s

Instituto Politécnico Nacional
Escuela Superior de Computo

Análisis de algoritmos

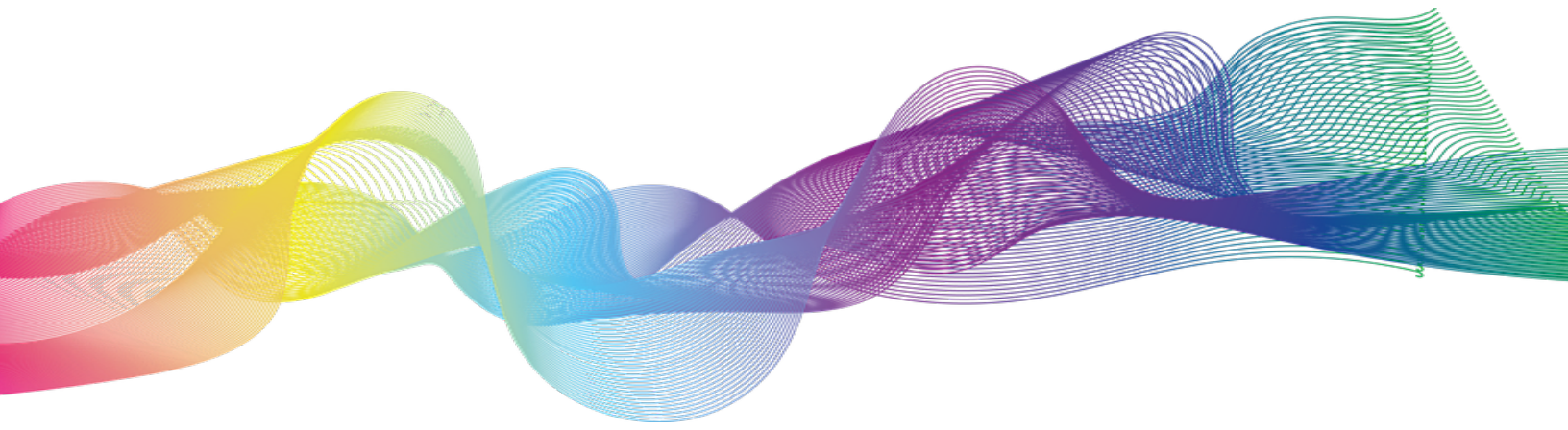
Sandra Díaz Santiago

Practica 3

Alumno:

- Hernández radilla José ángel.
- Vallejo serrano Ehecatzin.
- García González Aarón Antonio

Octubre, 2019



Puntos para cubrir:

1. Busca 2 implementaciones en lenguajes distintos a la FFT. Describe cuales son las diferencias (además de las obvias, por estar en lenguajes de programación distintos) entre ambas implementaciones.
2. Prueba cada una de las implementaciones anteriores con distintos ejemplos (al menos 3). Determina cual de las dos implementaciones funciona mejor.

Implementación 1

La primera implementación, está realizada en el lenguaje java y hace uso de otra clase llamada Complex, la cual contiene las operaciones elementales con números complejos. Algunas de las funciones se muestran a continuación:

Transformada.java Complex.java ×

home > catzin > Desktop > Complex.java

```java
import java.util.Objects;

public class Complex {
    private final double re;   // the real part
    private final double im;   // the imaginary part

    // create a new object with the given real and imaginary parts
    public Complex(double real, double imag) {
        re = real;
        im = imag;
    }
    // return a string representation of the invoking Complex object
    public String toString() {
        if (im == 0) return re + "";
        if (re == 0) return im + "i";
        if (im <  0) return re + " - " + (-im) + "i";
        return re + " + " + im + "i";
    }

    // return abs/modulus/magnitude
    public double abs() {
        return Math.hypot(re, im);
    }
```

Transformada.java Complex.java ×

home > catzin > Desktop > Complex.java

```
58
59      // return a new Complex object whose value is the conjugate of this
60      public Complex conjugate() {
61          return new Complex(re, -im);
62      }
63
64      // return a new Complex object whose value is the reciprocal of this
65      public Complex reciprocal() {
66          double scale = re*re + im*im;
67          return new Complex(re / scale, -im / scale);
68      }
69
70      // return the real or imaginary part
71      public double re() { return re; }
72      public double im() { return im; }
73
74      // return a / b
75      public Complex divides(Complex b) {
76          Complex a = this;
77          return a.times(b.reciprocal());
78      }
79
80      // return a new Complex object whose value is the complex exponential of this
81      public Complex exp() {
82          return new Complex(Math.exp(re) * Math.cos(im), Math.exp(re) * Math.sin(im));
83      }
84
```

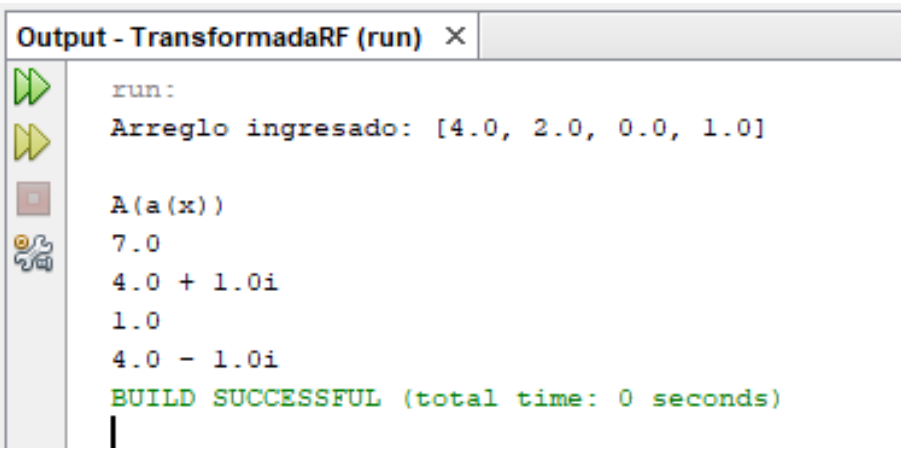
La implementación de la transformada rápida de Fourier es la siguiente:

```
public class Transformada{

    public static ArrayList<Complex> calcular(ArrayList<Complex> a){
        int n = a.size();
        if( n == 1){
            return a;
        }

        Complex w = new Complex(1,0);
        Complex wn = new Complex(Math.cos(2*Math.PI/n),Math.sin(2*Math.PI/n));
        ArrayList<Complex>pares = new ArrayList<Complex>();
        ArrayList<Complex>impares = new ArrayList<Complex>();

        ArrayList<Complex> y = new ArrayList();
        for(int i = 0; i < n; i++){
            if(i % 2 == 0){
                pares.add(a.get(i));
            }
            else{
                impares.add(a.get(i));
            }
```

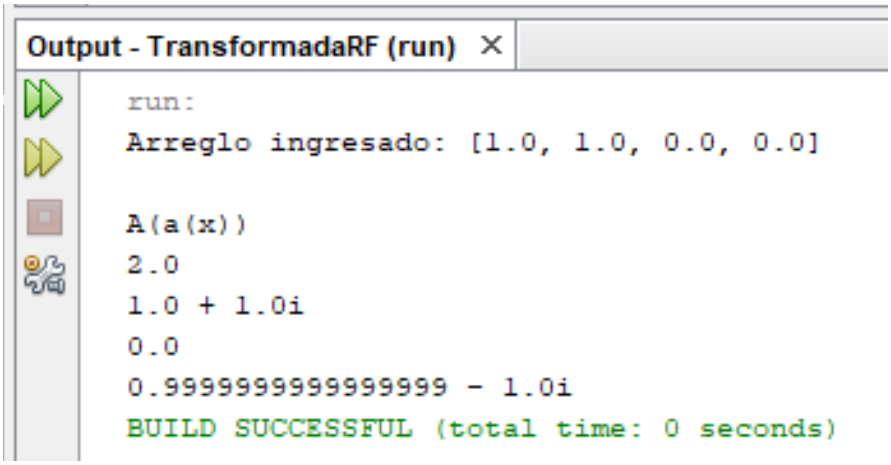
```
        }

        for(int i=0;i<n;i++)y.add(new Complex(0,0));

        pares = calcular(pares);
        impares = calcular(impares);

        for(int k = 0; k <=  (n/2)-1 ; k++){

            y.set(k,Complex.plus(pares.get(k), impares.get(k).times(w)));
            y.set(k+(n/2),pares.get(k).minus(impares.get(k).times(w)));
            w = w.times(wn);
        }
        return y;
    }
```

Pruebas

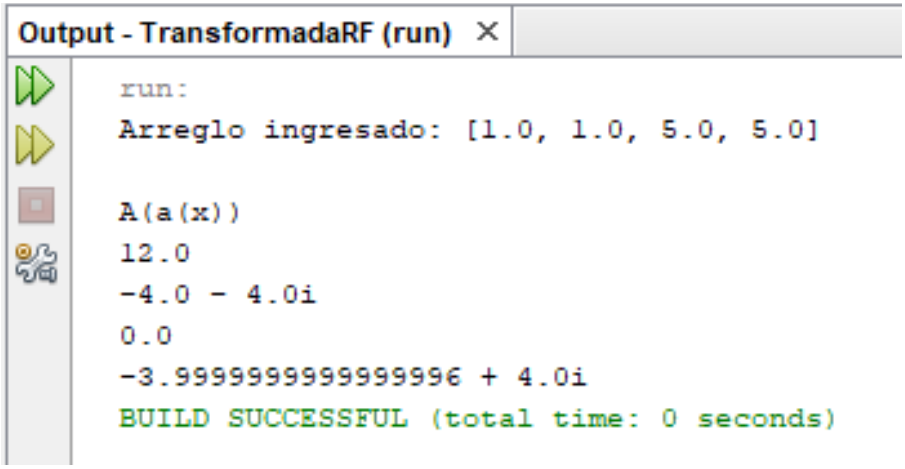
A continuación, se presentarán las pruebas con las siguientes entradas:
A [ ] = [4,2,0,1] = $4 + 2x + x^3$



A[] = [1,1,0,0] = 1 + x



A[ ] = [1,1,5,5] = $1 + x + 5x^2 + 5x^3$



Implementación 2

La segunda implementación está realizada en el lenguaje Python, y es la siguiente:

```
import math
import cmath

# Obtenemos la raiz principal
def omega(p, q):
   return cmath.exp((2.0 * cmath.pi * 1j * q) / p)

# transformada rapida de fourier
def fft(x):
    n = len(x)
    if n == 1:
        return x

    Feven = fft(x[0::2])
    Fodd = fft(x[1::2])
    combined = [0] * n

    for m in range(n//2):
        z = Feven[m] + omega(n, -m) * Fodd[m]
        p_r = round(z.real,4)
        p_c = round(z.imag,4)*1j
        combined[m] = p_r + p_c

        z = Feven[m] - omega(n, -m) * Fodd[m]
        p_r = round(z.real,4)
        p_c = round(z.imag,4)*1j
```

```
            combined[m + n//2] = p_r + p_c

    return combined
```

Pruebas

A continuación, se presentarán las pruebas con las mismas entradas que con la ejecución en java:

A [ ] = [4,2,0,1] = $4 + 2x + x^3$

```
Teclee el grado del polinomio A: 3
A0: 4
A1: 2
A2: 0
A3: 1
---------- Polinomio capturado ----------
A:  [4, 2, 0, 1]
---------- Transforada de Fourier ----------
T[A]:  [(7+0j), (4-1j), (1+0j), (4+1j)]
```

A[] = [1,1,0,0] = 1 + x

```
Teclee el grado del polinomio A: 1
A0: 1
A1: 1
---------- Polinomio capturado ----------
A:  [1, 1, 0, 0]
---------- Transforada de Fourier ----------
T[A]:  [(2+0j), (1-1j), 0j, (1+1j)]
```

A[ ] = [1,1,5,5] = $1 + x + 5x^2 + 5x^3$

```
Teclee el grado del polinomio A: 3
A0: 1
A1: 1
A2: 5
A3: 5
---------- Polinomio capturado ----------
A:  [1, 1, 5, 5]
---------- Transforada de Fourier ----------
T[A]:  [(12+0j), (-4+4j), 0j, (-4-4j)]
```

3. Usando la mejor de las implementaciones anteriores, diseña una función que calcule la suma de 2 polinomios de grado n. n>=2

```
# Sumar dos polinomios
def sumarPolinomios(A,B):
    grado_A = len(A)
    grado_B = len(B)
    resultado = []

    if(grado_A == grado_B):
        for i in range(0,grado_A):
            resultado.append(A[i]+B[i])
    elif(grado_A > grado_B): # El grado del polinomio A es mayor que el de B
        for i in range(0,grado_A):
            if i <= (grado_B - 1):
                resultado.append(A[i]+B[i])
            else:
                resultado.append(A[i])
    else: # El grado del polinomio B es mayor que el de A
        for i in range(0,grado_B):
            if i <= (grado_A - 1):
                resultado.append(A[i]+B[i])
            else:
                resultado.append(B[i])

    return resultado
```

Pruebas

A [ ] = [1,1,0,0] = $1 + x$
B [ ] = [1,0,1,0] = $1 + x^2$

```
Teclee el grado del polinomio A: 1
A0: 1
A1: 1
Teclee el grado del polinomio B: 2
B0: 1
B1: 0
B2: 1
---------- Polinomios capturados----------
A:  [1, 1, 0, 0]
B:  [1, 0, 1, 0]
---------- Suma de los polinomios ----------
[2, 1, 1, 0]
```

A [ ] = [1,1,0,0] = $1 + 2x + 3x^2 + 4x^3 + 5x^4 + 6x^5$
B [ ] = [1,0,1,0] = $6 + 5x + 4x^2 + 3x^3 + 2x^4 + x^5$

```
Teclee el grado del polinomio A: 5
A0: 1
A1: 2
A2: 3
A3: 4
A4: 5
A5: 6
Teclee el grado del polinomio B: 5
B0: 6
B1: 5
B2: 4
B3: 3
B4: 2
B5: 1

---------- Polinomios capturados----------
A:  [1, 2, 3, 4, 5, 6, 0, 0, 0, 0, 0]
B:  [6, 5, 4, 3, 2, 1, 0, 0, 0, 0, 0]
---------- Suma de los polinomios ----------
[7, 7, 7, 7, 7, 7, 0, 0, 0, 0, 0]
```

4. Repite el punto 3, pero ahora realiza la multiplicación. Considera que los polinomios pueden no sr del mismo grado.

```
# Multiplicacion de polinomios
'''
    ->Expresar cada uno de los números a multiplicar en base 10 y colocar los coeficientes de dicho
    polinomio en un vector.
    ->Completar, añadiendo ceros, el vector de coeficientes de cada uno de los números hasta la
    primera potencia de dos mayor o igual que 2n.
    ->Aplicar la FFT a cada vector obtenido de este modo.
    ->Multiplicar, coordenada a coordenada, los dos vectores obtenidos en el apartado anterior.
    ->Aplicar la FFT inversa al vector que resulta de la multiplicación anterior.
'''
def multiplicarPolinomios(A,B):
    fft_A = fft(A)
    fft_B = fft(B)
    tam_AB = len(fft_A)
    resultado = []
    for i in range(0,tam_AB):
        resultado.append(fft_A[i]*fft_B[i])
    return ifft(resultado)
```

Donde la funcion fft(A), se encuentra en el punto 1 y 2 del reporte, primeramente obtenemos la fft de cada uno de los polinomios, luego multiplicamos posicion a posicion las fft resultantes y finalmante a este ultimo paso le aplicamos transformada rapida inversa de furier (ifft) y dividimos entre el grado del polinomio resultante (es decir la suma del grado del polinomio A mas el grado del polinomio B), para ello la ifft, se implementa a continuacion:

```
# Transformada rapida de fourier inversa
''' Dado que la IFFT es la solucion, entonces basta con multiplicar los conjugados
    de las reaices n-esimas y dividir entre la suma de los grados mayores de los dos polinomios a
    sumar respectivamente '''
def ifft(X):
   x = fft([x.conjugate() for x in X])
   return [x.conjugate()/len(X) for x in x]
```

Para evitar Perdida de informacion y redundacia, la parte real y la parte imaginaria al calcular la fft, se redondea a 4 digitos

```
p_r = round(z.real,4)
p_c = round(z.imag,4)*1j
```

Tambien dado que el ejercicio nos pide que cuidemos que la multiplicacion se puede hacer entre polinomios de distintos grado, entonces despues de capturar los dos polinomios, nos encargamos de validar que los arreglos de coeficientes que le mandemos a multiplicar y a su vez a las trasnformadas sean los correctos, para ello, se codifico lo siguiente:

```
# Acondicionamos los polinomios para que tengan el mismo tamaño + 1
if len(A) == len(B):
    print("")
elif(len(A)<len(B)):
    for i in range(len(B)-len(A)):
        A.append(0)
else:
    for i in range(len(A)-len(B)):
        B.append(0)

diferiencia_grado = (grado_A + grado_B) - len(A) - 1
for i in range(diferiencia_grado):
    A.append(0)
    B.append(0)
```

Para que tengan el mismo tamaño, completamos con ceros de ser necesario.

Pruebas

A [ ] = [1,1,0,0] = $1 + x$
B [ ] = [1,0,1,0] = $1 + x^2$

```
--------------- Practica 3 - Divide y venceras ----------
Teclee el grado del polinomio A: 1
A0: 1
A1: 1
Teclee el grado del polinomio B: 2
B0: 1
B1: 0
B2: 1
---------- Polinomios capturados----------
A:  [1, 1, 0, 0]
B:  [1, 0, 1, 0]
---------- Multiplicacion de los polinomios ----------
[(1-0j), (1-0j), (1-0j), (1-0j)]
```

A [ ] = [1,3,0,0] = $1 + 3x$
B [ ] = [2,0,-1,0] = $2 - x^2$

```
--------------- Practica 3 - Divide y venceras ----------
Teclee el grado del polinomio A: 1
A0: 1
A1: 3
Teclee el grado del polinomio B: 2
B0: 2
B1: 0
B2: -1
---------- Polinomios capturados----------
A:  [1, 3, 0, 0]
B:  [2, 0, -1, 0]
---------- Multiplicacion de los polinomios ----------
[(2-0j), (6-0j), (-1+0j), (-3+0j)]
```

A [ ] = [1,1,1,1] = $1 + x + x^2 + x^3$
B [ ] = [1,1,1,0] $1 + x + x^2$

```
--------------- Practica 3 - Divide y venceras ---------------
Teclee el grado del polinomio A: 3
A0: 1
A1: 1
A2: 1
A3: 1
Teclee el grado del polinomio B: 2
B0: 1
B1: 1
B2: 1
---------- Polinomios capturados----------
A:  [1, 1, 1, 1, 0, 0]
B:  [1, 1, 1, 0, 0, 0]
---------- Multiplicacion de los polinomios ----------
[(2-0j), (2-0j), -0j, (2-0j), (2-0j), -0j]
```