РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ
Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

Отчёт по лабораторной работе №8
Целочисленная арифметика многократной точности

Дисциплина: *Математические основы защиты информации и информационной безопасности*

Студент: Агеева Анастасия Сергеевна, 1032212304
Группа: НФИмд-02-21
Преподаватель: Кулябов Дмитрий Сергеевич,
д-р.ф.-м.н., проф.

Москва 2021

Содержание

List of Figures

1 Цель работы

Цель данной лабораторной работы изучение алгоритмов целочисленной арифметики многократной точности.

2 Задание

1. Реализовать программно алгоритм сложения неотрицательных чисел;
2. Реализовать программно алгоритм вычитания неотрицательных чисел;
3. Реализовать программно алгоритм умножения неотрицательных целых чисел столбиком;
4. Реализовать программно алгоритм умножения “быстрый столбик”;
5. Реализовать программно алгоритм деления многоразрядных целых чисел.

3 Теоретическое введение

Длинная арифметика [1] — выполняемые с помощью вычислительной машины арифметические операции (сложение, вычитание, умножение, деление, возведение в степень, элементарные функции) над числами, разрядность которых превышает длину машинного слова данной вычислительной машины. Эти операции реализуются не аппаратно, а программно, с использованием базовых аппаратных средств работы с числами меньших порядков. Частный случай — арифметика произвольной точности — относится к арифметике, в которой длина чисел ограничена только объёмом доступной памяти.

3.1 Применение

Длинная арифметика применяется в следующих областях:

- составление кода для процессоров (микроконтроллеров) низкой разрядности. Например, микроконтроллеры серии AVR имеют АЦП с разрядностью 10 бит и регистры с разрядностью 8 бит. Этого недостаточно для обработки информации с АЦП; без длинной арифметики не обойтись;
- **криптография**. Большинство систем подписывания и шифрования данных используют целочисленную арифметику по модулю m, где m — очень большое натуральное число, не обязательно простое. Например, при реализации метода шифрования RSA, криптосистемы Рабина или схемы Эль-Гамаля требуется обеспечить точность результатов умножения и возведения в степень порядка 10309;

- математическое (см. список ПО) и финансовое ПО. Результат вычисления на бумаге должен совпадать с результатом работы компьютера с точностью до последнего разряда. В частности, калькулятор Windows (начиная с Windows 95) проводит четыре арифметических действия с намного большей точностью, чем позволяет процессор x86. Для научных и инженерных расчётов длинная арифметика применяется редко, так как ошибки во входных данных обычно намного больше, чем ошибки округления;
- стандартная тема в спортивном программировании.

3.2 Необходимые аппаратные средства для работы с длинной арифметикой

Строго говоря, для реализации арифметики произвольной точности от процессора требуется лишь косвенная адресация; в арифметике фиксированной точности можно обойтись даже без неё. Тем не менее, определённые функции процессора ускоряют длинную арифметику, одновременно упрощая её программирование.

- Флаг переноса. Операции «сложить/вычесть с переносом», «циклический сдвиг через бит переноса».
- Косвенная адресация с автоинкрементом и автодекрементом (индексный регистр после операции увеличивается или уменьшается).
- Умножение $ww = 2w$ (умножение слова на слово, результат — двойное слово), деление $2w \div w = w$ (с возможным переполнением).

3.3 Реализация в языках программирования

Языки программирования имеют встроенные типы данных, размер которых, в основном, не превышает 64 бита (около 1019). Десятичная длинная арифмети-

ка была реализована в советских языках программирования АЛМИР-65 на ЭВМ МИР-1 и АНАЛИТИК на ЭВМ МИР-2. Для работы с большими числами, в современных языках программирования существует довольно много готовых оптимизированных библиотек для длинной арифметики.

Большинство функциональных языков позволяют переключаться с обычной арифметики на длинную без необходимости изменения кода арифметических расчётов. Например, Erlang и Scheme всегда представляют точные числа длинными. В Standard ML реализации всех разновидностей целых чисел определяются на основании сигнатуры INTEGER, позволяя выбирать необходимую размерность,— в том числе присутствует модуль IntInf, реализующий целые числа произвольной точности; в реализации PolyML этот модуль используется по умолчанию.

Встроенные библиотеки работы с большими числами есть в Ruby, Python и Java.

4 Выполнение лабораторной работы

1. Импортирую библиотеку math, которая понадобится для выполнения работы.

In [1]:
```
import math
```

2. Задам два словаря, которые создают связь между числом и строковым символом (и обратно).

In [2]:
```
# словарь
# строковый символ : числовой аналог
str2num = {chr(letter_sym) : (letter_sym - ord("A") + 10) for letter_sym in range(ord("A"), ord("2") + 1)}
for i in "0123456789":
    str2num[i] = int(i)

# число : строковый аналог
num2str = {value : key for (key, value) in str2num.items()}
```

3. Задам функцию для перевода числа в десятичную систему счисления.

In [3]:
```
# перевод в 10-ную систему счисления
# array = True, если число передается в виде массива чисел

def to_10 (u_str, b, array = False):
    u_array = u_str if array else [str2num[letter] for letter in u_str]
    u = 0
    for i in range(len(u_array)):
        u += (b**i)*u_array[len(u_array) - i - 1]
    return u
```

4. Задам функцию для перевода числа в систему счисления с основанием b.

In [4]:
```
# перевод в систему счисления с основанием b
# n - минимальная разрядность возвращаемого числа
def to_b(number, b, n = 1):
    (q, r) = (math.floor(number / b), number % b)
    w = num2str[r]
    while q >= b:
        (q, r) = (math.floor(q / b), q % b)
        w = w + num2str[r]
    if q != 0:
        w = w + num2str[q]
    while len(w) < n:
        w = w + "0"
    return w[::-1]
```

5. Задам две функции, которые будут удалять или добавлять незначащие нули к числу.

In [5]:
```
# удаление 0 в начале числа
def trim_zero(a):
    while a[0] == '0' and len(a) > 1:
        a = a[:1]
    return a
```

In [6]:
```
# добавления 0 в начало числа
def fill_zero(u, n, array = False):
    result = [0]*(n - len(u))
    if array:
        result.extend(u)
        return result
    return "".join([str(i) for i in result]) + u
```

6. Задам функцию *addition()*, в которой реализовано сложение неотри-

цательных чисел по алгоритму, представленному в задании к работе.

```
In [7]:
def addition(u_str, v_str, b):
    u = [str2num[letter] for letter in u_str]
    v = [str2num[letter] for letter in v_str]
    if len(u) != len(v):
        if len(u) < len(v):
            u = fill_zero(u, len(v), True)
        else:
            v = fill_zero(v, len(u), True)
    n = len(u)
    k = 0
    w = []
    for j in range(n-1, -1, -1):
        w.append(((u[j] + v[j] + k)%b))
        k = math.floor((u[j] + v[j] + k)/b)
    w.append(k)
    w.reverse()
    return "".join([num2str[i] for i in w])
```

```
In [8]:
addition("12345", "6789", 10)
```

```
Out[8]: '019134'
```

```
In [9]:
addition("010101", "001001", 2)
```

```
Out[9]: '0011110'
```

7. Задам функцию *substraction()*, в которой реализовано вычитание неотрицательных чисел по алгоритму, представленному в задании к работе.

```
In [10]:
def substraction(u_str, v_str, b):
    u = [str2num[letter] for letter in u_str]
    v = [str2num[letter] for letter in v_str]
    if len(u) != len(v):
        if len(u) < len(v):
            u = fill_zero(u, len(v), True)
        else:
            v = fill_zero(v, len(u), True)
    if u < v:
        return "u must be more than v"
    n = len(u)
    k = 0
    w = []
    for j in range(n-1, -1, -1):
        w.append(((u[j] - v[j] + k)%b))
        k = math.floor((u[j] - v[j] + k)/b)
    w.reverse()
    return "".join([num2str[i] for i in w])
```

```
In [11]:
substraction("12345", "6789", 10)
```

```
Out[11]: '05556'
```

```
In [12]:
substraction("010101", "001001", 2)
```

```
Out[12]: '001100'
```

```
In [13]:
substraction("345", "6789", 10)
```

```
Out[13]: 'u must be more than v'
```

8. Задам функцию *column_multiply()*, в которой реализовано умножение неотрицательных целых чисел столбиком по алгоритму, представленному в за-

```
In [14]:
def column_multiply(u_str, v_str, b):
    u = [str2num[letter] for letter in u_str]
    v = [str2num[letter] for letter in v_str]
    n = len(u)
    m = len(v)
    w = [0]*(m + n)
    for j in range(m -1, -1, -1):
        if v[j] != 0:
            k = 0
            for i in range(n - 1, -1, -1):
                t = u[i]*v[j] + w[i+j+1] + k
                w[j+i+1] = t%b
                k = math.floor(t/b)
            w[j] = k
    return "".join([num2str[i] for i in w])

In [15]:
column_multiply("12345", "6789", 10)
Out[15]: '083810205'

In [16]:
column_multiply("010101", "001001", 2)
Out[16]: '000010111101'
```

дании к работе.

9. Задам функцию *quick_multiply()*, в которой реализовано умножение неотрицательных чисел быстрым столбиком по алгоритму, представленному в за-

```
In [17]:
def quick_multiply(u_str, v_str, b):
    u = [str2num[letter] for letter in u_str]
    v = [str2num[letter] for letter in v_str]
    n = len(u)
    m = len(v)
    w = [0]*(m + n)
    t = 0
    for s in range(0, m + n):
        for i in range(0, s + 1):
            if (0 <= n -i - 1 < n) and (0 <= m - s + i - 1 < m):
                t = t + u[n - i - 1] * v[m - s + i - 1]
        w[m + n - s - 1] = t%b
        t =  math.floor(t/b)
    return "".join([num2str[i] for i in w])

In [18]:
quick_multiply("12345", "6789", 10)
Out[18]: '083810205'

In [19]:
quick_multiply("010101", "001001", 2)
Out[19]: '000010111101'
```

дании к работе.

10. Задам функцию *division()*, в которой реализовано деление многоразрядных целых чисел по алгоритму, представленному в задании к работе.

In [20]:
```
def division(u_str, v_str, b):
    u = u_str
    v = v_str
    u_10 = to_10(u, b)
    v_10 = to_10(v, b)
    n = len(u) - 1
    t = len(v) - 1
    if v[0] == 0 or not (n >= t >= 1):
        return "Incorrect data"
    q = [0]*(n - t + 1)
    while u_10 >= v_10 * (b ** (n-t)):
        q[n-t] = q[n-t] + 1
        u_10 -= v_10 * b ** (n-t)
    u = to_b(u_10, b, n+1)
    u = [str2num[letter] for letter in u]
    v = [str2num[letter] for letter in v_str]
    for i in range(n, t, -1):
        if u[n - i] >= v[0]:
            q[i-t-1] = b - 1
        else:
            q[i-t-1] = math.floor((u[n -i] * b + u[n-i+1]) / v[0])
        while q[i-t-1] * (v[0]*b + v[1]) > u[n - i]*(b**2) + u[n-i+1]*b + u[n-i+2]:
            q[i-t-1] = q[i-t-1] - 1
        u_10 = to_10(u, b, True)
        u_10 -= v_10 * q[i-t-1] * (b**(i-t-1))
        if u_10 < 0:
            u_10 += v_10 * (b**(i-t-1))
            q[i-t-1] -= 1
        u = to_b(u_10, b, n+1)
        u = [str2num[letter] for letter in u]
    q = "".join([num2str[i] for i in q])
    r = "".join([num2str[i] for i in u])
    return trim_zero(q[::-1]), trim_zero(r)
```

In [21]:
```
division("225", "15", 10)
```

Out[21]: ('15', '0')

In [22]:
```
division("225", "0", 10)
```

Out[22]: 'Incorrect data'

In [23]:
```
division("010101", "010", 2)
```

Out[23]: ('1010', '0')

5 Выводы

В ходе данной лабораторной работы я реализовала программно 5 алгоритмов целочисленной арифметики многократной точности.

Список литературы

1. Длинная арифметика [Электронный ресурс]. Википедия, 2019. URL: https://ru.wikipedia.org/wiki/Длинная_арифметика.