Metodología de la Programación y Algoritmia

Convocatoria de Junio 2018
SOLUCIÓN

NºExpediente: _____ **DNI:** _____

Apellidos: _____ **Nombre:** _____

1.- Menú de Juan. Se quiere diseñar un algoritmo RyP para determinar qué platos debe elegir Juan de forma que cumpla todos los requisitos que se especifican.
1.a) Tipificación.
1.b) Función de cota y la estructura del nodo utilizando pseudocódigo.
1.c) Árbol de expansión. 4 platos con 500 kcal, 600 kcal, 300 kcal y 600 kcal. Máximo valor de kcal: 1600 kcal .

SOLUCIÓN:

1.a)
Este ejercicio se puede resolver utilizando el algoritmo de la Mochila que se encuentra en las diapositivas del Tema 7, considerando que el vector de valores es el mismo que el vector de pesos y contiene las kilocalorías de los platos. Consulta el tema para resolver este apartado. En las diapositivas se indica cuál es la tipificación.

1.b)
Se puede utilizar la misma función de cota que la incluida en las diapositivas del Tema 7. Tanto la implementación de esta función como la estructura del nodo se pueden encontrar en las diapositivas.

Estructura del nodo:

- X:{0,1}[n] — vector de decisión de tamaño n (n es el nº total de elementos que queremos introducir en el maletero) que toma los valores 1 o 0 para indicar si el plato i se selecciona o no. Cada elemento ocupa una posición del vector X, así X_1 se refiere al plato 1, X_2 al plato 2 donde i=1,..,n. En el ejemplo, n = 4
- k: natural U {0} — indica hasta qué elemento de X se ha completado el vector.
- valor: real+ — Kilocalorías consumidas. Se obtiene sumando los valores de los platos con X_i=1, i=1,…,k.
- cota: real+ — valor máximo de kilocalorías que se podría alcanzar del menú (cota)

Para las componentes del nodo se ha cogido el tipo de datos más genérico, se podría también haber cogido enteros en vez de reales.

1.c) Árbol de expansión.

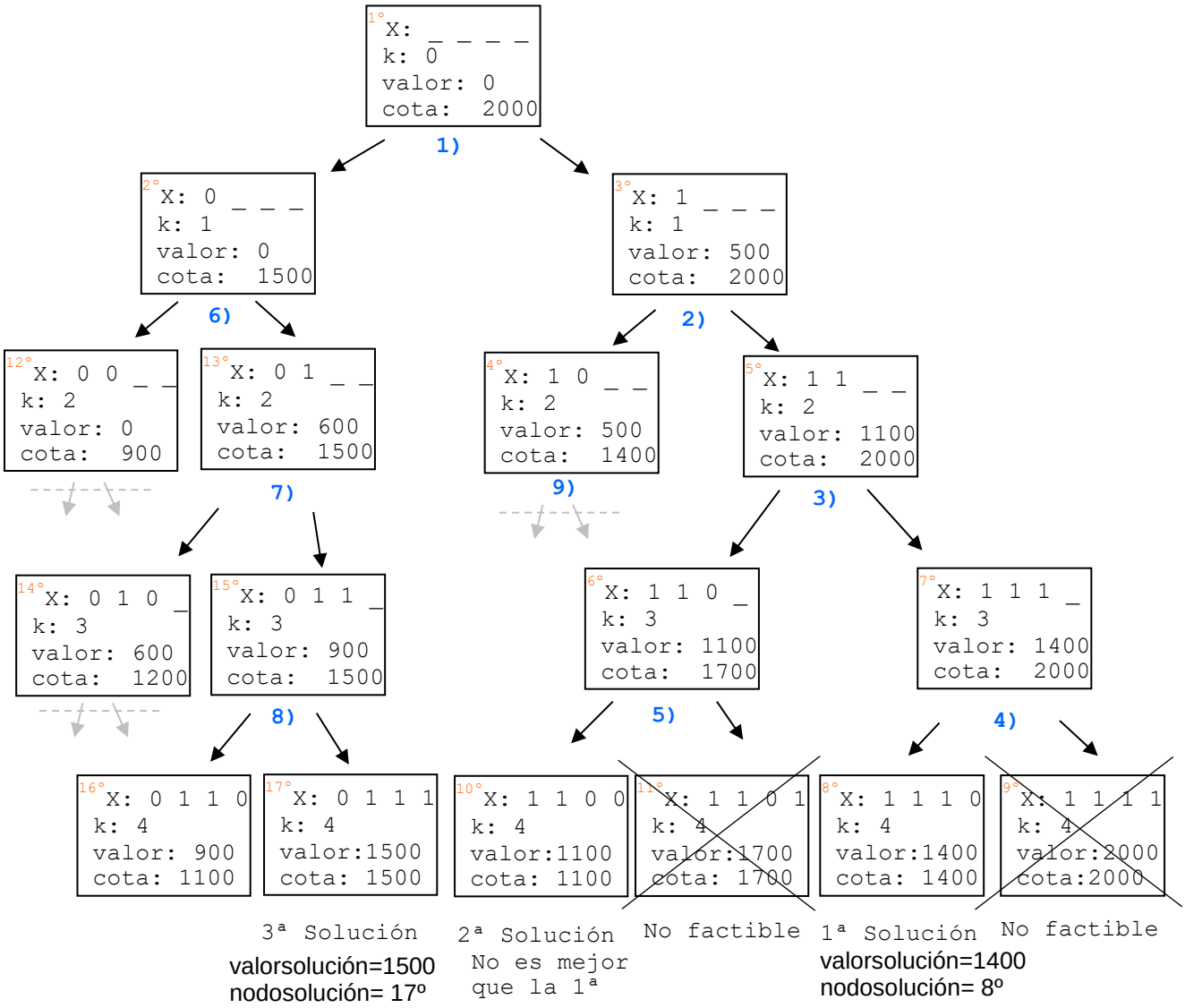
El árbol de expansión se va a realizar aplicando el algoritmo de la Mochila explicado en las diapositivas del Tema 7. El orden en que se generan los nodos se indica con un número en naranja dentro de cada nodo y el orden en que se extraen los nodos de la lista de nodos vivos con un número de color azul debajo del nodo.

La cota del problema la inicializamos a un valor de forma que cuando se encuentre la primera solución se actualice, por ejemplo le asignamos el valor -1. Cuando encontremos una solución actualizamos el valor de la solución si es mejor que el valor de la solución que se tenga. Los valores que va tomando la solución son:

valorsolucion= -1
valorsolucion = 1400 (nodosolución: 8º)
valorsolucion = 1500 (nodosolución: 17º)

Metodología de la Programación y Algoritmia

Convocatoria de Junio 2018
SOLUCIÓN



La evolución de la lista de nodos vivos es la siguiente:

Lista de nodos vivos	Nodo seleccionado	Estado de la lista tras extraer el nodo
{ }		
{1º}	1º	{ }
{2º, 3º}	3º	{2º}
{2º,4º,5º}	5º	{2º,4º}
{2º,4º,6º,7º}	7º	{2º,4º,6º}
{2º,4º,6º}	6º	{2º,4º}
{2º,4º}	2º	{4º}
{4º,13º}	13º	{4º}
{4º,15º}	15º	{4º}
{4º}	4º	{ }

Nº de nodos generados: 17
Nº de nodos podados: 3
Nº de nodos expandidos: 8

La solución del problema es el nodo 17º, que indica que la cantidad máxima de kilocalorías que puede consumir Juan sin sobrepasar las 1600kcal es 1500kcal y para ello pedirá los platos 2, 3 y 4.

Metodología de la Programación y Algoritmia
Convocatoria de Junio 2018
SOLUCIÓN

2.- Dado el siguiente algoritmo

```
función junio18(a:natural U {0}, b:natural U {0}):natural U {0}
   si a < b
      devolver b + junio18(a + 1, b - a)
   si no
      devolver a * b
   fsi
ffunción
```

2.a) Traza de llamadas recursivas para a=3 y b=20 e indica cuál es el resultado final.
2.b) Versión iterativa aplicando el esquema general más adecuado para este tipo de recursividad.

SOLUCIÓN:

2.a) Traza para a = 3 y b = 20.

1º) junio18(3,20) = 20 + 52 = 72
2º) junio18(4,17) = 17 + 35 = 52
3º) junio18(5,13) = 13 + 22 = 35
4º) junio18(6,8) = 8 + 14 = 22
5º) junio18(7,2) = 14

Resultado final: 72

2.b) Versión iterativa
La función presenta una recursividad lineal no final. Como existe la función inversa, aplicamos el esquema de transformación correspondiente a este tipo, que es más adecuado que utilizar pila.

```
función junio18_iterativo(a:natural U {0}, b:natural U {0}):natural U {0}

   a1, b1, s: natural U {0}

   a1 ← a
   b1 ← b
   mientras a1 < b1 hacer
      b1 ← b1 – a1
      a1 ← a1 + 1
   fmientras
   s ← a1 * b1
   mientras !(a1 = a) hacer
      a1 ← a1 - 1
      b1 ← b1 + a1
      s ← b1 + s
   fmientras
   devolver s

ffunción
```

Metodología de la Programación y Algoritmia
Convocatoria de Junio 2018
SOLUCIÓN

3.- Calcula la complejidad asintótica del algoritmo y **justifica** tu respuesta de forma que quede claro por qué sale dicha complejidad. La complejidad asintótica de calcular(A:real[n]):real es $O(n^2 \log n)$.

SOLUCIÓN:

La variable que indica el tamaño del problema es t.

```
        función junio18(X:real[t]):real      Nivel 1    Nivel 2    Nivel 3
(1)        i,j:entero
(2)        k,n:real
(3)        i ← 1                             O(1)
(4)        mientras i ≤ t hacer              O(logt)
(5)           j ← i                                   O(1)
(6)           mientras j > 1 hacer                    O(t)
(7)              k ← Xj - t / 2                                 O(1)
(8)              j ← j - 2                                      O(1)
(9)           fmientras
(10)          i ← i * 2                               O(1)
(11)       fmientras
(12)       n ← calcular(X) + t               O(t²logt)
(13)       devolver n + t                    O(1)
        ffunción
```

En el bucle de la línea (4) el número de iteraciones varía en función del logt y en el de la línea (6) lo hace de forma lineal. Como la condición es del O(1), la complejidad de estas líneas son O(logt) y O(t), respectivamente.

Para resolver el bucle de las líneas (6) – (9), aplicamos la regla de la secuencia de instrucciones a las instrucciones que están dentro del bucle: max{O(1), O(1)} = O(1) y resolvemos la complejidad aplicando la regla del producto O(t)*O(1) = O(t).

```
        función junio18(X:real[t]):real      Nivel 1    Nivel 2
(1)        i,j:entero
(2)        k,n:real
(3)        i ← 1                             O(1)
(4)        mientras i ≤ t hacer              O(logt)
(5)           j ← i                                   O(1)
(6)           mientras j > 1 hacer                    O(t)
(7)              k ← Xj - t / 2
(8)              j ← j - 2
(9)           fmientras
(10)          i ← i * 2                               O(1)
(11)       fmientras
(12)       n ← calcular(X) + t               O(t²logt)
(13)       devolver n + t                    O(1)
        ffunción
```

Resolvemos el bucle de las líneas (4) – (11) procediendo del mismo modo. Las instrucciones que están dentro son del orden de max{O(1), O(t), O(1)} = O(t) y aplicamos la regla del producto O(logt)*O(t) = O(t logt).

```
        función junio18(X:real[t]):real      Nivel 1
(1)        i,j:entero
(2)        k,n:real
(3)        i ← 1                             O(1)
(4)        mientras i ≤ t hacer              O(tlogt)
(5)           j ← i
(6)           mientras j > 1 hacer
(7)              k ← Xj - t / 2
(8)              j ← j - 2
(9)           fmientras
(10)          i ← i * 2
(11)       fmientras
(12)       n ← calcular(X) + t               O(t²logt)
(13        devolver n + t                    O(1)
        ffunción
```

Finalmente aplicamos la regla de instrucciones secuenciales a las instrucciones del nivel 1. Con lo cual, la complejidad del algoritmo junio18 es max{O(1), O(t logt), O(t^2 logt), O(1)} = O(t^2 logt).