Лабораторная работа №6

Разложение чисел на множители. Метод Полларда

Исламова С.М.

Информация

Докладчик

- Исламова Сания Маратовна
- студент уч. группы НПИмд-01-24
- Российский университет дружбы народов
- 1132249576@pfur.ru
- https://github.com/SaniyaIslamova26

Вводная часть

Актуальность

- Реализация ρ-алгоритма Полларда для разложения чисел на множители
- Работа с большими числами в языке Julia
- Понимание вероятностных методов факторизации

Объект и предмет исследования

- ρ-алгоритм Полларда
- Факторизация больших чисел
- Псевдослучайные последовательности
- Вычисление наибольшего общего делителя (НОД)
- Язык программирования Julia

Цели и задачи

- Реализовать ρ-алгоритм Полларда для факторизации чисел
- Исследовать эффективность метода на различных входных данных
- Проанализировать поведение алгоритма на составных числах

Теоретическая часть

Метод Полларда

- Вероятностный алгоритм для нахождения нетривиальных делителей
- Основан на поиске циклов в псевдослучайной последовательности
- Использует “черепаху” и “зайца” для обнаружения циклов
- Временная сложность: $O(\sqrt{p})$, где p - наименьший простой делитель

Алгоритм

1. Выбираем псевдослучайную функцию $f(x) = (x^2 + c) \bmod n$
2. Инициализируем $a = b = 1$
3. На каждой итерации:

- a = f(a) (один шаг)
- b = f(f(b)) (два шага)
- d = НОД(|a - b|, n)

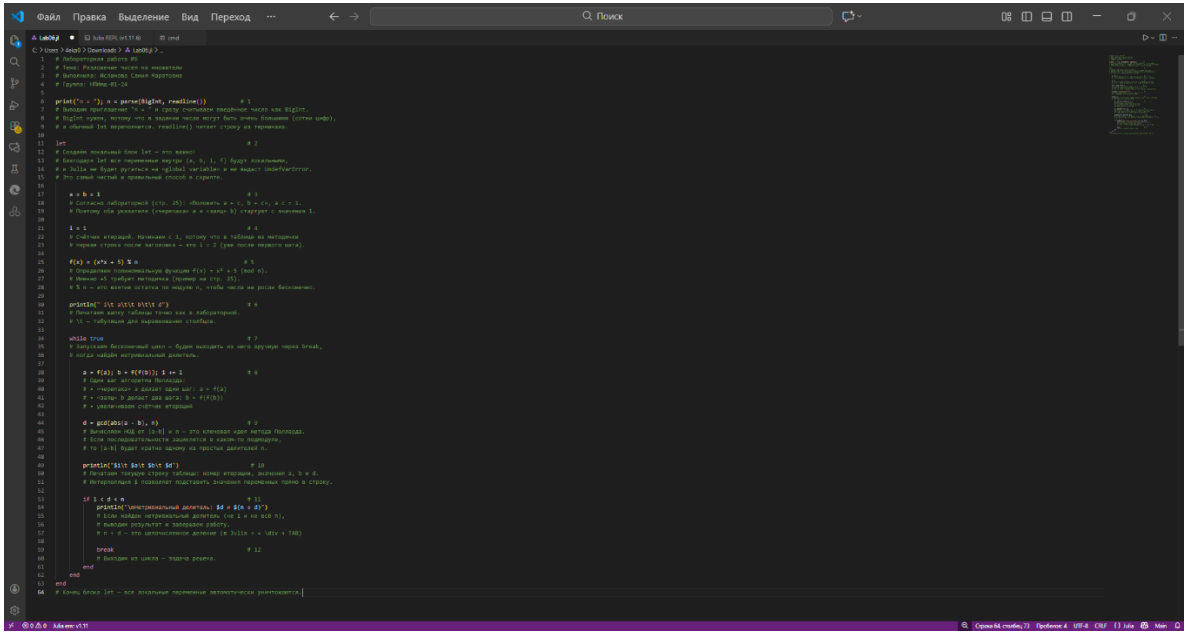
4. Если 1 < d < n - найден нетривиальный делитель

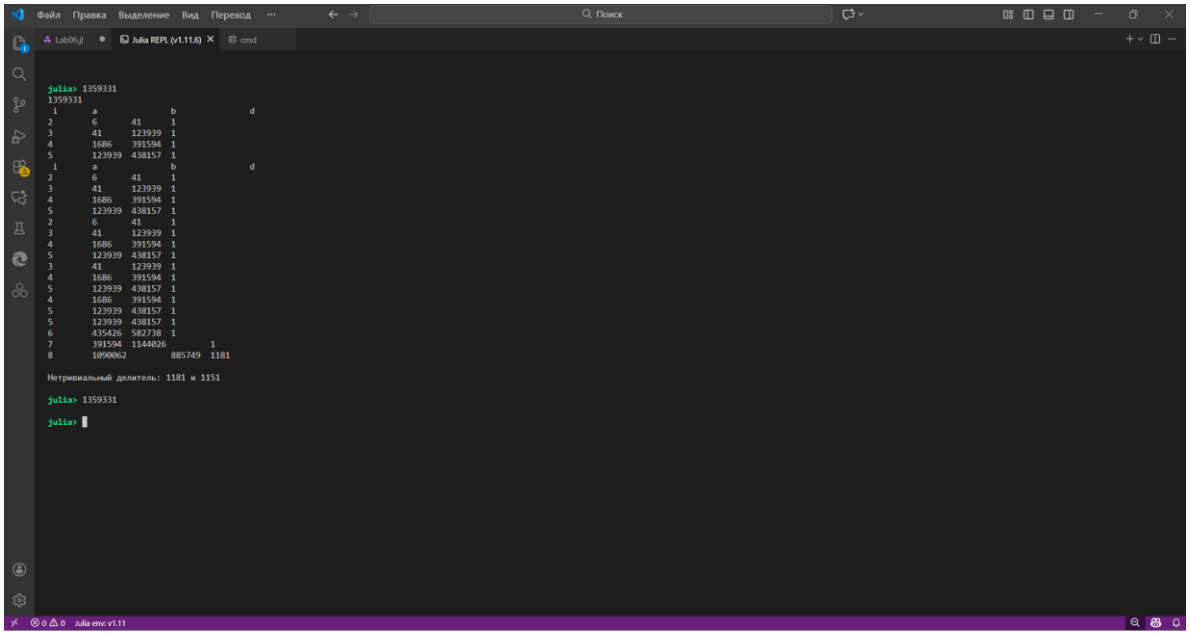
Практическая реализация

Реализация на языке программирования Julia разложение чисел на множители: ρ-алгоритма Полларда для разложения чисел на множители

```
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», а 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) = x² + 5 (mod 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\t b\t\t d")                          # 6
    # Печатаем шапку таблицы точно 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
end
# Конец блока let — все локальные переменные автоматически уничтожаются.
```


Файл Правка Выделение Вид Переход
Поиск
Lab06.jl
Julia REPL (v1.11.6)
cmd
# Лабораторная работа №6
# Тема: Разложение чисел на множители
# Выполнила: Исхакова Сания Харитовна
# Группа: НФИмд-01-24
print("n = "); n = parse(BigInt, readline())
# 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
# 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
# Согласно лабораторной (стр. 25): «Положить a ← 1, b ← c», а c = 1.
# Поэтому оба указателя («черепаха» a и «заяц» b) стартуют с значения 1.
i = 1
# 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^2 + 5) % n
# Определяем полиномиальную функцию f(x) = x² + 5 (mod 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\t b\t\t d")
# Печатаем шапку таблицы точно как в лабораторной.
# \t — табуляция для выравнивания столбцов.
while true
# Запускаем бесконечный цикл — будем выходить из него вручную через break,
# когда найдём нетривиальный делитель.
a = f(a); b = f(f(b)); i += 1
# 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)
# 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")
# Печатаем текущую строку таблицы: номер итерации, значения a, b и d.
# Интерполяция $ позволяет подставить значения переменных прямо в строку.
if 1 < d < n
println("\nНетривиальный делитель: $d = $(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
Выходим из цикла — задача решена.
end
end
end
Конец блока let — все локальные переменные автоматически уничтожаются.
Julia env: v1.11


Файл Правка Выделение Вид Переход
Поиск
Lab06.jl
Julia REPL (v1.11.6)
cmd
julia> 1359331
1359331
i 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
2 6 41 1
3 41 123939 1
4 1686 391594 1
5 123939 438157 1
3 41 123939 1
4 1686 391594 1
5 123939 438157 1
4 1686 391594 1
5 123939 438157 1
5 123939 438157 1
6 435426 582738 1
7 391594 1144026 1
8 1090862 885749 1181
Нетривиальный делитель: 1181 и 1151
julia> 1359331
julia>
Julia env: v1.11

Результаты

- Выполнены все необходимые действия для реализации задач лабораторной работы №6: успешно реализовано на языке программирования Julia разложение чисел на множители: ρ-алгоритма Полларда для разложения чисел на множители.

Вывод

Реализовано на языке программирования Julia разложение чисел на множители: ρ-алгоритма Полларда для разложения чисел на множители