

ГУАП
КАФЕДРА № 51

ПРЕПОДАВАТЕЛЬ

доцент, кандидат техн. наук		Е.М.Линский
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ (ПРОЕКТУ)
УМНОЖЕНИЕ С ПОМОЩЬЮ БПФ

по курсу: ОСНОВЫ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	5611		П.П.Недошивин
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2017

СОДЕРЖАНИЕ

1	Постановка задачи	2
2	Умножение «столибком»	4
3	Алгоритм. Базовая теория	6
4	Алгоритм. Быстрое преобразование Фурье	9
5	Алгоритм. Реализация	13
6	Инструкция пользователя.	18
7	Тестирование	19
	Список литературы	20

1 ПОСТАНОВКА ЗАДАЧИ

В некоторых сферах деятельности человека приходится работать с большими числами. Какие проблемы при этом возникают? Такие числа выходят за диапазон значений какого-либо встроенного типа данных. Также возникает вопрос, как проводить операции над ними. Можно создать свой класс, в котором реализованы считывание, вывод и собственно необходимые операции. При этом можно добиться сокращения используемых ресурсов памяти и значительного ускорения работы алгоритмов. Например, можно реализовать операцию умножения «в столбик» для динамического массива целых чисел. Единственным ограничением на размер массива будет размер оперативной памяти компьютера.

Однако, так как размер чисел достаточно велик, то классические алгоритмы, такие как умножение «в столбик», работают медленно. Были придуманы методы, позволяющие значительно сократить затраты времени при работе с длинными числами. Первым появился класс алгоритмов быстрого умножения.

Широкое применение получил алгоритм, основанный на быстром преобразовании Фурье (БПФ) (англ. Fast Fourier Transform (FFT)). Данный алгоритм используется в криптографии, в математическом и инженерном программном обеспечении, требующем сверхвысокой точности, например, при расчете траектории движения и необходимой силы тяги космических носителей. Также используется в бухгалтерии для точного подсчета денежных и других средств [1].

Целью данной курсовой работы является разработка программы для перемножения длинных чисел. Согласно цели, задачами работы были:

1. анализ алгоритмов перемножения длинных чисел (БПФ и «в столбик»);
2. разработка и реализация программы на языке C++ для перемножения длинных чисел, основанной на БПФ (нерекурсивный вариант);

3. тестирование и анализ программы для перемножения длинных чисел.

2 УМНОЖЕНИЕ «СТОЛБКОМ»

Рассмотрим классический алгоритм умножения «в столбик».

Data: x, y, n_1, n_2

Result: res

$length = n_1 + n_2 + 1;$

$i = 0;$

while $i < n_1$ **do**

$j = 0;$

while $j < n_2$ **do**

$res_{i+j-1} += x_i * y_j;$

$j++;$

end

$i++;$

end

for $i = 0$ **to** $length$ **do**

$res_{i+1} += res_i / 10;$

$res_i = res_i \bmod 10;$

end

На вход поступают два числа x и y длины n_1 и n_2 соответственно. На выход идет произведение $x*y$ — число res . Изначально все его разряды равны нулю, его длина не превышает n_1+n_2+1 . Каждый разряд первого числа умножается на каждый разряд второго числа. Произведение i -го разряда x и j -го разряда y добавляется к $(i+j-1)$ -му разряду произведения. Так как после перемножения $x*y$ разряды res могут быть больше 9, требуется пересчет разрядов. Это реализуется при помощи переноса частного от деления на 10 в более старший разряд.

Оценим сложность алгоритма умножения «в столбик». Если представить, что происходит перемножение чисел одинаковой длины, равной n , то видно, что умножение разрядов выполняется n^2 раз, а перенос разрядов — $2n$ раз. Следовательно, приближительная сложность алгоритма — $O(n^2)$. При достаточно больших числах (например, с 1000 разрядами) данный алгоритм

будет неэффективен. Необходима разработка более быстрого алгоритма.

3 АЛГОРИТМ. БАЗОВАЯ ТЕОРИЯ

Прежде, чем рассмотреть предлагаемый далее алгоритм, определим некоторые понятия.

Для начала рассмотрим ключевое для дальнейших рассуждений понятие полинома. **Полином** $A(x)$ от переменной x над полем F имеет вид (значения принадлежат полю F):

$$A(x) = a_0x^0 + a_1x^1 + \dots + a_{n-1}x^{n-1}$$

Произведение полиномов: $C(x) = A(x)B(x)$. Умножение выполняется таким образом: каждое слагаемое многочлена $A(x)$ умножается на каждое слагаемое многочлена $B(x)$, после чего выполняется приведение подобных слагаемых с одинаковыми степенями. Степень многочлена-произведения равна сумме степеней сомножителей:

$$\deg(C) = \deg(A) + \deg(B)$$

Задание многочлена вектором коэффициентов

Пусть дан полином:

$$A(x) = a_0x^0 + a_1x^1 + \dots + a_nx^n$$

Тогда вектором коэффициентов будет являться $a = (a_0, a_1, \dots, a_n)$.

Задание многочлена набором значений

Фиксируем n различных точек x_0, x_1, \dots, x_{n-1} . Многочлен $A(x)$ степени ниже n однозначно определяется своими значениями в этих точках, то есть набором из n пар аргумент-значение.

$$(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1}),$$

где $y_k = A(x_k)$, для $k = 0, 1, \dots, n-1$.

Представление многочлена с помощью значений в заданных точках удобно для многих операций: в частности, для умножения достаточно умножить значения многочленов в каждой из точек (значения многочленов должны быть заданы в одних и тех же точках). Надо иметь в виду, что при умножении степени многочленов складываются, и произведение двух многочленов степени меньше n может иметь степень больше n . При этом она заведомо меньше $2n$, так что для восстановления произведения достаточно $2n$ точек. Таким образом, умножая два многочлена степени меньше n , полезно с самого начала иметь значения многочленов не в n , а в $2n$ точках. Затем эти значения можно перемножить за время $O(n)$ и получить представление произведения в виде набора пар аргумент-значение.

Полиномиальная интерполяция — построение многочлена по имеющемуся набору пар аргумент-значение.

Теорема (однозначность интерполяции) [2]

Для любого множества $(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})$ пар аргумент-значение (все x_i различны) существует единственный многочлен $A(x)$ степени ниже n , для которого $y_k = A(x_k)$, для $k = 0, 1, \dots, n-1$.

Комплексные корни из единицы

Комплексным корнем степени n из единицы называют такое комплексное число ω , при котором $\omega^n = 1$. Имеется ровно n комплексных корней для данного уравнения. Эти корни равномерно распределены на окружности единичного радиуса с центром в нуле.

Решения уравнения (рис.1) $\omega^n = 1$ имеют вид $e^{\frac{2\pi k}{n}}$, для $k = 0, 1, \dots, n-1$.

Рассмотрим некоторые свойства комплексных корней из 1.

Лемма 1 (о сокращении)

Для любых целых $k \geq 0, n \geq 0, d > 0$ верно

$$\omega_{dn}^{dk} = \omega_n^k$$

Следствие

Для любого четного $m \geq 0$

$$\omega_{2m}^m = \omega_2^1 = -1$$

Лемма 2 (о делении пополам)

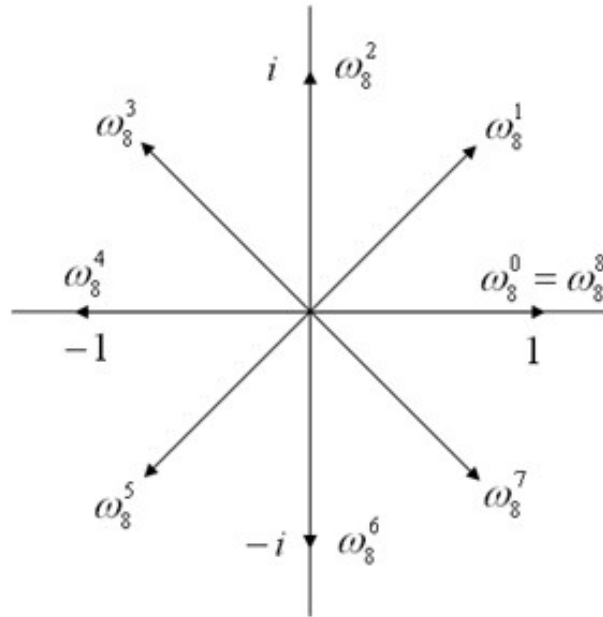


Рис. 1: Комплексные корни из 1

Если $n > 0$ чётно, то, возведя в квадрат все n комплексных корней степени n из единицы, мы получим все $n/2$ комплексных корней степени $n/2$ из единицы (каждый — по два раза).

Лемма 3 (о сложении)

Для любого целого $n \geq 0$ и неотрицательного целого k , не кратного n , выполнено равенство

$$(\omega_n^k)^0 + (\omega_n^k)^1 + \dots + (\omega_n^k)^{n-1} = 0$$

4 АЛГОРИТМ. БЫСТРОЕ ПРЕОБРАЗОВАНИЕ ФУРЬЕ

Дискретным преобразованием Фурье (ДПФ) (англ. Discrete Fourier transform (DFT)) многочлена $A(x)$ называются значения этого многочлена в точках $x = \omega_{n,k}$, т.е. это вектор:

$$\begin{aligned} DFT(a_0, a_1, \dots, a_{n-1}) &= (y_0, y_1, \dots, y_{n-1}) = (A(\omega_{n,0}), A(\omega_{n,1}), \dots, A(\omega_{n,n-1})) = \\ &= (A(\omega_n^0), A(\omega_n^1), \dots, A(\omega_n^{n-1})) \end{aligned}$$

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

$$InverseDFT(y_0, y_1, \dots, y_{n-1}) = (a_0, a_1, \dots, a_{n-1})$$

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

Даны два многочлена $A(x)$ и $B(x)$. Посчитаем ДПФ для каждого из них. Для умножения многочленов необходимо перемножить в каждой точке их значения. Это означает, что если умножить вектора $DFT(A)$ и $DFT(B)$, просто умножив каждый элемент одного вектора на соответствующий ему элемент другого вектора, то получится ДПФ от многочлена $A(x)B(x)$.

Применив обратное ДПФ, получаем:

$$A(x)B(x) = InverseDFT(DFT(A) * DFT(B))$$

Произведение ДПФ требует для вычисления только $O(n)$ операций (так как происходит попарное умножение точек). Таким образом, если вычислять ДПФ и обратное ДПФ за время $O(n \log n)$, то и произведение двух полиномов

(а, следовательно, и двух длинных чисел, которые можно представить в виде полиномов) можно найти за то же время.

Следует отметить, 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 n получается многочлен степени $2n-1$, поэтому, чтобы результат получился корректным, предварительно нужно удвоить степени каждого многочлена (опять же, дополнив их нулевыми коэффициентами).

Быстрое преобразование Фурье — это метод, позволяющий вычислять ДПФ за время $O(n \log n)$. Этот метод основывается на свойствах комплексных корней из единицы (а именно, на том, что степени одних корней дают другие корни). Основная идея БПФ заключается в разделении вектора коэффициентов на два вектора, рекурсивном вычислении ДПФ для них, и объединении результатов в одно БПФ [3].

Пусть имеется многочлен $A(x)$ степени n , где n — степень 2, $n > 1$:

$$A(x) = a_0x^0 + a_1x^1 + \dots + a_{n-1}x^{n-1}$$

Разделим его на два многочлена, один — с чётными, а другой — с нечётными коэффициентами:

$$A_0(x) = a_0x^0 + a_2x^1 + \dots + a_{n-2}x^{n-1}$$

$$A_1(x) = a_1x^0 + a_3x^1 + \dots + a_{n-1}x^{n-1}$$

Получить исходный многочлен можно по формуле (1):

$$A(x) = A_0(x^2) + xA_1(x^2)$$

Многочлены A_0 и A_1 имеют вдвое меньшую степень, чем многочлен A . Для эффективной работы алгоритма требуется за линейное время по вычисленным $DFT(A_0)$ и $DFT(A_1)$ вычислить $DFT(A)$. Искомый алгоритм быстрого преобразования Фурье — это стандартная схема алгоритма «разделяй и властвуй», для которой известна оценка $O(n \log n)$.

Итак, имея вычисленные вектора $(y_k^0)_{k=0}^{n/2-1} = DFT(A_0)$ и $(y_k^1)_{k=0}^{n/2-1} = DFT(A_1)$, найдём выражение для $(y_k)_{k=0}^{n-1} = DFT(A)$.

Значения для первой половины коэффициентов получены через (1):

$$y_k = y_k^0 + \omega_n^k y_k^1$$

при $k = 0 \dots n/2-1$

Аналогично для второй половины коэффициентов:

$$y_{k+n/2} = y_k^0 - \omega_n^k y_k^1$$

при $k = 0 \dots n/2-1$ (через (1) и свойства комплексных корней из 1).

Эти формулы получили название «преобразование бабочки» [4].

Обратное БПФ

Итак, пусть дан вектор $(y_0, y_1, \dots, y_{n-1})$, содержащий значения многочлена $A(x)$ степени n в точках ω_n^k . Требуется восстановить коэффициенты $(a_0, a_1, \dots, a_{n-1})$ многочлена. Эта известная задача называется интерполяцией; для решения этой задачи будет применен алгоритм, практически не отличающийся от прямого БПФ.

ДПФ можно представить в матричном виде:

$$\begin{pmatrix} \omega_n^0 & \omega_n^0 & \omega_n^0 & \omega_n^0 & \dots & \omega_n^0 \\ \omega_n^0 & \omega_n^1 & \omega_n^2 & \omega_n^3 & \dots & \omega_n^{n-1} \\ \omega_n^0 & \omega_n^2 & \omega_n^4 & \omega_n^6 & \dots & \omega_n^{2(n-1)} \\ \omega_n^0 & \omega_n^3 & \omega_n^6 & \omega_n^9 & \dots & \omega_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega_n^0 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \omega_n^{3(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Тогда вектор $(a_0, a_1, \dots, a_{n-1})$ можно найти, умножив вектор $(y_0, y_1, \dots, y_{n-1})$ на обратную матрицу к матрице, стоящей слева (которая, кстати, называется матрицей Вандермонда):

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} \omega_n^0 & \omega_n^0 & \omega_n^0 & \omega_n^0 & \dots & \omega_n^0 \\ \omega_n^0 & \omega_n^1 & \omega_n^2 & \omega_n^3 & \dots & \omega_n^{n-1} \\ \omega_n^0 & \omega_n^2 & \omega_n^4 & \omega_n^6 & \dots & \omega_n^{2(n-1)} \\ \omega_n^0 & \omega_n^3 & \omega_n^6 & \omega_n^9 & \dots & \omega_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega_n^0 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \omega_n^{3(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Непосредственной проверкой можно убедиться в том, что эта обратная матрица такова:

$$\frac{1}{n} \begin{pmatrix} \omega_n^0 & \omega_n^0 & \omega_n^0 & \omega_n^0 & \dots & \omega_n^0 \\ \omega_n^0 & \omega_n^{-1} & \omega_n^{-2} & \omega_n^{-3} & \dots & \omega_n^{-(n-1)} \\ \omega_n^0 & \omega_n^{-2} & \omega_n^{-4} & \omega_n^{-6} & \dots & \omega_n^{-2(n-1)} \\ \omega_n^0 & \omega_n^{-3} & \omega_n^{-6} & \omega_n^{-9} & \dots & \omega_n^{-3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega_n^0 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \omega_n^{-3(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{pmatrix}$$

Таким образом, получаем формулу:

$$a_k = \frac{1}{n} \sum_{j=0}^{n-1} y_j \omega_n^{-kj}$$

Сравниваем ее с формулой для y_k :

$$y_k = \sum_{j=0}^{n-1} a_j \omega_n^{kj}$$

Формулы почти не отличаются, поэтому коэффициенты a_k можно находить таким же алгоритмом «разделяй и властвуй», как и прямое БПФ, только вместо ω_n^k везде надо использовать ω_n^{-k} , а каждый элемент результата надо разделить на n .

Таким образом, вычисление обратного ДПФ почти не отличается от вычисления прямого ДПФ, и его также можно выполнять за время $O(n \log n)$.

5 АЛГОРИТМ. РЕАЛИЗАЦИЯ

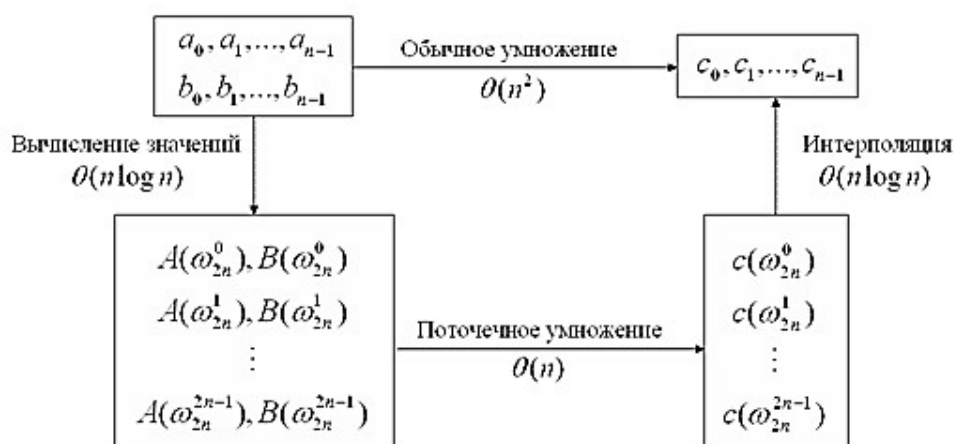


Рис. 2: Общая схема алгоритма программы

Алгоритм быстрого умножения двух полиномов можно записать так [5]:

1. Удвоение количества коэффициентов. Дополнить полиномы $A(x)$ и $B(x)$ нулевыми старшими коэффициентами так, чтобы количество элементов в каждом полиноме было $2n$ элементов.
2. Вычисление значений. При помощи быстрого преобразования Фурье вычислить значения многочленов $A(x)$ и $B(x)$ в точках, являющихся корнями степени $2n$ из единицы.
3. Поточечное умножение. Поточечно умножить полученные значения многочленов $A(x)$ и $B(x)$ друг на друга. В результате получаются значения многочлена $C(x) = A(x)B(x)$ в корнях степени $2n$ из единицы.
4. Интерполяция. Получить коэффициенты многочлена $C(x)$ при помощи обратного преобразования Фурье.

Общая схема алгоритма и сложности шагов показаны на рис. 2.

Способ хранения данных

Множители и произведения хранятся поразрядно в векторах целых чисел. Знаки чисел логически запоминаются и учитываются. В функции реализации БПФ используются векторы комплексных чисел.

Рекурсивная реализация [6]

Рассмотрим рекурсивную реализацию быстрого преобразования Фурье.

```
array RecursiveFFT(array a)
```

```
1 n = length(a)
2 if (n == 1) return a
3  $\omega_n = e^{2\pi/n}$ 
4  $\omega = 1$ 
5  $y^0 = \text{RecursiveFFT}(a_0, a_2, \dots, a_{n-2})$ 
6  $y^1 = \text{RecursiveFFT}(a_1, a_3, \dots, a_{n-1})$ 
7 for  $k = 0; k < n / 2; k++$  do
    8  $y_k = y_k^0 + \omega y_k^1$ 
    9  $y_{k+n/2} = y_k^0 - \omega y_k^1$ 
   10  $\omega = \omega \omega_n$ 
   11
end
12 return y
```

Функция RecursiveFFT работает следующим образом.

В строке 1 переменной n присваивается количество элементов в векторе а.

Строка 2 образует «базу рекурсии». Для вектора единичной длины дискретным преобразованием Фурье является сам вектор, так как степень корня нулевая.

В третьей строке соответствующей переменной присваивается главное значение корня степени n из единицы.

В строке номер 4 инициализируется единицей переменная, в которой будет храниться текущее значение корня.

В пятой и шестой строках вычисляется дискретное преобразование Фурье для векторов, состоящих из членов только при четных или при нечетных степенях.

В цикле, состоящем из строк 7, 8, 9, 10, собираются вместе результаты рекурсивных вычислений.

Восьмая строка основана на формуле (1) из предыдущего раздела.

Девятая же строка следует из того, что восьмая строка верна и из леммы о делении пополам.

Десятая и седьмая строки гарантируют нам то, что будут перебраны все комплексные корни степени n из единицы.

Время выполнения данного алгоритма равно $O(n \log n)$. Глубина дерева рекурсивных вызовов равна $\log n$, а время операций, выполняемых на каждом уровне $O(n)$. Дерево входных векторов рекурсивных вызовов для $n = 8$ приведено на рис. 3.

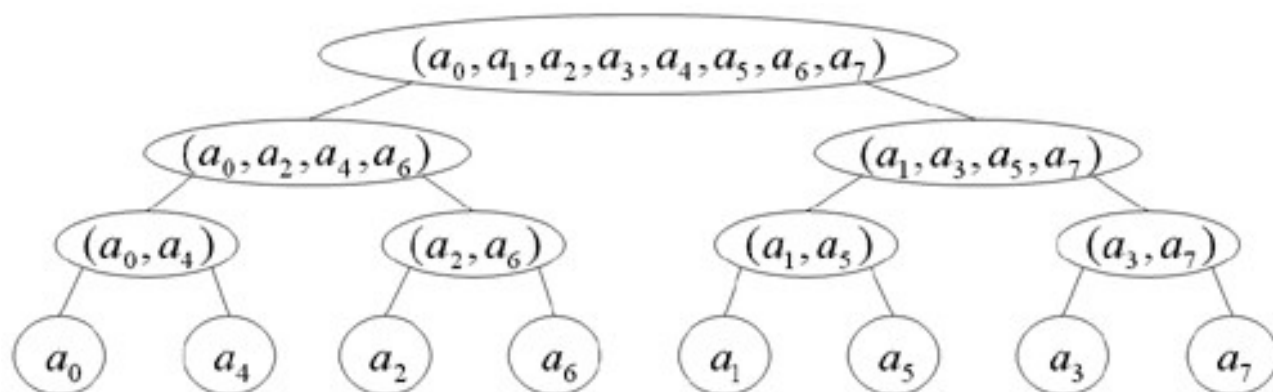


Рис. 3: Дерево рекурсивных вызовов

Итеративная (нерекурсивная) реализация

Рекурсию можно заменить вычислением снизу вверх, расположив элементы исходного вектора в том порядке, в котором они появляются в листьях дерева рекурсивных вызовов. В примере выше числа 0, 4, 2, 6, 1, 5, 3, 7 соответствуют обратной двоичной записи номеров позиций, на которых они стоят: 000, 100, 010, 110, 001, 101, 011, 111. Данная закономерность показывает, что индексы элементов исходного вектора определяются, начиная с младшего бита двоичной записи индекса. Поэтому если в позиции i каждого элемента a_i инвертировать порядок битов и переупорядочивать элементы вектора a в соответствии с новыми индексами, то получится **поразрядно обратная перестановка** вектора. Это позволит проводить вычисление БПФ в самом векторе a без создания временных векторов четных и нечетных коэффициентов.


```

typedef complex<double> base;
void fft (vector<base> a, bool invert)
int n = (int) a.size();
for int i = 1, j = 0; i < n; ++i do
    int bit = n » 1;
    for j >= bit; bit »= 1 do
        | j -= bit;
    end
    j += bit;
    if i < j then
        | swap (a[i], a[j]);
    end
end
for int len = 2; len <= n; len «= 1 do
    double ang = 2*PI/len * (invert ? -1 : 1);
    base wlen (cos(ang), sin(ang));
    for int i = 0; i < n; i += len do
        base w (1);
        for int j = 0; j < len/2; ++j do
            base u = a[i + j];
            base v = a[i + j + len / 2] * w;
            a[i + j] = u + v;
            a[i + j + len/2] = u - v;
            w *= wlen;
        end
    end
end
if invert then
    for int i = 0; i < n; ++i do
        | a[i] /= n;
    end
end

```

Необходимо провести поразрядно обратную перестановку вектора *a*. Для этого сначала вычисляется количество значащих битов ($\log n$) в числе *n*.

Затем для каждой позиции i находится соответствующая позиция j , битовая запись которой есть битовая запись числа i , записанная в обратном порядке. Если получившаяся в результате позиция оказалась больше i , то элементы в этих двух позициях надо обменять (если не это условие, то каждая пара обменяется дважды, и в итоге ничего не произойдёт).

Например, пусть j заранее подсчитано. Тогда, при переходе к числу $i+1$ нужно к j прибавить единицу, но в «инвертированной» системе счисления. В обычной двоичной системе счисления прибавить единицу — значит удалить все единицы, стоящие на конце числа, а перед ними поставить единицу. Соответственно, в «инвертированной» системе проход идет, начиная со старшего бита числа. Единицы обнуляются, остановка происходит в первом встретившемся нулевом бите, который заменяется единицей.

Затем выполняется $\log n - 1$ стадий алгоритма (начиная с 2 и до $\log n$), на каждой из которых вычисляются ДПФ для блоков длины 2^k , где k — счетчик этого цикла. Для всех блоков будет одно и то же значение корня ω_{2^k} , которое запоминается в переменной $wlen$. Цикл по i проходит по блокам, а вложенный цикл по j применяет преобразование бабочки ко всем элементам блока.

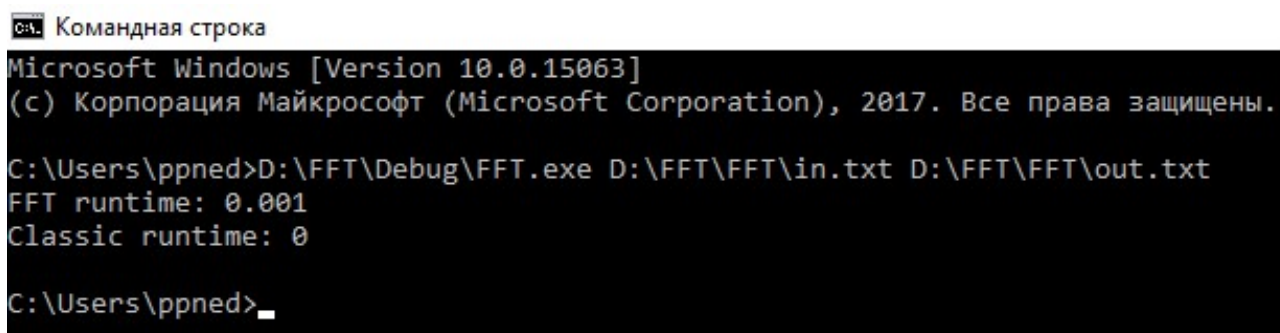
Как видно, алгоритм, основанный на БПФ, значительно превосходит классический алгоритм умножения «столбиком». Конкретные примеры, подтверждающие данное утверждение, находятся в разделе 7.

6 ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ

Программа запускается через командную строку. Аргументы командной строки:

1. Полный адрес исполняемого файла программы
2. Полный адрес входного файла программы
3. Полный адрес выходного файла программы

Входной и выходной файлы должны иметь формат txt (быть текстовыми файлами). На вход подается 2 целых числа в десятичной системе счисления, записанные по одному в строке без посторонних символов (например, пробелов). На выход поступают 2 целых числа – произведения чисел из входного файла: первое число подсчитано при помощи БПФ, второе – при помощи классического алгоритма «столбиком». В командной строке будет выведено время умножения каждым способом (рис. 4).



```
cmd Командная строка
Microsoft Windows [Version 10.0.15063]
(c) Корпорация Майкрософт (Microsoft Corporation), 2017. Все права защищены.

C:\Users\ppned>D:\FFT\Debug\FFT.exe D:\FFT\FFT\in.txt D:\FFT\FFT\out.txt
FFT runtime: 0.001
Classic runtime: 0

C:\Users\ppned>
```

Рис. 4: Пример работы программы в командной строке

Пример входного и выходного файлов:

in.txt

9876543210

123456789

out.txt

1219326311126352690

7 ТЕСТИРОВАНИЕ

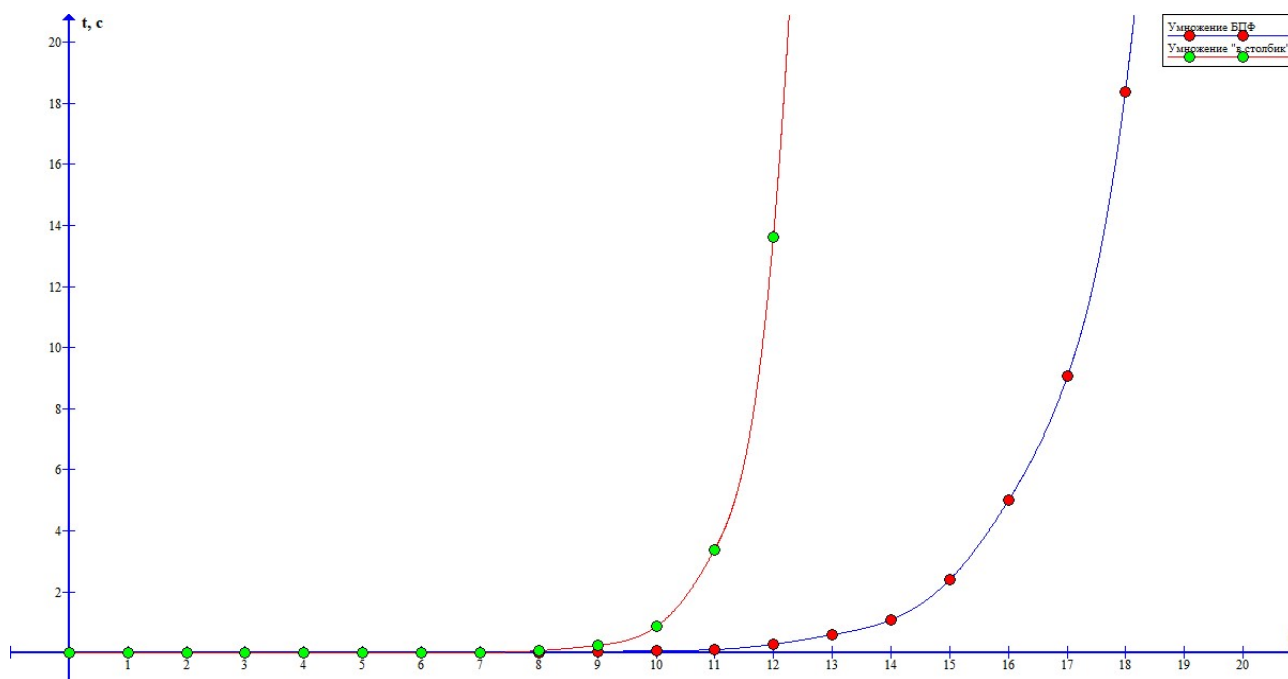


Рис. 5: График времени работы умножения «столбиком» и с помощью БПФ

Тестирование проводилось на числах разной длины. На графике представлены временные диаграммы умножения «столбиком» (красный цвет) и умножения с помощью БПФ (синий цвет). На оси ОХ отложена длина числа (2^i разрядов), на оси ОУ – время в секундах (рис. 5).

Как видно из графиков, на малых числах алгоритмы по времени работают практически одинаково. Расхождение начинается с приблизительно 1000-разрядных чисел. Функция БПФ начинает возрастать на более высоких значениях, что означает, что данный алгоритм дает больший выигрыш по времени, чем классический алгоритм «столбиком».

СПИСОК ЛИТЕРАТУРЫ

- [1] Быстрое умножение полиномов - ИТМО ФИТиП
<https://rain.ifmo.ru/cat/view.php/theory/math/fft-2004> (2004)
- [2] Р.Блейхут
Теория и практика кодов, контролирующих ошибки. М.:Мир, 1986
- [3] Быстрое преобразование Фурье - Википедия
<https://ru.wikipedia.org/wiki/Быстрое-преобразование-Фурье> (2017)
- [4] А.С.Глинченко
Цифровая обработка сигналов, ИПЦ СПбГУТ, 2001
- [5] Быстрое умножение многочленов при помощи преобразования Фурье - это просто
<https://habrahabr.ru/post/113642/> (2011)
- [6] Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн
Алгоритмы. Построение и анализ, М.:ООО«И.Д.Вильямс», 2013