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

Лабораторная работа №2

студента 5 курса 531 группы
специальности 10.05.01 «Компьютерная безопасность»
факультета компьютерных наук и информационных технологий
Енца Михаила Владимировича

Преподаватель профессор	___________ подпись, дата	В. А. Молчанов
Заведующий кафедрой д.ф.-м.н., доцент	___________ подпись, дата	М. Б. Абросимов

Саратов 2018

1. Постановка задачи.

Изучение свойств дискретного преобразования Фурье и программная реализация его приложений.

2. Теоретические сведения по рассмотренным темам с их обоснованием.

Дискретное преобразование Фурье.

Пусть имеется многочлен n-ой степени:

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

Не теряя общности, можно считать, что n является степенью 2. Если в действительности n не является степенью 2, то 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 $w_{n,k}, k = 0, \dots n-1$, 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: $w_{n,k} = (w_n)^k$.

Дискретным преобразованием Фурье (DFT) многочлена $A(x)$ называются значения этого многочлена в точках $x = w_{n,k}$, т.е. это вектор:

$$DFT(a_0, a_1, .., a_{n-1}) = (y_{0,}y_1, \dots, y_{n-1}) = \left(A(w_{n,0}), A(w_{n,1}), \dots, A(w_{n,n-1})\right)$$
$$= (A(w_n^0), A(w_n^1), .. A(w_n^{n-1})$$

Аналогично определяется и *обратное дискретное преобразование Фурье* (InverseDFT). Обратное ДПФ для вектора значений многочлена $(y_{0,}y_1, \dots, y_{n-1})$ – это вектор коэффициентов многочлена $(a_0, a_1, .., a_{n-1})$:

$$InverseDFT(y_{0,}y_1, \dots, y_{n-1}) = (a_0, a_1, .., a_{n-1})$$

Быстрое преобразование Фурье.

Быстрое преобразование Фурье (БПФ или FFT) — это метод, позволяющий вычислять ДПФ за время $O(nlogn)$. Этот метод основывается на свойствах комплексных корней из единицы.

Основная идея БПФ заключается в разделении вектора коэффициентов на два вектора, рекурсивном вычислении ДПФ для них, и объединении результатов в одно БПФ.

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

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

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

$$A_0(x) = a_0x^0 + a_2x^1 + \cdots + a_{n-2}x^{n/2-1}$$
$$A_1(x) = a_1x^0 + a_3x^1 + \cdots + a_{n-1}x^{n/2-1}$$

Нетрудно убедиться, что:

$$A(x) = A_0(x^2) + xA_1(x^2) \qquad (1)$$

Многочлены A_0 и A_1 имеют вдвое меньшую степень, чем многочлен A. Если мы сможем, за линейное время, по вычисленным $DFT(A_0)$ и $DFT(A_1)$, вычислить $DFT(A)$, то мы и получим искомый алгоритм быстрого преобразования Фурье.

Пусть имеем вычисленные вектора $\{y_k^0\}_{k=0}^{\frac{n}{2}-1} = DFT(A_0)$ и $\{y_k^1\}_{k=0}^{\frac{n}{2}-1} = DFT(A_1)$. Найдём выражения для $\{y_k\}_{k=0}^{n-1} = DFT(A)$.

Сразу получаем значения для первой половины коэффициентов:

$$y_k = y_k^0 + w_n^k y_k^1, \quad k = 0 \ldots \frac{n}{2} - 1.$$

Для второй половины коэффициентов после преобразований также получаем простую формулу:

$$y_{k+n/2} = A\left(w_n^{k+\frac{n}{2}}\right) = A_0(w_n^{2k+n}) + w_n^{k+\frac{n}{2}} A_1(w_n^{2k+n})$$

$$= A_0(w_n^{2k} w_n^n) + w_n^k w_n^{n/2} A_1(w_n^{2k} w_n^n) = A_0(w_n^{2k}) - w_n^k A_1(w_n^{2k})$$

$$= y_k^0 - w_n^k y_k^1$$

(Здесь воспользовались (1), а также тождествами $w_n^n = 1$ и $w_n^{n/2} = -1$.)

В результате получаем формулы для вычисления всего вектора $\{y_k\}$:

$$y_k = y_k^0 + w_n^k y_k^1, \quad k = 0 \dots \frac{n}{2} - 1.$$

$$y_{k+n/2} = y_k^0 - w_n^k y_k^1, \quad k = 0 \dots \frac{n}{2} - 1.$$

Обратное быстрое преобразование Фурье.

Пусть дан вектор $(y_0, y_1, \dots, y_{n-1})$ — значения многочлена A степени n в точках $x = w_n^k$. Требуется восстановить коэффициенты $(a_0, a_1, \dots, a_{n-1})$ многочлена. Эта задача называется *интерполяцией*, для этой задачи есть и общие алгоритмы решения, однако в данном случае будет получен простой алгоритм (простой тем, что он практически не отличается от прямого БПФ).

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

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 называется матрицей Вандермонда):

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

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

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

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:

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

Можно заметить, что эти две задачи почти ничем не отличаются, поэтому коэффициенты a_k можно находить таким же алгоритмом "разделяй и властвуй", как и прямое БПФ, только вместо w_n^k везде надо использовать w_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 $O(nlogn)$.

Алгоритм быстрого преобразования Фурье.

Вход. Вектор $a = (a_0, a_1, \dots, a_{n-1})^T$, где $n = 2^k, a_i \in Z/mZ$.

Выход. Вектор $b = F(a) = (b_0, b_1, \dots, b_{n-1})^T$, где $b_i \equiv \sum_{j=0}^{n-1} a_j w^{ij} (mod\ m)$. Для $0 \le i < n$.

1. Положить $r_{0,k} = \sum_{j=0}^{n-1} a_j x^j$

2. Для $s = k-1, k-2, \dots, 0$ выполнить следующие действия.

2.1. Положить $t = 0$.

2.2. Пока $t < n-1$:

2.2.1. Представить полином $r_{t,s-1}(x)$ в виде $\sum_{j=0}^{2^{s+1}-1} a_j x^j$

2.2.2. Положить $e = rev(t/2^s)$

2.2.3. Положить $r_{t,s}(x) = \sum_{j=0}^{2^s-1}(a_j + w^e a_{j+2^s})x^j mod(m)$.

2.2.4. Положить $r_{t+2^s,s}(x) = \sum_{j=0}^{2^s-1}\left(a_j + w^{e+\frac{n}{2}} a_{j+2^s}\right)x^j mod(m)$.

2.3. Положить $t = t + 2^{s+1}$ и вернуться на шаг 2.2.

3. Для $i = 0,1, \dots, n-1$ положить $b_{rev(i)} = r_{i,0}$

4. Результат: $b = (b_0, b_1, \dots, b_{n-1})^T$

Коды программ, реализующей рассмотренные алгоритмы

Программа реализована на языке Python (версия интерпретатора 3.6).

```
def fft(a, m, n, k):
    w = primitive_root(m, n)
    r = {(0, k): a}
    for s in range(k - 1, -1, -1):
        t = 0
        while t < n - 1:
            a_new = [r[(t, s + 1)][j] for j in range(pow(2, s + 1))]
            e = int(bin(t // pow(2, s))[2:].zfill(k)[::-1], 2)
            r[(t, s)] = [(a_new[j] + pow(w, e) * a_new[j + pow(2, s)]) %
m for j in range(pow(2, s))]
            r[(t + pow(2, s), s)] = [(a_new[j] + pow(w, e + n // 2) *
a_new[j + pow(2, s)]) % m for j in
                                         range(pow(2, s))]
            t += pow(2, s + 1)
    b = [0] * n
    for i in range(n):
        b[int(bin(i)[2:].zfill(k)[::-1], 2)] = r[(i, 0)][0]
    print('Вектор b = {}'.format(str(b)))
    return b


def ifft(b, m, n, k):
    w = utils.inverse(primitive_root(m, n), m) % m
    r = {(0, k): b}
    for s in range(k - 1, -1, -1):
        t = 0
        while t < n - 1:
            b_new = [r[(t, s + 1)][j] for j in range(pow(2, s + 1))]
```

```
                e = int(bin(t // pow(2, s))[2:].zfill(k)[::-1], 2)
                r[(t, s)] = [(b_new[j] + pow(w, e) * b_new[j + pow(2, s)]) %
m for j in range(pow(2, s))]
                r[(t + pow(2, s), s)] = [(b_new[j] + pow(w, e + n // 2) *
b_new[j + pow(2, s)]) % m for j in
                                            range(pow(2, s))]
                t += pow(2, s + 1)
        a = [0] * n
        for i in range(n):
            # print(int(bin(i)[2:].zfill(k)[::-1], 2))
            # print(utils.inverse(m, n % m))
            a[int(bin(i)[2:].zfill(k)[::-1], 2)] = (utils.inverse(n % m, m)
* r[(i, 0)][0]) % m
        print('Вектор a = {}'.format(str(a)))
```

Оценки сложности рассмотренных алгоритмов

Сложность алгоритма «быстрое преобразование Фурье» и «быстрое обратное преобразование Фурье» равна $O(n \log n)$.

Результаты тестирования програмы

Пример выполнения FFT:

Модуль $m = 65537$.

Вектор $a = [7, 6, 5, 4, 3, 2, 1, 0]$

Вектор $b = [28, 17476, 1028, 15428, 4, 50117, 64517, 48069]$

Пример выполнения IFFT:

Модуль $m = 65537$.

Вектор $b = [28, 17476, 1028, 15428, 4, 50117, 64517, 48069]$

Вектор $a = [7, 6, 5, 4, 3, 2, 1, 0]$