Лабораторная работа № 5. Метод квадратичного решета.

Десятов Александр, 09-641.

import math

import itertools

import numpy as np

from itertools import chain

import copy

from threading import Thread

def yakobi\_symbol(a\_f, b\_f):

if math.gcd(a\_f, b\_f) != 1:

return 0

r = 1

if a\_f < 0:

a\_f = -a\_f

if b\_f % 4 == 3:

r = -r

while a\_f != 0:

t = 0

# Пока a\_f четное

while not a\_f & 1:

t += 1

# деление на 2

a\_f >>= 1

# Если t нечетное

if t & 1:

b\_f\_mod\_8 = b\_f % 8

if b\_f\_mod\_8 == 3 or b\_f\_mod\_8 == 5:

r = -r

a\_f\_mod\_4 = a\_f % 4

b\_f\_mod\_4 = b\_f % 4

if a\_f\_mod\_4 == b\_f\_mod\_4 and a\_f\_mod\_4 == 3:

r = -r

c\_f = a\_f

a\_f = b\_f % c\_f

b\_f = c\_f

return r

def factorization(n\_f):

# Получить 4 миллиона первых простых чисел

all\_prime\_arr = []

for i in range(1, 5):

f = open('primes' + str(i) + '.txt', 'r')

all\_prime\_str = f.read()

f.close()

all\_prime\_arr.append([int(i) for i in all\_prime\_str.split()])

all\_prime\_arr = list(itertools.chain.from\_iterable(all\_prime\_arr))

# Не является ли степенью

n\_sqrt = math.floor(math.sqrt(n\_f))

if n\_f == n\_sqrt\*n\_sqrt:

print(str(n\_f) + ' = ' + str(n\_sqrt) + ' \* ' + str(n\_sqrt))

return

# Набор простых чисел в базу S

S = [-1, 2]

i\_prime = 1

k = 18

m = math.floor(math.sqrt(n\_f))

while True:

len\_S\_beg = len(S)

# Добор S

while i\_prime < len(all\_prime\_arr):

if yakobi\_symbol(n\_f, all\_prime\_arr[i\_prime]) == 1:

S.append(all\_prime\_arr[i\_prime])

i\_prime += 1

if len(S) == k:

break

if len\_S\_beg == len(S):

print("не хватает простых чисел (не дополнилось ничем)")

return

print("S = ", S)

t = k + 1

print("t = ", t)

k += 1

v\_matrix = []

a\_arr = []

b\_arr = []

x\_arr = [0]

for x in x\_arr:

if x >= 0:

x\_arr.append(x + 1)

x\_arr.append(-x - 1)

a = x + m

b = b\_for\_change = a\*a - n

v\_row = [0]\*k

for i\_s, s in enumerate(S):

if i\_s == 0:

if b\_for\_change < 0:

b\_for\_change = -b\_for\_change

v\_row[i\_s] = 1

continue

while b\_for\_change % s == 0:

b\_for\_change /= s

v\_row[i\_s] = (v\_row[i\_s] + 1) & 1

# Если удалось b разложить на множители из S, то есть b является p(k) гладкой

if b\_for\_change == 1:

v\_matrix.append(v\_row)

a\_arr.append(a)

b\_arr.append(b)

print("x =", x, "a =", a, "b =", b, "e =", v\_row)

if len(v\_matrix) == t:

break

# cV = 0

v\_matrix\_np = np.array(v\_matrix)

zero\_arr\_k = np.zeros(k)

# Все бинарные числа в двоичном списке

c\_arr = np.array(list(itertools.product([0, 1], repeat=t)))

for c in c\_arr:

# Если c является решением уравнения cV = 0

if np.array\_equal(c @ v\_matrix\_np % 2, zero\_arr\_k):

xx = b = 1

for i\_val in enumerate(c):

if i\_val[1]:

b \*= b\_arr[i\_val[0]]

xx = (xx \* a\_arr[i\_val[0]]) % n\_f

y = math.floor(math.sqrt(b))

# print("xx =", xx, "y =", y)

if (not xx == y % n\_f) and (not xx == (n\_f - y) % n\_f):

answer = math.gcd(xx + y, n\_f)

if not answer == 1:

print(str(n\_f) + ' = ' + str(answer) + ' \* ' + str(n\_f//answer))

return

n = 10002200057 # 100019\*100003

#n=1811706971 #17299\*104729

#n=2328707497 #61637\*37781

#n=100160063 #10009\*10007

#n=99400891 #9973\*9967

#n=1022117 #1009\*1013

#n=23123 #1217\*19

#n=91 #7\*13

# Запуск в отдельном потоке

thread = Thread(target=factorization,args=(n,))

thread.start()

thread.join()