Algoritmos y Estructuras de Datos II Clase 17

Carreras:

Licenciatura en Informática

Ingeniería en Informática

2024

Unidad III

Técnicas de diseño de algoritmos

Ramificación y poda(1)

(en inglés: ***Branch and Bound***)

Ramificación y Poda

- La idea central de backtracking es no expandir los nodos del árbol de expansión de estados si se puede deducir que no conducen a una solución del problema. Esta idea se puede reforzar cuando se trata con problemas de optimización.

- Un problema de optimización busca minimizar o maximizar una función objetivo, generalmente sujeto a algunas restricciones.

- En los problemas de optimización:

➢ Una **solución factible** es un punto en el espacio de soluciones que satisface las restricciones del problema.

➢ Una **solución óptima** es una solución factible que tiene el mejor valor de la función objetivo (mínimo o máximo).

Ramificación y Poda

En un algoritmo de ramificación y poda, en general se termina el camino de búsqueda desde un nodo del árbol de expansión de estados por alguna de las siguientes situaciones:

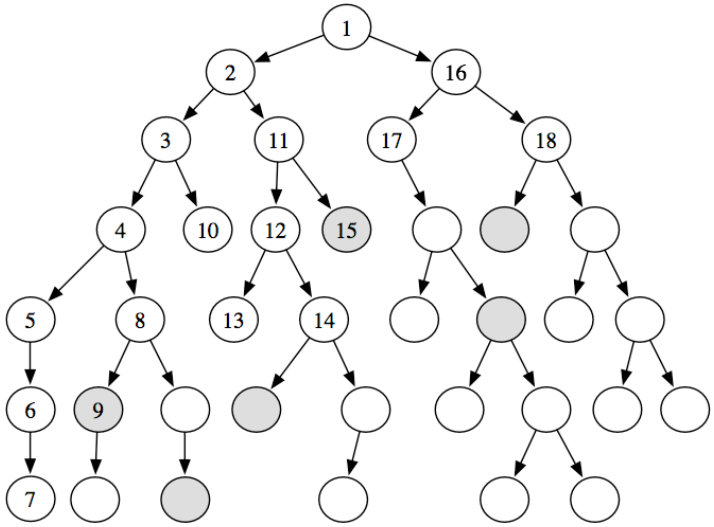
- El valor de la cota del nodo no es mejor que el valor de la mejor solución conseguida hasta el momento.
- El nodo representa una solución no factible porque viola las restricciones del problema.
- El conjunto de soluciones posibles representadas por ese nodo es un punto único y no puede expandirse. En este caso se compara el valor de la función objetivo de la solución factible con la de la mejor solución obtenida hasta el momento y se la actualiza en el caso de que la de este nodo sea mejor.

Ramificación y Poda

- Esta técnica de diseño se aplica generalmente para resolver *problemas de optimización*.
- Especialmente sirve para *Problemas de optimización combinatoria*: son problemas de optimización, muy fáciles para describir, pero tienen un gran número (finito) de soluciones posibles.
- La mayoría de estos problemas no tienen un algoritmo polinómico que los resuelva.
- Ramificación y poda es la técnica más eficiente para resolver los problemas NP de optimización combinatoria.

Ramificación y Poda

Ejemplos de aplicación:



- Problema de la Mochila
- Problema del camino más corto
- Problema del Agente viajero (TSP)
- Problema de las n reinas
- Juego del n^2-1
- Asignación de tareas
- Coloreado de un mapa
- Círculo más pequeño que cubre n puntos en un plano
- Problema de selección del k-ésimo menor en tiempo lineal
- Problema de la mediana en tiempo lineal
- Cubrimiento de área mínima con n piezas rectangulares
- Programación lineal con 2 variables

Ramificación y Poda

- La técnica de Ramificación y Poda, lo mismo que el diseño Vuelta Atrás, realiza una enumeración parcial del espacio de soluciones basándose en la generación de un árbol de expansión.
- Es una técnica de búsqueda de soluciones basada en el proceso de *explorar un árbol implícito*.

- Desventaja: el recorrido de todo el árbol llevaría un tiempo de ejecución prohibitivo en los casos de algoritmos basados en la exploración del árbol con todas las posibilidades.

Ramificación y Poda

- La principal diferencia con backtracking es que la generación de nodos del árbol de expansión se puede realizar aplicando distintas estrategias que se llaman *estrategias de ramificación*.

- Además se utilizan cotas que permiten podar ramas que no conducen a una solución óptima (se evita ramificar nodos), estas técnicas se conocen con el nombre de *estrategias de poda.* Esto lleva a no explorar aquellas ramas que no conducen a una solución válida u óptima (en el caso de problemas de optimización)

Ramificación y Poda

En líneas generales se pretende usar el siguiente Procedimiento:

- Aplicar una buena técnica de ***ramificación*** obteniendo para cada nodo una estimación de beneficio de la solución óptima que se puede encontrar a partir de él.
- Usar esta ***estimación*** para decidir qué zonas del árbol explorar primero, generando en cada paso los hijos del nodo con mayor beneficio estimado.
- La ***poda*** se realizará calculando en cada nodo cotas del beneficio que se puede alcanzar a partir de ese nodo.

Ramificación y Poda

- El diseño Vuelta Atrás realiza la generación de descendientes de una manera sistemática y de la misma forma para todos los problemas, haciendo un recorrido en profundidad del árbol que representa el espacio de soluciones.

- Una característica que diferencia a la estrategia de Ramificación y Poda del diseño Vuelta Atrás es la posibilidad de generar nodos siguiendo *distintas estrategias*.

Ramificación y Poda

El diseño Ramificación y Poda en su versión más sencilla puede seguir distintos recorridos:

- Un recorrido en ***profundidad*** (estrategia LIFO)
- Un recorrido en ***amplitud*** (estrategia FIFO)
- Un recorrido seleccionando el nodo que en principio parece ***más prometedor***, usando para ello los resultados de un cálculo realizado mediante funciones de costo (estrategia del *montículo*).

Ramificación y Poda

Además de estas estrategias de ramificación, esta técnica utiliza el cálculo de cotas para ***podar aquellas ramas del árbol que no conducen a la solución óptima***.

- Para poder realizar la poda:
  - Se calcula en cada nodo una cota del posible valor de aquellas soluciones alcanzables desde ése nodo.
  - Si la cota muestra que cualquiera de estas soluciones tiene que ser necesariamente peor que la mejor solución hallada hasta el momento no se necesita seguir explorando por esa rama del árbol, se realiza el ***proceso de poda***.

Nodo Vivo

- Se define ***nodo vivo*** del árbol a un nodo *con posibilidades de ser ramificado*, esto significa que no ha sido tratado ni podado.

- Los nodos vivos se almacenan en alguna Estructura de datos que se pueda recorrer (pila, fila, cola de prioridad), dependiendo de la estrategia de búsqueda seleccionada

- El algoritmo recorre la estructura de datos y determina cual es el próximo nodo vivo que va a ser expandido.

Ramificación y Poda

La idea básica es la siguiente:

- Sacar un elemento de la Estructura de nodos vivos.
- Generar todos los descendientes del nodo (ramificación).
- Se analiza cada nodo. Si el nodo no se poda y tampoco es solución, vuelve a la Estructura de nodos vivos.

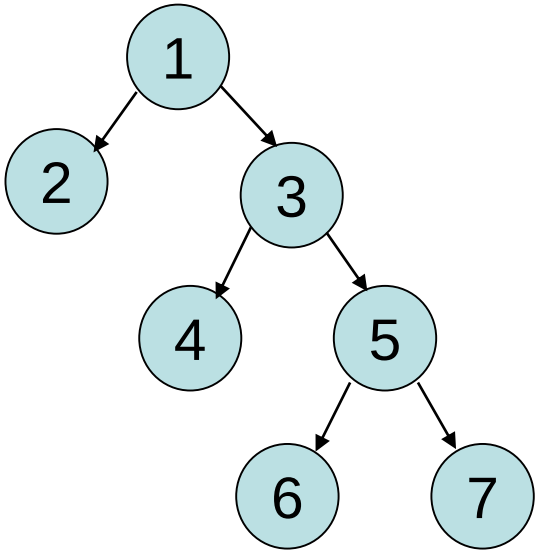
El recorrido del árbol depende de como se maneje la lista:

- Recorrido en profundidad (pila).
- Recorrido en amplitud (fila).
- Estrategia mas prometedor (cola con prioridades o montículo).

Estructura de nodos vivos

Estructura PILA (LIFO - Last In First Out)

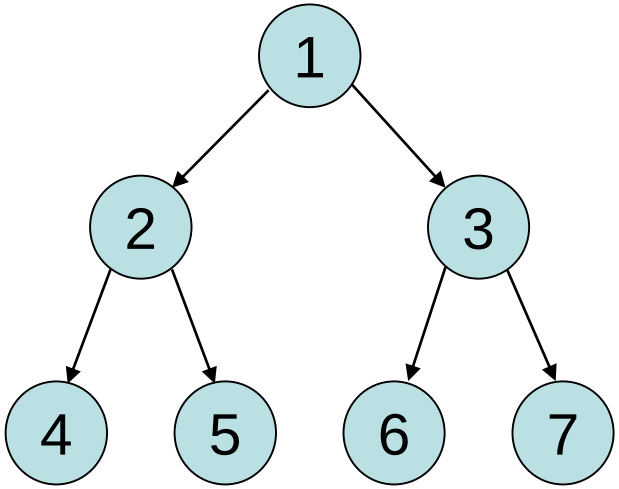
- La lista de nodos vivos se almacena en una pila
- El árbol se recorre en profundidad



Estructura de nodos vivos

Estructura FILA (FIFO - First In First Out)

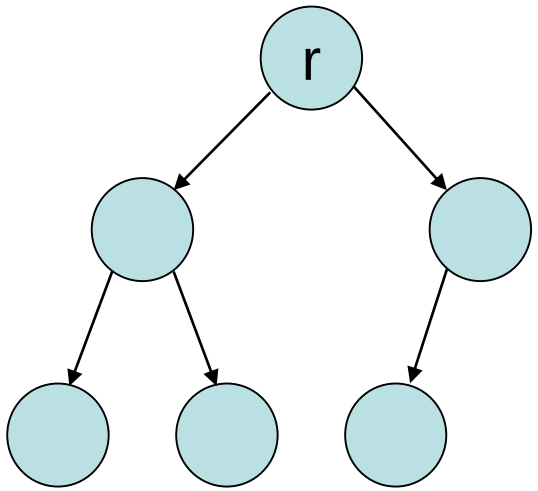
- La lista de nodos vivos se almacena en una fila
- El árbol se recorre en amplitud



Estructura de nodos vivos

Estructura Montículo (Heap)

- La lista de nodos vivos se almacena en un árbol binario, generalmente implementado con un arreglo.
- Del árbol se usa la raíz principal, se elimina y se mantiene la condición de heap (casi completo y semiordenado)



Etapas de Ramificación y Poda

Básicamente, se realizan tres etapas:

1. ***Selección***: se encarga de extraer un nodo de entre el conjunto de los nodos vivos. La elección depende de la estrategia de búsqueda que utilice el algoritmo.
2. ***Ramificación***: se construyen todos los posibles nodos hijos del nodo seleccionado en el paso anterior.
3. ***Poda***: se eliminan algunos de los nodos creados en la etapa anterior. Aquellos nodos no podados pasan a formar parte del conjunto de nodos vivos, y se comienza de nuevo por el proceso de selección.

El algoritmo finaliza cuando encuentra la solución, o bien cuando se agota el conjunto de nodos vivos.

Algoritmo Ramificación y Poda

Esquema de Diseño de alto nivel del algoritmo

Inicialización:

Introducir la raíz del árbol en la Estructura de nodos vivos

Inicializar la variable de cota de poda

Mientras la Estructura de nodos vivos no esté vacía hacer:

Sacar un nodo según la estrategia de ramificación

Comprobar el nodo debe ser podado según la estrategia de poda

Si el nodo no debe ser podado entonces

Generar todos sus hijos

Para todos los hijos desde primer hijo hasta ultimo hijo hacer

Comprobar si es solución y actualizar cota de poda

Si no debe ser podado entonces

Agregar el nodo a la Estructura

Fin

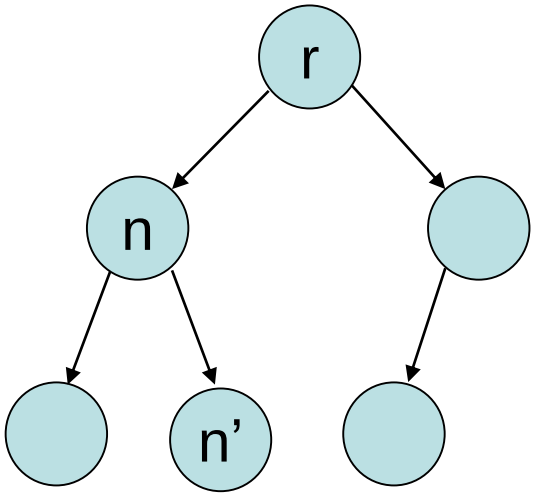
Función de Costo

- Para cada nodo del árbol se dispone de una ***función de costo*** que estima el valor óptimo de la solución si se continua por ese camino.
- De esta manera, si la cota que se obtiene para un nodo, es peor que una solución ya obtenida por otra rama, se puede podar esa rama porque no es interesante seguir por ella.
- Evidentemente *no se puede realizar ninguna poda hasta que se haya encontrado alguna solución.*
- Se puede arrancar con una solución obtenida por una técnica greedy que aunque no sea la óptima será cercana a la óptima.

Función de Costo

- Las funciones de costo deben ser crecientes respecto a la profundidad del árbol, es decir, si *h* es una función de costo entonces:

$h(n) \leq h(n')$ para todo n' nodo descendiente de n.



Función de Costo

- La dificultad en estos algoritmos está en **encontrar una *buena función de costo*** para cada problema.
- La función de estimación de costo tiene que ***"ser buena"*** en dos sentidos:
  - que garantice la poda
  - que su cálculo no sea muy costoso (tampoco demasiado simple porque probablemente pocas ramas puedan ser excluidas).
- La **eficiencia del algoritmo** reside en este punto.

Eficiencia de Ramificación y Poda

¿Cómo lograr que un algoritmo R y P sea eficiente?

Si las funciones de estimaciones de costo son:

- ***muy precisas***:→ poda exhaustiva del árbol
  Ventaja: se recorren pocos nodos.
  Desventaja: se gasta mucho tiempo en realizar cada una de las estimaciones.

- ***poco precisas***: → no se hace mucha poda
  Ventaja: se gasta poco tiempo en cada nodo.
  Desventaja: el número de nodos puede ser muy elevado.

Meta: Buscar equilibrio entre la exactitud de las cotas y el tiempo de calcularlas.

Complejidad de Ramificación y Poda

Complejidad de los algoritmos de Ramificación y Poda

El orden de complejidad depende de 2 factores:

- El *numero de nodos* recorridos.
- El *tiempo gastado en cada nodo.*

Así:

- El numero de nodos depende de la efectividad de la poda.
- El tiempo requerido para cada nodo depende de:
  - La complejidad en el manejo de la estructura de datos usada.
  - El tiempo usado en el calculo de las estimaciones del costo.

Complejidad de Ramificación y Poda

Conclusión:

- En el *caso promedio*, se obtienen mejoras con respecto a la técnica de backtracking.

- En el *peor caso*, se recorren tantos nodos como en backtracking !

- En algunos casos el tiempo incluso puede ser peor, porque se agrega el costo del calculo de cotas y de manejo de la estructura.

Ventajas y Desventajas

- Una ventaja adicional que poseen estos algoritmos: *la posibilidad de ejecutarlos en paralelo.*

- El disponer de *algoritmos paralelizables* es muy importante en muchas aplicaciones en las que es necesario abordar los problemas de forma paralela para resolverlos en tiempos razonables, debido a su complejidad intrínseca.

- Los *requerimientos de memoria son mayores* que en los de los algoritmos Vuelta Atrás. Ya no se puede disponer de una estructura global en donde ir construyendo la solución. Ahora se necesita que cada nodo sea autónomo, en el sentido que tiene que contener toda la información necesaria para realizar los procesos de ramificación y poda, y para reconstruir la solución encontrada hasta ese momento.

Ventajas y Desventajas

- Los algoritmos de ramificación y poda *resultan difíciles de programar sin una estructura de datos adecuada*: tienen la necesidad de mantener una lista de nodos que han sido generados pero no han sido explorados en su totalidad, situados en diferentes niveles del árbol y preferiblemente ordenados por orden de las cotas correspondientes. El *montículo* es una estructura de datos ideal para almacenar esta lista.

- A diferencia del recorrido en profundidad, el programador *no dispone de una formulación recursiva* elegante de la ramificación y poda.

- Siempre se debe llegar a un compromiso en lo concerniente a la *calidad de la cota calculada*.

Asignación De Tareas

El problema de la asignación tiene numerosas aplicaciones. Por ejemplo, el problema en términos de edificios y terrenos, trabajos y obreros, etc.

Este problema consiste en: dadas ***n personas*** y ***n tareas,*** asignar a cada persona una sola tarea ***minimizando el costo de la asignación total.***

- Hay n! posibles asignaciones para considerar.
- Por lo tanto es interesante recurrir a la metodología de ramificación y poda.

El problema de la asignación de tareas que no puede resolverse por la técnica Greedy, si puede resolverse utilizando una técnica de **Ramificación y Poda.**

Asignación De Tareas

Diseñar un algoritmo utilizando la técnica de *Ramificación y Poda* para resolver el problema de la *Asignación de Tareas minimizando el costo total.*

El algoritmo recibe una **matriz C_{nxn}** de valores que determina el costo de asignar a cada una de las **n** personas la ejecución de cada una de las **n** tareas.

Datos:

matriz C_{nxn},

donde C(i,j) es el costo de que la persona **i** realice la tarea **j**,

Con $1 \leq i \leq n$, $1 \leq j \leq n$

Salida:

Vector T(1..n), donde el valor de las componentes T(i) almacena el valor de la tarea j asignada al trabajador i.

Objetivo: $\sum_{i=1}^{n} C(i, T(i))$ sea minima

Asignación De Tareas

Ejemplo:

Datos:

n=4,

letras(personas),

números(tareas),

matriz de costos: C(i,j) →

Agente\Tareas	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Dos soluciones rápidas:

⇒ a:1, b:2, c:3, d:4 cuyo costo es: 11+15+19+28=73

⇒ d:1, c:2, b:3, a:4 cuyo costo es: 17+17+13+40=87

La solución óptima del problema entonces no puede costar más que 73.

cota superior=73

El mínimo costo posible para cada tarea (sin importar quien la realice) se obtiene calculando el mínimo de cada columna. 11+12+13+22=58

cota inferior=58

→La respuesta está en el intervalo [58..73]

Asignación De Tareas

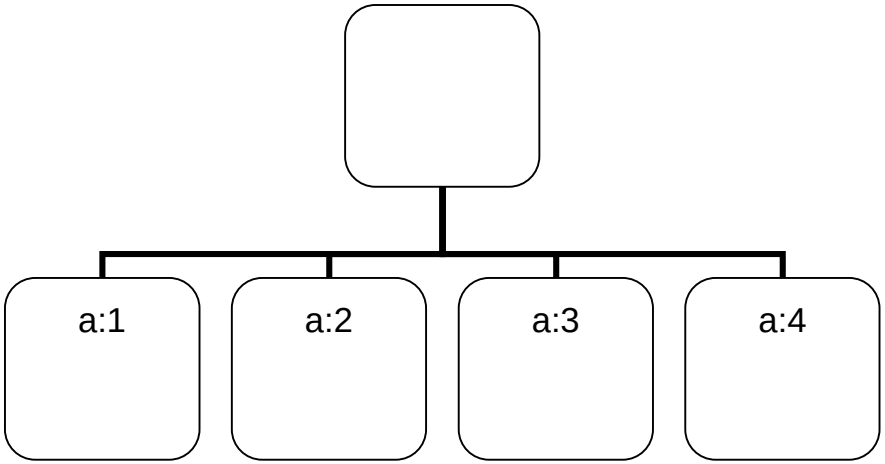
Para resolver el problema mediante ramificación y poda, se construye un árbol de expansión :

- En la raíz no se hacen asignaciones.
- Los nodos corresponden a asignaciones parciales.
- 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.
- Se usa esa cota para podar y para guiar la búsqueda.

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

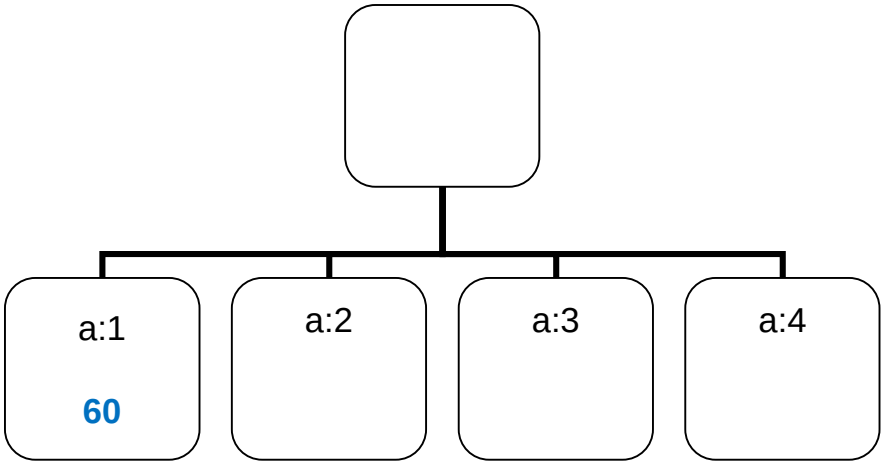


- Ahora hay que determinar la cota en cada nodo.

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

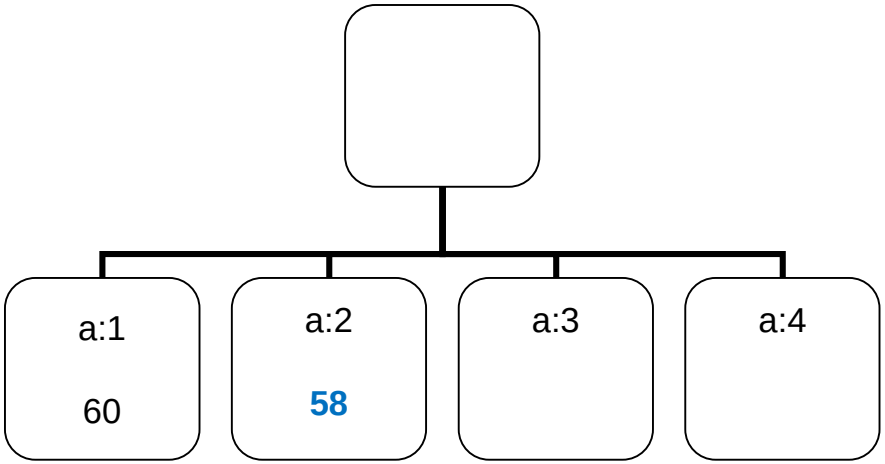


a:1 11+14+13+22=60

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

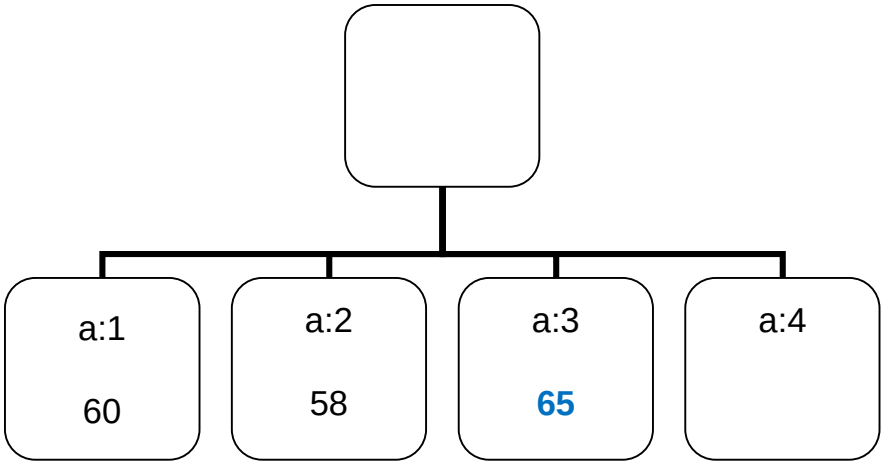


a:2 12+11+13+22=58

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

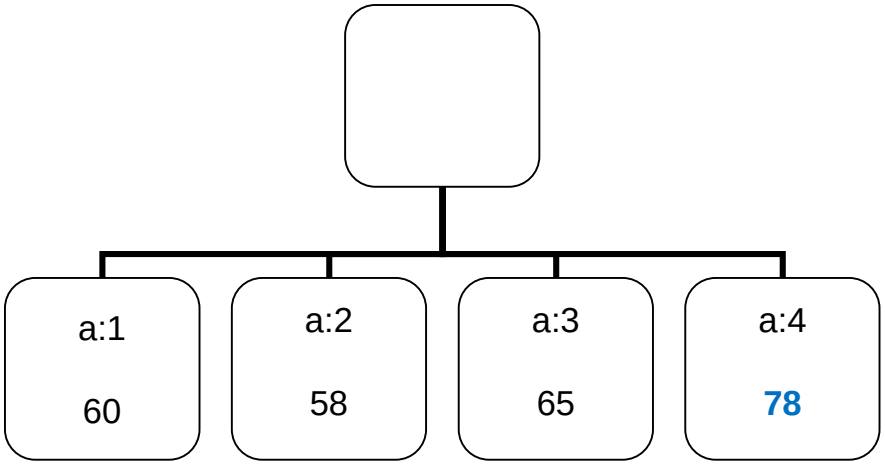


a:3 18+11+14+22=65

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

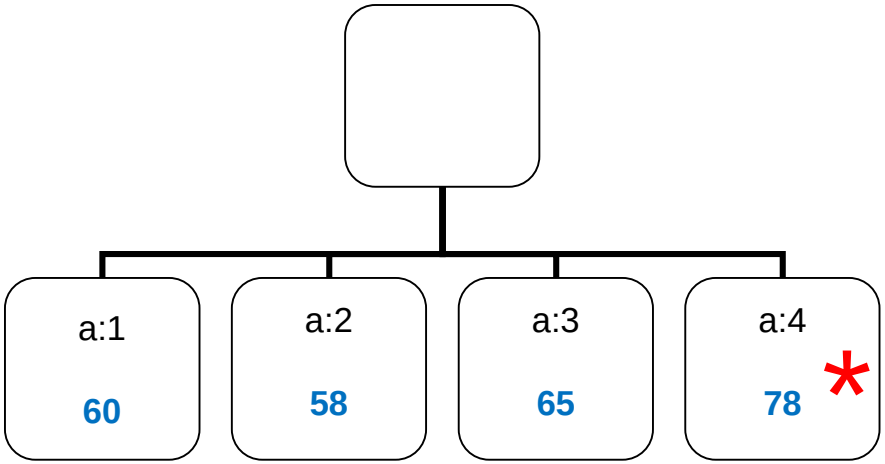


a:4 40+11+14+13=78

Asignación De Tareas

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

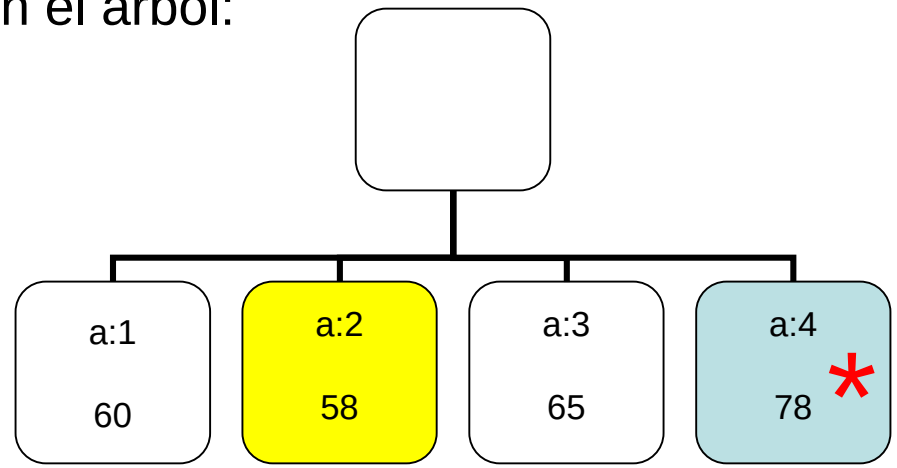
A/T	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28



- 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:

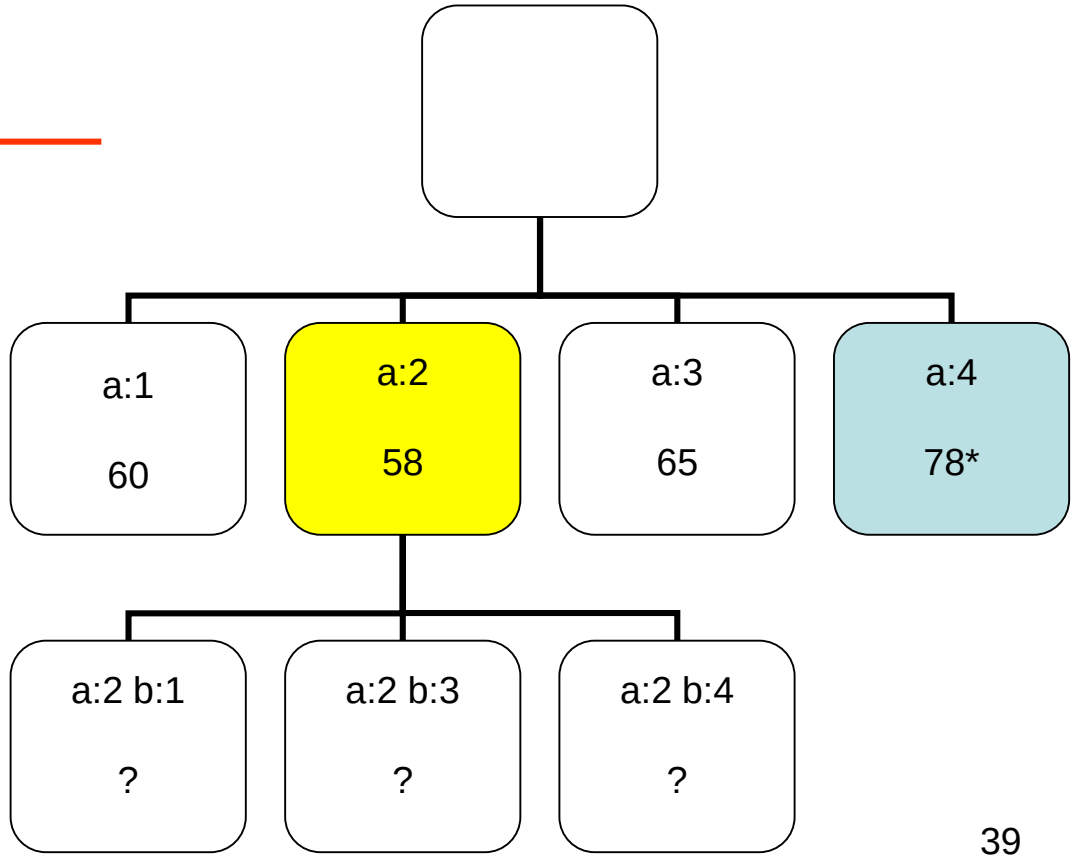


- La cifra obtenida para cada nodo es una cota inferior para las soluciones que se pueden obtener de ese nodo.
- NOTA: estos valores de cota inferior no son necesariamente alcanzables.

Asignación De Tareas

- De los 4 nodos, el nodo más prometedor es (a:2), tiene la cota inferior menor de todas (58), 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

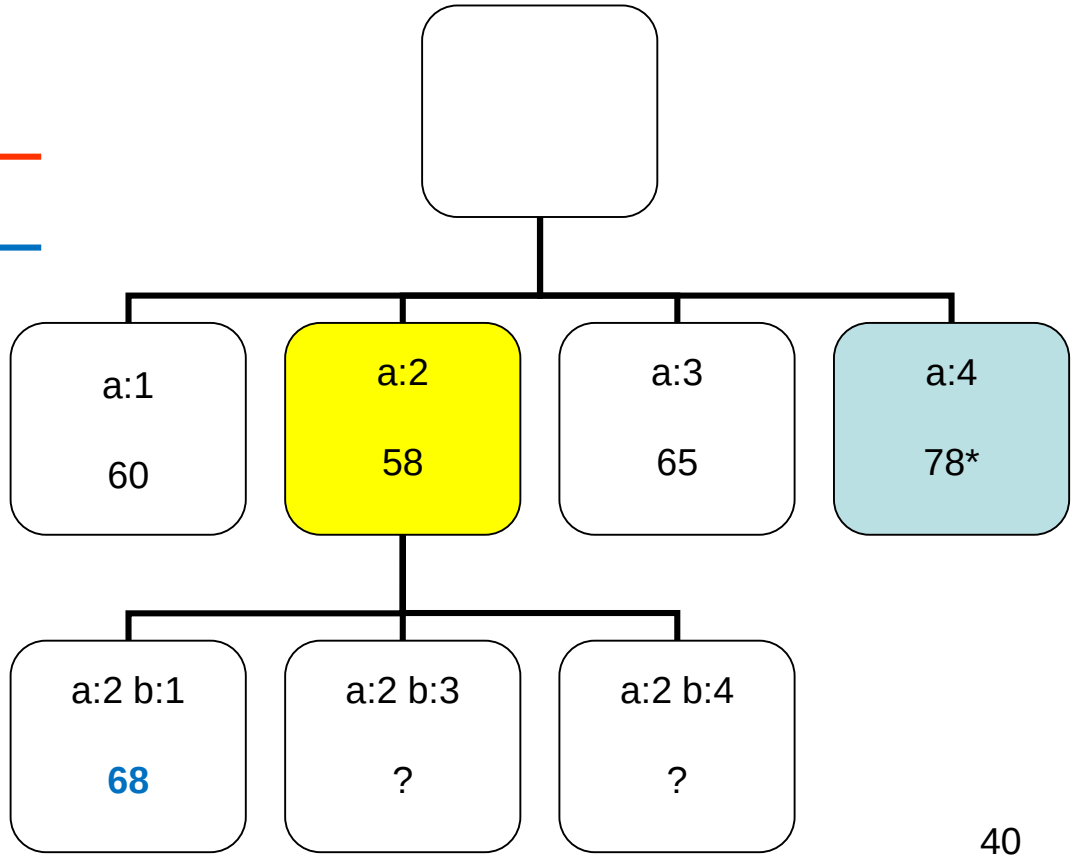


Asignación De Tareas

- Se explora 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+23=**68**

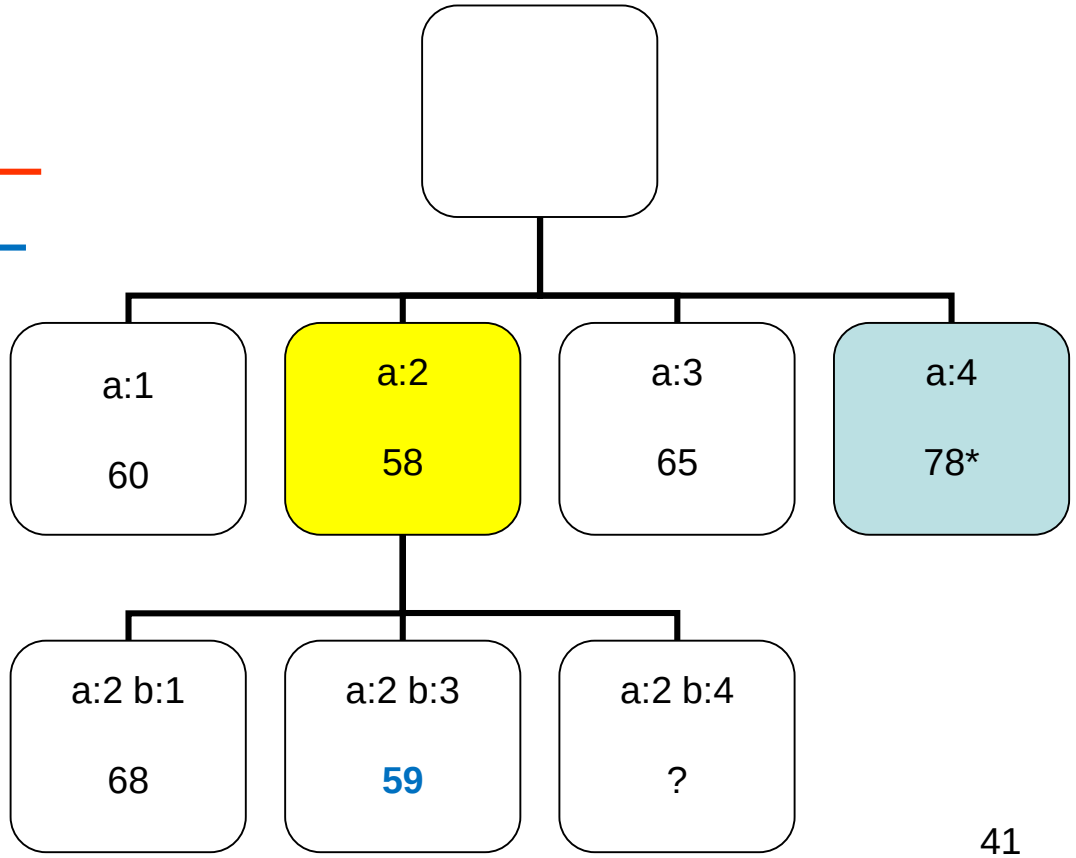


Asignación De Tareas

- Se explora 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+23=**59**

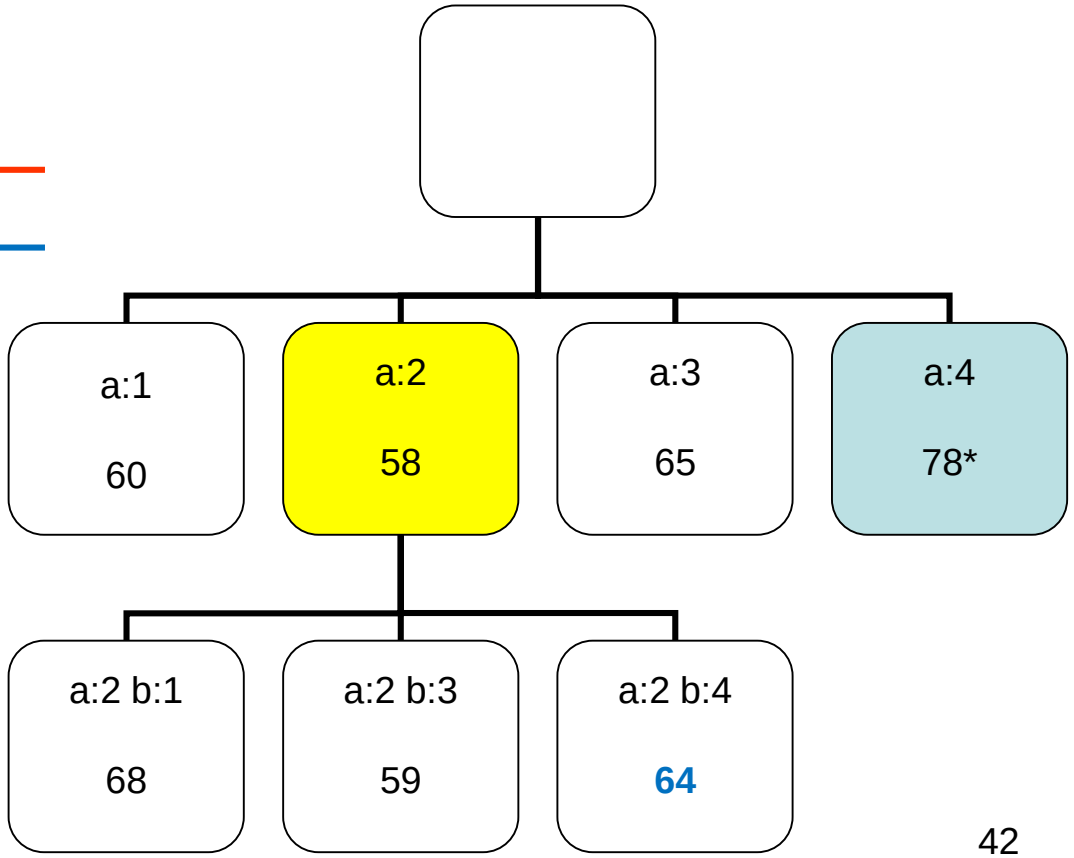


Asignación De Tareas

- Se explora 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

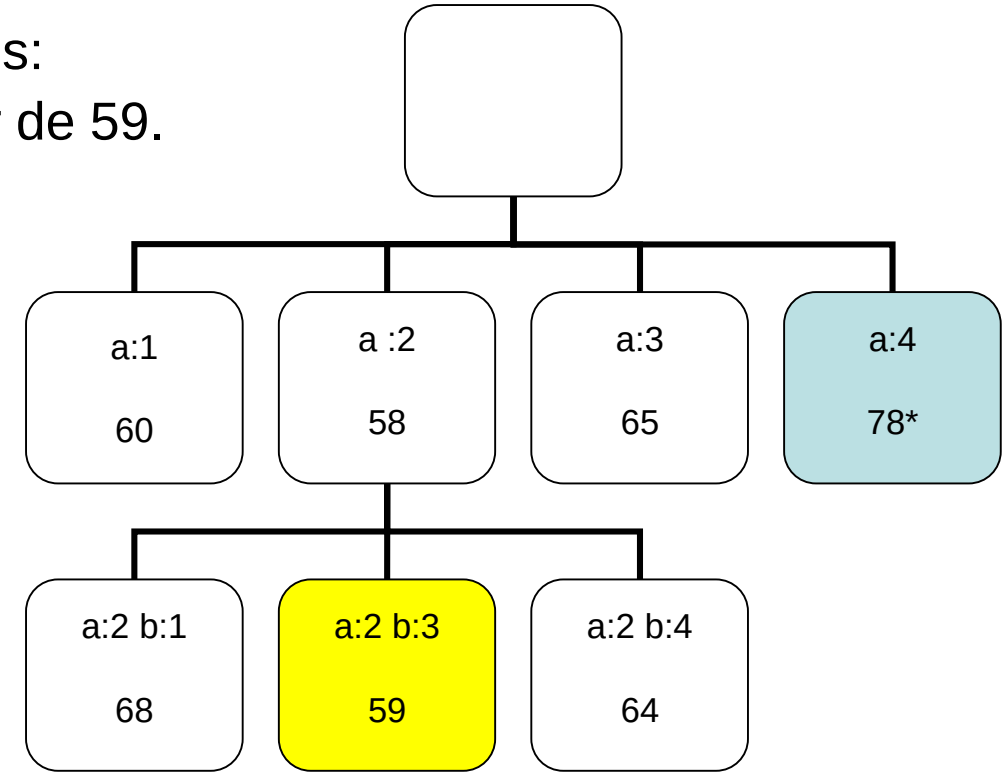
a:2 b:4 12+22+11+19=**64**



Asignación De Tareas

- El nodo más prometedor es:
  (a:2),(b:3) con cota inferior de 59.

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

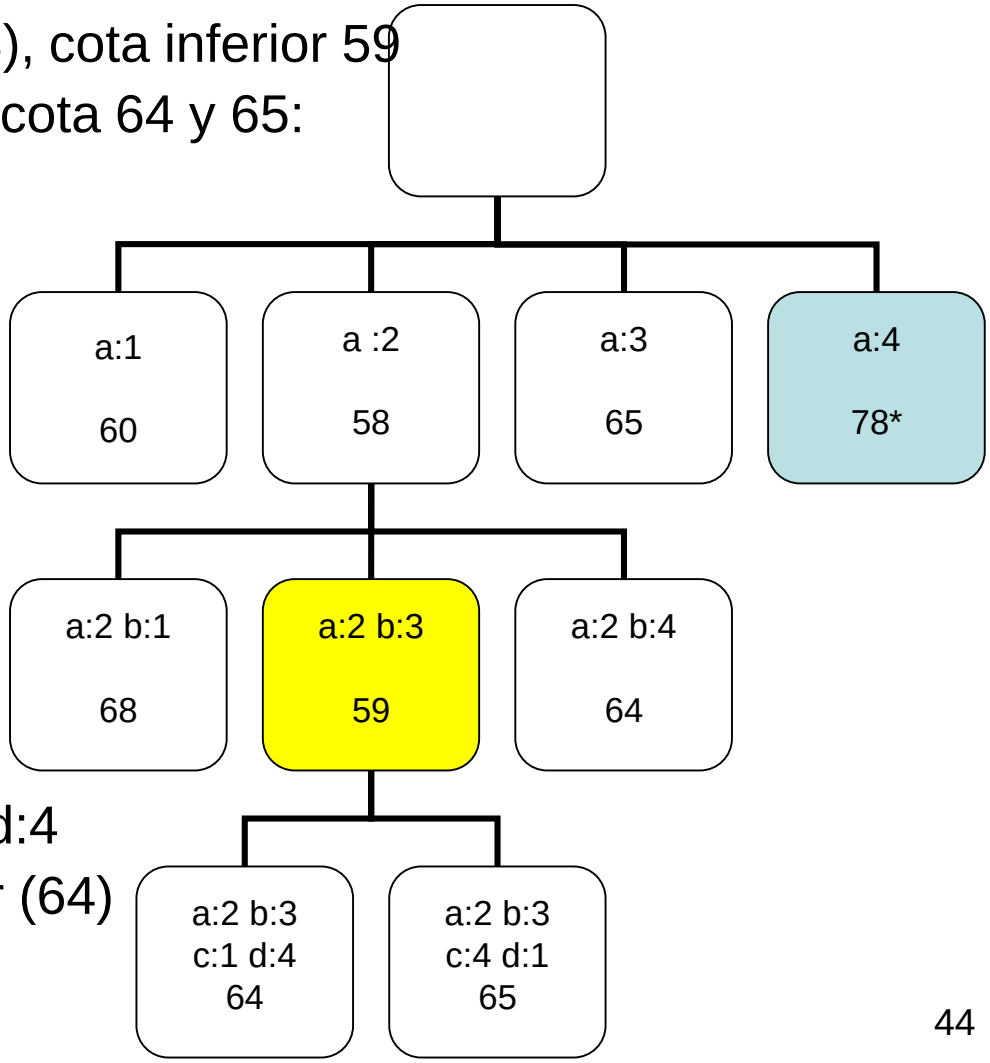


- Se ramifica el nodo de cota 59.

Asignación De Tareas

- Se ramifica el nodo (a:2),(b:3), cota inferior 59
- 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



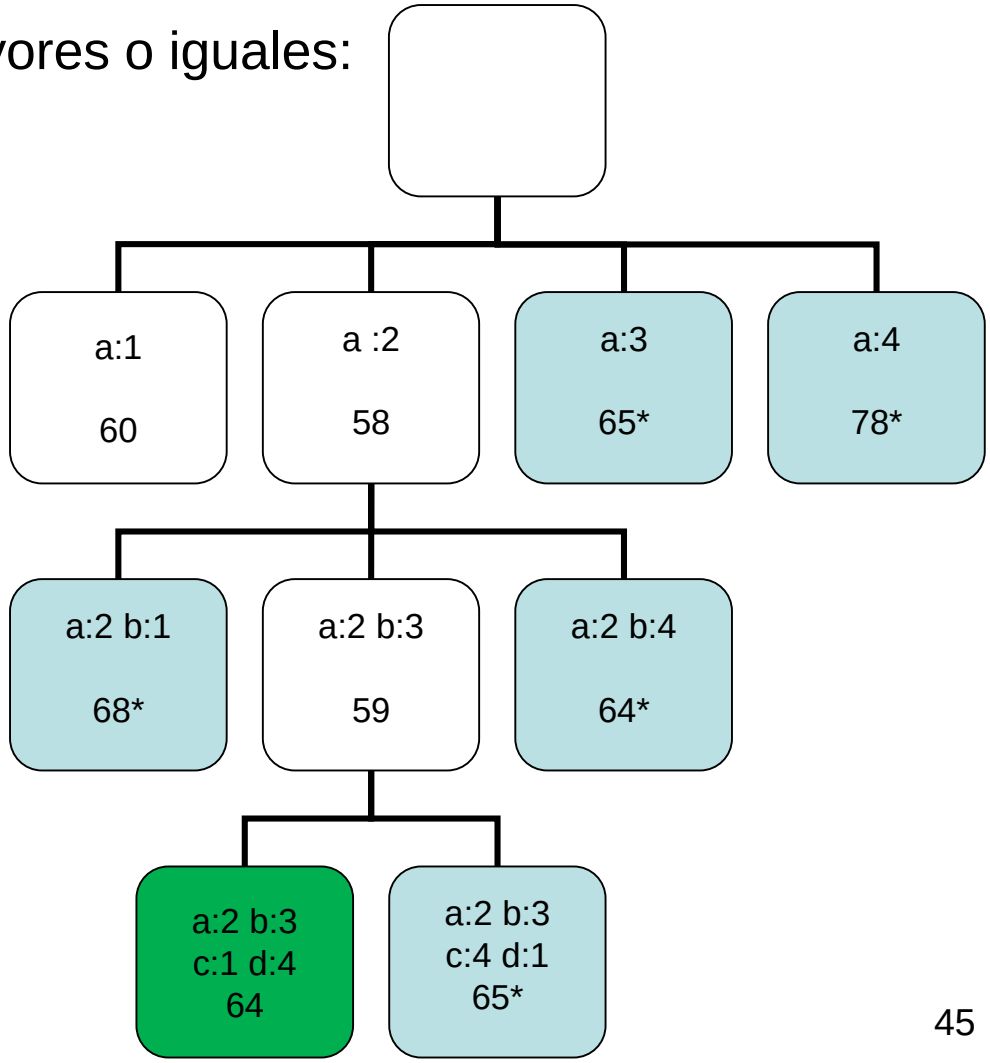
- El nodo solución a:2 b:3 c:1 d:4

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

Asignación De Tareas

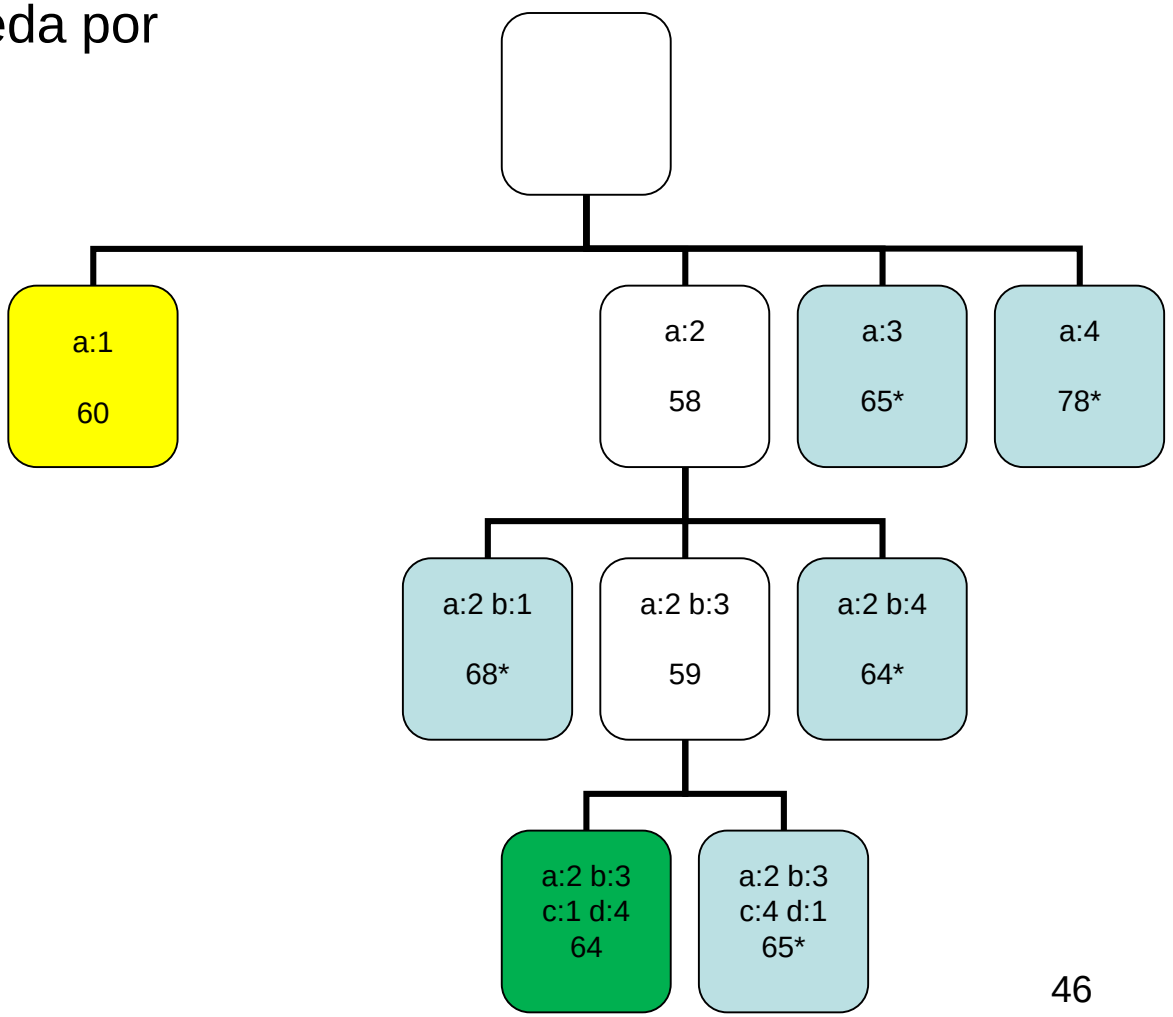
Se podan nodos con cota mayores o iguales: cota 64, 65 y 68

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



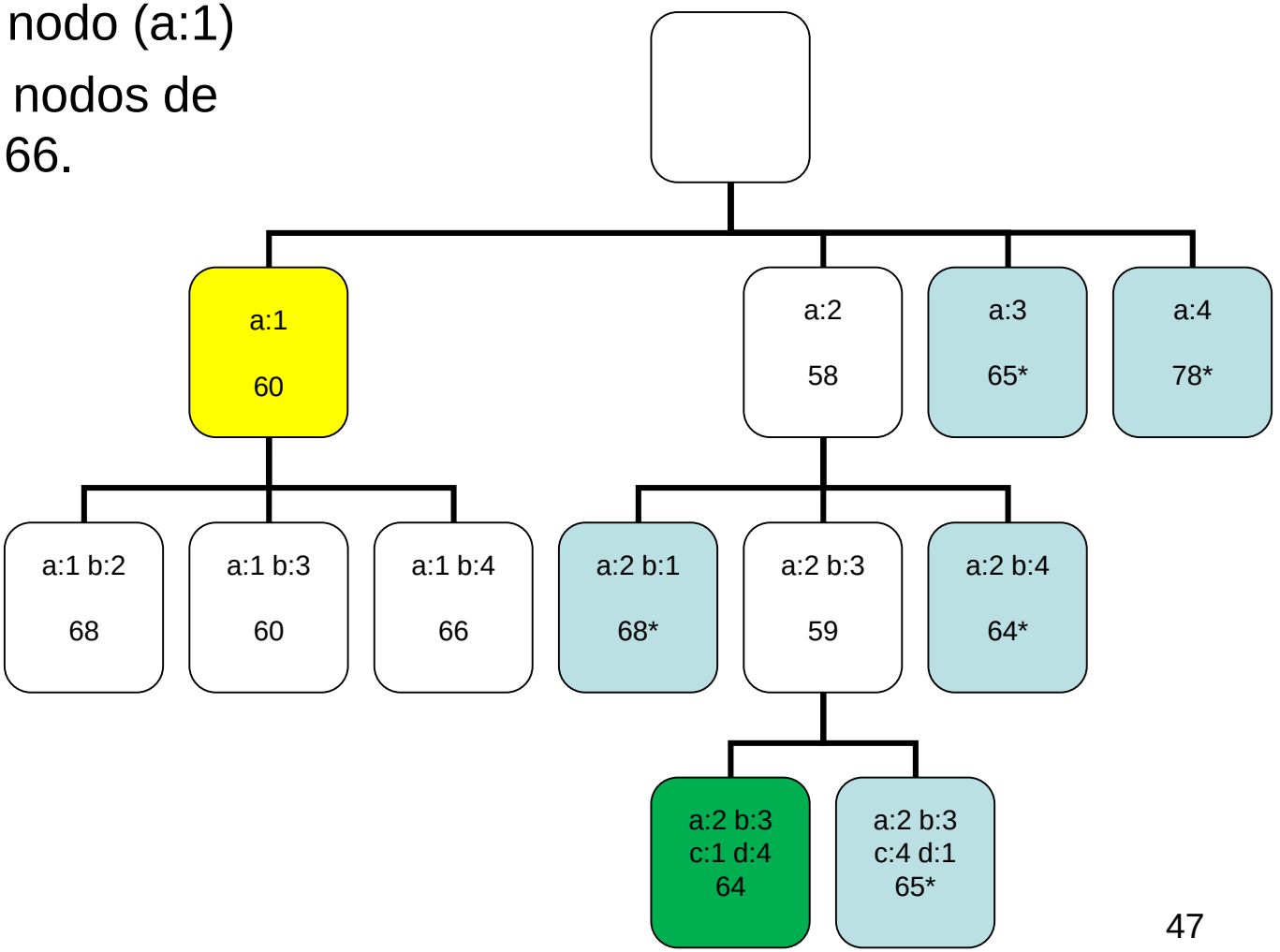
Asignación De Tareas

- El único nodo que queda por explorar es (a:1).



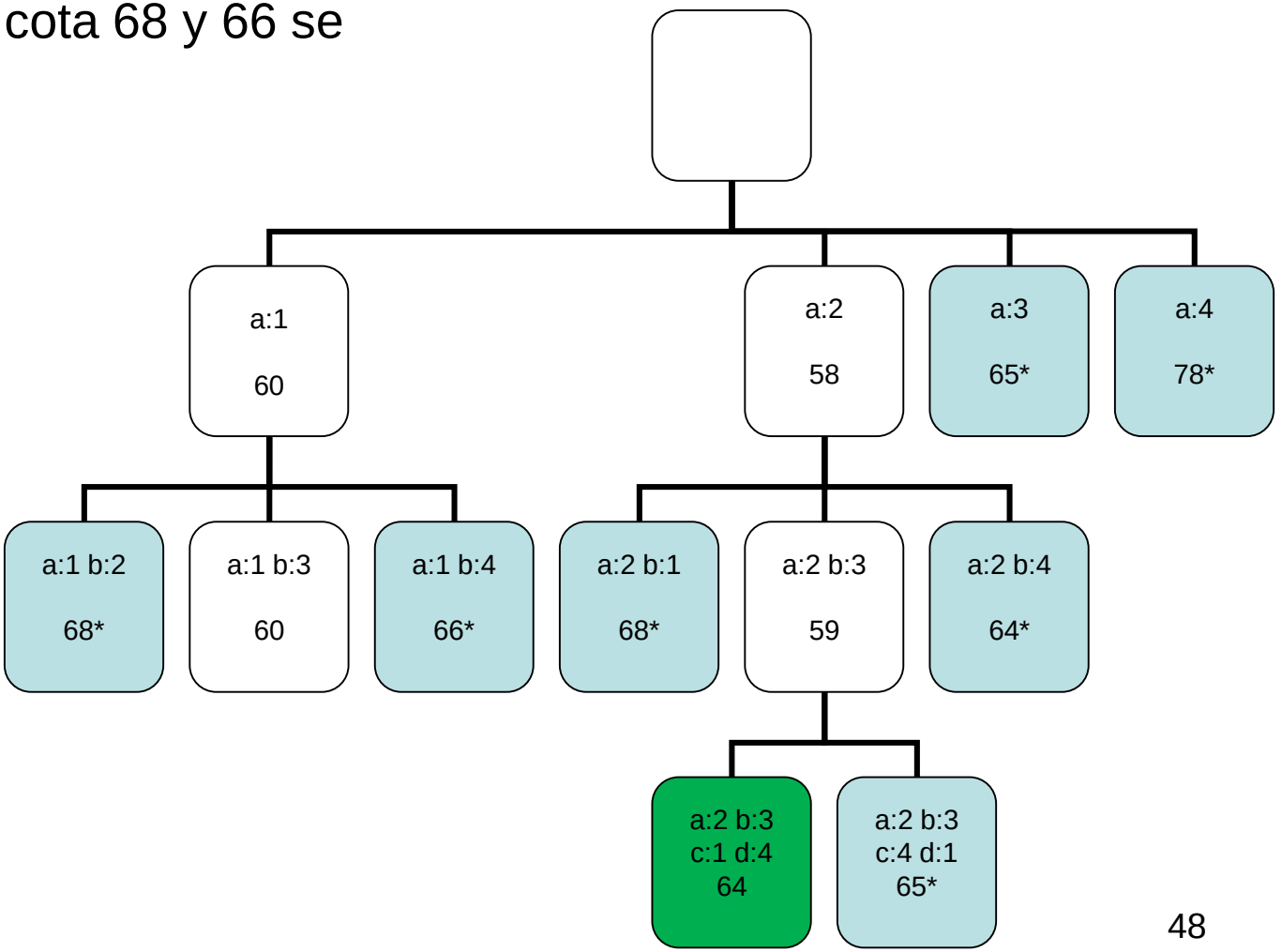
Asignación De Tareas

- Se ramifica el nodo (a:1)
- Se obtienen 3 nodos de cota: 68, 60 y 66.



Asignación De Tareas

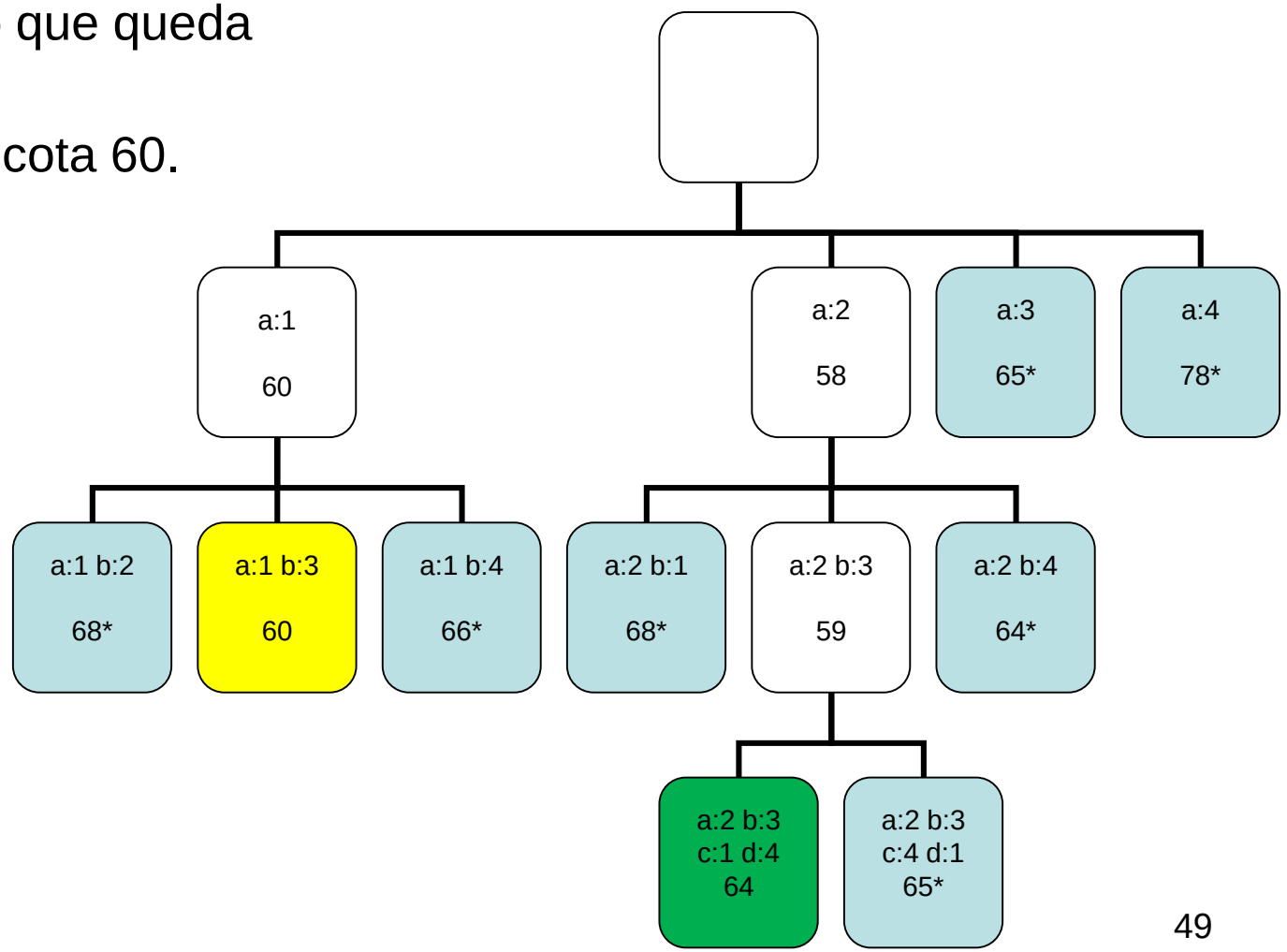
- Los nodos de cota 68 y 66 se podan.



Asignación De Tareas

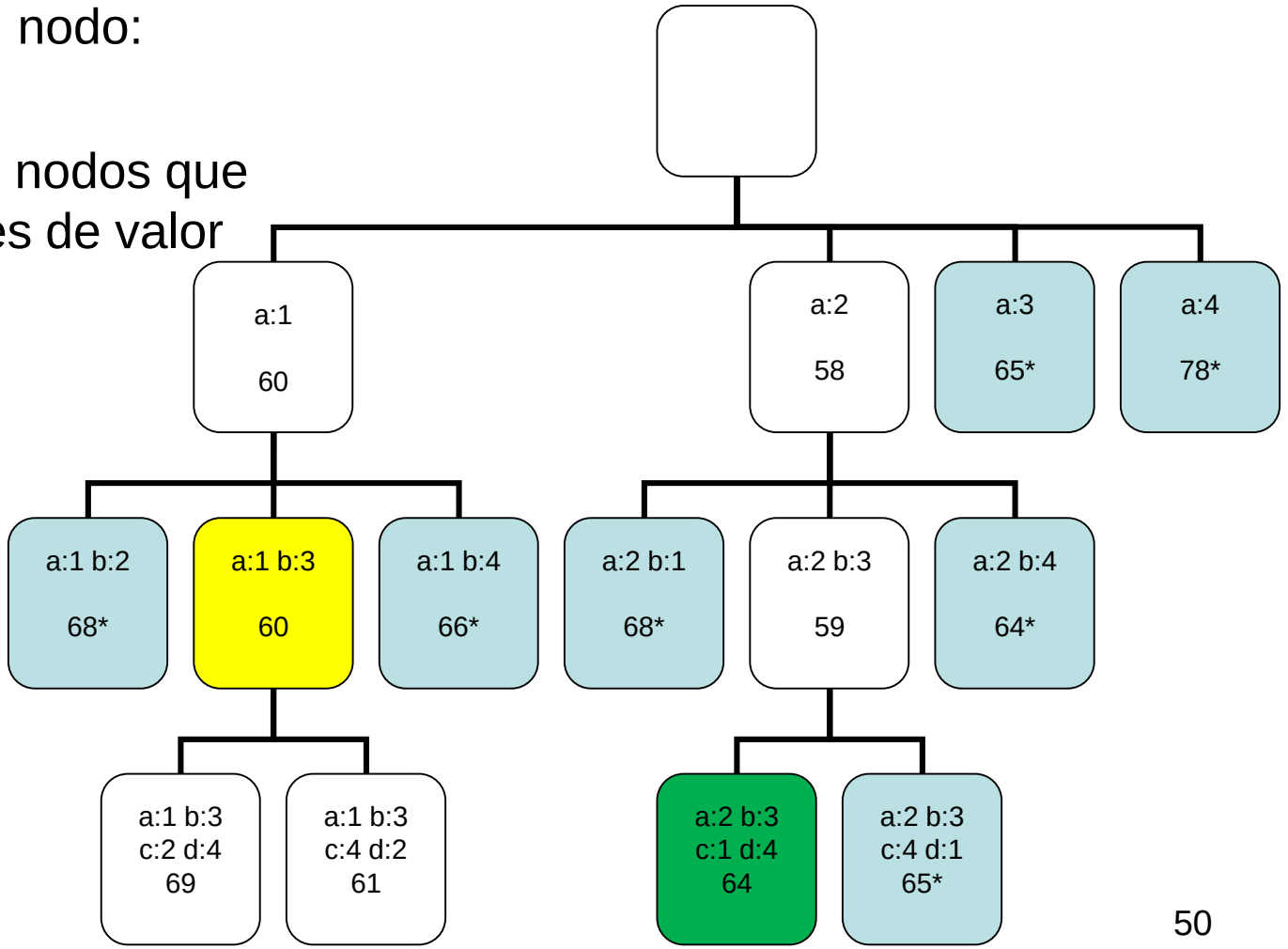
- El único nodo que queda explorar es:

  (a:1),(b:3) de cota 60.



Asignación De Tareas

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



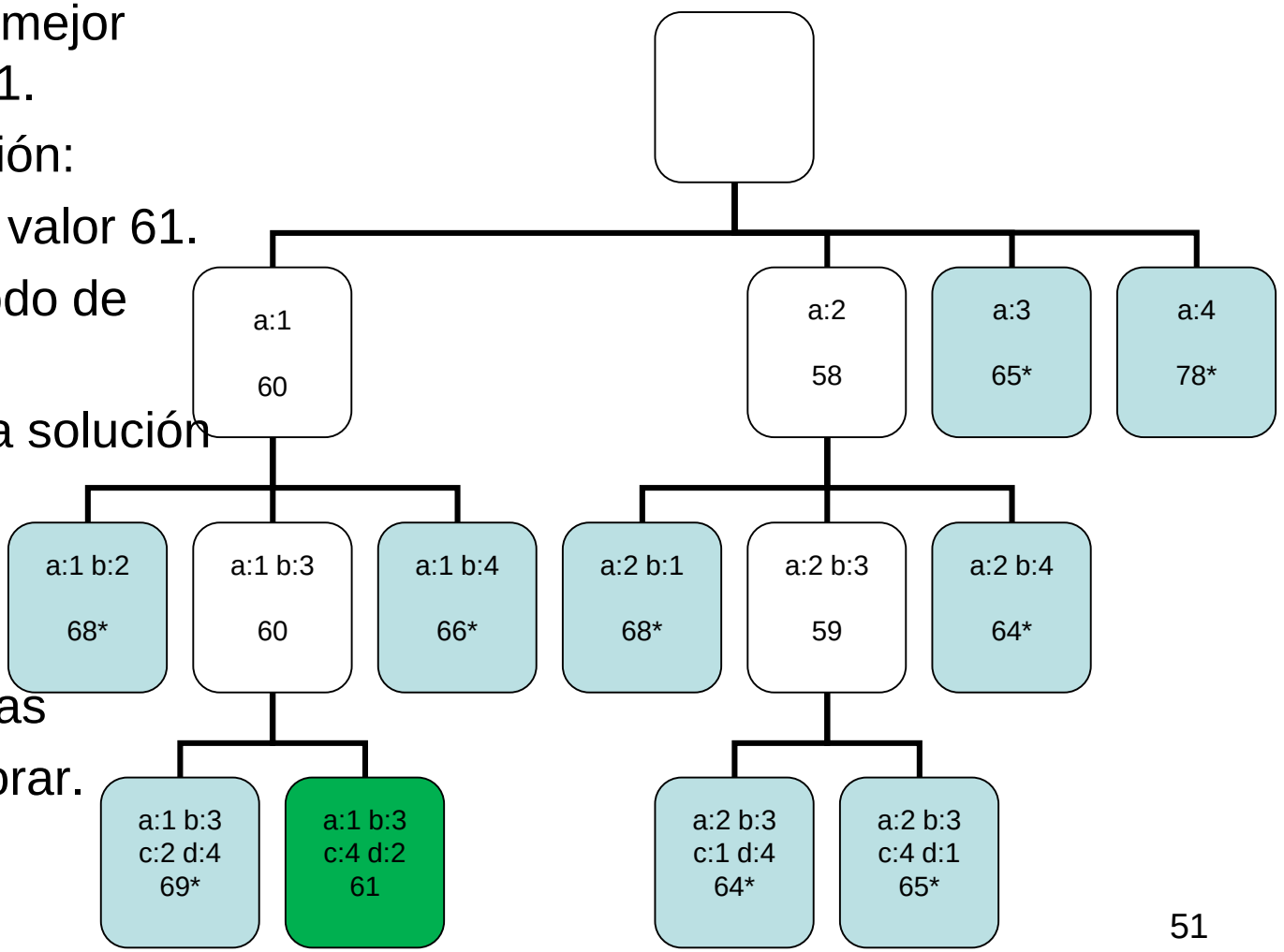
Asignación De Tareas

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

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

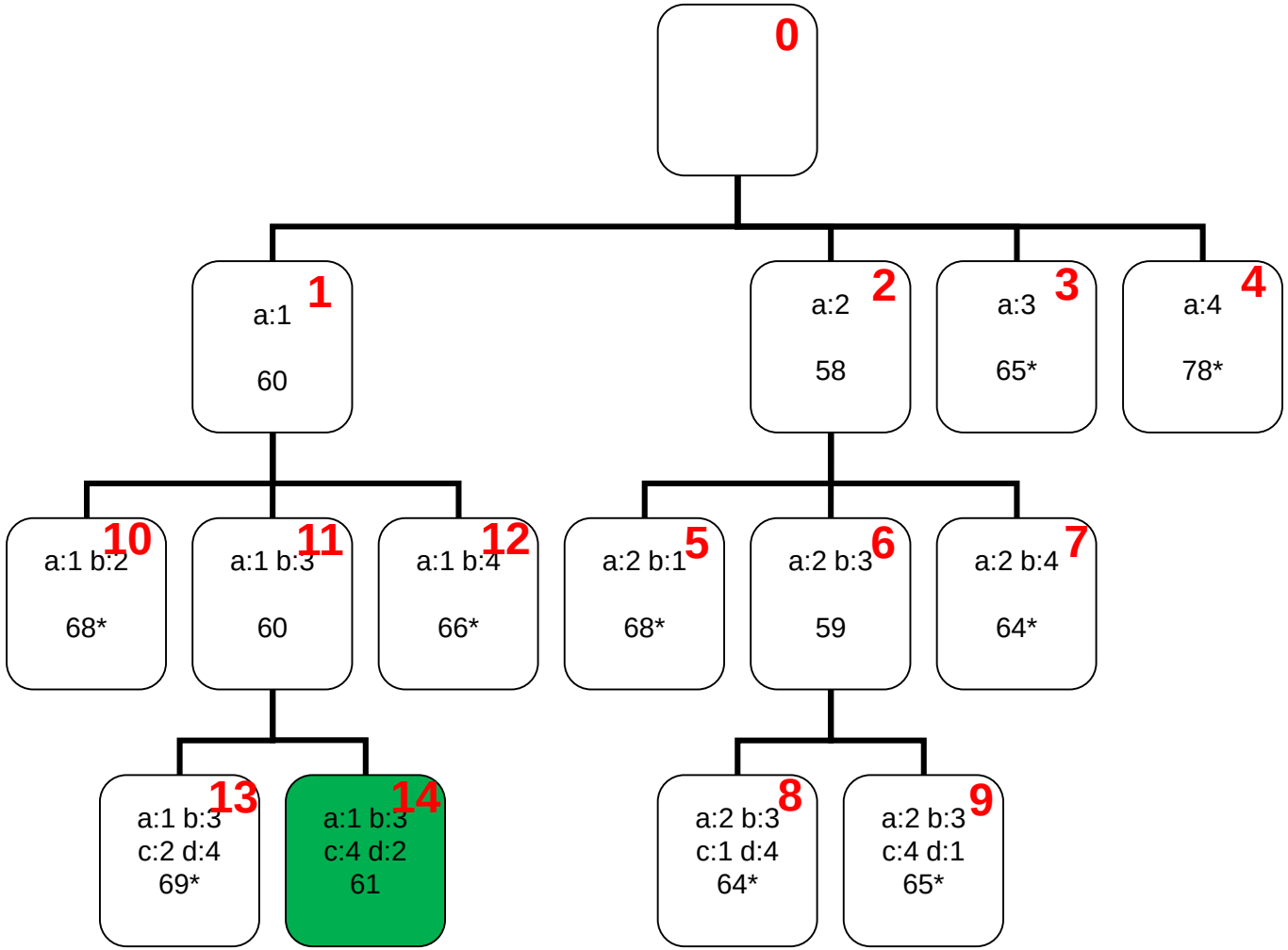
- Se poda el nodo de cota 69.
- Se descarta la solución de valor 64.
- No quedan mas

nodos para explorar.



Asignación De Tareas

- Árbol completo con el orden de generación de los nodos:



Asignación De Tareas

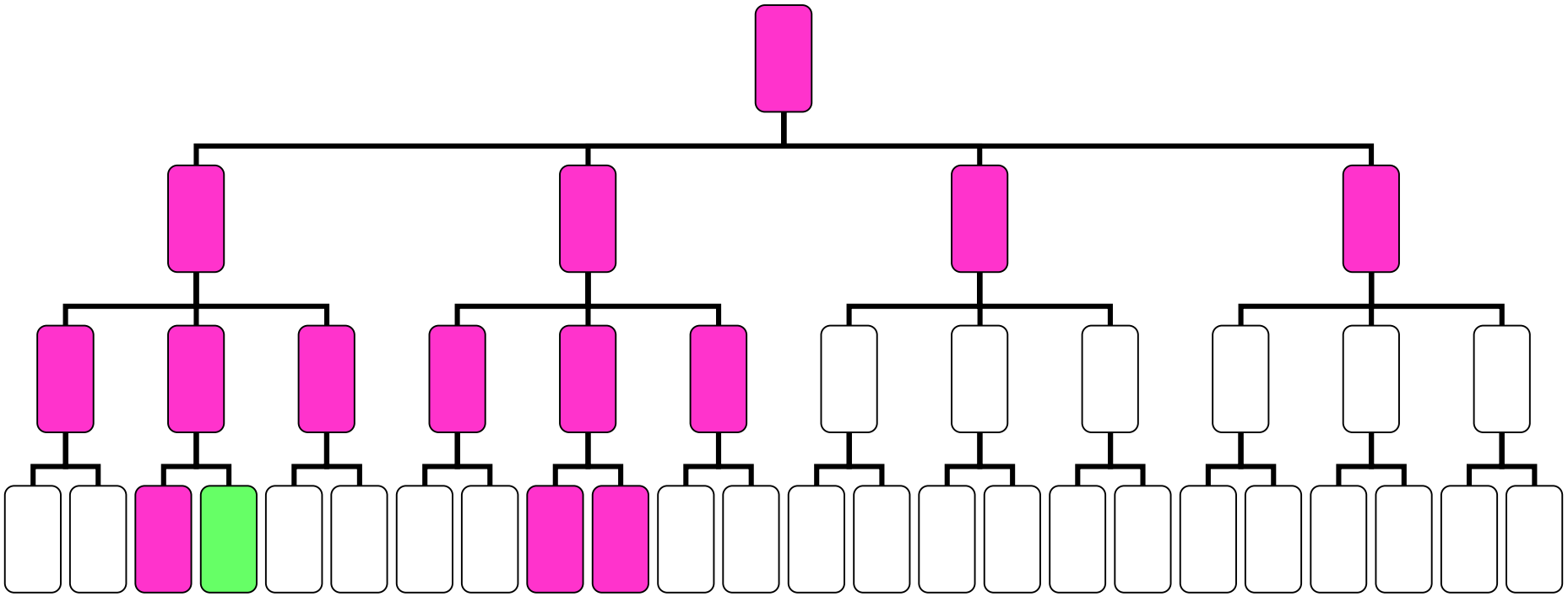
Conclusiones del ejemplo:

- Este ejemplo ilustra que a pesar que el nodo (a:2) era el más prometedor, la solución óptima no surgió de él.

- De los 41 nodos posibles, para obtener esta solución se han construido solamente 15 nodos (incluida la raíz).

- De las 24 soluciones posibles del problema, sólo 6 han sido examinadas (4 en el árbol y 2 al comienzo para determinar la cota superior). En el siguiente esquema se muestra esta situación.

Asignación De Tareas



Asignación De Tareas