## Отчет по лабораторной работе №8

### 1. Цель работы

Цель данной лабораторной работы – реализовать быстрые алгоритмы для сложения, вычитания, умножения столбиком, быстрого умножения и деления с большими числами.

### 2. Задание

### Алгоритм 1 (сложение неотрицательных целых чисел)

**Вход.** Два неотрицательных числа **u = u₁u₂…uₙ** и **v = v₁v₂…vₙ**; разрядность чисел *n*; основание системы счисления *b*.  
**Выход.** Сумма **w = w₀w₁…wₙ**, где **w₀** — цифра переноса — всегда равная 0 либо 1.

1. Присвоить **j := n, k := 0** (j идет по разрядам, k следит за переносом).
2. Присвоить **wⱼ = (uⱼ + vⱼ + k) (mod b)**, где **wⱼ** — наименьший неотрицательный вычет в данном классе вычетов;  
   **k = ⌊(uⱼ + vⱼ + k) / b⌋.**
3. Присвоить **j := j − 1**. Если **j > 0**, то возвращаемся на шаг 2; если **j = 0**, то присвоить **w₀ := k** и результат: **w**.

### Алгоритм 2 (вычитание неотрицательных целых чисел)

**Вход.** Два неотрицательных числа **u = u₁u₂…uₙ** и **v = v₁v₂…vₙ**, **u > v**; разрядность чисел *n*; основание системы счисления *b*.  
**Выход.** Разность **w = w₁w₂…wₙ = u − v.**

1. Присвоить **j := n, k := 0** (k — заем из старшего разряда).
2. Присвоить **wⱼ = (uⱼ − vⱼ + k) (mod b)**, где **wⱼ** — наименьший неотрицательный вычет в данном классе вычетов;  
   **k = ⌊(uⱼ − vⱼ + k) / b⌋.**
3. Присвоить **j := j − 1**. Если **j > 0**, то возвращаемся на шаг 2; если **j = 0**, то результат: **w**.

### Алгоритм 3 (умножение неотрицательных целых чисел столбиком)

**Вход.** Числа **u = u₁u₂…uₙ, v = v₁v₂…vₘ**; основание системы счисления *b*.  
**Выход.** Произведение **w = uv = w₁w₂…wₙ₊ₘ**.

1. Выполнить присвоения:  
   **wₘ₊₁ := 0, wₘ := 0, …, w₁ := 0**, **j := m**  
   (j перемещается по номерам разрядов числа **v** от младших к старшим).
2. Если **vⱼ = 0**, то присвоить **wⱼ := 0** и перейти на шаг 6.
3. Присвоить **i := n, k := 0** (Значение **i** идет по номерам разрядов числа **u**, **k** отвечает за перенос).
4. Присвоить  
   **t := uᵢ ⋅ vⱼ + wᵢ₊ⱼ + k, wᵢ₊ⱼ := t (mod b), k := ⌊t / b⌋**,  
   где **wᵢ₊ⱼ** — наименьший неотрицательный вычет в данном классе вычетов.
5. Присвоить **i := i − 1**. Если **i > 0**, то возвращаемся на шаг 4, иначе присвоить **wⱼ := k**.
6. Присвоить **j := j − 1**. Если **j > 0**, то вернуться на шаг 2. Если **j = 0**, то результат: **w**.

### Алгоритм 4 (быстрый столбик)

**Вход.** Числа **u = u₁u₂…uₙ, v = v₁v₂…vₘ**; основание системы счисления *b*.  
**Выход.** Произведение **w = uv = w₁w₂…wₙ₊ₘ**.

1. Присвоить **t := 0**.
2. Для **s от 0 до n + m − 1 с шагом 1** выполнить шаги 3 и 4.
3. Для **t = 0** до **n** выполнить присвоение  
   **t := t + uₙ₋ₛ₊ₜ ⋅ vₘ₋ₜ₊ₛ**.
4. Вычислить  
   **wₛ₊₁ := t (mod b), t := ⌊t / b⌋**,  
   где **wₛ₊₁** — наименьший неотрицательный вычет по модулю *b*.  
   Результат: **w**.

### Алгоритм 5 (деление многоразрядных целых чисел)

**Вход.** Числа **u = uₙ…u₁u₀, v = vₜ…v₁v₀**, **n ≥ t ≥ 1, vₜ ≠ 0**, разрядность чисел соответственно **n** и **t**.  
**Выход.** Частное **q = qₙ₋ₜ…q₀**, остаток **r = rₜ…r₀**.

1. Для **j** от 0 до **n − t** присвоить **qⱼ := 0**.
2. Пока **u ≥ v bⁿ⁻ᵗ**, выполнять:  
   **qₙ₋ₜ := qₙ₋ₜ + 1, u := u − v bⁿ⁻ᵗ.**
3. Для **i := n, n − 1, …, t + 1** выполнять пункты 3.1 – 3.4:  
   **3.1.** Если **uᵢ ≥ vₜ**, то присвоить  
   **qᵢ₋ₜ₋₁ := b − 1**,  
   иначе присвоить  
   **qᵢ₋ₜ₋₁ := ⌊(uᵢb + uᵢ₋₁) / vₜ⌋.**

* **3.2.** Пока  
  **qᵢ₋ₜ₋₁(vₜb + vₜ₋₁) > uᵢb² + uᵢ₋₁b + uᵢ₋₂**,  
  выполнять  
  **qᵢ₋ₜ₋₁ := qᵢ₋ₜ₋₁ − 1.**
* **3.3.** Присвоить  
  **u := u − qᵢ₋ₜ₋₁v bⁱ⁻ᵗ⁻¹.**
* **3.4.** Если **u < 0**, то присвоить  
  **u := u + v bⁱ⁻ᵗ⁻¹, qᵢ₋ₜ₋₁ := qᵢ₋ₜ₋₁ − 1.**

1. **r := u.**  
   Результат: **q и r.**

### 3. Теоретическое введение

Будем считать, что число записано в *b*-ичной системе счисления, *b* — натуральное число, *b ≥ 2*. Натуральное *b*-разрядное число будем записывать в виде:

**u = u₁u₂…uₙ.**

Арифметика многократной точности (или произвольной точности) позволяет выполнять операции с целыми числами, превышающими размер стандартных типов данных, предоставляемых языками программирования. Это необходимо в различных областях, включая криптографию, научные вычисления и обработку больших данных.

При работе с большими целыми числами знак такого числа удобно хранить в отдельной переменной. Например, при умножении двух чисел, знак произведения вычисляется отдельно. Квадратные скобки обозначают, что берется целая часть числа. Для представления больших чисел обычно используются массивы или списки, где каждый элемент соответствует отдельному разряду числа

### 4. Выполнение лабораторной работы

**Функции, необходимые для упрощения работы и многократно использующиеся в алгоритмах**

# Убираем нули  
function remove\_leading\_zeros(number::Vector{Int})  
 while length(number) > 1 && number[1] == 0  
 popfirst!(number)  
 end  
 return number  
end  
  
# Функция сравнения   
  
function compare(u::Vector{Int}, v::Vector{Int})::Int  
 u = remove\_leading\_zeros(copy(u))  
 v = remove\_leading\_zeros(copy(v))  
   
 if length(u) > length(v)  
 return 1  
 elseif length(u) < length(v)  
 return -1  
 else  
 for (digit\_u, digit\_v) in zip(u, v)  
 if digit\_u > digit\_v  
 return 1  
 elseif digit\_u < digit\_v  
 return -1  
 end  
 end  
 return 0  
 end  
end  
  
# Функция добавления нулей  
  
function pad\_with\_leading\_zeros(u::Vector{Int}, v::Vector{Int})::Tuple{Vector{Int}, Vector{Int}}  
 max\_len = max(length(u), length(v))  
 u\_padded = vcat(zeros(Int, max\_len - length(u)), u)  
 v\_padded = vcat(zeros(Int, max\_len - length(v)), v)  
 return (u\_padded, v\_padded)  
end  
  
# Перевод числа в вектор цифр  
  
function number\_to\_digits(number::Union{String, Int})::Vector{Int}  
 if isa(number, Int)  
 number = string(number)  
 elseif isa(number, String)  
 if !all(c -> isdigit(c), number)  
 error("Число должно содержать только цифры.")  
 end  
 else  
 error("Число должно быть строкой или целым числом.")  
 end  
 return [parse(Int, string(c)) for c in number]  
end  
  
# Перевод цифр в число  
  
function digits\_to\_number(digits::Vector{Int})::String  
 return join(string.(digits))  
end  
function number\_to\_digits\_func(number::Union{String, Int})::Vector{Int}  
 return number\_to\_digits(number)  
end  
  
function digits\_to\_number\_func(digits::Vector{Int})::String  
 return digits\_to\_number(digits)  
end

digits\_to\_number\_func (generic function with 1 method)

**Алгоритм 1 (сложение неотрицательных целых чисел)**

function add(u::Vector{Int}, v::Vector{Int}, b::Int)::Vector{Int}  
 u\_padded, v\_padded = pad\_with\_leading\_zeros(u,v)  
 n = length(u\_padded)  
 w = zeros(Int, n+1)  
 k = 0   
  
 for j in n:-1:1  
 total = u\_padded[j] + v\_padded[j] + k  
 w[j+1] = total % b  
 k = div(total,b)  
 end  
 w[1] = k   
 return w   
end

add (generic function with 1 method)

**Пример сложения**

u\_num = "123456789012345"  
v\_num = "987654321098765"  
b = 10  
u = number\_to\_digits\_func(u\_num)  
v = number\_to\_digits\_func(v\_num)  
sum\_result = add(u, v, b)  
sum\_str = digits\_to\_number\_func(sum\_result)  
println("u = $u\_num")  
println("v = $v\_num")  
println("w = $sum\_str")

u = 123456789012345  
v = 987654321098765  
w = 1111111110111110

**Алгоритм 2 (вычитание неотрицательных целых чисел)**

function subtract(u::Vector{Int}, v::Vector{Int}, b::Int)::Vector{Int}  
 u\_padded, v\_padded = pad\_with\_leading\_zeros(u, v)  
 n = length(u\_padded)  
 w = zeros(Int, n)  
 k = 0  
  
 for j in n:-1:1  
 diff = u\_padded[j] - v\_padded[j] + k   
 if diff < 0  
 diff += b  
 k = -1  
 else  
 k = 0  
 end  
 w[j] = diff   
 end  
 return remove\_leading\_zeros(w)  
end

subtract (generic function with 1 method)

**Пример вычитания**

u\_num = "987654321098765"  
v\_num = "123456789012345"  
b = 10  
u = number\_to\_digits\_func(u\_num)  
v = number\_to\_digits\_func(v\_num)  
diff\_result = subtract(u, v, b)  
diff\_str = digits\_to\_number\_func(sum\_result)  
println("u = $u\_num")  
println("v = $v\_num")  
println("w = $sum\_str")

u = 987654321098765  
v = 123456789012345  
w = 1111111110111110

**Алгоритм 3 (умножение неотрицательных целых чисел столбиком)**

function multiply\_long(u::Vector{Int}, v::Vector{Int}, b::Int)::Vector{Int}  
 n = length(u)  
 m = length(v)  
 w = zeros(Int, n + m)  
  
 for j in m:-1:1  
 carry = 0  
 for i in n:-1:1  
 t = u[i] \* v[j] + w[i + j] + carry  
 w[i + j] = t % b  
 carry = div(t, b)  
 end  
 w[j] += carry  
 end  
  
 return remove\_leading\_zeros(w)  
end

multiply\_long (generic function with 1 method)

**Пример умножения**

u\_num = "123456789012345"  
v\_num = "678901234567890"  
u = number\_to\_digits\_func(u\_num)  
v = number\_to\_digits\_func(v\_num)  
product\_long = multiply\_long(u, v, b)  
product\_long\_str = digits\_to\_number\_func(product\_long)  
println("u = $u\_num")  
println("v = $v\_num")  
println("w = $product\_long\_str")

u = 123456789012345  
v = 678901234567890  
w = 83814966476268537645950602050

**Алгоритм 4 (быстрый столбик)**

function multiply\_fast(u::Vector{Int}, v::Vector{Int}, b::Int)::Vector{Int}  
 n = length(u)  
 m = length(v)  
 w = zeros(Int, n + m)  
  
 for s in (n + m):-1:1  
 t = 0  
 for t\_idx in 1:n  
 v\_idx = s - t\_idx + 1  
 if v\_idx >= 1 && v\_idx <= m  
 t += u[n - t\_idx + 1] \* v[m - v\_idx + 1]  
 end  
 end  
 w[s] += t  
 carry = div(w[s], b)  
 w[s] = mod(w[s], b)  
 if s > 1  
 w[s - 1] += carry  
 end  
 end  
  
 return remove\_leading\_zeros(w)  
end

multiply\_fast (generic function with 1 method)

**Пример быстрого умножения**

product\_fast = multiply\_fast(u, v, b)  
product\_fast\_str = digits\_to\_number\_func(product\_fast)  
println("u = $u\_num")  
println("v = $v\_num")  
println("w = $product\_fast\_str")  
println()

u = 123456789012345  
v = 678901234567890  
w = 536504679225115307111781741960

**Алгоритм 5 (деление многоразрядных целых чисел)**

function divide(u::Vector{Int}, v::Vector{Int}, b::Int)::Tuple{Vector{Int}, Vector{Int}}  
 if compare(v, [0]) == 0  
 error("Деление на ноль невозможно.")  
 end  
  
 if compare(u, v) < 0  
 return ([0], copy(u))  
 end  
  
 n = length(u)  
 t\_len = length(v)  
 q = zeros(Int, n - t\_len + 1)  
 u\_copy = copy(u)  
  
 for i in 1:(n - t\_len + 1)  
 start = i  
 end\_idx = i + t\_len - 1  
 if end\_idx > length(u\_copy)  
 current = [0]  
 else  
 current = u\_copy[start:end\_idx]  
 end  
 current = remove\_leading\_zeros(current)  
 if length(current) < t\_len  
 current = vcat(zeros(Int, t\_len - length(current)), current)  
 end  
  
 if current[1] < v[1]  
 q\_digit = 0  
 else  
 q\_digit = div(current[1], v[1])  
 q\_digit = min(q\_digit, b - 1)  
 end  
  
 while true  
 v\_times\_q = multiply\_long(v, [q\_digit], b)  
 v\_times\_q = vcat(v\_times\_q, zeros(Int, length(current) - length(v\_times\_q)))  
 cmp = compare(v\_times\_q, current)  
 if cmp > 0  
 q\_digit -= 1  
 else  
 break  
 end  
 if q\_digit == 0  
 break  
 end  
 end  
  
 v\_times\_q = multiply\_long(v, [q\_digit], b)  
 v\_times\_q = vcat(v\_times\_q, zeros(Int, length(current) - length(v\_times\_q)))  
 current = subtract(current, v\_times\_q, b)  
  
 u\_copy[start:end\_idx] = current  
 q[i] = q\_digit  
 end  
  
 q = remove\_leading\_zeros(q)  
 r = remove\_leading\_zeros(u\_copy)  
 return (q, r)  
end

divide (generic function with 1 method)

**Пример деления**

u\_num = "1234567890123456789012345"  
v\_num = "123456789012345"  
u = number\_to\_digits\_func(u\_num)  
v = number\_to\_digits\_func(v\_num)  
quotient, remainder = divide(u, v, b)  
quotient\_str = digits\_to\_number\_func(quotient)  
remainder\_str = digits\_to\_number\_func(remainder)  
println("u = $u\_num")  
println("v = $v\_num")  
println("q = $quotient\_str")  
println("r = $remainder\_str")

### 5. Выводы

В ходе данной лабораторной работы было рассмотрено 5 алгоритмов, обспечивающих более высокую производительность машинного сложения, вычитание, умножения и деления больших чисел. Данные алгоритмы были программно реализованы на языке ***Julia***. Данные программы могут быть использованы для работы с числами любой счетной системы (в частности десятичной) и превышающими размер стандартных типов данных.

### 6. Список литературы

1. Knuth, D. E. The Art of Computer Programming, Volume 2: Seminumerical Algorithms. Addison-Wesley, 1997.
2. Sedgewick, R., Wayne, K. Algorithms. Addison-Wesley, 2011.
3. Liu, Y. Chinese Remainder Theorem and Its Applications. Springer, 2008.
4. Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. Introduction to Algorithms. MIT Press, 2009.
5. McIlroy, M. D. Implementing the Chinese Remainder Theorem, 1981.
6. Буряков, В. Г. Алгоритмы и структуры данных. М.: Наука, 2004.
7. Julia Documentation. https://docs.julialang.org/