

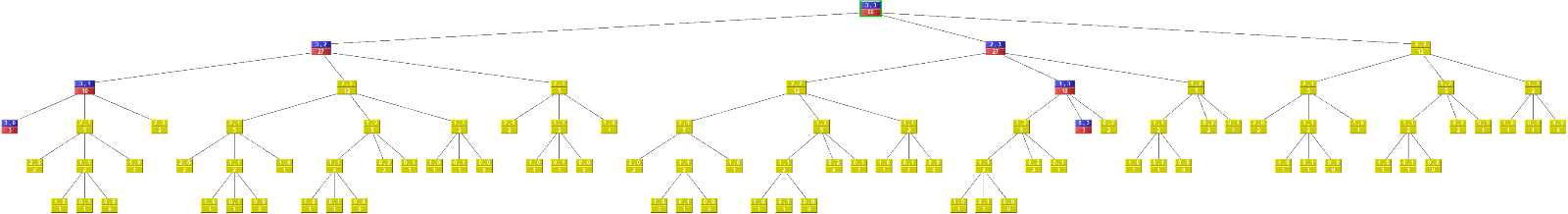
# Análisis de algoritmos: Practica 4.

# **Eliminación de la recursividad redundante.**

Daniel Lois Nuevo

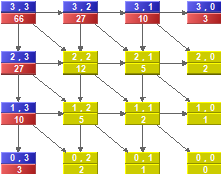
Adrián García Oller

1. **Análisis de la redundancia.**
2. Árbol recursivo.

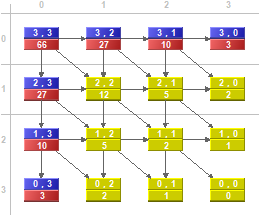


Con x=3 e y=3.Las llamadas marcadas en amarillos son las que se repiten y las no marcadas son las que son llamadas una única vez.

1. Grafo de dependencia.



1. **Diseño de la tabla.**
2. **Figura de la tabla.**



1. Declaración de la tabla. Como podemos observar en la figura anterior debemos usar una tabla con x+1 filas y y+1 columnas.

public static int fT (int x, int y) {

int[][] tabla=new int[x+1][y+1];

for(int i=0;i<x+1;i++) {

for(int j=0;j<y+1;j++) {

tabla[i][j]=-1;

}

}

frecT(x, y, tabla);

return tabla[x][y];

}

1. **Memorización.**
2. **Código.**

**public static void frecT (int x, int y, int[][] tabla) {**

**if(tabla[x][y]==-1) {**

**if (x==0)**

**tabla[x][y]=y;**

**else if (y==0)**

**tabla[x][y]=x;**

**else {**

**frecT(x,y-1, tabla);**

**frecT(x-1,y, tabla);**

**frecT(x-1,y-1, tabla);**

**tabla[x][y]=tabla[x][y-1]+ tabla[x-1][y] +tabla[x-1][y-1] ;**

**}**

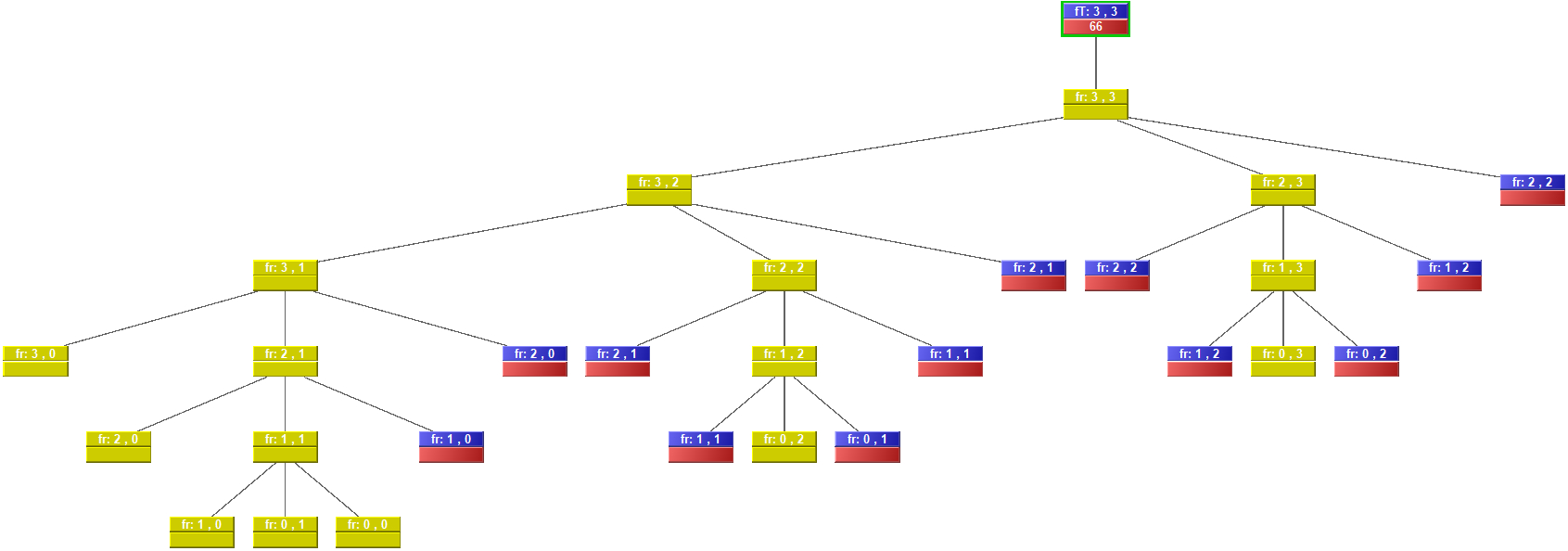
**}**

**}**

Como se puede observar en el código, una vez creada tabla se llama a este algoritmo de memorización el cual comprueba primero si esa llamada ya tiene una solución en la tabla, si no la tiene la calcula haciendo las llamadas correspondientes y se le asigna a la celda correspondiente, en el caso de que x o y sean cero se le añade a la celda de esa llamada el parámetro contrario.

1. **Árbol de recursión.**

Con x=3 e y=3.Las llamadas marcadas en amarillo son las nuevas y las no marcadas son las repetidas.



1. **Análisis de su complejidad en tiempo y en espacio.**

Como el número de llamadas nuevas es ((x+1) \*(y+1) +1) y el número de llamadas repetidas es ((2x-1) \*y-1) +x-1):

T(x,y)= ((x+1)\*(y+1)+1) +((2x-1)\*y-1)+x-1), por lo que por lo que T(x,y)=Ɵ(x\*y).

Como el tamaño de la tabla es (x+1) \* (y+1) y la rama más larga del árbol es x + y:

M(x,y)= ((x+1) \* (y+1)) + (x + y) , por lo que M(x,y)=Ɵ(x\*y).

1. **Tabulación.**
2. **Código**

public static int fTab(int x, int y) {

int [][] tab = new int[x+1][y+1];

for (int i = 0; i < x+1;i++) {

tab[i][0] = i;

}

for (int i = 0; i < y+1;i++) {

tab[0][i] = i;

}

for(int i=1;i<x+1;i++) {

for(int j=1;j<y+1;j++) {

tab[i][j]=tab[i][j-1]+tab[i-1][j]+tab[i-1][j-1];

}

}

return tab[x][y];

}

En este código aplicamos el método de tabulación para eliminar la recursividad redundante. Para ello nos ayudamos de una tabla en la cual iniciamos los valores de las entradas que sustituirán los valores de los casos base del algoritmo recursivo. Eso lo hacemos con los dos primeros bucles for.

Para calcular el resultado del algoritmo vamos actualizando todos los valores de la tabla con un bucle for anidado hasta llegar a la entrada (x,y) y la devolvemos.

1. **Complejidad en tiempo y espacio:**

La complejidad de este algoritmo es la siguiente:

El orden de complejidad del primer for es Ɵ(Y), mientras que en el segundo tenemos Ɵ(Y).

Con la cual al final del segundo for tenemos una complejidad del orden Ɵ(X+Y).

El bucle for anidado tiene una complejidad del orden Ɵ(X\*Y) lo cual nos deja una complejidad total del algoritmo del orden Ɵ(X+Y+X\*Y) que resumiendo nos dejaría la complejidad del orden Ɵ(X\*Y) al igual que el algoritmo de Memorización.

}

1. **Conclusiones.**

Esta práctica nos ayuda a profundizar más en la recursividad. Gracias a que profundizamos en las dos técnicas vistas en clase para eliminar la recursividad redundante, Tabulación y Memorización.

Al ser un algoritmo recursivo sencillo permite familiarizarse con estas técnicas con mayor facilidad, aun así hemos encontrado dificultades a la hora de examinar el árbol de recursión y observar el comportamiento del algoritmo, ya que era la primera vez que lo hacíamos.

Como conclusión nos parece una buena práctica para afianzar los conocimientos de eliminación de recursividad redundante que hemos obtenido en clase, ya que creemos que la dificultad es la adecuada para el nivel que tenemos.