Algoritmos y Estructuras de Datos II Clase 18

Carreras:

Licenciatura en Informática

Ingeniería en Informática

2024

Unidad III

Técnicas de diseño de algoritmos

Ramificación y poda(2)

(en inglés: ***Branch and Bound***)

Asignación De Tareas

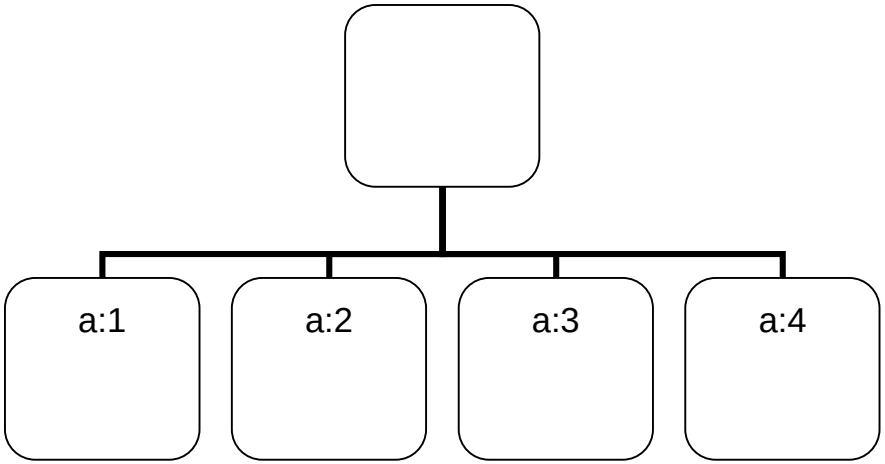
Para resolver el problema mediante ramificación y poda, se construye un árbol de expansión:

- En la raíz no se hacen asignaciones.
- Los nodos corresponden a asignaciones parciales de la solución.
- La ramificación se hace por personas: en cada nivel se determina la asignación de una persona más.
- Para cada nodo, se calcula una cota de las soluciones que se pueden obtener completando la asignación parcial correspondiente. La estimación de cota se hace tomando el valor mínimo de cada fila o columna.
- Se usa esa cota para podar los nodos que no van a llevar a una mejor solución.
- La búsqueda de la solución se continúa con el nodo que sea el candidato más prometedor (montículo) .

Asignación De Tareas

- Se comienza por el agente a, entonces hay 4 ramas desde la raíz.

A/T	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

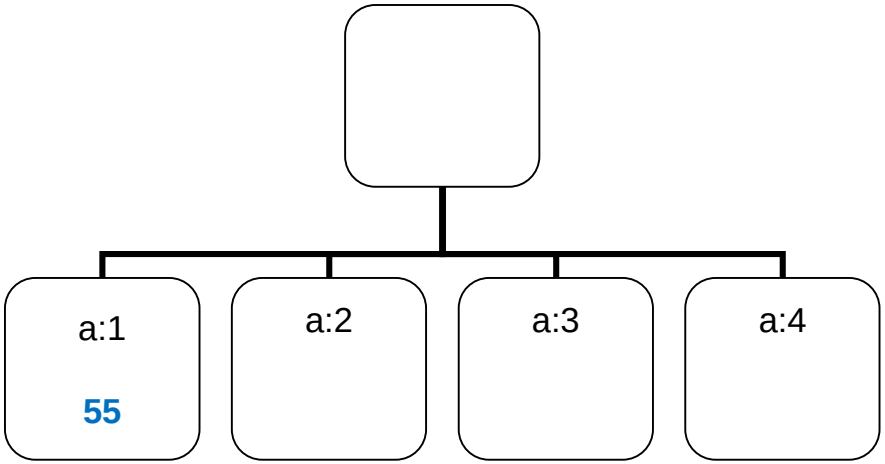


- Hay que determinar la cota en cada nodo.
- Se hace la estimación de la cota con el mínimo por fila.

Asignación De Tareas

- Calcular la cota inferior para el nodo (a:1)

A/T	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

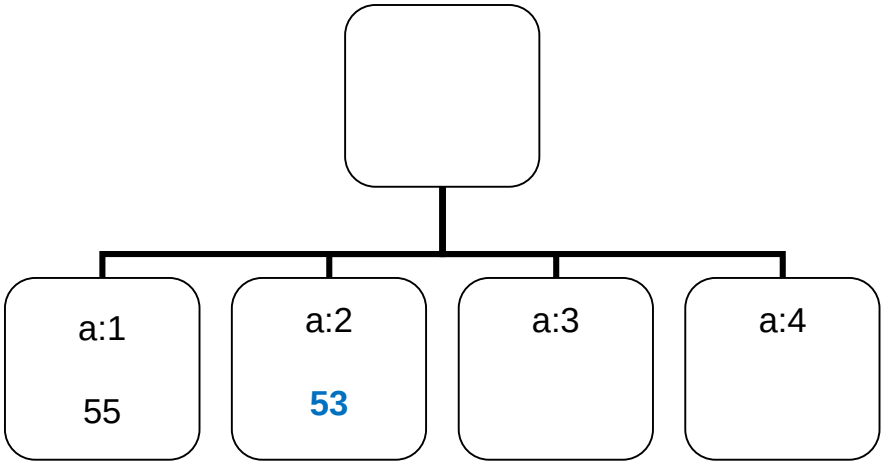


a:1 11+13+17+14=55

Asignación De Tareas

- Calcular la cota para inferior el nodo (a:2)

A/T	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

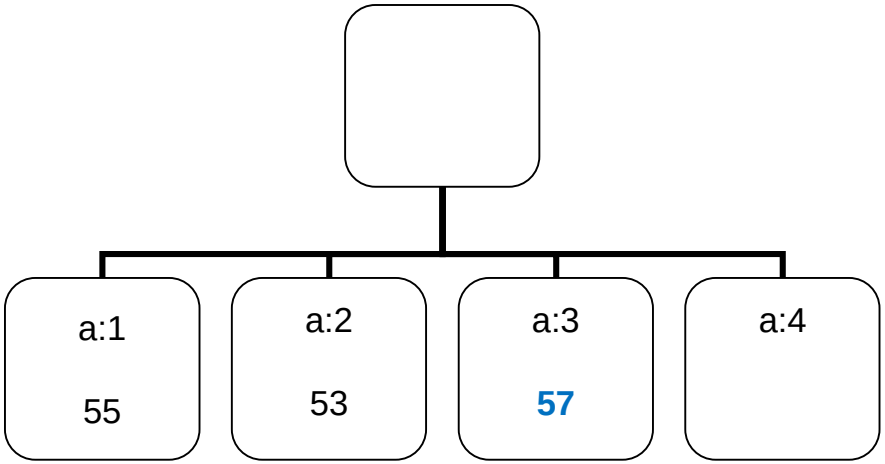


a:2 12+13+11+17=53

Asignación De Tareas

- Calcular la cota inferior para el nodo (a:3)

A/T	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

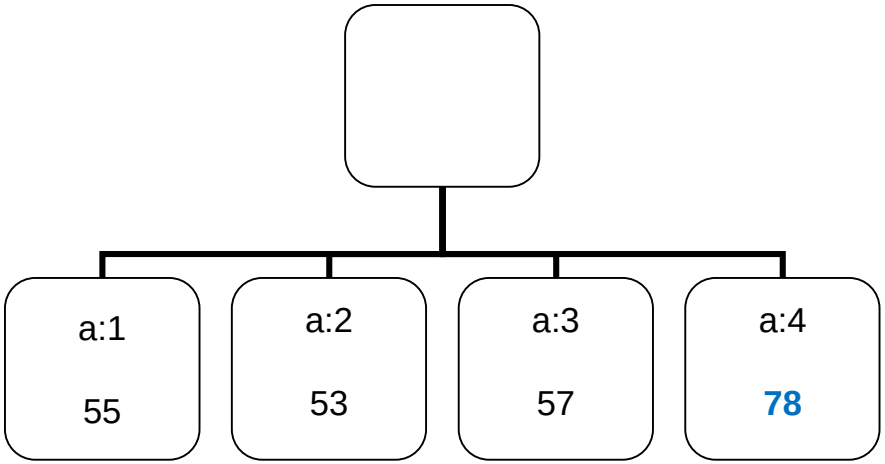


a:3 18+14+11+14=57

Asignación De Tareas

- Calcular la cota inferior para el nodo (a:4)

A/T	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

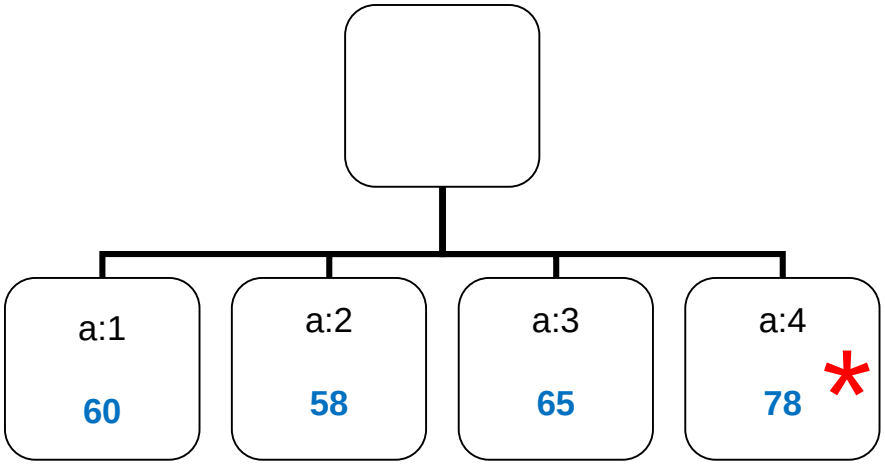


a:4 40+13+11+14=78

Asignación De Tareas

- Cota inferior para el agente a con cada una de las 4 tareas:

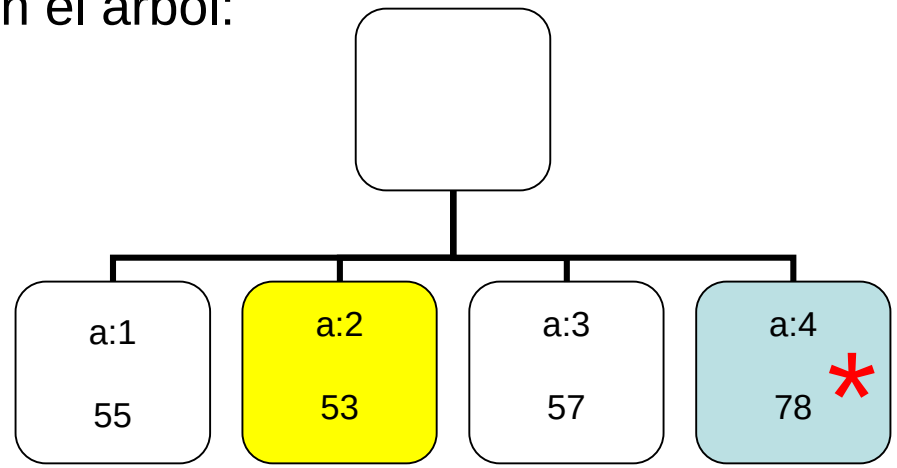
A/T	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28



- El asterisco * en el último nodo (a:4) denota que está podado porque ya se ha encontrado una solución de valor 73.

Asignación De Tareas

- Entonces si se comienza por el agente a se obtienen 4 nodos en el primer nivel en el árbol:

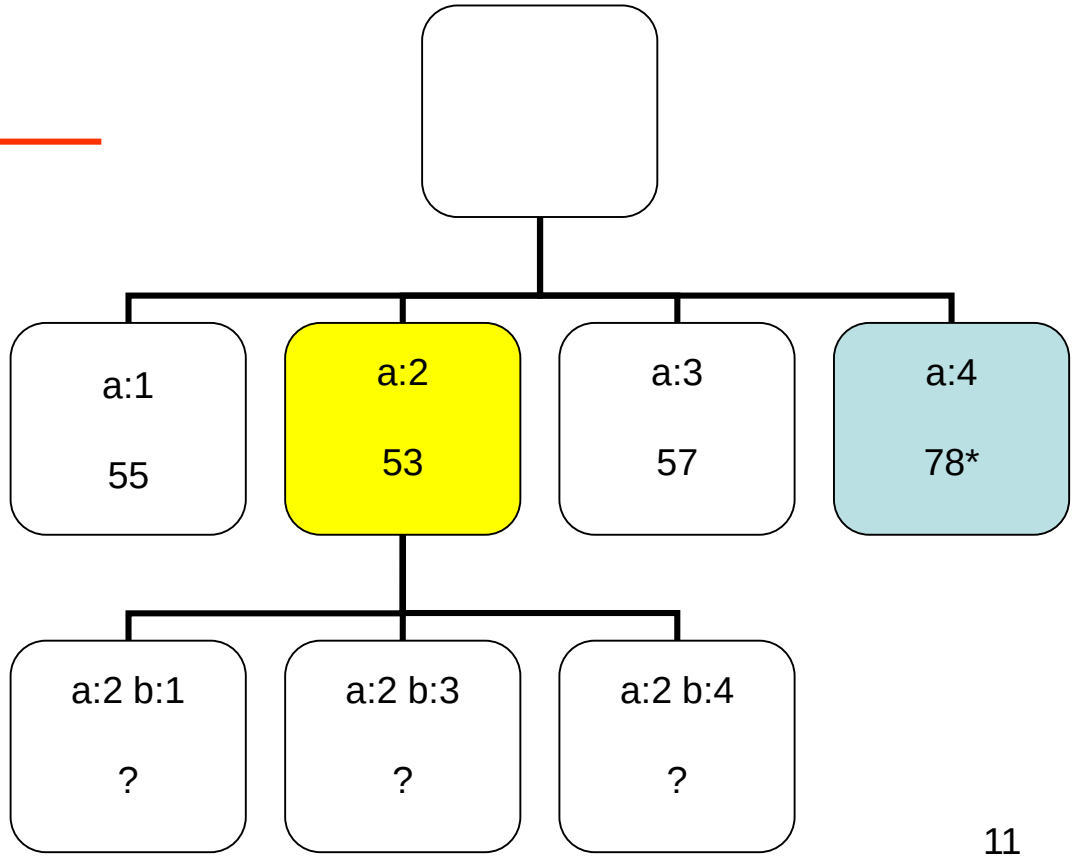


- La cifra obtenida para cada nodo es una cota inferior para las soluciones que se pueden obtener de ese nodo.
- NOTA: estos valores de cota inferior no son necesariamente alcanzables.

Asignación De Tareas

- De los 4 nodos, el nodo más prometedor es (a:2), tiene la cota inferior menor de todas (53), se explora primero esa asignación:

A/T	1	2	3	4
a	~~11~~	~~12~~	~~18~~	~~40~~
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

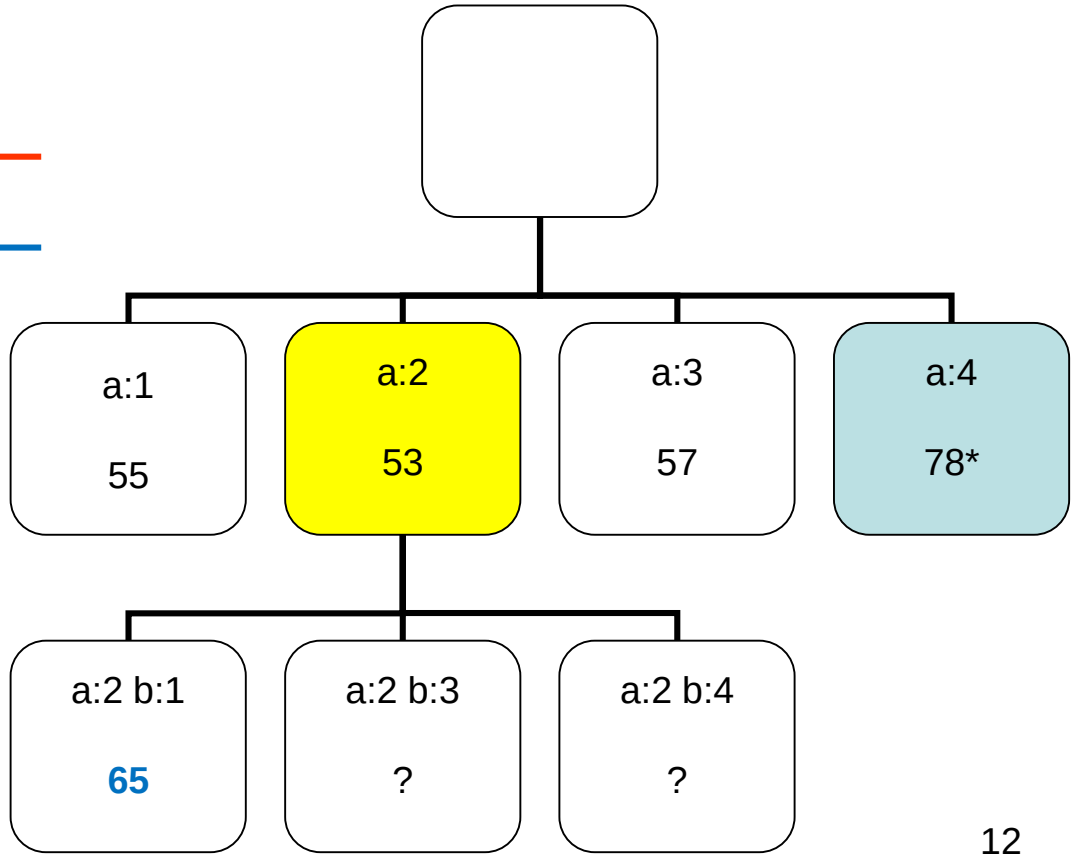


Asignación De Tareas

- Se calcula la cota de la asignación de (a:2), (b:1):

A/T	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

a:2 b:1 12+14+19+20=**65**

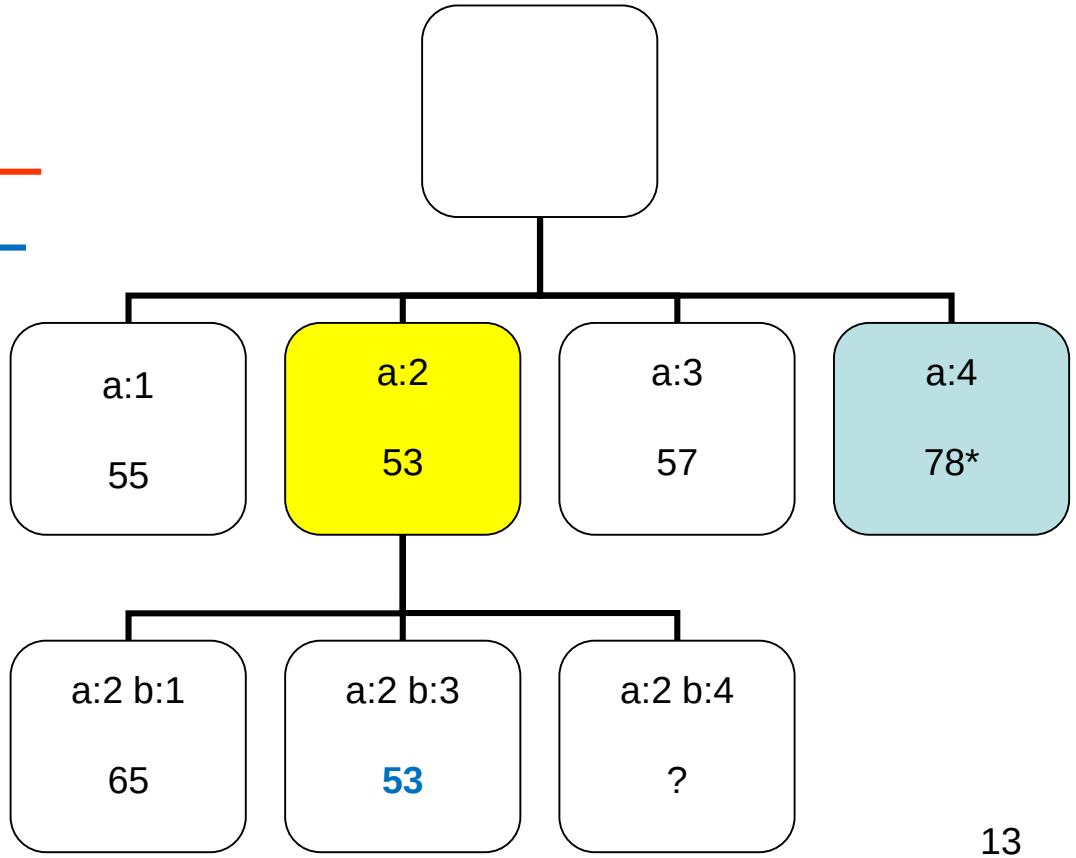


Asignación De Tareas

- Se calcula la cota de la asignación de (a:2), (b:3):

A/T	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

a:2 b:3 12+13+11+17=**53**

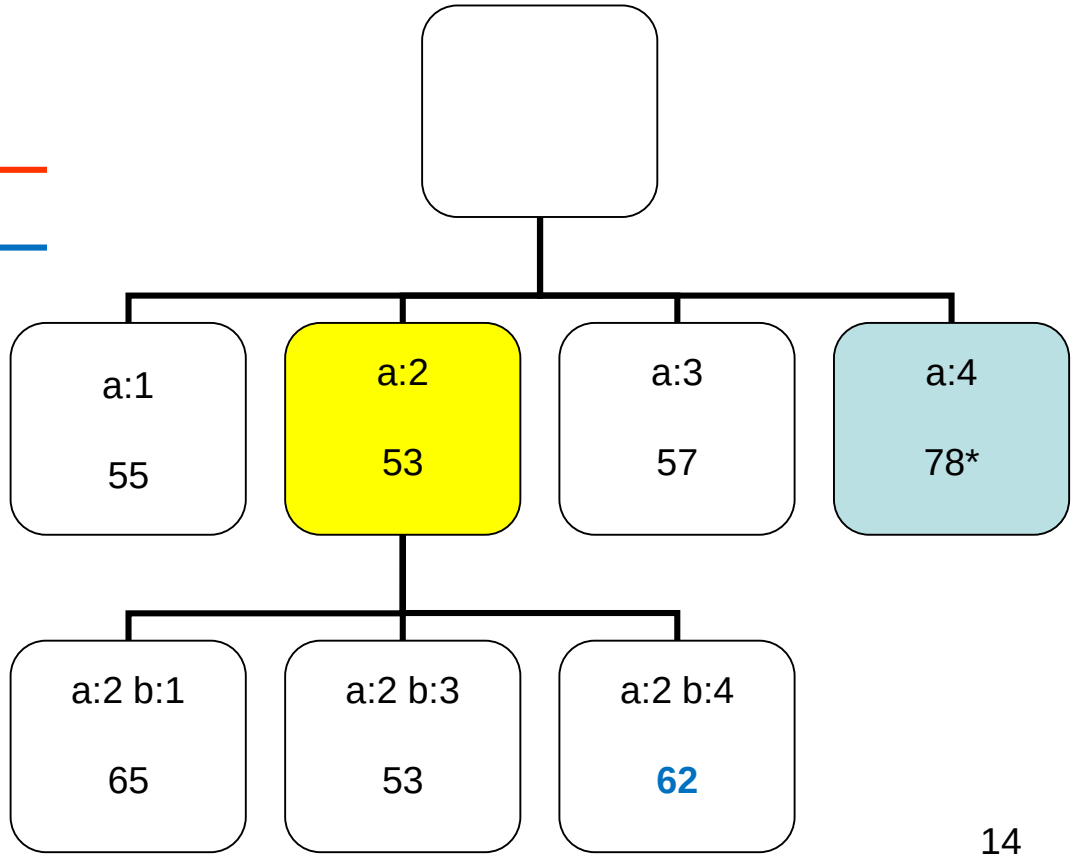


Asignación De Tareas

- Se calcula la cota de la asignación de (a:2), (b:4):

A/T	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

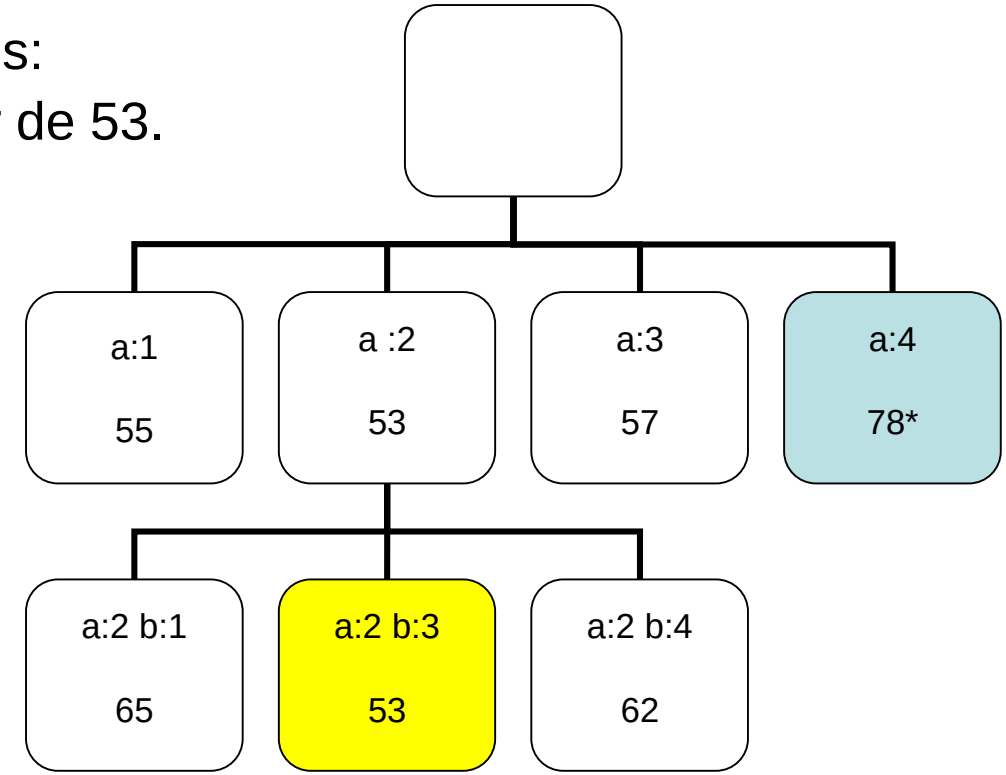
a:2 b:4 12+22+11+17=**62**



Asignación De Tareas

- El nodo más prometedor es:
  (a:2),(b:3) con cota inferior de 53.

A/T	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

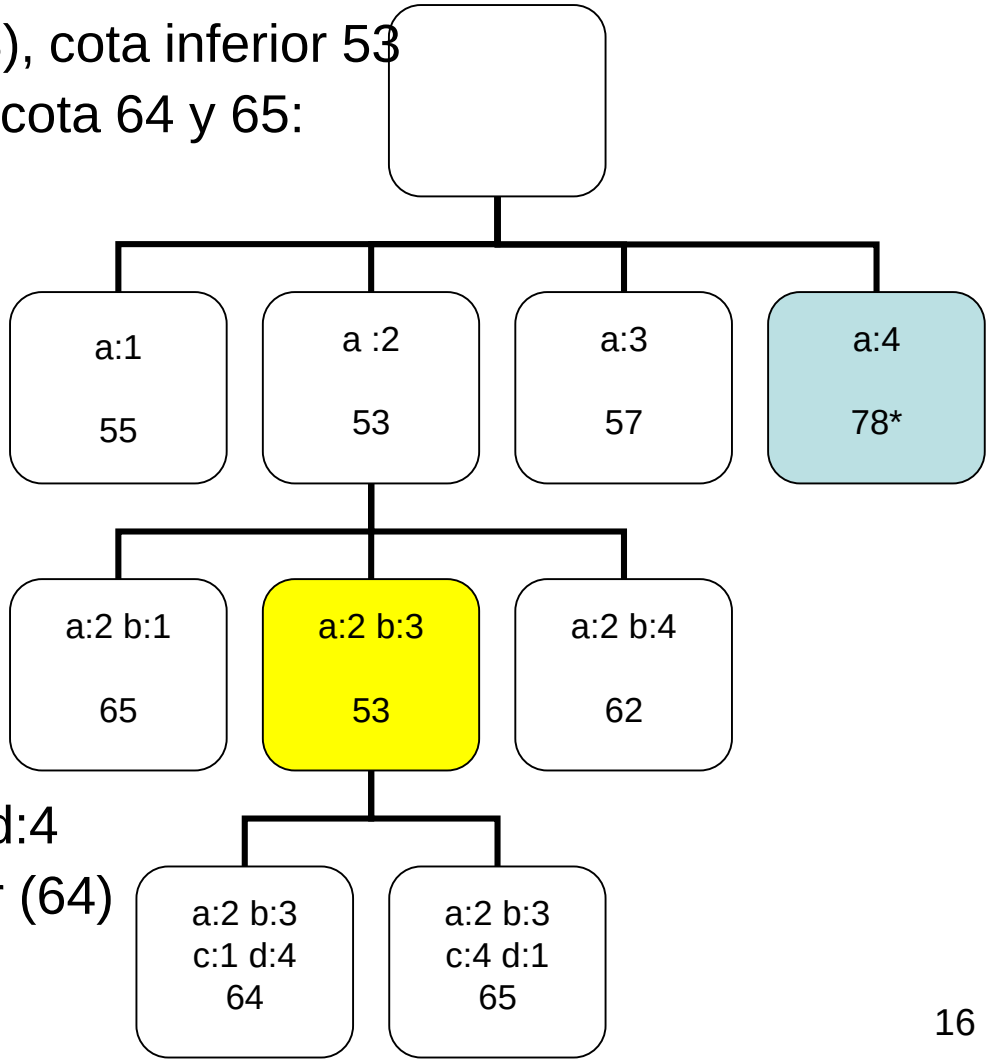


- Se ramifica el nodo de cota 53.

Asignación De Tareas

- Se ramifica el nodo (a:2),(b:3), cota inferior 53
- Se obtienen 2 soluciones de cota 64 y 65:

A/T	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28



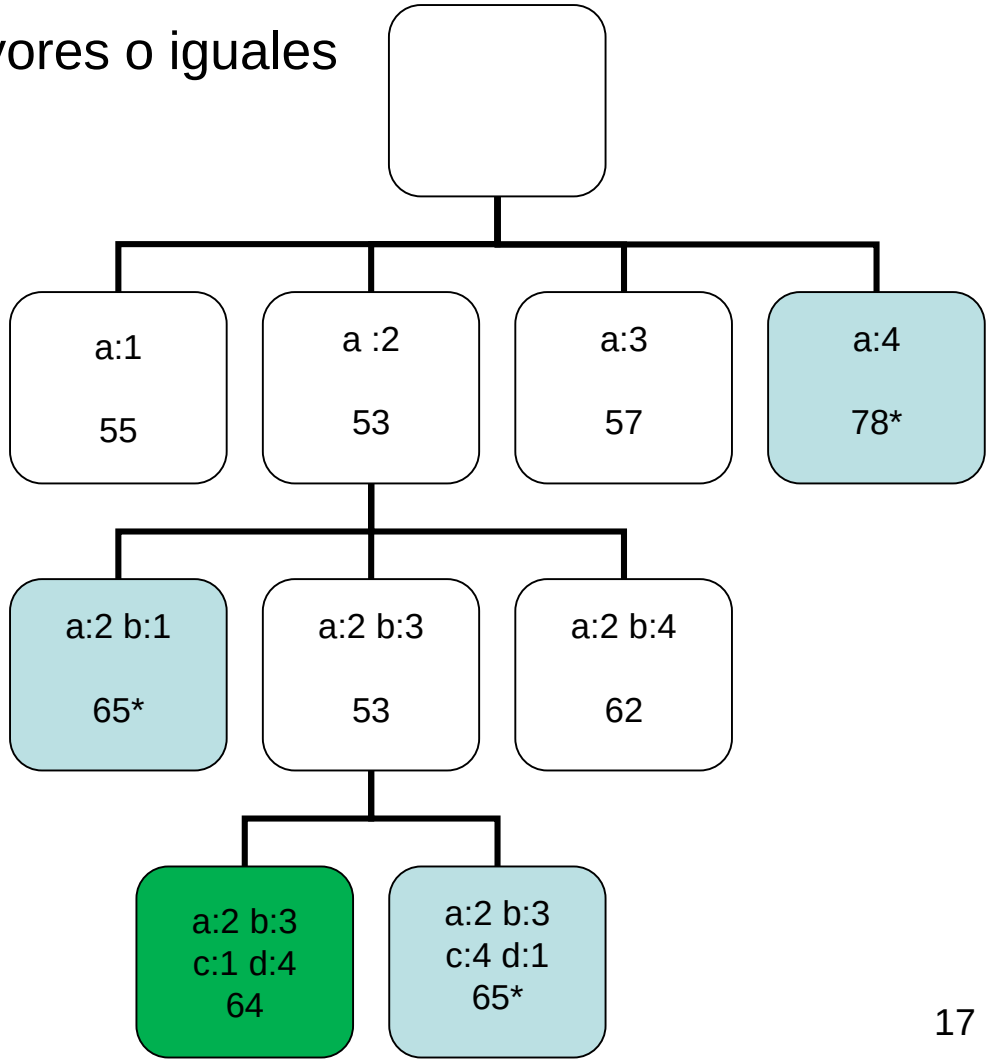
- El nodo solución a:2 b:3 c:1 d:4

es hasta ahora el de mejor valor (64) (nueva cota superior)

Asignación De Tareas

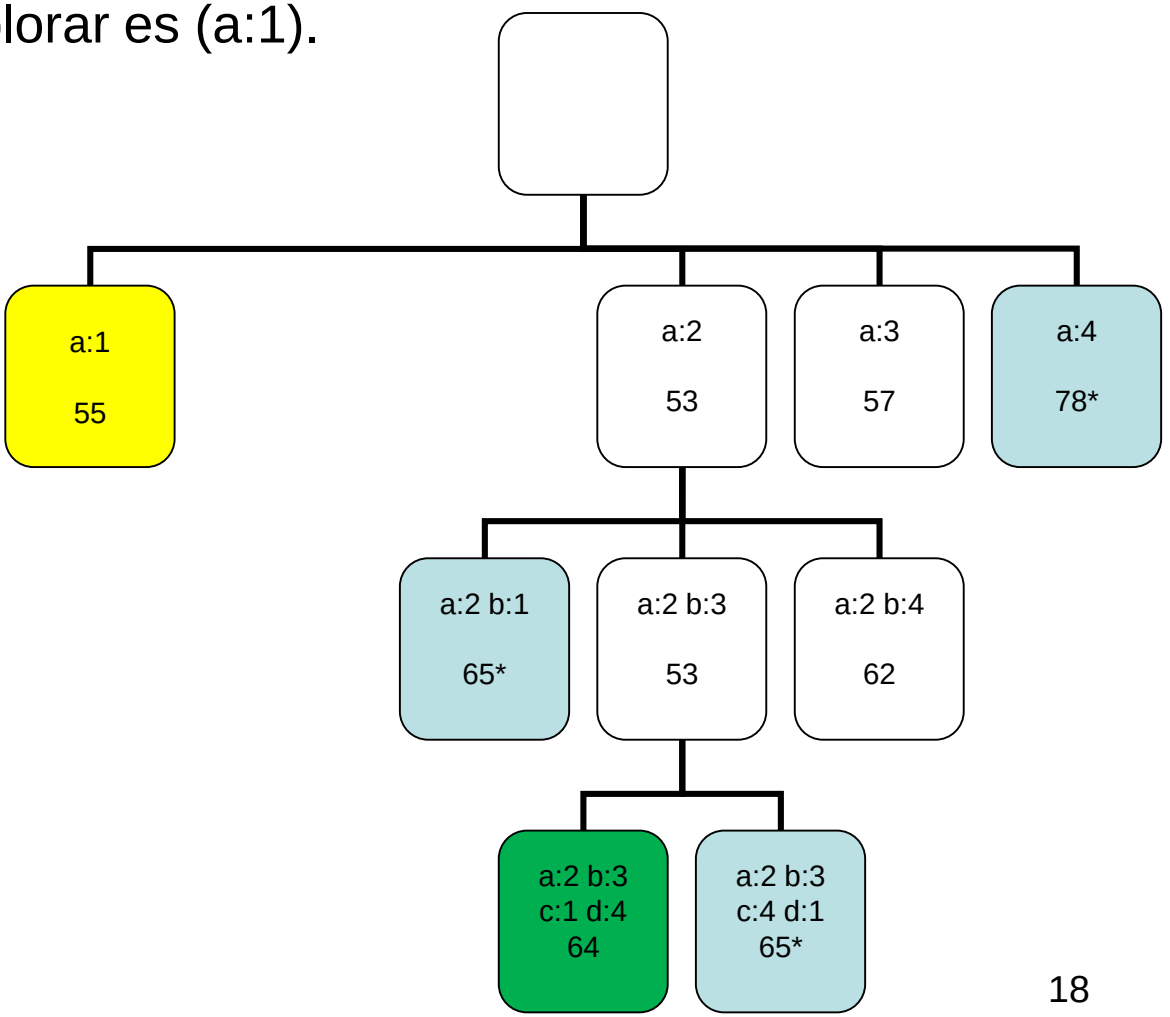
Se podan nodos con cota mayores o iguales a la cota obtenida de 64.

A/T	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28



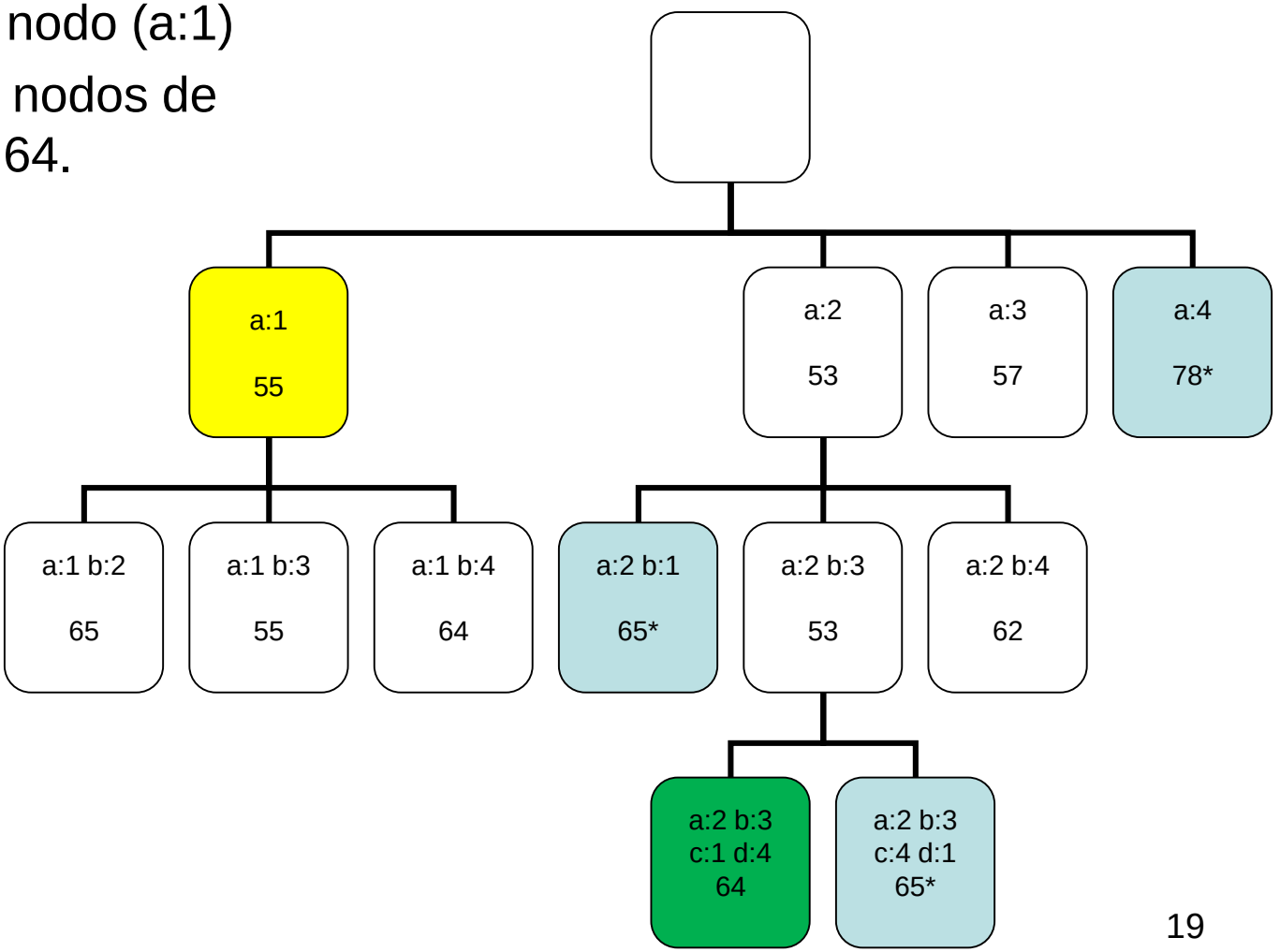
Asignación De Tareas

- El próximo nodo a explorar es (a:1).



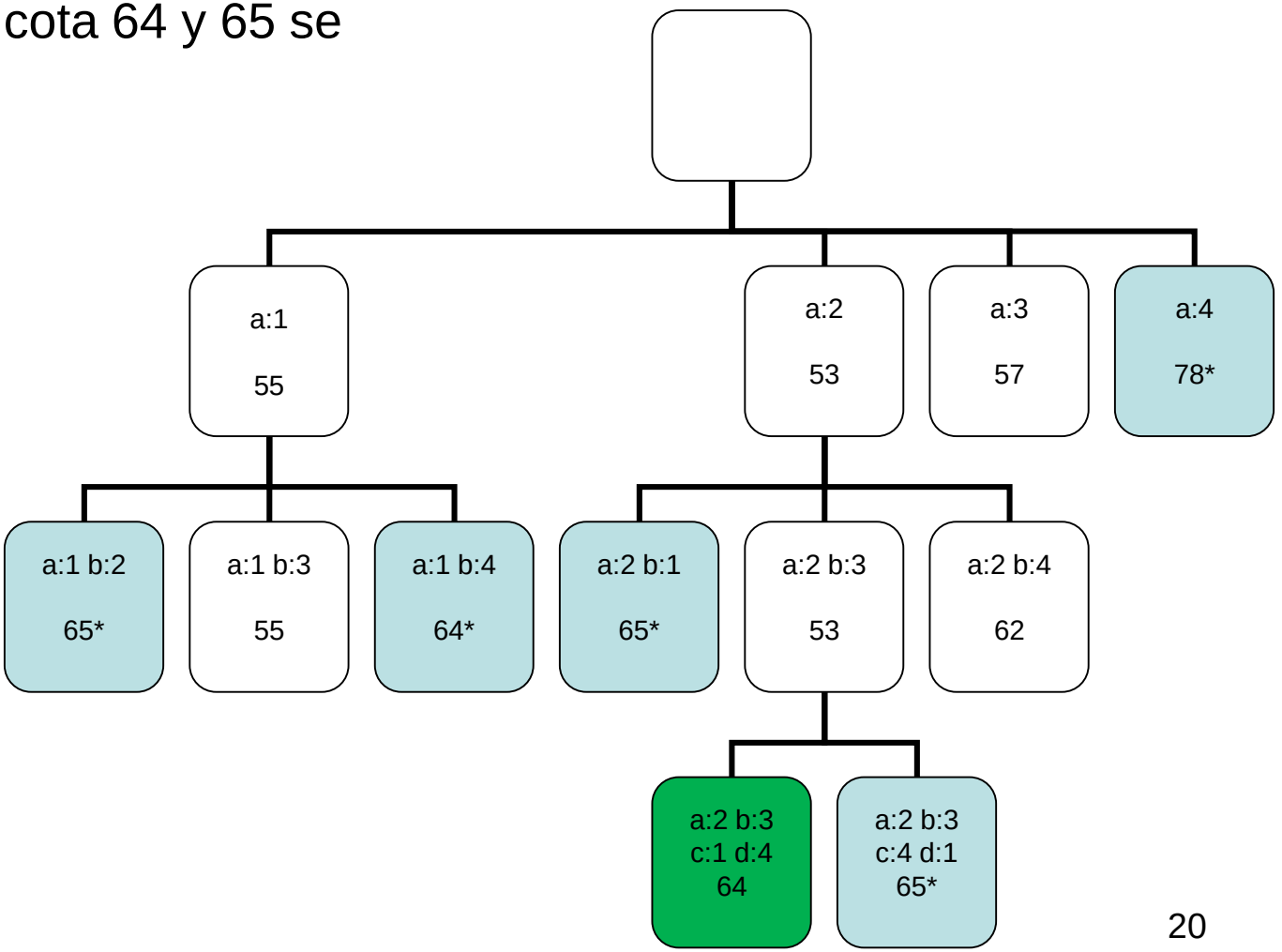
Asignación De Tareas

- Se ramifica el nodo (a:1)
- Se obtienen 3 nodos de cota: 65, 55 y 64.



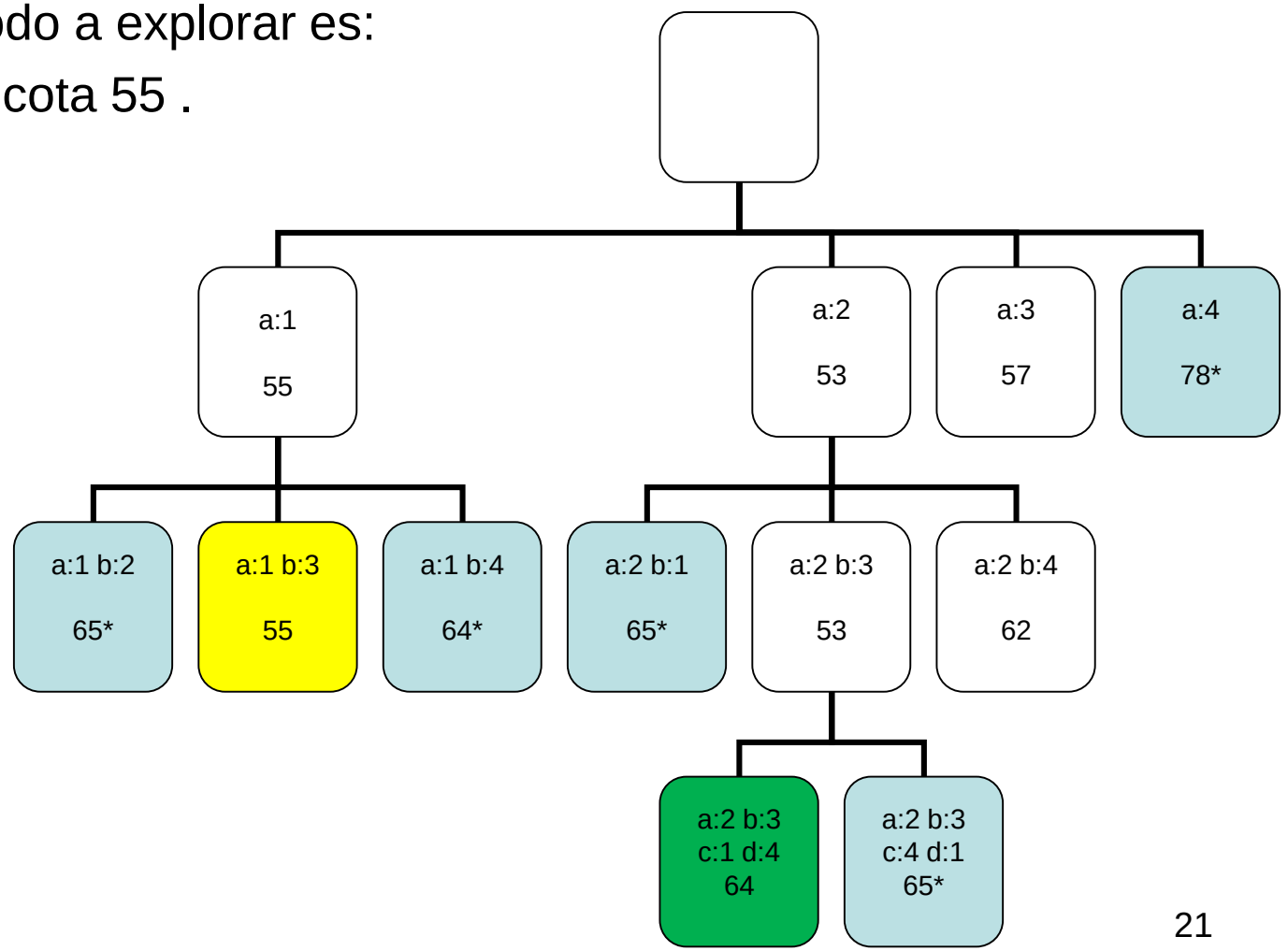
Asignación De Tareas

- Los nodos de cota 64 y 65 se podan.



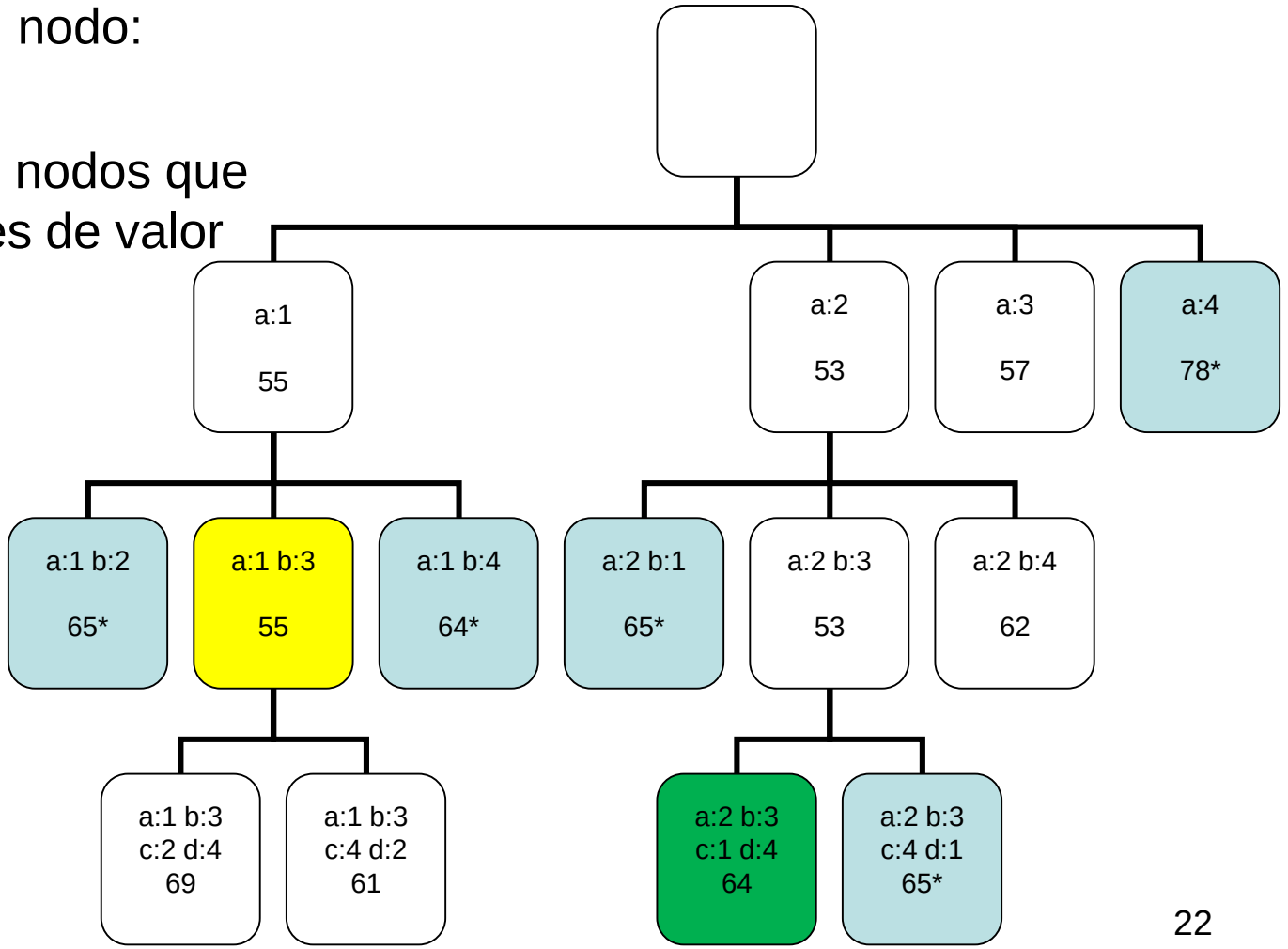
Asignación De Tareas

- El próximo nodo a explorar es:
  (a:1),(b:3) de cota 55 .



Asignación De Tareas

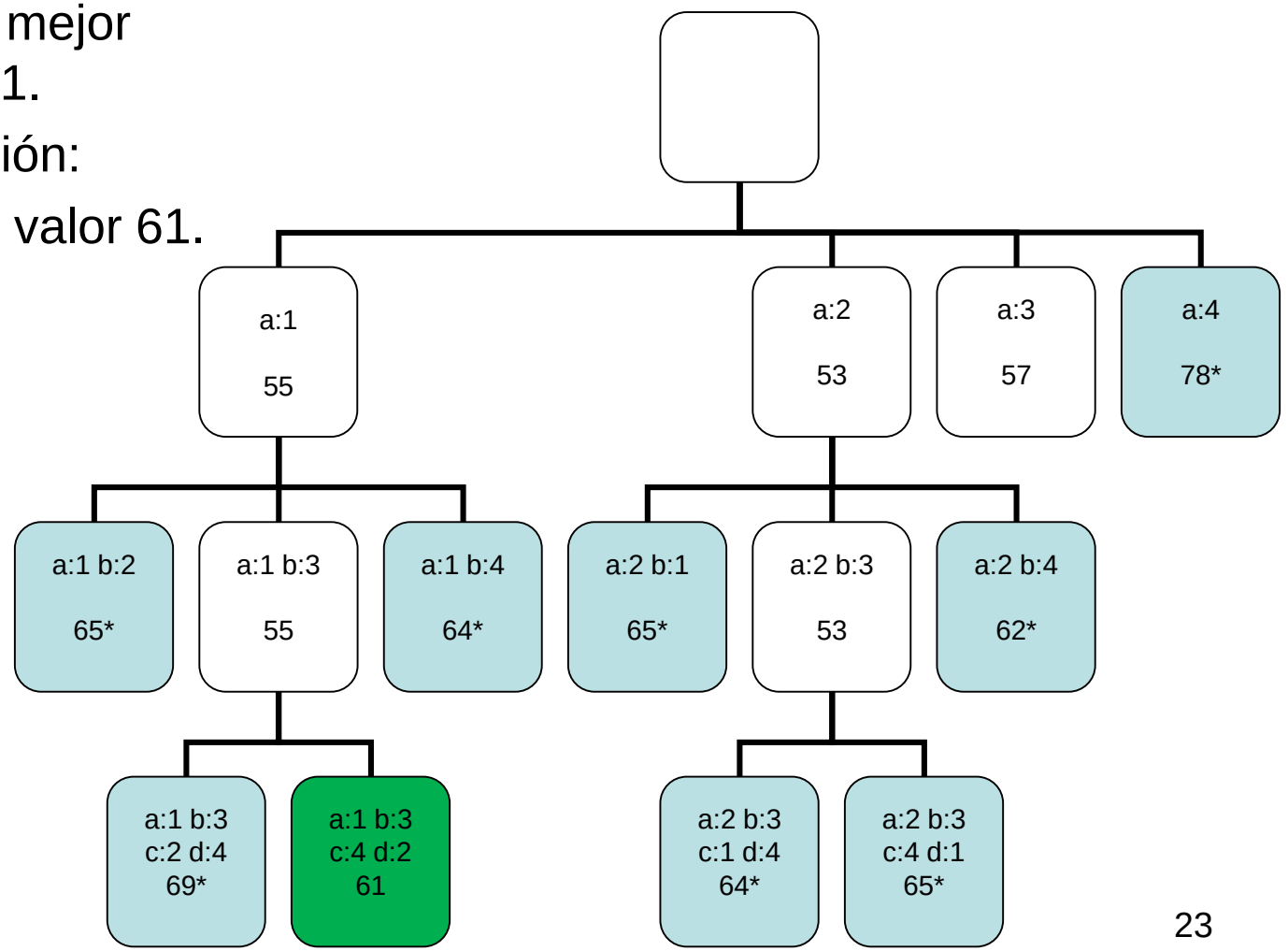
- Se ramifica el nodo:
  (a:1),(b:3).
- Se generan 2 nodos que son soluciones de valor 69 y 61.



Asignación De Tareas

- Se obtiene la mejor cota que es 61.
- El nodo solución:

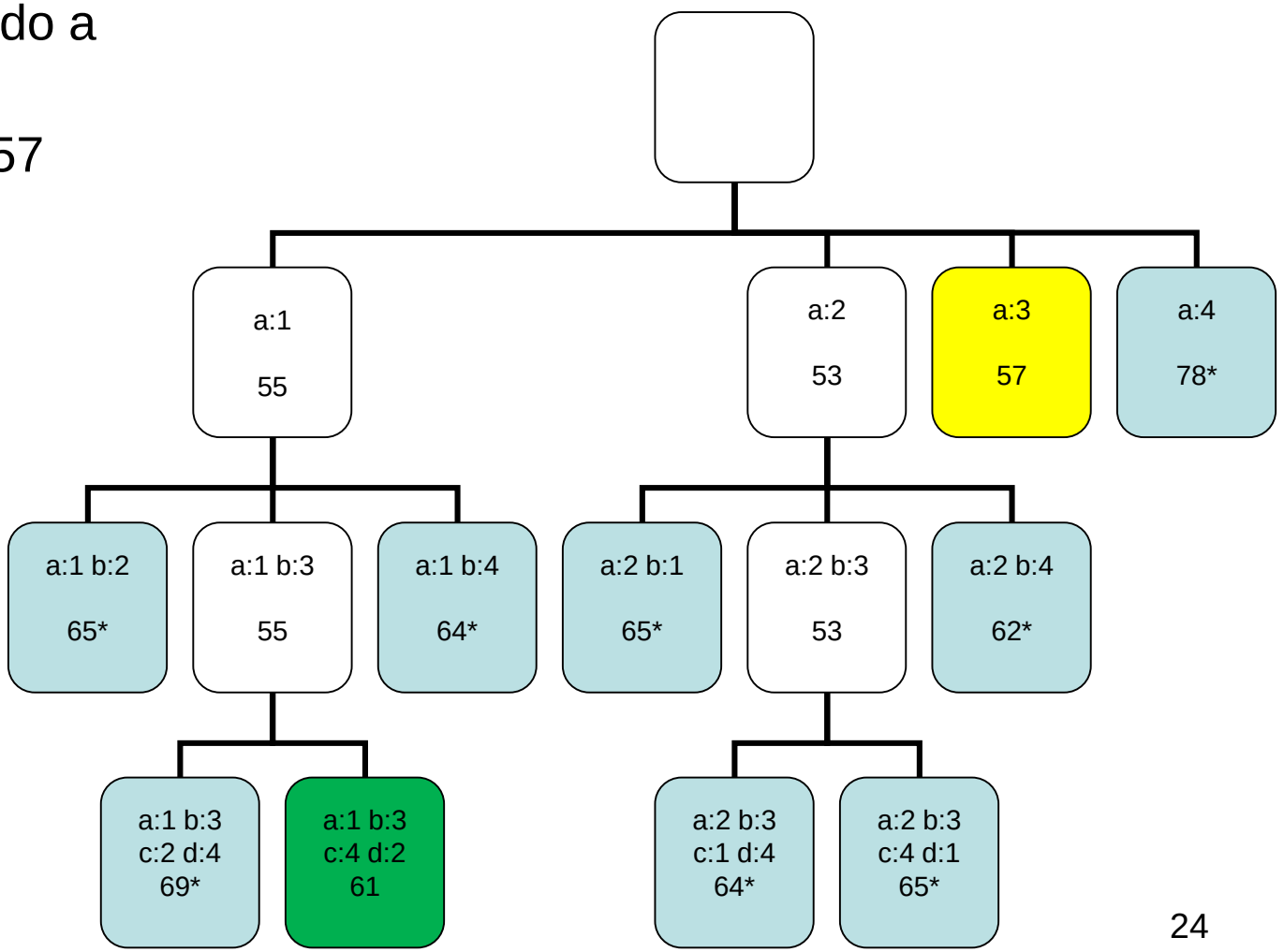
T=(1,3,4,2) tiene valor 61.



Asignación De Tareas

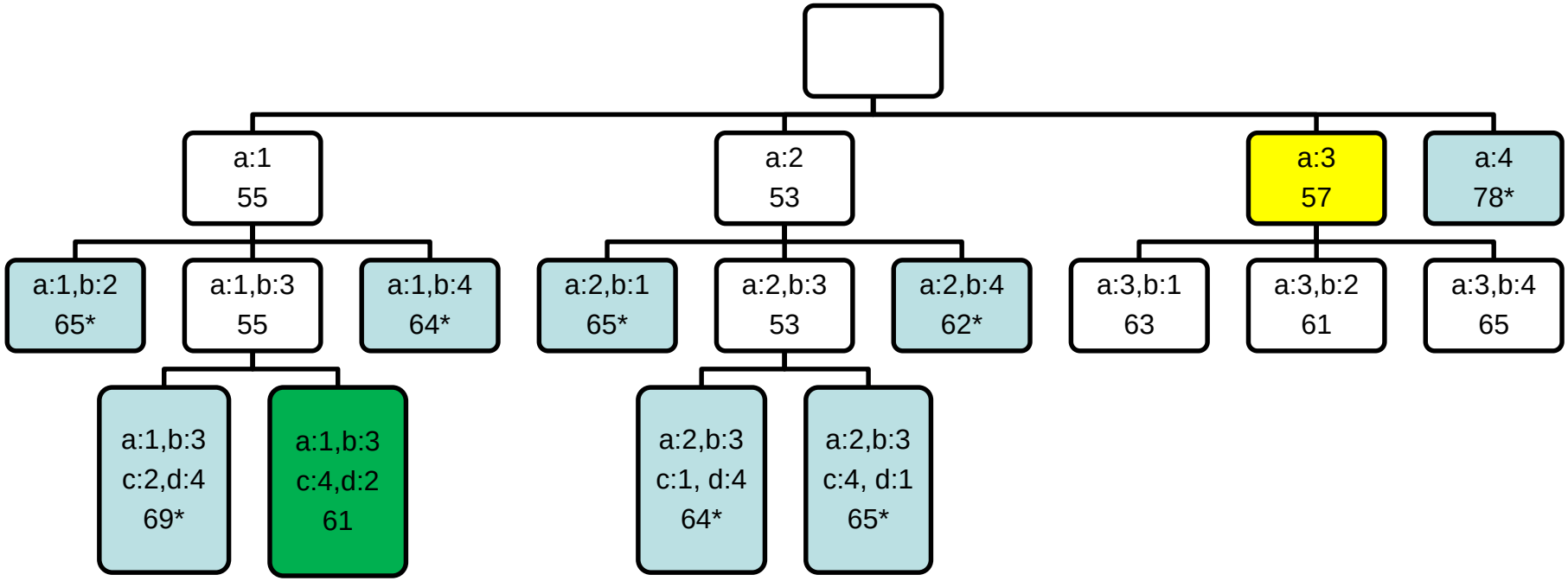
- El próximo nodo a explorar es:

  (a:3) de cota 57



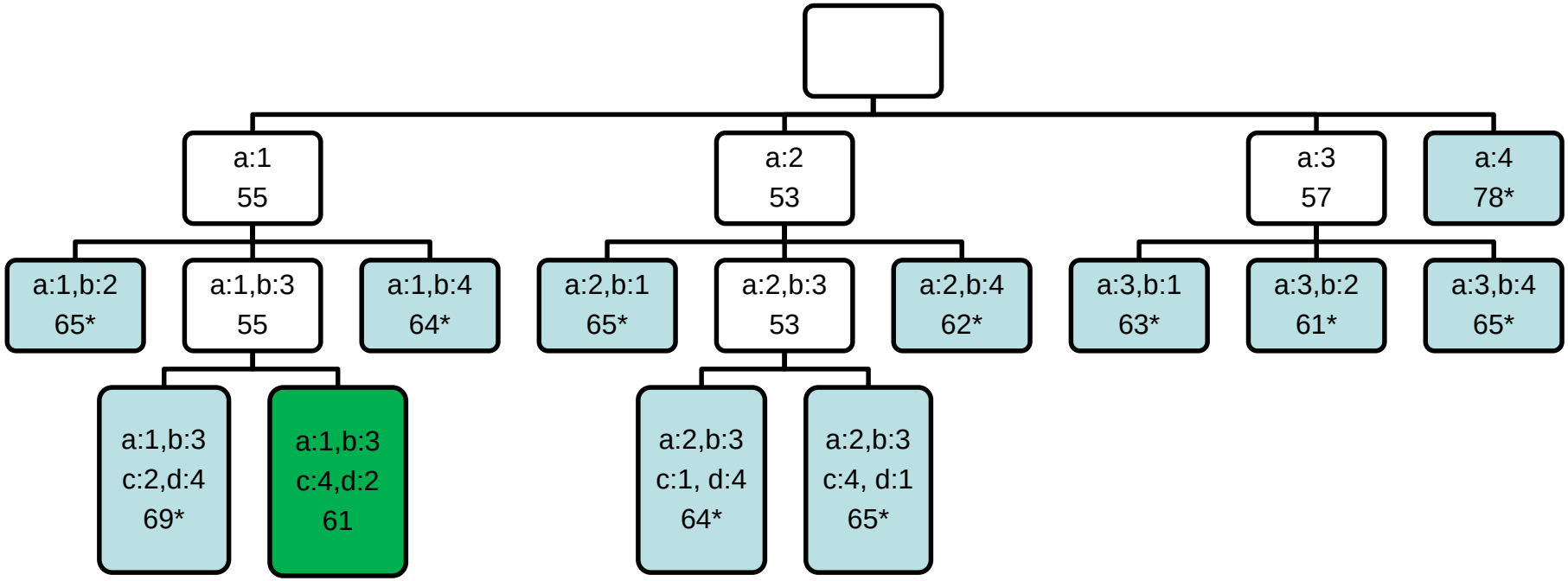
Asignación De Tareas

- El próximo nodo a explorar es: (a:3) de cota 57

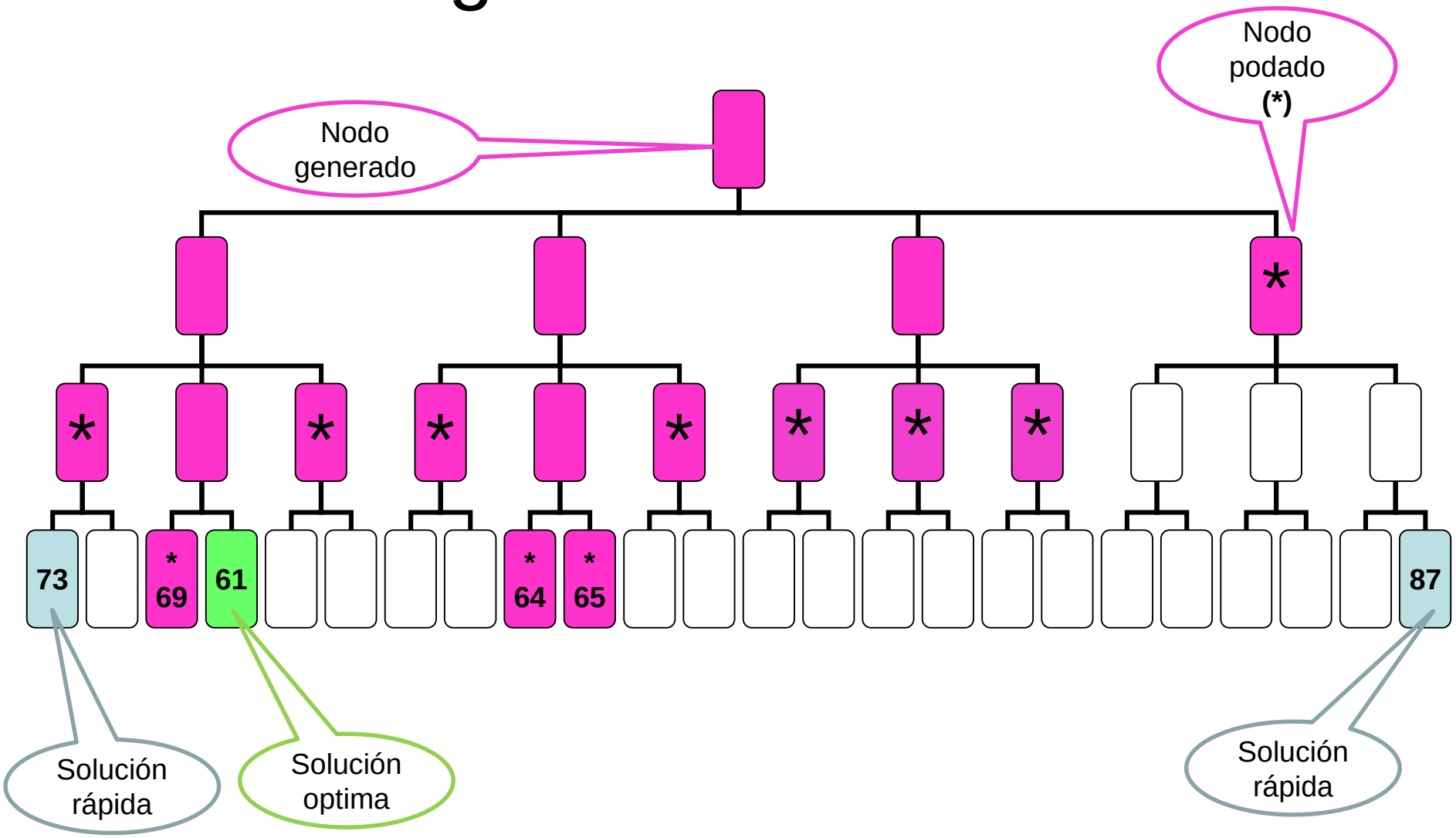


Asignación De Tareas

- Los 3 hijos generados desde el nodo (a:3) se podan.



Asignación De Tareas



ESQUEMA DE UN ALGORITMO DE RAMIFICACION Y PODA:

```
Crear Estructura de nodos vivos e inicializarla con la raíz de árbol
Inicializar cota de poda
Mientras NOT este vacía Estructura hacer
        Extraer de Estructura un nodo vivo N según estrategia elegida
        Si ese nodo N no tiene que ser podado entonces
                Generar todos los hijos de N
                Para j=1 a numero de hijos de N Hacer
                        Si hijo(j) es solución entonces
                                Escribir solución: hijo(j)
                                Actualizar cota de poda
                        Sino
                                Si hijo(j) no debe ser podado entonces
                                        Agregar nodo hijo(j) a Estructura
                                FinSi
                        FinSi
                Fin Para
        FinSi
FinMientras
Fin
```

Algoritmo Ramificación y Poda

Consideraciones de Implementación del algoritmo

Para escribir el algoritmo que resuelve el problema se deben programar las estructuras de datos:

- El adt **Estructura** que almacena los nodos que se van generando con las operaciones: Crear, Agregar, Extraer, EsVacia, Tamano, Destruir.

- El adt **nodo** que permita manejar toda la información asociada a cada nodo con sus operaciones: NodoInicial, Expandir, EsAceptable, EsSolucion, h, Eliminar, NoHaySolucion, Imprimir.

- Variables:

```
numgenerados     // numero total de nodos generados
numanalizados    // numero total de nodos analizados
numpodados       // numero total de nodos podados
```

- Constante:

```
MAXHIJOS = ...          //numero maximo de hijos de un nodo
```

Algoritmo Ramificación y Poda

En el ***tipo abstracto de datos*** **Estructura** que representa la estructura de datos donde se almacenan los nodos tiene operaciones con la siguiente sintaxis:

Crear():→Estructura

Agregar(e, n, prioridad): Estructura x nodo x entero≥0→Estructura

Extraer(e): Estructura → nodo

EsVacia(e): Estructura → BOOLEAN

Tamano(e): Estructura → entero≥0

Destruir(e): Estructura→ Estructura

Algoritmo Ramificación y Poda

- El ***tipo abstracto de datos*** **Estructura** que representa la estructura de datos puede implementarse como una pila, una fila o una cola de prioridad.

- Mediante la utilización del adt Estructura se puede disponer de esta forma fácil y modular de cambiar las estrategias de selección de los nodos vivos.

Esta estrategia de selección con los valores de estos tres parámetros: numgenerados, numanalizados, numpodados, permitirá analizar el algoritmo de Ramificación y Poda de una forma sencilla, cómoda y eficiente, y en consecuencia elegir la mejor de las estrategias para un problema dado.

Algoritmo Ramificación y Poda

El ***tipo abstracto de datos*** **nodo** tiene la siguiente sintaxis en sus operaciones:

NodoInicial(): →nodo *// devuelve el nodo raíz del árbol de expansión*
Expandir(n, hijos): nodo → entero x arreglo de nodo ≥0 ***II*** *(ramificación)*
// construye los nodos hijos de un nodo dado y devuelve también su cant.
EsAceptable(n): nodo → BOOLEAN *// determina si es nodo vivo*
EsSolucion(n): nodo →BOOLEAN
// decide cuándo un nodo es una hoja del árbol, (posible solución al problema)
h(n): nodo → entero≥0 *// función de costo que da la prioridad del nodo*
Valor(n): nodo → entero≥0 *// devuelve el valor asociado a un nodo*
ActualizarCota(c): entero≥0 → entero≥0 *// almacena cota mínima del problema*
Eliminar(n): nodo → nodo *// destruye un nodo*
NoHaySolucion():→nodo *// devuelve nodo con valor especial (indefinido)*
Imprimir(n): nodo → salida *// escribe un nodo*

Algoritmo Ramificación y Poda

Además de encontrar una solución, el algoritmo calcula tres datos:

a) ***numgenerados****:* da información sobre el recorrido total que ha tenido que realizar el algoritmo hasta encontrar la solución. Mientras más pequeño sea este valor, menos parte del árbol de expansión habrá tenido que construir, y por tanto más rápido será el proceso.

b) ***numanalizados****:* indica el número de nodos que el algoritmo ha tenido que analizar, para lo cual es necesario extraerlos de la estructura y comprobar si han de ser podados y, si no, expandirlos. Éste es el valor más importante de los tres, pues indica el número de nodos del árbol de expansión que se recorren efectivamente. En consecuencia, es deseable que este valor sea pequeño.

c) ***numpodados****:* da una indicación de la efectividad de la función de poda y las restricciones impuestas al problema. Mientras mayor sea este valor, más trabajo ahorra el algoritmo.

Algoritmo Ramificación y Poda

El diseño del esquema de programa principal para encontrar ***la primera solución*** en un problema de minimización:

```
//  Variables globales:
        E:Estructura          // estructura para almacenar los nodos
        n:nodo                // nodo del arbol
        numgenerados, numanalizados, numpodados: entero  // contadores

//  programa principal:
numgenerados← 0 ;  numanalizados← 0 ; numpodados← 0
n ← RamiYPoda()
Escribir("Solucion:" , Imprimir(n))
Escribir("Nodos Generados:" , numgenerados)
Escribir ("Nodos Analizados: ", numanalizados)
Escribir ("Nodos Podados: ", numpodados)
Fin.
```

```
Función RamiYPoda(): → nodo // encuentra la primera solucion
variables:
        E:Estructura          // estructura para almacenar los nodos
        n:nodo          // nodo vivo para expandir
        hijos:arreglo (1..MAXHIJOS) de nodo; // hijos de un nodo
        numhijos,i,j: entero⩾0
E←Crear()                     // inicializar la estructura
n←NodoInicial()               // inicializar el nodo
Agregar(E,n,h(n))             // h es la funcion de costo
numgenerados ← 1
Mientras NOT EsVacia(E) hacer
        n←Extraer(E)
        numanalizados← numanalizados+1
        numhijos ← Expandir(n,hijos)
        numgenerados ← numgenerados+numhijos
        Eliminar(n)
        Para i=1 a numhijos hacer
                Si EsAceptable(hijos(i)) entonces   // nodo vivo
                        Si EsSolucion(hijos(i)) entonces      // Eureka!
                                Para j=1 a numhijos Hacer
                                        Si i≠j  entonces     //eliminar resto de hijos
                                                Eliminar(hijos(j))
                                Destruir(E)
                                Retorna hijos(i)     // devolver la solucion
                        Sino
                                Agregar(E,hijos(i),h(hijos(i)))
                Sino  // se poda
                        Eliminar(hijos(i))
                        numpodados←numpodados+1
Destruir(E)
Retorna NoHaySolucion()
Fin
```

Algoritmo Ramificación y Poda

- Si lo que se quiere es encontrar no sólo una solución al problema sino **todas las soluciones**, es posible conseguirlo con una pequeña variación del esquema anterior:

Función RamiYPodaTodas(todas)

```
Función RamiYPodaTodas(todas): → Arreglo de nodo x entero⩾0
// encuentra todas las soluc. del problema y  retorna el nro. de soluc.
variables:
        E:Estructura // estructura para almacenar los nodos
        n:nodo       // nodo vivo para expandir
        hijos, todas:arreglo (1..MAXHIJOS) de nodo
        numhijos,i,j,numsol: entero⩾0

E←Crear()                   // inicializar la estructura
n←NodoInicial()             // inicializar el nodo
Agregar(E,n,h(n))           // h es la funcion de costo
numgenerados ← 1
numsol ←0
Mientras NOT EsVacia(E) hacer        // analizar todo el arbol
        n←Extraer(E)
        numanalizados← numanalizados+1
        numhijos ← Expandir(n,hijos)
        numgenerados ← numgenerados+numhijos
        Eliminar(n)
        Para i=1 a numhijos hacer
                Si EsAceptable(hijos(i)) entonces
                        Si EsSolucion(hijos(i)) entonces
                                numsol←numsol+1          // 1 soluc mas
                                todas(numsol) ←hijos(i)
                                Eliminar(hijos(i))
                        Sino
                                Agregar(E,hijos(i),h(hijos(i)))
                Sino
                        Eliminar(hijos(i))
                        numpodados ←numpodados+1
Destruir(E)
Retorna numsol
Fin
```

Algoritmo Ramificación y Poda

- Si lo que se quiere es encontrar no sólo una solución al problema sino **la mejor solución**, es posible conseguirlo con otra variante del esquema anterior que considera el valor de la mejor solución se presenta:

Función RamiYPodaOptima(): → nodo

```
Función RamiYPodaOptima(): → nodo // busca la solución entera de menor valor

E←Crear()                    // inicializar estructura
n←NodoInicial()              // inicializar el nodo
Agregar(E,n,h(n))            // h es la funcion de costo
numgenerados ← 1
solucion←NoHaySolucion()
valor_solucion←Max(entero)
ActualizarCota(valor_solucion)
Mientras NOT EsVacia(E) hacer
        n←Extraer(E)
        numanalizados← numanalizados+1
        Si EsAceptable(n) entonces
                numhijos ← Expandir(n,hijos)
                numgenerados ← numgenerados+numhijos
                Eliminar(n)
                Para i=1 a numhijos hacer
                        Si EsAceptable(hijos(i)) entonces
                                Si EsSolucion(hijos(i)) entonces  //mejor
                                        valor ←Valor(hijos(i) )
                                        Si valor<valor_solucion entonces
                                                Eliminar(solucion)
                                                solucion ← hijos(i)
                                                valor_solucion ← valor
                                                ActualizarCota(valor_solucion)
                                                Eliminar(hijos(i))
                                Sino
                                        Agregar(E,hijos(i),h(hijos(i)))
                        Sino
                                Eliminar(hijos(i))
                                numpodados ←numpodados+1
        Sino
                Eliminar(n)
                numpodados ←numpodados+1
Destruir(E)
Retorna solucion
Fin
```

Variables
E:Estructura // estructura para almacenar los nodos
n,solucion:nodo // nodo vivo para expandir
hijos:arreglo (1..MAXHIJOS) de nodo // hijos de un nodo
numhijos,i,j, valor,valor_solucion: entero≥0

Problema de la mochila 0/1

Ya se desarrolló una solución con programación dinámica y con vuelta atrás para este caso.

Se verá a continuación una solución con Ramificación y Poda.

Datos: Se tienen n objetos y una mochila para llevarlos.

Cada objeto tiene un peso: p_i y un beneficio asociado: b_i.

La mochila puede cargar un peso máximo dado: M.

Objetivo: *llenar la mochila de tal manera que se* ***maximice el beneficio*** *de los objetos transportados, respetando la limitación de la capacidad impuesta.*

Maximizar la cantidad: $\sum_{i=1}^{n} b_i x_i$

Restricción: $\sum_{i=1}^{n} p_i x_i \leq M$

Solución: Vector $\boldsymbol{X}$ de n componentes

Si $x_i = 0$ si el objeto i no va en la mochila

Si $x_i = 1$ si el objeto i si va en la mochila

Problema de la mochila 0/1

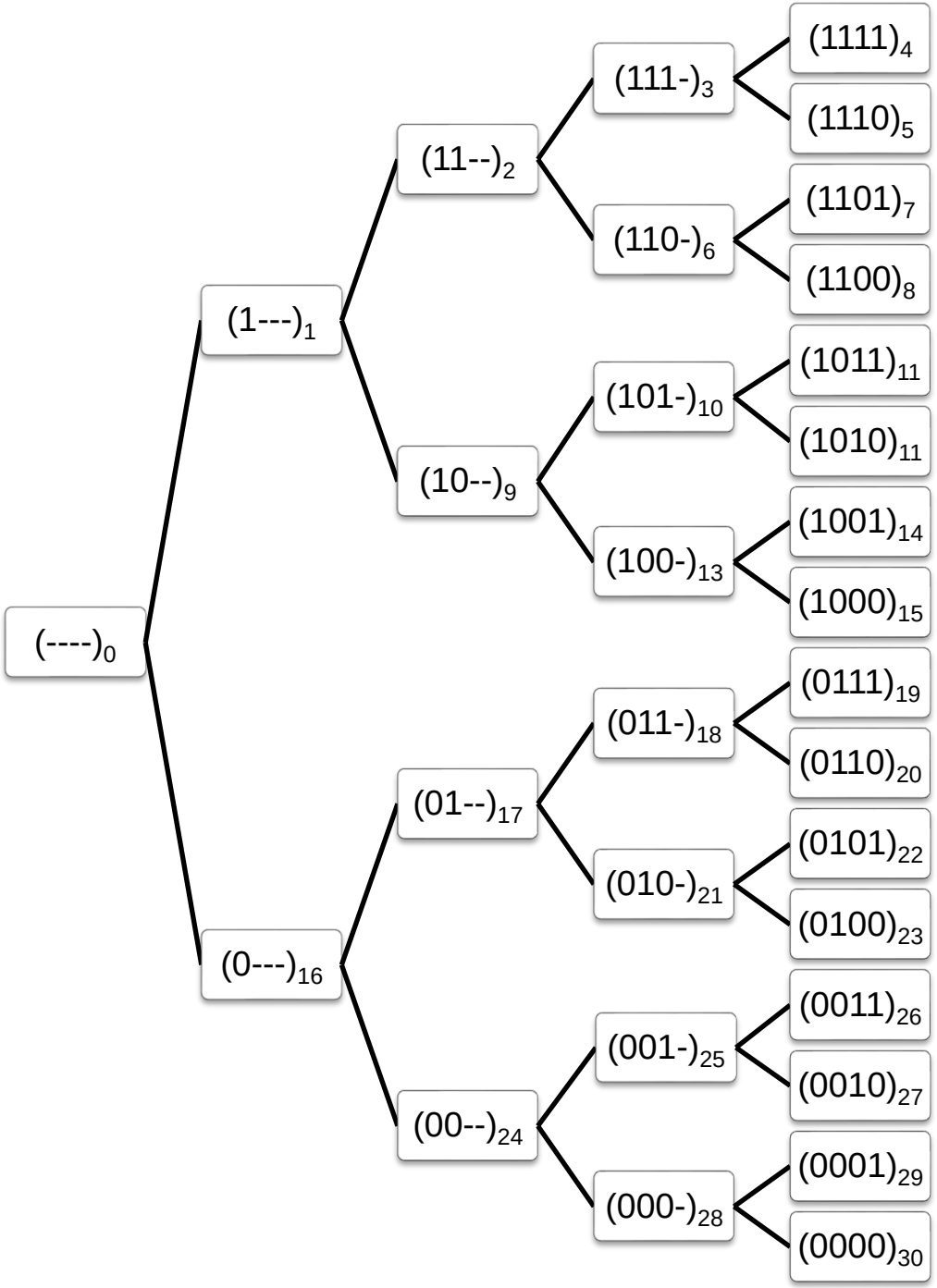
- Para construir el árbol de expansión del problema es necesario plantear la solución como una secuencia de decisiones, una en cada etapa o nivel del árbol.
- Representar la solución del problema mediante un vector solución X, en donde en cada posición podrá encontrarse uno de los valores 1 ó 0, indicando si se carga o no el elemento.
- Existen 2^n modos de asignar los valores 0 o 1 al vector solución X.
- Comenzando por el primer elemento, se irá recorriéndolos todos y decidiendo en cada paso si se incluye o no ese elemento, por lo tanto cada nodo va a dar lugar a lo sumo a dos hijos.
- Se puede suponer que los elementos tienen un preproceso de modo de que *están ordenados de forma decreciente* del cociente bi/pi.
- Respecto a la poda, es importante notar que éste es un problema de *maximización*.
- Hay que considerar hasta donde expandir y una buena función de **poda**.

Soluciones problema mochila 0/1

Ejemplo:

Para n=4
2^4=16
posibles soluciones.

Usando Estructura Pila sin poda:
ab de 31 nodos→



Problema de la mochila 0/1

Consideraciones de implementación:

Definir la estructura de datos : nodo. En ellos hay que almacenar información sobre la solución alcanzada hasta ese nodo:

Constante n = // numero de elementos distintos
solucion: arreglo(1..n) de enteros
nodo : con los siguientes campos:
    peso // peso acumulado
    beneficio // beneficio acumulado
    k: entero // nivel
    s: solucion // solución hasta el momento

variables globales:
cota: entero // cota superior alcanzada hasta el momento
capacidad: entero // M, capacidad máxima de carga de la mochila
tabla: arreglo (1..n) de registros con 2 campos: (beneficio,peso) //datos iniciales

Problema de la mochila 0/1

- La función *NodoInicial* debe generar un nodo vacío inicialmente, con *peso* y *beneficio* y *nivel*, todos en cero.

- La estrategia de ramificación está a cargo de la función *Expandir*. Cada nodo puede generar a lo sumo dos hijos, que corresponden a incluir el elemento o no en la mochila. Sólo serán generados aquellos nodos que sean válidos, esto es, si caben en la mochila, teniendo en cuenta la capacidad ya utilizada hasta el momento.

- Para implementar la función que realiza la *poda*, se puede suponer sin pérdida de generalidad que los objetos están numerados en manera decreciente de los cocientes beneficio/peso:

$$(b_i/p_i) \geq (b_{i+1}/p_{i+1}) \qquad 1 \leq i \leq n-1$$

Problema de la mochila 0/1

Si los valores: $x_1, x_2, \ldots x_k$, $0 \leq k \leq n$ se cargan en la mochila, con: $\sum_{i=1}^{k} p_i x_i \leq M$

El beneficio acumulado hasta el elemento k es: $B_k = \sum_{i=1}^{k} b_i x_i$

El peso acumulado hasta el elemento k es: $P_k = \sum_{i=1}^{k} p_i x_i$

El beneficio mejor que se puede alcanzar cargando el próximo elemento ($k+1$) no puede sobrepasar el valor:

$$B_M = B_k + (M - P_k)\frac{b_{k+1}}{p_{k+1}}$$

B_M representa la *cota superior* del valor que se puede alcanzar por esa rama del árbol.

Esta fórmula se usa para implementar la función h:

- si el nodo n es solución retorna su beneficio: $h(n) = B_k$
- si n no es solución (su beneficio + el beneficio del próximo): $h(n) = B_M$

Problema de la mochila 0/1

Funcion **h(n)** : nodo → entero≥0

// función de costo

Si EsSolucion(n) entonces

retorna B_k

Sino

retorna B_M

Como este es un problema de optimización para obtener un ***máximo*** beneficio, se va a minimizar MAXINT – h(n). La función valor se define por:

Funcion **Valor(n)** : nodo → entero≥0

// devuelve el valor asociado a un nodo

retorna Max(entero) – h(n)

Problema de la mochila 0/1

En lo que respecta a la estrategia de poda, hay que mantener actualizada una variable global de poda Cota. Cada vez que se guarda un nodo en la lista de nodos o se encuentra una solución, es necesario actualizar la cota con la función ActualizarCota. Luego se usa esa cota para decidir si se poda o no un nodo.

Funcion **EsAceptable(n)** : nodo → BOOLEAN

// decide si es nodo vivo

retorna Valor(n) ≤ Cota

En cada nodo se considera solamente los sucesores que satisfagan la restricción de peso. Cuando se genera un nodo se calcula el valor de la cota para el beneficio de la solución y se usa esa cota para podar aquellos nodos que superen la misma.

Problema de la mochila 0/1

Otra de las funciones que es necesario implementar es la que determina cuándo un nodo es *solución*. En este caso es muy simple, consiste en decidir cuándo se ha llegado a tratar hasta el *n*-ésimo elemento.

En cuanto a la función *NoHaySolucion*, que devuelve un valor especial para indicar que el problema no admite solución, se sabe que para este problema eso no ocurrirá nunca, porque siempre existe al menos una solución, que es la que representa el vector $X=(0,0,...,0)$, es decir, siempre se puede no incluir ningún elemento. Ésta es, por ejemplo, la solución a un problema en donde todos los pesos de los elementos superen la capacidad de la mochila.

Por su parte, la función *Eliminar* es la que va a devolver al sistema los recursos ocupados por un nodo, y es la que actúa como “destructor” del tipo abstracto de datos nodo.

Problema de la mochila 0/1

Se construye el nodo inicial X_0

Ejemplo:
n=3
M=3
p=(1,2,3)
b=(2,3,4)
b/p=(2, 1.5, 1.33)

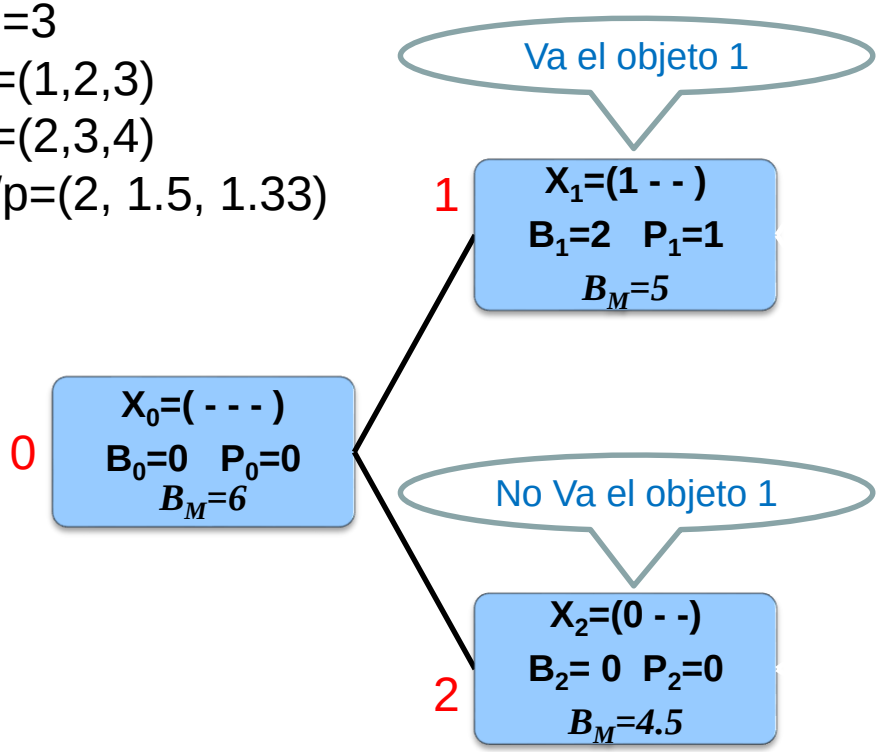
0 **X_0=(- - -)**
B_0=0 P_0=0
B_M=6

$$B_M = B_k + (M - P_k)\frac{b_{k+1}}{p_{k+1}}$$

Problema de la mochila 0/1

Ejemplo:
n=3
M=3
p=(1,2,3)
b=(2,3,4)
b/p=(2, 1.5, 1.33)

Se expande el nodo inicial 0

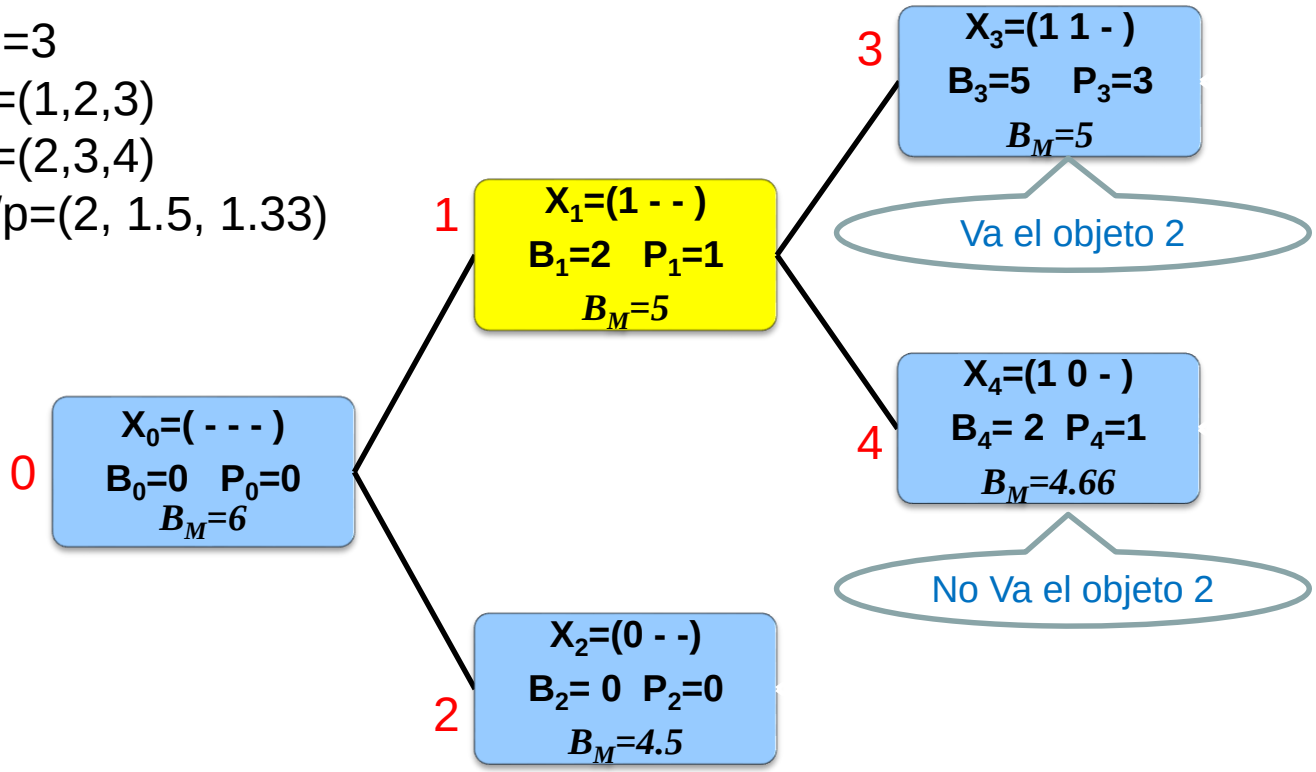


$$B_M = B_k + (M - P_k)\frac{b_{k+1}}{p_{k+1}}$$

Problema de la mochila 0/1

Ejemplo:
n=3
M=3
p=(1,2,3)
b=(2,3,4)
b/p=(2, 1.5, 1.33)

Se expande el nodo mas prometedor, el nodo 1

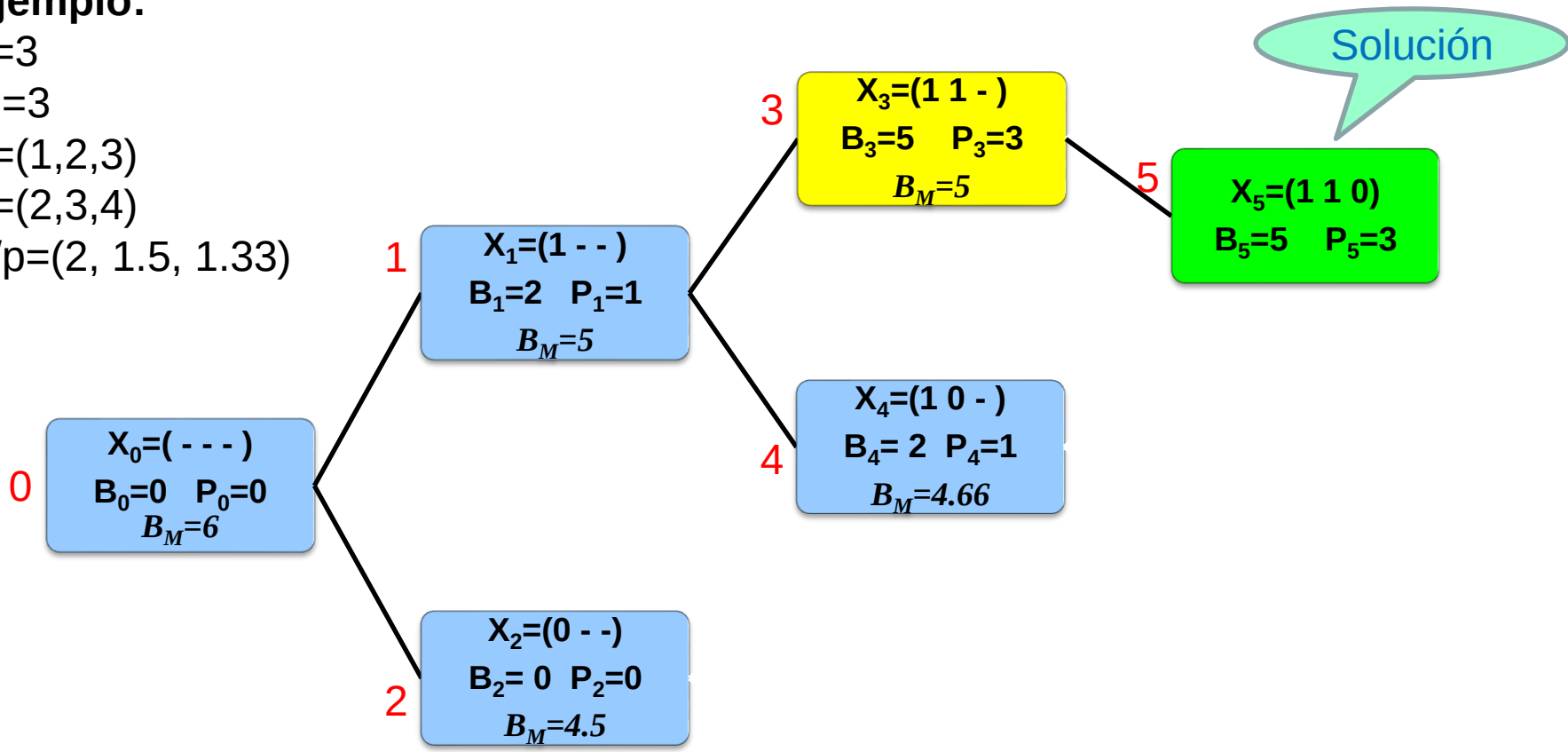


$$B_M = B_k + (M - P_k)\frac{b_{k+1}}{p_{k+1}}$$

Problema de la mochila 0/1

Ejemplo:
n=3
M=3
p=(1,2,3)
b=(2,3,4)
b/p=(2, 1.5, 1.33)

Se expande el nodo mas prometedor , el nodo 3

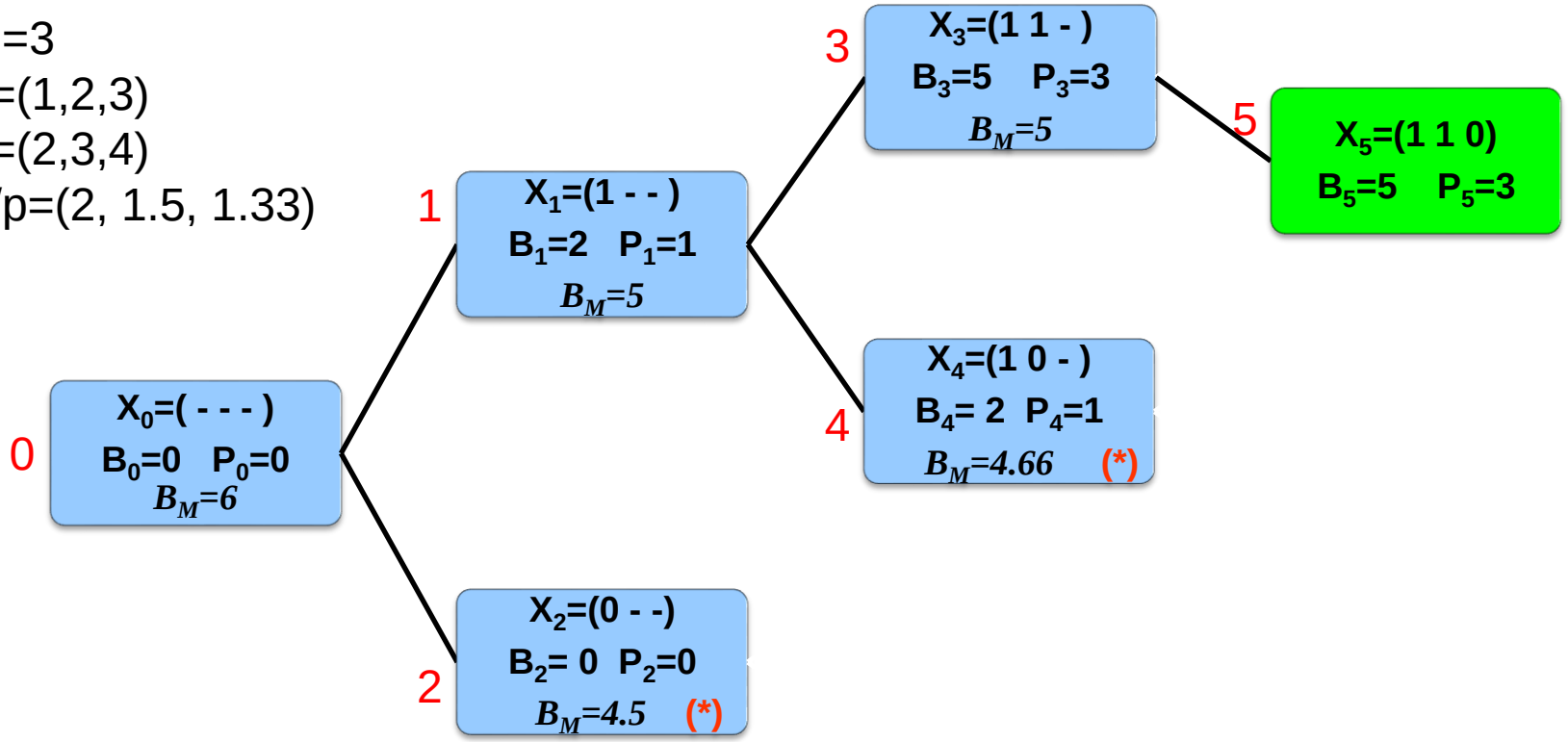


$$B_M = B_k + (M - P_k)\frac{b_{k+1}}{p_{k+1}}$$

Problema de la mochila 0/1

Ejemplo:
n=3
M=3
p=(1,2,3)
b=(2,3,4)
b/p=(2, 1.5, 1.33)

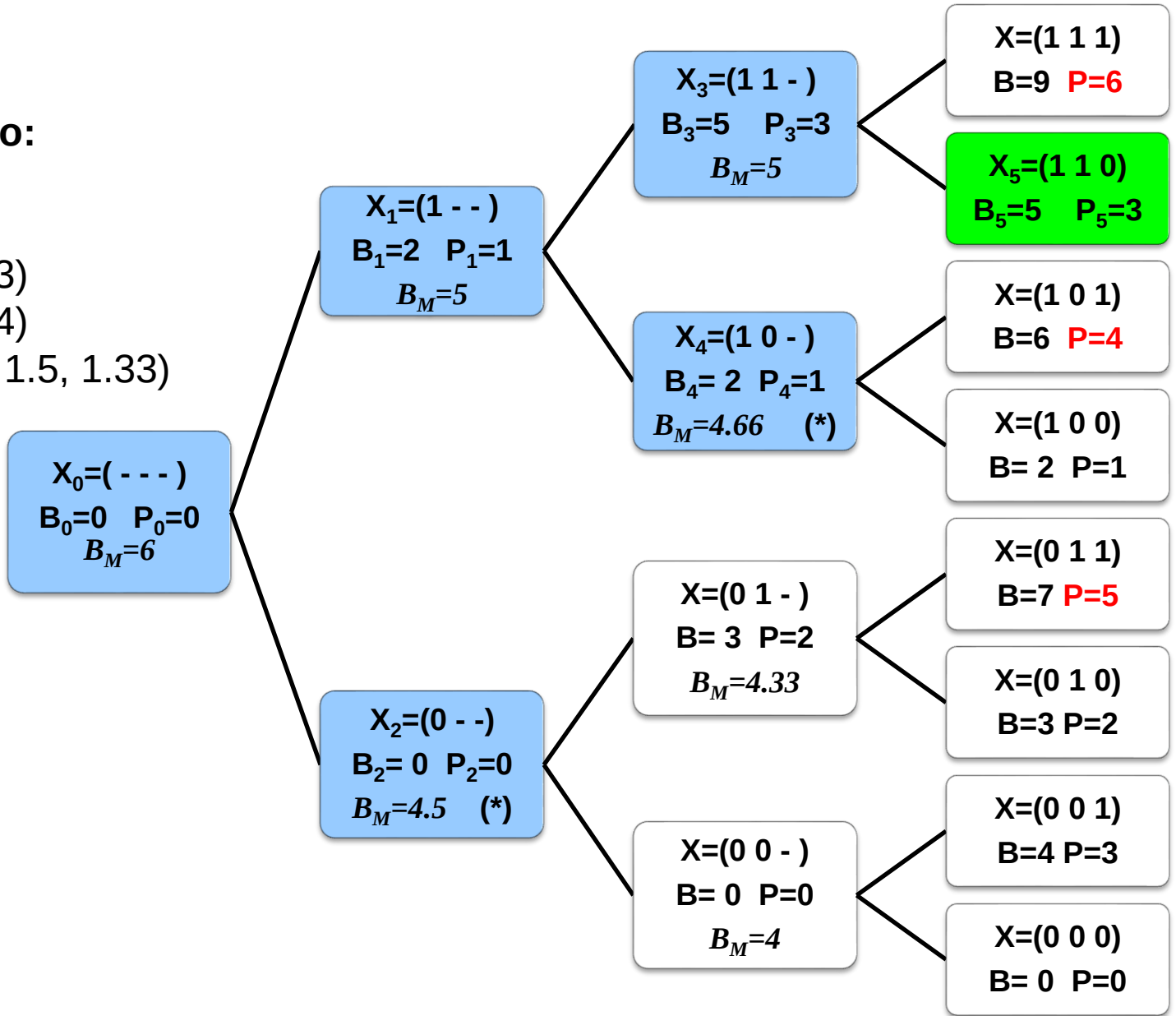
Con el valor de la solución obtenida X_5 se podan 2 y 4.
No hay mas nodos para expandir.



(*) Importante: los nodos 2 y 4 se podan después de obtener la solución X_5

Problema de la mochila 0/1 con las 8 soluciones posibles:

Ejemplo:
n=3
M=3
p=(1,2,3)
b=(2,3,4)
b/p=(2, 1.5, 1.33)



Problema de la mochila múltiple

Este problema está muy relacionado con el anterior y va a presentar muy pocas diferencias.

- En primer lugar, la solución va a seguir estando representada por un vector, pero esta vez no será de ceros y unos, sino que podrá tomar valores enteros positivos.

Este primer cambio no se ve reflejado en el algoritmo del problema anterior, porque el tipo *nodo* ya permitía almacenar valores positivos mayores que uno.

- En segundo lugar, cada nodo no generará a lo sumo dos hijos, sino que podrá generar **varios**, tantos como le permita la capacidad de la mochila.

Este segundo cambio tiene su reflejo en la función que expande los nodos.

- Las demás funciones del adt Nodo quedan igual.

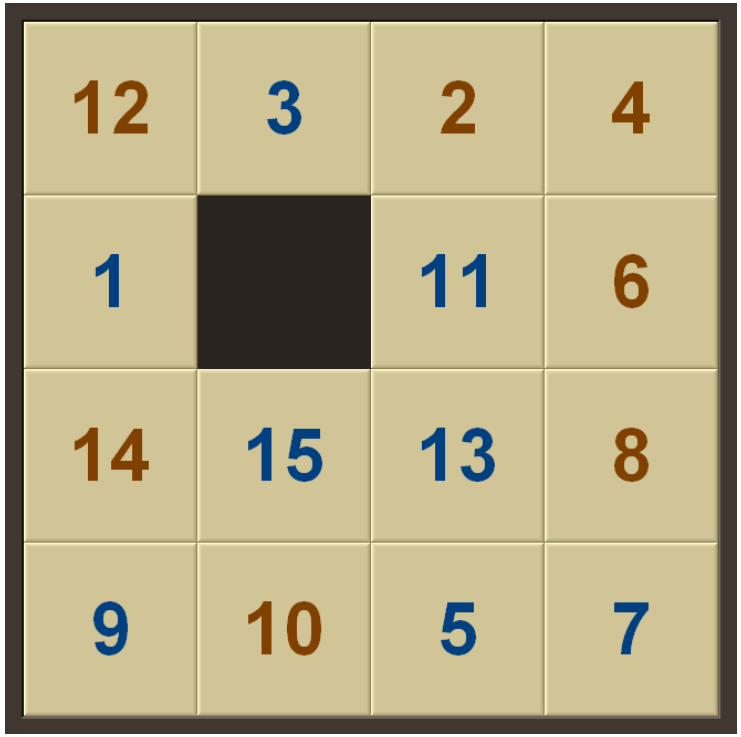
Problema del enigma n^2–1

Este problema se le atribuye a Sam Loyd , inventado en 1878.

Consiste de un tablero con:

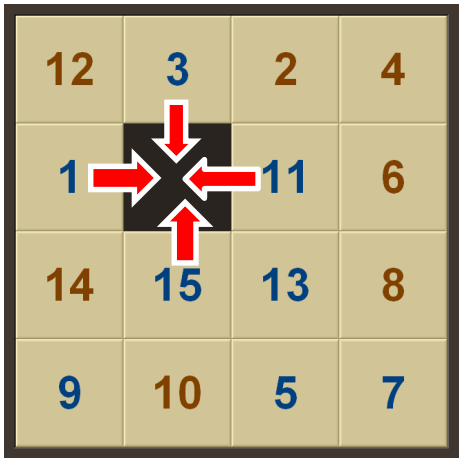
- n^2 casillas
- n^2–1 piezas
- numeradas de 1 a n^2–1.

Dada una ordenación inicial de todas las piezas en el tablero, queda sólo una casilla vacía: el “hueco”.

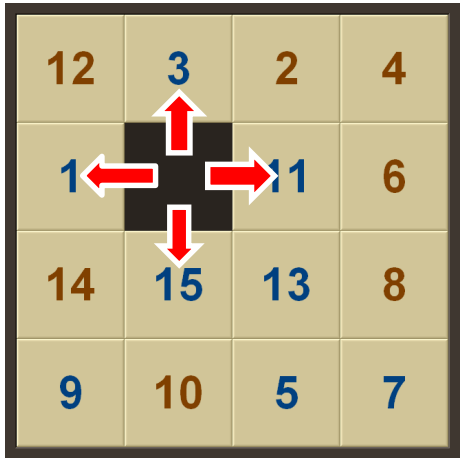


Problema del enigma n^2–1

Los únicos *movimientos permitidos son los de las piezas adyacentes (horizontal y verticalmente) al hueco*, que pueden ocuparlo; al hacerlo, dejan el hueco en la posición en donde se encontraba la pieza antes del movimiento.



Otra forma de abordar el problema es *considerar que lo que se mueve es el hueco*, pudiendo hacerlo hacia arriba, abajo, izquierda o derecha (siempre sin salirse del tablero). Al moverse, su casilla es ocupada por la pieza que ocupaba la casilla a donde se ha "movido" el hueco.



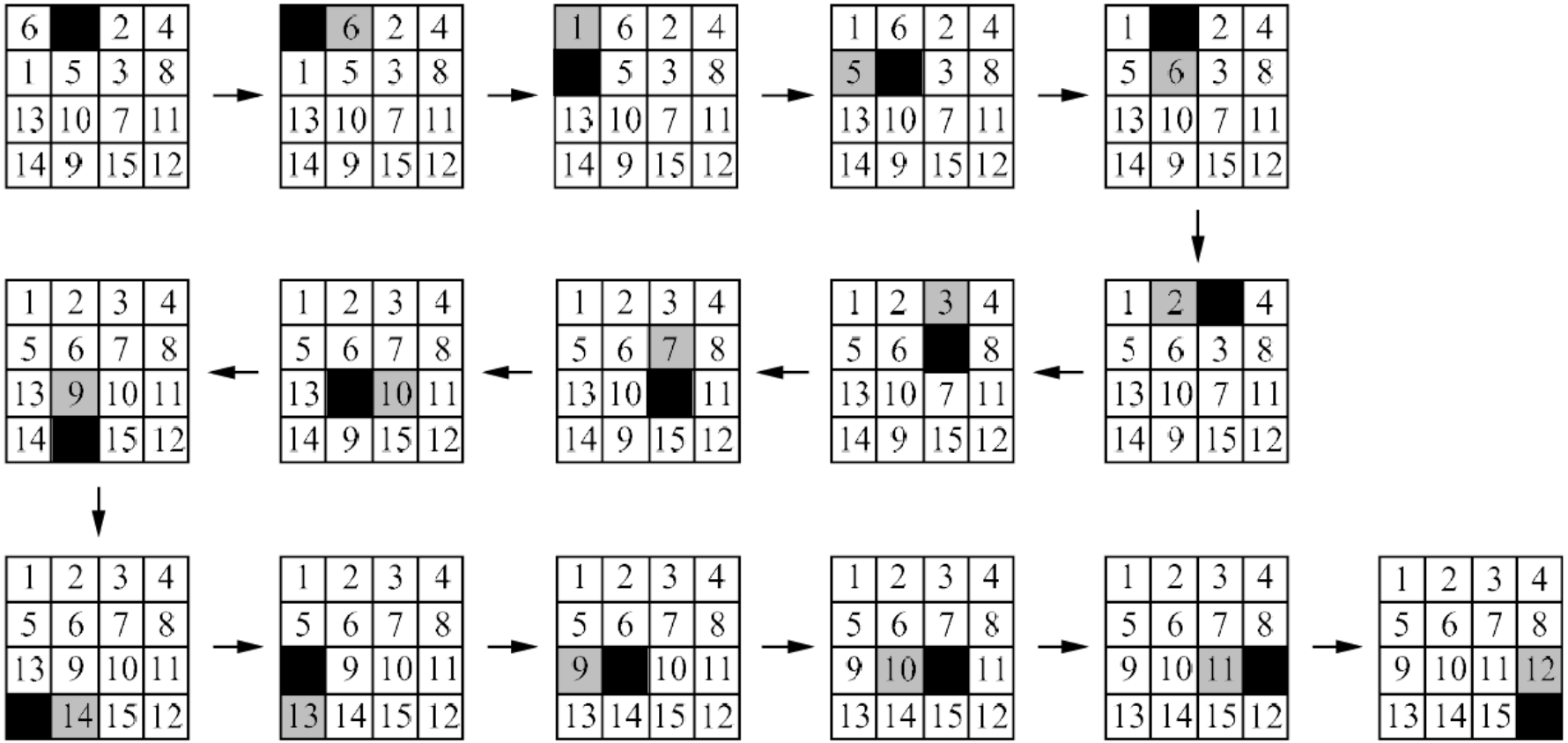
Problema del enigma n^2–1

Objetivo: mover las piezas desde la disposición inicial hasta una disposición final ordenada, en donde:

- en la casilla (i,j) se encuentra la pieza numerada $(i–1)*n + j$
- en la casilla (n,n) se encuentra el hueco.

Juego del 15

Ejemplo:



Juego del 15

Para jugar: https://lorecioni.github.io/fifteen-puzzle-game/

Inicial:

Final:

Problema del enigma n^2–1

Solución, primeras ideas:

Para resolver este problema con Ramificación y Poda es necesario en primer lugar construir el árbol de expansión.

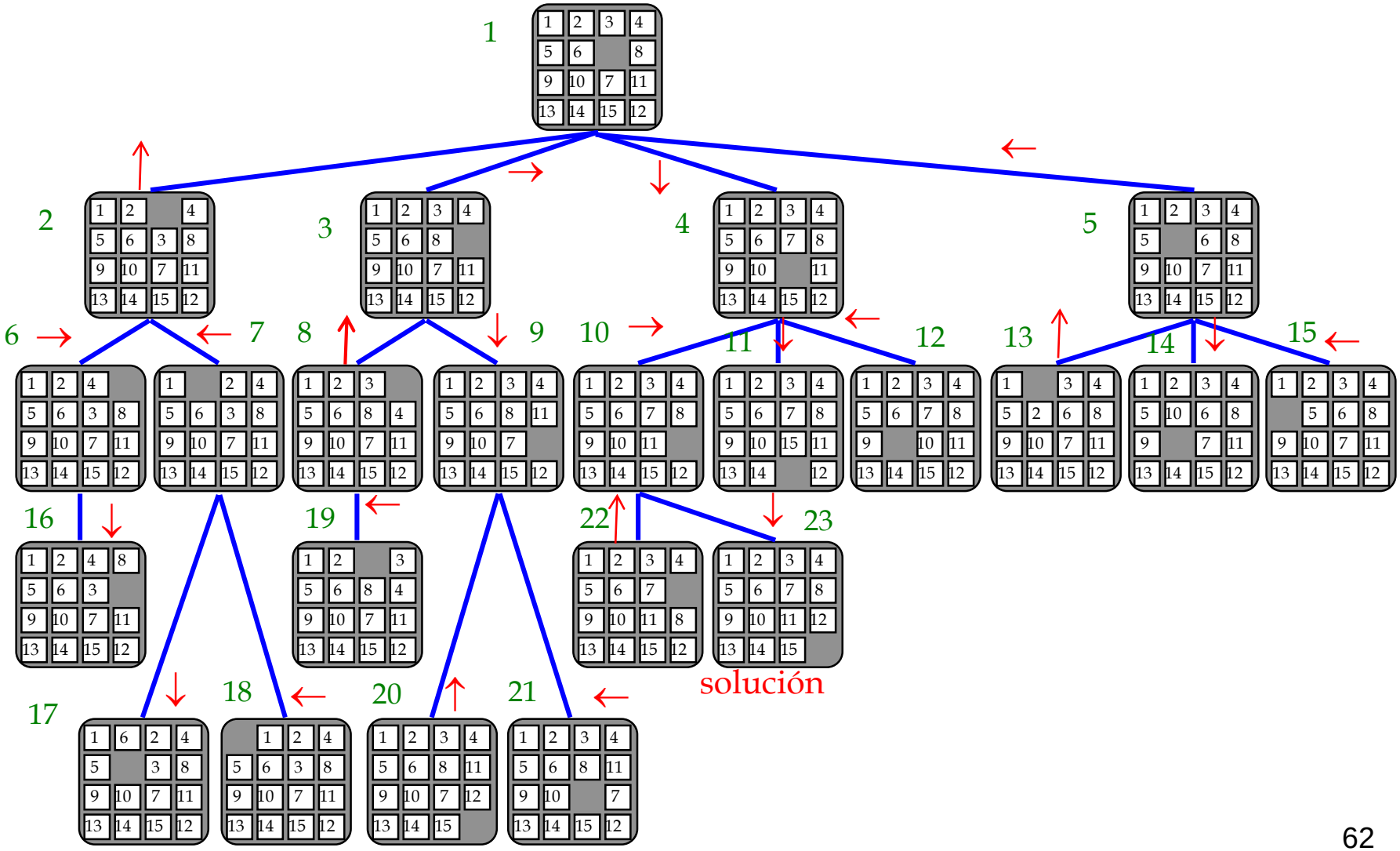
Plantearlo como una *secuencia de decisiones*, una en cada nivel del árbol.

Posibles decisiones: *cuatro movimientos del hueco* (arriba, abajo, izquierda y derecha) siempre que éstos sean válidos, es decir, siempre que no caigan fuera del tablero.

Se puede utilizar (entre otras) algunas de las siguientes *funciones de costo*:

- h1(n)= número de piezas que están en una posición distinta de la que les corresponde en la disposición final.
- h2(n)= suma, para cada ficha, de la distancia a la posición donde le tocaría estar. La distancia de Manhattan entre dos puntos del plano de coordenadas ($x1$,$y1$) y ($x2$,$y2$) viene dada por: $|x1 - x2| + |y1 - y2|$.

Juego del *15*



Problema del enigma n^2–1

Solución:

- Una primera aproximación a la solución del problema consiste en definir cada uno de los nodos como una matriz.

- Sin embargo, esto no va a ser suficiente puesto que no se dispone asi de “historia” sobre los movimientos ya realizados, lo cual llevaría posiblemente a ciclos en donde se repetirían indefinidamente movimientos de forma circular.

- Aparte de esta razón, también se sabe que los nodos utilizados en la técnica de Ramificación y Poda deben de ser autónomos, es decir, contienen toda la información necesaria para recuperar a partir de ellos la solución construida hasta el momento.

- En este caso la solución debe estar compuesta por una sucesión de matrices que muestran la serie de movimientos a realizar para llegar a la disposición final. Entonces cada nodo debe tener la lista los nodos que él ha generado

Problema del enigma n^2–1

Una vez que se dispone de la representación de los valores del tipo abstracto de datos, se implementan sus operaciones.

- La función *NodoInicial* *d*ebe de contener la disposición inicial del tablero.
- La estrategia de ramificación está a cargo de la *función Expandir*:
  Primero debe encontrar el hueco, luego se lo tiene que mover en 4 direcciones (si es que fuera posible el movimiento)
- Para implementar la función que realiza la *poda*, se va a podar aquellos nodos cuyo último movimiento haga aparecer un ciclo.
- Una función que también es necesario implementar es la que define la *función de costo* que va a contar el número de piezas que se encuentran fuera de lugar.
- Se necesita implementar una función para determinar cuándo *un nodo es solución*: es suficiente comprobar que su función de costo vale cero.
- La *función Eliminar* es la que va a devolver al sistema los recursos ocupados por un nodo, actuando como destructor del tipo abstracto de datos.

Problema del enigma n^2–1

Observación:

- Puede ocurrir puesto que no todas las disposiciones iniciales permitan llegar a la disposición final, como por ejemplo ocurre para la siguiente disposición inicial:

1	**3**	**2**
4	**5**	**6**
7	**8**	

- En este caso no habrá una solución para el problema.

Ramificación y poda

Trabajo Práctico no. 8