Міністерство освіти і науки України

Національний технічний університет України

"Київський політехнічний інститут імені Ігоря Сікорського"

Навчально-науковий Фізико-технічний інститут

Спеціальні розділи обчислювальної математики

Комп'ютерний практикум №1

Багаторозрядна арифметика

Виконав:
студент групи ФІ-12
Юрчук Олексій

Київ 2024

Тема: Багаторозрядна арифметика

Мета: Отримання практичних навичок програмної реалізації багаторозрядної арифметики; ознайомлення з прийомами ефективної реалізації критичних по часу ділянок програмного коду та методами оцінки їх ефективності.

Завдання:

А) 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.
Б) Проконтролювати коректність реалізації алгоритмів; наприклад, для декількох багаторозрядних a,b,c,n перевірити тотожності:
1. $(a+b)\cdot c = c\cdot(a+b) = a\cdot c + b\cdot c$;
2. $n\cdot a = \underbrace{a+a+...+a}_{n}$, де n повинно бути не менш за 100;

В) Обчислити середній час виконання реалізованих арифметичних операцій. Підрахувати кількість тактів процесора (або інших одиниць виміру часу) на кожну операцію. Результати подати у вигляді таблиць або діаграм.

Теоретичні відомості:

- Представлення багаторозрядних чисел

Натуральне число N завжди можна представити у системі числення із основою β:

$$N = \overline{a_{n-1}a_{n-2}...a_1a_0}_{\,\beta} = a_{n-1}\beta^{n-1} + a_{n-2}\beta^{n-2} + ... + a_1\beta + a_0,$$

де $n = \left|\log_\beta N\right|$ – довжина числа у даній системі числення, $a_i \in \{0, 1, ..., \beta-1\}$ – окремі цифри даного представлення. Зручним та природним способом зберігання таких чисел виявляються звичайні масиви.

- Додавання та віднімання багаторозрядних чисел

Отже, нехай дано два числа A та B фіксованої довжини n у системі числення із основою $\beta = 2^w$:

$$A = a_{n-1}\beta^{n-1} + \ldots + a_1\beta + a_0,$$
$$B = b_{n-1}\beta^{n-1} + \ldots + b_1\beta + b_0,$$

і необхідно обчислити їх суму $C = A + B$. Зазвичай сума двох чисел може бути на одну цифру довша за довжину аргументів (з'являється значуща цифра c_n); у моделі із фіксованою довжиною ця цифра або нехтується, або повертається окремим аргументом.

Додавання чисел виконується поцифрово, із використанням додаткової змінної *carry*, що містить так званий *біт переносу*: частину суми двох цифр, що виходить за допустимі межі для цифри, а тому повинен додаватись до наступної цифри результату.

По-перше, змінна temp повинна містити $w+1$ значущий біт (чому?), отже, якщо w дорівнює розміру регістру процесора, значення temp буде обчислюватись некоректно.

По-друге, здається, що алгоритм додавання містить багато важких операцій ділення та остачі від ділення, тобто він має бути повільним. Однак насправді це не так: ми будемо широко використовувати той факт, що ми працюємо у двійковій архітектурі, в якій ділення на степінь двійки є лише бітовим зсувом вправо (операція `lsr`, або >>), а остача від ділення на степінь двійки – це останні w біт числа, які можна знайти за допомогою операції логічного ТА (`and`, або `&`). Остаточно процедура додавання матиме такий вигляд:

Процедура LongAdd (A, B, C, carry)
Вхід: багаторозрядні числа A, B
Вихід: багаторозрядне число $C = A + B$; біт переносу carry, що виходить за довжину *C*.

```
carry := 0;
for i := 0 to n-1 do:
      temp  := a[i] + b[i] + carry;
      c[i]  := temp & (2^w - 1);          // чому саме так?
      carry := temp >> w;
return C, carry
```

Аналогічним чином будується процедура віднімання двох багаторозрядних чисел; вона буде використовувати *біт запозичення* borrow, який показує, що у молодших розрядах виникла від'ємна різниця, яку потрібно компенсувати за рахунок даного розряду. Якщо по закінченню процедури borrow не дорівнює нулю, то в процедурі віднімалось більше число від меншого; в залежності від реалізації ця ситуація або повинна оброблятись штатним чином, або викликати помилку.

Процедура LongSub (A, B, C, borrow)
Вхід: багаторозрядні числа A, B
Вихід: багаторозрядне число $C = A - B$; фінальний біт запозичення borrow.

```
borrow := 0;
for i := 0 to n-1 do:
      temp  := a[i] - b[i] - borrow;
      if temp >= 0 then:
            c[i]   := temp;
            borrow := 0;
      else:
            c[i]   := 2^w + temp;
            borrow := 1;
return C, borrow
```

- ## Порівняння багаторозрядних чисел

Реалізація операції порівняння може бути виконана в різний спосіб. Найпростіший варіант – це використання описаної вище процедури LongSub: дійсно, після виконання цієї процедури за значенням біту borrow можна зробити висновок про відносні значення аргументів: якщо borrow дорівнює нулю, то $A \geq B$, а якщо одиниці, то $A < B$.

Часто, однак, за результатом порівняння необхідно чітко виокремити всі три випадки $A > B$, $A = B$ та $A < B$. У моделі з фіксованою довжиною іноді простіше виконувати порівняння чисел згідно «шкільного» визначення.

Процедура LongCmp(A, B, r)
Вхід: багаторозрядні числа A, B
Вихід: результат порівняння *r*: 0, якщо числа рівні; 1, якщо $A > B$; –1, якщо $A < B$.

```
i := n-1;
while (a[i] = b[i]) do:
      i := i-1;

if (i = -1) then:             // всі цифри однакові
      return 0;
else:
      if a[i] > b[i] then:
            return 1
      else:
            return -1;
```

Множення та піднесення до квадрату багаторозрядних чисел

Множення багаторозрядних чисел відбувається дуже подібно до додавання. Дійсно, розглянемо добуток двох багаторозрядних чисел:

$$A \cdot B = A \cdot (b_{n-1}\beta^{n-1} + \ldots + b_1\beta + b_0) = (A \cdot b_{n-1})\beta^{n-1} + \ldots + (A \cdot b_1)\beta + A \cdot b_0 .$$

Бачимо, що для обчислення добутку необхідно навчитись множити багаторозрядні числа на одну цифру та на степінь β. Однак в нашій моделі чисел множення на степені β відбувається майже миттєво: по суті, це лише зсув комірок відповідного масиву цифр!

Таким чином, ми можемо зосередитись на першій підпроцедурі – множенню на одну цифру. Для спрощення опису відійдемо від моделі із фіксованою довжиною числа та будемо вважати, що, в загальному випадку, множення двох *n*-розрядних чисел дає 2*n*-розрядний результат.

Процедура LongMulOneDigit (A, b, C)
Вхід: багаторозрядне число A довжини *n*, цифра b.
Вихід: багаторозрядне число $C = A \cdot b$ довжини $n+1$.

```
carry := 0;
for i := 0 to n-1 do:
      temp  := a[i] * b + carry;
      c[i]  := temp & (2^w - 1);
      carry := temp >> w;          // скільки значущих біт містить carry?
C[n] := carry;
return C
```

Відповідно, процедура множення двох багаторозрядних чисел LongMul буде використовувати допоміжні підпроцедури LongMulOneDigit, описану вище, та LongShiftDigitsToHigh, яка зсуває комірки масиву цифр у бік старших індексів.

Процедура LongMul (A, B, C)
Вхід: багаторозрядні числа A, B довжини n.
Вихід: багаторозрядне число $C = A \cdot B$ довжини $2n$.

```
C := 0;
for i := 0 to n-1 do:
      temp := LongMulOneDigit(A, b[i]);
      LongShiftDigitsToHigh(temp, i);
      C := C + temp;          // багаторозрядне додавання!
return C
```

Піднесення чисел до квадрату, з одного боку, може бути обчислене як звичайне множення числа на само себе. З іншого боку, під час піднесення до квадрату серед одноцифрових добутків майже половина буде дублюватись; відштовхуючись від цього, можна побудувати реалізацію процедури LongSquare, що буде майже вдвічі швидшою за LongMul.

- ## Ділення багаторозрядних чисел

Ділення є однією з самих складних арифметичних операцій, оскільки навіть у простому варіанті «в стовпчик» вимагає виконання багатьох інших дій та пошукових евристик (знаходження цифр частки). В криптографічних застосуваннях намагаються зменшити кількість використовуваних операцій ділення, задля чого розробляються спеціальні алгоритми медулярної арифметики.

Однак повністю позбавитись ділення майже неможливо. Нижче наводиться один з найпростіших варіантів алгоритму ділення, що оперує із багаторозрядними числами у двійковій формі запису; зауважимо, що числа у системі числення із основою $\beta = 2^w$ легко переводяться у двійкову форму: достатньо кожну цифру перевести у двійковий запис, виділивши на неї w біт, а потім склеїти результати у порядку старшинства. Обернений перехід також не вимагає особливих зусиль.

Алгоритм, що пропонується, має неофіційну назву «зсувай@віднімай», оскільки основними його кроками є зсув дільника на максимально можливу кількість біт в сторону старших розрядів та віднімання його від подільного. Фактично цей алгоритм описує ділення в стовпчик у двійковій системі числення, однак використовує всі переваги останнього – зокрема, спрощення всіх операцій.

Процедура LongDivMod (A, B, Q, R)
Вхід: багаторозрядні числа A, B.
Вихід: багаторозрядні частка Q та остача від ділення R: $A = B \cdot Q + R$, $0 \le R < B$.

```
k := BitLength(B);
R := A;
Q := 0;
while R >= B do:             // багаторозрядне порівняння!
      t := BitLength(R);
      C := LongShiftBitsToHigh(B, t - k);
      if R < C then:         // багаторозрядне порівняння! вийшло забагато?
            t := t - 1;      // тоді повертаємось на біт назад
            C := LongShiftBitsToHigh(B, t - k);
      R := R - C;
      Q := Q + 2^(t - k);    // встановити в Q біт із номером (t - k)
return Q, R
```

- ## Піднесення багаторозрядних чисел до багаторозрядного степеня

Остання базова арифметична операція, яку ми ще не розглядали – це піднесення до степеня. Для зручності в цьому розділі ми також відходимо від моделі із фіксованою довжиною числа і вважатимемо, що результат піднесення до степеню повертається потрібної (порівняно великої) довжини.

Прямолінійний спосіб обчислення виразу A^B передбачає B раз перемножити майбутній результат на число A; цей спосіб вимагає B множень багаторозрядних чисел (де само число B чималеньке), а тому він не підходить для практичного застосування.

Значно ефективніший метод піднесення до степеня дає так звана *схема Горнера*, перенесена на цю задачу з задачі ефективного обчислення поліномів у точці. Схема Горнера має декілька можливих реалізацій та працює із двійковим записом степеня B.

Нехай $B = b_{m-1}2^{m-1} + \ldots + b_1 2 + b_0$, де $b_i \in \{0, 1\}$ – окремі біти числа B. Тоді можемо записати:

$$A^B = A^{b_{m-1}2^{m-1}+\ldots+b_1 2+b_0} = \prod_{i=0}^{m-1}\left(A^{2^i}\right)^{b_i}.$$

Бачимо, що результат можна одержати шляхом множення послідовних возведень числа A до квадрату, причому біти b_i фактично вказують, включається поточна степінь до результату чи не включається. Звідси маємо такий алгоритм піднесення до степеня.

Процедура LongPower1 (A, B, C)

Вхід: багаторозрядні числа A, B; B задане двійковим записом $B = b_{m-1}2^{m-1} + \ldots + b_1 2 + b_0$.

Вихід: багаторозрядне число $C = A^B$.

```
C := 1;
for i := 0 to m-1 do:
      if b[i] = 1 then:
            C := C * A;       // багаторозрядне множення!
      A := A * A;             // багаторозрядне множення!
return C
```

Бачимо, що схема Горнера виконує m піднесень до квадрату та не більш ніж m множень, тобто усього $\le 2\log B$ багаторозрядних множень (на відміну від прямолінійного способу, яке вимагає B множень).

Інший варіант схеми Горнера базується на такому представленні піднесення до степеня:

$$A^B = A^{b_{m-1}2^{m-1}+\ldots+b_1 2+b_0} = \left(\ldots\left(\left(A^{b_{m-1}}\right)^2 \cdot A^{b_{m-2}}\right)^2 \cdot \ldots \cdot A^{b_1}\right)^2 \cdot A^{b_0}.$$

В даному варіанті ми йдемо не від молодших бітів B, а від старших, на кожному кроці підносимо до квадрату *результат* та, якщо потрібно, множимо його на A. Маємо таку процедуру.

Процедура LongPower2 (A, B, C)

Вхід: багаторозрядні числа A, B; B задане двійковим записом $B = b_{m-1}2^{m-1} + \ldots + b_1 2 + b_0$.

Вихід: багаторозрядне число $C = A^B$.

```
C := 1;
for i := m-1 to 0 do:
      if b[i] = 1 then:
            C := C * A;       // багаторозрядне множення!
      if i <> 0 then:         // на останньому кроці до квадрату не підносимо
            C := C * C;       // багаторозрядне множення!
return C
```

Хід роботи

Введені числа:

number_one =
c2d1e0f1efed847dccb876543210fedcba9876543210fedcba9876543210fedcba9876d2c3b4a5e1d2a32edb098fa7f5e4d3c2b1a0f9e8d7c6b5a4f3e2d1c0b9a8f7e6d5c4b3a2f1e0d9c8b7

number_two =
6b5a4f3d7c6b5a97786a5b4c3d2e1f0e1d28586977c3b4a858697786a5b4c3d2e1f0e1d2c3b8e1f4a5968778695a4b3c2d1e0f86a5b4c3d2e1f0e1d2c3b4a86a5b4c3d24f3977c3b4a0e1d2586

number_three =
f4a5b6c7d8e9f0a1b2c3d4e5f7af890bde23a2d852d1e0a4b3c7793f42d3c4b5a68a4b3c2d1e0f1e2d3c4b5a68f0e1d2c3b4a5968778695a4f1e8b3c2d1e0bafecabfefcabcabcabcacab7d8e9f0a1b2c3d4e5f7af890bde23a2d1e5a4b3c2d6789abcdef0123456789abcdef01cabca01

Перевірка властивостей:

```
Check the properties of numbers:
First property:         a*b + a*c = a*(b+c)
ba2e26c5e0c300f45998375b78f31aa45505383881c2151ddae73981e26bbe8bddb91e9390c7926a846e92f66096ae22689b390f508af6e445a4ef6d8925931
6a8d75da84090a1fde116ac6b75c50269528384bf879420defd971d0c0403c7377c56e2950acb6a7342a101d9bd3da406c38dd91a6153c3bcb502281af9c3da
ba1bf4690e10c5ed876d57ce2695f330

ba2e26c5e0c300f45998375b78f31aa45505383881c2151ddae73981e26bbe8bddb91e9390c7926a846e92f66096ae22689b390f508af6e445a4ef6d8925931
6a8d75da84090a1fde116ac6b75c50269528384bf879420defd971d0c0403c7377c56e2950acb6a7342a101d9bd3da406c38dd91a6153c3bcb502281af9c3da
ba1bf4690e10c5ed876d57ce2695f330

Second property:        a + ... + a = a*n
9833f7bd03718f8247f01c71c71d471c71c71c71c71d471c71c71c71c71c80

9833f7bd03718f8247f01c71c71d471c71c71c71c71d471c71c71c71c71c80
```

Результат додавання, віднімання, множення, ділення, піднесення до степеню, взяття лишку:

```
sum of ONE and TWO is: 6b5a4f3d7c6c1d69595c4b39c1abebc6937c8a7a76a06f40cebda997a4917e6b584513e2
difference of ONE and TWO is: 6b5a4f3d7c6a97c597786b5eb8b05255a6d4265878e6fa0fe2154575a6d8093a6b9cafc2
production of ONE and TWO is: 51b2731bc3733e30e420d9ca00e2bf1c9658cb1bafbdf1e6310c1080349dec8537f7f1ea114b249e7174cf47ae7ba1071
4cebb8b9a98b7933feab673412010112120
production of ONE const number is: 7226f9cdc2952ba1b5f415555555f555555555555555f555555555555555560
square of THREE is: e9cc4fb2159c54b402db5fb1f5f55449156f30fe4db9dac38013cd2b78475ae08ee0152d8063fa9f1d84b5970c48bc1f0f84322d23e
e14d34436b7b5fea8b933b80fc51239b45b42528430d454c8672300e27ff0debd6bddfc886632abc4846e8700ddc5a766e10be91268d0aeb3ab51148d94fd7e
cf73158aef5c14acceedec19b79d14ccff260d06ce1a8a3bd6151b7b0127a41869ab9db563ed6f67dd47abaae2869b1bba5c66d26e9c3262f3f9e902a31abcb
ba8c68a337605c4ed55c85cf7d5653f08f19a10bac6b3a5f42be234f91643e26801967c2b399d228557b4bb9401
queotient of ONE and TWO is: 8d10b45242b3
remainder of ONE and TWO is: ad56289131f1f50d3f62e2a7c379e7255f0696b9b54cc63f18b2709c0a2
power of numbers A and B is: 312e9e7bb37a3dd4ca23bd5240b415378d1287adc824763d23a32050ce207efc0e183bfb543a529941c8d3ecb7a1417cc5
e3631c18db3cd40ea09b6338eeb896afb43262660b01c0edadb15820c40c19a91214402871ecf0d621eb0ab39ea774ccd58c3ae58f499ab88c58060f463593c
9b1f879ed3775bdef33c66c1bd340b30eb7dd2005e49ad8004ebc98a17369063850506cf52c3dfc438575e2bcd1435cfb6c0edc761c01000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
```

Усі операції працюють коректно, перевірено за допомогою https://srom-check.herokuapp.com

Також було виміряно середній час виконання кожної зазначеної операції та відповідна кількість тактів процесора, результати були згруповані у таблицю:

Операція	Середній час виконання (секунди)	Кількість тактів процесора
Додавання	1.85e-06	2405

Віднімання	1.98e-06	2574
Множення	0.00028705	373165
Множення на число	3.13e-06	4069
Ділення	6.688e-05	86944
Взяття лишка	4.215e-05	54795
Піднесення в квадрат	0.00030344	394472
Піднесення до степеня великого числа	0.0105533	13719290

Кількість тактів процесора обрахована за формулою:
кількість тактів = час виконання (секунди) * тактова частота (Гц)
Тактова частота комп'ютера: 1,30ГГц, тобто 1300000000Гц

Додатки:

Увесь код можна знайти за посиланням, на GitHub:
https://github.com/MansteinOrGuderian/SpRzOM-1