Практичне завдання:

«Робота з великими числами. Логічні та арифметичні операції»

Мета практичного завдання: отримати практичний досвід зі зберігання та обробки великих натуральних чисел під час програмування криптографічних алгоритмів.

1 Теоретичні відомості

У криптографії для безпеки даних часто використовуються великі натуральні числа. Обробка цих чисел може займати значну кількість часу та ресурсів обчислювальної системи. Одна з теорій полягає в тому, що важливо ефективно зберігати та обробляти великі числа шляхом використання різних оптимізацій, таких як розбиття чисел на менші блоки, використання алгоритмів ділення та множення зі зменшенням розрядності до найбільш ефективної, побітова обробка чисел та використання ефективних структур даних. Крім того, важливо враховувати вимоги до безпеки та захисту даних під час їх обробки.

Розбиття числа на блоки фіксованої довжини використовується для забезпечення ефективності обчислень при роботі з великими числами в криптографії. Наприклад, при роботі з алгоритами блочного симетричного шифрування або алгоритмами гешування (AES, SHA та інші), де використовуються великі числа для ключів, блоків та проміжних даних. Розбиття числа на блоки з фіксованою довжиною дозволяє зменшити розрядність чисел і тим самим покращити швидкодію обчислень.

Вибір довжини блоків може залежати від конкретного алгоритму, але часто використовують блоки по 8, 16, 32, 64 або 128 бітів. Також важливо виконувати операції з беззнаковими числами, оскільки в криптографії використовуються лише додатні значення, і операції зі знаковими числами можуть призвести до помилок в обчисленнях.

Для роботи з великими числами важливо розуміти, яким чином вони представлені в пам'яті комп'ютера, а також які операції над ними можуть бути виконані. Розбиття чисел на блоки з фіксованою довжиною є одним з ефективних способів роботи з великими числами в криптографії.

Наприклад, якщо необхідно обчислити результат операції XOR з двох чисел довжиною по 256 бітів, ефективним рішенням буде розбити кожне число на 8 блоків по 32 біти. Далі, 8 разів застосувати операцію XOR до кожної пари блоків, а потім сконкатенувати вісім проміжних результатів в одне велике число.

Цим самим принципом можна виконувати інші побітові операції, такі як побітовий зсув, побітова інверсія, побітові AND / OR. Для реалізації арифметичних операцій, таких як додавання і віднімання, потрібно додатково враховувати біт переносу. Але для операцій множення, ділення і піднесення до ступеню за модулем краще використовувати спеціалізовані алгоритми.

Загальні рекомендації щодо зберігання та обробки великих натуральних чисел при програмуванні криптографічних алгоритмів:

- використовуйте відповідний тип даних. Для зберігання великих натуральних чисел рекомендується використовувати тип даних змінної довжини (BigInteger, BigNumber або власну реалізацію);
- завжди перевіряйте коректність введених даних та забезпечуйте перевірку на розмір чисел. Великі числа можуть займати багато пам'яті та призводити до переповнення буферів;
- використовуйте ефективні алгоритми для виконання арифметичних операцій з великими числами, такі як алгоритм Карацуби для множення чисел або алгоритм Шенхаге – Штрассена для піднесення до ступеню за модулем;
- забезпечуйте безпеку при обробці та передачі великих чисел. Для захисту від атак на середовище виконання програми та від атак на передачу даних рекомендується використовувати криптографічні протоколи та захист від буферних переповнень;
- дотримуйтеся принципу ефективного використання розрядності процесора. Якщо у вас 32-бітний процессор, розділяйте велике число на блоки довжиною 32 біти. Якщо у вас 64-бітний процессор, розділяйте велике число на блоки довжиною 64 біти. Це забезпечить ефективність використання пам'яті та збільшить швидкість обчислень.

2 Хід роботи

Для виконання практичного завдання вам треба реалізувати власну бібліотеку для зберігання і обробки великих натуральних чисел. Ви можете використовувати будь-яку зручну для вас мову програмування, але уникати готових реалізацій і бібліотек. Натомість отримати власний досвід програмної обробки великих чисел і написати автоматичні тести для перевірки правильності обчислень.

Етапи виконання практичного завдання:

Етап	Стисле формулювання задачі	Максимальна кількість балів
1	Реалізація власного типу даних великого числа з методами setHex і getHex	3
2	Реалізація побітових операцій для власного типу даних	3
3	Реалізація арифметичних операцій для власного типу даних	4

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.

2.2 Реалізація побітових операцій

Реалізуйте методи, що будуть виконувати побітові операції з об’єктами 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 епаті 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.

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
```