Tema 2.4 Búsqueda con adversario

Tema 2. Resolución de problemas con búsqueda

- Representación de problemas y espacio de estados
- Búsqueda de soluciones
  - Búsqueda ciega
  - Búsqueda heurística
    - **Búsqueda con adversario**
    - Aprendizaje de heurísticas
  - Planificación
  - Búsqueda local
- Aplicaciones

Búsqueda con adversario

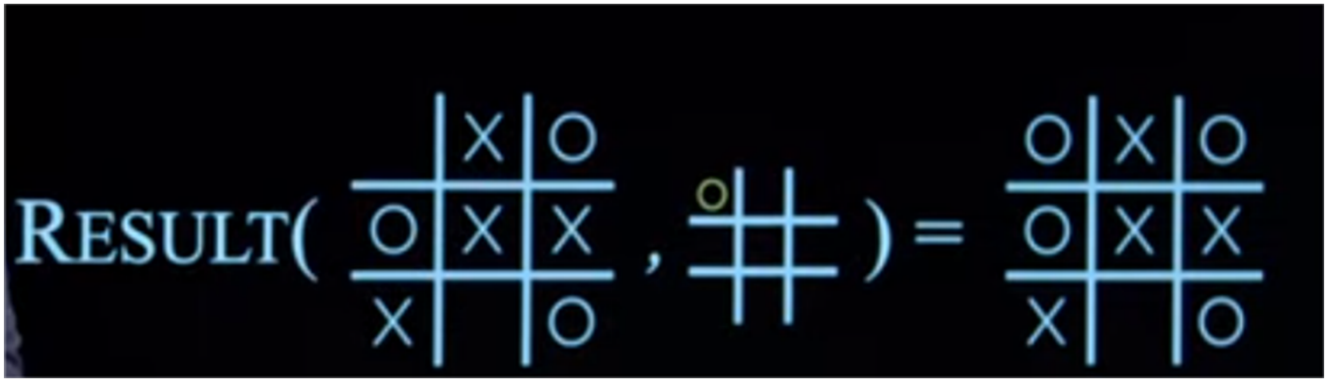
- Hasta ahora tenemos un único agente.
  - Intentando resolver un puzle, intentando ir de un punto a otro de un laberinto, ..
- Búsqueda en un **entorno hostil → juegos**
  - Juegos de 2 jugadores (alternos) de información perfecta (tablero a la vista)
  - Al acabar, cada jugador **pierde, gana o empata**
  - En principio se excluyen los juegos de azar
- *El árbol de juego* representa todas las jugadas posibles en una partida.
  - Estado inicial
  - Las hojas (fin de partida)
  - Partida completa → cada camino desde la raíz hasta una hoja
- Convenio
  - MAX es el jugador que abre el juego y al segundo, MIN.
  - Son posiciones MAX (MIN) aquéllas en las que tiene que jugar MAX (MIN)
  - Si identificamos la raíz con el nivel 0, y comienza jugando MAX, las posiciones de nivel par corresponden a MAX y las de nivel impar a MIN

Minimax

Si le damos estos valores a los tableros finales
el jugador (X) quiere maximizar la puntuación y el jugador (O) minimizarla

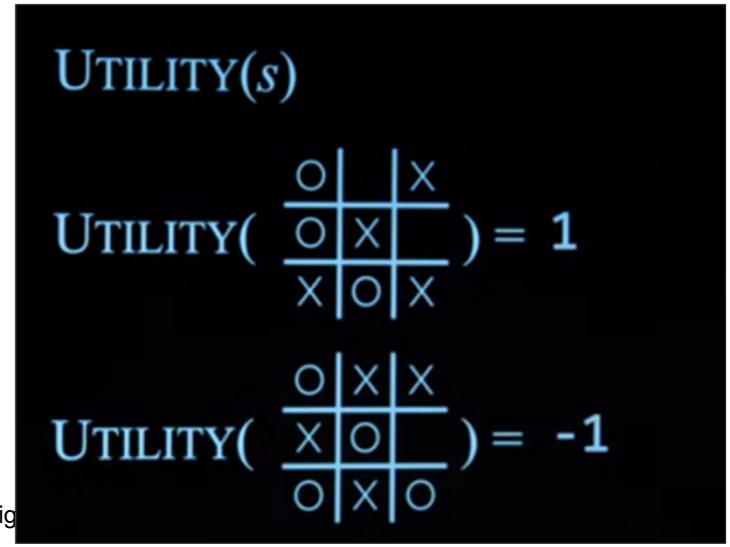
Definición del juego

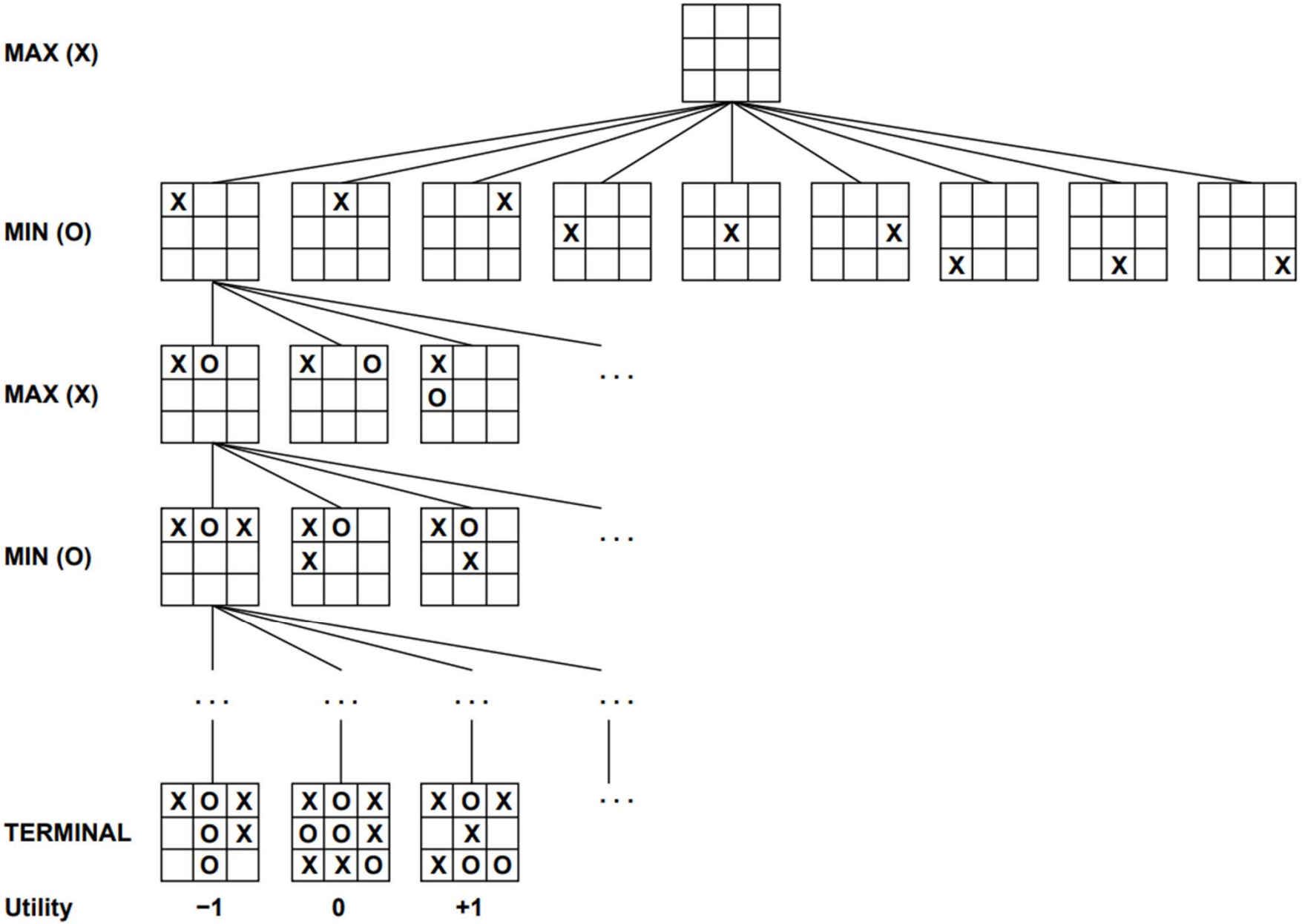
- Estado inicial
- Player (S): devuelve a qué jugador le toca en el estado S
- Actions (S): devuelve las acciones posibles en el estado S
- Result (S,A): devuelve el estado resultante de aplicar la acción A en el estado S



Definición del juego

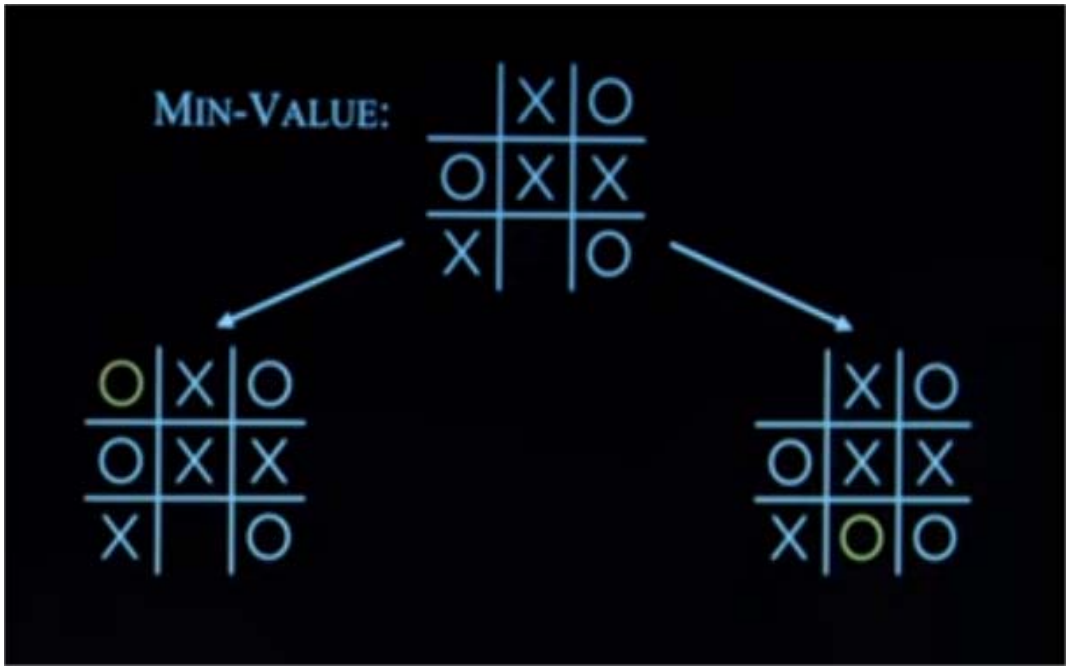
- Estado inicial
- Player (S): devuelve a qué jugador le toca en el estado S
- Actions (S): devuelve las acciones posibles en el estado S
- Result (S,A): devuelve el estado resultante de aplicar la acción A en el estado S
- Goal (S) o Terminal (S): comprueba si el estado S es un estado final.
- F_utilidad (S): devuelve un valor numérico asociado al estado terminal S




MAX (X)
MIN (O)
MAX (X)
MIN (O)
TERMINAL
Utility
−1
0
+1

Como calcular la utilidad cuando no es una situación final

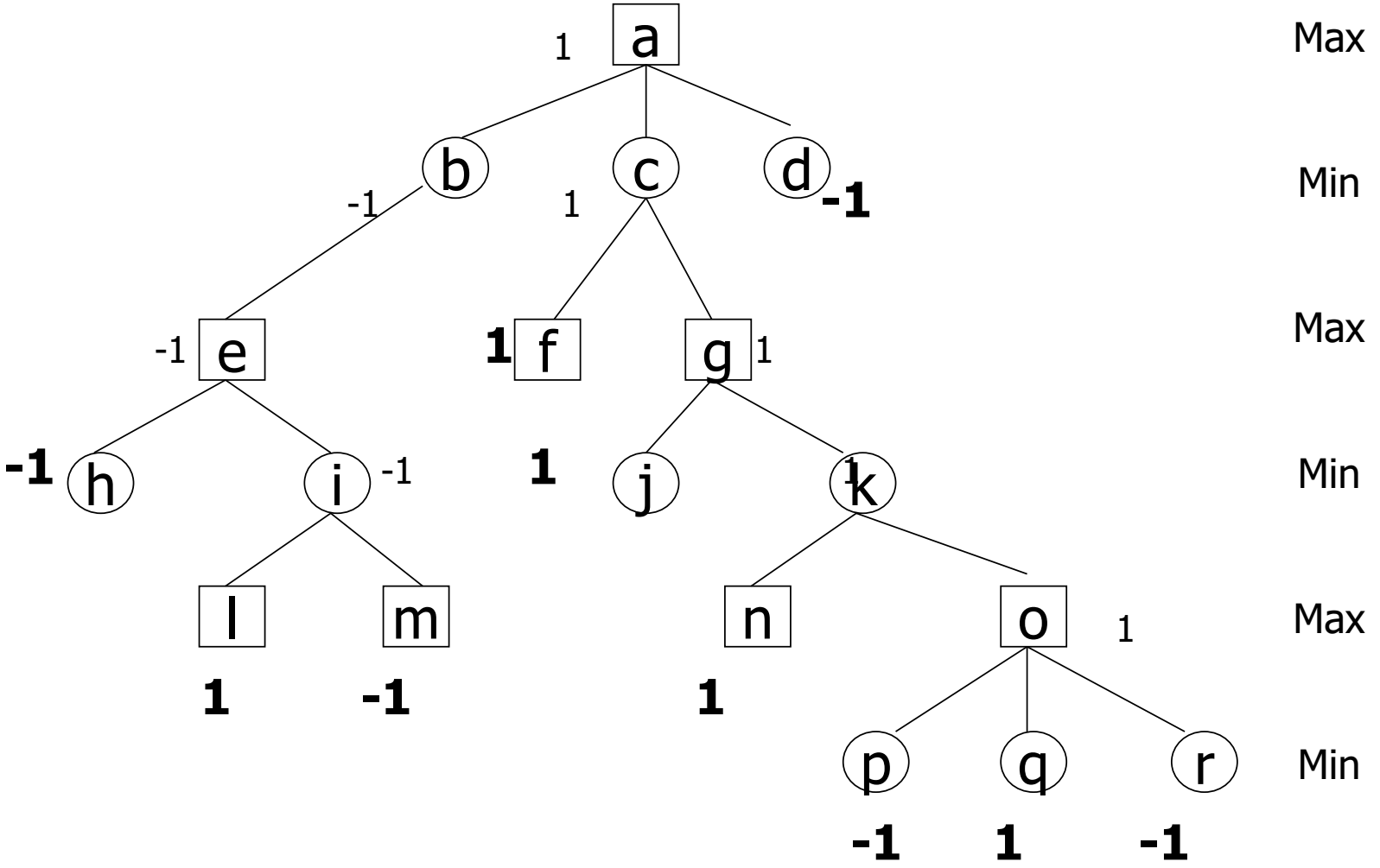
Tendremos que analizar las acciones y los estados siguientes
Le toca el turno a MIN



Convenio para Minimax

- En el árbol del juego, se asigna a los nodos terminales (final de partida) un valor de 1 (G), -1 (P) ó 0 (E)
  - 1 (o G) gana el primer jugador (maximizador)
  - -1 (o P) gana el segundo jugador (minimizador) es decir, pierde el primero
  - 0 (o E) los dos jugadores empatan
- Propagación hacia atrás de los valores de los nodos terminales:
  - A cada nodo **max** se le asigna el **máximo** de los valores de sus hijos.
  - A cada nodo **min** se le asigna el **mínimo** de los valores de sus hijos.
- El valor de la raíz sirve para tomar la decisión correspondiente a esa jugada.
- El árbol de juego se genera dinámicamente → no es viable mantener todo el árbol en memoria
  - recorrido **primero en profundidad**
  - inicializamos los valores de los nodos **max a -∞** y los nodos **min a +∞**

Etiquetado Minimax



Minimax

- Dado un estado s
  - MAX elige la acción a (a∈ACTIONS (s)) con el valor más alto de MIN-VALUE (RESULT(s,a))
  - MIN elige la acción a (a∈ACTIONS (s)) con el valor más bajo de MAX-VALUE (RESULT(s,a))

```
function MAX-VALUE(state):
    if TERMINAL(state):
        return UTILITY(state)
    v = -∞
    for action in ACTIONS(state):
        v = MAX(v, MIN-VALUE(RESULT(state, action)))
    return v
```

```
function MIN-VALUE(state):
    if TERMINAL(state):
        return UTILITY(state)
    v = ∞
    for action in ACTIONS(state):
        v = MIN(v, MAX-VALUE(RESULT(state, action)))
    return v
```

Aproximación recursiva

Minimax

- La etiqueta de un nodo nos indica lo mejor que podría jugar MAX en caso de que se enfrentase a un oponente perfecto.
- ***Resolver un árbol de juego*** significa asignar una etiqueta **G, P o E a la posición inicial.**
- Un *árbol solución (estrategia de juego) para MAX* es un subárbol del árbol de juego que:
  - contiene a la raíz
  - contiene **un sucesor** de cada posición MAX no terminal que aparezca en él
  - contiene **todos los sucesores** de cada nodo MIN que aparezca en él.
- Representa un plan (una estrategia) de los movimientos que debe realizar MAX, ante cualquier movimiento posible de MIN.
  - Conociendo **un árbol solución para MAX** ➔ programar a un computador para que juegue con un contrincante MIN
    - Un árbol solución para MAX se llamará *árbol ganador para MAX* (o estrategia ganadora para MAX) si **todas las posiciones terminales del árbol solución tienen etiqueta G**
    - Un árbol ganador para MAX asegura que el jugador MAX ganará, haga lo que haga MIN
    - **Existirá un árbol ganador para MAX si y sólo si al resolver el árbol de juego la etiqueta de la posición inicial es G**

Ejemplo

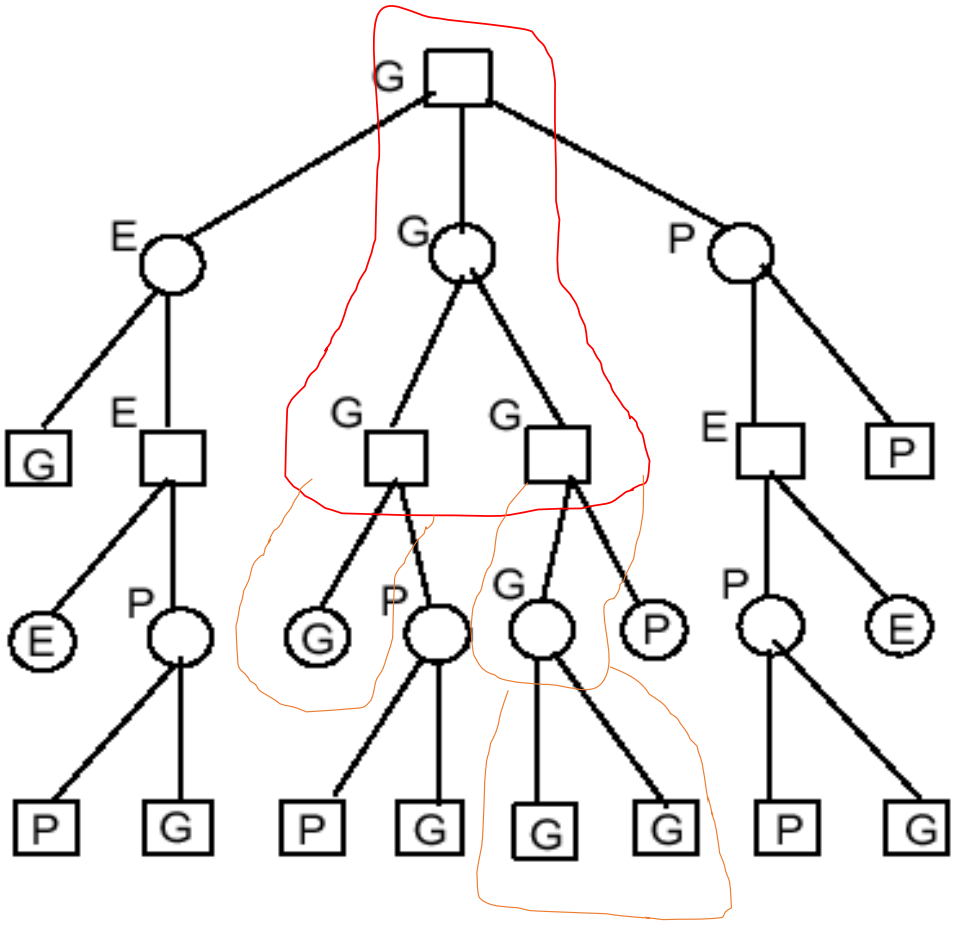
max

min

max

min

max



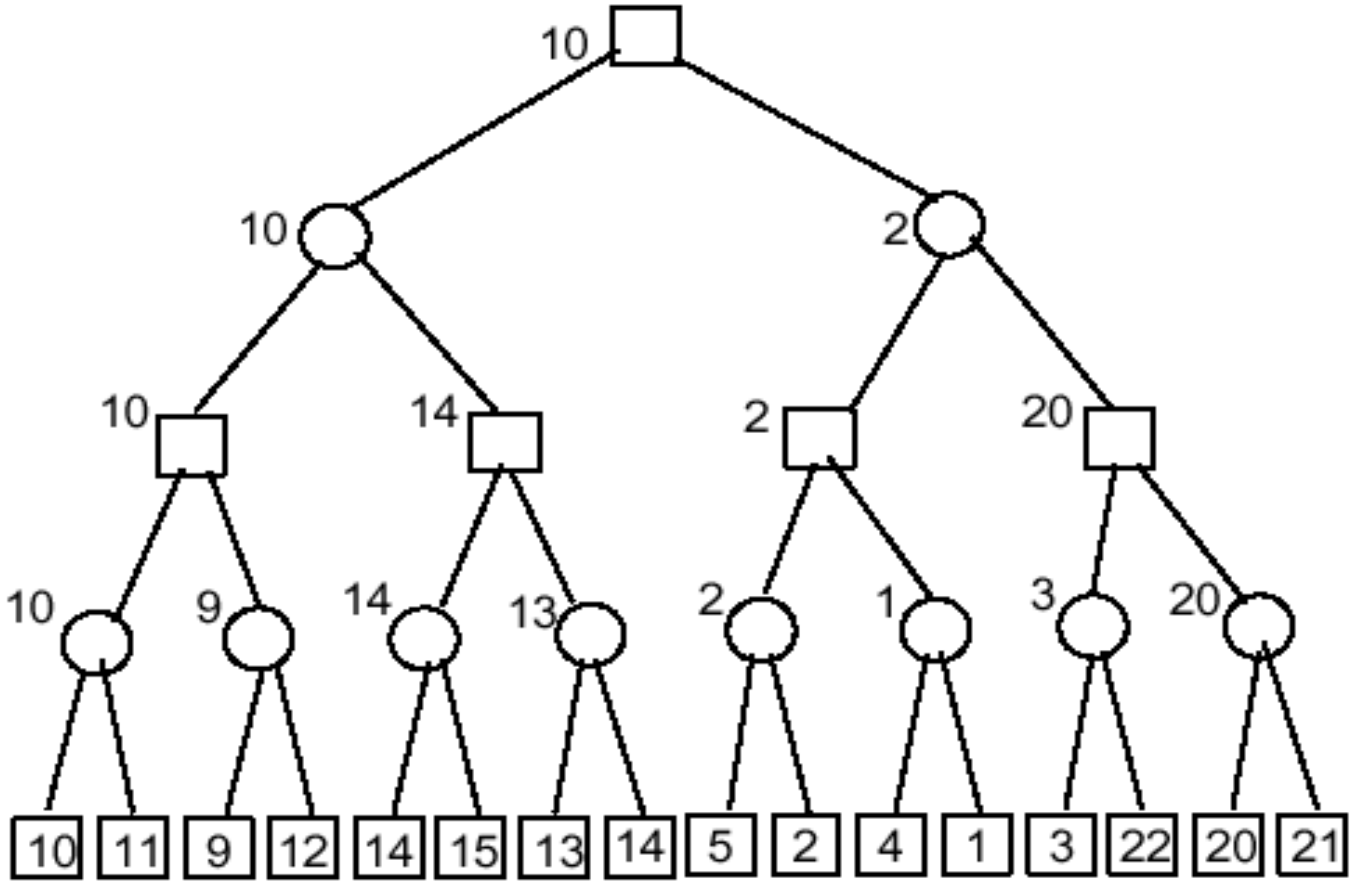
P -1
G +1
E 0

Existe un árbol ganador para MAX

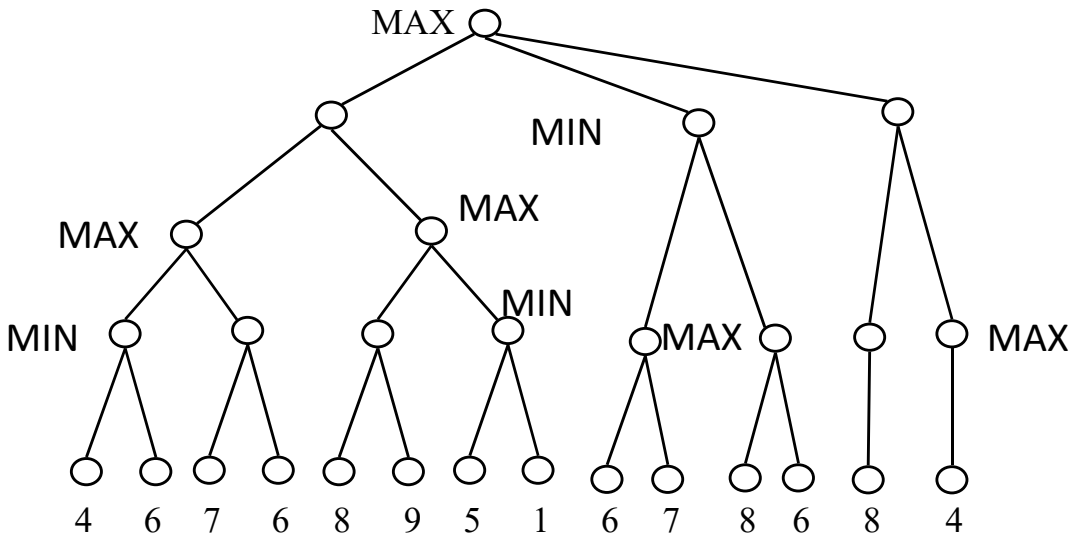
Minimax con estimación en nodos límite

- Minimax es exponencial en tiempo ➔ sólo aplicable directamente en juegos muy simples
- El método de etiquetado descrito requiere un árbol de juego completo.
- Para la mayoría de los juegos, desarrollar todo el árbol de juego es una tarea impracticable.
  - Un árbol de juego para las damas tiene aproximadamente 10^{40} posiciones no terminales. Generar el árbol completo requeriría 10^{21} siglos (3 billones de posiciones/ segundo)
  - Para el ajedrez unas 10^{120} posiciones y 10^{101} siglos
  - Incluso si un adivino nos proporcionase un árbol ganador (una estrategia ganadora) no tendríamos posibilidad de almacenarlo ni de recorrerlo.
- **Alternativa**: explorar sólo hasta una cierta profundidad límite y en los nodos de ese nivel utilizar una estimación **heurística** de la "bondad" del estado correspondiente (tendencia a ganar, perder o empatar)
  - Nodos terminales (finales de partida) ➔ función de **evaluación**
  - Nodos límite (nivel de exploración) ➔ función de **estimación**
- Aproximación heurística (profundidad limitada + estimación)
  - *Función de evaluación* (estática) h que nos indica cuál es el "mérito" de cada una de las posiciones.
  - h asigna valores grandes a las posiciones que son más favorables para MAX.
- Se asume que el valor ascendido a la raíz, obtenido mediante una profundización hasta un límite P, va a ser una estimación mejor que si aplicáramos directamente la función de estimación al nodo raíz

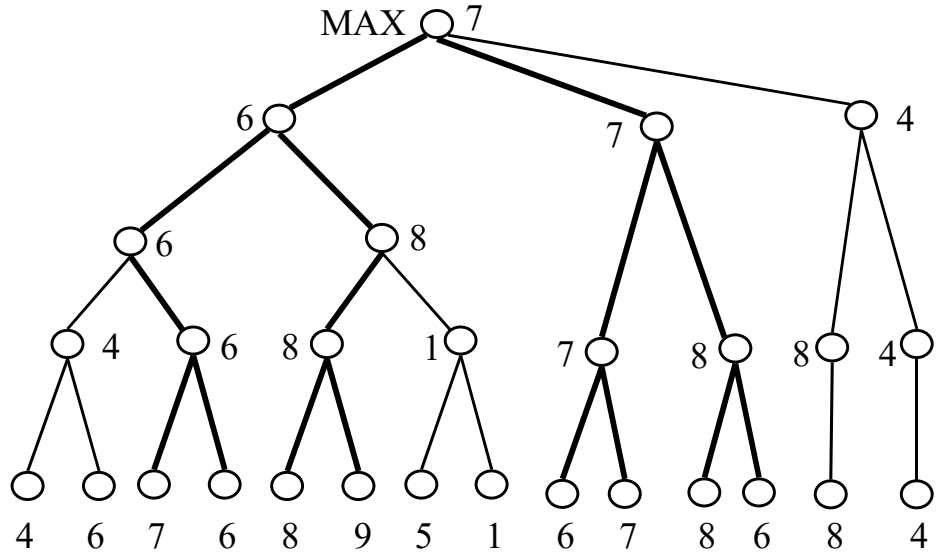
Ejemplo: minimax de 4 niveles con f estimación

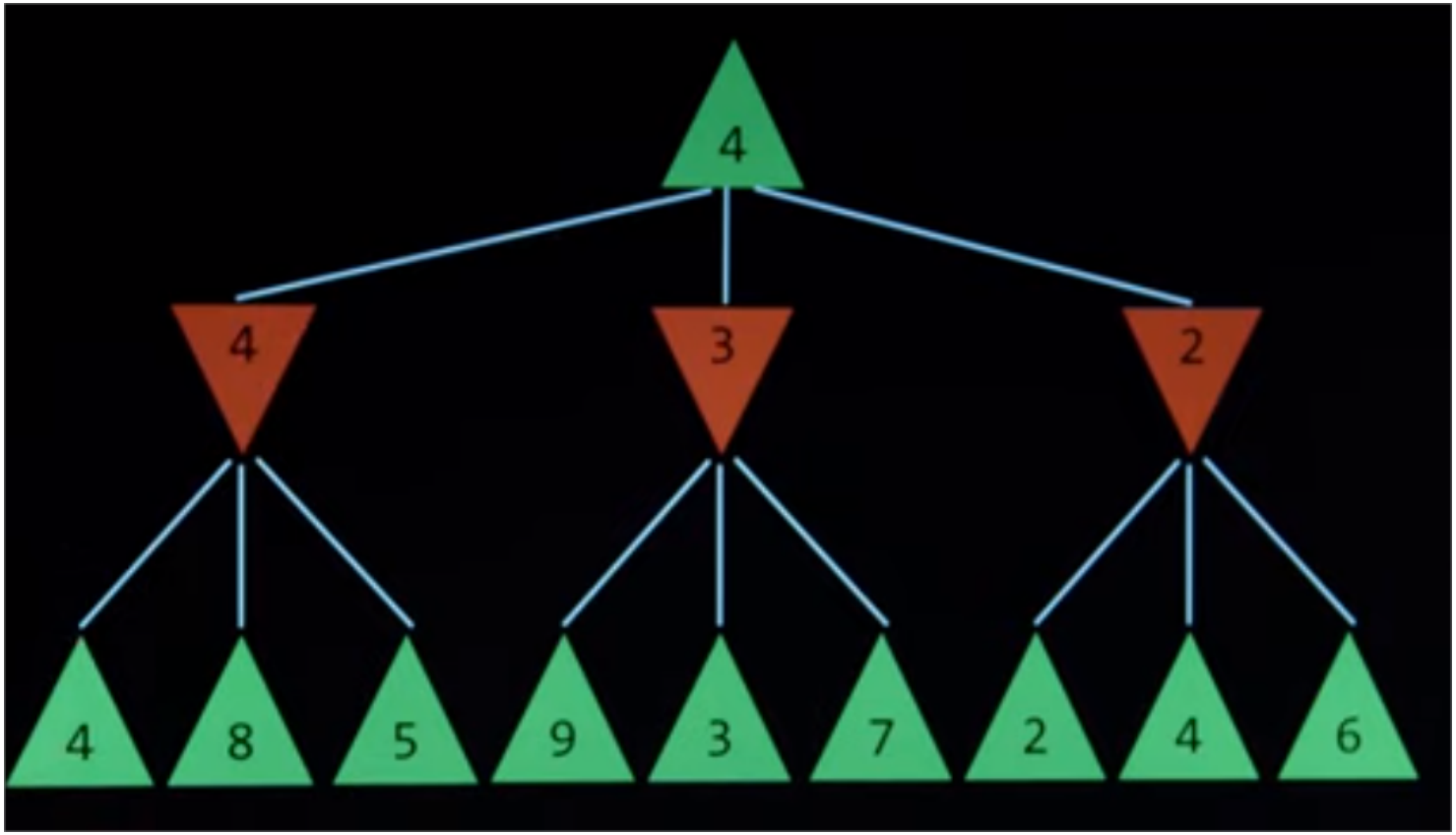


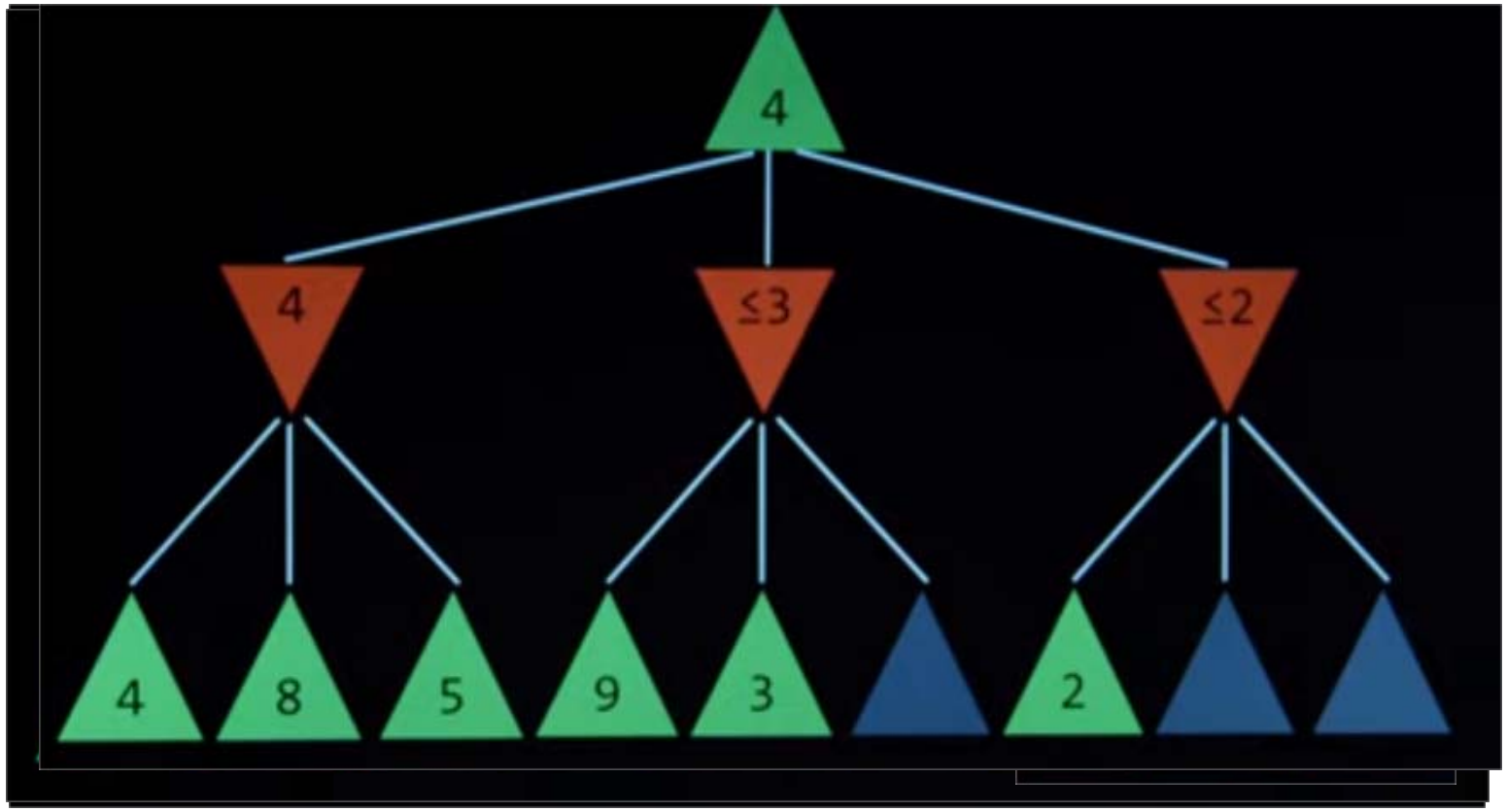
Árbol inicial



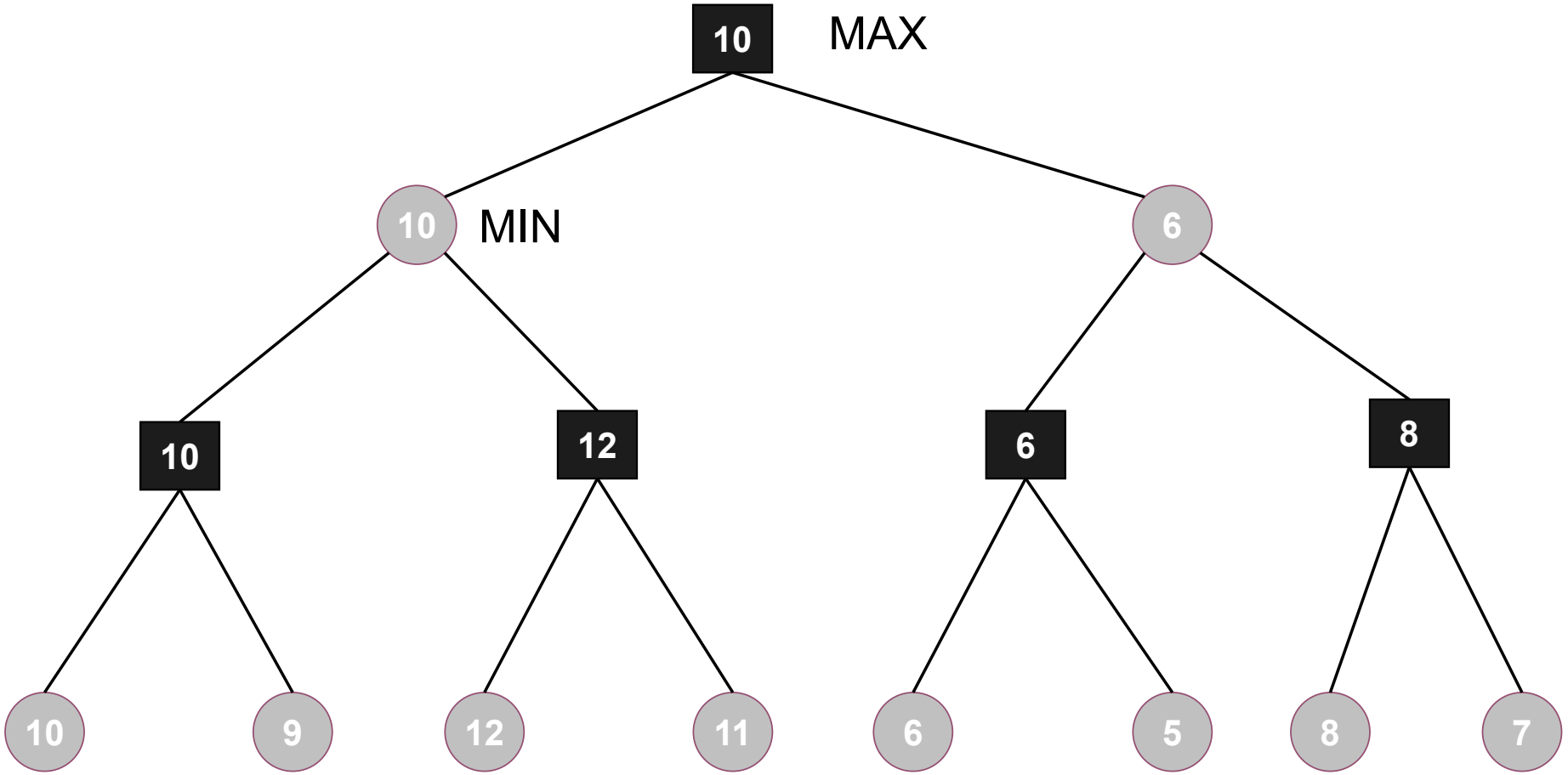
Etiquetado MINIMAX

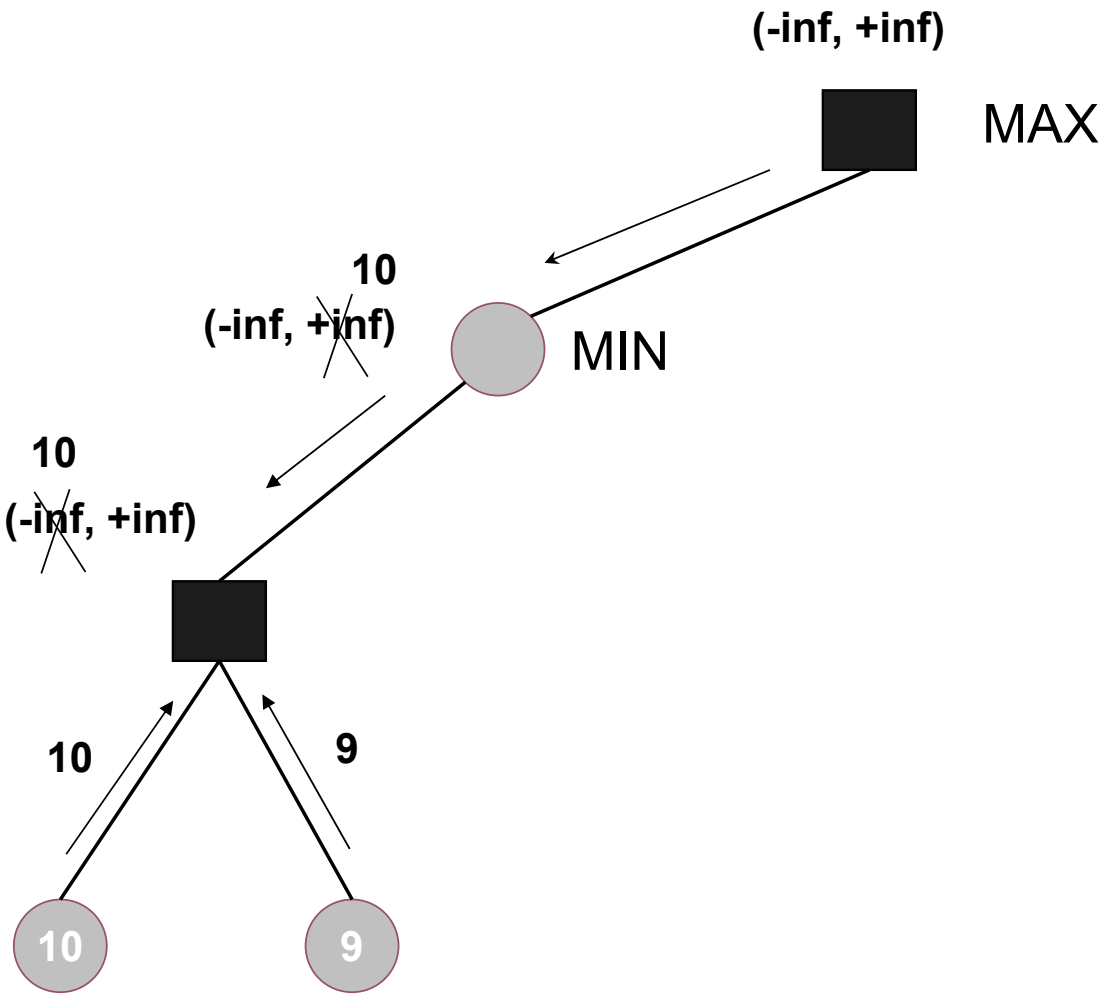


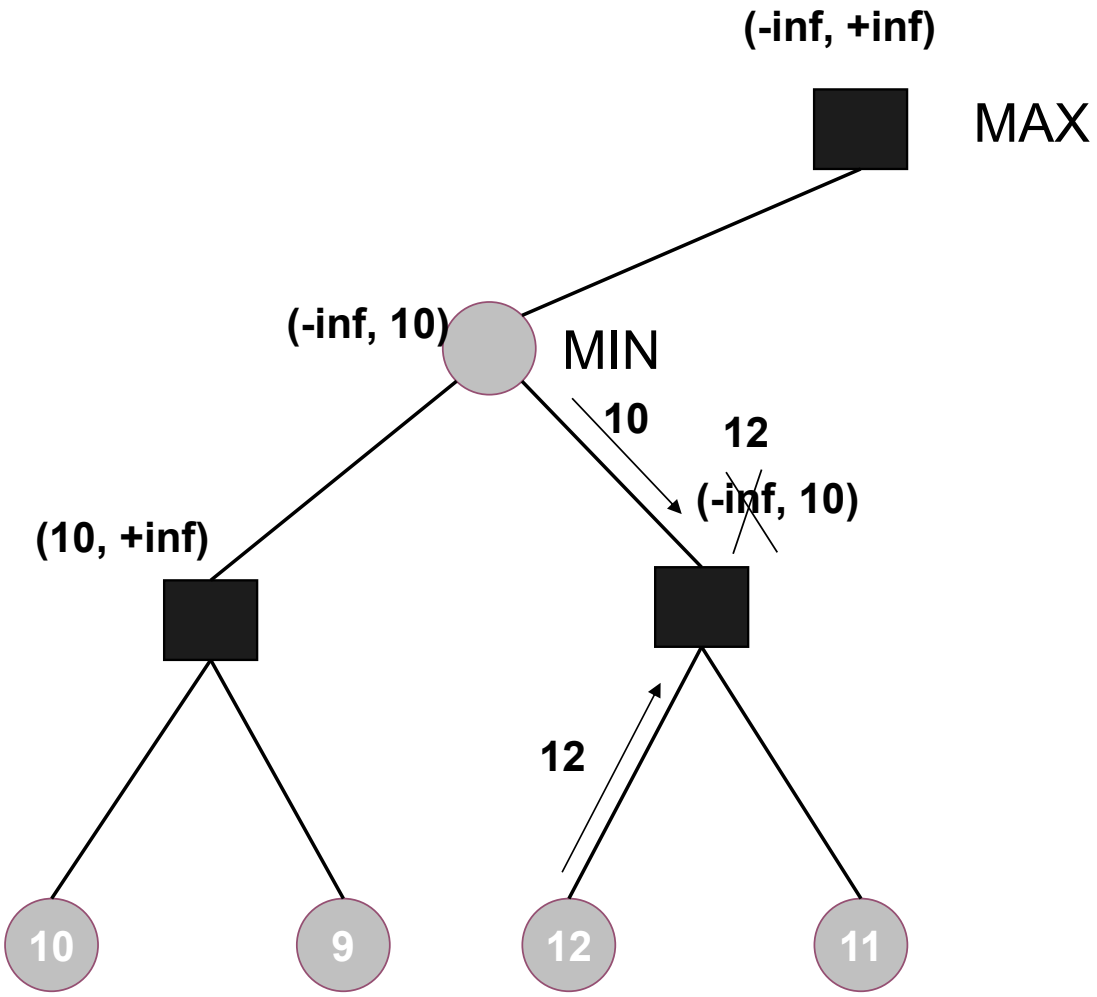

4
4
3
2
4
8
5
9
3
7
2
4
6

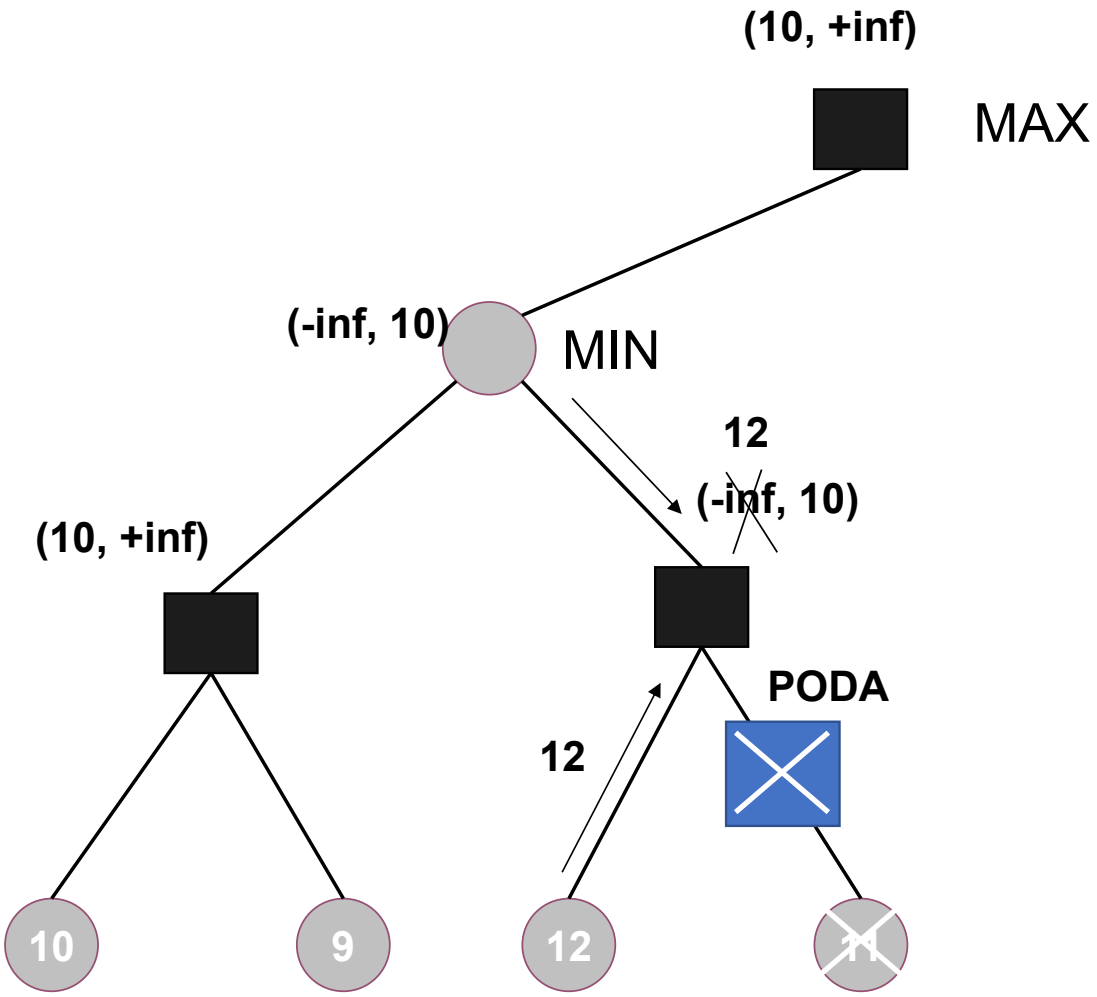

MAX
MIN
4
4
≤3
≤2
4
8
5
9
3
2

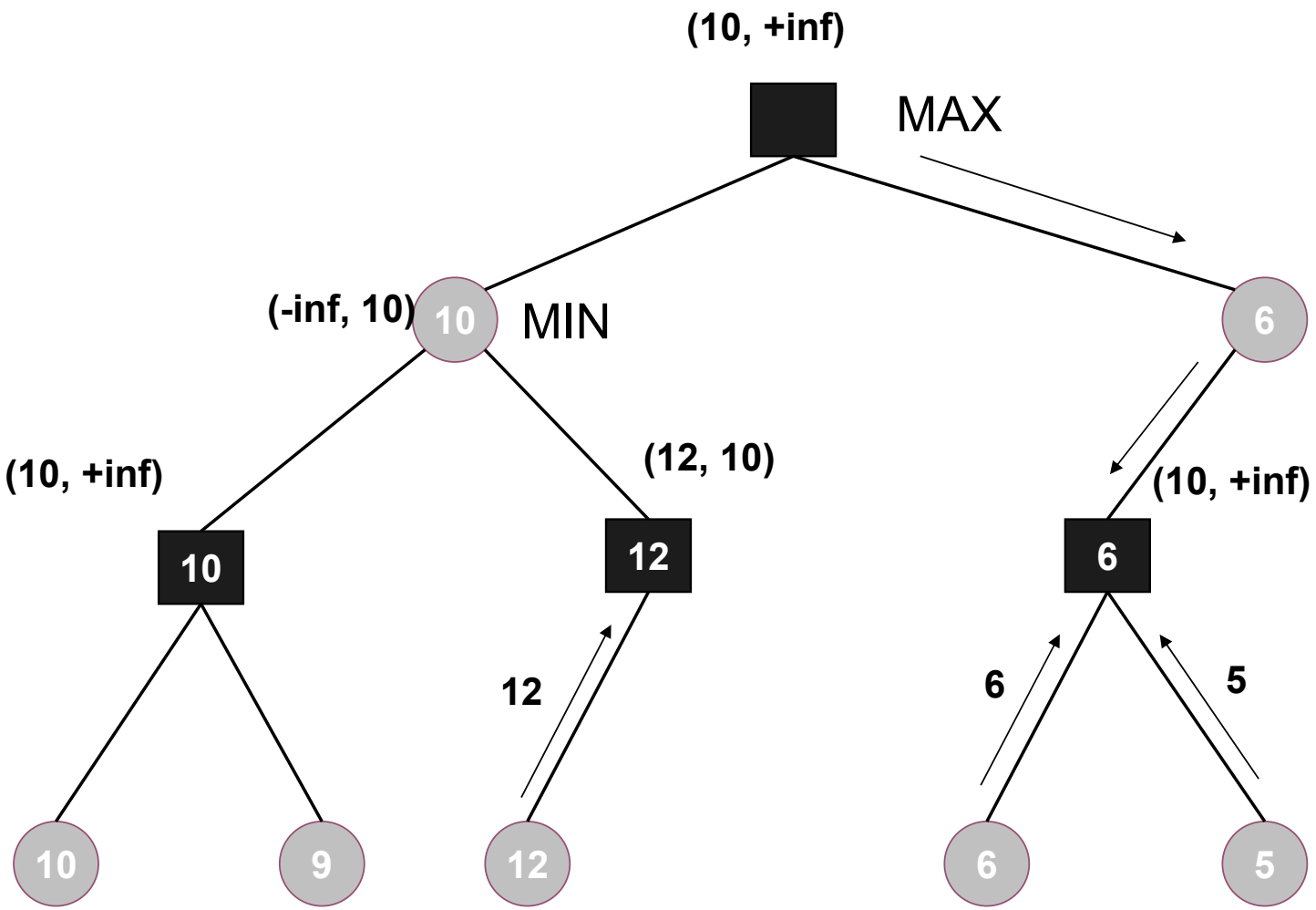
Poda alfa-beta

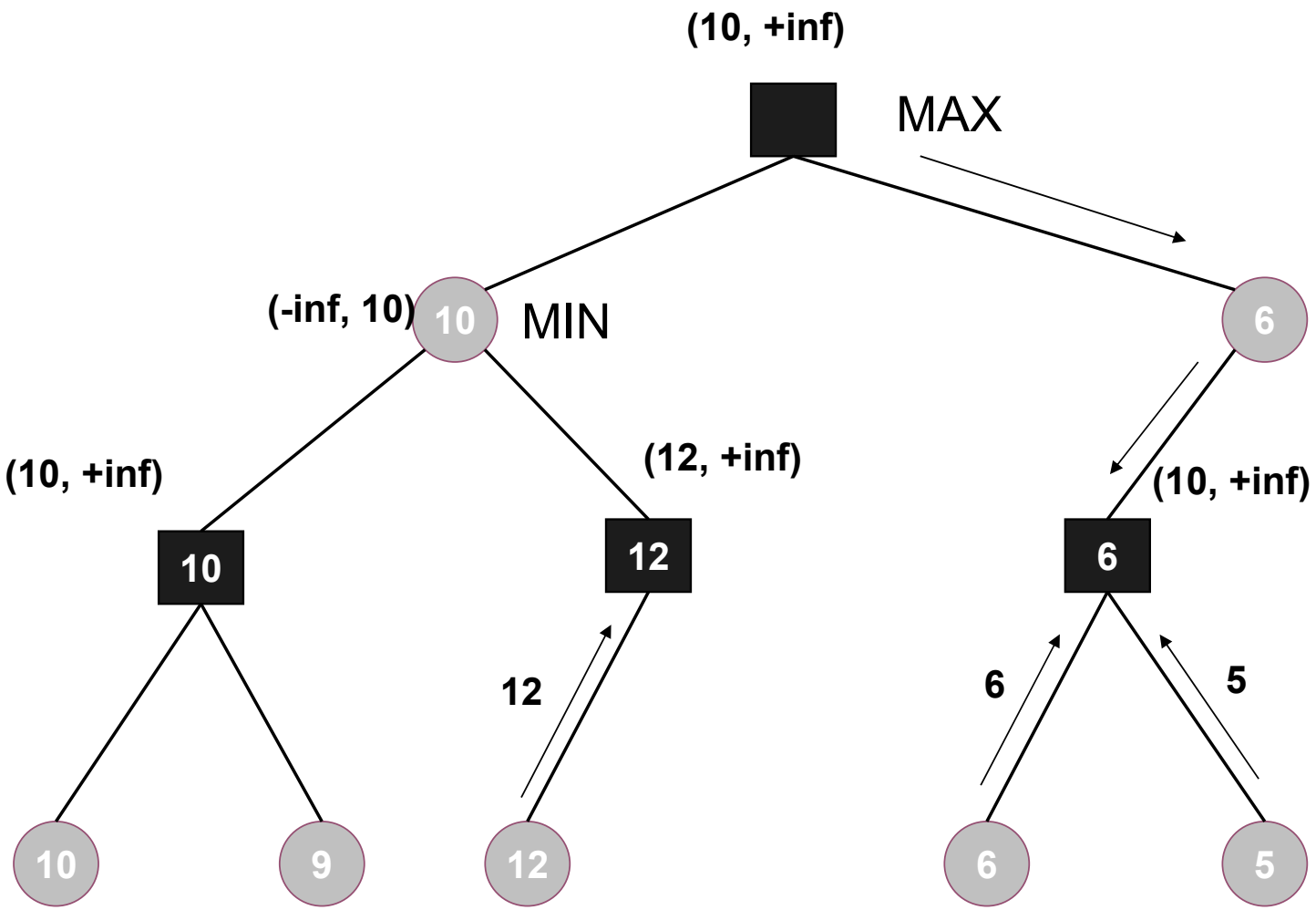



(-inf, +inf)
MAX
10
(-inf, +inf)
MIN
10
(-inf, +inf)
10
9
10
9

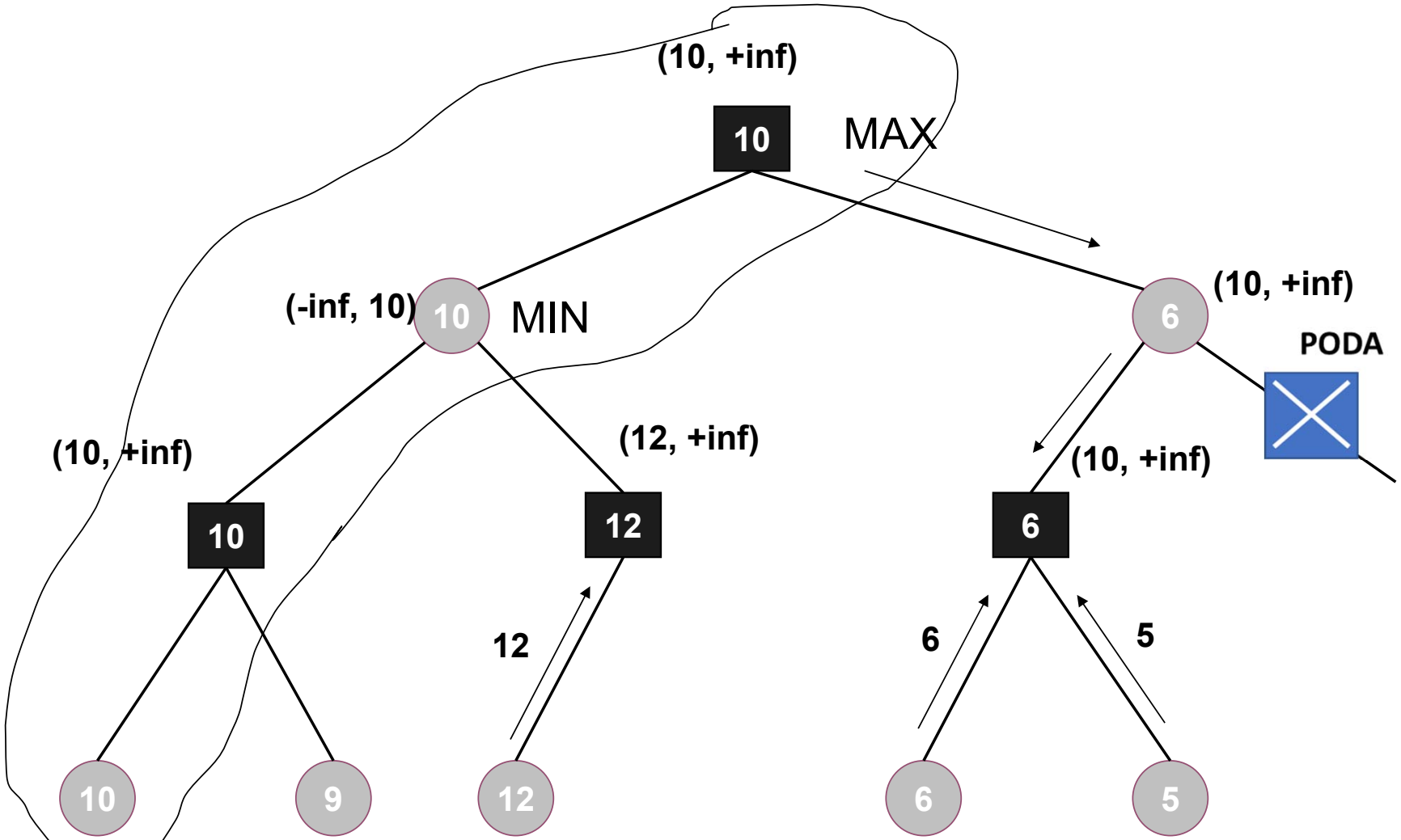

(-inf, +inf)
MAX
(-inf, 10)
MIN
10
12
(-inf, 10)
(10, +inf)
12
10
9
12
11


(10, +inf)
MAX
(-inf, 10)
MIN
12
(-inf, 10)
(10, +inf)
PODA
12
10
9
12
11


(10, +inf)
MAX
(-inf, 10)
10
MIN
6
(10, +inf)
(12, 10)
(10, +inf)
10
12
6
12
6
5
10
9
12
6
5



Tiene un valor de 6 o menos

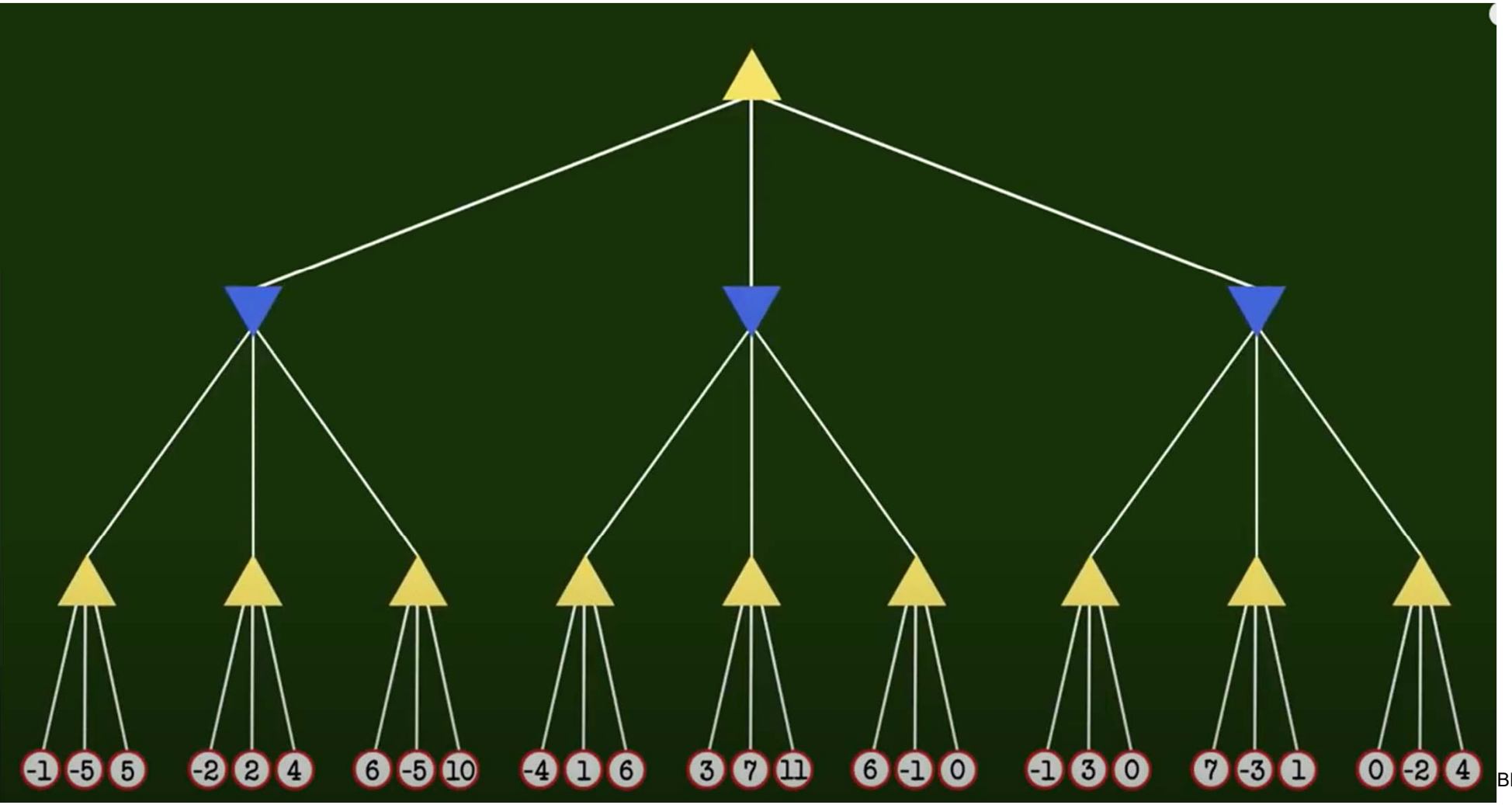

(10, +inf)
10
MAX
(-inf, 10)
10
MIN
6
(10, +inf)
PODA
(10, +inf)
(12, +inf)
(10, +inf)
10
12
6
12
6
5
10
9
12
6
5

The α–β algorithm

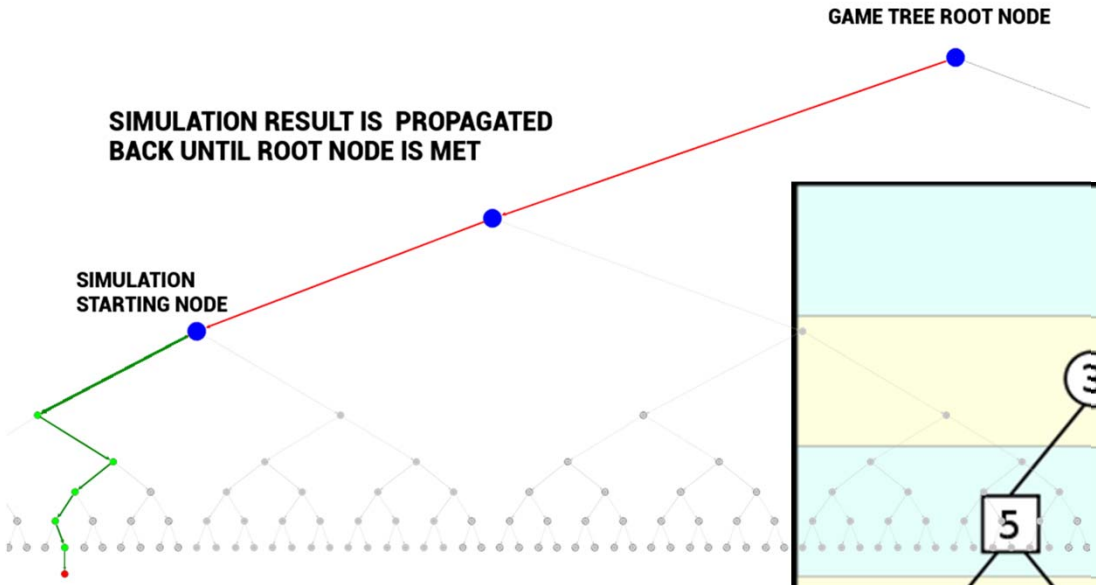
```
function ALPHA-BETA-DECISION(state) returns an action
    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v

function MIN-VALUE(state, α, β) returns a utility value
    same as MAX-VALUE but with roles of α, β reversed
```

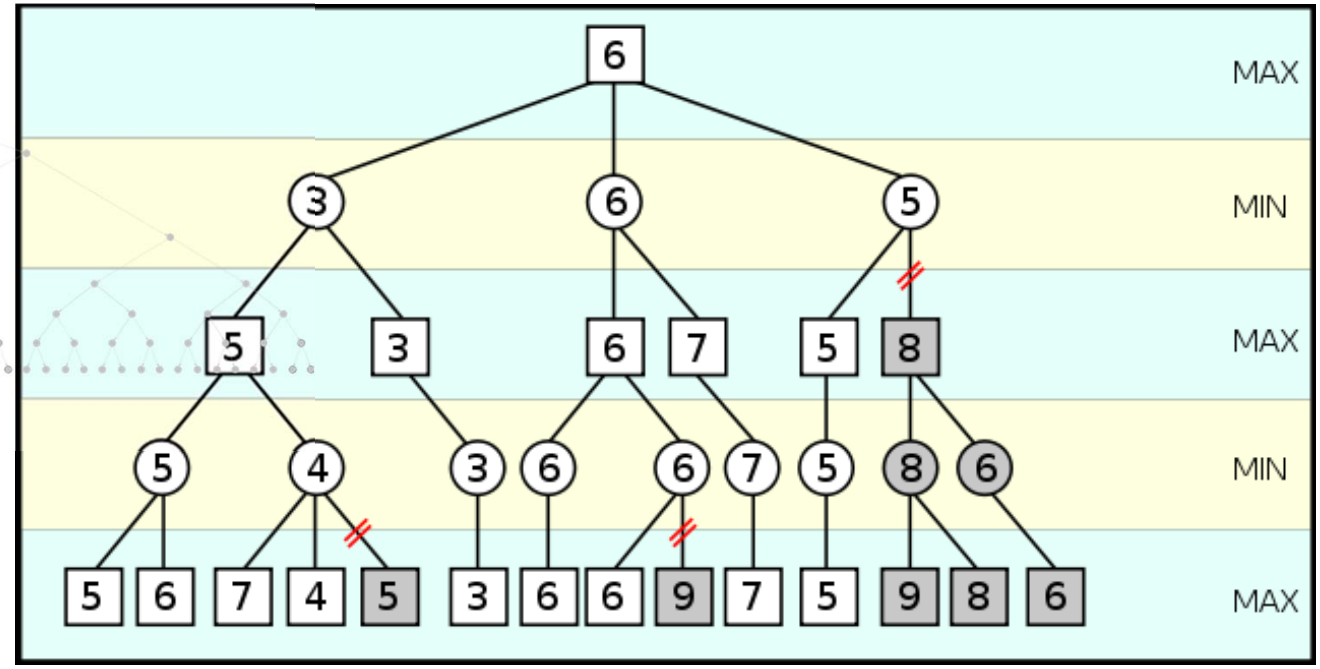
Ejercicio para practicar

https://www.youtube.com/watch?v=l0y-TGehf-4



Árboles de montecarlo

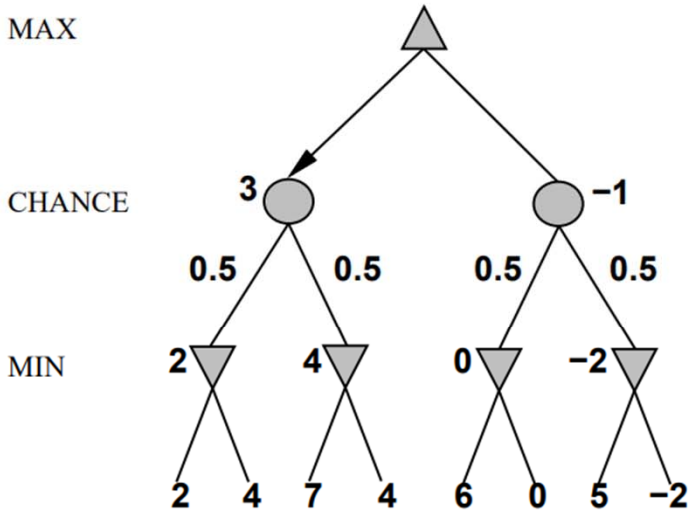




El origen de los arboles de Montecarlo **MCTS** fue la Tesis Doctoral de Bruce Abramson (1987) donde combinaba una búsqueda minimax con un modelo probabilístico para generar al azar jugadas completas, en vez de usar una función de evaluación estática.

Juegos no deterministas: expectiMinimax

- Moneda




```
...
if state is a MAX node then
        return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a MIN node then
        return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a chance node then
        return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
...
```