Міністерство освіти і науки України
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
Фізико-технічний інститут

“Спеціальні розділи обчислювальної математики”

Комп’ютерний практикум

Робота №1. Багаторозрядна арифметика

Виконала
студентка гр. ФБ-11 Данькова Єлізавета

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 згідно з варіантом №13

А) Згідно варіанту розробити клас чи бібліотеку функцій для роботи з m-бітними цілими числами. Бібліотека повинна підтримувати числа довжини до 2048 біт. Повинні бути реалізовані такі операції:

1) переведення малих констант у формат великого числа (зокрема, 0 та 1);

2) додавання чисел;

3) віднімання чисел;

4) множення чисел, піднесення чисел до квадрату;

5) ділення чисел, знаходження остачі від ділення;

6) піднесення числа до багаторозрядного степеня;

7) конвертування (переведення) числа в символьну строку та обернене перетворення символьної строки у число; обов’язкова підтримка шістнадцяткового представлення, бажана – десяткового та двійкового.

Бажано реалізувати такі операції:

1) визначення номеру старшого ненульового біта числа;

2) бітові зсуви (вправо та вліво), які відповідають діленню та множенню на степені двійки.

ХІД РОБОТИ:

Додавання

```
def LongAddition(a, b):
    max_len = max(len(a), len(b))
    carry = 0
    sum = []
    for i in range(max_len):
        temp_A = int(a[i]) if i < len(a) else 0
        temp_B = int(b[i]) if i < len(b) else 0
        temp = temp_A + temp_B + carry
        sum.append(temp & (2 ** 32 - 1))
        carry = temp >> 32
    if carry > 0:
        sum.append(carry)
    return sum
```

Алгоритм:

1. Визначення максимальної довжини:

Визначається максимальна довжина з двох списків, що представляють великі числа.

2. Ініціалізація переносу і суми:

Ініціалізуються змінні для переносу та для збереження результату суми. Перенос використовується для збереження значення, яке перевищує 32 біти при додаванні.

3. Проходження по всіх елементах списків:

Цикл проходить через всі елементи списків, від першого до останнього елемента довшого з двох списків.

4. Отримання значень з обох списків або нуль, якщо індекс виходить за межі:

Для кожного індексу перевіряється, чи існує елемент в обох списках. Якщо елемент існує, береться його значення, якщо ні, використовується значення нуль.

5. Додавання чисел і переносу:

Виконується додавання відповідних елементів з обох списків разом з поточним значенням переносу.

6. Збереження результату додавання і оновлення переносу:

Нижні 32 біти результату додавання зберігаються в список суми. Вищі 32 біти зберігаються як нове значення переносу.

7. Додавання залишкового переносу:

Якщо після завершення циклу значення переносу більше нуля, воно додається до результату як додатковий елемент.

8. Повернення результату:

Повертається список, що містить результат додавання двох великих чисел.

Віднімання

```
def LongSubstration(a, b):
    borrow = 0
    sub = []
    max_len = max(len(a), len(b))
    for i in range(max_len):
        temp_A = int(a[i]) if i < len(a) else 0
        temp_B = int(b[i]) if i < len(b) else 0
        temp = temp_A - temp_B - borrow
        if temp >= 0:
            sub.append(temp)
            borrow = 0
        else:
            sub.append((2 ** 32 + temp))
            borrow = 1
    if borrow != 0:
        return None
    else:
        while len(sub) > 1 and sub[-1] == 0:
            sub.pop()
        return sub
```

Алгоритм:

1. Ініціалізація запозичення і різниці:

- Змінна `borrow` використовується для збереження запозичення під час віднімання.

- Змінна `sub` ініціалізується як порожній список для збереження результату віднімання.

2. Визначення максимальної довжини:

- Визначається максимальна довжина з двох списків, що представляють великі числа.

3. Проходження по всіх елементах списків:

- Цикл проходить через всі елементи списків, від першого до останнього елемента довшого з двох списків.

4. Отримання значень з обох списків або нуль, якщо індекс виходить за межі:

- Для кожного індексу перевіряється, чи існує елемент в обох списках. Якщо елемент існує, береться його значення, якщо ні, використовується значення нуль.

5. Віднімання чисел і обробка запозичення:

- Виконується віднімання відповідних елементів з обох списків разом з поточним значенням запозичення.

- Якщо результат віднімання не негативний, він додається до списку різниці, а запозичення встановлюється в 0.

- Якщо результат віднімання негативний, до результату додається 2^32 (що еквівалентно позиченню одного бітового блоку), і запозичення встановлюється в 1.

6. Перевірка на наявність залишкового запозичення:

- Якщо після завершення циклу значення запозичення не дорівнює нулю, функція повертає `None`, що вказує на помилку (наприклад, якщо `a` було менше `b` і це не оброблено).

7. Видалення провідних нулів:

- Після завершення віднімання видаляються провідні нулі в результаті, щоб зберегти правильне представлення числа.

8. Повернення результату:

- Повертається список, що містить результат віднімання двох великих чисел.

Множення

```
def LongMultiply(a, b):
    if LongCompare(a, b) == -1:
        return LongMultiply(b, a)

    mul = [0] * (len(a) + len(b))

    for i in range(len(b)):
        carry = 0
        for j in range(len(a)):
            temp = mul[i + j] + a[j] * b[i] + carry
            mul[i + j] = temp & (2 ** 32 - 1)
            carry = temp >> 32

        if carry > 0:
            mul[i + len(a)] += carry

    while len(mul) > 1 and mul[-1] == 0:
        mul.pop()

    return mul
```

Алгоритм:

1. Перевірка і обмін чисел для оптимізації:

- Якщо перше число (`a`) менше другого (`b`), функція викликає сама себе з оберненими аргументами.

2. Ініціалізація результату множення:

- Створюється список `mul`, який має розмір, рівний сумі довжин обох чисел. Він заповнений нулями і буде використовуватися для зберігання результату множення.

3. Основний цикл множення:

- Зовнішній цикл проходить по всіх елементах списку `b`.

- Внутрішній цикл проходить по всіх елементах списку `a`.

- Кожен елемент `a[j]` множиться на `b[i]`, і результат додається до відповідного елемента `mul[i + j]`, разом з поточним значенням `carry`.

- Результат множення зберігається в `mul[i + j]` після взяття по модулю 2^32, щоб залишити тільки 32 біти.

- Перенос (`carry`) обчислюється шляхом зсуву на 32 біти вправо.

4. Додавання залишкового переносу:

- Після завершення внутрішнього циклу, якщо значення переносу (`carry`) більше 0, воно додається до наступного елемента результату `mul`.

5. Видалення провідних нулів:

- Після завершення всіх операцій множення видаляються всі провідні нулі з кінця списку результату, щоб отримати коректне представлення числа.

6. Повернення результату:

- Повертається список `mul`, що містить результат множення двох великих чисел.

Квадрат

```
def LongSquare(a):
    result = LongMultiply(a, a)
    return result
```

Ділення

```
def LongDivideModule(a, b):
    if a == b:
```

```
        return ([1], [0])
    k = BitLength(b)
    remainder = a.copy()
    quotient = [0]
    while LongCompare(remainder, b) != -1:
        t = BitLength(remainder)
        c = LongShiftBitsToHigh(b, t - k)
        if LongCompare(remainder, c) < 0:
            t -= 1
            length = t - k
            c = LongShiftBitsToHigh(b, length)
        remainder = LongSubstration(remainder, c)
        temp = LongShiftBitsToHigh([1], t - k)
        quotient = LongAddition(quotient, temp)
    return (quotient, remainder)
```

Алгоритм:

1. Перевірка рівності чисел:

Якщо числа a і b рівні, функція повертає частку [1] і залишок [0].

2. Ініціалізація змінних:

Змінна k визначає довжину числа b в бітах.

Змінна remainder (залишок) ініціалізується копією числа a.

Змінна quotient (частка) ініціалізується списком з одним елементом [0].

3. Основний цикл ділення:

Цикл продовжується, поки залишок remainder не стане меншим за b.

Змінна t визначає довжину залишку в бітах.

Змінна c представляє число b, зсунуте вліво на t - k бітів.

Якщо залишок remainder менший за c, значення t зменшується на 1, а c зсувається на t - k бітів.

4. Оновлення залишку і частки:

Залишок remainder зменшується на значення c.

Частка quotient збільшується на одиницю, зсунуту вліво на t - k бітів.

5. Повернення результату:

Повертається частка quotient і залишок remainder.

Степінь

```
def LongPower(a, b):
    pow = [1]
```

```
    for i in range(BitLength(b)):
        if BitCheck(b, i) == 1:
            pow = LongMultiply(pow, a)
        a = LongMultiply(a, a)
        while len(a) < len(pow):
            a.append(0)
    return pow
```

Приклад виконання:

A + B =

151a9ee8090b7e138ffddf45744afc80bd4757a
5684bdb3b109691177a8d11c7cc93c3cc69763
70d7e64aa2be7d174c334f23ef536aae0995c1
b6afd1762690f2d8f88cab6a2f5474c313fa7019
d6d37d30cdf45dcfff2e2c1d3ae49233ed7915a
3f17c94e3322ca0b76a43b691b62c5c363f047e
9b8b0e4f59a4e6a33326af15

A - B =

99aa49257f25ddf4b6487febd09c433606f8424
8ceeac66f1045a1a9bec251c9c6c2ca2c40fb9ff
eae674366f4642f460f7c1fce732552186af9be3
4e43113defa55ecbeb1659f97f262a844181a1c
fc0d69c0ecb0152831f05da0b33331099d96c5
d9ae522dbb6f361ac3f26206fc42e3c81fe845b
13576bba21acfcd26a951

A * B =

5848f681cd11fd72def2ed3613d51f3881e1120
2e2288be3aa35d877af44c3dbcaf8e49b59ac3c
34a626a2f65307f1b74985193aadc72203c0fc9
3dbad2e49c000e59bacf45a2fbf45af0171d563
91c17dd97bd797f76fa11bbf59a5ed10960642
608298434be9f8b9437d9aa759e378d6c3f34a
1e1dd7ffe44224916a52363d510e7b01236f76

A / B =

2

A mod B =

3daa7677f65cdc52917dc5b6169280cb2039a6
41f4014fce13b3e9c2c9baec70458610df15c7b
792127593bb300aa0cf6fa8380bf760f659bf9b
45669b3655550b049cc85500c5298a09ff2e173
bc1bb7bb7a7342020253 4d9eefec3b2d2d1e1
9030083b09ab82d6757304104a2f64363a8cad
9d1ac4490a6ca0b4e61a26a66f

A ^{2}

ebbf0a4ae47bf2da2adb16cf57a5614df4d44c9
43931e04ddcb3ee41f53916262e41f2ef5c8fe4
ec5bd99ae1f18a22d7984e6d83c28800f9980b
018e963ab79d7255bd032d4c08e90597791d5
3181b370800f68929d9a66e2d71fd41c3c291e
cff2f840a2ee0d44fea25a495d82569b0ce5debf
67024e86f22d44c5b9b26809a3e23bf7248073
a6367fe88f0675fe60fd195650f48e91450f0f99
4893b5ed4adc8a76803187621e0adb8302270
1dcc06d5935a19501697068a7cadd09e48b872
02164f35a51f6bc3b19405394db5567e195147
149b0f46534dc36e242a167fb7d623351d3d72
3a406471ba8b46f63eb69db8193a16e4c5b669
3093c092f89229

Performance calculation example

Operand length: 1024 bits

Experiment count: 1000
Running 1000 experiments on 1024-bit bigints; operation: +
 - Average operation duration: 0.00e+00 seconds
Running 1000 experiments on 1024-bit bigints; operation: -
 - Average operation duration: 0.00e+00 seconds
Running 1000 experiments on 1024-bit bigints; operation: *
 - Average operation duration: 1.99e-06 seconds
Running 1000 experiments on 1024-bit bigints; operation: /
 - Average operation duration: 6.63e-08 seconds

Experiment count: 10000
Running 10000 experiments on 1024-bit bigints; operation: +
 - Average operation duration: 5.05e-08 seconds
Running 10000 experiments on 1024-bit bigints; operation: -
 - Average operation duration: 9.97e-08 seconds
Running 10000 experiments on 1024-bit bigints; operation: *
 - Average operation duration: 1.87e-06 seconds
Running 10000 experiments on 1024-bit bigints; operation: /
 - Average operation duration: 3.47e-07 seconds

Experiment count: 100000
Running 100000 experiments on 1024-bit bigints; operation: +
 - Average operation duration: 7.87e-08 seconds
Running 100000 experiments on 1024-bit bigints; operation: -
 - Average operation duration: 1.46e-07 seconds

Running 100000 experiments on 1024-bit bigints; operation: *
 - Average operation duration: 1.43e-06 seconds
Running 100000 experiments on 1024-bit bigints; operation: /
 - Average operation duration: 1.80e-07 seconds

Operand length: 2048 bits

Experiment count: 1000
Running 1000 experiments on 2048-bit bigints; operation: +
 - Average operation duration: 0.00e+00 seconds
Running 1000 experiments on 2048-bit bigints; operation: -
 - Average operation duration: 0.00e+00 seconds
Running 1000 experiments on 2048-bit bigints; operation: *
 - Average operation duration: 5.20e-06 seconds
Running 1000 experiments on 2048-bit bigints; operation: /
 - Average operation duration: 0.00e+00 seconds

Experiment count: 10000
Running 10000 experiments on 2048-bit bigints; operation: +
 - Average operation duration: 2.03e-07 seconds
Running 10000 experiments on 2048-bit bigints; operation: -
 - Average operation duration: 2.54e-07 seconds
Running 10000 experiments on 2048-bit bigints; operation: *
 - Average operation duration: 5.20e-06 seconds
Running 10000 experiments on 2048-bit bigints; operation: /
 - Average operation duration: 1.62e-07 seconds

Experiment count: 100000
Running 100000 experiments on 2048-bit bigints; operation: +

- Average operation duration: 9.67e-08 seconds
Running 100000 experiments on 2048-bit bigints; operation: -
- Average operation duration: 1.57e-07 seconds
Running 100000 experiments on 2048-bit bigints; operation: *
- Average operation duration: 5.10e-06 seconds
Running 100000 experiments on 2048-bit bigints; operation: /
- Average operation duration: 1.82e-07 seconds

Operand length: 4096 bits

Experiment count: 1000
Running 1000 experiments on 4096-bit bigints; operation: +
- Average operation duration: 9.93e-07 seconds
Running 1000 experiments on 4096-bit bigints; operation: -
- Average operation duration: 0.00e+00 seconds
Running 1000 experiments on 4096-bit bigints; operation: *
- Average operation duration: 1.58e-05 seconds
Running 1000 experiments on 4096-bit bigints; operation: /
- Average operation duration: 2.65e-08 seconds

Experiment count: 10000
Running 10000 experiments on 4096-bit bigints; operation: +
- Average operation duration: 2.69e-07 seconds
Running 10000 experiments on 4096-bit bigints; operation: -
- Average operation duration: 4.30e-07 seconds
Running 10000 experiments on 4096-bit bigints; operation: *
- Average operation duration: 1.57e-05 seconds
Running 10000 experiments on 4096-bit bigints; operation: /
- Average operation duration: 2.04e-07 seconds

Experiment count: 100000

Running 100000 experiments on 4096-bit bigints; operation: +

- Average operation duration: 2.41e-07 seconds

Running 100000 experiments on 4096-bit bigints; operation: -

- Average operation duration: 3.15e-07 seconds

Running 100000 experiments on 4096-bit bigints; operation: *

- Average operation duration: 1.54e-05 seconds

Running 100000 experiments on 4096-bit bigints; operation: /

- Average operation duration: 4.77e-07 seconds