

Лабораторная работа №8

Тема: Целочисленная арифметика многократной точности

Выполнила: Исламова Сания Маратовна

Группа: НПИМд-01-24

Студ.билет: 1132249576

Задача лабораторной работы:

Реализовать рассмотренные алгоритмы программно: Сложение неотрицательных целых чисел, Вычитание неотрицательных целых чисел, Умножение неотрицательных целых чисел столбиком, Быстрый столбик, Деление многоразрядных целых числе

Описание хода выполнения лабораторной работы:

```
# Функция преобразования массива цифр в строку в системе счисления b
function digits_to_str(dig::Vector{Int}, b::Int)::String
    # Если массив пустой или состоит только из нулей — число равно 0
    if isempty(dig) || all(==(0), dig)
        return "0"
    end
    # Собираем строку из цифр, начиная со старшего разряда (reverse)
    # Если цифра ≥10, преобразуем в букву A–Z (для оснований >10)
    join( [d < 10 ? string(d) : string(Char('A' + d - 10)) for d in reverse(dig)], "")
    # Результат: строка, представляющая число в основании b, без ведущих нулей
end

# Функция преобразования строки в массив цифр (младший разряд — индекс 1)
function str_to_digits(s::String, b::Int)::Vector{Int}
    # Убираем пробелы и приводим к нижнему регистру для удобства
    s = strip(lowercase(s))
    # Пустая строка или "0" → представляем как [0]
    if s == "" || s == "0" return [0] end
    digits = Int[] # Создаём пустой массив для цифр
    # Проходим по символам строки справа налево (младшие разряды первые)
    for c in reverse(s)
        # Преобразуем символ в цифру: 0–9 или A–Z → 10–35
        d = isdigit(c) ? c - '0' : (uppercase(c) - 'A' + 10)
        # Проверяем, что цифра допустима в данном основании
        if d < 0 || d >= b
            error("Недопустимая цифра '$c' в основании $b")
        end
        push!(digits, d) # Добавляем цифру в массив (младшая первая)
    end
    # Удаляем ведущие нули (кроме случая, когда число 0)
    while length(digits) > 1 && digits[end] == 0
        pop!(digits)
    end
    digits # Результат : массив цифр , младший разряд — digits[1]
end

# Удаление ведущих нулей из массива цифр
function trim(dig::Vector{Int})::Vector{Int}
    # Пока больше одного разряда и старший разряд нулевой — удаляем его
    while length(dig) > 1 && dig[end] == 0
        pop!(dig)
    end
    dig # Результат: массив без ведущих нулей (кроме [0] для нуля)
end

# Алгоритм 1: Сложение неотрицательных чисел (по лабораторной)
```

```

function add_big(u::Vector{Int}, v::Vector{Int}, b::Int)::Vector{Int}
    n = max(length(u), length(v)) # Определяем максимальную разрядность
    w = zeros{Int, n + 1} # Результирующий массив на один разряд больше (для переноса)
    k = 0 # Перенос, изначально 0
    for j = 1:n # Проходим по разрядам от младшего к старшему
        uj = j <= length(u) ? u[j] : 0 # Берем цифру из u или 0, если разряд кончился
        vj = j <= length(v) ? v[j] : 0 # Аналогично для v
        s = uj + vj + k # Сумма цифр + перенос
        w[j] = s % b # Записываем младшую часть суммы в текущий разряд
        k = s ÷ b # Новый перенос в старший разряд
    end
    w[n+1] = k # Записываем финальный перенос (w[n+1] в алгоритме)
    trim(w) # Убираем ведущие нули
    # Результат: сумма u + v в основании b
end

```

Алгоритм 2: Вычитание $u - v$ ($u \geq v \geq 0$)

```

function sub_big(u::Vector{Int}, v::Vector{Int}, b::Int)::Vector{Int}
    n = length(u) # Разрядность берём по первому числу ( $u \geq v$ )
    w = zeros{Int, n} # Результирующий массив
    k = 0 # Заём из старшего разряда, изначально 0
    for j = 1:n
        uj = u[j] # Цифра из уменьшаемого
        vj = j <= length(v) ? v[j] : 0 # Цифра из вычитаемого или 0
        s = uj - vj - k # Разность с учётом займа
        if s < 0 # Если получилась отрицательная цифра
            s += b # Занимаем из старшего разряда (добавляем основание)
            k = 1 # Устанавливаем заём для следующего разряда
        else
            k = 0 # Заём не нужен
        end
        w[j] = s # Записываем цифру результата
    end
    trim(w) # Убираем ведущие нули
    # Результат: разность u - v в основании b
end

```

Алгоритм 3: Умножение "столбиком"

```

function mul_classic(u::Vector{Int}, v::Vector{Int}, b::Int)::Vector{Int}
    n, m = length(u), length(v) # Разрядности множимого и множителя
    w = zeros{Int, n + m} # Результат до n+m разрядов
    for j = 1:m # По разрядам множителя v (от младшего)
        vj = v[j]
        if vj == 0 && continue # Если цифра 0 — пропускаем (оптимизация)
        k = 0 # Перенос для текущего "столбика"
        for i = 1:n # По разрядам множимого u
            t = u[i] * vj + w[i+j-1] + k # Произведение + уже накопленное + перенос
            w[i+j-1] = t % b # Записываем в текущий разряд
            k = t ÷ b # Перенос в следующий разряд
        end
        w[n+j] = k # Записываем оставшийся перенос
    end
    trim(w) # Убираем ведущие нули
    # Результат: произведение u × v в основании b
end

```

Алгоритм 4: Умножение "быстрым столбиком"

```

function mul_fast(u::Vector{Int}, v::Vector{Int}, b::Int)::Vector{Int}
    n, m = length(u), length(v)

```

```

w = zeros(Int, n + m)      # Результат до n+m разрядов
t = 0                      # Накопитель промежуточной суммы
for s = 0:n+m-2            # s — диагональ в "столбике"
    low = max(0, s - m + 1) # Нижняя граница индекса i
    high = min(s, n - 1)    # Верхняя граница индекса i
    for i = low:high        # Суммируем все произведения на этой диагонали
        t += u[i+1] * v[s-i+1]
    end
    w[n+m-s-1] = t % b      # Записываем цифру в соответствующий разряд
    t ÷= b                  # Переносим остаток в следующую диагональ
end
w[1] = t                   # Последний перенос в старший разряд
trim(w)

# Результат: произведение u × v (тот же, что и в mul_classic, но другой алгоритм)
end

```

Алгоритм 5: Деление с остатком

```

function div_big(u_::Vector{Int}, v_::Vector{Int}, b::Int)
    u = copy(u_)            # Копируем, чтобы не изменять оригинал
    v = trim(copy(v_))      # Копируем и убираем ведущие нули у делителя
    n, t = length(u), length(v) # Разрядности делимого и делителя
    if t == 0 || all==(0), v) # Проверка деления на ноль
        error("Деление на ноль")
    end
    q = zeros(Int, n - t + 1) # Массив для частного (максимальная длина)
    for i = n:-1:t+1          # По разрядам делимого от старшего
        hi = i <= n ? u[i] : 0 # Текущий старший разряд
        mi = i-1 >= 1 ? u[i-1] : 0 # Следующий разряд
        lo = i-2 >= 1 ? u[i-2] : 0 # Ещё один для точной оценки
        # Оценка цифры частного
        qhat = hi >= v[t] ? b - 1 : (hi * b + mi) ÷ v[t]
        v1 = v[t] * b + (t >= 2 ? v[t-1] : 0) # Для проверки переполнения
        # Корректируем оценку вниз, если слишком большая
        while qhat > 0 && qhat * v1 > hi * b + mi * b + lo
            qhat -= 1
        end
        borrow = 0           # Заём при вычитании
        for j = 1:t          # Вычитаем qhat × v × b^(i-t)
            pos = i - t + j
            if pos > length(u) # Если нужно — расширяем массив u
                resize!(u, pos)
            end
            u[pos] = 0
            temp = qhat * v[j] + borrow
            u[pos] -= temp % b
            borrow = temp ÷ b
            if u[pos] < 0      # Если отрицательно — занимаем
                u[pos] += b
                borrow += 1
            end
        end
        pos_carry = i + 1
        if pos_carry <= length(u)
            u[pos_carry] -= borrow # Вычитаем заём из старшего разряда
            if u[pos_carry] < 0    # Если переполнение — корректируем
                u[pos_carry] += b
            end
            qhat -= 1
        end
    end
end

```

```

end
    q[i - t] = qhat      # Записываем цифру частного
end

r = length(u) >= t ? u[1:t] : u # Остаток — младшие t разрядов
trim(q), trim(r)          # Убираем ведущие нули

# Результат: кортеж (частное, остаток)
end

#
#
println ( "Лабораторная работа №8: Арифметика многократной точности")
println ( "0 — Выход")
println ( "1 — Сложение")
println ( "2 — Вычитание")
println ( "3 — Умножение столбиком")
println ( "4 — Умножение быстрым столбиком")
println ( "5 — Деление с остатком\n")

#
# Бесконечный цикл — программа работает, пока пользователь не выберет выход
while true
    print("Выберите алгоритм (0 для выхода): ")
    input = strip(readline())      # Считываем и убираем лишние пробелы
    input == "0" && (println("До свидания!"); break) # Выход по 0

    # Преобразуем ввод в число, если ошибка — сообщаем и продолжаем цикл
    alg = try parse{Int, input} catch
        println("Неверный выбор\n"); continue
    end

    if !(1 <= alg <= 5)             # Проверяем диапазон
        println("Выберите от 1 до 5\n"); continue
    end

    #
    print("Основание b (2–36): ")
    b = try parse{Int, readline()} catch # Считываем основание
        println("Неверное основание\n"); continue
    end

    if !(2 <= b <= 36)             # Проверяем допустимость
        println("b должно быть от 2 до 36\n"); continue
    end

    #
    print("Первое число (в системе $b): ")
    s1 = readline()                # Ввод первого числа как строки
    print("Второе число (в системе $b): ")
    s2 = readline()                # Ввод второго числа

    #
    try
        # Преобразуем строки в массивы цифр
        u = str_to_digits(s1, b)
        v = str_to_digits(s2, b)

        # Выполняем выбранный алгоритм
        if alg == 1
            res = add_big(u, v, b)
            println("Сумма : $(digits_to_str(res, b))\n")
        elseif alg == 2
            # Проверка условия u ≥ v для вычитания
            if length(u) < length(v) || (length(u) == length(v) && u < v)
                println("Ошибка: первое число должно быть ≥ второго\n")
            else

```

```
res = sub_big(u, v, b)

println("Разность : $(digits_to_str(res, b))\n")

end

elseif alg == 3

res = mul_classic(u, v, b)

println("Произведение (столбиком ): $(digits_to_str(res, b))\n")

elseif alg == 4

res = mul_fast(u, v, b)

println("Произведение (быстро ): $(digits_to_str(res, b))\n")

elseif alg == 5

q, r = div_big(u, v, b)

println("Частное : $(digits_to_str(q, b))")

println("Остаток : $(digits_to_str(r, b))\n")

end

catch e

# Ловим все ошибки ввода (недопустимые цифры и т.д.)

println ( "Ошибка : $(sprintf(showerror, e))\n")

end

end
```

Результат реализации рассмотренного алгоритма

```
1 # Лабораторная работа №8
2 # Тема: Целочисленная арифметика многократной точности
3 # Выполнила: Исламова Сания Маратовна
4 # Группа: ИВМд-01-24
5 |
6
7 # Функция преобразования массива цифр в строку в системе счисления b
8 function digits_to_str(dig::Vector{Int}, b::Int)::String
9     # Если массив пустой или состоит только из нулей – число равно 0
10    if isempty(dig) || all(==(0), dig)
11        return "0"
12    end
13    # Собираем строку из цифр, начиная со старшего разряда (reverse)
14    # Если цифра >10, преобразуем в букву A-Z (для оснований >10)
15    join([d < 10 ? string(d) : string(Char('A' + d - 10)) for d in reverse(dig)], "")
16    # Результат: строка, представляющая число в основании b, без ведущих нулей
17 end
18
19 # Функция преобразования строки в массив цифр (младший разряд – индекс 1)
20 function str_to_digits(s::String, b::Int)::Vector{Int}
21     # Убираем пробелы и приводим к нижнему регистру для удобства
22     s = strip(lowercase(s))
23     # Пустая строка или "0" → представляем как [0]
24     if s == "" || s == "0" return [0] end
25     digits = Int[] # Создаём пустой массив для цифр
26     # Проходим по символам строки справа налево (младшие разряды первые)
27     for c in reverse(s)
28         # Преобразуем символ в цифру: 0-9 или A-Z → 10-35
29         d = isdigit(c) ? c - '0' : (uppercase(c) - 'A' + 10)
30         # Проверяем, что цифра допустима в данном основании
31         if d < 0 || d >= b
32             error("Недопустимая цифра '$c' в основании $b")
33         end
34         push!(digits, d) # Добавляем цифру в массив (младшая первая)
35     end
36     # Удаляем ведущие нули (кроме случая, когда число 0)
37     while length(digits) > 1 && digits[end] == 0
38         pop!(digits)
39     end
40     digits # Результат: массив цифр, младший разряд – digits[1]
41 end
42
43 # Удаляем ведущие нули из массива цифр
44 function trim(dig::Vector{Int})::Vector{Int}
45     # Пока больше одного разряда и старший разряд нулевой – удаляем его
```

```
44 function trim(dig::Vector{Int})::Vector{Int}
45     # Пока больше одного разряда и старший разряд нулевой – удаляем его
46     while length(dig) > 1 && dig[end] == 0
47         pop!(dig)
48     end
49     dig # Результат: массив без ведущих нулей (кроме [0] для нуля)
50 end
51
52 # Алгоритм 1: Сложение неотрицательных чисел (по лабораторной)
53 function add_big(u::Vector{Int}, v::Vector{Int}, b::Int)::Vector{Int}
54     n = max(length(u), length(v)) # Определим максимальную разрядность
55     w = zeros{Int, n + 1} # Результирующий массив на один разряд больше (для переноса)
56     k = 0 # Перенос, изначально 0
57     for j = 1:n
58         u_j = j <= length(u) ? u[j] : 0 # Проходим по разрядам от младшего к старшему
59         v_j = j <= length(v) ? v[j] : 0 # Берём цифру из u или 0, если разряд кончился
60         s = u_j + v_j + k # Аналогично для v
61         w[j] = s % b # Сумма цифр + перенос
62         k = s ÷ b # Записываем младшую часть суммы в текущий разряд
63     end # Новый перенос в старший разряд
64     w[n+1] = k # Записываем финальный перенос (m_0 в алгоритме)
65     trim(w) # Убираем ведущие нули
66     # Результат: сумма u + v в основании b
67 end
68
69 # Алгоритм 2: Вычитание u - v (u ≥ v ≥ 0)
70 function sub_big(u::Vector{Int}, v::Vector{Int}, b::Int)::Vector{Int}
71     n = length(u) # Разрядность берём по первому числу (u ≥ v)
72     w = zeros{Int, n} # Результирующий массив
73     k = 0 # Заём из старшего разряда, изначально 0
74     for j = 1:n
75         u_j = u[j] # Цифра из уменьшаемого
76         v_j = j <= length(v) ? v[j] : 0 # Цифра из вычитаемого или 0
77         s = u_j - v_j - k # Разность с учётом займа
78         if s < 0 # Если получились отрицательная цифра
79             s = s + b # Занимаем из старшего разряда (добавляем основание)
80             k = 1 # Устанавливаем заём для следующего разряда
81         else
82             k = 0 # Заём не нужен
83         end
84         w[j] = s # Записываем цифру результата
85     end
86     trim(w) # Убираем ведущие нули
87     # Результат: разность u - v в основании b
88 end
```

```

100 # Алгоритм 1: Умножение "столбиком"
101 function mul_classic(v:Vector[Int], v2:Vector[Int], b:Int)::Vector[Int]
102     n, m = length(v), length(v2) # Размеры векторов и модуль
103     w = zeros{Int, n + m} # Результирующий массив
104     for j = 1:m
105         v[j] == 0 && continue # Если цифра 0 - пропускаем (оптимизация)
106         k = 0 # Перенос для текущего "столбика"
107         for i = 1:n
108             k = v[i] * v2[j] + w[i+j] + k # Произведение + уже накопленное + перенос
109             w[i+j] = k % b # Записываем в текущий разряд
110             k = k ÷ b # Перенос в следующий разряд
111         end
112         w[n+j] = k # Записываем оставшийся перенос
113     end
114     trim(w) # Убираем ведущие нули
115     # Результат: произведение n * m с модулем b
116 end
117
118 # Алгоритм 4: Умножение "быстрым столбиком"
119 function mul_fast(u:Vector{Int}, v:Vector{Int}, b:Int)::Vector{Int}
120     n, m = length(u), length(v) # Размеры до тех разрядов
121     w = zeros{Int, n + m} # Результирующий массив
122     t = 0 # t = 0 - делитель, n * столбика
123     for s = 1:m÷2
124         low = min(n, s + 1) # Нижняя граница индекса l
125         high = min(m, m - 1) # Верхняя граница индекса r
126         for l = 1:low:high
127             t += u[l+s] * v[s-l+1] # Суммируем все произведения на этой диагонали
128         end
129         w[n+s-1] = t % b # Записываем цифру в соответствующий разряд
130         t = t ÷ b # Переносим остаток в следующий диапазон
131     end
132     w[n] = t # Последний перенос в старый разряд
133     trim(w) # Последний перенос в старый разряд
134     # Результат: произведение n * m (то же, что и в mul_classic, но другой алгоритм)
135 end

```

```

127 # Алгоритм 5: Деление с остатком
128 function div_big(u:Vector{Int}, v:Vector{Int}, b:Int)
129     n = copy(u) # Копируем, чтобы не изменить оригинал
130     w = trim(copy(n)) # Копируем и убираем ведущие нули у делителя
131     m = length(w) # Размеры делимого и делителя
132     if t == 0 || all(==(0), v) # Проверка деления на ноль
133         error("Деление на ноль.")
134     end
135     q = zeros{Int, n - t + 1} # Массив для частного (максимальная длина)
136     for i = m-1:t-1
137         hi = i < n ? u[i] : 0 # Текущий (старый) разряд
138         hi = hi * 10 + u[i+1] # Следующий разряд
139         lo = hi >= 1 ? u[i+2] : 0 # Ещё один для точной цифры
140         # Поиск оптимального частного
141         qhat = hi ÷ v[t] * b - 1; (hi * b + lo) ÷ v[t]
142         v[t] * b + (t > 2 ? v[t-1] : 0) # Для проверки перемножения
143         # Сокращаем разряд, если число слишком большое
144         while qhat > 0 && qhat * v[t] + hi * b^b + lo * b > hi
145             qhat -= 1
146         end
147         borrow = 0 # Если при вычитании
148         for j = t+1:n
149             pos = i - t + j # Вычитаем qhat * v + b^j(i-1)
150             if pos < length(u) # Если нужно - расширяем массив u
151                 realize(u, pos)
152             end
153             u[pos] -= qhat * v[j] + borrow
154             u[pos] = max(0, u[pos]) # Если отрицательное - занимаем
155             borrow = u[pos] < 0 ? -u[pos] : 0
156             u[pos] += borrow
157         end
158         pos = i - t + 1
159         if pos < length(u) # Если отрицательное - занимаем
160             u[pos] += borrow
161         end
162         pos = i - t + 1
163         if pos < length(u) # Если отрицательное - занимаем
164             u[pos] += borrow
165         end
166         if u[pos] < 0 # Если отрицательное - занимаем
167             u[pos] += b
168             borrow += 1
169         end
170     end
171     pos = i - t + 1
172     if pos < length(u) # Если отрицательное - занимаем
173         u[pos] += borrow
174     end
175     q[i - t] = qhat # Записываем цифру частного
176 end

```

```

128 # Алгоритм 6: Деление с остатком
129 function div_big(u:Vector{Int}, v:Vector{Int}, b:Int)
130     n = length(u)
131     m = length(v)
132     if m == 0 || all(==(0), v)
133         error("Деление на ноль.")
134     end
135     q = zeros{Int, n - m + 1}
136     for i = m-1:n-1
137         hi = i < n ? u[i] : 0
138         hi = hi * 10 + u[i+1]
139         lo = hi >= 1 ? u[i+2] : 0
140         # Поиск оптимального частного
141         qhat = hi ÷ v[t] * b - 1; (hi * b + lo) ÷ v[t]
142         v[t] * b + (t > 2 ? v[t-1] : 0)
143         # Сокращаем разряд, если число слишком большое
144         while qhat > 0 && qhat * v[t] + hi * b^b + lo * b > hi
145             qhat -= 1
146         end
147         borrow = 0
148         for j = t+1:n
149             pos = i - t + j
150             if pos < length(u)
151                 realize(u, pos)
152             end
153             u[pos] -= qhat * v[j] + borrow
154             u[pos] = max(0, u[pos])
155             borrow = u[pos] < 0 ? -u[pos] : 0
156             u[pos] += borrow
157         end
158         pos = i - t + 1
159         if pos < length(u)
160             u[pos] += borrow
161         end
162         pos = i - t + 1
163         if pos < length(u)
164             u[pos] += borrow
165         end
166         if u[pos] < 0
167             u[pos] += b
168             borrow += 1
169         end
170     end
171     pos = i - t + 1
172     if pos < length(u)
173         u[pos] += borrow
174     end
175     q[i - t] = qhat
176 end

```

```

128 # Алгоритм 7: Деление с остатком
129 function div_big(u:Vector{Int}, v:Vector{Int}, b:Int)
130     n = length(u)
131     m = length(v)
132     if m == 0 || all(==(0), v)
133         error("Деление на ноль.")
134     end
135     q = zeros{Int, n - m + 1}
136     for i = m-1:n-1
137         hi = i < n ? u[i] : 0
138         hi = hi * 10 + u[i+1]
139         lo = hi >= 1 ? u[i+2] : 0
140         # Поиск оптимального частного
141         qhat = hi ÷ v[t] * b - 1; (hi * b + lo) ÷ v[t]
142         v[t] * b + (t > 2 ? v[t-1] : 0)
143         # Сокращаем разряд, если число слишком большое
144         while qhat > 0 && qhat * v[t] + hi * b^b + lo * b > hi
145             qhat -= 1
146         end
147         borrow = 0
148         for j = t+1:n
149             pos = i - t + j
150             if pos < length(u)
151                 realize(u, pos)
152             end
153             u[pos] -= qhat * v[j] + borrow
154             u[pos] = max(0, u[pos])
155             borrow = u[pos] < 0 ? -u[pos] : 0
156             u[pos] += borrow
157         end
158         pos = i - t + 1
159         if pos < length(u)
160             u[pos] += borrow
161         end
162         pos = i - t + 1
163         if pos < length(u)
164             u[pos] += borrow
165         end
166         if u[pos] < 0
167             u[pos] += b
168             borrow += 1
169         end
170     end
171     pos = i - t + 1
172     if pos < length(u)
173         u[pos] += borrow
174     end
175     q[i - t] = qhat
176 end

```

```

128 # Алгоритм 8: Деление с остатком
129 function div_big(u:Vector{Int}, v:Vector{Int}, b:Int)
130     n = length(u)
131     m = length(v)
132     if m == 0 || all(==(0), v)
133         error("Деление на ноль.")
134     end
135     q = zeros{Int, n - m + 1}
136     for i = m-1:n-1
137         hi = i < n ? u[i] : 0
138         hi = hi * 10 + u[i+1]
139         lo = hi >= 1 ? u[i+2] : 0
140         # Поиск оптимального частного
141         qhat = hi ÷ v[t] * b - 1; (hi * b + lo) ÷ v[t]
142         v[t] * b + (t > 2 ? v[t-1] : 0)
143         # Сокращаем разряд, если число слишком большое
144         while qhat > 0 && qhat * v[t] + hi * b^b + lo * b > hi
145             qhat -= 1
146         end
147         borrow = 0
148         for j = t+1:n
149             pos = i - t + j
150             if pos < length(u)
151                 realize(u, pos)
152             end
153             u[pos] -= qhat * v[j] + borrow
154             u[pos] = max(0, u[pos])
155             borrow = u[pos] < 0 ? -u[pos] : 0
156             u[pos] += borrow
157         end
158         pos = i - t + 1
159         if pos < length(u)
160             u[pos] += borrow
161         end
162         pos = i - t + 1
163         if pos < length(u)
164             u[pos] += borrow
165         end
166         if u[pos] < 0
167             u[pos] += b
168             borrow += 1
169         end
170     end
171     pos = i - t + 1
172     if pos < length(u)
173         u[pos] += borrow
174     end
175     q[i - t] = qhat
176 end

```

```
Лабораторная работа №8: Арифметика многократной точности
0 – Выход
1 – Сложение
2 – Вычитание
3 – Умножение столбиком
4 – Умножение быстрой столбиком
5 – Деление с остатком

Julia> 1
1
Основание b (2-36): 6
Первое число (в системе 6): 4
Второе число (в системе 6): 2
Сумма: 10

Выберите алгоритм (0 для выхода): 2
Основание b (2-36): 29
Первое число (в системе 29): 23
Второе число (в системе 29): 17
Ошибка: первое число должно быть > второго

Выберите алгоритм (0 для выхода): 2
Основание b (2-36): 30
Первое число (в системе 30): 29
Второе число (в системе 30): 10
Разности: 19

Выберите алгоритм (0 для выхода): 3
Основание b (2-36): 33
Первое число (в системе 33): 32
Второе число (в системе 33): 19
Произведение (столбиком): 311

Выберите алгоритм (0 для выхода): 4
Основание b (2-36): 10
Первое число (в системе 10): 2
Второе число (в системе 10): 8
Произведение (быстро): 1

Выберите алгоритм (0 для выхода): 5
Основание b (2-36): 16
Первое число (в системе 16): 15
Второе число (в системе 16): 12
Частное: 0
Остаток: 15

Выберите алгоритм (0 для выхода): 0
До свидания!
```