Diskretna Fourier-ova Transformacija – DFT

Zadan je $n \in \mathbb{N}$ i konačni niz ili vektor $a \in \mathbb{C}^n$ od n elemenata
$$a = (a_0, a_1, \ldots, a_{n-1})^T.$$
Taj niz/vektor možemo interpretirati i kao funkciju
$$a : \mathbb{Z}_n \to \mathbb{C}$$
gdje je $\mathbb{Z}_n = \{0, \ldots, n-1\}$ standardni sustav ostataka modulo n.

Elemente vektora a indeksiramo elementima iz $\mathbb{Z}_n$, a ne iz $\{1, \ldots, n\}$, kao što je uobičajeno u linearnoj algebri. Razlog nije samo tradicija u obradi signala, nego ima i matematičku pozadinu.

Vektor a možemo zamišljati kao diskretni uzorak nekog (može i kontinuiranog) signala, koji je "snimljen" u trenucima $t = 0, \ldots, n-1$.

Vektoru a možemo prirodno pridružiti polinom A nad $\mathbb{C}$, oblika
$$A(x) = \sum_{j=0}^{n-1} a_j x^j.$$
Takvi polinomi (kao i vektori a, duljine n) čine vektorski prostor nad $\mathbb{C}$, dimenzije n, izomorfan s $\mathbb{C}^n$. Zato kažemo da polinom A ima <u>red n</u>, tj. stupanj je $\leq n-1$.

Vektore, gledane kao funkcije na $\mathbb{Z}_n$, možemo i množiti – po točkama. Analogno, polinome možemo množiti (algebra!). Međutim, odmah uočavamo da ovakvo pridruživanje
$$a \longmapsto A$$
<u>nije</u> homomorfizam obzirom na množenje (Očito je homomorfizam na ostale operacije – zbrajanje i množenje skalarom – tj. između vektorskih prostora).

Naime, za "funkcijsko" ili Hadamard-ovo množenje vektora vrijedi

$$c = a \cdot b \iff c_j = a_j \cdot b_j, \quad j = 0, \ldots, n-1,$$

dok za množenje polinoma vrijedi

$$C = A \cdot B \iff c_j = \sum_{k=0}^{j} a_k \cdot b_{j-k}, \quad j = 0, \ldots, \underline{2n-2}$$

pa je produkt C reda $2n-1$.

Da bismo uspostavili vezu s produktom polinoma, na vektorima uvodimo novu operaciju $\otimes$, koju zovemo konvolucija, tako da je

$$(a \otimes b)_j = \sum_{k=0}^{n-1} a_k \, b_{j-k}, \quad j = 0, \ldots, n-1.$$

$\hookrightarrow$ periodičko pros.

Ovo još uvijek ne odgovara relaciji za koeficijente produkta polinoma. I ne može, jer redovi (duljine) nisu ujednačeni.

Međutim, ako nizove/vektore a i b produljimo nulama do dovoljne duljine N, pripadni polinomi A i B ostaju isti, samo ih gledamo u većem vektorskom prostoru. Naravno, i $C = A \cdot B$ se ne mijenja. Zbog toga je dovoljno uzeti $N \geq 2n-1$, pa da i niz c koeficijenata produkta bude cijeli prikaziv.

Dakle, napravimo

$$\left.\begin{aligned} a &\mapsto a' = (a_0, \ldots, a_{n-1}, 0, \ldots, 0)^T \\ b &\mapsto b' = (b_0, \ldots, b_{n-1}, 0, \ldots, 0)^T \end{aligned}\right\} \text{duljine } N$$

i $\quad c' = a' \otimes b' = (c'_0, \ldots, c'_{N-1})$

Ako je $N \geq 2n-1$, onda vrijedi

$$c'_j = \sum_{k=0}^{N-1} a'_k \, b'_{j-k} = \sum_{k=0}^{j} a_k \, b_{j-k}, \quad j = 0, \ldots, 2n-2$$

$$c'_j = 0, \quad j > 2n-2 \quad (j = 2n-1, \ldots, N-1)$$

i stavimo:

$$\underline{c := (c'_0, \ldots, c'_{2n-2})^T} \; - \text{ duljine } 2n-1.$$

Dobiveni niz c je pridružen polinomu $C = A \cdot B$.
(Preciznije, dobivenom nizu c pridružen je polinom $C = A \cdot B$).

Ovime smo uspostavili formalni "homomorfizam" konvolucije nizova i produkta polinoma. Stvarno, još ništa korisno nismo napravili, dok nemamo efikasne algoritme za računanje nečeg od toga!

Polinome zasad prikazujemo koeficijentima – u tzv. koeficijentnoj reprezentaciji. Ako se opet sjetimo interpolacije, možemo koristiti i drugačiji prikaz – vrijednostima na dovoljno velikom skupu točaka.

Sasvim općenito, za polinom A reda n možemo odabrati bilo koji skup od točno n različitih točaka $z_0, \ldots, z_{n-1} \in \mathbb{C}$ i dobiti tzv. vrijednosnu reprezentaciju – vektorom vrijednosti

$$\left(A(z_0), \ldots, A(z_{n-1})\right)^T \in \mathbb{C}^n.$$

Uočimo da u ovakvoj reprezentaciji, sve aritmetičke operacije na polinomima imaju jednostavan oblik – po točkama. To vrijedi i za produkt!

$$(A \cdot B)(z) = A(z) \cdot B(z)$$

Ako pazimo na red, sve operacije su efikasne – brzo se izvode – linearno u duljini nizova.

Ono što nam fali je efikasan prijelaz između te dvije reprezentacije

vektor koef. $\longleftrightarrow$ vektor vrijednosti.

Napomena: ovo ima smisla samo za produkt, ako nam uspije.

Zašto? Sve ostale operacije (zbrajanje, množenje skalarom) idu brzo u obje reprezentacije – tj. i u koeficijentnoj. Međutim, za množenje imamo samo standardni $O(n^2)$ algoritam u koeficijentnoj reprezentaciji, pa tu tražimo uštedu!

Efikasni prijelaz iz jedne u drugu reprezentaciju (i natrag) dobivamo pametnim izborom točaka $z_0, \dots, z_{n-1}$.

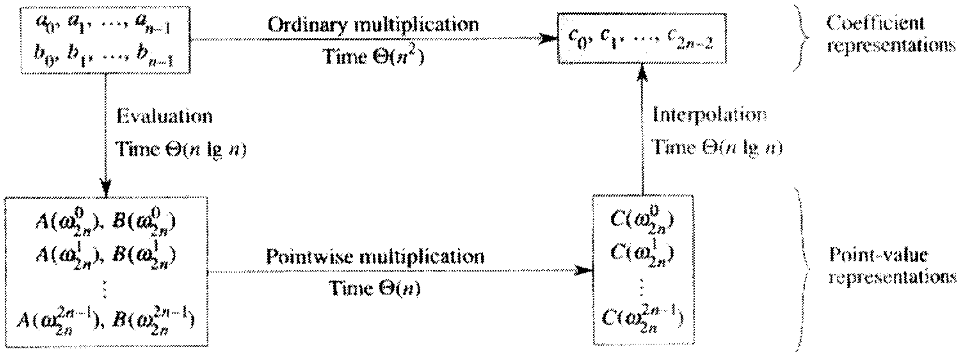

Figure 32.1 A graphical outline of an efficient polynomial-multiplication process. Representations on the top are in coefficient form, while those on the bottom are in point-value form. The arrows from left to right correspond to the multiplication operation. The ω_{2n} terms are complex $(2n)$th roots of unity.

input and output representations are in coefficient form. We assume that n is a power of 2; this requirement can always be met by adding high-order zero coefficients.

1. *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.
2. *Evaluate:* Compute point-value representations of $A(x)$ and $B(x)$ of length $2n$ through two applications of the FFT of order $2n$. These representations contain the values of the two polynomials at the $(2n)$th roots of unity.
3. *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.
4. *Interpolate:* Create the coefficient representation of the polynomial $C(x)$ through a single application of an FFT on $2n$ point-value pairs to compute the inverse DFT.

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 32.2
The product of two polynomials of degree-bound n can be computed in time $\Theta(n \lg n)$, with both the input and output representations in coefficient form. ∎

3. Konvolucijski teorem:

$$DFT_n(a \otimes b) = DFT_n(a) \cdot DFT_n(b)$$

(što smo već koristili i, zapravo, dokazali, ali za posebne nizove a', b').

32.2-3

Do Exercise 32.1-1 by using the $\Theta(n \lg n)$-time scheme.

32.2-4

Write pseudocode to compute DFT_n^{-1} in $\Theta(n \lg n)$ time.

32.2-5

Describe the generalization of the FFT procedure to the case in which n is a power of 3. Give a recurrence for the running time, and solve the recurrence.

32.2-6 ★

Suppose that instead of performing an n-element FFT over the field of complex numbers (where n is even), we use the ring $\mathbf{Z}_m$ of integers modulo m, where $m = 2^{tn/2} + 1$ and t is an arbitrary positive integer. Use $w = 2^t$ instead of ω_n as a principal nth root of unity, modulo m. Prove that the DFT and the inverse DFT are well defined in this system.

32.2-7

Given a list of values $z_0, z_1, \ldots, z_{n-1}$ (possibly with repetitions), show how to find the coefficients of the polynomial $P(x)$ of degree-bound n that has zeros only at $z_0, z_1, \ldots, z_{n-1}$ (possibly with repetitions). Your procedure should run in time $O(n \lg^2 n)$. (*Hint:* The polynomial $P(x)$ has a zero at z_j if and only if $P(x)$ is a multiple of $(x - z_j)$.)

32.2-8 ★

$y_k = \sum_{j=0}^{n-1} a_j z^{jk}$

The ***chirp transform*** of a vector $a = (a_0, a_1, \ldots, a_{n-1})$ is the vector $y = (y_0, y_1, \ldots, y_{n-1})$, where $y_k = \sum_{j=0}^{n-1} a_j z^{jk}$ and z is any complex number. The DFT is therefore a special case of the chirp transform, obtained by taking $z = \omega_n$. Prove that the chirp transform can be evaluated in time $O(n \lg n)$ for any complex number z. (*Hint:* Use the equation

$$y_k = z^{k^2/2} \sum_{j=0}^{n-1} \left(a_j z^{j^2/2}\right) \left(z^{-(k-j)^2/2}\right)$$

to view the chirp transform as a convolution.)

32.3 Efficient FFT implementations

Since the practical applications of the DFT, such as signal processing, demand the utmost speed, this section examines two efficient FFT implementations. First, we shall examine an iterative version of the FFT algorithm that runs in $\Theta(n \lg n)$ time but has a lower constant hidden in the Θ-notation than the recursive implementation in Section 32.2. Then, we shall use the insights that led us to the iterative implementation to design an efficient parallel FFT circuit.