**Práctica Final Metodología de la Programación y Algoritmia**

****

**Trabajo realizado por:**

**Sergio Sanchiz Villena -** [**sergio.sanchiz@goumh.umh.es**](mailto:sergio.sanchiz@goumh.umh.es) **- 74440566P**

**Lucas Borges de Castro -** [**lucas.borges@goumh.umh.es**](mailto:lucas.borges@goumh.umh.es) **- 58487778C**

**Pablo Casimiro Beneitez -** [**pablo.casimiro@goumh.umh.es**](mailto:pablo.casimiro@goumh.umh.es) **- 46085962B**

[**1.-Algoritmo Recursivo. 2**](#_8nh4ojv7sbjk)

[1.1.- Descripción del algoritmo. 2](#_bfkhnnma8rck)

[1.2.- Estrategia de programación y tipo de recursividad del algoritmo. 2](#_ksp5cp5nan4e)

[1.3 .- Pseudocódigo. 2](#_2quismcwuvdq)

[1.4. Ejemplo. 2](#_qjq239bk0l0a)

[1.5.- Complejidad asintótica. 3](#_a34cmdrm7crl)

[**2.- Algoritmo Iterativo 4**](#_fk67xqtb1hfu)

[2.1.- Descripción del algoritmo. 4](#_w2nwf4w4fp9h)

[2.2.- Estrategia de programación. 4](#_6bam04auw3qy)

[2.3.- Pseudocódigo. 4](#_4e53ar51pc5u)

[2.4.- Ejemplo. 5](#_k43odgy0vsvd)

[2.5.- Complejidad Asintótica. 5](#_2jiby0jl0kw2)

[**3.- Bibliografía 6**](#_wd0xzl7txor)

# **1.-Algoritmo Recursivo.**

## **1.1.- Descripción del algoritmo.**

Dada una posición de inicio, una posición destino, y las dimensiones (filas y columnas) de un plano, el algoritmo hace por cada “casilla”, todo el rango de movimientos posibles por el robot (abajo, derecha y diagonal) siempre y cuando estos no se salgan del plano.

Si se alcanza un caso base, es decir, que la posición sea un límite del plano o la posición destino, el algoritmo devuelve el valor de 0 o 1 respectivamente. De tal forma que, el algoritmo se va llamando a sí mismo, probando todas las combinaciones de movimientos posibles, si se alcanza un borde del plano, el algoritmo suma 0. Si se alcanza la posición destino, el algoritmo suma 1. Una vez probadas todas las combinaciones posibles, la suma de todos los “1” será el total de caminos encontrados.

## **1.2.- Estrategia de programación y tipo de recursividad del algoritmo.**

El tipo de recursividad de este algoritmo se trata de una “Recursividad Lineal: No Final”. Puesto que, al terminar la recursividad, quedan operaciones pendientes por realizarse, siendo dicha operación, la suma de los caminos hallados.

## **1.3 .- Pseudocódigo.**

función robot\_recursivo(x1:natural, y:natural, x2:natural, y2:natural, filas:natural, columnas:natural)

Si (x1 = x2 Y y = y2)

devolver 1

Si (x1 >= filas O y >= columnas)

devolver 0

Si no

devolver robot\_recursivo( x1+1, y+1, x2, y2, filas, columnas) + robot\_recursivo(x1+1, y, x2, y2, filas, columnas)+robot\_recursivo(x1, y+1, x2, y2, filas, columnas)

fSi

ffunción

## **1.4. Ejemplo.**

Para el comienzo del algoritmo tenemos un plano de f \* c con punto inicial (x,y) y punto final (x1,y1).

· Con la llamada a la función se guardan los datos guardados.

robot\_recursivo(x, y, x1, y1, f, c)

· Al principio de la función se comprueba si está en el punto final.

¿x = x1 Y y = y1?

Si no se continúa con el algoritmo.

· Después se comprueba si se ha pasado de los límites del plano.

¿x => f o y => c?

Si no se continúa con el algoritmo.

· Si no se ha dado ninguno de los dos casos anteriores se mueve el robot de una de las tres maneras posibles y se vuelve al inicio del algoritmo con las nuevas posiciones.

- Derecha: robot\_recursivo(x + 1, y, x1, y1, f, c)

- Abajo: robot\_recursivo((x, y + 1, x1, y1, f, c)

- Diagonal: robot\_recursivo(x + 1, y + 1, x1, y1, f, c)

En el caso de llegar al punto final se habrá repetido una cantidad z de veces(dependiendo del camino recorrido), y al llegar a la comprobación del punto final y ver que coincide se contará el camino seguido como camino válido y se repetirá el algoritmo con los puntos iniciales de partida

En caso de no llegar al punto final y pasarse de los límites del plano el algoritmo se habrá repetido z veces (depende del camino recorrido) y al comprobar si ha llegado a los límites del plano no se contará el camino recorrido y se volverá al principio del algoritmo con los puntos iniciales de partida.

(Las veces que se repita el algoritmo dependerá del tamaño del plano y el punto de partida de nuestro robot)

## **1.5.- Complejidad asintótica.**

función robot\_recursivo(x1:natural, y:natural, x2:natural, y2:natural, filas:natural, columnas:natural)

Si (x1 = x2 Y y = y2) || t(1)= 2

devolver 1 || t(2) = 1

Si (x1 >= filas O y >= columnas) || t(3) = 2

devolver 0 || t(4) = 1

Si no ||

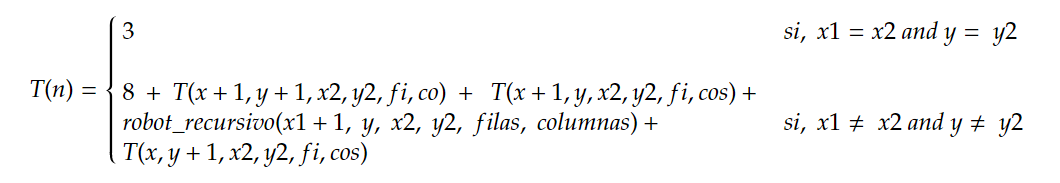
devolver robot\_recursivo( x1+1, y+1, x2, y2, filas, columnas) + || t(5) = 8

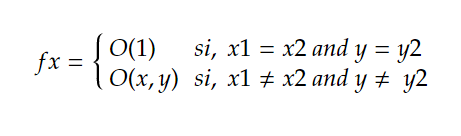
robot\_recursivo(x1+1, y, x2, y2, filas, columnas)+ || + T(x+1,y+1,x2,y2,fi,co)

robot\_recursivo(x1, y+1, x2, y2, filas, columnas) || + T(x+1,y,x2,y2,fi,cos)+

fSi || + T(x,y+1,x2,y2,fi,cos)

ffunción ||

****

****

# **2.- Algoritmo Iterativo**

## **2.1.- Descripción del algoritmo.**

Este algoritmo resuelve el problema de encontrar el número de caminos posibles desde un punto de partida hasta un punto en una matriz bidimensional de tamaño n\_filas x n\_columnas. El algoritmo utiliza una matriz bidimensional para almacenar información sobre los caminos posibles.

Inicialmente, todos los elementos de la matriz se inicializan a cero, excepto el punto inicial (x1, y) que se establece en 1 para indicar el punto de partida. A continuación, el algoritmo recorre iterativamente la matriz desde el punto inicial hasta el punto final. Para cada posición en la matriz, se realizan tres movimientos posibles: hacia abajo, hacia la derecha y en diagonal hacia abajo y derecha, incrementando los valores de las posiciones correspondientes en la matriz.

El algoritmo realiza estos movimientos de manera iterativa para cada posición de la matriz, asegurándose de no exceder los límites de las filas y columnas de la matriz.

## **2.2.- Estrategia de programación.**

Programación dinámica.

En lugar de mantener un seguimiento de todos los caminos posibles, el algoritmo utiliza una matriz bidimensional para almacenar y actualizar sólo los valores necesarios. Esta opción es la mejor para este problema puesto que resuelve el subproblema una vez, y lo almacena en una estructura (matriz) la solución más óptima

## **2.3.- Pseudocódigo.**

función robot\_iterativo ( x1:natural, y:natural, x2:natural, y2:natural, filas:natural, columnas:natural )

i, j, x, num\_caminos:natural

matriz:entero [filas]

Para i = 0 hasta filas - 1 hacer

matriz[filas][columnas]

Para j = 0 hasta columnas - 1 hacer

matriz [i][j] = 0

fPara

fPara

matriz [x1][y] = 0

Para x = x1 hasta x2 hacer

Para i = y hasta y2 hacer

Si(x < filas - 1)

matriz [x + 1][i] = matriz [x + 1][i] + matriz [x][i]

Si(i < columnas - 1)

matriz [x][i + 1] = matriz [x][i + 1] + matriz [x][i]

Si (x < filas - 1 Y i < columnas - 1)

matriz [x + 1][i + 1] = matriz [x + 1][i + 1] + matriz [x][i]

fsi

fPara

fPara

num\_caminos = matriz [x2][y2]

Para i = 0 hasta filas - 1 hacer

delete[] matriz[i]

fPara

delete[] matriz

devolver num\_caminos

ffunción

## **2.4.- Ejemplo.**

Para el comienzo del algoritmo tenemos un plano de f \* c con punto inicial (x,y) y punto final (x1,y1).

· Con la llamada al algoritmo introduciremos los datos elegido:

robot\_iterativo(x, y, x1, y1, f, c)

· La siguiente función de nuestro algoritmo es crear nuestra matriz con los puntos a seguir a 0 y luego colocamos nuestro punto inicial con valor de 1

matriz [filas, columnas]

matriz [x][y] = 1

· Una vez que tenemos nuestra matriz con el punto de inicio iremos comprobando en bucles si todos los caminos que puede hacer nuestro robot desde el inicio para contar los válidos y poder mostrarlos al usuario. En los casos de no llegar al punto final se mantendrá dentro del bucle y se repetirá el proceso desde el punto de inicio original.

· Una vez que tenga todos los caminos posibles los guardará en un contador y después vaciaremos nuestra matriz por completo para futuras llamadas al algoritmo.

## **2.5.- Complejidad Asintótica.**

función robot\_iterativo ( x1:natural, y:natural, x2:natural, y2:natural, filas:natural, columnas:natural )

i, j, x, num\_caminos:natural || t(1)= 1

matriz:entero [filas] || t(2)= 2

Para i = 0 hasta filas - 1 hacer || t(3)= 2\*filas

matriz[filas][columnas] || t(4)= 2

Para j = 0 hasta columnas - 1 hacer || t(5)= 2\*columnas

matriz [i][j] = 0 || t(6)= 2

fPara

fPara

matriz [x1][y] = 0 || t(9)= 2

Para x = x1 hasta x2 hacer || t(10)= x2-x1+1

Para i = y hasta y2 hacer || t=(y2-y+1)\*(x2-x1+1)

Si(x < filas - 1) || t(12)=2

matriz [x + 1][i] = matriz [x + 1][i] + matriz [x][i] || t(13)=5

Si(i < columnas - 1) || t(14)=2

matriz [x][i + 1] = matriz [x][i + 1] + matriz [x][i] || t(15)=5

Si (x < filas - 1 Y i < columnas - 1) || t(16)=4

matriz [x + 1][i + 1] = matriz [x + 1][i + 1] + matriz [x][i] || t(17)=5

fsi

fPara

fPara

num\_caminos = matriz [x2][y2] || t(21)=2

Para i = 0 hasta filas - 1 hacer || t(22)=2\*filas

delete[] matriz[i] || t(23)=2

fPara

delete[] matriz || t(24)=2

devolver num\_caminos || t(25)=1

ffunción



Notación Asintótica: O(y2x2 - yx1)

# **3.- Bibliografía**

- Práctica 4 Ejercicio 4.1: Recursividad y su transformación a iterativo.

Metodología de la Programación y Algoritmia (UMH).

Curso 2022/2023.

- Apuntes del Tema 5: Programación dinámica. Diapositivas 10 a la 17.

Metodología de la Programación y Algoritmia (UMH)

Curso 2022/2023.

- Título: Programación dinámica: qué es, cómo funciona y recursos de aprendizaje.

Url: https://geekflare.com/es/dynamic-programming