МИНОБРНАУКИ РОССИИ
ФГБОУ ВО «СГУ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

АЛГОРИТМЫ АЛГЕБРЫ И ТЕОРИИ ЧИСЕЛ

ЛАБОРАТОРНАЯ РАБОТА №12

студента 4 курса 431 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Никитина Арсения Владимировича

Проверил
доцент _____ А. С. Гераськин

Саратов 2022

СОДЕРЖАНИЕ

1 Задание лабораторной работы

Осуществить факторизацию с помощью алгоритма Диксона.

2 Теоретическая часть

Алгоритм Диксона факторизации чисел использует в своей основе идею Лежандра, заключающуюся в поиске пары целых чисел x и y таких, что:

$$x^2 \equiv y^2 \pmod{n},\ x \not\equiv \pm y \pmod{n}$$

.

Описание алгоритма

1. Составить факторную базу $B = \{p_1, p_2, \ldots, p_h\}$, состоящую из всех простых чисел $p \leq M = L(n)^{\frac{1}{2}}$, где $L(n) = \exp\left(\sqrt{\ln n \cdot \ln\ln n}\right)$.
2. Выбрать случайное b, $\sqrt{n} < b < n$.
3. Вычислить $a = b^2 \bmod n$.
4. Проверить число a на гладкость пробными делениями. Если a является B-гладким числом, то есть $a = \prod_{p \in B} p^{\alpha_p(b)}$, следует запомнить вектора $\vec{\alpha}(b)$ и $\vec{\varepsilon}(b)$:
   а) $\vec{\alpha}(b) = (\alpha_{p_1}(b), \ldots, \alpha_{p_h}(b))$,
   б) $\vec{\varepsilon}(b) = (\alpha_{p_1}(b) \bmod 2, \ldots, \alpha_{p_h}(b) \bmod 2)$.
5. Повторять процедуру генерации чисел b до тех пор, пока не будет найдено $h+1$ B-гладких чисел $b_1, ..., b_{h+1}$.
6. Методом Гаусса найти линейную зависимость среди векторов $\vec{\varepsilon}(b_1), \ldots, \vec{\varepsilon}(b_{h+1})$:

$$\vec{\varepsilon}(b_{i_1}) \oplus \cdots \oplus \vec{\varepsilon}(b_{i_t}) = \vec{0},\ t = \overline{1, m}.$$

7. Положить $x = b_{i_1} \ldots b_{i_t} \bmod n$ и $y = \prod_{p \in B} p^{\frac{\left(\alpha_p\left(b_{i_1}\right) + \cdots + \alpha_p\left(b_{i_t}\right)\right)}{2}} \bmod n$.
8. Проверить $x \equiv \pm y \pmod{n}$. Если это так, то повторить процедуру генерации. Если нет, то найдена нетривиальное разложение:

$$n = u \cdot v, u = gcd(x + y, n), v = gcd(x - y, n).$$

3 Практическая часть

3.1 Пример работы алгоритма

```
L = 194.17460574500024
M = 13.93465484843454
Фактор-база B = { 2 3 5 7 11 13 }

b_i [43805, 3300, 31577, 36314, 68820, 49325, 8128]

a_i:
[3, 1, 1, 2, 0, 0]
[0, 0, 1, 2, 2, 0]
[1, 0, 0, 1, 3, 0]
[3, 3, 0, 0, 2, 0]
[4, 0, 1, 1, 0, 0]
[0, 1, 1, 3, 1, 0]
[5, 1, 0, 2, 0, 0]

e_i:
[1, 1, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0]
[1, 0, 0, 1, 1, 0]
[1, 1, 0, 0, 0, 0]
[0, 0, 1, 1, 0, 0]
[0, 1, 1, 1, 1, 0]
[1, 1, 0, 0, 0, 0]


Номера линейно зависимых векторов в матрице e_i: (3, 6)
619 * 145 = 89755
```

Рисунок 1

3.2 Код программы, реализующей рассмотренный алгоритм

```
from sympy import *
import math
import random
import numpy as np
```

```
from numpy import sum as npsum
from itertools import chain, combinations



def combs(matr):
    arr = []
    for k in range(2, len(matr) + 1):
        rows = combinations(matr,k)
        indices = list(combinations(range(len(matr)), k))
        for i, j in enumerate(rows):
            if sum(npsum(j, axis=0) % 2) == 0:
                arr.append(indices[i])
    return min(arr, key=len)


def base_smoothness(a, base):

    vector = [0] * len(base)

    for i, divider in enumerate(base):
        while not a % divider:
            a //= divider
            vector[i] += 1

    if a == 1:
        return True, vector, [elem % 2 for elem in vector]
    else:
        return False, None, None


def dixon_factorization(n):

    l = math.exp(math.sqrt(math.log(n) * math.log(math.log(n))))
    print('L =', l)

    m = l ** (1 / 2)
    print('M =', m)

    factor_base = list(sieve.primerange(m))
    print('Фактор-база B = {', *factor_base, '}')
```

```
46
47      a_vectors = []
48      e_vectors = []
49      b_i = []
50
51      h = len(factor_base)
52
53      left = math.floor(math.sqrt(n)) + 1
54      right = n - 1
55
56      while len(a_vectors) - 1 != h:
57
58          b = random.randint(left, right)
59
60          a = (b * b) % n
61          # if a == 0:
62          #     continue
63
64          is_smooth, a_vector, e_vector = base_smoothness(a, factor_base)
65
66          if is_smooth and b not in b_i and any(e_vector):
67              a_vectors.append(a_vector)
68              e_vectors.append(e_vector)
69              b_i.append(b)
70
71
72      print('\n b_i', b_i)
73      print('\n a_i:', *a_vectors, sep='\n')
74      print('\n e_i:', *e_vectors, '\n', sep='\n')
75
76
77      lin_res = combs(e_vectors)
78
79      if len(lin_res) == 0:
80          return(dixon_factorization(n))
81      else:
82          print(f'Номера линейно зависимых векторов в матрице e_i: {lin_res}
            ↪  ')
83
84      x = np.prod([b_i[number] for number in lin_res]) % n
85
```

```
86      powers_list = [0] * len(factor_base)
87      for i in lin_res:
88          for number, power in enumerate(a_vectors[i]):
89              powers_list[number] += power
90
91      y = prod([factor_base[i] ** (elem // 2) for i, elem in \
92          enumerate(powers_list)]) % n
93
94
95      if y != x and x != (n - y):
96          u = gcd(n, x - y)
97          v = gcd(n, x + y)
98          if u * v != n:
99              return dixon_factorization(n)
100         else:
101             print(f'{u} * {v} = {n}')
102             return
103     else:
104         return dixon_factorization(n)
105
106
107 def main():
108
109     while True:
110
111         print('\nВыполнить факторизацию методом Диксона - \enter')
112         print('Выход из программы - 2')
113
114         try:
115             value = int(input('Введите значение: '))
116
117         except ValueError:
118             value = 1
119
120         if value == 1:
121
122             n = int(input('Введите число, которое требуется факторизовать: '))
123             dixon_factorization(n)
124
125         elif value == 2:
126             print('Работа программы завершена')
```

```
127                 return
128
129
130 if __name__ == "__main__":
131     main()
132
```