

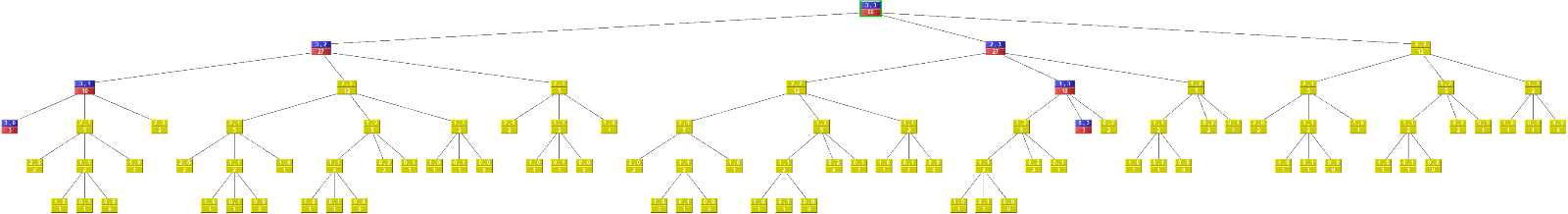
# Análisis de algoritmos: Practica 4.

# **Eliminación de la recursividad redundante.**

Daniel Lois Nuevo

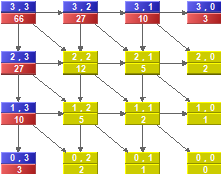
Adrián García Oller

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

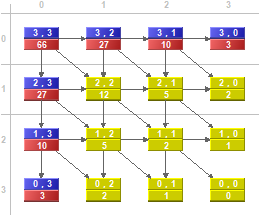


Con x=3 e y=3.Las llamadas marcadas en amarillos son las que se repiten y las no marcadas son las que son llamadas una única vez.

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] ;**

**}**

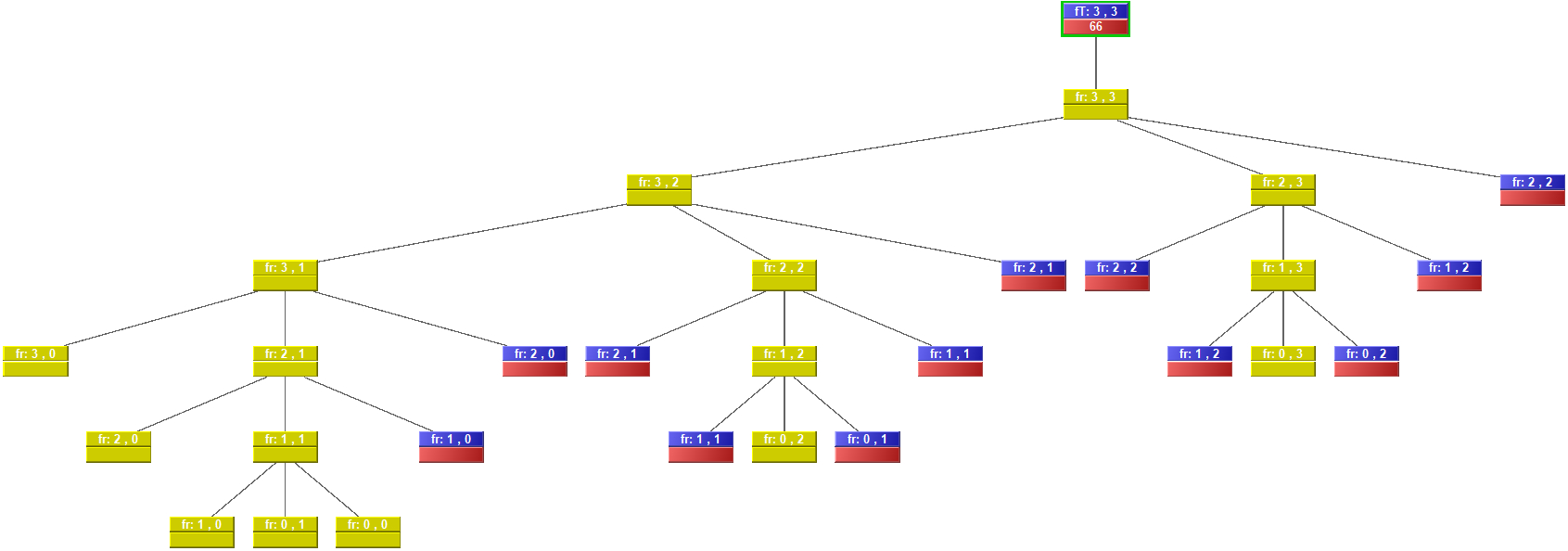
**}**

**}**

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. **Conclusiones.**

Esta práctica ayuda a comprender los algoritmos de ramifica y poda y a practicar con herramientas de estudio de algoritmos. En cuanto a la dificultad de la práctica creemos que tiene una complejidad notablemente mayor a la anterior por la complejidad en la decisión del tipo de función de cota a emplear. Gracias a optimex hemos podido comprobar que tanto el backtracking como el ramifica y poda son exactos. Nos costó entender el funcionamiento de la función de cota para minimización.