marp: true css: custom-theme.css title: "Лабораторная работа №6"на простоту" subtitle: "Дисциплина: Математические основы защиты информации и информационной безопасности" author: Хосе Фернандо Леон Атупанья, НФИмд-01-24, 1032249918 institute: Российский Университет Дружбы Народов, Москва, Россия date: 26 октября 2024

Лабораторная работа № 8

Дисциплина: Математические основы защиты информации и информационной безопасности

Тема: Целочисленная арифметика многократной точности.

Студент: Леон Фернандо Хосе Фернандо

Цель работы

Ознакомиться с темой целочисленная арифметика многократной точности, используя материал, представленный в лабораторной работе № 8, и используя концепции, представленные в предыдущих работах, такие как модули, максимальный общий делитель и шифрование.

Задание

1. Реализовать рассмотренные алгоритмы программно

2. Выполнение лабораторной работы

1. Алгоритм 1 (сложение неотрицательных целых чисел)

u и v - это векторы цифр чисел в базе b, w инициализируется как массив нулей с n + 1 элементами для хранения результата, включая переносимую цифру. Цикл перебирает цифры от наименее значимой до наиболее значимой (j = n, n-1, ..., 1). Вычисляет сумму цифр из u и v в позиции j, включая перенос k.

1. Код алгоритмы

```
function add_large_integers(u::Vector{Int}, v::Vector{Int}, b::Int)
    n = length(u)  # Number of digits in the numbers
    w = zeros(Int, n + 1)  # Result array (extra space for carry)
    k = 0  # Initial carry

    for j in n:-1:1
        # Compute the sum of digits, including the carry
        digit_sum = u[j] + v[j] + k
        w[j + 1] = digit_sum % b  # Least significant digit of the result
```

```
        k = digit_sum ÷ b        # Carry for the next position
    end

    # Set the carry to the first digit (w₀)
    w[1] = k
    return w
end
```

---

2. Алгоритм 2 (вычитание неотрицательных целых чисел)

```
function subtract_large_integers(u::Vector{Int}, v::Vector{Int}, b::Int)
    n = length(u)  # Number of digits in the numbers
    w = zeros(Int, n)  # Result array
    k = 0  # Initial borrow

    for j in n:-1:1
        # Compute the difference, including the borrow
        digit_diff = u[j] - v[j] + k
        if digit_diff < 0
            digit_diff += b  # Adjust with base if negative
            k = -1           # Borrow for the next position
        else
            k = 0            # No borrow needed
        end
        w[j] = digit_diff  # Store the result
    end
    return w
end
```

---

3. Алгоритм 3 (умноджение неотрицательных целых чисел)

```
function multiply_large_integers(u::Vector{Int}, v::Vector{Int}, b::Int)
    n = length(u)  # Number of digits in u
    m = length(v)  # Number of digits in v
    w = zeros(Int, n + m)  # Result array (enough space for the product)

    for j in m:-1:1
        if v[j] == 0
            continue  # Skip if the digit in v is 0
        end

        k = 0  # Initial carry
        for i in n:-1:1
            t = u[i] * v[j] + w[i + j] + k  # Multiply, add to the result and
carry
            w[i + j] = t % b  # Store least significant digit
            k = t ÷ b         # Carry for the next iteration
```

```
        end

        w[j] = k  # Store remaining carry
    end
    return w
end
```

---

4. Алгоритм 4 (быстрый столбик)

```
function fast_column_multiply(u::Vector{Int}, v::Vector{Int}, b::Int)
    n = length(u)  # Number of digits in u
    m = length(v)  # Number of digits in v
    w = zeros(Int, n + m)  # Result array
    t = 0  # Intermediate sum

    for s in 0:(m + n - 2)
        t = 0  # Reset intermediate sum for each position
        for i in 0:s
            # Include valid products where indices align
            if n - i > 0 && m - (s - i) > 0
                t += u[n - i] * v[m - (s - i)]
            end
        end
        # Compute result digit and carry
        w[m + n - 1 - s] = t % b
        t = t ÷ b
    end

    # Handle the final carry
    w[1] = t
    return w
end
```

---

5. Алгоритм 5 (деление многоразрядных целых чисел)

```
function divide_large_integers(u::Vector{Int}, v::Vector{Int}, b::Int)
    n = length(u) - 1  # Degree of dividend u
    t = length(v) - 1  # Degree of divisor v
    q = zeros(Int, n - t + 1)  # Quotient array
    r = copy(u)  # Copy of u, will hold the remainder
    # Step 2: Adjust for high powers of b
    while compare_large_integers(r, v, n - t, b) >= 0
        q[end] += 1
        r = subtract_scaled(r, v, n - t, b)
    end
    # Step 3: Perform the main division algorithm
    for i in n:-1:(t + 1)
```

```
        if r[i + 1] >= v[t + 1]
            q[i - t - 1] = b - 1
        else
            q[i - t - 1] = (r[i + 1] * b + r[i]) ÷ v[t + 1]
        end
        while q[i - t - 1] * (v[t + 1] * b + v[t]) >
              (r[i + 1] * b^2 + r[i] * b + r[i - 1])
            q[i - t - 1] -= 1
        end
        # Subtract q[i-t-1] * v * b^(i-t-1) from r
        r = subtract_scaled(r, v, i - t - 1, b, q[i - t - 1])
        if r[1] < 0
            r = add_scaled(r, v, i - t - 1, b)
            q[i - t - 1] -= 1
        end
    end
    return q, r
end
```

Вывод

В заключение хотелось бы отметить, что разработанные алгоритмы для многоточной целочисленной арифметики демонстрируют надежные решения для таких фундаментальных операций, как сложение, вычитание, умножение и деление. Каждый алгоритм был тщательно разработан и реализован, чтобы справиться со сложностями послойных вычислений в любой базовой системе счисления. Корректность этих методов обеспечивается строгим соблюдением принципов модульной арифметики и управлением переносами, заимствованиями и промежуточными результатами.