0 Identificación:

Juan Camilo Ruiz

Sergio Guzmán

1 Algoritmo de solución

Se decidió usar programación dinámica con el fin de priorizar el espacio, de este modo se usa solamente un arreglo de tamaño N+1, siendo N el tiempo dado en las entradas del problema. Así, se efectúa un ciclo en el cual se calculan todos los elementos del arreglo, posteriormente, se calculan la mitad de los valores de CP para multiplicarlo por 2, debido a que al calcular la multiplicación de K[i] con K[j], siendo inicialmente i = 0 y j = N, e ir aumento i y disminuyendo j, eventualmente, cuando i>j, se realizarán las mismas operaciones que se efectuaron hasta el momento. Por tanto, es posible calcular CP al calcular CP hasta el punto en que i>j, y luego multiplicarlo por 2. Finalmente, una vez efectuado el ciclo para calcular todos los valores de r y luego calcular CP hasta el punto en que i>j, se debe verificar si N es par o impar, debido a que en caso de que sea impar, se tiene que eliminar el valor de la multiplicación cuando i=j, ya que este solo debería ocurrir una vez y al ser par, ocurre dos veces. Por último, se usa una función redondear() para redondear los 4 decimales de los números.

Para este grafo se utilizaron las siguientes funciones matemáticas de verificación del programa:

r(k+2) = C\*r(k) + D\*r(k+1), k≥0

cp(r,n) = (+k| 0≤k≤n: r(k)\*r(n-k)), n≥0.

Anotaciones:

**public** **class** ProblemaA {

**public** **double** problema1 (**int** n,**double** A, **double** B,**double** C,**double** D) {

//Q: true

**double** [] reserva = **new** **double** [n+1];

reserva[0] = A;

reserva[1] = B;

//P1: 2 <= i <= N ^ (ForAll k| 0 <= k <= i : reserva[k] = r(k))

//t1: N - i

**for** (**int** i = 2; i <= n; i++) {

reserva[i] = C\*reserva[i-2] + D\*reserva[i-1];

}

//R1: ForAll k| 0 <= k <= N : reserva[k] = r(k)

**double** cp = 0;

//P2: 0 <= i <= N/2 ^ cp = cp(r,i)

//t2: N/2 - i

**for** (**int** i=0; i <= n-i; i++) {

cp += reserva[i]\*reserva[n-i];

}

//R2: cp = cp(r, n/2)

cp = cp\*2;

//Q2: cp = cp(r,N) + max(0,((n+1)%2)\*(N/2)^2)

**if** (n%2==0) {

cp -= reserva[n/2]\*reserva[n/2];

}

//R3: cp = cp(r,N)

**return** redondear(cp);

}

**public** **double** redondear(**double** numero) {

BigDecimal bd = **new** BigDecimal(numero);

bd = bd.setScale(4, RoundingMode.***HALF\_UP***);

**return** bd.doubleValue();

}

**public** **static** **void** main(String[] args) **throws** Exception{

//Instancia del problema

ProblemaA pd = **new** ProblemaA();

ArrayList<Double> resp = **new** ArrayList<>();

**try** (

InputStreamReader is= **new** InputStreamReader(System.***in***);

BufferedReader br = **new** BufferedReader(is);

) {

String line = br.readLine();

**while**(line!=**null** && line.length()>0) {

String valores [] = line.split(" ");

**if** (valores.length != 5) **throw** **new** Exception("Valores ingresados errados");

resp.add(pd.problema1(Integer.*parseInt*(valores[0]), Double.*parseDouble*(valores[1]), Double.*parseDouble*(valores[2]), Double.*parseDouble*(valores[3]), Double.*parseDouble*(valores[4])));

line=br.readLine();

}

//If no one is typing anymore end the algorithm and show the answers

**for**(Double r:resp)

System.***out***.println(r);

}

}

}

2 Análisis de complejidades espacial y temporal

La complejidad espacial es de O(N+1), ya que el único espacio que se usa es un arreglo de espacio N+1.

La complejidad temporal es de O(N+N/2), ya que en el primer ciclo se calculan todos los elementos de r, los cuales son N elementos. Por otro lado, el ciclo para calcular CP, es efectuado N/2 veces, ya que solo calcula la mitad de los valores de CP, que son N/2, debido a que en vez de usar dos variables i y j, se usa i y N-i, de modo que es posible calcular CP/2 en un ciclo que se efectúa N/2 veces.

3. Comentarios finales

La solución cumple con priorizar la complejidad espacial, además la complejidad temporal es lo suficientemente efectiva.