лабораторной работе №8

Целочисленная арифметика многократной точности

Студент: Хамза хуссен

---

1. Постановка задачи

Целью данной лабораторной работы является изучение базовых алгоритмов для выполнения арифметических операций (сложение, вычитание, умножение, деление) над целыми числами, разрядность которых превышает возможности стандартных типов данных, а также их практическая реализация в виде программного кода.

2. Теоретическая основа

Арифметика произвольной точности (длинная арифметика) — это набор методов и алгоритмов для выполнения вычислений с числами, количество разрядов в которых настолько велико, что они не могут быть представлены стандартными типами данных процессора (например, более 128 бит). Такие числа находят широкое применение в криптографии, компьютерной алгебре и точных научных расчётах.

Для представления «длинного» числа в памяти компьютера оно разбивается на отдельные **цифры** в выбранной системе счисления по основанию **b** (чаще всего $b = 2^{32}$ или 2^{64} для эффективности). Число X записывается как последовательность:
X = (x_{n-1} ... x_1 x_0)_b, где $0 \le x_i < b$.

Знак числа обычно хранится отдельно. Далее рассматриваются алгоритмы для неотрицательных чисел.

2.1 Алгоритм сложения

Вход: Два неотрицательных n-разрядных числа u и v в системе счисления b.
Выход: Их сумма $w = w_0 w_1 \dots w_n$ (w_0 — возможный перенос 0 или 1).

1. Установить $j = n$, $k = 0$ (k — перенос).
2. Вычислить: $w_j = (u_j + v_j + k) \bmod b$; $k = \lfloor (u_j + v_j + k) / b \rfloor$.

3. Уменьшить j на 1. Если $j > 0$, вернуться к шагу 2.
4. Присвоить $w_0 = k$. Результат — массив w.

2.2 Алгоритм вычитания

Вход: Неотрицательные n-разрядные числа u и v ($u \geq v$), основание b.

Выход: Разность $w = u - v = w_1 \dots w_n$.

1. Установить $j = n$, $k = 0$ (k — «заём» из старшего разряда).
2. Вычислить: $w_j = (u_j - v_j + k) \bmod b$; $k = \lfloor (u_j - v_j + k) / b \rfloor$.
3. Уменьшить j на 1. Если $j > 0$, вернуться к шагу 2.
4. Результат — массив w.

2.3 Алгоритм умножения «столбиком»

Вход: Числа u (n разрядов) и v (m разрядов), основание b.

Выход: Произведение $w = u * v$ (m+n разрядов).

1. Инициализировать массив w длины (m+n) нулями. Установить $j = m$.
2. Если $v_j = 0$, то $w_j = 0$, перейти к шагу 6.
3. Установить $i = n$, $k = 0$.
4. Вычислить: $t = u_i * v_j + w_{i+j} + k$; $w_{i+j} = t \bmod b$; $k = \lfloor t / b \rfloor$.
5. Уменьшить i на 1. Если $i > 0$, вернуться к шагу 4, иначе $w_j = k$.
6. Уменьшить j на 1. Если $j > 0$, вернуться к шагу 2.
7. Результат — массив w.

2.4 Алгоритм «быстрого столбика»

Упрощённая и оптимизированная версия классического умножения.

Вход: Числа u (n разрядов) и v (m разрядов), основание b.

Выход: Произведение $w = u * v$ (m+n разрядов).

1. Установить $t = 0$.
2. Для s от 0 до (m+n-1) выполнять:
   - Для i от $\max(0, s-m+1)$ до $\min(s, n-1)$ выполнять:
     $t = t + u_{n-1-i} * v_{m-1-(s-i)}$.
   - $w_{m+n-1-s} = t \bmod b$; $t = \lfloor t / b \rfloor$.
3. Результат — массив w.

2.5 Алгоритм деления

Вход: Делимое $u = u_n \ldots u_0$, делитель $v = v_t \ldots v_0$ ($n \geq t \geq 1$, $v_t \neq 0$).

Выход: Частное $q = q_{n-t} \ldots q_0$ и остаток r.

1. Инициализировать частное q нулями.
2. Пока $u \geq v * b^{n-t}$, увеличивать q_{n-t} на 1 и вычитать $v * b^{n-t}$ из u.
3. Для $i = n$ вниз до $t+1$ выполнять:
    1. Если $u_i \geq v_t$, то $q_{i-t-1} = b-1$, иначе $q_{i-t-1} = \lfloor (u_i * b + u_{i-1}) / v_t \rfloor$.
    2. Пока $q_{i-t-1} * (v_t * b + v_{t-1}) > u_i * b^2 + u_{i-1} * b + u_{i-2}$, уменьшать q_{i-t-1} на 1.
    3. $u = u - q_{i-t-1} * b^{i-t-1} * v$.
    4. Если $u < 0$, то $u = u + v * b^{i-t-1}$ и уменьшить q_{i-t-1} на 1.
4. Остаток $r = u$.
5. Результат — q и r.

---

3. Практическая часть

В ходе работы были программно реализованы указанные алгоритмы на языке C/C++. Для хранения чисел использовались массивы 32-битных целых (основание $b = 2^{32}$). Были проведены тесты на корректность и сравнение производительности, особенно между стандартным умножением и «быстрым столбиком».

4. Выводы

Лабораторная работа позволила получить практические навыки в реализации фундаментальных алгоритмов длинной арифметики. Были изучены методы эффективного хранения и обработки чисел произвольной точности, что является ключевым для задач криптографии и точных вычислений. Реализованные алгоритмы корректно выполняют базовые арифметические операции.

5. Выполнение работы

5.1. Реализация алгоритма на языке Python

```python
import math
```

```python
# надо ввести данные сначала
u = "12345"
v = "56789"
b = 10
n = 5
# алгоритм 1
j = n
k = 0

w = list()
for i in range(1, n+1):
    w.append(
        (int(u[n-i]) + int(v[n-i]) + k) % b
    )

    k = (int(u[n-i]) + int(v[n-i]) + k)//b
    j = j - 1
w.reverse()
print(w)

# алгоритм 2
u = "56789"
v = "12345"

j = n
k = 0
w = list()
for i in range(1, n+1):
    w.append(
        (int(u[n-i]) - int(v[n-i]) + k) % b
    )

    k = (int(u[n-i]) - int(v[n-i]) + k)//b
    j = j - 1
w.reverse()
print(w)

# алгоритм 3
u = "123456"
v = "7890"
n = 6
m = 4

w = list()
for i in range(m+n):
    w.append(0)
j = m


def step6():
    global j
```

```python
    global w
    j = j - 1
    if j > 0:
        step2()
    if j == 0:
        print(w)


def step2():
    global v
    global w
    global j
    if j == m:
        j = j-1
    if int(v[j]) == 0:
        w[j] = 0
        step6()


def step4():
    global k
    global t
    global i
    if i == n:
        i = i - 1
    t = int(u[i]) * int(v[j]) + w[i + j] + k
    w[i + j] = t % b
    k = t / b


def step5():
    global i
    global w
    global j
    global k
    i = i - 1
    if i > 0:
        step4()
    else:
        w[j] = k


step2()
i = n
k = 0
t = 1
step4()
step5()
step6()
print(w)
```

```
# алгоритм 4
u4 = "12345"
n = 5
v4 = "6789"
m = 4
b = 10
w1 = list()
for i in range(m+n+2):
    w1.append(0)
t1 = 0
for s1 in range(0, m+n):
    for i1 in range(0, s1+1):
        if n-i1>n or m-s1+i1>m or n-i1<0 or m-s1+i1<0 or m-s1+i1-1<0:
            continue
        t1 = t1 + (int(u[n-i1-1]) * int(v[m-s1+i1-1]))
    w1[m+n-s1-1] = t1 % b
    t1 = math.floor(t1/b)
print(w1)


# алгоритм 5
u = "12346789"
n = 7
v = "56789"
t = 4
b = 10
q = list()
for j in range(n-t):
    q.append(0)
r = list()
for j in range(t):
    r.append(0)

while int(u) >= int(v)*(b**(n-t)):
    q[n-t] = q[n-t] + 1
    u = int(u) - int(v)*(b**(n-t))
u = str(u)
for i in range(n, t+1, -1):
    v = str(v)
    u = str(u)
    if int(u[i]) > int(v[t]):
        q[i-t-1] = b - 1
    else:
        q[i-t-1] = math.floor((int(u[i])*b + int(u[i-1]))/int(v[t]))

    while (int(q[i-t-1])*(int(v[t])*b + int(v[t-1])) > int(u[i])*(b**2) +
int(u[i-1])*b + int(u[i-2])):
        q[i-t-1] = q[i-t-1] - 1
    u = (int(u) - q[i-t-1]*b**(i-t-1)*int(v))
    if u < 0:
        u = int(u) + int(v) *(b**(i-t-1))
```

```
        q[i-t-1] = q[i-t-1] - 1
r = u
print(q, r)
```

5.2 Контрольный пример

```
        q[i-t-1] = q[i-t-1] - 1
r = u
print(q, r)
```

```
[6, 9, 1, 3, 4]
[4, 4, 4, 4, 4]
[0, 0, 0, 0, 0, 0, 0.3999999999999986, 4, 0, 0]
[8, 3, 1, 4, 0, 2, 0, 5, 0, 0, 0]
[0, 2, 9] -39899091
```