UNIVERSITATEA TRANSILVANIA DIN BRASOV
FACULTATEA DE MATEMATICA SI INFORMATICA

PROGRAMUL DE STUDII - INFORMATICA

ALGORITMI FUNDAMENTALI
Înmulțirea polinoamelor utilizând FFT

-proiect-

STUDENT:
Opriș Andrei Ioan

BRAȘOV

2020

CUPRINS

1 Enunțul problemei

Fie A(x) si B(x) - două polinoame de tipul $A(x) = \sum_{i=1}^{n} a_i x^i$, $B(x) = \sum_{i=1}^{m} b_i x^i$. Se cere să se utilizeze un algoritm de calul al polinomul $C(x) = \sum_{k=1}^{n+m} c_i x^i$, care sa aibe o complexitate mai mica de $O(n^2)$ utilizând o tehnică de programare de tip Divide et Impera.

2 Explicație rezolvare

2.1 Contextul problemei

Un polinom se poate reprezenta prin:

- coeficienți $[a_0, a_1, a_2, \dots, a_k]$
- grafic $\{(x_0, A(x_0)),(x_1, A(x_1)), \dots, (x_k, A(x_k))\}$

Reprezentarea grafic a polinoamelor duce la următoarea proprietate:

- Un polinom de grad n poate fi unic definit de n+1 puncte - $\{(x_0, A(x_0)),(x_1, A(x_1)), \dots, (x_k, A(x_k))\}$

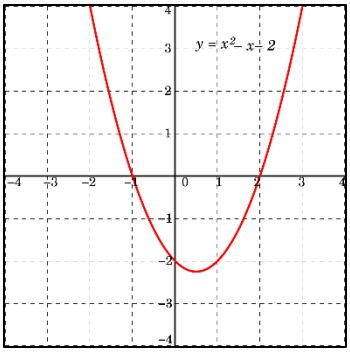


Fig 1 – polinom de grad 2 definit prin 3 puncte – sursa – https://en.wikipedia.org/wiki/Quadratic_function

Exemplu – (Fig. 1) - 3 puncte în spațiu {(a1, a2),(b1, b2),(c1, c2)} pot definii un polinom de gradul 2 de forma: $y = ax^2 + bx + c$.

Utilizând proprietatea de mai sus, rezulta faptul că un polinom poate fi determinat utilizând următorul sistem:

$$\begin{aligned} A(x_0) &= a_0 + a_1 x_0^1 + a_2 x_0^2 + \cdots + a_n x_0^n \\ A(x_1) &= a_0 + a_1 x_1^1 + a_2 x_1^2 + \cdots + a_n x_1^n \\ &\vdots \\ A(x_n) &= a_0 + a_1 x_n^1 + a_2 x_n^2 + \cdots + a_n x_n^n \end{aligned} \tag{2.1}$$

De asemenea, sistemul (2.1) se poate scrie si sub forma:

$$\begin{bmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_n) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^2 \\ 1 & x_1 & x_1^2 & \cdots & x_1^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \tag{2.2}$$

Matricea valorilor din relația (2.2) este o matrice de tip Vandermonde – pentru care se poate scrie următoarea proprietate:

$$\det(V) = \prod_{0 \le j \le k \le n-1} (x_k - x_j) \neq 0 \tag{2.3}$$

$$\Rightarrow \begin{cases} \text{coeficientii } (a_0, a_1, \ \dots, \ a_n) \text{ există} \\ \text{polinomul poate fi definit de } A(x) \end{cases}$$

2.2 FFT (Transformata Rapidă Fourier) – Principiul algoritmului FFT

Transformata rapidă Fourier este un algoritm ce ajută la diminuarea complexității problemei de înmulțire a polinoamelor de la o complexitate inițială $O(n^2)$ la una de tip O(nlogn) – utilizând o metoda de programare de tip Divide et Impera.

Maniera principală prin care acest algoritm are o complexitate e tip O(nlogn) o reprezintă împărțirea polinomului inițial in doua jumătăți:

- una conținând termenii de grad par
- iar cealaltă pe cei de grad impar

Astfel, polinomul inițial:

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} \tag{2.4}$$

Se va transforma într-unul de forma:

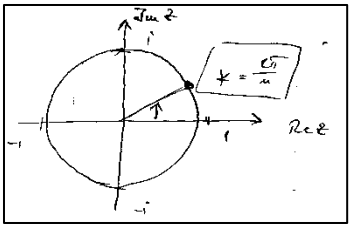
$$A(x) = (a_0 + a_2 x^2 + a_4 x^4 \dots + a_{n-2} x^{\frac{n-2}{2}}) + (a_1 x + a_3 x^3 + a_5 x^5 \dots + a_{n-1} x^{\frac{n-2}{2}}) \tag{2.5}$$

Se va nota cu A_e polinomul de grad par, iar A_o polinomul de grad impar. Aceste două polinoame se vor exprima în funcție de x^2, iar in polinomul A_o se va da factor comun un x, ceea ce va duce la o exprimare a celor două polinoame un funcție de același x^2, la fel ca in relația de mai jos:

$$A(x) = A_e(x^2) + x\, A_o(x^2) \tag{2.6}$$

Pentru calcularea noului polinom obținut este nevoie de $\left(x_1^2, x_2^2, \dots, x_{n/2}^2\right)$ valori pentru a calcula polinomul – adică jumătate din valorile inițiale, ducând la o complexitate de tip O(nlogn), deoarece polinomul obținut trebuie calculat pentru jumătate din valorile inițiale, necesitând timp liniar de calcul.

2.3 Rădăcinile de ordin N ale unității



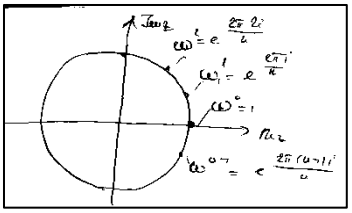
Un polinom, de asemenea, poate fi caracterizat prin rădăcinile complex conjugate de ordin n ale unității.

Așadar, dat fiind o ecuație de tipul $z^n=1$ – rădăcinile sale de ordin N ale unității au următoarele caracteristici:

- sunt complex conjugate
- rădăcinile de ordin n sunt exprimate prin relația

$$\omega = e^{\frac{2\pi i}{n}} \tag{2.7}$$

- trecerea de la forma exponențială la forma trigonometrică (Euler formula)



$$e^{i\theta} = \cos(\theta) + i\sin(\theta) \tag{2.8}$$

- rădăcinile de ordin N definesc un cerc de raza R=1 – i.e. cercul trigonometric

Ceea de rezultă că, pentru un polinom dat, acesta poate fi reprezentat prin rădăcinile de ordin N ale unității : $A(x) \rightarrow [1, \omega^1, \omega^2, \dots, \omega^{n-1}]$.

Însă, având polinomul împărțit în termeni de grad par și impar, polinomul trebuie calculat doar pentru N/2 rădăcini de ordin N ale unității $[1, \omega^2, \omega^4, \dots, \omega^{\frac{n-1}{2}}]$.

Astfel, principiul de funcționare al algoritmului FFT se rezuma la următorii pași:

- Pasul 1 – Identificarea problemei:

Dat fiind : $A(x)$: $[a_0, a_1, a_2, \dots, a_{n-1}]$ - coeficienții polinomului
Calculăm rădăcinile de ordin n: $\omega = e^{\frac{2\pi i}{n}}$:$[1, \omega^1, \omega^2, \dots, \omega^{n-1}]$

- Pasul 2 – Împărțirea polinomului în termeni pari/impari și determinarea rădăcinilor de ordin N ale unității

FFT – coeficienți pari $A_e(x)$: $[a_0, a_2, a_4, \dots, a_{n-2}]$ rad n: $\omega = e^{\frac{2\pi i}{n}}$:$[1, \omega^1, \omega^2, \dots, \omega^{n-2}]$

FFT – coeficienți impari $A_o(x)$: $[a_1, a_3, a_5, \dots, a_{n-1}]$ rad n: $\omega = e^{\frac{2\pi i}{n}}$:$[1, \omega^1, \omega^2, \dots, \omega^{n-2}]$

- Pasul 3 – rezolvarea sistemului pentru a obține reprezentarea prin valori a polinomului inițial a rădăcinilor complex conjugate

Datorită proprietăților funcțiilor pare/impare:

$$\begin{cases} A(\omega_i) = A_e(\omega_i^2) + \omega_i A_o(\omega_i^2) \\ A(-\omega_i) = A_e(\omega_i^2) - \omega_i A_o(\omega_i^2) \end{cases}$$

Reprezentarea prin valori a polinomului:

$$y_e = [A_e(1), A_e(\omega^2), \dots, A_e(\omega^{n-2})]$$

Sistemul necesar pentru a obține reprezentarea prin valori a polinomului se poate rezolva astfel:

$$\begin{cases} A(x_i) = A_e(x^{2i}) + x^i A_o(x^{2i}) \\ A(-x_i) = A_e(x_i^{2i}) - x^i A_o(\omega^{2i}) \end{cases}, i = 0,1, \dots, \frac{n-2}{2} \quad (2.9)$$

Dar $x_i = \omega^i$

$$\begin{cases} A(\omega^i) = A_e(\omega^{2i}) + \omega^i A_o(\omega^{2i}) \\ A(-\omega^i) = A_e(\omega_i^{2i}) - \omega^i A_o(\omega^{2i}) \end{cases}, i = 0,1, \dots, \frac{n-2}{2} \quad (2.10)$$

Dar $-\omega^i = \omega^{i+\frac{n}{2}}$

$$\begin{cases} A(\omega^i) = A_e(\omega^{2i}) + \omega^i A_o(\omega^{2i}) \\ A(-\omega^i) = A_e(\omega_i^{2i}) - \omega^i A_o(\omega^{2i}) \end{cases}, i = 0,1, \dots, \frac{n-2}{2} \quad (2.11)$$

Dar $A_e(\omega^{2i}) = y_{e_i}; \quad A_o(\omega^{2i}) = y_{o_i}$

$$\begin{cases} A(\omega^i) = y_{e_i} + \omega^i y_{o_i} \\ A\left(\omega^{i+\frac{n}{2}}\right) = y_{e_i} - \omega^i y_{o_i} \end{cases}, i = 0,1, \dots, \frac{n-2}{2} \quad (2.12)$$

2.4 FFT in pseudocod

function FFT(A):

input A – $[a_0, a_1, a_2, \dots, a_{n-1}]$ – coef. polin.

n = len(A) – putere a lui 2

if (n==1)

return A – A este o ct

$\omega = e^{\frac{2\pi i}{n}}$ - def ω pt det răd de ordin n ale unității

$A_e, A_o = [a_0, a_1, a_2, \dots, a_{n-2}][a_1, a_2, a_3, \dots, a_{n-1}]$ – împărțirea polin

$y_e, y_o = FFT(A_e), FFT(A_o)$ – apelarea recursivă a funcției

$y = [0] * n$ – inițializare pentru output

for i in range (n/2)

$y[i] = y_e[i] + \omega^i y_0[i]$

$y[i + \frac{n}{2}] = y_e[i] - \omega^i y_0[i]$

return y

2.5 Interpolarea rezultatelor pentru obținerea noului polinom - IFFT

Având sistemul inițial de la relația (2.1) si matriceal scris in relația (2.2), utilizând algoritmul FFT sistemul matriceal din relația (2.2) se poate scrie sub următoarea forma:

$$\begin{bmatrix} A(\omega_0^0) \\ A(\omega_1^1) \\ \vdots \\ A(\omega_n^n) \end{bmatrix} = \begin{bmatrix} 1 & \omega^0 & \omega^0 & \cdots & \omega^0 \\ 1 & \omega & \omega^2 & \cdots & \omega^{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^n & \omega^{2n} & \dots & \omega^{n^2} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \quad (2.13)$$

Matricea rădăcinilor de ordin N ale unităților necesare obținerii reprezentării prin valori a polinomului se mai numește și matricea transformării discrete Fourier (DFT), care inversată va rezulta în determinarea coeficienților polinomului, astfel:

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} 1 & \omega^0 & \omega^0 & \cdots & \omega^0 \\ 1 & \omega & \omega^2 & \cdots & \omega^{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^n & \omega^{2n} & \dots & \omega^{n^2} \end{bmatrix}^{-1} \begin{bmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_n) \end{bmatrix} \quad (2.14)$$

Iar procesând matricea DFT se poate ajunge la forma finala necesară determinării coeficienților noului polinom:

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \frac{1}{n} \begin{bmatrix} 1 & \omega^0 & \omega^0 & \cdots & \omega^0 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-n} & \omega^{-2n} & \dots & \omega^{-n^2} \end{bmatrix} \begin{bmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_n) \end{bmatrix} \quad (2.15)$$

2.6 IFFT in pseudocod

function IFFT(A):

input A – $[A(\omega^0), A(\omega^1), \dots, A(\omega^{n-1}),]$ – rep prin valori

n = len(A) – putere a lui 2

if (n==1)

return A – A este o ct

$\omega = (1/n)e^{\frac{-2\pi i}{n}}$ - def ω pt det răd de ordin n ale unității

$A_e, A_o = [\omega_0, \omega_1, \omega_2, \dots, \omega_{n-2}][\omega_1, \omega_2, \omega_3, \dots, \omega_{n-1}]$

$y_e, y_o = FFT(A_e), FFT(A_o)$ – apelarea recursivă a funcției

$y = [0] * n$ – initializare pentru output

for i in range (n/2)

$y[i] = y_e[i] + \omega^i y_0[i]$

$y[i + \frac{n}{2}] = y_e[i] - \omega^i y_0[i]$

return y

3 Demonstrație complexitate

3.1 Divide

Dat fiind un polinom A(x) cu x ∈ X, acesta necesita împărțit in termeni de grad par si impar, astfel:

$$A_e(x) = \sum_{k=0}^{\frac{n-2}{2}} a_{2k}x^k = (a_0 + a_2x^2 + a_4x^4 \dots + a_{n-2}x^{\frac{n-2}{2}})$$

$$A_o(x) = \sum_{k=0}^{n-1} a_{2k+1}x^k = (a_1x + a_3x^3 + a_5x^5 \dots + a_{n-1}x^{\frac{n-1}{2}}) \quad (3.1)$$

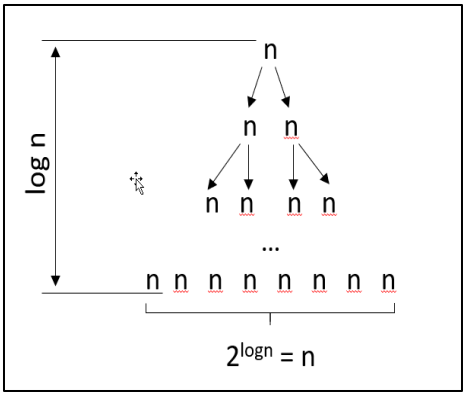
3.2 Stăpânește

Pentru toți termenii de grad par si impar recent divizați in subcapitolul 4.1 Divide trebuie calculate valorile noilor polinoame obținute astfel:

$A_e(x)$ si $A_o(x)$ trebuie calculat y_e si y_o astfel încât y ∈ X^2 = {x^2 | x ∈ X} (3.2)

Timpul de execuție pentru cazul general este:

$$T(n, |X|) = 2T(\frac{n}{2}, |X|) + O(n + |X|) = O(n^2) \quad (4.3)$$



Introducând conceptul de rădăcini de ordin n ale unității, se pot scrie următoarele relații:

- Pentru cazul de baza:
  |X| = 1 => ω ∈ {1} (3.3)

După care:

|X| = 2 => ω ∈{-1, 1}

|X| = 4 => ω ∈{-i, i, -1, 1}

|X| = 6 => ω ∈ $\left\{\pm\frac{\sqrt{2}}{2}(1+i), \pm\frac{\sqrt{2}}{2}(1-i), -i,\ i,\ -1,\ 1\right\}$ (3.4)

- Pentru cazul general:

$|X| = n \Rightarrow \omega = e^{\frac{2\pi i}{n}} = \cos\left(\frac{2\pi}{n}\right) + i \cdot \sin\left(\frac{2\pi}{n}\right)$ (3.5)

Expresia din relația 4.6 este defapt trecerea de la forma exponențială la forma trigonometrică (formula lui Euler).

Din relațiile 3.3, 3.4, 3.5 rezulta faptul ca timpul de execuție devine:

$$T(n, |\frac{X}{2}|) = 2T(\frac{n}{2}, |\frac{X}{2}|) + O(n + |\frac{X}{2}|) \quad (3.6)$$

,care poate fi rescris astfel:

$$T(n) = 2T(\frac{n}{2}) + O(n) = O(n \cdot \log n) \quad (3.7)$$

3.3 Combină

După determinarea rădăcinilor de ordin n ale unității pentru noile polinoame obținute din împărțirea celui inițial in termeni de grad par si impar, acestea trebuie combinate pentru a obține reprezentarea prin valori a polinomului inițial, astfel:

$$A(x) = A_e(x^2) + x\, A_o(x^2) \quad (3.8)$$

$$A(\omega^i) = A_e(\omega^{2i}) + \omega^i A_o(\omega^{2i}) \quad (3.9)$$

$$A(\omega^i) = y_{e_i} + \omega^i y_{o_i} \quad (3.10)$$

4 Demonstrație corectitudine prin inducție matematică

După cum am demonstrat in capitolul 3, pentru înmulțirea polinoamelor este necesară trecerea de la reprezentarea prin coeficienți la reprezentarea prin valori. Aceasta se face introducând in ecuație rădăcinile de ordin n ale unității.

Utilizând rădăcinile de ordin n ale unității vom demonstra corectitudinea algoritmului astfel:

- Pentru cazul de baza unde polinomul este o constanta:

$A(x) = x \Rightarrow y = \omega = x$ (4.1)

- După care daca polinomul este reprezentat de 2 coeficienți:

$|X| = 2 \Rightarrow \omega \in \{-1, 1\}$ (4.2)

- După care daca polinomul este reprezentat de 4 coeficienți:

$|X| = 4 \Rightarrow \omega \in \{-i, i, -1, 1\}$ (4.3)

- Pentru cazul general unde polinomul este reprezentat de n coeficienți:

$$|X| = n \Rightarrow \omega = e^{\frac{2\pi i}{n}} = \cos\left(\frac{2\pi}{n}\right) + i \cdot \sin\left(\frac{2\pi}{n}\right) \quad (4.4)$$

Presupunem ca pentru n – relația 4.4 este adevărata, vom demonstra ca pentru n+1 relația de mai sus este adevărata:

$$|X| = n+1 \Rightarrow \omega = e^{\frac{2\pi i}{n+1}} = \cos\left(\frac{2\pi}{n+1}\right) + i \cdot \sin\left(\frac{2\pi}{n+1}\right) \quad (4.5)$$

5 Execuție

5.1 Programul de înmulțire a două polinoame

```
// Inmult_poli.cpp : Acest fisier contine alforitmul de inmultire a doua polinoame
utuluzant Tranfsormata Rapida Fourider.
//

#define _USE_MATH_DEFINES
#include <bits.h>
#include <math.h>
#include <iostream>
#include <complex>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

// Pentru a stoca valorile complexe ale radacinilor de ordin n ale unitatilor folosim
complex<double>
typedef complex<double> cd;


// Functia recursiva pentru FFT - Transformata Rapida Fourier
vector<cd> fft(vector<cd> a)
{
	int n = a.size();

	// verificare daca polinomul introdus nu este cumva o constanta
	if (n == 1)
		return vector<cd>(1, a[0]);

	// definire vector pentru stocarea radacinilor de ordin n ale unitatilor
	vector<cd> w(n);
	for (int i = 0; i < n; i++)
	{
		double alpha = 2 * M_PI * i / n;
		w[i] = cd(cos(alpha), sin(alpha));
	}

	vector<cd> Ae(n / 2), Ao(n / 2);
	for (int i = 0; i < n / 2; i++)
```

```
	{
		// stocarea elementelor de grad par ale polinomului
		Ae[i] = a[i * 2];

		// stocarea elementelor de grad impar ale polinomului
		Ao[i] = a[i * 2 + 1];
	}

	// apelarea recursiva a functiei pentru elementele de ordin par
	vector<cd> y0 = fft(Ae);

	// apelarea recursiva a functiei pentru elementele de ordin impar
	vector<cd> y1 = fft(Ao);

	// definire vector pentru stocarea valorilor polinomului y0, y1, y2, ..., yn-1.
	vector<cd> y(n);

	for (int k = 0; k < n / 2; k++)
	{
		y[k] = y0[k] + w[k] * y1[k];
		y[k + n / 2] = y0[k] - w[k] * y1[k];
	}
	return y;
}



// Functia recursiva pentru IFFT - Inversa Transformatei Rapide Fourier
vector<cd> ifft(vector<cd> y)
{
	int n = y.size();
	// verificare daca valorea introdusa este un punct
	if (n == 1)
		return vector<cd>(1, y[0]);

	// definire vector pentru stocarea radacinilor de ordin n ale unitatilor
	vector<cd> wn(n);
	for (int i = 0; i < n; i++)
	{
		double alpha = -2 * M_PI * i / n;
		wn[i] = cd(cos(alpha), sin(alpha));
	}

	vector<cd> Ye(n / 2), Yo(n / 2);
	for (int i = 0; i < n / 2; i++)
	{
		// stocarea valorilor de grad par pentru determinarea coeficientilor
polinomului
		Ye[i] = y[i * 2];

		// stocarea valorilor de grad impar pentru determinarea coeficientilor
polinomului
		Yo[i] = y[i * 2 + 1];
	}

	// apelarea recursiva a functiei pentru determinarea coeficientilor pari ai
polinimului
	vector<cd> a0 = ifft(Ye);
```

```
	// Rapelarea recursiva a functiei pentru determinarea coeficientilor impari ai
polinimului
	vector<cd> a1 = ifft(Yo);

	// definire vector pentru stocarea valorilor polinomului a0, a1, a2, ..., an-1.
	vector<cd> a(n);

	for (int k = 0; k < n / 2; k++)
	{
		a[k] = ((a0[k] + wn[k] * a1[k]));
		a[k + n / 2] = ((a0[k] - wn[k] * a1[k]));
	}
	return a;
}


void afisarePolinom1(vector<cd>& polinom1)
{
	cout << "Polinom 1: " << "\n";
	for (int indexPoli1 = 0; indexPoli1 < polinom1.size(); ++indexPoli1)
	{
		cout << polinom1[indexPoli1] << "\n";
	}
	cout << "\n";
}

void afisarePolinom2(vector<cd>& polinom2)
{
	cout << "Polinom 2: " << "\n";
	for (int indexPoli2 = 0; indexPoli2 < polinom2.size(); ++indexPoli2)
	{
		cout << polinom2[indexPoli2] << "\n";
	}
	cout << "\n";
}

void fftPolinom1(vector<cd> polinom1, vector<cd>& vectorFftPolinom1)
{
	cout << "FFT pentru polinomul 1: " << "\n";
	vectorFftPolinom1 = fft(polinom1);
	for (int indexFftPoli1 = 0; indexFftPoli1 < polinom1.size(); ++indexFftPoli1)
	{
		cout << vectorFftPolinom1[indexFftPoli1] << "\n";
	}
	cout << "\n";
}

void fftPolinom2(vector<cd> polinom2, vector<cd>& vectorFftPolinom2)
{
	cout << "FFT pentru polinomul 2: " << "\n";
	vectorFftPolinom2 = fft(polinom2);
	for (int indexFftPoli2 = 0; indexFftPoli2 < polinom2.size(); ++indexFftPoli2)
	{
		cout << vectorFftPolinom2[indexFftPoli2] << "\n";
	}
	cout << "\n";
}
```

```
cd inmultire(cd termen1, cd termen2)
{
	cd produs = termen1 * termen2;
	return produs;
}


void inmultireValori(vector<cd> vectorFftPolinom1, vector<cd> vectorFftPolinom2,
vector<cd>& vectorMultipicareValori)
{
	cout << "Inmultirea valorilor celor doua polinoame: " << "\n";
	for (int indexMultip = 0; indexMultip < vectorFftPolinom1.size(); ++indexMultip)
	{
		vectorMultipicareValori[indexMultip] =
inmultire(vectorFftPolinom1[indexMultip], vectorFftPolinom2[indexMultip]);
		cout << vectorMultipicareValori[indexMultip] << "\n";
	}
	cout << "\n";
}


void ifftPolinom(vector<cd> vectorMultipicareValori, vector<cd> vectorIfftPolinom)
{
	cout << "IFFT: " << "\n";
	cd N = vectorIfftPolinom.size();
	cd finalIfft = 0;
	vectorIfftPolinom = ifft(vectorMultipicareValori);
	for (int indexIfft = 0; indexIfft < vectorIfftPolinom.size(); ++indexIfft)
	{
		finalIfft = vectorIfftPolinom[indexIfft] / N;
		cout << finalIfft << "\n";
	}
	cout << "\n";
}


int main()
{
	//definire polinoame pentru inmultit
	vector<cd> polinom1{ 1, 2, 1 }, polinom2{ 1, -2, 1 };

	//definire vectori pentru FFT si IFFT
	vector<cd> vectorFftPolinom1, vectorFftPolinom2;
	vector<cd> vectorIfftPolinom(polinom1.size() + polinom2.size());
	vector<cd> vectorMultipicareValori(polinom1.size() + polinom2.size() - 1);

	afisarePolinom1(polinom1);
	afisarePolinom2(polinom2);
	fftPolinom1(polinom1, vectorFftPolinom1);
	fftPolinom2(polinom2, vectorFftPolinom2);

	inmultireValori(vectorFftPolinom1, vectorFftPolinom2, vectorMultipicareValori);
	ifftPolinom(vectorMultipicareValori, vectorIfftPolinom);

	return 0;
}
```

5.2 Exemplu de execuție

Pașii de execuție a algoritmului FFT se pot vizualiza în fișierul PowerPoint din link-ul următor :

- Opris_Andrei_10LF204_Inmult_poli_FFT_Example

De asemenea, pentru o prezentare generala a algoritmului FFT se poate vizualiza fișierul PowerPoint din următorul link:

- Opris_Andrei_10LF204_Inmult_poli_FFT

6 Bibliografie

[1] - R. C. T. Lee, S. Tseng, Introduction to the Design and Analysis of Algorithms: A Strategic Approach (2005)

[2] - V Pan, Structured Matrices and Polynomials: Unified Superfast Algorithms, (2001)

[3] – Determinarea radacinilor de ordin n ale unitatii
https://physics.uvt.ro/~eugeniat/mate_gen/Cursuri2019/CURS9-2019.pdf

[4] – P. Bürgisser, M. Clausen, A. Shokrollahi Algebraic Complexity Theory, (1997)