НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Звіт з виконання комп'ютерного практикуму

ДОСЛІДЖЕННЯ БІБЛІОТЕК БАГАТОСЛІВНОЇ АРИФМЕТИКИ

Виконали студенти
групи ФІ-32мн
Величко Олена,
Мельник Ілля,
Міснік Аліна

Перевірили:
Селюх П.В.

Київ — 2023

Мета роботи: Дослідження алгоритмів реалізації арифметичних операцій над великими (багато розрядними) числами над скінченими полями та групами з точки зору їх ефективності за часом та пам'яттю для різних програмно-апаратних середовищ.

Постановка задачі: Розглянути бібліотеки багаторозрядної арифметики, вбудовані в програмні платформи Scala (BigInt), Java (BigInteger) та Python (Mod) для процесорів із 32-розрядною архітектурою та обсягом оперативної пам'яті до 8 ГБ.

1 ХІД РОБОТИ

1.1 Теоретичний опис

Як ми знаємо, модулярна арифметика великих чисел наразі є невіддільною частиною криптографії. Завдяки її властивостям, ми можемо забезпечити стійкість до розсекречування інформації. Але проблема у тому, що перевага модулярної арифметики великих чисел є і її мінусом при реалізації у різних середовищах. По-перше, необхідно зберігати десь ці великі числа, що зазвичай не розраховані на розмір звичайних типів чисел. А по-друге, деякі алгоритми піднесення до великого степеня можуть потребувати досить великого часу обрахунку. Саме тому постала нагальна потреба у спеціальних бібліотеках, що містили б оптимальні методи обрахунку великих чисел, а також мали можливість зберігати їх без використання великої кількості пам'яті. Насправді таких бібліотек існує досить багато, але нашою задачею буде дослідити саме вбудовані бібліотеки модулярної арифметики великих чисел. Ми розглянемо три мови програмування та відповідно порівняємо три бібліотеки: Java (BigInteger), Scala(BigInt) та Python(Mod).

1.2 Java

1.2.1 Java library – BigInteger

Клас BigInteger у Java використовує деяку властивість для зберігання чисел, які перевищують обсяг пам'яті примітивних типів даних. Тобто, він зберігає значення в масиві за допомогою двійкового представлення. BigInteger по суті групує двійкове представлення в 32-розрядні частини. Це скасовує обмеження на кількість пам'яті, яку мають інші типи даних у java.

Які методи містить клас BigInteger?

Насправді їх досить багато. Відзначимо, що найголовнішими для нас

наразі є методи додавання, віднімання, множення, ділення (як із залишком, так і цілочисельно), модуль, порівняння двох чисел, піднесення до степеня за модулем. Також варто зазначити зручність подання числа, оскільки є можливість представити число у будь-якій системі числення та переведення його в іншу систему числення, система дозволяє це гнучко робити. Дивлячись на список методів у класі java.math.BigInteger, стає зрозуміло, що ця конструкція слугує декільком цілям:

1) Він представляє як завгодно довгі цілі числа у двійковому вигляді та надає аналоги всіх примітивних цілочисельних арифметичних операцій.

2) Він надає деякі додаткові корисні методи, такі як bitLength(), testBit() та генерування рівномірно випадкового n-бітового беззнакового значення.

3) Також клас містить декілька спеціальних методів для обробки всієї арифметики сирої криптосистеми RSA. Це дозволяє легко і швидко реалізувати RSA як перевірку концепції. Крім того, ці методи не часто використовуються за межами RSA, тому RSA, ймовірно, є їх основним варіантом використання.

Також, важливими для криптографії є такі вбудовані функції як:

1) gcd() – функція, яка повертає значення НСД, що часто використовується у тестах простоти.

2) isProbablePrime(int certainty) – функція для визначення простоти числа. Certainty (вірогідність) – міра невизначеності: якщо функція повертає true, то ймовірність того, що це BigInteger є простим, перевищує (1 - $\frac{1}{2}^{certainty}$). Час виконання цього методу пропорційний значенню цього параметра. Функція використовує алгоритм перевірки на простоту Міллера-Рабіна.

3) nextProbablePrime() – повертає перше ціле число, більше за BigInteger, яке, ймовірно, є простим.

4) modPow() – обчислює піднесення до степеня за заданим модулем, використовуючи алгоритм адитивного ланцюга.

Методи класу BigInteger корисні і в інших криптосистемах з

Виміри/операції	$+$	$-$	$*$	$/$	a^2	$(a^p) mod n$	$(a^-1) mod n$
Час, мкс	**0,2**	**0,3**	**0,6**	**1,8**	**1,4**	**32,2**	**28,4**

Таблиця 1.1 – Операції BigInteger in Java

відкритим ключем, які використовують обчисленням за простим модулем. Наприклад, в алгоритмі цифрового підпису (DSA), обміну ключами Діффі-Геллмана. Криптографія на еліптичних кривих (наприклад, ECDSA) використовує піднесення до степеня (після додавання, віднімання і множення) і пошук оберненого за модулем.

1.2.2 Java package – Crypto

Досить цікавою та корисною бібліотекою є Cipher, яка являє собою набір симетричних та асиметричних алгоритмів шифрування, таких як: AES, DES та RSA з різними режимами шифрування, а також можливостями автоматичного падінгу.

Бібліотека KeyAgreement створена для підтримки алгоритму обміну ключами Діффі-Геллмана.

Mac – бібліотека для обчислення кодів автентифікації, що забезпечує перевірку цілісності повідомлення.

KeyGenerator – бібліотека для генерації секретного ключа.

1.3 Scala

Ми вирішили обрати мову Scala для нашого огляду, оскільки було цікаво поглянути на те, як впорається із задачею обчислення у великій арифметиці мова, що поєднує властивості об'єктноорієнтованого та функціонального програмування.

Scala використовує JVM (*Java Virtual Machine*) для виконання свого коду. Це дає змогу використовувати оптимізації JIT-компілятора (*dynamic translation або run-time compilation*) для збільшення продуктивності. По

суті, це означає виконання програмного коду безпосередньо під час роботи програми. Оскільки мова Scala функціональна, це дає змогу зручно працювати з математичним апаратом. Основними плюсами такого підходу у Scala є:

1) типи даних визначаються як множини значень, а не набори операцій. Такі типи даних забезпечують вищий ступінь абстракції та узагальнення ;

2) lazy-оцінювання (або обчислення) — це концепція, яка дозволяє відкладати обчислення значень до того моменту, коли вони фактично стають необхідними.

1.3.1 Scala library – BigInt

Аналогічно до BigInteger у Java, BigInt це бібліотека, що дозволяє зберігати числа, які перевищують обсяг пам'яті примітивних типів даних. Якщо заглянути у середину типу BigInt, то виявиться, що дана конструкція використовує клас "java.math.BigInteger"для свого представлення. Тобто BigInt є по суті прямою похідною бібліотеки Java.

Відповідно, як і BigInteger, ця бібліотека містить усі необхідні нам методи для модулярної арифметики великих чисел.

Виміри/операції	$+$	$-$	$*$	$/$	a^2	$(a^p)modn$	$(a^-1)modn$
Час, мкс	**1,5**	**3**	**2,4**	**1,6**	**2,6**	**15,6**	**10,7**

Таблиця 1.2 – Операції BigInt in Scala

1.4 Python

1.4.1 Вбудований тип int

У мові програмування Python вбудований тип даних int не має межі розміру для цілих чисел, і вони можуть займати стільки пам'яті, скільки

доступно системі користувача.

Ціле число, подібно до кортежу або списку, має змінну довжину і підлаштовується в залежності від потреби. В структурі цього типа даних є спеціальна змінна, яка зберігає кількість елементів в масиві, що відповідає нашому числу, і може бути перевизначена для підвищення ефективності виділення пам'яті.

Крім цього, замість зберігання тільки однієї десяткової цифри в кожному елементі масиву числа, Python перетворює числа із системи числення з основою 10 у числа в системі з основою 2^{30} і викликає кожен елемент як цифру, значення якої коливається від 0 до $2^{30} - 1$.

У шістнадцятковій системі числення, основа $16 = 2^4$ означає, що кожна «цифра» шістнадцяткового числа коливається від 0 до 15 в десятковій системі числення. У Python аналогічно, «число» з основою 2^{30}, що означає, що число варіюватиметься від 0 до $2^{30} - 1 = 1073741823$ у десятковій системі числення.

Таким чином, Python ефективно використовує майже весь виділений простір в 32 біти на одну цифру, економить ресурси і все ще виконує прості операції, такі як додавання та віднімання.

Залежно від платформи Python використовує або 32-бітові цілочислові беззнакові масиви, або 16-бітові цілочисельні беззнакові масиви з 15-бітними цифрами. Отже, для виконання складних операцій з дуже великими числами знадобиться лише кілька бітів і немає потреби у використанні спеціалізованих бібліотек.

1.4.2 Python library – Mod

Для модулярної арифметики в Python використовується вбудована бібліотека Mod, яка містить функції для виконання операцій з модулем числа. Завдяки цій бібліотеці можна легко і швидко знаходити обернені за модулем числа, перемножати числа за модулем, а також підносити за модулем одне число в степінь іншого, не кажучи вже про найпростіші математичні операції такі як додавання та віднімання.

Отже, за допомогою типу даних int були записані великі числа довжини 512 біт, потім до них були застосовані арифметичні операції за модулем за допомогою бібліотеки Mod, після чого ми порахували час виконання кожної операції, який можна побачити в таблиці.

Виміри/операції	$+$	$-$	$*$	$/$	a^2	$(a^p) mod n$	$(a^{-}1) mod n$
Час, мкс	**9**	**8**	**8**	**9**	**9**	**187**	**43**

Таблиця 1.3 – Операції Mod in Python

ВИСНОВКИ

У даній роботі ми провели аналіз трьох бібліотек, що підтримують арифметику великих чисел: BigInteger (Java), BigInt (Scala) та Mod (Python). Кожна з бібліотек має широкий спектр застосування та багато різних методів. Для аналізу були обрані числа довжини 512 бітів.

Якщо порівнювати BigInteger та BigInt, то на простих операціях BigInteger показує себе краще за BigInt, а на складних навпаки – бібліотека Java програє Scala. Однозначно насправді важко сказати, чому саме так відбувається. У вузькому спектрі розгляду – BigInt використовує BigInteger для своїх методів. Можливо, оптимізація Java бібліотеки є більшою на простих операціях ніж у її суперниці. Також не варто забувати, що про властивість Scala – функціональність. Методи відкладання обчислень є дуже зручними, особливо у контексті важких обчислень, такі як піднесення до великого степеня за модулем.

Аналізуючи досліджені бібліотеки, можна відмітити, що найменш оптимальною по швидкості є бібліотека Mod на Python, не дивлячись на те, що ця мова програмування оптимізована і використовує ефективну систему числення для зберігання масивів чисел, отже повинна мати перевагу і в часі обчислень, і у виділеній пам'яті. Причиною такого результату може бути те, що в Python немає спеціалізованої бібліотеки для обчислення великих чисел, тому арифметичні операції можуть бути менш оптимізовані ніж в інших мовах програмування.