Estrategias algorítmicas

Tema 3(IV)

Algorítmica y Modelos de Computación

1. Introducción.
2. Algoritmos divide y vencerás.
3. Algoritmos voraces.
4. Programación dinámica.
5. Algoritmos Bactracking (vuelta atrás).
6. Ramificación y poda (branch and bound).

6. Ramificación y poda (branch and bound).

1. Introducción.
2. Método general.
3. Análisis de tiempos de ejecución.
4. Ejemplos de aplicación.
   4.1. Problema de la mochila 0-1.
   4.2. Problema de la asignación.

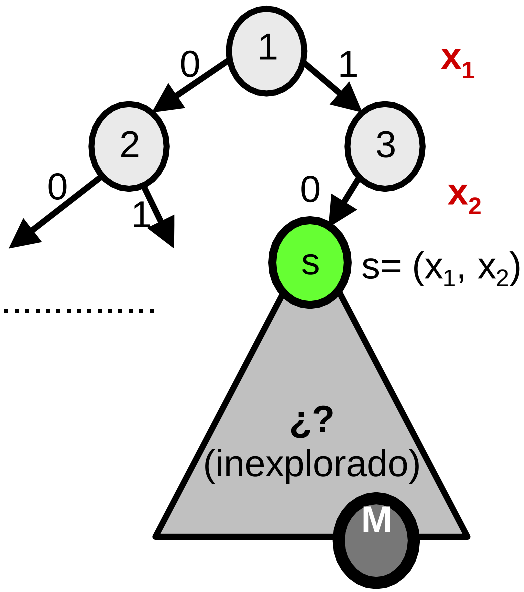
6. Ramificación y poda (branch and bound). Introducción.

- 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. Ramificación y poda. 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) ≤ Valor(M) ≤ CS(s)**

M= (x_1, x_2, x_3, x_4,..., x_n)
Valor(M) = ¿?

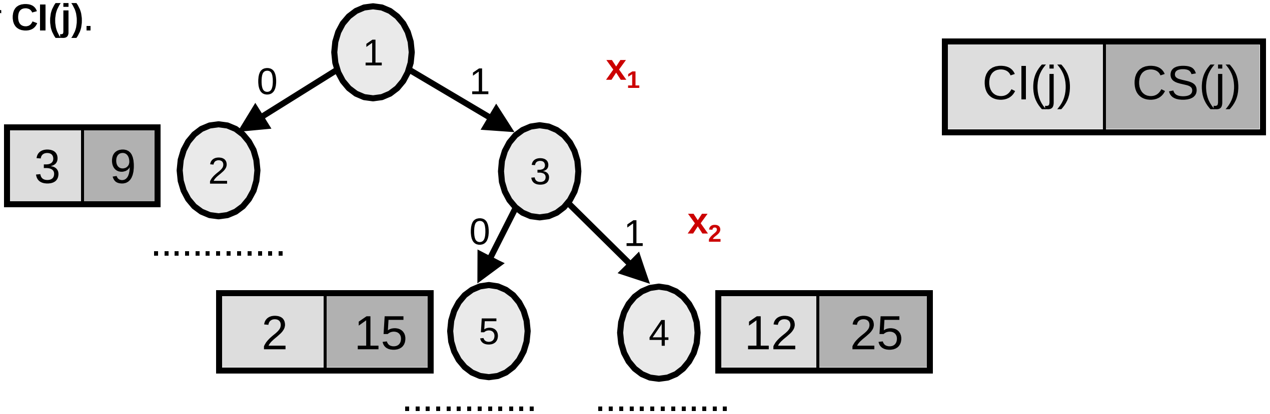


- **Para cada nodo s** tendremos:
  - **CS(s): Cota superior** del beneficio (o coste) óptimo que podemos alcanzar a partir del nodo **s**.
  - **CI(s): Cota inferior** del beneficio (o coste) óptimo que podemos alcanzar a partir del nodo **s**.
  - **BE(s): Beneficio estimado** (o coste) óptimo que se puede encontrar a partir del nodo **s**.

- Las cotas **CS(s)**, **CI(s)** deben ser “fiables”: determinan cuándo se puede podar.
- El beneficio (o coste) estimado **BE(s)** ayuda a decidir qué parte del árbol evaluar primero.
- Poda según los valores de **CI(s)** y **CS(s)** y ramificación según valores de **BE(s)**

6. Ramificación y poda. Método general.

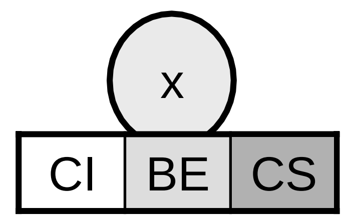
- **Estrategia de poda**
  - Supongamos un problema de **maximización**.
  - Hemos recorrido varios nodos, estimando para cada uno, la cota superior **CS(j)** e inferior **CI(j)**.



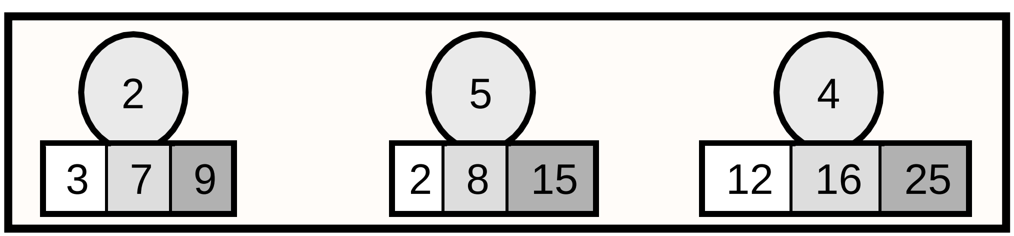
- ¿Merece la pena seguir explorando por el nodo 2? ¿Y por el 5? podar 2 porque CS(2)=9 ≤ CI(4)=12
- **Estrategia de poda (maximización).** Podar un nodo **i** si se cumple que:
  - **CS(i)** ≤ **CI(j)**, para algún nodo **j** generado
    o bien
  - **CS(i)** ≤ **Valor(s)**, para algún nodo **s** solución final
- **Implementación.** Usar una variable de poda **C**:
  C = max({**CI(j)** | ∀ **j** generado}, {**Valor(s)** | ∀ **s** solución final})
  - Podar **i** si: **CS(i)** ≤ **C**
- ¿Cómo sería para el caso de minimización? **C** = min({**CS(j)** | ∀ **j** generado}, {**Valor(s)** | ∀ **s** solución final}). Podar **i** si: **CI(i)** ≥ **C**

6. Ramificación y poda. Método general.

- **Estrategias de ramificación**
  - Igual que en backtracking, hacemos un recorrido en un **árbol** de soluciones (que es **implícito** → no se genera).
  - **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.
- **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

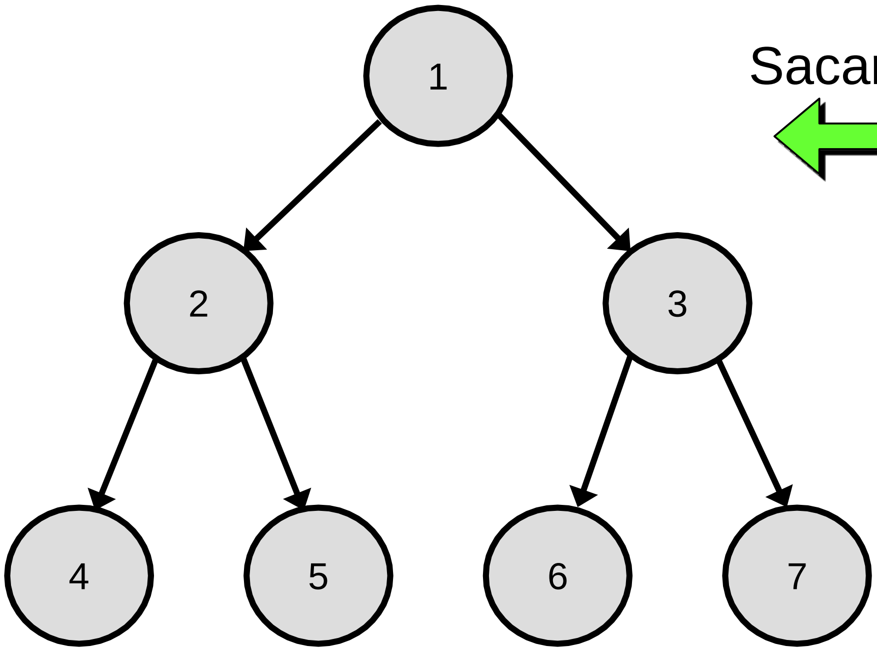


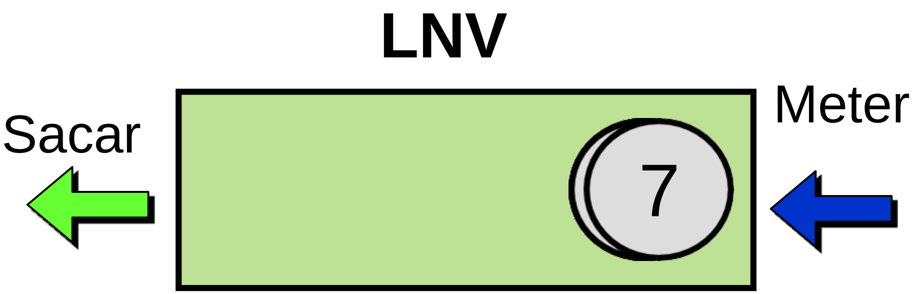
- ¿En qué orden se sacan y se meten?
- Según cómo se maneje esta lista, el recorrido será de uno u otro tipo.

6. Ramificación y poda. 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 en anchura

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



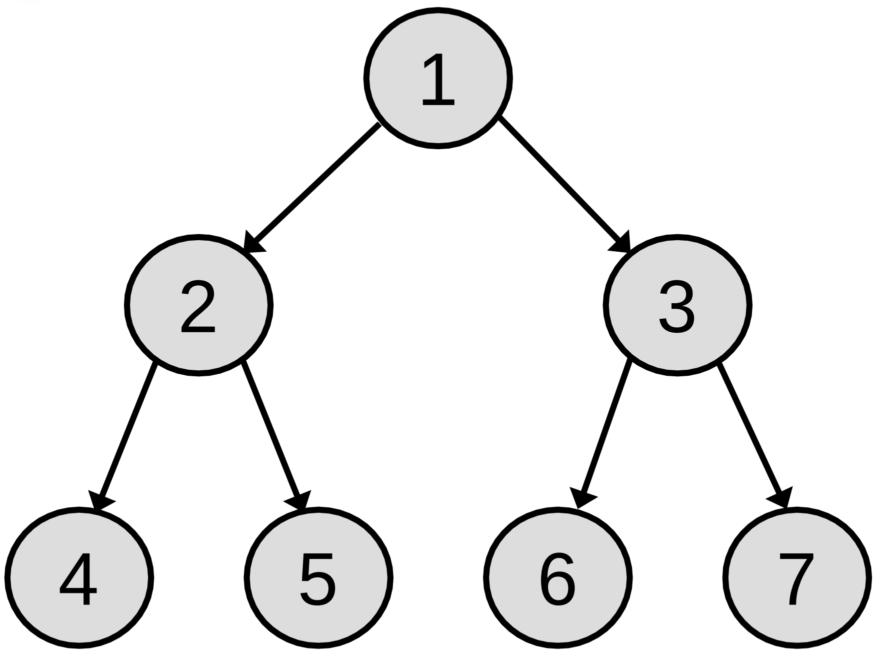


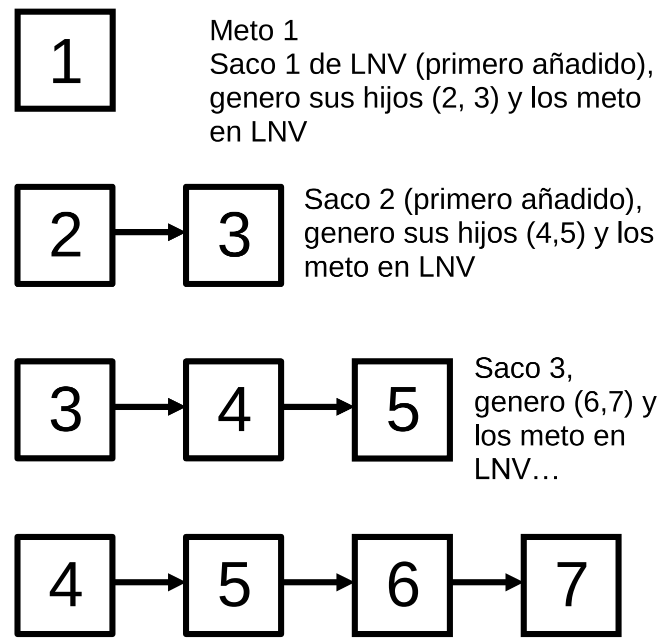
Estrategia de ramificación FIFO (First In First Out)

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

Idea básica del algoritmo:

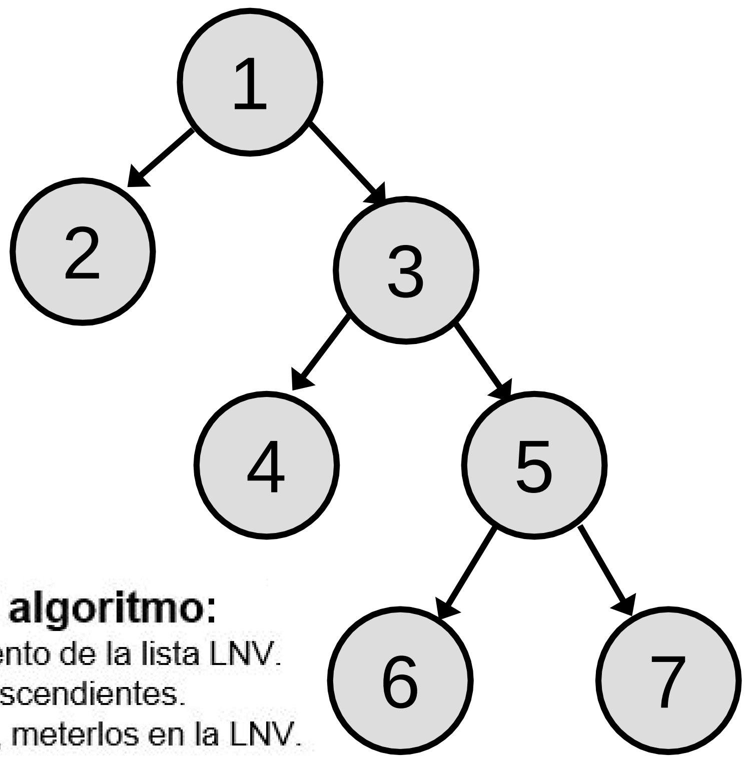
- Sacar un elemento de la lista LNV.
- Generar sus descendientes.
- Si no se podan, meterlos en la LNV.

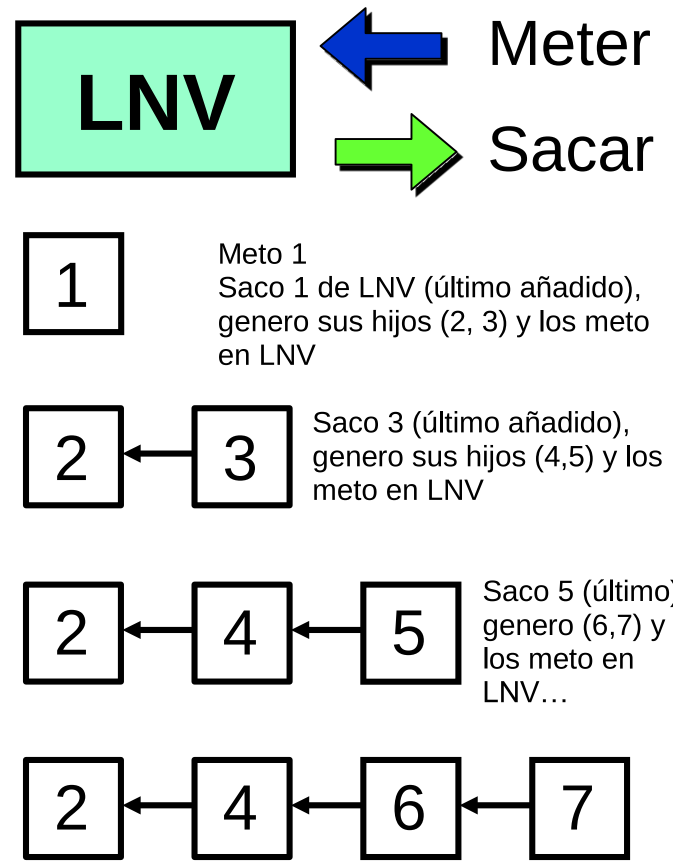




6. Ramificación y poda. Método general.

- **Estrategia de ramificación LIFO (Last In First Out)**
  - Si se usa la estrategia LIFO, la LNV es una **pila** y el recorrido es en profundidad



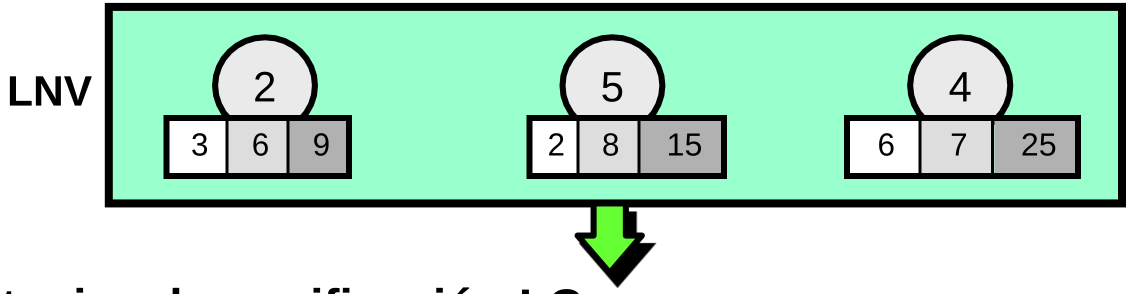


Idea básica del algoritmo:

- Sacar un elemento de la lista LNV.
- Generar sus descendientes.
- Si no se podan, meterlos en la LNV.

6. Ramificación y poda. 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 LNV, elegir el que tenga mayor beneficio (MB) (o menor coste LC) para explorar a continuación.



- **Estrategias de ramificación LC**
  - En caso de empate (de beneficio o coste estimado) deshacerlo usando un criterio **FIFO** ó **LIFO**.
  - **Estrategia LC-FIFO:** Seleccionar de LNV el nodo que tenga mayor beneficio (menor coste) 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 (menor coste) 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 (BE).

6. Ramificación y poda. 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 **BE** usando **LC-FIFO** o **LC-LIFO**.

- **Ejemplo.** Recorrido con **ramificación y poda,** usando **LC-FIFO.**
  - Suponemos un problema de minimización (maximización).
  - Para realizar la poda usamos una variable **C** = valor de la menor (mayor) de las cotas superiores (inferiores) hasta ese momento, o de alguna solución final.
    - **C** = min({**CS(j)** | ∀ **j** generado}, {**Valor(s)** | ∀ **s** solución final})
    - **C** = max({**CI(j)** | ∀ **j** generado}, {**Valor(s)** | ∀ **s** solución final})
  - Si para algún nodo **i**, **CI(i) ≥ C** (**CS(i) ≤ C**), entonces podar **i**.

Ej: Recorrido con ramificación y poda, usando LC-FIFO (minimización)

LC-FIFO: Seleccionar de **LNV** nodo con mayor beneficio (menor coste), si empate escoger el primero que se metió

C = valor de la menor de las cotas superiores hasta ese momento

Si para algún nodo **i**, **CI(i) ≥ C**, entonces podar **i**

Al sacar 4, CI(4)=4 < C=10 → genero 6 y 7, que son solución final **s** → No se meten en LNV

Valor(6)=5 y **Valor**(7)=6 → C=5

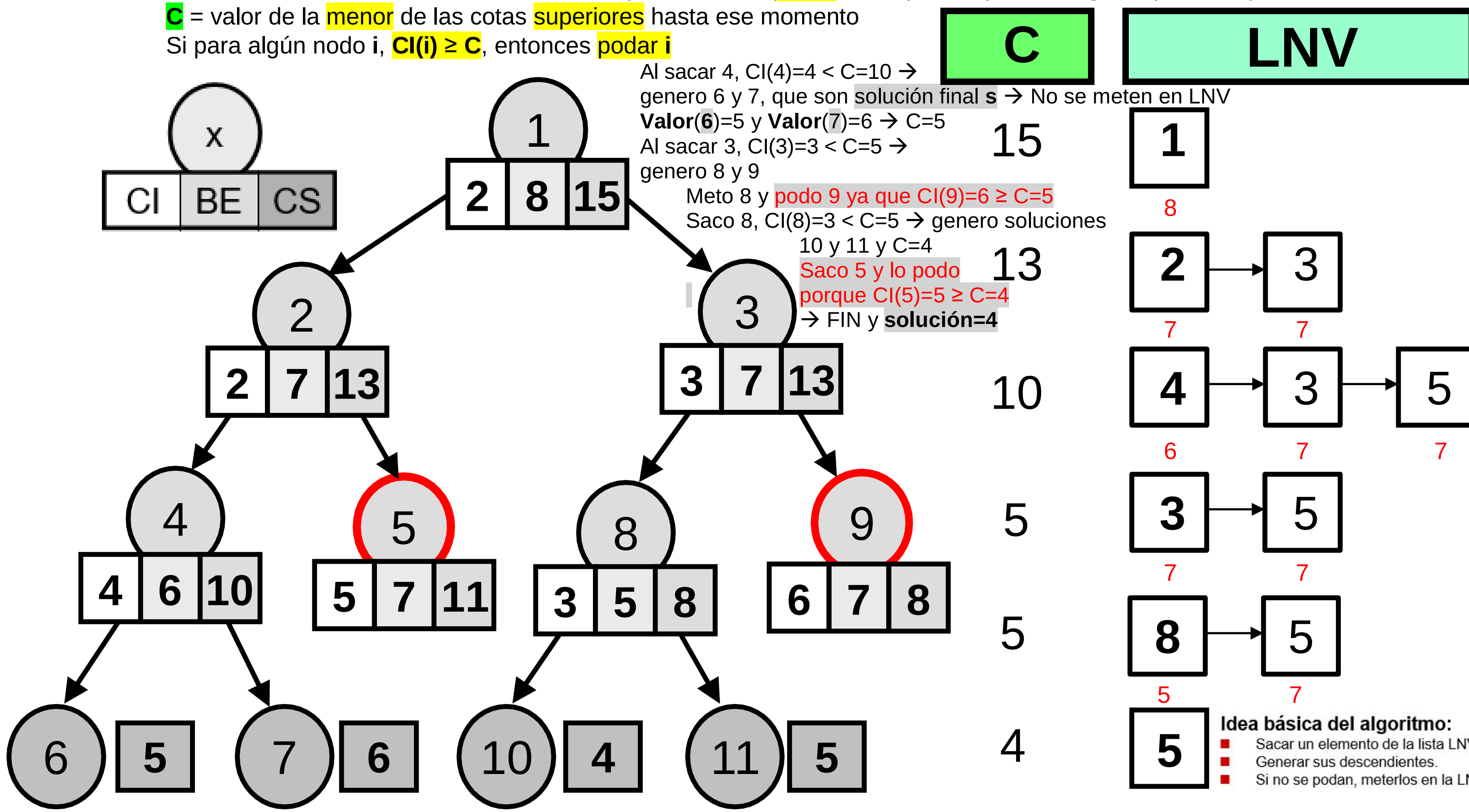
Al sacar 3, CI(3)=3 < C=5 → genero 8 y 9

Meto 8 y podo 9 ya que CI(9)=6 ≥ C=5

Saco 8, CI(8)=3 < C=5 → genero soluciones 10 y 11 y C=4

Saco 5 y lo podo porque CI(5)=5 ≥ C=4

→ FIN y **solución=4**



Idea básica del algoritmo:

- Sacar un elemento de la lista LNV.
- Generar sus descendientes.
- Si no se podan, meterlos en la LNV.

6. Ramificación y poda. 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. Ramificación y poda. Método general.

- **Algoritmo de ramificación y poda.**

RamificacionYPoda (raiz: Nodo; var s: Nodo) // *Minimización* *Maximizacion*

> **LC-FIFO:** Seleccionar de **LNV** el nodo con mayor beneficio (menor coste), en caso de empate escoger el primero que se introdujo
> **C** = valor de la menor mayor de las cotas superiores inferiores hasta ahora
> Si para algún nodo **i**, **CI(i) ≥ C** **CS(i) ≤ C**, entonces podar **i**

```
LNV:= {raiz}
C:= CS(raiz)
s:= ∅
mientras LNV ≠ ∅ hacer
    x:= Seleccionar(LNV)                    // Estrategia ramificación
    LNV:= LNV - {x}
    si CI(x) < C  CS(x) > C entonces        // Estrategia poda (ci(x)≥C cs(x)≤c poda)
        para cada y hijo de x hacer
            si Solución(y) AND (Valor(y)< > Valor(s)) entonces
                s:= y
                C:= min max (C, Valor(y))
            sino si NO Solución(y) AND (CI(y) < C  CS(y)>C) entonces
                LNV:= LNV + {y}
                C:= min max (C, CS(y) CI(y))
            finsi
        finpara
finmientras
```

6. Ramificación y poda. 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(nivel(x)+1, y) **hacer**

        Generar(nivel(x)+1, y)

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

6. Ramificación y poda. 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
  finregistro

  Almacenar para no recalcular.
  ¿Todos?

- ¿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. Ramificación y poda. 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.
- **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. Ramificación y poda. 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.

- Datos del problema:
  - **n**: número de objetos disponibles.
  - **M**: capacidad de la mochila.
  - **p = (p_1, p_2, ..., p_n)** pesos de los objetos.
  - **b = (b_1, b_2, ..., b_n)** beneficios de los objetos.
- Formulación matemática:

  Maximizar $\sum_{i=1}^{n} x_i b_i$;

  sujeto a la **restricción**

  $\sum_{i=1}^{n} x_i p_i \leq \mathrm{M}$ y $x_i \in \{0,1\}$
- **Ejemplo:** $n = 4$; $M = 7$;
  $b = (2, 3, 4, 5)$; $p = (1, 2, 3, 4)$

1. Representación de la solución.

Con un árbol binario:

s= (x_1, x_2, ..., x_n), con $x_i \in \{0,1\}$

- **x_i = 0** → No se coge el objeto **i**;
- **x_i = 1** → 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?

1.a) Nodo raíz

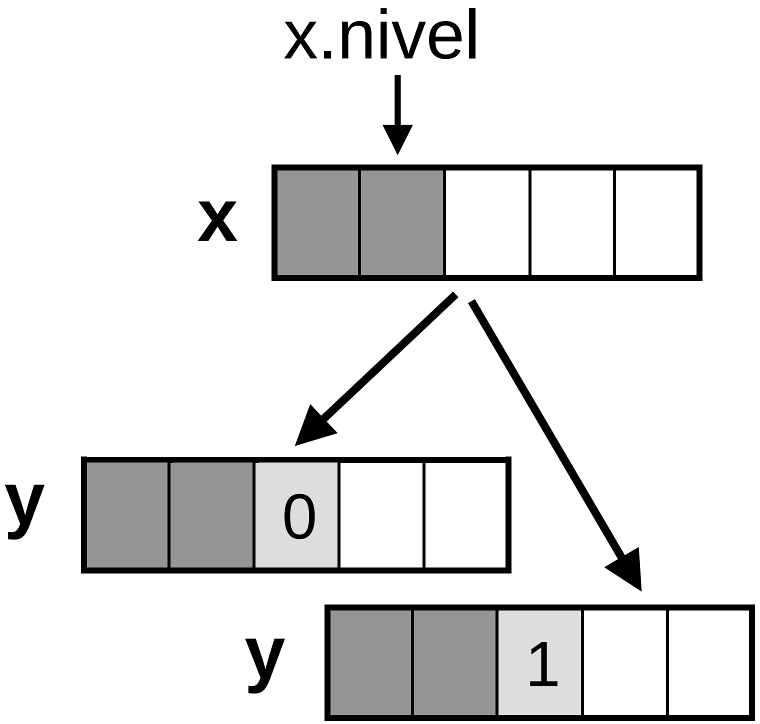
raiz.nivel:= 0
raiz.bact:= 0
raiz.pact:= 0

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

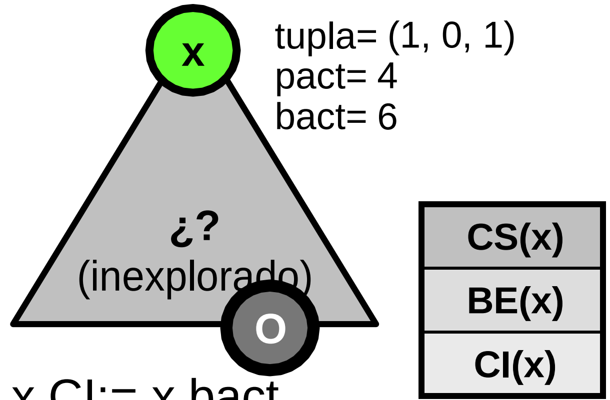
para i:= 0, 1 **hacer**
y.nivel:= x.nivel+1
y.tupla:= x.tupla
y.tupla[y.nivel]:= i
y.bact:= x.bact + i * b[y.nivel]
y.pact:= x.pact + i * p[y.nivel]
si y.pact > M **entonces break**
....

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

devolver x.nivel==n



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



n= 6, M= 11
b= (2, 3, 4, 5, 6, 7)
p= (1, 2, 3, 4, 5, 6)
b/p=(2, 1. 5, 1.3, 1.25, 1.2, 1.16)

2.a) Cálculo de CI(x)

Posibilidad 1. El beneficio acumulado hasta ese momento: x.CI:= x.bact

2.b) Cálculo de CS(x)

- **Idea** (igual que con **backtracking**)**:** la solución de la mochila (no 0/1) **MochilaVorazNo01**[1] es una cota superior válida de la mochila 0/1.
  x.CS:= x.bact + ⌊**MochilaVorazNo01**(x.nivel+1, n, M-x.pact)⌋

  MochilaVorazNo01[1]**(a, b, Q):** Problema de la mochila no 0/1 de peso Q con los objetos (a, ..., b).

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 + **MochilaVoraz01**(x.nivel+1, n, M-x.pact)

[1] dados b y p, el vorazNo01 coge 1º los objetos con mayor b/p (entero si cabe o la fracción que quepa, que será el último de los metidos)

2.c) Cálculo de BE(x)

```
MochilaVoraz01 (a, b, Q): entero
  bacum:= 0
  pacum:= 0
  para i:= a hasta b hacer
    si pacum + p[i] ≤ Q entonces
      pacum:= pacum + p[i]
      bacum:= bacum + b[i]
    finsi
  finpara
  devolver bacum
```

CI(x) = x.**bact** (beneficio acumulado hasta ese momento)
CS(x) = x.**bact +** $\lfloor$ **MochilaVorazNo01**(x. nivel+1, n, M-x.pact) $\rfloor$
BE(x) = x.**bact +** $\lfloor$ **MochilaVoraz01**(x. nivel+1, n, M-x.pact) $\rfloor$

MochilaVorazNo01 escoge 1º los objetos con mayor b/p (entero si cabe o la fracción que quepa)

MochilaVoraz01 escoge 1º los objetos con mayor b/p (entero si cabe, si no cabe entero no lo escoge)

$\lfloor$ **MochilaVorazNo01** (4, 6, 11-4) $\rfloor$
p[4..6]=4+3/5 5=7
b[4..6]=5+3/5 6=8.6 → **6**+8
CS(x) = 6 + $\lfloor$ **8.6** $\rfloor$ → **6**+8 → **CS(x) = 14**

$\lfloor$ **MochilaVoraz01** (4, 6, 11-4) $\rfloor$
p[4..6]=4
b[4..6]=5 → **5**
BE(x) = 6 + $\lfloor$ **5** $\rfloor$ → **6+5** → **BE(x) = 11**

- **NOTA:** se supone que los objetos están ordenados por b/p.
- **Ejemplo.** Cálculo de las funciones **CI(x), CS(x), BE(x)** cuando tupla **x= (1, 0, 1)** → pact=4 bact=6
  n= 6 M= 11 p= (1, 2, 3, 4, 5, 6) b= (2, 3, 4, 5, 6, 7)
  - ¿Cuánto valen **CI(x)**, **BE(x), CS(x)**? → 6, 11, 14
  - ¿Cuánto es la solución óptima? ¿Son buenas las funciones? → x=(1,1,1,0,1,0) solución óptima=15
  - **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?

3. Estrategia de ramificación y de poda ***Maximizacion***

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 ≤ 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)

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

MB-LIFO: mayor beneficio-LNV es una pila (1º en entrar 1º en salir) → recorrido en profundidad

- **Algoritmo de ramificación y poda.**

//minimización

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

MB-LIFO: Seleccionar de **LNV** el nodo con mayor beneficio (menor coste), en caso de empate escoger el último que se introdujo
C = valor de la menor mayor de las cotas superiores inferiores hasta ahora
Si para algún nodo **i**, **CI(i) ≥ C** **CS(i) ≤ C**, entonces podar **i**

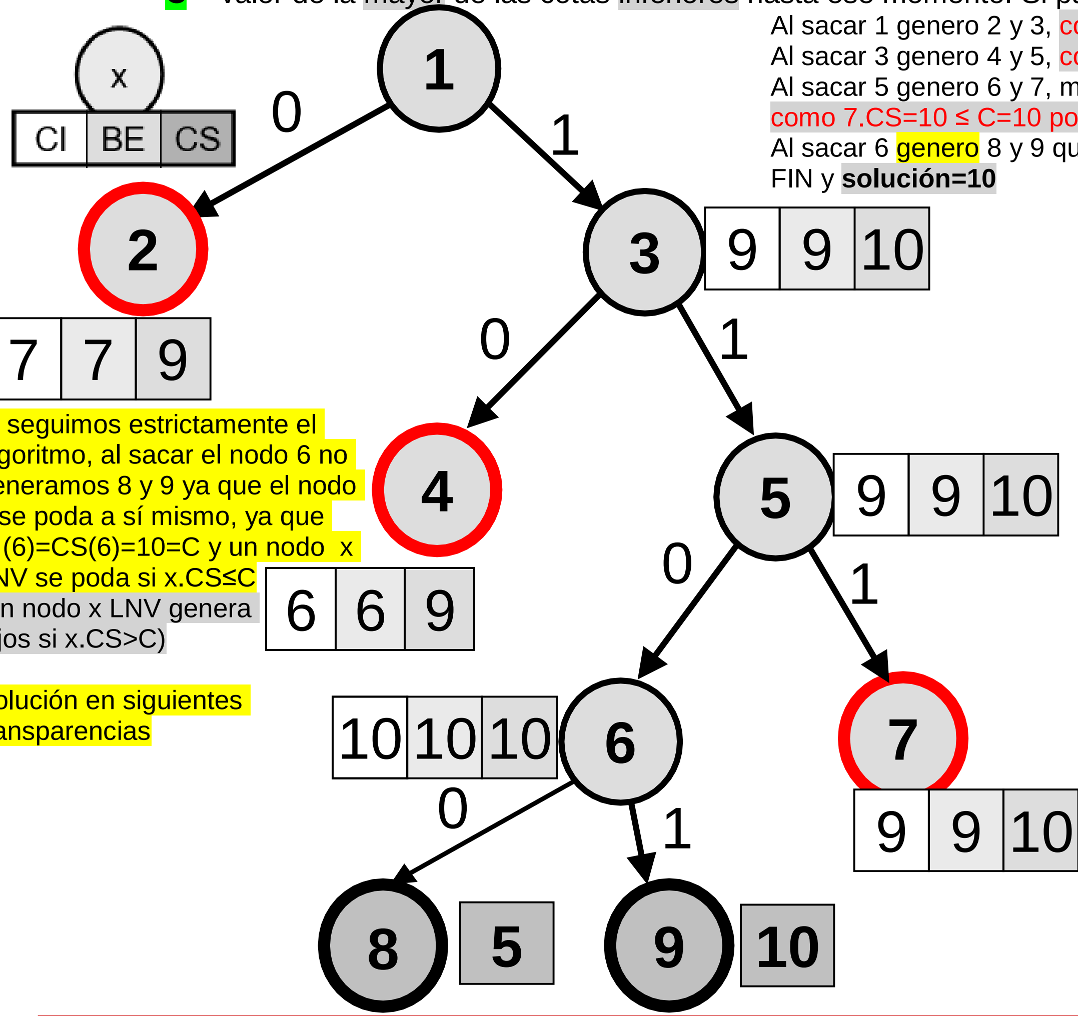
```
LNV:= {raiz}
C:= raiz.CI
s:= ∅
mientras LNV ≠ ∅ hacer
    x:= Seleccionar(LNV)       // Estrategia de ramificación MB-LIFO
    LNV:= LNV - {x}
    si x.CI < C  x.CS > C entonces // Estrategia de poda (x.CI≥C  x.CS≤C poda)
        para cada y hijo de x hacer
            si Solución(y) AND (y.bact < > s.bact) entonces
                s:= y
                C:= min max (C, y.bact)
            sino si NO Solución(y) AND (y.CI < C  y.CS > C) entonces
                LNV:= LNV + {y}
                C:= min  max (C, y.CS  y.CI)
            finsi
        finpara
finmientras
```

- **Ejemplo.** n= 4, M= 7, b= (2, 3, 4, 5), p= (1, 2, 3, 4) (**x.BE** se usa como valor **x.CI**)

MB-LIFO: Seleccionar de **LNV** nodo con mayor beneficio (menor coste), si empate escoger el último que se metió

C = valor de la mayor de las cotas inferiores hasta ese momento. Si para algún nodo **x**, **x.CS ≤ C**, entonces podar **x**

Al sacar 1 genero 2 y 3, como 2.CS=9 ≤ C=9 podo 2 y solo meto 3 (3.CS=10 > C=9)
Al sacar 3 genero 4 y 5, como 4.CS=9 ≤ C=9 podo 4 y solo meto 5 (5.CS=10 > C=9)
Al sacar 5 genero 6 y 7, meto 6 (6.CS=10 > C=9) y actualizo C=max(C,6.CI) → C=10
como 7.CS=10 ≤ C=10 podo 7
Al sacar 6 genero 8 y 9 que son solución final **s Valor**(8)=5 y **Valor**(9)=10 →
FIN y **solución=10**



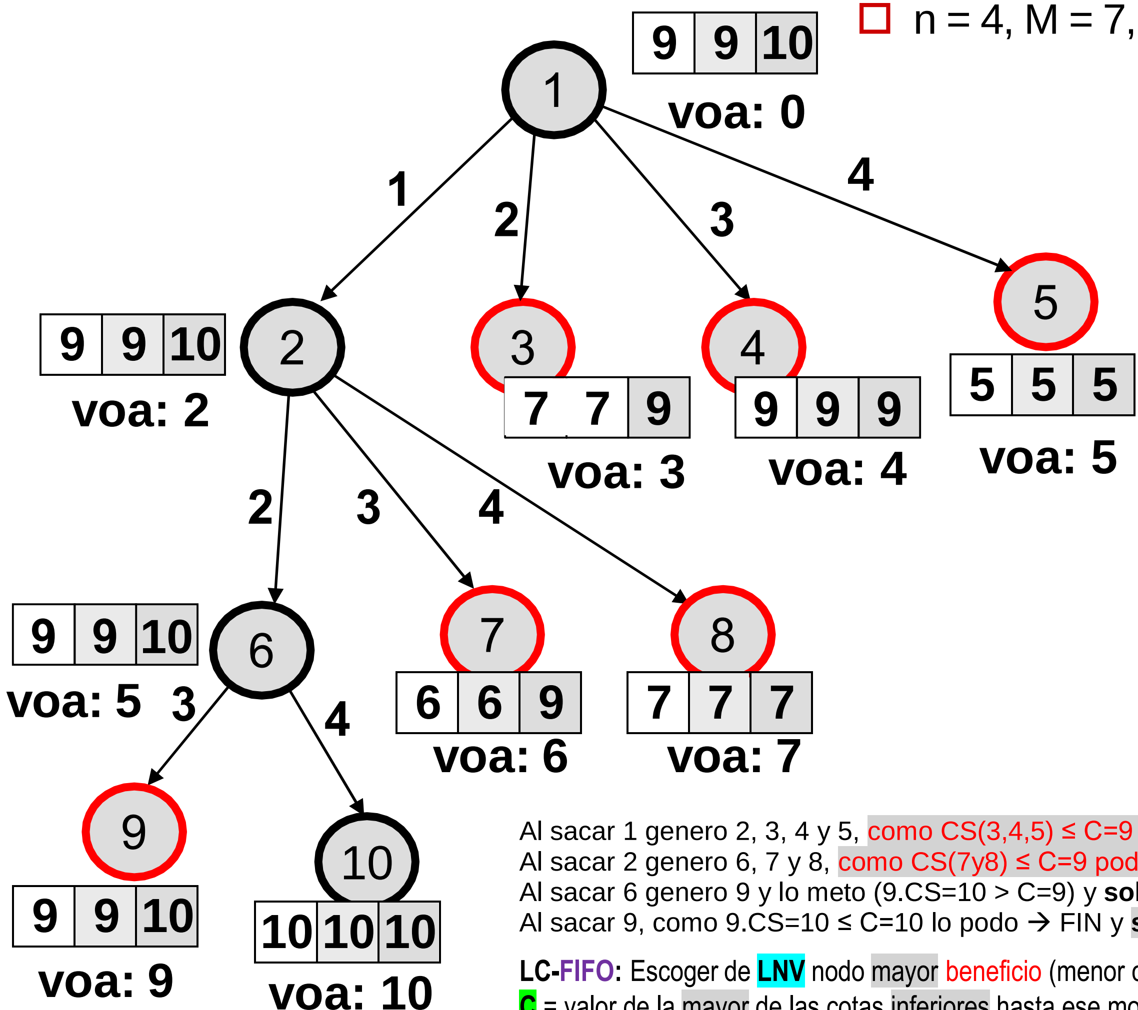
Si seguimos estrictamente el algoritmo, al sacar el nodo 6 no generamos 8 y 9 ya que el nodo 6 se poda a sí mismo, ya que CI(6)=CS(6)=10=C y un nodo x LNV se poda si x.CS≤C
(un nodo x LNV genera hijos si x.CS>C)

Solución en siguientes transparencias

C	LNV
9	1
9	3
9	5
10	6

s= (1, 1, 0, 1)
Árbol binario

- **Ejemplo.** Utilizando un **árbol combinatorio** (en vez de binario) 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)

Al sacar 1 genero 2, 3, 4 y 5, como CS(3,4,5) ≤ C=9 podo 3,4 y 5 y solo meto 2 (2.CS=10 > C=9)
Al sacar 2 genero 6, 7 y 8, como CS(7y8) ≤ C=9 podo 7 y 8 y solo meto 6 (6.CS=10 > C=9)
Al sacar 6 genero 9 y lo meto (9.CS=10 > C=9) y **solución 10 Valor**(10)=10 → actualizo C, C=10
Al sacar 9, como 9.CS=10 ≤ C=10 lo podo → FIN y **solución=10**

LC-FIFO: Escoger de **LNV** nodo mayor beneficio (menor coste), si empate escoger el primero que se metió.
C = valor de la mayor de las cotas inferiores hasta ese momento. Podar **x** si **x.CS ≤ C**,

- **NOTA:** si el nodo **x** es tal que **CI(x)** = **CS(x)**, entonces se poda a sí mismo[2], 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** ≤ **voa**)
    - **Posibilidad 3.** Generar directamente el nodo solución:

      si y.CI == y.CS **entonces**

      y:= SolucionMochilaVoraz(y.nivel+1, n, M-y.pact)

(**x.BE** se usa como valor **x.CI)**

CI(x)=BE(x) = x.**bact +** ⌊**MochilaVoraz01**(x. nivel+1, n, M-x.pact)⌋

CS(x) = x.**bact +** ⌊**MochilaVorazNo01**(x. nivel+1, n, M-x.pact)⌋

Si CI(x)=CS(x) → ⌊**MochilaVorazNo01**(x. nivel+1, n, M-x.pact)⌋ = ⌊**MochilaVoraz01**(x. nivel+1, n, M-x.pact)⌋ **→ MochilaVoraz01**

[2] **C** = mayor de las cotas inferiores hasta ese momento. Si algún nodo **x**, **x.CS ≤ C**, podar **x** → como **x.CI=x.CS** se poda a si mismo

- ¿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

    b= (3, 3, 3, 3, 3, 3, 3, 3, 3, 3, ..., 3, 3, 3, 4)
    p= (3, 3, 3, 3, 3, 3, 3, 3, 3, 3, ..., 3, 3, 3, 5)
  - **Problema:** CI, CS y BE son poco informativas.

Enunciado del problema de asignación maximizando el beneficio.

- Existen **n** personas y **n** trabajos.
- Cada persona **i** puede realizar un trabajo **j** con más o menos rendimiento: **B[i, j]**.
- **Objetivo:** asignar una tarea a cada trabajador (asignación uno-a-uno), de maneraque se **maximice** la suma de rendimientos.
- **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 $\{(p_1, t_1), (p_2, t_2), \ldots, (p_n, t_n)\}$.
- **Formulación matemática:**

Maximizar $\sum_{i=1}^{n} B[p_i, t_i]$ sujeto a la **restricción** $p_i \neq p_j$, $t_i \neq t_j$, $\forall\ i \neq j$

- **Ejemplo.** (P1, T3), (P2, T2), (P3, T1)
- B_{TOTAL}= 4+8+6= 18

Tareas

Personas

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

1. Representación de la solución

- Desde el punto de vista de las **personas:**

$s = (t_1, t_2, ..., t_n)$, siendo $t_i \in \{1, ..., n\}$, con $t_i \neq t_j$, $\forall\ i \neq j$

- $\mathbf{t_i}$ → 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. Ramificación y poda. Ejemplos de aplicación. **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
         CI, BE, CS: entero
   finregistro

1.a) Nodo raíz
      raiz.nivel:= 0
      raiz.bact:= 0

1.b) Para cada y hijo de un nodo x
      para i:= 1, ..., 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]
               .....
      Usada(m: Nodo; t: entero): booleano
         para i:= 1,..., m.nivel hacer
            si m.tupla[i]==t entonces
               devolver TRUE
         devolver FALSE

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

```
tipo
   Nodo = registro
         tupla: array [1..n] de entero
         nivel: entero
         bact: entero
         usadas: array [1..n] de booleano
         CI, BE, CS: entero
   finregistro

1.a) Nodo raíz
      raiz.nivel:= 0
      raiz.bact:= 0
      para i:= 1, ..., n hacer usadas[i]=false

1.b) Para cada y hijo de un nodo x
      para i:= 1, ..., n hacer
         y.nivel:= x.nivel+1
         y.tupla:= x.tupla
         y.usadas:= x.usadas
         si y.usadas[i] entonces break
         y.tupla[y.nivel]:= i
         y.usadas[i]:=true
         y.bact:= x.bact + B[y.nivel, i]
               .....
1.c) Función Solución(x: Nodo): booleano
         devolver x.nivel==n
```

Resultado: tarda menos tiempo pero usa más memoria

Cada nodo contiene su propia copia de tupla, (el array tupla no es compartido por los nodos), ídem con array usadas

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

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[·,·])
- **BE.** La media de las cotas: x.BE:= (x.CI+x.CS)/2

2. Posibilidad 2. Estimaciones precisas:

- **CI.** Resolver el problema usando un algoritmo voraz.
  x.CI:= x.bact + AsignaciónVoraz(x)
- **AsignaciónVoraz(x):** Asignar a cada persona la tarea libre con más beneficio (sin repetir).

AsignaciónVoraz(m: Nodo): entero

bacum:= 0

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

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

m.usadas[k]:= TRUE

bacum:= bacum + B[i, k]

finpara

devolver bacum

Tareas

Personas

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

2. Posibilidad 2. Estimaciones precisas: (…continuación)

- **CS.** Asignar a cada persona la tarea libre con mayor beneficio (aunque se repitan).
  x.CS:= x.bact + MáximosTareas(x)

  MáximosTareas(m: Nodo): entero
  bacum:= 0
  para i:= m.nivel+1, ..., n **hacer**
  k:= $\text{argmax}_{\forall j \in \{1..n\}}$ 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

Posibilidad 1. Estimaciones triviales
CI(raiz) = raíz.bact = **0**
CS(raíz) = raíz.bact + (n - raiz.nivel) * max(B[-,-]) =
= 0 + (3 - 0) * 8 = **24**
BE(raíz) = (raíz.CI+raíz.CS) / 2 = (**0** + **24**) / 2 = **12**

Posibilidad 2. Estimaciones precisas
CI(raiz) = raíz.bact + AsignaciónVoraz(raíz) =
= 0 + (6 + 3 + 1) = **10**
CS(raíz) = raíz.bact + MaximosTareas(raíz) =
= 0 + (6 + 8 + 6) = **20**
BE(raíz) = (raíz.CI+raíz.CS) / 2 = (**10** + **20**) / 2 = **15**

La solución óptima es:
s = (3, 2, 1) cuyo beneficio es 4+8+6 **= 18**

- **Cuestión clave:** ¿podemos garantizar que la solución óptima a partir de **x** estará entre **CI(x)** y **CS(x)**?

- **Ejemplo. Cálculo de las funciones CI(x), CS(x), BE(x)**
  - **Ejemplo. n**= 3. ¿Cuánto serían **CI(raíz)**, **CS(raíz)** y **BE(raíz)**?
  - ¿Cuál es la solución óptima del problema?

Tareas (columnas) / Personas (filas)

B	**1**	**2**	**3**
1	5	6	4
2	3	8	2
3	6	5	1

3. Estrategia de ramificación y de poda ***Maximizacion***

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 ≤ C**

3.b) Estrategia de ramificación

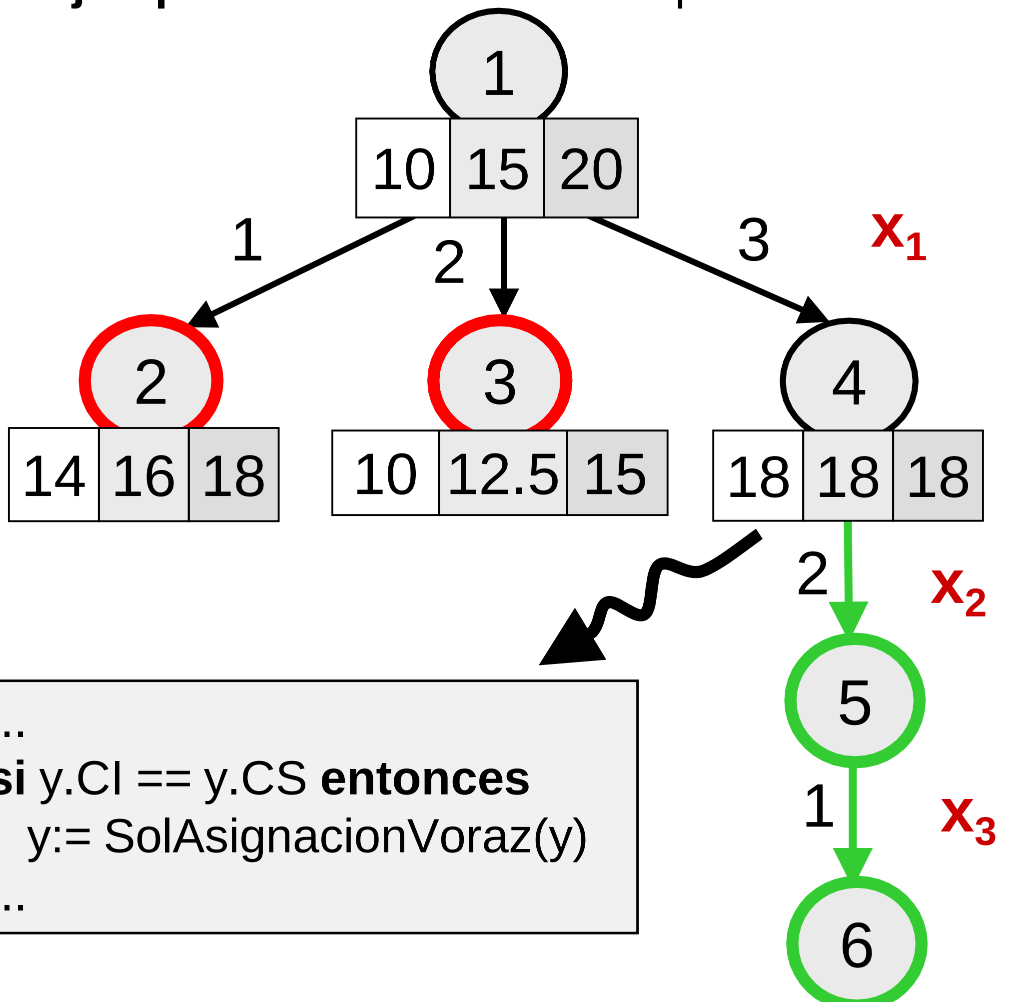
- **Usar una estrategia MB-LIFO:** explorar primero los nodos con mayor BE y en caso de empate seguir por la rama más profunda.

4. Algoritmo de ramificación y poda. (Exactamente el mismo que antes)

> **MB-LIFO:** Seleccionar de **LNV** el nodo con mayor beneficio (menor coste), en caso de empate escoger el último que se introdujo
> **C** = valor de la mayor de las cotas inferiores hasta ahora
> Si para algún nodo **i**, **CS(i)** ≤ **C**, entonces **podar i**

```
AsignaciónRyP (n: ent; B: array[1..n,1..n] de ent; var s: Nodo) // Maximizar b
    LNV:= {raiz}
    C:= raiz.CI
    s:= ∅
    mientras LNV ≠ ∅ hacer
        x:= Seleccionar(LNV)          // Estrategia de ramificación MB-LIFO
        LNV:= LNV - {x}
        si x.CS > C entonces          // Estrategia de poda ( x.CS≤C 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
```

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



...
si y.CI == y.CS **entonces**
   y:= SolAsignacionVoraz(y)
...

s= (3, 2, 1)

Tareas

Personas

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

C	LNV	
10	1	
18	2	3
18	3	

6. Ramificación y poda. Ejemplos de aplicación. **2_Problema de asignación.**

MB-LIFO: Seleccionar de **LNV** el nodo con mayor beneficio, en caso de empate escoger el último que se introdujo (rama más profunda)

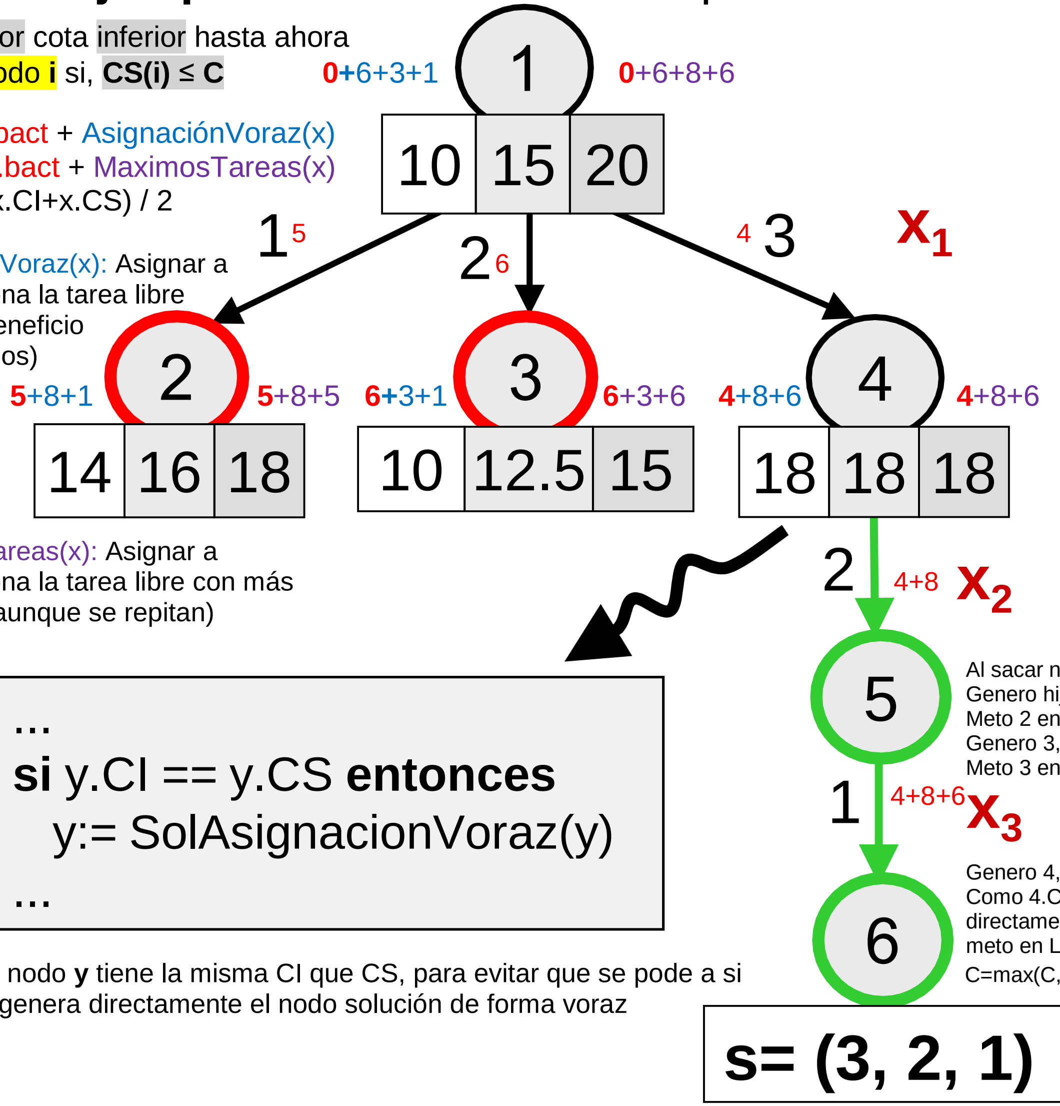
☐ **Ejemplo. n**= 3. Estimaciones precisas.

C = mayor cota inferior hasta ahora
Podar nodo **i** si, **CS(i) ≤ C**

CI(x) = x.bact + AsignaciónVoraz(x)
CS(x) = x.bact + MaximosTareas(x)
BE(x) = (x.CI+x.CS) / 2

AsignaciónVoraz(x): Asignar a cada persona la tarea libre con más beneficio (sin repetidos)

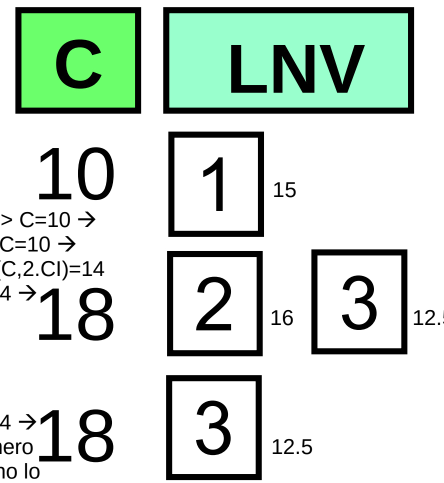
MaximosTareas(x): Asignar a cada persona la tarea libre con más beneficio (aunque se repitan)



Tareas

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

Personas



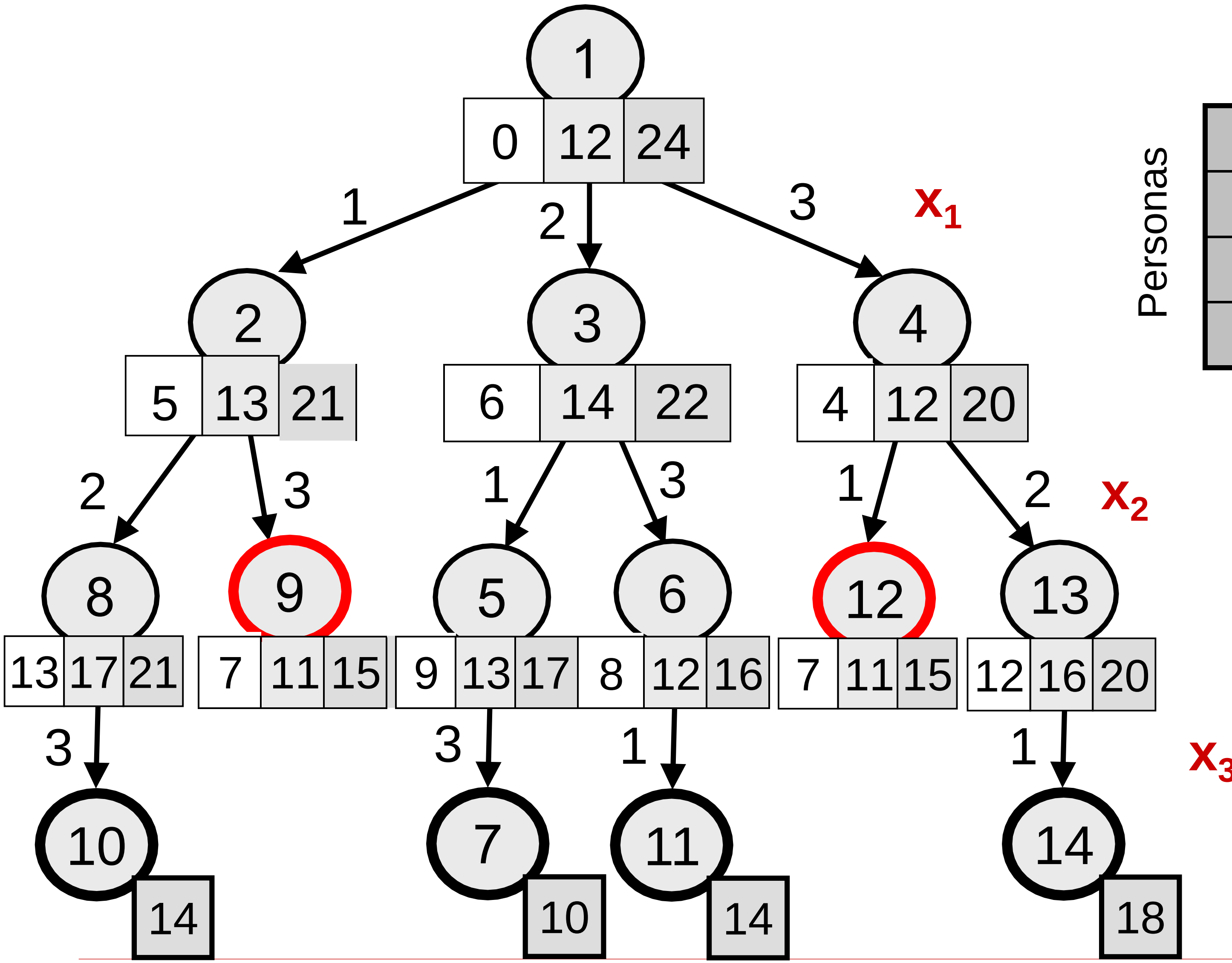
Al sacar nodo 1, 1.CS=20 > C=10 →
Genero hijo 2, 2.CS=18 > C=10 →
Meto 2 en LNV → C=max(C,2.CI)=14
Genero 3, 3.CS=15 > C=14 →
Meto 3 en LNV y C=14

Genero 4, 4.CS=18 > C=14 →
Como 4.CI==$.CS=18 genero directamente la solución (no lo meto en LNV) s=(3,2,1) y s.bact=18
C=max(C,s.bact)=max(14,18)→C=18 → Al sacar 2, 2.CS=18 ≤ C=18 → podo
→ Al sacar 3, 3.CS=15 ≤ C=18 → podo

```
...
si y.CI == y.CS entonces
   y:= SolAsignacionVoraz(y)
...
```

Cuando un nodo **y** tiene la misma CI que CS, para evitar que se pode a si mismo, se genera directamente el nodo solución de forma voraz

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



Tareas

Personas

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

s= (3, 2, 1)

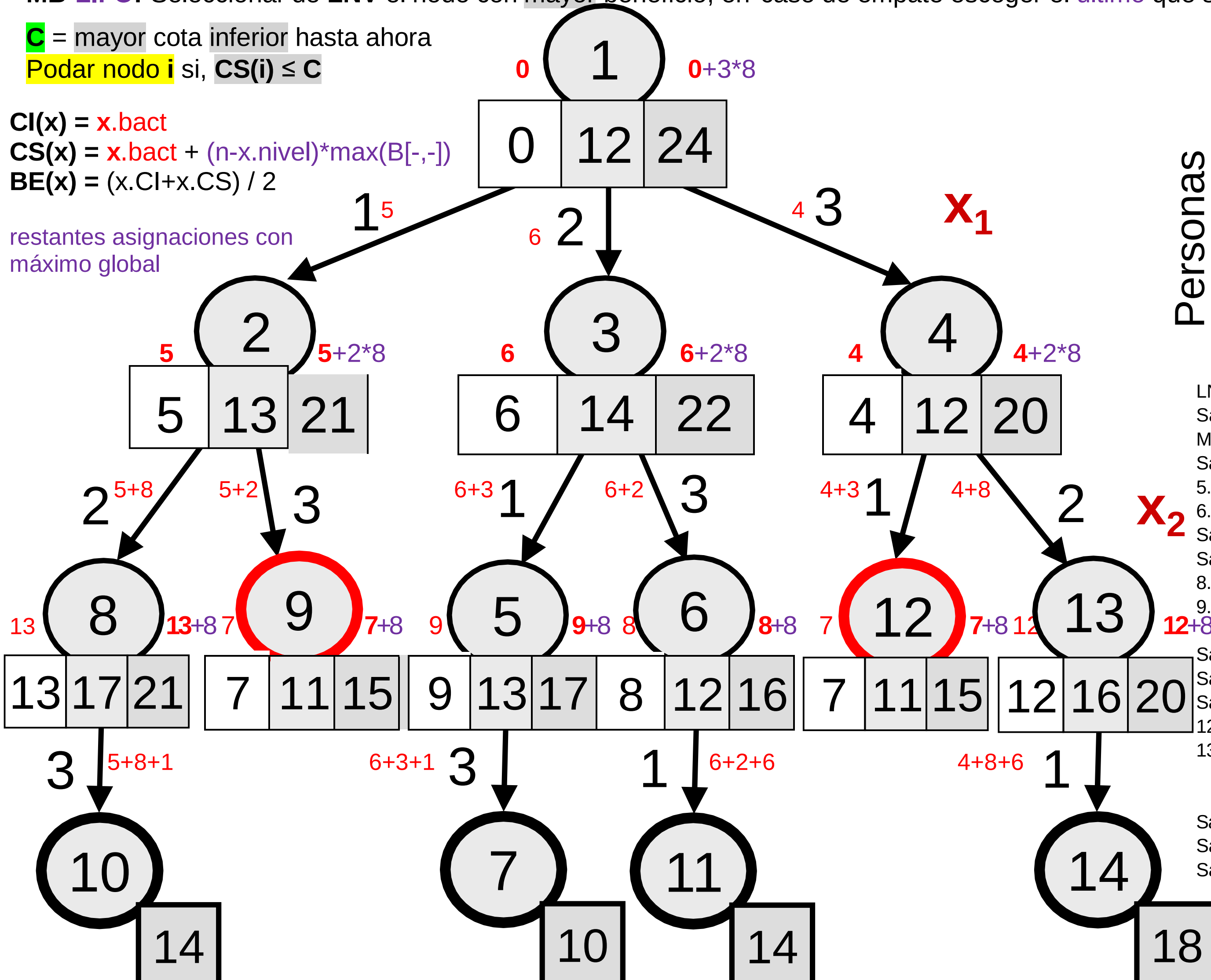
❑ **Ejemplo. n**= 3. Usando las estimaciones triviales.

MB-LIFO: Seleccionar de **LNV** el nodo con mayor beneficio, en caso de empate escoger el último que se introdujo (rama más profunda)

C = mayor cota inferior hasta ahora
Podar nodo i si, **CS(i) ≤ C**

CI(x) = **x**.bact
CS(x) = **x**.bact + (n-x.nivel)*max(B[-,-])
BE(x) = (x.CI+x.CS) / 2

restantes asignaciones con máximo global



Tareas

Personas

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

LNV = raíz = nodo 1, C=0
Saco 1 de LNV, 1.CS=24 > C=0 → genero hijos 2,3,4
Meto 2, C=5, meto 3, C=6, meto 4 → orden LNV 3,2,4
Saco 3 de LNV, 3.CS=22 > C=6 → genero hijos 5, 6
5.CS=17 > C=6 → Meto 5, C=9
6.CS=16 > C=9 → Meto 6 → orden LNV 5, 2, 6, 4
Saco 5 de LNV, 5.CS=17 > C=9 → genero sol 7, C=10
Saco 2 de LNV, 2.CS=21 > C=10, genero hijos 8, 9
8.CS=21 > C=10 → Meto 8, C=13
9.CS=15 > C=13 → Meto 9 → orden LNV 8, 6, 4, 9

Saco 8 LNV, 8.CS=21 > C=13 → genero sol 10, C=14
Saco 6 de LNV, 6.CS=16 > C=14 → genero sol 11
Saco 4 de LNV, 4.CS=20 > C=14 → genero hijo 12,13
12.CS=15 > C=14 → Meto 12
13.CS=20 > C=14 → Meto 13 → orden LNV 13, 12, 9

Saco 13 LNV, 13.CS=20 > C=14 → genero sol 14, Ç=18
Saco 12 de LNV, 12.CS=15 ≤ C=18 → **podo 12**
Saco 9 de LNV, 9.CS=15 ≤ C=18 → **podo 9** → **FIN**

s= (3, 2, 1)

- 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(n-nivel))**

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.