

Быстрое преобразование Фурье глазами программиста

А. А. Григорьев

Дискретное преобразование Фурье (ДПФ) - это, пожалуй, самый распространенный инструмент цифровой обработки сигналов. В значительной мере это связано с существованием быстрого алгоритма его вычисления, известного как быстрое преобразование Фурье - БПФ (Fast Fourier Transform – FFT). В наши дни мало кому выпадает доля реализовать алгоритм FFT самостоятельно – готовые библиотечные FFT-модули широко представлены во всевозможных средах разработки. Фокус внимания смещен в сторону применений FFT. Тем не менее, хотя бы общее знакомство с принципами его реализации полезно любому имеющему отношение к цифровой обработке – пользуясь неким инструментом нeliшне представлять себе как он устроен. К тому же, анализ реализаций FFT весьма поучителен для начинающего программиста.

1. Целеопределение и лобовая атака

Если формально, то N -точечное ДПФ – это алгоритм, который принимает на вход N -блок $X = [x_0, x_1, \dots, x_{N-1}]$ вообще говоря комплексных выборочных значений сигнала и формирует N -блок $Y = [y_0, y_1, \dots, y_{N-1}]$ комплексных же выборочных значений спектра по правилу:

$$y_f = \sum_{t=0}^{N-1} x_t \xi^{ft}.$$

Здесь $\xi = e^{-j \frac{2\pi}{N}}$ – это примитивный комплексный корень степени N из единицы (одно из N решений уравнения $z^N = 1$). По сути ДПФ – это просто умножение вектора X на квадратную матрицу $H = \{h_{f,t} = \xi^{ft}\}$.

Степени ξ^k , $k = [0 : N - 1]$ примитивного корня ξ пробегают все множество корней степени N из единицы. Перемножение корней сводится к сложению показателей: $\xi^k \xi^l = \xi^{(k+l)}$, а возведение корня в целую степень выливается в перемножение показателей: $(\xi^k)^l = \xi^{kl}$. Условие $\xi^N = 1 = \xi^0$ обеспечивает при этом автоматическое приведение показателей по модулю N , так что запись ξ^{ft} следует читать как $\xi^{ft \bmod N}$.

Фигурирующие в определении ДПФ индексы f, t удобно с самого начала рассматривать как элементы кольца Z_N целых чисел от 0 до $N - 1$ с операциями сложения и умножения по модулю N . Как видно из рис. 1, объект Z_N называется кольцом ровно потому, что он на самом деле круглый. Отрезок $[0 : N - 1]$ замыкает в круг переход из $N - 1$ в 0 при добавлении очередной единицы.

Полезно немного поупражняться в устном счете в кольце Z_{15} . Для пытливого ума открывается много нового. Обнаруживается, к примеру, что $5 \times 3 = 0$. С обычными числами такого не бывает. Говорят, что числа 3 и 5 являются делителями нуля в Z_{15} . Список делителей нуля в Z_{15} исчерпывается числами 3, 5, 6, 9, 10, 12, не взаимно простыми с модулем 15. Прочие же 8 ненулевых чисел обратимы – для каждого a из них найдется b , такое что $ab = 1$. Все обратимые числа являются примитивными

элементами Z_{15} , в том смысле, что их кратные ax , $x \in Z_{15}$ пробегают все элементы Z_{15} . Перечисленные факты иллюстрируют практически все базовые представления алгебраической теории модульных колец.

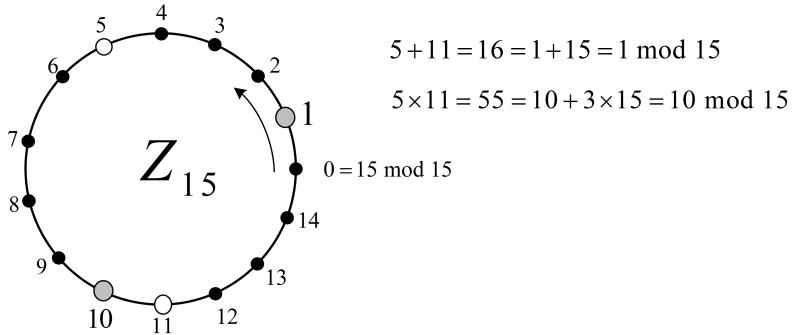


Рис. 1. Кольцо Z_{15}

Представление индексов элементами кольца позволяет переписать формулу для ДПФ в удобной форме

$$y_f = \sum_{t \in Z_N} x_t \xi^{ft}; \quad f, t \in Z_N, \quad (1)$$

избавляющей от необходимости каждый раз оговаривать, что арифметические операции над индексами f, t выполняются по модулю N .

ДПФ обратимо. Блок отсчетов x_t сигнала восстанавливается по блоку отсчетов y_f спектра по схеме:

$$x_t = \frac{1}{N} \sum_{f \in Z_N} x_f \xi^{-tf}. \quad (2)$$

Обратное ДПФ отличается от прямого только знаком показателя степени корня ξ и нормировочным множителем $\frac{1}{N}$.

Формы (1) и (2) прямого и обратного ДПФ просты по структуре, но малопригодны на практике. Проблема в том, что сложность вычислений по ним растет как N^2 с увеличением числа точек N – для вычисления каждого из N значений x_f требуется N умножений x_t на ξ^{ft} и $(N-1)$ -о сложение. Уже при $N = 2^{16}$ речь пойдет о выполнении порядка $2^{32} \simeq 4 \times 10^9$ операций над комплексными числами.

Быстрые алгоритмы, о которых речь пойдет далее, позволяют снизить сложность вычисления ДПФ с квадратичной $O(N^2)$ до логарифмически-линейной $O(N \log N)$.

2. О природе ускорения

В основе всех регулярных алгоритмов быстрого вычисления ДПФ лежит факторизация – разложение модуля N на сомножители.

Целое N раскладывается в произведение $N = \prod_{k=1}^K P_k$, $P_k = p_k^{n_k}$ степеней $n_k \geq 1$ простых чисел p_k . Стандартный прием факторизации кольца Z_N в декартово произведение координатных колец Z_{P_k} , опирающийся на знаменитую китайскую теорему об остатках, позволяет свести вычисление ДПФ на N -точках к вычислениям ДПФ на P_k -точках. Это снижает сложность ДПФ с N^2 до $N \sum_{k=1}^K P_k$.

В плане программирования факторизация на взаимно простые множители реализуется штатно. Она выливается в векторизацию – переход от работы с одномерным массивом, индексированным элементами $t \in Z_N$, к работе с многомерным массивом,

индексированным вектор-индексом $(t^{(1)}, t^{(2)}, \dots, t^{(K)})$, $t^{(k)} \in Z_{P_k}$. Мы вернемся к ней позднее.

Более замысловат логически прием ускорения ДПФ на числе точек $N = p^k$, являющимся степенью простого. Число $N = p^k$ прекрасно раскладывается на k одинаковых множителей p . Но эти множители не взаимно просты. Так что стандартная факторизация в декартово произведение здесь не работает. На замену ей приходят иные методы, опирающиеся на представление элементов кольца Z_{p^k} в позиционной системе счисления по основанию p .

Наконец, для малых простых p существует целая плеяда эвристических алгоритмов ускорения ДПФ, известных как алгоритмы Винограда.

В компьютере все двоичное. Поэтому почти исключительное распространение получил алгоритм FFT на числе точек $N = 2^n$, выражаящемся степенью двойки. Когда заходит речь о быстром преобразовании Фурье, почти наверняка имеется в виду FFT на 2^n -точках. Начнем с обсуждения именно этого случая.

3. FFT на 2^n точках

Взгляните на псевдокод алгоритма 1 ниже. Нечто похожее Вы увидите, анализируя любой попавший под руку исходный текст реализации FFT.

```
// tw[n] = e^{-j \frac{2\pi n}{2^n}}, n = [0 : 2^{(n-1)} - 1] - массив поворачивающих множителей
Function FFT(X) // X-блок данных для вычисления FFT на 2^n точках
    N=Length(X); XOR=N/2; twSTEP = 1;
    for it = 1 to n do
        base = 0;
        for ib = 1 to twSTEP do
            for ind = 0 to XOR-1 do
                // Вычисление бабочек
                temp = X[base+ind];
                X[base+ind] = temp+X[base+ind+XOR];
                X[base+ind+XOR] = temp-X[base+ind+XOR];
                // Умножение на поворачивающие множители
                X[base+ind+XOR] = X[base+ind+XOR]*tw[ind*twSTEP];
            end
            base = base +2*XOR;
        end
        XOR=XOR/2; twSTEP = 2*twSTEP;
    end
    BrPermutation(X); // Бит-реверс перестановка элементов блока X
end
```

Алгоритм 1: Алгоритм FFT на 2^n точках

С виду ничего сложного. Весь алгоритм укладывается в десяток строк. Он принимает на вход 2^n -блок X временных выборок x_t . По завершении алгоритма в этом же блоке образуются выборки y_f спектра. По дороге алгоритм обращается к массиву поворачивающий множителей (twiddles) $tw[n] = \xi^n = e^{-j \frac{2\pi n}{2^n}}$, в котором лежат предвычисленные значения степеней корня ξ . Вот и весь интерфейс.

Внутри же находится тройной вложенный цикл, в теле которого производятся некоторые манипуляции над элементами X . Обнаруживаются сложения-вычитания (слой бабочек) и умножения на поворачивающие множители. Вызываемая в конце процедура реверс перестановки ($BrPermutation(X)$) к вычислениям отношения не

имеет. Она просто некоторым образом перетасовывает элементы X . Анализ ее текста (алгоритм 7 ниже) только усугубляет ощущение полного кошмара.

Нормальный человек вряд ли сразу поймет, почему предписанная этим алгоритмом последовательность действий над элементами X действительно приводит к вычислению ДПФ (1). И все же этот алгоритм написан человеком, отправляясь именно от этой формулы. Просто этот человек руководствовался некоторыми соображениями, нам пока недоступными. Убедительный пример того, что исходный текст алгоритма – это еще не все. Объяснение алгоритма порой требует больших усилий, чем его написание.

3.1. Перейдем к объяснению

В основе FFT на $N = 2^n$ точках лежит все та же факторизация. Только теперь, когда N делится на 2 много раз, факторизация становится рекурсивной.

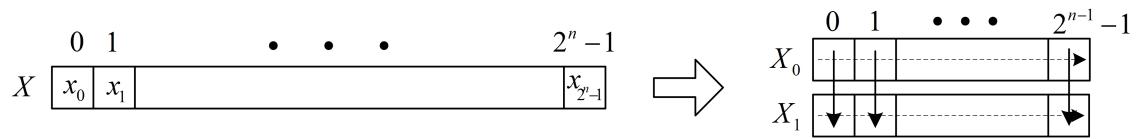


Рис. 2. Шаг рекурсивного разбиения блока

Разобьем блок X из 2^n точек на два блока X_0, X_1 по 2^{n-1} точек в каждом, рис. 2, так, чтобы вычисление ДПФ на X удалось свести к вычислению вначале 2-точечных ДПФ по всем 2^{n-1} столбцам, а затем – двух ДПФ по строкам X_0, X_1 .

Пусть C_n – сложность (Complexity) вычисления ДПФ на 2^n точках. Тогда

$$C_n = 2^{n-1}C_1 + 2C_{n-1}.$$

Каждый из субблоков X_0, X_1 можно, в свою очередь, разбить на два. Это даст

$$C_{n-1} = 2^{n-2}C_1 + 2C_{n-2},$$

или

$$C_n = 2^{n-1}C_1 + 2(2^{n-2}C_1 + 2C_{n-2}) = 22^{n-1}C_1 + 2^2C_{n-2}.$$

После k -й итерации рекурсивного разбиения блоков получится

$$C_n = k2^{n-1}C_1 + 2^kC_{n-k}.$$

На итерации $k = n - 1$, когда длина строк снизится до $2^{n-k} = 2^{n-n+1} = 2$, получится

$$C_n = (n-1)2^{n-1}C_1 + 2^{n-1}C_1 = n2^{n-1}C_1 = \frac{C_1}{2}N \log_2(N), \quad N = 2^n.$$

Таким образом, рекурсивное разбиение – это именно то, что снижает сложность алгоритма с N^2 до $\frac{C_1}{2}N \log_2(N)$, где C_1 – сложность ДПФ на $2^1 = 2$ точках.

ДПФ же на двух точках вычисляется совсем просто. При $N = 2$ $\xi = e^{-j\frac{2\pi}{N}} = e^{-j\pi}$ и сумма (1) сводится к

$$y_f = \sum_{t=0}^1 x_t e^{-j\pi f t},$$

или

$$y_0 = x_0 + x_1,$$

$$y_1 = x_0 - x_1.$$

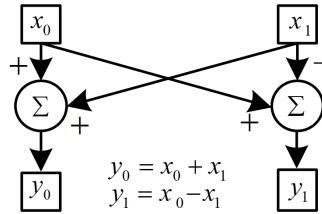


Рис. 3. Операция бабочки

Это базовая для FFT на 2^n точках операция бабочки (Butterfly), рис. 3, которая уже была мельком упомянута в алгоритме 1. Для ее вычисления требуется одно сложение и одно вычитание.

3.2. Схемы рекурсивного разбиения

Встает вопрос – как именно выполнять разбиения. Возможна масса способов разбить блок на два. Не все они годятся. Нужно, чтобы ДПФ по исходному блоку сводилось к ДПФ по столбцам и строкам разбиения. На самом деле при $N = 2^n$, не представим произведением взаимно простых множителей, таких «по настоящему хороших» разбиений не существует вовсе. Приходится придумывать некие «просто хорошие» разбиения. Это приводит к появлению упомянутых выше «поворачивающих множителей» (twiddles), которые и привносят в FFT операцию умножения. Оказывается, что не все сводится только к бабочкам.

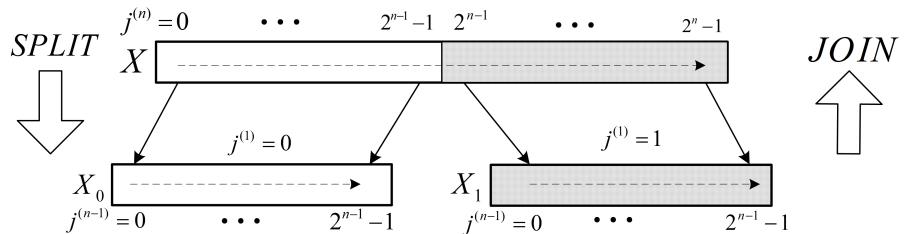


Рис. 4. Разбиение по дихотомии

Первой приходит в голову очевидная схема рекурсивного разбиения по дихотомии, показанная на рис. 4. На каждом шаге левая и правая половины элементов блока X , индексированных n -разрядными числами $j^{(n)}$ из кольца Z_{2^n} , переносятся в левый и правый субблоки X_0, X_1 , индексированные $(n-1)$ -разрядными числами $j^{(n-1)}$ из $Z_{2^{n-1}}$. Сами же блоки разбиения нумеруются двоичным числом $j^{(1)}$ из Z_{2^1} – старшим разрядом двоичного кода числа $j^{(n)}$.

При разбиении дихотомией скалярный индекс $j^{(n)}$ вычисляется по вектор-индексу $(j^{(1)}, j^{(n-1)})$ как

$$j^{(n)} = 2^{n-1}j^{(1)} + j^{(n-1)}.$$

Двойственная схема рекурсивного разбиения на рис. 5 по понятной причине называется прореживанием, реже – децимацией (Decimation). В левый субблок X_0 попадают четные элементы X , в правый X_1 – нечетные. Номером субблока $j^{(1)}$ оказывается младший разряд двоичного кода числа $j^{(n)}$, а формула для вычисления индекса $j^{(n)}$ по вектор-индексу $(j^{(1)}, j^{(n-1)})$ принимает вид:

$$j^{(n)} = j^{(1)} + 2j^{(n-1)}.$$

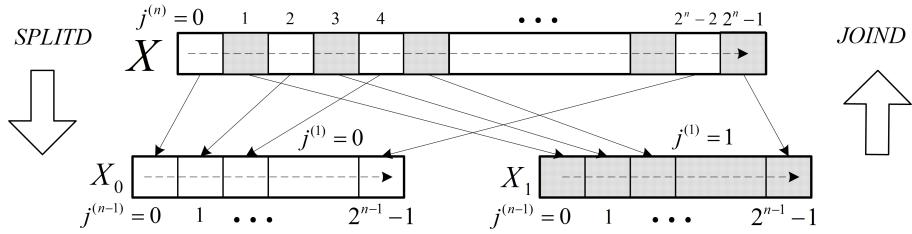


Рис. 5. Разбиение прореживанием

Ключевая идея ускорения ДПФ на 2^n -точках заключается в применении двойственных схем разбиения для временной и частотной областей. Алгоритм FFT с прореживанием по выходу получается, когда во временной области используется разбиение дихотомией

$$t^{(n)} = 2^{n-1}t^{(1)} + t^{(n-1)},$$

а в частотной - прореживанием:

$$f^{(n)} = f^{(1)} + 2f^{(n-1)}$$

Перемножение этих представлений для временного и частотного индексов дает четыре слагаемых

$$f^{(n)}t^{(n)} = 2^{n-1}f^{(1)}t^{(1)} + 2f^{(n-1)}t^{(n-1)} + f^{(1)}t^{(n-1)} + 2^n t^{(1)}f^{(n-1)},$$

из которых последнее оказывается кратным 2^n , то есть равным нулю по модулю 2^n . Поэтому в кольце Z_{2^n}

$$f^{(n)}t^{(n)} = 2^{n-1}f^{(1)}t^{(1)} + 2f^{(n-1)}t^{(n-1)} + f^{(1)}t^{(n-1)}.$$

Исчезновение четвертого слагаемого – это и есть ключевой момент. Оставшиеся три слагаемые – это перемножение индексов $f^{(1)}, t^{(1)}$ в кольце Z_2 , индексов $f^{(n-1)}, t^{(n-1)}$ в кольце $Z_{2^{n-1}}$ и перепутывающий фактор $f^{(1)}t^{(n-1)}$ – произведение элемента $f^{(1)}$ из Z_2 на элемент $t^{(n-1)}$ из $Z_{2^{n-1}}$.

То, что у нас получилось, в алгебре известно как факторизация кольца Z_{2^n} в скрещенное (twisted) произведение колец Z_2 и $Z_{2^{n-1}}$.

3.3. Схемы рекурсивной факторизации ДПФ

Переход к вектор-индексам $(f^{(1)}, f^{(n-1)}), (t^{(1)}, t^{(n-1)})$ в формуле для ДПФ (1) дает

$$y_{f^{(1)}, f^{(n-1)}} = \sum_{t^{(n-1)} \in Z_{2^{n-1}}} (\xi^2)^{f^{(n-1)}t^{(n-1)}} \xi^{f^{(1)}t^{(n-1)}} \sum_{t^{(1)} \in Z_2} x_{t^{(1)}, t^{(n-1)}} (\xi^{2^{n-1}})^{f^{(1)}t^{(1)}}. \quad (3)$$

Внутренняя сумма здесь – это ДПФ на двух точках, которое вычисляется по индексу $t^{(1)}$ много раз для всех возможных значений $t^{(n-1)}$ ($\xi^{2^{n-1}} = e^{-j\pi}$ – это как раз корень из единицы для двухточечного ДПФ). Внешняя сумма – это ДПФ на 2^{n-1} -й точке по индексу $t^{(n-1)}$ (ξ^2 – это корень из единицы для него). Картина портит только промежуточный поворачивающий множитель $\xi^{f^{(1)}t^{(n-1)}}$. Его присутствие – это плата за разложение 2^n на не взаимно простые множители 2 и 2^{n-1} . С ним приходится жить. После вычисления всех внутренних ДПФ получается двумерный массив со смешанным индексом $(f^{(1)}, t^{(n-1)})$. Перед вычислением внешних ДПФ элементы этого массива приходится домножить на поворачивающие множители $\xi^{f^{(1)}t^{(n-1)}}$. Внешних ДПФ два. Одно вычисляется при $f^{(1)} = 0$, второе – при $f^{(1)} = 1$.

Имея факторизацию (3) перед глазами, несложно набросать рекурсивную процедуру вычисления FFT. Выглядит она примерно так:

```

Function  $Y = FFT(X)$ 
  if (Length(X) = 1) return X;
  X0,X1 = SPLIT(X);
  BATTERFLY(X0,X1); // Бабочки между элементами блоков X0,X1
  TWIDDLES(X1); // Поворачивающие множители
  Y0=FFT(X0);
  Y1=FFT(X1);
  Y = JOIND(Y0,Y1);
end

```

Алгоритм 2: FFT с прореживанием по выходу

Процедура принимает блок временных выборок X и возвращает блок Y выборок спектра. Если длина (Length) блока X равна единице (ДПФ на одной точке), ничего делать не нужно – возвращаем X . Иначе разбиваем блок на два – $X0 = \{x_{0,t^{(n-1)}}\}$ и $X1 = \{x_{1,t^{(n-1)}}\}$ по дихотомии – см. переход *SPLIT* на рис. 4. На парах элементов блоков X_0, X_1 вычисляем бабочки (двуточечные ДПФ) – процедура *BATTERFLY* (алгоритм 3). В блоке $X0$ оказываются числа с $f^{(1)} = 0$, в блоке $X1$ – с $f^{(1)} = 1$. С внутренней суммой в (3) покончено. Теперь домножаем элементы блока $X1$ на поворачивающие множители – процедура *TWIDDLES* (алгоритм 4). Блок $X0$ в домножении не нуждается – все его поворачивающие множители $\xi^{0t^{(n-1)}}$ равны единице. Осталось вычислить два внешние ДПФ. Для этого дважды рекурсивно вызываем саму функцию $Y = FFT(X)$, передавая ей блоки $X0, X1$. Получив результаты $Y0, Y1$ с компонентами спектров, собираем из них единый блок Y обращением прореживания (см. переход *JOIND* на рис. 4) и возвращаем его в качестве результата.

```

Function  $BATTERFLY(B0,B1)$ 
  N = Length(B0);
  for  $n=0$  to  $N-1$  do
    | Temp = B0[n];
    | B0[n] = Temp+B1[n];
    | B1[n] = Temp-B1[n];
  end
end

```

Алгоритм 3: Бабочки между элементами блоков

```

Function  $TWIDDLES(B)$ 
  N = Length(B);
   $w = e^{j\pi/N}$ ;
  for  $n=0$  to  $N-1$  do
    | B[n] = B[n]* $w^n$ ;
  end
end

```

Алгоритм 4: Поворачивающие множители

Разработка алгоритма FFT с прореживанием по выходу завершена. Получившаяся простая рекурсивная процедура действительно работает, причем, при любой длине 2^n входного блока.

Двойственный алгоритм FFT с прореживанием по входу получается, когда разбиение прореживанием применяется во временной области

$$t^{(n)} = t^{(1)} + 2t^{(n-1)},$$

а разбиение дихотомией – в частотной

$$f^{(n)} = 2^{n-1}f^{(1)} + f^{(n-1)}.$$

Формула для перемножения индексов в кольце Z_{2^n} принимает при этом вид

$$f^{(n)}t^{(n)} = 2^{n-1}f^{(1)}t^{(1)} + 2f^{(n-1)}t^{(n-1)} + f^{(n-1)}t^{(1)},$$

отличающийся только формой перепутывающего фактора. И это небольшое отличие в буквальном смысле ставит все с ног на голову. Аналогичное (3) выражение для ДПФ принимает вид

$$y_{f^{(1)}, f^{(n-1)}} = \sum_{t^{(1)} \in Z_2} (\xi^{2^{n-1}})^{f^{(1)}t^{(1)}} \xi^{f^{(n-1)}t^{(1)}} \sum_{t^{(n-1)} \in Z_{2^{n-1}}} x_{t^{(1)}, t^{(n-1)}} (\xi^2)^{f^{(n-1)}t^{(1)}}. \quad (4)$$

Отличие в том, что «большие» ДПФ на 2^{n-1} точках теперь оказались внутри, а малые, двухточечные – снаружи. Первыми приходится вычислять внутренние ДПФ, а это требует разбиения на более мелкие блоки. Рекурсия уходит вглубь: пока разбиение на самые мелкие блоки не достигнуто, вычислять просто нечего.

Набросать же рекурсивную процедуру вычисления FFT с прореживанием по входу ничуть не сложнее, чем ее аналог с прореживанием по выходу:

```

Function Y = FFT(X)
  if (Length(X) = 1) return X;
  X0, X1 = SPLITD(X);
  Y0 = FFT(X0);
  Y1 = FFT(X1);
  TWIDDLE(Y1); // Поворачивающие множители
  BATTERFLY(Y0, Y1); // Бабочки на элементах блоков X0, X1
  Y = JOIN(Y0, Y1);
end

```

Алгоритм 5: FFT с прореживанием по входу

В ней все то же самое, но в обратном порядке. Входной блок разбивается на два прореживанием. Затем вызываются FFT для подблоков, потом идет умножение на поворачивающие множители, и, наконец – бабочки между блоками. Завершается все сборкой результата по дихотомии. Короче, все в полном соответствии в (4).

Рекурсивные процедуры FFT отличаются ясностью логической структуры, но не слишком эффективны. Многочисленные рекурсивные вызовы приводят к большим накладным расходам на управление вычислительным процессом и копирование данных из одного блока в другой. Процессор оказывается занятым не столько арифметикой, сколько вызовами подпрограмм и пересылками данных. Но, прежде чем приступать к улучшению этих процедур, нужно познакомиться с концепцией битреверса.

3.4. Бит реверс

Элементы 2^n -блоков X индексируются числами j из кольца Z_{2^n} . Эти числа представляют двоичными кодами в n -разрядном индексном регистре, рис. 6.



Рис. 6. Бит-реверс

Порядок нумерации разрядов регистра существенен. Именно он определяет веса двоичных разрядов при интерпретации содержимого регистра (j_{n-1}, \dots, j_0) как двоичного позиционного кода числа $j \in Z_{2^n}$. При стандартной (прямой) кодировке разряды нумеруются справа налево. Нумерация в обратном порядке дает реверсную кодировку чисел.

Один и тот же двоичный код (j_{n-1}, \dots, j_0) в регистре можно интерпретировать либо как код числа j в прямой кодировке, либо как код совсем другого числа $Br(j)$ в реверсной кодировке рис. 6. К примеру, $Br(1) = 4$ в Z_8 . Это определяет реверс-преобразование (Bit Revert) – отображение $Br(j)$ кольца Z_{2^n} в себя. Реверс преобразование обратимо: $Br(Br(j)) = j$. Это означает, что если $Br(j) = k$, то $Br(k) = j$. Существуют числа палиндромы, не изменяющиеся при реверсе битов: $Br(j) = j$. Нетрудно сообразить, что в Z_{2^n} имеется ровно $2^{n/2}$ палиндромов при четных n и $2^{(n+1)/2}$ – при нечетных.

Формально, прямая и реверсная кодировки вполне равнозначны. Но с позиций компьютера это совсем не так. При выполнении арифметических операций над числами происходят переносы из младших разрядов в старшие. Для прямой кодировки чисел цепи переносов реализованы аппаратно, для реверсной – они отсутствуют. Поэтому программная реализация арифметики в реверсной кодировке чисел – это серьезная головная боль.

Реверс преобразование индексов вызывает реверс-перестановку элементов x_j индексируемого ими блока X . Эта перестановка – $BrPermutation(X)$ – уже упоминалась выше с тексте алгоритма 1. Она попросту меняет местами элементы x_j и $x_{Br(j)}$ с индексами j и $Br(j)$, связанными реверс преобразованием.

Присутствие реверса в алгоритмах FFT вполне ожидаемо. Вернемся к факторизациям индексов дихотомией

$$j^{(n)} = 2^{n-1} j^{(1)} + j^{(n-1)}$$

и прореживанием

$$j^{(n)} = j^{(1)} + 2j^{(n-1)}.$$

Заметим, что при факторизации числа $j^{(n)}$ с позиционным кодом $j_0 + 2j_1 + \dots + 2^{n-1} j_{n-1}$ по дихотомии бит $j^{(1)}$ – это значение j_{n-1} старшего разряда позиционного кода, а при факторизации прореживанием – значение j_0 его младшего разряда. Последовательная, рекурсивная факторизация $j^{(n)}$ даст либо бит-вектор $(j_0, j_1, \dots, j_{n-1})$ – прореживание, либо реверсный вектор $(j_{n-1}, j_{n-2}, \dots, j_0)$ – дихотомия.

Для алгоритма ускорения ДПФ принципиально использование разных схем факторизации во временной и частотной областях. Но это с неизбежностью приводит к тому, что прямой кодировке индексов в одной области будет отвечать реверсная кодировка в другой. В какой из областей пойти на реверсную кодировку – не все равно.

На этапе умножении на поворачивающие множители ξ^{kj} существенно, чтобы кодировка индекса j была прямой – вычисление произведения kj в реверсной кодировке ограничено с безумием. При вычислении FFT с прореживанием по выходу умножение на поворачивающие множители ведется по временному индексу – необходима прямая кодировка во времени. При прореживании по входу требуется прямая кодировка частотного индекса.

Различие в кодировках временного и частотного индексов приводит к тому, что элементы временного блока X и частотного блока Y оказываются упорядоченными по разному. Поэтому присутствие реверс-перестановки в алгоритмах FFT вполне естественно.

В наших рекурсивных процедурах FFT реверс-перестановка выполняется «на лету» за счет того, что блоки разбиваются на субблоки по одной схеме, а сливаются по другой. Если блок X разбить на «кирпичи» (блоки длины 1) по дихотомии, а затем собрать блок Y прореживанием, то как раз получится реверс-перестановка. Соответствующий рекурсивный алгоритм $Y = BrPermutation(X)$ выглядит так.

```
Function Y = BrPermutation(X)
  if (Length(X) = 1) return X;
  X0,X1 = SPLIT(X);
  Y0=BrPermutation(X0);
  Y1=BrPermutation(X1);
  Y = JOIN(Y0,Y1);
end
```

Алгоритм 6: Рекурсивная реверс-перестановка

По существу, это рекурсивный алгоритм FFT , из которого убрана вся арифметика. Остался лишь скелет из разбиений/слияний блоков.

Реверс-перестановка эквивалента переходу от одной схемы разбиения к другой. Нетрудно сообразить, что разбиение по дихотомии эквивалентно реверс-перестановке элементов блока с последующим разбиением прореживанием и наоборот.

```
Function BrPermutation(X)
  N=Length(X); Bh = N/2; // Бит в старшем разряде
  j=0;
  for i= 1 to N-1 do
    m = Bh; // Поиск первого нуля со стороны старшего разряда
    while j>=m do
      | m=m/2;
    end
    j = j - m; // и вычитание единицы в этой позиции
    if i<j then
      | // перестановка
      | temp = X[i]; X[i] = X[j]; X[j] = temp;
    end
  end
end
```

Алгоритм 7: Реверс-перестановка

Для реализации реверс-перестановки элементов блока придуман гениальный алгоритм 7, текст которого кочует из одной реализации FFT в другую почти без изменений. Идея этого алгоритма такова: будем параллельно считать от 1 до $2^n - 1$ по i в прямой кодировке и по j – в реверсной, и переставлять элементы с индексами

i, j , если $i < j$. Условие $i < j$ позволяет обойти тривиальные перестановки с $i = j$, и блокирует обратные перестановки элементов, уже переставленных ранее: числа i и $j = Br(i)$ либо равны (ничего переставлять не нужно), либо одно из них меньше другого – перестановка выполняется только при $i < j$. Обратная перестановка при $i > j$ отменяется.

Реализация счета в реверс-кодировке так же весьма остроумна. Переносы из старших разрядов в младшие при сложении не реализованы, зато реализованы заемы из старших разрядов при вычитании. Будем отыскивать в текущем коде первый нуль со стороны старшего разряда, и вычитать в позиции этого нуля единицу. Возникающая цепочка заемов и обеспечивает счет в реверс-кодировке. Несколько первых этапов показаны на рис. 7.

$$\begin{array}{r}
 \begin{array}{r}
 -00000 = 0 \\
 -1 \\
 \hline
 -10000 = 1 \\
 -1 \\
 \hline
 -01000 = 2 \\
 -1 \\
 \hline
 -11000 = 3 \\
 -1 \\
 \hline
 00100 = 4
 \end{array}
 \end{array}$$

Рис. 7. Счет в реверс-кодировке

3.5. Маскируем рекурсию

Рекурсивное вычисление FFT обладает структурой двоичного дерева, рис. 8. Корневой узел, получив входной блок X , разбивает его блок на два субблока – X_0, X_1 , и передает их на обработку левому и правому узлам с нижнего слоя. Все узлы работают одинаково. Различие только в размере входных блоков. По мере работы узлов блоки последовательно расщепляются и передаются по дереву вниз на все более глубокие слои.

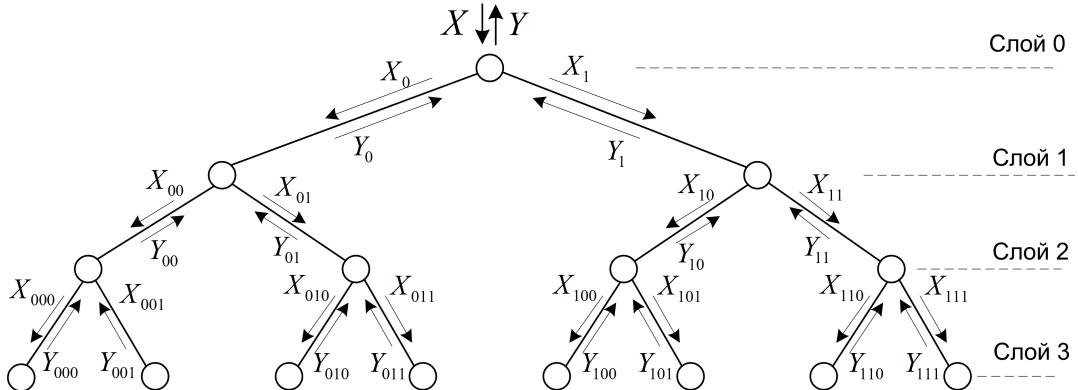


Рис. 8. Дерево рекурсивного вычисления

Наконец, узлы последнего слоя получают блоки длины 1, разбивать которые далее уже невозможно. Эти узлы просто возвращают полученные блоки наверх как блоки Y . Это запускает процесс встречного распространения блоков по дереву снизу вверх. Каждый из узлов, получив с нижнего слоя левый и правый блоки, собирает их в единый блок Y и передает его на верхний слой. Все завершается, когда последний Y -блок, собранный корневым узлом, выдается потребителю.

Рекурсивный алгоритм 2 с прореживанием по выходу устроен так, что все арифметические операции над элементами блоков выполняются на этапе их распространения вниз – перед передачей пар субблоков на нижний слой. Этап же распространения вверх просто реализует реверс-перестановку данных за счет сборки блоков прореживанием. Напротив, в алгоритме 3 с прореживанием по входу вся арифметика сосредоточена в фазе распространения блоков вверх.

Выигрыш в эффективности вычисления FFT достигается искусственной маскировкой его рекурсивной природы. Маскировку обеспечивает переход от вычислений по дереву к вычислениям по горизонтальным слоям его узлов. Заметим, что все узлы данного слоя выполняют одинаковые операции над блоками одного и того же размера. Причем, обрабатываемые ими блоки – это неперекрывающиеся субблоки некоторого большого блока X . Весь функционал слоя можно реализовать, пройдя поперек дерева по всем его узлам. При этом не обязательно разбивать блок X на субблоки реально. Достаточно научиться эффективно выделять эти субблоки в теле самого блока X .

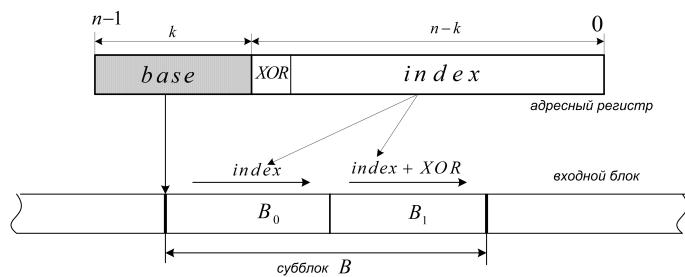


Рис. 9. Схема адресации блоков

В варианте FFT с прореживанием по выходу входной блок разбивается по дихотомии. Структура подблоков оказывается при этом совершенно прозрачной. Чтобы выделить подблоки для 2^k узлов слоя номер k нужно просто разбить блок X на 2^k равных частей. Это достигается мысленным разбиением n -разрядного адресного регистра – указателя на элементы блока X – на k -битовую базовую часть $base$ и $n - k$ -битовую индексную часть, рис. 9. Базовая часть служит указателем на голову обрабатываемого блока B , а индексная часть $index$ – задает смещения элементов внутри блока.

Старший разряд индексной части – это так называемый XOR -бит. По значению этого бита блок B разбивается на субблоки B_0, B_1 , между элементами которых и вычисляются бабочки. Ясно, что положение XOR -бита полностью определяет текущий формат адресного регистра. При проходе по самому верхнему слою дерева с единственным корневым узлом этот бит находится в старшем разряде – имеется всего один бит длины 2^n . Каждому спуску на слой вниз отвечает сдвиг XOR -бита на позицию вправо.

Теперь становится ясно, из каких соображений был написан алгоритм 1 (см. алгоритм 8 ниже). Автор имел перед глазами картинки дерева вычислений, рис. 8, и схемы адресации, рис. 9. Он сообразил, что нужно пройти сверху вниз по всем слоям дерева и организовал для этого внешний цикл. Внутри он устроил цикл перебора по всем узлам текущего слоя. Наконец, в следующем по вложенности цикле он реализовал обработку блока данных внутри узла – бабочки между его половинками и умножение старшей половины на поворачивающие множители. Потом он вспомнил, что фаза распространения блоков снизу вверх по дереву, которая и обеспечивает реверс-перестановку данных, полностью проигнорирована, и добавил в конец явный вызов процедуры $BrPermutation(X)$. Вот и вся работа.

```

//  $tw[n] = e^{-j \frac{2\pi n}{2^n}}$ ,  $n = [0 : 2^{(n-1)} - 1]$  - массив поворачивающих множителей
Function FFT(X) // X-блок данных для вычисления FFT на  $2^n$  точках
    N=Length(X); n=log2(N);
    XOR=N/2; twSTEP = 1;
    for it = 1 to n do // Цикл по слоям
        base = 0;
        for ib = 1 to twSTEP do // Цикл по узлам
            for ind = 0 to XOR-1 do // Цикл по индексу внутри блока
                // Вычисление бабочек
                temp = X[base+ind];
                X[base+ind] = temp+X[base+ind+XOR];
                X[base+ind+XOR] = temp-X[base+ind+XOR];
                // Умножение на поворачивающие множители
                X[base+ind+XOR] = X[base+ind+XOR]*tw[ind*twSTEP];
            end
            base = base +2*XOR;
        end
        XOR=XOR/2; twSTEP = 2*twSTEP;
    end
    BrPermutation(X); // Реверс-перестановка элементов блока X
end

```

Алгоритм 8: Еще раз FFT с прореживанием по выходу (дихотомия по выходу)

```

//  $tw[n] = e^{-j \frac{2\pi n}{2^n}}$ ,  $n = [0 : 2^{(n-1)} - 1]$  - массив поворачивающих множителей
Function FFT(X,n) // X-блок данных для вычисления FFT размера  $2^n$ 
    BrPermutation(X); // Реверс-перестановка элементов блока X
    N=Length(X); n=log2(N);
    XOR=N/2; twSTEP = 1;
    for it = 1 to n do // Цикл по слоям
        base = 0;
        for ib = 1 to twSTEP do // Цикл по узлам
            for ind = 0 to XOR-1 do // Цикл по индексу внутри блока
                // Умножение на поворачивающие множители
                X[base+ind+XOR] = X[base+ind+XOR]*tw[ind*twSTEP];
                // Вычисление бабочек
                temp = X[base+ind];
                X[base+ind] = temp+X[base+ind+XOR];
                X[base+ind+XOR] = temp-X[base+ind+XOR];
            end
            base = base +2*XOR;
        end
        XOR=2*XOR; twSTEP = twSTEP/2;
    end
end

```

Алгоритм 9: FFT с прореживанием по выходу (дихотомия по выходу)

Двойственный алгоритм 9 с прореживанием по входу написать ничуть не сложнее. Нужно только учесть, что теперь все наоборот – вычисления выполняются как раз на этапе распространения блоков снизу вверх, а при распространении сверху вниз просто происходит разбиение блоков прореживанием. Достаточно переставить вызов реверс-перестановки в начало, и изменить порядок обхода слоев на снизу вверх.

Процедуры 8 и 9 оптимизированы по числу арифметических операций – поворачивающие множители предвычислены и хранятся в памяти. Если существенны ограничения на объем памяти, несложно организовать динамическое вычисление этих множителей, сэкономив память ценой дополнительных умножений.

Два нижние слоя дерева FFT отличает то, что нетривиальные умножения в их узлах фактически отсутствуют. Узлы самого нижнего слоя обрабатывают блоки длины 2 и умножают на поворачивающий множитель подблоки длины 1. Множитель этот неизменно равен единице. Бессмысленные умножения на единицу можно опустить. На предыдущем слое обрабатываются блоки длины 4 и умножаются на поворачивающие множители блоки длины 2. Множители это оказываются равными единице и $\xi^{-j\frac{\pi}{2}} = -j$. Умножение на единицу излишне, а умножение на $-j$ реализуется перестановкой мнимой и вещественной частей с заменой знака: $-j(x + jy) = y - jx$.

Воспользовавшись этим, можно сэкономить на умножениях на двух нижних слоях. К сожалению, это нарушает стройность структуры алгоритмов из трех вложенных циклов и требует дополнительны накладных затрат на управление вычислениями.

Наконец, для перехода к обратным ДПФ (2) достаточно изменить знак в показателях поворачивающих множителей – выполнить их комплексное сопряжение.

Принципы организации FFT на 2^n точках элементарно переносятся на случай $N = p^n$. Применяются две двойственные схемы рекурсивного разбиения боков для временной и частотной областей – разбиение на p одинаковых блоков (p -томия)

$$j^{(n)} = p^{n-1}j^{(1)} + j^{(n-1)}; \quad j^{(1)} \in Z_p, \quad j^{(n-1)} \in Z_{p^{n-1}}$$

и разбиение децимацией через p (p -прореживание)

$$j^{(n)} = j^{(1)} + pj^{(n-1)},$$

которые оказываются ассоциированными с реверсной и прямой кодировками скалярного индекса $j^{(n)}$ позиционным кодом по основанию p . Факторизации, аналогичные (3),(4), сводят БПФ на p^n точках к последовательному БРФ на p и p^{n-1} точках с промежуточным умножением на поворачивающие множители. БПФ же на p точках приобретает статус элементарной операции, которая заменяет собой бабочку.

4. Разложение на взаимно простые

Разобравшись с FFT на p^n точках, вернемся к БПФ на числе точек $N = \prod_{k=1}^K P_k$, являющимся произведением K взаимно простых множителей $P_k = p_k^{n_k}$, $n_k \geq 1$. На помощь здесь приходит уже упоминавшаяся китайская теорема об остатках.

В концентрированном виде, это теорема о том, что большое кольцо Z_N по составному модулю N факторизуется в декартово произведение K малых координатных колец Z_{P_k} : $Z_N = \prod_{k=1}^K Z_{P_k}$. Попросту это означает, что элементы j из кольца Z_N можно векторизовать – представить их блоками $(j^{(1)}, j^{(2)}, \dots, j^{(K)})$ элементов $j^{(k)}$ из координатных колец Z_{P_k} , причем так, что арифметические операции в Z_N сводятся к тем же операциям в каждом из координатных колец в отдельности. Часто об этом говорят как о реализации арифметики в Z_N арифметикой в остаточных классах – координатных колцах Z_{P_k} .

Все довольно просто. В кольце Z_N существует и единственное разложение единицы

$$1 = R_1 + R_2 + \cdots + R_K \quad (5)$$

в сумму ортогональных идемпотентов – чисел R_k со свойствами

$$R_k \bmod P_j = \delta_{k,j}; \quad k, j = [1 : K]. \quad (6)$$

где $\delta_{k,j}$ – символ Кронекера.

Идемпотенты R_k эффективно вычисляются. В качестве «нулевого приближения» к R_k возьмем число $Q_k = \frac{N}{P_k}$. Это произведение всех P_j , кроме одного. Оно уже равно нулю по всем модулям P_j , но пока еще может отличаться от единицы по модулю P_k : $Q_k \bmod P_k = a$. Осталось найти число b , такое что $ab = 1 \bmod P_k$ (это эквивалентно обращению a в кольце Z_{P_k}). Тогда $R_k = bQ_k$ и будет искомым идемпотентом.

К примеру, идемпотенты кольца Z_{15} ($15 = 3 \times 5$) – это числа $R_3 = 10$ и $R_5 = 6$.

Если свойства (6) выполнены, то любое из произведений $R_k R_j$, $j \neq k$ равно нулю по всем модулям P_k , то есть делится на все взаимно простые числа P_k . Следовательно, оно кратно N , то есть равно нулю в Z_N . Это означает, что R_k попарно ортогональны в Z_N : $R_k R_j = 0$ при $j \neq k$. Умножив теперь обе части разложения единицы (5) на любое из R_k с учетом ортогональности найдем: $R_k = R_k^2$. Это свойство идемпотентности (похожести на единицу) чисел R_k : $R_k^2 = R_k$. В компактной форме свойства ортогональности и идемпотентности выражаются тождеством,

$$R_k R_j = R_k \delta_{k,j}, \quad k, j = [1 : K], \quad (7)$$

которое эквивалентно модульным свойствам (6).

Умножение на R_k реализует проектирование элементов j кольца Z_N в координатные кольца Z_{P_k} с элементами

$$j^{(k)} = j \bmod P_k, \quad k = [1 : K]. \quad (8)$$

В самом деле,

$$R_k j = R_k (j^{(k)} + mP_k) = R_k j^{(k)} + mR_k P_k = R_k j^{(k)}$$

ввиду того, что произведение $R_k P_k$, кратное всем P_k , равно нулю в Z_N .

Умножив обе части разложения (5) на j , получим представление элемента $j \in Z_N$ через набор его проекций $j^{(k)}$ в координатные кольца Z_{P_k} :

$$j = R_1 j^{(1)} + R_2 j^{(2)} + \cdots + R_K j^{(K)} \quad (9)$$

Возвращаясь к ДПФ на N точках, разложим временнной и частный индексы по координатным кольцам

$$t = R_1 t^{(1)} + R_2 t^{(2)} + \cdots + R_K t^{(K)},$$

$$f = R_1 f^{(1)} + R_2 f^{(2)} + \cdots + R_K f^{(K)}.$$

С учетом свойств ортогональности-идемпотентности (7), для произведения ft это дает:

$$ft = R_1 f^{(1)} t^{(1)} + R_2 f^{(2)} t^{(2)} + \cdots + R_K f^{(K)} t^{(K)}.$$

Вот она – праведная факторизация. Никакого перепутывания. Умножение в большом кольце сведено к умножениям в координатных кольцах.

Для степени корня ξ в формуле ДПФ (1) получается

$$\xi^{ft} = (\xi^{R_1})^{f^{(1)}t^{(1)}} (\xi^{R_2})^{f^{(2)}t^{(2)}} \dots (\xi^{R_K})^{f^{(K)}t^{(K)}} = \xi_1^{f^{(1)}t^{(1)}} \xi_2^{f^{(2)}t^{(2)}} \dots \xi_K^{f^{(K)}t^{(K)}},$$

а сама эта формула принимает вид:

$$y_{(f^{(1)}, \dots, f^{(K)})} = \sum_{t^{(1)} \in Z_{P_1}, \dots, t^{(K)} \in Z_{P_K}} x_{(t^{(1)}, \dots, t^{(K)})} \xi_1^{f^{(1)}t^{(1)}} \dots \xi_K^{f^{(K)}t^{(K)}}.$$

Никаких поворачивающих множителей. Представляем N -блок X временных выборок K -мерным массивом с длиной P_k по k -ой координате и вычисляем ДПФ по всем его строкам, столбцам и так далее в произвольном порядке.

Переносить элементы блока X в отдельный многомерный массив не обязательно. Можно обращаться непосредственно к элементам X , каждый раз вычисляя скалярный индекс $t = R_1 t^{(1)} + R_2 t^{(2)} + \dots + R_K t^{(K)}$ по текущему набору координат векторного индекса $(t^{(1)}, t^{(2)}, \dots, t^{(K)})$. Разумеется, это потребует дополнительных накладных затрат на адресную арифметику.

Пусть сложность ДПФ на N точках составляет N^2 . По каждой данной координате k придется вычислить P_k -точечное ДПФ $Q_k = \frac{N}{P_k}$ раз. Сложность этих вычислений составит $P_k^2 Q_k = P_k N$. Вычисления эти придется повторить по всем координатам. В конечном итоге получится сложность $(P_1 + P_2 + \dots + P_K)N$, которая меньше N^2 в той мере, в которой сумма взаимно простых сомножителей P_k меньше их произведения.

5. Маленькие хитрости

Элементами j кольца Z_N на самом деле являются классы $\{j + mN\}$, $m \in Z$ эквивалентных целых чисел, отличающихся на любое кратное модуля N . Их называют смежными классами.

Степени ξ^{ft} не зависят от выбора конкретных представителей смежных классов $\{f + mN\}$, $\{t + kN\}$, поскольку $\xi^N = 1$. Если участвующие в ДПФ блоки выборок X, Y периодически продолжить вдоль осей времени и частоты, так чтобы выполнялись тождества $x_t = x_{t+kN}$, $y_f = y_{f+mN}$, то индексы f, t также приобретут статус произвольных представителей смежных классов. Тогда окажется, что традиционный диапазон $[0 : N - 1]$ значений индексов – это не более как один из вариантов выбора системы представителей. Вообще же возможно множество других вариантов их выбора.

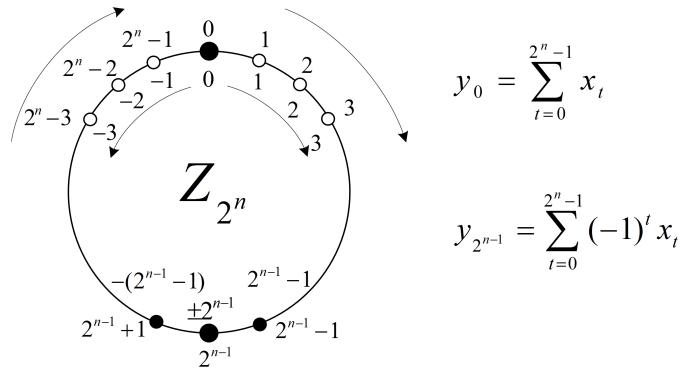


Рис. 10. Варианты выбора представителей Z_{2^n}

При анализе ДПФ-спектров удобна знаковая система представителей, включающая наряду с положительными числами f отрицательные числа $-f = N - f$. Представление о соотношении между этими двумя системами дает рис. 10.

Забавно, что в знаковой системе представителей точке 2^{n-1} отвечают как положительная частота $f = 2^{n-1}$, так и отрицательная частота $f = -2^{n-1}$. Это связано с тем, что спектр дискретизованного сигнала периодичен, а точки $\pm 2^{n-1}$ как раз совпадают с левой и правой границами периода – границами полосы Найквиста.

Для частот $f = 0$ и $f = 2^{n-1}$ формулы ДПФ выглядят особенно просто. Они приведены на рисунке. Выборка y_0 спектра на нулевой частоте – это просто сумма всех выборок сигнала, а выборка $y_{2^{n-1}}$ спектра на границе полосы Найквиста – это та же сумма, но с переменами знака.

Взглянув на определение ДПФ

$$y_f = \sum_{t \in Z_N} x_t \xi^{ft}; \quad f, t \in Z_N,$$

нетрудно заметить, что значение ξ^{ft} не изменяется при изменение знаков обоих f и t , а также при комплексном сопряжении и изменении знака одного из f или t . Отсюда вытекают следующие свойства инвариантности сигналов и спектров при отражениях – заменах $t \rightarrow -t$, $f \rightarrow -f$ и комплексных сопряжениях (*):

$$y_{-f} \leftrightarrow x_{-t}; \quad y_{-f}^* \leftrightarrow x_t^*; \quad y_f^* \leftrightarrow x_{-t}^*.$$

Отсюда следует, что вещественным сигналам с $x_t = x_t^*$ отвечают сопряженно-симметричные спектры с $y_f = y_{-f}^*$, а мнимым сигналам с $x_t = -x_t^*$ – сопряжено-антисимметричные спектры с $y_f = -y_{-f}^*$.

Этот факт лежит в основе распространенного приема параллельного вычисления спектров сразу двух вещественных сигналов a_t , b_t . Эти сигналы упаковывают в виртуальный комплексный сигнал $x_t = a_t + jb_t$ и вычисляют FFT y_f от него. Спектры a_f , b_f сигналов a_t , b_t выделяют как сопряженно симметричную и антисимметричную части y_f :

$$a_f = \frac{y_f + y_{-f}^*}{2}; \quad b_f = \frac{y_f - y_{-f}^*}{2}.$$

Одна из областей применения FFT – это вычисление циклических сверток сигналов a_t , b_t – сумм вида

$$x_t = \sum_{u \in Z_N} a_u b_{t-u}.$$

В основе метода лежит теорема о свертке, согласно которой спектр свертки есть произведение спектров сигналов. Доказывается это в одну строку:

$$y_f = \sum_{t \in Z_N} x_t \xi^{ft} = \sum_{t \in Z_N} \sum_{u \in Z_N} a_u b_{t-u} \xi^{ft} = \sum_{u \in Z_N} a_u \xi^{fu} \sum_{t \in Z_N} b_{t-u} \xi^{f(t-u)} = a_f b_f.$$

Для вычисления свертки вычисляют спектры сигналов, перемножают их поточечно и затем вычисляют обратное ДПФ.

При поточечном перемножении порядок расположения элементов внутри блоков безразличен. Поэтому на реверс-перестановках удается сэкономить. К сигналам применяют FFT с прореживанием по выходу – реверс-перестановка на выходе, а обратное преобразование выполняют прореживанием по входу – реверс-перестановка на входе. При последовательном выполнении прямого и обратного FFT реверс-перестановки взаимно аннигилируют – их можно исключить. Пример ситуации, когда существование двух двойственных вариантов FFT приносит ощутимую пользу.