СПЕЦІАЛЬНІ РОЗДІЛИ ОБЧИСЛЮВАЛЬНОЇ МАТЕМАТИКИ

Лабораторна 1

Терпило Софія

ФБ-06

Завдання до комп’ютерного практикуму:

А) Згідно варіанту розробити клас чи бібліотеку функцій для роботи з m-бітними цілимичислами. 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.

Хід роботи:

```python
def long_add(num1, num2, hex_alphabet, base):
    summary = ''
    num1 = num1[::-1]
    num2 = num2[::-1]
    num1, num2 = normalize(num1, num2)
    carry = 0
    if num1 == 0:
        return num2
    if num2 == 0:
        return num1
    for digit1, digit2 in zip(num1, num2):
        temp = hex_alphabet.index(digit1) + hex_alphabet.index(digit2) + carry
        sum_digit = temp % base
        carry = temp // base
        summary += hex_alphabet[sum_digit]
    if carry != 0:
        summary += hex_alphabet[carry]
    return summary[::-1]
```

```python
def long_sub(num1, num2, hex_alphabet, base):
    comparison = long_compare(num1, num2, hex_alphabet)
    if comparison == 1:
        difference = ''
        num1 = num1[::-1]
        num2 = num2[::-1]
        num1, num2 = normalize(num1, num2)
        borrow = 0
        for digit1, digit2 in zip(num1, num2):
            temp = hex_alphabet.index(digit1) - hex_alphabet.index(digit2) - borrow
            if temp >= 0:
                difference += hex_alphabet[temp]
                borrow = 0
            else:
                difference += hex_alphabet[temp]
                borrow = 1
        return difference[::-1]
    elif comparison == 0:
        return '0'
    else:
        return 'error'
```

```python
def long_mul_one_digit(num, digit, hex_alphabet, base):
    product = ''
    carry = 0
    for d in range(len(num)):
        temp = int(hex_alphabet.index(num[d])) * int(digit) + carry
        p = temp % base
        carry = temp // base
        product += hex_alphabet[p]
    if carry != 0:
        product += hex_alphabet[carry]
    return product


def long_mul(num1, num2, hex_alphabet, base):
    num1 = num1[::-1]
    num2 = num2[::-1]
    carry = '0'
    carry, num1 = normalize(carry, num1)
    for digit2 in range(len(num2)):
        temp_mul = long_mul_one_digit(num1, hex_alphabet.index(num2[digit2]), hex_alphabet, base)
        temp_mul = temp_mul[::-1]
        if digit2 != 0:
            n = 0
            while n != digit2:
                temp_mul += '0'
                n += 1
        carry = long_add(carry, temp_mul, hex_alphabet, base)
    return carry
```

```python
def long_pow(num, p, hex_alphabet, base):
    p = convert_to_binary(p)
    product = '1'
    for digit in range(len(p)):
        if p[digit] == '1':
            product = long_mul(product, num, hex_alphabet, base)
        if digit != len(p) - 1:
            product = long_mul(product, product, hex_alphabet, base)
    return product


def long_compare(num1, num2, hex_alphabet):
    num1 = num1.lstrip('0')
    num2 = num2.lstrip('0')
    num1 = num1[::-1]
    num2 = num2[::-1]

    if len(num1) > len(num2):
        return 1
    elif len(num1) < len(num2):
        return -1
    for i in range(len(num1) - 1, -1, -1):
        if hex_alphabet.index(num1[i]) < hex_alphabet.index(num2[i]):
            return -1
        elif hex_alphabet.index(num1[i]) > hex_alphabet.index(num2[i]):
            return 1
    return 0
```

```python
def long_div(num1, num2, hex_alphabet, base):
    quotient = ''
    remainder = ''
    dividend = ''
    if long_compare(num1, num2, hex_alphabet) == -1:
        remainder = num1
    else:
        if len(num2) >= 2:
            remainder = num1[: len(num2) - 1]
        for i in range(0, len(num1) - len(num2) + 1):
            temp_remainder = remainder
            dividend = temp_remainder + num1[i + len(num2) - 1]
            q_next = 0
            for j in range(base - 1, -1, -1):
                if j != 0:
                    mul_for_compare = long_mul(num2, hex_alphabet[j], hex_alphabet, base)
                    sub_for_compare = long_sub(dividend, mul_for_compare, hex_alphabet, base)
                    if sub_for_compare != 'error':
                        comparison = long_compare(sub_for_compare, dividend, hex_alphabet)
                        if comparison == -1 or comparison == 0:
                            q_next = j
                            break
            temp = long_mul(num2, hex_alphabet[q_next], hex_alphabet, base)
            remainder = long_sub(dividend, temp, hex_alphabet, base)
            quotient = quotient + str(hex_alphabet[q_next])
    if remainder == '0':
        return [quotient.lstrip('0'), '0']
    else:
        return [quotient.lstrip('0'), remainder.lstrip('0')]
```

Контроль коректності:

```python
# TEST: (a + b) * c = c * a + c * b
sum = long_add(a, b, hex_alphabet, base)
prod = long_mul(sum, c, hex_alphabet, base)
print('(a + b) * c =', prod)

prod_2 = long_mul(a, c, hex_alphabet, base)
prod_3 = long_mul(c, b, hex_alphabet, base)
sum_2 = long_add(prod_2, prod_3, hex_alphabet, base)
print('a * c + c * b =', sum_2)

if prod == sum_2:
    print('TEST : (a + b) * c = a * c + c * b => passed')
```

```
(a + b) * c = 36EBFA7ABB3981B29BEB9312ED9BFC371CCBFCBBC42E10FC659DD1A59A25A4F
a * c + c * b = 36EBFA7ABB3981B29BEB9312ED9BFC371CCBFCBBC42E10FC659DD1A59A25A
TEST : (a + b) * c = a * c + c * b => passed
```

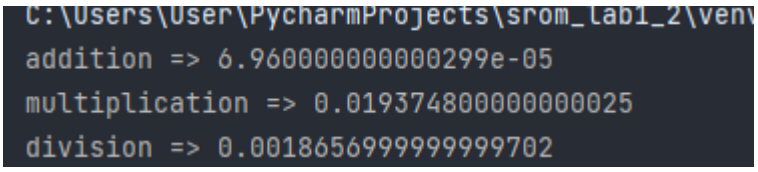
Час виконання операцій:

```python
addition_start = timer()
long_add(a, b, hex_alphabet, base)
addition_end = timer()
addition_time = addition_end - addition_start
print('addition =>', addition_time)

multiplication_start = timer()
long_mul(a, b, hex_alphabet, base)
multiplication_end = timer()
multiplication_time = multiplication_end - multiplication_start
print('multiplication =>', multiplication_time)

division_start = timer()
long_div(a, b, hex_alphabet, base)
division_end = timer()
division_time = division_end - division_start
print('division =>', division_time)

power_start = timer()
pow = long_pow(c, n, hex_alphabet, base)
power_end = timer()
power_time = power_end - power_start
print('power =>', power_time)
```



Час виконання піднесення в багаторозрядний степінь непристойно довго виконується, я її не дочекалась(