

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Дальневосточный федеральный университет»**

**(ДВФУ)**

|  |
| --- |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ**  **Департамент математического и компьютерного моделирования** |

**ДОКЛАД**

**о практическом задании по дисциплине АИСД**

«Эффективная длинная арифметика»

направление подготовки 09.03.03 «Прикладная информатика»

профиль «Прикладная информатика в компьютерном дизайне»

|  |  |  |
| --- | --- | --- |
|  |  | Выполнил студент  гр. Б9121-09.03.03пикд  Сербинович М.Д.  \_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)* |
| Доклад защищен:  С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Руководитель практики  Доцент ИМКТ А.С Кленин  *(должность, уч. звание)*  \_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)*  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2022г. |
| Рег. № \_\_\_\_\_\_  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г. |  |  |

г. Владивосток

2022

# Введение

Решая задачи, многим из нас приходилось сталкиваться с тем, что нам просто не хватало размерностей типов для, казалось, простейших операций: сложение, вычитание и умножение. Все эти операции знакомы Вам с ранних классов. Но что делать, если одну из этих операций необходимо применить для огромных чисел, скажем так 1000 знаков или более… (насколько хватит фантазии!).

**Применение**

* [Составление кода](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) для [процессоров](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) ([микроконтроллеров](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%BB%D0%B5%D1%80)) низкой [разрядности](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D1%8F%D0%B4%D0%BD%D0%BE%D1%81%D1%82%D1%8C). Например, микроконтроллеры серии [AVR](https://ru.wikipedia.org/wiki/AVR) имеют [АЦП](https://ru.wikipedia.org/wiki/%D0%90%D0%A6%D0%9F) с разрядностью 10 [бит](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82) и регистры с разрядностью 8 бит. Этого недостаточно для обработки информации с АЦП; без длинной арифметики не обойтись;
* [Криптография](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F). Большинство систем [подписывания](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%8C) и [шифрования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) данных используют целочисленную арифметику по модулю m, где m — очень большое [натуральное](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%82%D1%83%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) число, не обязательно [простое](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE). Например, при реализации метода [шифрования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [RSA](https://ru.wikipedia.org/wiki/RSA), [криптосистемы Рабина](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%A0%D0%B0%D0%B1%D0%B8%D0%BD%D0%B0) или [схемы Эль-Гамаля](https://ru.wikipedia.org/wiki/%D0%A1%D1%85%D0%B5%D0%BC%D0%B0_%D0%AD%D0%BB%D1%8C-%D0%93%D0%B0%D0%BC%D0%B0%D0%BB%D1%8F) требуется обеспечить точность результатов [умножения](https://ru.wikipedia.org/wiki/%D0%A3%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) и [возведения в степень](https://ru.wikipedia.org/wiki/%D0%92%D0%BE%D0%B7%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B2_%D1%81%D1%82%D0%B5%D0%BF%D0%B5%D0%BD%D1%8C) порядка 10309;
* Математическое и финансовое [ПО](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). Результат вычисления на бумаге должен совпадать с результатом работы компьютера с точностью до последнего разряда. В частности, [калькулятор Windows](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BB%D1%8C%D0%BA%D1%83%D0%BB%D1%8F%D1%82%D0%BE%D1%80_Windows) (начиная с [Windows 95](https://ru.wikipedia.org/wiki/Windows_95)) проводит четыре арифметических действия с намного большей точностью, чем позволяет процессор [x86](https://ru.wikipedia.org/wiki/X86). Для научных и инженерных расчётов длинная арифметика применяется редко, так как ошибки во входных данных обычно намного больше, чем ошибки округления;
* [Спортивном программировании](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%BE%D1%80%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

# Длинная арифметика

**Длинная арифметика**— это набор программных средств (структуры данных и алгоритмы), которые позволяют работать с числами гораздо больших величин, чем это позволяют стандартные типы данных.

Цифры могут использоваться из той или иной системы счисления, обычно применяются десятичная система счисления и ее степени (десять тысяч, миллиард), либо двоичная система счисления.

Операции над числами в этом виде длинной арифметики производятся с помощью "школьных" алгоритмов сложения, вычитания, умножения, деления столбиком. Впрочем, к ним также применимы алгоритмы быстрого умножения: Быстрое преобразование Фурье и Алгоритм Карацубы,

Источник: [27]

Прежде всего необходимо понимать, что бесконечно длинное число можно представить только в виде динамического массива, что нам и предстоит сделать. Но даже, если числа представлять в виде динамических массивов, некоторые ограничения будут накладываться все равно. Например, длинна такого числа будет ограничена объемом памяти компьютера. Также следует понимать, что при использовании операций сложения и умножения, результат будет занимать больше места в памяти компьютера, нежели операнды.

Каждую арифметическую операцию рассмотрим отдельно

## Сложение длинных чисел

Рассмотрим арифметическую операцию сложения, применяемую в длинной арифметике. Алгоритм этого нехитрого арифметического действия, на удивление, простой. Он выглядит так:

// определяем длину массива суммы

**if** (size\_a > size\_b)

    length = size\_a + 1;

**else**

    length = size\_b + 1;

**for** (**int** ix = 0; ix < length; ix++)

{

    b[ix] += a[ix]; // суммируем последние разряды чисел

    b[ix + 1] += (b[ix] / 10); // если есть разряд для переноса, переносим его в следующий разряд

    b[ix] %= 10; // если есть разряд для переноса он отсекается

}

**if** (b[length - 1] == 0)

    length--;

“Виновники торжества” – то есть числа, которые мы будем складывать, предположительно записаны в массивы a и b. Необходимо учесть, что они записаны “зеркально”, то есть первый элемент массива соответствует последней цифре соответствующего числа, второй элемент – предпоследней, и т. д. Размеры длин чисел хранятся в переменных size\_a и size\_b, но Вы можете использовать любые другие. Вы конечно же задумались: “Зачем здесь оператор выбора if, строки 2–5?” или “Для чего он здесь?”. В этом блоке кода мы определяем максимальную длину числа, полученного в результате суммирования. Ведь, чаще всего суммируемые числа, разной длинны, одно больше, другое меньше, а нам нужно выделить память так, чтобы каждое число вместилось.

Далее, в алгоритме, делаем так, как нас учили на уроках математики: сначала складываем отдельные разряды, начиная с конца, строка 9; делим получившуюся сумму на 10 и получаем целую часть от деления на десять, которую мы сразу прибавляем к следующему разряду, строка 10.  В строке 11, мы отсекаем первый разряд полученного числа, если, конечно, он есть.  
Вот и все. Главное не забыть, что число будет храниться в массиве b и выводить его следует с конца.

## Вычитание длинных чисел

Вторая, наиболее используемая арифметическая операция – это вычитание. И эта часть статьи поможет Вам научиться писать алгоритмы вычитания больших и огромных чисел.   
Будем считать, что наши числа хранятся в массивах a и b, m и n – длинны этих чисел соответственно. Следует учесть, что числа записаны “зеркально”(см. выше). Конечно, если мы знаем какое число больше, то задача упрощается. Но мы можем и не знать этого. Тогда нам следует сперва найти, какое из чисел больше. Это нам понадобится для определения знака получившегося числа, то есть, если первое число меньше второго, то в ответе появится минус. И так, приступим к написанию первой части алгоритма, то есть определению большего числа. Алгоритм выглядит так:

**int** k = 3; // если к == 3, значит числа одинаковой длинны

length = size\_a;

**if** (size\_a > size\_b)

{

    length = size\_a;

    k = 1; // если к == 1, значит первое число длиннее второго

}

**else**

**if** (size\_b > size\_a)

    {

        length = size\_b;

        k = 2; // если к == 2, значит второе число длиннее первого

    }

**else** // если числа одинаковой длинны, то необходимо сравнить их веса

**for** (**int** ix = 0; ix < length;) // поразрядное сравнение весов чисел

        {

**if** (a[ix] > b[ix]) // если разряд первого числа больше

            {

                k = 1; // значит первое число длиннее второго

**break**; // выход из цикла for

            }

**if**(b[ix] > a[ix]) // если разряд второго числа больше

            {

                k = 2; // значит второе число длиннее первого

**break**; // выход из цикла for

            }

        } // конец for

Теперь поясню написанное. Сначала Вы можете увидеть, что переменной k придается значение 3. В данной части алгоритма переменная k является флагом результата проверки. Если числа равны, то k останется равно 3, если первое больше второго, то k примет значение 1, если второе больше первого, то k примет значение 2. Переменная length примет значение длинны большего числа. Теперь перейдем к обоснованию работоспособности этого алгоритма. Сравнение чисел происходит в два этапа. Сначала мы сравниваем длинны чисел: какое число длиннее, то и больше, строки 1 — 11. Если числа одинаковой длинны, то переходим к по разрядовому сравнению, строки 13 — 26. Начинаем по порядку сравнивать разряды начиная с самого старшего, так мы определим, больший вес числа. В этом и заключается суть и сложность первой части. Теперь перейдем ко второй части алгоритма — вычитание. Она выглядит так:

**int** difference (**int** \*x, **int** \*y, **int** \*z, **int** length)

{

**for** (**int** ix = 0; ix < (length - 1); ix++) // проход по всем разрядам числа, начиная с последнего, не доходя до первого

    {

**if** (ix < (length - 1)) // если текущий разряд чисел не первый

        {

            x[ix + 1]--; // в следующуем разряде большего числа занимаем 1.

            z[ix] += 10 + x[ix]; // в ответ записываем сумму значения текущего разряда большего числа и 10-ти

        } **else**  // если текущий разряд чисел - первый

                z[ix] += x[ix]; // в ответ суммируем значение текущего разряда большего числа

        z[ix] -= y[ix]; // вычитаем значение текущего разряда меньшего числа

**if** (z[ix] / 10 > 0) // если значение в текущем разряде двухразрядное

        {

            z[ix + 1]++; // переносим единицу в старший разряд

            z[ix] %= 10; // в текущем разряде отсекаем ее

        }

    }

**return** 0;

}

Для самого вычитания удобно написать функцию, ведь нам тогда не придется писать два алгоритма для двух случаев: когда первое число больше второго, и наоборот. В массиве x мы содержим большее число, в массиве y – меньшее, в массиве z – результат. Алгоритм довольно простой: для каждого разряда мы добавляем 10, с учетом вычитания из старшего разряда — 1. Это делается для упрощения вычитания разрядов. Эта операция делается лишь в том случае, когда рассматриваемый разряд не является последним в массиве (первым в числе). После вычитания разрядов мы проверяем получившееся число в данном разряде в массиве z. Ответ запишется в массив z, причем “зеркальным” (см. выше) способом. Процедуру следует вызывать следующим образом:

**if** (k == 1) difference(a,b,c, length); - если первое число больше второго,

**if** (k == 2) difference(b,a,c, length); - если второе число больше первого.

Теперь ответ будет храниться в массиве c все в том же “зеркальном” порядке. Вот мы и научились вычитать большие числа.

## Умножение длинных чисел (базовый метод)

length = size\_a + size\_b + 1;

**for** (**int** ix = 0; ix < size\_a; ix++)

**for** (**int** jx = 0; jx < size\_b; jx++)

        c[ix + jx - 1] += a[ix] \* b[jx];

**for** (**int** ix = 0; ix < length; ix++)

{

    c[ix + 1] +=  c[ix] / 10;

    c[ix] %= 10;

}

**while** (c[length] == 0)

    length-- ;

Вот так выглядит алгоритм задачи. Теперь попробуем “это” разобрать, точнее разобраться, как это работает. Сначала у нас имелось два числа в массивах a и b все в том же “зеркальном” виде. Длинны чисел хранятся в переменных size\_a и size\_b. В переменной length хранится длинна результирующего числа. Она будет равна либо сумме длин первоначальных чисел, либо этой сумме увеличенной на единицу. Но так, как мы не знаем точной длинны полученного числа, то возьмем длину побольше, то есть второй вариант. Теперь после этих не хитрых подсчетов, приступим к перемножению чисел. Будем их перемножать так, как нас учили в школе. Для этого запускаем два цикла: один до size\_a, другой до size\_b. После этих циклов вы можете увидеть еще один до length. благодаря ему в записи числа в массиве, в каждой ячейке массива мы получаем по одной цифре полученного числа. Последний цикл нужен, что бы узнать точную длину полученного числа, ведь предположенная нами длина числа может быть больше действительной. Ответ будет храниться в массиве c, все в том же “зеркальном” виде.

## Умножение длинных чисел (Алгоритм Карацубы)

**История алгоритма**

В 1960 году [Андрей Колмогоров](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D0%BC%D0%BE%D0%B3%D0%BE%D1%80%D0%BE%D0%B2,_%D0%90%D0%BD%D0%B4%D1%80%D0%B5%D0%B9_%D0%9D%D0%B8%D0%BA%D0%BE%D0%BB%D0%B0%D0%B5%D0%B2%D0%B8%D1%87) проводил семинар, посвящённый математическим задачам кибернетики. Одной из рассматриваемых на семинаре задач стало [умножение](https://ru.wikipedia.org/wiki/%D0%A3%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) двух {n}-разрядных целых чисел. Основным известным методом умножения в то время было умножение «в столбик», которое при алгоритмической реализации требовало O(n^{2}) [элементарных операций](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D0%B9) (сложений или умножений одноразрядных чисел). Колмогоров выдвинул гипотезу, что умножение «в столбик» является оптимальным алгоритмом умножения двух чисел в том смысле, что [время работы](https://ru.wikipedia.org/wiki/%D0%92%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B0) любого метода умножения не меньше  n^{2} по порядку величины. На правдоподобность «гипотезы  n^{2}» указывало то, что метод умножения «в столбик» известен не менее четырёх тысячелетий, и если бы был более быстрый метод умножения, то он, вероятно, уже был бы найден. Однако, через неделю 23-летний [Анатолий Карацуба](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%80%D0%B0%D1%86%D1%83%D0%B1%D0%B0,_%D0%90%D0%BD%D0%B0%D1%82%D0%BE%D0%BB%D0%B8%D0%B9_%D0%90%D0%BB%D0%B5%D0%BA%D1%81%D0%B5%D0%B5%D0%B2%D0%B8%D1%87) предложил новый метод умножения двух {n}-значных чисел с оценкой времени работы O() и тем самым опроверг «гипотезу  n^{2}».

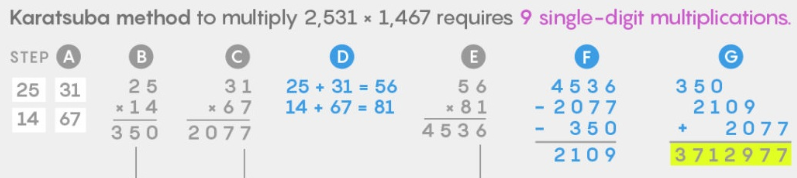
Метод Карацубы относится к алгоритмам вида «[разделяй и властвуй](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D1%8F%D0%B9_%D0%B8_%D0%B2%D0%BB%D0%B0%D1%81%D1%82%D0%B2%D1%83%D0%B9_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0))», наравне с такими алгоритмами как [двоичный поиск](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA), [быстрая сортировка](https://ru.wikipedia.org/wiki/%D0%91%D1%8B%D1%81%D1%82%D1%80%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0) и др. Формулы рекурсивного сведения, используемые в методе Карацубы, были известны ещё [Чарльзу Бэббиджу](https://ru.wikipedia.org/wiki/%D0%91%D1%8D%D0%B1%D0%B1%D0%B8%D0%B4%D0%B6,_%D0%A7%D0%B0%D1%80%D0%BB%D0%B7), который, однако, не обратил внимания на возможность использования лишь трёх рекурсивных умножений вместо четырёх.

**Описание алгоритма**

a) разбиваем числа по середине  
b) перемножаем левые части первого и второго

c) перемножаем правые части части первого и второго

d) складываем левую и правую часть первого числа (также поступаем со вторым числом)  
e) перемножаем эти суммы  
f) считаем e – b – c  
g) собираем итоговую сумму из b, c и f



## Умножение длинных чисел (Быстрое и Дискретное преобразование фурье, Алгоритм Кули-Тьюки)

**История алгоритма**

Алгоритм и его рекурсивная реализация были изобретены около 1805 года [К. Гауссом](https://ru.wikipedia.org/wiki/%D0%93%D0%B0%D1%83%D1%81%D1%81,_%D0%9A%D0%B0%D1%80%D0%BB_%D0%A4%D1%80%D0%B8%D0%B4%D1%80%D0%B8%D1%85) при [интерполировании](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D1%8F) траекторий астероидов [Паллада](https://ru.wikipedia.org/wiki/(2)_%D0%9F%D0%B0%D0%BB%D0%BB%D0%B0%D0%B4%D0%B0) и [Юнона](https://ru.wikipedia.org/wiki/(3)_%D0%AE%D0%BD%D0%BE%D0%BD%D0%B0). Тогда открытие не получило широкого распространения и было опубликовано лишь после смерти учёного на [новой латыни](https://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BB%D0%B0%D1%82%D1%8B%D0%BD%D1%8C). Результат Гаусса несколько раз переоткрывался в различных формах в течение последующих 150 лет и стал популярным после публикации в 1965 году статьи [Дж. Кули](https://en.wikipedia.org/wiki/James_Cooley)  (англ.)[рус.](https://ru.wikipedia.org/w/index.php?title=%D0%9A%D1%83%D0%BB%D0%B8,_%D0%94%D0%B6%D0%B5%D0%B9%D0%BC%D1%81&action=edit&redlink=1) из [IBM](https://ru.wikipedia.org/wiki/IBM) и [Дж. Тьюки](https://ru.wikipedia.org/wiki/%D0%A2%D1%8C%D1%8E%D0%BA%D0%B8,_%D0%94%D0%B6%D0%BE%D0%BD) из [Принстона](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%81%D1%82%D0%BE%D0%BD%D1%81%D0%BA%D0%B8%D0%B9_%D1%83%D0%BD%D0%B8%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%82%D0%B5%D1%82), в которой алгоритм был в очередной раз переоткрыт, а также описывалась удобная реализация для [ЭВМ](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%BE-%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0).

Тот факт, что первооткрывателем алгоритма является Гаусс, был обнаружен лишь через несколько лет после публикации Кули и Тьюки. В своей статье они ссылались только на работу [И. Дж. Гуда](https://ru.wikipedia.org/wiki/%D0%93%D1%83%D0%B4,_%D0%98%D1%80%D0%B2%D0%B8%D0%BD%D0%B3_%D0%94%D0%B6%D0%BE%D0%BD), в которой был описан [алгоритм Гуда — Томаса](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%93%D1%83%D0%B4%D0%B0_%E2%80%94_%D0%A2%D0%BE%D0%BC%D0%B0%D1%81%D0%B0).

**Дискретное преобразование Фурье (ДПФ)**

Пусть имеется многочлен n-ой степени:

 A(x) = a_0 x^0 + a_1 x^1 + \ldots + a_{n-1} x^{n-[...]

Не теряя общности, можно считать, что nявляется степенью 2. Если в действительности nне является степенью 2, то мы просто добавим недостающие коэффициенты, положив их равными нулю.

Из теории функций комплексного переменного известно, что комплексных корней n-ой степени из единицы существует ровно n. Обозначим эти корни через w_{n,k}, k = 0 \ldots {n-1}, тогда известно, что w_{n,k} = e^{ i \frac{ 2 \pi k }{ n } }. Кроме того, один из этих корней w_n = w_{n,1} = e^{ i \frac{ 2 \pi }{ n } }(называемый главным значением корня n-ой степени из единицы) таков, что все остальные корни являются его степенями: w_{n,k} = (w_n)^k.

Тогда **дискретным преобразованием Фурье (ДПФ)** (discrete Fourier transform, DFT) многочлена A(x)(или, что то же самое, ДПФ вектора его коэффициентов (a_0, a_1, \dots, a_{n-1})) называются значения этого многочлена в точках x = w_{n,k}, т.е. это вектор:  {\rm DFT}(a_0, a_1, \ldots, a_{n-1}) = (y_0, y_1,[...]  
 = (A(w_n^0), A(w_n^1), \ldots, A(w_n^{n-1})). 

Аналогично определяется и **обратное дискретное преобразование Фурье** (InverseDFT). Обратное ДПФ для вектора значений многочлена (y_0, y_1, \ldots y_{n-1})— это вектор коэффициентов многочлена (a_0, a_1, \ldots, a_{n-1}):

 {\rm InverseDFT}(y_0, y_1, \ldots, y_{n-1}) = (a_[...]

Таким образом, если прямое ДПФ переходит от коэффициентов многочлена к его значениям в комплексных корнях n-ой степени из единицы, то обратное ДПФ — наоборот, по значениям многочлена восстанавливает коэффициенты многочлена.

**Применение ДПФ для быстрого умножения полиномов**

Пусть даны два многочлена Aи B. Посчитаем ДПФ для каждого из них: {\rm DFT}(A)и {\rm DFT}(B)— это два вектора-значения многочленов.

Теперь, что происходит при умножении многочленов? Очевидно, в каждой точке их значения просто перемножаются, т.е.

 (A \times B)(x) = A(x) \times B(x). 

Но это означает, что если мы перемножим вектора {\rm DFT}(A)и {\rm DFT}(B), просто умножив каждый элемент одного вектора на соответствующий ему элемент другого вектора, то мы получим не что иное, как ДПФ от многочлена A \times B:

 {\rm DFT} (A \times B) = {\rm DFT} (A) \times {\r[...]

Наконец, применяя обратное ДПФ, получаем:

 A \times B = {\rm InverseDFT}( {\rm DFT} (A) \tim[...]

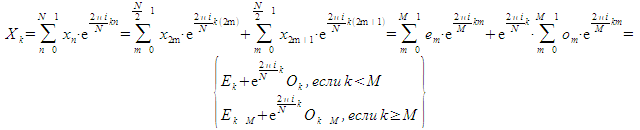
где, повторимся, справа под произведением двух ДПФ понимается попарные произведения элементов векторов. Такое произведение, очевидно, требует для вычисления только O(n)операций. Таким образом, если мы научимся вычислять ДПФ и обратное ДПФ за время O(n \log n), то и произведение двух полиномов (а, следовательно, и двух длинных чисел) мы сможем найти за ту же асимптотику.

Следует заметить, что, во-первых, два многочлена следует привести к одной степени (просто дополнив коэффициенты одного из них нулями). Во-вторых, в результате произведения двух многочленов степени nполучается многочлен степени 2n-1, поэтому, чтобы результат получился корректным, предварительно нужно удвоить степени каждого многочлена (опять же, дополнив их нулевыми коэффициентами).

**Алгоритм Cooley-Tukey**

Рассмотрим случай, когда . Так как для многочленов это всегда можно сделать, дополнив нулями в конце, не будем останавливаться на других случаях. В теории рассмотрим только прямое преобразование, а при реализации учтем оба варианта.

Основная идея ― разбить сумму на половину слагаемых с четными номерами и половину с нечетными. Построим для них преобразованные последовательности. Обозначим указанные последовательности и для четных номеров, и для нечетных ().



**Подключение к длинной арифметике**

Для умножения двух многочленов нужно представить их в виде своих значений в корнях из 1, перемножить эти значения, а потом восстановить многочлен по значениям. Из показанного выше следует, что это легко сделать тремя преобразованиями Фурье (двумя прямыми и одним обратным). Единственное, где можно ошибиться – это неправильно задать количество точек. Оно должно соответствовать степени не только исходных многочленов, но и степени их произведения.

Для перемножения длинных чисел нужно не забыть про переходы через разряд, а также про ограничение на основание системы счисления, связанное с «переполнением» мантиссы вещественного числа.

# Список литературы

1. <https://habr.com/ru/post/124258/>
2. <https://ru.wikipedia.org/wiki/%D0%94%D0%BB%D0%B8%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B0%D1%80%D0%B8%D1%84%D0%BC%D0%B5%D1%82%D0%B8%D0%BA%D0%B0#%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9>
3. <https://cyberleninka.ru/article/n/dlinnaya-arifmetika-v-issledovaniyaah-statistiki-pervyh-tsifr-stepeney-dvoyki-chisel-fibonachchi-i-prostyh-chisel>
4. <https://forany.xyz/a-836>
5. <https://habr.com/ru/post/207754/>
6. <https://habr.com/ru/post/113642/>
7. <https://algorithmica.org/ru/fft>
8. <https://parallel.ru/fpga/fft>
9. <https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F>
10. <https://habr.com/ru/post/113642/>
11. <https://habr.com/ru/company/otus/blog/449996/>
12. <https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D1%83%D0%BB%D0%B8_%E2%80%94_%D0%A2%D1%8C%D1%8E%D0%BA%D0%B8>
13. <http://cppstudio.com/post/5036/>
14. <https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE%D1%87%D0%BB%D0%B5%D0%BD%D0%BE%D0%B2>
15. <https://algowiki-project.org/ru/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D1%83%D0%BB%D0%B8-%D0%A2%D1%8C%D1%8E%D0%BA%D0%B8_%D0%B1%D1%8B%D1%81%D1%82%D1%80%D0%BE%D0%B3%D0%BE_%D0%BF%D1%80%D0%B5%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%A4%D1%83%D1%80%D1%8C%D0%B5_%D0%B4%D0%BB%D1%8F_%D1%81%D1%82%D0%B5%D0%BF%D0%B5%D0%BD%D0%B5%D0%B9_%D0%B4%D0%B2%D0%BE%D0%B9%D0%BA%D0%B8>
16. <http://www.ict.nsc.ru/jct/getfile.php?id=1853>
17. <https://kaf401.rloc.ru/DSP/FFTalg.pdf>
18. <https://ru.wikibrief.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm>
19. <https://ru.algorithmica.org/cs/numerical/newton/>
20. <https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm>
21. <https://russianblogs.com/article/71671149759/>
22. <https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B5%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%A4%D1%83%D1%80%D1%8C%D0%B5>
23. <https://ru.dsplib.org/content/dft/dft.html>
24. <https://www.math.spbu.ru/ru/mmeh/AspDok/pub/2008/prosekov.pdfhttps://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A4%D1%8E%D1%80%D0%B5%D1%80%D0%B0#:~:text=F%C3%BCrer's%20algorithm)%20%E2%80%94%20%D0%B1%D1%8B%D1%81%D1%82%D1%80%D1%8B%D0%B9%20%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%20%D1%83%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F,%D0%A8%D1%82%D1%80%D0%B0%D1%81%D1%81%D0%B5%D0%BD%D0%B0%2C%20%D0%BE%D0%BF%D1%83%D0%B1%D0%BB%D0%B8%D0%BA%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9%20%D0%B2%201971%20%D0%B3%D0%BE%D0%B4%D1%83>.
25. <http://omega.sp.susu.ru/books/conference/PaVT2011/short/015.pdf>
26. <https://russianblogs.com/article/6737570753/>
27. <https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F#%D0%94%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D1%83%D1%82%D1%91%D0%BC_%D0%B2%D1%8B%D1%87%D0%B8%D1%82%D0%B0%D0%BD%D0%B8%D0%B9>
28. <https://intellect.icu/dlinnaya-arifmetika-s-primerami-na-si-8291>