

**PRÁCTICA 4:**

**ELIMINACIÓN DE LA RECURSIVIDAD Y PROGRAMACIÓN DINÁMICA**

**Algoritmos Avanzados. Grado en Ingeniería Informática**



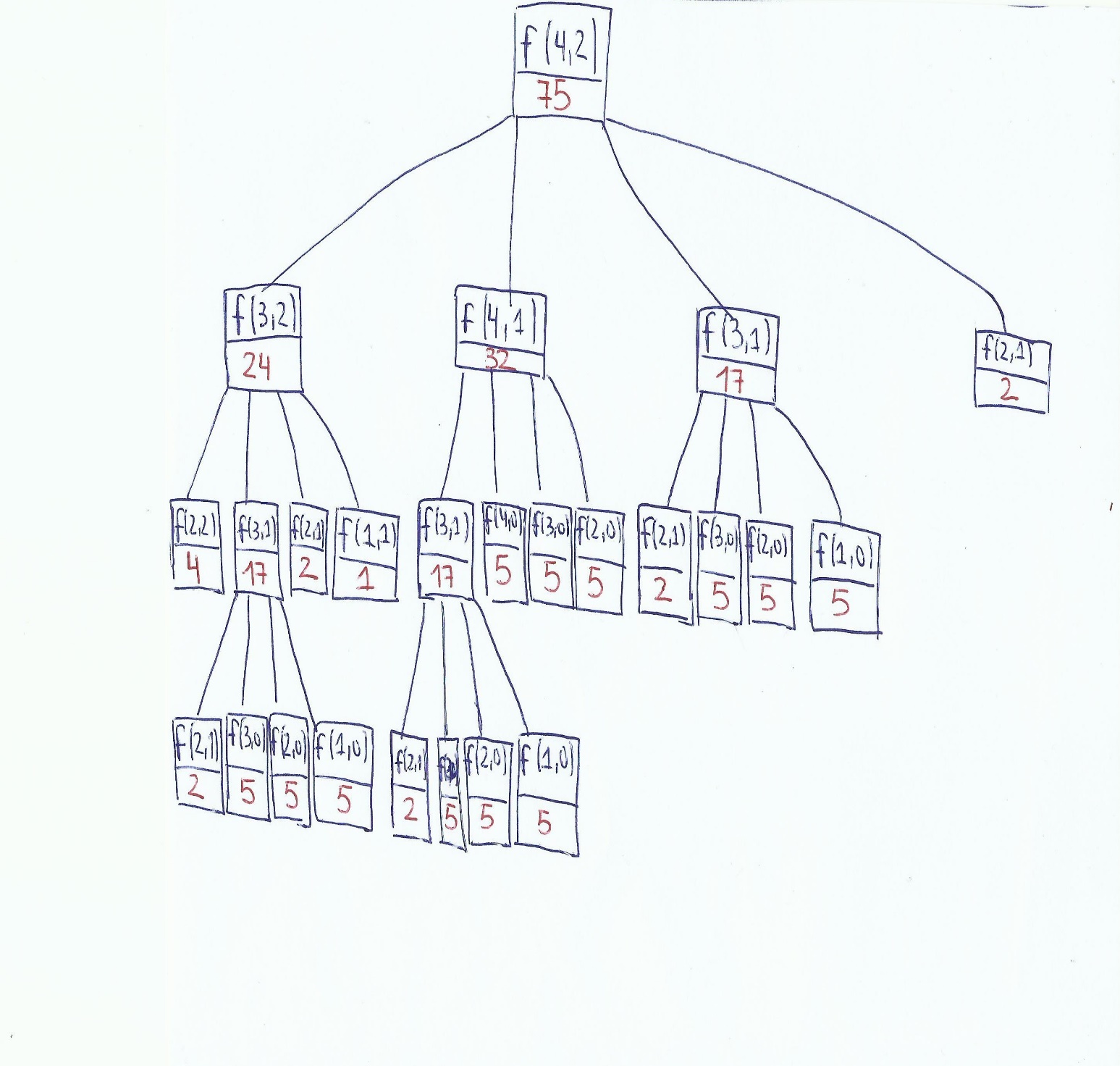
11 de DICiembre de 2017

JOSE VICENTE BAÑULS GARCÍA

JORGE ARANDA GARCÍA

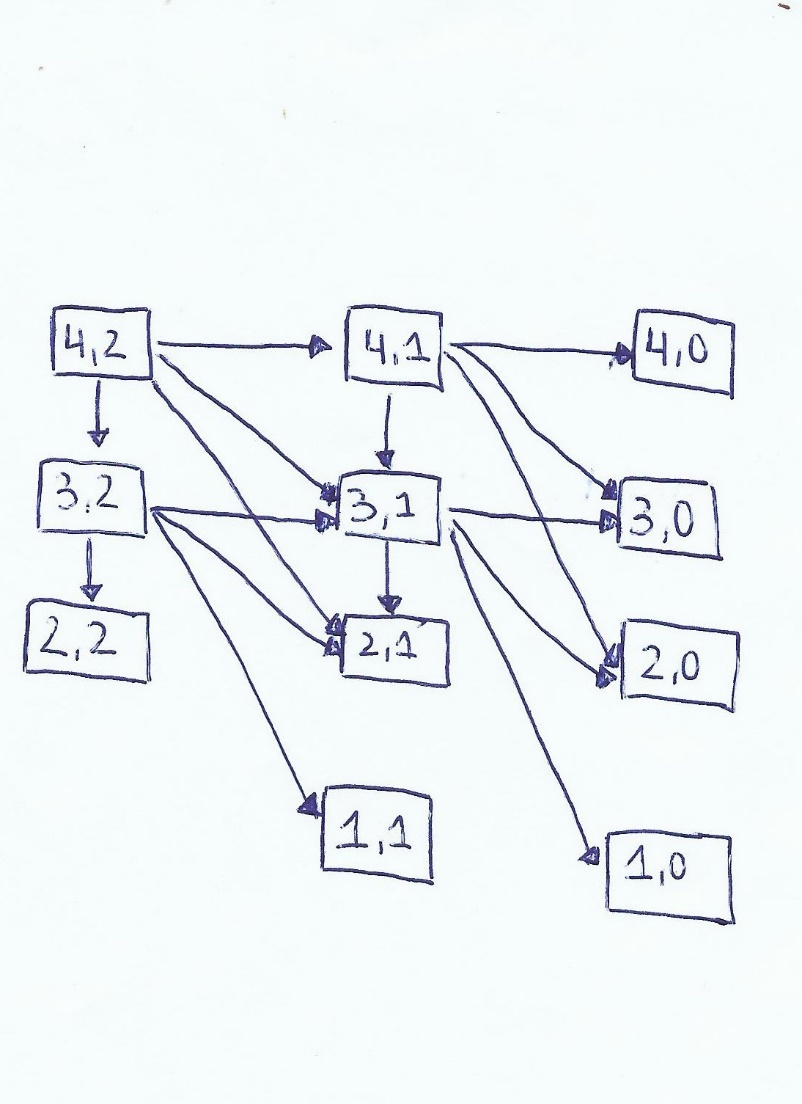
# ELIMINACIÓN DE LA RECURSIVIDAD

## Dibujar y analizar el árbol de recursión para la llamada f(4,2). Añadir la solución para cada subproblema.



En cada uno de los nodos, se muestra la llamada recursiva que se hace en azul, y en rojo se muestra el resultado de dicha llamada. Si un nodo en concreto no tiene hijos, es porque las condiciones de la llamada recursiva se corresponden con alguno de los casos base, bien multiplicar x\*y si x<=2, y>0, bien 5, si y<0.

## Dibujar y analizar el grafo de dependencia asociado al árbol anterior



Como se puede observar las llamadas que no son casos base, es decir cuando la x es menor que 2 o la y es 0, dependen siempre de otras cuatro posiciones de la tabla, que se corresponden con las llamadas recursivas.

f(x,y)= f(x-1,y) + f(x,y-1) + f(x-1,y-1) + f(x-2,y-1).

Por ejemplo : f(3,2) = f(2,2)+ f(3,1)+ f(2,1)+ f(1,1).

## Dibujar la tabla asociada especificando las dimesiones de la misma y los subproblemas que se almacenan en cada celda de la tabla

|  |  |  |
| --- | --- | --- |
| f(1,0)  5 | f(1,1)  1 |  |
| f(2,0)  5 | f(2,1)  2 | f(2,2)  4 |
| f(3,0)  5 | f(3,1)  17 | f(3,2)  24 |
| f(4,0)  5 | f(4,1)  32 | f(4,2)  75 |

El tamaño de la tabla es de **m \* n+1**. Cada posición de la tabla almacena el resultado de la llamada justo encima. Si es un caso base se rellena sin más y si no, se rellena con el resultado de las posiciones que le sean necesarias. Cabe destacar que la tercera posición de la primera fila no está rellena porque para el caso que nos concierne no es necesaria.

# PROGRAMACIÓN DINÁMICA

## Definición de la función global.

Beneficio(t) = Cantidad máxima de beneficio que se puede obtener al invertir durante t meses.

## Demostración del principio de optimalidad.

En el instante i la cantidad se ha escogido de forma óptima puesto que en todas las etapas anteriores se ha escogido de forma óptima.

## Definición de los casos base.

Beneficio(0) = C, donde C es la cantidad de la que disponemos.

## Definición de la función recursiva.

Beneficio(i) =

máx {beneficio (i-1), (beneficio (i-1) – GCD[i-1]) \* RCD[i-1]}

si 1 <= i < 6

máx {beneficio(i-1), (beneficio(i-1)-GCD[i-1]) \* RCD[i-1], (beneficio(i-6)-GBT[i-6]) \* RBT[i-6]}

si i >= 6

## Construcción del árbol recursivo para una llamada concreta.

## Construcción del grafo de dependencia para dicha llamada.

## Diseño y construcción de la tabla necesaria para eliminar la recursividad de la llamada anterior.

## Implementación en Java del algoritmo iterativo completo.

## Implementación en Java del algoritmo para recuperar la solución.

## Justificar si es posible reducir la memoria en el algoritmo iterativo. En caso afirmativo, implementar dicho algoritmo en Java

# CUESTIONES TEÓRICAS

# CONCLUSIONES