

DALGO Proyecto - Problema A

Daniel Zambrano H. - 201914912 , Angel Restrepo - 201914073

12 de diciembre de 2021

Algoritmo de solucion

El algoritmo calcula la convolucion ponderada $cp(r,n)$, dados los datos A, B, C y n recibidos .

```
public class ProblemaA {  
  
    public static void main(String args[]) throws IOException  
    {  
        try (  
            InputStreamReader is= new InputStreamReader(System.in);  
            BufferedReader br = new BufferedReader(is);  
        ) {  
            String line = br.readLine();  
            ArrayList<Double> resultados = new ArrayList<Double>();  
  
            while(line!=null && line.length()>0)  
            {  
                String [] dataStr = line.split("\\s+");  
                if(dataStr.length == 1 && dataStr[0].equals("0") ) {  
                    break;  
                }  
                resultados= cp(dataStr,resultados);  
            }  
        }  
    }  
}
```

Figura 1

En esta primera parte del algoritmo se lee la primera linea de la entrada y tambien se inicializa un arreglo de resultados que va a almacenar todos los cp calculados. Luego, se inicia un ciclo que va a leer todas las lineas a las que se les debe calcular el cp, hasta que la linea sea un solo cero y se le hace una division por espacios (" ") a la linea para tener en un arreglo los datos ingresados por consola. Despues se manda el arreglo de la entrada y el arreglo de resultados al metodo cp que va a calcular la convolucion.

```

public static ArrayList<Double> cp(String[] division, ArrayList<Double> resultados) {
    int n = Integer.parseInt(division[0]);

    double a = Double.parseDouble(division[1]);

    double b = Double.parseDouble(division[2]);

    double c = Double.parseDouble(division[3]);

    double d = Double.parseDouble(division[4]);

    double[] r = new double[n+1];

    Double cp = 0.0000;

    r[0] = a;
    r[1] = b;

    for(int i=0; i<(r.length-2);i++) {
        r[i+2] = c*r[i] + d*r[i+1];
    }
}

```

Figura 2

En esta segunda parte del algoritmo se le asignan los valores de n,A,B,C,D a sus variables respectivas y comienza el calculo de r teniendo en cuenta que:

$$\begin{aligned}
 r(0) &= A \\
 r(1) &= B \\
 r(k+2) &= C*r(k) + D*r(k+1), \quad k \geq 0
 \end{aligned}$$

Figura 3

Y almacenando cada resultado en el arreglo r.

```
for(int i=0; i<r.length ; i++) {  
    cp = cp + r[i]*r[n-i];  
}  
  
cp = Math.ceil(cp * 10000.0) / 10000.0;  
  
resultados.add(cp);  
return resultados;  
}
```

Figura 4

Teniendo ya todos los r calculados se procede a hacer la sumatoria del cp en un for usando la respectiva formula. Despues de calcular cp, se le hace un redondeo a 4 cifras decimales y se agrega a la lista de cp calculados hasta el momento y se retorna el arreglo.

```
    line=br.readLine();  
}  
  
for(int i=0; i<resultados.size(); i++) {  
    System.out.println(resultados.get(i));  
}  
}  
}
```

Figura 5

Por ultimo se lee la siguiente linea ingresada por consola y se divide por espacios igual que con la linea anterior. El algoritmo finaliza imprimiendo todos los cp almacenados en el arreglo.

Este algoritmo se realizo de esta forma porque era la manera mas optima de realizar los calculos y de almacenar cada dato, calculando los r y el cp apenas se van recibiendo los datos para cada linea ingresada con sus respectivas formulas dadas en el enunciado

Precondicion:

$$Q : \{A \in \mathbb{R} \geq 0, B \in \mathbb{R} \geq 0, C \in \mathbb{R} \geq 0, D \in \mathbb{R} \geq 0, n \in \mathbb{NAT} \geq 0\}$$

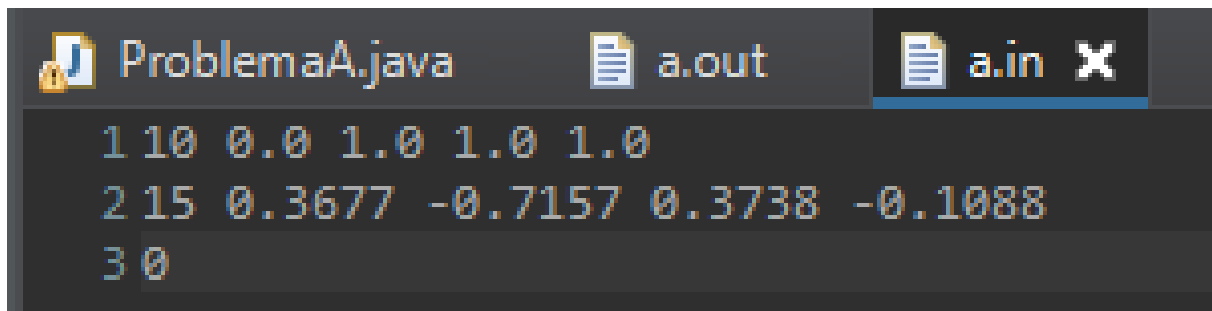
Postcondicion:

$$S : \{cp = (+k | 0 \leq k \leq n : r(k) * r(n - k)), n \geq 0\}$$

Análisis de complejidad espacial y temporal

Complejidad temporal:

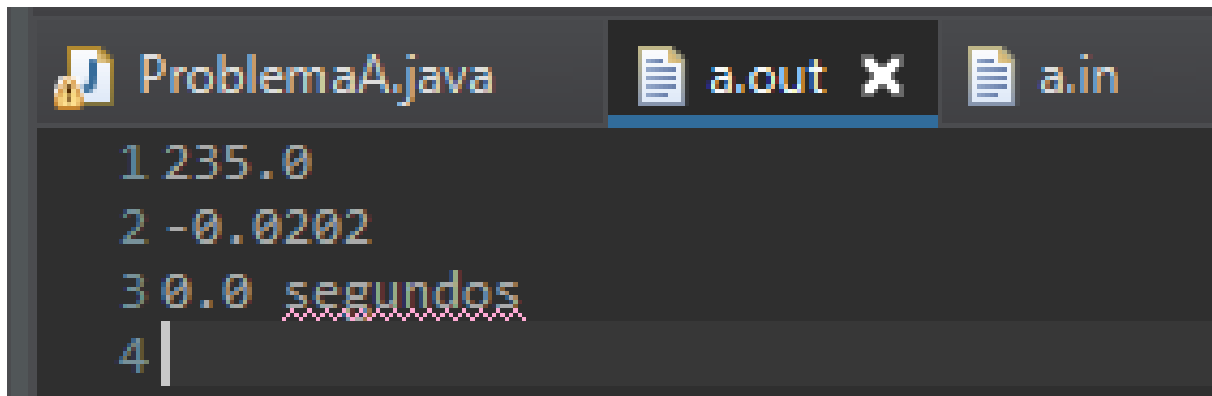
Se realizó el siguiente caso de prueba dado en el enunciado:



```
ProblemaA.java a.out a.in X
1 10 0.0 1.0 1.0 1.0
2 15 0.3677 -0.7157 0.3738 -0.1088
3 0
```

Figura 6

Y dando como resultado:

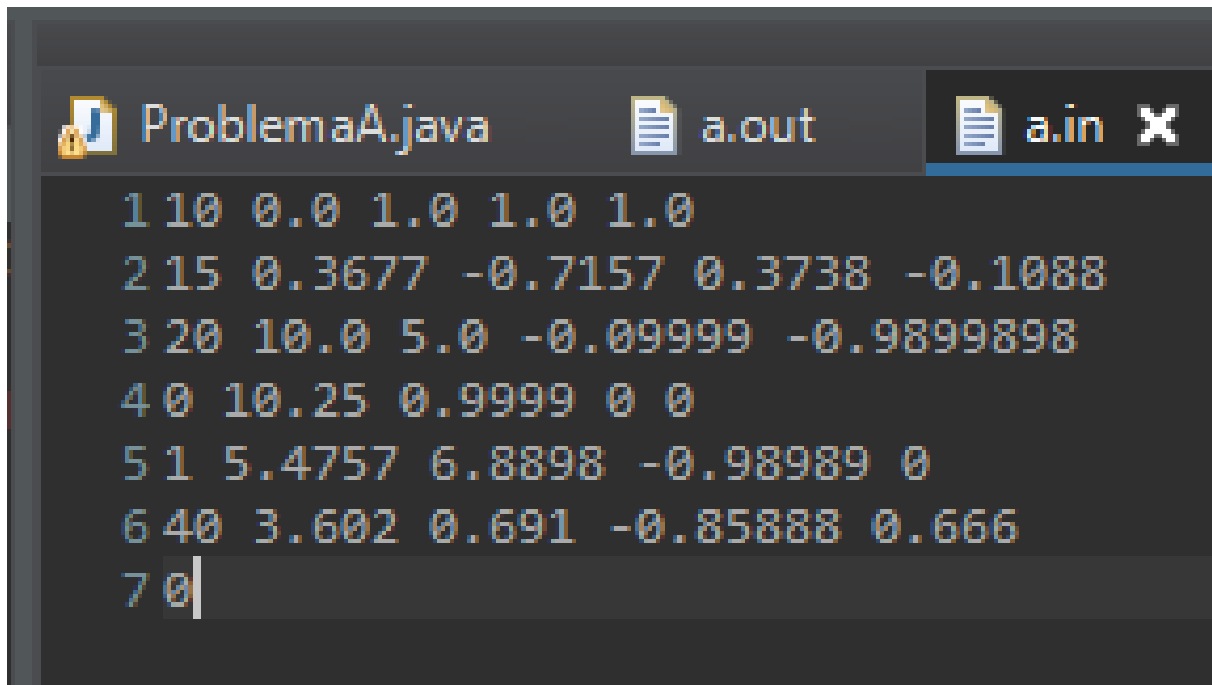


```
ProblemaA.java a.out X a.in
1 235.0
2 -0.0202
3 0.0 segundos
4 |
```

Figura 7

Como se puede ver en la imagen da el resultado esperado y muestra un tiempo de ejecución de 0.0 segundos. Volviendo a ejecutar el programa con el mismo caso se obtuvo un promedio de 0.0 segundos

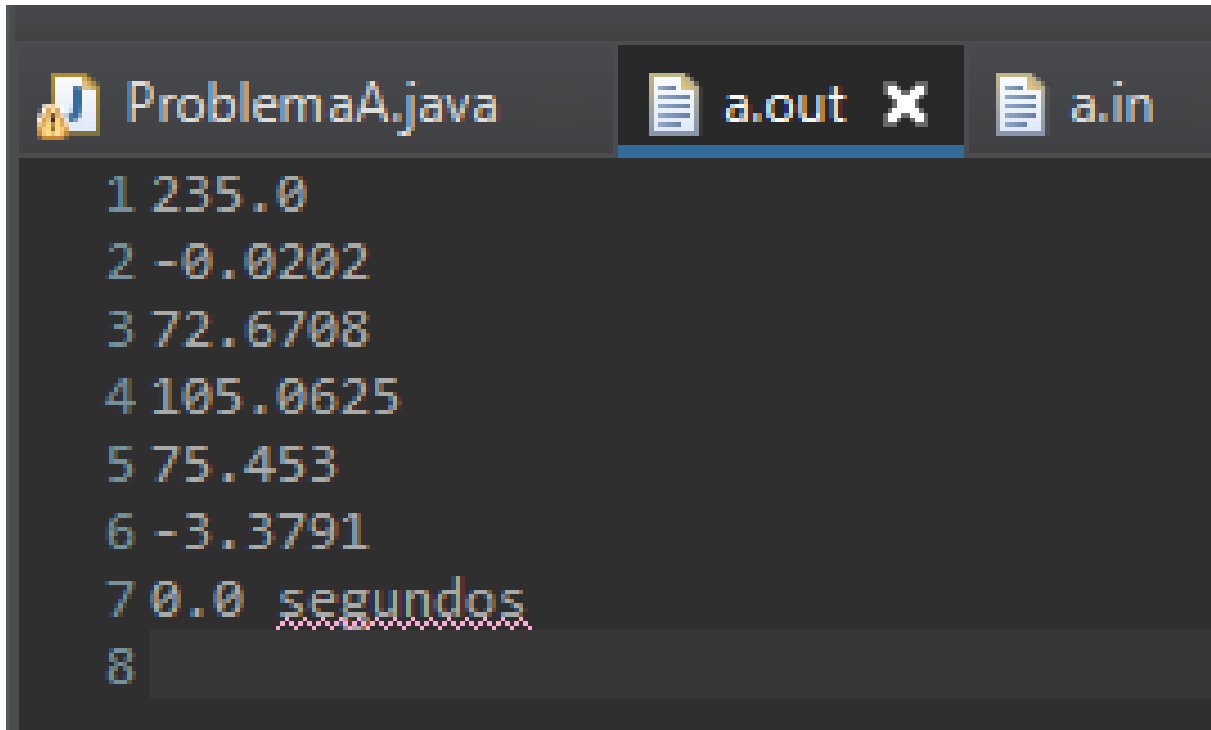
También se probó el algoritmo haciendo cambios al caso añadiendo más datos y haciendo el cálculo manual de los resultados esperados.



```
ProblemaA.java  a.out  a.in X
1 10 0.0 1.0 1.0 1.0
2 15 0.3677 -0.7157 0.3738 -0.1088
3 20 10.0 5.0 -0.09999 -0.9899898
4 0 10.25 0.9999 0 0
5 1 5.4757 6.8898 -0.98989 0
6 40 3.602 0.691 -0.85888 0.666
7 0
```

Figura 8

dando como resultado:



```
ProblemaA.java  a.out  a.in
1 235.0
2 -0.0202
3 72.6708
4 105.0625
5 75.453
6 -3.3791
7 0.0 segundos
8
```

Figura 9

Comentarios finales

Esta solución muestra un desempeño de tiempo excelente y con los resultados esperados al correr la solución con diferentes casos de diferentes tamaños.