

## 1.00 Clase adicional 10

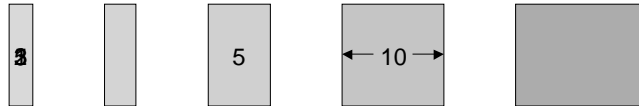
29/4/2002

### Correcciones del BP9 en el sitio web

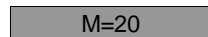
- Regla de reducción nº 2 actualizada (debería de ser " $\leq$ ", no " $<$ ")
- Diagrama corregido

## BP9: ¿Para qué sirve?

Dada (1) una serie de componentes de equipos de distintas longitudes,



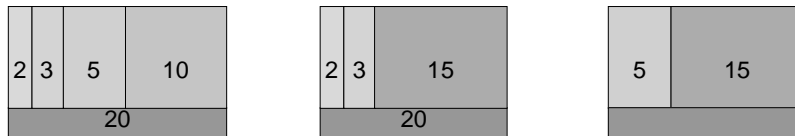
y (2) un *palet* de anchura  $M$ ,



¿De cuántas formas podemos embalar un subconjunto de equipos de música, uno al lado del otro, para que encajen en el *palet* de forma exacta?

## BP9: ¿Para qué sirve?

Para este ejemplo existen tres soluciones

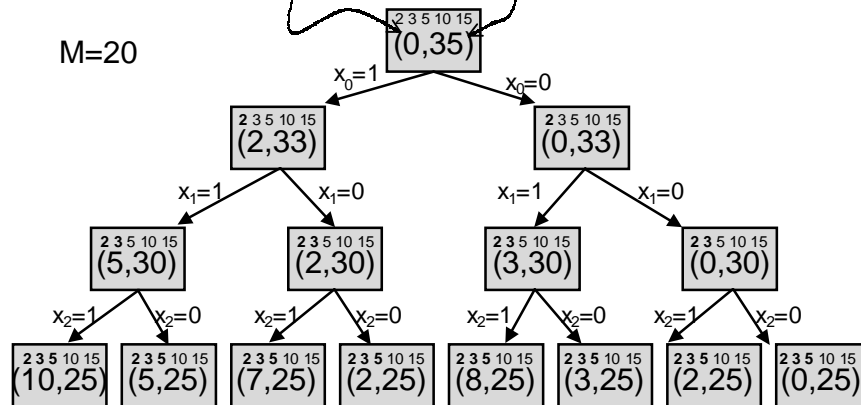


Pero, ¿cómo resolveríamos este problema en general?

## BP9: Algoritmo de árbol

s=Suma de long. de esta solución      r=Suma de long. a tener en cuenta

M=20



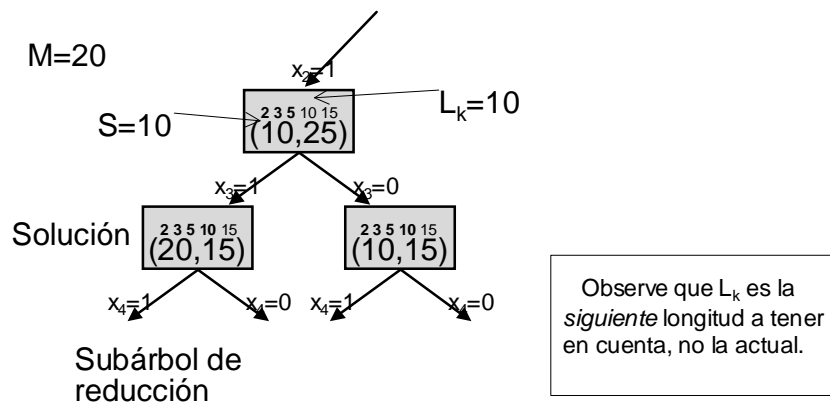
más dos niveles más

## BP9: Algoritmo de árbol

- Enumera todas las combinaciones
- Crece exponencialmente:  $O(2^n)$
- Se encuentra la solución siempre que  $s=M$
- $x[ ]$  es la solución actual, toma valores 0 si la longitud  $L_k$  no está en la solución, o 1 si lo está.
- Las reglas de reducción disminuyen el número de nodos “podando” subárboles. (Tenga en cuenta que nuestras “reglas de reducción” son en realidad “reglas de generación de subárboles”, ya que la acción realizada es generar nodos, no reducir los que ya existen.)

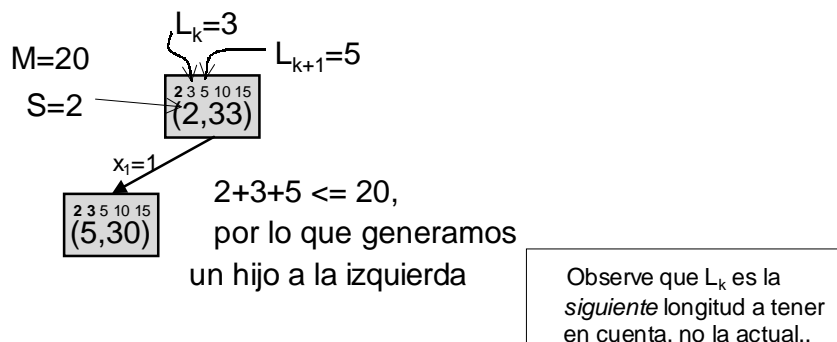
## Regla de reducción nº 1

- Si  $(s + L_k = M)$ , hay solución en la salida (y no se sigue a ningún subárbol)



## Regla de reducción nº 2

- Si  $(s + L_k + L_{k+1} \leq M)$ , se genera un hijo a la izquierda y se define  $x_k = 1$  (de lo contrario, no se hace)



## Regla de reducción nº 3

- Si  $(s + r - L_k \geq M)$  y  $(s + L_{k+1} \leq M)$ , se genera un hijo a la derecha y se define  $x_k = 0$
- Parte 1: si íbamos a generar un nodo a la derecha, las long. máx. que podríamos agregar a nuestra solución actual serían  $r - L_k$ . Si dicha cantidad es  $< M$ , nunca podremos llenar el *palet*, por lo que no merece la pena seguir.
- Parte 2: si la siguiente long. ( $L_{k+1}$ ) es mayor que la anchura del *palet*, tampoco podremos utilizarla y no merece la pena seguir.

Observe que  $L_k$  es la *siguiente* longitud a tener en cuenta, no la actual.

## Ejercicio

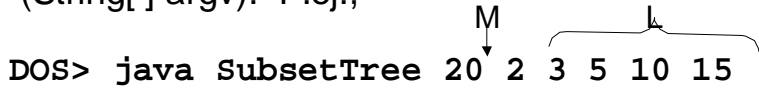
- Aplique las 3 reglas de reducción para decidir si se generan o no nodos a la derecha y a la izquierda cuando  $M=25$ ,  $L = \{10, 12, 13, 30\}$ 
  - (1) Para el nodo raíz ( $k=0$ )
  - (2) Cuando  $k = 2$ ,  $x=\{0, 1\}$
  - (3) Cuando  $k = 2$ ,  $x=\{1, 1\}$

La solución en la siguiente diapositiva...

## Solución al ejercicio

1. Se generan nodos a la derecha y a la izquierda. (Reglas 2 y 3)
2. Solución encontrada con la regla 1. No se genera ningún nodo más
3. No se generan nodos ni a la derecha ni a la izquierda. (Reglas 2 y 3.) Es un callejón sin salida.

## BP9: Implementación

- Los datos de entrada se leen desde la línea de comandos  
(String[] argv). P.ej.,  


```
DOS> java SubsetTree 20 2 3 5 10 15
```
- Escriba una clase `SubsetTree` como clase principal. Debería crear el nodo raíz y llamar a `root.visit()`
- La implementación sugerida se utiliza para crear un árbol que utilice una clase `Node`.
  - Cada nodo debe llevar un array de *solución* (`int x[]`) de tamaño `L.longitud`. ¿Cómo debería inicializarse?
  - ¿Qué otras variables/métodos necesitará?

## Método `visit()` del nodo

- El método `visit()` hace todo el trabajo:
  - Implementa reglas de reducción
  - Crea nodos a la izquierda y a la derecha, y llama a (`left.visit()`, `right.visit()`) repetidamente tantas veces como se necesite
  - Establece `x[i]` en 0 o en 1
  - Imprime las soluciones si las encuentra (se recomienda tener un método `printSolution` a mano)

## Otras soluciones

- Existen muchas soluciones posibles
- Ni siquiera necesita crear nodos (puede utilizar sólo la recursión para recorrer un árbol implícito)