# 4 Ramificación y acotación

# 4 Introducción: (1) Ramificación…

- Al igual que los métodos de búsqueda con retroceso:
  - se aplica a problemas de optimización con restricciones,
  - se genera el espacio de soluciones, organizándolo en un árbol (en general en un grafo),
  - no se genera el espacio de soluciones completo, sino que se podan bastantes estados.

- Terminología:
  - Nodo **vivo**: nodo del espacio de soluciones del que no se han generado aún todos sus hijos.
  - Nodo **muerto**: nodo del que no se van a generar más hijos porque:
    - no hay más
    - no es completable, i.e., viola las restricciones
    - no producirá una solución mejor que la solución en curso
  - Nodo **en curso** (o en expansión): nodo del que se están generando hijos

# 4 *Introducción: (1) Ramificación…*

- Diferencia fundamental con el método de búsqueda con retroceso:
  - Búsqueda con retroceso:

    Tan pronto como se genera un nuevo hijo del nodo en curso, este hijo pasa a ser el nodo en curso.
  - Ramificación y acotación:

    Se generan todos los hijos del nodo en curso antes de que **cualquier otro nodo vivo** pase a ser el nuevo nodo en curso.

- En consecuencia:
  - Búsqueda con retroceso:

    Los únicos nodos vivos son los que están en el camino de la raíz al nodo en curso.
  - Ramificación y acotación:

    Puede haber más nodos vivos.
    Se deben almacenar en una estructura de datos auxiliar: **lista de nodos vivos**.

# 4 *Introducción: (1) Ramificación…*

- Diferentes estrategias de elegir el siguiente nodo de la lista de nodos vivos

  ⇒ Distintos órdenes de recorrido del árbol de soluciones

  – **FIFO**: la lista de nodos vivos es una cola

    ⇒ recorrido por niveles (en anchura)

  – **LIFO**: la lista de nodos vivos es una pila

    ⇒ ≈ recorr. en profundidad (D–búsqueda)

  – **Mínimo coste**: la lista es una cola con prioridades

    ⇒ recorrido "extraño"

    La prioridad de un nodo se calcula de acuerdo con una **función de estimación** que mide cuánto de "prometedor" es un nodo.

# 4 *Introducción: (1) Ramificación…*

- Primer punto **clave** de los métodos de ramificación y acotación:

  Encontrar un buen orden de recorrido (o ramificación) de los nodos,

  es decir,

  definir una buena función de prioridad de los nodos vivos,

  para que las soluciones buenas se encuentren rápidamente.

# 4 Introducción: (1) Ramificación…

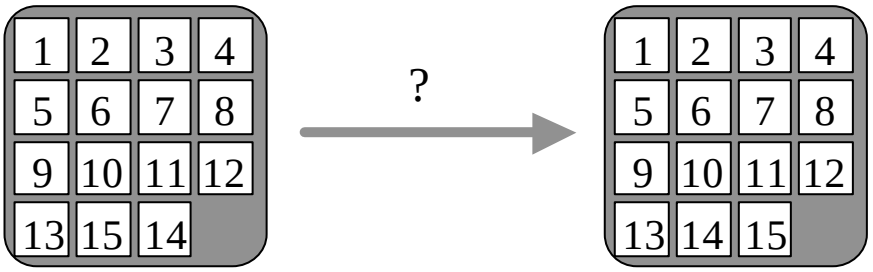
- El esquema es el siguiente:

```
repetir
  expandir el nodo vivo más prometedor;
  generar todos sus hijos;
  una vez generados, el padre se mata;
  para cada hijo hacer
    si tiene un coste esperado peor que
       el de la mejor solución en curso
       entonces se mata sino
    si tiene un coste esperado mejor
       que el de la mejor solución en
       curso y no es solución
       entonces se pasa a la lista de
                 nodos vivos
    sino {tiene un coste esperado mejor que
          el de la mejor solución en curso
          y es solución (el coste no es
          estimado sino real)}
       pasa a ser la mejor solución en
       curso y se revisa toda la lista
       de nodos vivos, eliminando los
       que prometen algo peor de lo
       conseguido fsi fsi
  fpara
hasta que la lista está vacía
```

# 4 Un primer ejemplo: El juego del 15

Samuel Loyd: El juego del 15 o "taken".
Problema publicado en un periódico de Nueva York en 1878 y por cuya solución se ofrecieron 1000 dólares.

- El problema original:



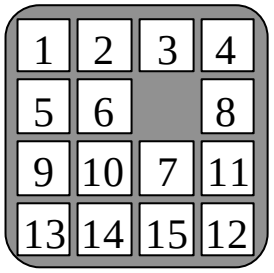

Problema de Lloyd  El objetivo

- La solución de "fuerza bruta":
  - Buscar en el espacio de estados (es decir en todos los estados posibles alcanzados desde el problema propuesto) hasta encontrar el objetivo.
  - Nótese que hay $16! \sim 20'9 \times 10^{12}$ posiciones, aunque sólo (?) la mitad pueden alcanzarse desde la posición inicial propuesta por Lloyd…

# 4 Un primer ejemplo: El juego del 15

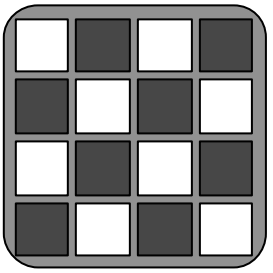
- ¿Qué estados son alcanzables?

  - Numeremos las casillas de 1 a 16.
  - Dada una configuración o estado, sea *Pos*(*i*) la posición (entre 1 y 16) de la ficha con el nº *i*, y sea *Pos*(16) la posición de la casilla vacía.
  - Dado un estado, para cada ficha *i*, sea *m*(*i*) el número de fichas *j* tales que $j<i$ y $Pos(j)>Pos(i)$.




*m*(*i*)=0, *i*=1,2,3,4,5,6,7
*m*(8)=*m*(9)=*m*(10)=1
*m*(11)=0
*m*(12)=0
*m*(13)=*m*(14)=*m*(15)=1
*m*(16)=9

  - Dado un estado, sea *x*=1 si la casilla vacía está en alguna de las posiciones sombreadas y *x*=0 en caso contrario.



# 4 Un primer ejemplo: El juego del 15

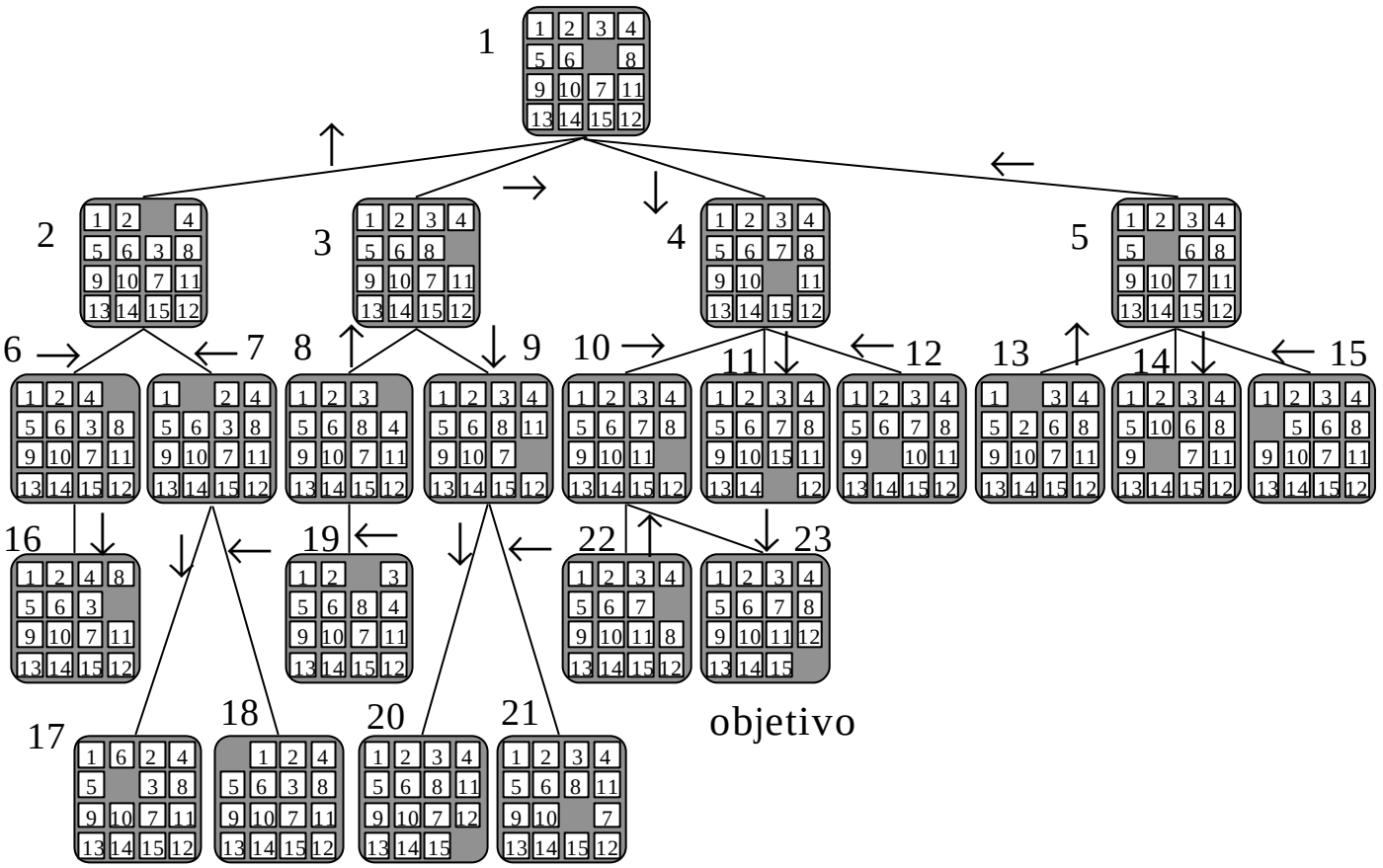
- **Teorema.** El estado objetivo es alcanzable desde un cierto estado si para ese estado:

$$\sum_{i=1}^{16} m(i)+x \qquad \text{es par}$$

  - Para el ejemplo anterior, dicha función vale 26, luego el estado objetivo es alcanzable.

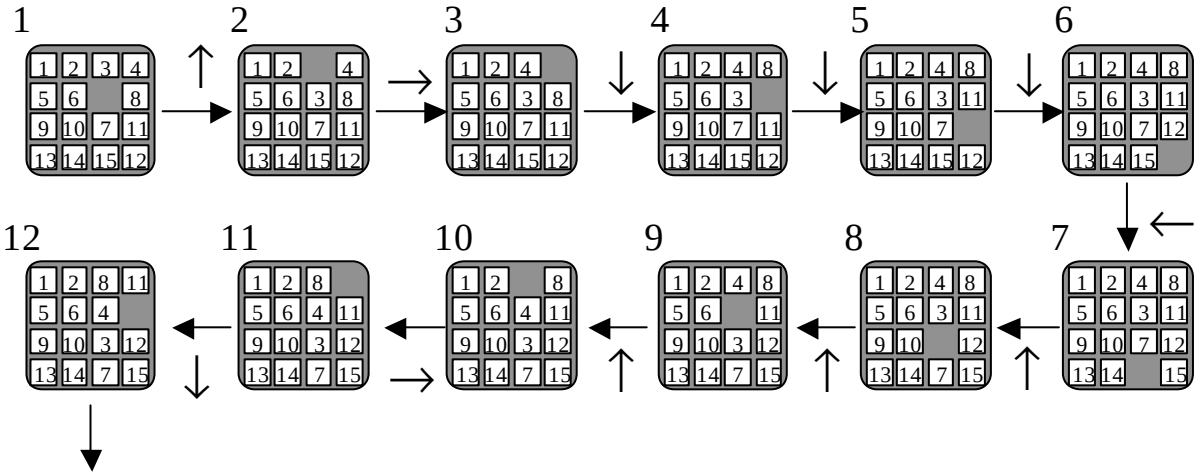
# 4 Un primer ejemplo: El juego del 15

- Hijos de cada nodo = movimientos de la casilla vacía (arriba, abajo, izquierda, derecha)
- poda inicial: ningún nodo tiene un hijo igual a su padre



# 4 Un primer ejemplo: El juego del 15

- Estrategias **ciegas** de elección del siguiente nodo en la lista de nodos vivos:
  - FIFO = recorrido por niveles o en anchura
    - tal como aparecen numerados en la figura
    - relativamente útil si el nodo solución está cerca de la raíz
  - Recorrido en profundidad
    - orden de los movimientos: ↑ → ↓ ←




...

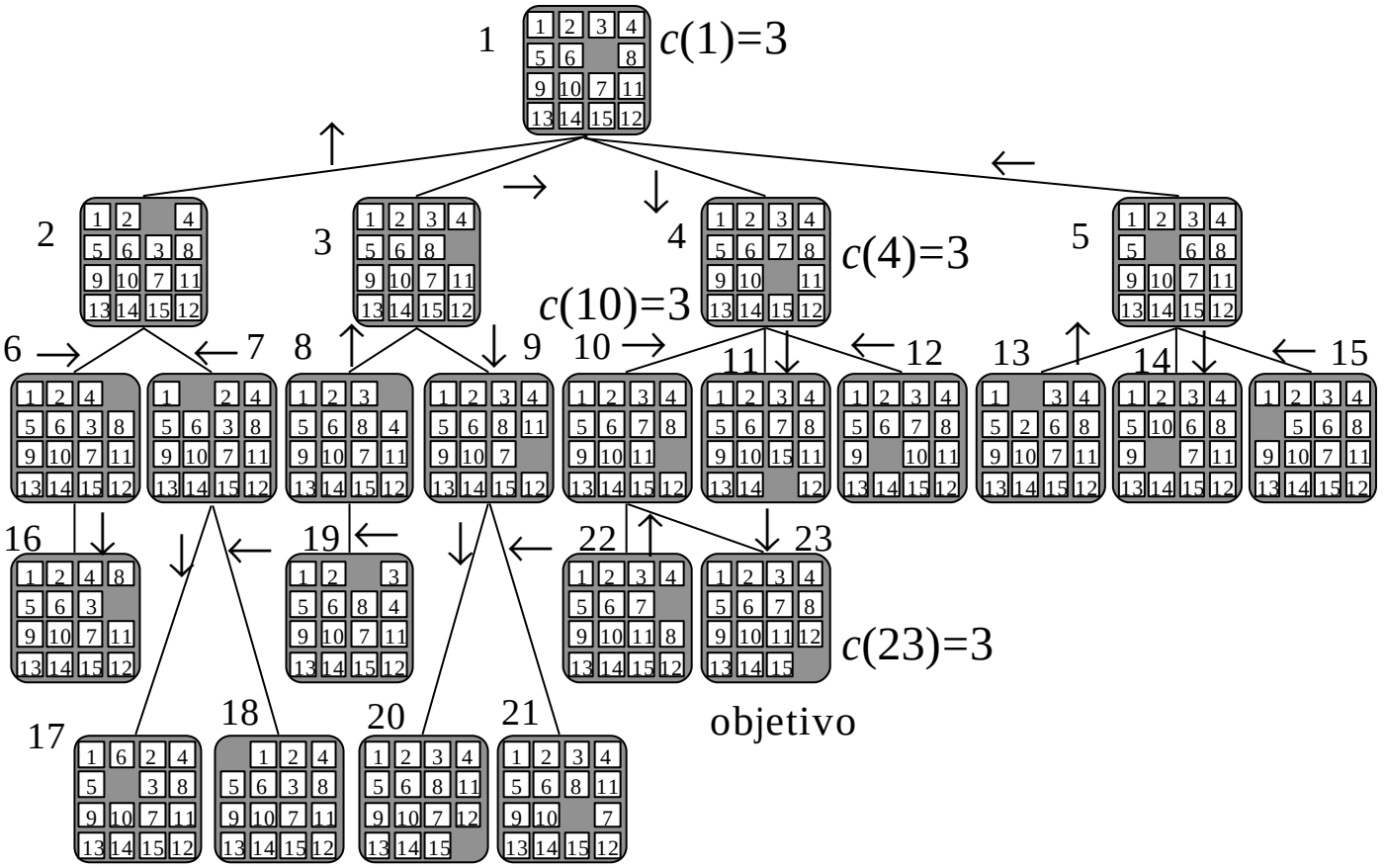
    - sigue la rama izquierda del árbol
    - no encuentra nunca la solución (a menos que exista una solución en esa rama)
  - LIFO = D-búsqueda
    - problemas similares al recorrido en profundidad

# 4 Un primer ejemplo: El juego del 15

- Estrategias no ciegas
  - *¿inteligentes*?
  - también se llaman técnicas de investigación operativa
  - Supongamos que fuese posible calcular la siguiente función para cada nodo *x* del árbol:

    $c(x)$ = longitud desde la raíz hasta el estado objetivo más cercano que es descendiente de *x*

# 4 Un primer ejemplo: El juego del 15




– Método:

- la raíz es el nodo en curso
- generar sus hijos hasta encontrar uno con igual valor de *c* que él (en el ejemplo, 4)
- repetir lo mismo (en el ejemplo, 10)
- hasta encontrar el objetivo (en el ejemplo, 23)

→ Los únicos nodos en curso son los del camino desde la raíz hasta la solución más cercana.

# 4 Un primer ejemplo: El juego del 15

– Problema: el cálculo de la función $c$.

– Un ejemplo de problemas en los que es posible: aquéllos en los que hay soluciones voraces.

– En general, usar una **función de estimación**:

$$\hat{c}(x) = f(x) + \hat{g}(x),$$

con $f(x)$ = longitud del camino de la raíz a $x$

y $\hat{g}(x)$ = estimación de la longitud del camino desde $x$ hasta la solución mas cercana

– $\hat{c}(x)$ es una función heurística; debe ser:

- fácil de calcular
- si $x$ es una hoja o una solución: $c(x) = \hat{c}(x)$

– Por ejemplo, en el juego del 15:

$\hat{g}(x)$ = nº de casillas no vacías que no están en su sitio objetivo

$(\hat{c}(x) \leq c(x), \forall x)$

# 4 Un primer ejemplo: El juego del 15

- Estrategia de mínimo coste con $\hat{c}(x)$
  - La lista de nodos vivos es una cola de nodos $x$ con prioridades $\hat{c}(x)$
  - En el ejemplo:

1 

$\hat{c}(1) = 0 + 3$

  - Se generan sus hijos y se añaden en la cola:

2 

$\hat{c}(2) = 1 + 4$

3 

$\hat{c}(3) = 1 + 4$

4 

$\hat{c}(4) = 1 + 2$

5 

$\hat{c}(5) = 1 + 4$

  - Se elige el mínimo, el 4, se elimina de la cola y se generan sus hijos:

2 

$\hat{c}(2) = 1 + 4$

3 

$\hat{c}(3) = 1 + 4$

5 

$\hat{c}(5) = 1 + 4$

10 

$\hat{c}(10) = 2 + 1$

11 

$\hat{c}(11) = 2 + 3$

12 

$\hat{c}(12) = 2 + 3$

# 4 *Un primer ejemplo: El juego del 15*

– Se elige el mínimo, el 10, se elimina de la cola y se añaden sus hijos:

2 


$\hat{c}(2) = 1 + 4$

3 


$\hat{c}(3) = 1 + 4$

5 


$\hat{c}(5) = 1 + 4$

11 


$\hat{c}(11) = 2 + 3$

12 


$\hat{c}(12) = 2 + 3$

22 


$\hat{c}(22) = 3 + 2$

23 


$\hat{c}(23) = 3 + 0$

– El nodo 23 es el objetivo y termina la búsqueda.

– En este caso, el método es casi (hay que almacenar la cola con prioridades) tan eficiente como si se hubiera utilizado $c(x)$.

# 4 Un primer ejemplo: El juego del 15

```
algoritmo mínimoCoste(ent x0:nodo)
variables c:cola; {con prior. de <x,coste(x)>}
          éxito:booleano; xcurso,x:nodo
principio
  si esSol(x0) entonces escribir(x0)
  sino
    creaVacía(c); {cola de nodos vivos}
    añadir(c,<x0,coste(x0)>);
    éxito:=falso;
    mq not éxito and not esVacía(c) hacer
      xcurso:=min(c); {nodo en curso}
      eliminarMin(c);
      mq not éxito and
         hay otro hijo x de xcurso hacer
        si esSol(x)
        entonces
          escribir(x); éxito:=verdad
        sino
          añadir(c,<x,coste(x)>)
        fsi
      fmq
    fmq;
    si not éxito
      entonces escribir("No hay solución")
    fsi
  fsi
fin
```

# 4 Un primer ejemplo: El juego del 15

– Comentarios al esquema algorítmico anterior:

- Si se desea escribir todo el camino desde la raíz hasta la solución, cada vez que se añade un nodo a la cola, hay que guardar en una **tabla auxiliar de padres de nodos** ese nodo junto a su padre.

- La terminación del algoritmo sólo está garantizada si el espacio de estados es finito.

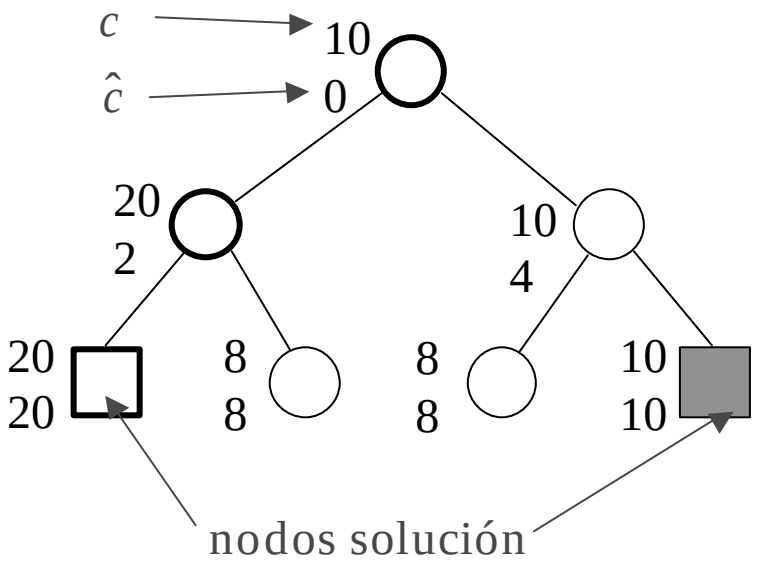
- Si el espacio de estados es infinito y existe al menos una solución, se puede garantizar la terminación con una elección adecuada de la función de estimación.

- Si el espacio de estados es infinito y no hay solución, el algoritmo no termina…

  – Se suele restringir la búsqueda a nodos con coste estimado menor que *C*.

# 4 Aplicación a problemas de optimización

- Problemas de optimización:
  - Ahora la función o coste a minimizar ya no es la distancia de la raíz a la solución, sino una cierta función objetivo dada
  - ¿Qué ocurre si interesa buscar una solución de coste mínimo?
    ¿Encuentra el algoritmo esa solución? **No**.




Se expande la raíz, sus hijos se añaden a la cola de nodos vivos, se elige el hijo izquierdo, se expande y se obtiene una solución de coste 20 (cuando la solución óptima tiene coste 10).

¿Por qué? Porque $\exists x, y$: $\hat{c}(x) < \hat{c}(y) \ \wedge \ c(x) > c(y)$

# 4 Aplicación a problemas de optimización

– **Teorema.** Si para cada nodo $x$ del árbol del espacio de estados $\hat{c}(x)$ es una estimación de $c(x)$ tal que para todo par de nodos $y$, $z$:

$$\hat{c}(y) < \hat{c}(z) \;\Leftrightarrow\; c(y) < c(z)$$

entonces el algoritmo `mínimoCoste` termina y encuentra siempre una solución de mínimo coste.

– **Problema:** no es fácil encontrar una función de estimación con tales características y fácil de calcular.

– **Conformarse con…**

Una función $\hat{c}(x)$ fácil de calcular y tal que para todo nodo $x$:

$$\hat{c}(x) \leq c(x)$$

y para cada solución:

$$\hat{c}(x) = c(x)$$

En este caso, el algoritmo `mínimoCoste` no siempre encuentra una solución de mínimo coste, pero puede modificarse ligeramente…

# 4 Aplicación a problemas de optimización

```
algoritmo mínimoCoste2(ent x0:nodo)
variables c:cola; {con prior. de <x,coste(x)>}
          éxito:booleano; xcurso,x:nodo
principio
  creaVacía(c); {cola de nodos vivos}
  añadir(c,<x0,coste(x0)>);
  éxito:=falso;
  mq not éxito and not esVacía(c) hacer
    xcurso:=min(c); {nodo en curso}
    eliminarMin(c);
    si esSol(xcurso)
      entonces
        escribir(xcurso);
        éxito:=verdad
      sino
        para todo x hijo de xcurso hacer
          añadir(c,<x,coste(x)>)
        fpara
    fsi
  fmq;
  si not éxito
    entonces escribir("No hay solución")
  fsi
fin
```

# 4 Aplicación a problemas de optimización

- Diferencia con el algoritmo anterior:

  - el algoritmo primero: si encuentra un hijo que sí es solución factible ya no incluye el resto de los hijos en la cola de nodos vivos con prioridades;

  - el segundo algoritmo: incluye en la cola de nodos vivos con prioridades a todos sus hijos, sin mirar si son solución factible o no.

    - el algoritmo segundo es más “prudente”

# 4 Aplicación a problemas de optimización

– **Teorema.** Si para cada nodo $x$ del árbol del espacio de estados $\hat{c}(x)$ es una estimación de $c(x)$ tal que:

$$\hat{c}(x) \leq c(x)$$

y para cada nodo solución $x$ se verifica:

$$\hat{c}(x) = c(x)$$

entonces si el algoritmo `mínimoCoste2` encuentra una solución, ésta es de mínimo coste.

**Demostración:** Si se encuentra la solución `xcurso`, se tiene que

$$\hat{c}(\texttt{xcurso}) \leq \hat{c}(x)$$

para todo nodo $x$ de la cola de nodos vivos.

Además,

$$\hat{c}(\texttt{xcurso}) = c(\texttt{xcurso}) \text{ y } \hat{c}(x) \leq c(x)$$

para todo nodo $x$ de la cola de nodos vivos.

Luego $c(\texttt{xcurso}) \leq c(x)$, y `xcurso` es de mínimo coste.

# 4 Introducción: (2) … y acotación

- Ramificación… y acotación:
  - Problema de minimización de una función $c(x)$ (**función objetivo**).
  - Se utiliza una función de estimación $\hat{c}(x)$ que sea una cota inferior de todas las soluciones obtenidas desde $x$:

$$\hat{c}(x) \leq c(x)$$

  - Suponer que se conoce una cota superior, $U$, del valor mínimo de $c$.

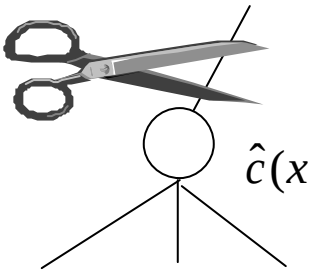
$$\rightarrow \quad c(x^*) = \min_x c(x) \leq U$$

(i.e., cota pesimista de la solución del problema)

# 4 Introducción: (2) … y acotación

– Regla de acotación:

"acotación" = poda



$\hat{c}(x) > U$

$( U \geq c(x^*) )$

$x$: $\hat{c}(x) > U$, $x$ puede ser podado porque

$\forall y$ solución descendiente de $x$: $c(y) \geq c(x) \geq \hat{c}(x) > U$

– Si se actualiza el valor de $U$ (al alcanzar una solución con un coste menor), todos los nodos vivos de coste estimado mayor también se pueden podar (otra cosa es que no se haga por no disminuir la eficiencia)

– Valor inicial de $U$:

- utilizar alguna heurística (basada en información extra sobre el problema), o
- $\infty$

– Si el valor inicial de $U$ es mayor o igual que el coste de una solución de mínimo coste, la regla de poda no elimina ningún nodo ascendente de una solución de coste mínimo.

# 4 Introducción: (2) … y acotación

- Construcción del algoritmo:
  - Definición de la función de coste $c(x)$ de forma que $c(x)$ sea mínimo para todos los nodos que representen una solución óptima.
  - ¿Cómo?
    - si $x$ es solución factible del problema de optimización: $c(x)$ = función objetivo($x$);
    - si $x$ no es factible: $c(x) = \infty$;
    - si $x$ representa una solución parcial (i.e. $x$ puede ser completada a factible):

$$c(x) = \min\{c(y) \mid y \text{ es descendiente de } x\}$$

¡Esta función es tan difícil de calcular como resolver el problema original!

  - Se utiliza $\hat{c}(x)$ tal que $\hat{c}(x) \leq c(x)$ para todo $x$.

ß

  - La función de estimación **estima el valor de la función objetivo y no el coste computacional** de alcanzar una solución.

# 4 *Introducción: (2) … y acotación*

- Puntos clave del método:

1. (**Recordar**) Encontrar un buen orden de recorrido o ramificación de los nodos, es decir, definir una buena función de prioridad de los nodos vivos para que las soluciones buenas se encuentren rápidamente.

2. **Encontrar una buena función de acotación o poda, *U*, para que se produzca el retroceso lo antes posible.**

# 4 Un problema de planificación de tareas a plazo fijo

- Generalización del problema resuelto con un algoritmo voraz (*Algoritmos voraces*, pág. 44).
- Se tiene un conjunto $C=\{1,2,\ldots,n\}$ de tareas y en cada instante sólo se puede realizar una tarea.
- Cada tarea $i$, $1 \leq i \leq n$, tiene asociada una terna $(t_i,d_i,w_i)$ de forma que:
  - $t_i$ es la duración de la tarea $i$ (el caso $t_i=1$ para todo $i$ fue resuelto con un algoritmo voraz);
  - la tarea $i$ debe terminarse antes del instante (o **plazo**) $d_i$;
  - hay una **penalización** $w_i$ si la tarea $i$ no termina antes de su plazo.
- Se trata de encontrar un subconjunto $S$ de tareas de $C$ tal que todas las tareas de $S$ puedan ser ejecutadas antes de sus plazos y la penalización total sea mínima.

# 4 Un problema de planificación de tareas a plazo fijo

- Ejemplo:

  $n = 4$;

  $(t_1,d_1,w_1) = (1,1,5)$

  $(t_2,d_2,w_2) = (2,3,10)$

  $(t_3,d_3,w_3) = (1,2,6)$

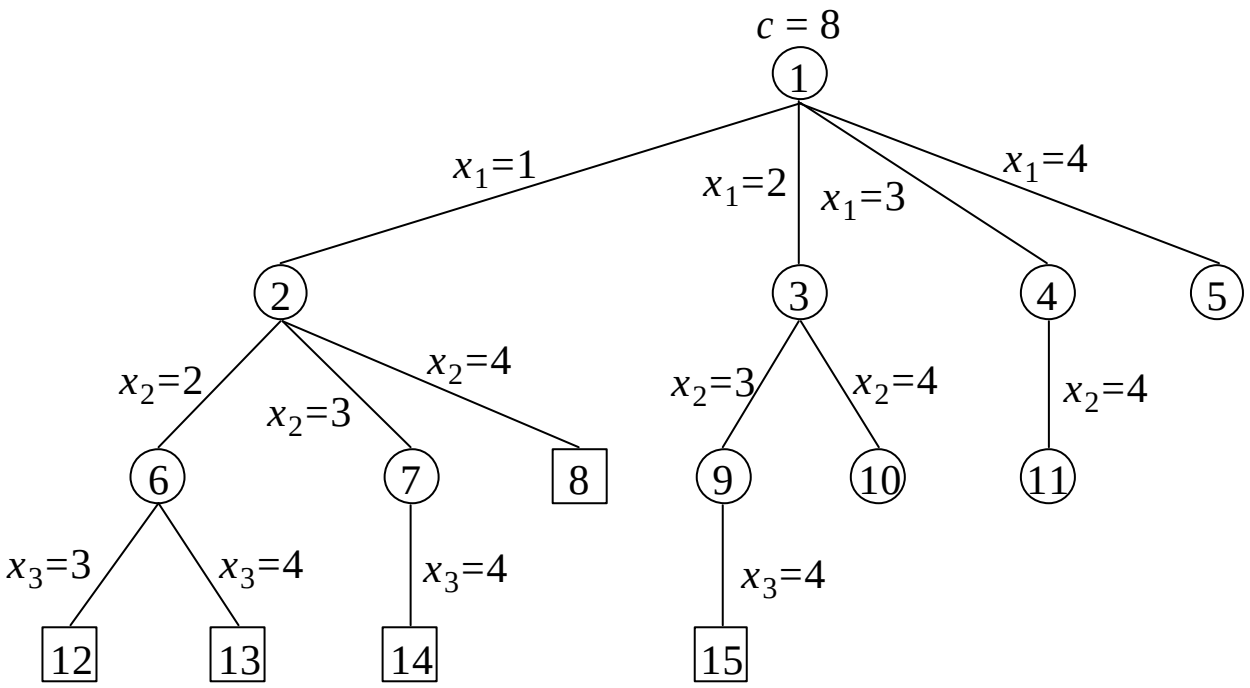
  $(t_4,d_4,w_4) = (1,1,3)$

- Espacio de estados:
  todos los subconjuntos de {1,2,3,4}.

- Dos representaciones posibles:
  - tuplas de tamaño variable y
  - tuplas de tamaño fijo.

# 4 Un problema de planificación de tareas a plazo fijo

- Representación con tuplas de tamaño variable:




- Nodos cuadrados: estados no factibles
- Nodos circulares: nodos factibles (o soluciones)

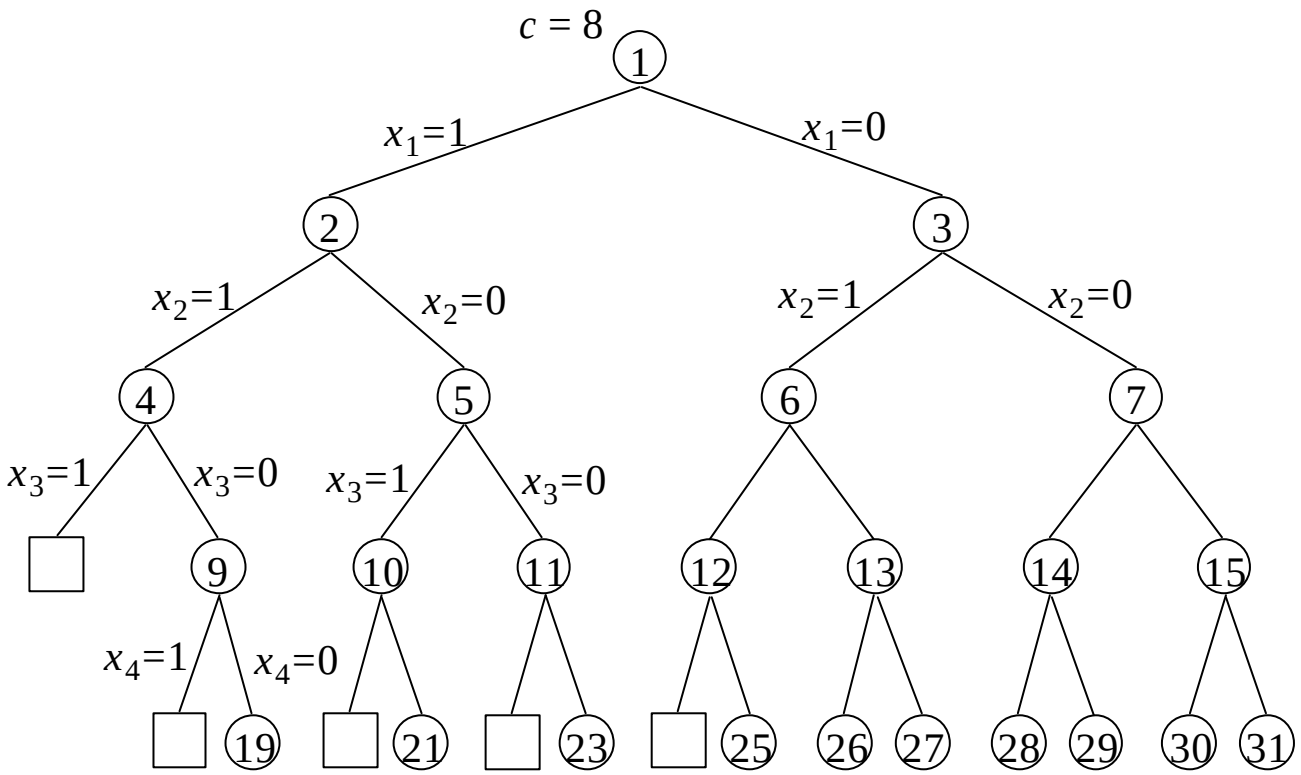
- Nodo 9: solución óptima (es la única)

  $S$ = {2,3}

  Penalización total = 8

# 4 Un problema de planificación de tareas a plazo fijo

- Representación con tuplas de tamaño fijo:



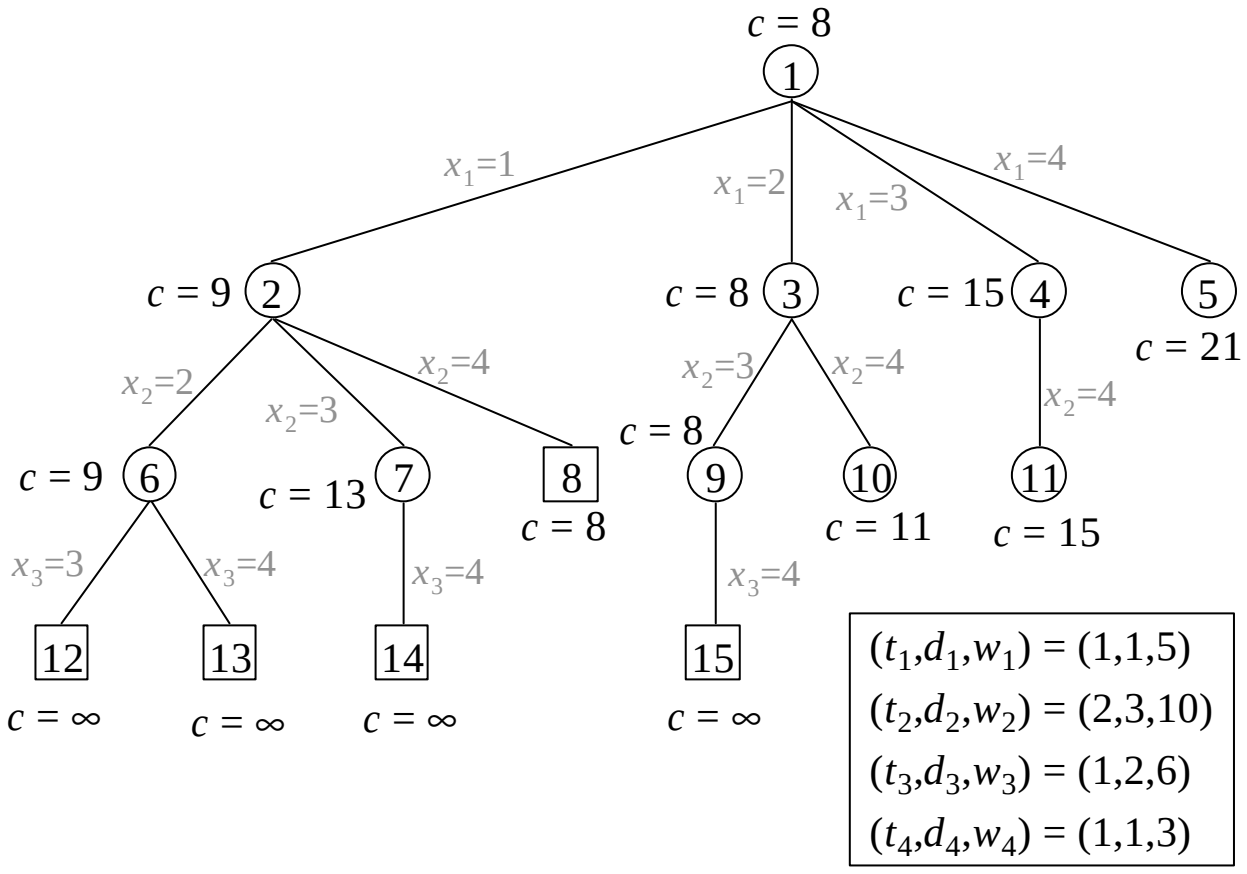

- Soluciones factibles: representadas por hojas circulares
- Solución óptima: nodo 25

  $S$ = {2,3}

  Penalización total = 8

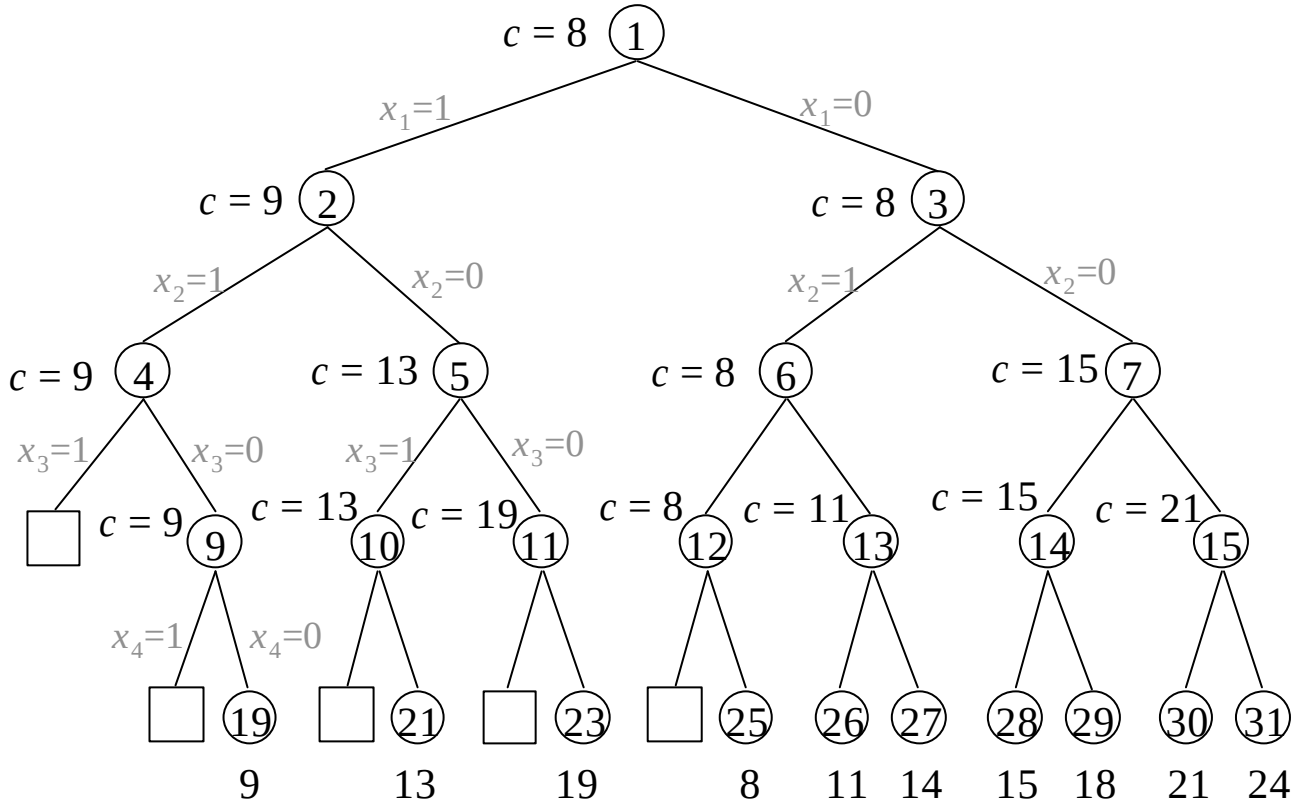
# 4 Un problema de planificación de tareas a plazo fijo

- Definición de la función de coste $c(x)$. Caso de representación de tamaño variable:
  - Para todo nodo circular $x$:
    $c(x)$ = mínima penalización correspondiente a todos los nodos descendientes de $x$.
  - Para todo nodo cuadrado $x$: $c(x) = \infty$.



# 4 Un problema de planificación de tareas a plazo fijo

- Definición de la función de coste $c(x)$. Caso de representación de tamaño fijo:
  - Para todo nodo circular $x$: $c(x)$ = mínima penalización correspondiente a todos los nodos descendientes de $x$.
  - Para todo nodo cuadrado $x$: $c(x) = \infty$.




$(t_1,d_1,w_1) = (1,1,5)$
$(t_2,d_2,w_2) = (2,3,10)$
$(t_3,d_3,w_3) = (1,2,6)$
$(t_4,d_4,w_4) = (1,1,3)$

# 4 Un problema de planificación de tareas a plazo fijo

- Definición de una función de estimación:

$$\hat{c}(x) \leq c(x)$$

  - Sean
    - $S_x$ = {tareas seleccionadas para $S$ hasta el nodo $x$}
    - $m_x = \max \{i \mid i \in S_x\}$
  - Entonces:

$$\hat{c}(x) = \sum_{\substack{i < m_x \\ i \notin S_x}} w_i$$

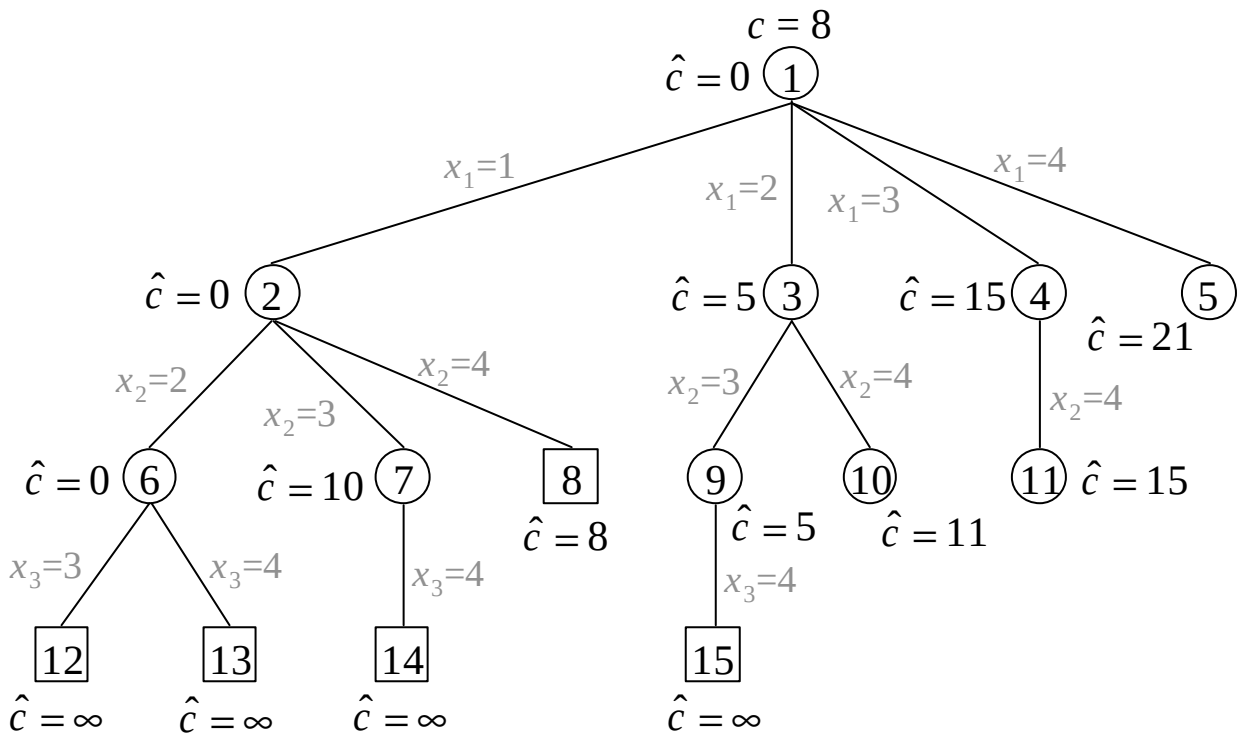
penalización debida a las tareas que ya han quedado excluidas de $x$

  - Para nodos cuadrados:

$$\hat{c}(x) = \infty$$

# 4 Un problema de planificación de tareas a plazo fijo

– En el caso de formulación de tamaño variable:

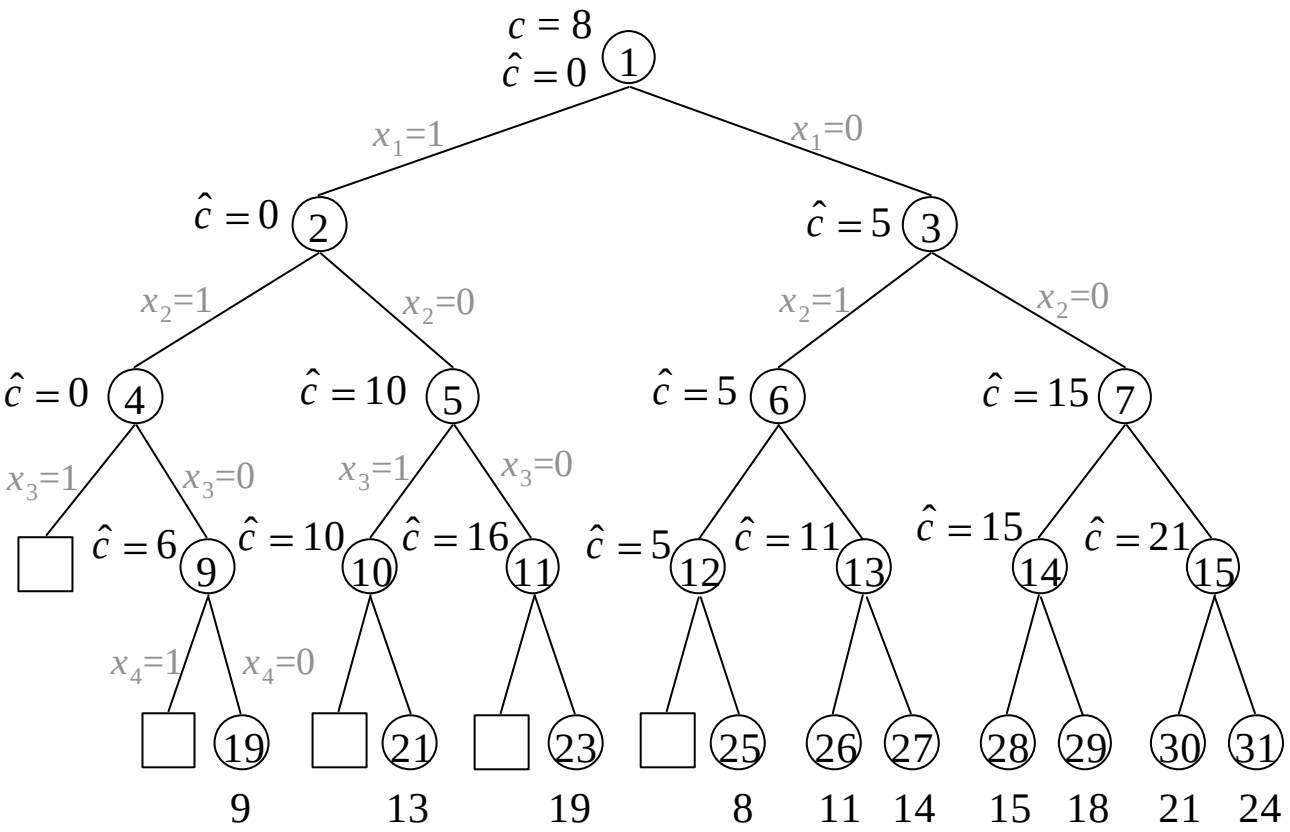



$$\hat{c}(x) = \sum_{\substack{i < m_x \\ i \notin S_x}} w_i$$

$(t_1,d_1,w_1) = (1,1,5)$
$(t_2,d_2,w_2) = (2,3,10)$
$(t_3,d_3,w_3) = (1,2,6)$
$(t_4,d_4,w_4) = (1,1,3)$

# 4 Un problema de planificación de tareas a plazo fijo

– En el caso de formulación de tamaño fijo:




$$\hat{c}(x) = \sum_{\substack{i < m_x \\ i \notin S_x}} w_i$$

$(t_1,d_1,w_1) = (1,1,5)$
$(t_2,d_2,w_2) = (2,3,10)$
$(t_3,d_3,w_3) = (1,2,6)$
$(t_4,d_4,w_4) = (1,1,3)$

# 4 Un problema de planificación de tareas a plazo fijo

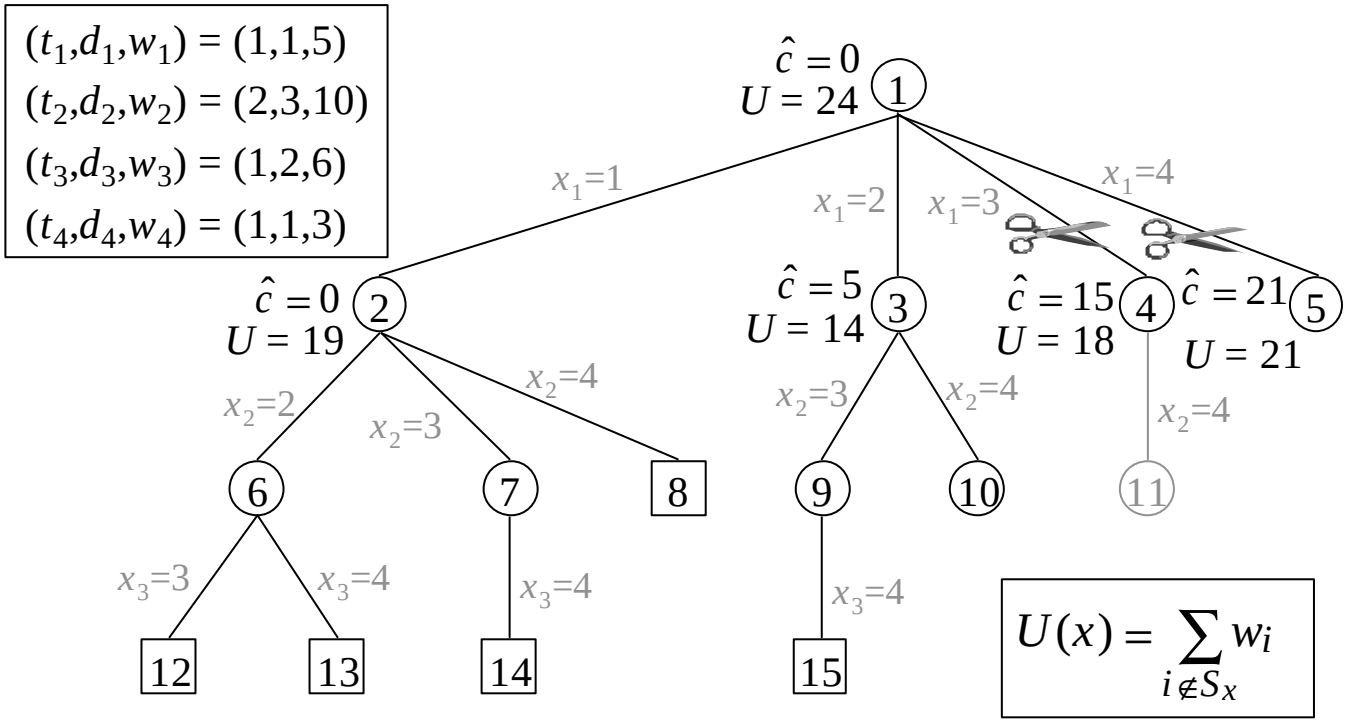
- Acotación:
  - Definición de $U(x)$:

    (Recordar: $U$ debe ser cota superior del coste de una solución de mínimo coste)

$$U(x) = \sum_{i \notin S_x} w_i$$

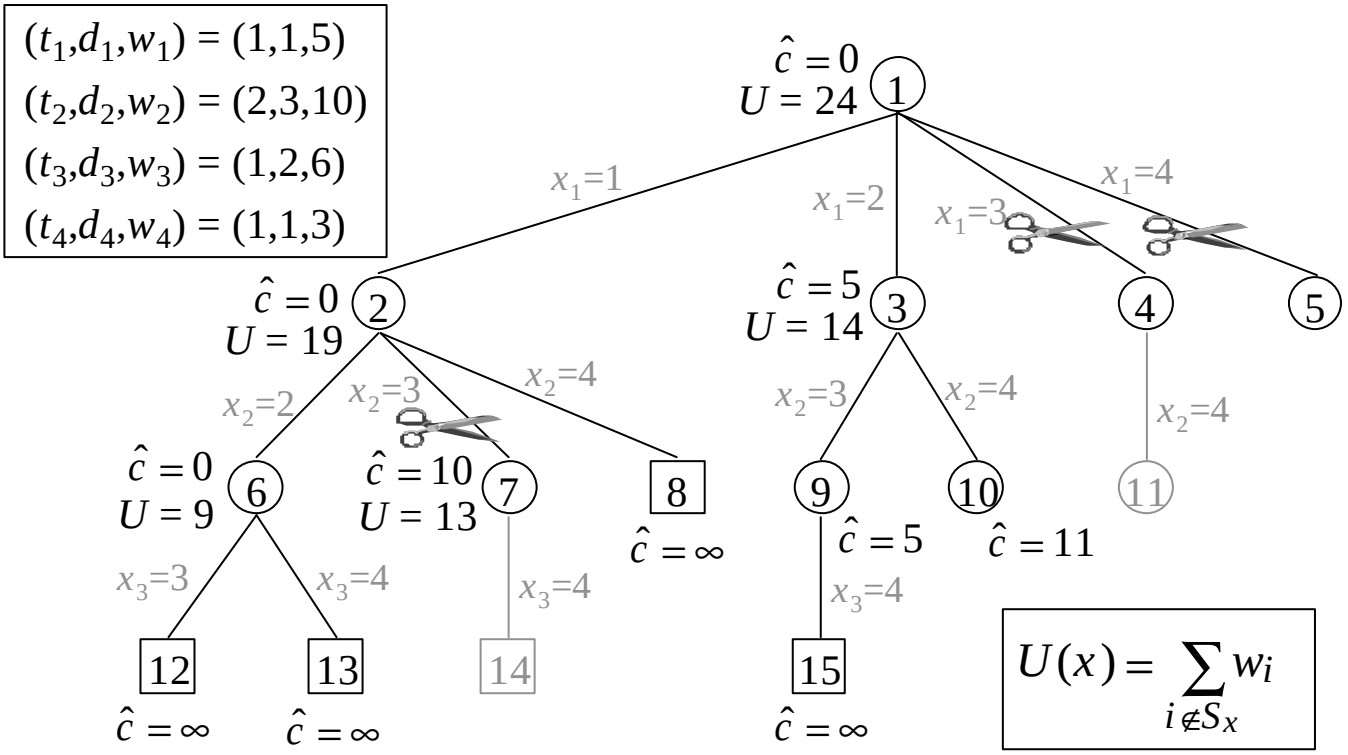
# 4 Un problema de planificación de tareas a plazo fijo

- Algoritmo de ramificación y acotación con estrategia de mínimo coste: (representación de tamaño variable)




- Empieza con $U = \infty$ (o $U = 24$) y con (1) como nodo en curso (único nodo vivo).
- Se expande (1) generando: (2), (3), (4), (5).
- $U$ se actualiza a 14 al generar (3);
  (4) y (5) se matan porque $\hat{c}(4) = 15 > U$, $\hat{c}(5) = 21 > U$

# 4 Un problema de planificación de tareas a plazo fijo



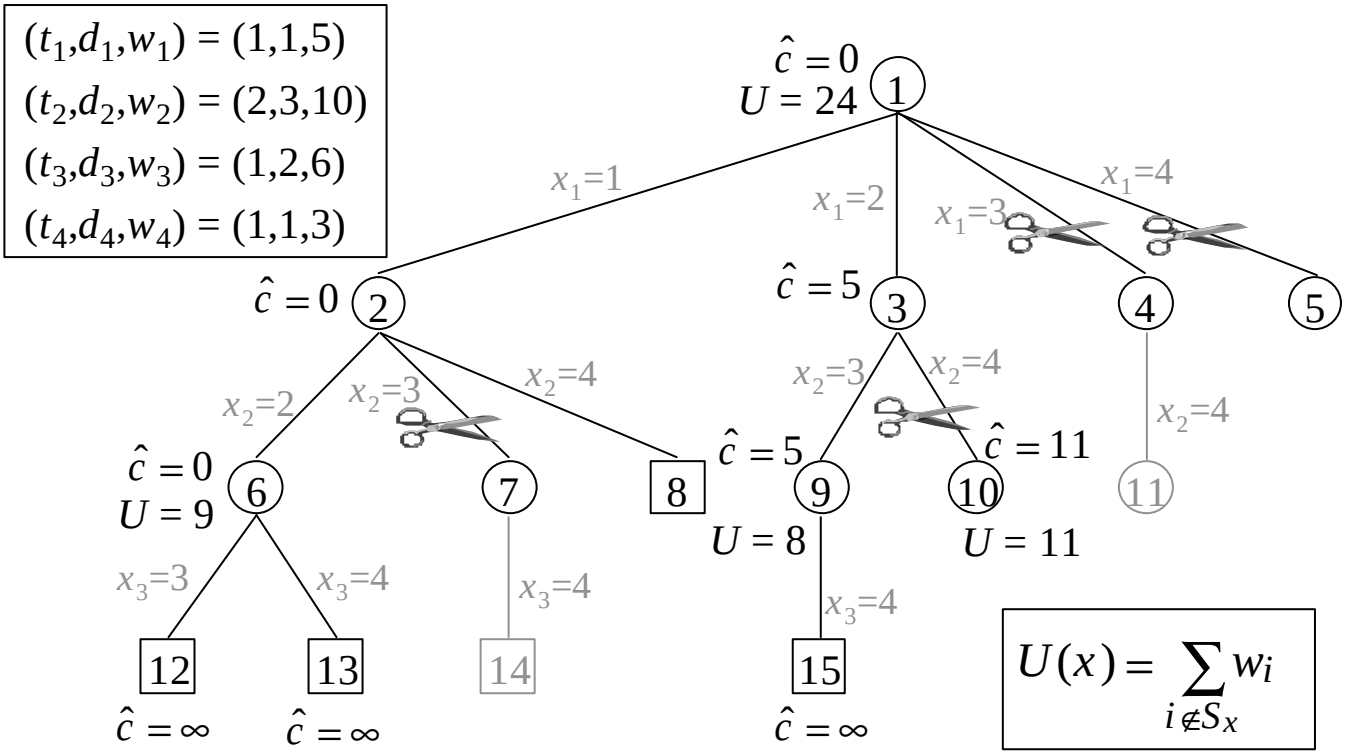

– El siguiente nodo en curso es (2).

$$\left(\hat{c}(2) = 0 < 5 = \hat{c}(3)\right)$$

– Se añaden a la cola: (6), (7), (8).
– Se actualiza $U$ a 9 al generar (6).
– El nodo (7) se mata porque $\hat{c}(7) = 10 > U = 9$.
– El nodo (8) se mata porque no es factible.
– Los nodos vivos son (3) y (6).
– El siguiente nodo en curso es (6).

$$\left(\hat{c}(6) = 0 < 5 = \hat{c}(3)\right)$$

# 4 Un problema de planificación de tareas a plazo fijo




- Los dos hijos de (6) son no factibles, luego el siguiente nodo en curso es el (3) .
- Al generar (9) , $U$ se actualiza a 8.
- El nodo (10) se mata porque $\hat{c}(10) = 11 > U = 8$.
- El único nodo vivo es (9) , luego se convierte en nodo actual. Su único hijo es no factible, por tanto se acaba la búsqueda y la solución es el nodo (9) .
- El algoritmo hubiera terminado también si el siguiente nodo actual, $x$, extraído de la cola fuese tal que $\hat{c}(x) \geq U$.

# 4 Un problema de planificación de tareas a plazo fijo

```
algoritmo RamAcotMinCoste(ent x0:nodo)
{Busca en el árbol cuya raíz es x0 una solución de
 mínimo coste. Se asume que existe al menos un nodo
 de mínimo coste y c_gorro(x)≤c(x)≤U(x). }
variables q:cola; {con prior. de <x,c_gorro(x)>}
          xcurso,x:nodo
principio
  creaVacía(q); {cola de nodos vivos}
  añadir(q,<x0,c_gorro(x0)>);
  mq not esVacía(q) hacer
    xcurso:=min(q); {nodo en curso}
    eliminarMin(q);
    para todo x hijo de xcurso hacer
      si c_gorro(x)≤U entonces
        añadir(q,<x,c_gorro(x)>);
        si esSol(x) and U(x)<U
          entonces U:=U(x)
        fsi
      fsi
    fpara;
    si esVacía(q) or c_gorro(min(q))≥U entonces
      escribir(U);
      creaVacía(q)
    fsi
  fmq
fin
```

# 4 Un problema de planificación de tareas a plazo fijo

- Si, como es lógico, se desea escribir la solución óptima además del valor óptimo de la función objetivo:
  - cada vez que se añade un nodo a la cola, hay que guardar en una tabla auxiliar de padres de nodos ese nodo junto a su padre o bien **guardar con cada nodo una identificación de su padre**, y
  - hay que mantener una variable auxiliar con el valor del último nodo solución que ha permitido actualizar *U*.

# 4.4 El problema de la mochila 0-1

- Recordar el problema de la mochila…
  - Se tienen $n$ objetos fraccionables y una mochila.
  - El objeto $i$ tiene peso $p_i$ y una fracción $x_i$ ($0 \leq x_i \leq 1$) del objeto $i$ produce un beneficio $b_i x_i$.
  - El objetivo es llenar la mochila, de capacidad $C$, de manera que se maximice el beneficio.

$$\text{maximizar} \quad \sum_{1 \leq i \leq n} b_i x_i$$

$$\text{sujeto a} \quad \sum_{1 \leq i \leq n} p_i x_i \leq C$$

$$\text{con} \quad 0 \leq x_i \leq 1,\ b_i > 0,\ p_i > 0,\ 1 \leq i \leq n$$

- … y su variante 0-1…
  - $x_i$ sólo toma valores 0 ó 1, indicando que el objeto se deja fuera o se mete en la mochila.
  - Los pesos, $p_i$, los beneficios, $b_i$, y la capacidad son números naturales.

- … pero sin exigir ahora que $p_i$, $b_i$ y $C$ sean naturales (sino simplemente reales positivos).

# 4 El problema de la mochila 0-1

- Soluciones previas a problemas parecidos:
  - Mochila con objetos fraccionables, es decir, $0 \leq x_i \leq 1$, $1 \leq i \leq n$: solución voraz.
  - Mochila 0-1 pero siendo los pesos, beneficios y capacidad de la mochila números naturales: solución de programación dinámica.

  → Ninguna de las dos soluciones funciona en el caso general de la mochila 0-1.

  - Caso general de la mochila 0-1: solución de búsqueda con retroceso.

  → Puede mejorarse la eficiencia.

# 4 El problema de la mochila 0-1

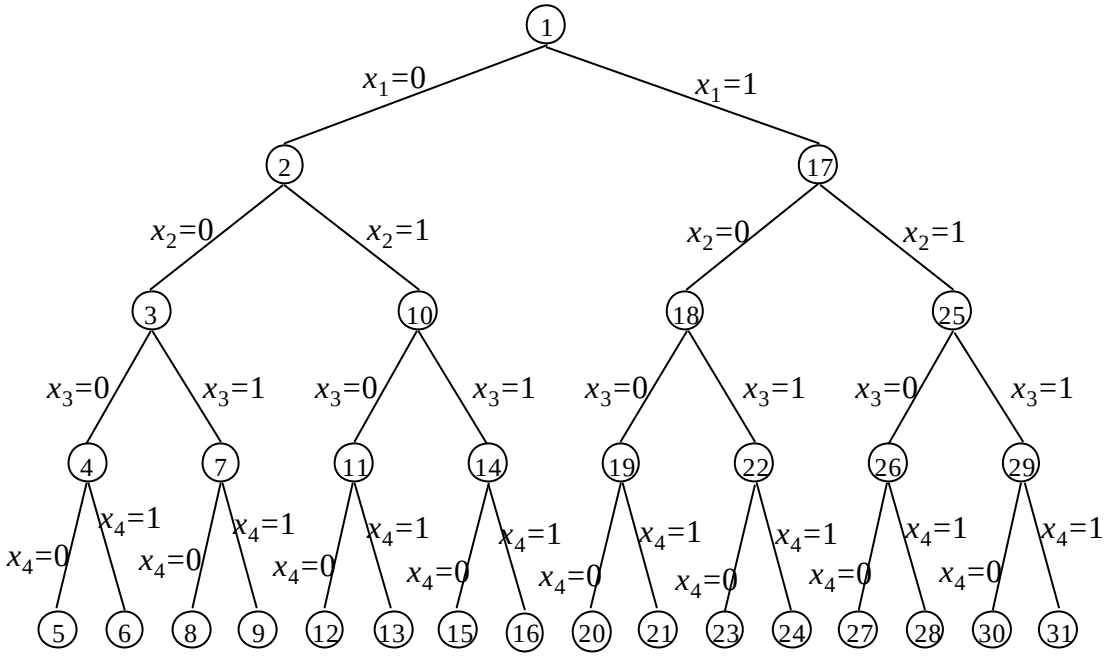
- Transformación en problema de minimización:

$$\text{minimizar} \quad -\sum_{1\le i\le n} b_i x_i$$

$$\text{sujeto a} \quad \sum_{1\le i\le n} p_i x_i \le C$$

$$\text{con} \quad x_i \in \{0,1\},\ b_i > 0,\ p_i > 0,\ 1 \le i \le n$$

- Espacio de soluciones:
  - $2^n$ modos de asignar los valores 0 ó 1 a las $x_i$.
  - Dos formas de representar la solución: tuplas de tamaño fijo o variable. Tamaño fijo:



# El problema de la mochila 0-1

- Definición de la función de coste, $c$:
  - Las hojas $x$ que ponen fin a caminos desde la raíz tales que

$$\sum_{1 \le i \le n} p_i x_i > C$$

    representan soluciones no factibles y para ellas $c(x)=\infty$.

  - El resto de las hojas $x$ representan soluciones factibles. Para ellas

$$c(x) = -\sum_{1 \le i \le n} b_i x_i$$

  - Para nodos $x$ que no sean hojas, $c(x)$ es el mínimo entre $c(x_{iz})$ y $c(x_{de})$, siendo $x_{iz}$ y $x_{de}$ los hijos izquierdo y derecho, resp., de $x$.

    → Por supuesto, su cálculo es tan difícil como resolver el problema original…

# El problema de la mochila 0-1

- Definición de dos funciones $\hat{c}$ y $U$ tales que

$$\hat{c}(x) \leq c(x) \leq U(x)$$

- Función $U$:
  - Solución más sencilla:

    Si $x$ es un nodo de nivel $j$, con $0 \leq j \leq n$, se han asignado ya valores a $x_i$, $1 \leq i \leq j$, por tanto:

$$c(x) \leq -\sum_{1 \leq i \leq j} b_i x_i = U(x)$$

    (i.e., beneficio actual cambiado de signo).

# 4 El problema de la mochila 0-1

– Solución mejorada:

```
constante n=... {número de objetos}
tipo vectReal=vector[1..n] de real
variables C:real; benef,peso:vectReal
{Pre: ∀i∈1..n:peso[i]>0 ∧
 ∀i∈1..n-1:benef[i]/peso[i]≥benef[i+1]/peso[i+1]}
```

```
función U_mejor(cap,Ufácil:real; obj:entero)
        devuelve real
{cap=capacidad ya ocupada de la mochila;
 Ufácil=valor de U calculado de forma fácil;
 obj=índice del primer objeto a considerar}
variable ca:real
principio
  U:=Ufácil; ca:=cap;
  para i:=obj hasta n hacer
    si ca+peso[i]≤C entonces
      ca:=ca+peso[i]; U:=U-benef[i]
    fsi
  fpara;
  devuelve U
fin
```

$$U(x) = \mathrm{U\_mejor}\left( \sum_{1 \le i \le j} p_i x_i \ , \ - \sum_{1 \le i \le j} b_i x_i \ , \ j+1 \right)$$

# El problema de la mochila 0-1

- Función $\hat{c}$:
  - Utilizar la función acotadora (o "de poda") que se definió para el método de búsqueda con retroceso:
    - era una cota superior del valor de la mejor solución posible al expandir el nodo y sus descendientes,
    - ahora hemos cambiado de signo a la función objetivo,

    → luego, cambiada de signo, es una cota inferior de $c(x)$

  - ¿Cómo se calculó esa cota?
    - en el nodo actual ya se han determinado $x_i$, $1 \le i \le j$;
    - relajar el requisito de integridad: $x_i \in \{0,1\}$, $j+1 \le i \le n$ se sustituye por $0 \le x_i \le 1$, $j+1 \le i \le n$
    - aplicar el algoritmo voraz

# 4 El problema de la mochila 0-1

```
constante n=... {número de objetos}
tipo vectReal=vector[1..n] de real
variables C:real; benef,peso:vectReal
{Pre: ∀i∈1..n:peso[i]>0 ∧
 ∀i∈1..n-1:benef[i]/peso[i]≥benef[i+1]/peso[i+1]}
```

```
función cota(cap,ben:real; obj:entero)
        devuelve real
{cap=capacidad aún libre de la mochila;
 ben=beneficio actual;
 obj=índice del primer objeto a considerar}
principio
  si obj>n or cap=0.0
    entonces devuelve ben
    sino
      si peso[obj]>cap
        entonces
          dev ben+cap/peso[obj]*benef[obj]
        sino
          dev cota(cap-peso[obj],
                   ben+benef[obj],obj+1)
      fsi
  fsi
fin
```

$$\hat{c}(x) = -\text{cota}\left( C - \sum_{1 \le i \le j} p_i x_i \ , \ \sum_{1 \le i \le j} b_i x_i \ , \ j+1 \right)$$

# El problema de la mochila 0-1

- Ejemplo:

| |
|---|
| $n = 4$; |
| $(b_1,b_2,b_3,b_4) = (10,10,12,18)$ |
| $(p_1,p_2,p_3,p_4) = (2,4,6,9)$ |
| $C = 15$ |

① primer nodo vivo (raíz, nivel 0, ninguna $x_i$ asignada)

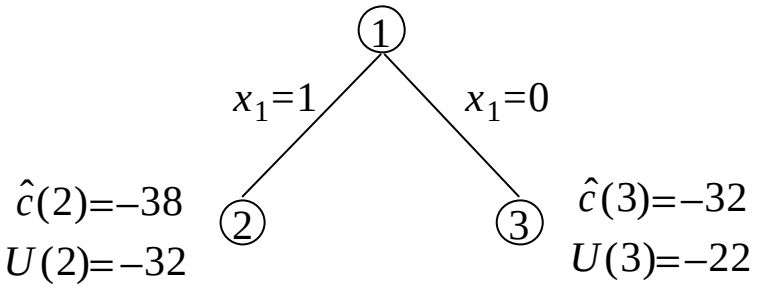
$$\hat{c}(1) = -\texttt{cota}(15,0,1) = -\texttt{cota}(13,10,2) =$$

$$= -\texttt{cota}(9,20,3) = -\texttt{cota}(3,32,4) =$$

$$= -(32 + 3/9 * 18) = -38$$

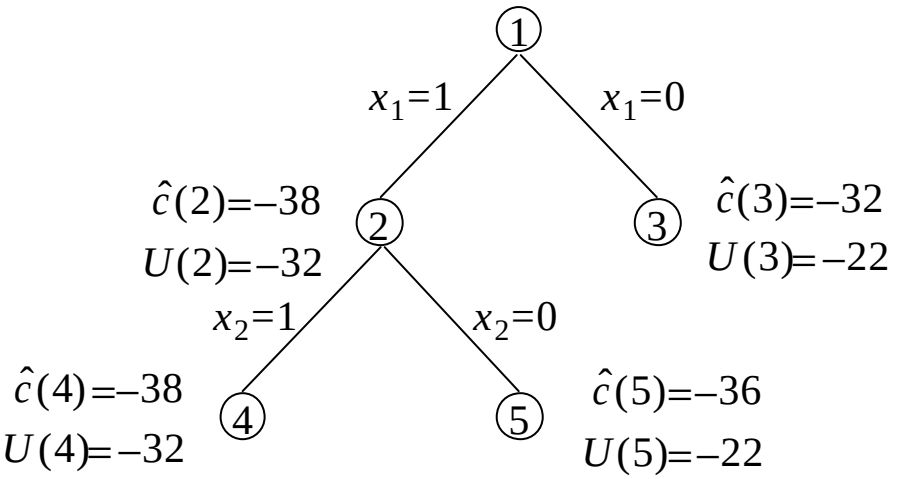
$$U(1) = -32$$

El valor actual de $U$ es -32; el nodo se expande:

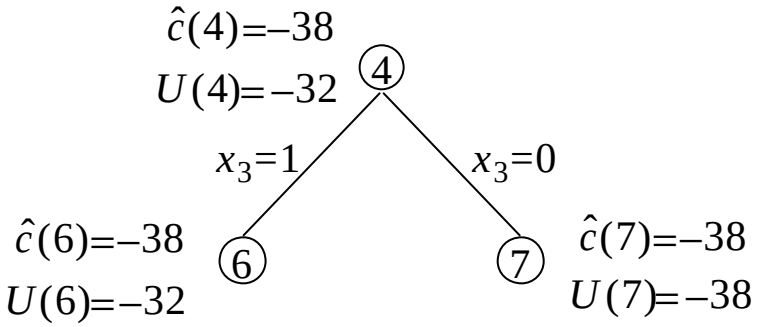


# 4 El problema de la mochila 0-1

② Siguiente nodo en curso. Se expande…



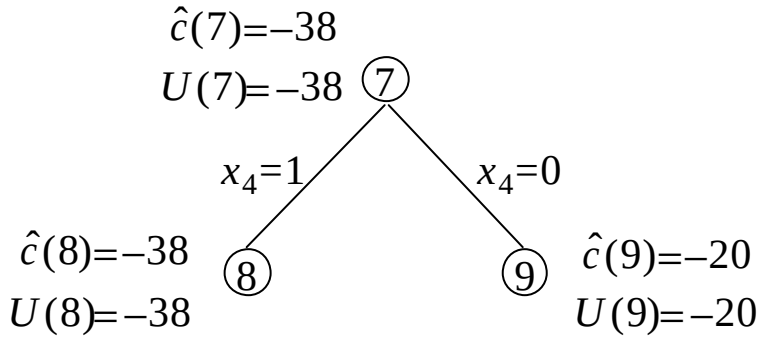

④ Siguiente nodo en curso. Se expande…




¿Siguiente nodo en curso? Hay dos con igual prioridad: 6 y 7.
Si es elegido el 6, se generan sus hijos:
el izquierdo es eliminado por no ser factible,
el derecho se añade a la cola de vivos (con prioridad 32,
menor que el nodo 7), luego el siguiente elegido es el 7.

# 4 El problema de la mochila 0-1

⑦ Siguiente nodo en curso.
Nuevo valor de $U$ = -38. Se expande...




El nodo 9 se mata (no alcanza el valor de $U$).
El nodo 8 es el único en la cola.
El valor óptimo es 38.

# 4 El problema de la mochila 0-1

- Comentarios adicionales:
  - Para poder imprimir la solución (además del valor óptimo) hay que:
    - guardar con cada nodo una identificación de su padre (identificador, puntero, cursor, …),
    - mantener una variable auxiliar con el valor del último nodo solución que ha permitido actualizar *U*, y
    - asociar con cada nodo un booleano que diga si es un hijo izquierdo o derecho (es decir, si mete en la mochila el objeto correspondiente o no).
  - Cabe pensar en una versión con estrategia ciega de ramificación (FIFO, por ejemplo) en lugar de la de mínimo coste:
    - por un lado la intuición nos dice que la estrategia de mínimo coste se acerca más deprisa a la solución óptima,
    - por otra parte, la inserción y borrado en la cola con prioridades de nodos vivos tiene coste logarítmico, mientras que con una cola FIFO, el coste es constante…

      → depende de los datos de entrada…

# 4 *El problema del viajante de comercio*

- Recordar:


  - Encontrar un recorrido de longitud mínima para un viajante que tiene que visitar varias ciudades y volver al punto de partida, conocida la distancia existente entre cada dos ciudades.

  - Es decir, dado un grafo dirigido con arcos de longitud no negativa, se trata de encontrar un circuito de longitud mínima que comience y termine en el mismo vértice y pase exactamente una vez por cada uno de los vértices restantes (circuito *hamiltoniano*).

# 4 *El problema del viajante de comercio*

- Soluciones ensayadas hasta ahora:
  - Heurística voraz: calcula una solución subóptima
  - Programación dinámica:
    - coste en tiempo: $\Theta(n^2 2^n)$
    - coste en espacio: $\Omega(n 2^n)$

- ¡Hay muchas más soluciones!
  - "…El problema del viajante de comercio es probablemente el problema *NP-completo* más estudiado en términos de soluciones propuestas…"

    (U. Manber: *Introduction to Algorithms. A Creative Approach.* Addison-Wesley, 1989.)

  - Ya en el año 85…

    E.L. Lawler, J.K. Lenstra,
    A.H.G. Rinnooy Kan, y D.B. Shmoys:
    *The Traveling Salesman Problem.*
    John Wiley & Sons, New York, 1985.

# 4 *El problema del viajante de comercio*

- El coste del caso peor de la solución que vamos a ver seguirá estando en $O(n^2 2^n)$.

- Sin embargo, el uso de buenas funciones de acotación mejora la eficiencia en muchos casos con respecto a la solución de programación dinámica.

# 4 El problema del viajante de comercio

- Formalización:
  - Sean $G=(V,A)$ un grafo orientado,
    $V=\{1,2,\ldots,n\}$,
    $L_{ij}$ la longitud de $(i,j)\in A$,
    $L_{ij}=\infty$ si no existe el arco $(i,j)$.

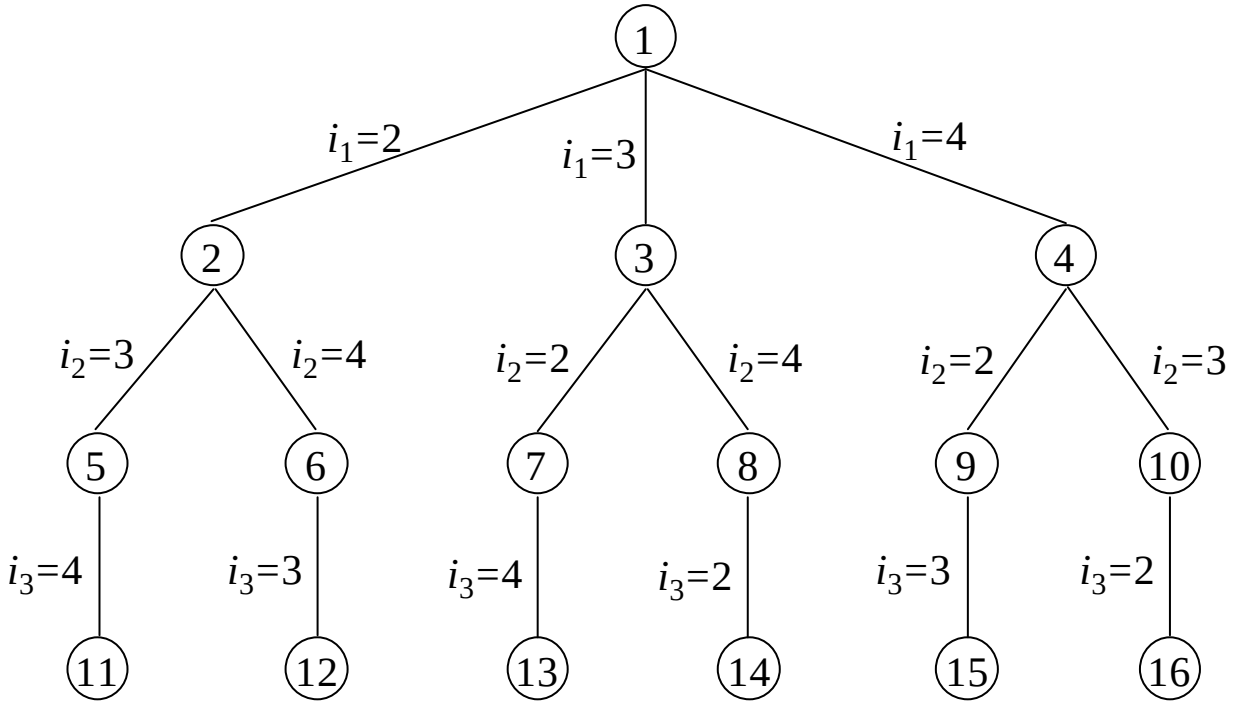
  - El circuito buscado empieza en el vértice 1.

  - Espacio de soluciones:
    $E = \{\ 1,p,1 \mid$ p es una permutación de $(2,3,\ldots,n)\ \}$
    $|E| = (n\text{-}1)!$

  - Reducción del espacio de soluciones:
    $E = \{\ 1,p,1 \mid p = i_1,i_2,\ldots,i_{n-1}$, es una permutación de
    $(2,3,\ldots,n)$ tal que $(i_j,i_{j+1})\in A$, $0\le j\le n\text{-}1$,
    $i_0=i_n=1\ \}$

# 4 El problema del viajante de comercio

- Representación del espacio de estados:
  - Caso de un grafo completo con $|V| = 4$.




Cada hoja es una solución y representa el viaje definido por el camino desde la raíz hasta la hoja.

# 4 *El problema del viajante de comercio*

- Definición de la función de coste, *c*:

  – Debe ser tal que el nodo solución *x* con valor mínimo de *c*(*x*) corresponda a un recorrido de longitud mínima.

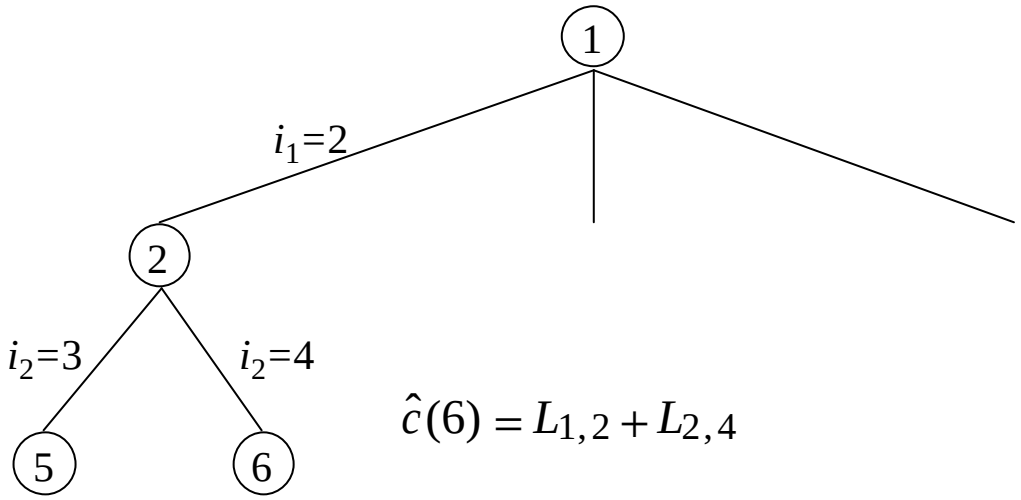
  – Por ejemplo,

$$c(x) = \begin{cases} \text{longitud del recorrido definido por el camino desde la raíz hasta } x, \text{ si } x \text{ es una hoja} \\ \text{coste de una hoja de coste mínimo del subárbol con raíz } x, \text{ si } x \text{ no es una hoja} \end{cases}$$

# 4 El problema del viajante de comercio

- Definición de la función de estimación $\hat{c}$
  - Debe ser tal que: $\hat{c}(x) \leq c(x)$
  - Una muy fácil:

$\hat{c}(x) =$ distancia total del recorrido definido por el camino desde la raíz hasta $x$




Si $x$ es una hoja, es obvio que:

$$\hat{c}(x) = c(x)$$

# 4 *El problema del viajante de comercio*

- Puede mejorarse usando la **matriz de distancias reducida**:
  - Una fila (columna) de la matriz de distancias se dice **reducida** si sus elementos son no negativos y contiene al menos un 0.
  - Una matriz de distancias se dice reducida si cada fila y columna es reducida.

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

Ejemplo de matriz **no** reducida.

  - Para cada $k$, $1 \le k \le n$, todo circuito hamiltoniano incluye exactamente un arco de la forma $(k,-)$ y exactamente un arco de la forma $(-,k)$.

ß

  - Si se resta una constante $t$ de cada elemento de una fila (columna) de la matriz de distancias, la longitud de todo hamiltoniano se reduce exactamente en $t$ y un hamiltoniano de distancia mínima lo sigue siendo.

# 4 El problema del viajante de comercio

– Si se elige $t$ como el mínimo de los elementos de la fila (columna) $i$-ésima y se resta $t$ de todos los elementos de esa fila (columna), la fila resultante es **reducida**.

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \quad \begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

Reducción de la fila 1, $t = 10$.

– Repitiendo el proceso para filas y columnas, siempre se puede conseguir que la matriz de distancias sea reducida.

Reducción de la matriz, $L = 25$.

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \quad \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

La cantidad total $L$ restada de filas y columnas es una cota inferior de la longitud de un hamiltoniano de longitud mínima, luego sirve como $\hat{c}(x)$ para el nodo $x$ raíz.

# 4 El problema del viajante de comercio

– Cálculo de $\hat{c}(x)$ para los nodos distintos de la raíz y de las hojas:

- Sea $A$ la matriz de distancias reducida para el nodo $y$.
- Sea $x$ un hijo de $y$ que corresponda a incluir el arco $(i,j)$ en el recorrido y que no sea hoja.
- La matriz $B$ reducida para $x$, y por tanto $\hat{c}(x)$, se calcula de la siguiente forma:

1. Cambiar todos los elementos de la fila $i$ y de la columna $j$ de $A$ por $\infty$.
   – Esto evita el incluir más arcos que salgan de $i$ o lleguen a $j$.
2. Cambiar el elemento $(j,1)$ de $A$ por $\infty$.
   – Esto evita considerar el arco $(j,1)$.
3. $B$ es la matriz que se obtiene al reducir todas las filas y columnas de la matriz resultante (excepto aquéllas formadas sólo por "$\infty$").

- Si $r$ es el valor total restado en el paso (3):

$$\hat{c}(x) = \hat{c}(y) + A(i,j) + r$$

# 4 *El problema del viajante de comercio*

- Definición de la función de acotación $U$:
  - Por simplificar la exposición, podemos adoptar la función de acotación trivial:
    - Inicialmente, $U = \infty$.
    - Se modifica $U$ sólo cuando se llega a un nodo $x$ que es una hoja; entonces:

$$U = \min \{U, c(x)\}$$

- Ejercicio:

  Pensar mejores funciones de acotación.

# 4 El problema del viajante de comercio

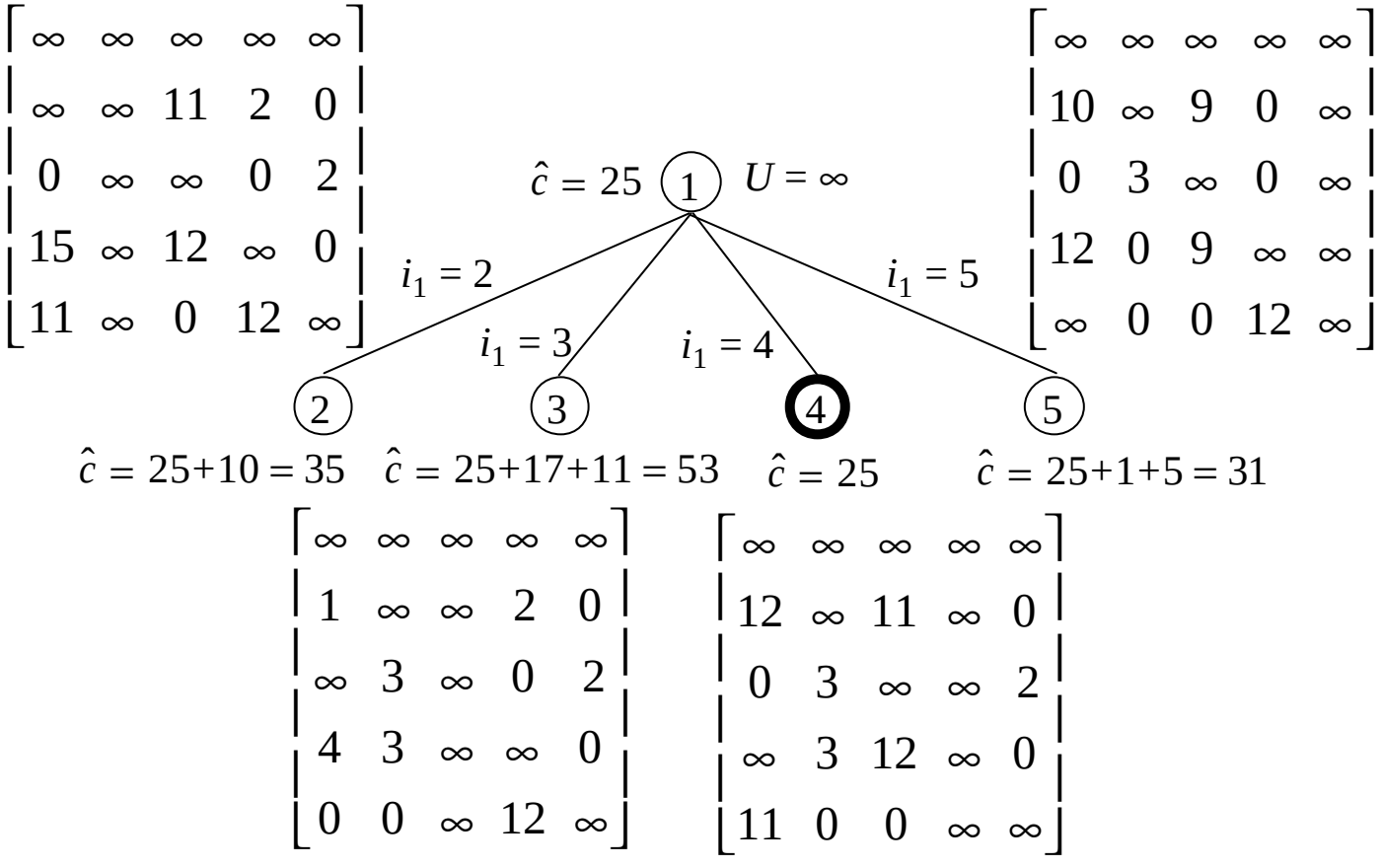
- Ejemplo:

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

Grafo original.

$$\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

Matriz reducida, $L = 25$.



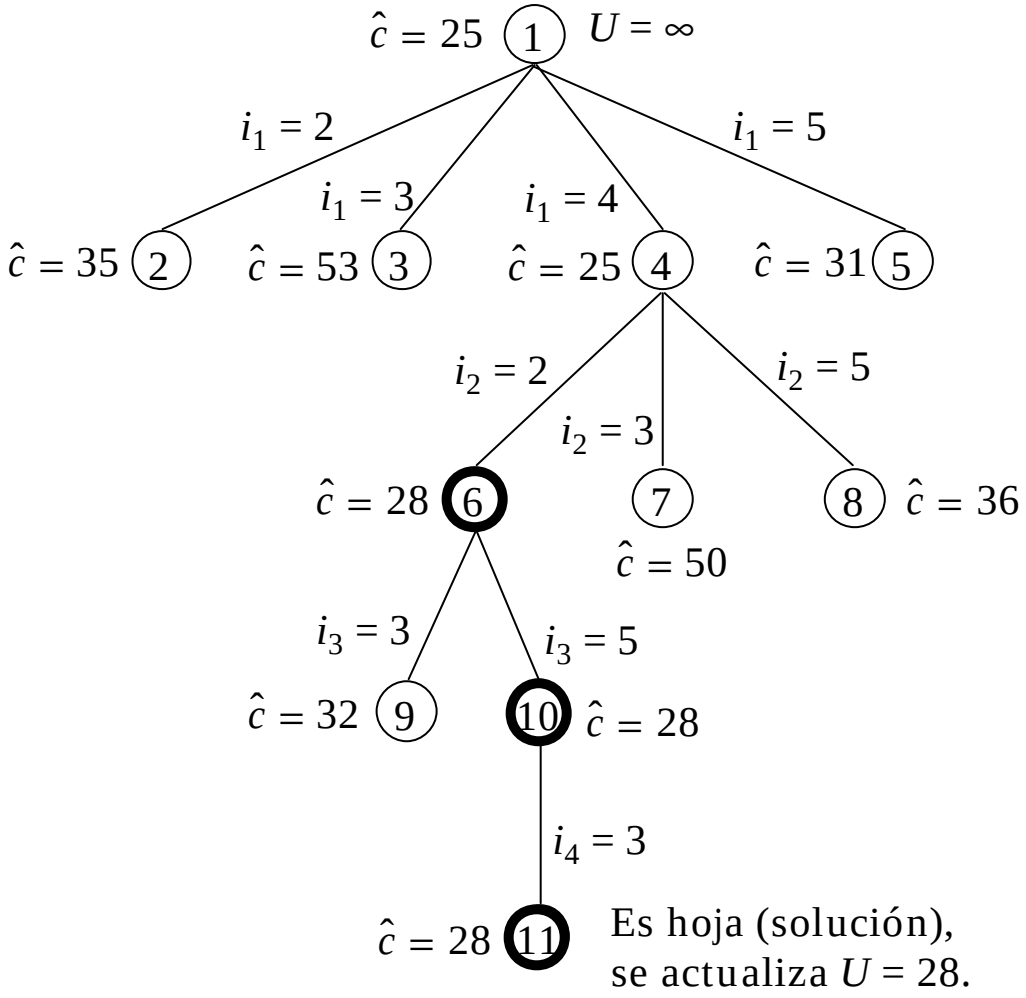

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}$$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

# 4 El problema del viajante de comercio



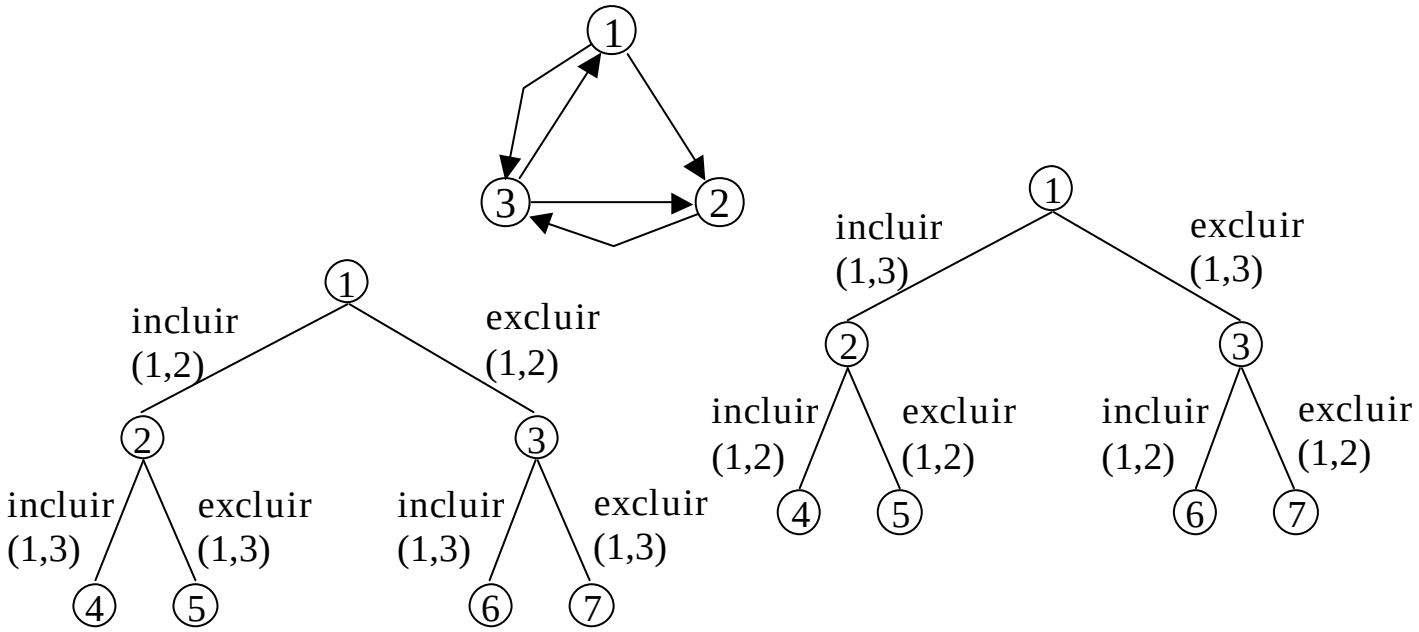

El siguiente nodo en curso sería el 5, pero $\hat{c}(5) = 31 > U$ luego el algoritmo termina y el hamiltoniano mínimo es 1,4,2,5,3,1.

# 4 *El problema del viajante de comercio*

- Otras versiones, basadas en otra representación del espacio de estados:
  - Un hamiltoniano es un **conjunto** de $n$ arcos.
  - Además, para cada vértice $i$, $1 \leq i \leq n$, debe haber en ese conjunto exactamente un arco de la forma $(i,j)$ y uno de la forma $(k,i)$.
  - Espacio de estados = árbol binario:
    - Un hijo izquierdo representa la inclusión de un determinado arco en el hamiltoniano mientras que su hermano derecho representa la exclusión de ese arco.

# 4 *El problema del viajante de comercio*

– Hay distintos árboles, dependiendo de qué arco se elige para incluir / excluir en cada paso.

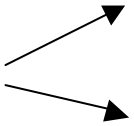



– ¿Cómo elegir el arco por el que empezar?
Depende de los datos del problema.

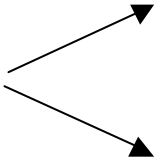
– Árbol de estados **dinámico**: el método de construcción depende de los datos del problema.

# 4 El problema del viajante de comercio

– Si se elige, para empezar, el arco $(i,j)$:



- el subárbol izquierdo representa todos los recorridos que incluyen el arco $(i,j)$, y
- el subárbol derecho los recorridos que no lo incluyen;



- si hay un recorrido óptimo incluido en el subárbol izquierdo, entonces sólo faltan por seleccionar $n$-1 arcos para encontrarlo,
- mientras que si todos están en el derecho, hay que seleccionar todavía $n$ arcos.

$\Rightarrow$ Hay que intentar seleccionar un arco que tenga **la máxima probabilidad de estar en el recorrido óptimo**.

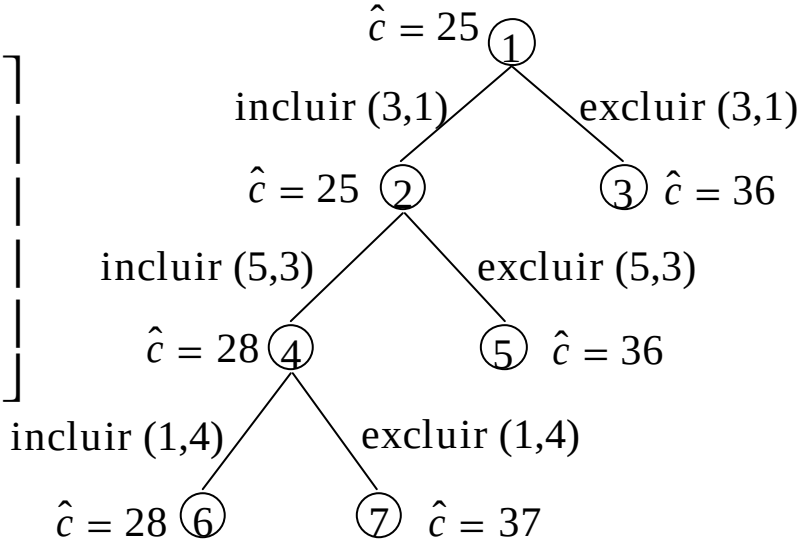
Hay varias heurísticas posibles, por ejemplo:

- Elegir un arco que produzca un subárbol derecho cuya raíz $x$ tenga $\hat{c}(x)$ máxima.
- Elegir un arco tal que la diferencia entre la función $\hat{c}$ para el hijo izquierdo y el derecho sea máxima.

# 4 El problema del viajante de comercio

– Aplicando la primera de las heurísticas al mismo grafo:

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$




Se llega al nodo 6.

Se han elegido ya tres arcos: (3,1), (5,3), (1,4).

Para los dos restantes, sólo queda ya una opción:

(4,2) y (2,5).

Así, se obtiene el recorrido: 5,3,1,4,2,5.

Con distancia total: 28 (así, $U = 28$)

El siguiente nodo en curso es el 3, con $\hat{c}(3) = 36 > U$ y el algoritmo acaba.

# 4 El problema del viajante de comercio

- Comentarios:

  - En el último ejemplo, el método de ramificación y acotación se usa sólo para subárboles grandes;

    una vez que se llega a subárboles pequeños (4 ó 6 nodos) es más eficiente construirlos enteros y calcular el valor exacto de la función de coste.

  - No hay forma de calcular analíticamente cuál de las soluciones del problema del viajante de comercio es más eficiente.

# 4 Consideraciones finales sobre eficiencia:

- Es casi imposible de estimar el rendimiento de la técnica para un problema y funciones de estimación y acotación dados.

- ¿Es posible disminuir en algún caso el número de nodos generados mediante la expansión de nodos $x$ con $\hat{c}(x) > U$ ?

  **Nunca** (porque el valor de $U$ no será modificado si se expande tal nodo $x$, y es el valor de $U$ el que determina qué nodos se expanden en el futuro).

- Si se tienen dos funciones de acotación $U_1$ y $U_2$ diferentes y $U_1 < U_2$ entonces **siempre** es mejor usar $U_1$ (i.e., nunca dará lugar a más nodos).

- ¿Si se usa una función de estimación mejor, decrece el número de nodos a expandir?
  ($\hat{c}_2$ es mejor que $\hat{c}_1$ si $\hat{c}_1(x) \leq \hat{c}_2(x) \leq c(x)$ para todo $x$)

  **Depende**.

  Si se usa una función de estimación mejor con las estrategias de ramificación **FIFO** o **LIFO**, **nunca** se incrementa el número de nodos generados.

  Si se usa una función de estimación mejor con la estrategia de ramificación de **mínimo coste**, el número de nodos generados **puede** crecer.