BÚSQUEDAS CON RETROCESO, RAMIFICACIÓN Y ACOTAMIENTO

Aránzazu Jurío

ALGORITMIA

2018/2019

Índice

Introducción

- Problemas que no se pueden resolver con las técnicas estudiadas
- Búsqueda exhaustiva de soluciones. Asociar un árbol que represente el espacio de estados y hacer una búsqueda sobre él
- Cada nodo representa un estado del problema. Un nodo es solución cuando el camino desde la raíz hasta él define una tupla que pertenece al conjunto de las soluciones
- Los nodos son de dos tipos: nodos vivos (todavía no se han generado todos sus descendientes) y nodos muertos (en caso contrario)
- El orden a la hora de estudiar los nodos da lugar a dos técnicas: búsquedas con retroceso y ramificación y acotamiento

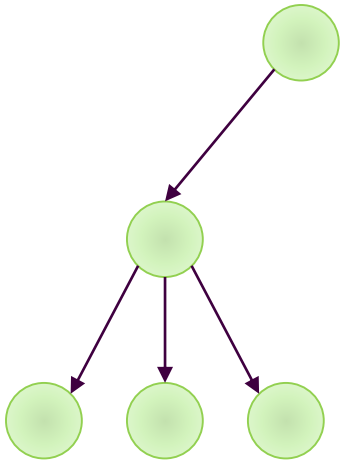
Índice

- Introducción
- Búsquedas con retroceso (backtracking)
  - Problema de las n reinas
  - Problema de la mochila 0-1
- Ramificación y acotamiento (branch and bound)
  - Problema de la mochila 0-1
  - Problema del viajante

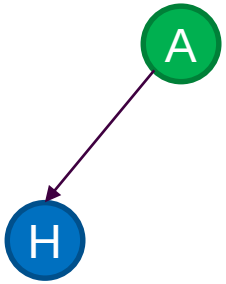
Backtracking

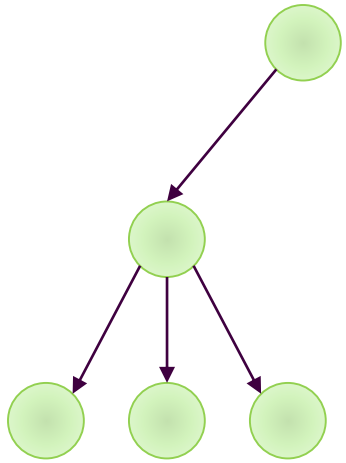
- En los algoritmos de búsqueda con retroceso (Backtracking), los nodos del árbol del espacio de estados se generan de la siguiente forma: si A es el nodo actual (el que se está extendiendo en un determinado momento), cada vez que se genere un nuevo hijo suyo H, éste pasará a ser el nodo actual y únicamente volverá a ser nodo actual A cuando el subárbol generado por H haya sido completamente explorado

Backtracking

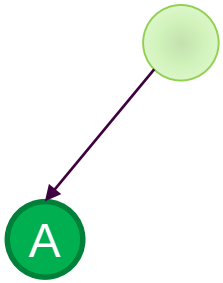


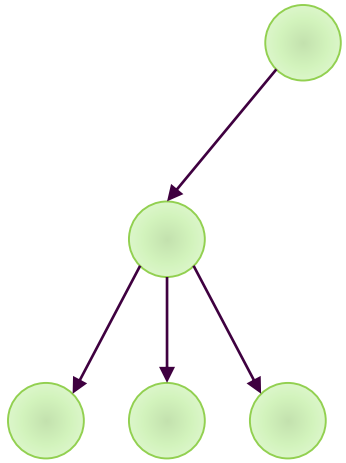
Backtracking



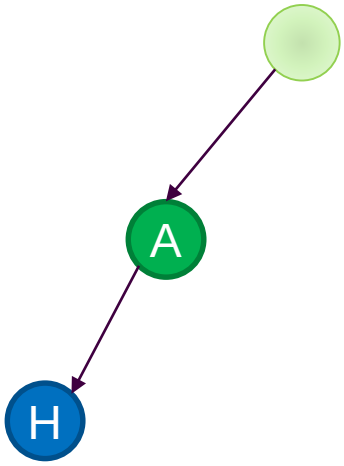


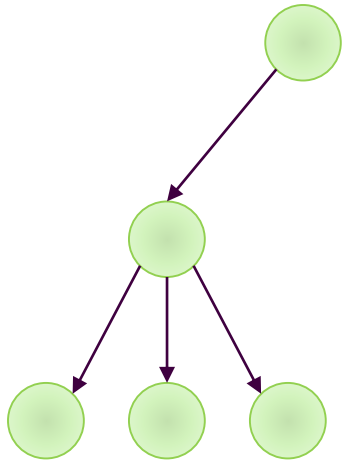
Backtracking



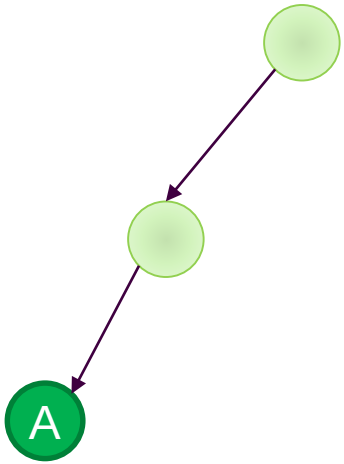


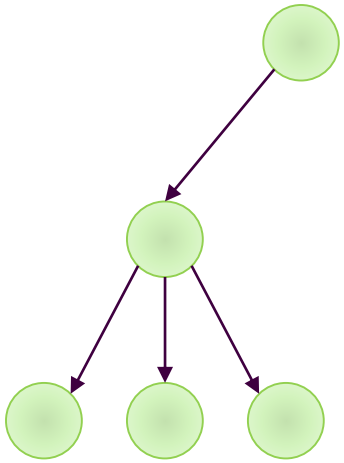
Backtracking



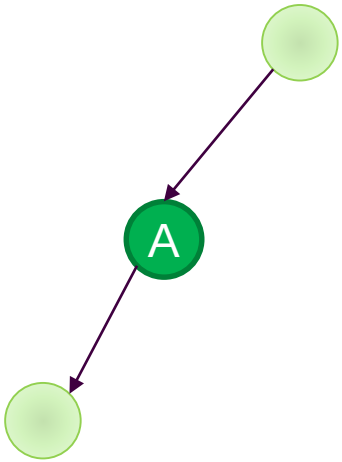


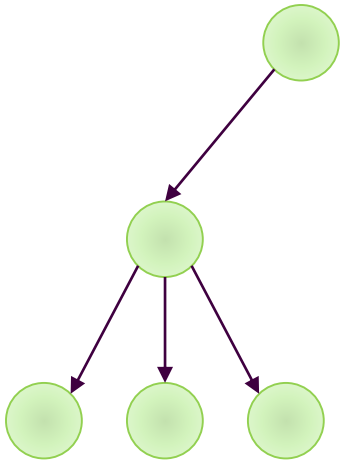
Backtracking



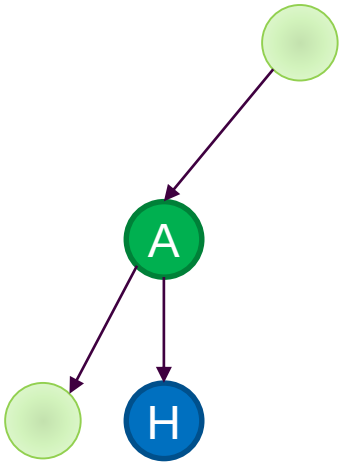


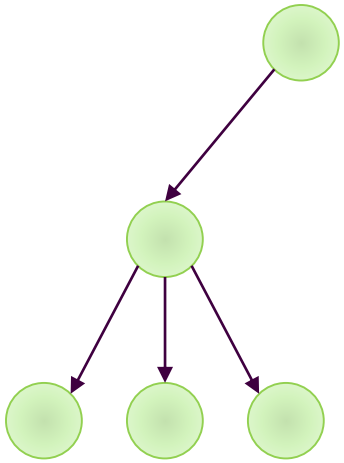
Backtracking



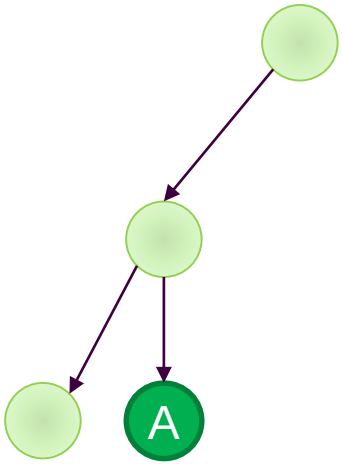


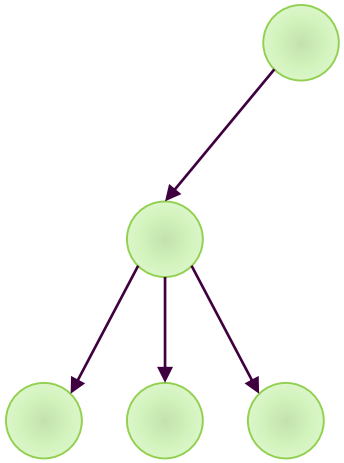
Backtracking



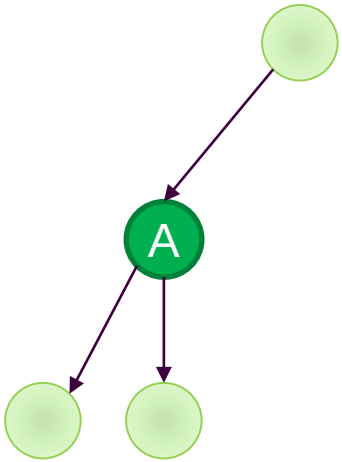


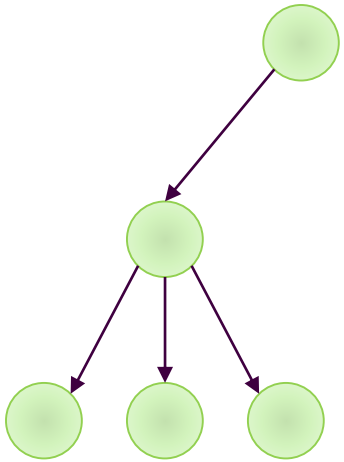
Backtracking



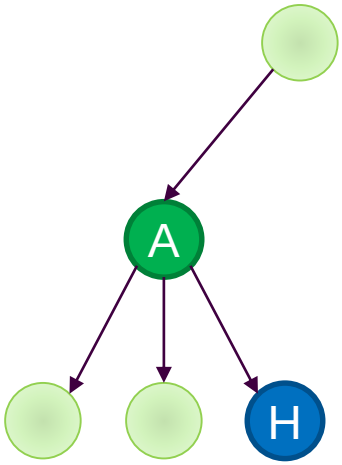


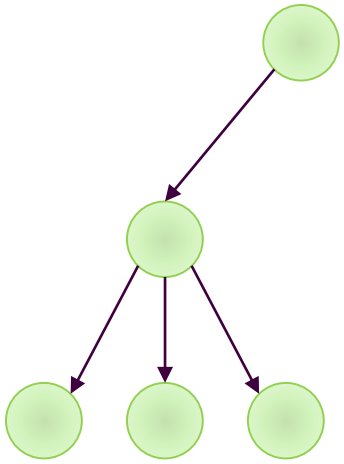
Backtracking



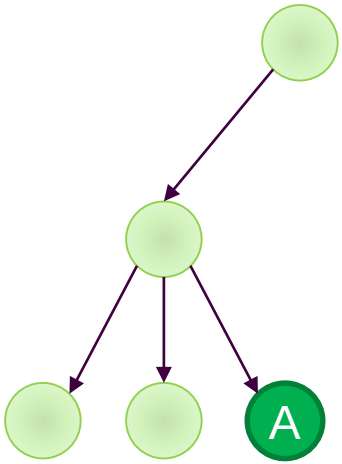


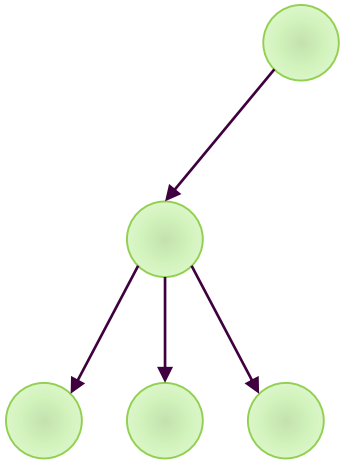
Backtracking



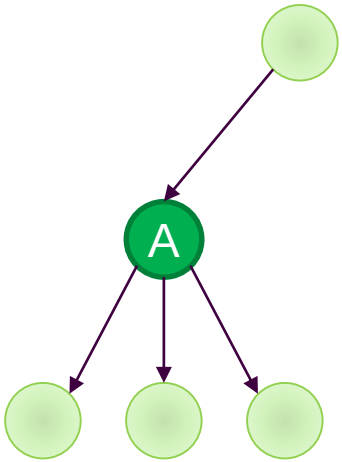


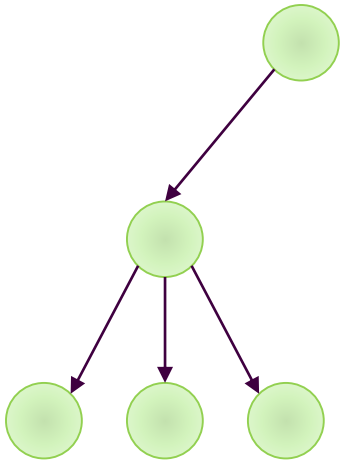
Backtracking





Backtracking



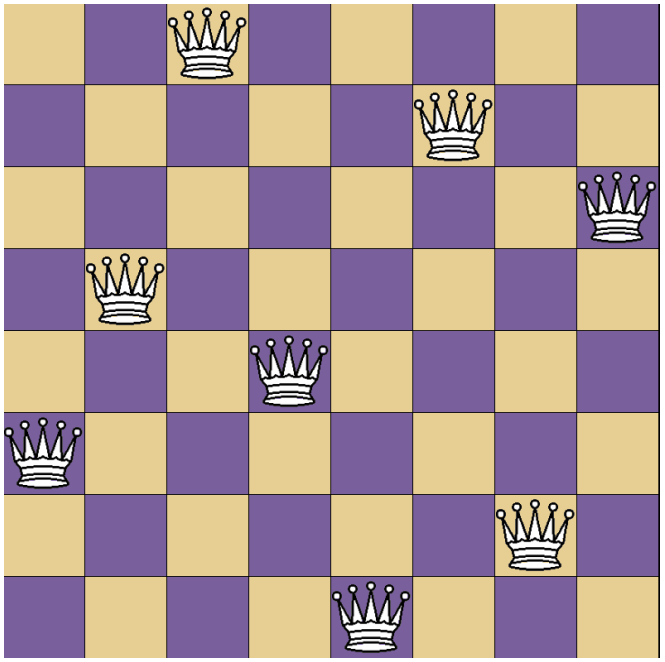


Backtracking

Índice

Problema de las n reinas

- Colocar 8 reinas en un tablero de ajedrez sin que se den jaque (dos reinas se dan jaque si están situadas en la misma fila, columna o diagonal del tablero)



Problema de las n reinas

- Calcular número de estados
  - Variaciones sin repetición de 64 elementos tomados de 8 en 8

$$V_{64,8} = \frac{64!}{(64-8)!} = 178.462.987.637.760$$

  - Combinaciones sin repetición de 64 elementos tomados de 8 en 8

$$C_{64,8} = \binom{64}{8} = \frac{64!}{8!\,(64-8)!} = 4.426.165.368$$

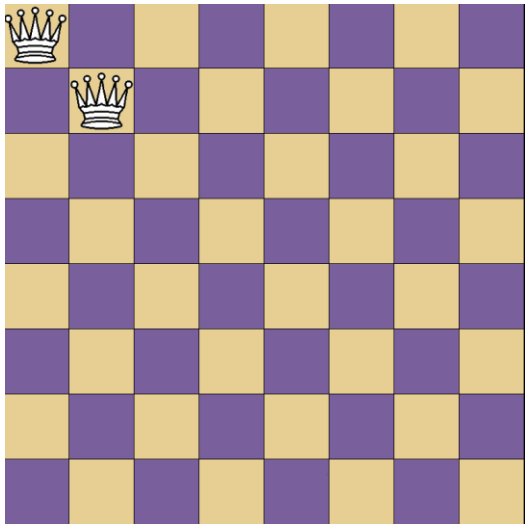
Problema de las n reinas

- Calcular número de estados
  - En los estados solución, sólo hay una reina en cada fila (puedo representar como una tupla de 8 elementos) y una reina en cada columna (no hay elementos repetidos en la tupla)
  - Permutaciones sin repetición de 8 elementos

$$P_8 = 8! = 40320$$

Problema de las n reinas

- Hemos conseguido reducir el número de posibles estados
- Pero no se verifica ningún estado hasta que todas las reinas han sido colocadas.
  - Por ejemplo, después de colocar dos reinas que se dan jaque, se verifican inútilmente 720 formas de colocar las otras 6 reinas



Problema de las n reinas

- Planteamos un algoritmo de búsqueda con retroceso
- Sólo expandimos aquellos nodos que representen estados completables (prometedores)
  - Un estado es completable cuando las reinas colocadas hasta ahora no se dan jaque entre ellas
- El árbol de estados tiene 2057 nodos. Encontramos la primera solución en el nodo 114

Problema de las 4 reinas

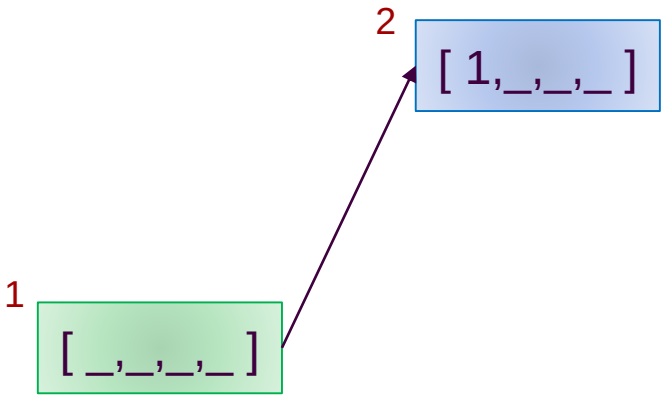
- Empezamos con el nodo raíz, en el que todavía no hemos situado ninguna reina, por lo que tenemos una tupla vacía
- El primer descendiente consiste en asignar al primer número de la tupla la posición 1
- En este momento, ese descendiente pasa a ser el nodo actual
- …

Problema de las 4 reinas

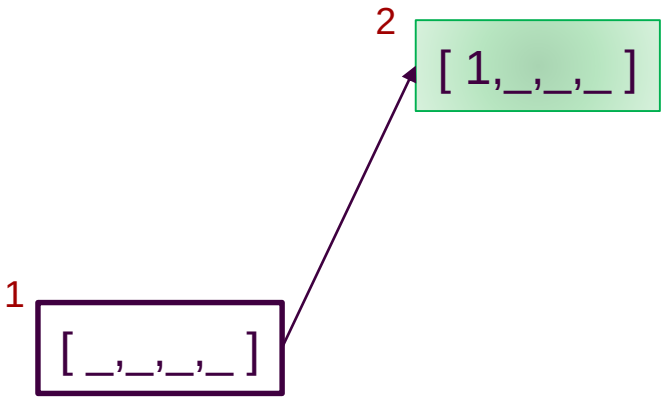
1

[_,_,_,_]

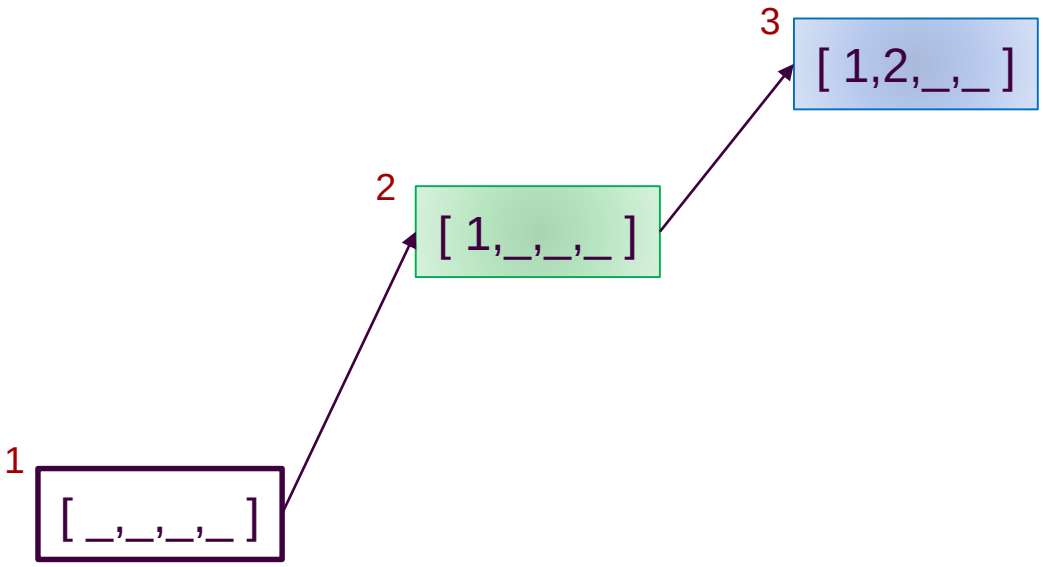
Problema de las 4 reinas



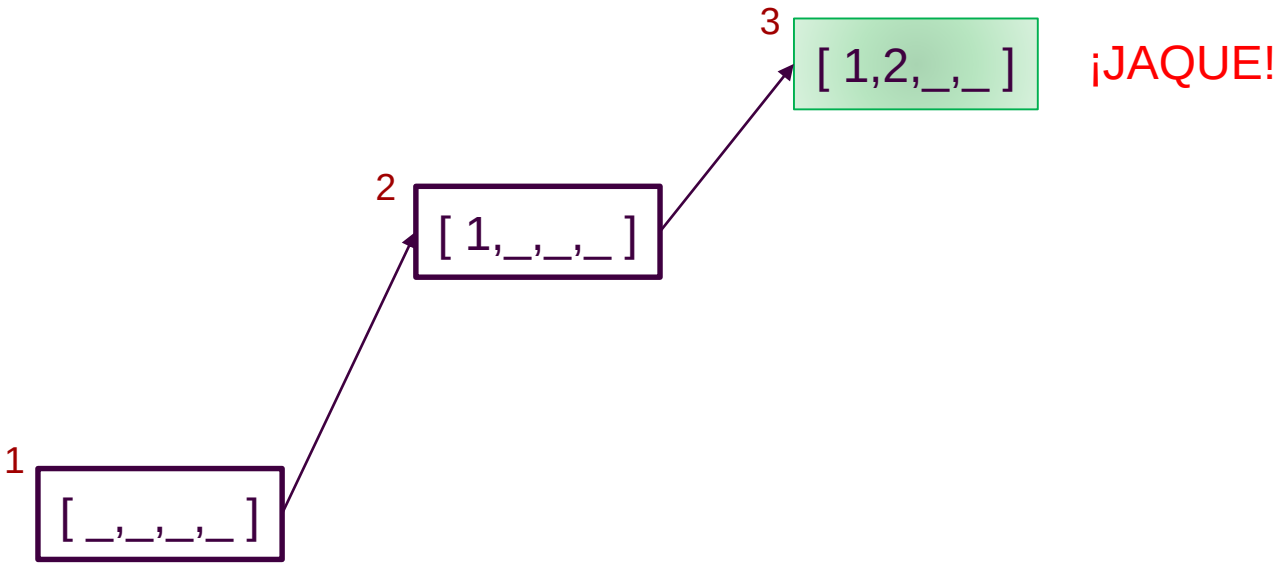
Problema de las 4 reinas



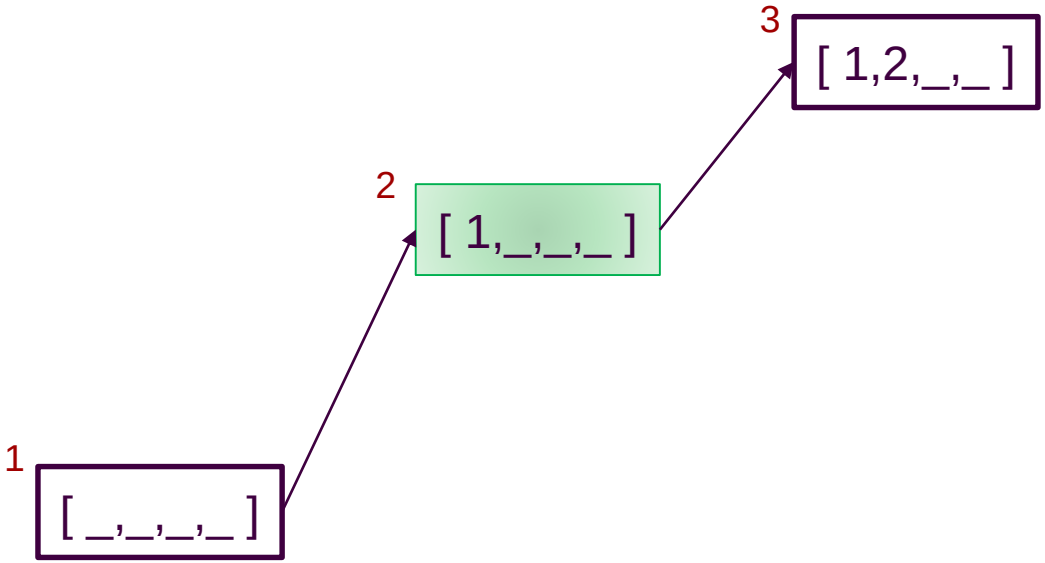
Problema de las 4 reinas



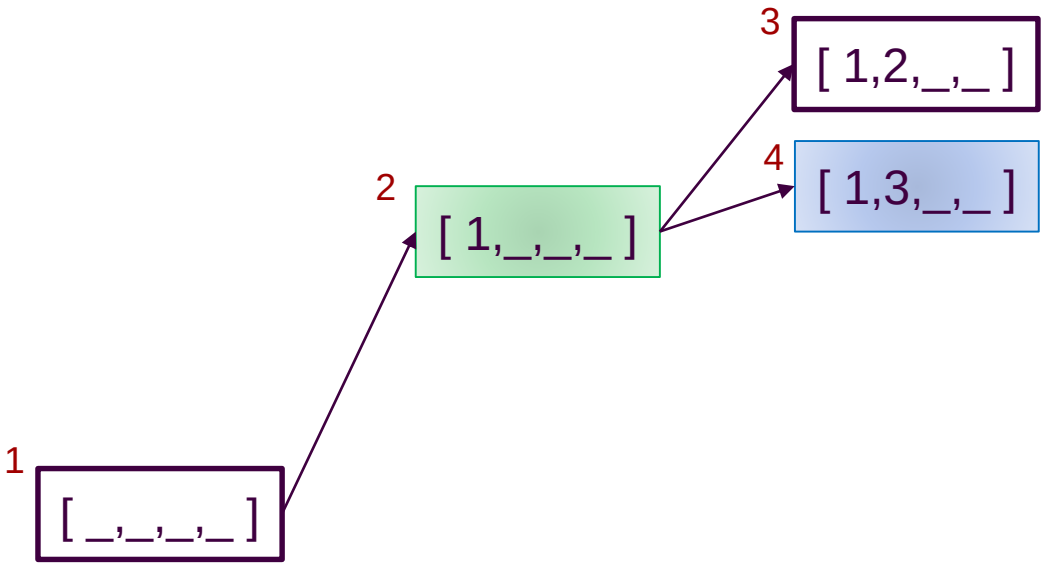
Problema de las 4 reinas



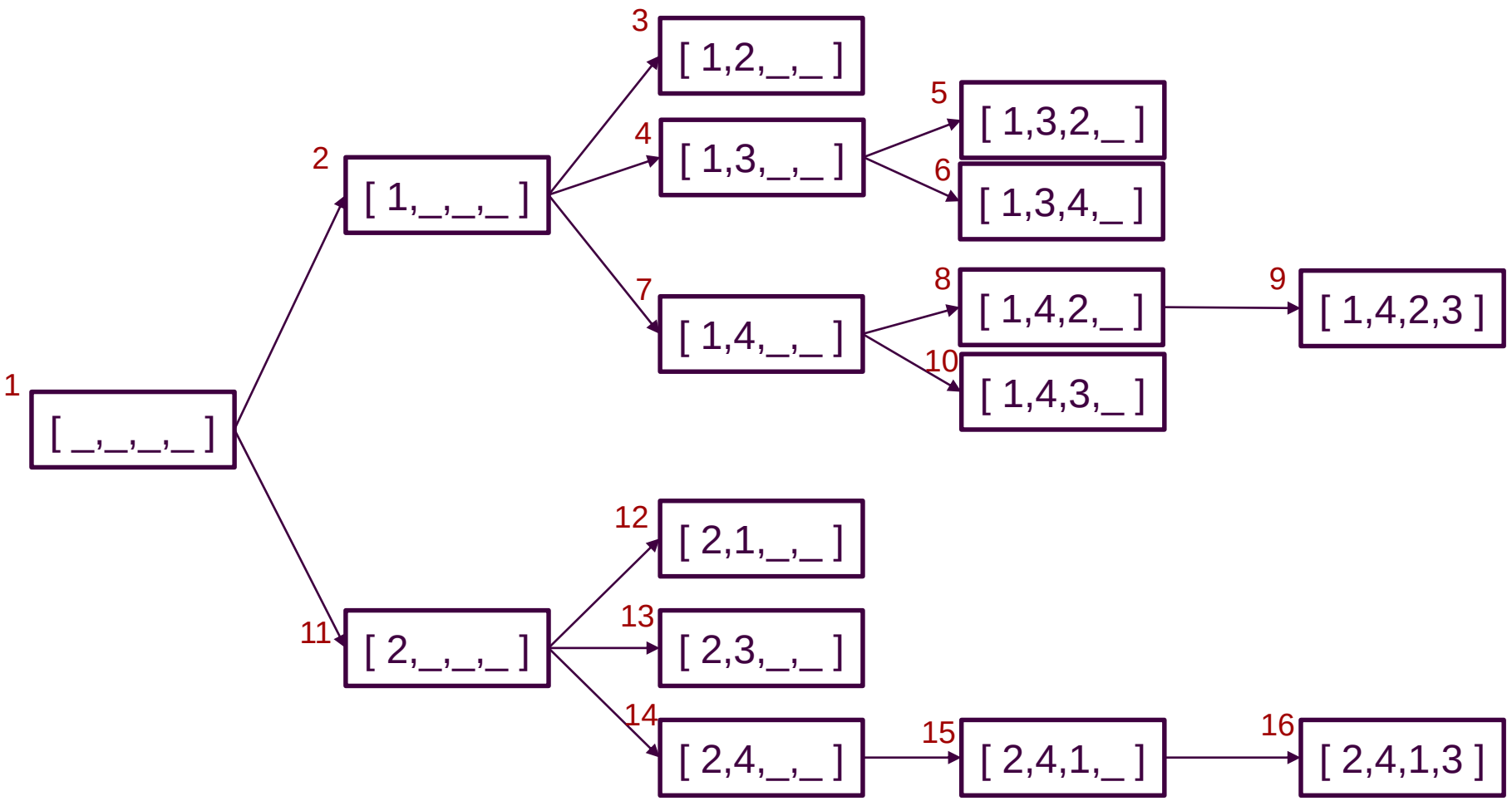
Problema de las 4 reinas



Problema de las 4 reinas



Problema de las 4 reinas



Índice

Problema de la mochila 0-1

- Dados n elementos $e_1, e_2, \ldots, e_n$ con pesos $p_1, p_2, \ldots, p_n$ y beneficios $b_1, b_2, \ldots, b_n$, y dada una mochila con capacidad de albergar hasta un máximo de peso M, queremos encontrar aquellos elementos que tenemos que introducir en la mochila de forma que la suma de los beneficios de los elementos escogidos sea máxima

Problema de la mochila 0-1

- Número de estados: $2^n - 1$
- ¿Cómo podemos mejorar el árbol para no estudiar todos estos estados?

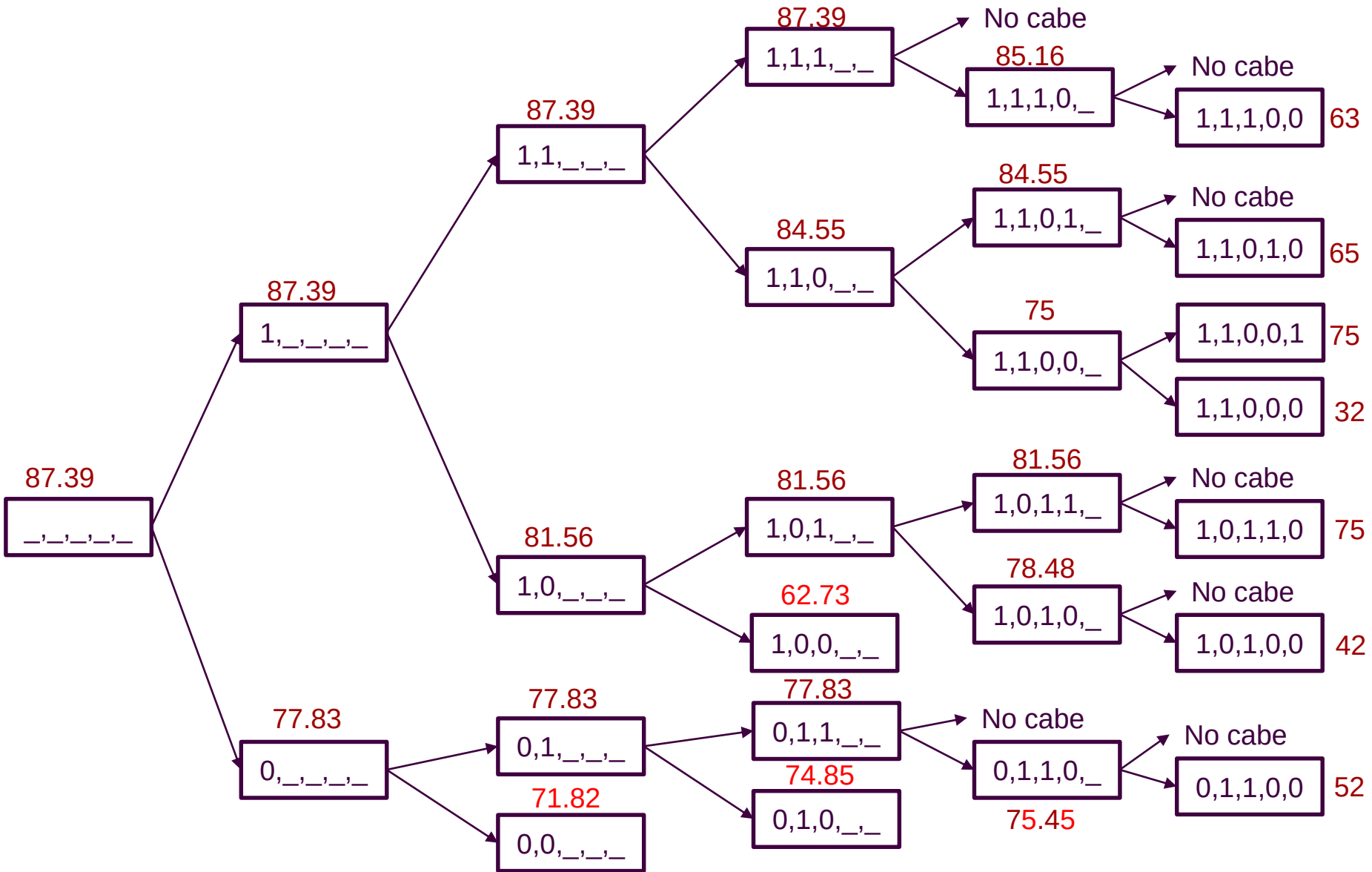
Problema de la mochila 0-1

- Número de estados: $2^n - 1$
- ¿Cómo podemos mejorar el árbol para no estudiar todos estos estados?
  - Función cota: si el mejor resultado que podemos obtener como descendiente de un nodo es peor que un resultado ya obtenido, no seguimos explorando ese hijo
  - Para saber la cota superior de beneficio utilizamos el algoritmo voraz relajando la condición de que los objetos no se pueden introducir fraccionados
  - Por facilidad, ordenamos los objetos en orden decreciente de beneficio por peso, de tal forma que la cota de introducir el siguiente objeto sea igual que la cota actual (padre)

Problema de la mochila 0-1

- Ejemplo
  - Capacidad: 50
  - Beneficios: (11 21 31 33 43)
  - Pesos: (1 11 21 23 33)

Problema de la mochila 0-1



Problema de la mochila 0-1

- Ejercicio
  - Calcular el beneficio máximo del problema de la mochila 0-1 con los siguientes datos, mediante la técnica de backtracking
    - Capacidad: 100
    - Objeto 1
      - Peso=45, beneficio=170
    - Objeto 2
      - Peso=20, beneficio=100
    - Objeto 3
      - Peso=100, beneficio=400
    - Objeto 4
      - Peso=50, beneficio=350

Índice

- Introducción
- Búsquedas con retroceso (backtracking)
  - Problema de las n reinas
  - Problema de la mochila 0-1
- Ramificación y acotamiento (branch and bound)
  - Problema de la mochila 0-1
  - Problema del viajante

Branch and bound

- Al igual que el backtracking, realiza una enumeración parcial del espacio de soluciones
- Se generan todos los hijos de un nodo, de manera que este nodo seguirá siendo el nodo actual hasta que muera
- De entre los nodos vivos, elegimos como nodo actual aquel que tenga una cota más prometedora

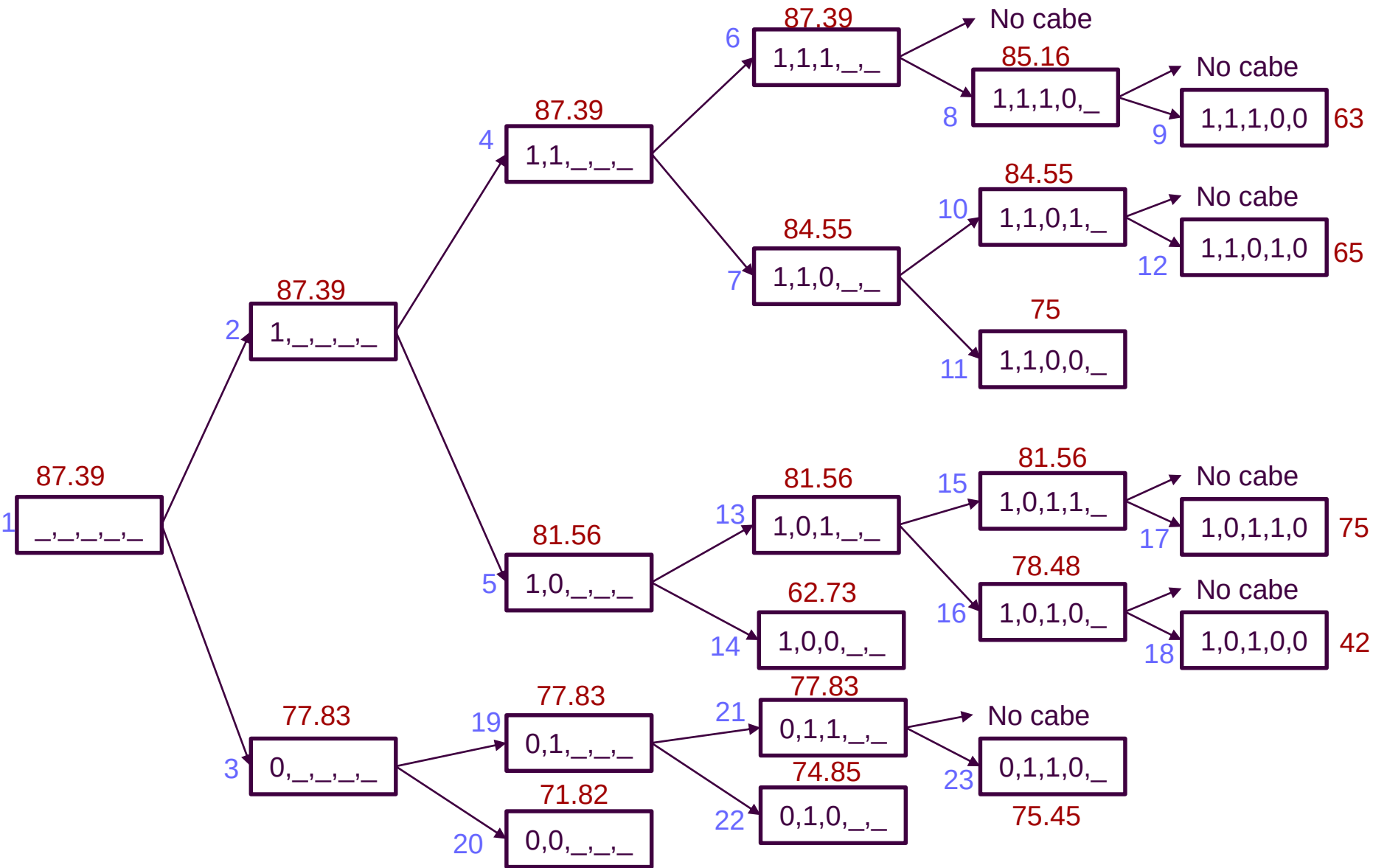
Índice

- Introducción
- Búsquedas con retroceso (backtracking)
  - Problema de las n reinas
  - Problema de la mochila 0-1
- Ramificación y acotamiento (branch and bound)
  - Problema de la mochila 0-1
  - Problema del viajante

Problema de la mochila 0-1

- Utilizamos como cota el beneficio máximo que se puede obtener si fraccionamos los objetos (algoritmos voraces)
- Ejemplo:
  - Capacidad: 50
  - Objeto 1
    - Peso = 1, beneficio = 11
  - Objeto 2
    - Peso = 11, beneficio = 21
  - Objeto 3
    - Peso = 21, beneficio = 31
  - Objeto 4
    - Peso = 23, beneficio = 33
  - Objeto 5
    - Peso = 33, beneficio = 43

Problema de la mochila 0-1



Índice

Problema del viajante

- Un viajante tiene que recorrer una serie de ciudades, partiendo de una de ellas, regresando a la misma e invirtiendo un tiempo mínimo
- Mediante técnicas de programación dinámica este problema se puede solucionar con una complejidad $\mathrm{O}(n^2 2^n)$
- Si se utilizan buenas funciones de acotación, se pueden resolver algunos problemas concretos utilizando mucho menos tiempo que mediante programación dinámica

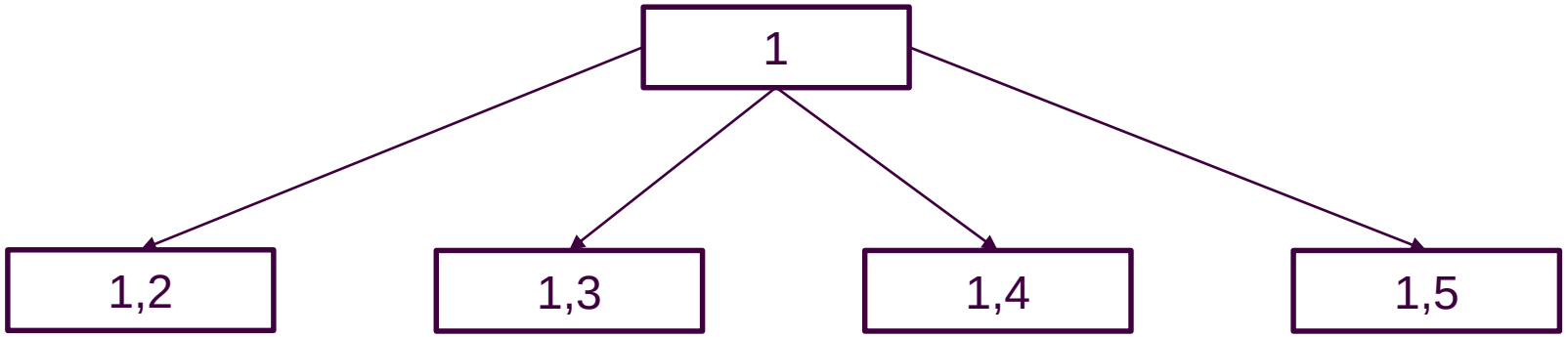
Problema del viajante

- Tenemos 5 ciudades con las siguientes distancias

$$\begin{pmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{pmatrix}$$

- Comenzamos el recorrido en la ciudad 1 y tenemos que volver hasta ella

Problema del viajante



Problema del viajante

- Necesito una cota para ver qué nodo es más prometedor, y seguir estudiando ese
- Además, la cota me ayuda a podar aquellas ramas cuya solución no va a ser mejor que una ya encontrada

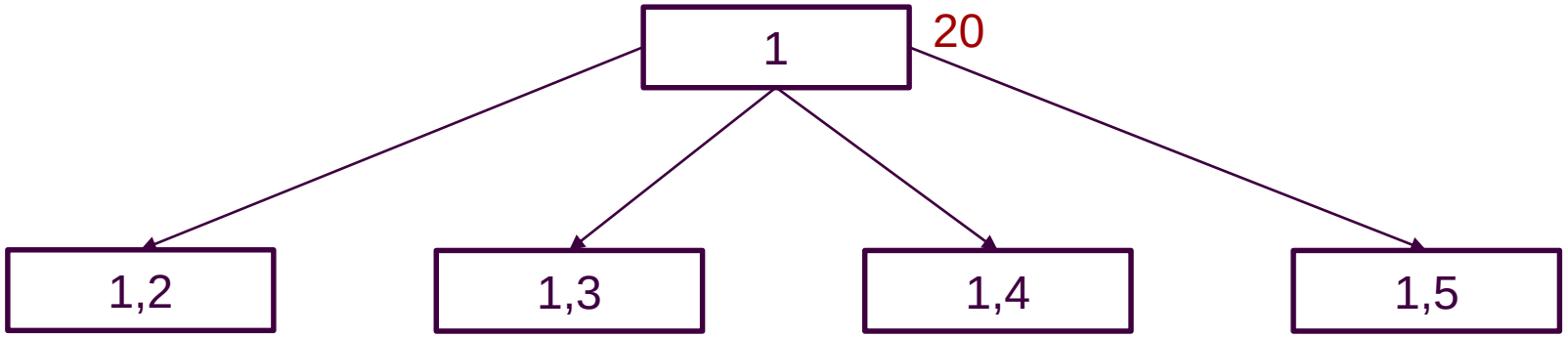
Problema del viajante

- Necesito una cota para ver qué nodo es más prometedor, y seguir estudiando ese
- Además, la cota me ayuda a podar aquellas ramas cuya solución no va a ser mejor que una ya encontrada
- Consideramos que la mitad de la distancia entre los vértices i y j corresponde a salir del vértice i y la otra mitad a llegar al vértice j
- ¿Cuál es la cota inferior del coste de salir del vértice 1, pasar una vez por el resto de vértices y volver al vértice 1?

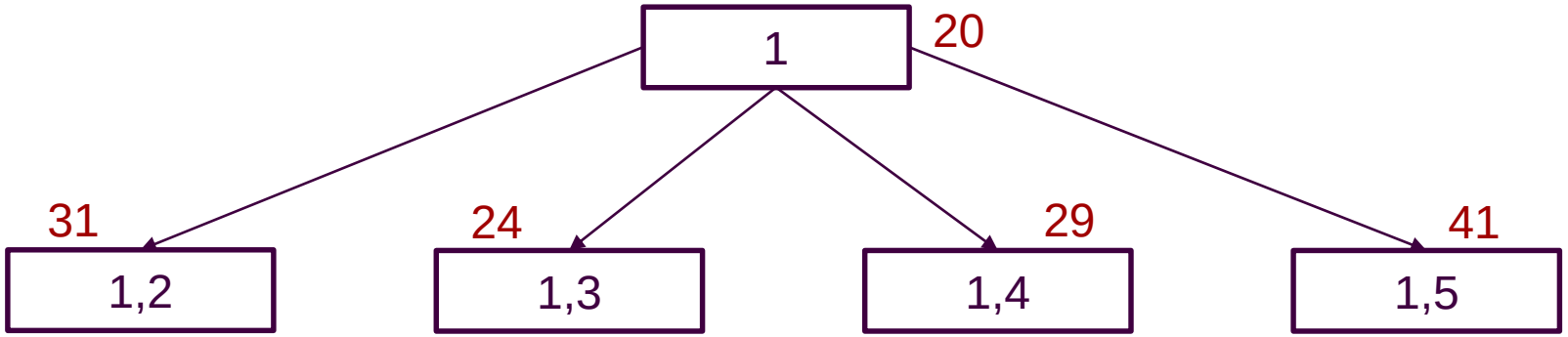
Problema del viajante

- $\begin{pmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{pmatrix}$
- Salir del vértice 1
  - Min(14/2, 4/2,10/2, 20/2) = 2
- Pasar por el vértice 2
  - Llegar: min(14/2, 5/2, 7/2, 7/2) = 2.5
  - Salir: min(14/2, 7/2, 8/2, 7/2) = 3.5
- …
- Cota inferior: 2 + 6 + 4 + 3 + 3 + 2 = 20

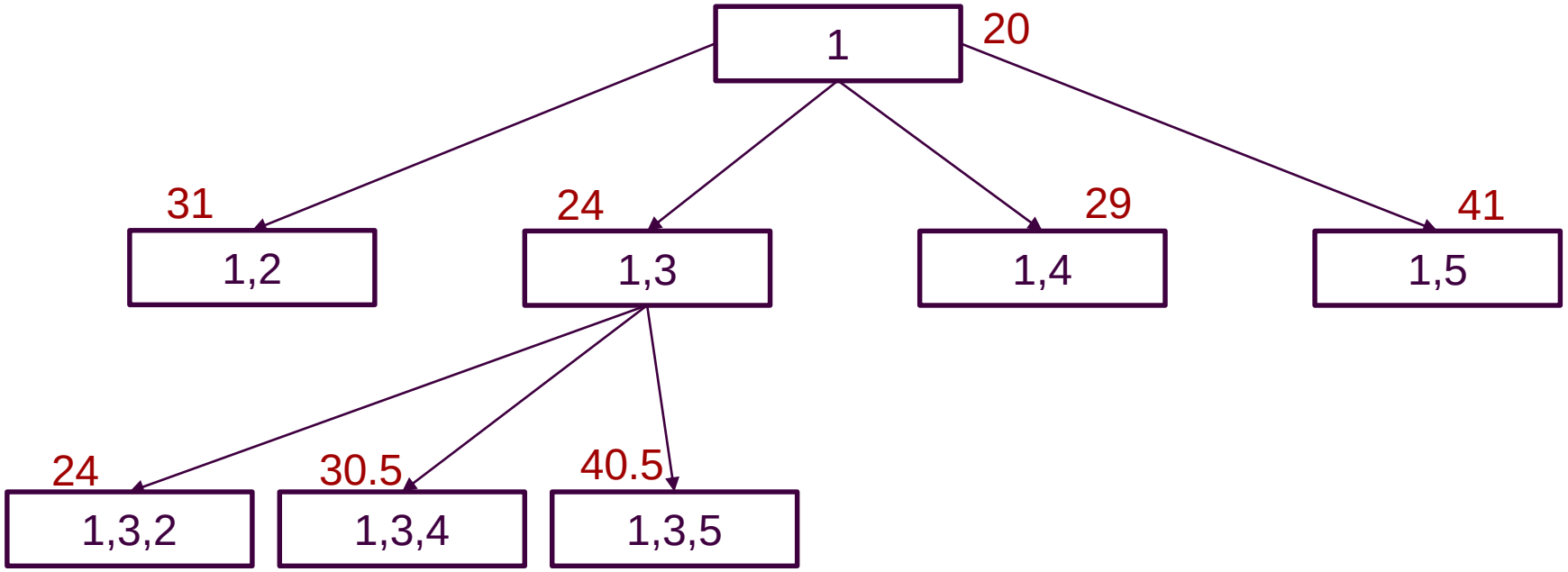
Problema del viajante



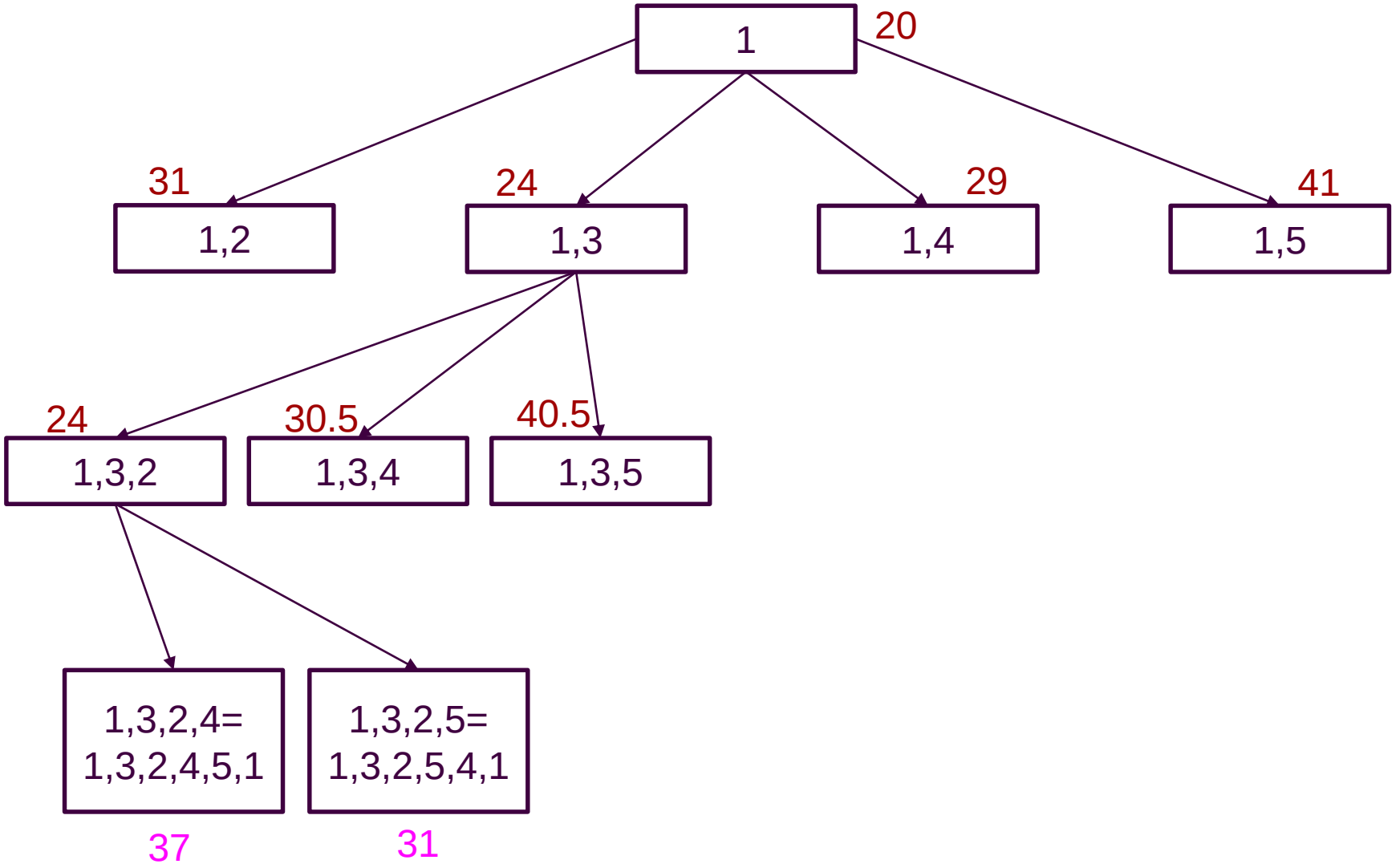
Problema del viajante



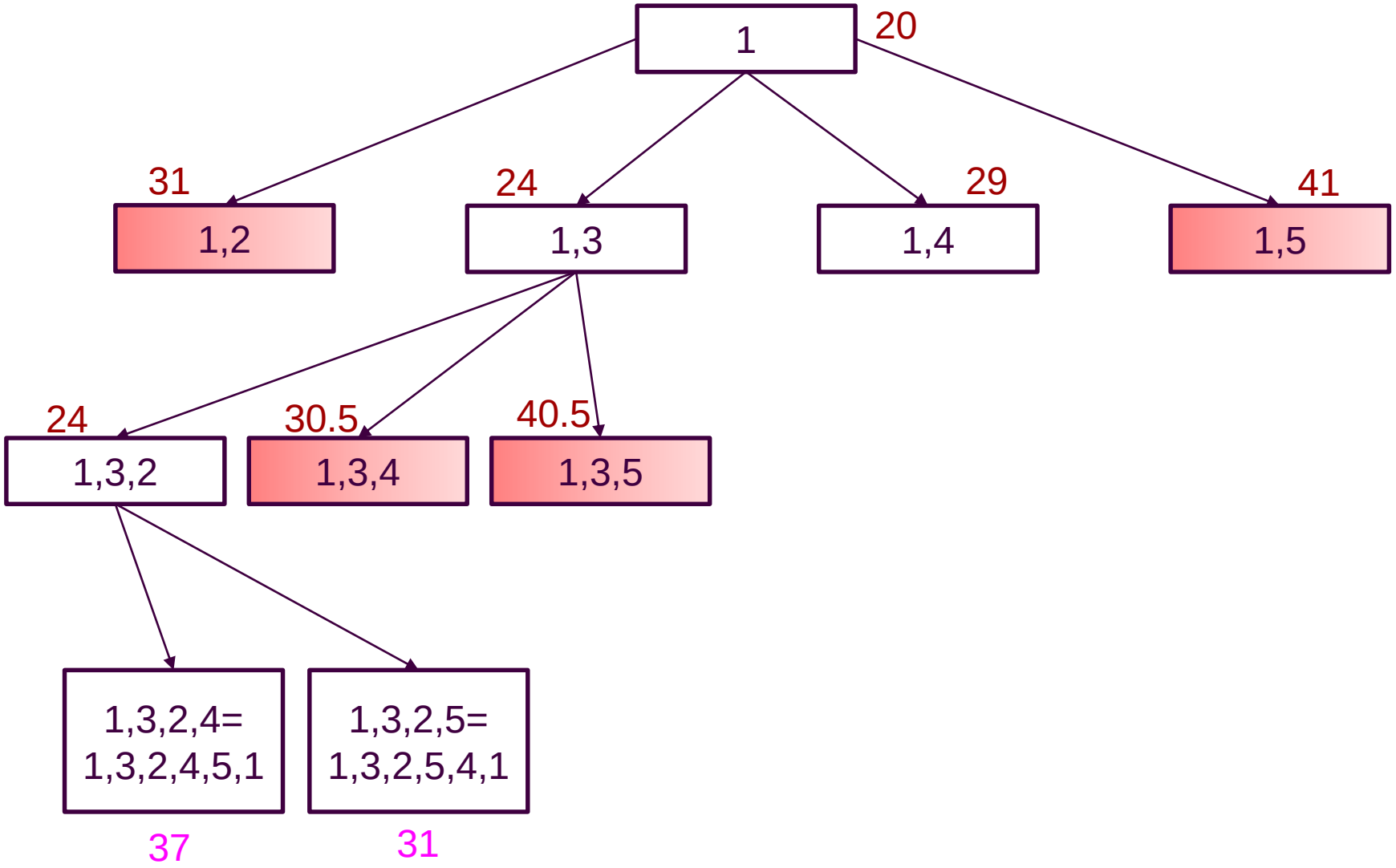
Problema del viajante



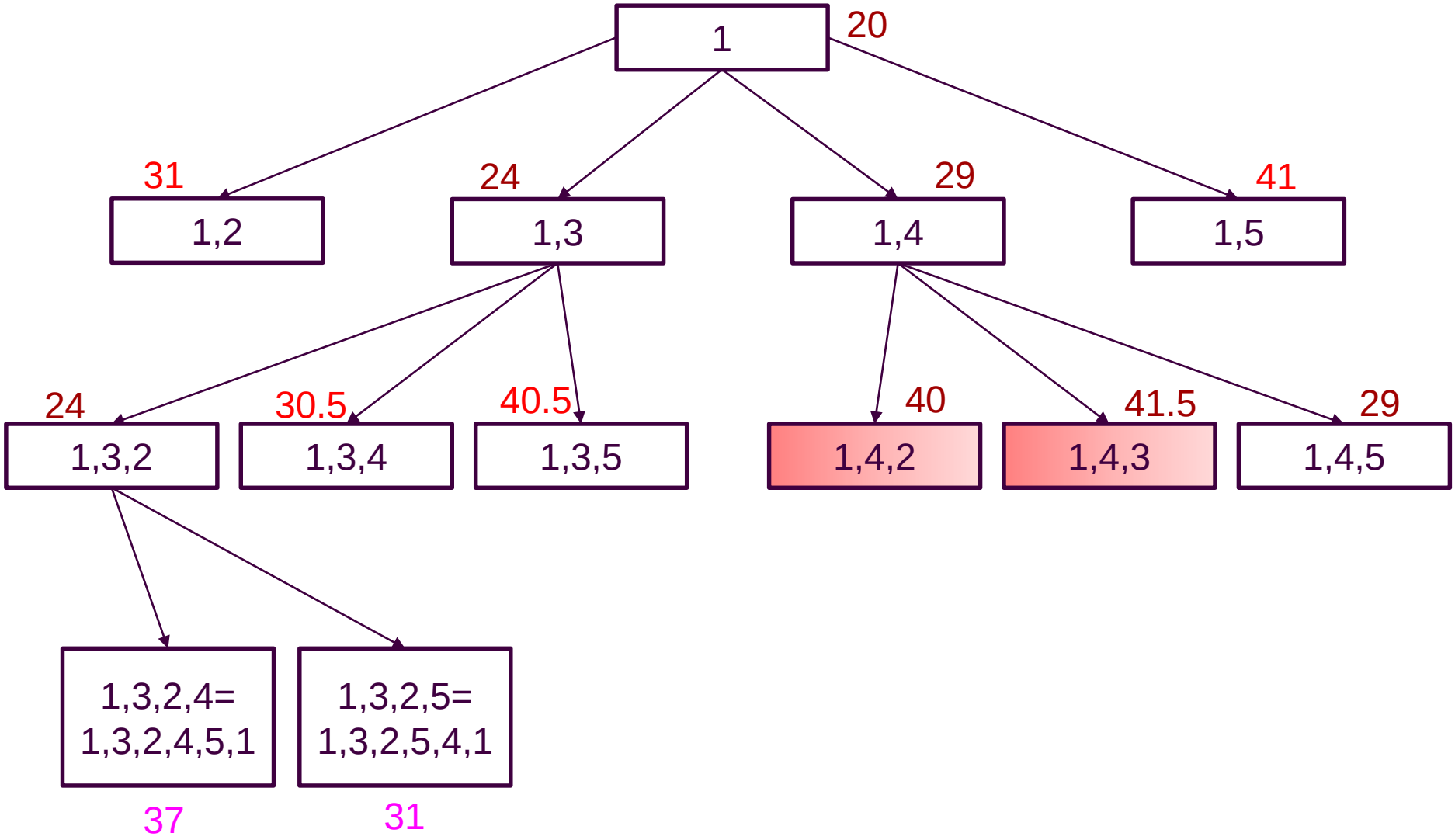
Problema del viajante



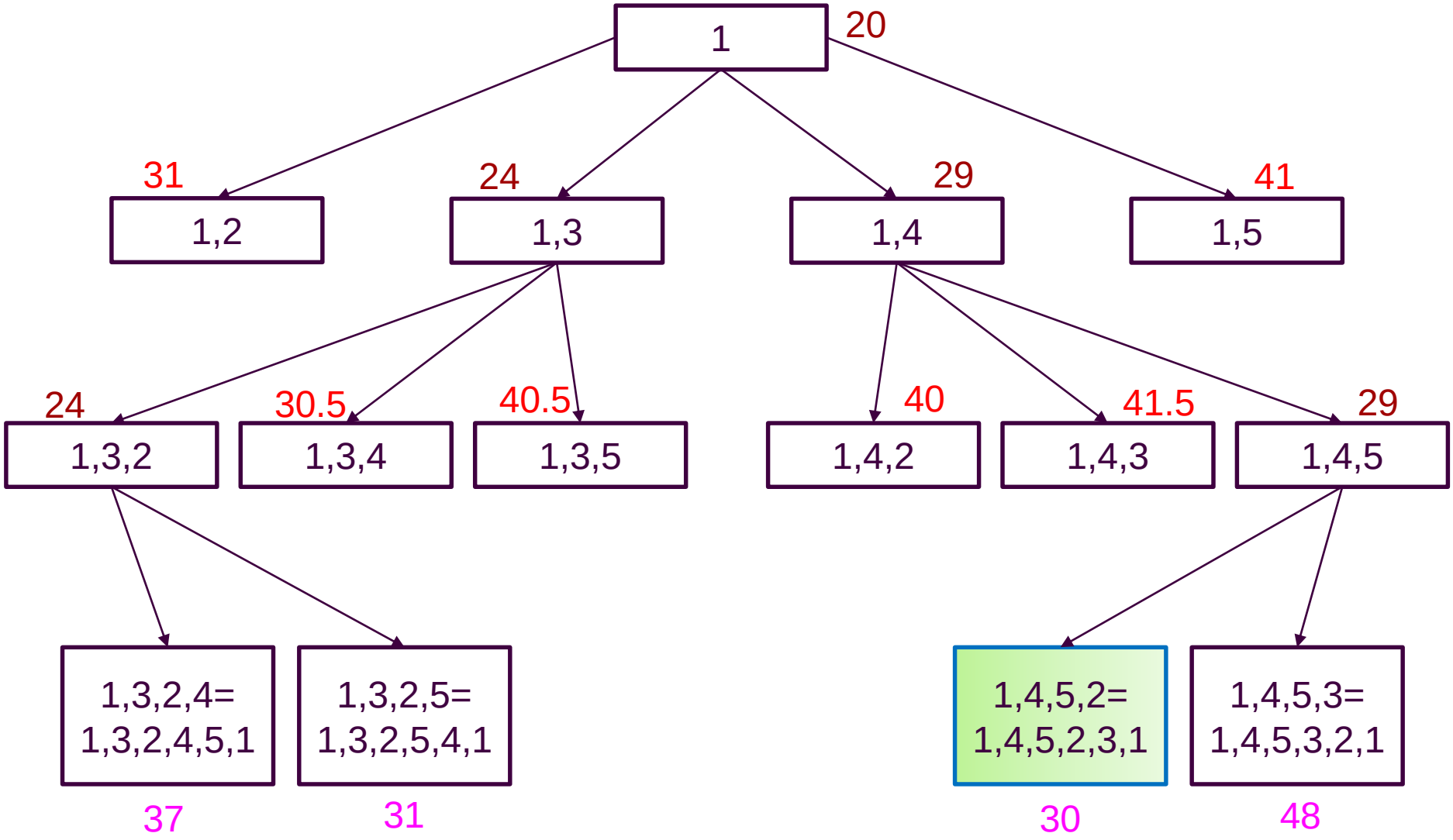
Problema del viajante



Problema del viajante



Problema del viajante



Problema del viajante

- Para llegar a la solución óptima sólo hemos explorado 15 de los 41 nodos del árbol
- Es más complicada de programar, porque hay que mantener una lista actualizada de nodos generados pero aún no explorados en diferentes niveles del árbol