Tema 6

Juegos de 2 Adversarios

elisa.guerrero@uca.es

Objetivos

- Saber reconocer el tipo de problemas a los que se les puede aplicar estas estrategias
- Saber formalizar problemas para juegos de dos adversarios
- Conocer y saber aplicar las estrategias Minimax y Poda alfa-beta
- Saber definir funciones heurísticas para nodos no terminales
- Conocer las ventajas y limitaciones de cada estrategia y plantear alternativas (efecto horizonte, minimax incremental …)
- Implementar Minimax y Poda en un lenguaje de programación

Juegos de 2 Adversarios

1. Generalidades
2. El Juego del Grundy
3. Algoritmo Minimax
4. Ejemplos: Aplicación Minimax
5. TicTacToe
6. Funciones Heurísticas
7. Poda alfa-beta
8. Ejemplos: Aplicación de Poda
9. Análisis y Variantes

1. Características de los Juegos

- Juegos para 2 jugadores (bipersonales)
- Los jugadores mueven alternativamente
  - Jugada: Acción que lleva a cabo un jugador cada vez que le toca jugar
- La ventaja para un jugador es desventaja para el otro
- Los jugadores poseen toda la información sobre el estado del juego
- Hay un número finito de estados y decisiones
- No interviene el azar

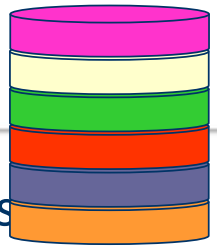
Ejemplo: **3 en Raya o TicTacToe**

X	O	O
O	X	
		X

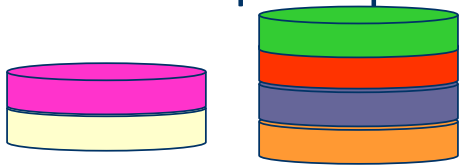
1.1 Formalización

- Un Problema de Búsqueda entre 2 Adversarios se define formalmente como:
  - **Nodo Inicial,** describiendo la situación inicial del juego.
  - **Jugadas** o movimientos que puede realizar cada jugador.
  - **Función esValida** para determinar si una determinada jugada/movimiento se puede realizar de acuerdo a las reglas del juego.
  - **Función aplicaJugada**, para aplicar una jugada cuando esta sea válida.
  - **Test Terminal** que determina cuándo acaba el juego, porque gana un jugador o se produce un empate.
  - **Función Utilidad**, da una puntuación a los nodos terminales.

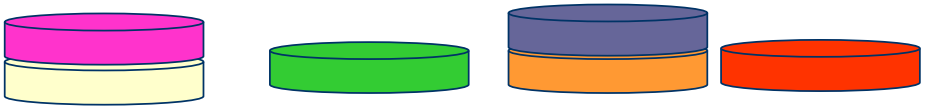
2. Juego del Grundy



- Se dispone de una pila de N fichas
- El primer jugador divide la pila original en dos pilas que deben ser desiguales.



- Después alternativamente, cada jugador hace lo mismo con alguna de las pilas.
- El juego sigue hasta que cada pila tiene sólo una o dos fichas, lo que hace imposible su continuación.

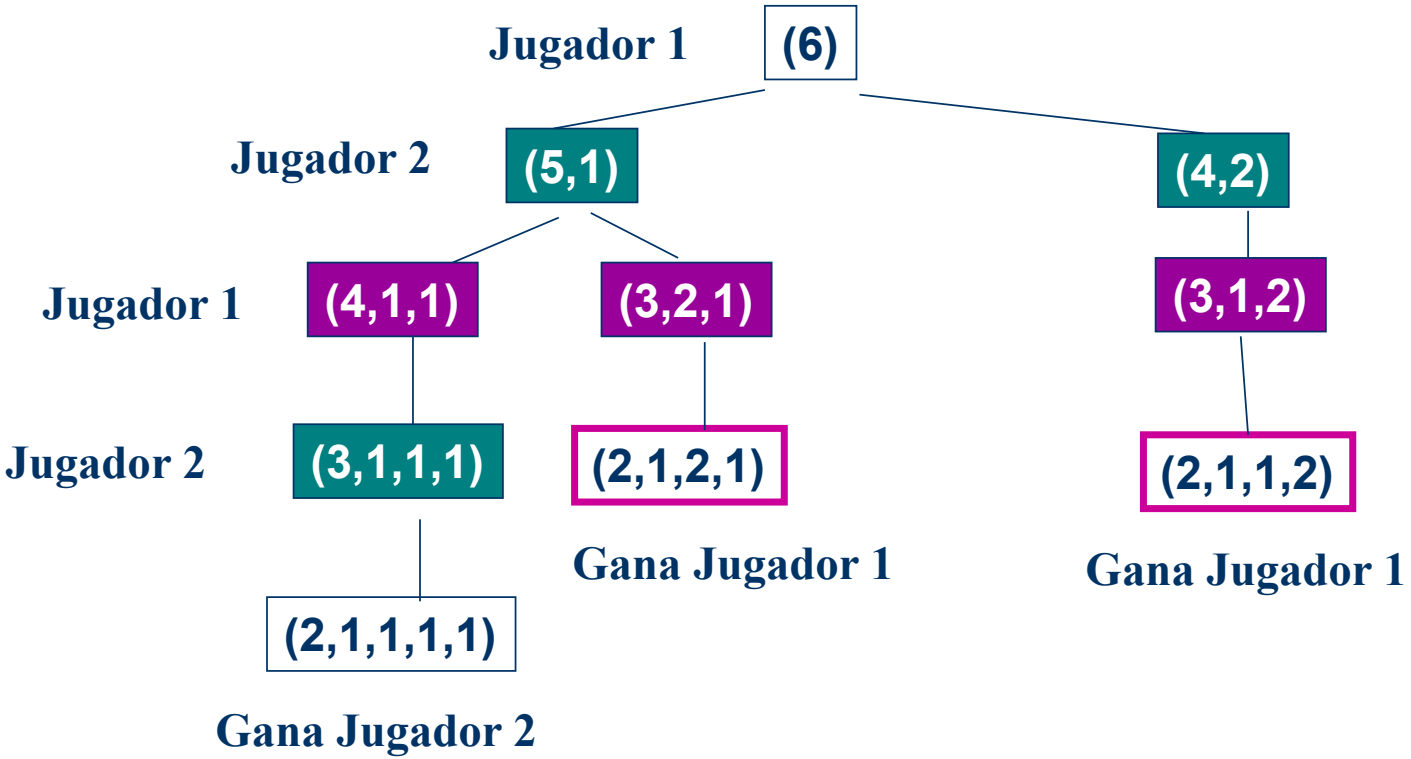


- El primer jugador que no pueda realizar un movimiento válido, pierde.

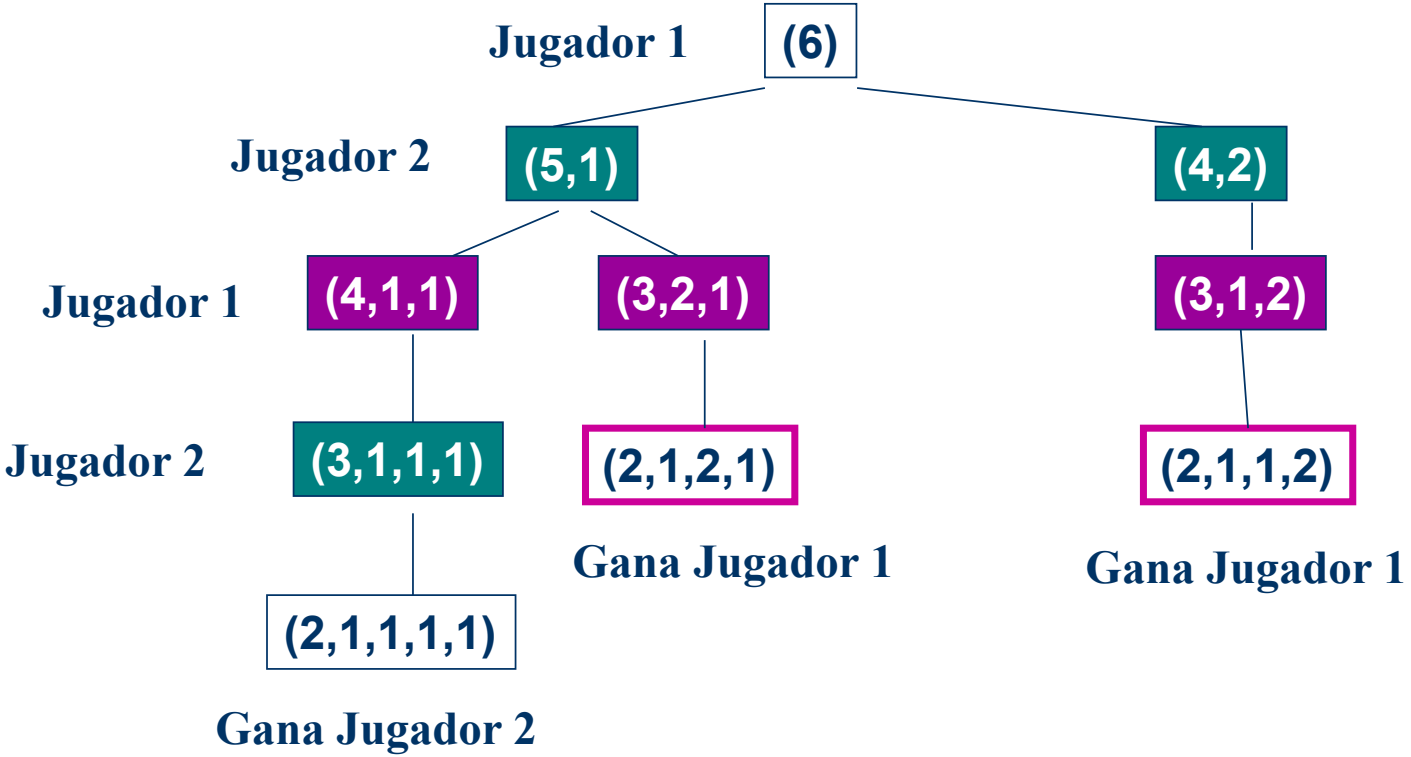
2.1 Juego del Grundy

- **Estado Inicial**: N fichas.
- **Jugadas**: Dividir un lote de fichas en dos lotes de tamaño desigual, cada una de las posibles divisiones de cada pila será una jugada.
- **esValida**: comprueba que la posible división produce dos lotes desiguales.
- **aplicaJugada**: realiza la división sobre una pila y obtiene dos pilas desiguales.
- **Test Terminal**: comprueba que todos los lotes constan de 1 o 2 fichas.
- **Función Utilidad:** si el nodo terminal pertenece a un movimiento ejecutado por el Jugador 1 devuelve +100, si el nodo terminal es del Jugador 2 devuelve -100.

2.2 Juego del Grundy con 6 fichas



¿Qué estrategia seguir para que el Jugador 1 elija un camino ganador?



3. MiniMax: Decidir la mejor jugada

- Dos adversarios o contrincantes: MAX y MIN.
- Diseñado para conseguir la mejor jugada para MAX, por tanto MAX suele representar al Agente Inteligente (ordenador).
- En cada turno de MAX hay que construir el árbol de juego y deberá elegir la mejor jugada posible pero teniendo en cuenta que MIN también intenta ganar el juego, por tanto hará las mejores jugadas para sí mismo también.

3.1 MiniMax: Estrategia básica

En **cada turno** MAX debe:

Construir el árbol de juego completo cuyo nodo raíz sea la situación actual (situación después de un movimiento de MIN o el estado inicial).

Valorar los estados finales según la función de utilidad.

Propagar hasta la raíz los valores de la función.

Elegir la jugada de MAX que tenga la mejor valoración.

3.2 MiniMax: Propagación de valores

La propagación de valores se hace según el principio minimax:

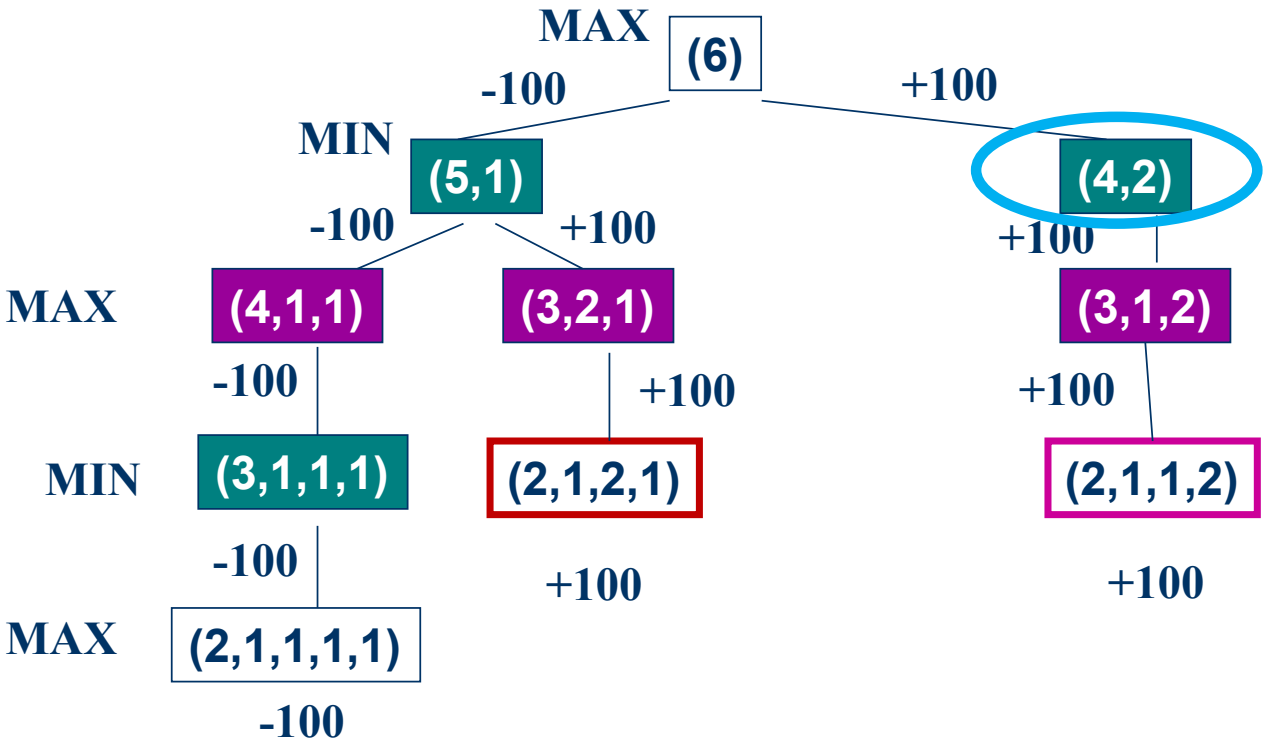
"MAX siempre escogerá lo mejor para MAX y
MIN lo mejor para MIN que es también lo peor para MAX":

- MAX toma el valor del sucesor con mayor valor

  - Utilidad(n) Si n es un nodo terminal
  - **Max** (ValorMin (s)) $_{s \in Sucesores(n)}$

- MIN toma el valor del sucesor con menor valor

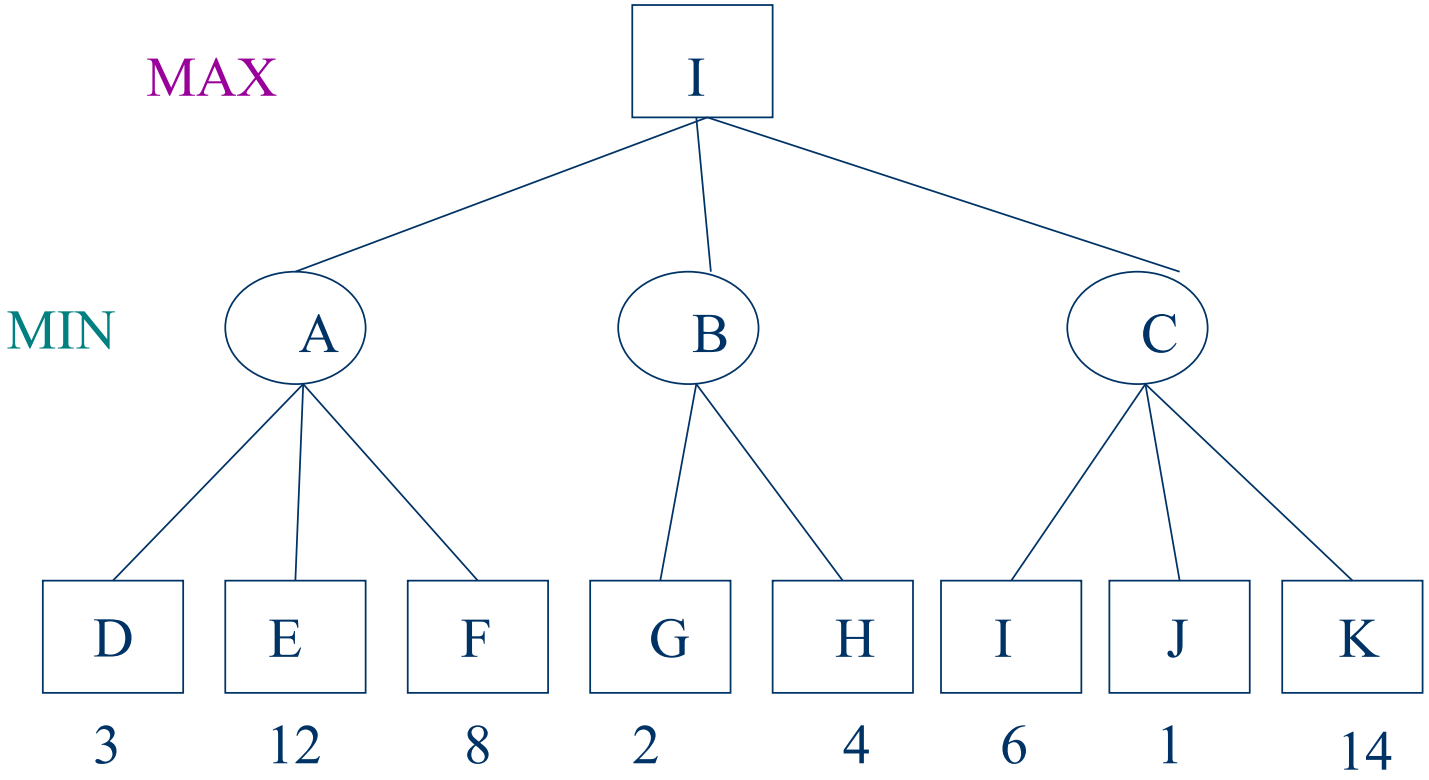
  - Utilidad(n) Si n es un nodo terminal
  - **Min** (ValorMax(s)) $_{s \in Sucesores(n)}$



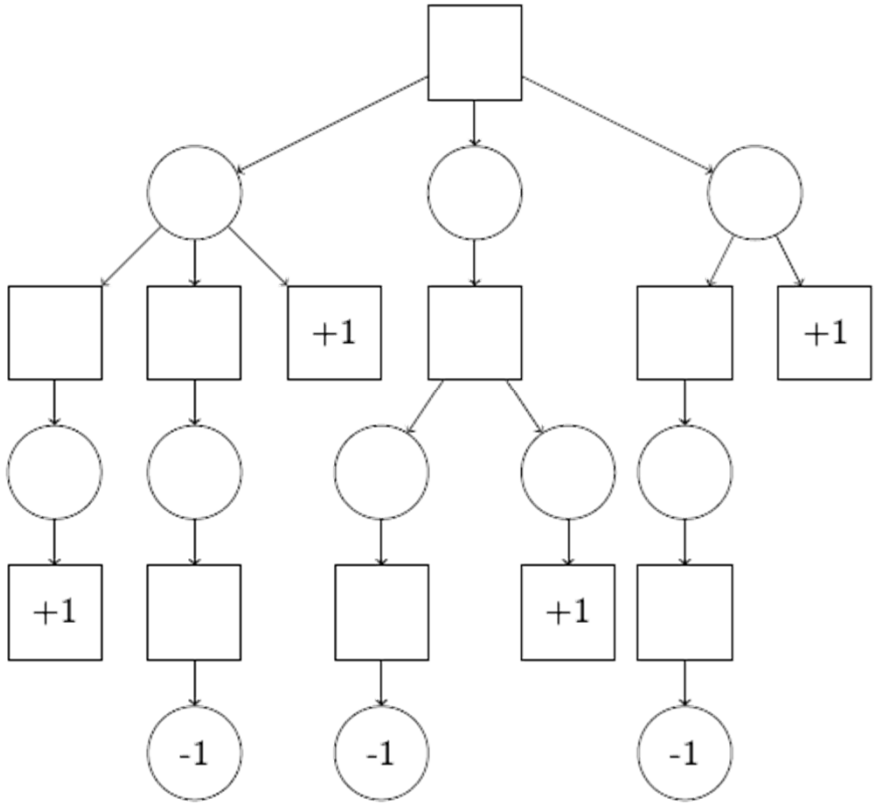
F_utilidad(Jugador1)= +100

F_utilidad(Jugador2)= -100

4. Ejemplo1: Aplicación Minimax (2 niveles)



4.6 Ejemplo 2: Aplicación Minimax



3.4 Función Minimax

```
tNodo: función minimax(E/S tNodo: nodo, E entero: jugador)
var
  entero: max, max_actual, jugada, mejorJugada      tNodo: intento
inicio
      max ← -10000
       desde jugada ← 1 hasta N hacer
              si esValida(nodo, jugada) entonces
                            intento ← aplicaJugada(nodo,jugador,jugada)
                            max_actual ← valorMin(intento, opuesto(jugador))
                            si max_actual > max  entonces
                                  max ← max_actual
                                  mejorJugada ← jugada
                            fin_si
              fin_si
       fin_desde
      fin_si
      nodo=aplicaJugada(nodo,jugador,mejorJugada);
      devolver nodo
fin_función
```

3.4 Función Minimax

```
desde jugada ← 1 hasta N hacer
        si esValida(nodo, jugada) entonces
            intento ← aplicaJugada(nodo,jugador,jugada)
            max_actual ← valorMin(intento, opuesto(jugador))
            si max_actual > max  entonces
                        max ← max_actual
                        mejorJugada ← jugada
            fin_si
           fin_si
fin_desde
(…)

nodo=aplicaJugada(nodo, jugador, mejorJugada);
```

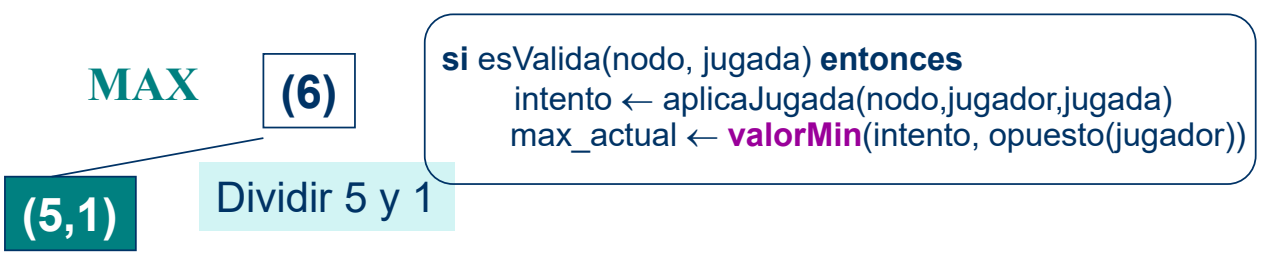
3.5 Función valorMin

```
entero: función valorMin(E tNodo: nodo, E entero: jugador)
(…)
    si terminal(nodo) entonces
            valor_min← utilidad(nodo)
    si_no
            valor_min ← +100000
            desde jugada ← 1 hasta N hacer
                si esValida(nodo, jugada) entonces
                        intento ← aplicaJugada(nodo,jugador,jugada)
                        aux ← valorMax(intento, opuesto(jugador))
                        valor_min ← minimo(valor_min, aux)
                fin_si
            fin_desde
   fin_si
   devuelve valor_min
fin_función
```

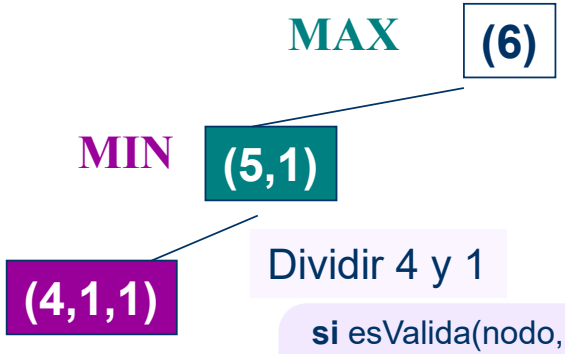
3.6 Función valorMax

```
entero: función valorMax(E tNodo: nodo, E entero: jugador)
(…)
si terminal(nodo) entonces
        valor_max← utilidad(nodo)
   si_no
        valor_max ← -100000
        desde jugada ← 1 hasta N hacer
          si esValida(nodo, jugada) entonces
                intento ← aplicaJugada(nodo,jugador,jugada)
                aux ← valorMin(intento, opuesto(jugador)
                valor_max ← maximo(valor_max, aux)
          fin_si
        fin_desde
   fin_si
   devuelve valor_max
fin_función
```

3.7 Traza Minimax



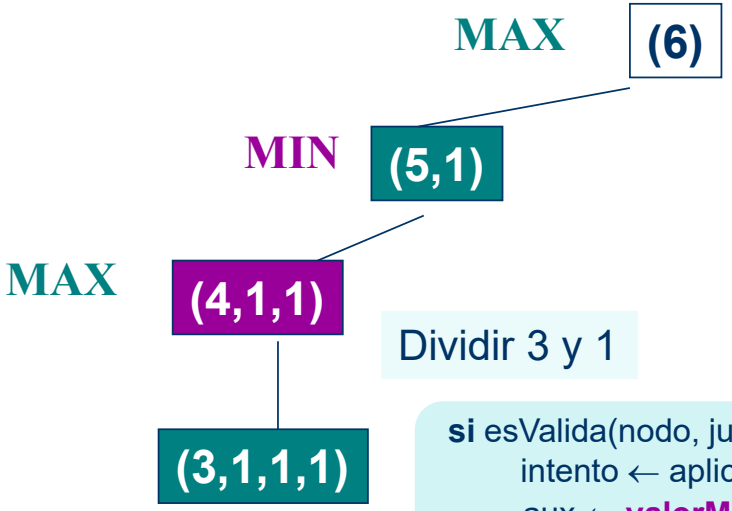
3.7 Traza Minimax



Dividir 4 y 1

```
si esValida(nodo, jugada) entonces
        intento ← aplicaJugada(nodo,jugador,jugada)
        aux ← valorMax(intento, opuesto(jugador))
```

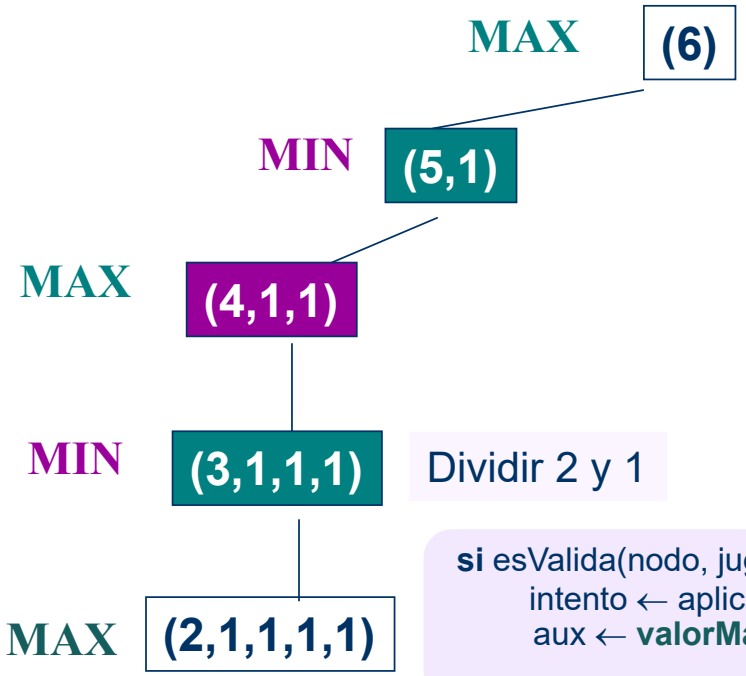
3.7 Traza Minimax



Dividir 3 y 1

```
si esValida(nodo, jugada) entonces
      intento ← aplicaJugada(nodo,jugador,jugada)
      aux ← valorMin(intento, opuesto(jugador))
```

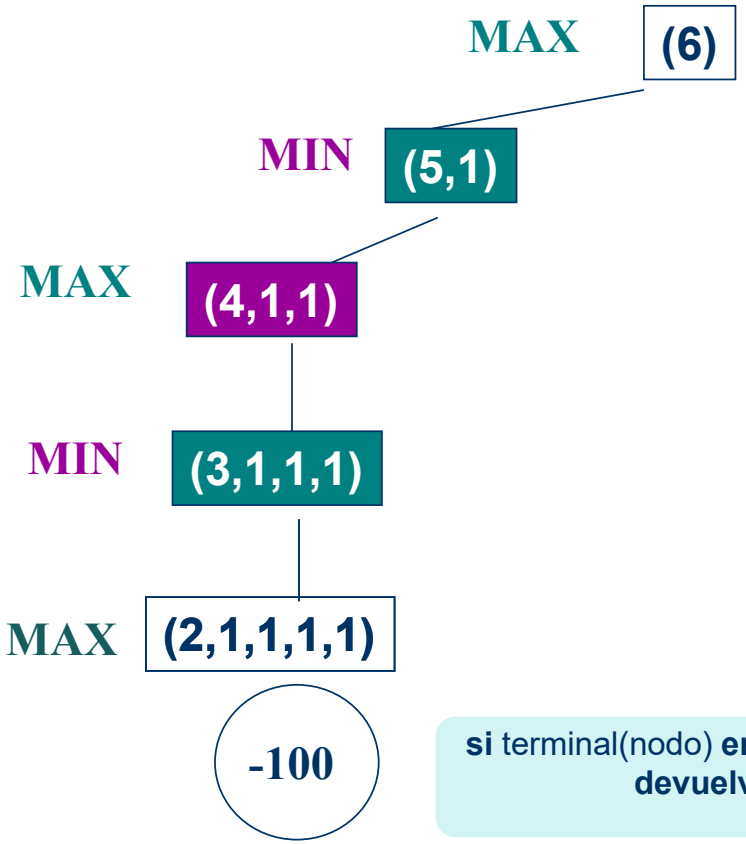
3.7 Traza Minimax



Dividir 2 y 1

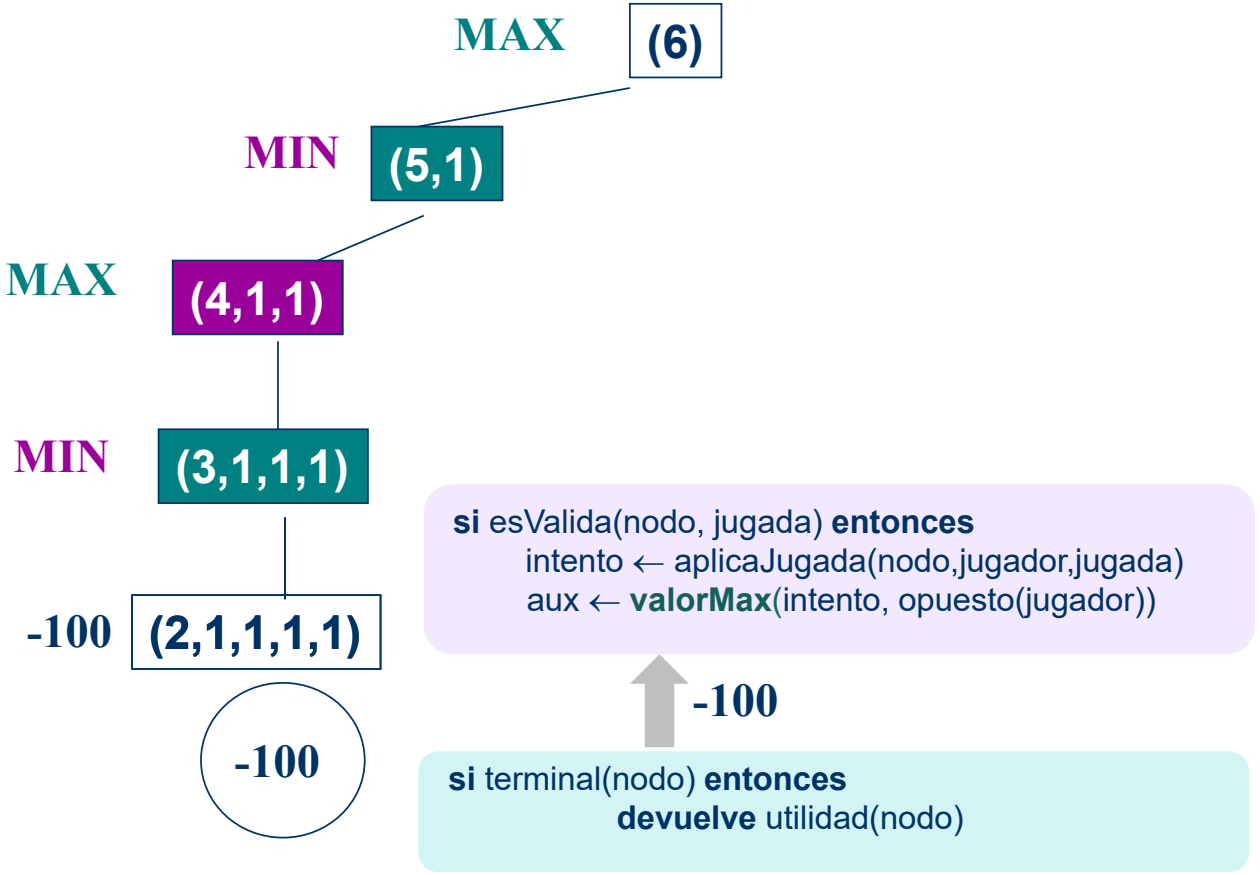
```
si esValida(nodo, jugada) entonces
    intento ← aplicaJugada(nodo,jugador,jugada)
    aux ← valorMax(intento, opuesto(jugador))
```

3.7 Traza Minimax

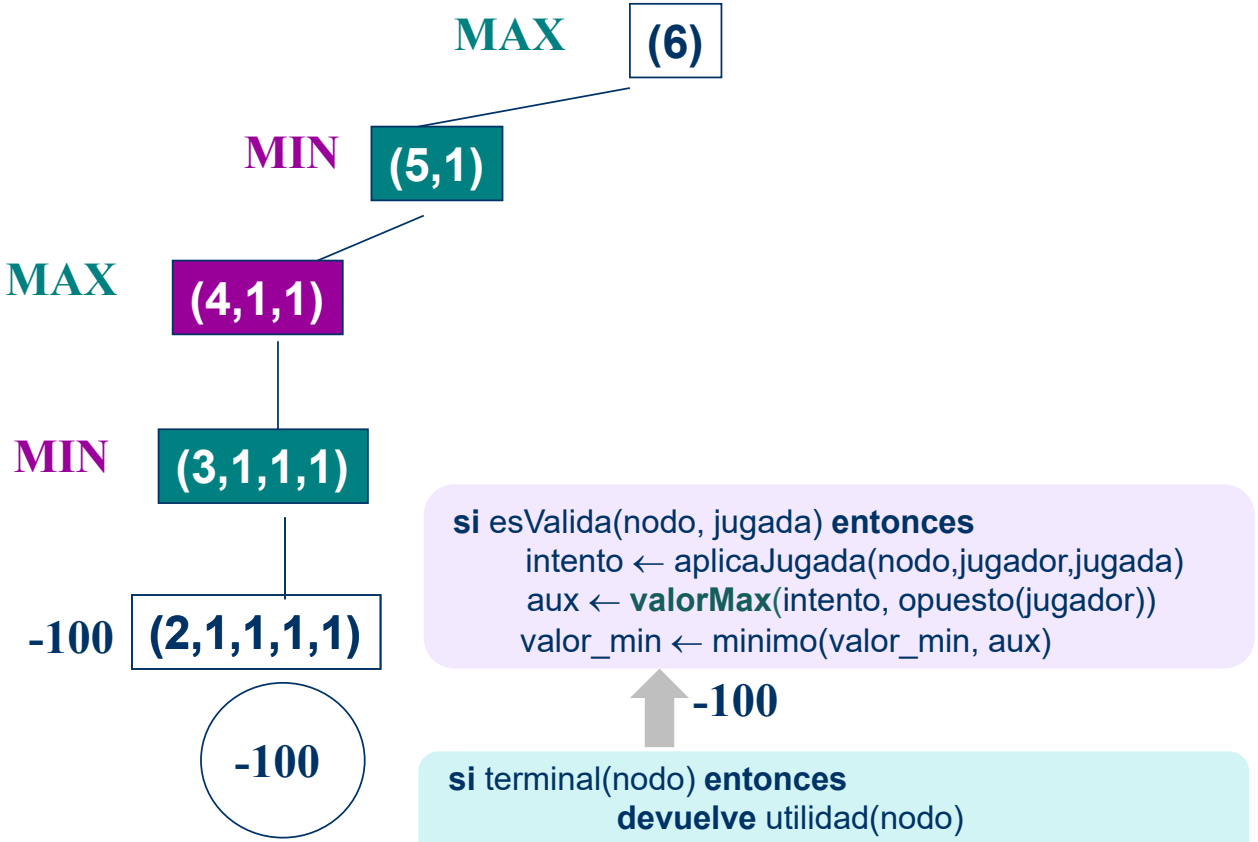



```
si terminal(nodo) entonces
        devuelve utilidad(nodo)
```

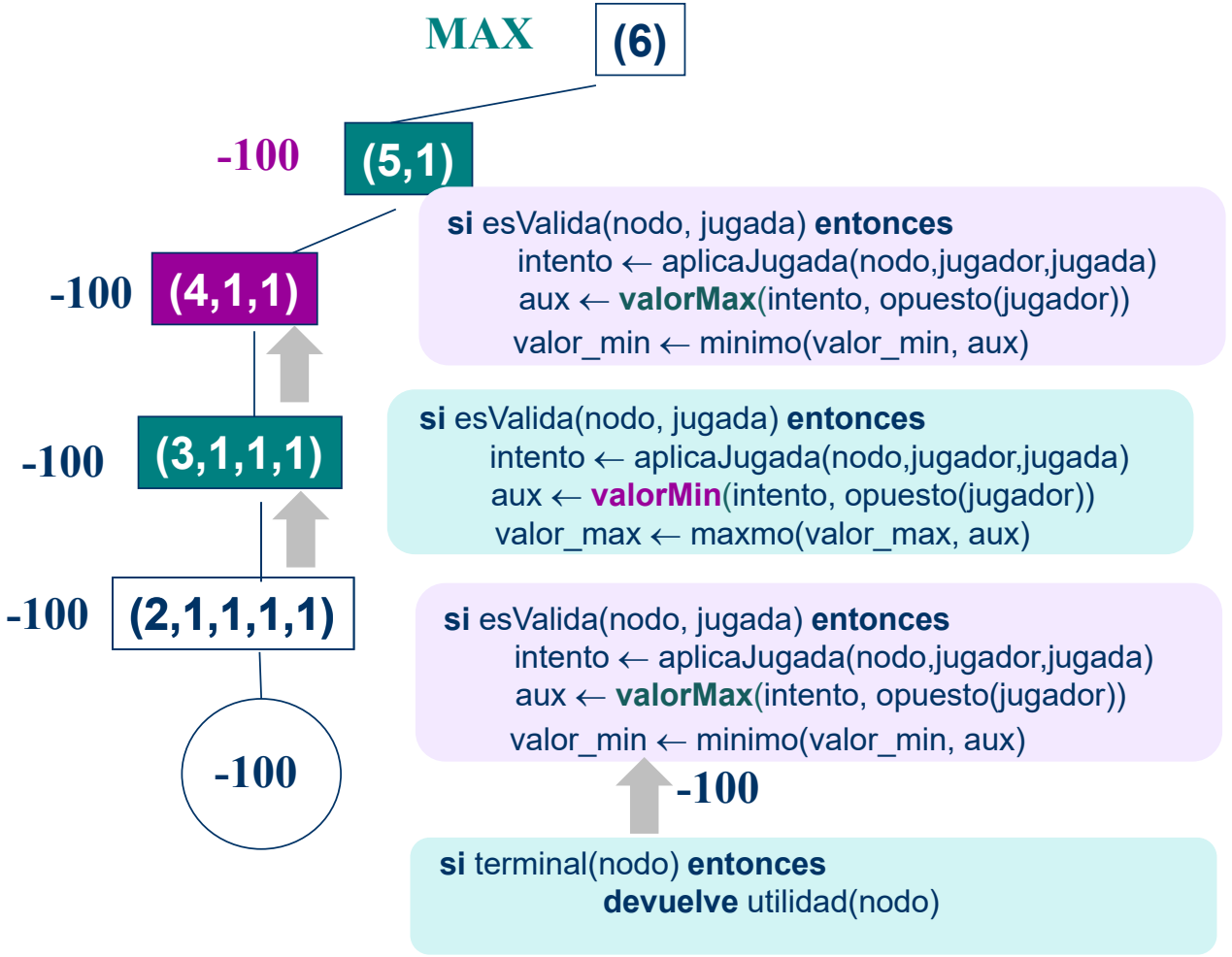
3.7 Traza Minimax



3.7 Traza Minimax

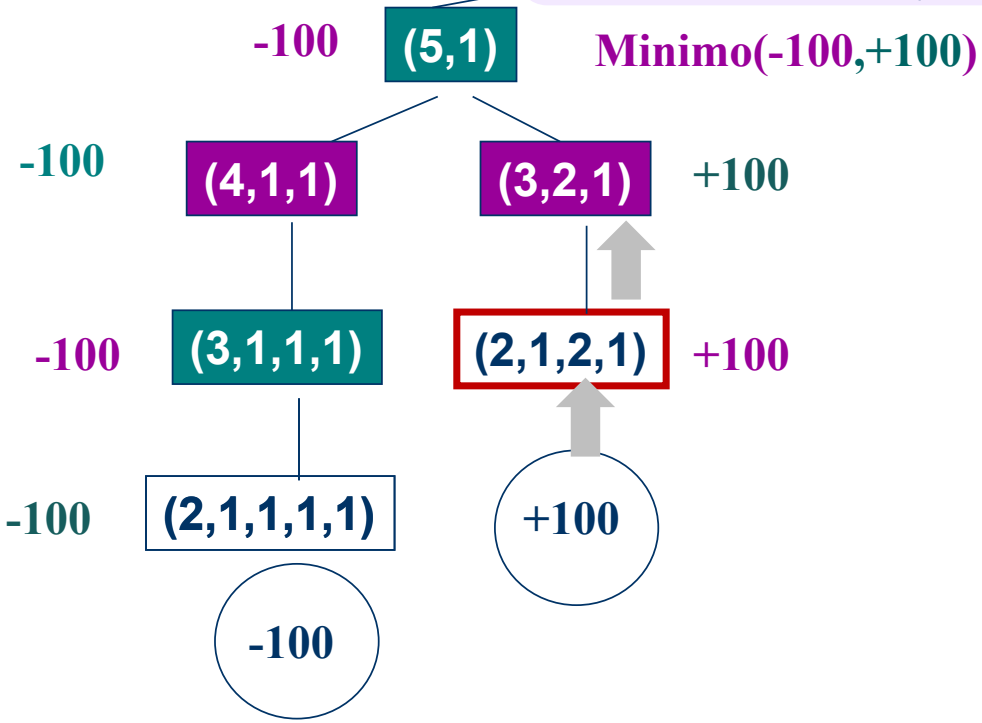


3.7 Traza Minimax

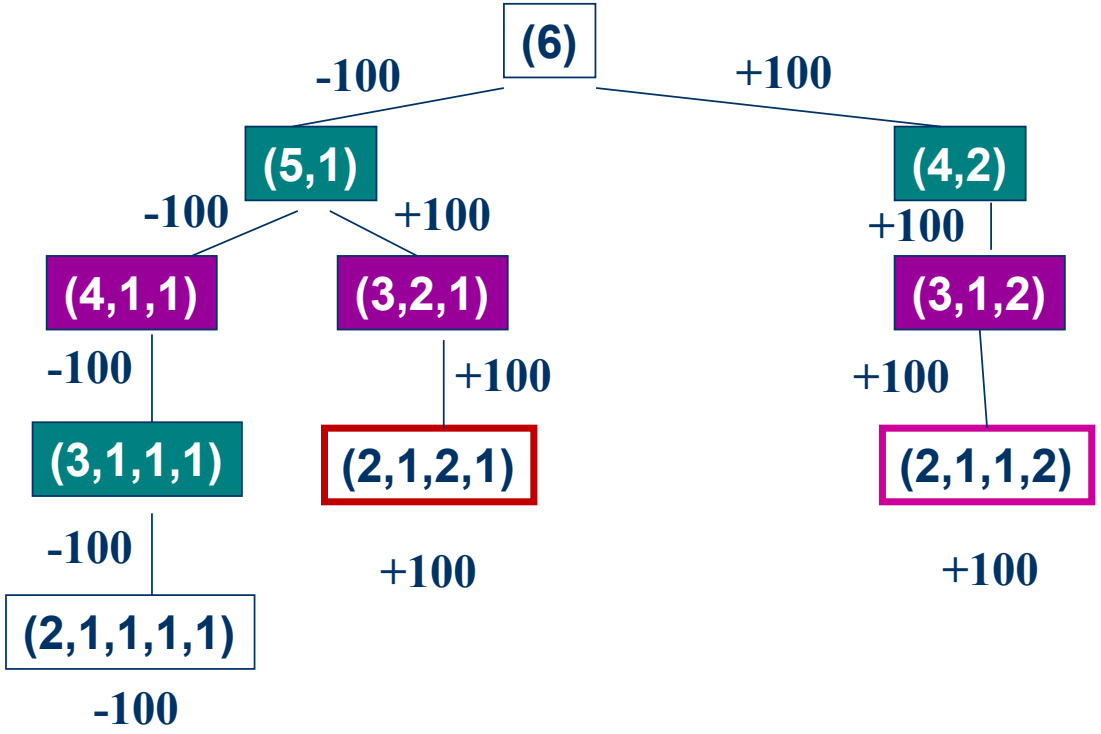


3.7 Traza Minimax

```
si esValida(nodo, jugada) entonces
        intento ← aplicaJugada(nodo,jugador,jugada)
        aux ← valorMax(intento, opuesto(jugador))
        valor_min ← minimo(valor_min, aux)
```



Haga clic para modificar el estilo de título del patrón



F_utilidad(Jugador1)= +100

F_utilidad(Jugador2)= -100

Haga clic para modificar el estilo de título del patrón

Jugada seleccionada por Max

(6)

MAX

Dividir 4 y 2

(4,2)

5. Tic-Tac-Toe

Estados: Tableros NxN (normalmente N=2)

Jugadas: Viene dada por la fila y la columna de una celda del tablero.

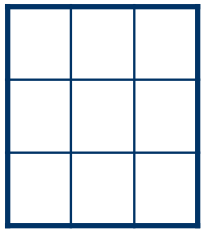
esValida: Comprobar que es una Jugada válida.

aplicaJugada: Colocar la ficha en la celda dada en Jugada.

Test Terminal: Comprobar que se ha acabado el juego.

Función Utilidad: +100 para Max, -100 para Min, 0 empate (por ejemplo)

Estado



Jugada:

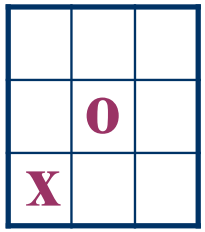
fila, columna

5. Tic tac toe

Estados: Tableros NxN

(normalmente N=2)

Estado



Representación Interna:
X=Max (+1) O=Min (-1)

```
@dataclass
class Nodo:
    tablero: np.array
    vacias: int
    N: int
```

Jugadas: Viene dada por la fila y la columna de una celda del tablero.

```
@dataclass
class Jugada:
    x: int
    y: int
```

5. Tic tac toe

esValida: Comprobar que es una Jugada Válida

1. Fila y Columna están dentro de los límites del tablero
2. La celda está vacía

aplicaJugada: Colocar la ficha en la celda dada en Jugada

1. Rellenar la celda con el valor dado al Jugador (Max o Min)
2. Actualizar el contador de celdas vacías

Test Terminal: Comprobar si se ha acabado el juego.

1. Hay 3 fichas alineadas en fila, columna o diagonal.
2. No quedan celdas vacías.

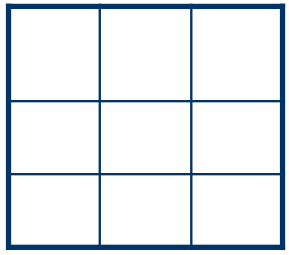
Función Utilidad: +100 para Max, -100 para Min,

0 empate (por ejemplo)

1. Si hay 3 celdas en fila, diagonal o columna con la marca del mismo jugador.
2. Si el tablero está lleno pero no gana nadie, devuelve valor de Empate.

5.2 Árbol para Tic-Tac-Toe

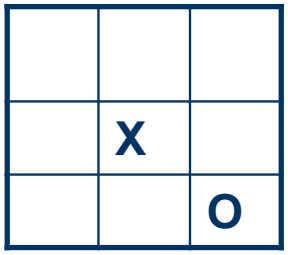
Turno de MAX

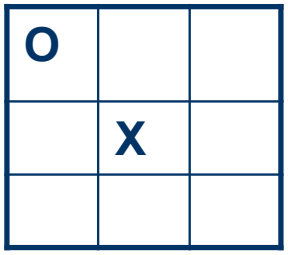


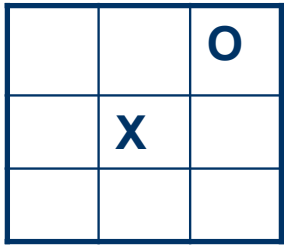
5.1 Tic-Tac-Toe

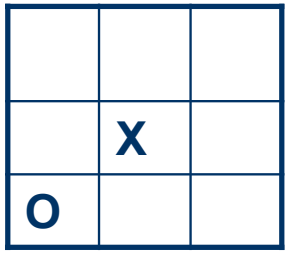
- Por simetría hay estados que se consideran idénticos:

Por ejemplo en las DIAGONALES

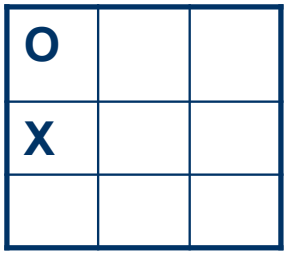


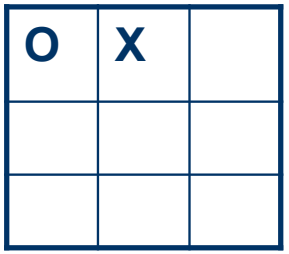




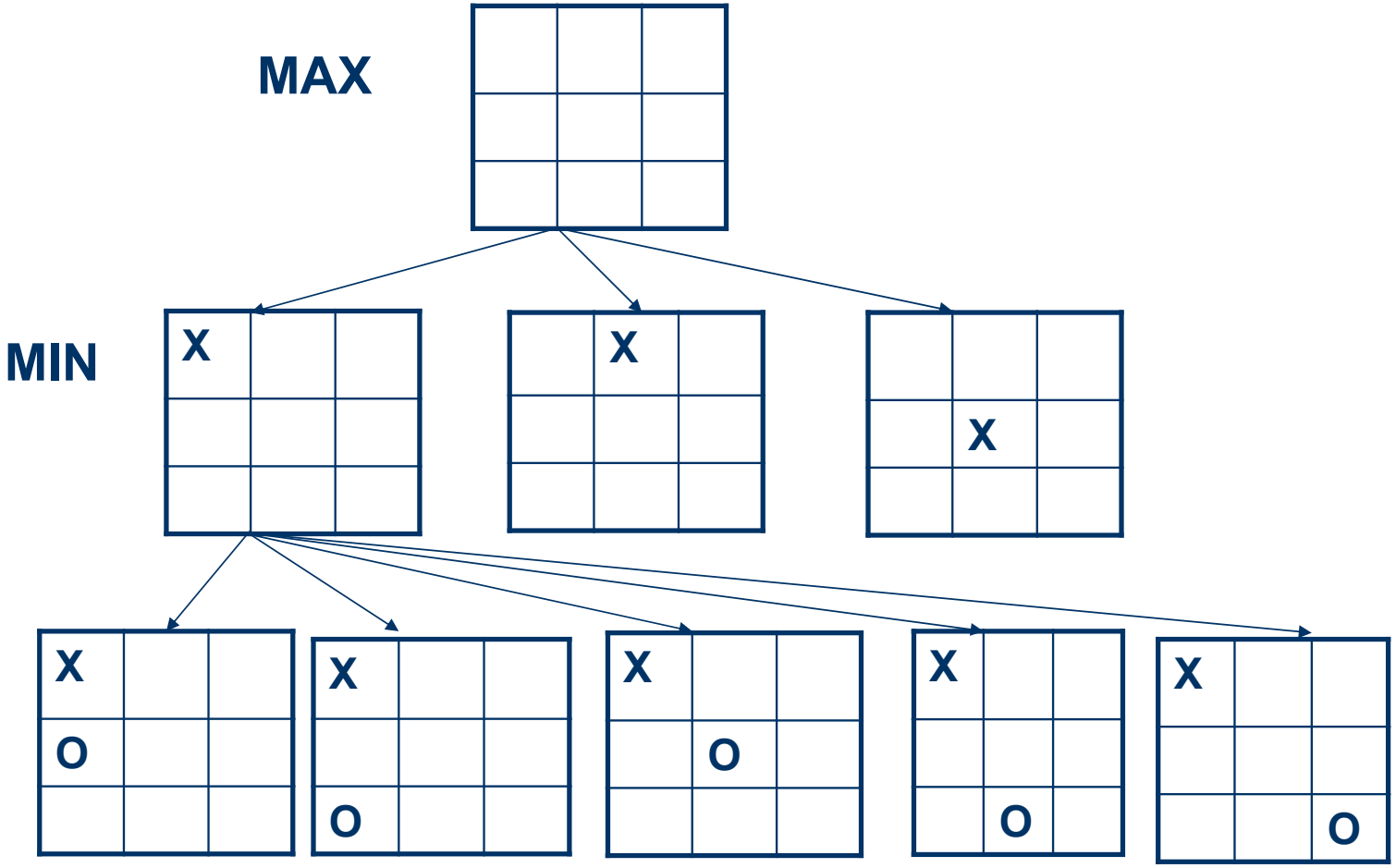


Otro caso





5.1 Tic-Tac-Toe



Análisis de Juegos

6.1 Características de las Funciones Heurísticas

NO representa ningún coste de llegar a la solución, ni es una estimación de la distancia al objetivo en pasos.

Expresa cómo de buena es la situación actual del juego para un jugador dado (Max o Min).

Debe reflejar de manera fiable las posibilidades actuales de ganar.

Debe estar de acuerdo con la función de utilidad para los nodos terminales.

No debe ser muy complejas (evitar altos costes computacionales).

6. Funciones Heurísticas para Nodos No Terminales

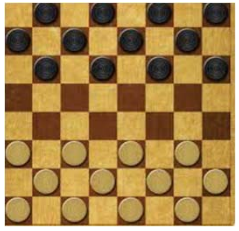
- Se utilizan cuando no se realiza la exploración completa del árbol de juego.
- Idea básica: estimar cómo de buena es su situación y cómo es para su oponente y entonces restar las puntuaciones de los jugadores:

  - Ajedrez:

    (valor blancas – valor negras)

  - Damas:

    (nº blancas – nº negras)



¿3-en-raya?

X	O	
	O	
	X	X

6.2 Función Heurística para el TicTacToe

- Tic-Tac-Toe: **Posibilidades de hacer fila, columna o diagonal:**
  - Nº de columnas, filas y diagonales con X y sin O
  - Nº de columnas, filas y diagonales con O y sin X

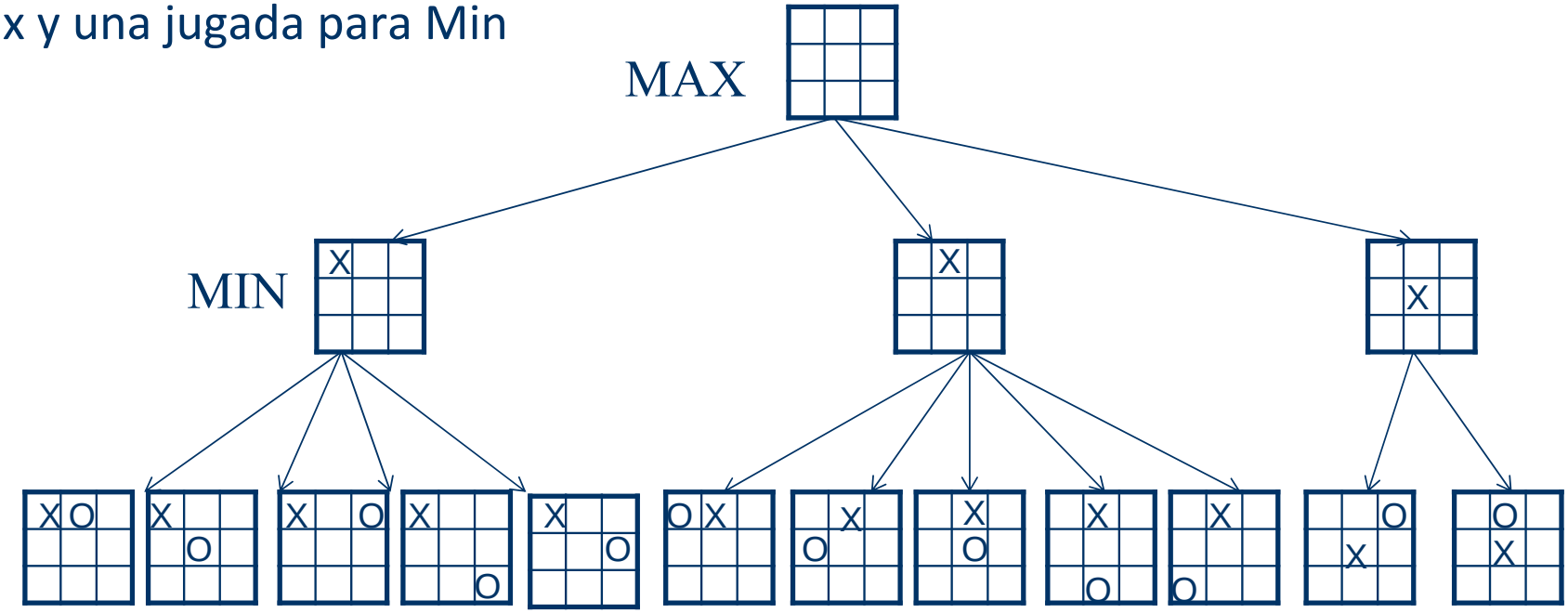
X	O	
	O	
	X	X

MAX : columna 1, columna 3, fila 3
MIN: fila 2, diagonal 2

f = 3 -2 =1

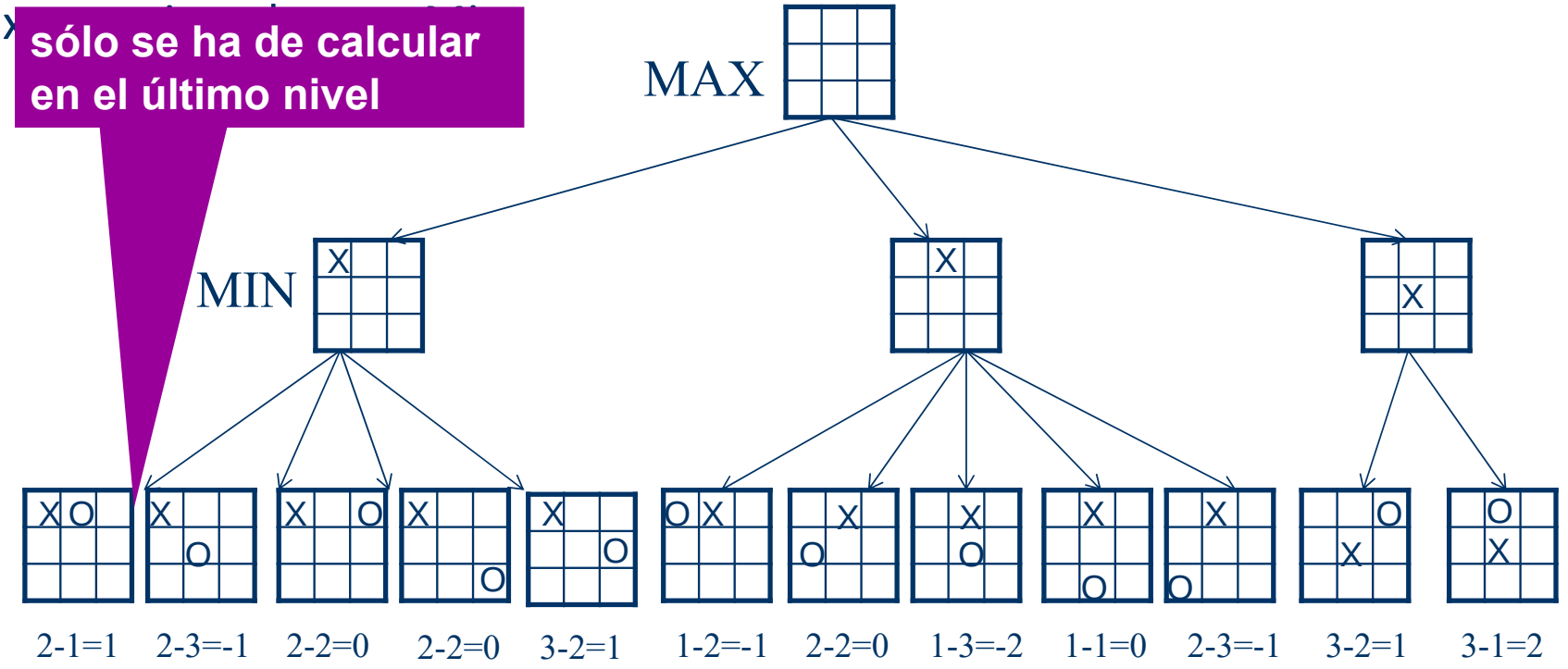
6.2 Función Heurística para TicTacToe

- Calcula la función heurística en este árbol donde sólo se considera una jugada para Max y una jugada para Min

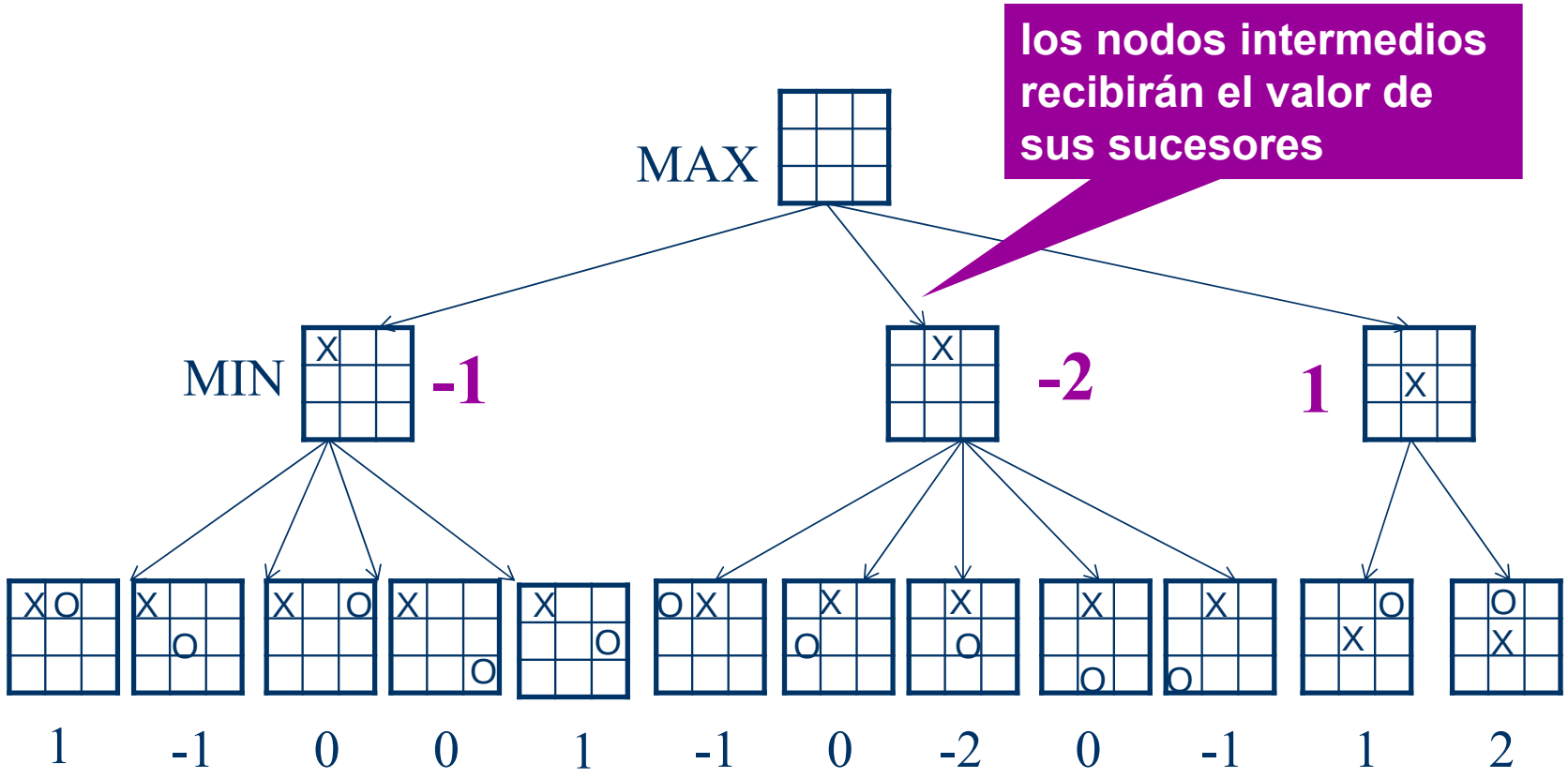


6.4 Funciones Heurísticas para Nodos No Terminales

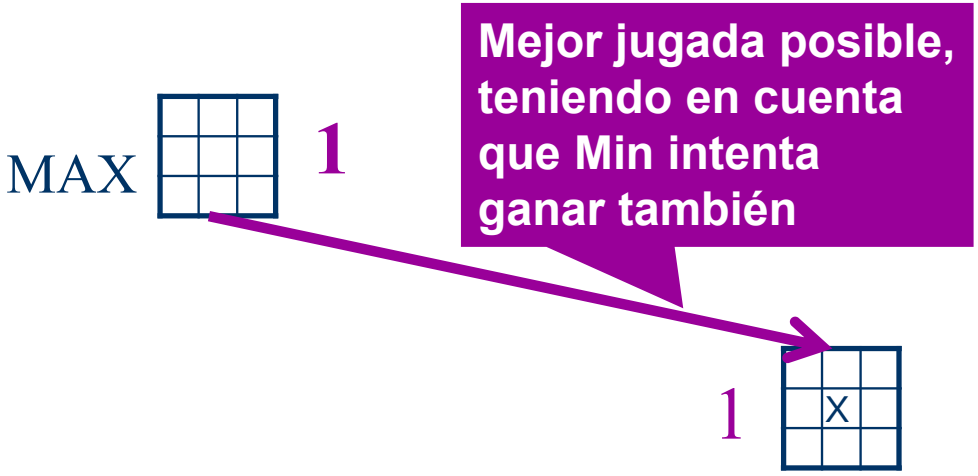
- Calcula la función heurística en este árbol donde sólo se considera una jugada para Max



6.5 Aplicación de la estrategia MiniMax con una Función Heurística



6.6 Aplicación de la estrategia MiniMax con una Función Heurística



7. Poda α-β

- A menudo es posible calcular el valor minimax sin tener que evaluar todos los nodos hoja.
- Puede que una parte del subárbol no vaya a añadir información sobre la evaluación minimax realizada.
- Los nodos cuya evaluación puede obviarse se **podan.**

7.1 Poda α-β

- α: Valor de la mejor opción hasta ahora para MAX,

  se actualiza cuando un nuevo valor, v, es $v > \alpha$

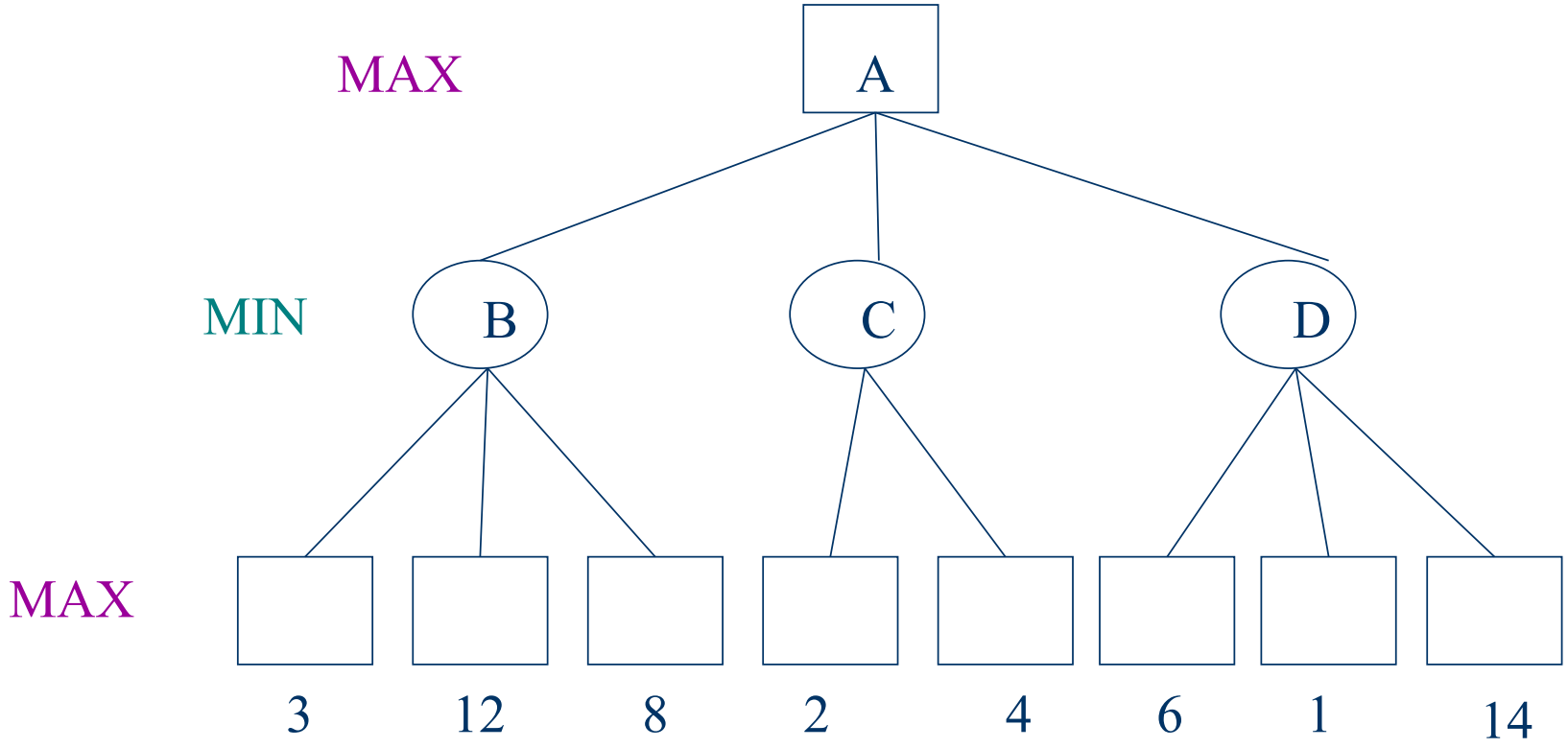
- β : Valor de la mejor opción hasta ahora para MIN, se actualiza cuando un nuevo valor, v, es $v < \beta$

Estrategia: **Búsqueda en Profundidad**

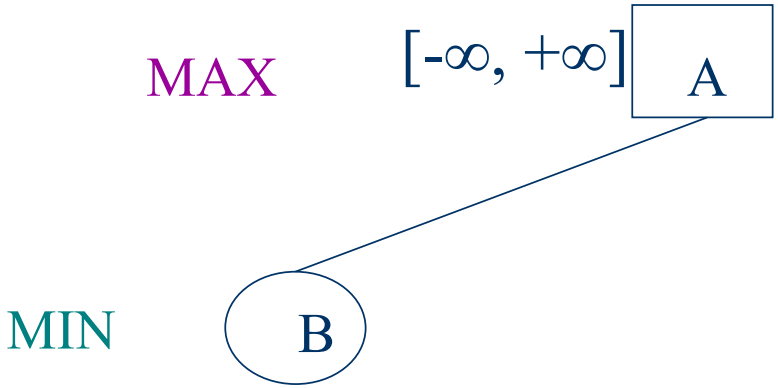
- Actualizar los valores α-β a medida que recorre el árbol y encuentra valores Minimax mejores.
- Podar las ramas en donde los valores Minimax no pueden mejorar los valores α-β actuales:

  Poda cuando en un nodo $\alpha >= \beta$

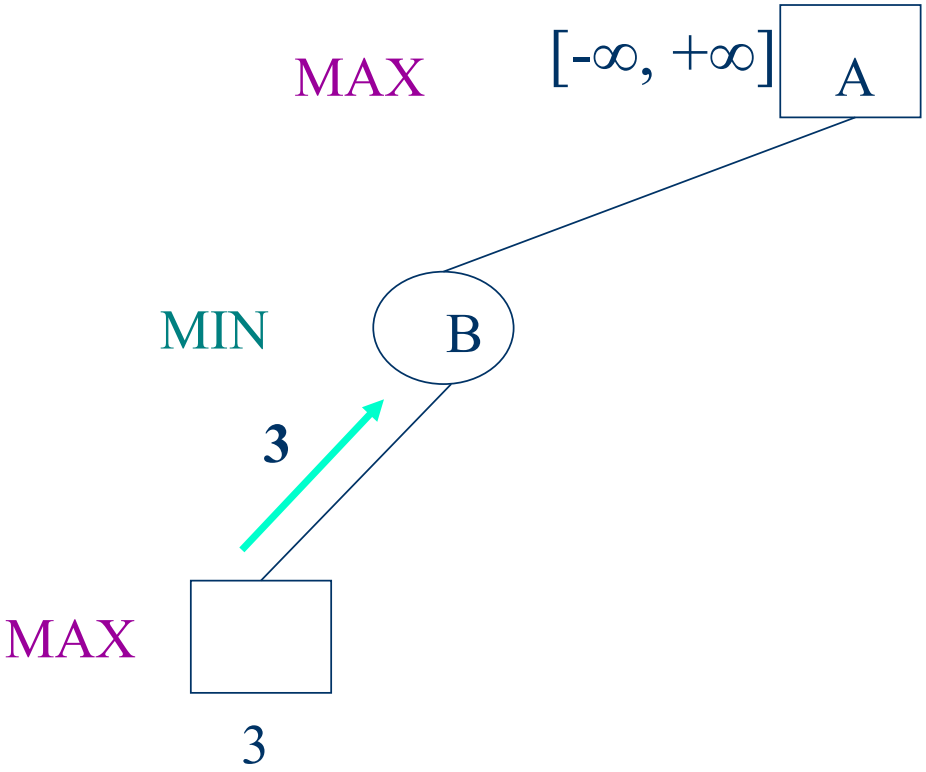
8. Ejemplo 3: Poda α-β



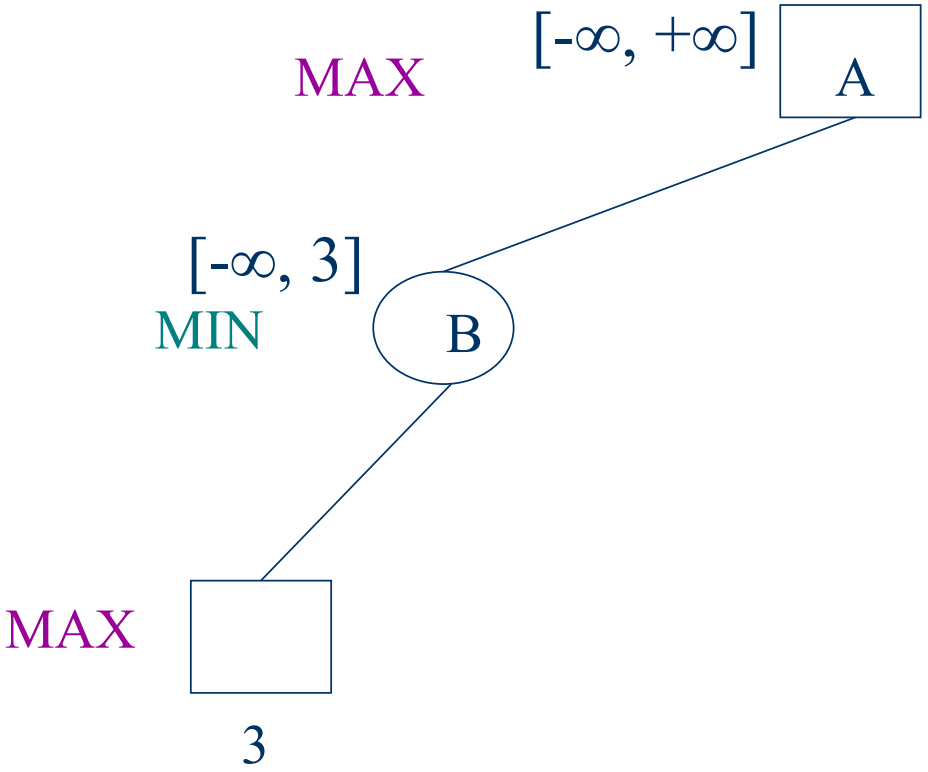
8. Ejemplo 3: Poda α-β



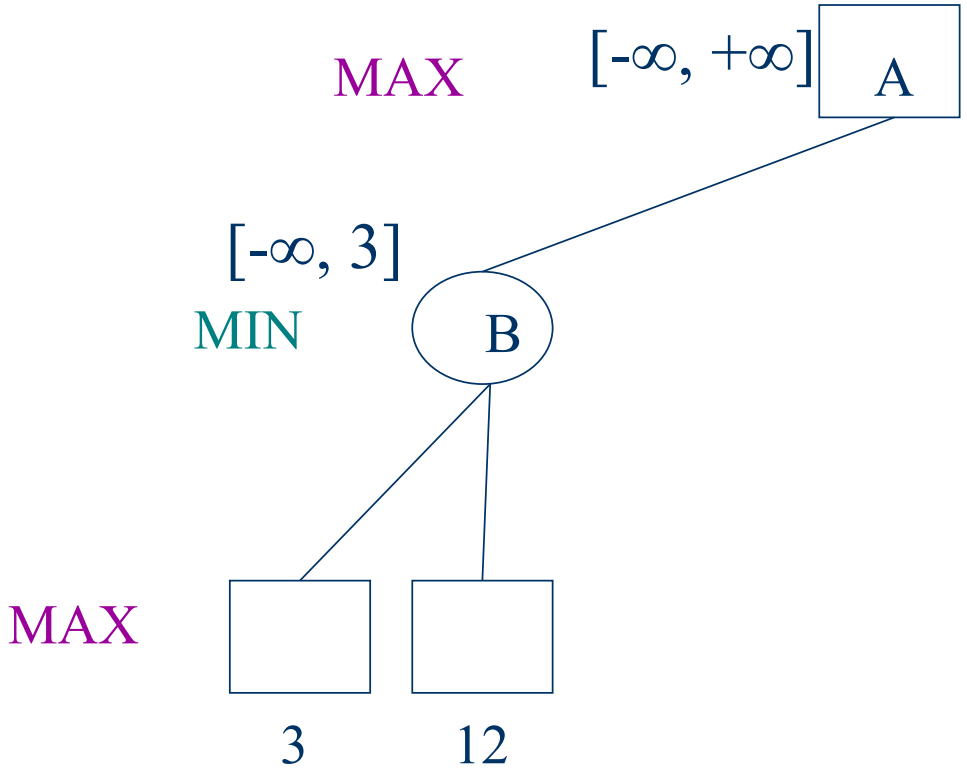
8. Ejemplo 3: Poda α-β



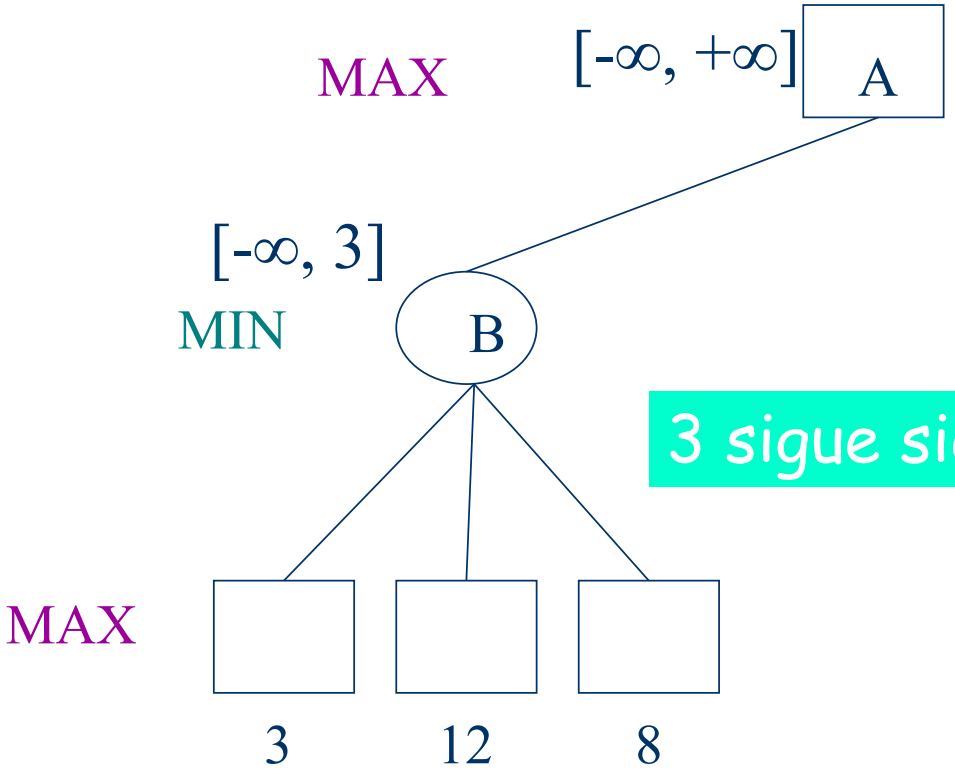
8. Ejemplo 3: Poda α-β



8. Ejemplo 3: Poda α-β

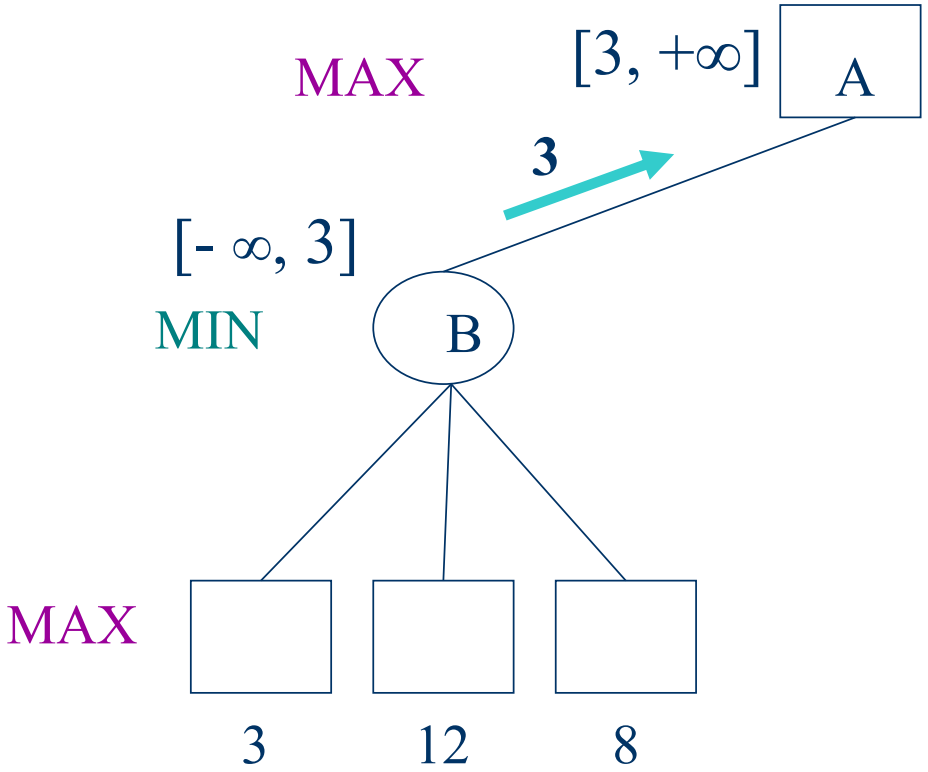


8. Ejemplo 3: Poda α-β

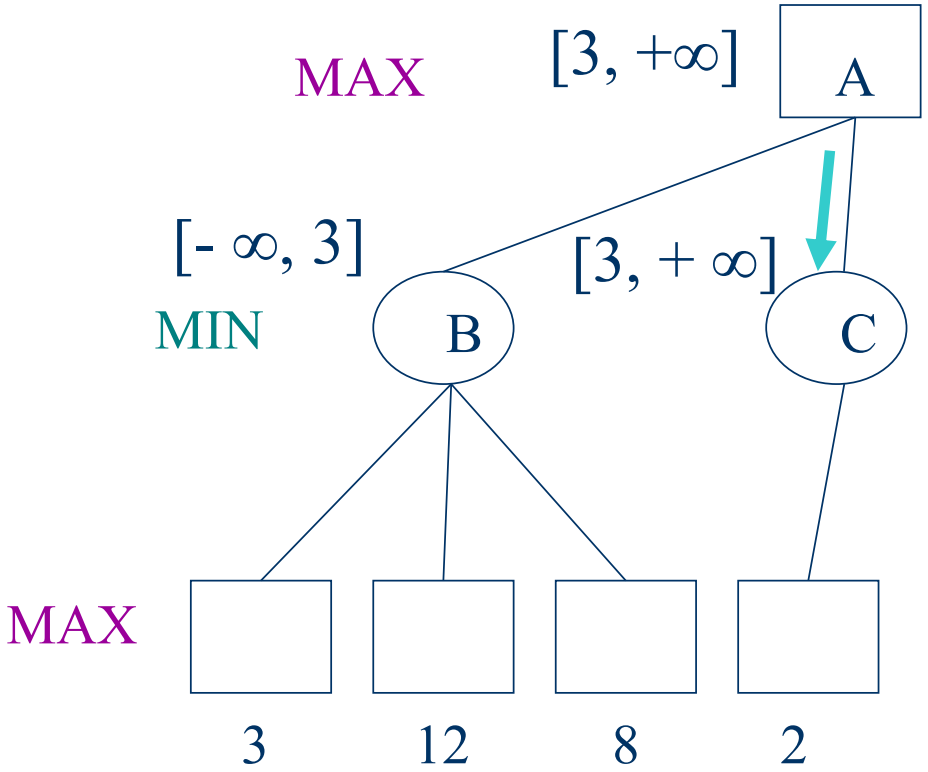


3 sigue siendo el menor valor

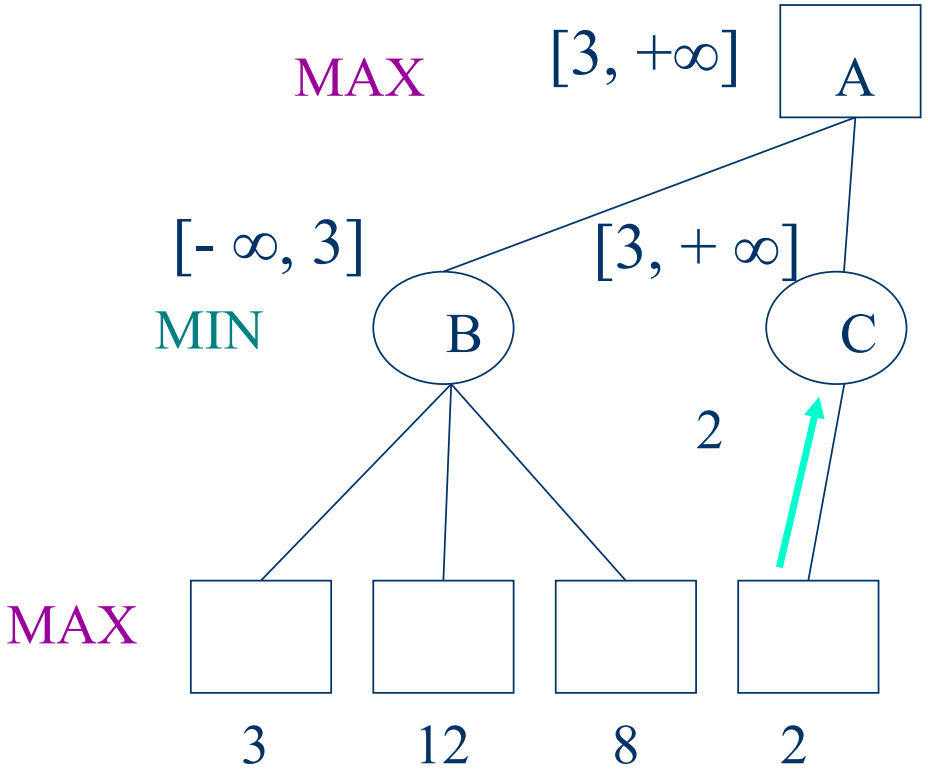
8. Ejemplo 3: Poda α-β



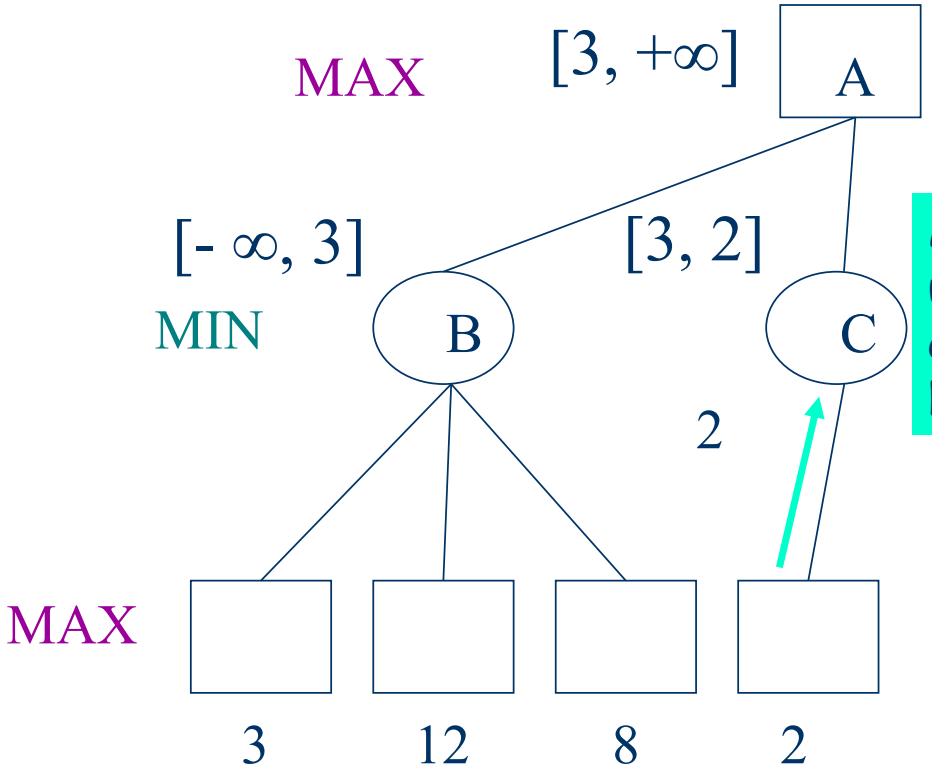
8. Ejemplo 3: Poda α-β



8. Ejemplo 3: Poda α-β

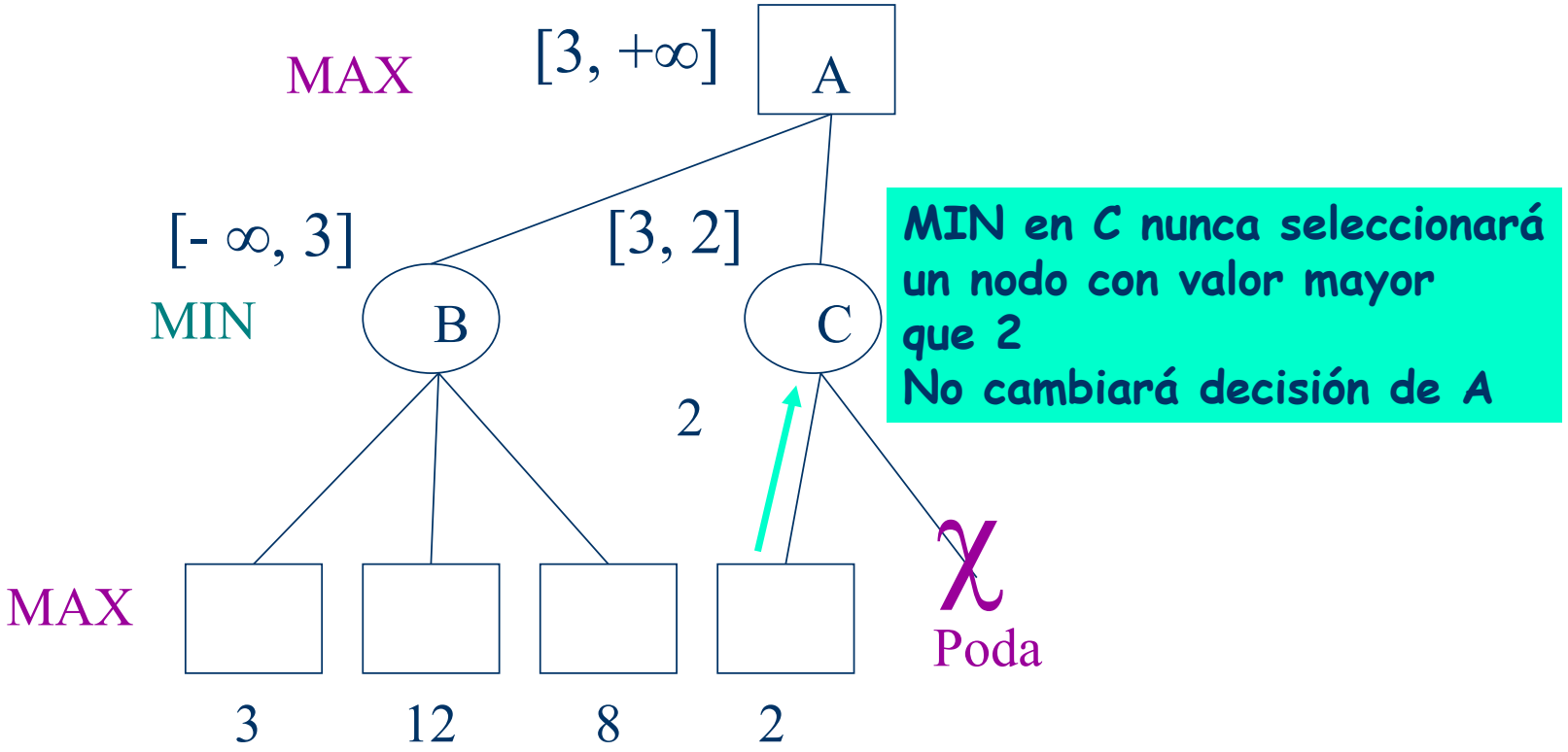


8. Ejemplo 3: Poda α-β



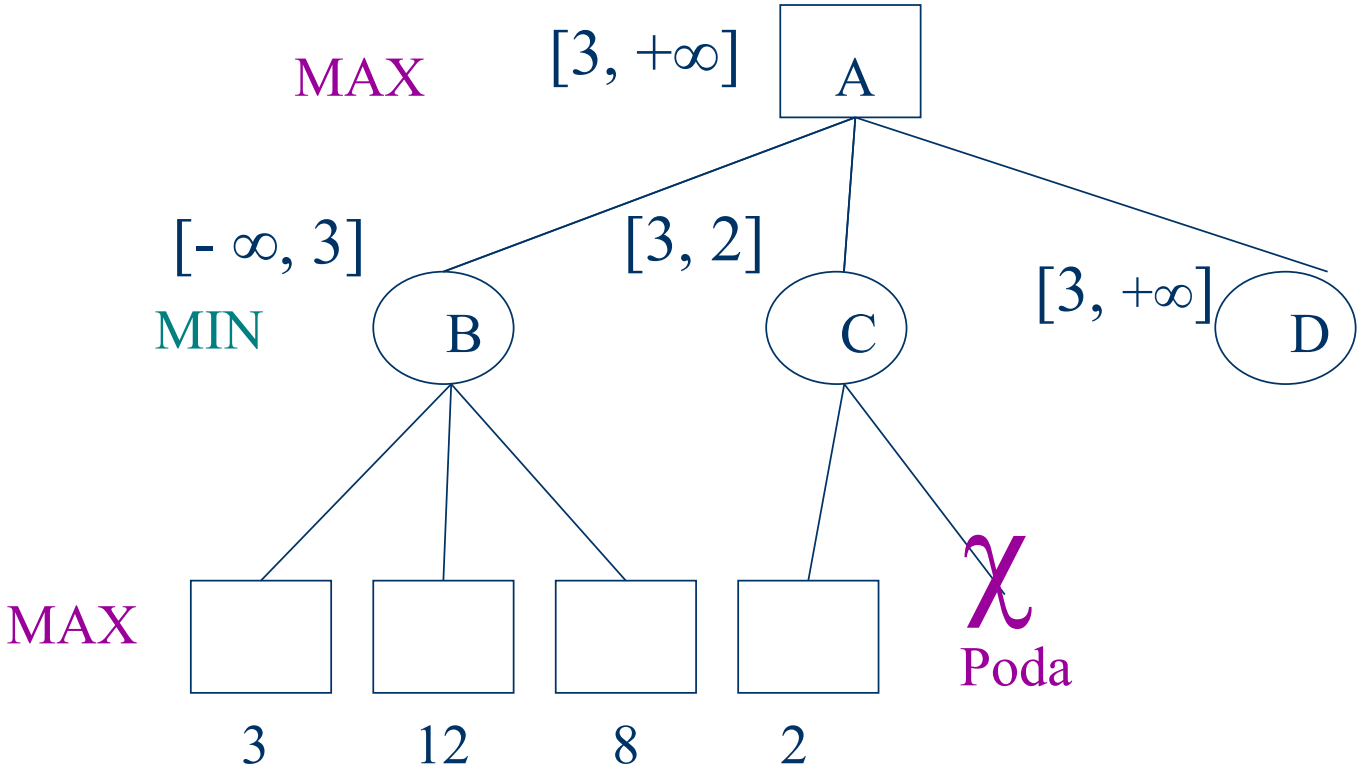
MIN en C nunca seleccionará un nodo con valor mayor que 2
No cambiará decisión de A

8. Ejemplo 3: Poda α-β

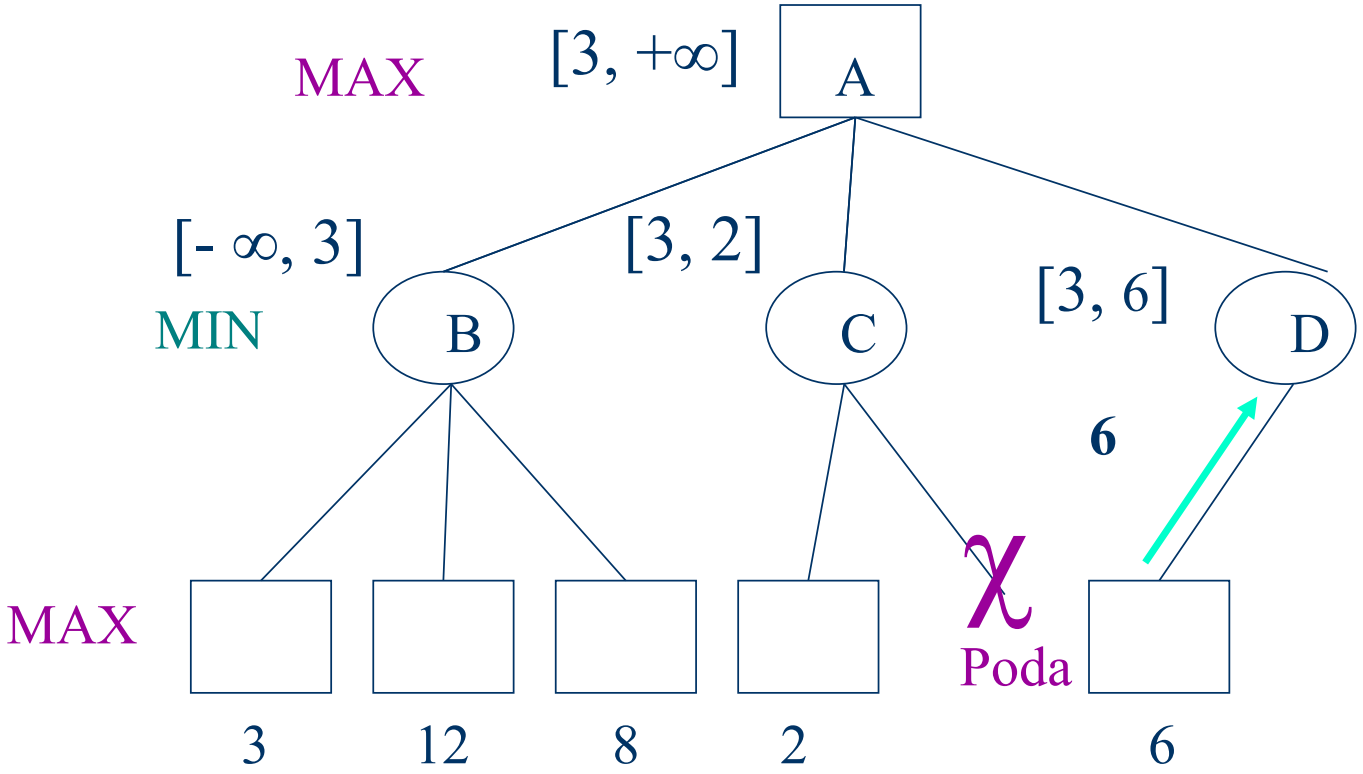


MIN en C nunca seleccionará un nodo con valor mayor que 2
No cambiará decisión de A

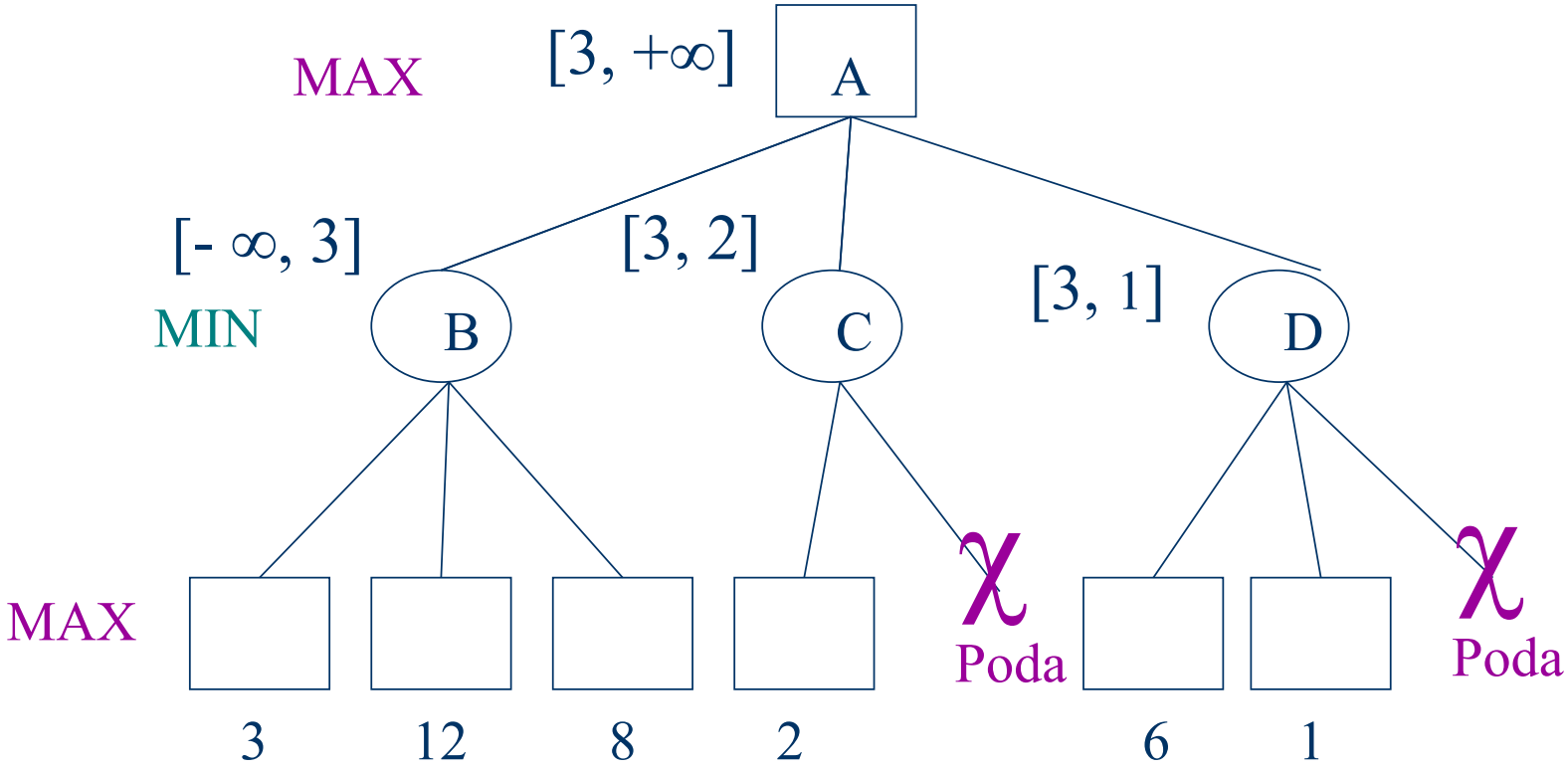
8. Ejemplo 3: Poda α-β



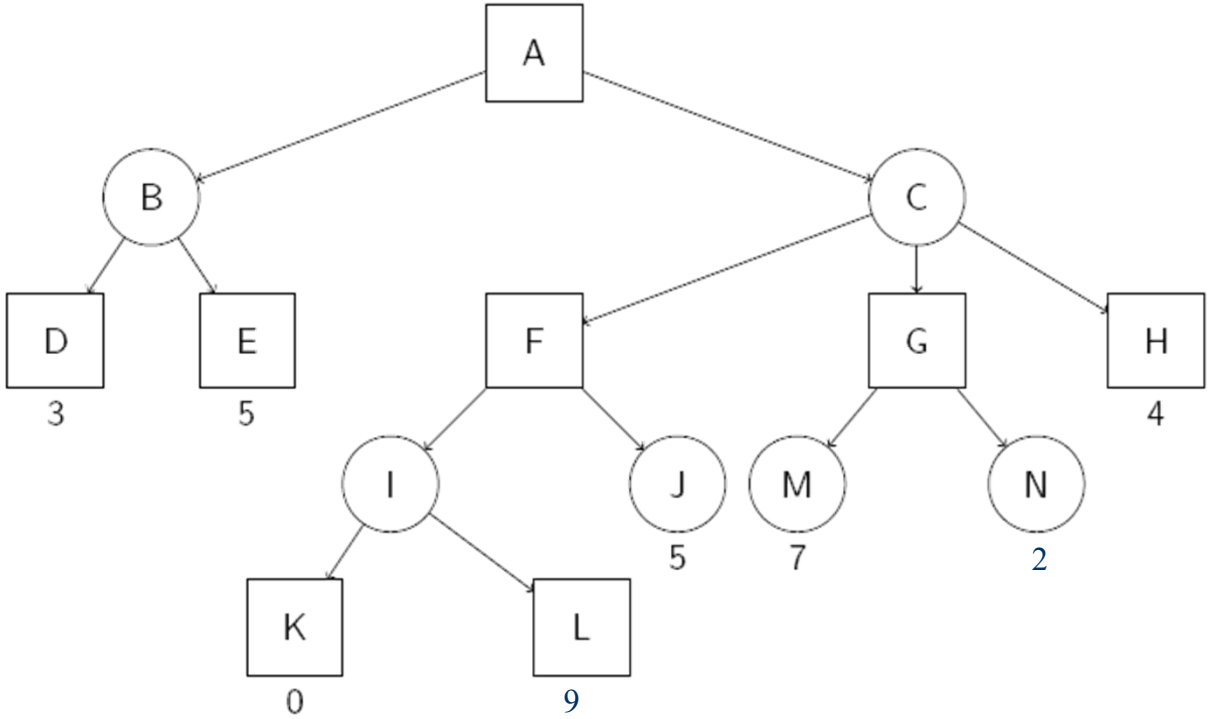
8. Ejemplo 3: Poda α-β



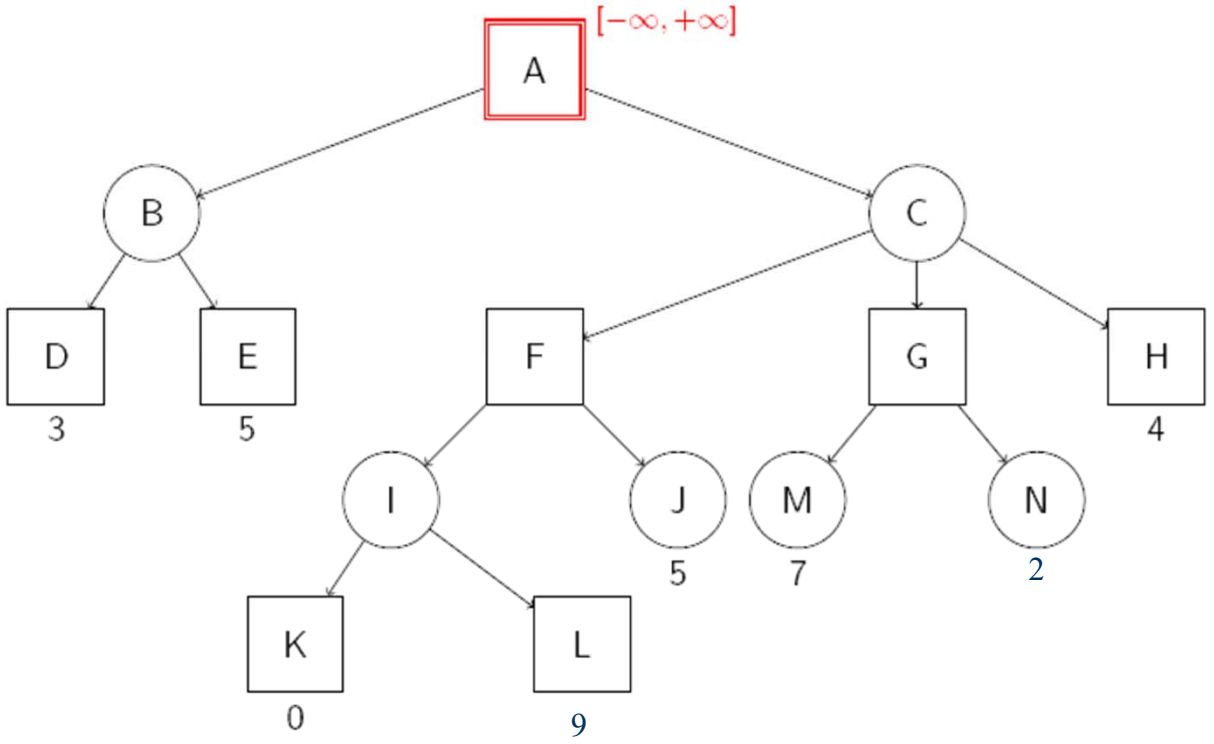
8. Ejemplo 3: Poda α-β



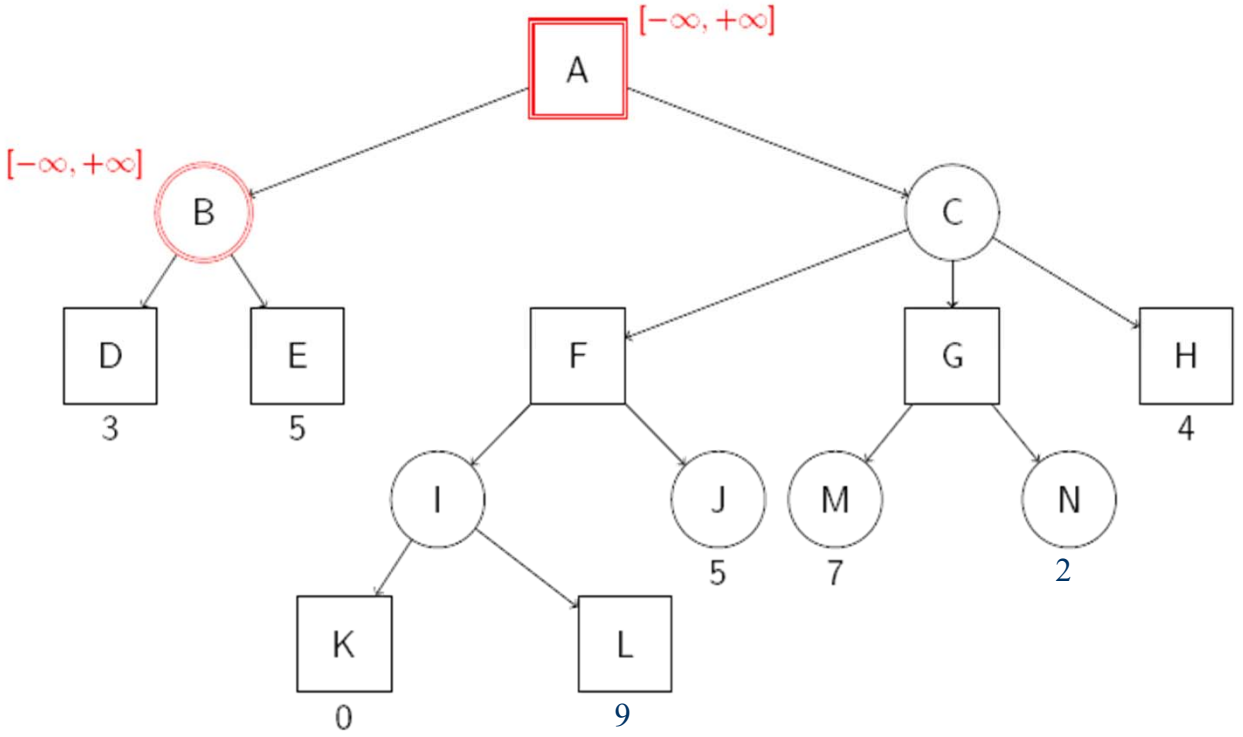
8.2 Ejemplo 4: Poda α-β



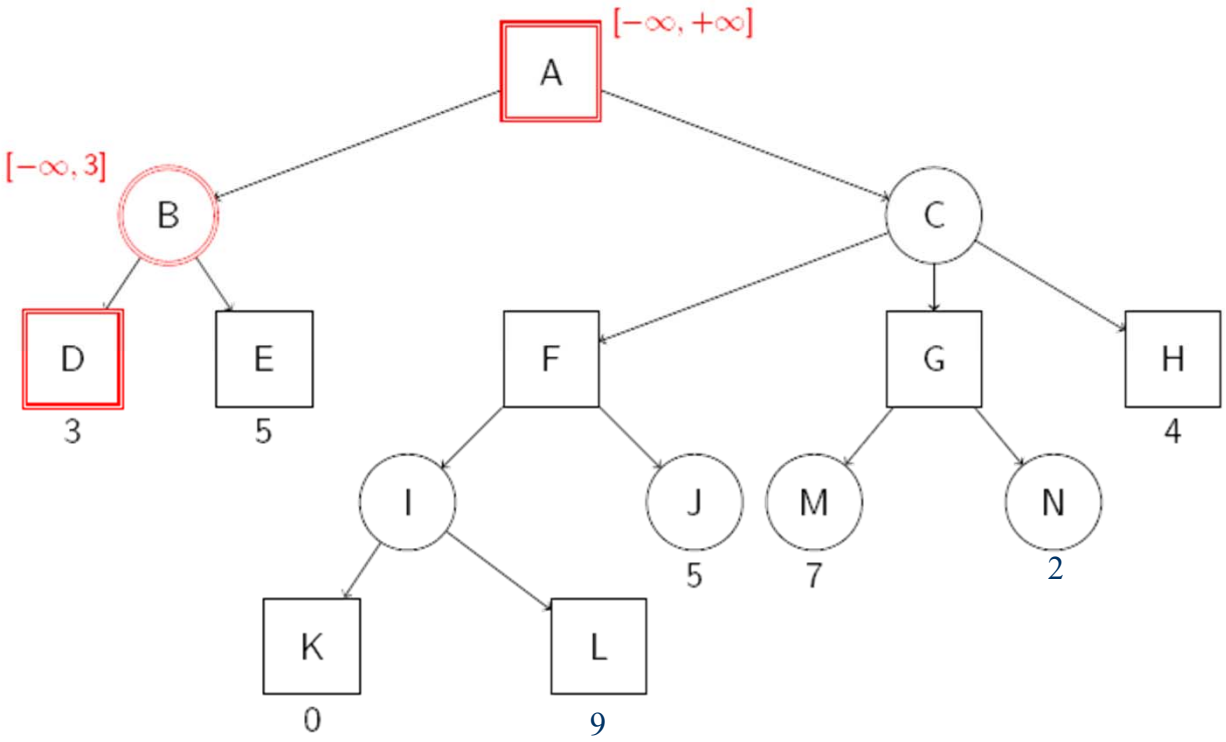
8. Ejemplo 4: Poda α-β



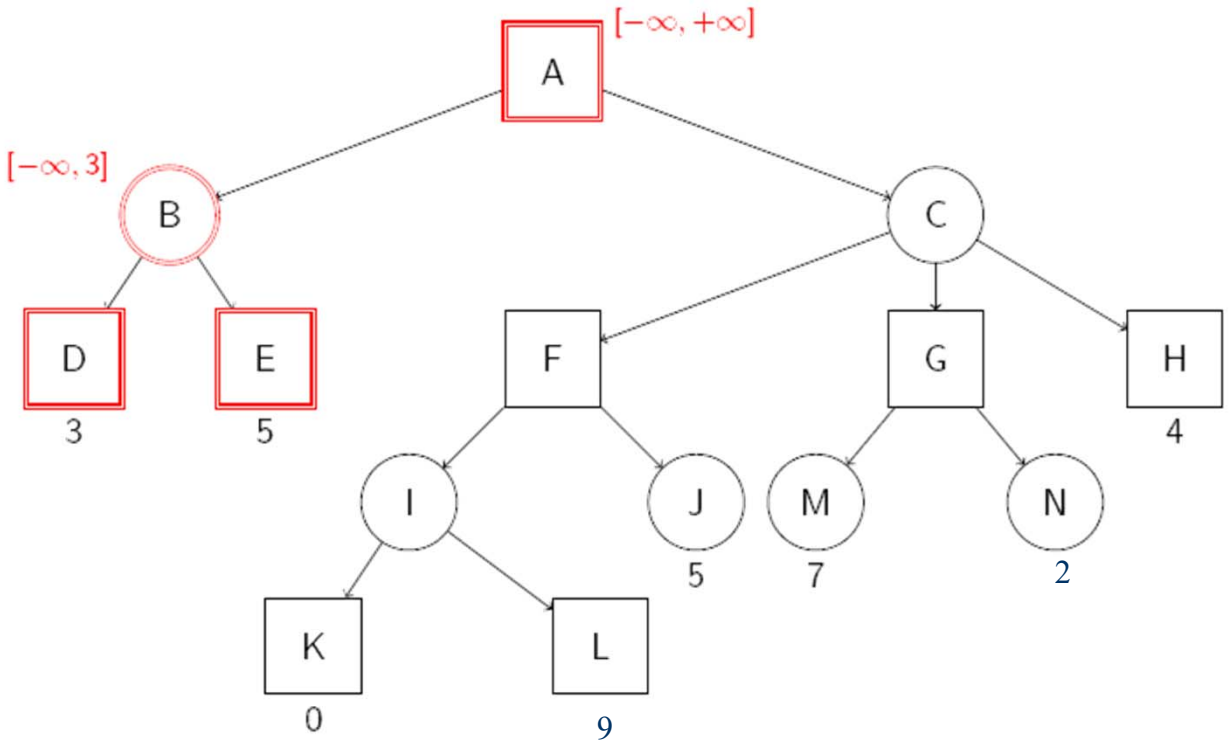
8. Ejemplo 4: Poda α-β



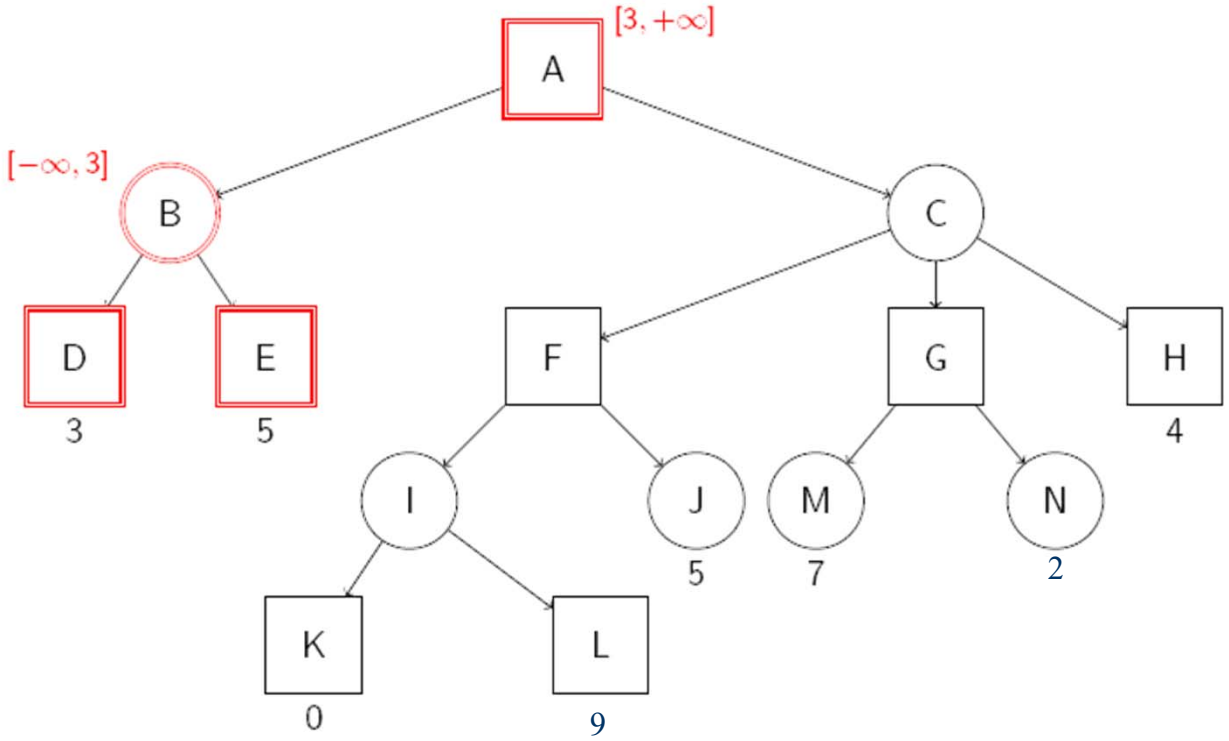
8. Ejemplo 4: Poda α-β



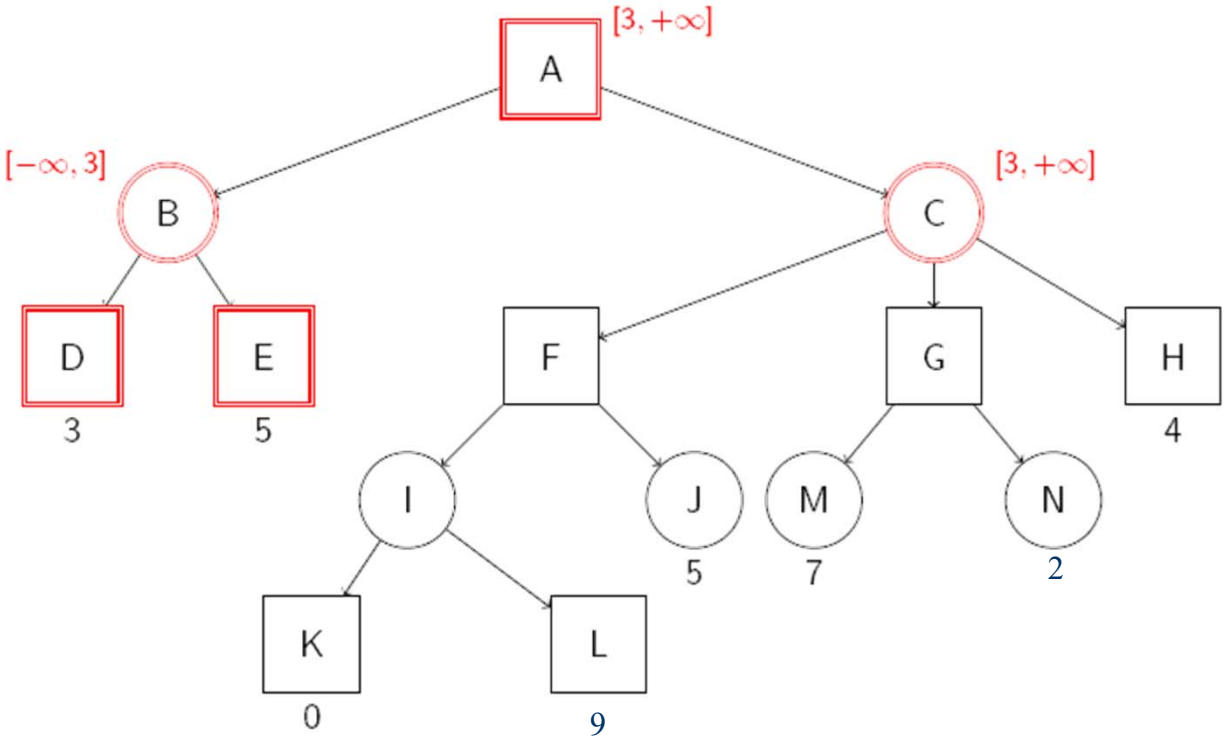
8. Ejemplo 4: Poda α-β



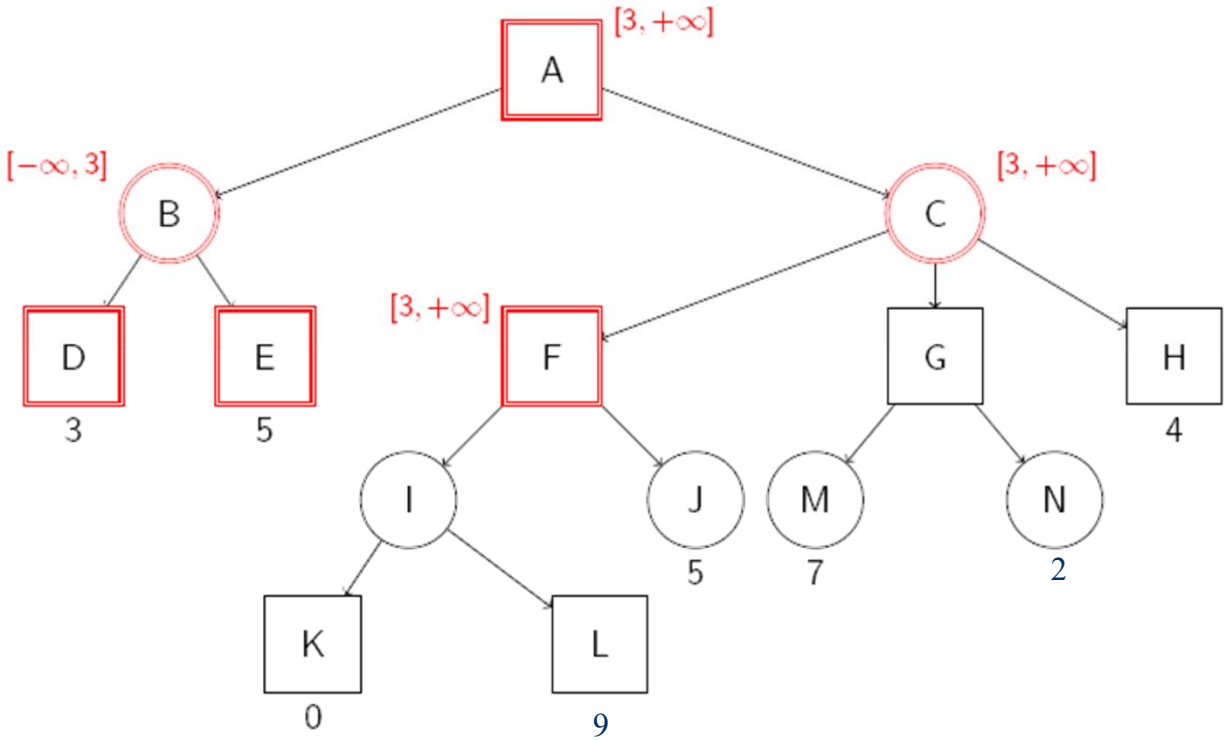
8. Ejemplo 4: Poda α-β



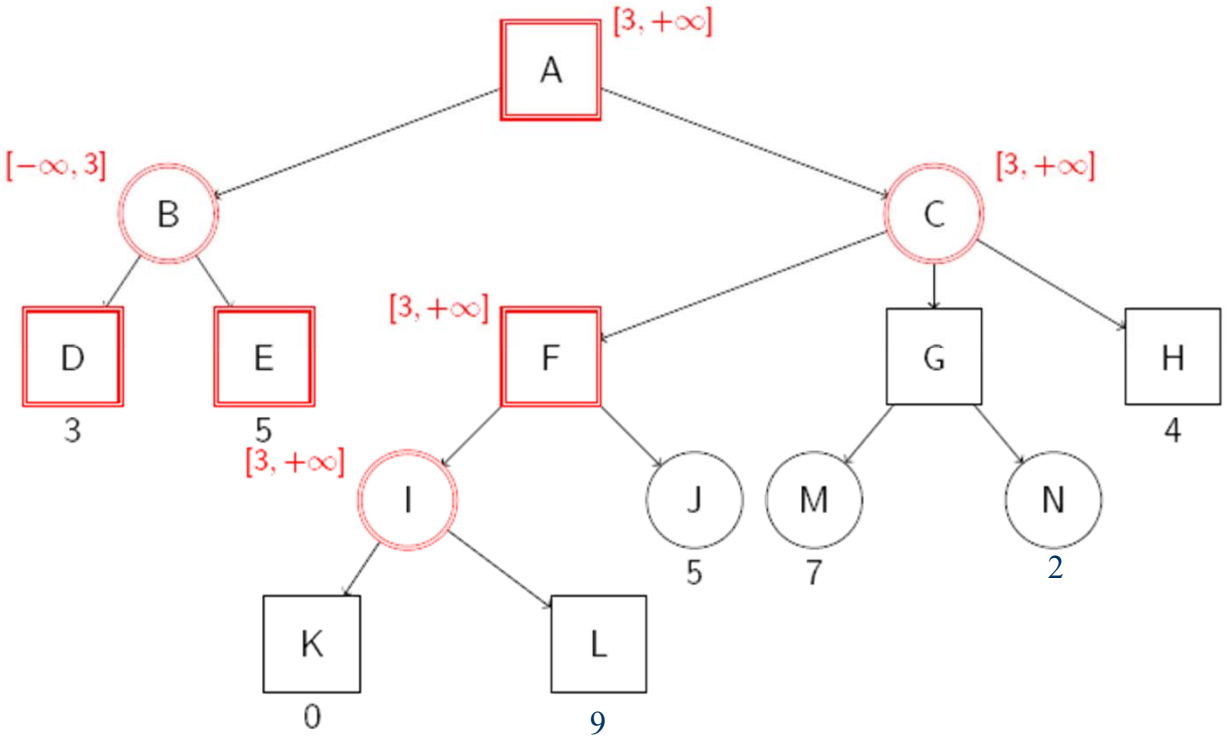
8. Ejemplo 4: Poda α-β



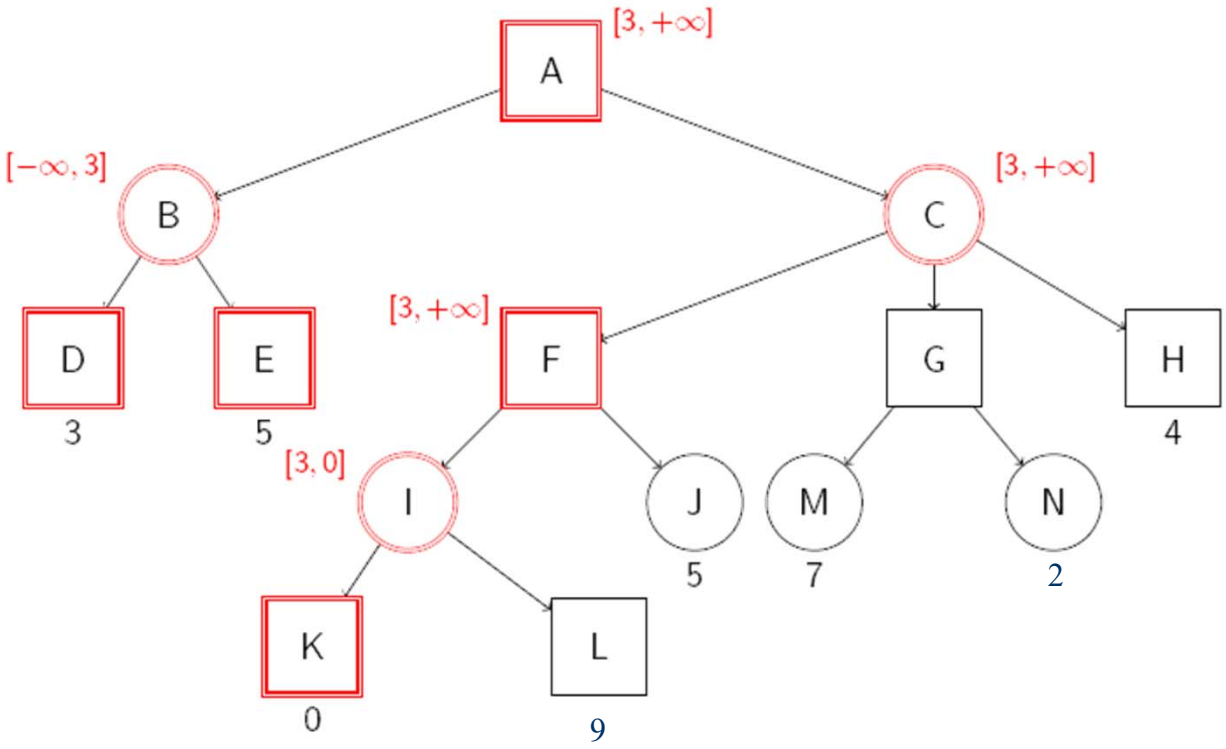
8. Ejemplo 4: Poda α-β



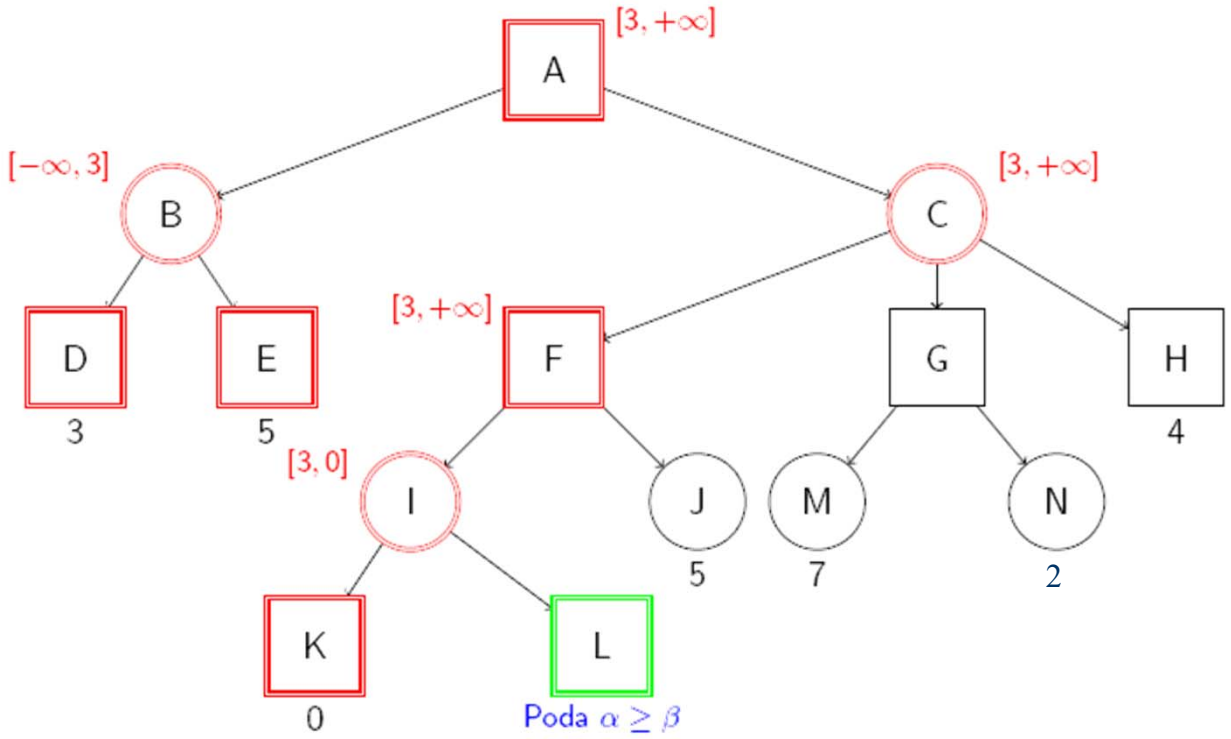
8. Ejemplo 4: Poda α-β



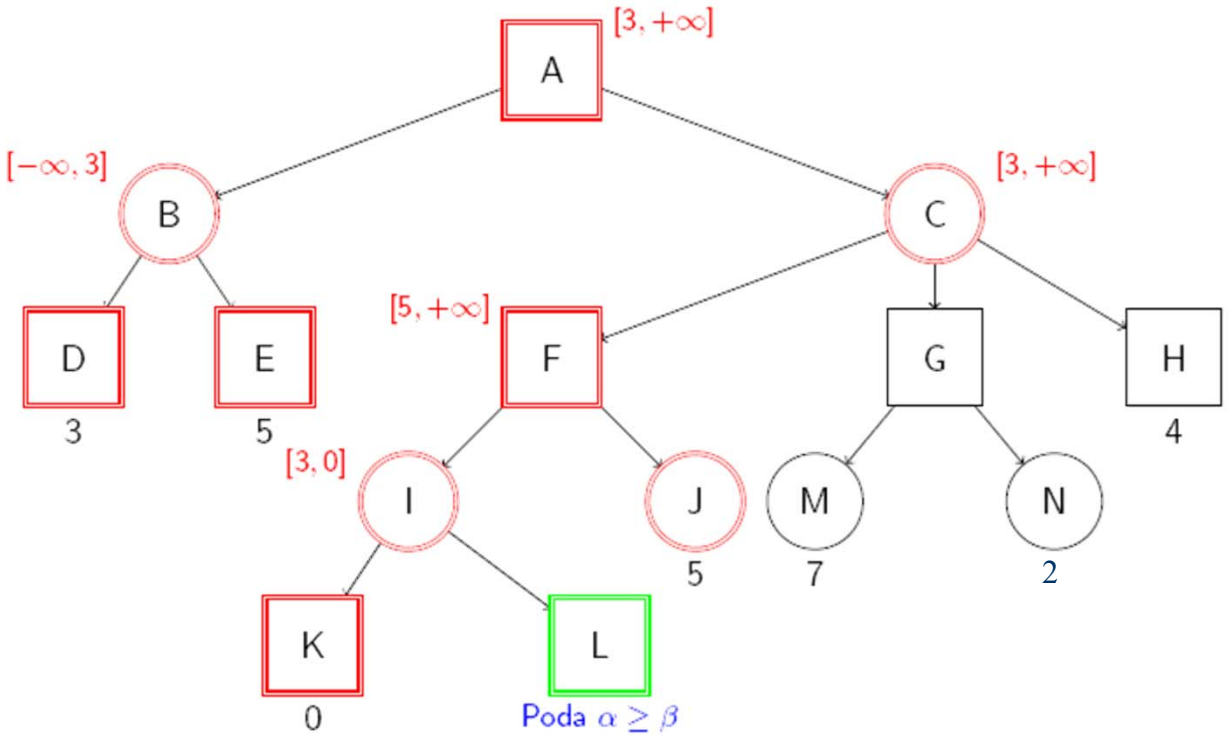
8. Ejemplo 4: Poda α-β



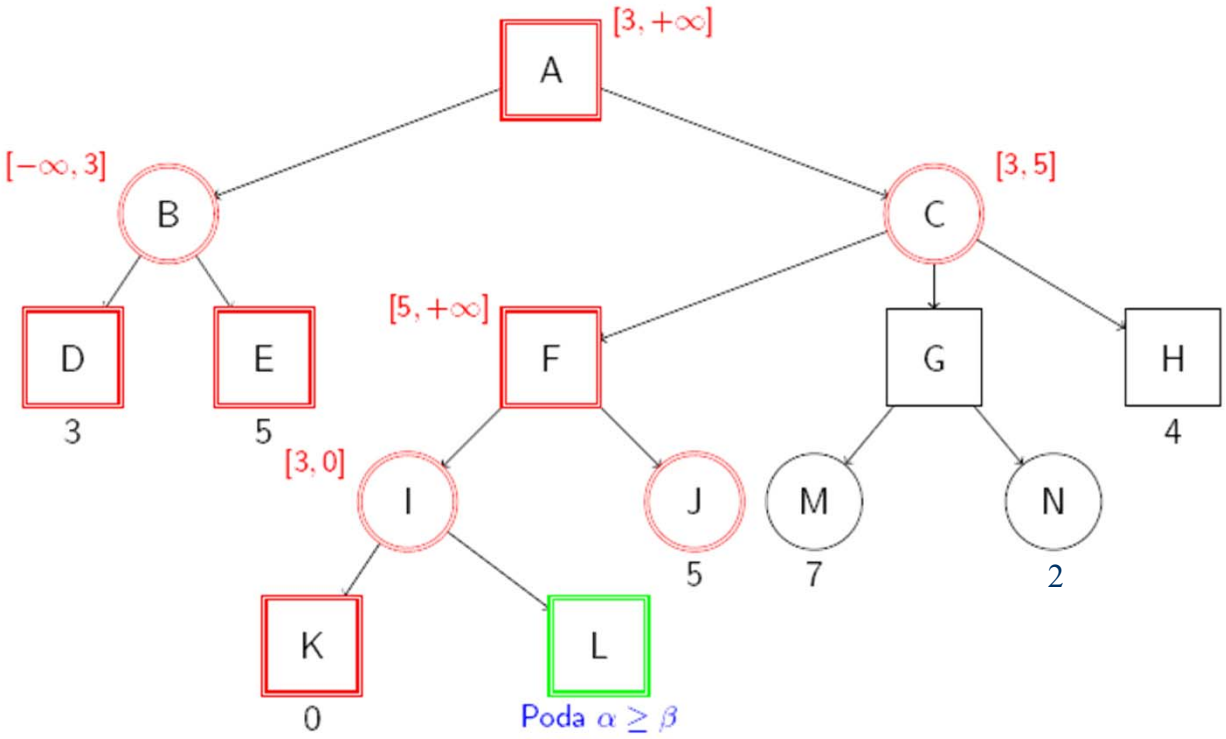
8. Ejemplo 4: Poda α-β



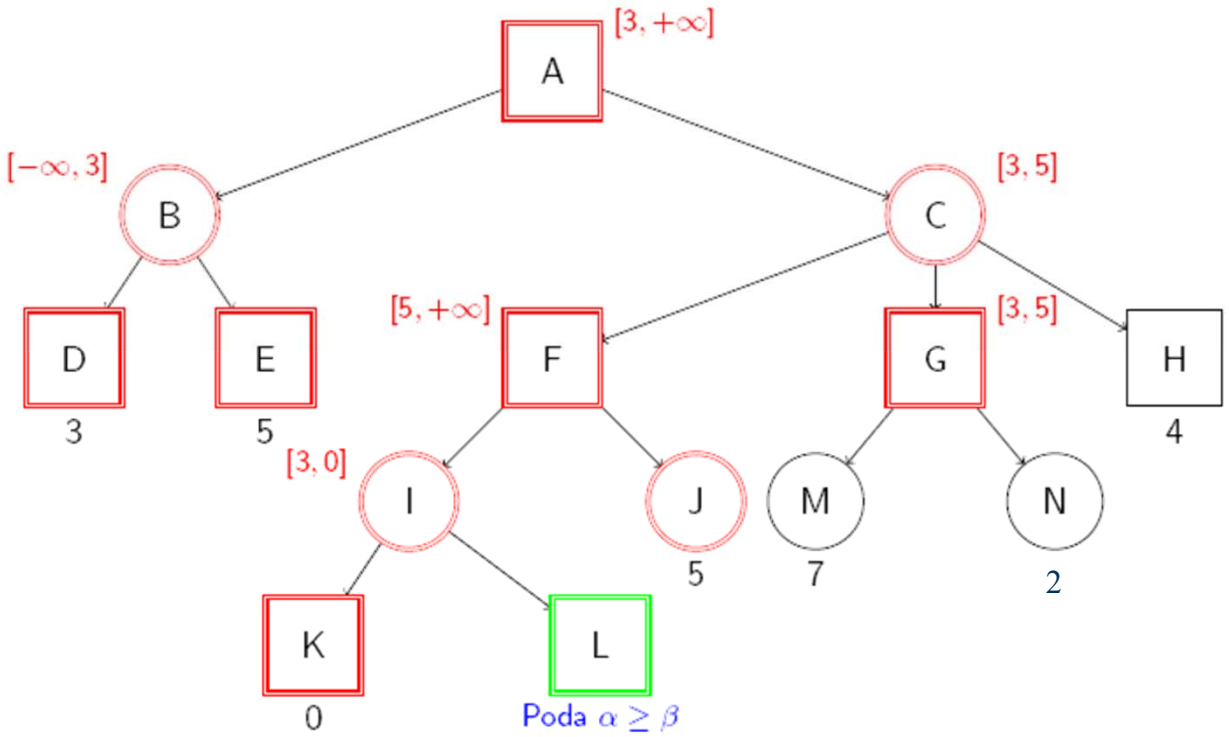
8. Ejemplo 4: Poda α-β



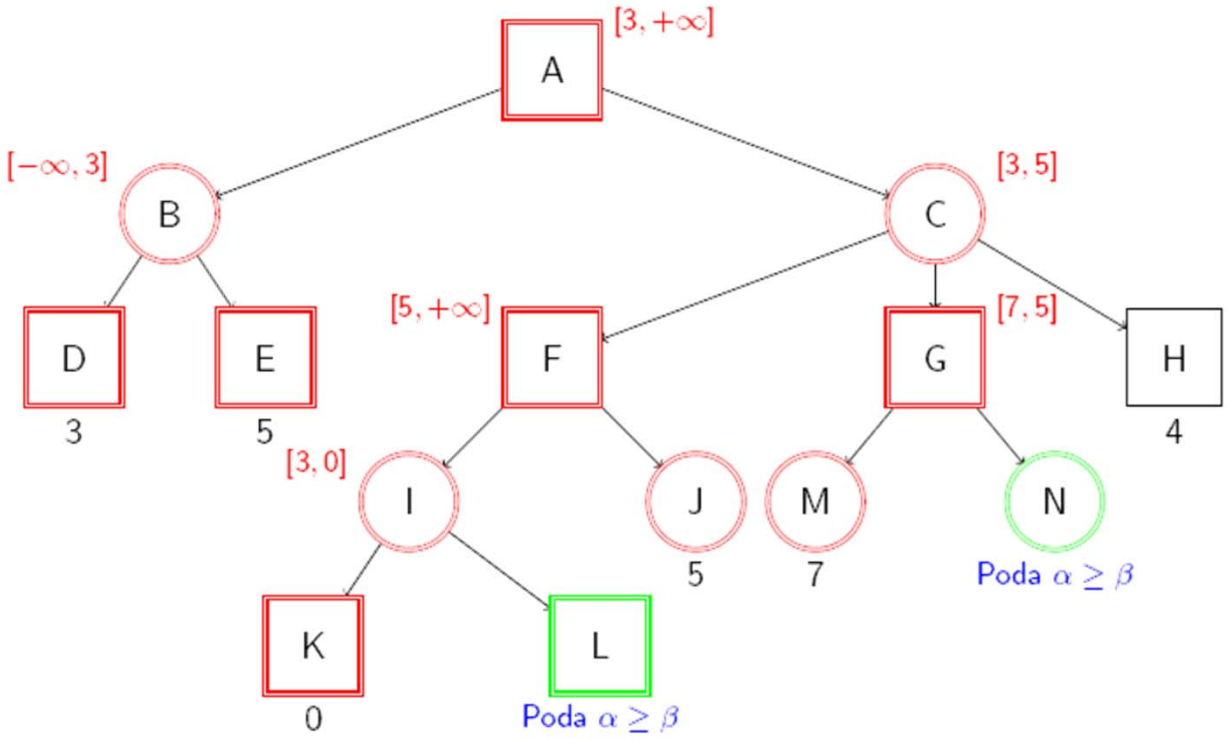
8. Ejemplo 4: Poda α-β



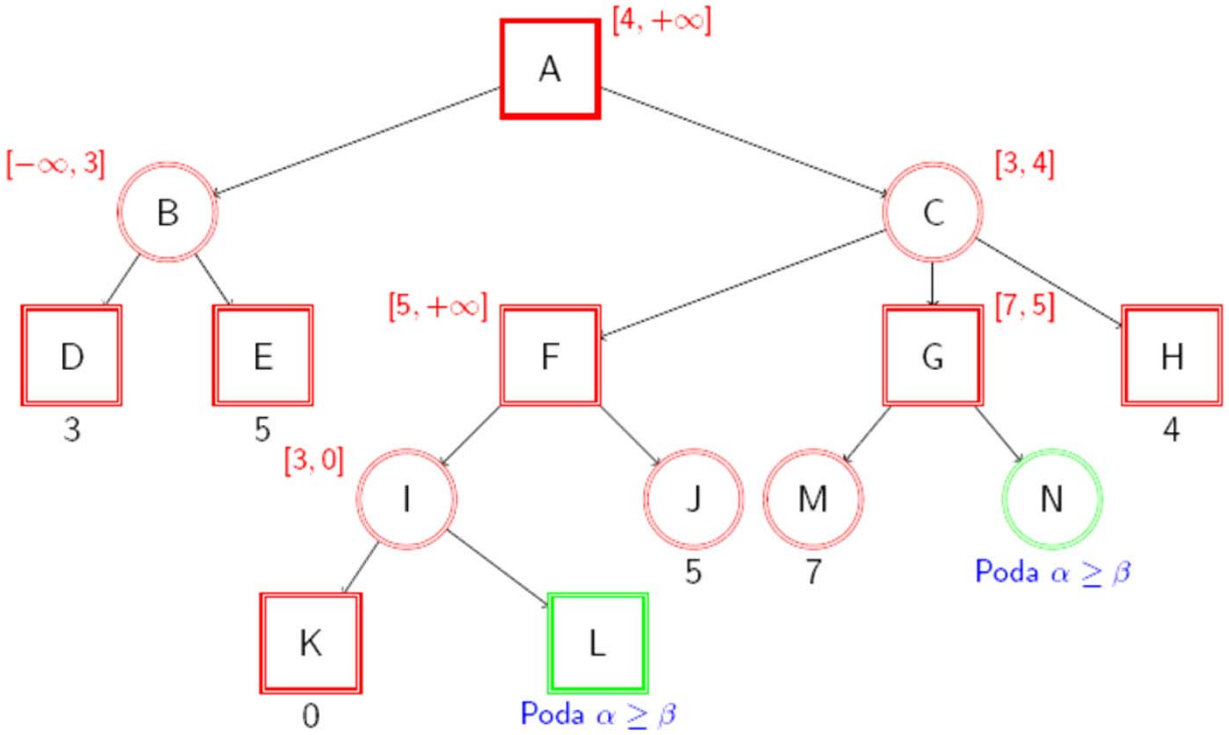
8. Ejemplo 4: Poda α-β



8. Ejemplo 4: Poda α-β



8. Ejemplo 4: Poda α-β



7.2 Función Poda α-β

```
tNodo: función poda_ab(E/S tNodo: nodo, E entero: jugador)
(…)
    alfa ← -infinito   beta← +infinito      prof ← 1
    desde jugada ← 1 hasta N hacer
        si esValida(nodo, jugada) entonces
            intento ← aplicaJugada(nodo,jugador,jugada)
            v ← valorMin_ab(intento, opuesto(jugador), prof+1, alfa, beta)
            si v > alfa  entonces
                alfa ← v
                mejorJugada ← jugada
            fin_si
        fin_si
    fin_desde
    si esValida(nodo, jugada) entonces
        nodo=aplicaJugada(nodo,jugador,mejorJugada)
    fin_si
    devolver nodo
fin_función
```

7.3 Función valorMin α-β

```
entero: función valorMin_ab(E tNodo: nodo, E entero: jugador, E entero: prof, E entero: alfa, E entero: beta)
(…)
    si terminal(nodo) entonces
            vmin← utilidad(nodo)
    si_no si prof=LIMITE entonces
            vmin← heuristica(nodo)
    si_no
            mientras jugada <N  Y alfa<beta hacer
                    si esValida(nodo, jugada) entonces
                        intento←aplicaJugada(nodo, jugador, jugada)
                        beta ← minimo(beta, valorMax_ab(intento, opuesto(jugador),
                                                        prof+1, alfa, beta)
                    fin_si
                    jugada← jugada+1
            fin_mientras
            vmin←beta
        fin_si
    fin_si
    devuelve vmin
fin_función
```

7.4 Función valorMin α-β

```
entero: función valorMax_ab(E tNodo: nodo, E entero: jugador, E entero: prof, E entero: alfa, E entero: beta)
(…)
    si terminal(nodo) entonces
            vmax← utilidad(nodo)
    si_no si prof=LIMITE entonces
            vmax← heuristica(nodo)
        si_no
            mientras  jugada <N  Y alfa<beta hacer
                    si esValida(nodo, jugada) entonces
                        intento←aplicaJugada(nodo, jugador, jugada))
                        alfa ← maximo(alfa, valorMin_ab(intento, opuesto(jugador),
                                                        prof+1, alfa, beta)
                    fin_si
                    jugada← jugada+1
            fin_mientras
            vmax← alfa
        fin_si
   fin_si
   devuelve  vmax
fin_función
```

9. Análisis de la Poda

- En el peor de los casos funciona como MiniMax.
- La expansión del árbol depende del orden de generación de los nodos.
- Problemas con la profundidad fija del árbol explorado:
  - **Efecto Horizonte:** se evalúa como buena o mala una situación sin saber que a la siguiente jugada la situación revierte.
  - **Falta de Equilibrio:** valores de evaluación muy inestables y dependientes del límite de profundidad impuesto.
- Solución: **Búsqueda Secundaria o de profundidad variable.**

9.1 Variantes

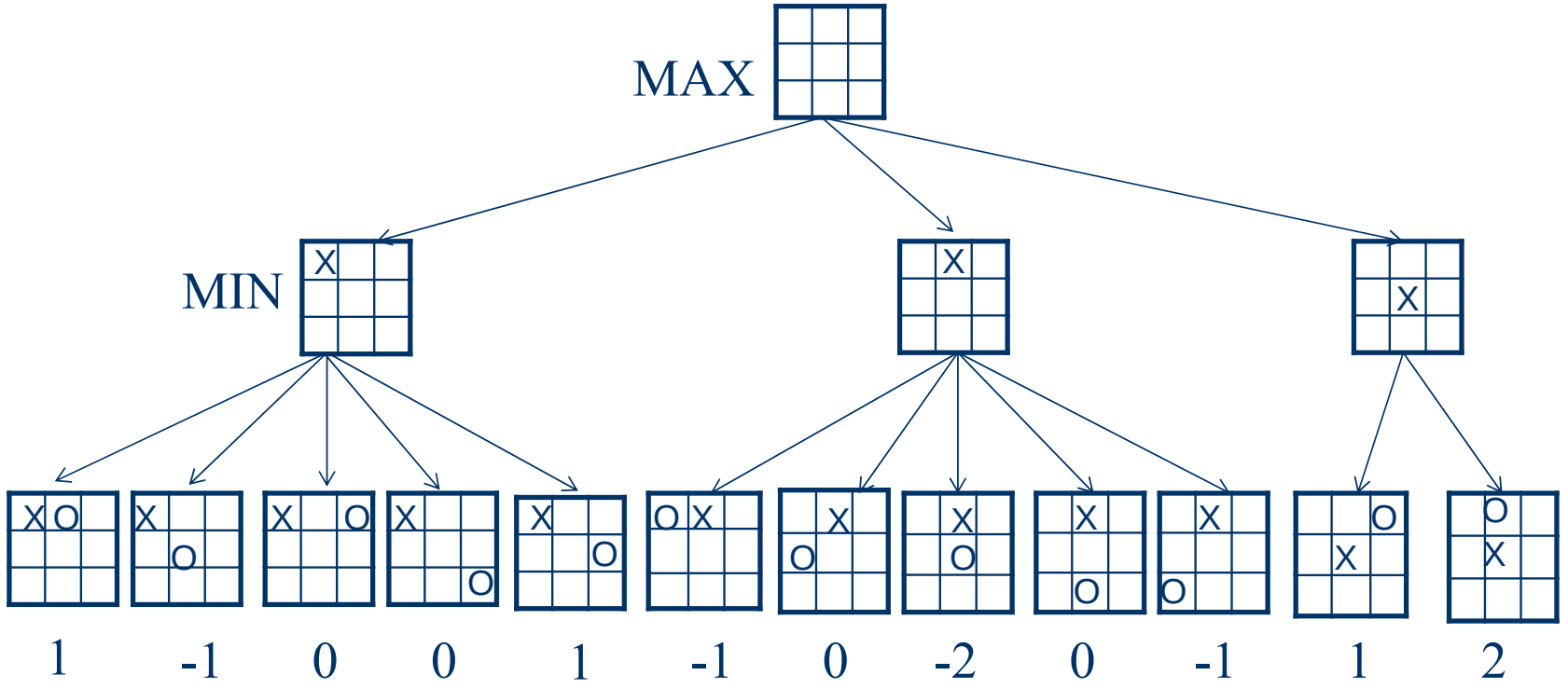
- **Poda α-β con Ventanas**: usar cotas (a,b) para los valores α-β de menor valor que ($-\infty$, $+\infty$).

- **Poda α-β de Profundidad Limitada**

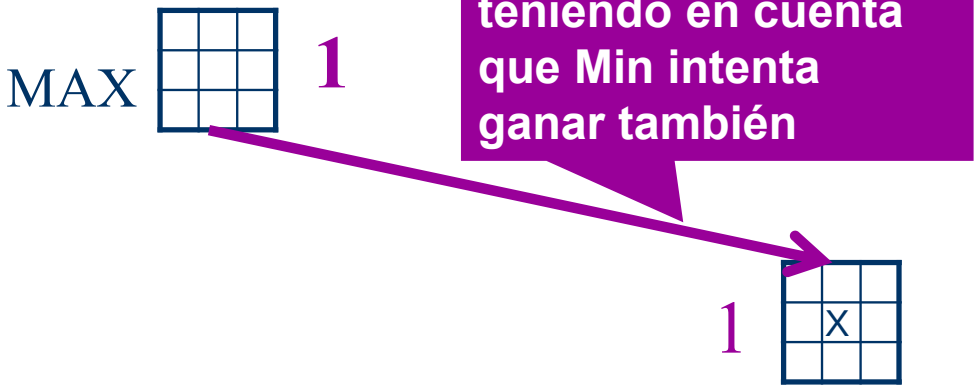
- **Métodos de Ordenación Fija y Dinámica**

10. Ejercicio

- Realizar la poda alfa-beta de izquierda a derecha.



Ejercicio

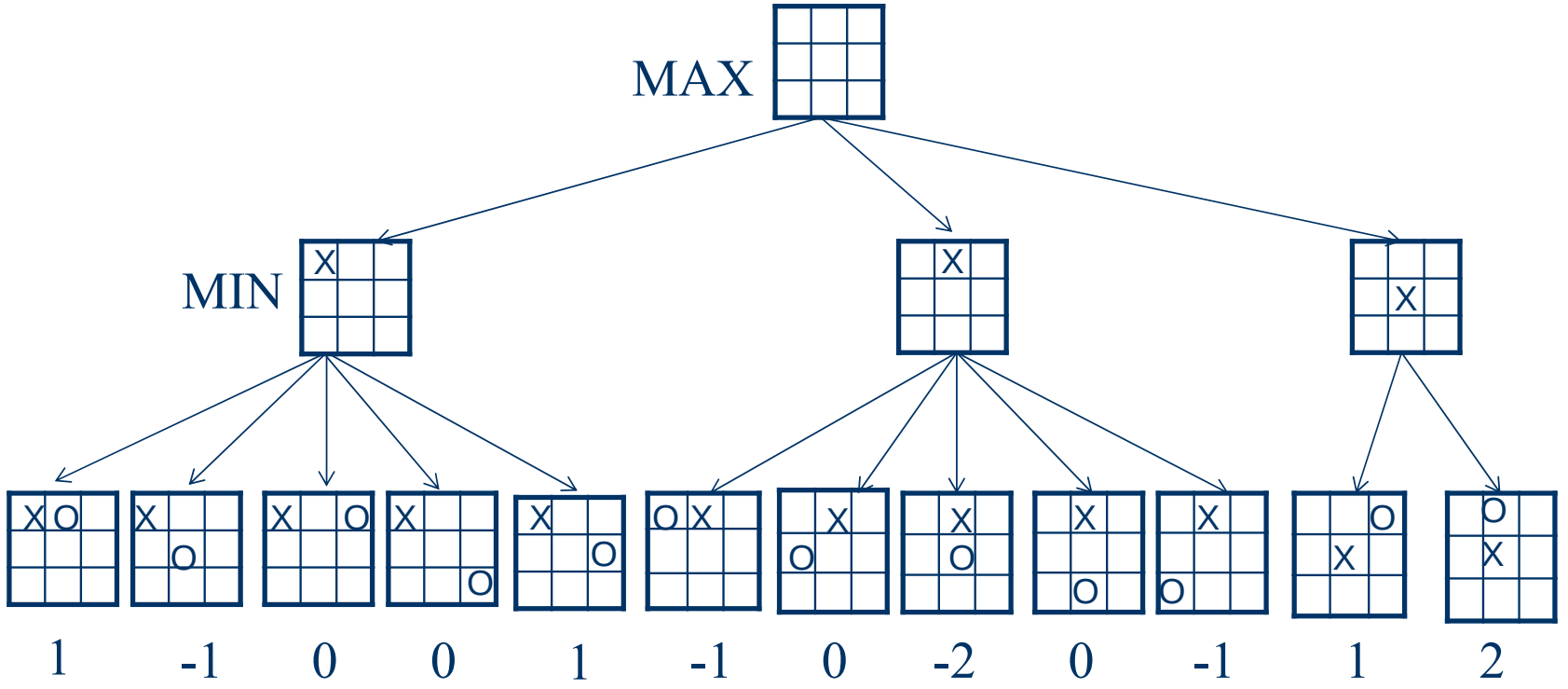

MAX
1
Mejor jugada posible, teniendo en cuenta que Min intenta ganar también
1
X

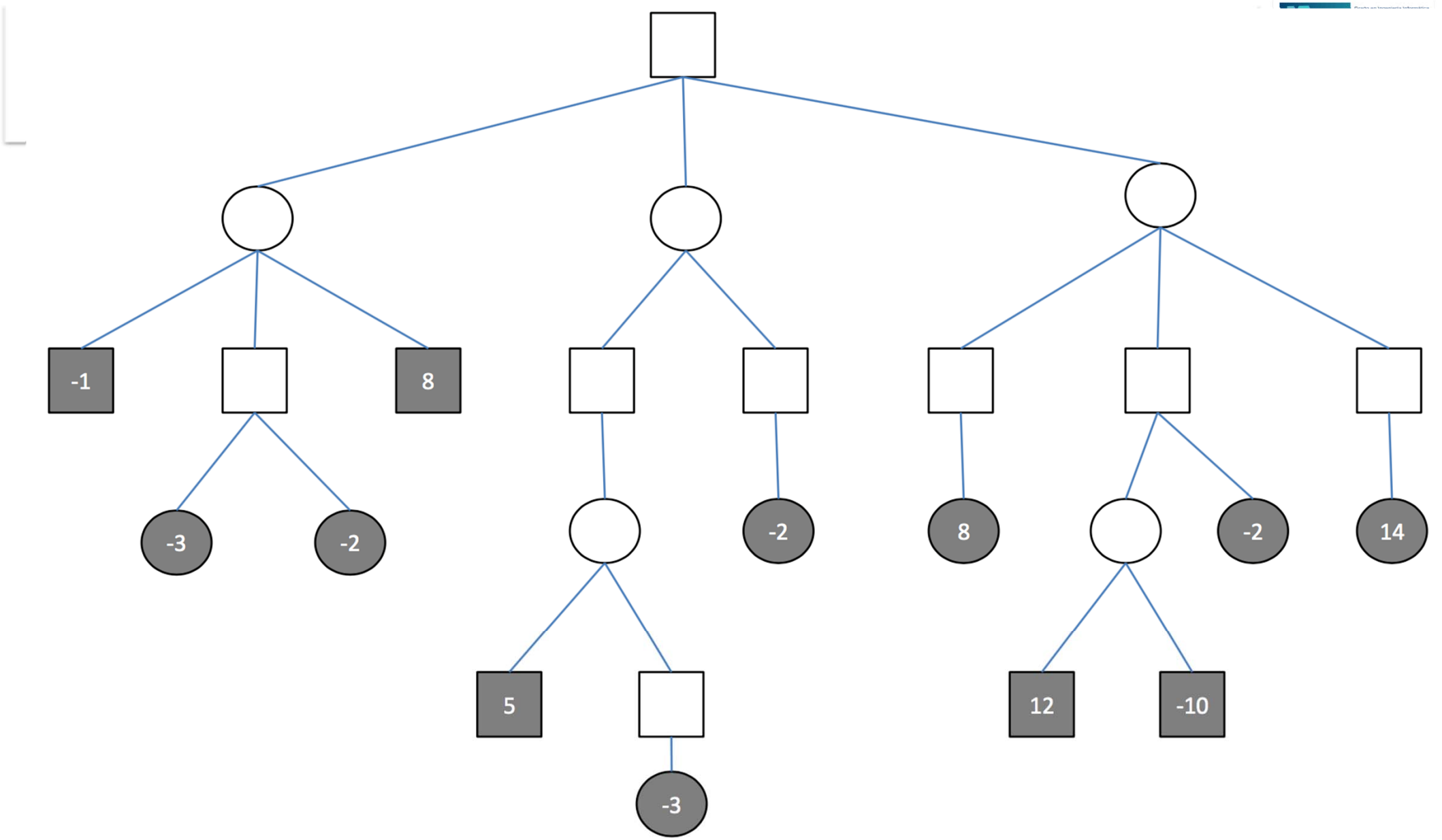
Orden de operadores

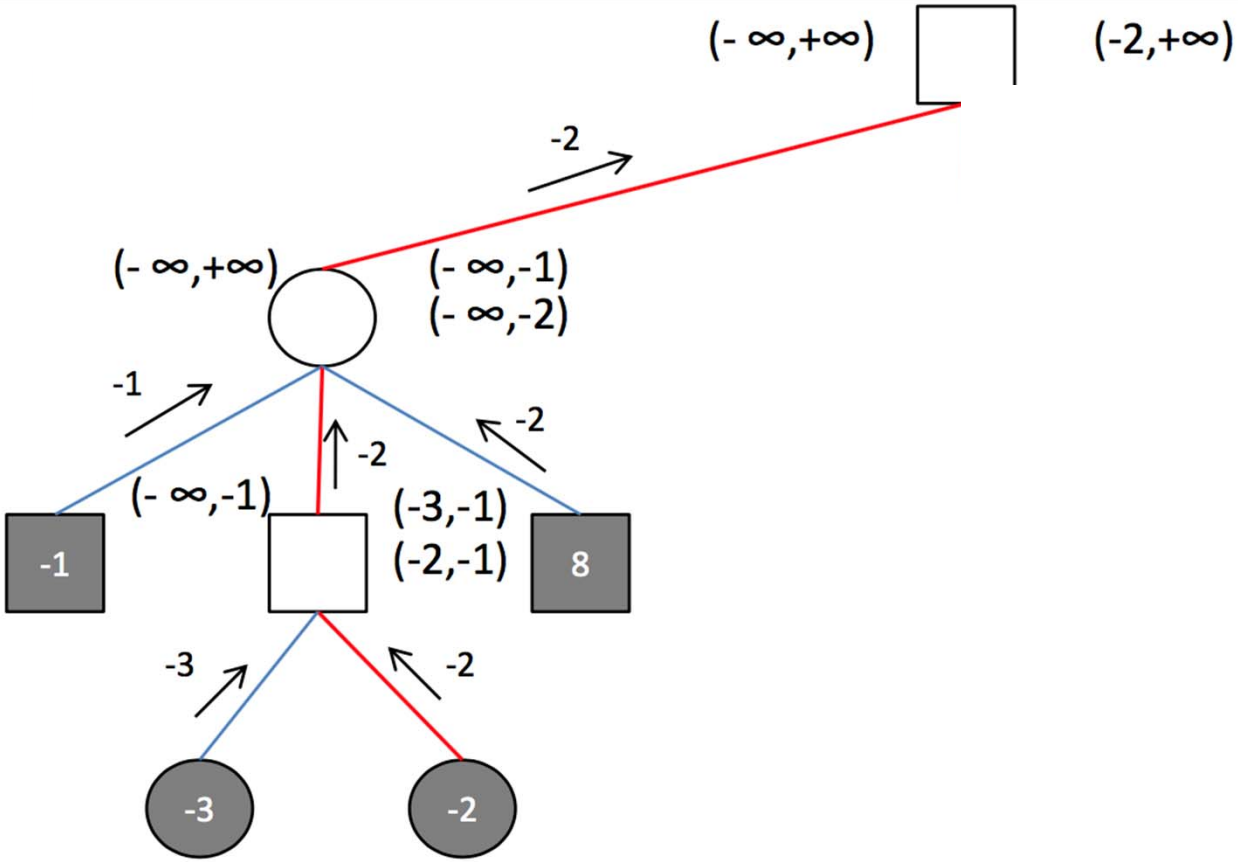
- Pero el orden de aplicación de los Operadores Sí puede modificar la Poda

Ejercicio

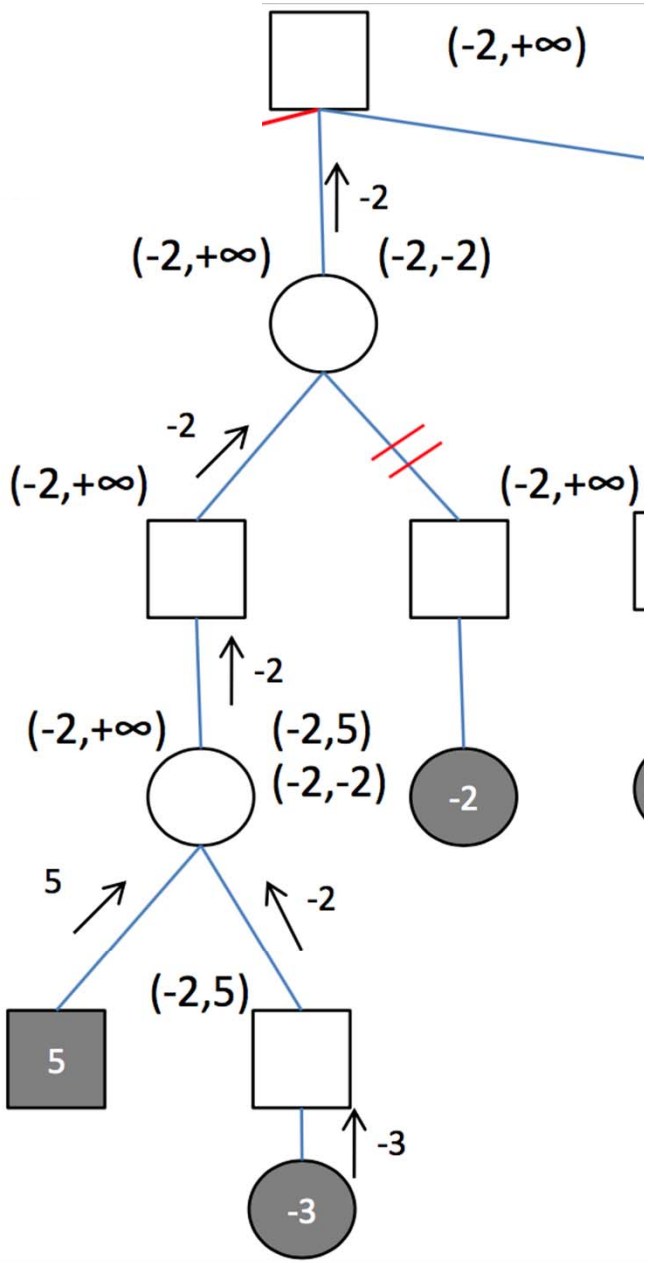
- Realizar la poda alfa-beta de derecha a izquierda.

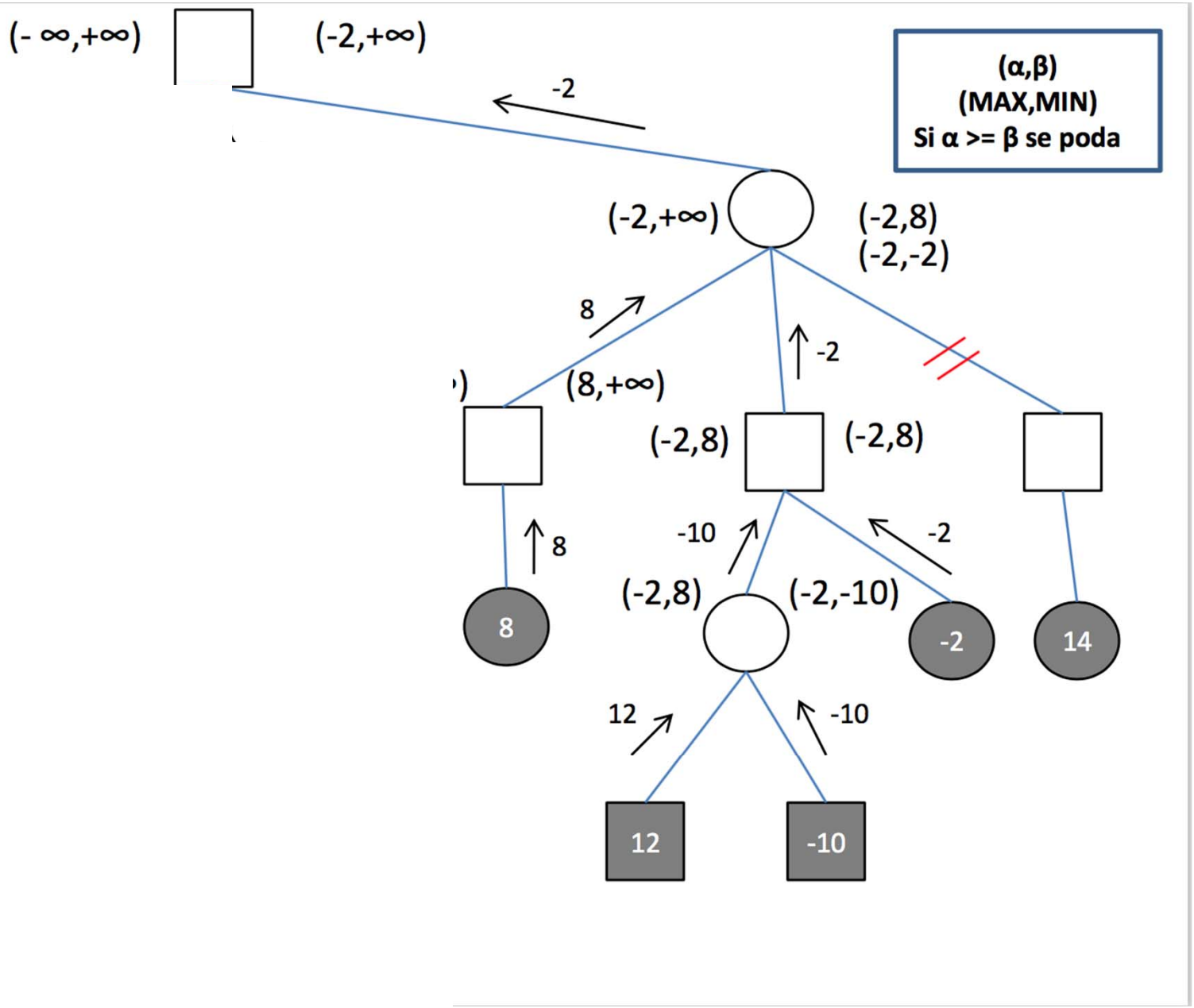


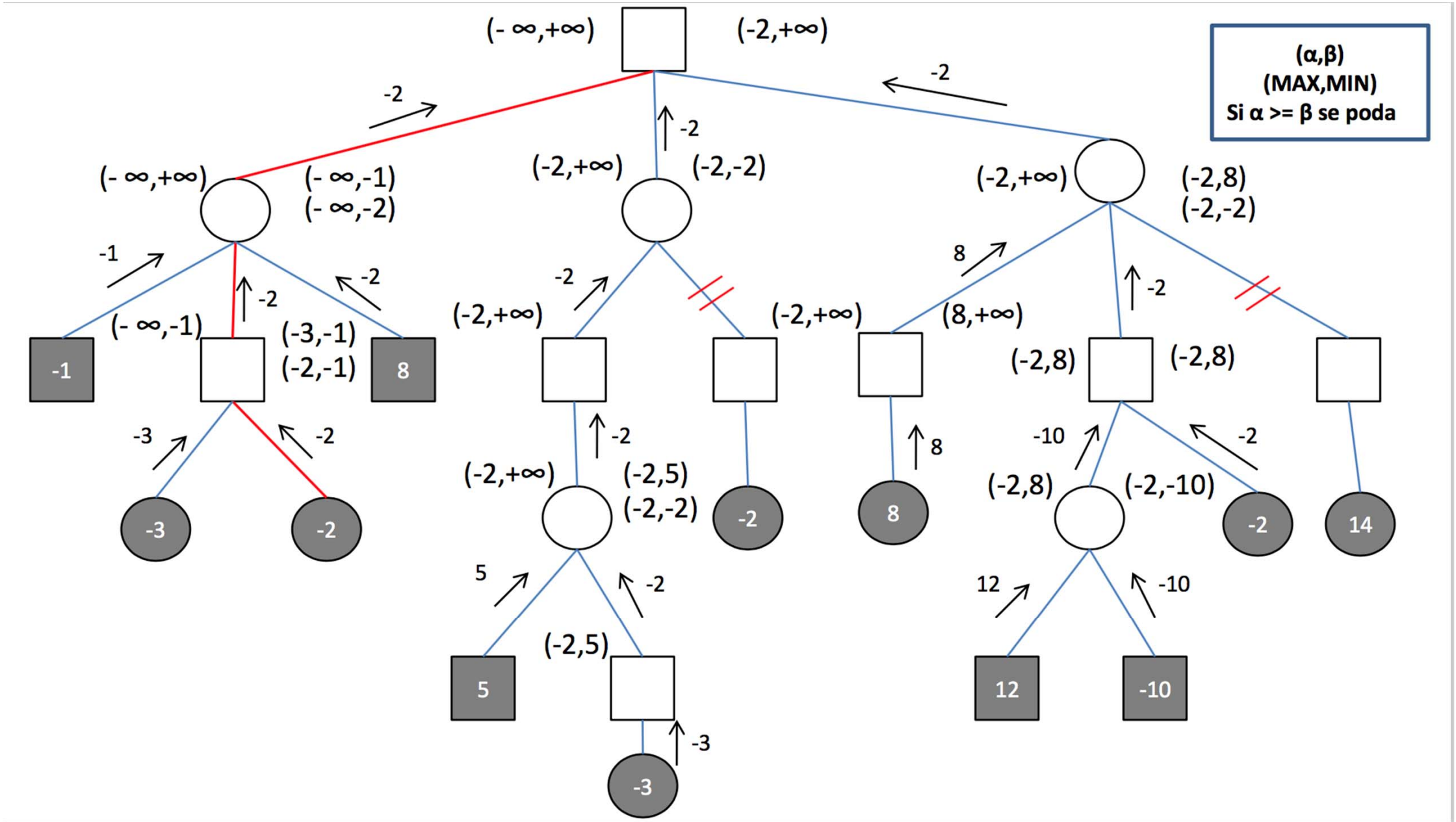

-1
8
-3
-2
-2
5
-3
8
12
-10
-2
14



(α,β)
(MAX,MIN)
Si α >= β se poda


(-2,+∞)
-2
(-2,+∞)
(-2,-2)
-2
(-2,+∞)
(-2,+∞)
-2
(-2,+∞)
(-2,5)
(-2,-2)
-2
5
-2
(-2,5)
5
-3
-3


(- ∞,+∞)
(-2,+∞)
-2
(α,β)
(MAX,MIN)
Si α >= β se poda
(-2,+∞)
(-2,8)
(-2,-2)
8
-2
(8,+∞)
(-2,8)
(-2,8)
8
-10
-2
(-2,8)
(-2,-10)
8
-2
14
12
-10
12
-10


(α,β)
(MAX,MIN)
Si α >= β se poda
(- ∞,+∞)
(-2,+∞)
-2
(- ∞,+∞)
(- ∞,-1)
(- ∞,-2)
(-2,+∞)
(-2,-2)
(-2,+∞)
(-2,8)
(-2,-2)
-1
-2
-2
-2
8
-2
(- ∞,-1)
(-3,-1)
(-2,-1)
(-2,+∞)
(-2,+∞)
(8,+∞)
-2
-1
8
-3
-2
-2
(-2,8)
(-2,8)
-2
-3
-2
-2
(-2,+∞)
(-2,5)
(-2,-2)
8
-10
-2
(-2,8)
(-2,-10)
-2
14
5
-2
12
-10
(-2,5)
5
12
-10
-3
-3

Referencias

- Borrajo D. Et al. (1997), Inteligencia artificial : Métodos y técnicas. Editorial Centro de Estudios Ramón Areces. Madrid
- RUSSELL, S. y NORVIG, P.(2004) Inteligencia Artificial. Un enfoque moderno (2ª Ed.). Pearson Educación, S.A. Madrid.
- Fernández Galán y otros autores (203): Problemas Resueltos de Inteligencia Artificial Aplicada. Pearson.
- NILSSON N (2001): Inteligencia Artificial: Una nueva síntesis. McGrawHill.
- RICH, E. and KNIGHT, K., (1994) Inteligencia artificial. McGraw-Hill. Edición original: Artificial Intelligence.
- WINSTON, P. H. (1994), Inteligencia Artificial. Tercera Edición, p. xxv+805, Addison-Wesley Iberoamericana, Wilmington, Delaware, EE.UU
- Además de los libros recomendados para todos los temas de búsqueda .... Se pueden visitar webs dedicadas a la investigación en juegos y jugar on-line en
- http://www.mundopc.net/ocio/demesa/conecta4/index.php
- http://www.cs.ualberta.ca/~games/
- http://www.alumni.caltech.edu/~leif/games/Morris/morris9b.html
- http://www.delphiforfun.org/Programs/NIM_Minimax.htm