_sub}")

    result_mul = big_multiply(u, v, 10)
    print(f"Умножение: Результат = {result_mul}")

    result_div_q, result_div_r = big_divide(u, v, 10)
    print(f"Деление: Частное = {result_div_q}, остаток = {result_div_r}")

    print("Тест 2 (числа с разной длиной в десятичной системе):")
    u = [9, 8, 9, 9]
    v = [2, 5]
    print(f"u = {u}, v = {v}")

```

Рис. 2.6: Проверка

Проверим работу алгоритмов (рис. 2.7):

```

Untitled3.ipynb ☆ ⓘ
Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda
Q Comandos + Código + Texto | Executar tudo ▾
[1] ✓ Os
print("Сложение: Результат = (result_add)")

result_sub = big_subtract(u, v, 10)
print("Вычитание: Результат = (result_sub)")

result_mul = big_multiply(u, v, 10)
print("Умножение: Результат = (result_mul)")

result_div_q, result_div_r = big_divide(u, v, 10)
print("Деление: Частное = (result_div_q), Остаток = (result_div_r)")

print("Тест 3 (деление на 1 в десятичной системе):")
u = [7, 8, 9]
v = [1]
print(f"u = {u}, v = {v}")

result_div_q, result_div_r = big_divide(u, v, 10)
print("Деление: Частное = (result_div_q), остаток = (result_div_r)")

print("Тест 4 (двоичная система счисления):")
u = [1, 0, 1, 1] # 11 в двоичной
v = [1, 0, 1] # 5 в двоичной
print(f"u = {u} (двоичное), v = {v} (двоичное)")

result_add = big_add(u, v, 2)
print("Сложение: Результат = (result_add) (двоичное)")

result_mul = big_multiply(u, v, 2)
print("Умножение: Результат = (result_mul) (двоичное)")

```

Рис. 2.7: Проверка

Проверим работу алгоритмов (рис. 2.8):

```

Untitled3.ipynb ☆ ⓘ
Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda
Q Comandos + Código + Texto | Executar tudo ▾
[1] ✓ Os
print("Деление: частное = (result_div_q) (двоичное), остаток = (result_div_r) (двоичное)")

if __name__ == "__main__":
    test_arithmetic()

... Тест 1 (деление числа в десятичной системе):
... u = [1, 0, 1, 1], v = [1, 1]
... Сложение: Результат = [1, 0, 0, 0, 1]
... Вычитание: Результат = [1, 1]
... Умножение: Результат = [1, 0, 0, 0, 1]
... Деление: Частное = [1], остаток = [1]

... Тест 2 (числа с разной длиной в десятичной системе):
... u = [9, 9, 9], v = [2, 5]
... Сложение: Результат = [1, 0, 0, 0, 2, 4]
... Вычитание: Результат = [9, 9, 7, 4]
... Умножение: Результат = [2, 4, 9, 9, 7, 5]
... Деление: Частное = [3, 9, 9], остаток = [2, 4]

... Тест 3 (деление на 1 в десятичной системе):
... u = [7, 8, 9], v = [1]
... Деление: Частное = [7, 8, 9], остаток = [0]

... Тест 4 (двоичная система счисления):
... u = [1, 0, 1, 1] (двоичное), v = [1, 0, 1] (двоичное)
... Сложение: Результат = [1, 0, 0, 0, 0, 0] (двоичное)
... Умножение: Результат = [1, 1, 0, 1, 1] (двоичное)
... Деление: Частное = [1, 0] (двоичное), остаток = [1] (двоичное)

```

Рис. 2.8: Проверка

3 Список литературы. Библиография

:::