**Кощавцев Даниил, 4ПМ, Дискретное логарифмирование**

**7 вариант.**

1) 11^x = 172 mod(251)

**а) Шаг младенца-шаг великана**

С помощью функции gcdex реализуется расширенный алгоритм Евклида.

Она может быть использована для поиска обратного элемента.

def gcdex(a, b):

if b == 0:

return a, 1, 0

else:

d, x, y = gcdex(b, a % b)

return d, y, x - y \* (a // b)

Функция obr() используется для нахождения обратного элемента к a по модулю b. Находит а^(-1) mod b, то есть такое с, что ac=1(mod b).

def obr(a, b):

g = gcdex(a, b)

return g[1] % b if g[0] == 1 else 0

Через функцию baby\_giant реализуется сам алгоритм, используя вышеуказанные функции:

def baby\_giant(a, b, p):

m = int(sqrt(p)) + 1

baby\_steps = {pow(a, l, p): l for l in range(m)}

c = obr(pow(a, m, p), p)

current = b

for k in range(m):

if current in baby\_steps:

l = baby\_steps[current]

return m \* k + l

current = (current \* c) % p

return None

Используя этот алгоритм, получаем, что x = 97

**б) Исчисление порядка**

С помощью функцииgenerate\_factor\_base генерируем факторную базу из первых t=7 простых элементов. Получаем: [2, 3, 5, 7, 11, 13, 17]

def generate\_factor\_base(t):

base = []

num = 2

while len(base) < t:

if sp.isprime(num):

base.append(num)

num += 1

return base

factor\_base = generate\_factor\_base(t)

Следующим шагом вычисляем g^k, раскладываем его по факторной базе и получаем систему уравнений, которую нужно решить, это реализуется через функции:

def decompose (g\_k, factor\_base):

exponents = []

cur = g\_k

for prime in factor\_base:

exponent = 0

while cur % prime == 0:

cur //= prime

exponent += 1

exponents.append(exponent)

if cur == 1:

return exponents

return None

def collect\_equations(g, n, p, factor\_base, n\_equations):

A = np.zeros((n\_equations, n\_equations), dtype=np.int32)

b = np.zeros(n\_equations, dtype=np.int32)

count = 0

k = 0

while count < n\_equations:

k += 1

g\_k = pow(g, k, p)

exponents = decompose\_in\_base(g\_k, factor\_base)

if exponents and not any(map(lambda x: mgcd(x, n, 1) > 1, exponents)):

A[count] = exponents

b[count] = k

if(np.linalg.matrix\_rank(A) < (count + 1)):

A[count] = np.zeros(n\_equations)

b[count] = 0

continue

count += 1

print("Получили следующую систему уравнений, которую надо решить")

for i in range(len(A)):

if len(A) // 2 == i:

print(f"{A[i]} = {b[i]}")

else:

print(f"{A[i]} {b[i]}")

return A, b

Получили такую систему:

[0 0 0 0 1 0 0] x = 1

[0 1 0 0 0 0 0] x = 6

[0 0 1 0 0 0 1] x = 14

[1 0 0 1 0 1 0] x =15

[1 0 0 1 0 0 1] x = 37

[1 0 0 0 0 1 0] x = 47

[0 0 1 1 0 0 0] x = 48

Дальше решаем полученную систему и находим вектор:

[[135], [6], [80], [218], [1], [162], [184]]

def solve\_system(A, b, n):

A = sp.Matrix(A).applyfunc(lambda x: x % n)

b = sp.Matrix(b).applyfunc(lambda x: x % n)

# Применяем метод Гаусса по модулю p

m = A.shape[0]

logs = sp.zeros(m, 1) # Инициализируем вектор решений

for i in range(m):

# Ищем ненулевой элемент в столбце i начиная с строки i

for j in range(i, m):

if A[j, i] != 0:

A.row\_swap(i, j)

b.row\_swap(i, j)

break

else:

# Если столбец нулевой, система может не иметь решения

print(f"Система уравнений имеет нулевой столбец, не возможно найти ее решение. Попробуйте уменьшить t")

return None

# Нормализуем строку

inv = sp.mod\_inverse(A[i, i], n)

for j in range(i, m):

A[i, j] = (A[i, j] \* inv) % n

b[i] = (b[i] \* inv) % n

# Обнуляем остальные элементы в столбце i

for j in range(m):

if j != i:

factor = A[j, i]

for l in range(i, m):

A[j, l] = (A[j, l] - factor \* A[i, l]) % n

b[j] = (b[j] - factor \* b[i]) % n

# Считываем решение

for i in range(m):

logs[i] = b[i]

print(f"Получили следующее решение системы:\n logs={logs}")

return logs

Финальным шагом раскладываем a\*g^k по факторной базе и логарифмируя получаем решение x = 97

def discrete\_log(g, a, n, p, factor\_base, logs):

while True:

k = random.randint(0, n - 1)

target = (a \* pow(g, k)) % p

exponents = decompose(target, factor\_base)

if exponents:

print(f"На следующем этапе получили разложение по факторной базе числа {a}\*{g}^{k}: exponents={exponents}")

x = sum(exponents[i] \* logs[i] for i in range(len(logs))) - k

return x % n

2) 7 ^x = 15 mod(263)

Через алгоритм Шаг младенца-шаг великана получаем, что x = 193

Через алгоритм исчисление порядка получаем, что x = 193, проделав те же действия, с факторной базой [2, 3, 5, 7, 11, 13, 17] и системой уравнений:

[0 0 0 1 0 0 0] x = 1

[1 0 0 0 0 0 1] x = 4

[3 0 0 0 1 0 0] x = 6

[3 0 0 0 0 1 0] x = 8

[1 1 1 0 0 0 0] x = 13

[0 1 0 0 1 0 0] x = 16

[1 0 1 0 0 0 0] x = 19