*Данькова Єлізавета ФБ-11*

СПЕЦІАЛЬНІ РОЗДІЛИ

ОБЧИСЛЮВАЛЬНОЇ МАТЕМАТИКИ

КОМП’ЮТЕРНИЙ ПРАКТИКУМ No2

Багаторозрядна модулярна арифметика

**1. Мета роботи**

Отримання практичних навичок програмної реалізації багаторозрядної арифметики; ознайомлення з прийомами ефективної реалізації критичних по часу ділянок програмного коду та методами оцінки їх ефективності.

**Завдання до комп’ютерного практикуму**

А) Доопрацювати бібліотеку для роботи з m-бітними цілими числами, створену на комп’ютерному практикумі No1, додавши до неї такі операції:

1) обчислення НСД та НСК двох чисел;

2) додавання чисел за модулем;

3) віднімання чисел за модулем;

4) множення чисел та піднесення чисел до квадрату за модулем;

5) піднесення числа до багаторозрядного степеня d по модулю n .

обчислення НСД та НСК двох чисел;

def gcd(self, other):  
 # GCD using Euclidean algorithm  
 a = self  
 b = other  
  
 if b == BigInt():  
 return a  
  
 while b != BigInt():  
 a, b = b, a % b  
  
 return a  
  
def lcm(self, other):  
 # LCM calculation using GCD  
 gcd\_ab = self.gcd(other)  
 if gcd\_ab == BigInt():  
 return BigInt() # LCM is not defined for 0  
 else:  
 return (self \* other) // gcd\_ab

2) додавання чисел за модулем;

3) віднімання чисел за модулем;

4) множення чисел та піднесення чисел до квадрату за модулем;

5) піднесення числа до багаторозрядного степеня d по модулю n .

def mod\_add(self, other, mod):  
 result = self + other  
 return result % mod  
  
def mod\_sub(self, other, mod):  
 result = self - other  
 return result % mod  
  
def mod\_mul(self, other, mod):  
 # Modular multiplication using Barrett reduction  
 reduction\_param = 2  
 mu = BigInt((1 << (reduction\_param \* self.size)) // mod.value[0])  
 q1 = (self \* other) >> (reduction\_param \* self.size)  
 q2 = mu \* (q1 \* mod)  
 result = (self \* other - q2 \* mod) % mod  
 return result  
  
def mod\_square(self, mod):  
 # Modular squaring using Barrett reduction  
 reduction\_param = 2  
 mu = BigInt((1 << (reduction\_param \* self.size)) // mod.value[0])  
 q1 = (self \* self) >> (reduction\_param \* self.size)  
 q2 = mu \* (q1 \* mod)  
 result = (self \* self - q2 \* mod) % mod  
 return result  
  
def mod\_pow(self, exponent, mod):  
 # Exponentiation using Horner's scheme  
 result = BigInt(1)  
 base\_power = self % mod  
  
 while exponent > 0:  
 if exponent % 2 == 1:  
 result = (result \* base\_power) % mod  
 exponent //= 2  
 base\_power = (base\_power \* base\_power) % mod  
  
 return result