Цель работы

Познакомиться с дискретным логарифмированием в конечном поле.

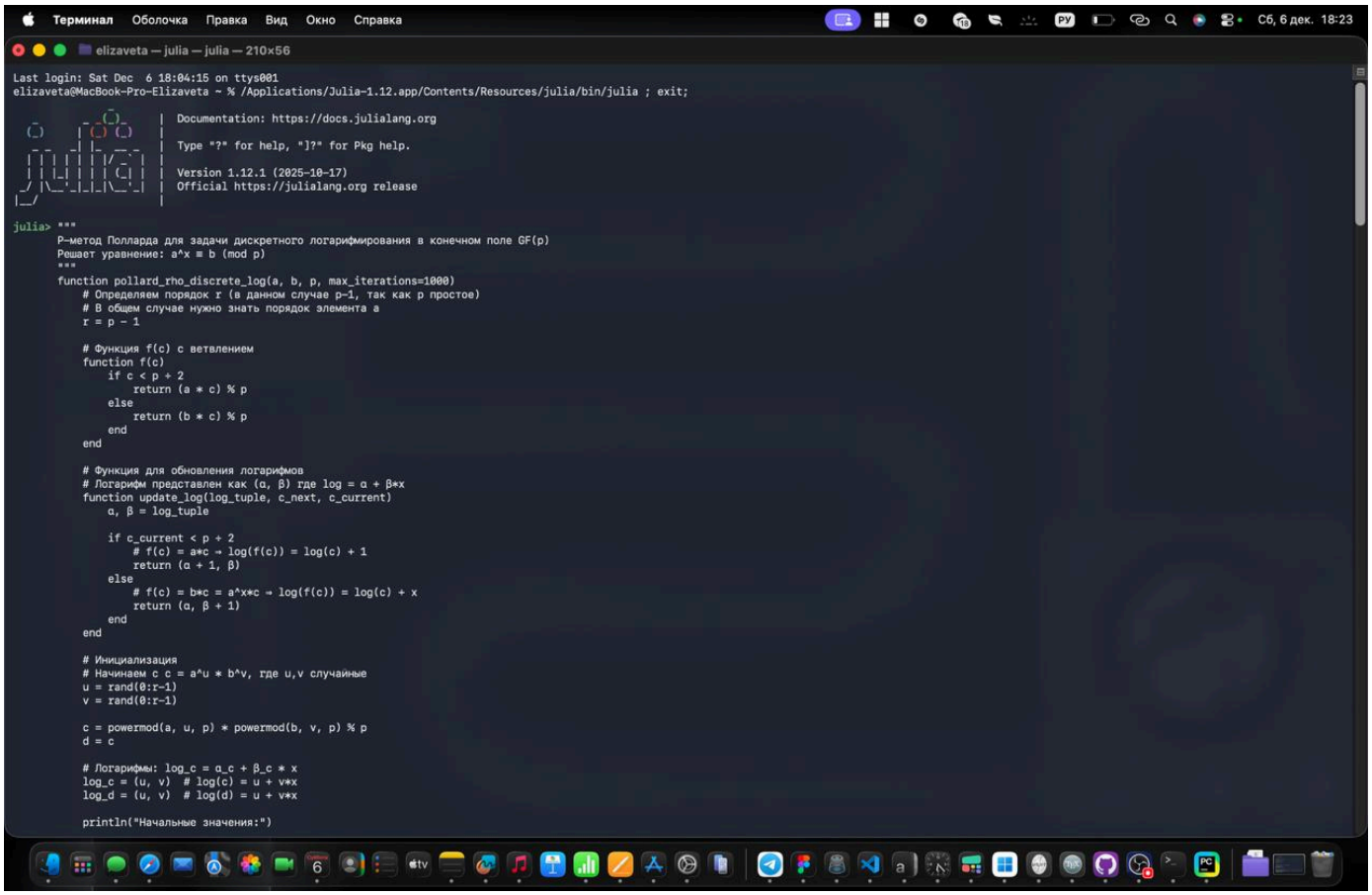
Задание

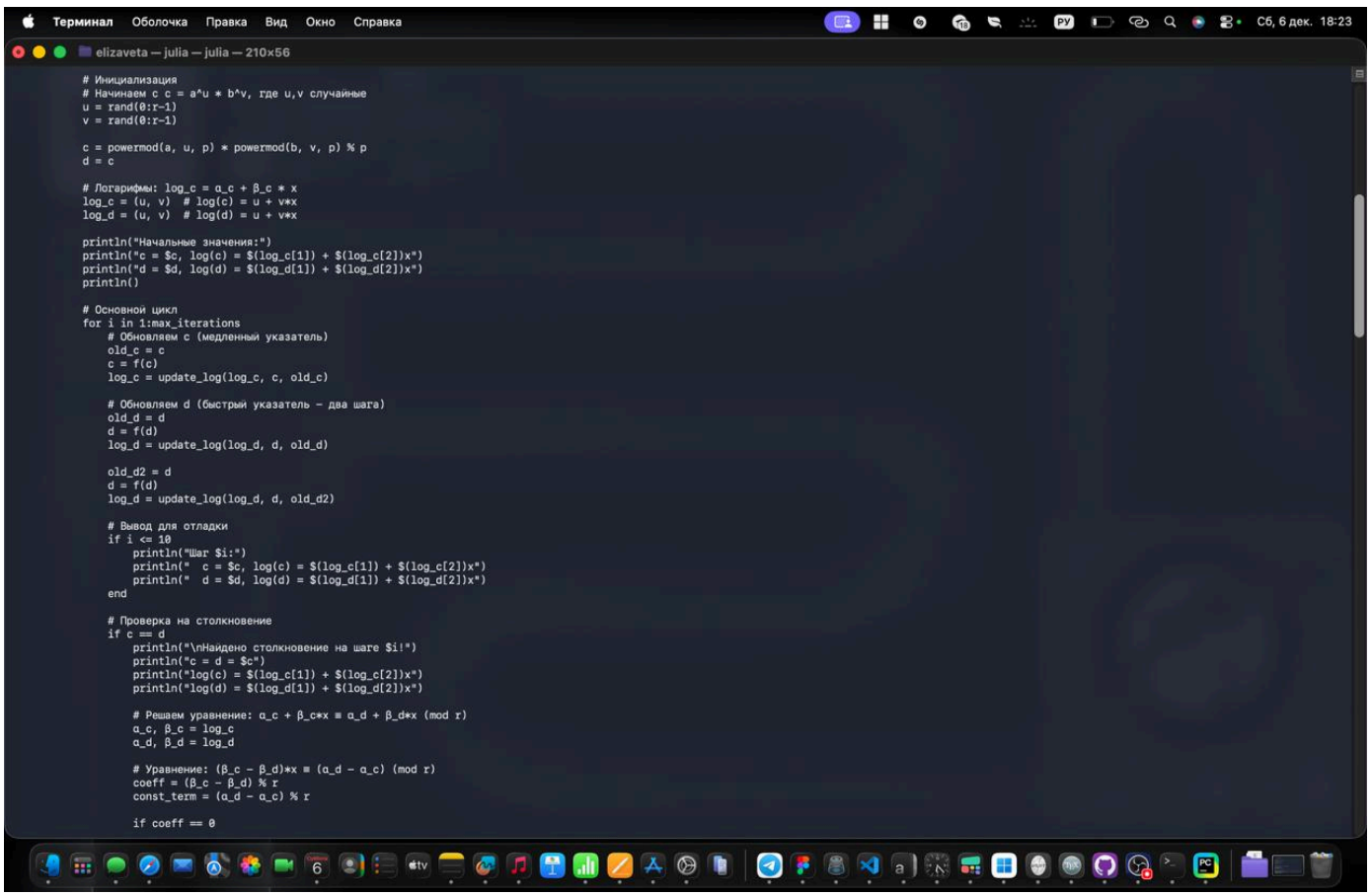
Реализовать алгоритм, реализующий p-метод Полларда.

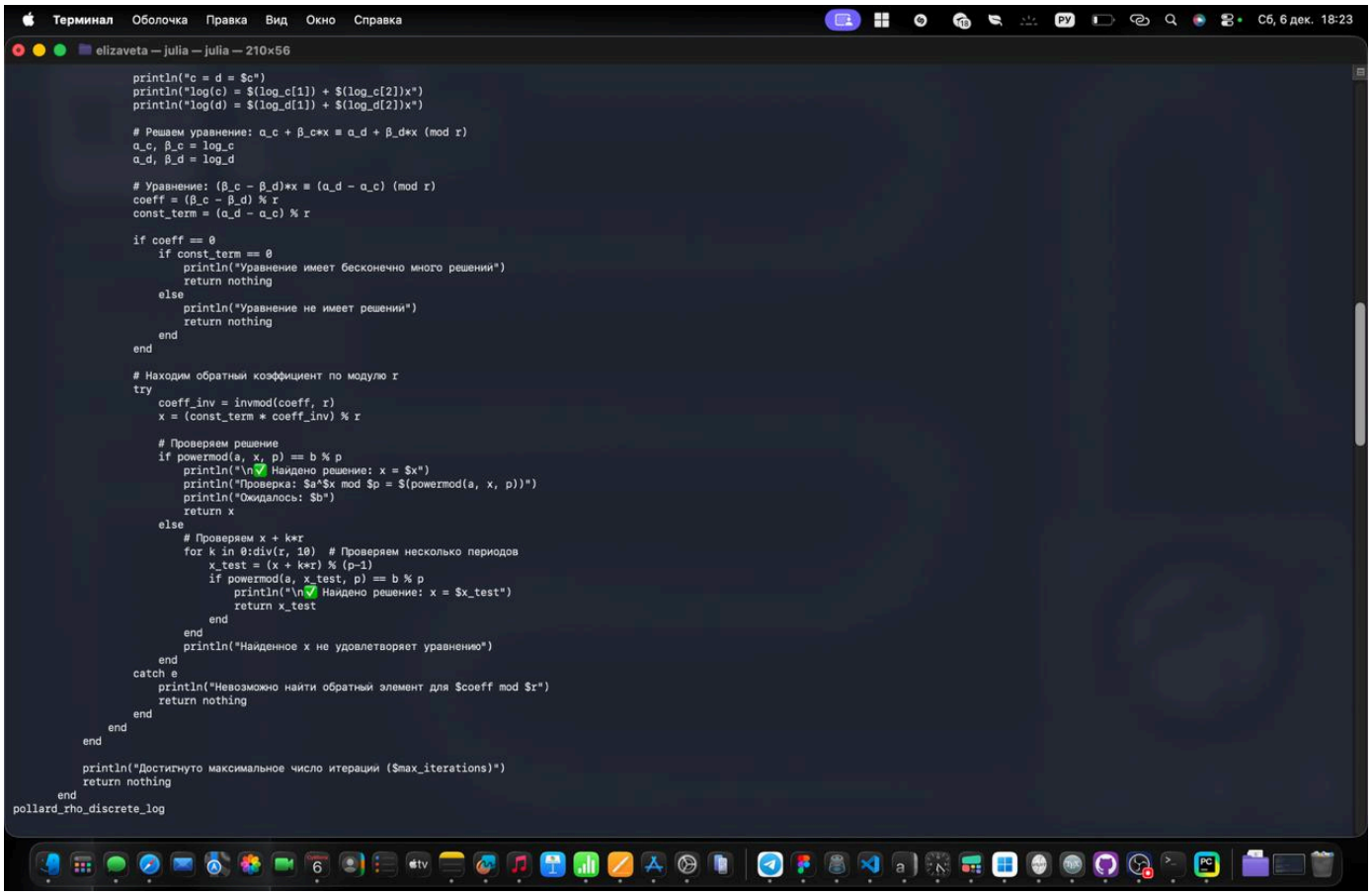
Выполнение лабораторной работы

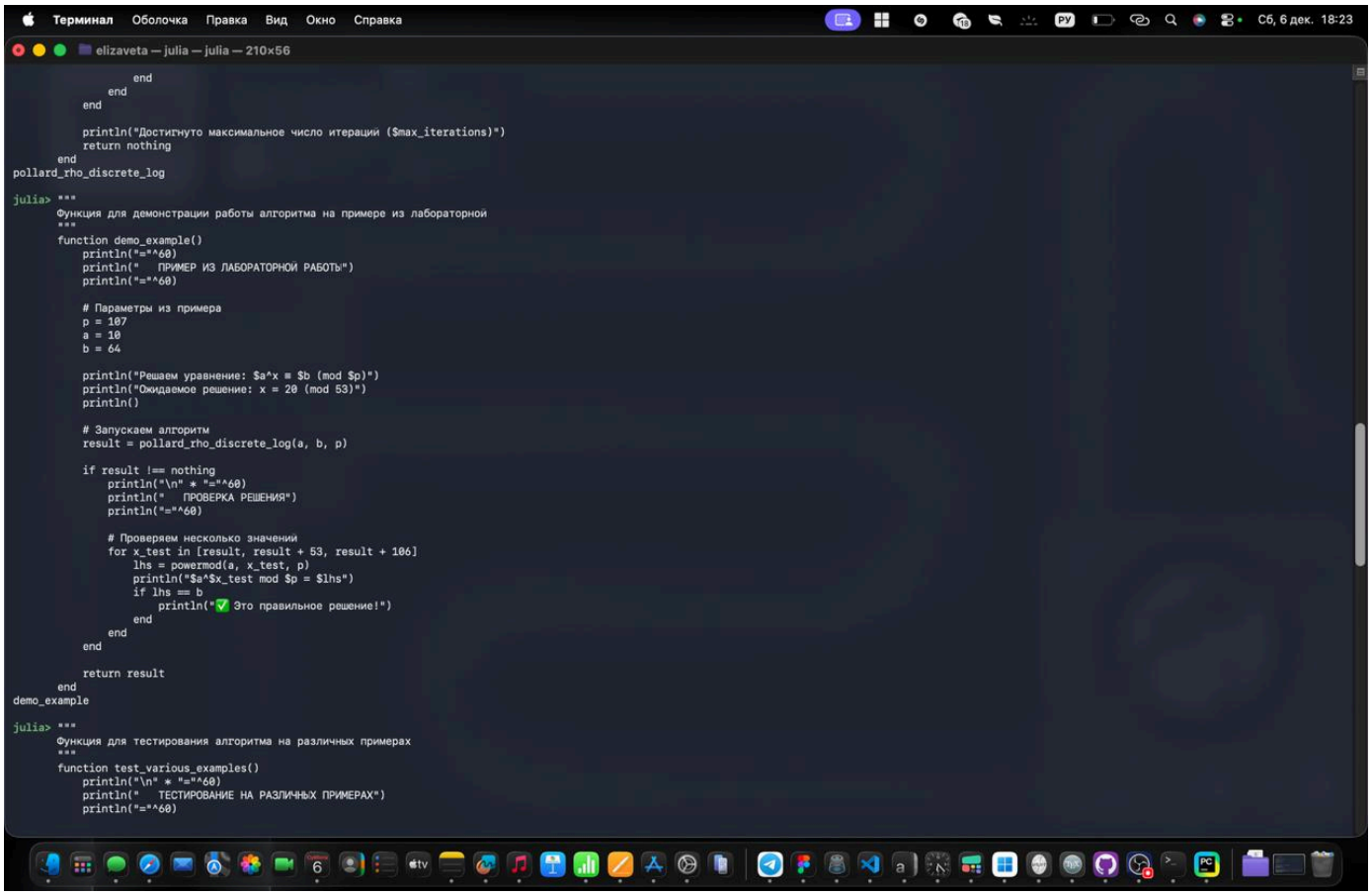
Данная работа была выполнена на языке Julia.

Для реализации p-метода Полларда была написана следующая программа (рис. [-@fig:001]) (рис. [-@fig:002]):


Терминал Оболочка Правка Вид Окно Справка
Сб, 6 дек. 18:23
elizaveta — julia — julia — 210×56
Last login: Sat Dec 6 18:04:15 on ttys001
elizaveta@MacBook-Pro-Elizaveta ~ % /Applications/Julia-1.12.app/Contents/Resources/julia/bin/julia ; exit;
Documentation: https://docs.julialang.org
Type "?" for help, "]?" for Pkg help.
Version 1.12.1 (2025-10-17)
Official https://julialang.org release
julia> """
P-метод Полларда для задачи дискретного логарифмирования в конечном поле GF(p)
Решает уравнение: a^x ≡ b (mod p)
"""
function pollard_rho_discrete_log(a, b, p, max_iterations=1000)
# Определяем порядок r (в данном случае p-1, так как p простое)
# В общем случае нужно знать порядок элемента a
r = p - 1
# Функция f(c) с ветвлением
function f(c)
if c < p ÷ 2
return (a * c) % p
else
return (b * c) % p
end
end
Функция для обновления логарифмов
Логарифм представлен как (α, β) где log = α + β*x
function update_log(log_tuple, c_next, c_current)
α, β = log_tuple
if c_current < p ÷ 2
f(c) = a*c → log(f(c)) = log(c) + 1
return (α + 1, β)
else
# f(c) = b*c = a^x*c → log(f(c)) = log(c) + x
return (α, β + 1)
end
end
# Инициализация
# Начинаем с c = a^u * b^v, где u,v случайные
u = rand(0:r-1)
v = rand(0:r-1)
c = powermod(a, u, p) * powermod(b, v, p) % p
d = c
# Логарифмы: log_c = α_c + β_c * x
log_c = (u, v) # log(c) = u + v*x
log_d = (u, v) # log(d) = u + v*x
println("Начальные значения:")


Терминал Оболочка Правка Вид Окно Справка
Сб, 6 дек. 18:23
elizaveta — julia — julia — 210×56
# Инициализация
# Начинаем с c = a^u * b^v, где u,v случайные
u = rand(0:r-1)
v = rand(0:r-1)
c = powermod(a, u, p) * powermod(b, v, p) % p
d = c
# Логарифмы: log_c = α_c + β_c * x
log_c = (u, v) # log(c) = u + v*x
log_d = (u, v) # log(d) = u + v*x
println("Начальные значения:")
println("c = $c, log(c) = $(log_c[1]) + $(log_c[2])x")
println("d = $d, log(d) = $(log_d[1]) + $(log_d[2])x")
println()
# Основной цикл
for i in 1:max_iterations
# Обновляем c (медленный указатель)
old_c = c
c = f(c)
log_c = update_log(log_c, c, old_c)
# Обновляем d (быстрый указатель - два шага)
old_d = d
d = f(d)
log_d = update_log(log_d, d, old_d)
old_d2 = d
d = f(d)
log_d = update_log(log_d, d, old_d2)
# Вывод для отладки
if i <= 10
println("Шаг $i:")
println(" c = $c, log(c) = $(log_c[1]) + $(log_c[2])x")
println(" d = $d, log(d) = $(log_d[1]) + $(log_d[2])x")
end
Проверка на столкновение
if c == d
println("\nНайдено столкновение на шаге $i!")
println("c = d = $c")
println("log(c) = $(log_c[1]) + $(log_c[2])x")
println("log(d) = $(log_d[1]) + $(log_d[2])x")
# Решаем уравнение: α_c + β_c*x ≡ α_d + β_d*x (mod r)
α_c, β_c = log_c
α_d, β_d = log_d
# Уравнение: (β_c - β_d)*x ≡ (α_d - α_c) (mod r)
coeff = (β_c - β_d) % r
const_term = (α_d - α_c) % r
if coeff == 0





В данной программе:

1 строка: подключение библиотеки для нахождения НОД

3: задание функции

4-16: задание внутренней функции для вывода результатов

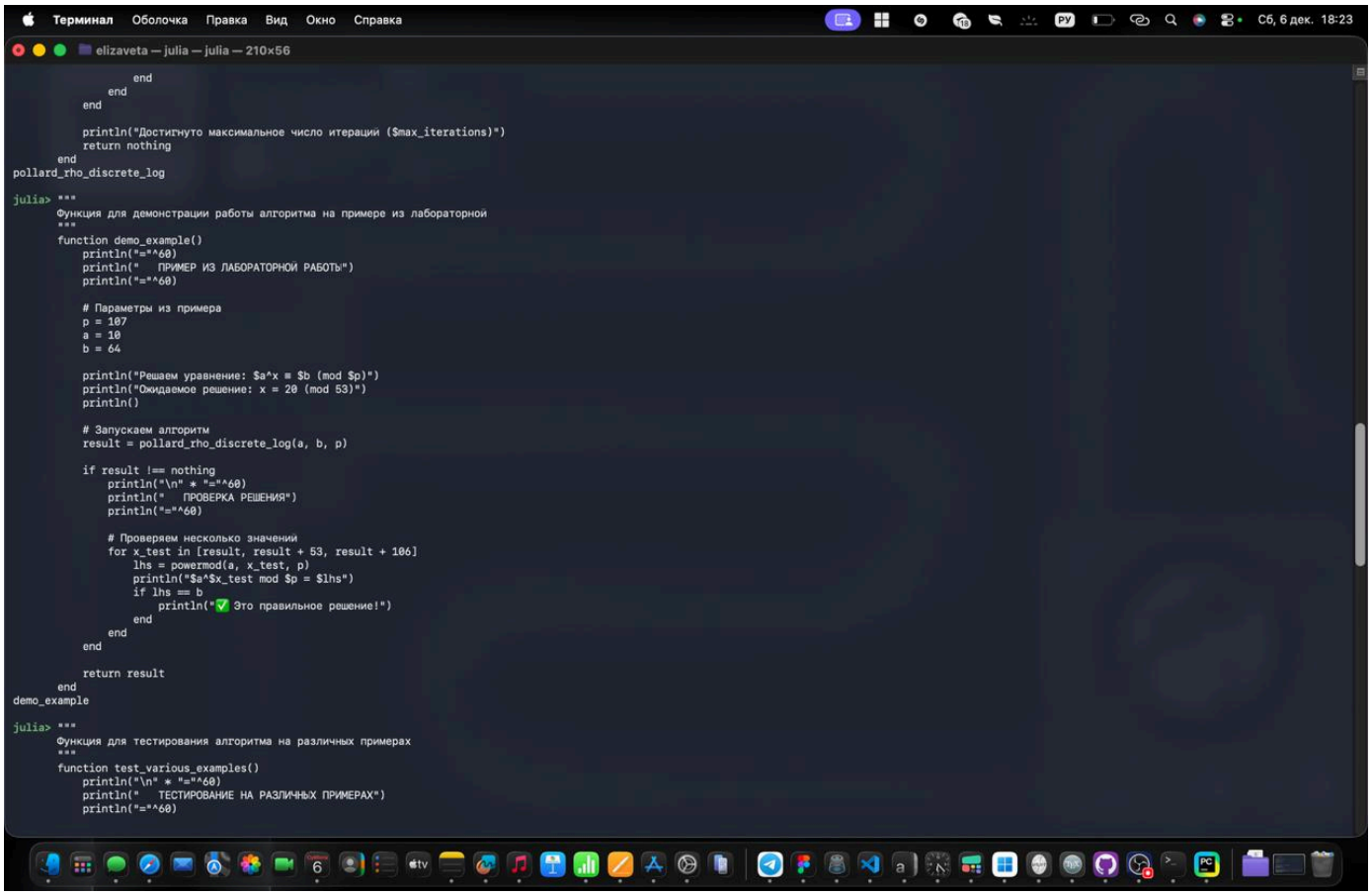
17: Задаём начальные значения

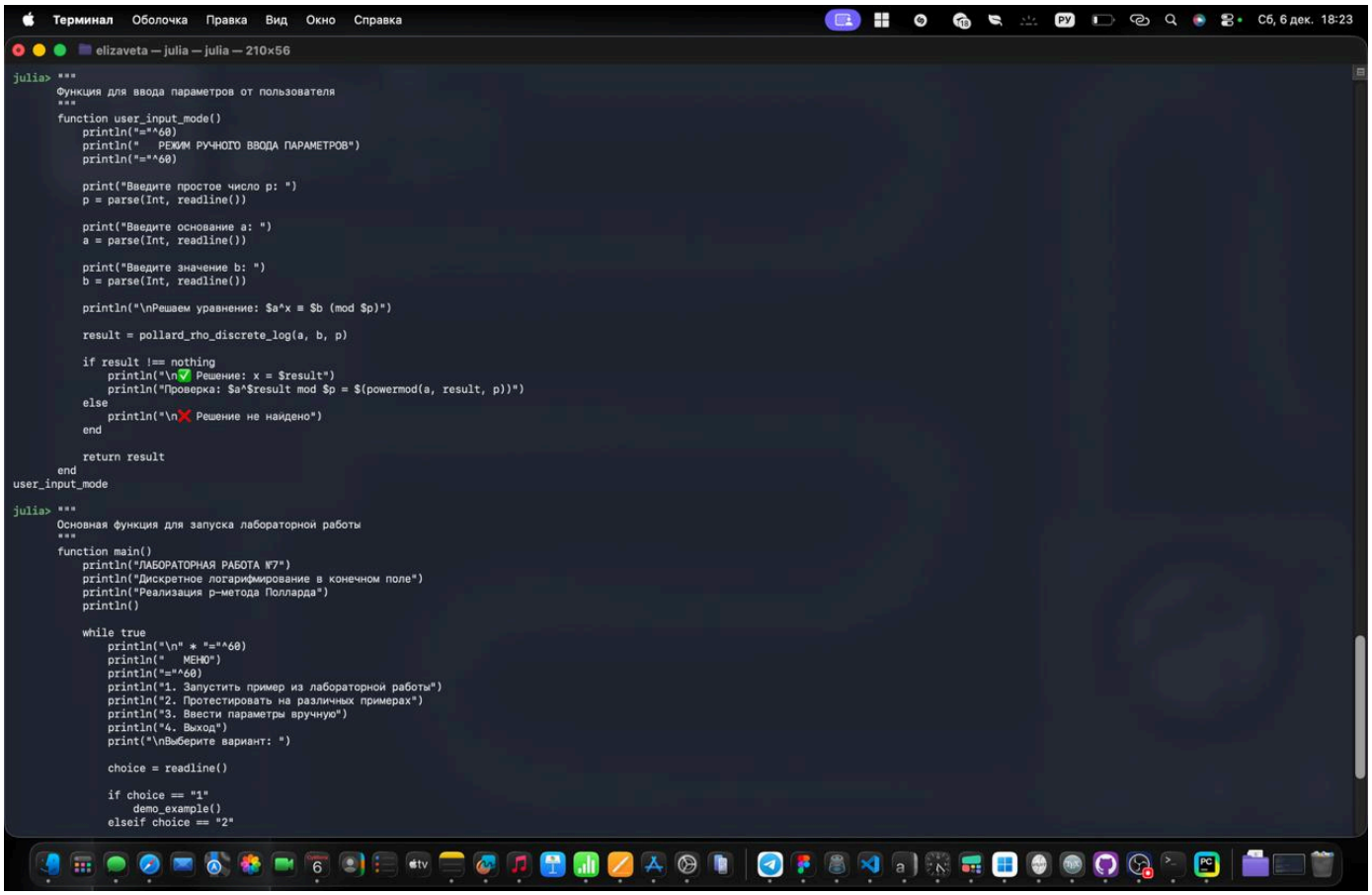
18: Начинаем вычисление, пока не получим равенство

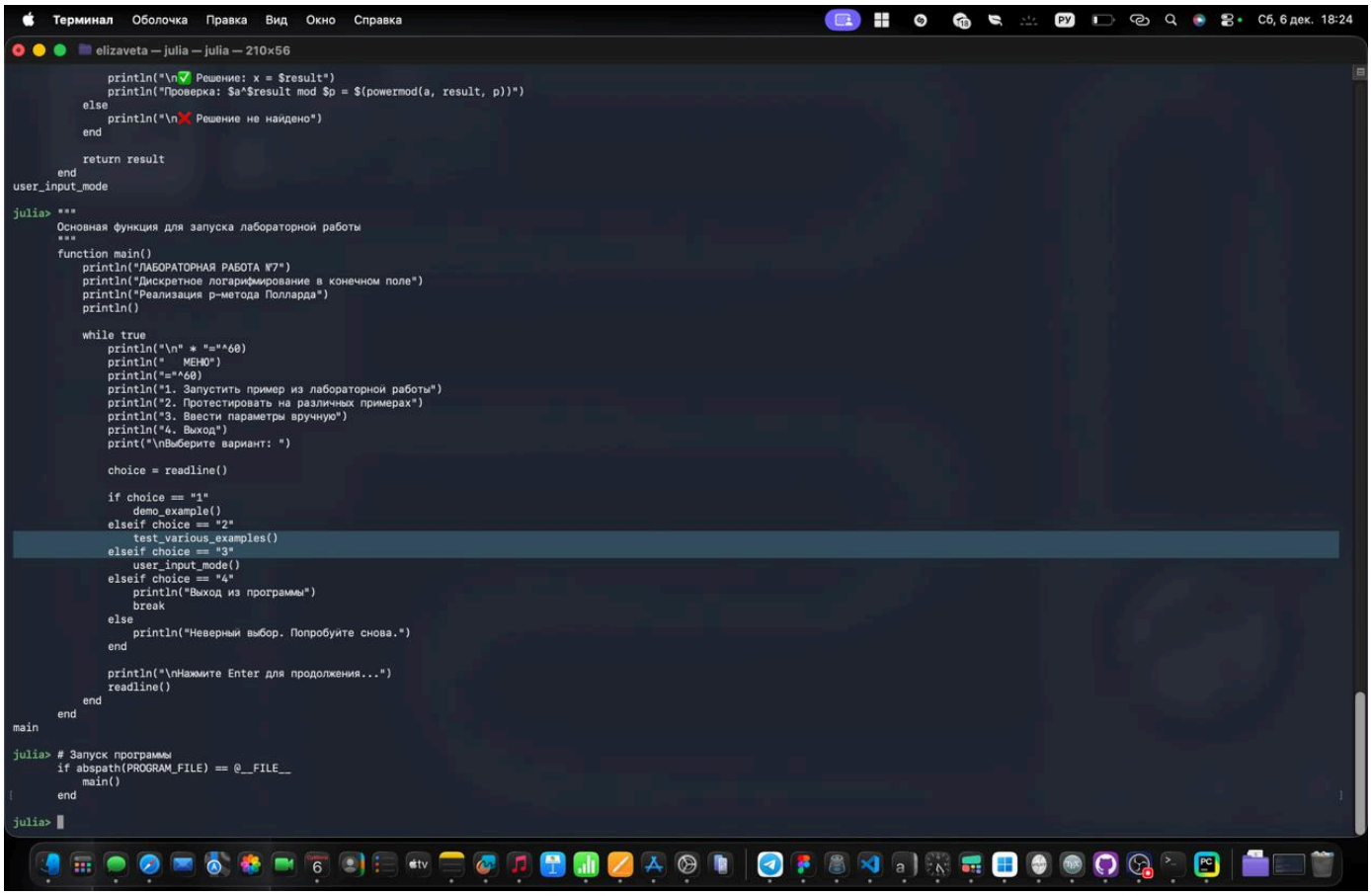
18-36: запускаем основной алгоритм, который с помощью вычисления остатков от деления и формул, представленных в лабораторной работе, формирует таблицу ответов.

39: запускаем функцию.

Мы можем видеть результат на (рис. [-@fig:003]) . Программа работает верно.


Терминал Оболочка Правка Вид Окно Справка
Сб, 6 дек. 18:23
elizaveta — julia — julia — 210×56
end
end
end
println("Достигнуто максимальное число итерации ($max_iterations)")
return nothing
end
pollard_rho_discrete_log
julia> """
Функция для демонстрации работы алгоритма на примере из лабораторной
"""
function demo_example()
println("="^60)
println(" ПРИМЕР ИЗ ЛАБОРАТОРНОЙ РАБОТЫ")
println("="^60)
# Параметры из примера
p = 107
a = 10
b = 64
println("Решаем уравнение: $a^x ≡ $b (mod $p)")
println("Ожидаемое решение: x = 20 (mod 53)")
println()
# Запускаем алгоритм
result = pollard_rho_discrete_log(a, b, p)
if result !== nothing
println("\n" * "="^60)
println(" ПРОВЕРКА РЕШЕНИЯ")
println("="^60)
# Проверяем несколько значений
for x_test in [result, result + 53, result + 106]
lhs = powermod(a, x_test, p)
println("$a^$x_test mod $p = $lhs")
if lhs == b
println("✅ Это правильное решение!")
end
end
end
return result
end
demo_example
julia> """
Функция для тестирования алгоритма на различных примерах
"""
function test_various_examples()
println("\n" * "="^60)
println(" ТЕСТИРОВАНИЕ НА РАЗЛИЧНЫХ ПРИМЕРАХ")
println("="^60)


Терминал Оболочка Правка Вид Окно Справка
Сб, 6 дек. 18:23
elizaveta — julia — julia — 210×56
julia> """
Функция для ввода параметров от пользователя
"""
function user_input_mode()
println("="^60)
println(" РЕЖИМ РУЧНОГО ВВОДА ПАРАМЕТРОВ")
println("="^60)
print("Введите простое число p: ")
p = parse(Int, readline())
print("Введите основание a: ")
a = parse(Int, readline())
print("Введите значение b: ")
b = parse(Int, readline())
println("\nРешаем уравнение: $a^x ≡ $b (mod $p)")
result = pollard_rho_discrete_log(a, b, p)
if result !== nothing
println("\n✅ Решение: x = $result")
println("Проверка: $a^$result mod $p = $(powermod(a, result, p))")
else
println("\n❌ Решение не найдено")
end
return result
end
user_input_mode
julia> """
Основная функция для запуска лабораторной работы
"""
function main()
println("ЛАБОРАТОРНАЯ РАБОТА №7")
println("Дискретное логарифмирование в конечном поле")
println("Реализация ρ-метода Полларда")
println()
while true
println("\n" * "="^60)
println(" МЕНЮ")
println("="^60)
println("1. Запустить пример из лабораторной работы")
println("2. Протестировать на различных примерах")
println("3. Ввести параметры вручную")
println("4. Выход")
print("\nВыберите вариант: ")
choice = readline()
if choice == "1"
demo_example()
elseif choice == "2"



Выводы

Познакомилась с алгоритмом разбора числа на множители и реализовала алгоритм p-метод Полларда.

Список литературы

Лабораторная работа №7

Разложение чисел на множители [Электронный ресурс]. URL: https://esystem.rudn.ru/mod/folder/view.php?id=1150980