Лабораторная работа №8

Математические основы защиты информации и информационной безопасности

Колчева Юлия Вячеславовна

Содержание

List of Tables

List of Figures

1 Цель работы

Познакомиться с целочисленной арифметикой многократной точности.

2 Задание

Реализовать алгоритмы, производящие операции над числами.

3 Выполнение лабораторной работы

Для всех последующих алгоритмов были использованы числа u и v (рис. 3.1) :

```
n [25]:
1 from math import floor
2 n = 3
3 m = 3
4 u = 124
5 v = 148
6 b = 10
7
```

Figure 3.1: Начальные данные

Для реализации алгоритма сложения целых чисел была написана следующая программа (рис. 3.2)

In [26]:
```
1  def sum_ (n,u,v,b):
2      k = 0
3      w = []
4      for j in range(n,0,-1):
5          u_j = u % b
6          v_j = v % b
7          w.append((u_j+v_j+k) % b)
8          k = floor((u_j+v_j+k) / b)
9          u = u // b
10         v = v // b
11     w0 = k
12     if w0 == 1:
13         w.append(w0)
14     return w
15
16 w = sum_(n,u,v,b)
```

In [27]:
```
1 w.reverse()
2 print(*w, sep = '')
```

```
272
```

Figure 3.2: сложение целых чисел

В данной программе:

1-3 строки: задаём функцию и начальные данные

4-10: реализация алгоритма: отделяем от числа цифры, производим с ними вычисления при помощи формул из лабораторной и отсекаем цифру.

13: запись цифры ответа в список.

В данном случае я вычислила сумму 124 и 148. Вывод представлен на скриншоте.

Для реализации разности была написана следующая программа (рис. 3.3)

In [28]:
```
def sub_ (n,u,v,b):
    k = 0
    w = []
    for j in range(n,0,-1):
        u_j = u % b
        v_j = v % b
        w.append((u_j-v_j+k) % b)
        k = floor((u_j-v_j+k) / b)
        u = u // b
        v = v // b
    return w
w = sub_(n,u,v,b)
```

In [29]:
```
w.reverse()
print(*w, sep = '')
```

```
976
```

Figure 3.3: разность целых чисел

Программа реализована аналогично предыдущей, только со знаком минуса. Вывод представлен на сриншоте (рис. 3.3)

Для реализации теста умножения столбиком была написана следующая программа (рис. 3.4) (рис. 3.5)

In [30]:

```
def mul1(uu,vv,b):
    u = []
    v = []
    for i in str(uu):
        u.append(int(i))
    for i in str(vv):
        v.append(int(i))
    n = len(u) - 1
    m = len(v) - 1
    j = m
    w = [0] * (len(u) + len(v))
    while j >= 0:
        if v[j] == 0:
            w[j] == 0
            j = j - 1
        else:
            i = n
            k = 0
            while i >= 0:
                t = u[i] * v[j] + w[i + j + 1] + k
                w[i + j + 1] = t % b
                k = t // b
                i = i - 1
            w[j] = k
            j = j - 1
        z = 0
```

Figure 3.4: Программа умножения столбиком1

```
        z = 0
        while w[z] == 0:
            z = z + 1
    return print(''.join(str(i) for i in remove_zeros(w)))

mul1(u,v,b)
```

```
18352
```

Figure 3.5: Программа умножения столбиком2

В данной программе:

1-3 строка: задаём функцию и подготавливаем переменные.

4-29: реализация алгоритма: присваиваем нулевые значения, отделяем цифры от числа и вычисляем новое значение по нескольким формулам, затем отсекаем цифру от числа и начинаем алгоритм заново.

Результаты работы программы с числами 124 и 148 (рис. 3.5)

Для реализации теста умножения быстрым столбиком была написана следующая программа (рис. 3.6)

```
In [32]:
def mult2(uu,vv,b):
    u = [int(i) for i in str(uu)]
    v = [int(i) for i in str(vv)]
    n = len(u) - 1
    m = len(v) - 1
    w = [0] * (len(u) + len(v))

    t = 0
    for s in range(m + n + 2):
        for i in range(s + 1):
            if (n - i < 0) or (m - s + i < 0):
                t = t
            else:
                t = t + u[n - i] * v[m - s + i]
        w[m + n - s + 1] = t % b
        t = t // b
    return print(''.join(str(i) for i in remove_zeros(w)))

In [33]:
mult2(u,v,b)

18352
```

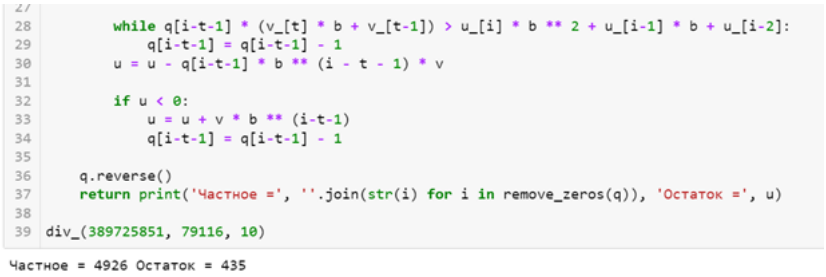
Figure 3.6: Быстрый столбик

Данная программа считает произведение более коротким образом. Вывод можно увидеть на скриншоте, он такой же, как и в предыдущем алгоритме, но считается быстрее. (рис. 3.6)

И в конце, алгоритм для деления многоразрядных целых чисел (рис. 3.7) (рис. 3.8)

```
In [34]:
def div_(uu,vv,b):
    u = uu
    v = vv

    n = len([int(i) for i in str(uu)]) - 1
    t = len([int(i) for i in str(vv)]) - 1
    q = [0] * (n - t + 1)
    r = [0] * (t + 1)


    while u >= v * b ** (n - t):
        q[n-t] = q[n-t] + 1
        u = u - v * b ** (n - t)

    n = len([int(i) for i in str(u)]) - 1
    t = len([int(i) for i in str(v)]) - 1

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

Figure 3.7: Деление



```
        while q[i-t-1] * (v_[t] * b + v_[t-1]) > u_[i] * b ** 2 + u_[i-1] * b + u_[i-2]:
            q[i-t-1] = q[i-t-1] - 1
        u = u - q[i-t-1] * b ** (i - t - 1) * v

        if u < 0:
            u = u + v * b ** (i-t-1)
            q[i-t-1] = q[i-t-1] - 1

    q.reverse()
    return print('Частное =', ''.join(str(i) for i in remove_zeros(q)), 'Остаток =', u)

div_(389725851, 79116, 10)
```

```
Частное = 4926 Остаток = 435
```

Figure 3.8: Деление

Данный алгоритм так же путём отделения цифр от чисел считает их частное и записывает остаток. С каждой цифрой работаем отдельно и записываем что мы взяли от других разрядов.

Выводы представлены на скриншоте (рис. 3.8)

4 Выводы

Познакомилась с целочисленной арифметикой многократной точности. Реализовала пять алгоритмов действий с многоразрядными числами.

5 Список литературы

Лабораторная работа №8

Целочисленная арифметикой многократной точности [Электронный ресурс]. URL: https://esystem.rudn.ru/mod/folder/view.php?id=1150982