Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА № 2**

**ПО КУРСУ «КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_К.В.Стасюк

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А. Крамаренко

**Задание:**

№ 12. Реализовать класс вычисления целых степеней по заданному модулю(для вычисления положительной степени воспользоваться малой теоремой Ферма и реализованными отдельном методами умножения и сложения по заданному модулю, для вычисления отрицательной степени воспользоваться теоремой Эйлера).

**Ход работы:**

**Малая теорема Ферма для положительной степени:**

Данная теорема утверждает, что если *p* - простое число, то для любого целого числа *a*, не кратного *p*, ≡1 (mod *p*).

**Проверка взаимной простоты:**

Малая Теорема Ферма требует, чтобы основание было взаимно просто с модулем. Это значит, что НОД (наибольший общий делитель) этих двух чисел должен быть равен 1. Если это условие не выполняется, вызывается исключение.

**Цель использования Малой Теоремы Ферма:**

Целью использования Малой Теоремы Ферма в методе \_modular\_power\_positive является эффективное вычисление положительных степеней числа по модулю. Эта теорема предоставляет математическое свойство, которое позволяет снизить вычислительную сложность возведения в степень.

Оптимизация умножений:

Поскольку ≡1 (mod *p)*, мы можем уменьшить количество умножений в вычислении (mod *p)* Вместо полного перемножения a само на себя

e раз, мы будем возводить a в квадрат и умножать на текущий результат только в тех случаях, когда бит степени e равен 1. Это происходит в цикле, где мы проверяем четность текущей степени.

**Теорема Эйлера для подсчета отрицательной степени:**

**Теорема Эйлера:**

Если *a* и *n* являются взаимно простыми положительными целыми числами , то для любого положительного целого числа *a* и положительного целого числа *n*, теорема Эйлера утверждает следующее:

≡ 1(mod *n*)

**Использование Теоремы Эйлера в вычислении обратных элементов:**

Теорема Эйлера играет важную роль при вычислении обратных элементов по модулю. Если у нас есть целое число a, взаимно простое с модулем n, то теорема Эйлера говорит нам, что: ≡ 1(mod *n*)

Так как ϕ(n) представляет количество взаимно простых с n чисел в интервале от 1 до n−1, мы можем переписать теорему Эйлера как:

≡ (mod *n*)

Таким образом, чтобы найти обратный элемент по модулю n, мы можем возвести a в степень ϕ(n)−1 и взять остаток от деления на n.

**Реализация в методе:**

Метод \_modular\_power\_negative класса ModularExponentiation предназначен для вычисления отрицательных степеней числа по теореме Эйлера.

**Проверка взаимной простоты:**

Первым шагом метод проверяет взаимную простоту между основанием base и модулем self.modulus. Это важно, так как теорема Эйлера требует, чтобы основание и модуль были взаимно просты.

**Вычисление функции Эйлера от модуля:**

Затем метод находит функцию Эйлера (phi\_modulus) для модуля.

**Нахождение обратного элемента по теореме Эйлера:**

Согласно теореме Эйлера, если a и n взаимно просты, то

≡ (mod *n*). Таким образом, метод находит обратный элемент inverse\_base. Далее метод возвращает результат вызова \_modular\_power\_positive с применением найденного обратного элемента и отрицательной экспонентой. Это делается для использования оптимизированного вычисления положительных степеней с использованием малой теоремы Ферма.

Теорема Эйлера позволяет эффективно вычислять обратные элементы по модулю, что полезно при работе с отрицательными степенями.

Использование оптимизированного вычисления положительных степеней для обработки отрицательных степеней, что улучшает эффективность алгоритма.

**Реализация класса :**

Класс ModularExponentiation предоставляет метод calculate\_power, который вычисляет степень числа по модулю, учитывая знак экспоненты.

Для оптимизации вычислений, класс использует малую теорему Ферма и теорему Эйлера.

Метод \_modular\_power\_positive реализует вычисление положительных степеней.

Метод \_modular\_power\_negative использует теорему Эйлера для вычисления отрицательных степеней.

Пример использования класса показывает, как использовать его для конкретного вычисления.

Код программы:

from lab1 import euler\_phi\_standard\_formula\_opt  
  
  
class ModularExponentiation:  
 def \_\_init\_\_(self, base, exponent, modulus):  
 # Инициализация объекта с основанием, степенью и модулем  
 self.base = base  
 self.exponent = exponent  
 self.modulus = modulus  
  
 def \_modular\_multiply(self, a, b):  
 # Метод для умножения по модулю  
 return (a \* b) % self.modulus  
  
 def \_are\_coprime(self, a, b):  
 # Метод для проверки взаимной простоты(Алгоритм Евклида)  
 while b:  
 a, b = b, a % b  
 return a == 1  
  
 # Малая теорема Ферма утверждает, что: если p — простое число, то для любого целого числа a, не кратного p,  
 # a^(p-1) при делении на p дает остаток 1.  
 def \_modular\_power\_positive(self, base, exponent):  
 result = 1  
 # приводим основание к модулю  
 base = base % self.modulus  
  
 # Проверка взаимной простоты между основанием и модулем  
 if not self.\_are\_coprime(base, self.modulus):  
 raise ValueError("Основание и модуль должны быть взаимно просты")  
  
 while exponent > 0:  
 # степень числа нечетная  
 if exponent % 2 == 1:  
 # умножаем наше число на себя (сделать степень четной)  
 result = self.\_modular\_multiply(result, base)  
 # уменьшаем степень вдвое  
 exponent //= 2  
 # возводим основание в квадрат по модулю  
 base = self.\_modular\_multiply(base, base)  
  
 return result  
  
 def \_modular\_power\_negative(self, base, exponent):  
 # Метод для вычисления отрицательной степени по теореме Эйлера  
 # проверяем взаимную простоту между основанием и модулем  
 if not self.\_are\_coprime(base, self.modulus):  
 raise ValueError("Основание и модуль должны быть взаимно просты")  
  
 # находим функцию Эйлера от модуля  
 phi\_modulus = euler\_phi\_standard\_formula\_opt((self.modulus))  
 # находим обратный эл-т по теореме Эйлера  
 inverse\_base = pow(base, phi\_modulus - 1, self.modulus) # a^(phi(n)-1) % n  
  
 return self.\_modular\_power\_positive(inverse\_base, -exponent)  
  
 def calculate\_power(self):  
 # Вычисление степени в зависимости от знака экспоненты  
 if self.exponent >= 0:  
 return self.\_modular\_power\_positive(self.base, self.exponent)  
 else:  
 return self.\_modular\_power\_negative(self.base, self.exponent)  
  
  
# Пример использования класса  
base = 3  
exponent = 5  
modulus = 11  
  
mod\_exp = ModularExponentiation(base, exponent, modulus)  
result = mod\_exp.calculate\_power()  
  
print(f"{base}^{exponent} mod {modulus} = {result}")