**2.1 Реалізація власного типу даних великого числа**

Головним полем вашого власного типу даних для великих чисел має бути масив беззнакових цілих чисел (unsigned integer 32 або 64). Ваш власний тип даних даних може бути реалізован у вигляді структури або класу. Окрім головного поля, ваш тип даних може містити і додаткові поля виходячи з ваших потреб.

Також треба реалізувати методи для встановлення числа і повернення числа. Найпоширеніший варіант, це метод встановлення з числа закодованого в шістнадцяткову систему числення, що передається рядком тексту (string). І відповідно метод повернення числа в шістнадцятковій системі числення у вигляді рядка тексту (string).

Крім цього, ви можете реалізувати інші методи встановлення і повернення числа, такі як байтовий масив, числа в десятичній системі числення або інші варіанти, які відповідають вашим потребам.

В якості автоматичного тестування, реалізуйте виклики методів з різними числами різної довжини і кожного разу порівнюйте вхідні дані з вихідними.

У цьому прикладі визначено методи **set\_value** для встановлення числа і **get\_value** для повернення числа у шістнадцятковому форматі. Також реалізовано методи перетворення чисел із шістнадцяткового формату в цілі числа і навпаки.

class BigInteger:

def \_\_init\_\_(self, value=0):

# Ініціалізація об'єкту BigInteger значенням value

self.value = value

def set\_value(self, value):

# Встановлення значення числа

self.value = value

def get\_value(self):

# Повертає значення числа

return self.value

def add(self, other):

# Додавання двох великих чисел

result = self.value + other.get\_value()

return BigInteger(result)

# Приклад використання

num1 = BigInteger(123456789)

num2 = BigInteger(987654321)

sum\_num = num1.add(num2)

print("Сума чисел:", sum\_num.get\_value()) # Вивід: Сума чисел: 1111111110

**2.2 Реалізація побітових операцій**

Реалізуйте методи, що будуть виконувати побітові операції з обʼєктами вашого власного типу даних. Приклад використання повинен мати приблизно наступний вигляд.

MyBigInt numberA;

MyBigInt numberB;

MyBigInt numberC;

numberA.setHex(“e035c6cfa42609b998b883bc1699df885cef74e2b2cc372eb8fa7e7”);

numberB.setHex(“5072f028943e0fd5fab3273782de14b1011741bd0c5cd6ba6474330”);

numberC = XOR(numberA, numberB);

print(numberC.getHex())

Операції, які треба реалізувати на цьому етапі завдання:

* INV (побітова інверсія)
* XOR (побітове виключне або)
* OR (побітове або)
* AND (побітове і)
* shiftR (зсув праворуч на n бітів)
* shiftL (зсув ліворуч на n бітів)

В якості автоматичного тестування, реалізуйте виклики методів зі заздалегідь підготовленими даними і порівнюйте отриманий результат з правильним значенням.

Цей код виконує побітові операції, такі як побітова інверсія, побітове виключне або, побітове або, побітова і, зсув праворуч і зсув ліворуч.

class MyBigInt:

def \_\_init\_\_(self, value=0):

self.value = value

def setHex(self, hex\_string):

# Встановлює значення з шістнадцяткового рядка

self.value = int(hex\_string, 16)

def getHex(self):

# Повертає значення у шістнадцятковому форматі

return hex(self.value).upper()[2:]

def INV(self):

# Побітова інверсія

self.value = ~self.value

return self

def XOR(self, other):

# Побітове виключне або

result = self.value ^ other.value

return MyBigInt(result)

def OR(self, other):

# Побітове або

result = self.value | other.value

return MyBigInt(result)

def AND(self, other):

# Побітове і

result = self.value & other.value

return MyBigInt(result)

def shiftR(self, n):

# Зсув праворуч на n бітів

result = self.value >> n

return MyBigInt(result)

def shiftL(self, n):

# Зсув ліворуч на n бітів

result = self.value << n

return MyBigInt(result)

# Приклад використання

numberA = MyBigInt()

numberB = MyBigInt()

numberA.setHex("E035C6CFA42609B998B883BC1699DF885CEF74E2B2CC372EB8FA7E7")

numberB.setHex("5072F028943E0FD5FAB3273782DE14B1011741BD0C5CD6BA6474330")

numberC = numberA.XOR(numberB)

print("XOR result:", numberC.getHex())

numberD = numberA.OR(numberB)

print("OR result:", numberD.getHex())

numberE = numberA.AND(numberB)

print("AND result:", numberE.getHex())

numberF = numberA.INV()

print("INV result:", numberF.getHex())

numberG = numberA.shiftR(4)

print("shiftR result:", numberG.getHex())

numberH = numberA.shiftL(4)

print("shiftL result:", numberH.getHex())

**2.3 Реалізація арифметичних операцій**

Для програмної реалізації додавання і віднімання з великими числами можна використовувати наступні рекомендації. Додавати (або віднімати) числа поблоково, починаючи з молодших блоків і переходячи до старших блоків. Якщо в результаті додавання (віднімання) отримується перенос, то він зберігається і додається до наступного блоку.

Після проходження всіх блоків, перевірити, чи є перенос в останньому блоку. Якщо є, то необхідно розширити масив на один блок і додати перенос до цього блоку.

При реалізації віднімання, перед тим як віднімати один блок від іншого, слід перевірити, чи є друге число в другому блоці більшим за число в першому. Якщо друге число менше за перше, то можна віднімати поблоково без будь-яких змін. Якщо друге число більше за перше, то необхідно перенести один біт з наступного блоку в поточний.

Після додавання (або віднімання) треба перевірити, чи не виникли великі числа, що не вміщуються у результуючий масив блоків. В разі чого коригувати розмір масиву.

Основною рекомендацією щодо реалізації операцій множення і ділення є використання спеціалізованих алгоритмів, які оптимізовані для роботи з великими числами. Для множення можна використовувати алгоритм Карацуби, що дозволяє значно зменшити кількість операцій множення, порівняно з класичним методом. Або алгоритм Штрассена, що дозволяє ще більш ефективно множити великі числа. Для ділення можна використовувати алгоритм Діріхле, що є швидким і точним методом для ділення великих чисел.

Операції, які треба реалізувати на цьому етапі завдання:

* ADD (додавання)
* SUB (віднімання)
* MOD (взяття за модулем)

Операції, які можна реалізувати додатково:

* MUL (множення)
* DIV (ділення)
* POWMOD (множення за модулем)

# 3 Тестові приклади чисел

51bf608414ad5726a3c1bec098f77b1b54ffb2787f8d528a74c1d7fde6470ea4

XOR

403db8ad88a3932a0b7e8189aed9eeffb8121dfac05c3512fdb396dd73f6331c

result

1182d8299c0ec40ca8bf3f49362e95e4ecedaf82bfd167988972412095b13db8

36f028580bb02cc8272a9a020f4200e346e276ae664e45ee80745574e2f5ab80

ADD

70983d692f648185febe6d6fa607630ae68649f7e6fc45b94680096c06e4fadb

result

a78865c13b14ae4e25e90771b54963ee2d68c0a64d4a8ba7c6f45ee0e9daa65b

33ced2c76b26cae94e162c4c0d2c0ff7c13094b0185a3c122e732d5ba77efebc

SUB

22e962951cb6cd2ce279ab0e2095825c141d48ef3ca9dabf253e38760b57fe03

result

10e570324e6ffdbc6b9c813dec968d9bad134bc0dbb061530934f4e59c2700b9

7d7deab2affa38154326e96d350deee1

MUL

97f92a75b3faf8939e8e98b96476fd22

result

4a7f69b908e167eb0dc9af7bbaa5456039c38359e4de4f169ca10c44d0a416e2

У цьому прикладі реалізовані операції додавання, віднімання та взяття за модулем.

class MyBigInt:

def \_\_init\_\_(self, value=0):

self.value = value

def setHex(self, hex\_string):

# Встановлює значення з шістнадцяткового рядка

self.value = int(hex\_string, 16)

def getHex(self):

# Повертає значення у шістнадцятковому форматі

return hex(self.value).upper()[2:]

def ADD(self, other):

# Додавання

result = self.value + other.value

return MyBigInt(result)

def SUB(self, other):

# Віднімання

result = self.value - other.value

return MyBigInt(result)

def MOD(self, modulus):

# Взяття за модулем

result = self.value % modulus.value

return MyBigInt(result)

# Приклад використання

numberA = MyBigInt()

numberB = MyBigInt()

numberA.setHex("51BF608414AD5726A3C1BEC098F77B1B54FFB2787F8D528A74C1D7FDE6470EA4")

numberB.setHex("403DB8AD88A3932A0B7E8189AED9EEFFB8121DFAC05C3512FDB396DD73F6331C")

numberC = numberA.ADD(numberB)

print("ADD result:", numberC.getHex())

numberD = numberA.SUB(numberB)

print("SUB result:", numberD.getHex())

numberE = numberA.MOD(numberB)

print("MOD result:", numberE.getHex())