Шаблон отчёта по лабораторной работе №8

Целочисленная арифметика многократной точности

Коне Сирики

Содержание

# 1 Цель работы

Целью данной лабораторной работы является ознакомление с алгоритмами целочисленной арифметики многократной точности, а также их последующая программная реализация.

# 2 Задание

Рассмотреть и реализовать на языке программирования Python:

1. Алгоритм сложения неотрицательных целых чисел;
2. Алгоритм вычитания неотрицательных целых чисел;
3. Алгоритм умножения неотрицательных целых чисел столбиком;
4. Алгоритм умножения неотрицательных целых чисел быстрым столбиком;
5. Алгоритм деления многоразрядных целых чисел.

# 3 Теоретическое введение

## 3.1 Арифметика многократной точности

Опр. 3.1.

:*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) [1].

В современных асимметричных криптосистемах в качестве ключей, как правило, используются целые числа длиной 1000 и более битов [2]. 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 ): 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 [3]. Например, при умножении двух чисел знак произведения вычисляется отдельно.

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

## 3.2 Сложение неотрицательных целых чисел

**Алгоритм 1. Сложение неотрицательных целых чисел** [3]

*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: .

## 3.3 Вычитание неотрицательных целых чисел

**Алгоритм 2. Вычитание неотрицательных целых чисел** [3]

*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: .

## 3.4 Умножение неотрицательных целых чисел столбиком

**Алгоритм 3. Умножение неотрицательных целых чисел столбиком** [3]

*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: .

## 3.5 Быстрый столбик

**Алгоритм 4. Быстрый столбик** [3]

*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: .

## 3.6 Деление многоразрядных целых чисел

**Алгоритм 5. Деление многоразрядных целых чисел** [3]

*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 , иначе присвоить .
* 3.2. пока выполнять .
* 3.3. присвоить .
* 3.4. если , то присвоить , .

1. Результат: и .

# 4 Выполнение лабораторной работы

Реализуем описанные выше алгоритмы на языке **Python** в среде Jupyter Notebook. Для работы нам понадобятся библиотека math, словари, отображающие буквы в их числовые аналоги и наоборот ( и т.д.), а также функция добавления ведущих нулей к числу :

import math  
  
# словарь вида {строковый символ : его числовой аналог}  
str2num = {chr(letter\_ord) : (letter\_ord - ord("A") + 10)  
 for letter\_ord in range(ord("A"), ord("Z") + 1)}  
for digit in "0123456789":  
 str2num[digit] = int(digit)  
# словарь вида {число : его стокровый аналог}  
num2str = {value : key for (key, value) in str2num.items()}  
  
def fill0(u, n, array = False):  
 """  
 Добавляет ведущие нули к числу u до разрядности n;  
 array = True, если число u передано в виде массива чисел  
 """  
 result = [0] \* (n - len(u))  
 if array:  
 result.extend(u)  
 return result  
 return "".join([str(i) for i in result]) + u

## 4.1 Сложение неотрицательных целых чисел

Создадим функцию addition(u\_str, v\_str, b) следующего вида:

def addition(u\_str, v\_str, b):  
 """  
 Складывает два неотрицательных числа в системе счисления b,  
 u\_str и v\_str, представленных в строчном виде  
 """  
 # представляем числа в виде массивов чисел  
 u = [str2num[letter] for letter in u\_str]  
 v = [str2num[letter] for letter in v\_str]  
  
 if len(u) != len(v): # если разрядности чисел не совпадают..  
 # добавляем к меньшему числу ведущие нули  
 if len(u) < len(v): u = fill0(u, len(v), True)  
 else: v = fill0(v, len(u), True)  
  
 # n - разрядность числа, k следит за переносом  
 # шаг 1  
 n = len(u); k = 0  
 w = [] # сумма  
  
 for j in range(n - 1, -1, -1): #  
 w.append(((u[j] + v[j] + k) % b)) # шаг 2-3  
 k = math.floor((u[j] + v[j] + k) / b) #  
  
 w.append(k); w.reverse() # шаг 3  
 # преобразуем сумму-массив в строчный вид  
 return "".join([num2str[digit] for digit in w])

Теперь с помощью данной функции найдём суммы пар чисел в разных системах счисления (см. Рис. 1).

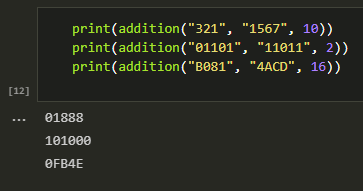


Рис. 1: Примеры нахождения сумм пар чисел в разных системах счисления

## 4.2 Вычитание неотрицательных целых чисел

Создадим функцию subtraction(u\_str, v\_str, b) следующего вида:

def subtraction(u\_str, v\_str, b):  
 """  
 Вычитает число v\_str из u\_str, представленных в строчном виде,  
 в системе счисления b  
 """  
 # представляем числа в виде массивов чисел  
 u = [str2num[letter] for letter in u\_str]  
 v = [str2num[letter] for letter in v\_str]  
  
 # если разрядности чисел не совпадают..  
 if len(u) != len(v):  
 # добавляем к меньшему числу ведущие нули  
 if len(u) < len(v): u = fill0(u, len(v), True)  
 else: v = fill0(v, len(u), True)  
 elif u < v: # если u меньше v  
 return "u должно быть больше v"  
  
 # n - разрядность числа, k следит за переносом,  
 # w - разность  
 n = len(u); w = []; k = 0 # шаг 1  
  
 for j in range(n - 1, -1, -1): #  
 w.append(((u[j] - v[j] + k) % b)) # шаг 2-3  
 k = math.floor((u[j] - v[j] + k) / b) #  
  
 w.reverse() # записываем массив в обратном порядке  
  
 # преобразуем результат в виде массива в строчный вид  
 return "".join([num2str[digit] for digit in w])

Теперь с помощью данной функции найдём разности пар чисел в разных системах счисления (см. Рис. 2).

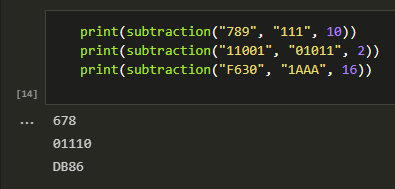


Рис. 2: Примеры нахождения разностей пар чисел в разных системах счисления

## 4.3 Умножение неотрицательных целых чисел столбиком

Создадим функцию multiply\_column(u\_str, v\_str, b) следующего вида:

def multiply\_column(u\_str, v\_str, b):  
 """  
 Умножает столбиком два неотрицательных числа  
 в системе счисления b, u\_str и v\_str,  
 представленных в строчном виде  
 """  
 # представляем числа в виде массивов чисел  
 u = [str2num[letter] for letter in u\_str]  
 v = [str2num[letter] for letter in v\_str]  
  
 # n - разрядность u,  
 # m - разрядность v  
 n = len(u); m = len(v)  
 # произведение  
 w = [0] \* (m + n) # шаг 1  
  
 # шаг 2 опускаем, поскольку весь массив w  
 # изначально заполнен нулями  
  
 for j in range(m - 1, -1, -1):   
 if v[j] != 0:   
 k = 0 # шаг 3   
 for i in range(n - 1, -1, -1): #  
 t = u[i] \* v[j] + w[i + j + 1] + k # шаг 4  
 w[i + j + 1] = t % b #  
 k = math.floor(t / b) #  
 w[j] = k # шаг 5   
  
 # преобразуем результат в виде массива в строчный вид  
 return "".join([num2str[digit] for digit in w]) # шаг 6

Теперь с помощью данной функции найдём произведения пар чисел в разных системах счисления (см. Рис. 3).

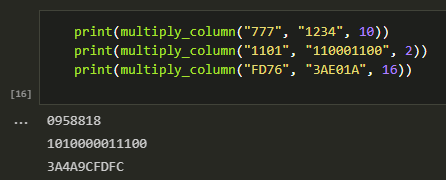


Рис. 3: Примеры нахождения произведения пар чисел в разных системах счисления

## 4.4 Быстрый столбик

Создадим функцию multiply\_quick(u\_str, v\_str, b) следующего вида:

def multiply\_quick(u\_str, v\_str, b):  
 """  
 Умножает быстрым столбиком два неотрицательных числа  
 в системе счисления b, u\_str и v\_str,  
 представленных в строчном виде  
 """  
 # представляем числа в виде массивов чисел  
 u = [str2num[letter] for letter in u\_str]  
 v = [str2num[letter] for letter in v\_str]  
  
 # n - разрядность u,  
 # m - разрядность v  
 n = len(u); m = len(v)  
 # произведение  
 w = [0] \* (m + n)  
  
 t = 0 # шаг 1  
 for s in range(0, m + n): # шаг 2  
 for i in range(0, s + 1): #  
 if (0 <= n - i - 1 < n) and (0 <= m - s + i - 1 < m): # шаг 3  
 t = t + u[n - i - 1] \* v[m - s + i - 1] #  
 w[m + n - s - 1] = t % b #  
 t = math.floor(t / b) # шаг 4  
  
 # преобразуем результат в виде массива в строчный вид  
 return "".join([num2str[digit] for digit in w])

Теперь с помощью данной функции найдём произведения пар чисел в разных системах счисления (см. Рис. 4).

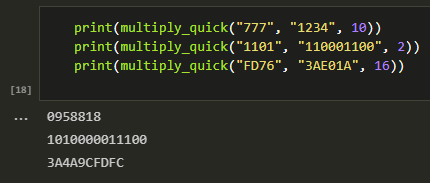


Рис. 4: Примеры нахождения произведения пар чисел быстрым столбиком в разных системах счисления

## 4.5 Деление многоразрядных целых чисел

Начнём с реализации дополнительных функций: для смены системы счисления числа и для удаления ведущих нулей из его представления.

def to10(u\_str, b, array = False):  
 """  
 Переводит число u\_str в системе счисления b  
 в десятичную систему исчисления;  
 array = True, если число u передано в виде массива чисел  
 """  
 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  
  
def to\_b(number, b, n = 1):  
 """  
 Переводит десятичное число number в систему счисления  
 с основанием b; n - минимальная разрядность  
 результирующей записи числа   
 """  
 # будем последовательно делить number на b и сохранять остатки  
 # q - очередное частое, r - очередной остаток  
 # w - результат, в который поелсдовательно записываем остатки  
 (q, r) = (math.floor(number / b), number % b); w = num2str[r]  
  
 # пока частоное больше основания системы счисления  
 while q >= b:  
 # продолжаем деление  
 (q, r) = (math.floor(q / b), q % b)  
 w = w + num2str[r]  
  
 # если частное ненулевое, тоже добавляем его в результат  
 if q != 0: w = w + num2str[q]  
  
 # если разрядность меньше желаемой..  
 while len(w) < n:  
 # добавляем ведущие нули  
 w = w + "0"  
  
 # записываем число в обратном порядке  
 return w[::-1]  
  
def trim\_zero(a):  
 """  
 Удаляет ведущие нули числа a  
 """  
 while a[0] == '0' and len(a) > 1:  
 a = a[1:]  
 return a

Теперь создадим функцию division(u\_str, v\_str, b) следующего вида:

def division(u\_str, v\_str, b):  
 """  
 Производит деление целых неотрицательных чисел,  
 записанных в строчном виде (u\_str на v\_str),  
 в системе счисления с основанием b  
 Результат: (q, r), где q - частное, r - остаток  
 """  
 u = u\_str; v = v\_str  
 n = len(u) - 1; t = len(v) - 1 # разрядности чисел  
  
 # проверка условий  
 if v[0] == 0 or not (n >= t >= 1):  
 return "Некорректные входные данные"  
  
 q = [0] \* (n - t + 1) # шаг 1  
  
 while to10(u, b) >= to10(v, b) \* (b \*\* (n - t)): #  
 q[n - t] = q[n - t] + 1 #  
 a = to\_b(b \*\* (n - t), b) # шаг 2  
 a = multiply\_column(v, a, b) #  
 u = subtraction(u, a, b) #  
 if len(u) > len(u\_str): # сохраняем начальную  
 u = u[1:] if u[0] == '0' else u # разрядность числа  
  
 # переводим числа в вид массивов  
 u = [str2num[letter] for letter in u]  
 v = [str2num[letter] for letter in v\_str]  
  
 for i in range(n, t, -1): # шаг 3  
 if u[n - i] >= v[0]: #  
 q[i - t - 1] = b - 1 # шаг 3.1.  
 else: #  
 q[i - t - 1] = math.floor((u[n - i] \* b + u[n - i + 1]) / v[0])  
  
 # шаг 3.2  
 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 = to10(u, b, True); v\_10 = to10(v, b, True) #  
 a = v\_10 \* q[i - t - 1] \* (b \*\* (i - t - 1)) # шаг 3.3  
 u\_10 = u\_10 - a #  
  
 if u\_10 < 0: #  
 u\_10 = u\_10 + v\_10 \* (b \*\* (i - t - 1)) # шаг 3.4  
 q[i - t - 1] = q[i - t - 1] - 1 #  
  
 # возвращаем число u в систему с основанием b  
 u = to\_b(u\_10, b, n + 1); u = [str2num[letter] for letter in u]  
  
 # преобразуем массивы обратно в строки  
 (q, r) = ("".join([num2str[digit] for digit in q]),  
 "".join([num2str[digit] for digit in u]))  
  
 # удаляем ведущие нули и записываем частное в обратном порядке  
 return (trim\_zero(q[::-1]), trim\_zero(r))

Теперь с помощью данной функции найдём частные и остоток от деления пар чисел в разных системах счисления (см. Рис. 5).

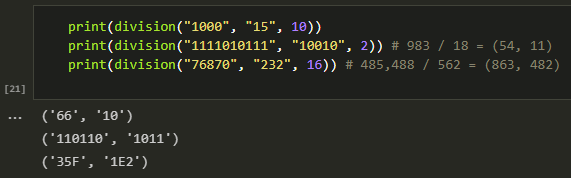


Рис. 5: Примеры нахождения частных и остатков от деления пар чисел в разных системах счисления

# 5 Выводы

Таким образом, была достигнута цель, поставленная в начале лабораторной работы: было проведено краткое знакомство с алгоритмами целочисленной арифметики многократной точности (сложение неотрицательных целых чисел, вычитание неотрицательных целых чисел, умножение неотрицательных целых чисел столбиком и быстрым столбиком, деление многоразрядных целых чисел), после чего все пять алгоритмов были успешно реализованы на языке программирования **Python**.

# Список литературы

1. Исупов К.С. Методы и алгоритмы организации высокоточных вычислений в арифметике остаточных классов для универсальных процессорных платформ: phdthesis. Вятский государственный университет, 2014.

2. Панкратова И.А. Теоретико-числовые методы в криптографии: учебное пособие. Томск: Томский государственный университет, 2009. С. 120.

3. Бубнов С.А. Лабораторный практикум по основам криптографии: учебно-методическое пособие. Саратов; <http://elibrary.sgu.ru/uch_lit/656.pdf>: Саратовский государственный университет им. Н.Г. Чернышевского, 2012.