FAST FOURIER TRANSFORM

Prof. Zheng Zhang

Harbin Institute of Technology, Shenzhen

Fast Fourier Transform

- The straightforward method of **adding two polynomials** of degree n takes $\Theta(n)$ time, while the straightforward method of **multiplying them** takes $\Theta(n^2)$ time.
- We shall show how the fast Fourier transform, or **FFT**, can reduce the time of multiply polynomials to $\Theta(n \lg n)$ time.
- The most common use for Fourier transforms, and hence the FFT, is in signal processing.
- A signal is given in **the time domain**: as a function mapping time to amplitude.

Fast Fourier Transform

- Fourier analysis allows us to express the signal as a **weighted sum of phase-shifted sinusoids of varying frequencies**. The **weights** and **phases** associated with the frequencies characterize the signal in the frequency domain.
- Among the many everyday applications of FFT's are **compression techniques** used to encode digital video and audio information, including MP3 files.
- How could we use FFT to reduce the time of multiply polynomials to $\Theta(n \lg n)$ time?

POLYNOMIALS

- A polynomial in the variable ***x*** over an **algebraic field** *F* is a representation of a function $A(x)$ as a formal sum:

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

where, $a_0, a_1, \ldots, a_{n-1}$ are the coefficients of the polynomial.

- **Degree** of $A(x)$ is k if the highest nonzero coefficient is a_k . We can denote it as: **degree(*A*) = *k***.
- Obviously, n is the degree bound of $A(x)$.

POLYNOMIAL OPERATIONS

$$A(x) = \sum_{j=0}^{n-1} a_j x^j \quad \text{and} \quad B(x) = \sum_{j=0}^{n-1} b_j x^j$$

If $C(x) = A(x) + B(x)$, then $c_j = a_j + b_j$

If $C(x) = A(x)B(x)$, then $c_j = \sum_{k=0}^{j} a_k b_{j-k}$

Polynomial Operations

$$A(x) = \sum_{j=0}^{n-1} a_j x^j \quad B(x) = \sum_{j=0}^{n-1} b_j x^j \qquad C(x) = A(x)B(x)$$

- For polynomial multiplication, if ***A***(x) and ***B***(x) are polynomials of degree-bound ***n***, their product ***C***(x) is a polynomial of degree-bound 2n-1 such that ***C***(x) for all x in the underlying field.
- You probably have multiplied polynomials before, by multiplying each term in ***A***(x) by each term in ***B***(x) and then combining terms with equal powers.

POLYNOMIAL OPERATIONS

$$A(x) = \sum_{j=0}^{n-1} a_j x^j \quad B(x) = \sum_{j=0}^{n-1} b_j x^j \qquad C(x) = A(x)B(x)$$

- $\boldsymbol{A}(x) = 6x^3 + 7x^2 + 10x + 9$
- $\boldsymbol{B}(x) = -2x^3 + 4x - 5$

$$C(x) = \sum_{j=0}^{2n-2} c_j x^j$$

$$c_j = \sum_{k=0}^{j} a_k b_{j-k}$$

$$\begin{array}{rrrrrrr}
 & & & 6x^3 & +\ 7x^2 & -\ 10x & +\ 9 \\
 & & & -\ 2x^3 & & +\ 4x & -\ 5 \\
\hline
 & & & -\ 30x^3 & -\ 35x^2 & +\ 50x & -\ 45 \\
 & & 24x^4 & +\ 28x^3 & -\ 40x^2 & +\ 36x & \\
-\ 12x^6 & -\ 14x^5 & +\ 20x^4 & -\ 18x^3 & & & \\
\hline
-\ 12x^6 & -\ 14x^5 & +\ 44x^4 & -\ 20x^3 & -\ 75x^2 & +\ 86x & -\ 45
\end{array}$$

Representing Polynomials

- Two ways to represent polynomials:
  - The coefficient representation
  - The point-value representation.
- The coefficient and point-value representations of polynomials are **in a sense equivalent**.
- That is, a polynomial in point-value form has a **unique counterpart** in coefficient form.
- We will show how to combine them so that we can multiply **two degree-bound *n* polynomials in** $\Theta(n \lg n)$ time.

Coefficient Representation of Polynomial

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

Coefficient vector $a = (a_0, a_1, \ldots, a_{n-1})$

The operation of evaluating $A(x)$ at a given point x_0 with **Horner's rule**:

$$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \ldots + x_0(a_{n-2} + x_0(a_{n-1}))\ldots))$$

The complexity of Polynomial addition: $\Theta(n)$

The complexity of Polynomial multiplication: $\Theta(n^2)$

Point-value Representation of Polynomial

Point-value representation of Polynomial $A(x)$ is,

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$$ ***point-value pairs***

such that all of the x_k are distinct and a

$$y_k = A(x_k)$$

An Interpolating Polynomial

- The inverse of evaluation—determining the coefficient form of a polynomial from a point-value representation—is **interpolation**.
- The following theorem shows that **interpolation** is **well defined** when the desired interpolating polynomial must have a degree-bound **equal** to the given number of point-value pairs.

Theorem 30.1 Uniqueness of an Interpolating Polynomial

For any set $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n\text{-}1}, y_{n\text{-}1})\}$ of n point-value pairs such that all the x_k value are distinct, there is a **unique** polynomial $\boldsymbol{A}(x)$ of **degree-bound *n*** such that $y_k = \boldsymbol{A}(x_k)$ for $k = 0, 1, \ldots, n\text{-}1$.

Proof:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

Uniqueness of an Interpolating Polynomial

The matrix on the left id denoted as $V(x_0, x_1, \ldots, x_{n\text{-}1})$, and is known as a Vandermonde matrix, of which the determinant is

$$\prod_{0 \le j < k \le n-1} (x_k - x_j)$$

The matrix is invertible if the x_k are distinct.

Thus, the **coefficients** can be solved for **uniquely** given the point-value representation.

$$a = V(x_0, x_1, \ldots, x_{n-1})^{-1} y$$

Addition of Point-value Representation

$$C(x) = A(x) + B(x)$$

$\boldsymbol{A}(x) = \{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$

$\boldsymbol{B}(x) = \{(x_0, y'_0), (x_1, y'_1), \ldots, (x_{n-1}, y'_{n-1})\}$

$\boldsymbol{C}(x) = \{(x_0, y_0 + y'_0), (x_1, y_1 + y'_1), \ldots, (x_{n-1}, y_{n-1} + y'_{n-1})\}$

The complexity of Polynomial addition: $\Theta(n)$

Multiplication of point-value representation

$$C(x) = A(x)B(x)$$

Since the degree-bound of ***C*** is $2n$, we need $2n$ point-value pairs for a point-value presentation of ***C***.

Extend point-value of ***A*** and ***B*** to

$A(x) = \{(x_0, y_0), (x_1, y_1), \ldots, (x_{2n-1}, y_{2n-1})\}$

$B(x) = \{(x_0, y'_0), (x_1, y'_1), \ldots, (x_{2n-1}, y'_{2n-1})\}$

The a point-value representation of ***C*** is

$C(x) = \{(x_0, y_0\, y'_0), (x_1, y_1\, y'_1), \ldots, (x_{2n-1}, y_{2n-1}\, y'_{2n-1})\}$

The complexity of Polynomial multiplication: $\Theta(n)$

Fast multiplication of polynomials in coefficient form

- Can we use the linear-time multiplication method for polynomials in **point-value form** to **expedite** polynomial multiplication in coefficient form?
- The answer hinges on whether we can **convert** a polynomial quickly from **coefficient** form to **point-value** form (evaluate) and vice versa (interpolate).
- We can **use any points** we want as evaluation points, but by choosing the evaluation points carefully, we can **convert** between representations in only $\Theta(n \lg n)$ time.

Fast multiplication of polynomials in coefficient form

- We shall see if we choose "**complex roots of unity**" as the evaluation points, we can produce a point-value representation by taking the discrete Fourier transform (or DFT) of a coefficient vector.
- We can perform the **inverse operation**, interpolation, by taking the "**inverse DFT**" of point-value pairs, yielding a coefficient vector.
- We will show how the FFT accomplishes the DFT and inverse DFT operations in $\Theta(n \lg n)$ time.
- We **assume** that n is a power of 2; we can always meet this requirement by adding **high-order** zero coefficients.

Graphical outline of efficient polynomial multiplication

Coefficient Representation

$$a_0, a_1, \ldots, a_{n-1}$$
$$b_0, b_1, \ldots, b_{n-1}$$

Ordinary multiplication $\longrightarrow$

Time $\Theta(n^2)$

$$c_0, c_1, \ldots, c_{2n-2}$$

Evaluation

Time $\Theta(n \lg n)$

Interpolation

Time $\Theta(n \lg n)$

$$A(w_{2n}^0), B(w_{2n}^0)$$
$$A(w_{2n}^1), B(w_{2n}^1)$$
$$\vdots$$
$$A(w_{2n}^{2n-1}), B(w_{2n}^{2n-1})$$

Pointwise multiplication $\longrightarrow$

Time $\Theta(n)$

$$C(w_{2n}^0)$$
$$C(w_{2n}^1)$$
$$\vdots$$
$$C(w_{2n}^{2n-1})$$

Point-value Representation

Fast multiplication of polynomials in coefficient form

- **Double degree-bound**: Create **coefficient representations** of $A(x)$ and $B(x)$ as degree-bound $2n$ polynomials by adding n high-order zero coefficients to each.
- **Evaluate**: Compute **point-value representations** of $A(x)$ and $B(x)$ of length $2n$ by applying the FFT of order $2n$ on each polynomial. These representations contain the values of the two polynomials at the $(2n)$th roots of unity.
- **Pointwise multiply**: Compute a point-value representation for the polynomial $C(x)=A(x)B(x)$ by multiplying these values together pointwise. This representation ***contains*** the value of $C(x)$ at each (2n)th root of unity.
- **Interpolate**: Create the coefficient representation of the polynomial $C(x)$ by applying the FFT on $2n$ point-value pairs to compute the inverse DFT.

Fast multiplication of polynomials in coefficient form

- Steps (1) and (3) take time $\Theta(n)$, and steps (2) and (4) take time $\Theta(n \lg n)$. Thus, once we show how to use the FFT, we will have proven the following.

Theorem 30.2

We can multiply two polynomials of degree-bound n in time $\Theta(n \lg n)$ with both the input and output representations in coefficient form.

Complex roots of unity

- A **complex *n*-th** root of unity is a complex number w such that $w^n = 1$.
- There are exactly n complex n-th roots of unity:

$$e^{2\pi ik/n} \qquad \text{for } k = 0, 1, \ldots, n\text{-}1$$

- **Lemma 30.3** (Cancellation lemma)

For any integers $n \geq 0$, $k \geq 0$, an $d > 0$, we have $w_{dn}^{dk} = w_n^k$.

- **Corollary**: For any even integer $n > 0$, we have $w_n^{n/2} = w_2 = -1$.
- **Lemma 30.5** (Halving lemma)

If $n > 0$ is even, then the squares of the n complex n-th roots of unity are the $n/2$ complex $n/2$-th roots of unity.

- **Lemma 30.6** (Summation lemma)

For any integer $n \geq 1$ and nonzero integer k not divisible by n, we have

$$\sum_{j=0}^{n-1} (w_n^k)^j = 0$$

DFT

Evaluating a polynomial, $A(x) = \sum_{j=0}^{n-1} a_j x^j$

of degree bound n at $w_n^0, w_n^1, w_n^2, \ldots, w_n^{n-1}$

$$y_k = A(w_n^k) = \sum_{j=0}^{n-1} a_j w_n^{kj} \qquad k = 0,1,\ldots,n-1$$

The vector $y = (y_0, y_1, \ldots, y_{n-1})$ is the discrete Fourier transform of the coefficient vector $a = (a_0, a_1, \ldots, a_{n-1})$.

Framework of FFT

***n* is an exact power of 2.**

The FFT method employs a **divide-and-conquer strategy**.

$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$$

where, $A^{[0]}(x) = a_0 + a_2 x + a_4 x^2 + \ldots + a_{n-2} x^{n/2-1}$

$A^{[1]}(x) = a_1 + a_3 x + a_5 x^2 + \ldots + a_{n-1} x^{n/2-1}$

So that, the problem of evaluating ***A***(x) at $w_n^0, w_n^1, \ldots, w_n^{n-1}$

reduces to

(1) Evaluating the degree-bound n/2 polynomial $A^{[0]}(x)$ and $A^{[1]}(x)$ at $(w_n^0)^2, (w_n^1)^2, \ldots, (w_n^{n-1})^2$. **(Halving lemma)**

Framework of FFT

(2) Combing the results. $A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$

According to halving lemma, the polynomials $A^{[0]}$ and $A^{[1]}$ of degree-bound $n/2$ are recursively evaluated at the $n/2$ complex $(n/2)$th roots of unity.

The recurrence for the running time is,

$$T(n) = 2\,T(n/2) + \Theta(n) = \Theta(n \lg n)$$

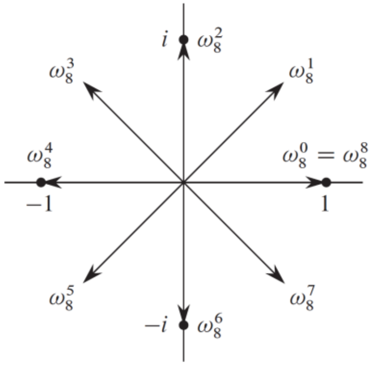
1-D Discrete Fourier Transform

$$X(j)=\sum_{k=0}^{N-1}A(k)\bullet W_N^{jk} \quad (j = 0, 1, \ldots, N\text{-}1) \quad (1)$$

$$A(k)=\frac{1}{N}\sum_{j=0}^{N-1}X(j)\bullet W_N^{-jk} \quad (k = 0, 1, \ldots, N\text{-}1) \quad (2)$$

where, $W_N = \boldsymbol{e}^{\mathbf{2\pi i/N}}$

$e^{ui} = \cos(u) + i * \sin(u)$



Eq. (1) can be rewritten in matrix form as

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ \vdots \\ X(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & W_N^{1\bullet 1} & W_N^{1\bullet 2} & \cdots & W_N^{1\bullet (N-1)} \\ 1 & W_N^{2\bullet 1} & W_N^{2\bullet 2} & \cdots & W_N^{2\bullet (N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{N-1} & W_N^{(N-1)\bullet 2} & \cdots & W_N^{(N-1)\bullet (N-1)} \end{bmatrix} \begin{bmatrix} A(0) \\ A(1) \\ A(2) \\ \vdots \\ A(N-1) \end{bmatrix} \qquad (3)$$

Eq. (3) can be simply denoted as

$$X = F_N A \, .$$

Examples

$$F_1 = [1] \qquad F_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad F_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}$$

Please note that,

(a) $W^0 = 1, W^{N/2} = -1$

(b) $W^{N+r} = W^r, W^{N/2+r} = -W^r$ $\mathbf{W^{N/2} * W^r = -W^r}$

$\mathbf{W_N = e^{2\pi i/N}}$

Idea of Fast Fourier Transform (FFT)

By exchanging the 2nd and 3rd column of F_4, we have

$$F_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} \qquad F_4 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & i & -i \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -i & i \end{bmatrix}$$

Denote,

$$\Pi_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \Omega_2 = \begin{bmatrix} 1 & \\ & i \end{bmatrix} \qquad F_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Idea of Fast Fourier Transform (FFT) (con't)

Then, we have

$$F_4 \prod_4 = \begin{bmatrix} F_2 & \Omega_2 F_2 \\ F_2 & -\Omega_2 F_2 \end{bmatrix} \qquad (4)$$

Thus, $\boxed{X = F_N A}$

$$F_4 \begin{bmatrix} A(0) \\ A(1) \\ A(2) \\ A(3) \end{bmatrix} = \begin{bmatrix} F_2 & \Omega_2 F_2 \\ F_2 & -\Omega_2 F_2 \end{bmatrix} \begin{bmatrix} A(0) \\ A(2) \\ A(1) \\ A(3) \end{bmatrix} = \begin{bmatrix} I & \Omega_2 \\ I & -\Omega_2 \end{bmatrix} \begin{bmatrix} F_2 \begin{bmatrix} A(0) \\ A(2) \end{bmatrix} \\ F_2 \begin{bmatrix} A(1) \\ A(3) \end{bmatrix} \end{bmatrix} \qquad (5)$$

Idea of Fast Fourier Transform (FFT) (con't)

Similarly, if $N = 2M$,

$$F_N \prod_N = \begin{bmatrix} F_M & \Omega_M F_M \\ F_M & -\Omega_M F_M \end{bmatrix} \quad (6)$$

Here, $\Omega_M = diag(1, W, \cdots, W^{M-1})$

So,

$$F_N A = \begin{bmatrix} I & \Omega_M \\ I & \Omega_M \end{bmatrix} \begin{bmatrix} F_M A_1 \\ F_M A_2 \end{bmatrix} \quad (7)$$

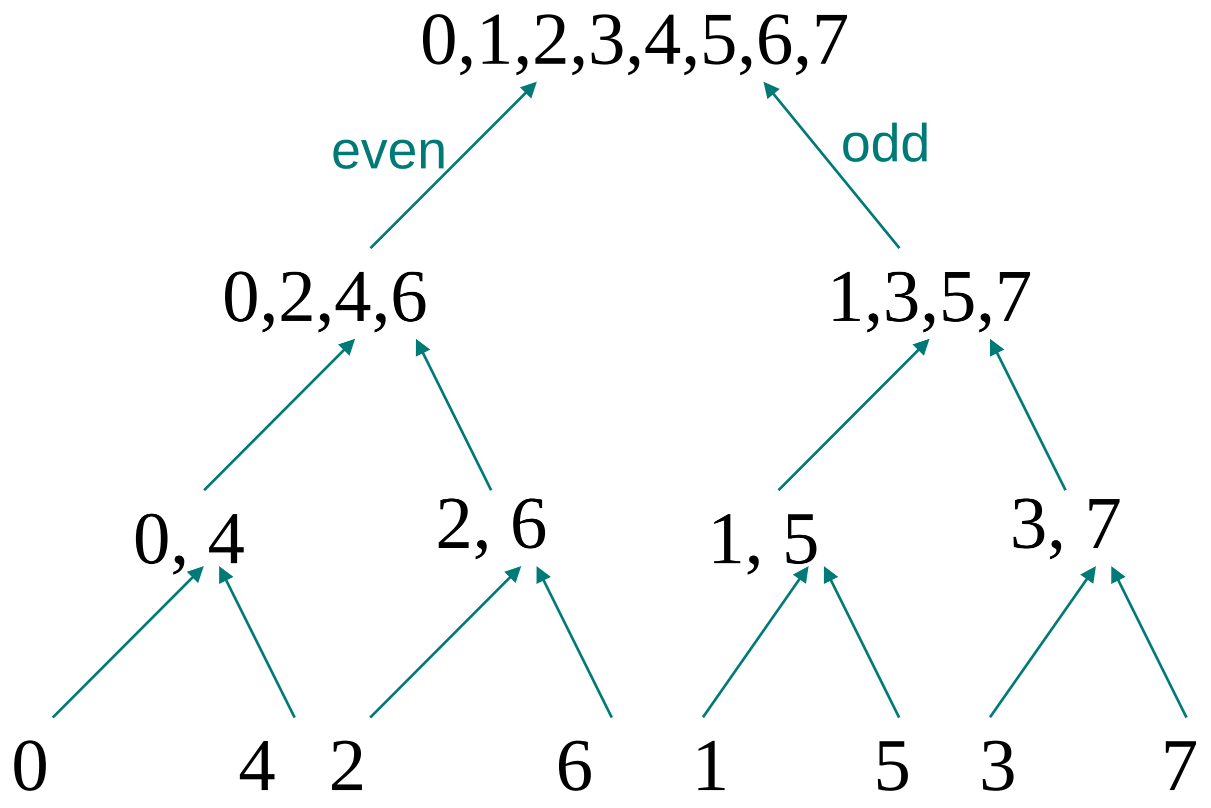
Here, $A_1 = [A(0), A(2), \cdots, A(N-2)]^T$

and $A_2 = [A(1), A(3), \cdots, A(N-1)]^T$

Idea of Fast Fourier Transform (FFT) (con't)

One example

If $N = 8$



Implementation of FFT

$$X(k) = \sum_{r=0}^{N-1} A(r) \bullet W_N^{rk}$$

Set $N = 2^m$

$$X(k) = \sum_{r=0}^{N/2-1} x(2r) W_N^{2rk} + \sum_{r=0}^{N/2-1} x(2r+1) W_N^{(2r+1)k}$$

$$= \sum_{r=0}^{N/2-1} x(2r) W_{N/2}^{rk} + W_N^k \sum_{r=0}^{N/2-1} x(2r+1) W_{N/2}^{rk} \qquad (8)$$

Here, $W_{N/2} = e^{\frac{2\pi i}{N/2}}$ and $r = 0, 1, \cdots, \frac{N}{2} - 1$

Implementation of FFT (con’t)

Set

$$A(k) = \sum_{r=0}^{N/2-1} x(2r) W_{N/2}^{rk} \qquad k = 0,1,\cdots,\tfrac{N}{2}-1$$

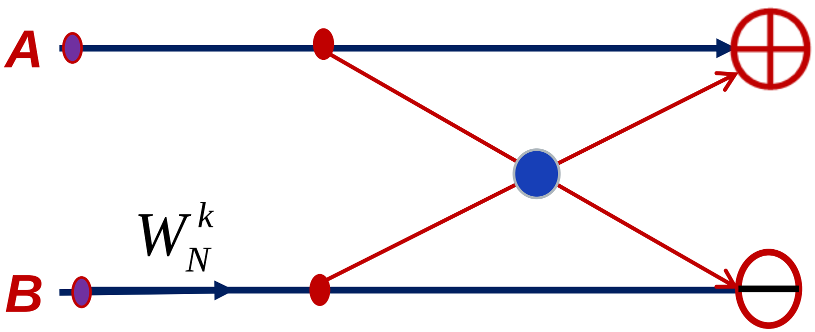
$$B(k) = \sum_{r=0}^{N/2-1} x(2r+1) W_{N/2}^{rk} \tag{9}$$

Then,

$$X(k) = A(k) + W_N^k B(k) \qquad X(k+N/2) = A(k) - W_N^k B(k) \tag{10}$$

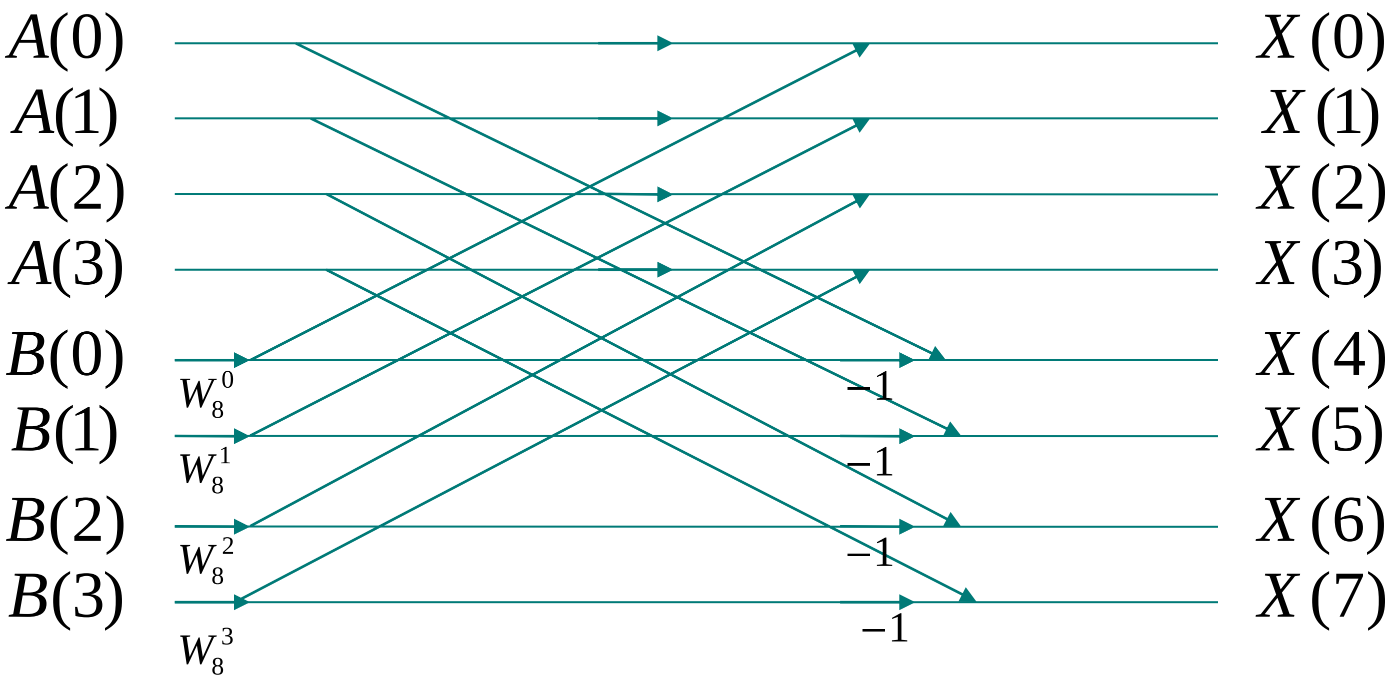
$X(k)$: DFT with N points; $A(k), B(k)$: DFT with N/2 points.

Implementation of FFT (con't)



$$X(k) = A(k) + W_N^k B(k)$$

$$X(k + N/2) = A(k) - W_N^k B(k)$$



Implementation of FFT (con't)

$$A(k) = \sum_{l=0}^{N/4-1} x(4l) W_{N/2}^{2lk} + \sum_{l=0}^{N/4-1} x(4l+2) W_{N/2}^{(2l+1)k}$$

$$= \sum_{l=0}^{N/4-1} x(4l) W_{N/4}^{lk} + W_{N/2}^{k} \sum_{l=0}^{N/4-1} x(4l+3) W_{N/4}^{lk}$$

Here, $r = 2l$ and $l = 0, 1, \cdots, N/4 - 1$

Implementation of FFT (con't)

Set

$$C(k) = \sum_{l=0}^{N/4-1} x(4l) W_{N/4}^{lk} \qquad k = 0,1,\cdots,N/4-1$$

$$D(k) = \sum_{l=0}^{N/4-1} x(4l+2) W_{N/2}^{lk}$$

Then

$$A(k) = C(k) + W_{N/2}^{k} D(k)$$

$$A(k+\tfrac{N}{4}) = C(k) - W_{N/2}^{k} D(k)$$

Implementation of FFT (con't)

Similarly, set

$$E(k)=\sum_{l=0}^{N/4-1}x(4l+1)W_{N/4}^{lk} \qquad k=0,1,\cdots,N/4-1$$

$$F(k)=\sum_{l=0}^{N/4-1}x(4l+3)W_{N/2}^{lk}$$

Then

$$B(k)=E(k)+W_{N/2}^{k}F(k)$$

$$B(k+\tfrac{N}{4})=E(k)-W_{N/2}^{k}F(k)$$

Implementation of FFT (con't)

If N=8, then $C(k)$ $D(k)$ $F(k)$ $E(k)$

all are DFT with 2 points.

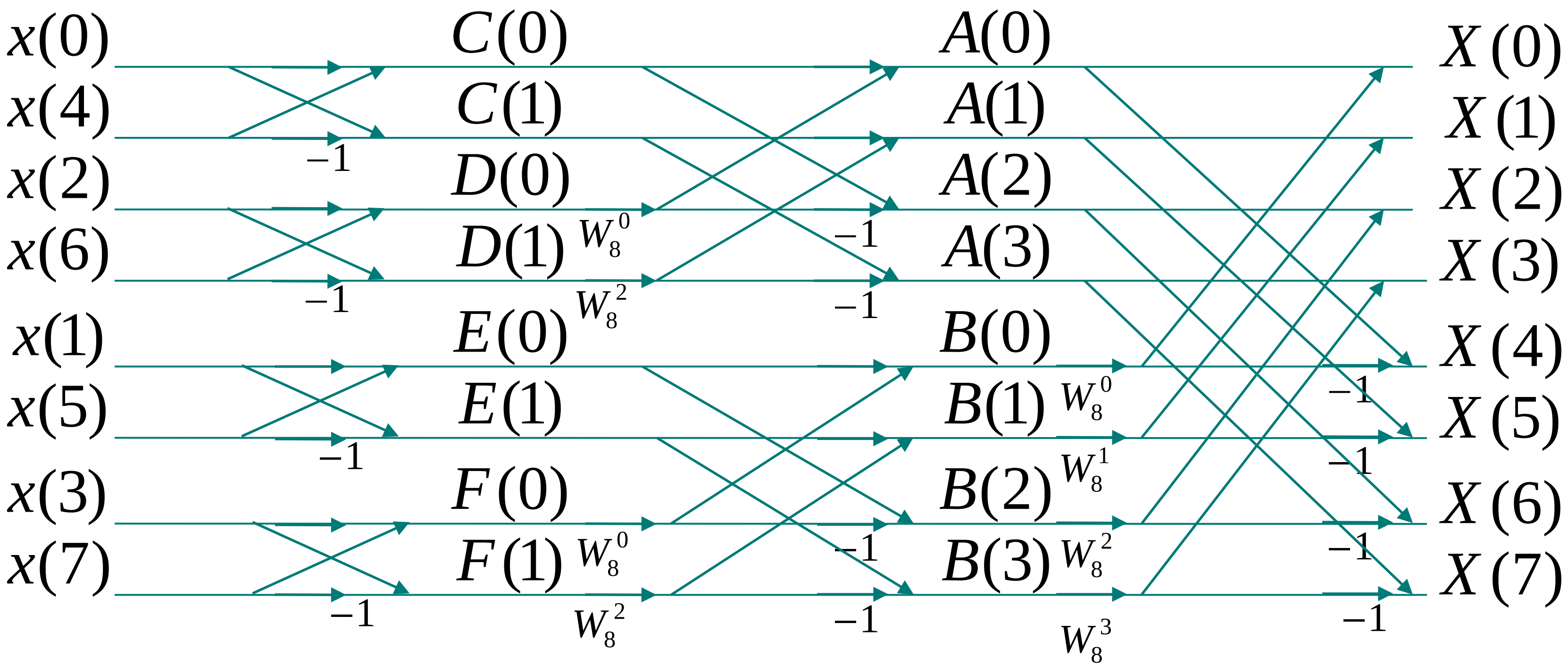
$$C(0) = x(0) + x(4) \qquad E(0) = x(1) + x(5)$$

$$C(1) = x(0) - x(4) \qquad E(1) = x(1) - x(5)$$

$$D(0) = x(2) + x(6) \qquad F(0) = x(3) + x(7)$$

$$D(1) = x(2) - x(6) \qquad F(1) = x(3) - x(7)$$

Implementation of FFT (con't)



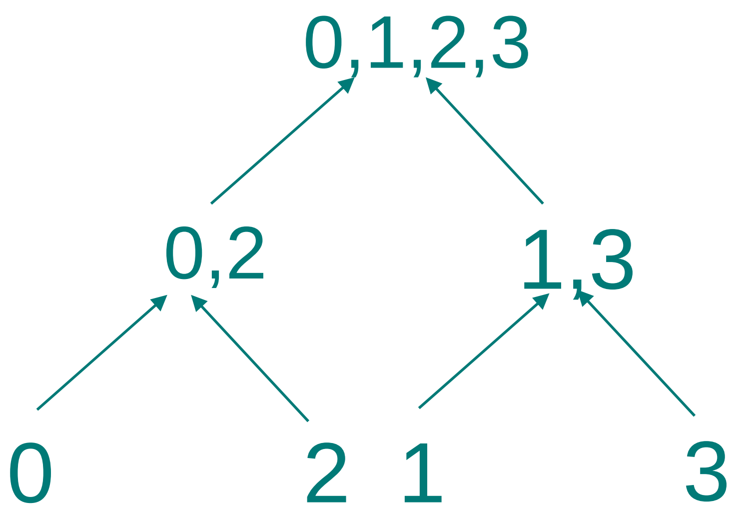
bit-reverse

Original index 1, binary code (001)→ (100), output index 4.

Exercise 1

Compute DFT of vector (0,1,2,3)

Solution 1 (FFT):



Where N=4,

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)\, W_{N/2}^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)\, W_{N/2}^{rk}$$

$$= \sum_{r=0}^{1} x(2r)\, W_2^{rk} + W_4^k \sum_{r=0}^{1} x(2r+1)\, W_2^{rk}$$

Here, $W_4 = e^{\pi i/2} = i,$ k=0,1,2,3

Set $A(k) = \sum_{r=0}^{1} x(2r)\, W_2^{rk}$
$B(k) = \sum_{r=0}^{1} x(2r+1)\, W_2^{rk}$
Here $W_2 = e^{\pi i} = -1$,and k=0,1

Then
$$X(k) = A(k) + W_4^k B(k)$$
$$X(k+2) = A(k) - W_4^k B(k) \qquad \text{k=0,1}$$

If N=4, $A(k)$ and $B(k)$ are DFT with 2 point.
So we have :
$$A(0) = x(0) + x(2) = 2$$
$$A(1) = x(0) - x(2) = -2$$
$$\mathrm{B}(0) = x(1) + x(3) = 4$$
$$\mathrm{B}(1) = x(1) - x(3) = -2$$

Thus,

According to $X(k) = A(k) + W_4^k B(k)$

$$\mathrm{X}(0) = A(0) + W_4^0 B(0) = 2 + 1 * 4 = 6$$

$$\mathrm{X}(1) = A(1) + W_4^1 B(1) = -2 - 2i$$

According to $X(k+2) = A(k) - W_4^k B(k)$

$$\mathrm{X}(2) = A(0) - W_4^0 B(0) = 2 - 1 * 4 = -2$$

$$\mathrm{X}(3) = A(1) - W_4^1 B(1) = -2 - (-2i) = -2 + 2i$$

So $\mathrm{X}(k)$ (where k=0,1,2,3) are DFT of (0,1,2,3),

It's $(6, -2 - 2\mathrm{i}, -2, -2 + 2\mathrm{i})$.

Solution 2 (Definition of DFT):

$$A(x) = \sum_{j=0}^{n-1}\left(a_j x^j\right) = 0x^0 + 1x^1 + 2x^2 + 3x^3 = x + 2x^2 + 3x^3$$

$$\omega_4 = e^{2\pi i/n} = e^{\pi i/2} = i$$

$$y_0 = A\left(\omega_4^0\right) = A(1) = 6$$

$$y_1 = A\left(\omega_4^1\right) = A(i) = -2 - i$$

$$y_2 = A\left(\omega_4^2\right) = A(-1) = -2$$

$$y_3 = A\left(\omega_4^3\right) = A(-i) = -2 + 2i$$

The DFT is vector $y = (y_0, y_1, y_2, y_3)$,

So the DFT of vector (0,1,2,3) is (6, -2-i, -2, -2+2i).