Отчёт по лабораторной работе №8

Целочисленная арифметика многократной точности

Милёхин Александр НПМмд-02-21

Содержание

[1 Цель работы 1](#_Toc95081546)

[2 Теоретические сведения 1](#_Toc95081547)

[2.1 Сложение неотрицательных целых чисел 2](#_Toc95081548)

[2.2 Вычитание неотрицательных целых чисел 2](#_Toc95081549)

[2.3 Умножение неотрицательных целых чисел столбиком 2](#_Toc95081550)

[2.4 Быстрый столбик 3](#_Toc95081551)

[2.5 Деление многоразрядных целых чисел 3](#_Toc95081552)

[3 Выполнение работы 3](#_Toc95081553)

[3.1 Реализация алгоритмов на языке Python 3](#_Toc95081554)

[3.2 Контрольный пример 6](#_Toc95081555)

[4 Выводы 7](#_Toc95081556)

[Список литературы 7](#_Toc95081557)

# 1 Цель работы

Ознакомление с алгоритмами целочисленной арифметики многократной точности, а также их последующая программная реализация.

# 2 Теоретические сведения

Высокоточная (длинная) арифметика — это операции (базовые арифметические действия, элементарные математические функции и пр.) над числами большой разрядности (многоразрядными числами), т.е. числами, разрядность которых превышает длину машинного слова универсальных процессоров общего назначения (более 128 бит).

В современных асимметричных криптосистемах в качестве ключей, как правило, используются целые числа длиной 1000 и более битов. Для задания чисел такого размера не подходит ни один стандартный целочисленный тип данных современных языков программирования. Представление чисел в формате с плавающей точкой позволяет задать очень большие числа (например, тип long double языка C++ – до ), но не удовлетворяет требованию абсолютной точности, характерному для криптографических приложений. Поэтому большие целые числа представляются в криптографических пакетах в виде последовательности цифр в некоторой системе счисления (обозначим основание системы счисления ): $x = (x\_{n-1} x\_{n-2} x\_1 x\_0)\_b, $ где .

Основание системы счисления выбирается так, чтобы существовали машинные команды для работы с однозначными и двузначными числами; как правило, равно , или .

При работе с большими целыми числами знак такого числа удобно хранить в отдельной переменной. Например, при умножении двух чисел знак произведения вычисляется отдельно.

Далее при описании алгоритмов квадратные скобки означают, что берётся целая часть числа.

## 2.1 Сложение неотрицательных целых чисел

\*Вход. Два неотрицательных числа и ; разрядность чисел ; основание системы счисления .

\*Выход. Сумма , где - цифра переноса, всегда равная либо .

1. Присвоить ( *идет по разрядам, следит за переносом*).
2. Присвоить , где .
3. Присвоить . Если , то возвращаемся на шаг 2; если , то присвоить и результат: .

## 2.2 Вычитание неотрицательных целых чисел

\*Вход. Два неотрицательных числа и , ; разрядность чисел ; основание системы счисления .

\*Выход. Разность .

1. Присвоить ( – заём из старшего разряда).
2. Присвоить ; .
3. Присвоить . Если , то возвращаемся на шаг 2; если , то результат: .

## 2.3 Умножение неотрицательных целых чисел столбиком

\*Вход. Числа , ; основание системы счисления .

\*Выход. Произведение .

1. Выполнить присвоения: ( *перемещается по номерам разрядов числа от младших к старшим*).
2. Если , то присвоить и перейти на шаг 6.
3. Присвоить (*значение идет по номерам разрядов числа , отвечает за перенос*).
4. Присвоить .
5. Присвоить . Если , то возвращаемся на шаг 4, иначе присвоить .
6. Присвоить . Если , то вернуться на шаг 2. Если , то результат: .

## 2.4 Быстрый столбик

\*Вход. Числа , ; основание системы счисления .

\*Выход. Произведение .

1. Присвоить .
2. Для от до с шагом 1 выполнить шаги 3 и 4.
3. Для от до с шагом 1 выполнить присвоение .
4. Присвоить . Результат: .

## 2.5 Деление многоразрядных целых чисел

\*Вход. Числа , .

\*Выход. Частное , остаток .

1. Для от до присвоить .
2. Пока , выполнять: .
3. Для выполнять пункты 3.1 – 3.4: 3.1. если , то присвоить , иначе присвоить . 3.2. пока выполнять . 3.3. присвоить . 3.4. если , то присвоить , .
4. . Результат: и .

# 3 Выполнение работы

## 3.1 Реализация алгоритмов на языке Python

import math  
  
#Algorithm 1  
u = "25534"  
v = "34789"  
b = 10  
n = 5  
  
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)  
  
#Algorithm 2  
u = "45678"  
v = "23456"  
  
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)  
  
#Algorithm 3  
u = "123450"  
v = "7895"  
n = 6  
m = 4  
  
w = list()  
for i in range(m+n):  
 w.append(0)  
j = m  
  
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  
   
def step6():  
 global j  
 global w  
 j = j - 1  
 if j > 0:  
 step2()  
 if j == 0:  
 print(w)  
  
step2()  
i = n  
k = 0  
t = 1  
step4()  
step5()  
step6()  
print(w)  
  
#Algorithm 4  
u4 = "12345"  
n = 5  
v4 = "6780"  
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)  
  
#Algorithm 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)

## 3.2 Контрольный пример

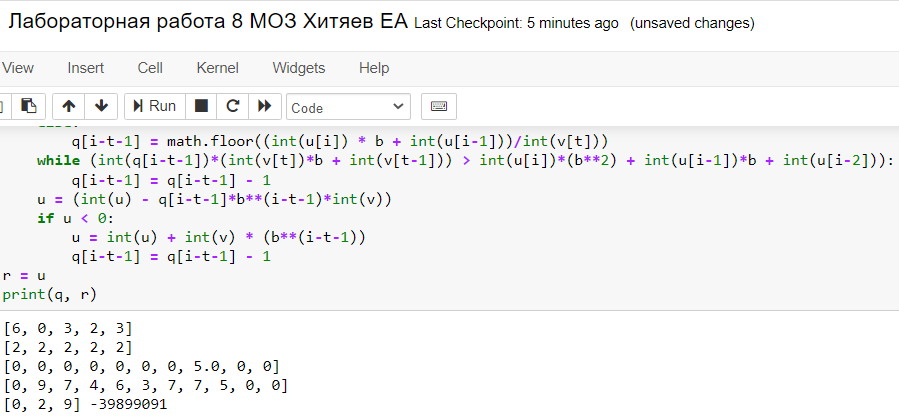


Figure 1: Пример работы алгоритмов

# 4 Выводы

Я изучил алгоритмы целочисленной арифметики, а также реализовал их программно на языке Python.

# Список литературы

1. [Длинная арифметика от Microsoft](https://habr.com/ru/post/207754/)
2. [Как оперировать числами, не помещающимися ни в один из числовых типов](https://programforyou.ru/poleznoe/dlinnaya-arifmetika-kak-operirovat-chislami-ne-pomeshchayushchimisya-ni-v-odin-iz-chislovyh-tipov)