Universidad Rey Juan Carlos

ETSII Escuela Técnica Superior de Ingeniería Informática

Algoritmos Avanzados

-Práctica 4-

Eliminación de la recursividad redundante

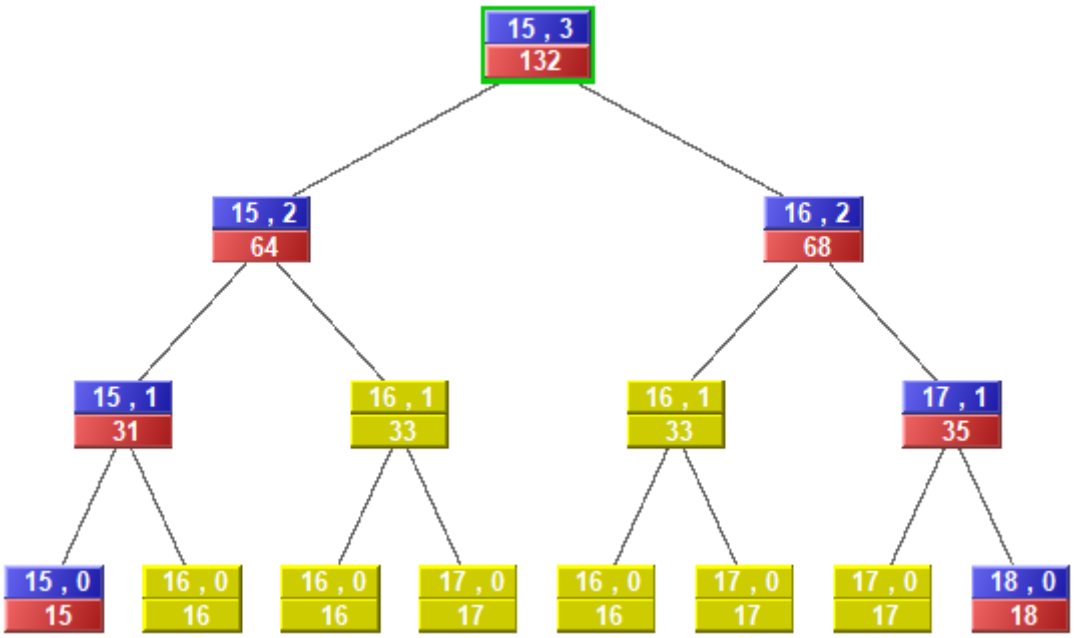
Curso 2022/2023

GII

Alberto Radlowski Nova
Pablo Rodea Piornedo

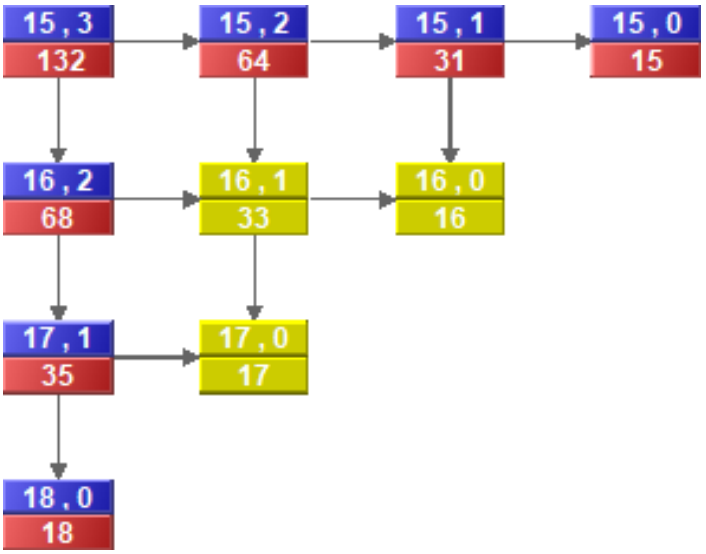
Análisis de la redundancia y diseño de la tabla

Árbol de recursión



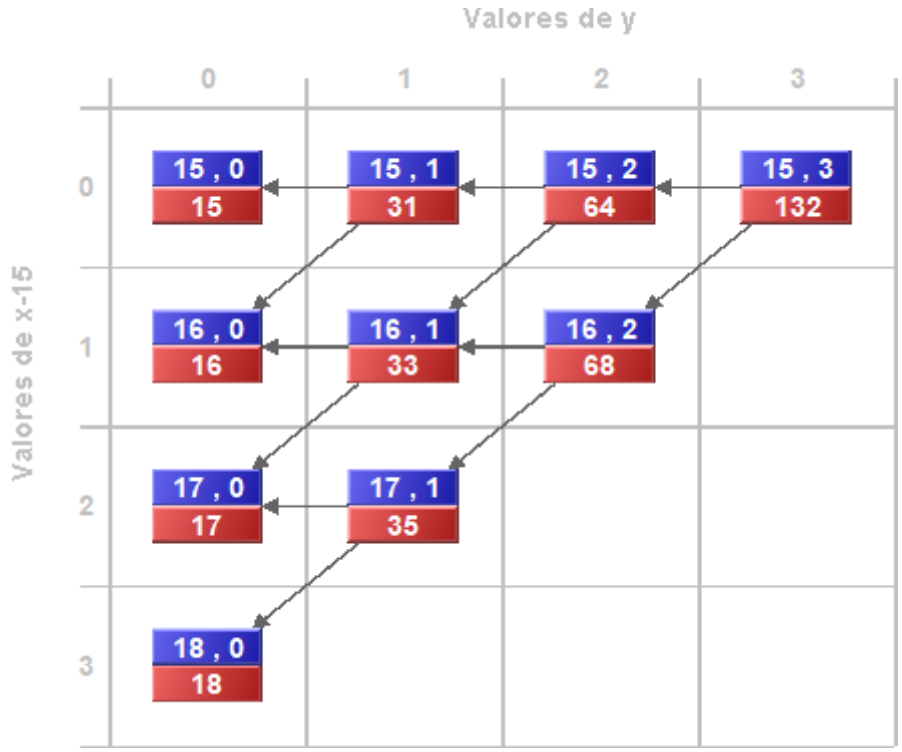
Hemos utilizado valores de *f* tales que x=15 e y=3.

Grafo de dependencia



Tabla

- Figura de la tabla



- Declaración en java de la tabla

```
public static int f2(int x, int y) {
    int[][] tabla = new int[y + 1][y +
1];
    for (int i = 0; i <= y; i++) {
        for (int j = 0; j <= y; j++) {
            tabla[i][j] = -1;
        }
    }
    int x2 = x;
    f2Mem(x, y, tabla, x2);
    return tabla[x-x2][y];
}
```

Memorización

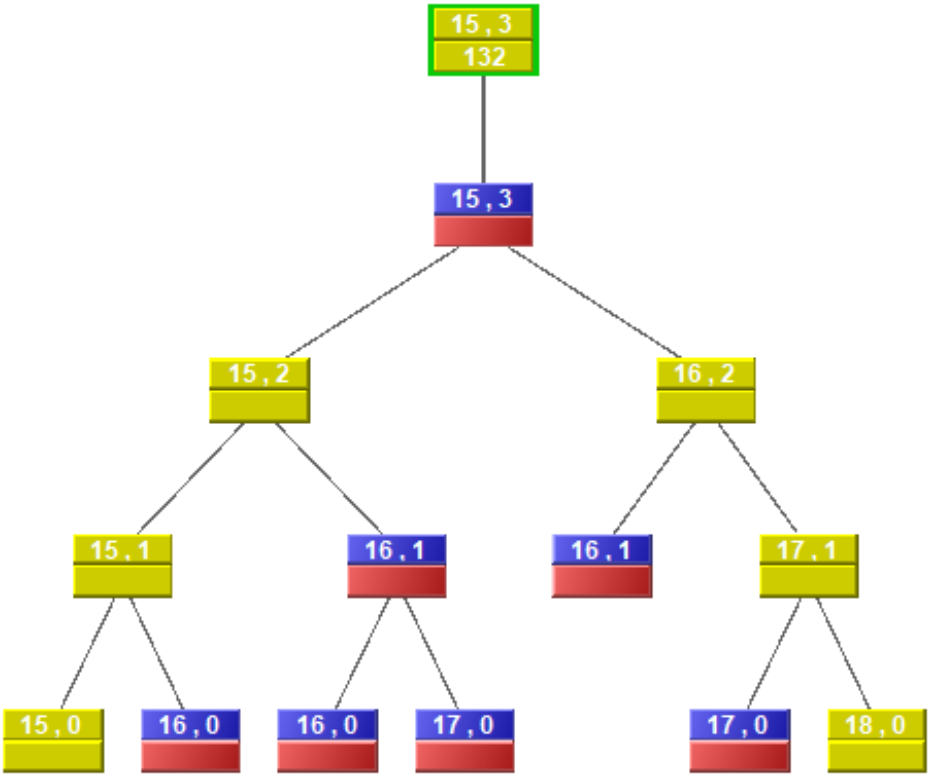
Código

```
public static void f2Mem(int x, int y,
int[][] tabla, int x2) {
    if ((tabla[x-x2][y] == -1)) {
        if(y==0){
            tabla[x-x2][y] = x;
        }
        else {
            f2Mem(x, y - 1, tabla, x2);
            f2Mem(x + 1, y - 1, tabla,
x2);
            tabla[x-x2][y] = tabla[x-x2][y
- 1] + tabla[x-x2+1][y - 1];
        }
    }
}
```

Una vez creada nuestra tabla con todos los valores inicializados a -1, nuestro método hace las llamadas recursivas propias hasta llegar a un “nodo hoja” que se comprueba en `if(y==0)` y le da a este nodo el mismo valor que tiene la x en ese momento.

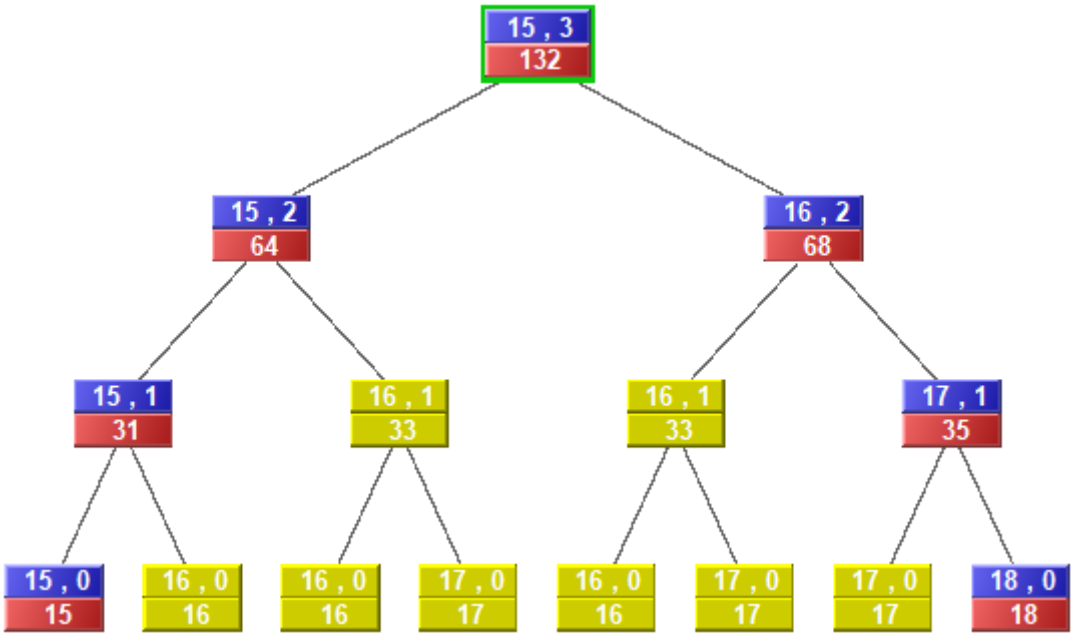
Por último, a la hora de calcular el valor para todos los demás nodos simplemente usamos los valores de los nodos ya calculados de forma inversa a la que se vé en las flechas en la figura de nuestra tabla. Ej: 15 + 16 = 31 | 31 + 33 = 64 | 64 + 68 = 132…

Árbol de recursión



Los nodos marcados en amarillo son las llamadas que se ejecutan una única vez, en cambio las otras son las que se repiten.

árbol recursivo del método original:



Análisis de Complejidad

Complejidad en Tiempo

Las llamadas que se realizan por primera vez son las mismas que aparecen en la tabla declarada anteriormente, es decir:

1) (y+1) * (y+1) -> tabla completa rellena de -1's

2) (y*(y-1)) / 2 -> llamadas que no llegan a realizarse
(los huecos que quedan el la tabla del grafo)

Quedando en total de (y+1)*(y+1) - (y*(y-1)) / 2 llamadas.

Adicionalmente a esto hay que sumar las llamadas repetidas:

((y-1)*(y-2)) / 2

Total de llamadas:

(y+1)*(y+1) - (y*(y-1)) / 2 + ((y-1)*(y-2)) / 2 = y^2+y+2

La complejidad en tiempo es por tanto = y^2+y+2 => $\theta(x^2)$

Complejidad en Espacio

Parámetros: "x", "y" y la tabla:
Rama más larga del árbol: y+1
Tamaño de la tabla: (y+1)*(y+1)

nº parámetros * rama más larga + tamaño de tabla ->

3*(y+1) + (y+1)*(y+1) = y^2+5y+4 => $\theta(x^2)$

Tabulación

Código - T

```
public static int f2Tab(int x, int y)
{
    int [][] tabla = new int[y+1][y+1];
    int x2 = x;
    int suma = 1;
    for(int i=0; i<=y; i++){
        tabla[i][0] = x2;
        x2++;
    }
    int y2 = y;
    for(int i=0; i<=y; i++){
        for(int j=1; j<=y2; j++){
            tabla[i][j] =
tabla[i][j-1]*2+suma;
            suma=suma*2;
        }
        y2--;
        suma = 1;
    }
     return tabla[0][y];
}
```

El primer for inicializa la columna 0 de la matriz con sus correspondientes valores empezando en el [0][0] = x e incrementando el valor de x y de la fila en 1 hasta fila = y.

Es decir: [0][0] = x; [1][0] = x+1; [2][0] = x+2 y así sucesivamente.

Los for anidados recorren la matriz calculando los valores en base a la columna 0 previamente calculada. La fórmula que siguen consiste en multiplicar el valor del nodo de la columna anterior de tu misma fila * 2 y sumar una cantidad.

Esta cantidad comienza siendo 1 y se va multiplicando * 2 según avanza en las columnas.
Es decir, en nuestro ejemplo el nodo [0][0] = 15, entonces el nodo [0][1] = 15*2+1 = 31, y el nodo [0][2] = 31*2+1*2=64 y el nodo [0][3] = 64*2+2*2 = 132

Se sigue el mismo procedimiento para completar la matriz bajando por las filas.

Análisis de Complejidad - T

Complejidad en Tiempo

$$\sum_{i=0}^{y} 1 + \sum_{i=0}^{y} \left(\sum_{j=1}^{y} 1 \right) = y+1+(y+1-1)*(y+1) =$$

$$y^2 + 2 * y + 1 \Rightarrow \theta(x^2)$$

Complejidad en Espacio

- 1 + 1 de variables de entrada “x” e “y”
- Tamaño de tabla -> (y+1) * (y+1)
- 1 + 1 + 1 de variables locales “x2”, “y2” y “suma”
- 1 + 1 de variables de los bucles “i”, “j

$$E(n) = 1 + 1 + (y+1)*(y+1) + 1 + 1 + 1 + 1 + 1 =$$

$$y^2 + 2 * y + 8 \Rightarrow \theta(x^2)$$

Conclusiones

Como conclusión final esta práctica ha servido para afianzar nuestro conocimientos acerca de la recursividad.

En cuanto a las dos formas de eliminar redundancia hemos preferido la tabulación debido a que la consideramos más eficaz y fácil de realizar en contraste a la memorización.

Por último, cabe decir que hemos encontrado la herramienta SRec increíblemente útil y dinámica para la realización de grafos, árboles de recursión y representar gráficamente nuestros métodos (tablas).