SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Brza Fourierova transformacija

Kristijan Verović
Voditelj: *Adrian Satja Kurdija*

Zagreb, veljača 2023.

SADRŽAJ

1. Uvod **1**

2. FFT algoritam **2**

3. Performanse **4**

4. Primjene **6**

5. Problemi i stanje tehnike u množenju polinoma **8**

6. Literatura **9**

7. Sažetak **10**

1. Uvod

Fourierova transformacija je samo jedna od bezbroj mogućih transformacija u teoriji obrade signala, otkud dolazi i ime algoritma koji je tema ovog izlaganja. Fourierova transformacija može biti kontinuirana i diskretna, kao što često biva u računarstvu ovdje se proučava diskretan slučaj odnosno Diskretna Fourierova transformacija (DFT). Računamo li DFT naivno po njegovoj definiciji imat ćemo složenost $O(n^2)$ što je često u praksi previsoko. Brza Fourierova transformacija (eng. Fast Fourier transform) ili FFT je "Podijeli pa vladaj" (eng. divide and conquer) algoritam koji u $O(nlogn)$ vremenskoj složenosti izračunava DFT. Za razumijevanje DFT-a i FFT-a nije potrebno znanje iz područja obrade signala.

2. FFT algoritam

Za cijeli opis algoritma s dokazima međukoraka (algebarske prirode) pogledati [6] i [5]. Na slici 2.1 ispod nalazi se definicija DFT-a, definiran kao množenje matrice nxn i vektora $nx1$ što ima složenost $O(n^2)$. Radi se o evaluaciji polinoma s koeficijentima prikazanim u vektoru $(a_0, a_1, ..., a_{n-1})$ u točkama w_n^k gdje k ide od 0 do $(n-1)$. Ovdje je $w_n^k = e^{\frac{2\Pi * i}{n}}$. Korištenje kompleksnih brojeva je ključan dio rada FFT algoritma jer će se pomoću njih dobiti algebarska svojstva koja omogućuje podjeli pa vladaj pristup računanju DFT-a.

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

Slika 2.1: Matrična definicija DFT-a.

Želimo pomnožiti dva polinoma A i B. Neka je n jednak $n = max(deg(A), deg(B))$. Dopunit ćemo koeficijente nulama i A i B do druge najveće potencije broja 2 nakon broja n i reći da je to novi n. Sada računamo $DFT(A)$ i $DFT(B)$, konkretni izračun DFT(A) u algoritmu je sljedeći. Razdijelimo A na A_0 koji ima samo koeficijente uz parne eksponente polinoma i na A_1 koji ima samo neparne:

$$A_0(x) = a_0 + a_2x + a_4x^2 + ... + a_{n/2-2}x^{n/2-1}$$

$$A_1(x) = a_1 + a_3x + a_5x^2 + ... + a_{n/2-1}x^{n/2-1}$$

$$ako(kn/2) : DFT(A)_k = A_0(w_{n/2}^{k-n/2}) - w_n^{k-n/2} * A_1(w_{n/2}^{k-n/2})$$

Ovdje je jasno u kojem smjeru ide divide and conquer, samo cijepamo dalje A_0 i A_1 na manje polovične podsegmente. Potom množimo $DFT(A)$ i $DFT(B)$ jer $DFT(A *$

$B) = DFT(A) * DFT(B)$. Sada kada imamo evaluiran polinom $A * B$ u točkama definiranim DFT-om mi želimo napraviti inverznu transformaciju, odnosno iz evaluiranih točaka polinoma naći koeficijente originalnog polinoma. Želimo InverseDFT kojim bismo postigli $A * B = InverseDFT(DFT(A) * DFT(B))$. To možemo postići ako nađemo inverznu matricu matrici iz slike 2.1. Lako se provjeri da je inverzna matrica toj matrici ona prikaza sljedećom slikom 2.2.

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

Slika 2.2: Inverzni DFT.

Ovdje se radi o idejno identičnom izračunu kao i za DFT stoga je složenost i ovog $O(nlogn)$, naime:

$a_k = \frac{1}{n} \sum_{j=0}^{n-1} y_j * w_n^{-kj}$

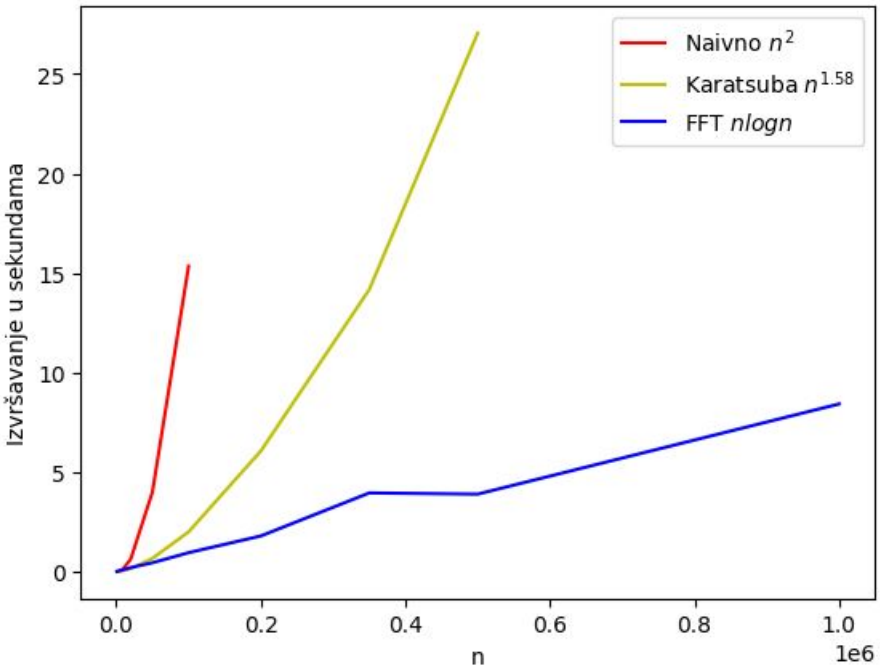
Što jako liči na izraz za DFT:

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

Iz čega se može zaključiti da se inverzni FFT može riješiti istim divide and conquer algoritmom uz minimalnu izmjenu koda za FFT.

3. Performanse

Usporedit ćemo vremenske karakteristike 2 algoritma za množenje polinoma s FFT-om: naivni $O(n^2)$ i Karatsubin algoritam $O(n^{1.58})$ [2]. Implementacije kodova pojedinih algoritama rađene su u programskom jeziku C++ i nalaze se na [1]. Algoritmi su kompajlirani u GCC-u sa zastavicama "-std=c++11 -O2", ovdje je -O2 često korištena optimizacijska zastavica. Performanse su prikazane u grafu na slici 3.1.



Slika 3.1: Usporedba algoritama za množenje polinoma.

Na grafu se zastoj brzine FFT-a između 350000 i 500000 može objasniti zbog dizanja n na potenciju broja 2, a 350000 i 500000 će se oboje dići na istu potenciju jer su oboje između $2^{18} = 262144$ i $2^{19} = 524288$. U tablici 3.1 ispod se nalaze konkretni brojevi preko kojeg je dobiven gornji graf 3.1, mjereno na procesoru "AMD A10-5757M", laptop Acer-Aspire V5-552G.

Tablica 3.1: Usporedba prosječnih vremena tri algoritma u sekundama.

N	naivni	Karatsuba	FFT
1000	0.0017	0.0021	0.0055
2000	0.006	0.0064	0.011
5000	0.039	0.0207	0.046
10000	0.16	0.051	0.093
20000	0.62	0.149	0.198
50000	4	0.66	0.44
100000	15.34	1.99	0.95
200000	-	6.06	1.79
350000	-	14.17	3.92
500000	-	27	3.88
1000000	-	-	8.42

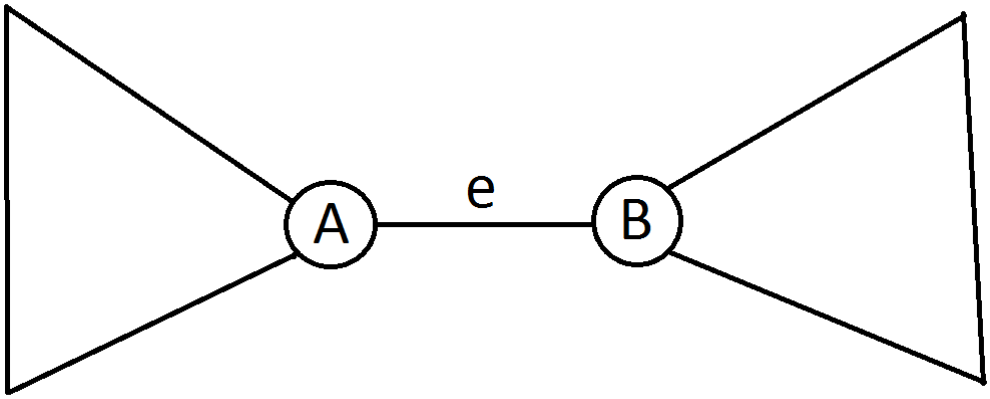
Budući da se radi o Podjeli pa vladaj algoritmu, posve je prirodno uvesti ovdje paralelizam. Pretpostavimo da raspolažemo s $p = 2^k$ procesora te možemo jednostavno razdijeliti memoriju svakom procesoru, onda sve od prve do uključivo $k-te$ razine sve se izračuna u $O(n)$, a onda tek na dalje izračun ide kao da smo imali jedan procesor. Preciznije, složenost je $O(n * (logn - logp)) = O(nlog(n/p))$, detaljnije na [4].

4. Primjene

Mi želimo FFT koristiti prvenstveno u svrhu množenja polinoma, a posljedično i velikih brojeva. Za velike brojeve, samo stavimo $x = 10$ (ili drugu bazu samo onda malo drugačije), pomnožimo dva polinoma, i tamo gdje imamo koeficijente > 10 guramo znamenke prema većim potencijama, složenost ovog djela je $O(Nlog_bN)$ što je pokriveno nlogn složenošću. Razlog iza $O(Nlog_bN)$ je da uz 10^k koeficijent može biti najviše $(k * b)^2$, gdje je b baza, a logaritam od toga je jednak $log_b(k * b)^2$ = $2log_b(k * b)$ = $2log_b(k) + 2$ što je asimptotski jednako $O(log_b(k))$ odnosno $O(log_b(N))$. To je naravno bila složenost potrebna da pomaknemo viška znamenke na gornje eksponente broja nakon množenja, pa onda to ispadne uzevši u obzir svaku znamenku $O(n * logn)$. Implementacija u C++-u nalazi se na [1].

Očita primjena je u području obrade signala otkud uopće i dolazi definicija DFT-a te se ekstenzivno koristi. Studenti na diplomskom studiju Fakulteta Elektrotehnike i Računarstva u Zagrebu koriste gotove module za izračun FFT-a na nekim predmetima u sklopu laboratorijskih vježbi.

Primjena je ogromna u izračunu kombinatoričkih problema uz pomoć polinoma. Primjer jednog takvog problema bio bi sljedeći. Pretpostavimo da imamo graf G koji je stablo, proučavamo u stablu jedan brid e. Zanima nas koliko puteva koje duljine prolazi kroz taj brid. Ilustrirano slikom 4.1.



Slika 4.1: Stablo

Promatramo A i B kao korijene svojih podstabla. Izračunamo koliko puteva koje duljine ide od nekog čvora u podstablu do korijena podstabla, to je izračunljivo jednostavno u $O(n)$. Kada bismo u podstablu A imali 5 puteva duljine jedan, 3 puteva duljine dva, 9 puteva duljine tri te 6 puteva duljine četiri, onda bismo to prezentirali kao (dodano +1 za sam čvor A puta duljine 0): $A = 1 + 5x + 3x^2 + 9x^3 + 6x^4$ Analogno bi napravili za podstablo B te bi pomnožili ta dva polinoma, u rezultirajućem polinomu koeficijent c_k uz x^k bi označavao koliko ima puteva duljine $k + 1$.

5. Problemi i stanje tehnike u množenju polinoma

Fokusiramo li se samo na cjelobrojne koeficijente (s realnim koeficijentima FFT ima manju grešku od naivnih pristupa), postavlja se pitanje kad će početi dolaziti do pogreške budući da FFT koristi tipove podataka pomičnog zareza u svom izračunu. Postoje algoritmi koji nemaju problema s preciznošću te daju točan rezultat za veće polinome i koeficijente jer koriste Number-theoretic transform (NTT). Za usporedbu DFT i NTT pogledati [3]. Schönhage–Strassen algoritam iz 1971-e se može koristiti za množenje velikih brojeva u $O(n * logn * loglogn)$. Algoritam iz 2019-e Harvey-van der Hoeven ima složenost $O(nlogn)$, ali tek za jako veliki n asimptotski dolazi do izričaja (Galactic algorithm) i nije primjenjiv.

6. Literatura

[1] https://github.com/Weramajstor/Seminar-1.

[2] https://en.wikipedia.org/wiki/Karatsuba_algorithm.

[3] https://en.wikipedia.org/wiki/Discrete_Fourier_transform_over_a_ring#Properties.

[4] https://cs.wmich.edu/gupta/teaching/cs5260/5260Sp15web/studentProjects/tiba&hussein/03278999.pdf.

[5] https://cp-algorithms.com/algebra/fft.html, 2008.

[6] https://cp-algorithms.com/algebra/fft.html, 2022.

7. Sažetak

Prolazi se kroz algoritam FFT. Analiziraju se performanse istog u usporedbi s drugim algoritmima. Spominje se mogućnost paralelizma algoritma. Prezentiraju se primjene FFT-a te se na kraju osvrće na probleme FFT-a i stanje tehnike (eng. state-of-the-art) u području množenja polinoma i velikih brojeva.