

Лабораторная работа №6

Тема: Разложение чисел на множители

Выполнила: Исламова Сания Маратовна

Группа: НПИмд-01-24

Студ.билет: 1132249576

Задача лабораторной работы:

Реализовать рассмотренный алгоритм программно: Алгоритм, реализующий р-метод Полларда

Описание хода выполнения лабораторной работы:

```
print("n = "); n = parse(BigInt, readline())          # 1
# Выводим приглашение "n = " и сразу считываем введённое число как BigInt.
# BigInt нужен, потому что в задании числа могут быть очень большими (сотни
# цифр),
# а обычный Int переполнится. readline() читает строку из терминала.

let                                         # 2
# Создаём локальный блок let – это важно!
# Благодаря let все переменные внутри (a, b, i, f) будут локальными,
# и Julia не будет ругаться на «global variable» и не выдаст UndefVarError.
# Это самый чистый и правильный способ в скрипте.

a = b = 1                                     # 3
# Согласно лабораторной (стр. 25): «Положить a ← c, b ← c», a c = 1.
# Поэтому оба указателя («черепаха» a и «заяц» b) стартуют с значения 1.

i = 1                                         # 4
# Счётчик итераций. Начинаем с 1, потому что в таблице из методички
# первая строка после заголовка – это i = 2 (уже после первого шага).

f(x) = (x*x + 5) % n                         # 5
# Определяем полиномиальную функцию  $f(x) = x^2 + 5 \pmod{n}$ .
# Именно +5 требует методичка (пример на стр. 25).
# % n – это взятие остатка по модулю n, чтобы числа не росли бесконечно.

println(" i\t a\t b\t d")                      # 6
# Печатаем шапку таблицы точно так же, как в лабораторной.
# \t – табуляция для выравнивания столбцов.

while true                                      # 7
# Запускаем бесконечный цикл – будем выходить из него вручную через break,
# когда найдём нетривиальный делитель.
```

```

a = f(a); b = f(f(b)); i += 1           # 8
# Один шаг алгоритма Полларда:
# • «черепаха» a делает один шаг: a ← f(a)
# • «заяц» b делает два шага: b ← f(f(b))
# # увеличиваем счётчик итераций

d = gcd(abs(a - b), n)                  # 9
# Вычисляем НОД от |a-b| и n – это ключевая идея метода Полларда.
# Если последовательности зациклятся в каком-то подмодуле,
# то |a-b| будет кратно одному из простых делителей n.

println("$i\t$a\t$b\t$d")               # 10
# Печатаем текущую строку таблицы: номер итерации, значения a, b и d.
# Интерполяция $ позволяет подставить значения переменных прямо в
строку.

if 1 < d < n                          # 11
    println("\nНетривиальный делитель: $d и $(n ÷ d)")
    # Если найден нетривиальный делитель (не 1 и не всё n),
    # выводим результат и завершаем работу.
    # n ÷ d – это целочисленное деление (в Julia ÷ = \div + TAB)

break                                    # 12
# Выходим из цикла – задача решена.

end
end
# Конец блока Let – все локальные переменные автоматически уничтожаются.

```

Результат реализации рассмотренного алгоритма

```

Файл Правка Выделение Вид Переход ...
C:\Users>let&gt; Download & Lab03.jl... 
1 # Алгоритмическая работа №9
2 # Вычисление делителей чисел на множестве
3 # Бинарного Истолка Семёна Ивановича
4 # Группа: ИМ-01-24
5
6 println("n: "); n = parse(Int64, readline())          # 1
# Вводим приведение "n" к типу сразу считывая введенное число как BigInt.
7 # Алгоритм работает потому что в дальнейшем мы будем брать остатки по модулю n.
8 # в дальнейшем readline() читает строку из терминала.
9
10 let
11     # Создаём локальный блок let – это важно!
12     # Алгоритм let все переменные внутри (a, b, i, f) будут локальными,
13     # а не глобальными, поэтому результат работы не будет влиять на терминал.
14     # Это самая частая ошибка в программировании – не всегда блок let помогает.
15
16     a = b = 1          # 2
17     # Создаём лабораторный (стр. 25) локальную a = c, b = c+, a <= 1.
18     # Поэтому оба указателя (=черепаха a и =заяц b) стартуют с значением 1.
19
20     i = 1            # 3
21     # Считаем итерации. Начинаем с 1, потому что в таблице из логарифма
22     # первая строка после заголовка – это 1 = 1 = 1 (также после первого шага).
23
24     f(x) = (x*x + 5) % n      # 4
25     # Рассчитываем квадратичную функцию f(x) = x*x + 5 (mod n).
26     # Вложенный if требует метасинтаксис (пример из стр. 23).
27     # И n – это вспомогательный модуль n, чтобы числа не росли бесконечно.
28
29     while true          # 5
30         # Внешний цикл бесконечный пока – будем выходит из него прорыв через break,
31         # когда найдем нетривиальный делитель.
32
33         a = f(x); b = f(f(x)); i += 1          # 6
34         # Шаги алгоритма Полларда:
35         # a – черепаха, b делает один шаг: b ← f(a)
36         # i – счетчик итераций
37         # f – квадратичная функция f(x) = x*x + 5 (mod n).
38         # x – увеличение счётчик итераций
39
40         d = gcd(a, b); n                         # 7
41         # Вычисление НОД от |a-b| и n – это ключевая идея метода Полларда.
42         # Если последовательности зациклились в каком-то подмодуле,
43         # то |a-b| будет кратно одному из простых делителей n.
44
45         println("$i\t$a\t$b\t$d")               # 8
46         # Печатаем текущую строку таблицы: номер итерации, значения a, b и d.
47         # Интерполяция $ позволяет подставить значения переменных прямо в строку.
48
49         if a < d & d < n                         # 9
50             println("Нетривиальный делитель: $d и $(n ÷ d)") # 10
51             # Если найден нетривиальный делитель (не 1 и не всё n),
52             # выводим результат и завершаем работу.
53             # n ÷ d – это целочисленное деление (в Julia ÷ = \div + TAB)
54
55         break                                    # 12
56         # Выходим из цикла – задача решена.
57
58 end
59
60 end
61
62 # Конец блока Let – все локальные переменные автоматически уничтожаются.

```

© 9 □ 9 Julia REPL v1.11 Окно 64 мониторов 12 Процессор: 4 УП-8 ОЗУ 1 Гб 8 Мб 0

The screenshot shows a Julia REPL session within a Visual Studio Code (VS Code) interface. The title bar indicates the window is titled "Julia REPL (v1.11.6)". The code editor area contains the following Julia code and its execution results:

```
julia> 1359331
1359331
1   a      b      d
2   6      41     1
3   41    123939  1
4   1686   391594 1
5   123939  438157 1
i   a      b      d
2   6      41     1
3   41    123939  1
4   1686   391594 1
5   123939  438157 1
6   123939  438157 1
7   391594  1144026 1
8   1090062   885749  1181
Нетривиальный делитель: 1181 и 1151
julia> 1359331
julia>
```

The output shows the result of the division of 1359331 by 1181, which yields a quotient of 1151 and a remainder of 1. The code editor also displays the status bar at the bottom with "Julia env v1.11".