


Programa de teoría

Parte I. Estructuras de Datos

1. Abstracciones y especificaciones
2. Conjuntos y diccionarios
3. Representación de conjuntos mediante árboles
4. Grafos

Parte II. Algorítmica

1. Análisis de algoritmos
2. Divide y vencerás
3. Algoritmos voraces
4. Programación dinámica
5. Backtracking
-  6. Ramificación y poda

PARTE II: ALGORÍTMICA

Tema 6. Ramificación y poda

6.1. Método general

6.2. Análisis de tiempos de ejecución

6.3. Ejemplos de aplicación

6.3.1. Problema de la mochila 0/1

6.3.2. Problema de la asignación

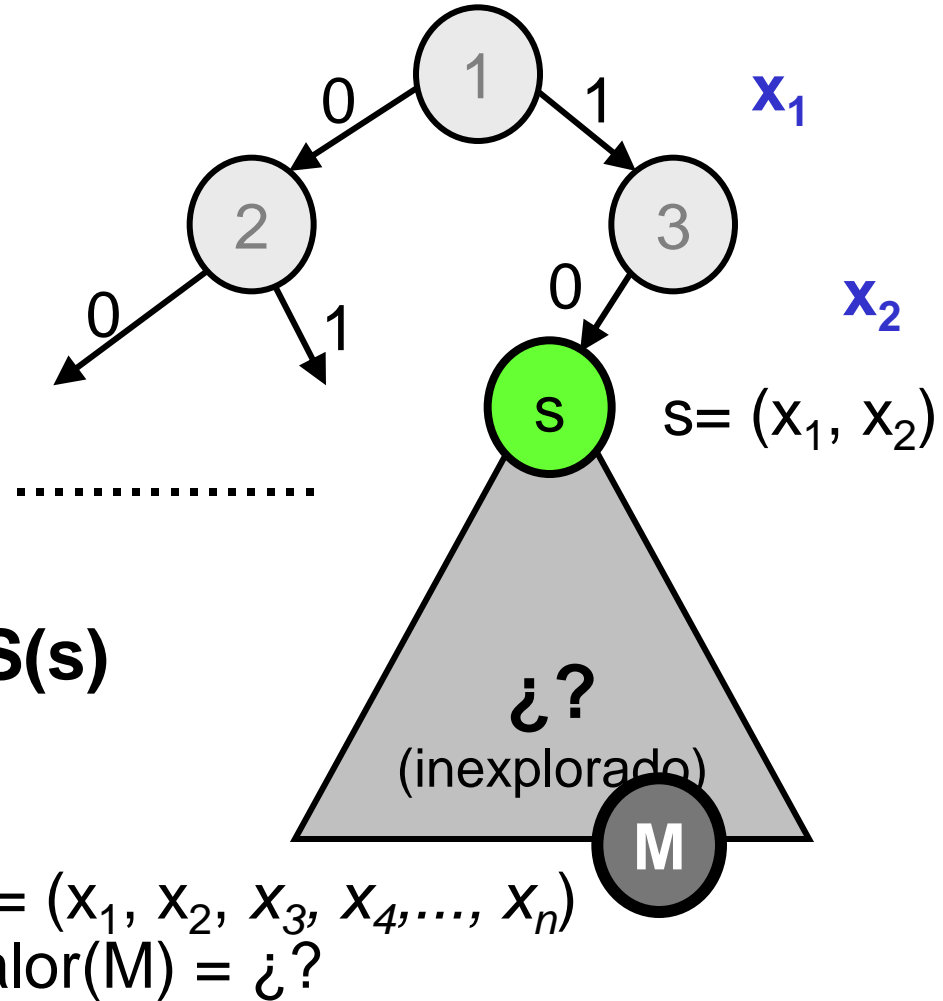
6.1. Método general

- La **ramificación y poda (branch and bound)** se suele utilizar en problemas de **optimización discreta** y en problemas de **juegos**.
- Puede ser vista como una **generalización** (o mejora) de la técnica de **backtracking**.
- **Similitud:**
 - Igual que backtracking, realiza un **recorrido sistemático** en un árbol de soluciones.
- **Diferencias:**
 - **Estrategia de ramificación:** el recorrido no tiene por qué ser necesariamente en profundidad.
 - **Estrategia de poda:** la poda se realiza **estimando** en cada nodo **cotas** del beneficio óptimo que podemos obtener a partir del mismo.

6.1. Método general

Estimación de cotas a partir de una solución parcial

- **Problema:** antes de explorar s , acotar el beneficio de la mejor solución alcanzable, M .



- **$CI(s) \leq \text{Valor}(M) \leq CS(s)$**

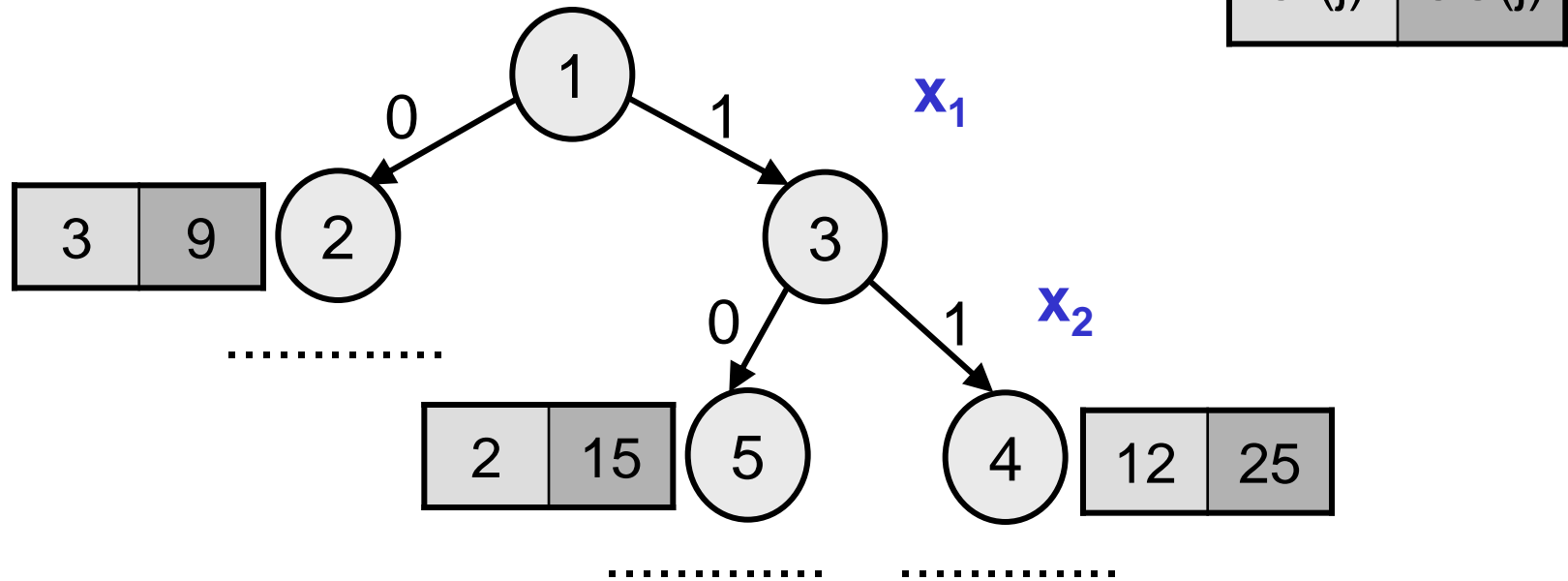
6.1. Método general

- Para cada nodo i tendremos:
 - **CS(i): Cota superior** del beneficio (o coste) óptimo que podemos alcanzar a partir del nodo i .
 - **CI(i): Cota inferior** del beneficio (o coste) óptimo que podemos alcanzar a partir del nodo i .
 - **BE(i): Beneficio estimado** (o coste) óptimo que se puede encontrar a partir del nodo i .
- Las cotas deben ser “fiables”: determinan cuándo se puede realizar una poda.
- El beneficio (o coste) estimado ayuda a decidir qué parte del árbol evaluar primero.

6.1. Método general

Estrategia de poda

- Supongamos un problema de **maximización**.
- Hemos recorrido varios nodos, estimando para cada uno la cota superior **CS(j)** e inferior **CI(j)**.



- ¿Merece la pena seguir explorando por el nodo 2?
- ¿Y por el 5?

6.1. Método general

- **Estrategia de poda (maximización).** Podar un nodo **i** si se cumple que:
 - $CS(i) \leq CI(j)$** , para algún nodo **j** generado
 - o bien
 - $CS(i) \leq Valor(s)$** , para algún nodo **s** solución final
- **Implementación.** Usar una variable de poda **C**:
 $C = \max(\{CI(j) \mid \forall j \text{ generado}\}, \{Valor(s) \mid \forall s \text{ solución final}\})$
 - Podar **i** si: **$CS(i) \leq C$**
- ¿Cómo sería para el caso de minimización?

6.1. Método general

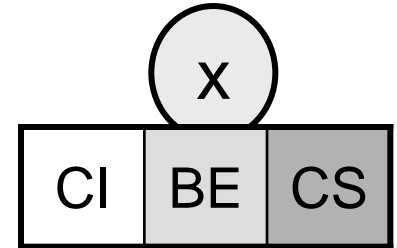
Estrategias de ramificación

- Igual que en backtracking, hacemos un recorrido en un **árbol** de soluciones (que es **implícito**).
- **Distintos tipos de recorrido:** en profundidad, en anchura, según el beneficio estimado, etc.
- Para hacer los recorridos se utiliza una **lista de nodos vivos**.
- **Lista de nodos vivos (LNV):** contiene todos los nodos que han sido generados pero que no han sido explorados todavía. Son los nodos pendientes de tratar por el algoritmo.

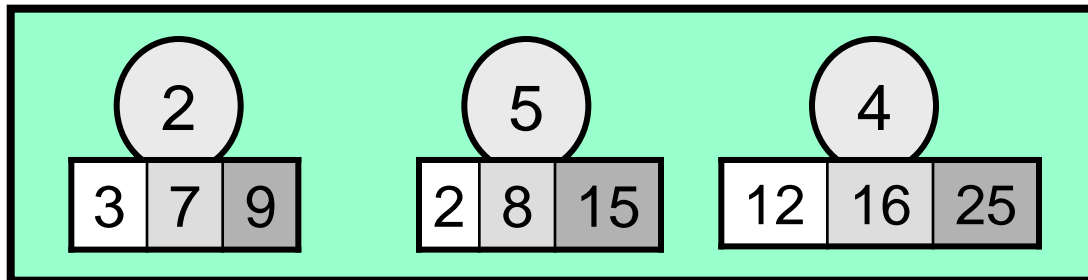
6.1. Método general

Estrategias de ramificación

- **Idea básica del algoritmo:**
 - Sacar un elemento de la lista LNV.
 - Generar sus descendientes.
 - Si no se podan, meterlos en la LNV.



LNV

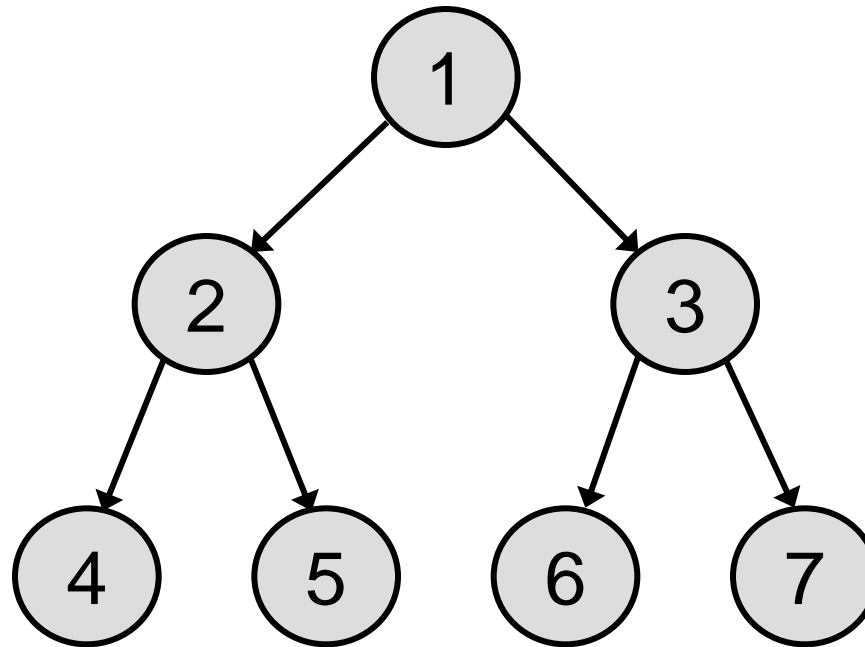


- ¿En qué orden se sacan y se meten?
- Según cómo se maneje esta lista, el recorrido será de uno u otro tipo.

6.1. Método general

Estrategia de ramificación FIFO (First In First Out)

- Si se usa la estrategia FIFO, la LNV es una **cola** y el recorrido es:
 - a) En profundidad
 - b) En anchura
 - c) NS/NC

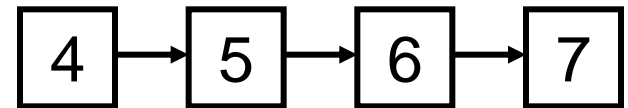
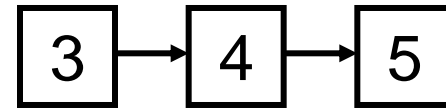
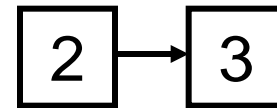
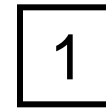
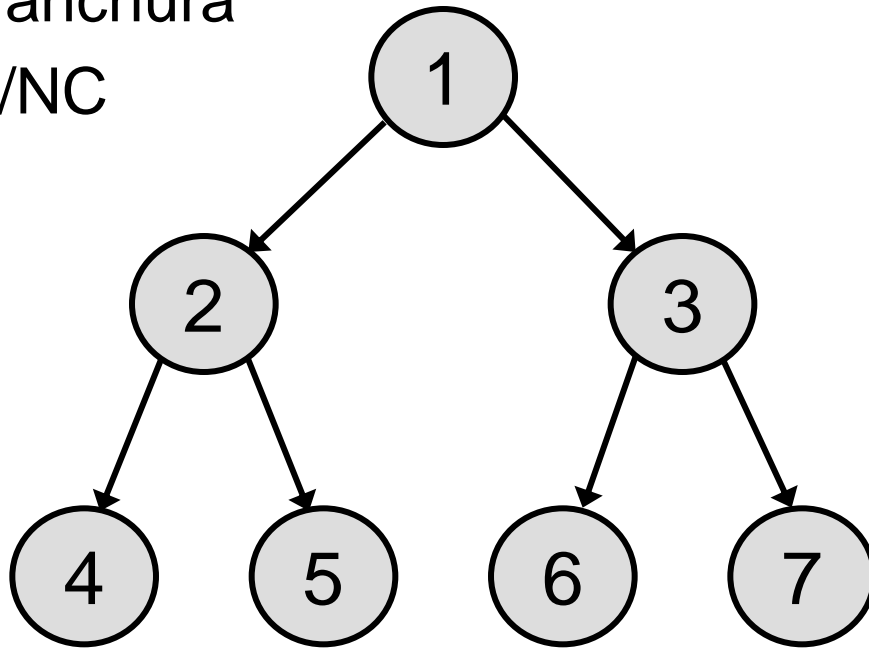


6.1. Método general

Estrategia de ramificación FIFO (First In First Out)

- Si se usa la estrategia FIFO, la LNV es una **cola** y el recorrido es:

- a) En profundidad
- b) En anchura
- c) NS/NC

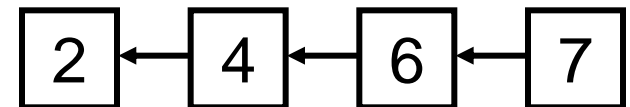
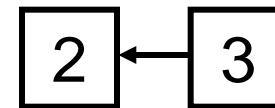
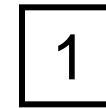
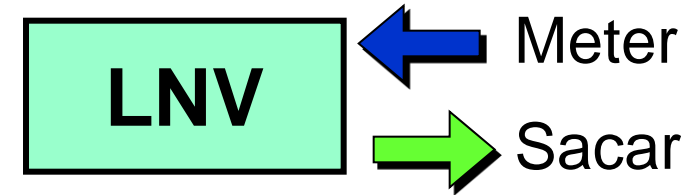
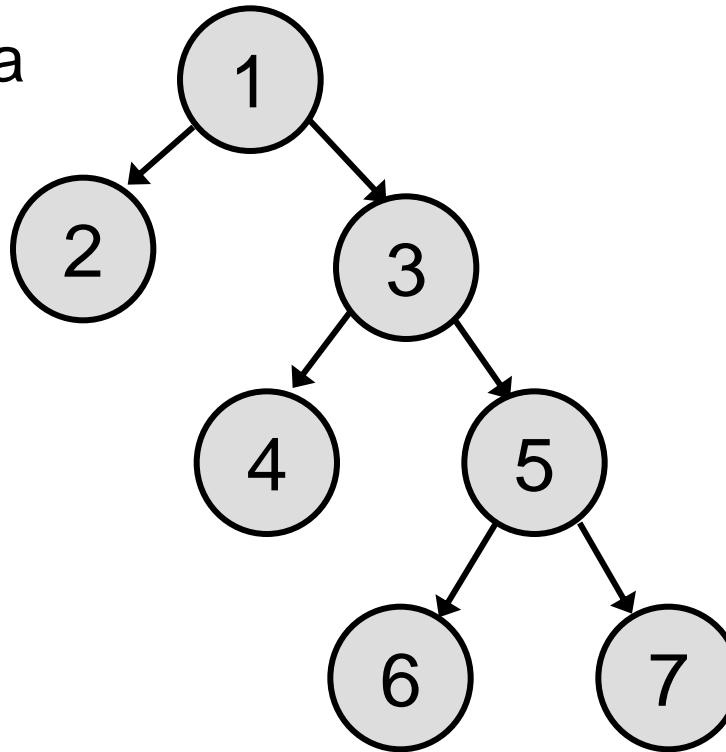


6.1. Método general

Estrategia de ramificación LIFO (Last In First Out)

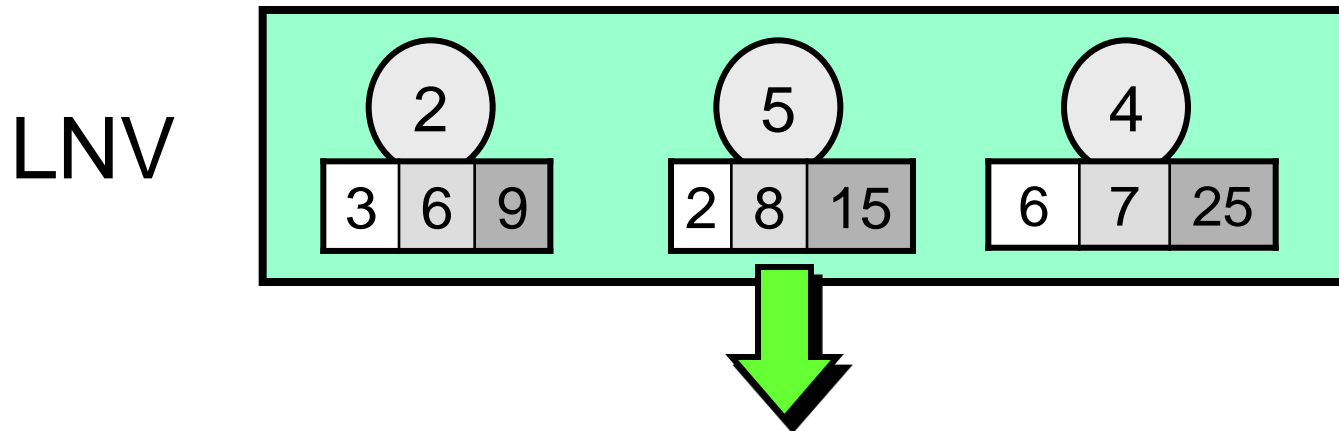
- Si se usa la estrategia LIFO, la LNV es una **pila** y el recorrido es:

- a) En profundidad
- b) En anchura
- c) NS/NC



6.1. Método general

- Las estrategias FIFO y LIFO realizan una búsqueda “a ciegas”, sin tener en cuenta los beneficios.
- **Usamos la estimación del beneficio:** explorar primero por los nodos con mayor valor estimado.
- **Estrategias LC (Least Cost):** entre todos los nodos de la lista de nodos vivos, elegir el que tenga mayor beneficio (o menor coste) para explorar a continuación.



6.1. Método general

Estrategias de ramificación LC

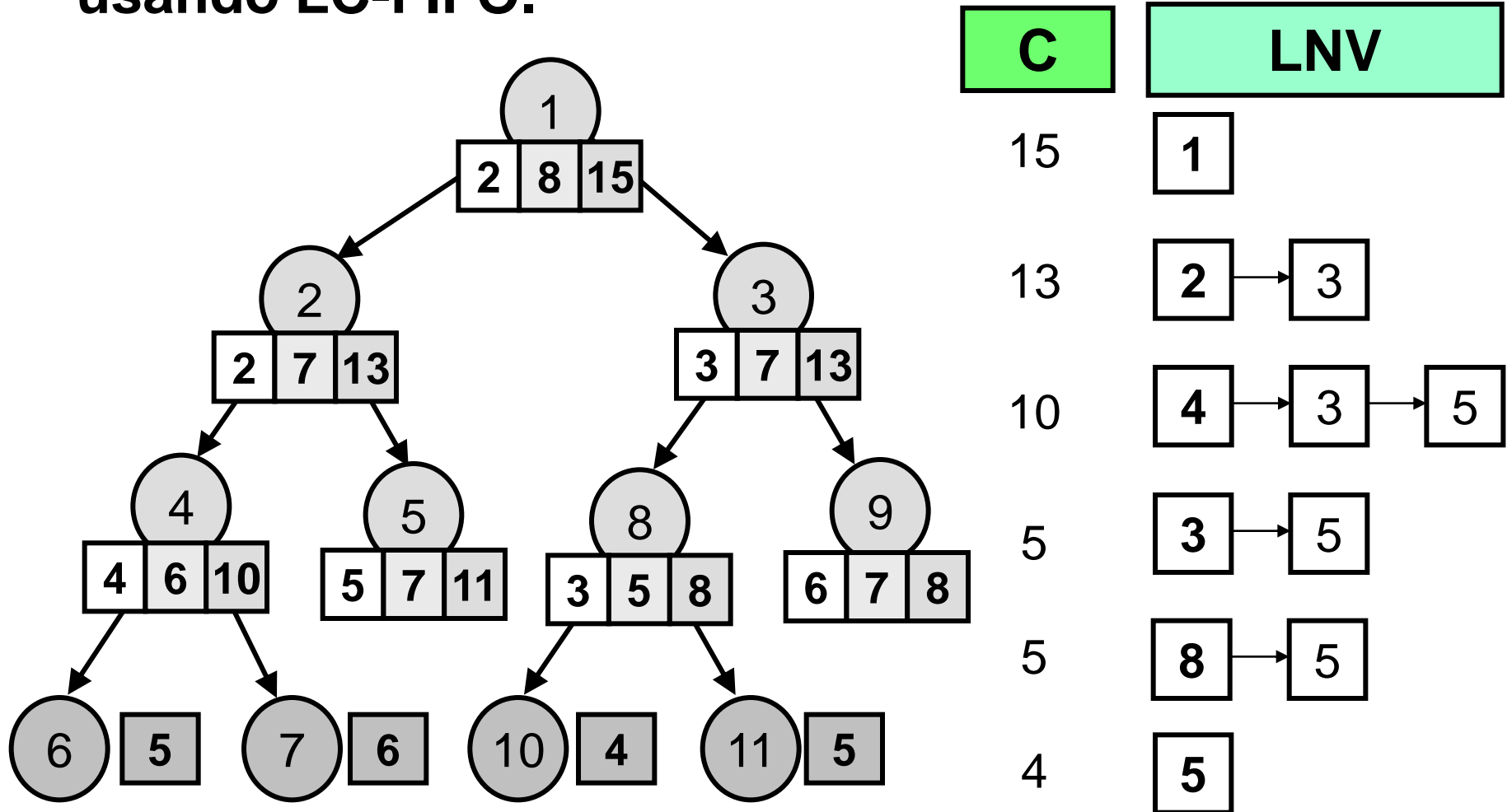
- En caso de empate (de beneficio o coste estimado) deshacerlo usando un criterio **FIFO** o **LIFO**.
- **Estrategia LC-FIFO**: seleccionar de LNV el nodo que tenga mayor beneficio y en caso de empate escoger el primero que se introdujo (de los que empatan).
- **Estrategia LC-LIFO**: seleccionar el nodo que tenga mayor beneficio y en caso de empate escoger el último que se introdujo (de los que empatan).
- ¿Cuál es mejor?
- Se diferencian si hay muchos “empates” a beneficio estimado.

6.1. Método general

- **Resumen:**
 - En cada nodo i tenemos: **CI(i)**, **BE(i)** y **CS(i)**.
 - **Podar** según los valores de **CI** y **CS**.
 - **Ramificar** según los valores de **BE**.
- **Ejemplo. Recorrido con ramificación y poda, usando LC-FIFO.**
 - Suponemos un problema de minimización.
 - Para realizar la poda usamos una variable **C** = valor de la menor de las cotas superiores hasta ese momento, o de alguna solución final.
 - Si para algún nodo i , **CI(i) \geq C**, entonces podar i .

6.1. Método general

- Ejemplo. Recorrido con ramificación y poda, usando LC-FIFO.



6.1. Método general

- **Esquema algorítmico de ramificación y poda.**
 - **Inicialización:** meter la raíz en la LNV, e inicializar la variable de poda **C** de forma conveniente.
 - **Repetir** mientras no se vacíe la LNV:
 - **Sacar un nodo** de la LNV, según la estrategia de ramificación.
 - Comprobar si debe ser podado, según la **estrategia de poda**.
 - En caso contrario, **generar sus hijos**. Para cada uno:
 - Comprobar si es una **solución final** y tratarla.
 - Comprobar si debe ser **podado**.
 - En caso contrario, **meterlo en la LNV** y **actualizar C** de forma adecuada.

6.1. Método general

Ramificación Y Poda (raiz: Nodo; var s: Nodo) // *Minimización*

LNV := {raiz}

C := **CS**(raiz)

s := \emptyset

mientras LNV $\neq \emptyset$ hacer

 x := **Seleccionar**(LNV) // *Estrategia de ramificación*

 LNV := LNV - {x}

 si **CI**(x) < C entonces // *Estrategia de poda*

 para cada y hijo de x hacer

 si **Solución**(y) AND (**Valor**(y) < **Valor**(s)) entonces

 s := y

 C := min (C, **Valor**(y))

 sino si NO **Solución**(y) AND (**CI**(y) < C) entonces

 LNV := LNV + {y}

 C := min (C, **CS**(y))

 finsi

 finpara

finmientras

6.1. Método general

- **Funciones genéricas:**

- **CI(i), CS(i), CE(i).** Cota inferior, superior y coste estimado, respectivamente.
- **Solución(x).** Determina si **x** es una solución final válida.
- **Valor(x).** Valor de una solución final.
- **Seleccionar(LNV): Nodo.** Extrae un nodo de la LNV según la estrategia de ramificación.
- **para cada y hijo de x hacer.** Iterador para generar todos los descendientes de un nodo. Equivalente a las funciones de backtracking.

y:= x

mientras MasHermanos(y) **hacer**

Generar(nivel(x)+1, y)

si Criterio(y) **entonces** ...

6.1. Método general

Algunas cuestiones

- Se comprueba el criterio de poda al meter un nodo y al sacarlo. ¿Por qué esta duplicación?
- ¿Cómo actualizar **C** si el problema es de maximizar? ¿Y cómo es la poda?
- ¿Qué información se almacena en la LNV?

LNV: Lista[Nodo]

tipo

Nodo = registro

tupla: TipoTupla // P.ej. array [1..n] de entero

nivel: entero

CI, CE, CS: real



Almacenar para no recalcular. ¿Todos?

finregistro

6.1. Método general

- ¿Qué pasa si para un nodo i tenemos que $CI(i)=CS(i)$?
- ¿Cómo calcular las cotas?
- ¿Qué pasa con las cotas si a partir de un nodo puede que no exista ninguna solución válida (factible)?

6.2. Análisis de tiempos de ejecución

- El tiempo de ejecución depende de:
 - **Número de nodos recorridos:** depende de la efectividad de la poda.
 - **Tiempo gastado en cada nodo:** tiempo de hacer las estimaciones de coste y tiempo de manejo de la lista de nodos vivos.
- En el caso promedio se suelen obtener mejoras respecto a backtracking...
- En el peor caso, se generan tantos nodos como en backtracking → El tiempo puede ser peor según lo que se tarde en calcular las cotas y manejar la LNV.
- ¿Cuántos nodos, como máximo, puede tener la LNV?

6.2. Análisis de tiempos de ejecución

- **Problema:** complejidad exponencial tanto en tiempo como en uso de memoria.
- ¿Cómo hacer más eficiente un algoritmo de RyP?
 - **Hacer estimaciones y cotas muy precisas** → Poda muy exhaustiva del árbol → Se recorren menos nodos pero se tardará mucho en hacer estimaciones.
 - **Hacer estimaciones y cotas poco precisas** → No se hace mucha poda → Se gasta poco tiempo en cada nodo, pero el número de nodos es muy elevado.
- Se debe buscar un equilibrio entre la exactitud de las cotas y el tiempo de calcularlas.

6.3. Ejemplos de aplicación

Aplicación de ramificación y poda (proceso metódico):

- 1) Definir la representación de la solución. A partir de un nodo, cómo se obtienen sus descendientes.
- 2) Dar una manera de calcular el valor de las cotas y la estimación del beneficio.
- 3) Definir la estrategia de ramificación y de poda.
- 4) Diseñar el esquema del algoritmo.

6.3. Ejemplos de aplicación

6.3.1. Problema de la mochila 0/1

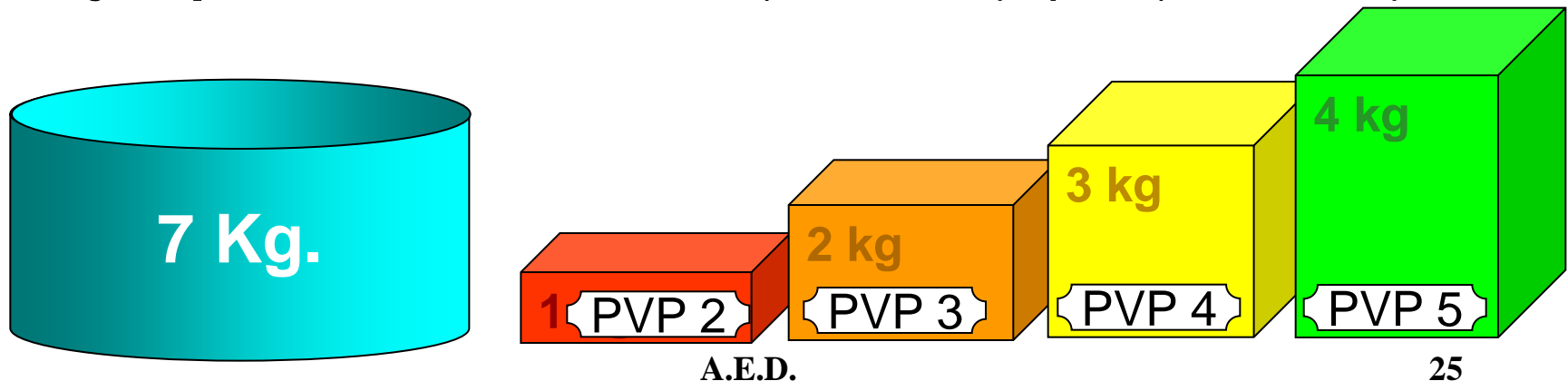
- **Datos del problema:**

- n : número de objetos disponibles.
- M : capacidad de la mochila.
- $\mathbf{p} = (p_1, p_2, \dots, p_n)$ pesos de los objetos.
- $\mathbf{b} = (b_1, b_2, \dots, b_n)$ beneficios de los objetos.

- **Formulación matemática:**

Maximizar $\sum_{i=1..n} x_i b_i$; sujeto a la restricción $\sum_{i=1..n} x_i p_i \leq M$, y $x_i \in \{0, 1\}$

- **Ejemplo:** $n = 4$; $M = 7$; $\mathbf{b} = (2, 3, 4, 5)$; $\mathbf{p} = (1, 2, 3, 4)$



6.3.1. Problema de la mochila 0/1

1) Representación de la solución

- Con un árbol binario: $\mathbf{s} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$, con $x_i \in \{0, 1\}$
 - $x_i = 0 \rightarrow$ No se coge el objeto i ; $x_i = 1 \rightarrow$ Sí se coge i

tipo

Nodo = registro

tupla: **array** [1..n] **de** entero

nivel: entero

bact, pact: entero

CI, BE, CS: entero

finregistro

1.a) ¿Cómo es el nodo raíz?

1.b) ¿Cómo generar los hijos de un nodo?

1.c) ¿Cómo es la función Solución(x: Nodo): booleano?

6.3.1. Problema de la mochila 0/1

1.a) Nodo raíz

raiz.nivel:= 0

raiz.bact:= 0

raiz.pact:= 0

1.b) Para cada y hijo de un nodo x

para $i := 0, 1$ hacer

$y.nivel := x.nivel + 1$

$y.tupla := x.tupla$

$y.tupla[y.nivel] := i$

$y.bact := x.bact + i * b[y.nivel]$

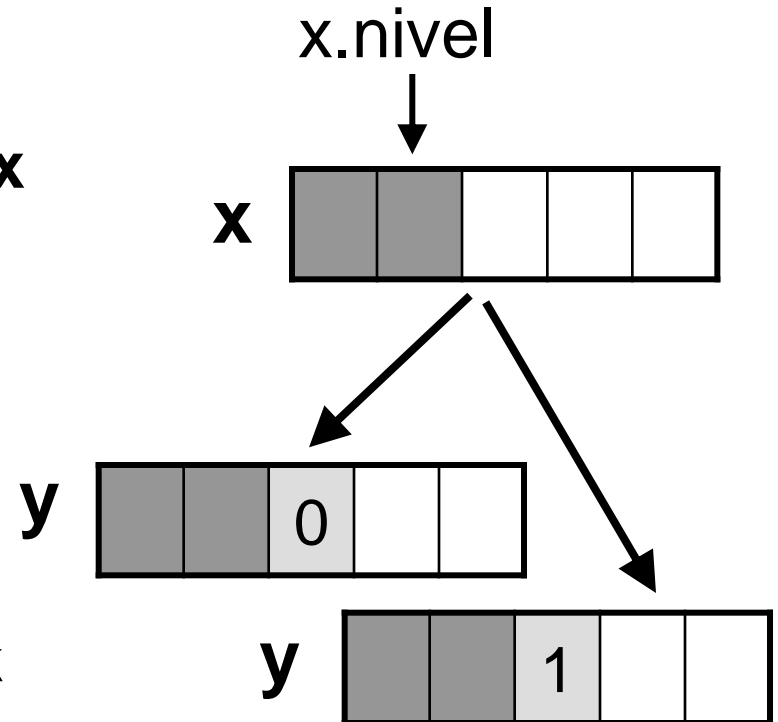
$y.pact := x.pact + i * p[y.nivel]$

si $y.pact > M$ entonces break

....

1.c) Función Solución(x: Nodo): booleano

devolver $x.nivel == n$



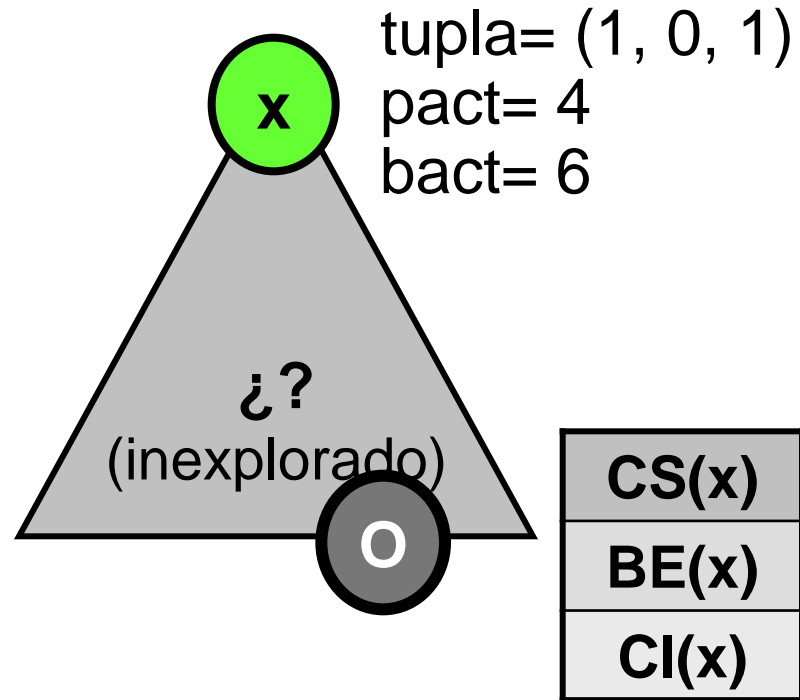
6.3.1. Problema de la mochila 0/1

2) Cálculo de las funciones $CI(x)$, $CS(x)$, $BE(x)$

$n = 6, M = 11$

$p = (1, 2, 3, 4, 5, 6)$

$b = (2, 3, 4, 5, 6, 7)$



2.a) Cálculo de $CI(x)$

Posibilidad 1. El beneficio acumulado hasta ese momento:

$$x.CI := x.bact$$

6.3.1. Problema de la mochila 0/1

2.b) Cálculo de CS(x)

- **Idea** (igual que con backtracking): la solución de la mochila no 0/1 es una cota superior válida de la mochila 0/1.

$$x.CS := x.bact + \lfloor \text{MochilaNo01}(x.nivel+1, n, M-x.pact) \rfloor$$

MochilaNo01(a, b, Q): problema de la mochila no 0/1 con los objetos (a, ..., b) y peso Q.

2.c) Cálculo de BE(x)

- **Idea:** usar un algoritmo voraz para el caso 0/1. Añadir objetos enteros, si caben enteros, por orden de b/p.

$$x.BE := x.bact + \text{MochilaVoraz01}(x.nivel+1, n, M-x.pact)$$

6.3.1. Problema de la mochila 0/1

2.c) Cálculo de BE(x)

MochilaVoraz01(a, b, Q): entero

 bacum:= 0

 pacum:= 0

para i:= a, ..., b **hacer**

si pacum+p[i] < Q **entonces**

 pacum:= pacum + p[i]

 bacum:= bacum + b[i]

finsi

finpara

devolver bacum

- **Ojo:** se supone que los objetos están ordenados por b/p.

6.3.1. Problema de la mochila 0/1

2) Cálculo de las funciones $CI(x)$, $CS(x)$, $BE(x)$

- **Ejemplo.** $x = (1, 0, 1)$
 $n = 6$, $M = 11$
 $p = (1, 2, 3, 4, 5, 6)$
 $b = (2, 3, 4, 5, 6, 7)$
- ¿Cuánto valen $CI(x)$, $CS(x)$, $BE(x)$?
- ¿Cuánto es la solución óptima? ¿Son buenas las funciones?
- **Idea:** el valor calculado para $BE(x)$ puede usarse como un valor de $CI(x)$:
 $x.CI := x.bact + MochilaVoraz01(x.nivel+1, n, M-x.pact)$
- ¿Por qué?

6.3.1. Problema de la mochila 0/1

3) Estrategia de ramificación y de poda

3.a) Estrategia de poda

- **Variable de poda C:** valor de la mayor cota inferior o solución final del problema.
- **Condición de poda:** podar i si: $i \cdot CS \leq C$

3.b) Estrategia de ramificación

- **Usar una estrategia LC:** explorar primero los nodos con mayor BE (**estrategia MB**).
- ¿LC-FIFO ó LC-LIFO? LC-LIFO: en caso de empate seguir por la rama más profunda. (MB-LIFO)

6.3.1. Problema de la mochila 0/1

4) Esquema del algoritmo

- Usar un esquema parecido al genérico.
- Idea básica:
 - Meter el nodo raíz en la LNV
 - Mientras no se vacíe la LNV
 - Sacar el siguiente nodo, según estrategia MB-LIFO
 - Generar sus hijos (iterador **para cada hijo...**)
 - Si no se podan, meterlos en la LNV

6.3.1. Problema de la mochila 0/1

Mochila01RyP (n: ent; b, p: array[1..n] de ent; var s: Nodo)

LNV:= {raiz}

C:= raiz.CI

s:= \emptyset

mientras LNV $\neq \emptyset$ **hacer**

 x:= Seleccionar(LNV) // Estrategia MB-LIFO

 LNV:= LNV - {x}

si x.CS > C **entonces** // Estrategia de poda

para cada y hijo de x hacer

si Solución(y) AND (y.bact > s.bact) **entonces**

 s:= y

 C:= max (C, y.bact)

sino si NO Solución(y) AND (y.CS > C) **entonces**

 LNV:= LNV + {y}

 C:= max (C, y.CI)

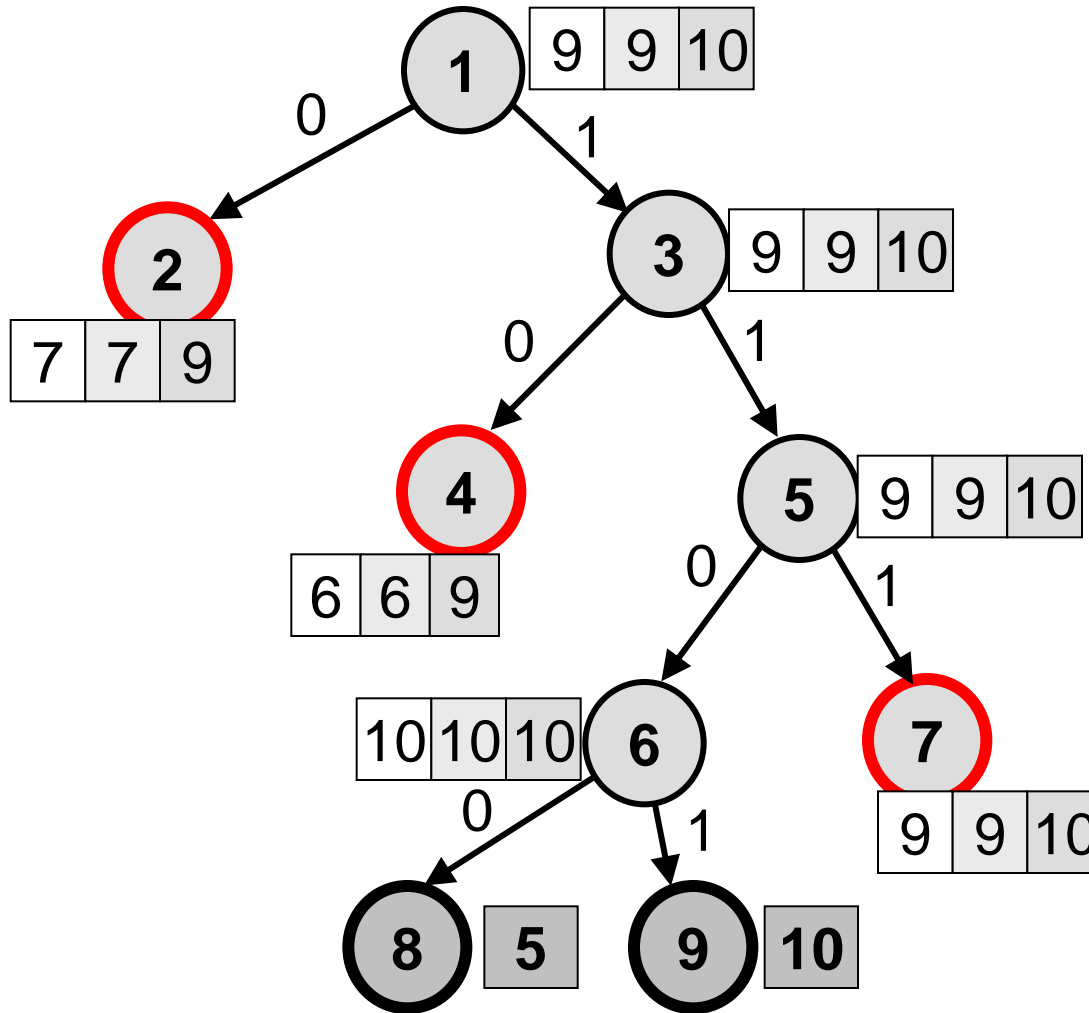
finsi

finpara

finmientras

6.3.1. Problema de la mochila 0/1

- Ejemplo. $n=4$, $M=7$, $b=(2, 3, 4, 5)$, $p=(1, 2, 3, 4)$



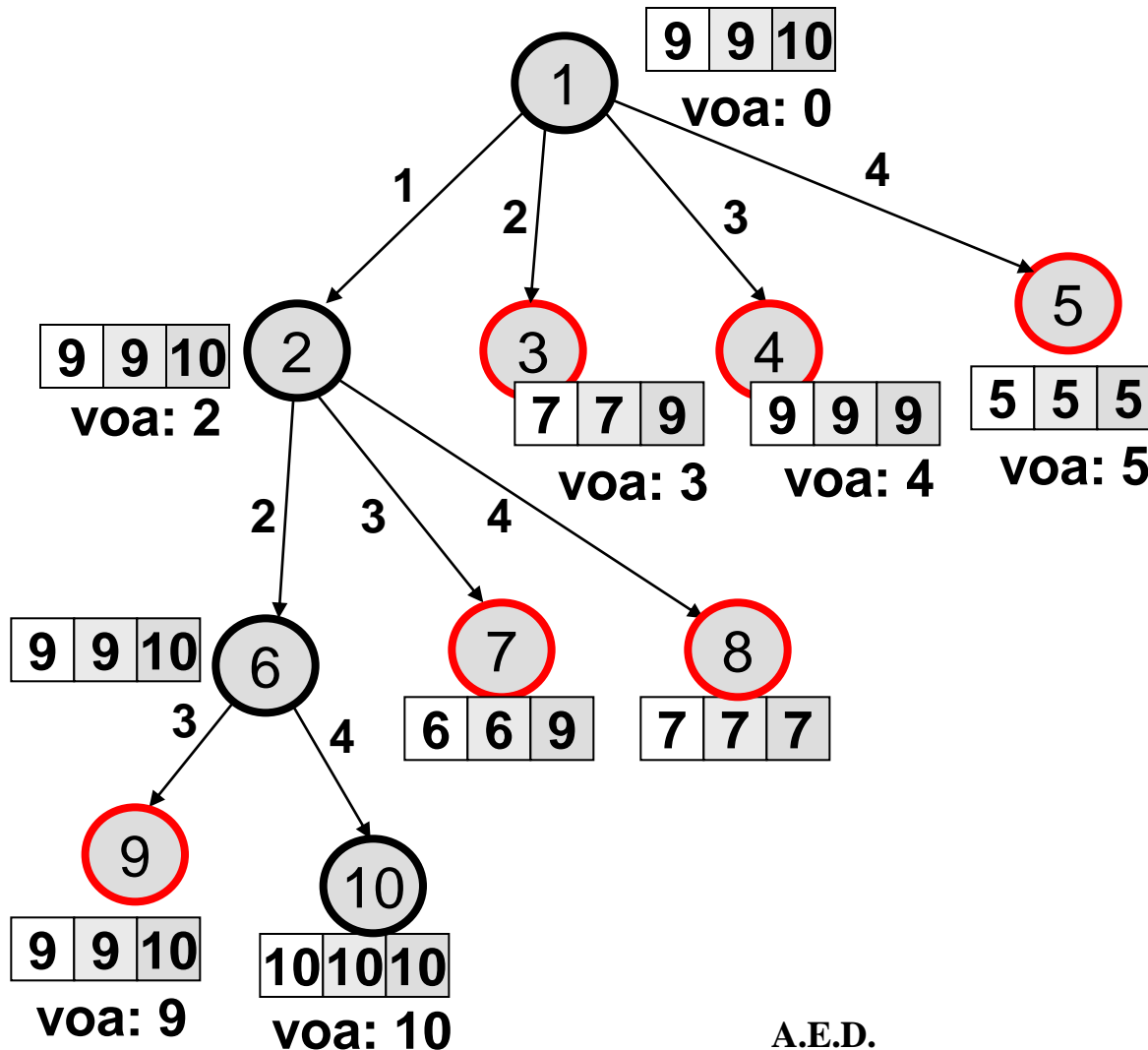
C	LNV
9	1
9	3
9	5
10	6
$s = (1, 1, 0, 1)$	

6.3.1. Problema de la mochila 0/1

- **Ojo:** si el nodo x es tal que $CI(x) = CS(x)$, entonces se poda a sí mismo, antes de haber generado sus descendientes.
- ¿Cómo solucionarlo?
- **Posibilidad 1.** Cambiar la condición de poda:
Podar i si: $i.CS < C$
- **Posibilidad 2.** Usar dos variables de poda C , voa :
 voa : valor óptimo actual
Podar i si: $(i.CS < C)$ OR $(i.CS \leq voa)$
- **Posibilidad 3.** Generar directamente el nodo solución:
si $y.CI == y.CS$ entonces
 $y := SolucionMochilaVoraz(y.nivel+1, n, M-y.pact)$

6.3.1. Problema de la mochila 0/1

- Ejemplo.** Utilizando un árbol combinatorio y LC-FIFO.
 $n = 4$, $M = 7$, $b = (2, 3, 4, 5)$, $p = (1, 2, 3, 4)$



C	LNV
9	1
9	2
9	6
10	9

$s = (1, 2, 4)$

6.3.1. Problema de la mochila 0/1

- ¿Cuánto es el orden de complejidad del algoritmo, en el peor caso?
- ¿Y en el mejor caso? ¿Y en promedio?
- En los ejemplos anteriores el algoritmo encuentra la solución muy rápidamente, pero...
- ¿Ocurrirá siempre así?
- **Ejemplo.** $n= 101$, $M= 155$
 $\mathbf{b} = (3, 3, 3, 3, 3, 3, 3, 3, 3, 3, \dots, 3, 3, 3, 4)$
 $\mathbf{p} = (3, 3, 3, 3, 3, 3, 3, 3, 3, 3, \dots, 3, 3, 3, 5)$
- **Problema:** CI, CS y BE son poco informativas.

6.3.2. Problema de asignación

Enunciado del problema de asignación

- **Datos del problema:**

- **n**: número de personas y de tareas disponibles.
- **B**: array [1..n, 1..n] de entero. Rendimiento o beneficio de cada asignación. **B[i, j]** = beneficio de asignar a la persona **i** la tarea **j**.

- **Resultado:**

- Realizar **n** asignaciones $\{(p_1, t_1), (p_2, t_2), \dots, (p_n, t_n)\}$.

- **Formulación matemática:**

Maximizar $\sum_{i=1..n} B[p_i, t_i]$, sujeto a

la restricción $p_i \neq p_j, t_i \neq t_j, \forall i \neq j$

		Tareas			
		B	1	2	3
Personas	1	5	6	4	
	2	3	8	2	
	3	6	5	1	

6.3.2. Problema de asignación

1) Representación de la solución

- Desde el punto de vista de las **personas**:
 $s = (t_1, t_2, \dots, t_n)$, siendo $t_i \in \{1, \dots, n\}$, con $t_i \neq t_j, \forall i \neq j$
 - $t_i \rightarrow$ número de tarea asignada a la persona i .

tipo

Nodo = registro

tupla: **array** [1..n] **de** entero

nivel: entero

bact: entero

CI, BE, CS: entero

finregistro

1.a) ¿Cómo es el nodo raíz?

1.b) ¿Cómo generar los hijos de un nodo?

1.c) ¿Cómo es la función **Solución(x: Nodo): booleano**?

6.3.2. Problema de asignación

1.a) Nodo raíz

raiz.nivel:= 0

raiz.bact:= 0

1.b) Para cada y hijo de un nodo x

para $i:= 1, \dots, n$ hacer

 y.nivel:= x.nivel+1

 y.tupla:= x.tupla

 si Usada(x, i) entonces break

 y.tupla[y.nivel]:= i

 y.bact:= x.bact + B[y.nivel, i]

.....

operación Usada(m: Nodo; t: entero): booleano

 para $i:= 1, \dots, m.nivel$ hacer

 si m.tupla[i]==t entonces devolver TRUE

 devolver FALSE

6.3.2. Problema de asignación

- **Otra posibilidad:** almacenar las tareas usadas en el nodo.
tipo

Nodo = registro

tupla: **array** [1..n] **de** entero

nivel: entero

bact: entero

usadas: **array** [1..n] **de** booleano

CI, BE, CS: entero

finregistro

- **Resultado:** se tarda menos tiempo pero se usa más memoria.

1.c) Función Solución(x: Nodo): booleano
devolver x.nivel==n

6.3.2. Problema de asignación

2) Cálculo de las funciones $CI(x)$, $CS(x)$, $BE(x)$

		Tareas			
Personas		B	1	2	3
1		5	6	4	
2		3	8	2	
3		6	5	1	

2) Posibilidad 1. Estimaciones triviales:

- **CI.** Beneficio acumulado hasta ese momento: $x.CI := x.bact$
- **CS.** CI más suponer las restantes asignaciones con el máximo global: $x.CS := x.bact + (n-x.nivel) * \max(B[\cdot, \cdot])$
- **BE.** La media de las cotas: $x.BE := (x.CI + x.CS) / 2$

6.3.2. Problema de asignación

2) Posibilidad 2. Estimaciones precisas:

- **CI.** Resolver el problema usando un algoritmo voraz.
 $x.CI := x.bact + \text{AsignaciónVoraz}(x)$
- **AsignaciónVoraz(x):** asignar a cada persona la tarea libre con más beneficio.

operación AsignaciónVoraz(m: Nodo): entero

 bacum := 0

para i := m.nivel+1, ..., n **hacer**

 k := argmax <sub>$\forall j \in \{1..n\}$
 m.usadas[j] == FALSE</sub> B[i, j]

 m.usadas[k] := TRUE

 bacum := bacum + B[i, k]

finpara

devolver bacum

6.3.2. Problema de asignación

2) Posibilidad 2. Estimaciones precisas:

- **CS.** Asignar las tareas con mayor beneficio (aunque se repitan).

$x.CS := x.bact + \text{MáximosTareas}(x)$

operación MáximoTareas(m: Nodo): entero

 bacum := 0

para i := m.nivel+1, ..., n **hacer**

 k := argmax _{$\forall j \in \{1..n\}$} B[i, j]
 m.usadas[j] == FALSE

 bacum := bacum + B[i, k]

finpara

devolver bacum

- **BE.** Tomar la media: $x.BE := (x.CI + x.CS) / 2$

6.3.2. Problema de asignación

2) Cálculo de las funciones $CI(x)$, $CS(x)$, $BE(x)$

- **Cuestión clave:** ¿podemos garantizar que la solución óptima a partir de x estará entre $CI(x)$ y $CS(x)$?
- **Ejemplo.** $n=3$. ¿Cuánto serían $CI(\text{raíz})$, $CS(\text{raíz})$ y $BE(\text{raíz})$? ¿Cuál es la solución óptima del problema?

		Tareas			
		B	1	2	3
Personas	1	5	6	4	
	2	3	8	2	
	3	6	5	1	

6.3.2. Problema de asignación

3) Estrategia de ramificación y de poda

3.a) Estrategia de poda

- **Variable de poda C:** valor de la mayor cota inferior o solución final del problema.
- **Condición de poda:** podar i si: $i.CS \leq C$

3.b) Estrategia de ramificación

- **Usar una estrategia MB-LIFO:** explorar primero los nodos con mayor BE y en caso de empate seguir por la rama más profunda.

6.3.2. Problema de asignación

4) Esquema del algoritmo. (Exactamente el mismo que antes)

AsignaciónRyP (n: ent; B: array[1..n,1..n] de ent; var s: Nodo)

LNV:= {raiz}

C:= raiz.Cl

s:= \emptyset

mientras LNV \neq \emptyset **hacer**

 x:= Seleccionar(LNV) // Estrategia MB-LIFO

 LNV:= LNV - {x}

si x.CS > C **entonces** // Estrategia de poda

para cada y hijo de x hacer

si Solución(y) AND (y.bact > s.bact) **entonces**

 s:= y

 C:= max (C, y.bact)

sino si NO Solución(y) AND (y.CS > C) **entonces**

 LNV:= LNV + {y}

 C:= max (C, y.Cl)

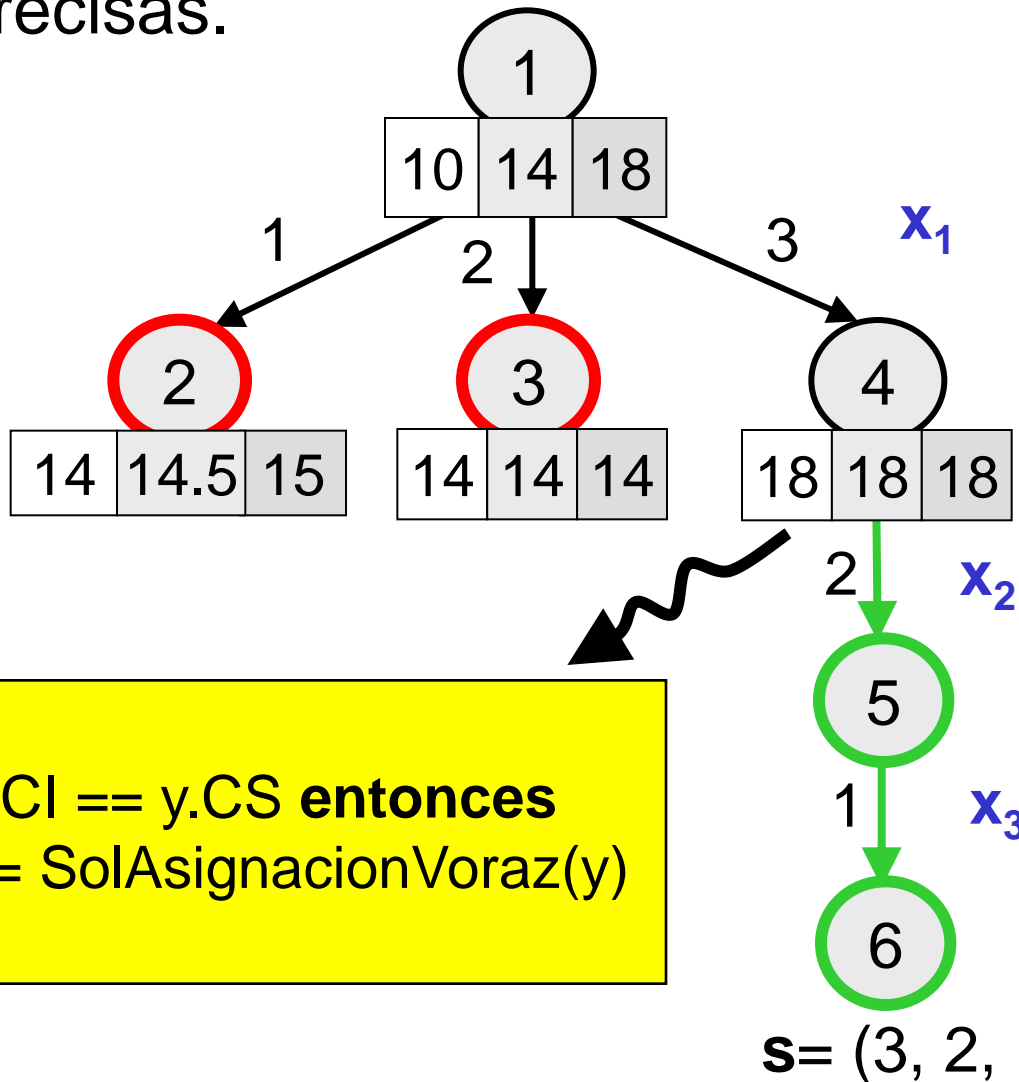
finsi

finpara

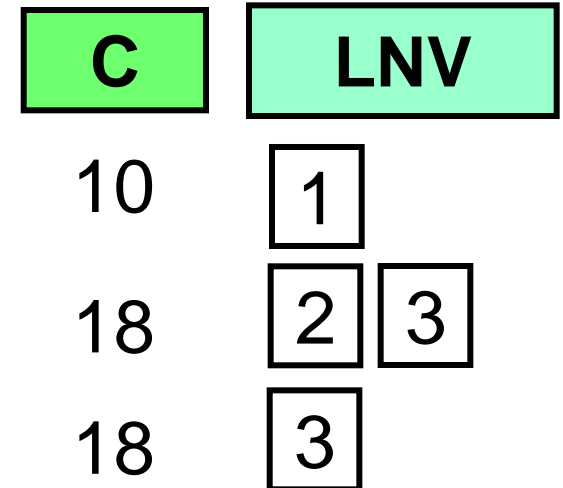
finmientras

6.3.2. Problema de asignación

- Ejemplo. $n=3$. Estimaciones precisas.

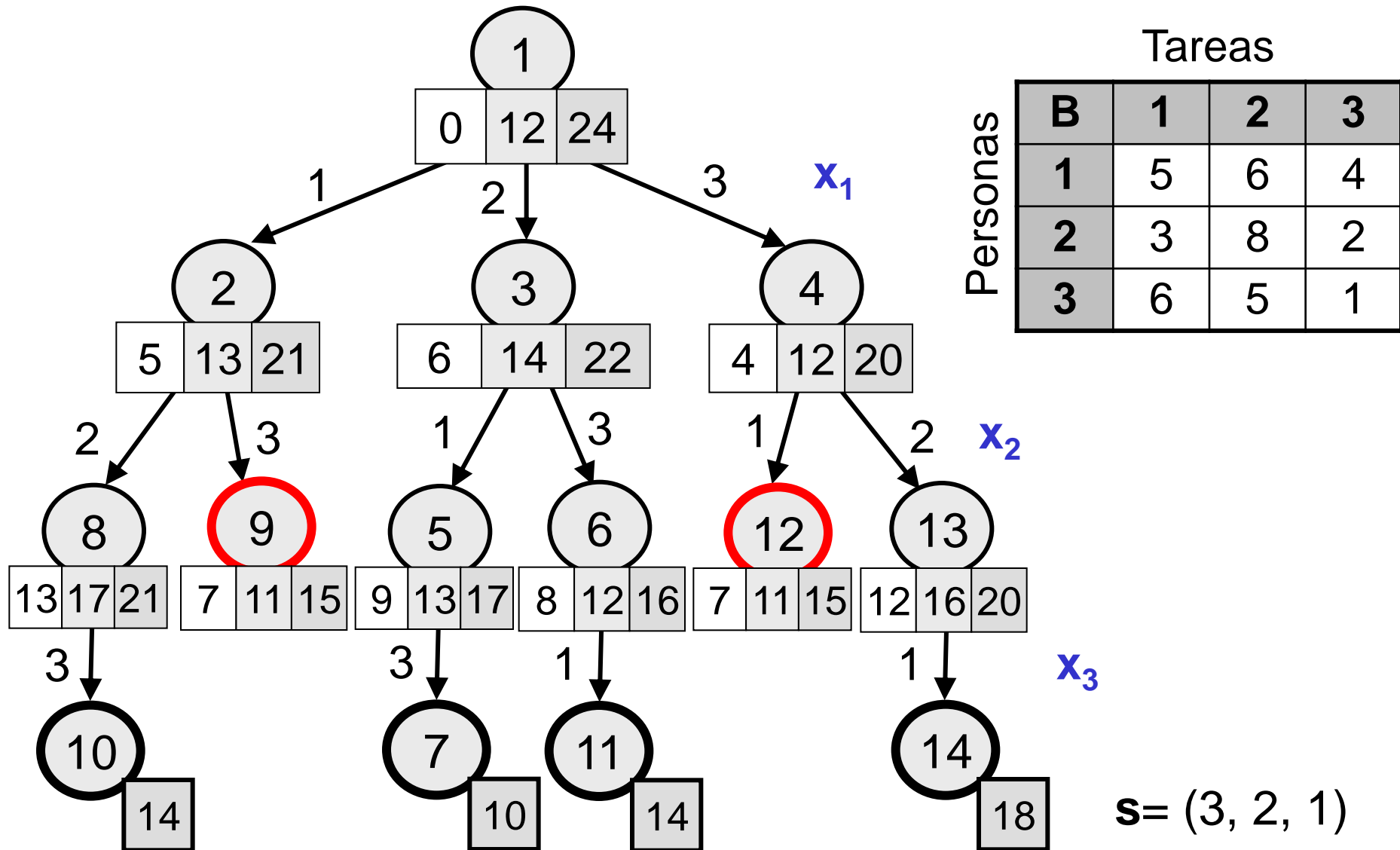


		Tareas		
Personas	B	1	2	3
	1	5	6	4
	2	3	8	2
	3	6	5	1



6.3.2. Problema de asignación

- Ejemplo. $n=3$. Usando las estimaciones triviales.



A.E.D.

6.3.2. Problema de asignación

- Con estimaciones precisas: 4 nodos generados.
- Con estimaciones triviales: 14 nodos generados.
- ¿Conviene gastar más tiempo en hacer estimaciones más precisas?
- ¿Cuánto es el tiempo de ejecución en el peor caso?
- Estimaciones triviales: **$O(1)$**
- Estimaciones precisas: **$O(n\text{-nivel})$**

6. Ramificación y poda

Conclusiones

- **Ramificación y poda:** mejora y generalización de la técnica de backtracking.
- **Idea básica.** Recorrido implícito en árbol de soluciones:
 - Distintas estrategias de ramificación.
 - Estrategias LC: explorar primero las ramas más prometedoras.
 - Poda basada en acotar el beneficio a partir de un nodo: CI, CS.
- **Estimación de cotas:** aspecto clave en RyP. Utilizar algoritmos de avance rápido.
- **Compromiso tiempo-exactitud.** Más tiempo → mejores cotas. Menos tiempo → menos poda.