Seksjon 2.4: FFT (Fast Fourier Transform)
(effektiv algoritme for DFT)

Bevis for teorem 2.30

Anta at $0 \leq n < \frac{N}{2}$:

$$y_n = \sum_{k=0}^{N-1} x_k e^{-2\pi i k n/N} = \sum_{k=0}^{\frac{N}{2}-1} x_{2k} e^{-2\pi i (2k) n/N} + \sum_{k=0}^{\frac{N}{2}-1} x_{2k+1} e^{-2\pi i (2k+1) n/N}$$

$$= \sum_{k=0}^{\frac{N}{2}-1} x_{2k} e^{-2\pi i k n/(\frac{N}{2})} + e^{-2\pi i n/N} \sum_{k=0}^{\frac{N}{2}-1} x_{2k+1} e^{-2\pi i k n/(\frac{N}{2})}$$

$$= \left(DFT_{\frac{N}{2}}(x^{(e)})\right)_n + e^{-2\pi i n/N} \left(DFT_{\frac{N}{2}}(x^{(o)})\right)_n$$

Gjør vi dette for alle n slik at $0 \leq n < \frac{N}{2}$ så får vi:

$$(y_0, y_1, \ldots, y_{\frac{N}{2}-1}) = DFT_{\frac{N}{2}}(x^{(e)}) + D_{\frac{N}{2}} DFT_{\frac{N}{2}}(x^{(o)})$$

Helt tilsvarende vises

$$(y_{\frac{N}{2}}, \ldots, y_{N-1}) = DFT_{\frac{N}{2}}(x^{(e)}) - D_{\frac{N}{2}} DFT_{\frac{N}{2}}(x^{(o)})$$

Formlene

$$(y_0, \cdots, y_{\frac{N}{2}-1}) = DFT_{\frac{N}{2}}(\vec{x}^{(e)}) + D_{\frac{N}{2}} DFT_{\frac{N}{2}}(\vec{x}^{(o)})$$

$$(y_{\frac{N}{2}}, \ldots, y_{N-1}) = DFT_{\frac{N}{2}}(\vec{x}^{(e)}) - D_{\frac{N}{2}} DFT_{\frac{N}{2}}(\vec{x}^{(o)})$$

Disse kan også skrives på blokkmatriseform:

$$\vec{y} = \begin{pmatrix} DFT_{\frac{N}{2}} & D_{\frac{N}{2}} DFT_{\frac{N}{2}} \\ DFT_{\frac{N}{2}} & -D_{\frac{N}{2}} DFT_{\frac{N}{2}} \end{pmatrix} \begin{pmatrix} x^{(e)} \\ x^{(o)} \end{pmatrix}$$

$$= \begin{pmatrix} I_{\frac{N}{2}} & D_{\frac{N}{2}} \\ I_{\frac{N}{2}} & -D_{\frac{N}{2}} \end{pmatrix} \begin{pmatrix} DFT_{\frac{N}{2}} & 0 \\ 0 & DFT_{\frac{N}{2}} \end{pmatrix} \begin{pmatrix} x^{(e)} \\ x^{(o)} \end{pmatrix}$$

Hva er matrisen P ?

$$\vec{X} \to \begin{pmatrix} X^{(e)} \\ X^{(o)} \end{pmatrix} \begin{array}{l} \to \text{siste bit: indeksen er } 0 \\ \to \text{siste bit: indeksen er } 1 \end{array}$$

Dette svarer til å reorganisere $\vec{X}$ ved å sette siste bit først i indeksen.

På samme måte:

$$\begin{pmatrix} X^{(e)} \\ X^{(o)} \end{pmatrix} \to \begin{pmatrix} X^{(ee)} \\ X^{(eo)} \\ X^{(oe)} \\ X^{(oo)} \end{pmatrix}$$ svarer til å sette nest siste bit som andre bit.

Vi fortsetter slik: P svarer da til en permutasjon, der bitene reverseres.

feks: $100_2 \to 001_2$

$4 \to 1$

DFT krever N^2 komplekse multiplikasjoner
og $\approx N^2$ komplekse addisjoner.

en kompleks add. $\longleftrightarrow$ to reelle add.
en kompleks mult. $\longleftrightarrow$ 4 reelle mult. og 2 reelle add.

$$(x_1+iy_1)(x_2+iy_2) = x_1x_2 - y_1y_2 + i(y_1x_2 + x_1y_2)$$

N^2 komplekse mult: $\rightarrow$ $4N^2$ reelle mult., og $2N^2$ reelle add.
N^2 komplekse add: $\rightarrow$ $2N^2$ reelle add.

til sammen $8N^2$ operasjon
(like mange add. som mult.)

La M_N, A_N være antall kompleks multiplikasjoner og addisjoner i FFT algoritmen. Vi har at: (N = matrisestørrelse)

$$M_N = 2M_{\frac{N}{2}} + \frac{N}{2} \quad \xrightarrow{} \begin{array}{l} 2N \text{ reelle mult} \\ + N \text{ reelle adds.} \end{array} \qquad A_N = 2A_{\frac{N}{2}} + N$$

$DFT_{\frac{N}{2}}(x^{(e)}) + D_{\frac{N}{2}} DFT_{\frac{N}{2}}(x^{(o)})$

$DFT_{\frac{N}{2}}(x^{(e)}) - D_{\frac{N}{2}} DFT_{\frac{N}{2}}(x^{(o)})$ — $2N$ reelle adds.

La istedet M_N være reelle mult.: Da blir

$M_N = 2M_{\frac{N}{2}} + 2N$, $A_N = 2A_{\frac{N}{2}} + 3N$

Sett $x_n = M_{2^n}$ $\quad 2^n$

$M_{2^n} = 2M_{2^{n-1}} + 2N$

$x_n = 2x_{n-1} + 2\cdot 2^n$

$x_{n+1} = 2x_n \cdot 2\cdot 2^{n+1}$

$x_{n+1} - 2x_n = 4\cdot 2^n$

homogen løsning: $x_{n+1} - 2x_n = 0$ har løsning $x_n^h = C\cdot 2^n$

inhomogen: Vi prøver $x_n^p = A\cdot n 2^n$:

$A(n+1)2^{n+1} - 2An2^n = 4\cdot 2^n$

$\cancel{2An2^n} + 2A2^n - \cancel{2An2^n} = 4\cdot 2^n$

$2A2^n = 4\cdot 2^n \Rightarrow A = 2 \Rightarrow x_n^p = 2n2^n$

$x_n = x_n^p + x_n^h = 2n2^n + C\cdot 2^n$.

Initialbetingelse: $x_1 = M_{2^1} = 0$, og $x_2 = M_{2^2} = 0$

Setter inn $x_2 = 0$ og får: $0 = 2\cdot 2\cdot 4 + C\cdot 2^2$

$C = -4$

$\Rightarrow x_n = 2n2^n - 4\cdot 2^n$

$\overset{N=2^n}{\Rightarrow}$ $\underset{n=\log_2 N}{\Rightarrow}$ $M_N = 2N\log_2 N - 4N$

Tilsvarende får vi for addisjoner at $A_N = 3N\log_2 N - N$

Tilsammen: $M_N + A_N = 5N\log_2 N - 5N$

Vi har redusert antall operasjoner fra

$8N^2$ til $5N\log_2 N - 5N$

Anta at en algoritme for dimensjon N trenger R_N regneops.
Vi sier at algoritmen har orden $f(N)$ hvis $\lim_{N\to\infty} \frac{R_N}{f(N)} = 1$,
og sier at algoritmen er $O(f(N))$

Eks: FFT-algoritmen er $O(5N\log_2 N)$, siden

$$\frac{R_N}{f(N)} = \frac{5N\log_2 N - 5N}{5N\log_2 N} \to 1 \text{ når } N\to\infty$$