Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

**Дискретное преобразование Фурье**

ОТЧЕТ ПО ДИСЦИПЛИНЕ

«ТЕОРЕТИКО-ЧИСЛОВЫЕ МЕТОДЫ В КРИПТОГРАФИИ»

Лабораторная работа №3

студентки 5 курса 531 группы

специальности 10.05.01 «Компьютерная безопасность»

факультета компьютерных наук и информационных технологий

Громовой Наталии Викторовны

|  |  |  |
| --- | --- | --- |
| Преподаватель  профессор | \_\_\_\_\_\_\_\_\_\_ | В.А. Молчанов |
|  | подпись, дата |  |

Саратов 2018

**Цель работы**: изучение свойств дискретного преобразования Фурье.

**Теоретическая часть**

Пусть имеется многочлен *n*-ой степени:

Не теряя общности, можно считать, что *n* является степенью 2. Если в действительности *n* не 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 , тогда известно, что  . Кроме того, один из этих корней  (называемый главным значением корня *n*-ой степени из единицы) таков, что все остальные корни являются его степенями: .

Тогда дискретным преобразованием Фурье (ДПФ) (discrete Fourier transform, DFT) многочлена *A(x)* называются значения этого многочлена в точках , т.е. это вектор:

Аналогично определяется и обратное дискретное преобразование Фурье (InverseDFT). Обратное ДПФ для вектора значений многочлена  – это вектор коэффициентов многочлена :

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.

**Быстрое преобразование Фурье** (fast Fourier transform) — это метод, позволяющий вычислять ДПФ за время *O*(*nlogn*). 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.

Итак, пусть имеется многочлен *A(x)* степени *n*, где *n* — степень двойки, и *n>*1:

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

Нетрудно убедиться, что:

Многочлены *A0* и *A1* имеют вдвое меньшую степень, чем многочлен *A*. Если мы сможем за линейное время по вычисленным DFT(*A0* ) и DFT(*A1)* вычислить DFT(*A*), то мы и получим искомый алгоритм быстрого преобразования Фурье (т.к. это стандартная схема алгоритма "разделяй и властвуй", и для неё известна асимптотическая оценка *O*(*nlogn*)).

Итак, пусть мы имеем вычисленные вектора  и  . Найдём выражения для .

Во-первых, вспоминая (1), мы сразу получаем значения для первой половины коэффициентов:

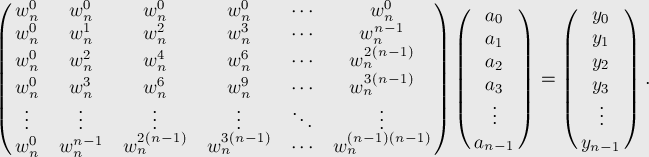
Для второй половины коэффициентов после преобразований также получаем простую формулу:

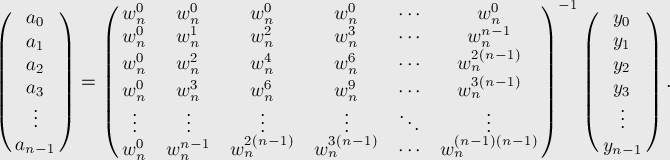
(Здесь мы воспользовались (1), а также тождествами   и  .)

Итак, в результате мы получили формулы для вычисления всего вектора {*yk*}:

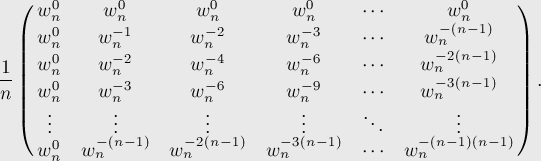
**Обратное БПФ**

Итак, пусть дан вектор  — значения многочлена A степени *n* в точках . Требуется восстановить коэффициенты  многочлена. Эта задача называется **интерполяцией**, для этой задачи есть и общие алгоритмы решения, однако в данном случае будет получен очень простой алгоритм (простой тем, что он практически не отличается от прямого БПФ).

ДПФ мы можем записать, согласно его определению, в матричном виде:

Тогда вектор  можно найти, умножив вектор  на обратную матрицу к матрице, стоящей слева (которая, кстати, называется матрицей Вандермонда):

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



Таким образом, получаем формулу:

 a_k = \frac{1}{n} \sum_{j=0}^{n-1} y_j w_n^{-kj}.[...]

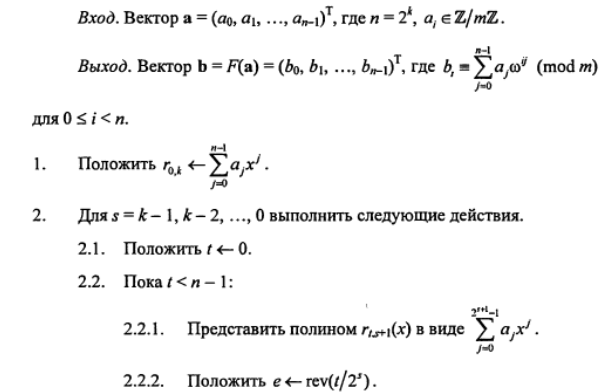
Сравнивая её с формулой для y_k:

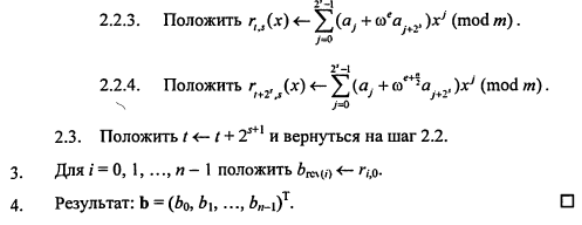
 y_k = \sum_{j=0}^{n-1} a_j w_n^{kj}, 

мы замечаем, что эти две задачи почти ничем не отличаются, поэтому коэффициенты *ak* можно находить таким же алгоритмом "разделяй и властвуй", как и прямое БПФ, только вместо  везде надо использовать , 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*(*nlogn)*

Алгоритм БПФ (аналогичный для обратного БПФ, но *w* меняем на *w-1(* mod *m),*а на шаге 3 полагаем .

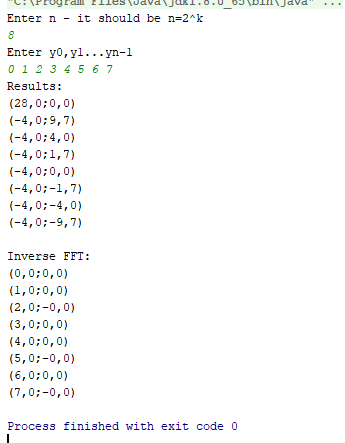




**Практическая часть**

Мною была реализована программа на языке Java, реализующая быстрое преобразование Фурье, а также обратное быстрое преобразование Фурье.

Пример работы программы:



**Программная реализация**

**class** Complex {  
 **public final double re**;  
 **public final double im**;  
  
 **public** Complex() {  
 **this**(0, 0);  
 }  
  
 **public** Complex(**double** r, **double** i) {  
 **re** = r;  
 **im** = i;  
 }  
 **public** Complex conj() {  
 **return new** Complex(**re**, -**im**);  
 }  
 **public** Complex mul(Complex b) {  
 **return new** Complex(**re** \* b.**re** - **im** \* b.**im**, **re** \* b.**im** + **im** \* b.**re**);  
 }  
 **public** Complex mul(**double** b) {  
 **return new** Complex(**re** \* b, **im** \* b);  
 }  
 **public** Complex div(Complex b) {  
 **return this**.mul(b.conj()).mul(1 / b.len2());  
 }  
 **public double** len2() {  
 **return re** \* **re** + **im** \* **im**;  
 }  
 **public** Complex add(Complex b) {  
 **return new** Complex(**this**.**re** + b.**re**, **this**.**im** + b.**im**);  
 }  
  
 **public** Complex sub(Complex b) {  
 **return new** Complex(**this**.**re** - b.**re**, **this**.**im** - b.**im**);  
 }  
  
 **public** Complex mult(Complex b) {  
 **return new** Complex(**this**.**re** \* b.**re** - **this**.**im** \* b.**im**,  
 **this**.**re** \* b.**im** + **this**.**im** \* b.**re**);  
 }  
  
 @Override  
 **public** String toString() {  
 **return** String.*format*(**"(%.1f;%.1f)"**, **re**, **im**);  
 }  
}

**import** java.util.Scanner;  
  
**import static** java.lang.Math.\*;  
  
**public class** FFT2 {  
  
  
 **public static int** bitReverse(**int** n, **int** bits) {  
 **int** reversedN = n;  
 **int** count = bits - 1;  
  
 n >>= 1;  
 **while** (n > 0) {  
 reversedN = (reversedN << 1) | (n & 1);  
 count--;  
 n >>= 1;  
 }  
  
 **return** ((reversedN << count) & ((1 << bits) - 1));  
 }  
  
 **static void** fft(Complex[] buffer,**int** inverse) {  
  
 **int** bits = (**int**) (*log*(buffer.**length**) / *log*(2));  
 **for** (**int** j = 1; j < buffer.**length** / 2; j++) {  
 **int** swapPos = *bitReverse*(j, bits);  
 Complex temp = buffer[j];  
 buffer[j] = buffer[swapPos];  
 buffer[swapPos] = temp;  
 }  
  
 **for** (**int** N = 2; N <= buffer.**length**; N <<= 1) {  
 **for** (**int** i = 0; i < buffer.**length**; i += N) {  
 **for** (**int** k = 0; k < N / 2; k++) {  
 **int** evenIndex = i + k;  
 **int** oddIndex = i + k + (N / 2);  
 Complex even = buffer[evenIndex];  
 Complex odd = buffer[oddIndex];  
  
 **double** term = inverse\* (-2 \* ***PI*** \* k) / (**double**) N;  
 Complex exp = (**new** Complex(*cos*(term), *sin*(term)).mult(odd));  
 buffer[evenIndex] = even.add(exp);  
 buffer[oddIndex] = even.sub(exp);  
 }  
 }  
 }  
 }  
 **public static void** main(String[] args) **throws** Exception {Scanner scanner=**new** Scanner(System.***in***);  
 System.***out***.println(**"Enter n - it should be n=2^k"**);  
 **int** n=scanner.nextInt();  
 **if**(n>0&&(n&(n-1))!=0){  
 **throw new** IllegalArgumentException(**"Wrong number"**);  
 }  
 System.***out***.println(**"Enter y0,y1...yn-1"**);  
 **int** input[]=**new int**[n];  
 **for** (**int** i = 0; i < n; i++) {  
 input[i]=scanner.nextInt();  
 }  
  
 Complex[] cinput = **new** Complex[input.**length**];  
 **for** (**int** i = 0; i < input.**length**; i++)  
 cinput[i] = **new** Complex(input[i], 0);  
  
 *fft*(cinput,1);  
  
 System.***out***.println(**"Results:"**);  
 **for** (Complex c : cinput) {  
 System.***out***.println(c);  
 }  
 System.***out***.println();  
 *fft*(cinput,-1);  
 System.***out***.println(**"Inverse FFT:"**);  
 **for** (Complex c : cinput) {  
 **if**(c.**re**/cinput.**length**<1&&c.**re**/cinput.**length**>0) {  
 System.***out***.println( String.*format*(**"(%.1f;%.1f)"**,0d,0d ));  
 **continue**;  
 }  
 System.***out***.println( String.*format*(**"(%.1f;%.1f)"**,c.**re**/cinput.**length**,c.**im**/cinput.**length** ));  
 }  
 }  
 }