

Tema 5. Búsqueda entre 2 Adversarios

Análisis de Juegos

Elisa Guerrero Vázquez

Esther L. Silva Ramírez

Análisis de Juegos

1. Juegos de 2 adversarios: generalidades
2. Algoritmo Minimax
3. Funciones Heurísticas
4. Poda alfa-beta
5. Análisis y Variantes

Objetivos

- Saber reconocer el tipo de problemas a los que se les puede aplicar estas estrategias
- Saber formular 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

1. Características de los Juegos

- Juegos para 2 jugadores (bipersonales)
- Los jugadores mueven alternativamente
- 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

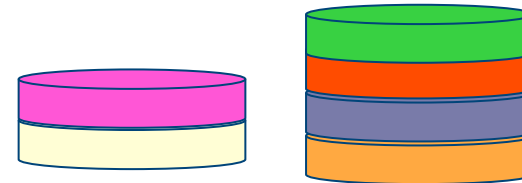
X	O	O
O	X	
		X

1.1 Formulación

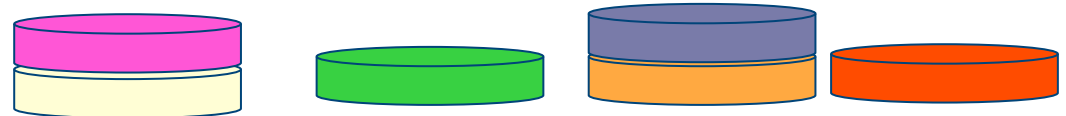
- Un Problema de Búsqueda entre 2 Adversarios se define formalmente como:
 - **Estado Inicial**, describiendo la situación inicial del juego
 - **Lista de Operadores o Jugadas**
 - **Funciones aplicaJugada y esValida**, para aplicar una Jugada cuando el operador es válido
 - **Test Terminal** que determina cuándo acaba el juego porque gana un jugador o se produce un empate
 - **Función Utilidad**, da una puntuación a los estados terminales

1.2 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 la continuación

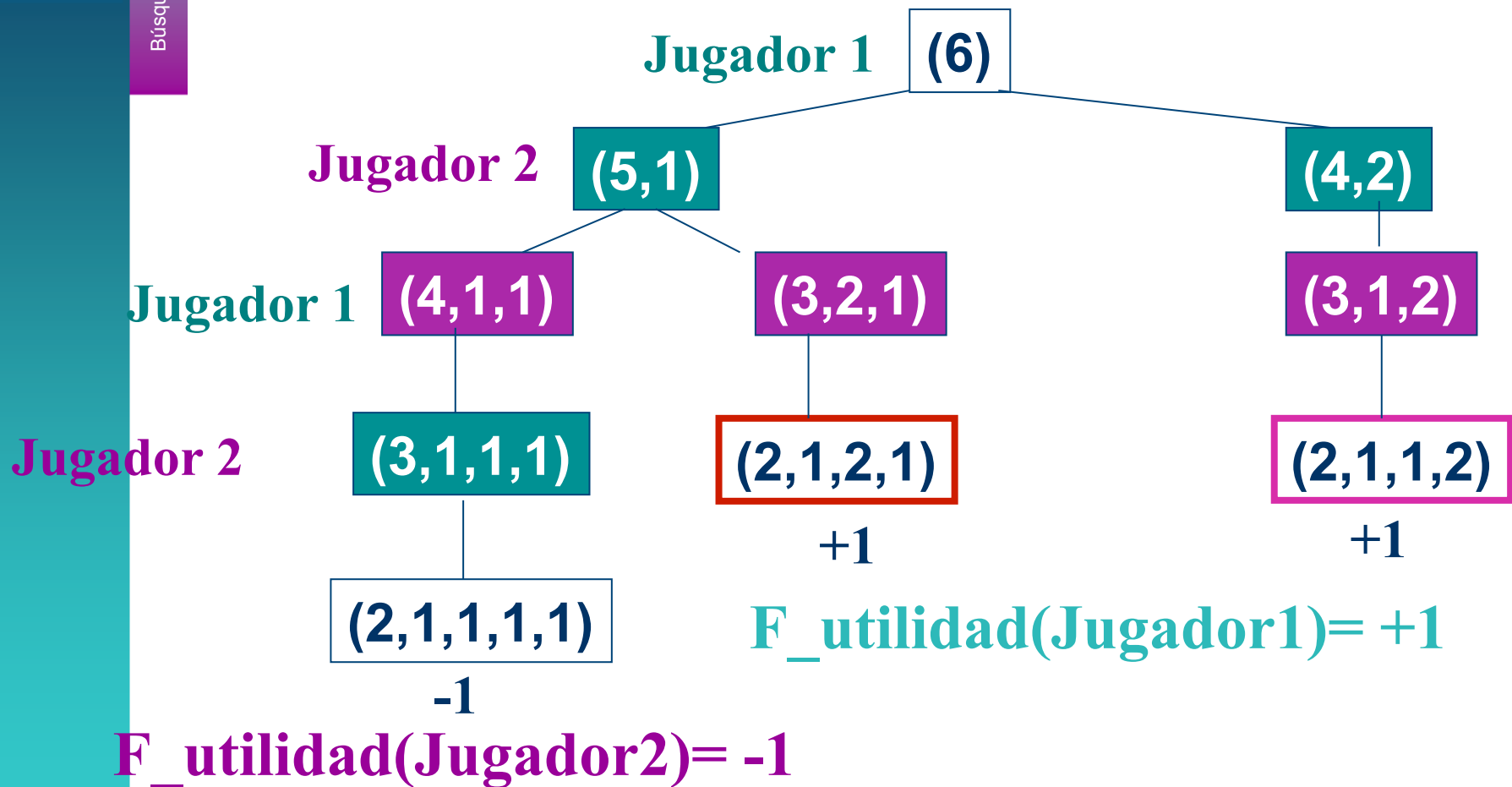


- El primer jugador que no pueda realizar un movimiento válido, pierde

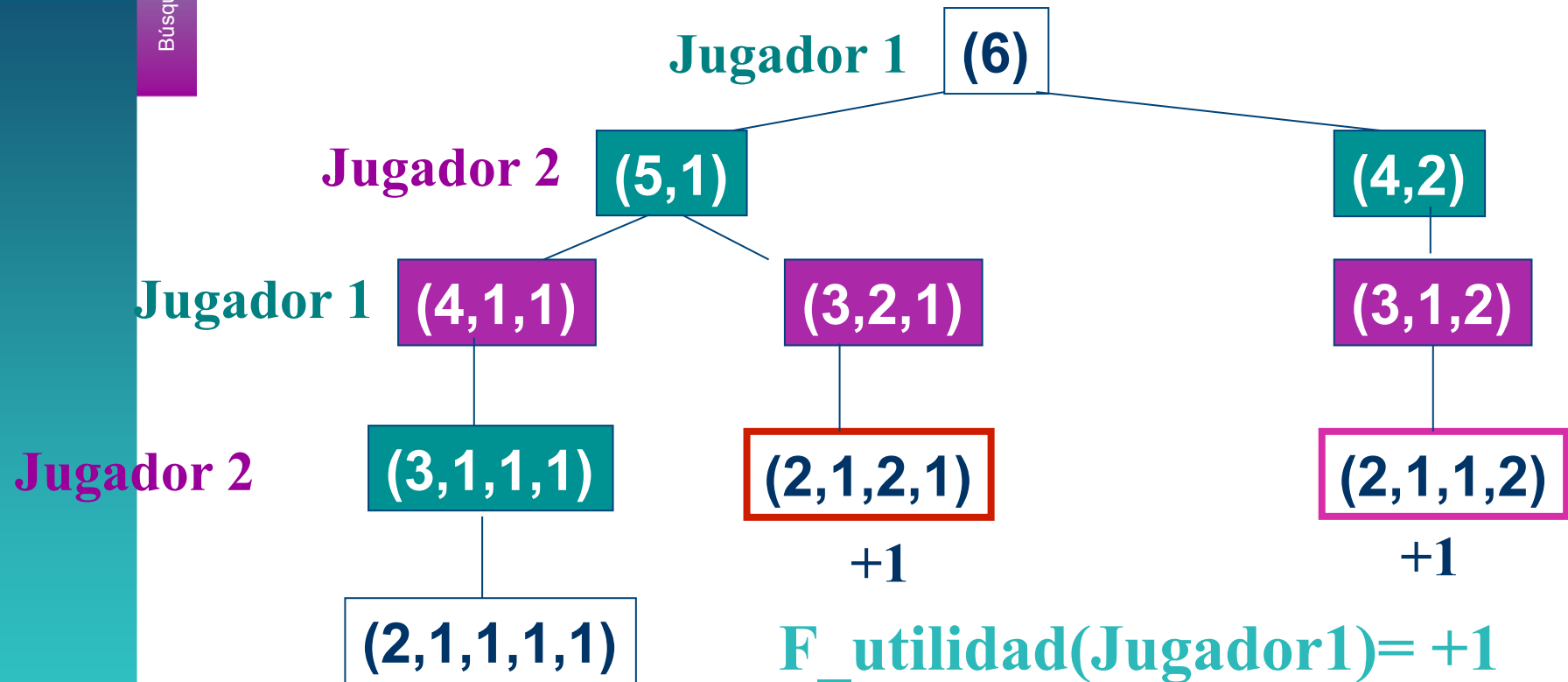
1.2 Juego del Grundy

- **Estado Inicial:** N fichas
- **Jugadas:** **Dividir** un lote de fichas en dos lotes de tamaño desigual.
- **Test Terminal:** Comprobar 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 +1, si el nodo terminal es del Jugador 2 devuelve -1

1.2 Juego del Grundy con 6 fichas





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



$F_{\text{utilidad}}(\text{Jugador2}) = -1$

2. 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 los movimientos de MIN son también los mejores posibles

2.1 MiniMax: Estrategia básica

- En cada turno de MAX:
 1. Construir el árbol de juego completo cuyo nodo raíz sea la situación actual (situación después de un movimiento de MIN o el estado inicial)
 2. Valorar los estados finales según la función de utilidad
 3. *Propagar* hasta la raíz los valores de la función
 4. Elegir el movimiento que lleve al sucesor con mejor valoración

2.2 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***

❖ **MIN** toma el valor del sucesor con ***menor valor***

2.3 Evaluación Minimax

■ Valor Minimax(n) =

- Utilidad(n) Si n es un estado terminal
- $\text{Max} (\text{ValorMinimax} (s))_{s \in \text{Sucesores}(n)}$ Si n es un estado Max
- $\text{Min} (\text{ValorMinimax}(s))_{s \in \text{Sucesores}(n)}$ Si n es un estado Min

2.4 Seudocódigo 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_desde
  nodo=aplicaJugada(nodo,jugador,mejorJugada);
devolver nodo
fin_función
```

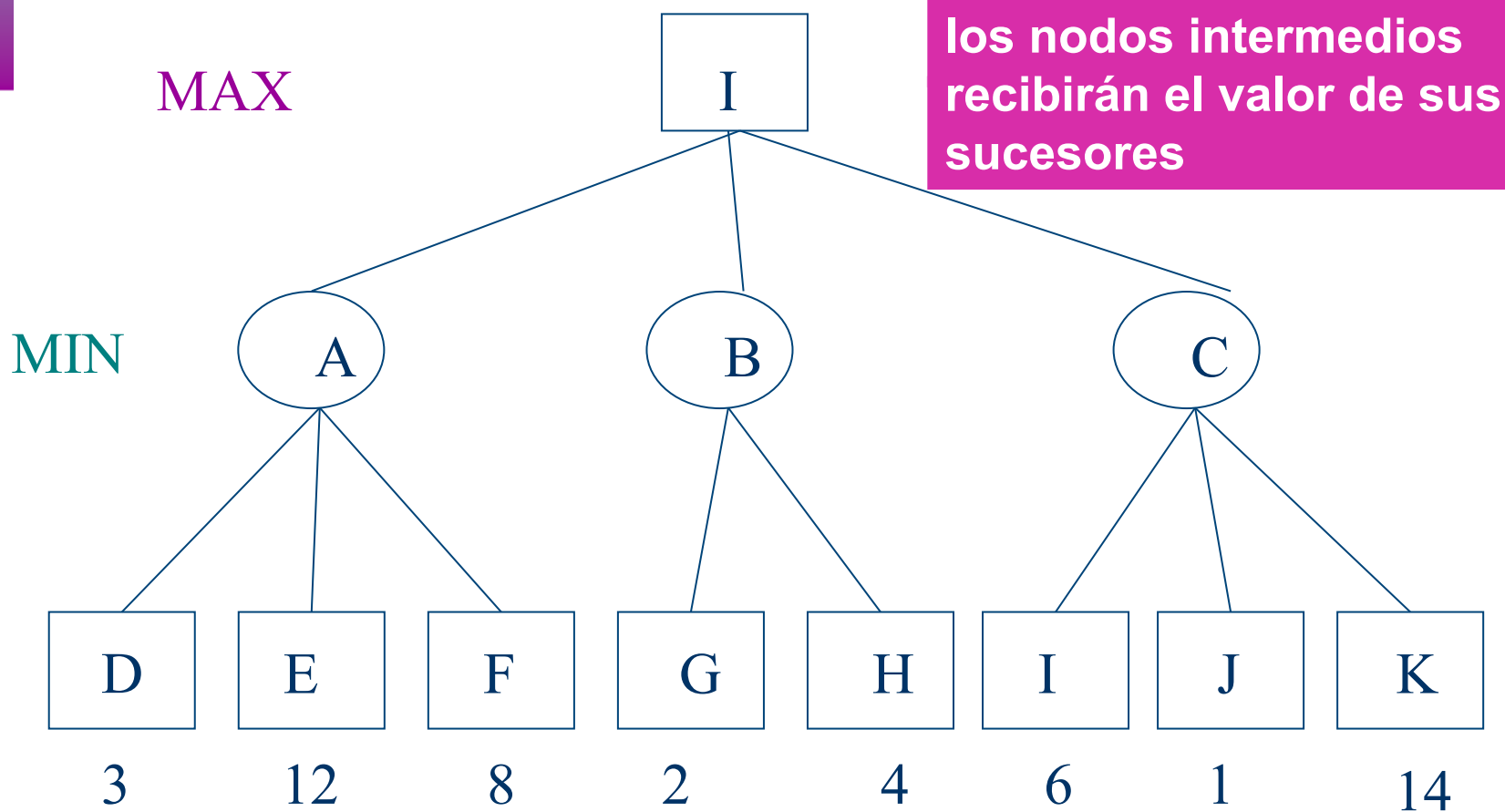
2.4 Seudocódigo función valorMin

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

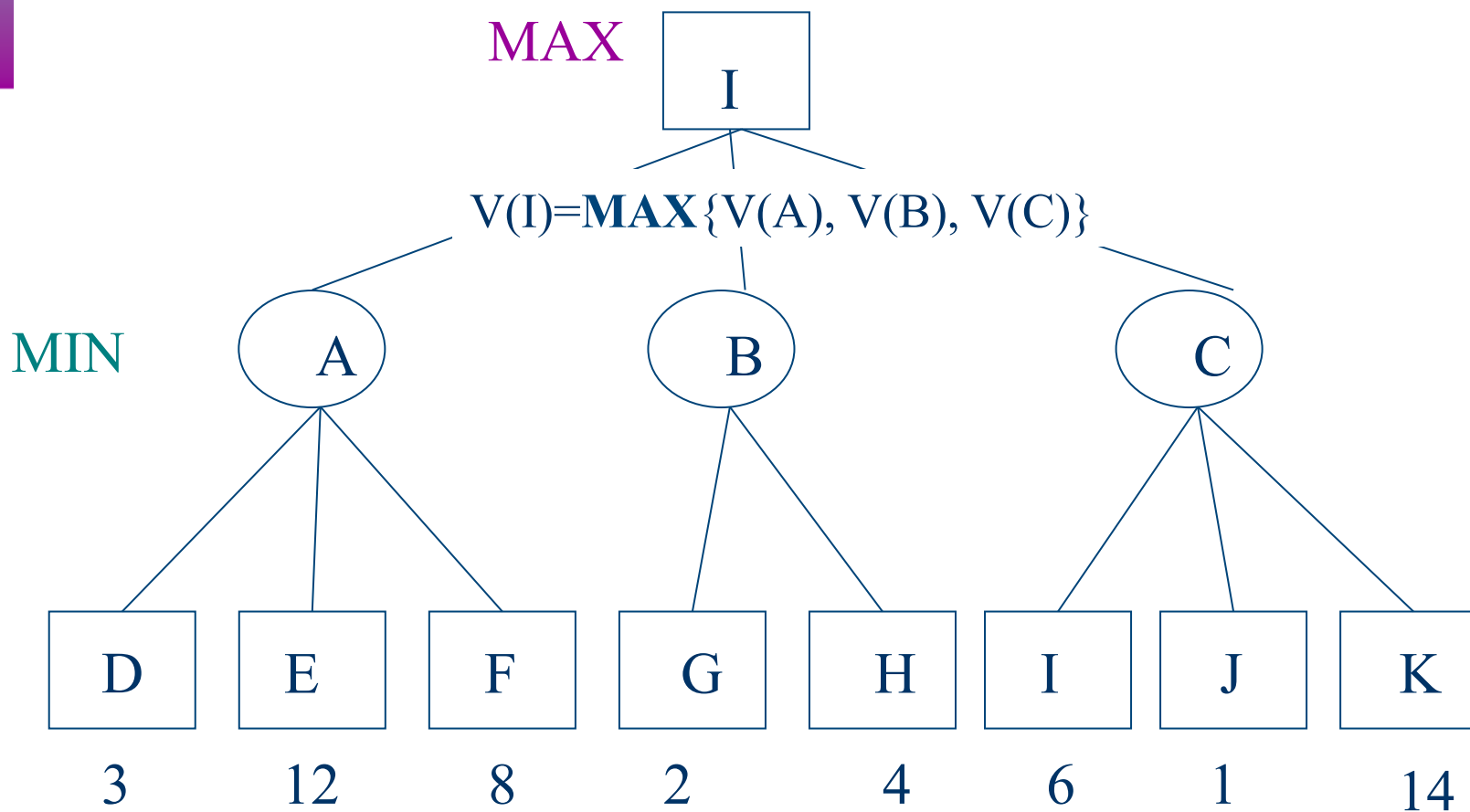
2.4 Seudocódigo función valorMax

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

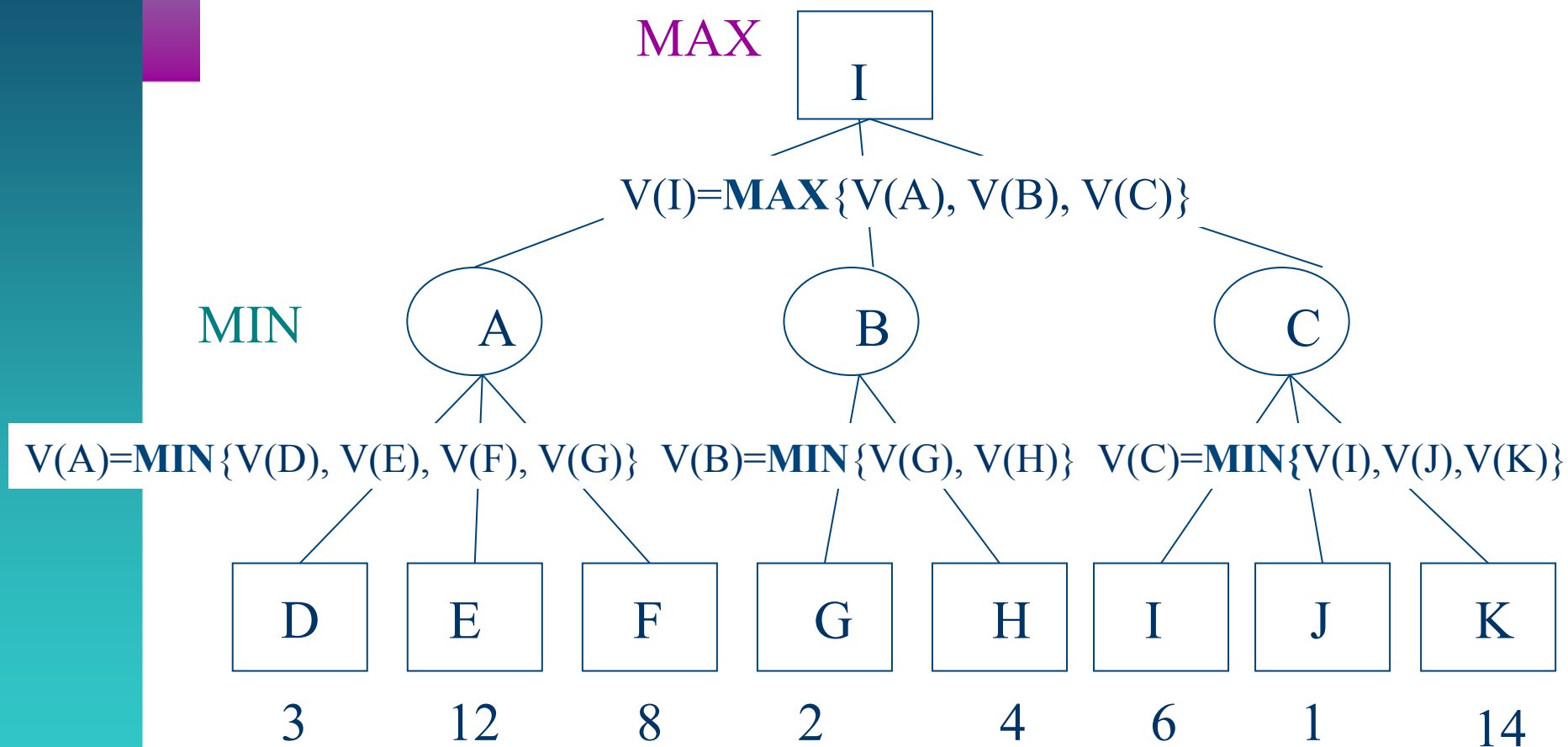

2.5 Ejemplo1: Aplicación Minimax (2 niveles)



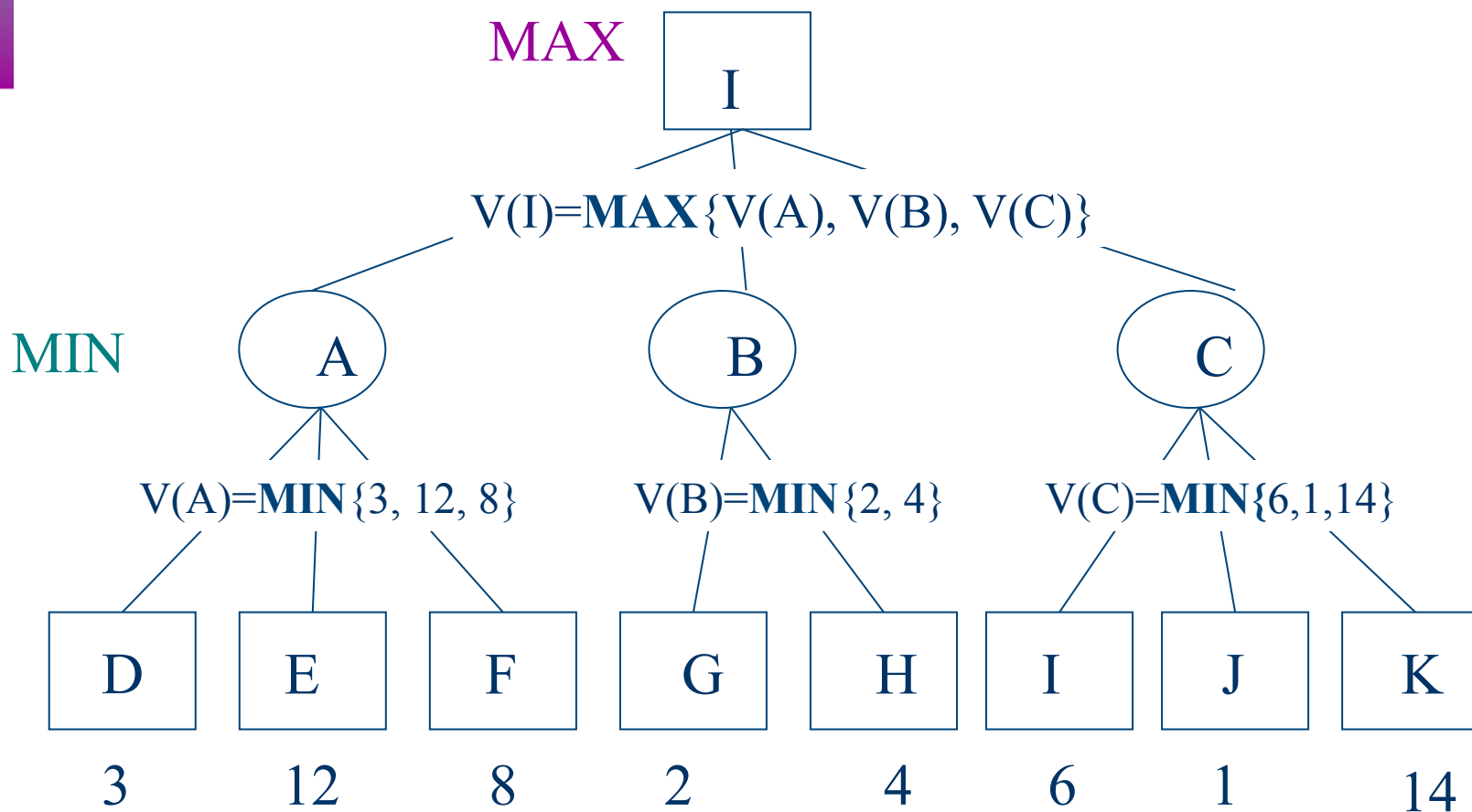
2.5 Ejemplo1: Aplicación Minimax (2 niveles)



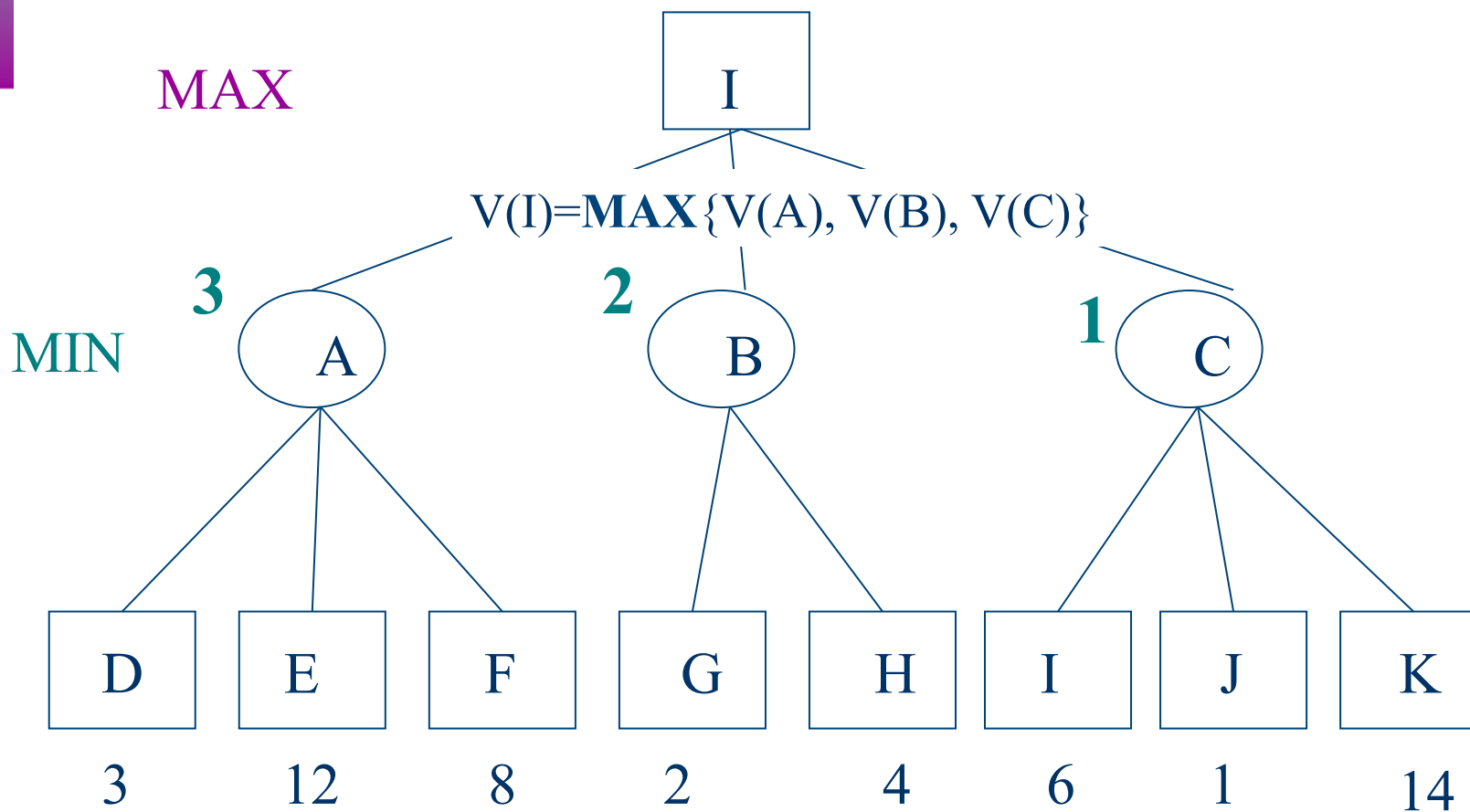
2.5 Ejemplo1: Aplicación Minimax (2 niveles)



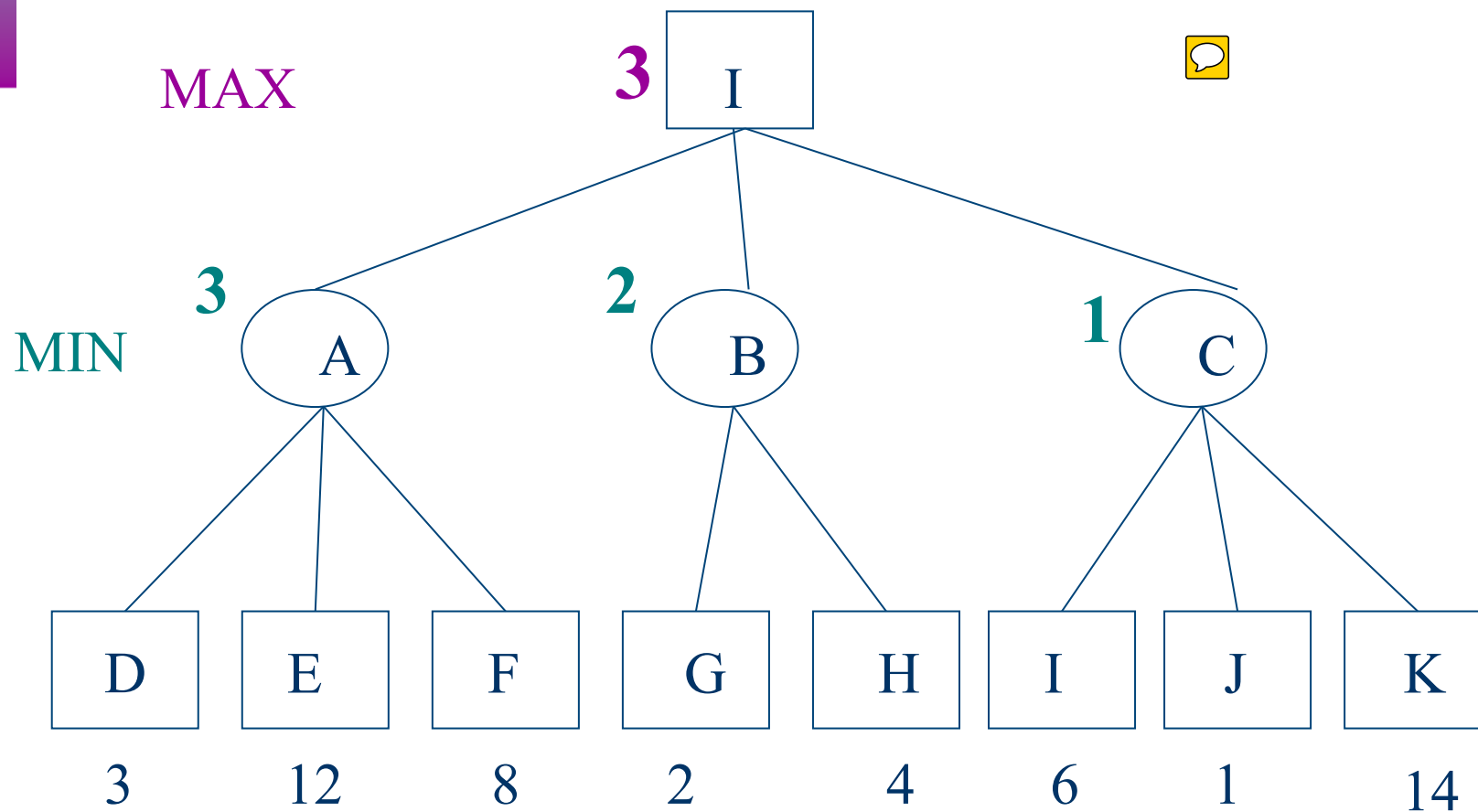
2.5 Ejemplo1: Aplicación Minimax (2 niveles)



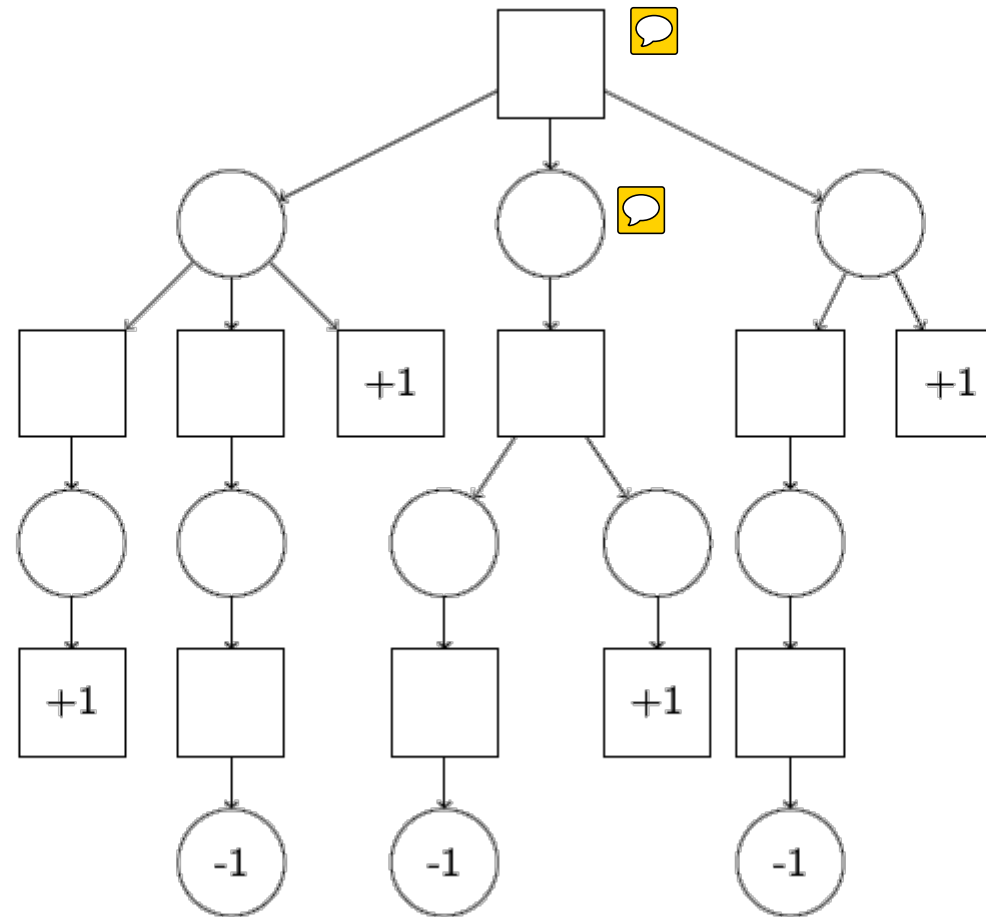
2.5 Ejemplo1: Aplicación Minimax (2 niveles)



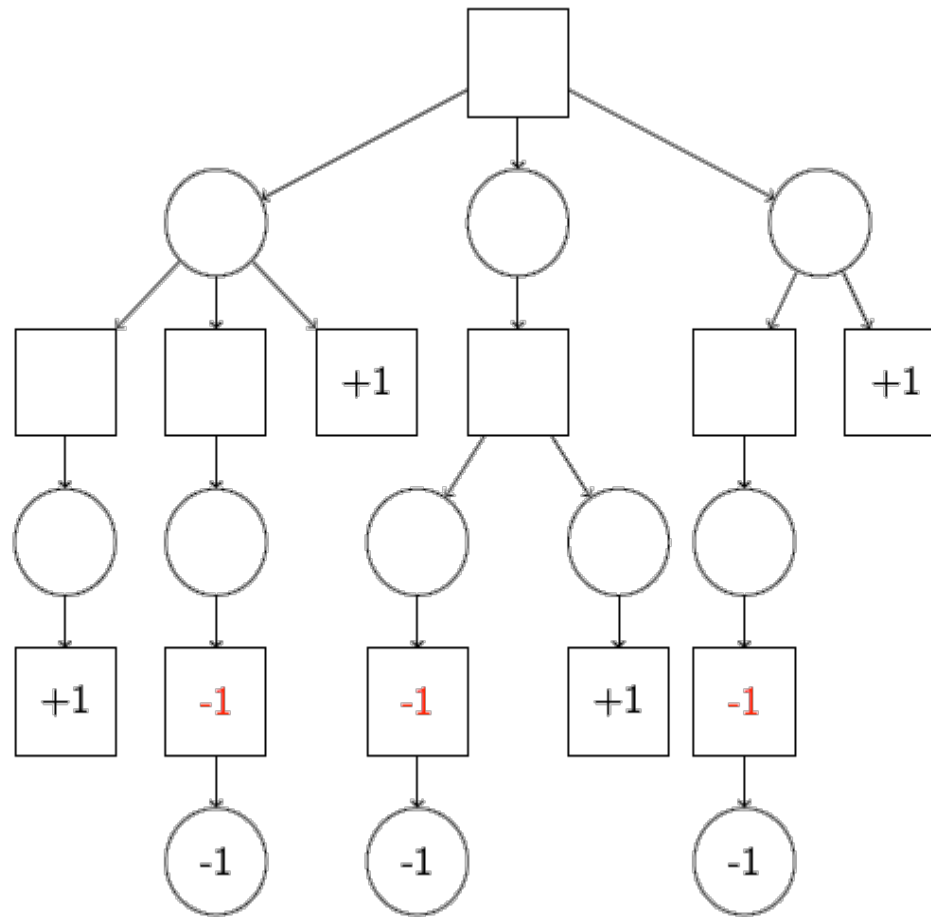
2.5 Ejemplo1: Aplicación Minimax (2 niveles)



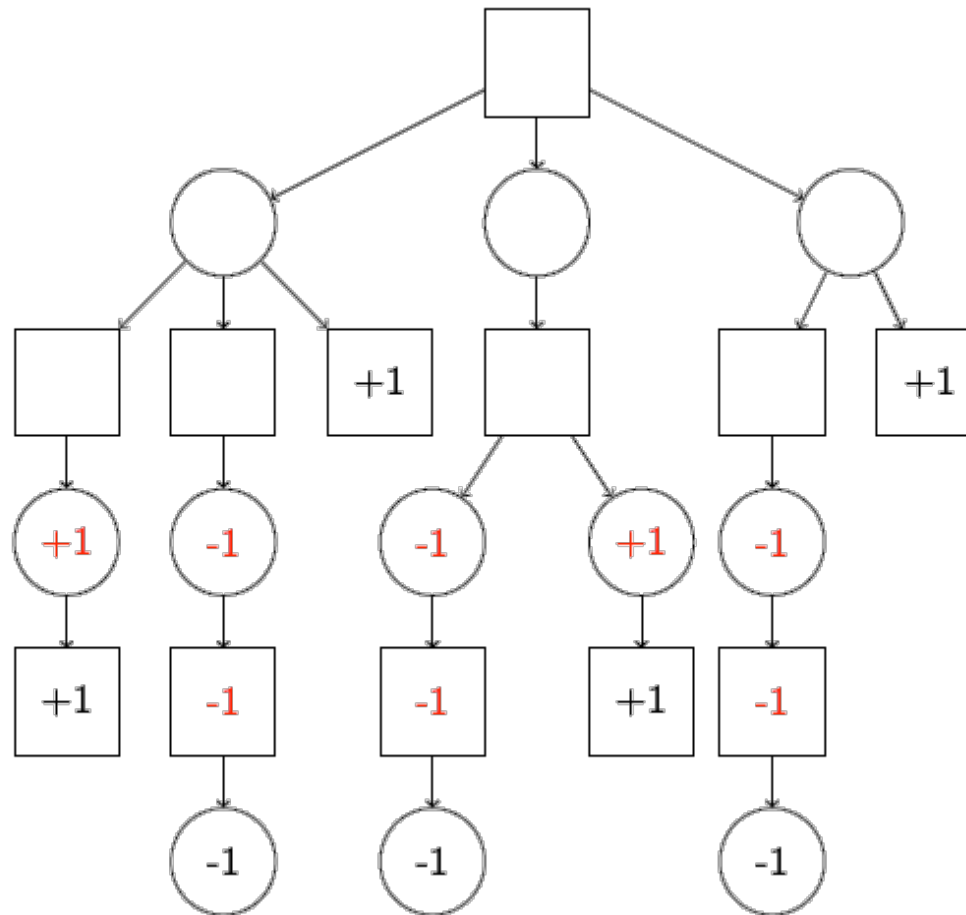
2.6 Ejemplo 2: Aplicación Minimax



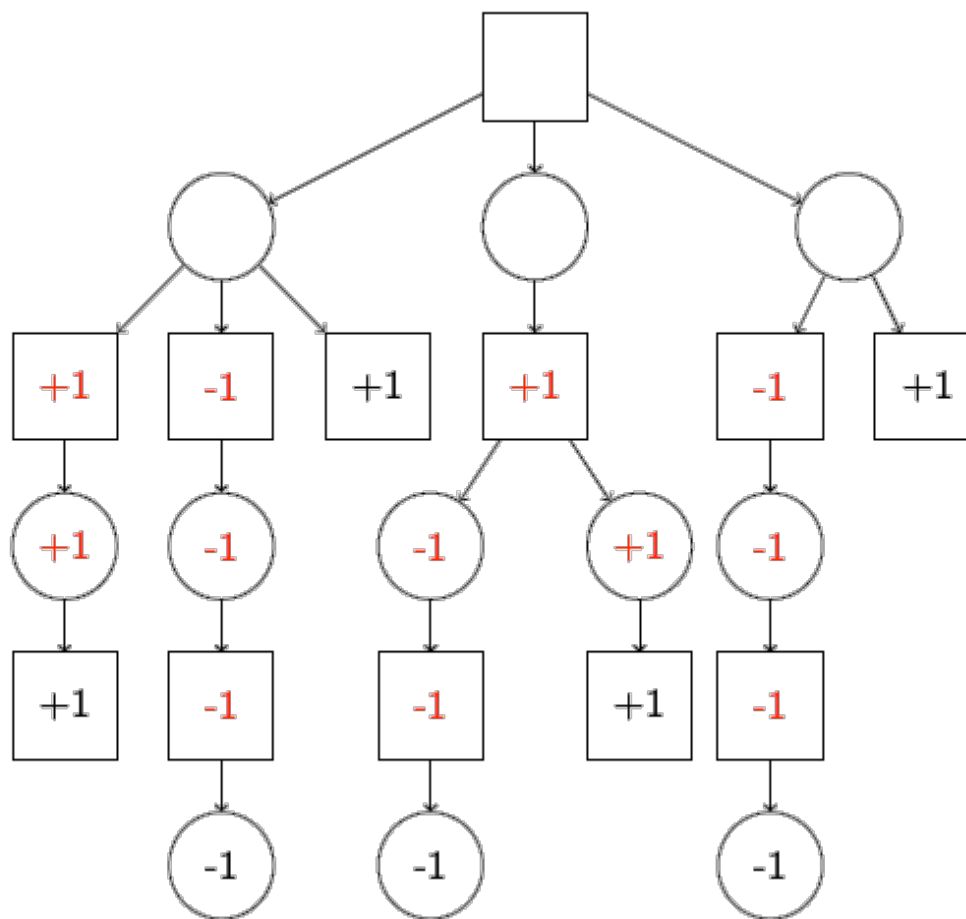
2.6 Ejemplo 2: Aplicación Minimax



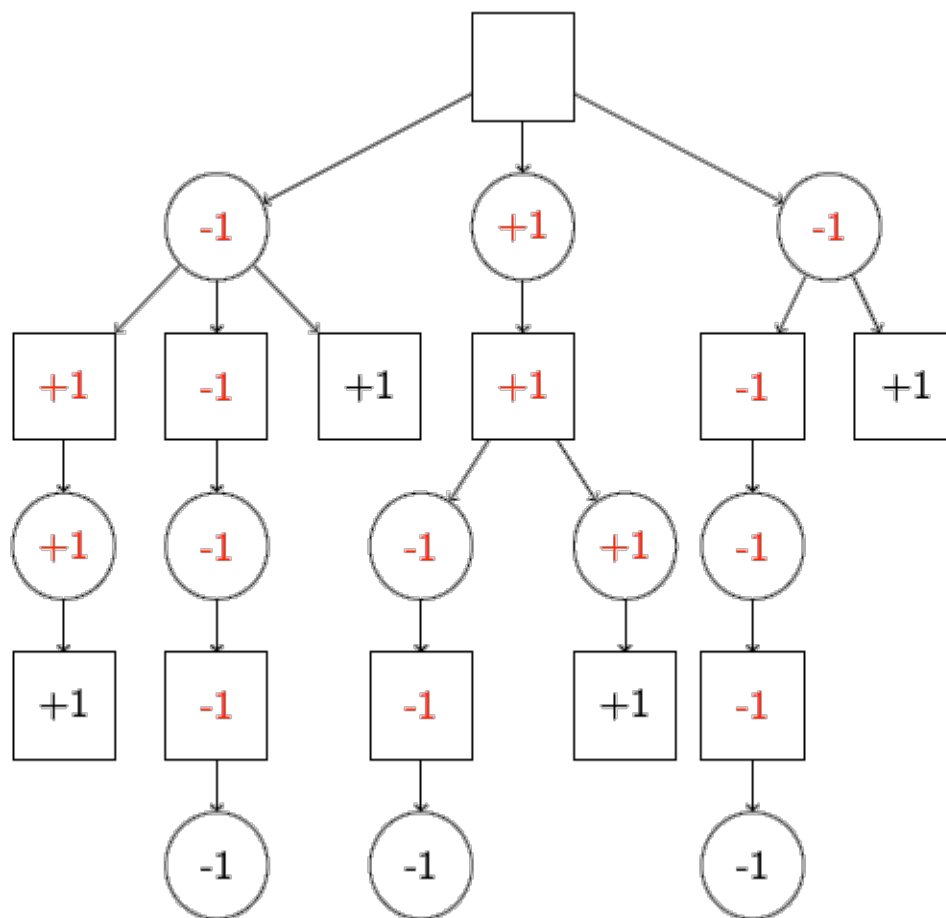
2.6 Ejemplo 2: Aplicación Minimax



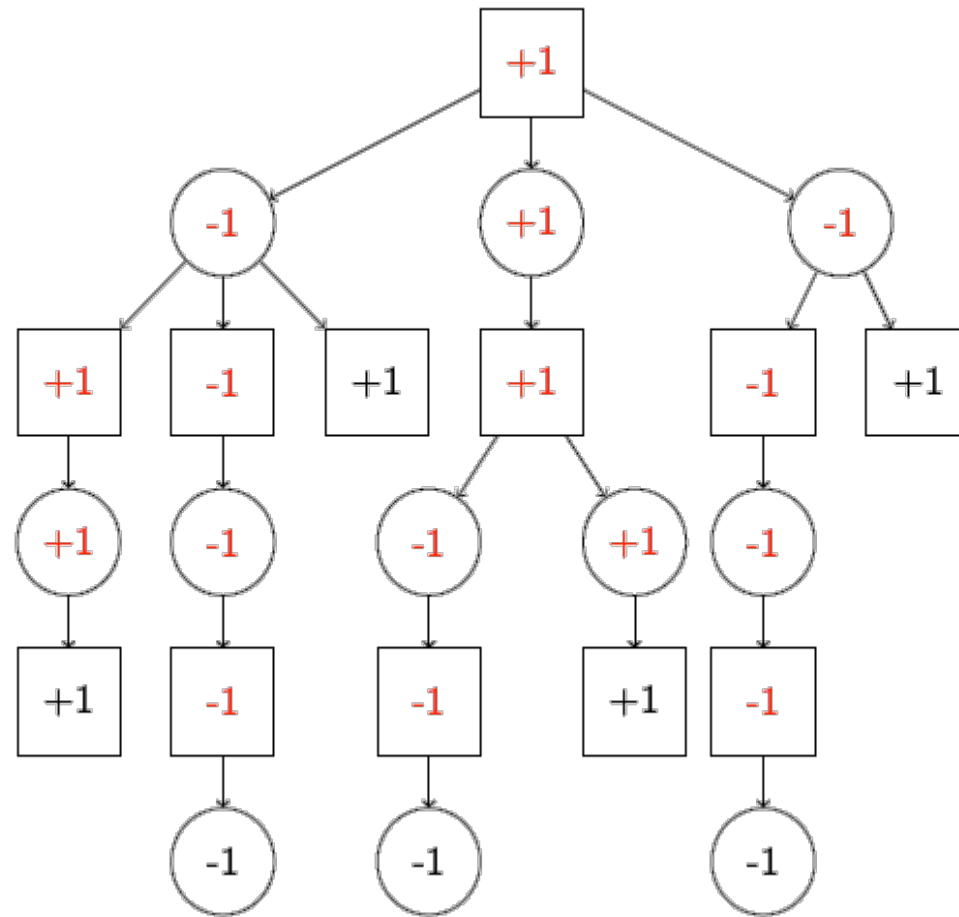
2.6 Ejemplo 2: Aplicación Minimax



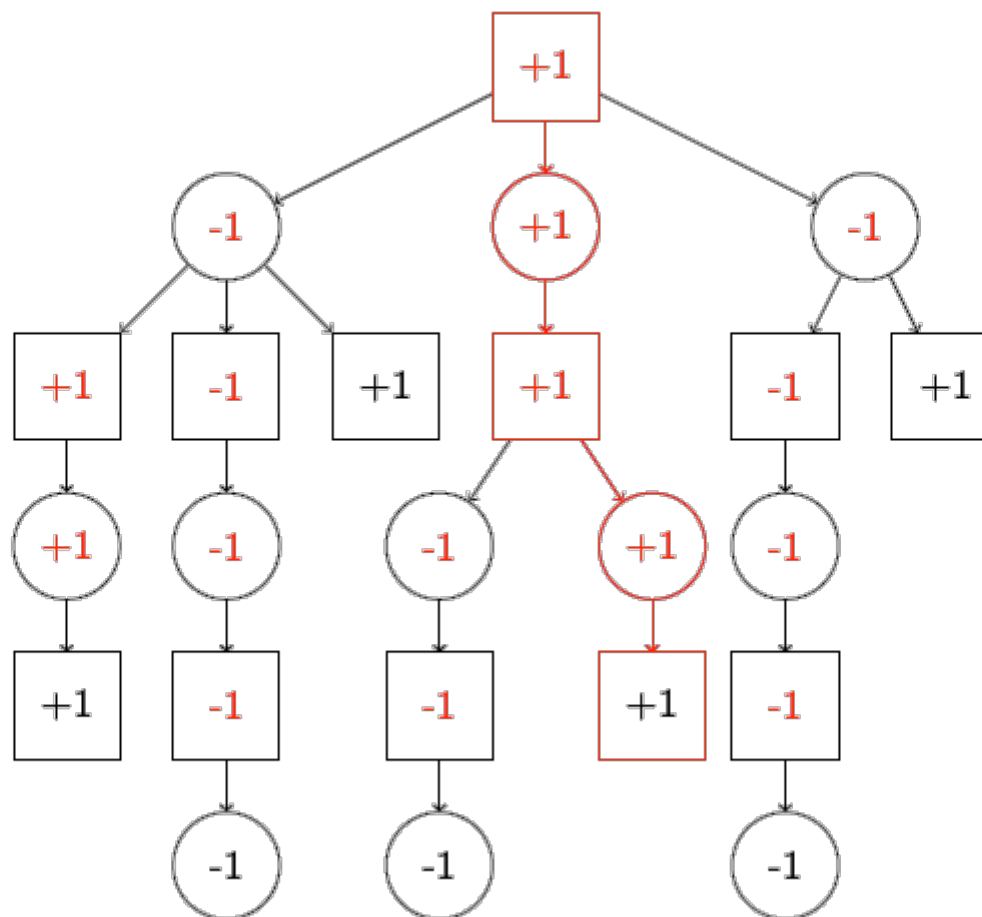
2.6 Ejemplo 2: Aplicación Minimax



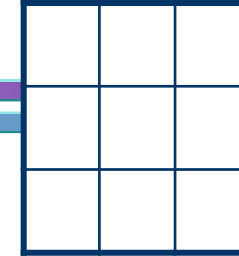
2.6 Ejemplo 2: Aplicación Minimax



2.6 Ejemplo 2: Aplicación Minimax



2.7 Tic-Tac-Toe



- **Estado Inicial:** Tablero Vacío
- **Jugadas:** **Colocar** las fichas en una posición vacía hasta llenar el tablero.
 - **esValida:** comprueba que la posición está vacía
 - **aplicaJugada:** coloca la ficha en la posición
- **Test Terminal:** Comprobar si hay 3 fichas iguales alineadas en la misma fila, en la misma columna o en diagonal, o tablero lleno que no se pueda realizar ningún otro movimiento.
- **Función Utilidad:** +100 para Max, -100 para Min, 0 empate (por ejemplo)

2.7 Tic-Tac-Toe

- Por simetría hay estados que se consideran idénticos:

Por ejemplo en las DIAGONALES

	X	
		O

O		
	X	

		O
	X	

	X	
O		

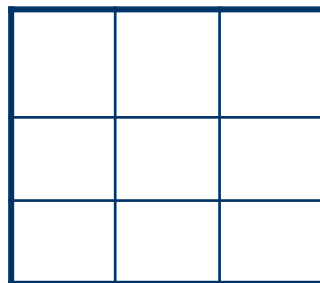
Otro caso

O		
X		

O	X	

2.7 Árbol para Tic-Tac-Toe

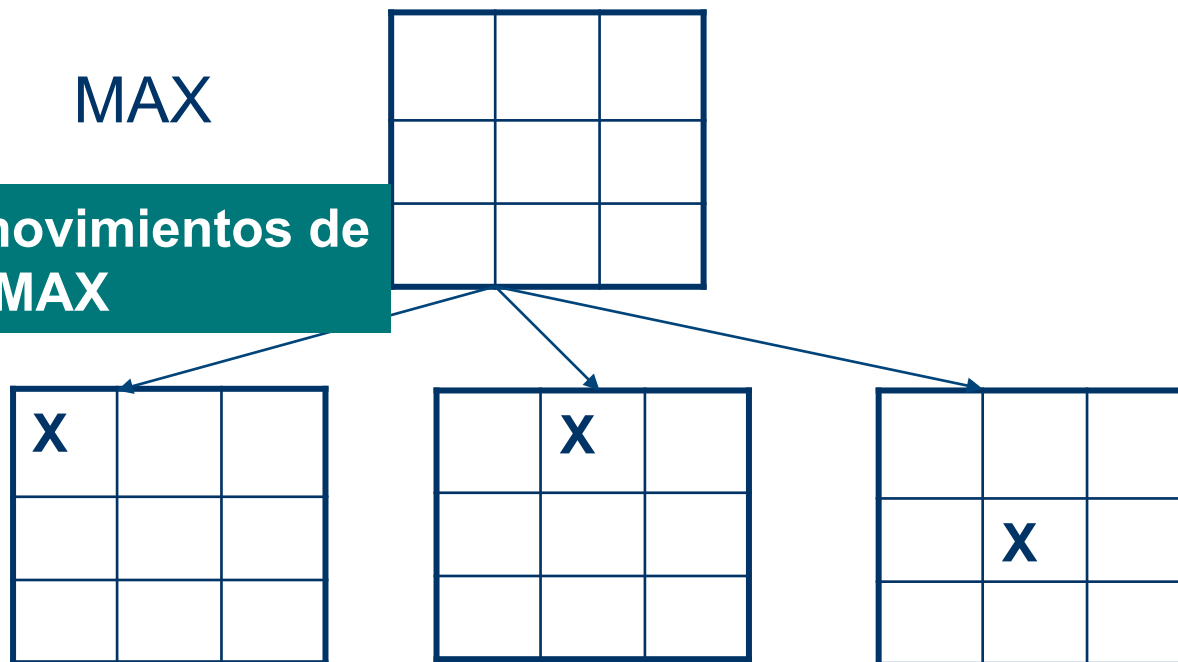
Turno de
MAX



2.7 Árbol para Tic-Tac-Toe

MAX

Posibles movimientos de
MAX

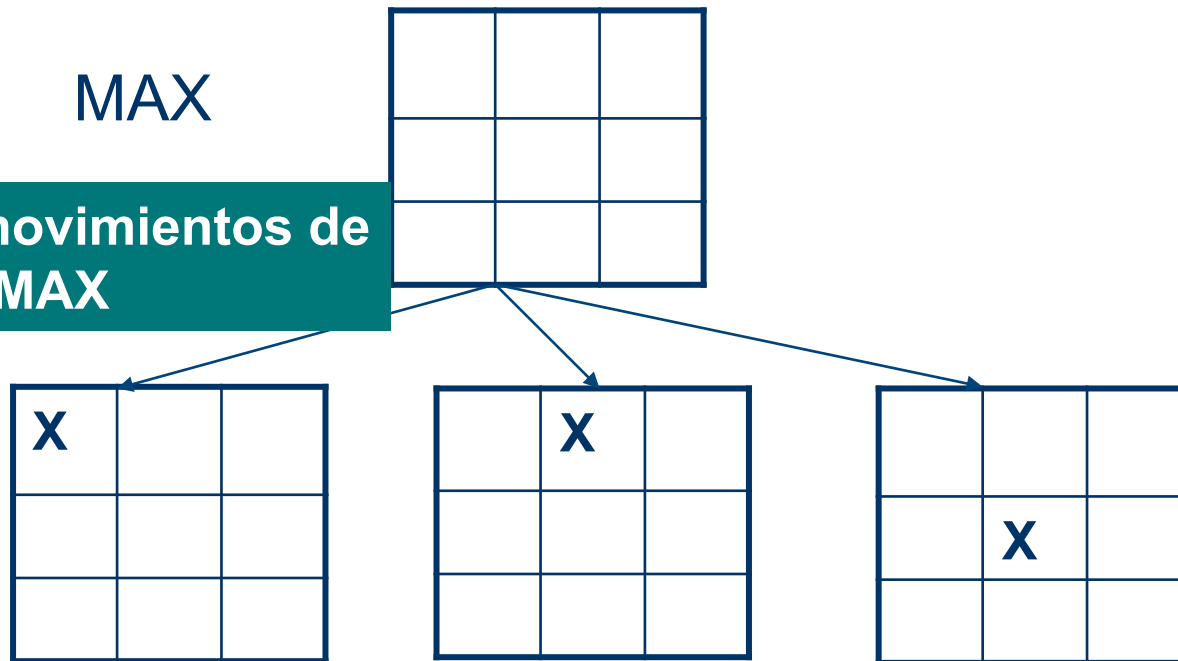


Árbol para Tic-Tac-Toe

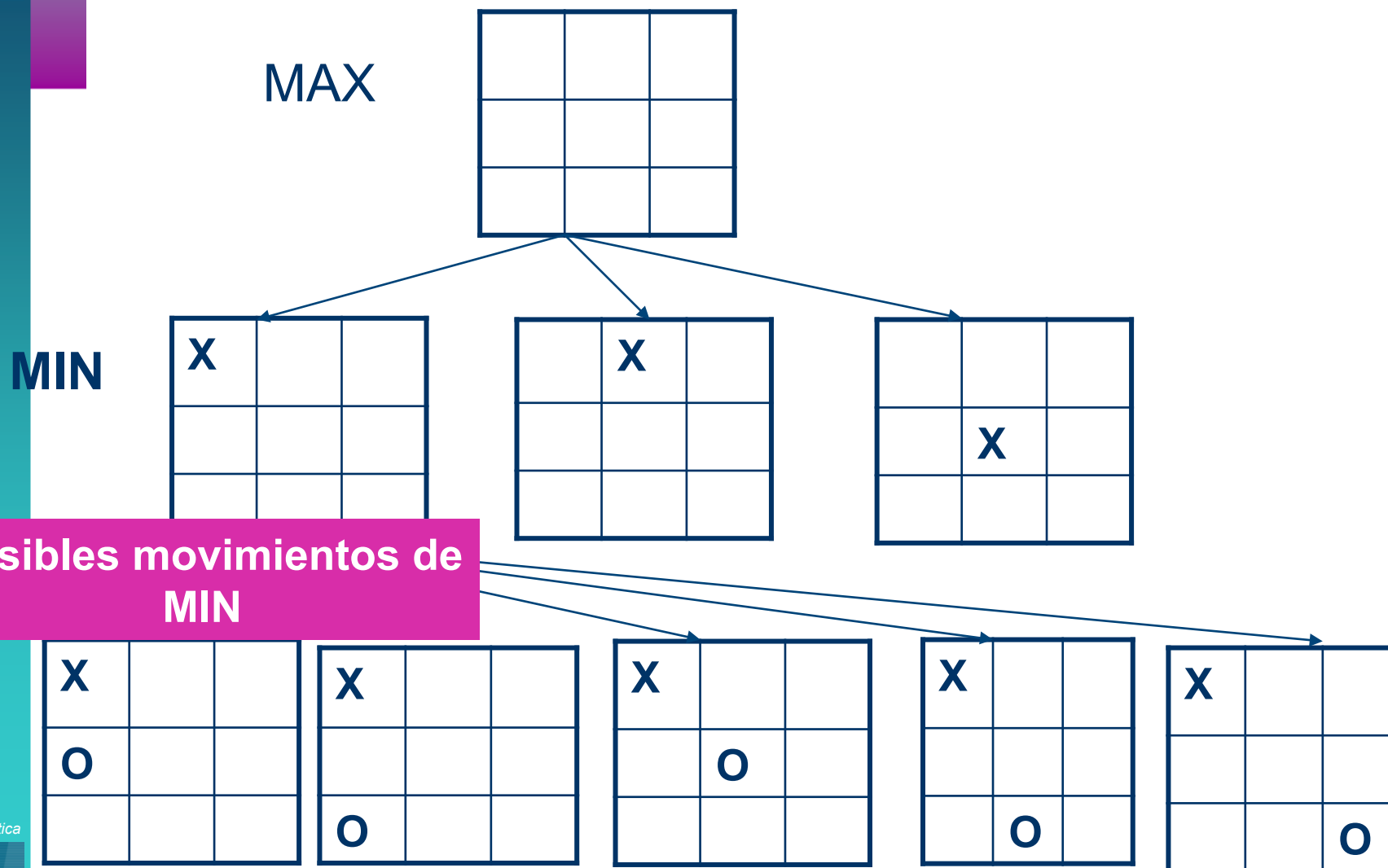
MAX

Posibles movimientos de
MAX

Turno de
MIN



2.7 Árbol para Tic-Tac-Toe



3. 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:
 $(n^{\circ}\ blancas - n^{\circ}\ negras)$
 - 3-en-raya?

3. Funciones Heurísticas para Nodos No Terminales

- Características de las funciones heurísticas:
 - Expresan cómo de buena es la situación actual del juego para un jugador dado (Max o Min)
 - Deben reflejar de manera fiable las posibilidades actuales de ganar
 - Deben estar de acuerdo con la función de utilidad para los nodos terminales
 - No deben ser muy complejas (evitar altos costes computacionales)
 - Esta función no representa ningún coste de llegar a la solución, ni es una estimación de la distancia al objetivo en pasos

3.1 Funciones Heurísticas para Nodos No Terminales

- Tic-Tac-Toe:

- N° de columnas, filas y diagonales con X y sin O

- N° de columnas, filas y diagonales con O y sin X



X	O	
	O	
	X	X

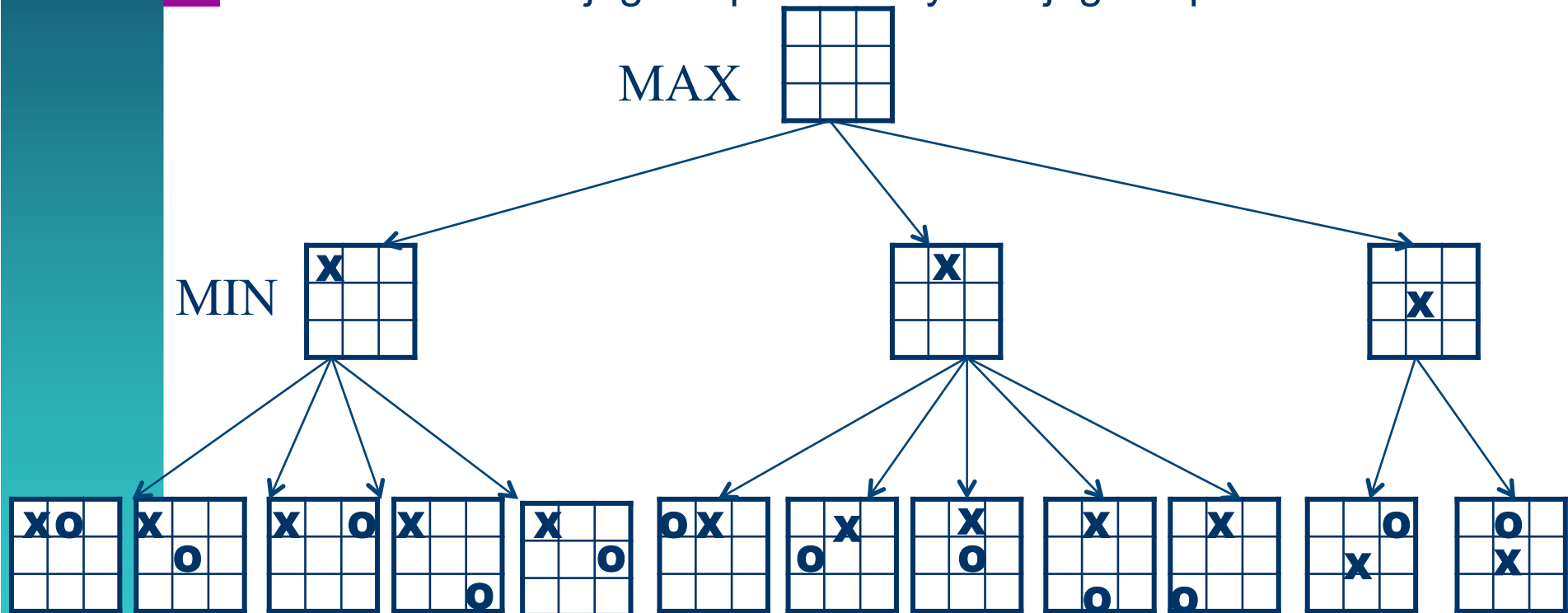
MAX : columna 1, columna 3, fila 3

MIN: fila 2, diagonal 2

$$f = 3 - 2 = 1$$

3.2 Función Heurística para Nodos No Terminales

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

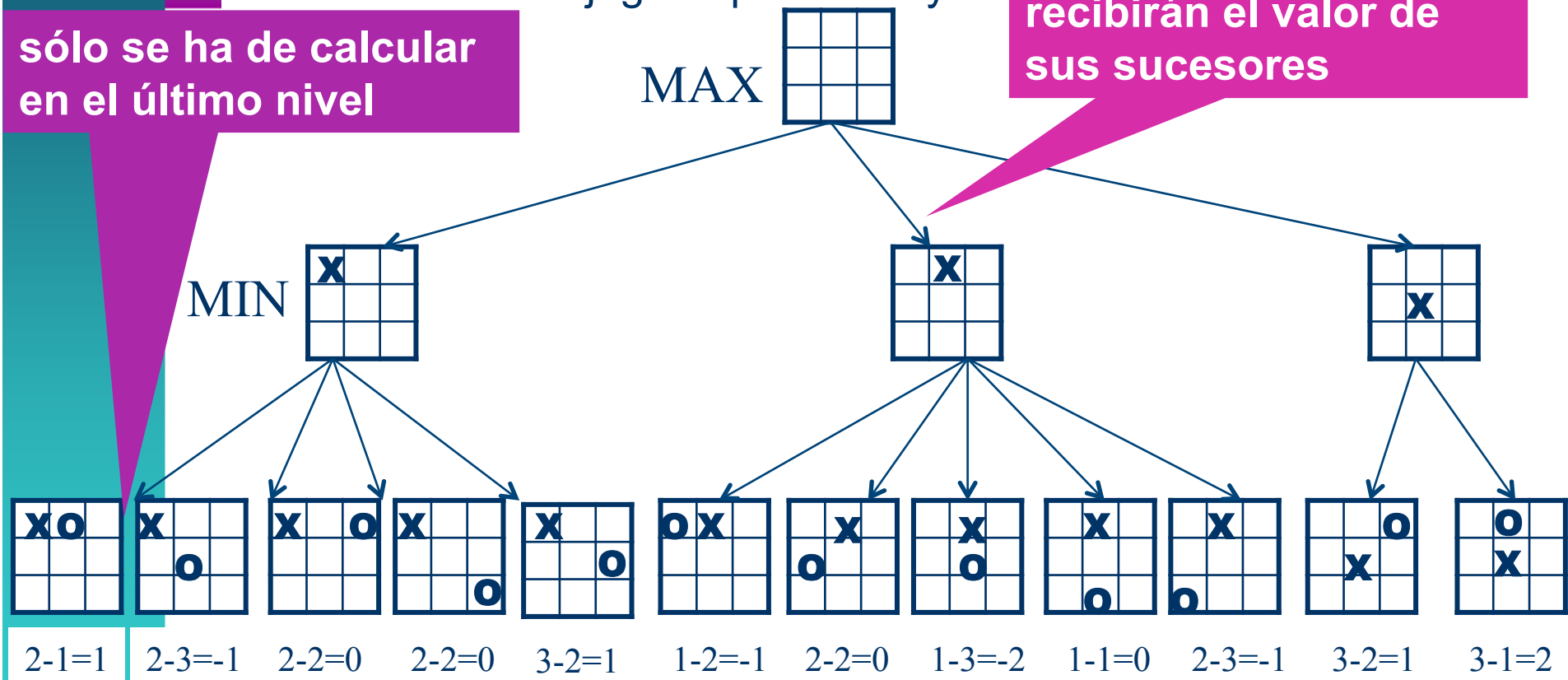


3.2 Función Heurística para Nodos No Terminales

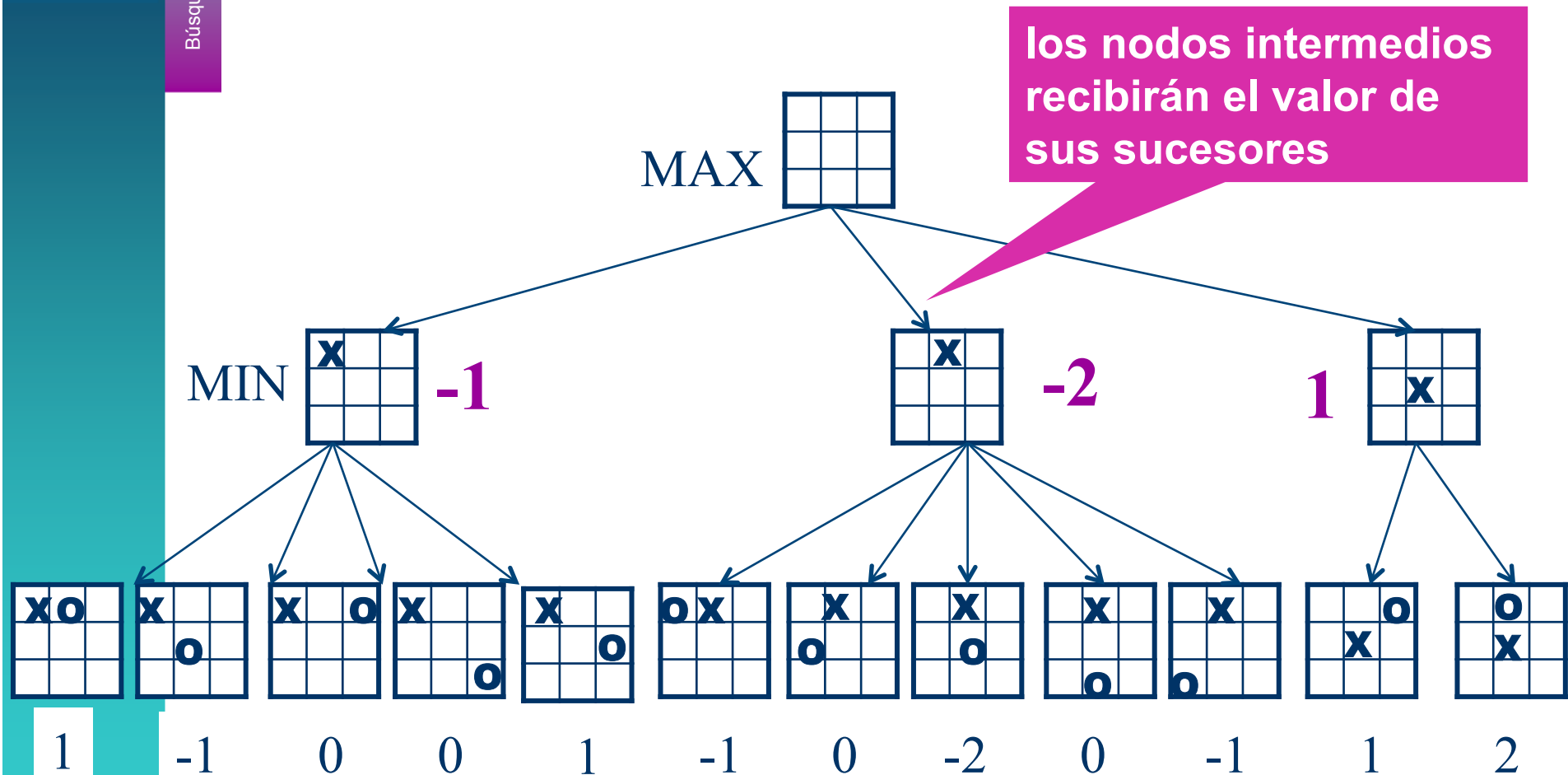
- Calcula la función heurística en este nivel y considera una jugada para Max y un

sólo se ha de calcular en el último nivel

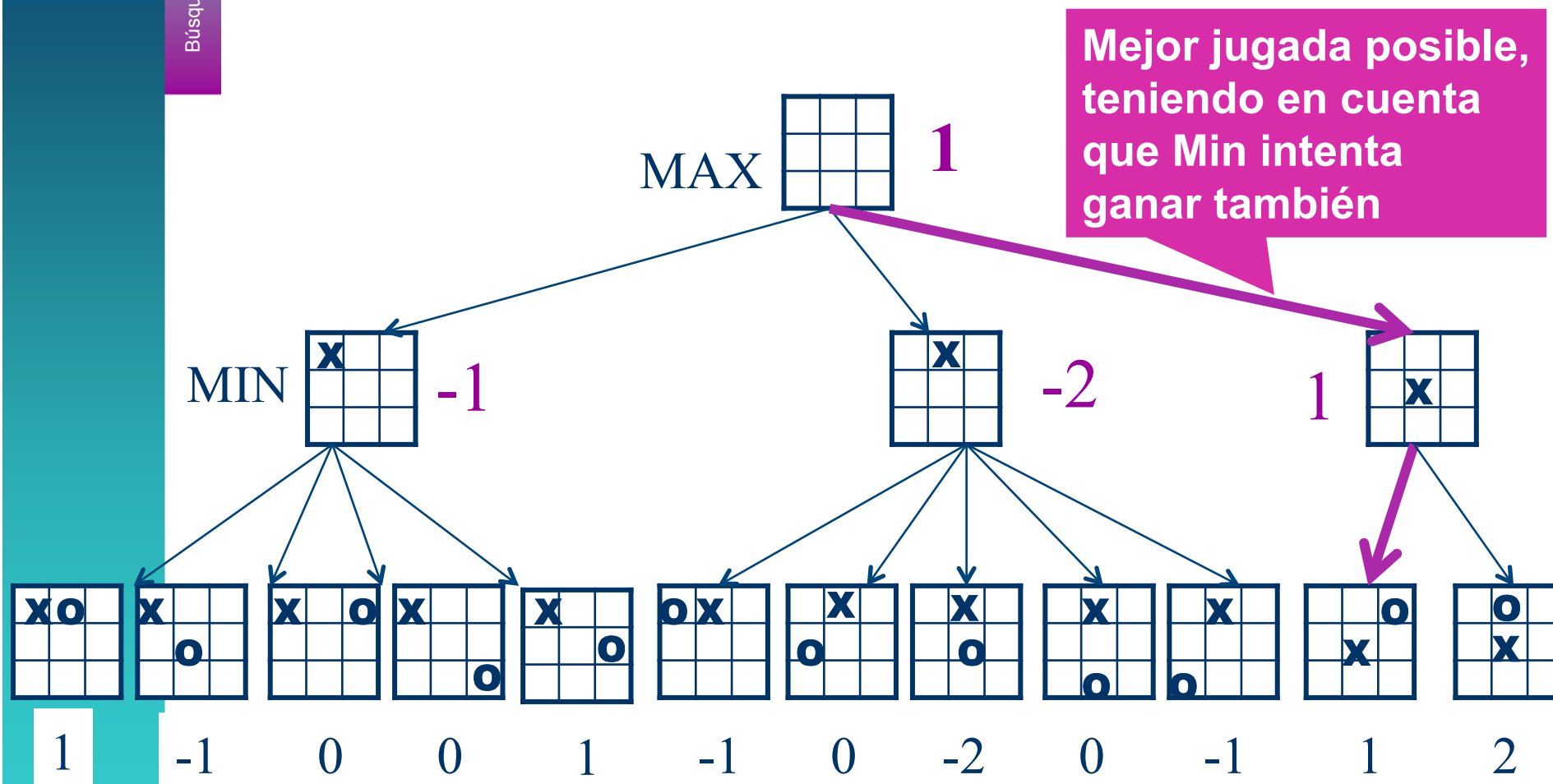
los nodos intermedios recibirán el valor de sus sucesores



3.3 Aplicación de la estrategia MiniMax con una Función Heurística



3.3 Aplicación de la estrategia MiniMax con una Función Heurística



4. 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**

4. 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

Podar cuando en un nodo $\alpha \geq \beta$

4.1 Algoritmo Poda α - β

```
tNodo: función poda_ab(E/S tNodo: nodo, E entero: jugador)
var
  entero: max_actual, jugada, mejorJugada, prof, v
  tNodo: intento
inicio
  alfa ← -infinito   beta ← +infinito   prof ← 1
  mientras jugada ≤ 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
    jugada ← jugada+1
  fin_mientras
  nodo = aplicaJugada(nodo, jugador, mejorJugada);
  devolver nodo
fin_función
```

4.1 Seudocódigo función valorMin α - β

```
entero: función valorMin_ab(E tNodo: nodo, E entero: jugador, E entero: prof,  
                             E entero: alfa, E entero: beta)  
  
var  
    entero: vmin, jugada  tNodo: intento  
  
inicio  
    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  
    devuelve vmin  
fin_función
```

4.1 Seudocódigo función valorMax α - β

entero: función **valorMax_ab**(E tNodo: nodo, E entero: jugador, E entero: prof,
E entero: alfa, E entero: beta)

var

entero: vmax, jugada **tNodo:** intento

inicio

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

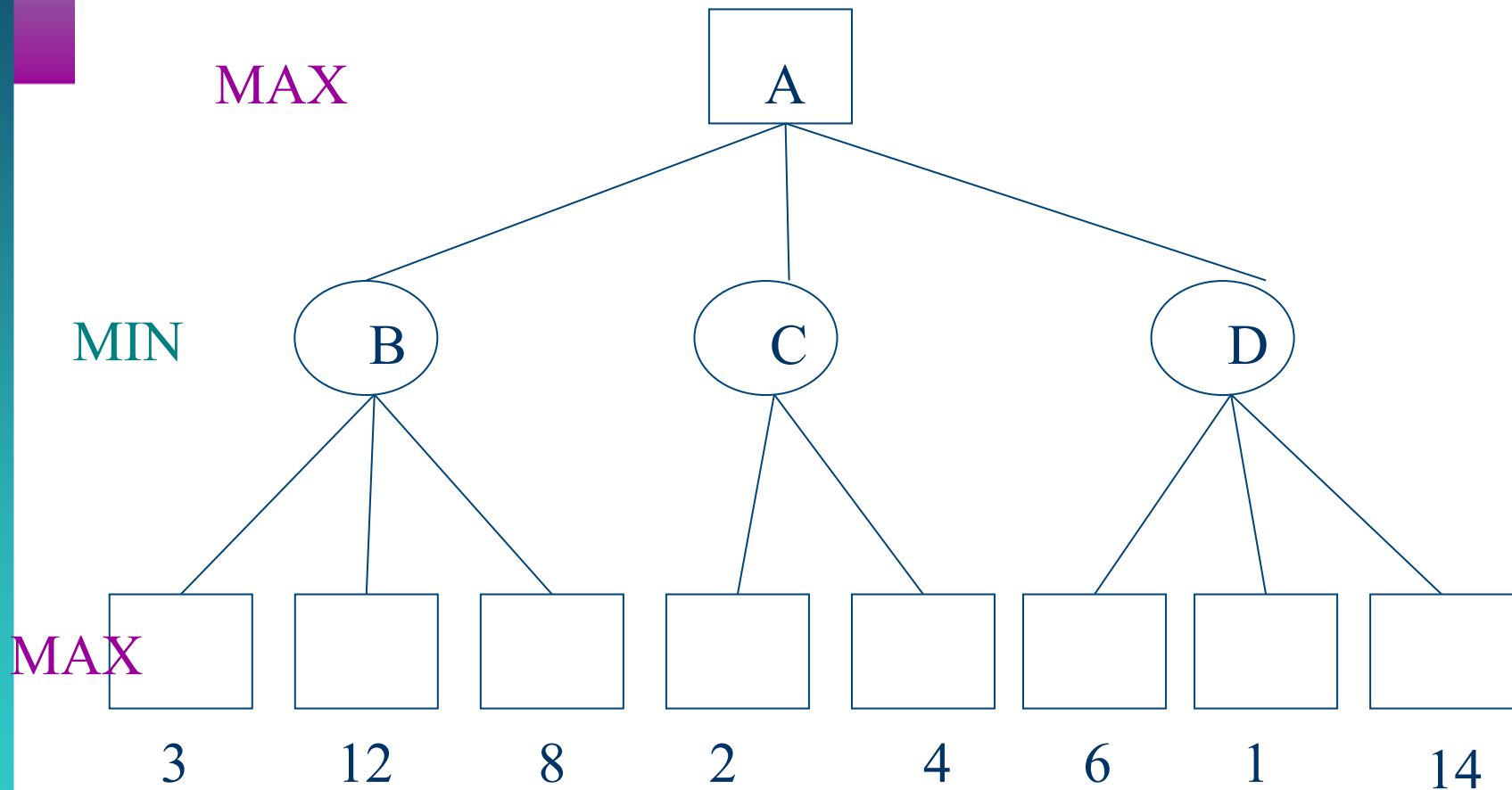
vmax ← alfa

fin_si

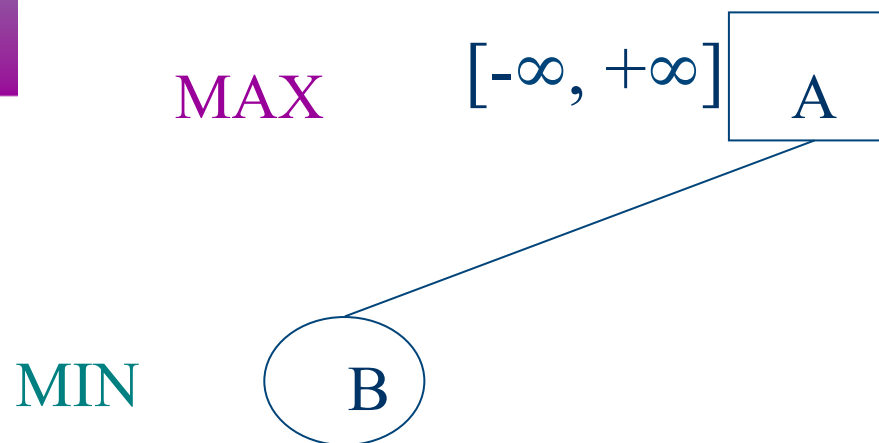
devuelve vmax

fin_función

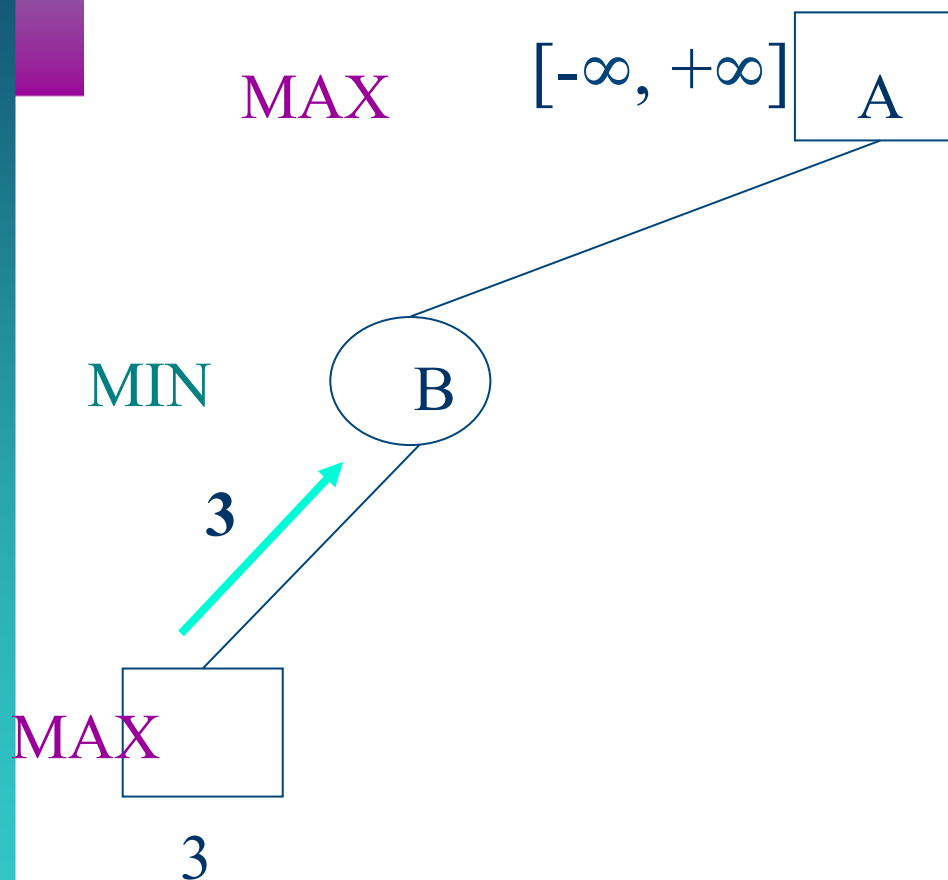
4.2 Ejemplo 3: Poda α - β



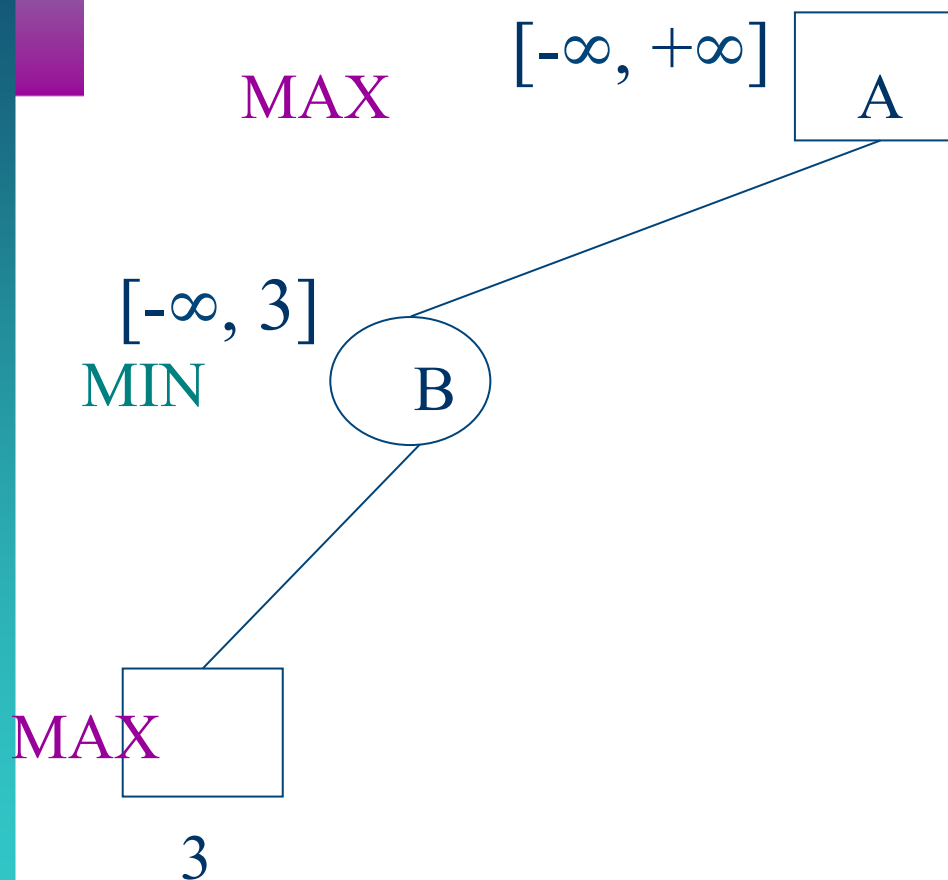
4.2 Ejemplo 3: Poda α - β



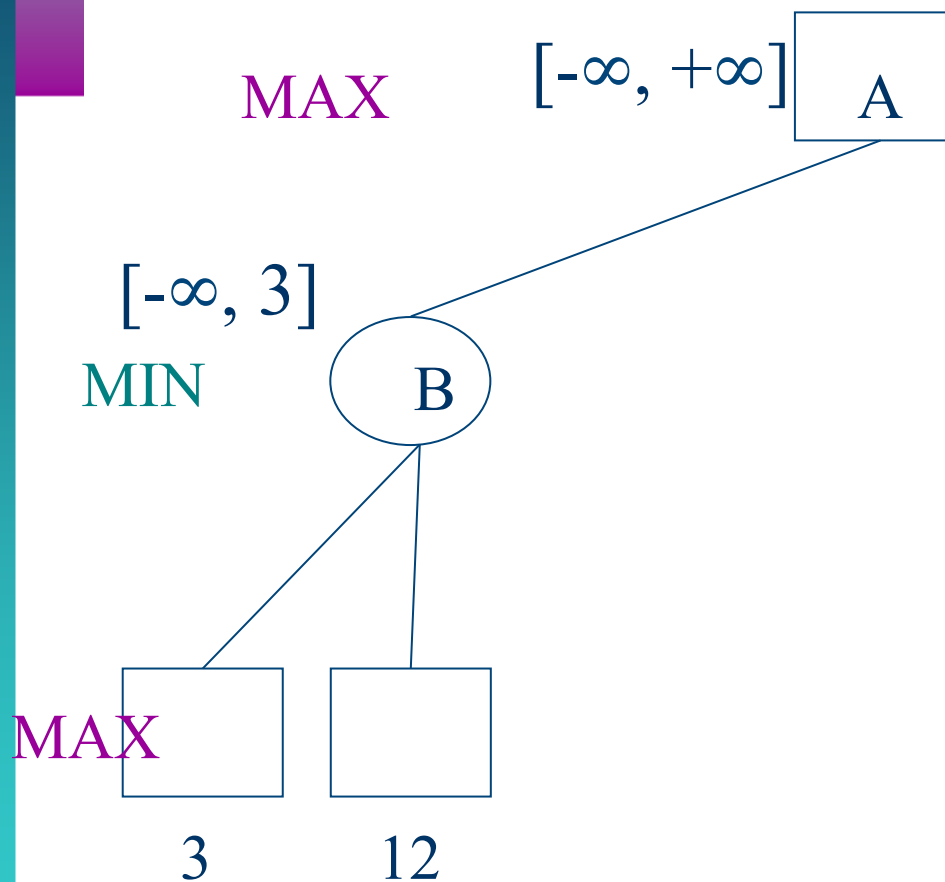
4.2 Ejemplo 3: Poda α - β



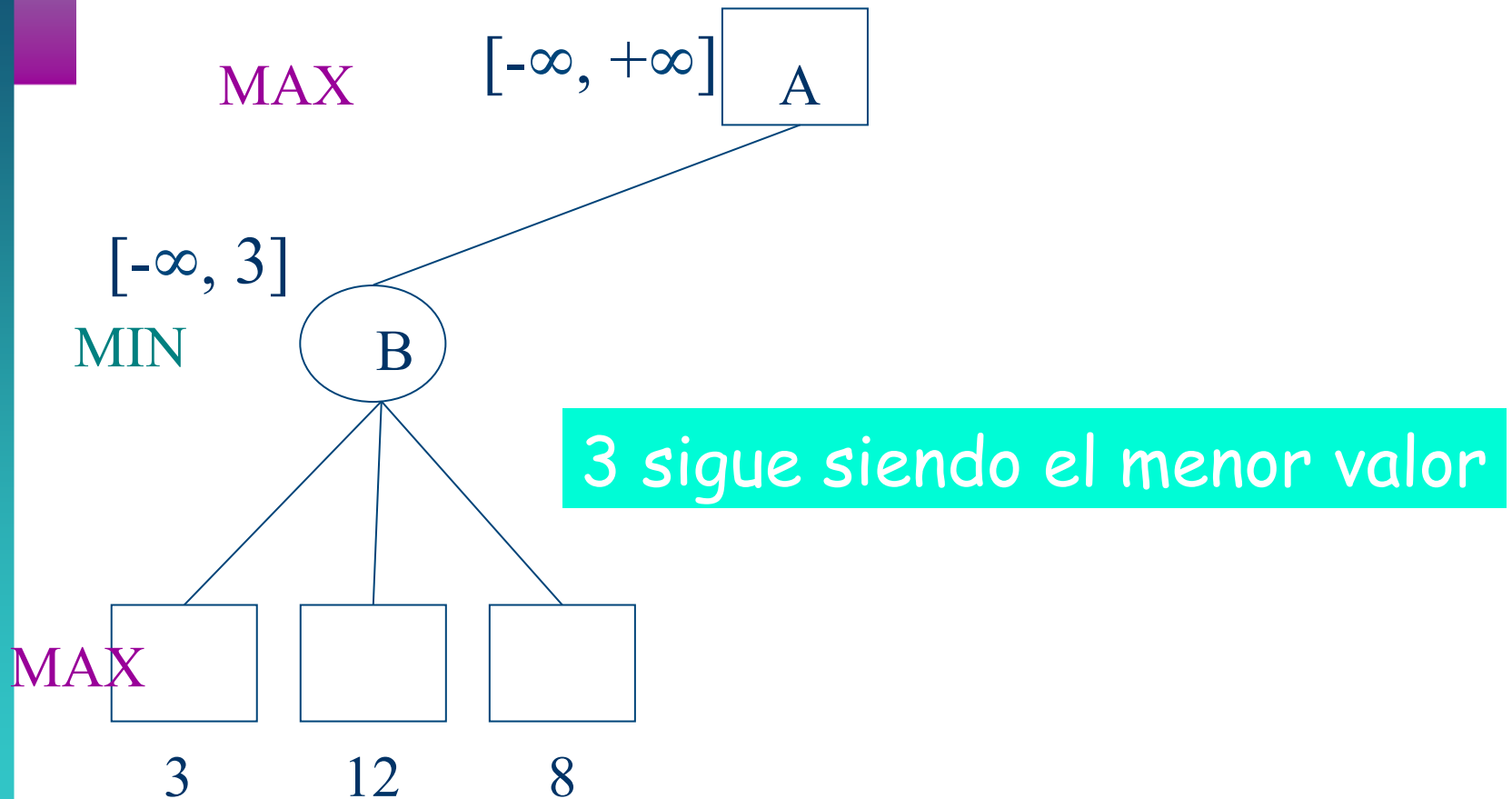
4.2 Ejemplo 3: Poda α - β



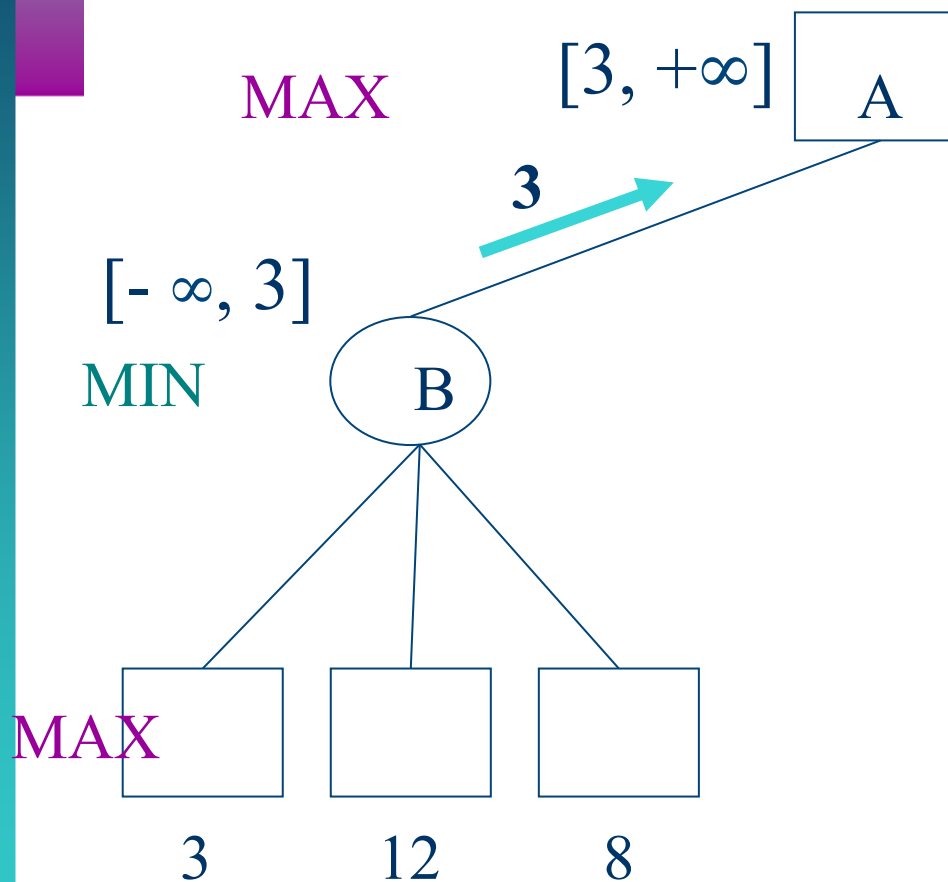
4.2 Ejemplo 3: Poda α - β



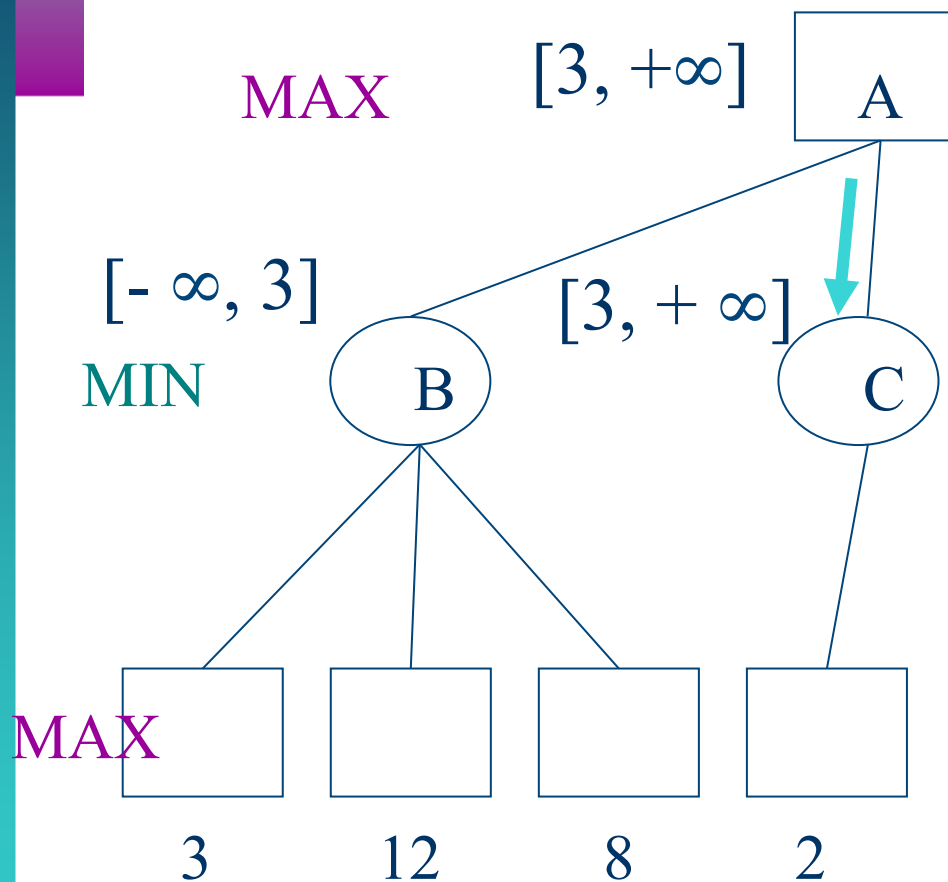
4.2 Ejemplo 3: Poda α - β



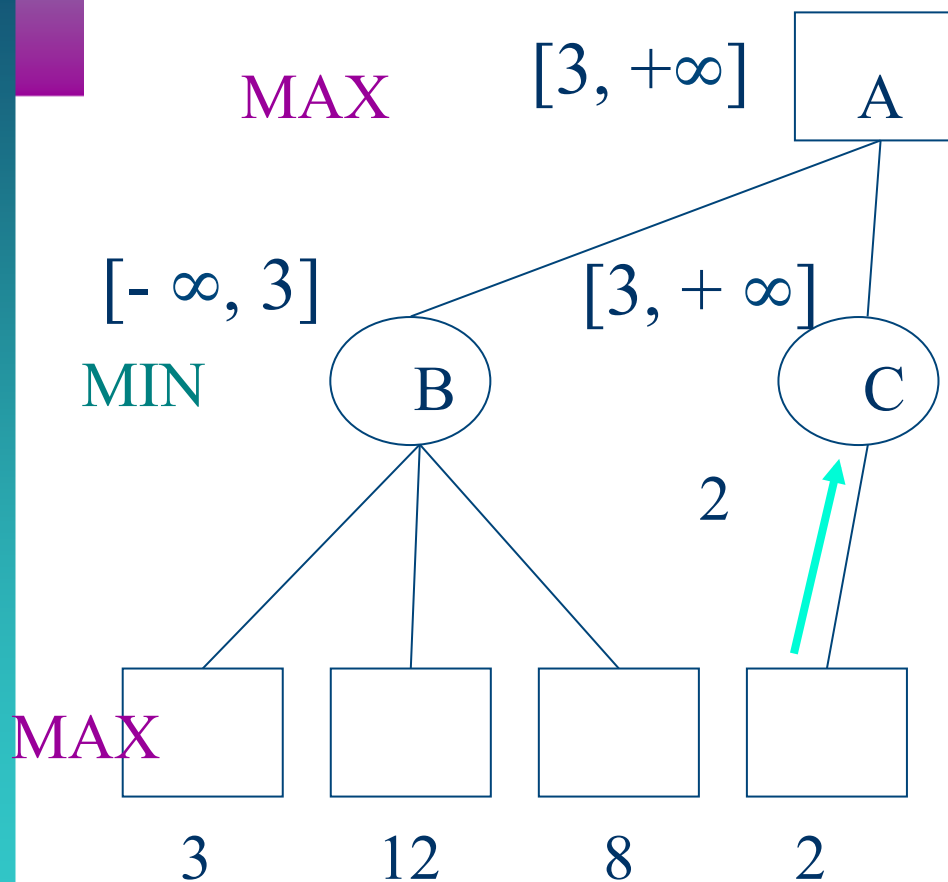
4.2 Ejemplo 3: Poda α - β



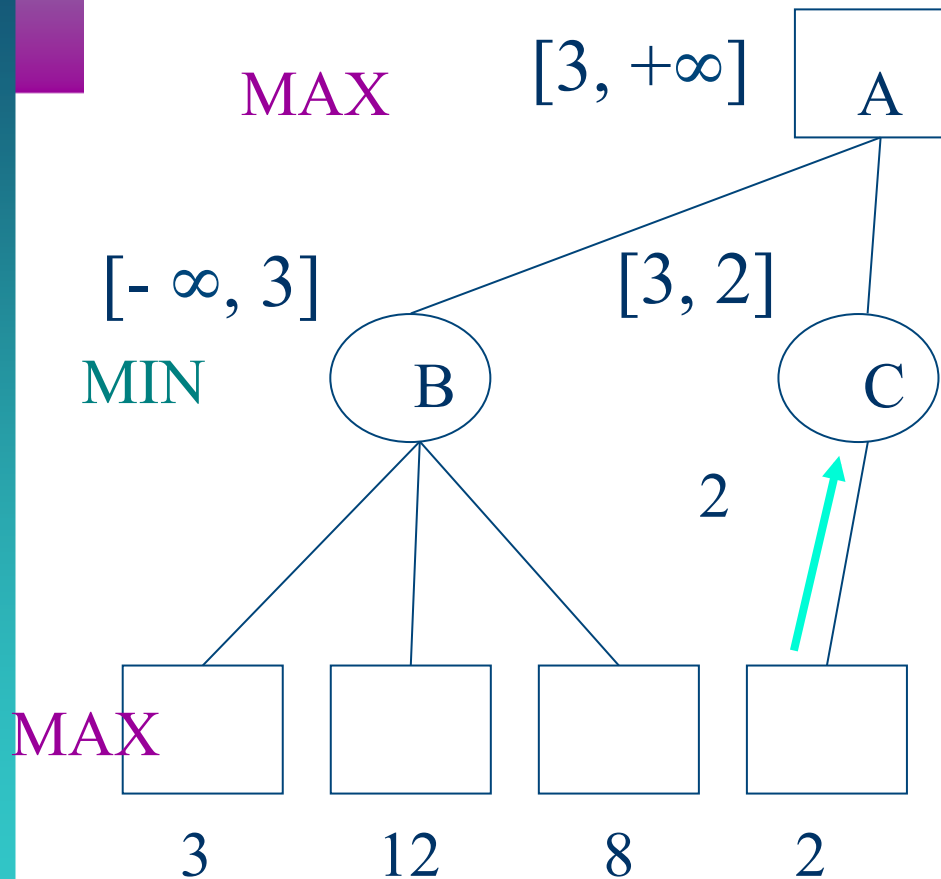
4.2 Ejemplo 3: Poda α - β



4.2 Ejemplo 3: Poda α - β

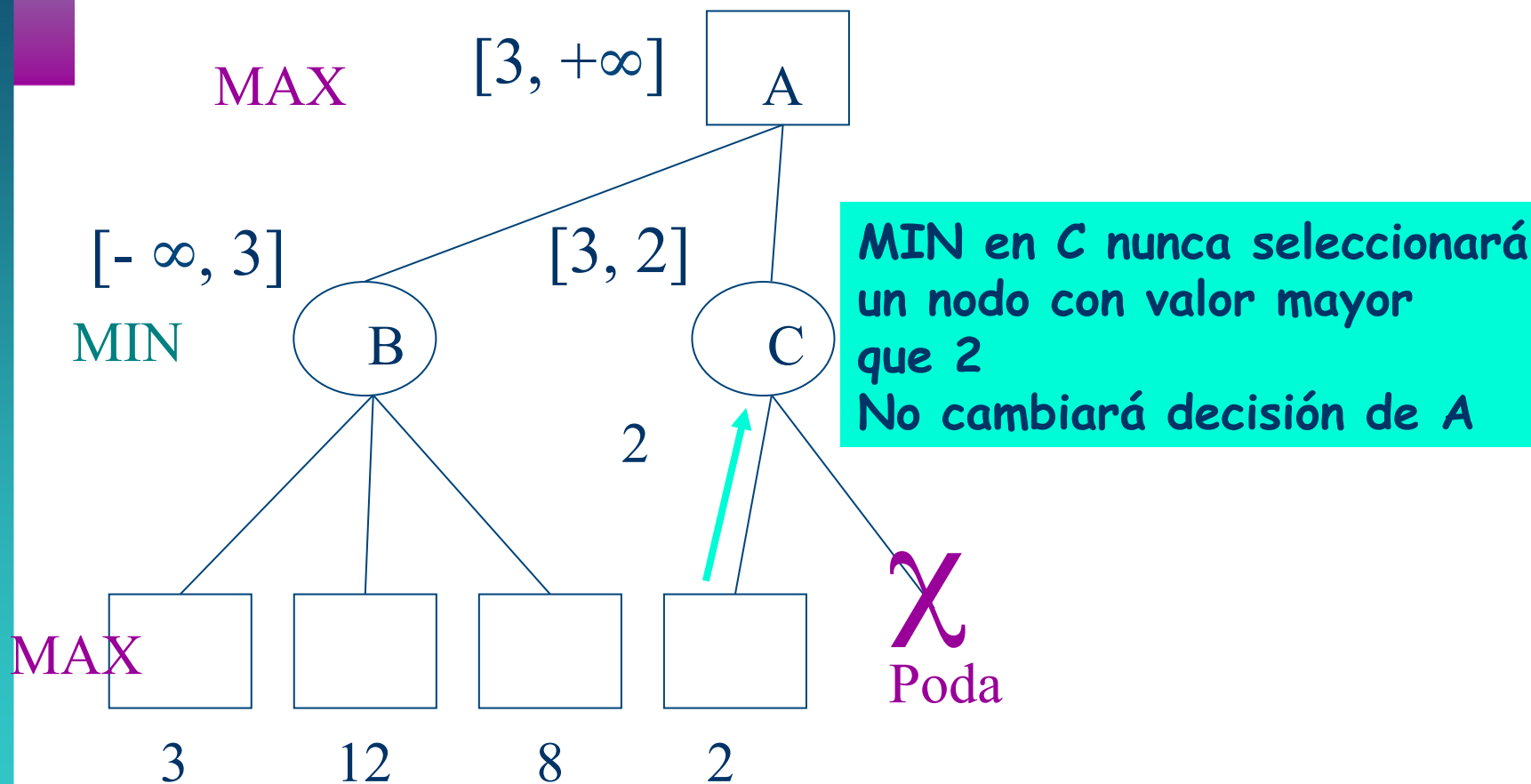


4.2 Ejemplo 3: Poda α - β

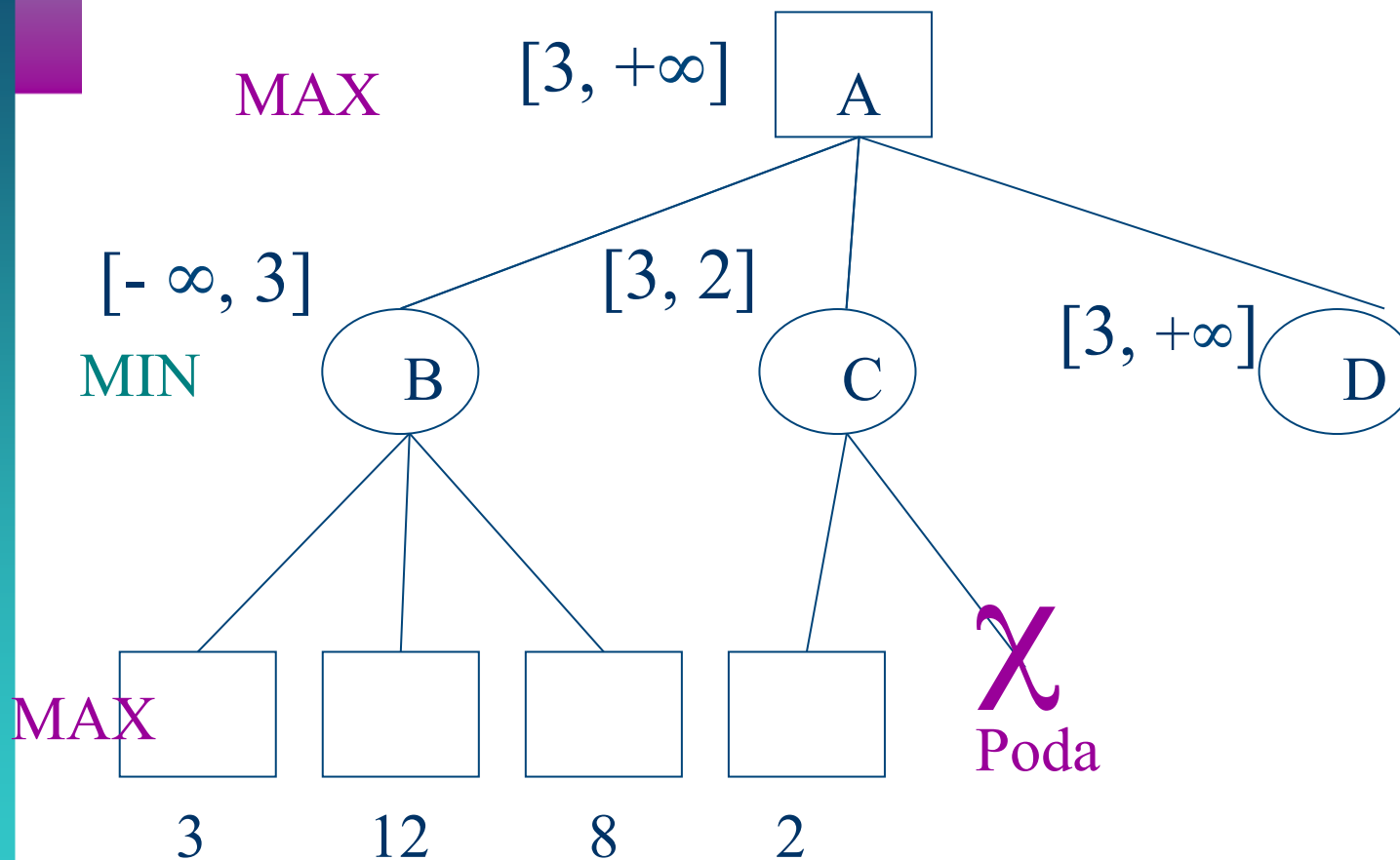


MIN en C nunca seleccionará un nodo con valor mayor que 2
No cambiará decisión de A

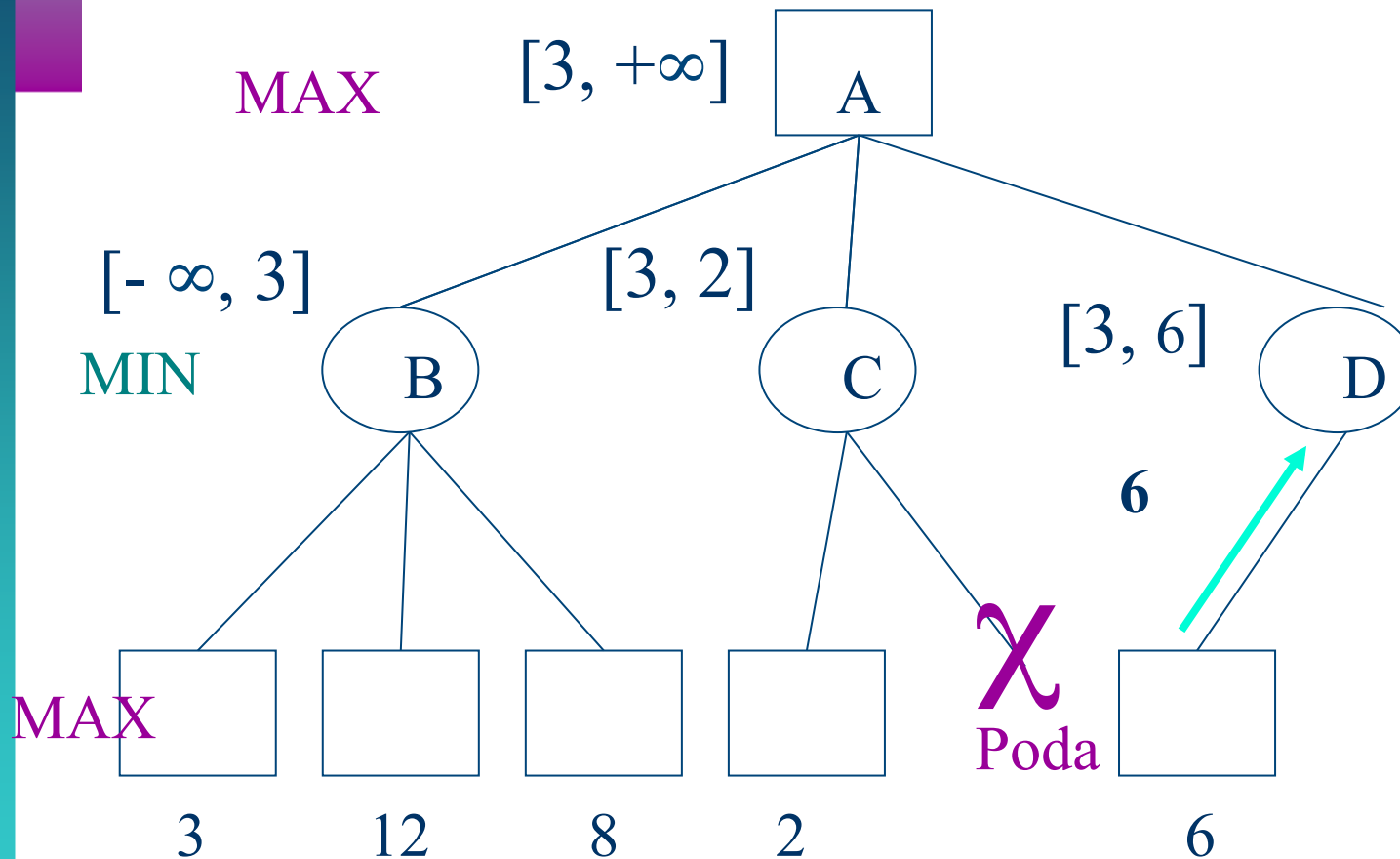
4.2 Ejemplo 3: Poda α - β



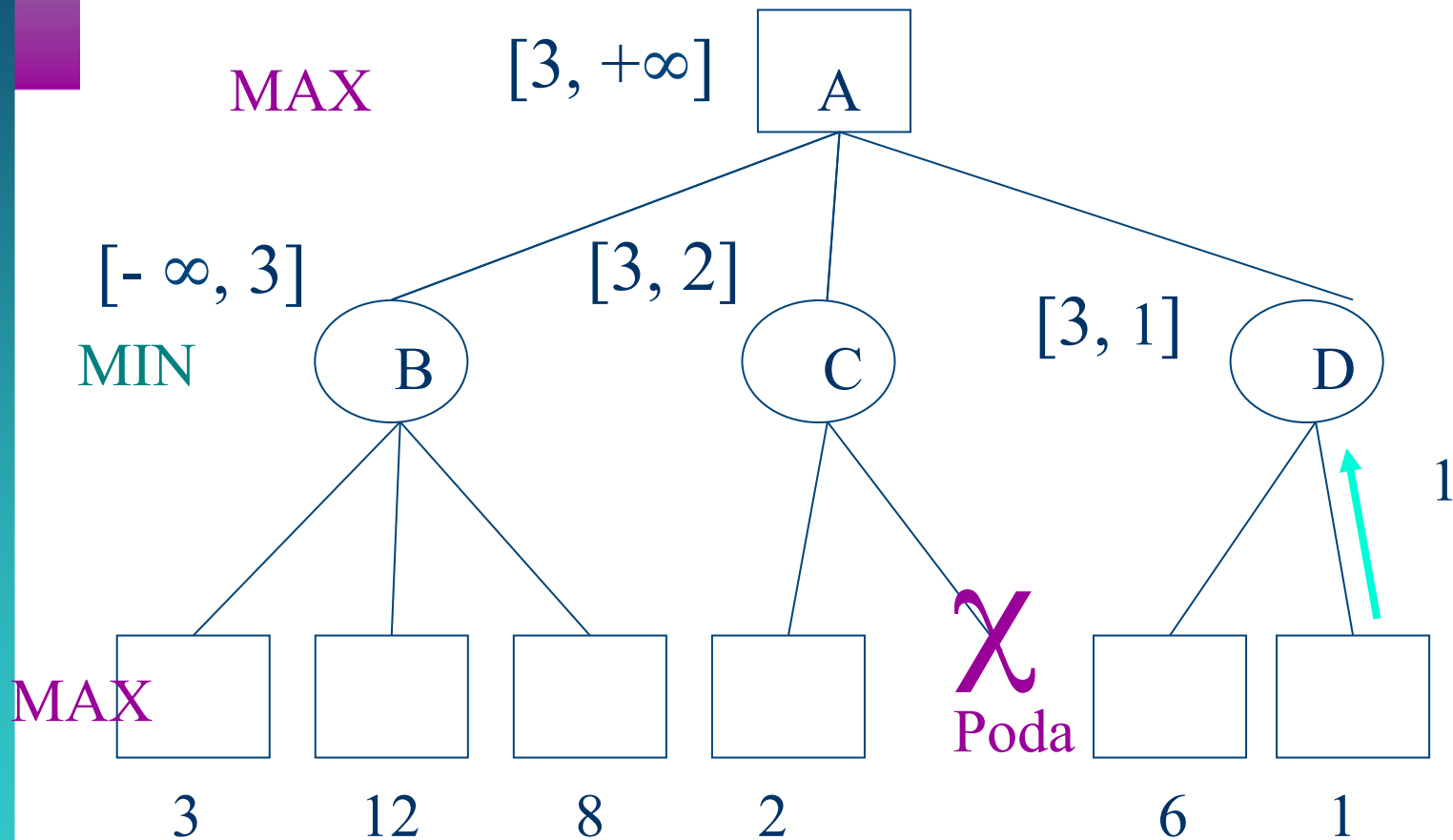
4.2 Ejemplo 3: Poda α - β



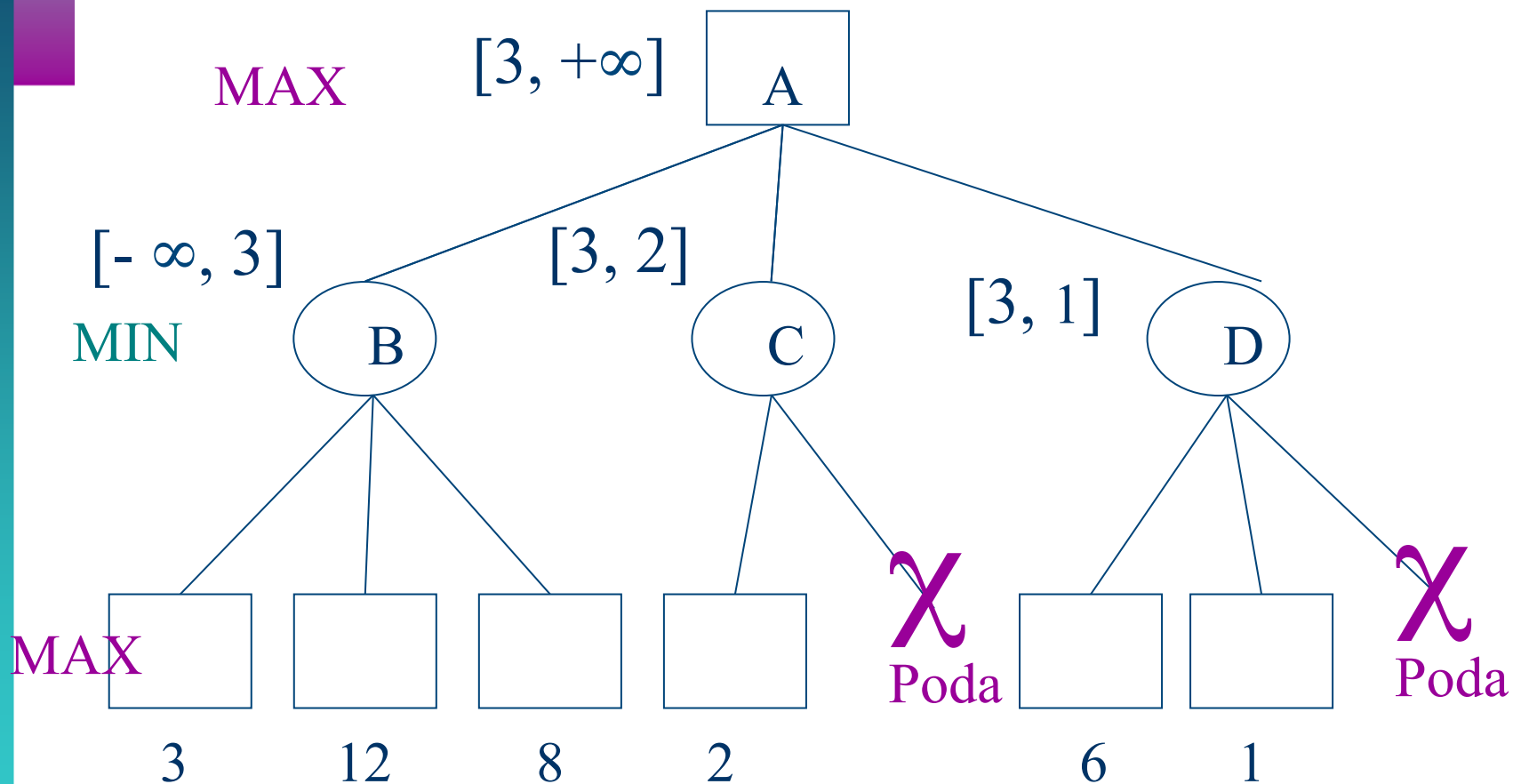
4.2 Ejemplo 3: Poda α - β



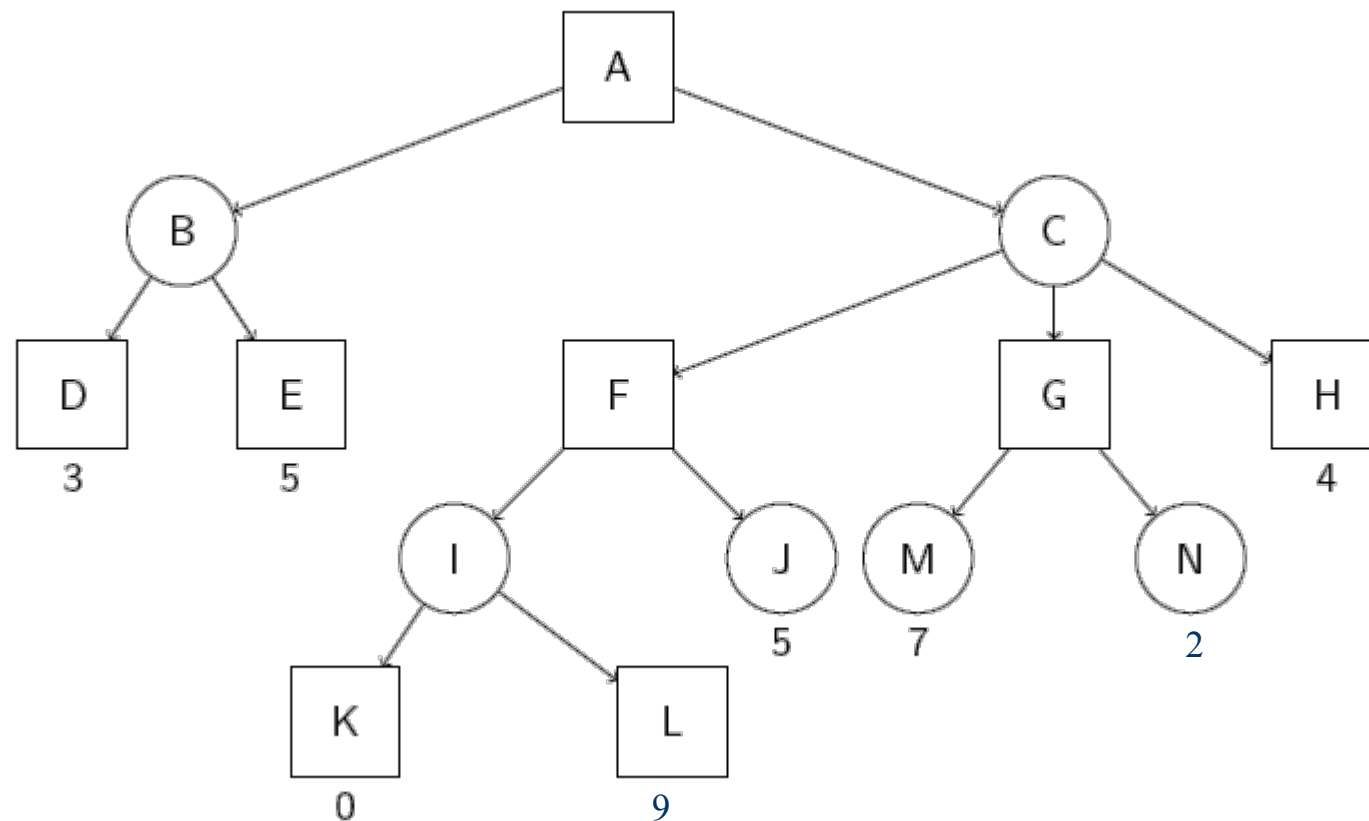
4.2 Ejemplo 3: Poda α - β



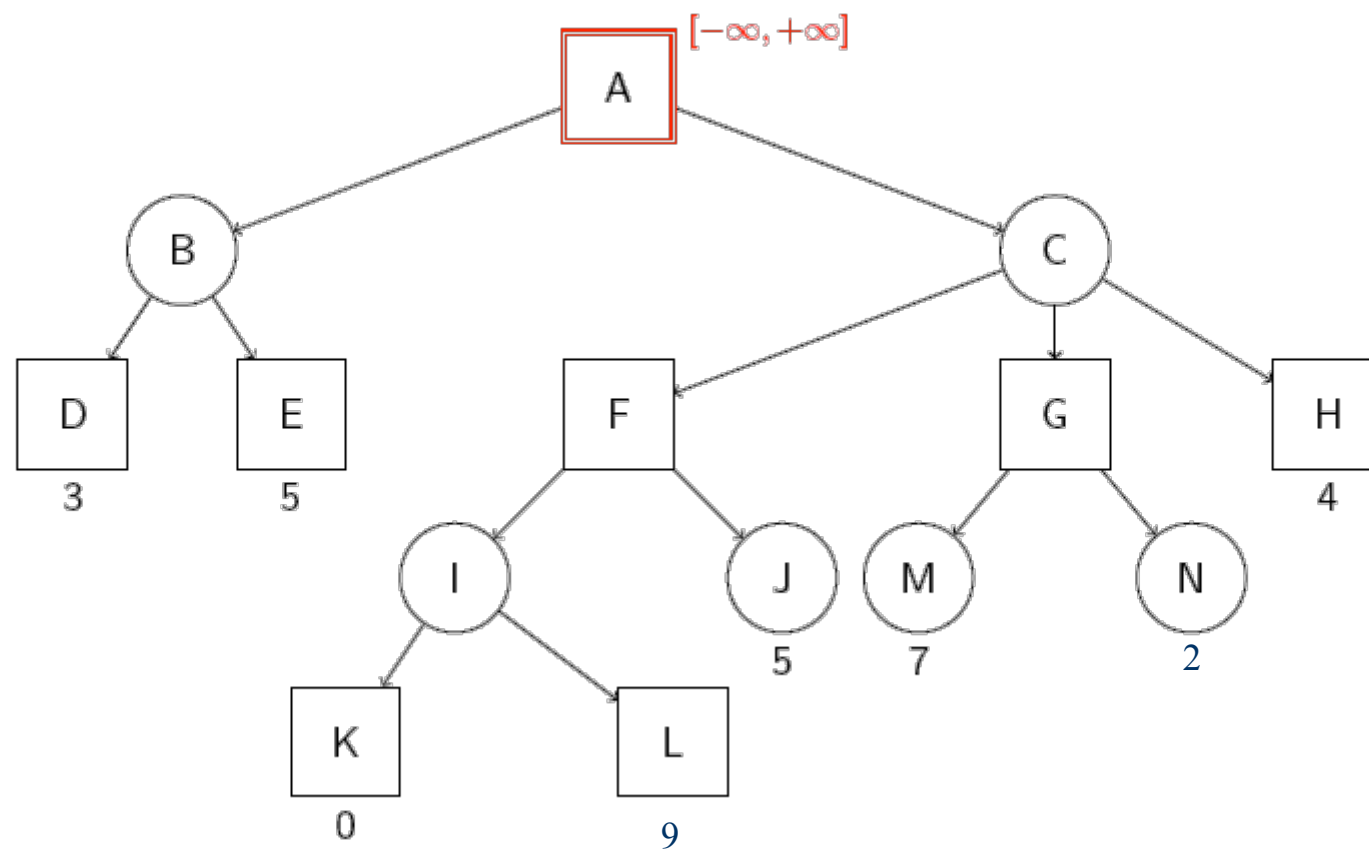
4.2 Ejemplo 3: Poda α - β



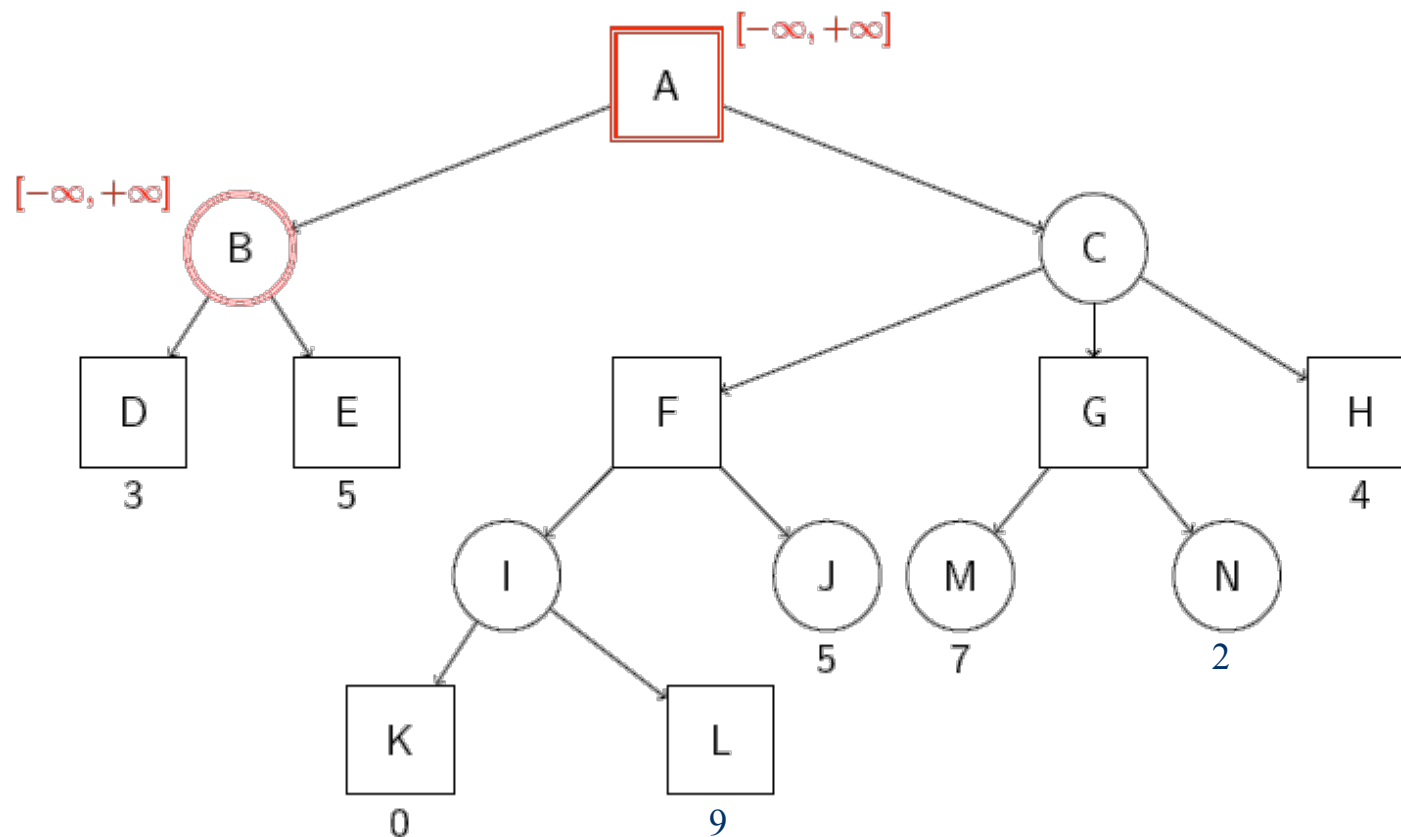
4.3 Ejemplo 4: Poda α - β



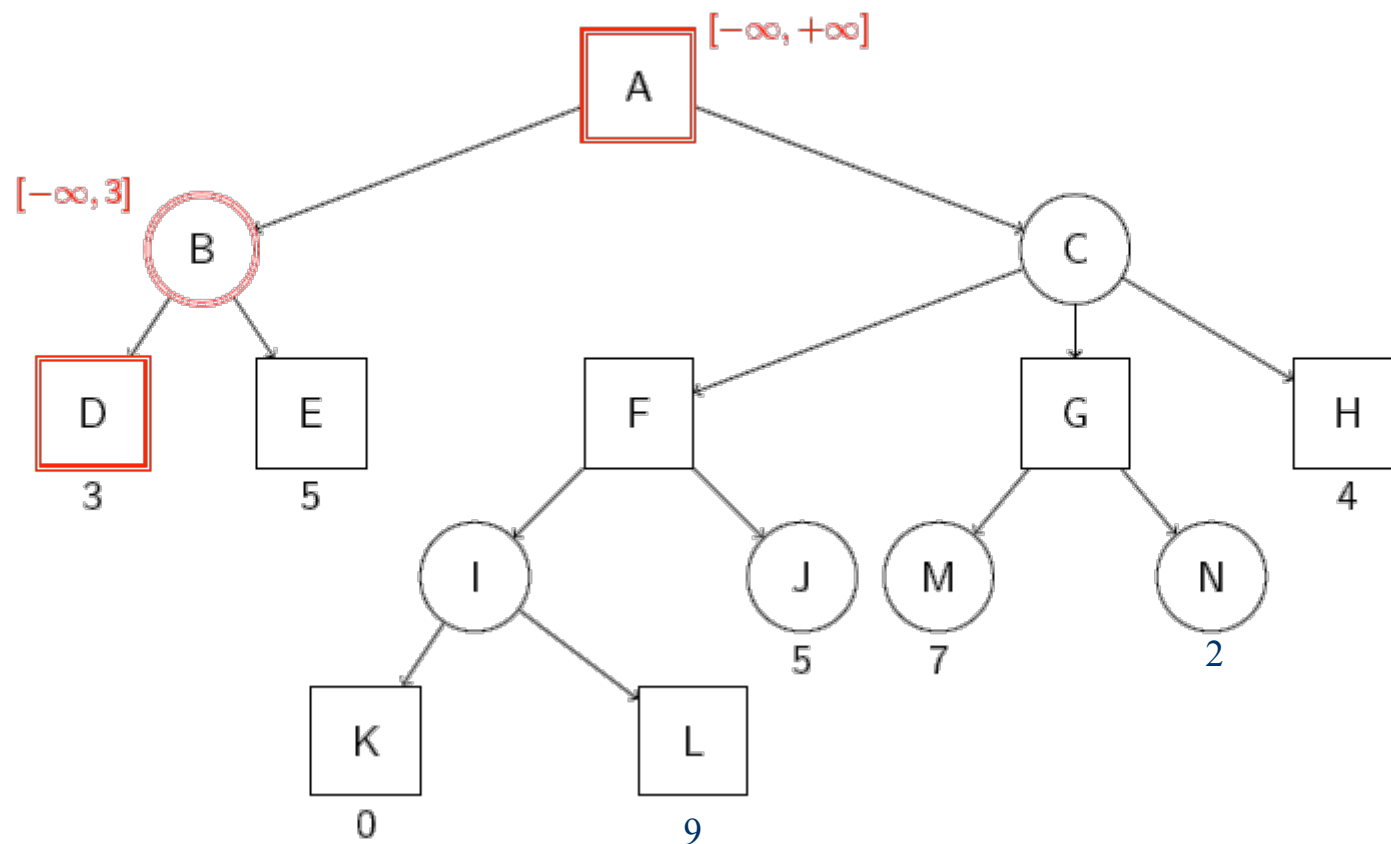
4.3 Ejemplo 4: Poda α - β



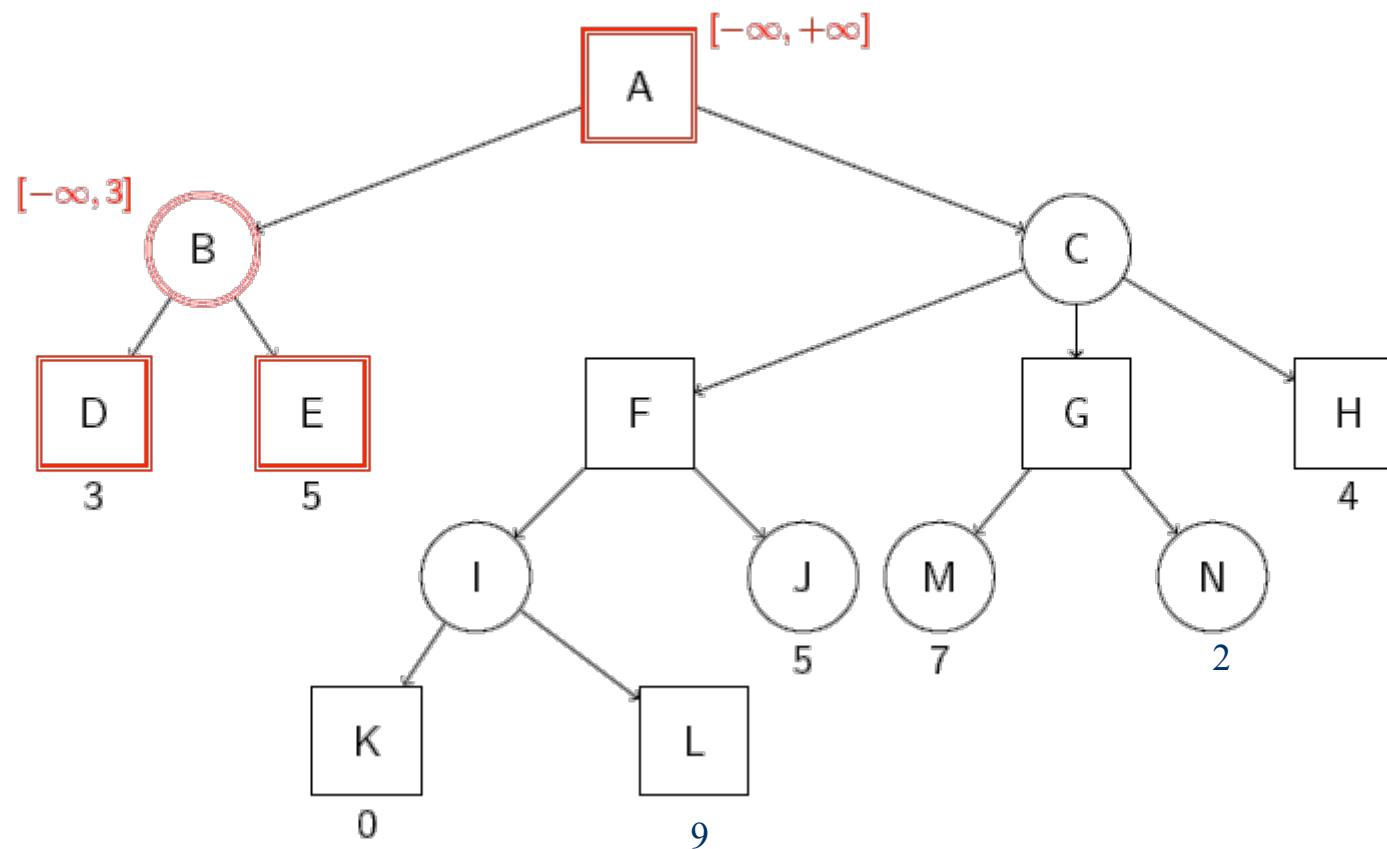
4.3 Ejemplo 4: Poda α - β



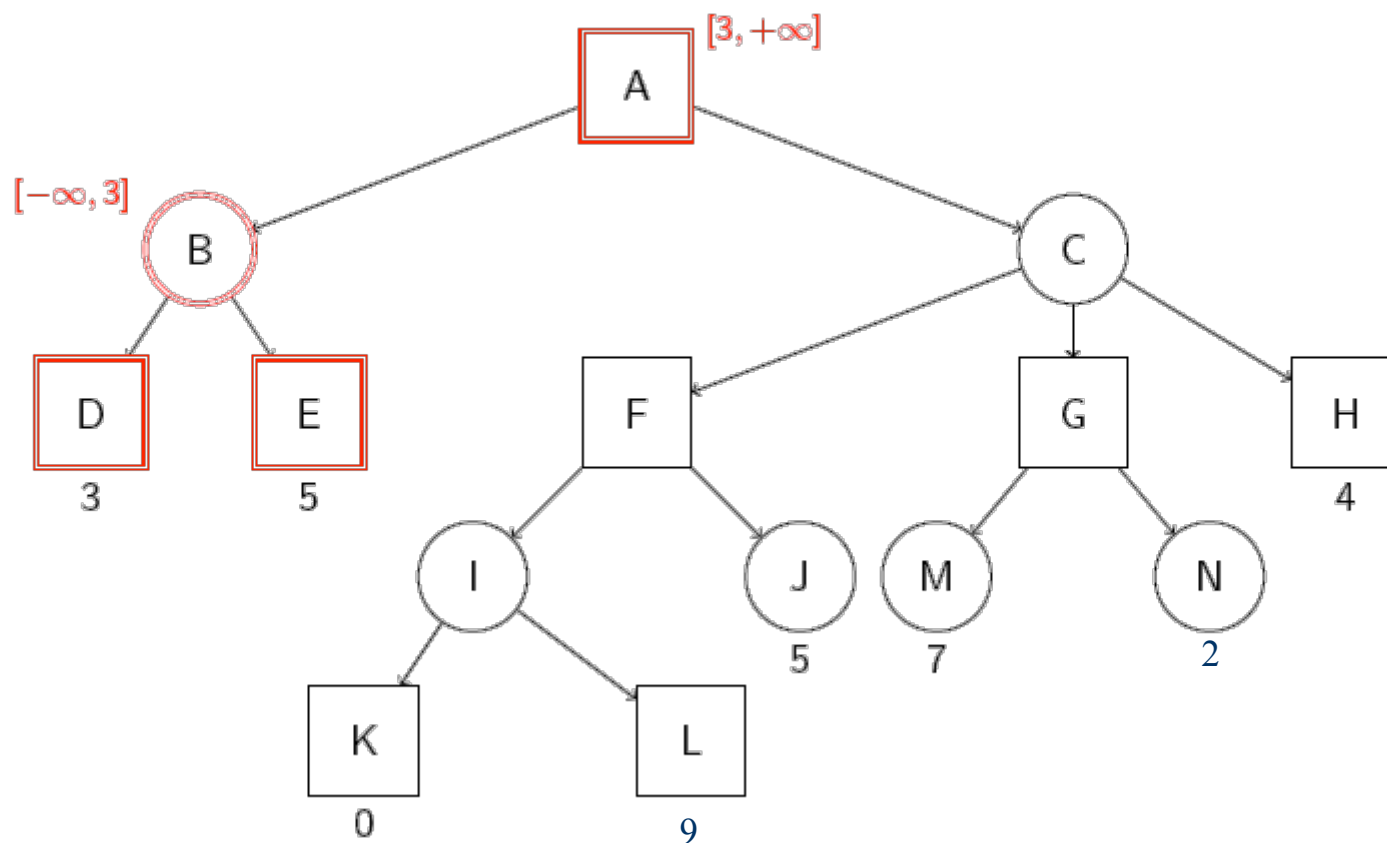
4.3 Ejemplo 4: Poda α - β



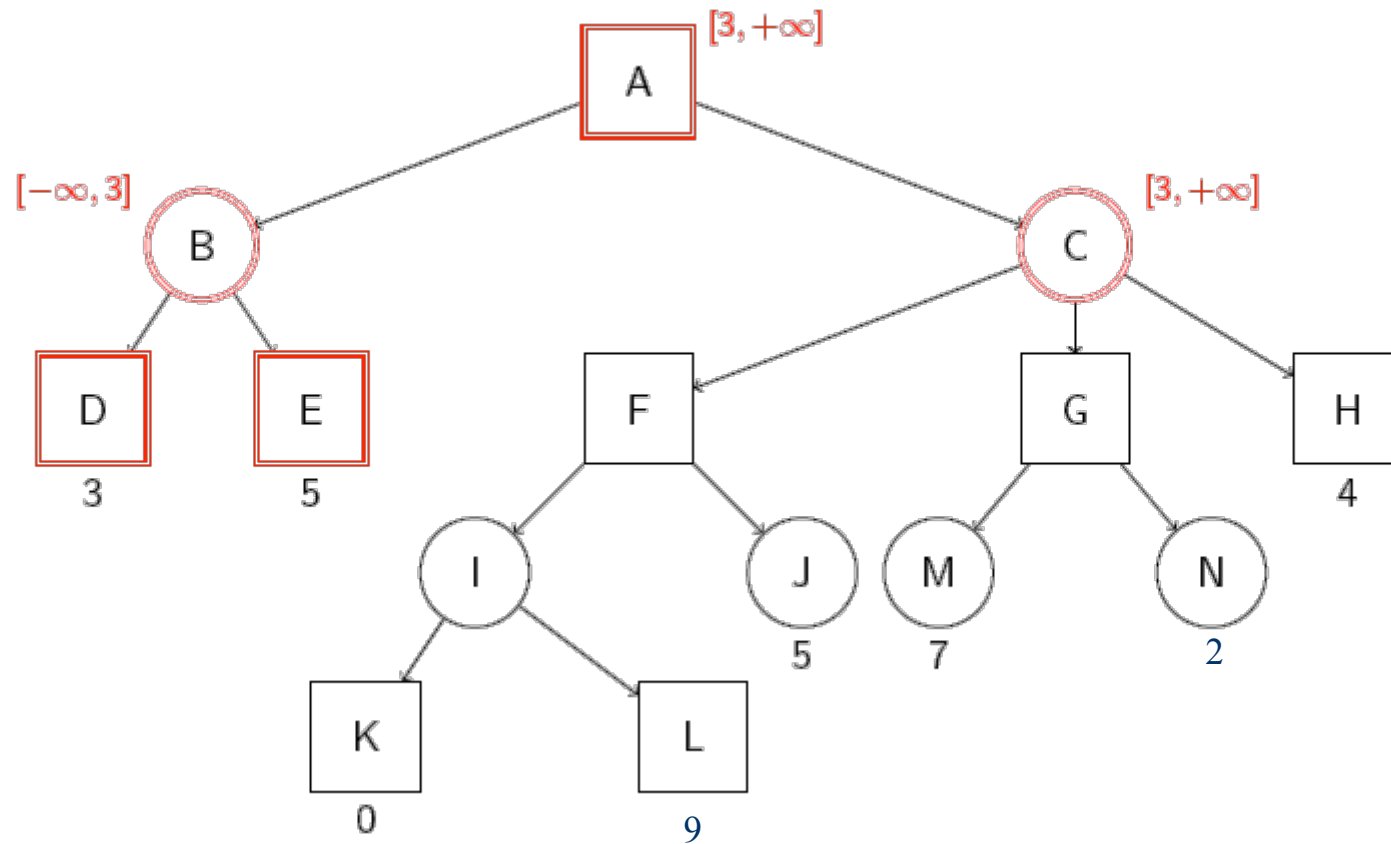
4.3 Ejemplo 4: Poda α - β



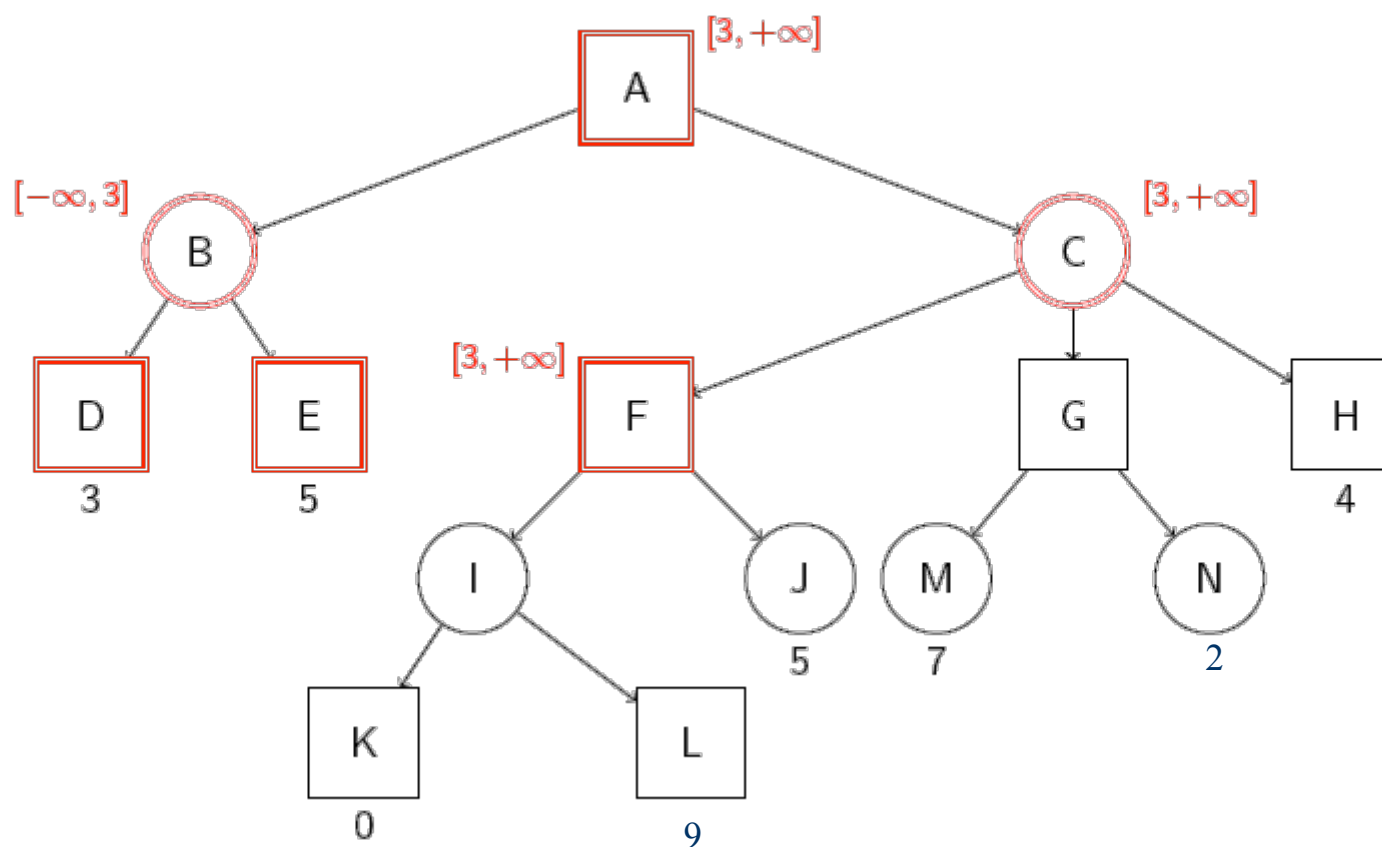
4.3 Ejemplo 4: Poda α - β



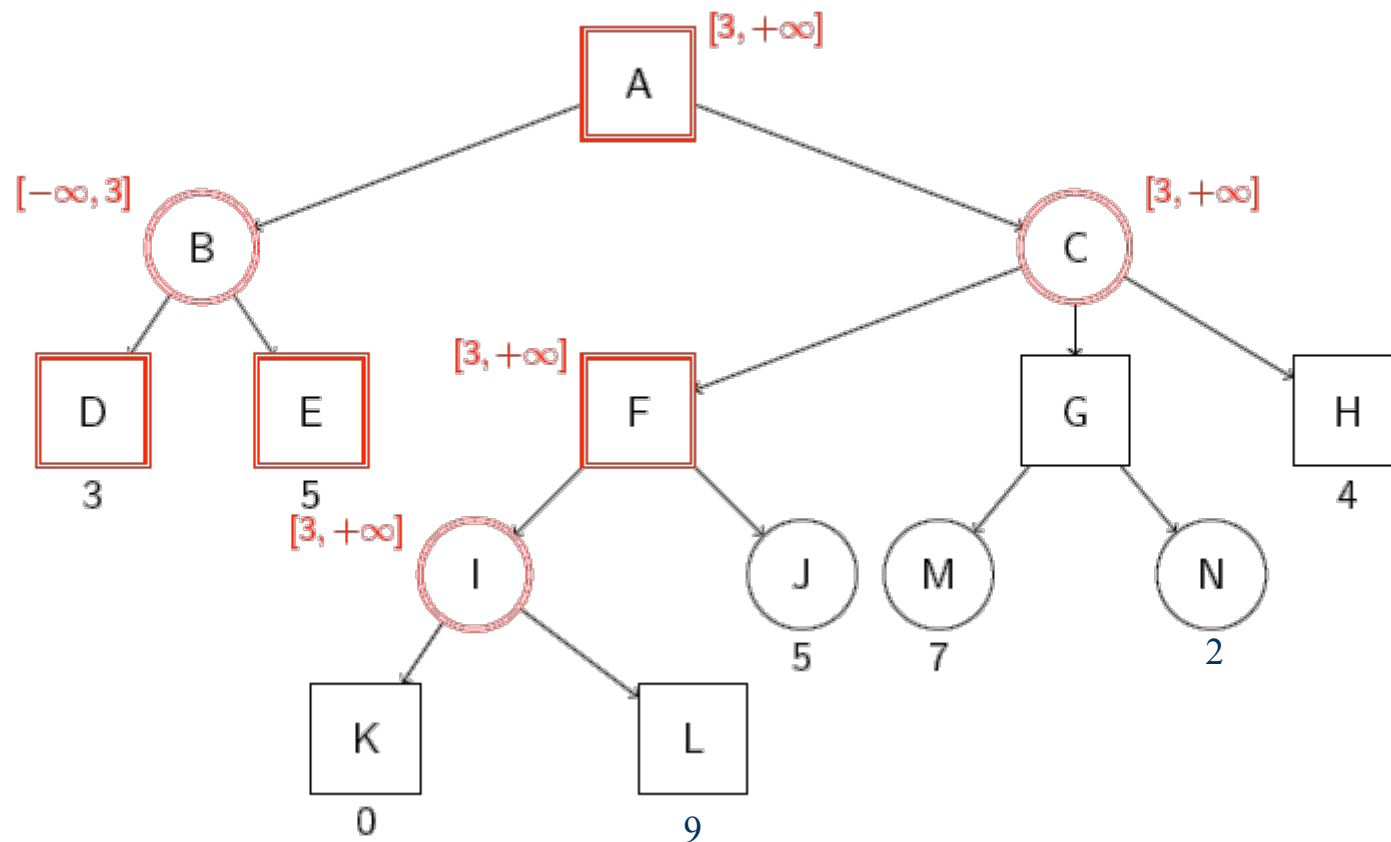
4.3 Ejemplo 4: Poda α - β



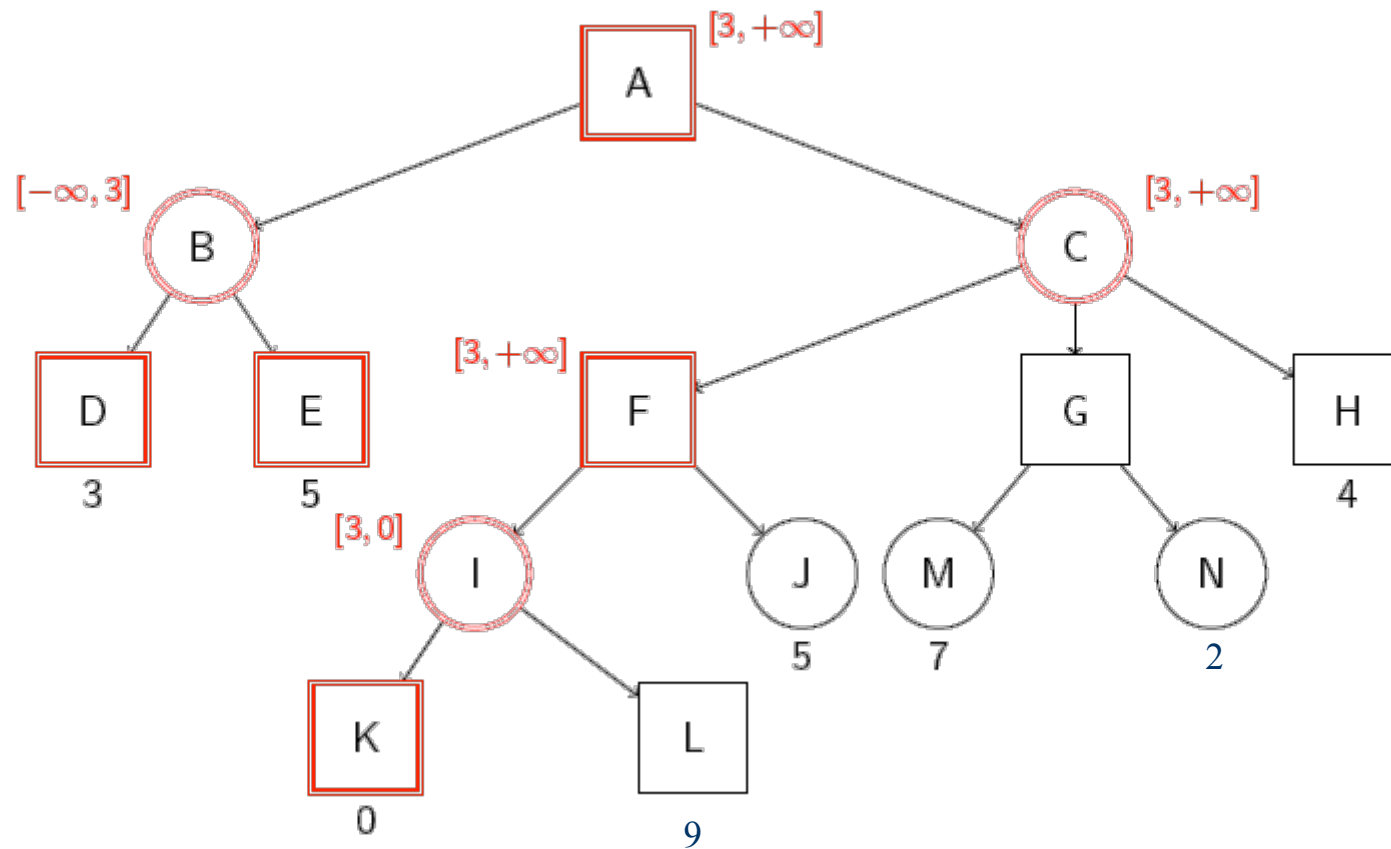
4.3 Ejemplo 4: Poda α - β



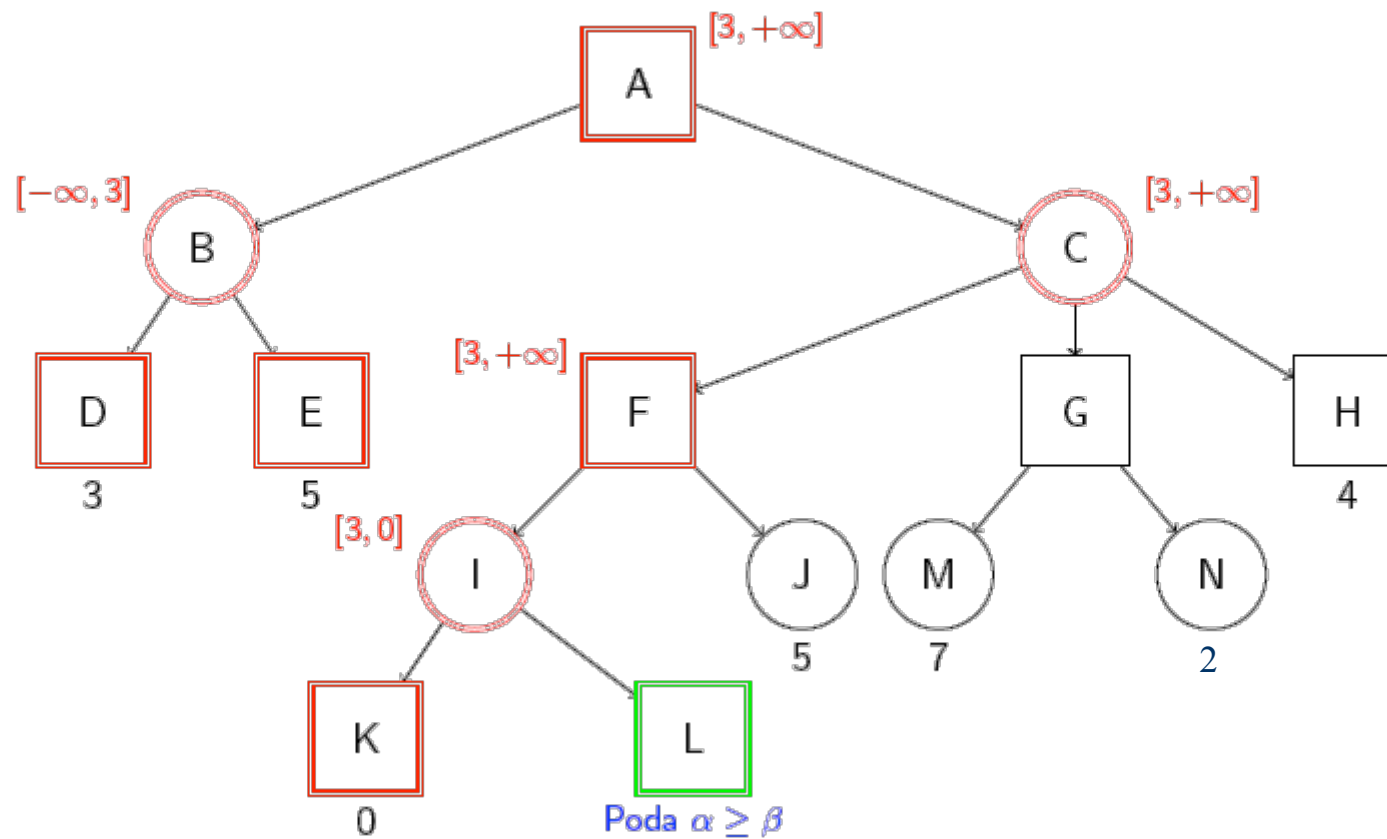
4.3 Ejemplo 4: Poda α - β



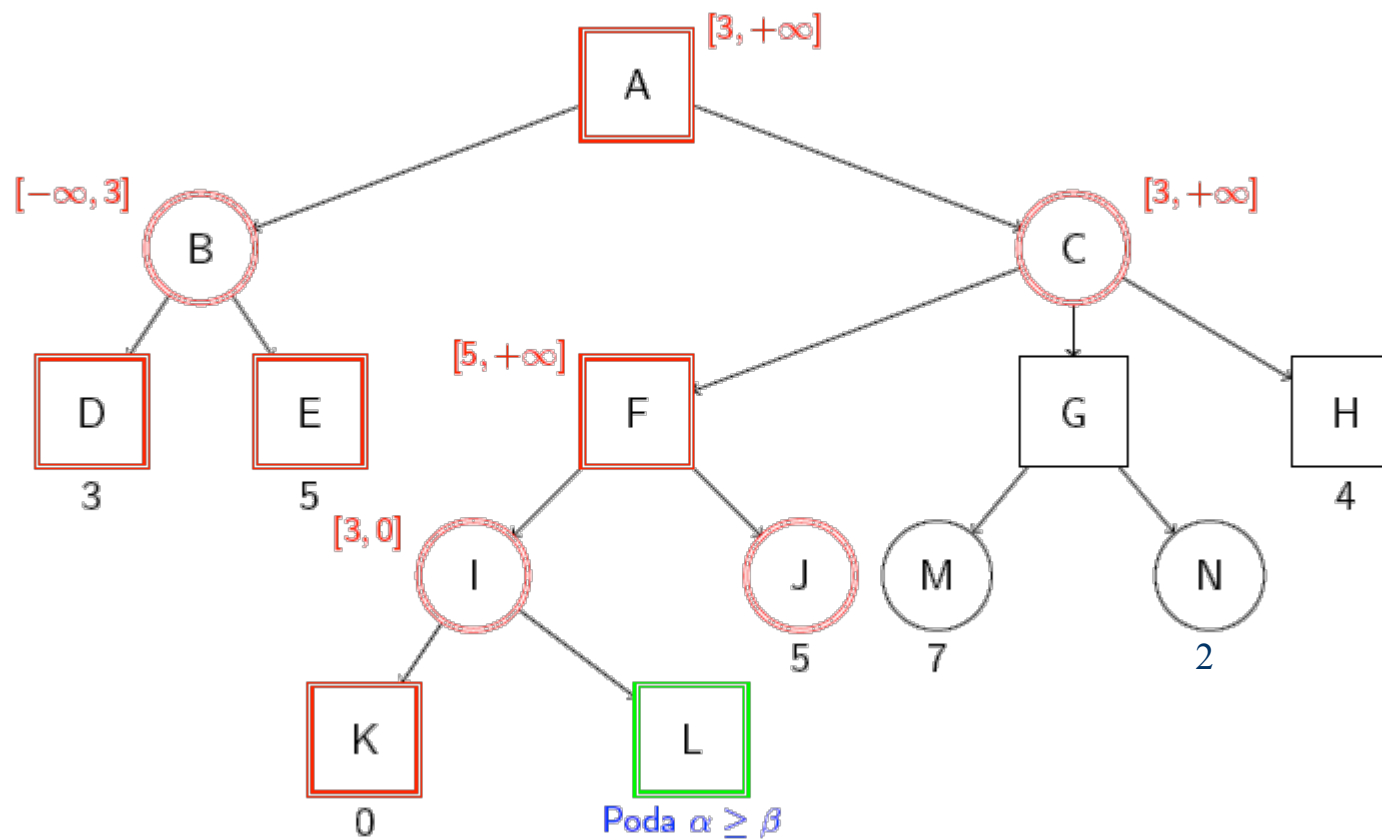
4.3 Ejemplo 4: Poda α - β



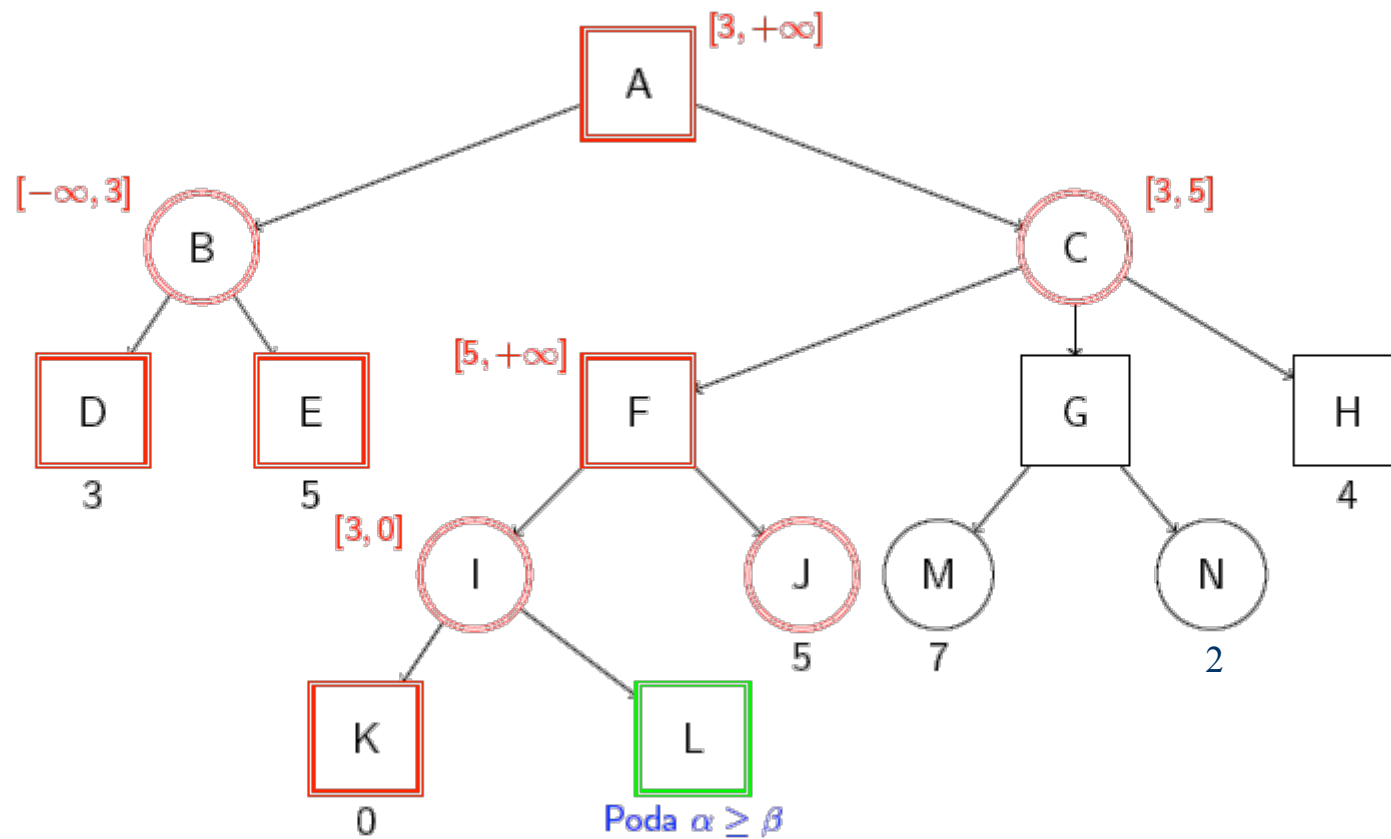
4.3 Ejemplo 4: Poda α - β



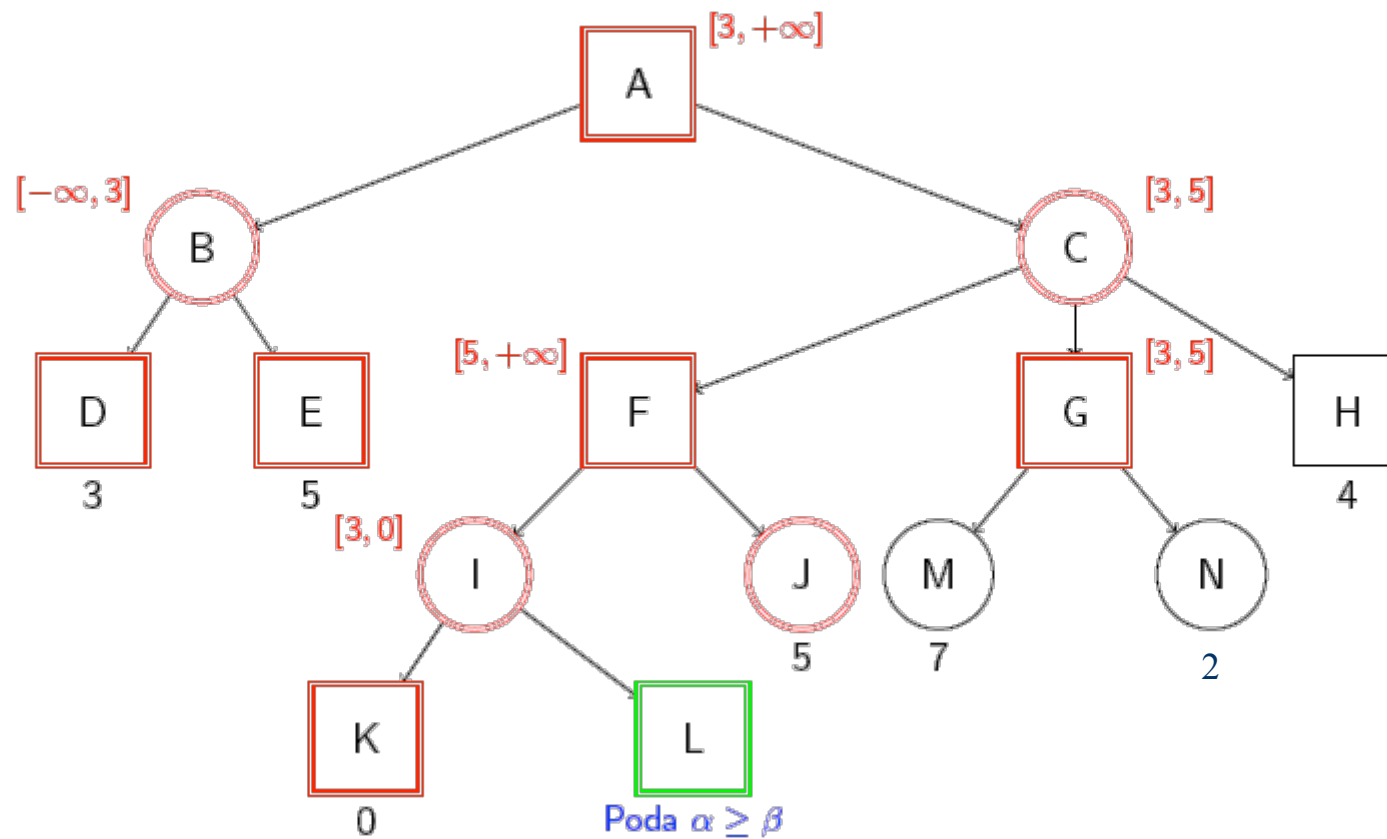
4.3 Ejemplo 4: Poda α - β



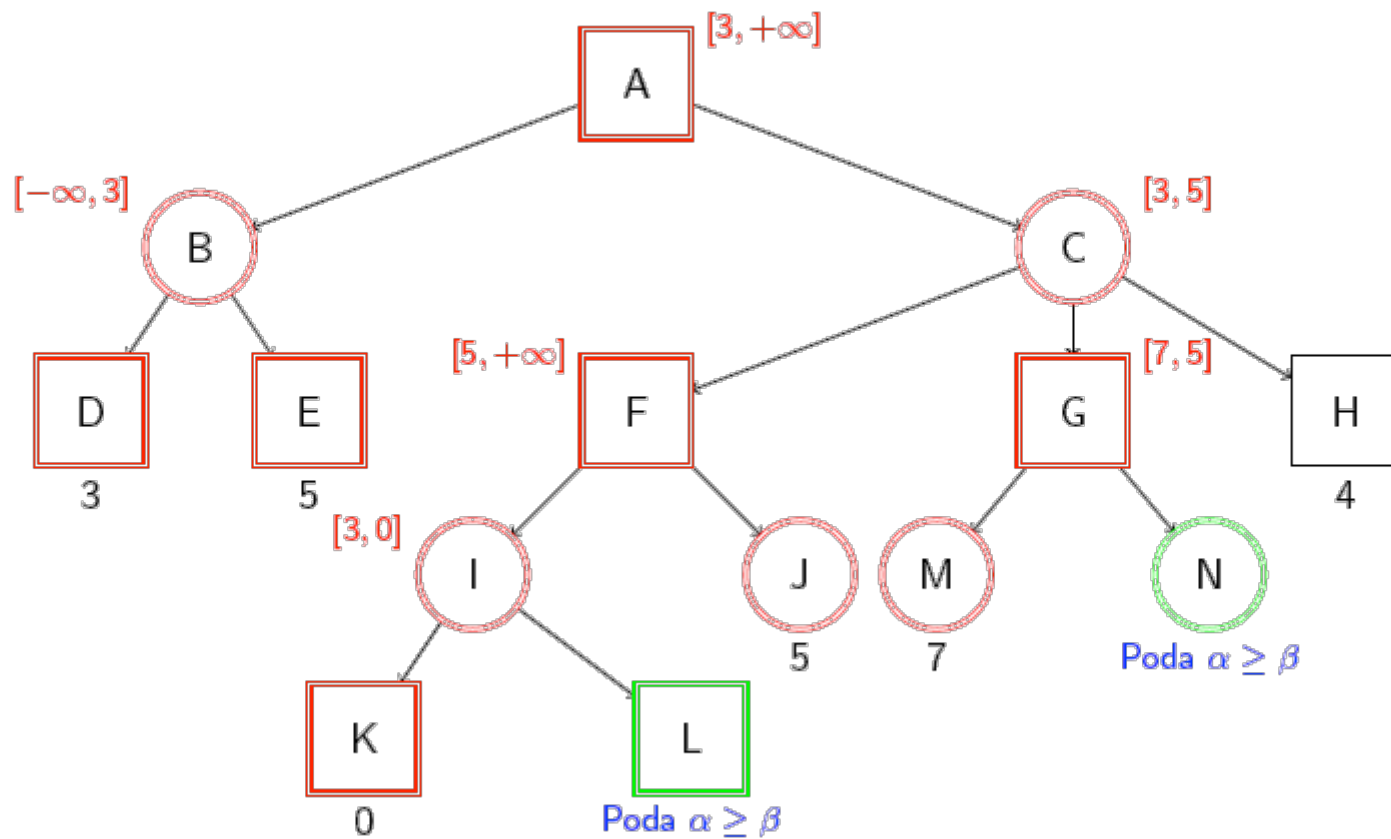
4.3 Ejemplo 4: Poda α - β



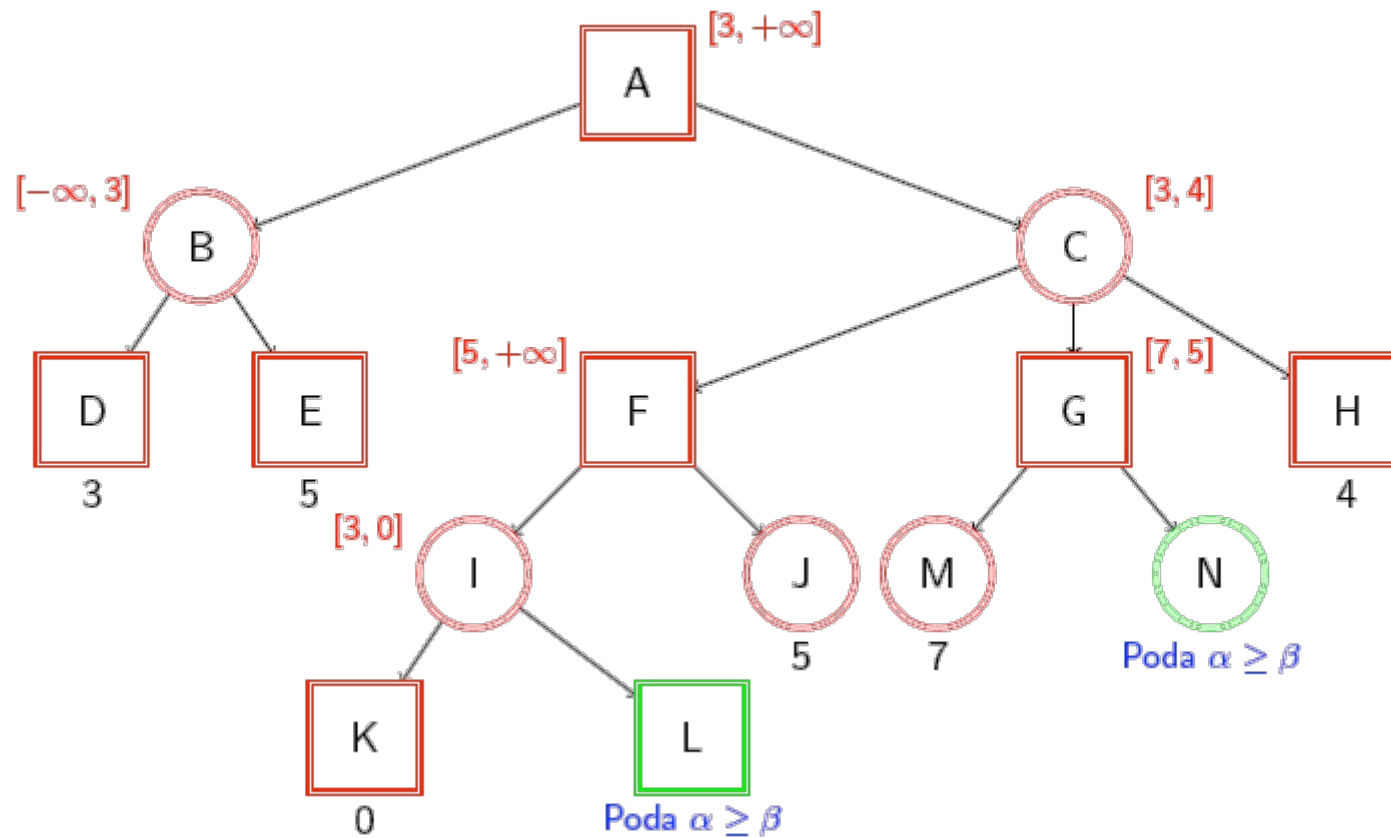
4.3 Ejemplo 4: Poda α - β



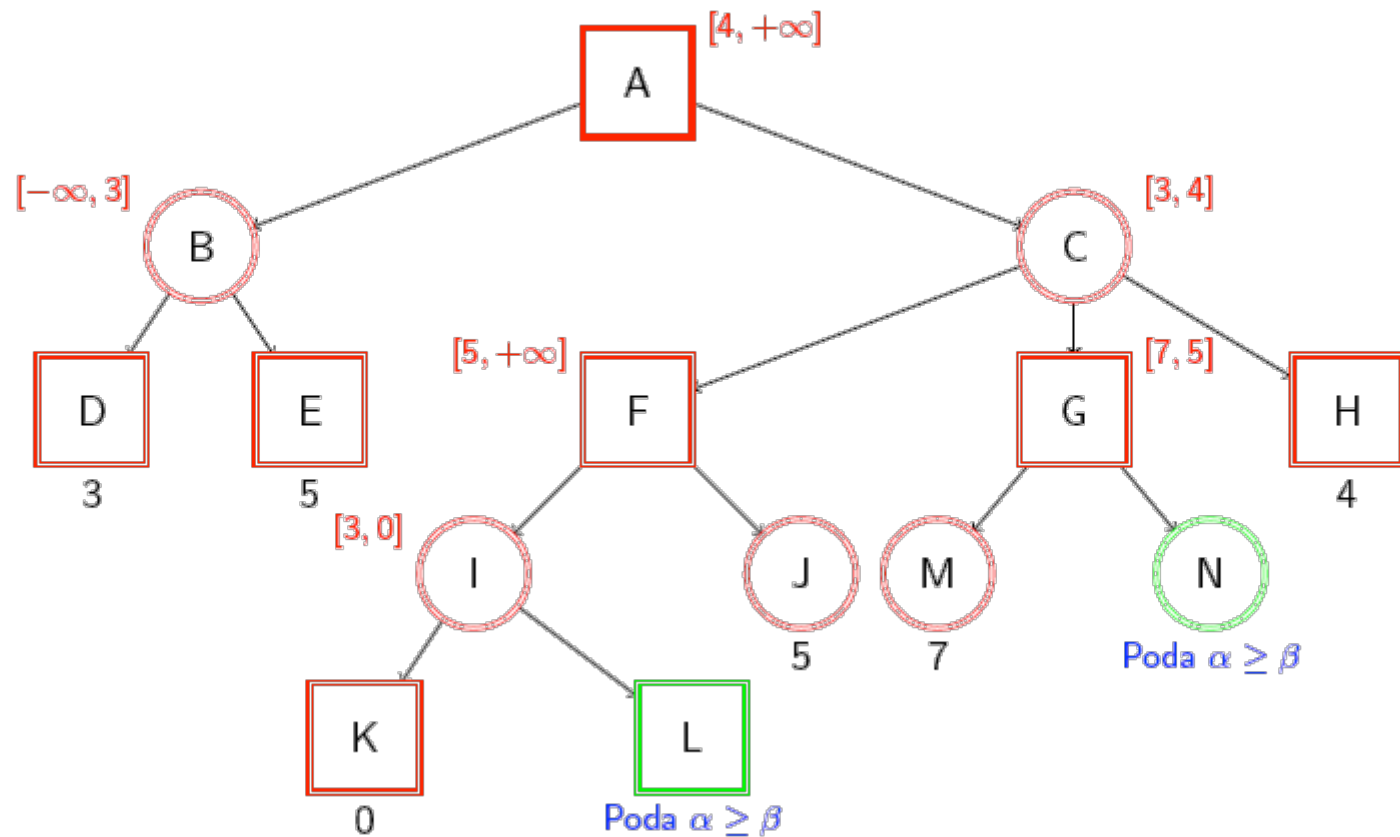
4.3 Ejemplo 4: Poda α - β



4.3 Ejemplo 4: Poda α - β



4.3 Ejemplo 4: Poda α - β



5.1 Análisis de la Poda

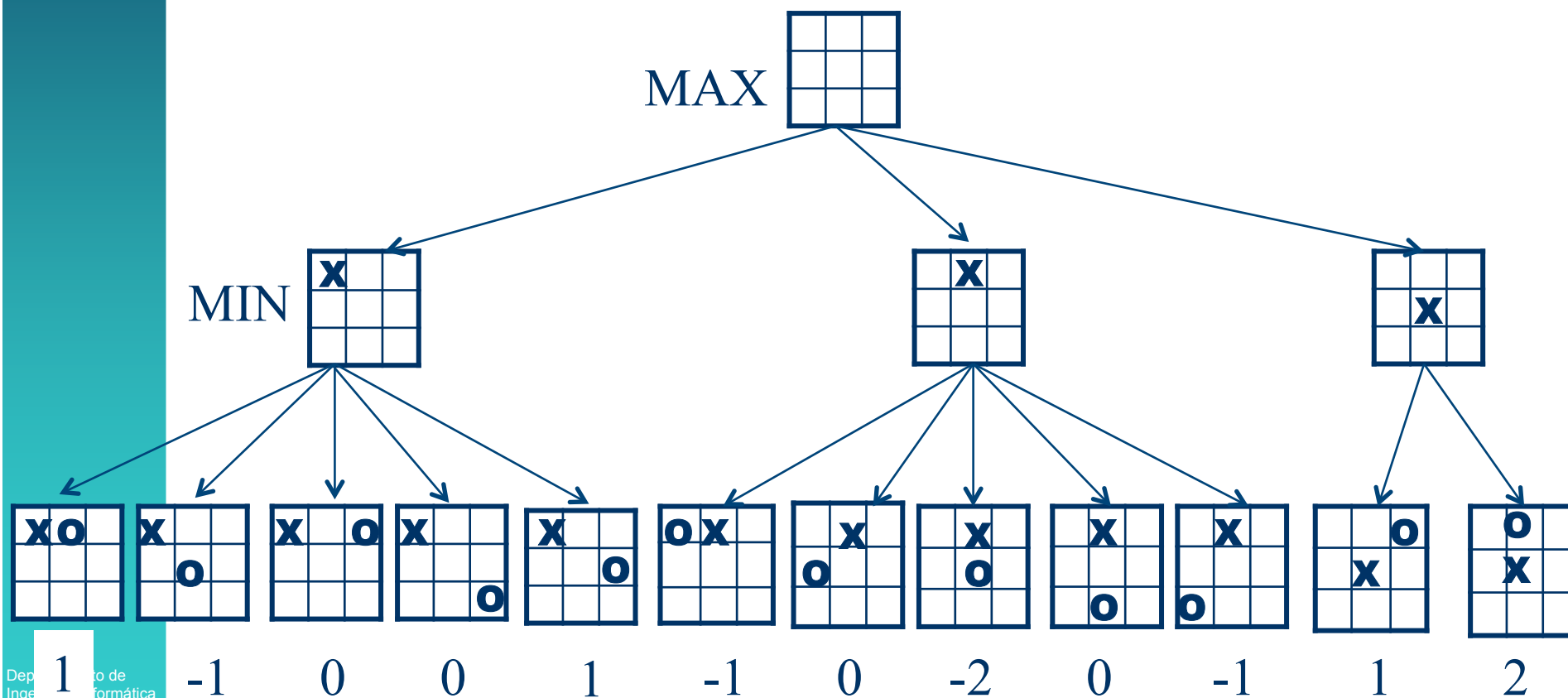
- En el peor de los casos funciona como MiniMax
- La expansión del árbol depende del orden de generación de los nodos
- Problemas con la profundidad fija del árbol explorado:
 - **Efecto Horizonte:** se evalúa como buena o mala una situación sin saber que a la siguiente jugada la situación revierte
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

5.2 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**

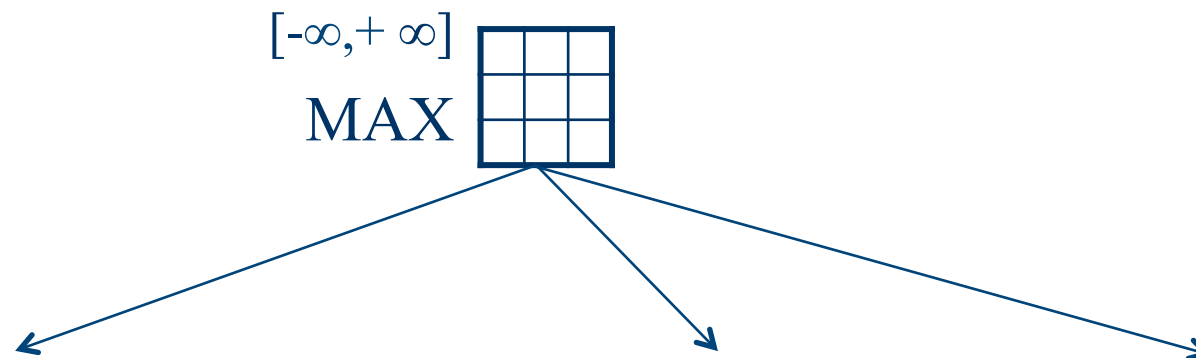
Ejercicio

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



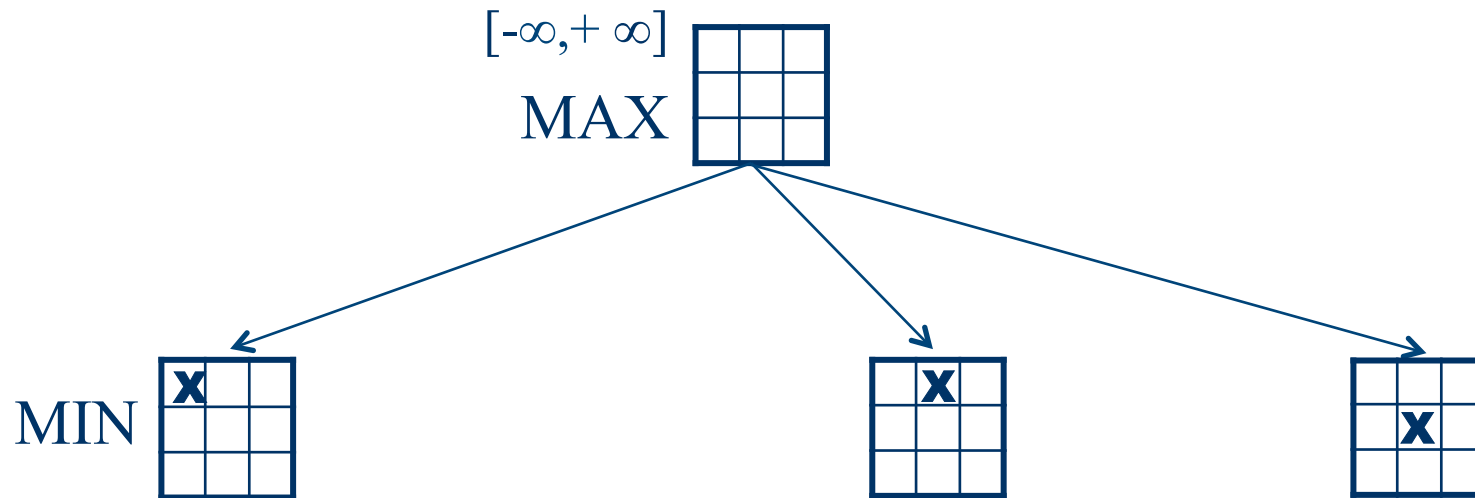
Ejercicio

- Poda alfa-beta de izquierda a derecha:



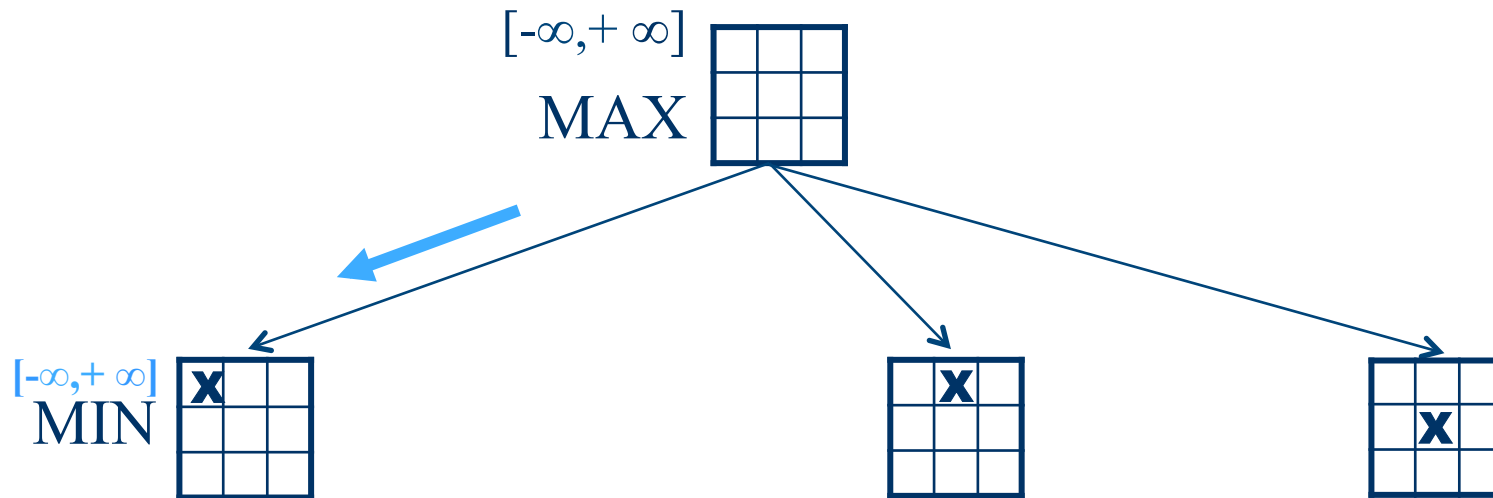
Ejercicio

- Poda alfa-beta de izquierda a derecha:



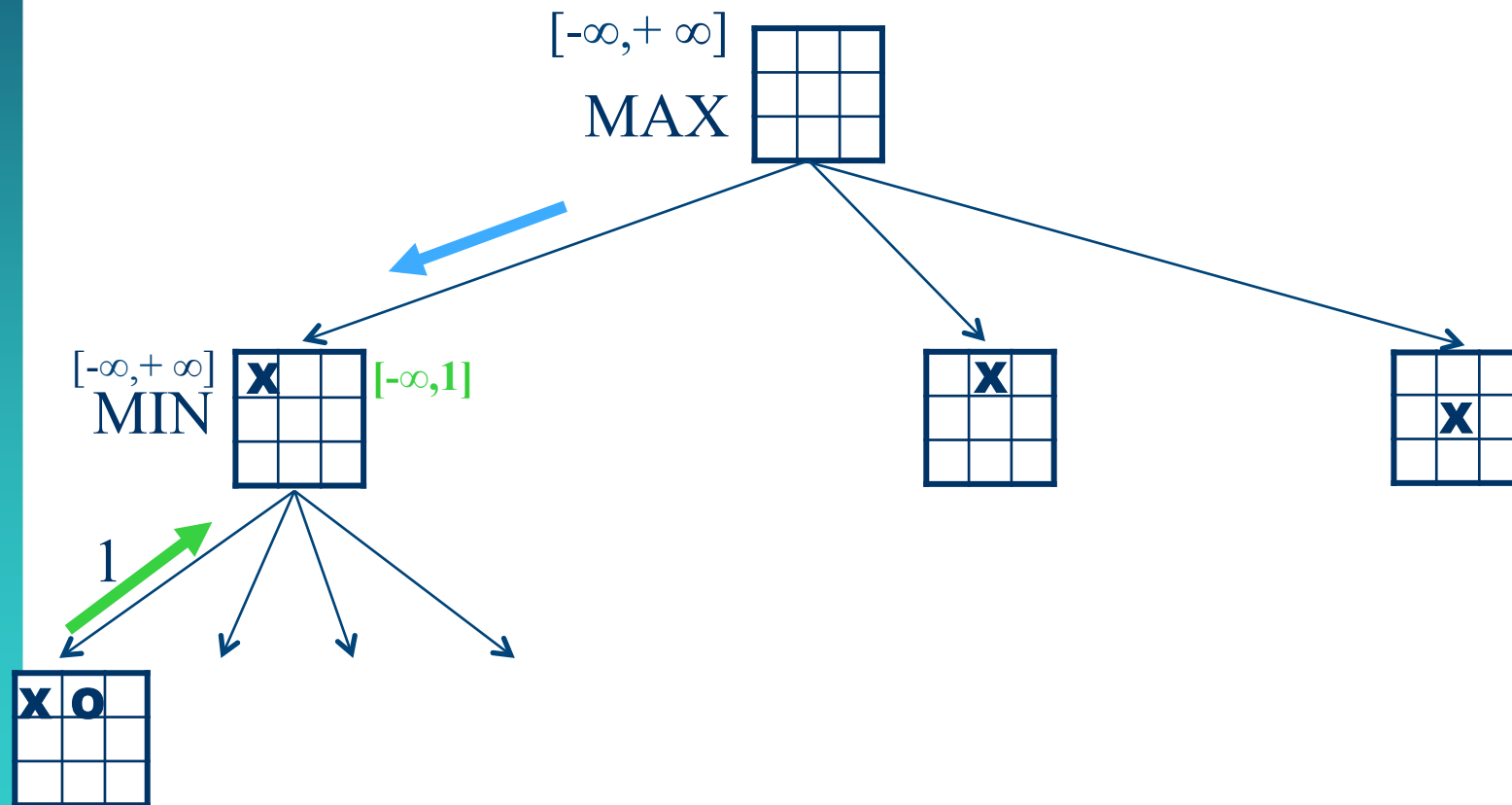
Ejercicio

- Poda alfa-beta de izquierda a derecha:



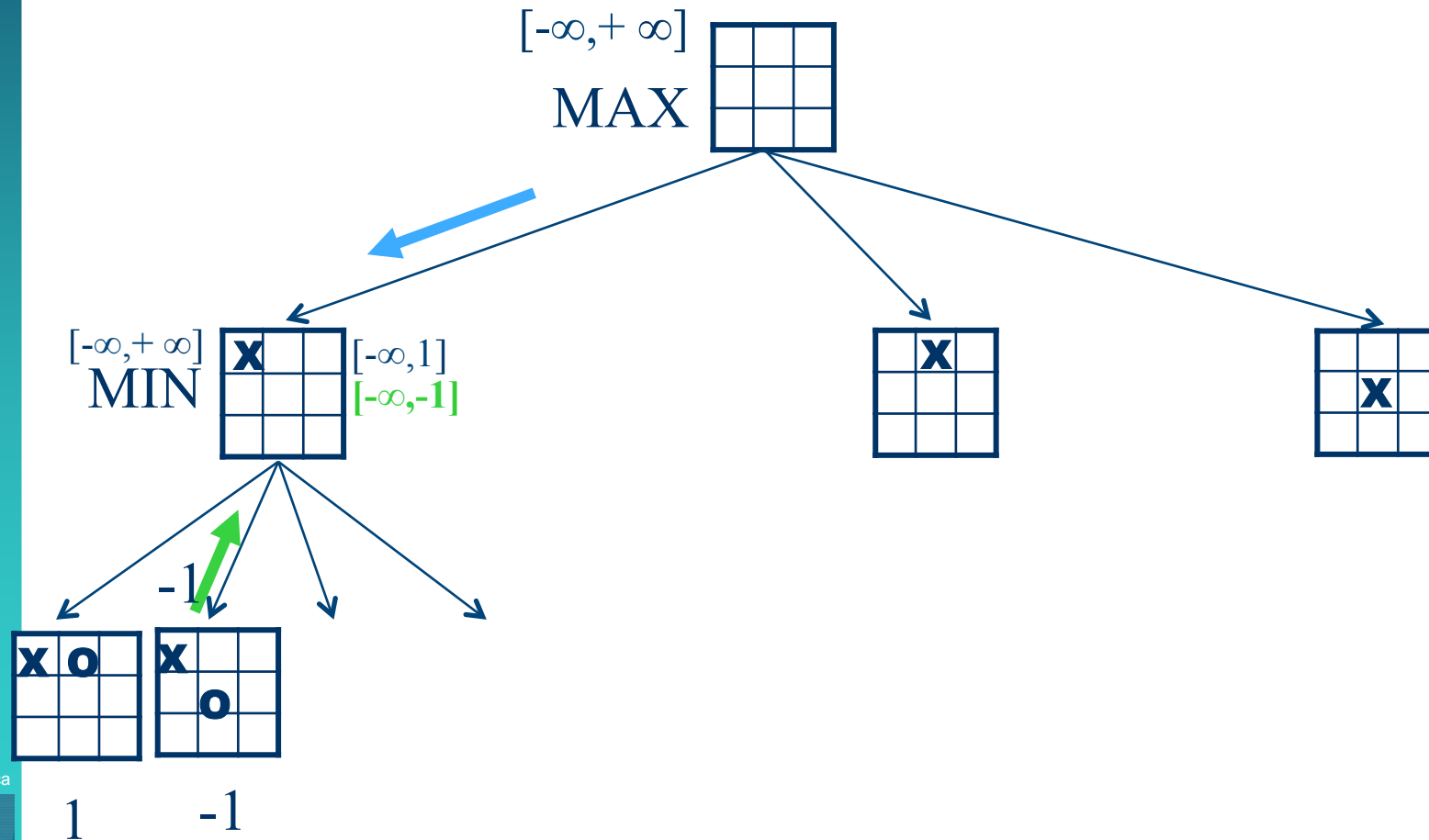
Ejercicio

- Poda alfa-beta de izquierda a derecha:



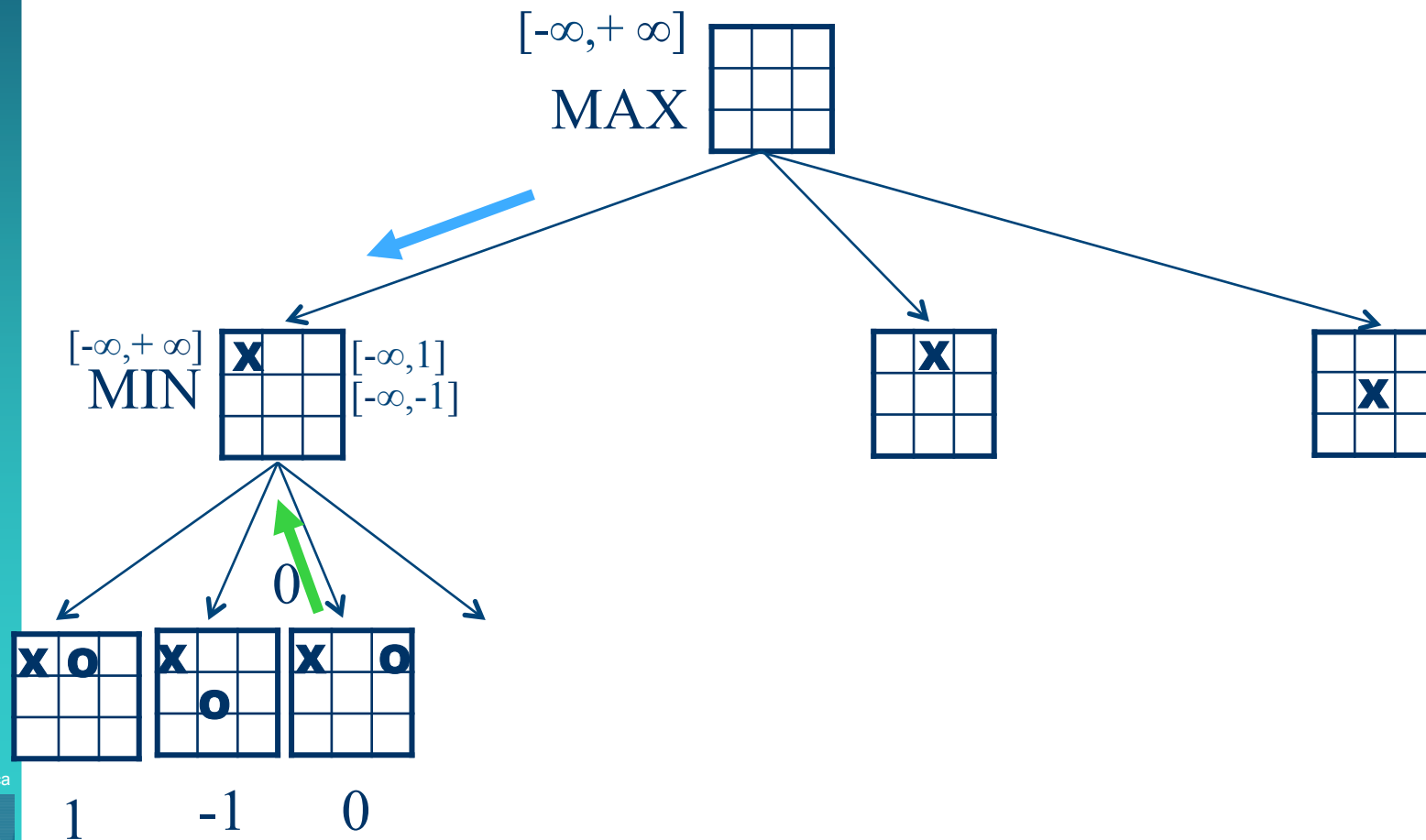
Ejercicio

- Poda alfa-beta de izquierda a derecha:



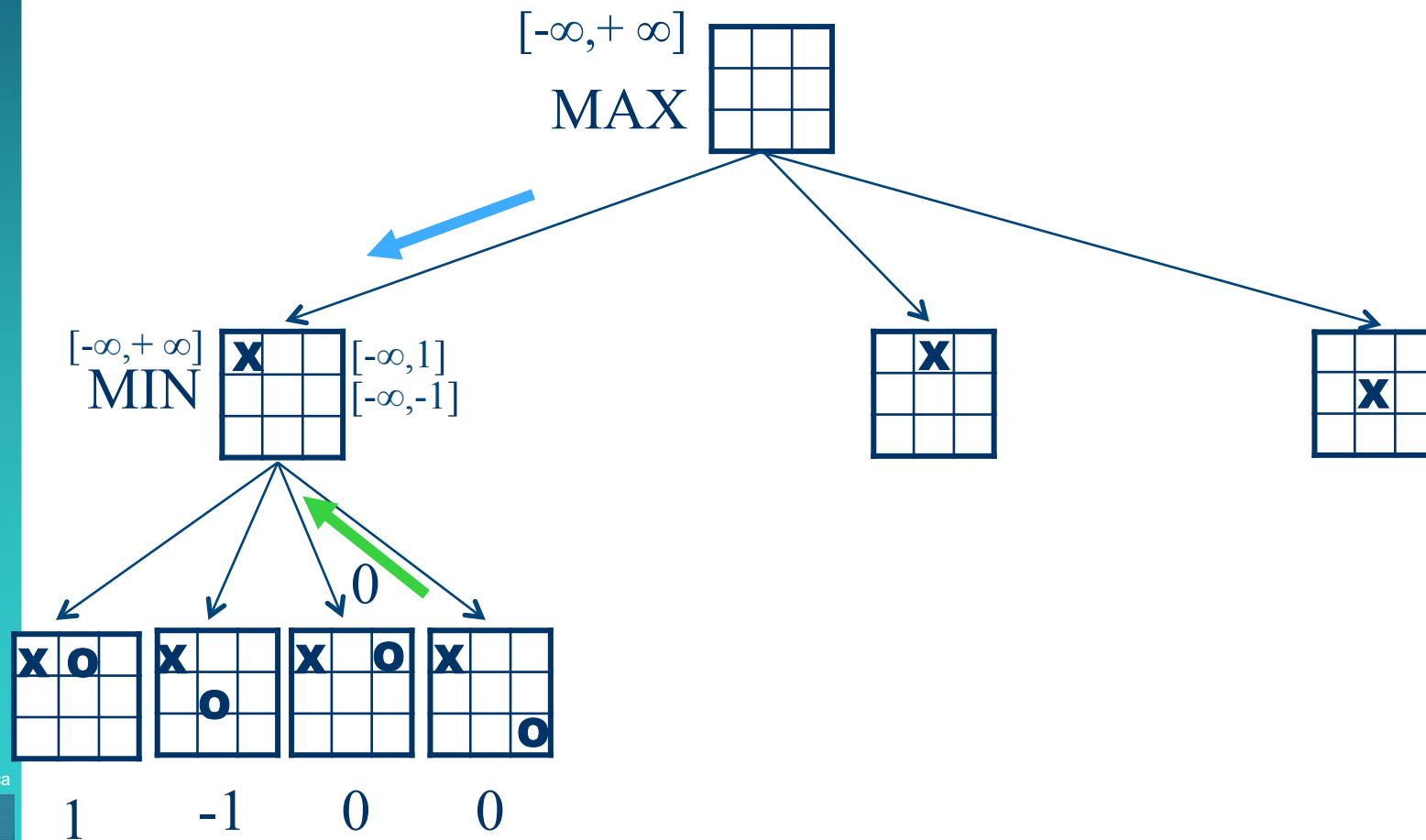
Ejercicio

- Poda alfa-beta de izquierda a derecha:



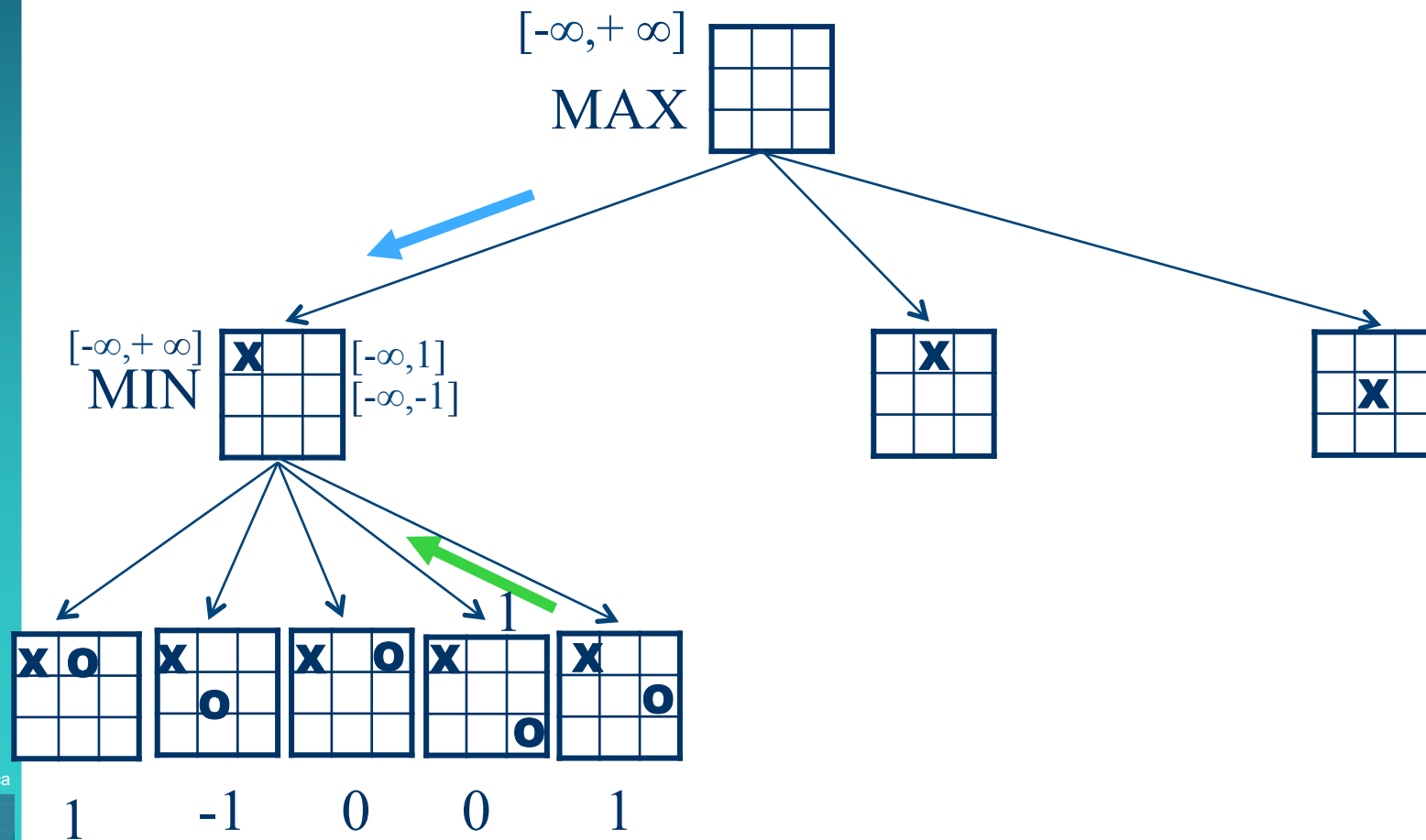
Ejercicio

- Poda alfa-beta de izquierda a derecha:



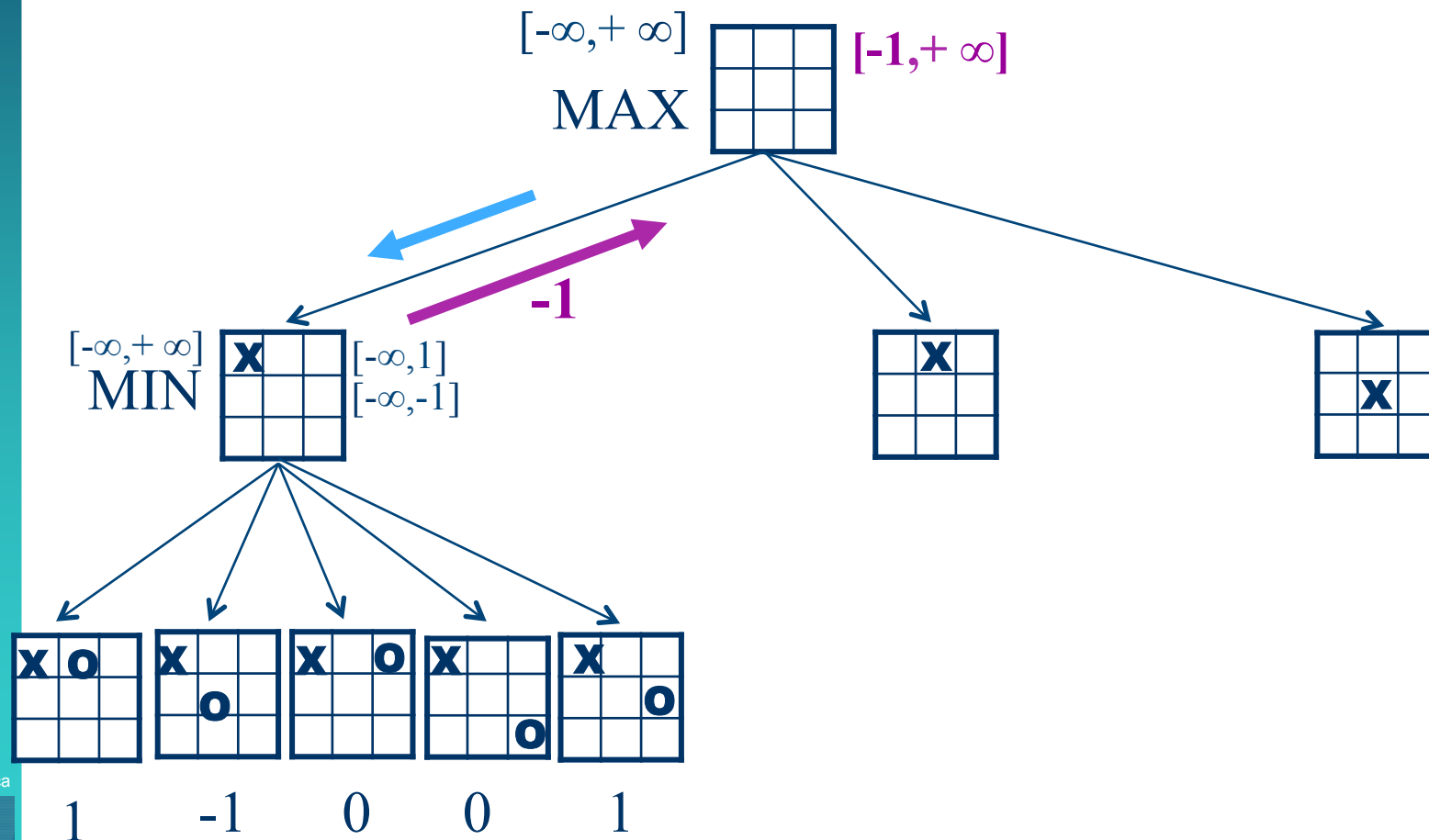
Ejercicio

- Poda alfa-beta de izquierda a derecha:



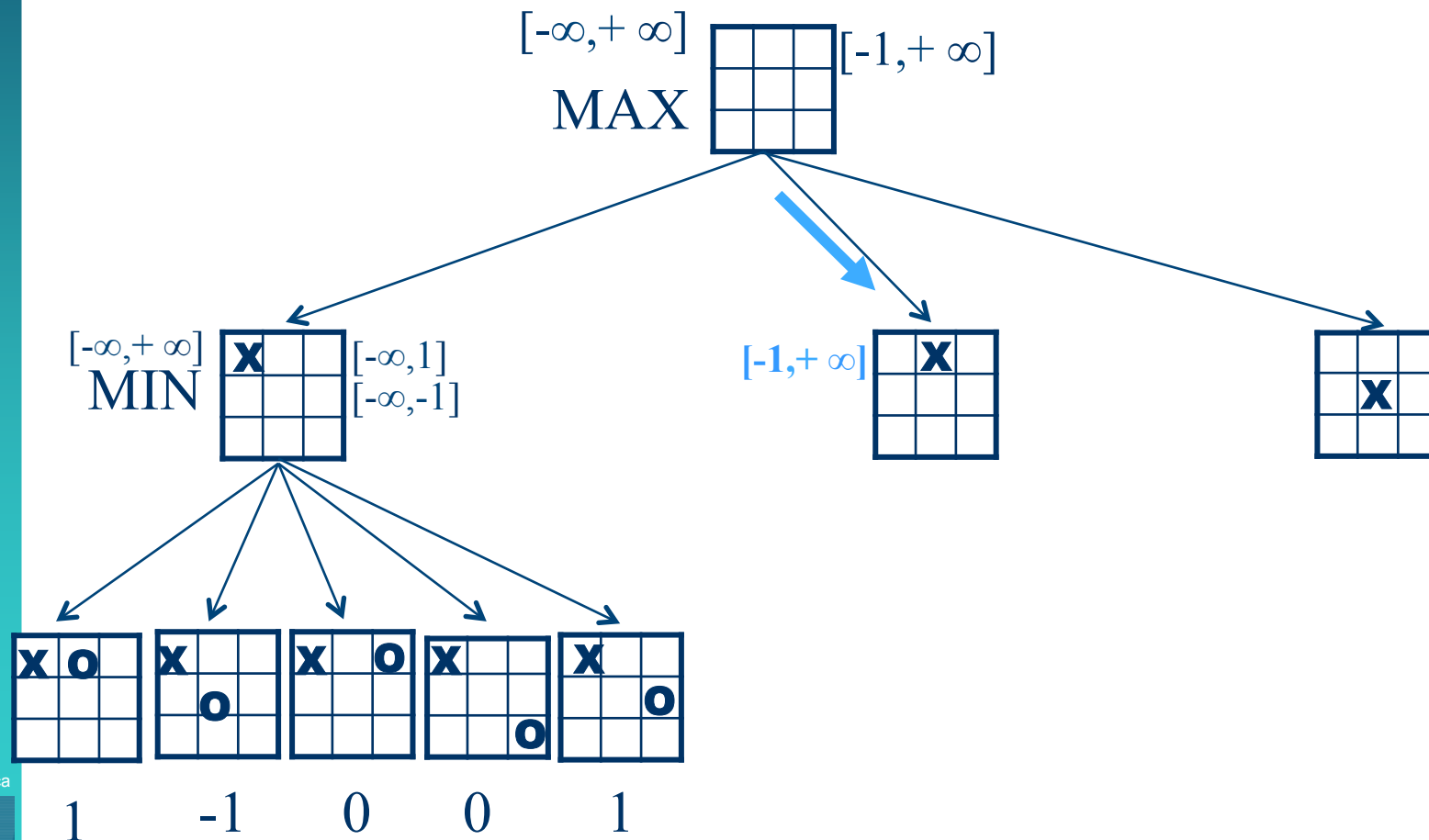
Ejercicio

- Poda alfa-beta de izquierda a derecha:



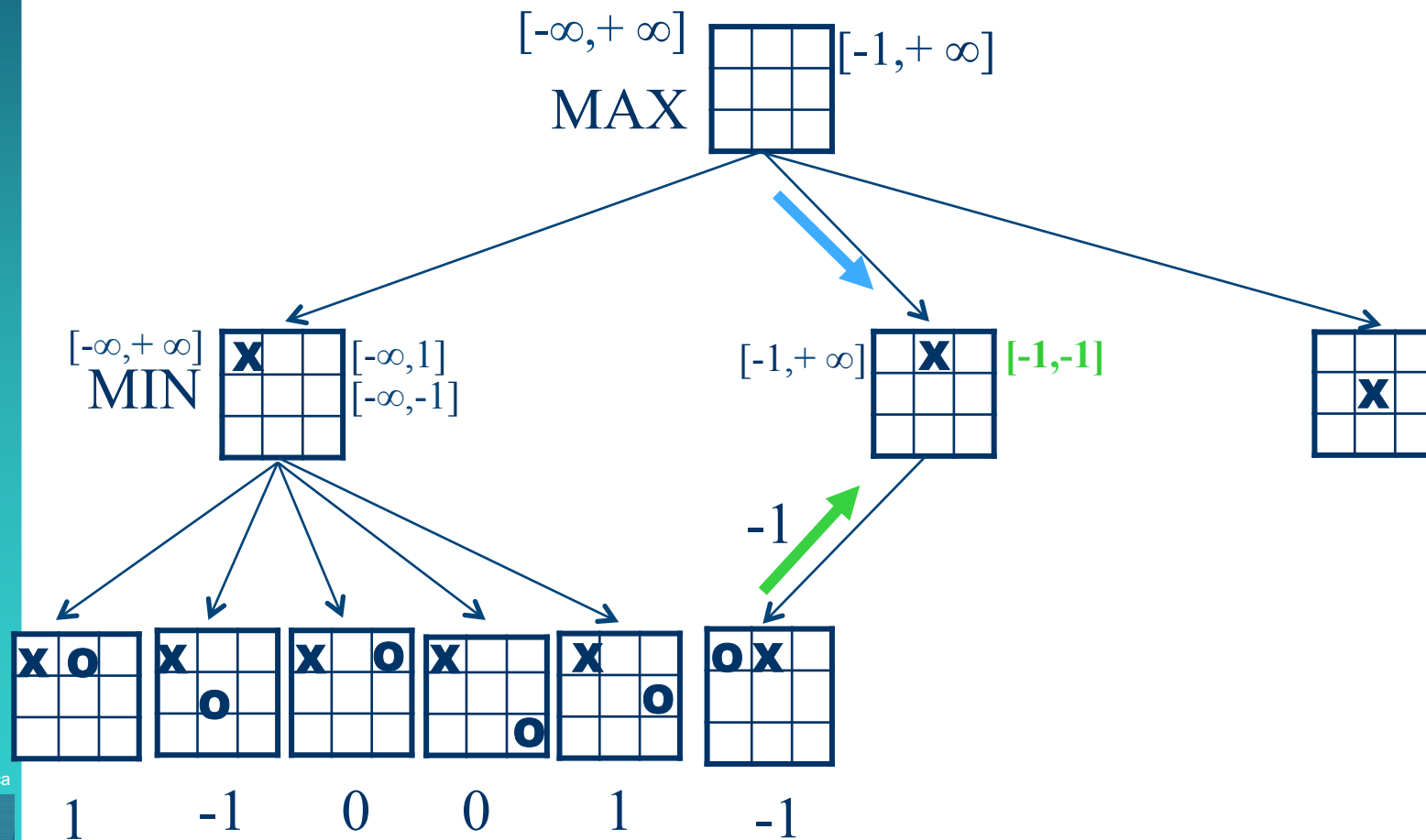
Ejercicio

- Poda alfa-beta de izquierda a derecha:



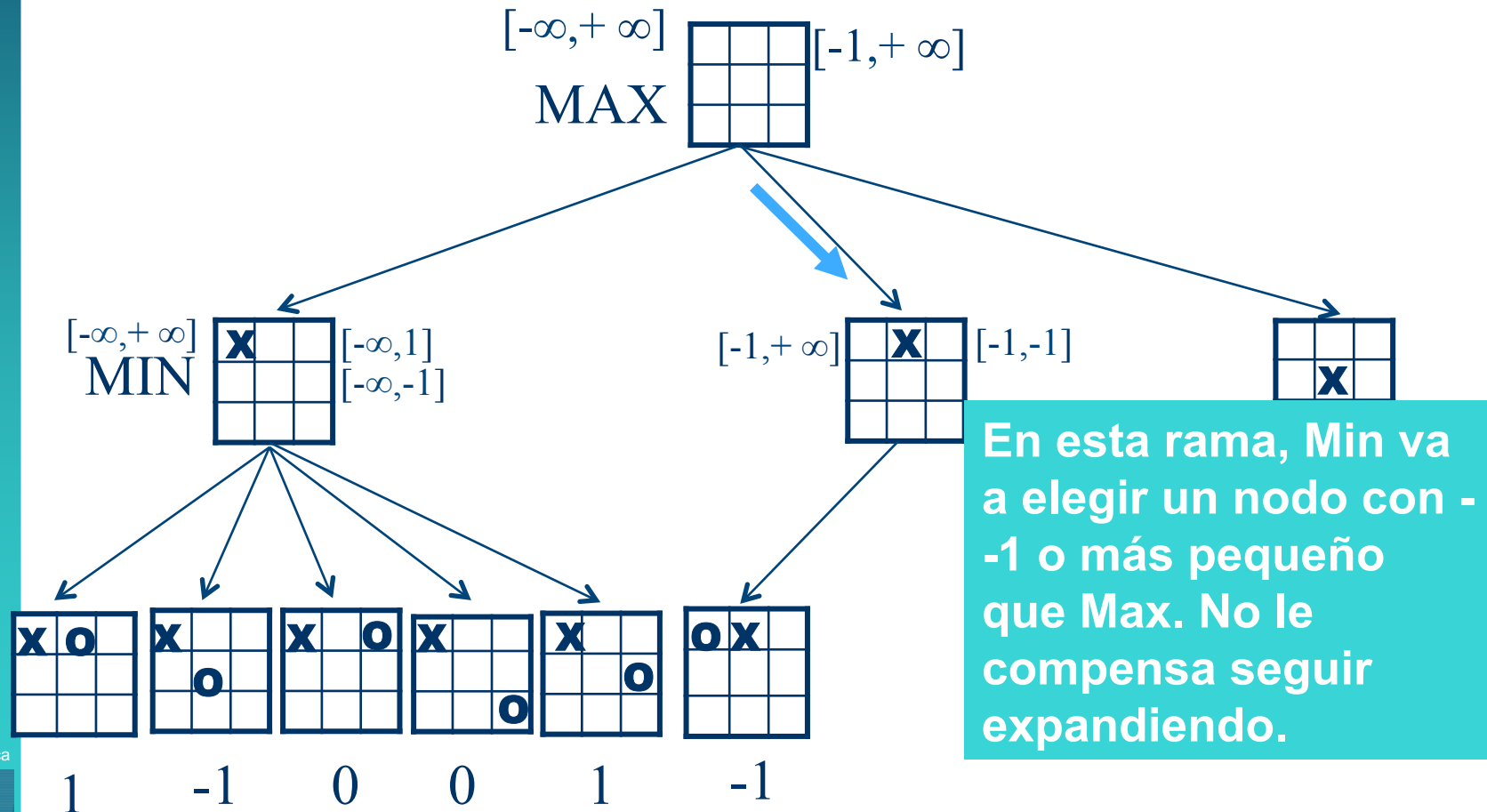
Ejercicio

- Poda alfa-beta de izquierda a derecha:



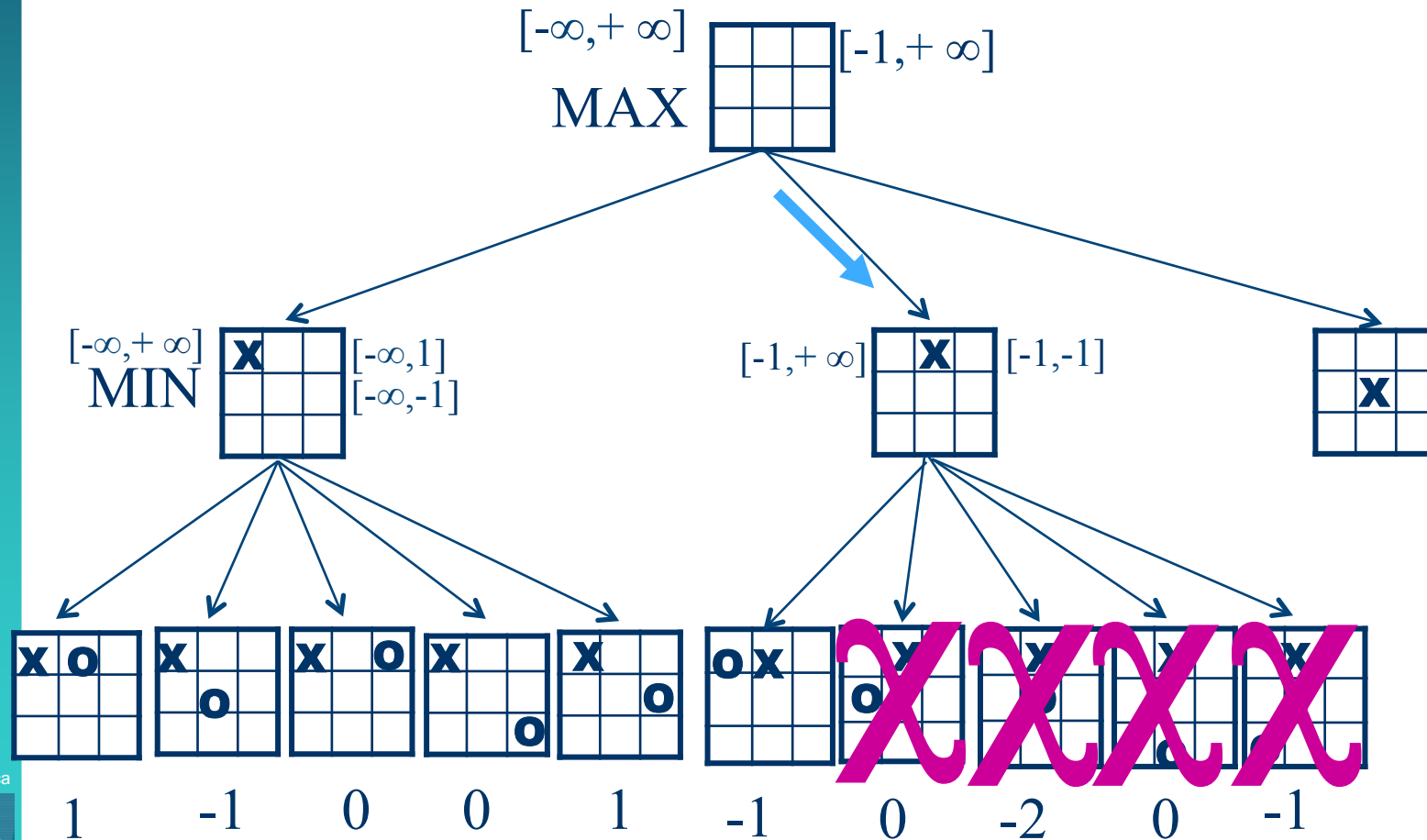
Ejercicio

- Poda alfa-beta de izquierda a derecha:



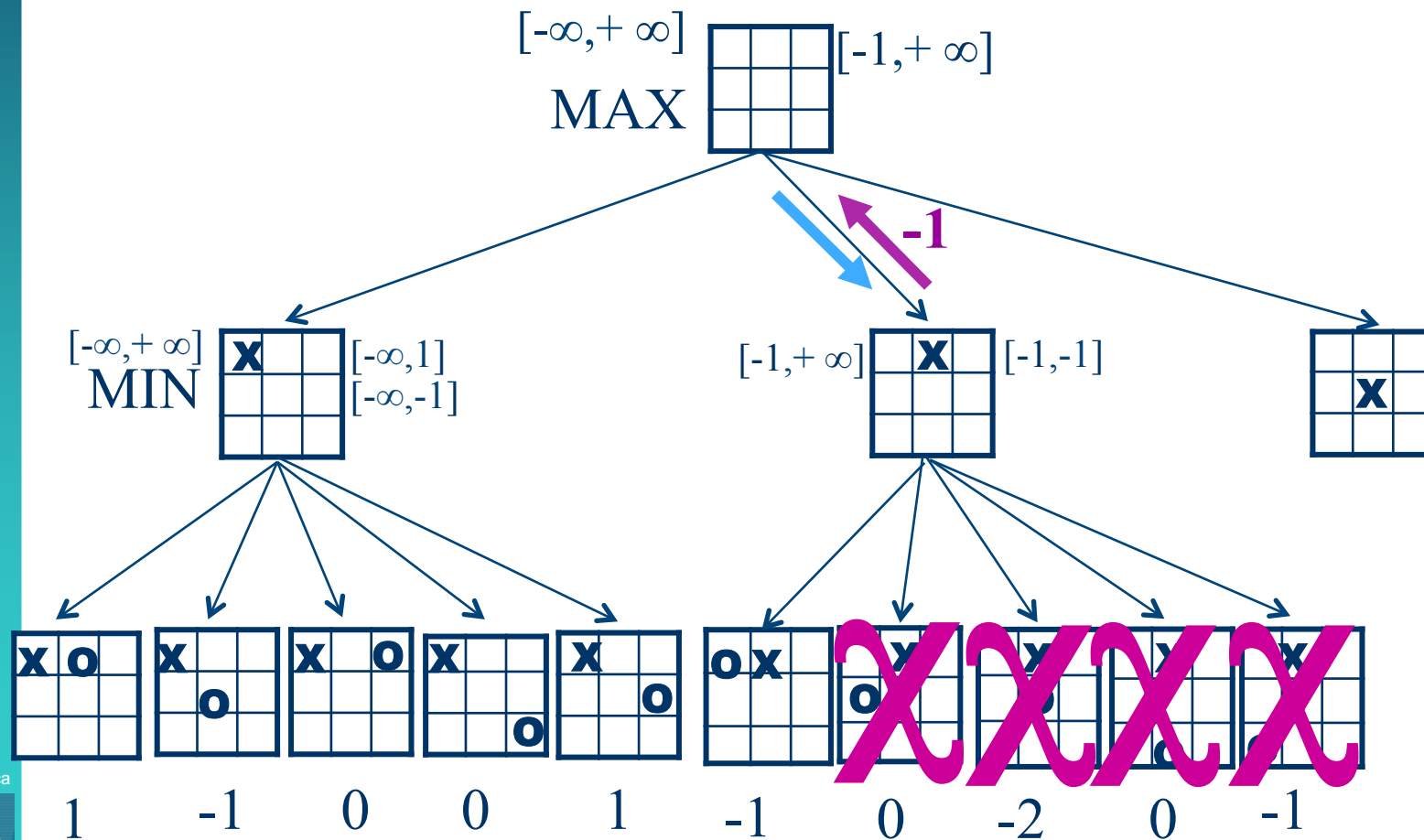
Ejercicio

- Poda alfa-beta de izquierda a derecha:



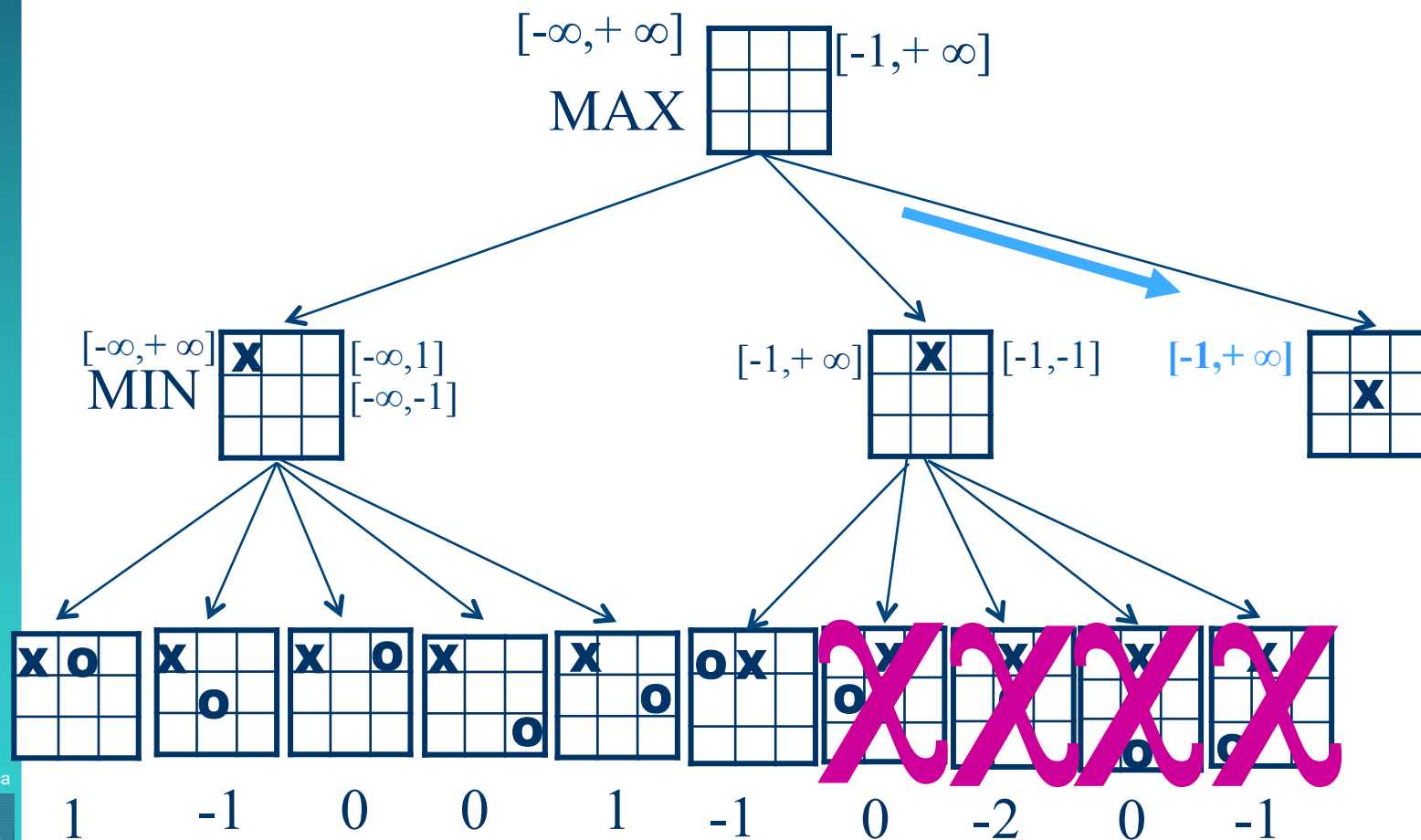
Ejercicio

- Poda alfa-beta de izquierda a derecha:



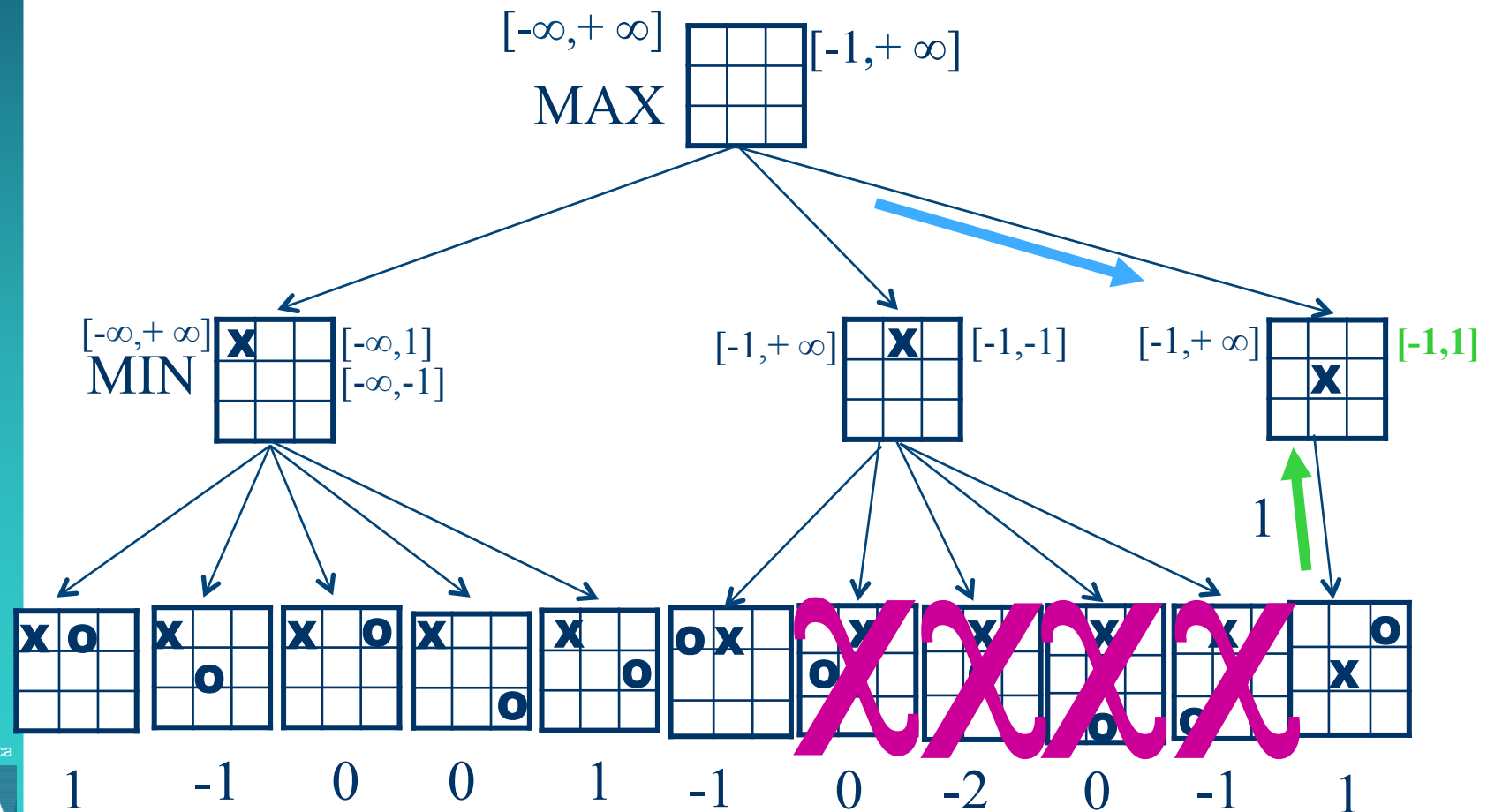
Ejercicio

- Poda alfa-beta de izquierda a derecha:



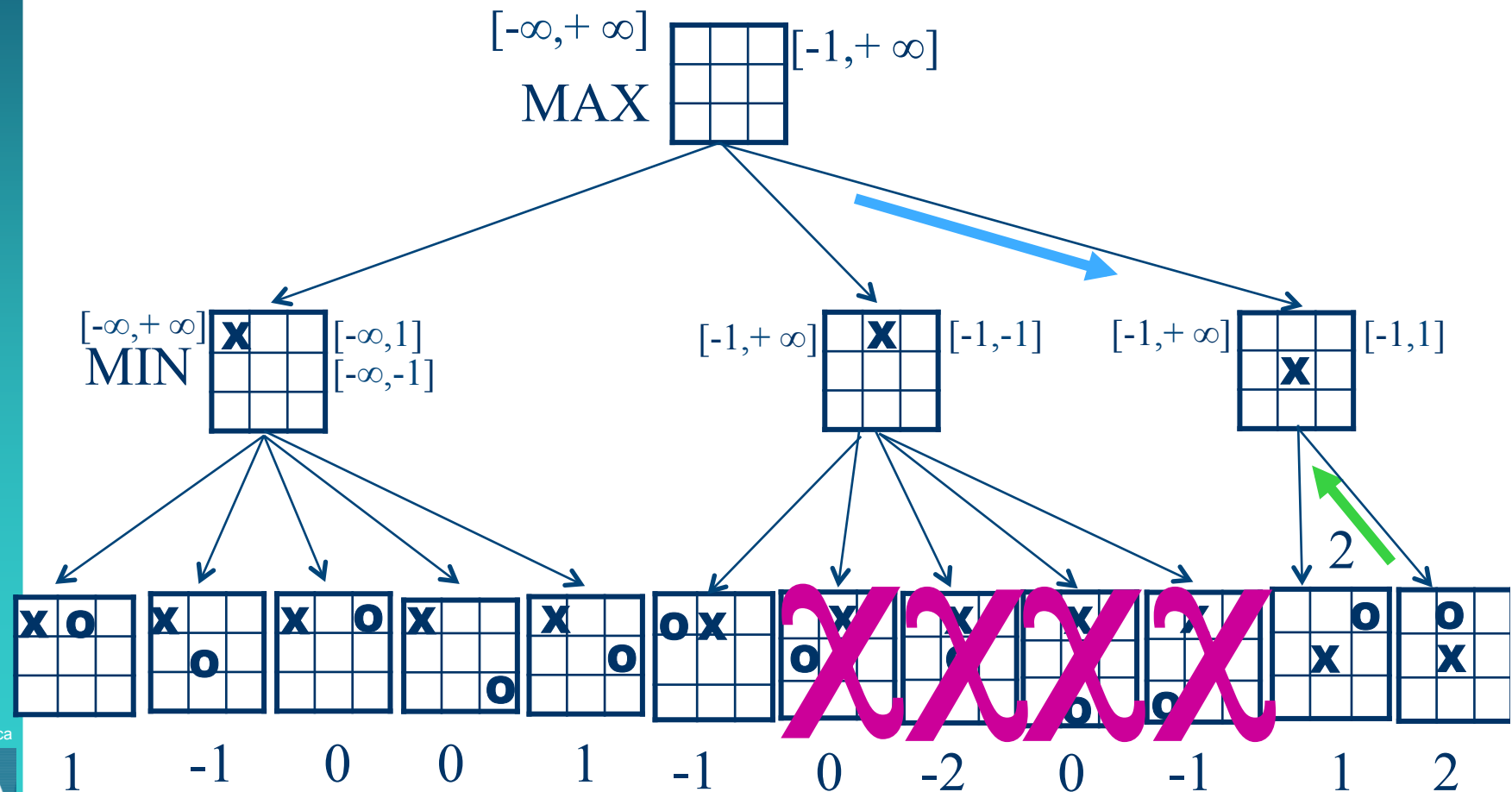
Ejercicio

- Poda alfa-beta de izquierda a derecha:



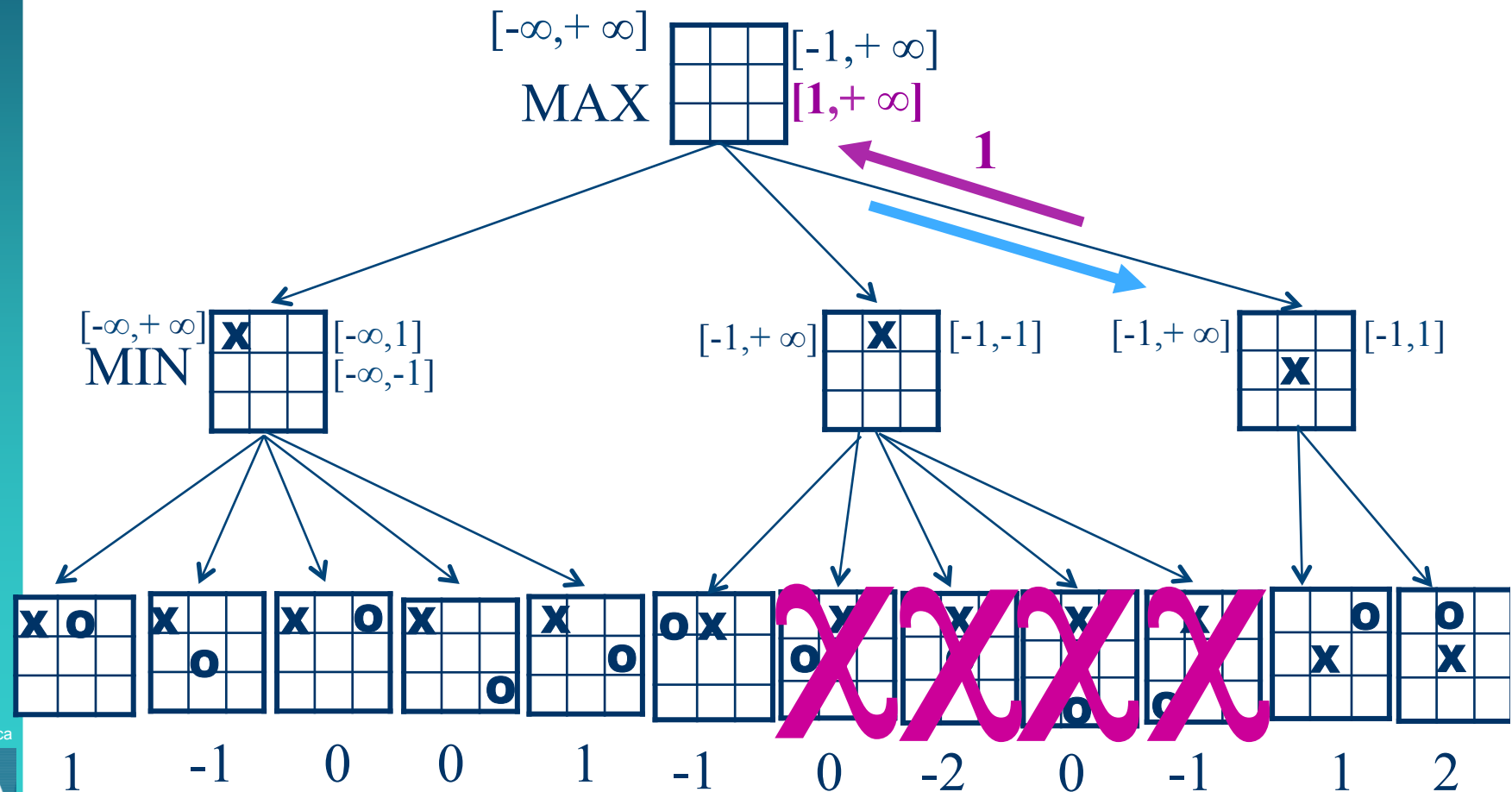
Ejercicio

- Poda alfa-beta de izquierda a derecha:



Ejercicio

- Poda alfa-beta de izquierda a derecha:





Inteligencia
Artificial

Búsqueda entre adversarios

Orden de aplicación de los operadores

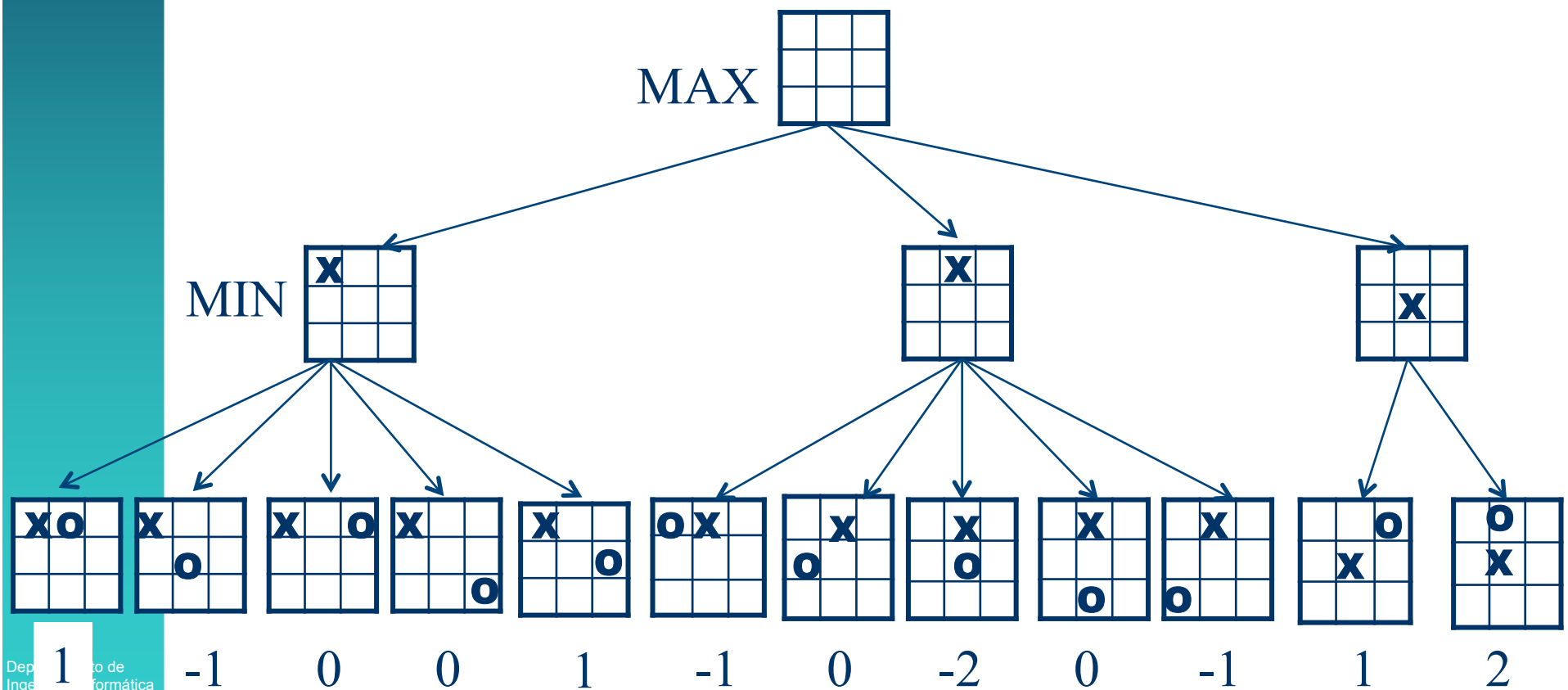
Pero el orden de aplicación de los Operadores Sí puede modificar la Poda

Departamento de
Ingeniería Informática



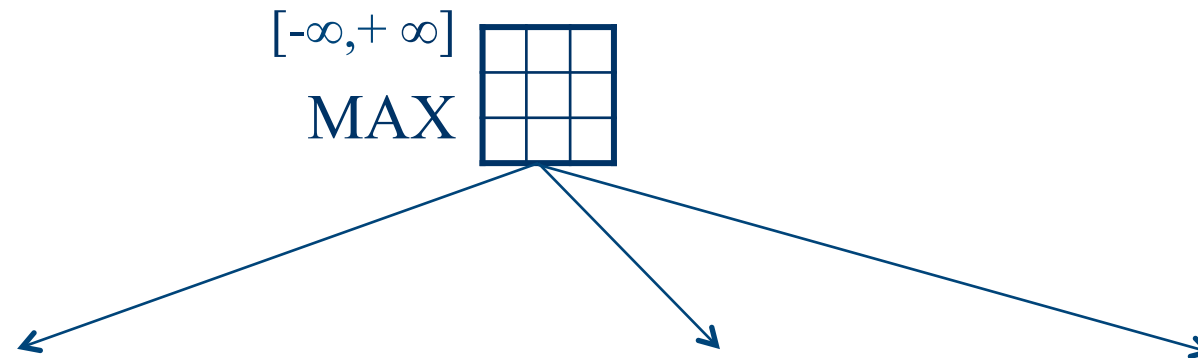
Ejercicio

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



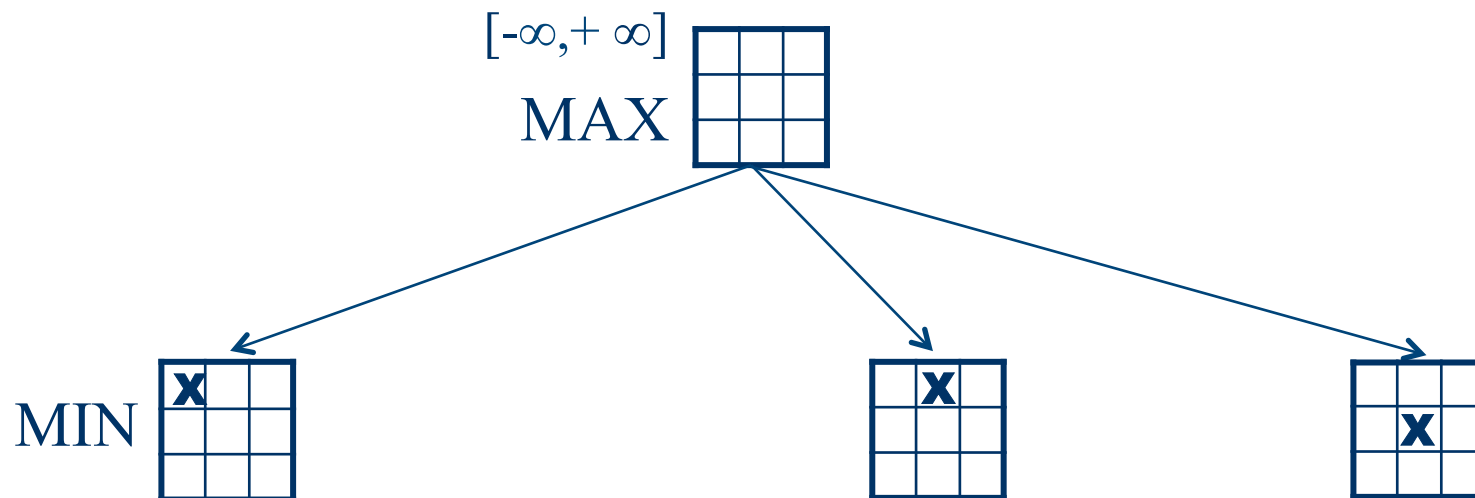
Ejercicio

- Poda alfa-beta de derecha a izquierda:



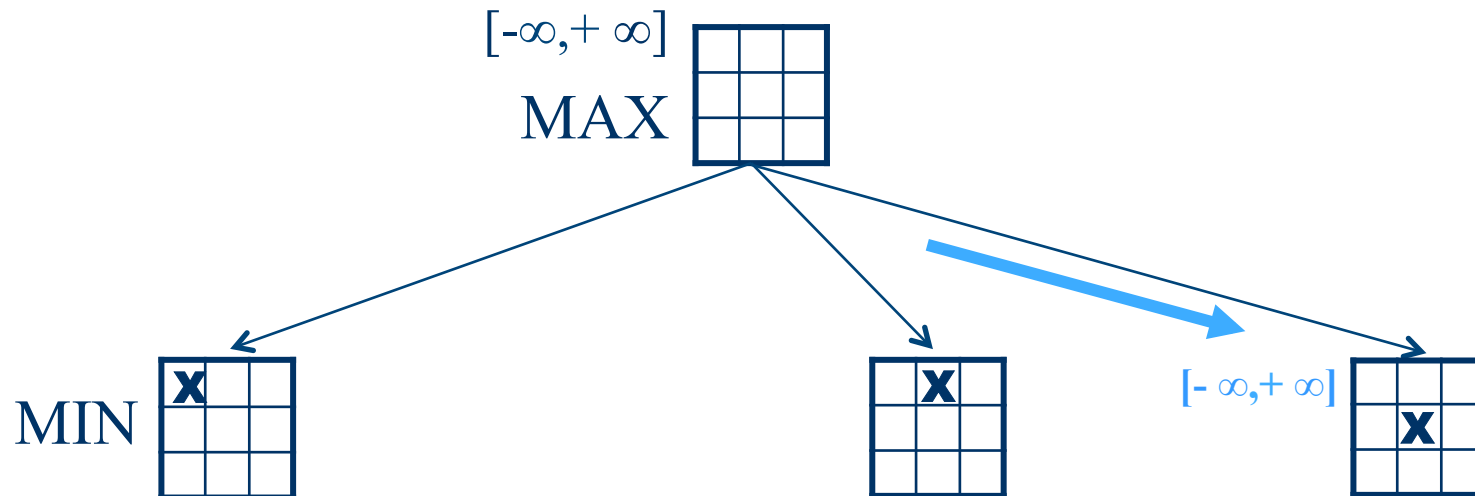
Ejercicio

- Poda alfa-beta de derecha a izquierda:



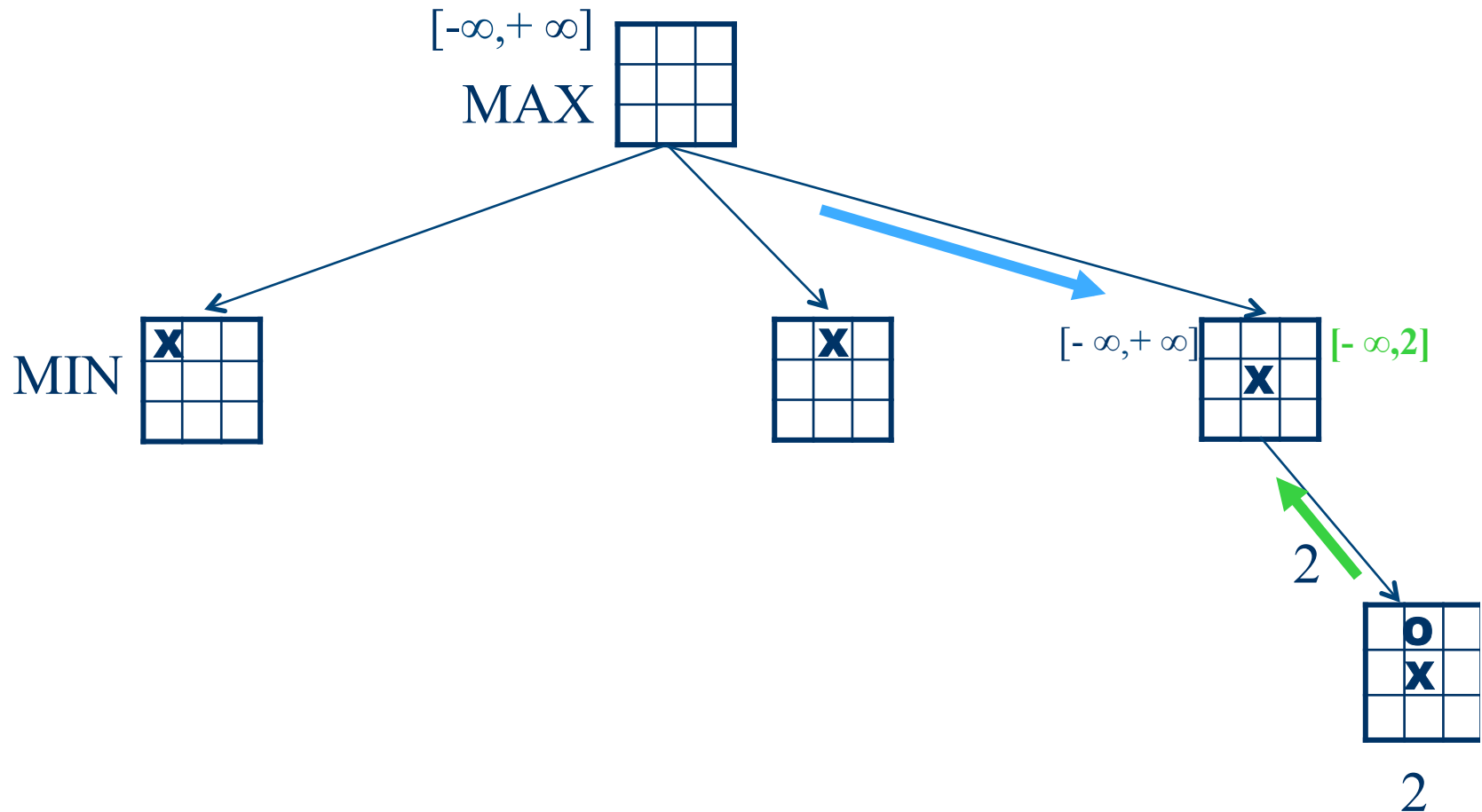
Ejercicio

- Poda alfa-beta de derecha a izquierda:



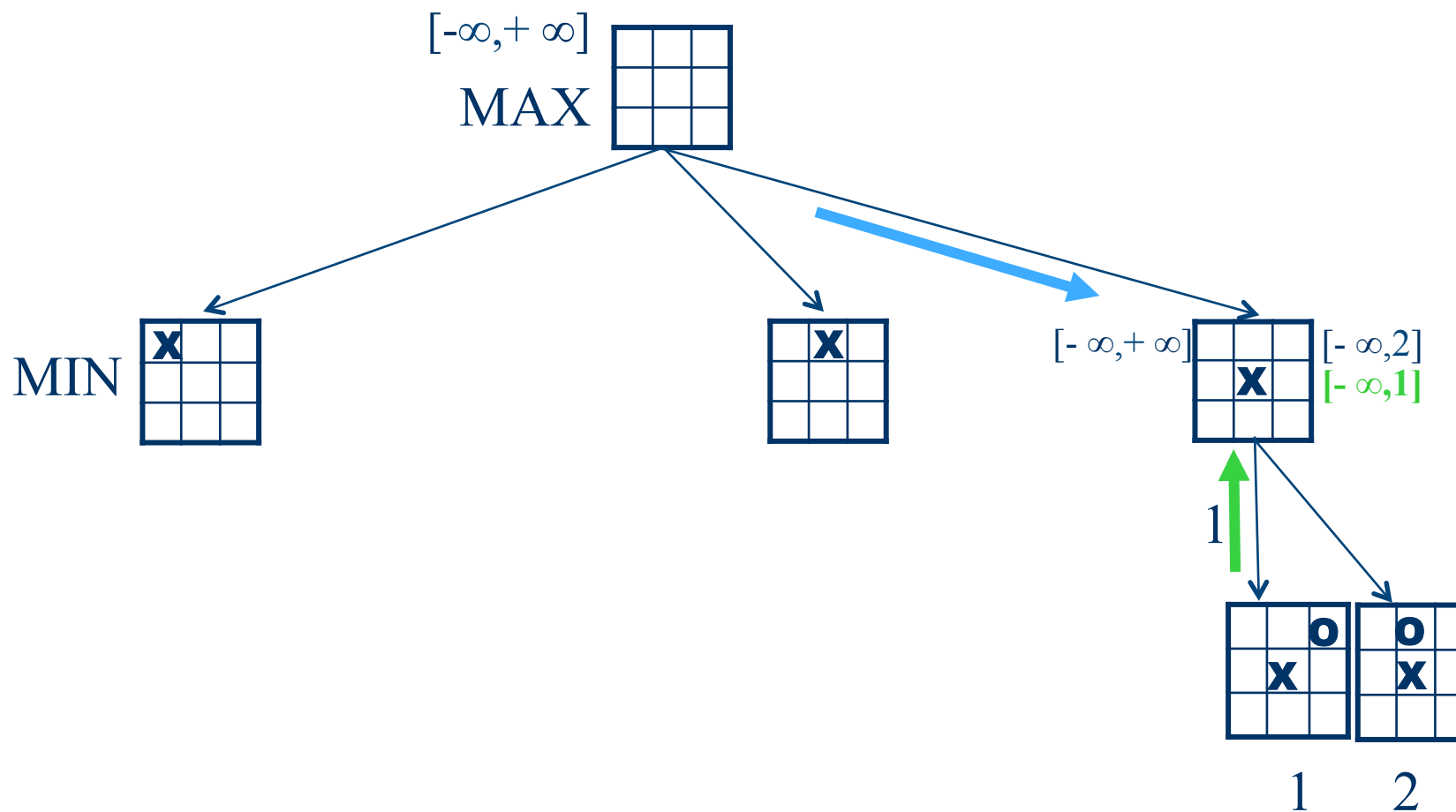
Ejercicio

- Poda alfa-beta de derecha a izquierda:



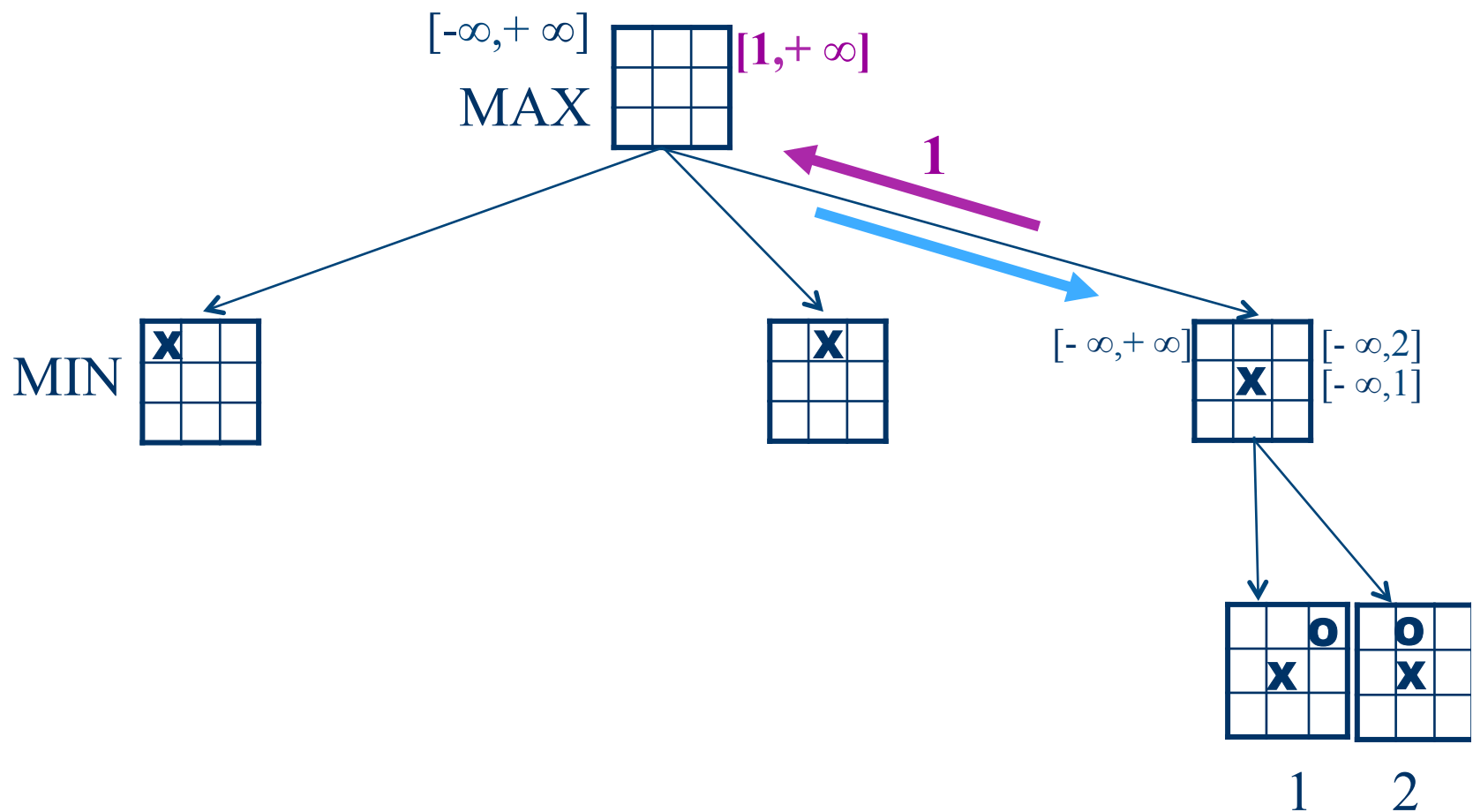
Ejercicio

- Poda alfa-beta de derecha a izquierda:



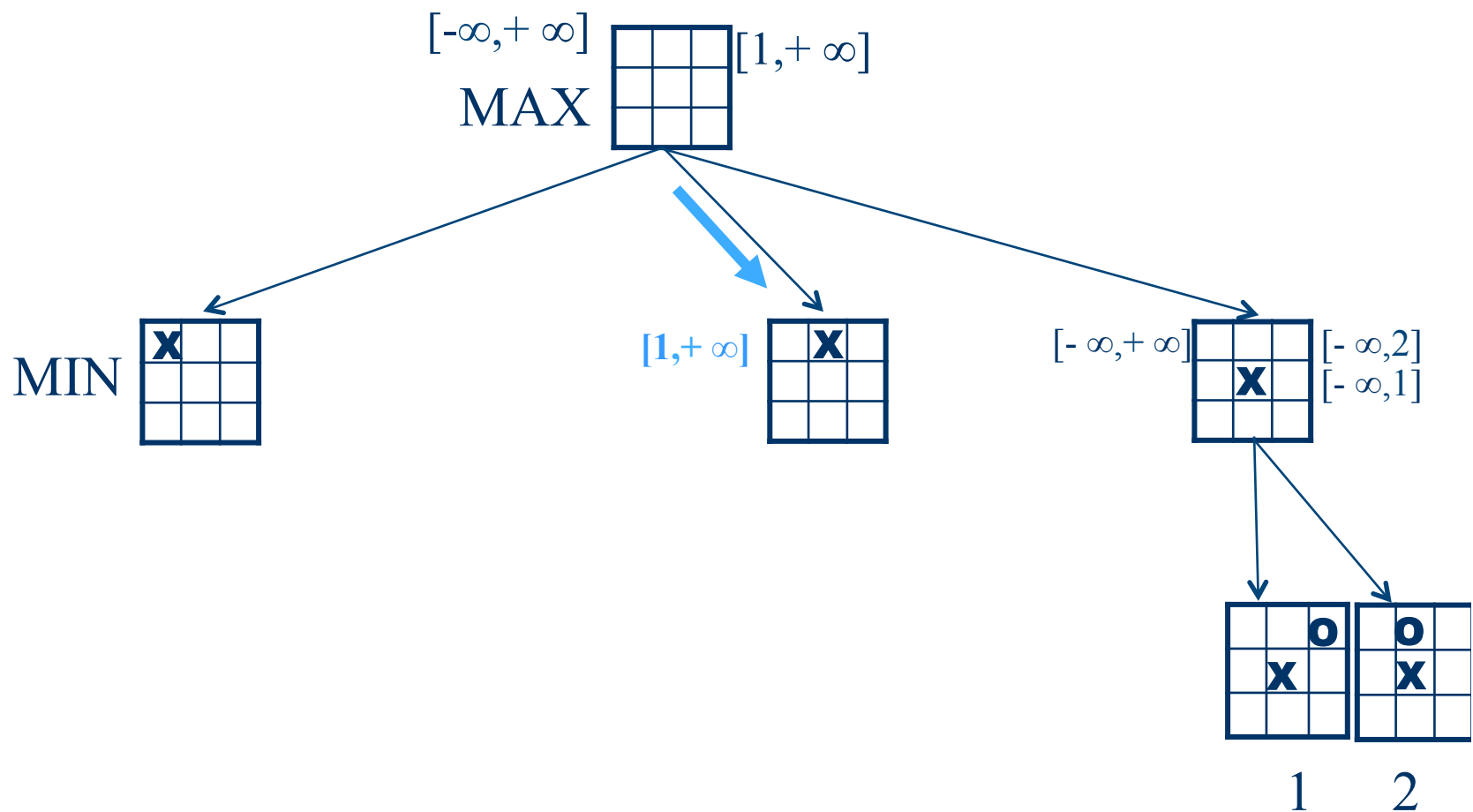
Ejercicio

- Poda alfa-beta de derecha a izquierda:



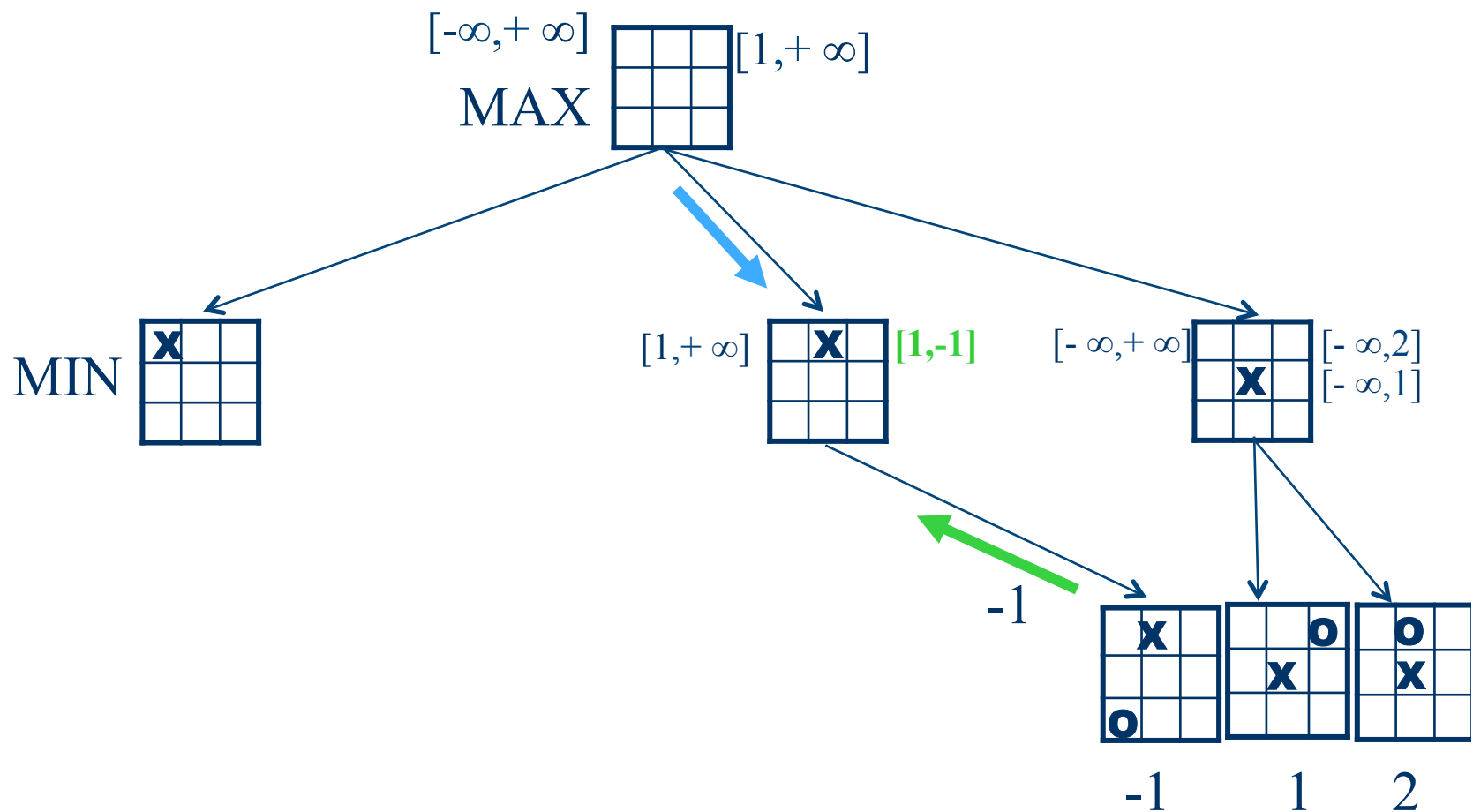
Ejercicio

- Poda alfa-beta de derecha a izquierda:



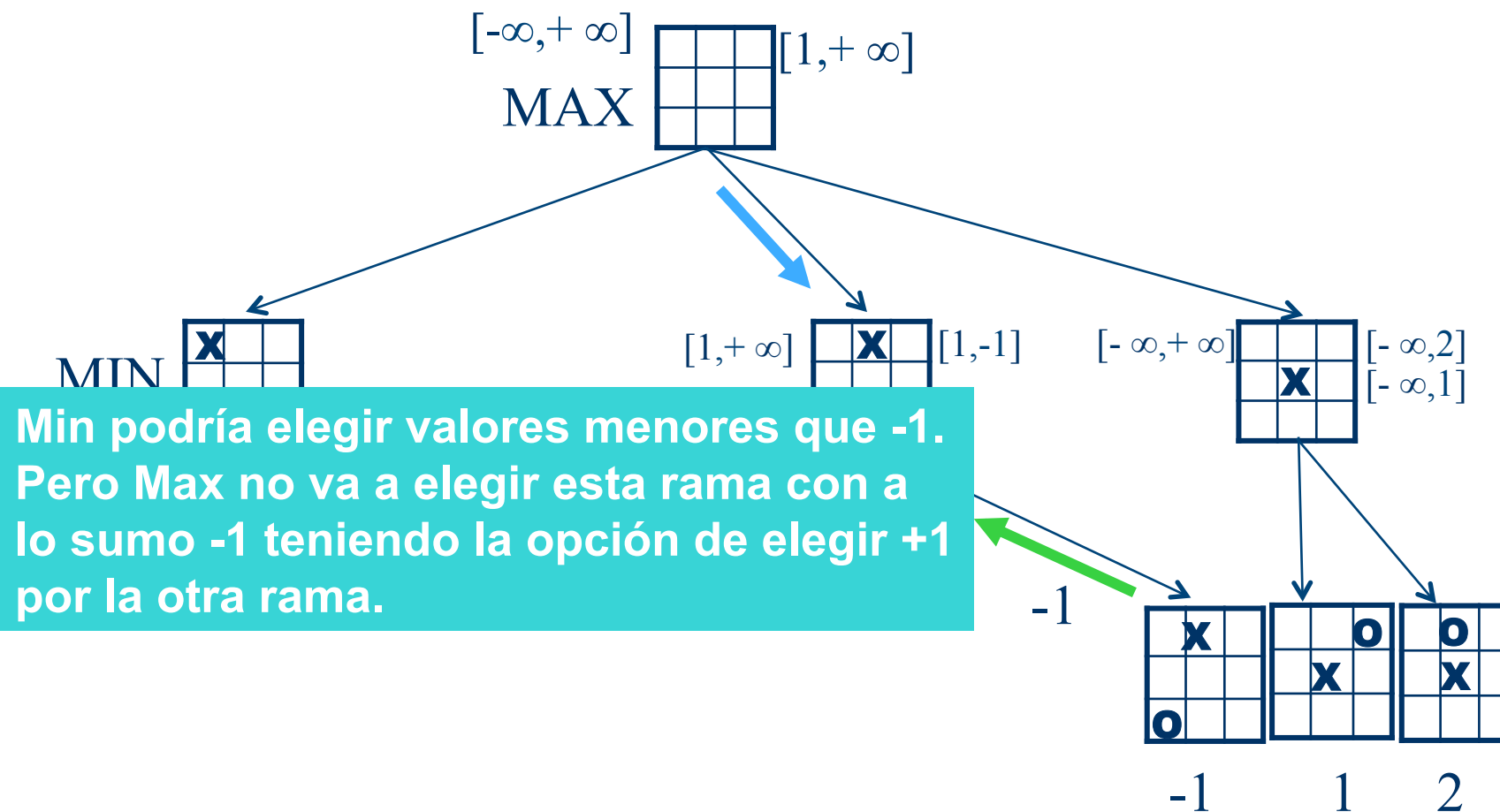
Ejercicio

- Poda alfa-beta de derecha a izquierda:



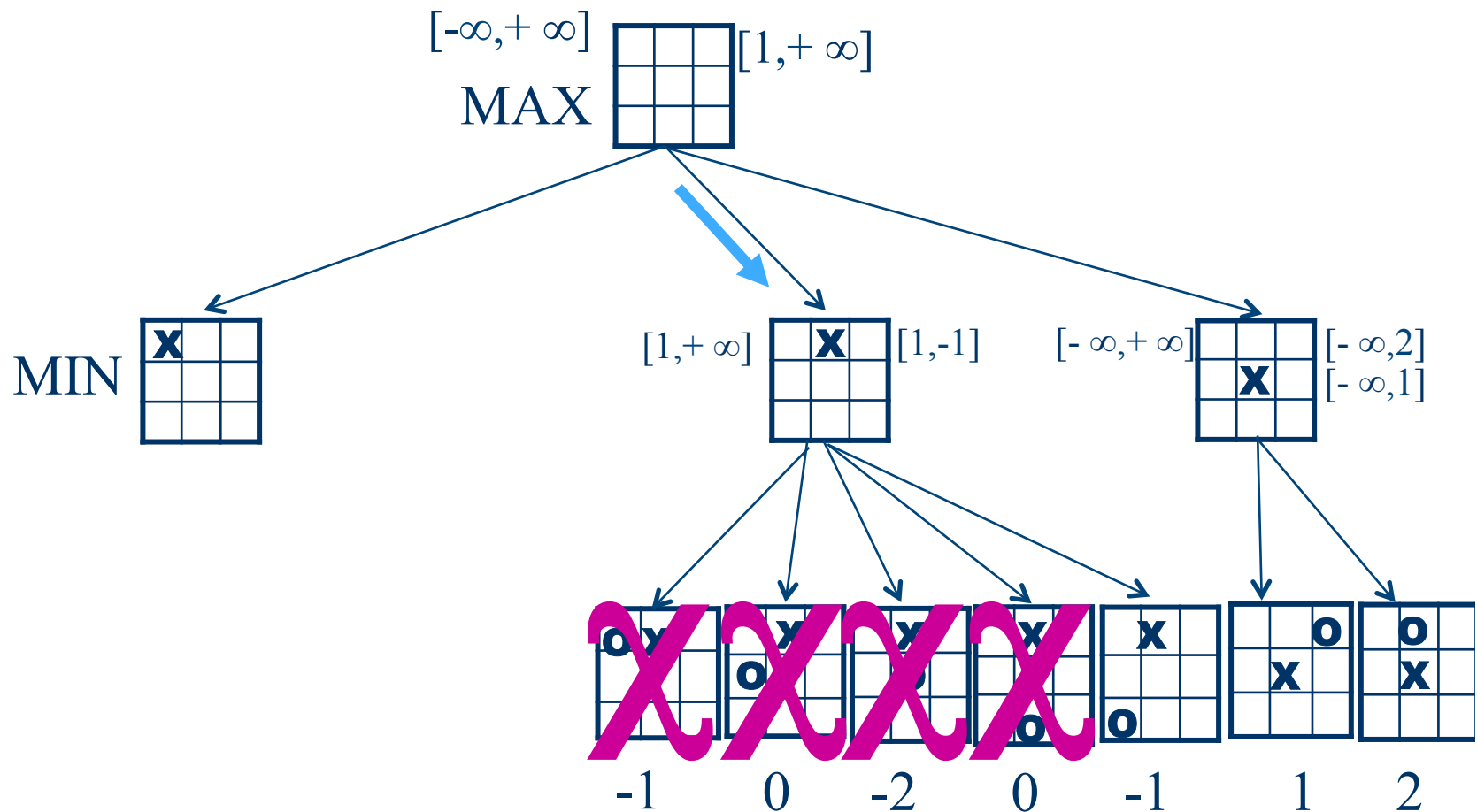
Ejercicio

- Poda alfa-beta de derecha a izquierda:



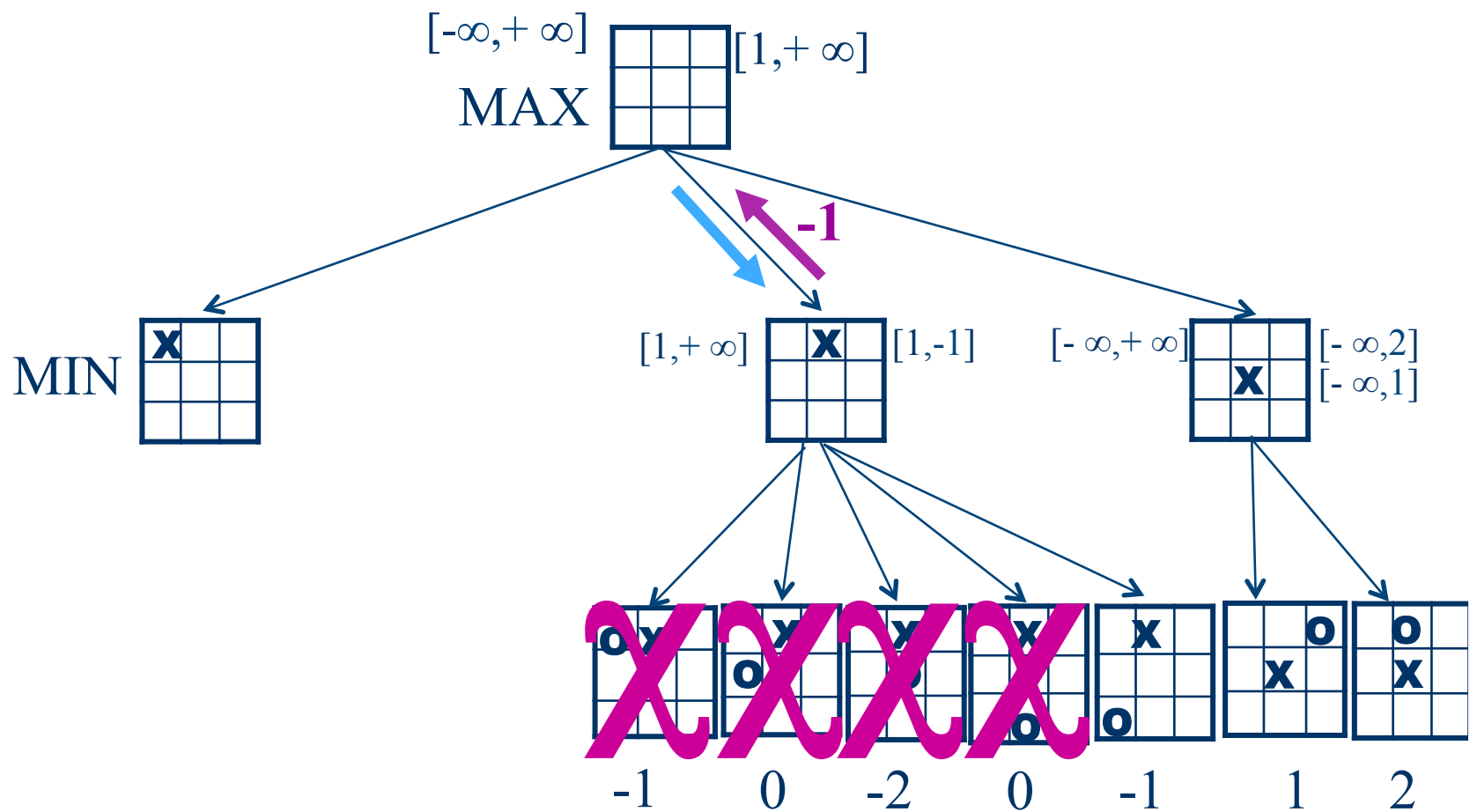
Ejercicio

- Poda alfa-beta de derecha a izquierda:



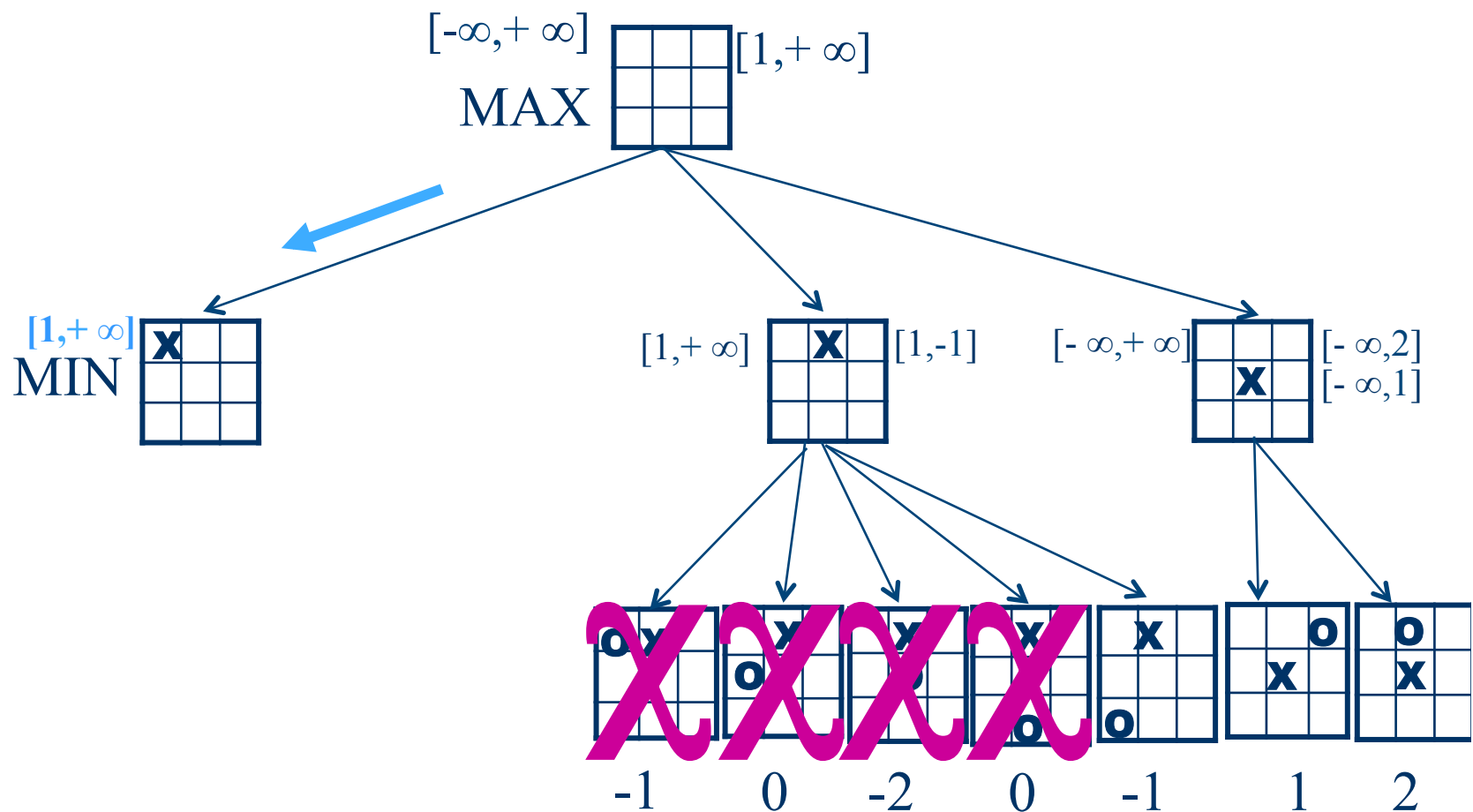
Ejercicio

- Poda alfa-beta de derecha a izquierda:



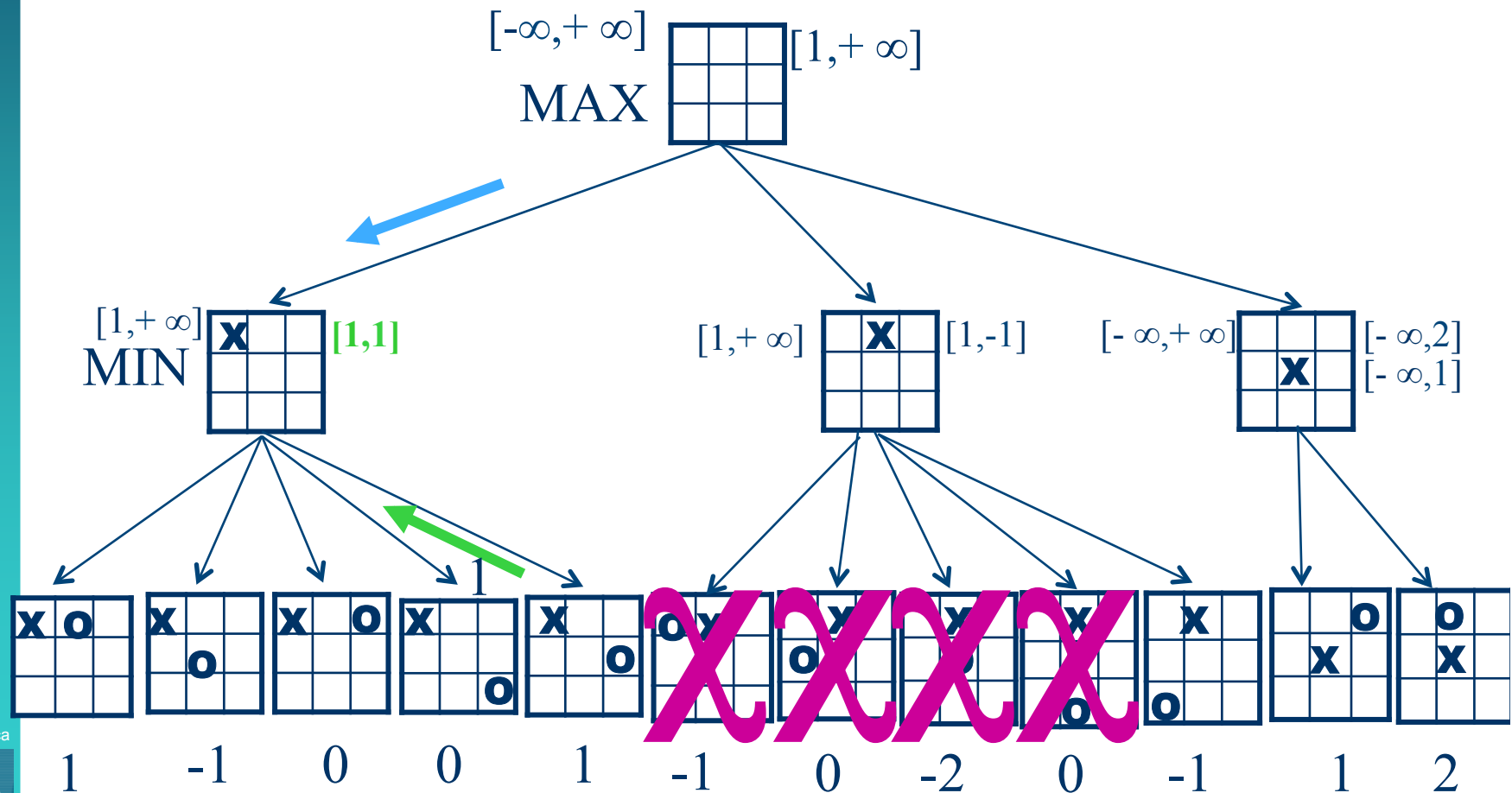
Ejercicio

- Poda alfa-beta de derecha a izquierda:



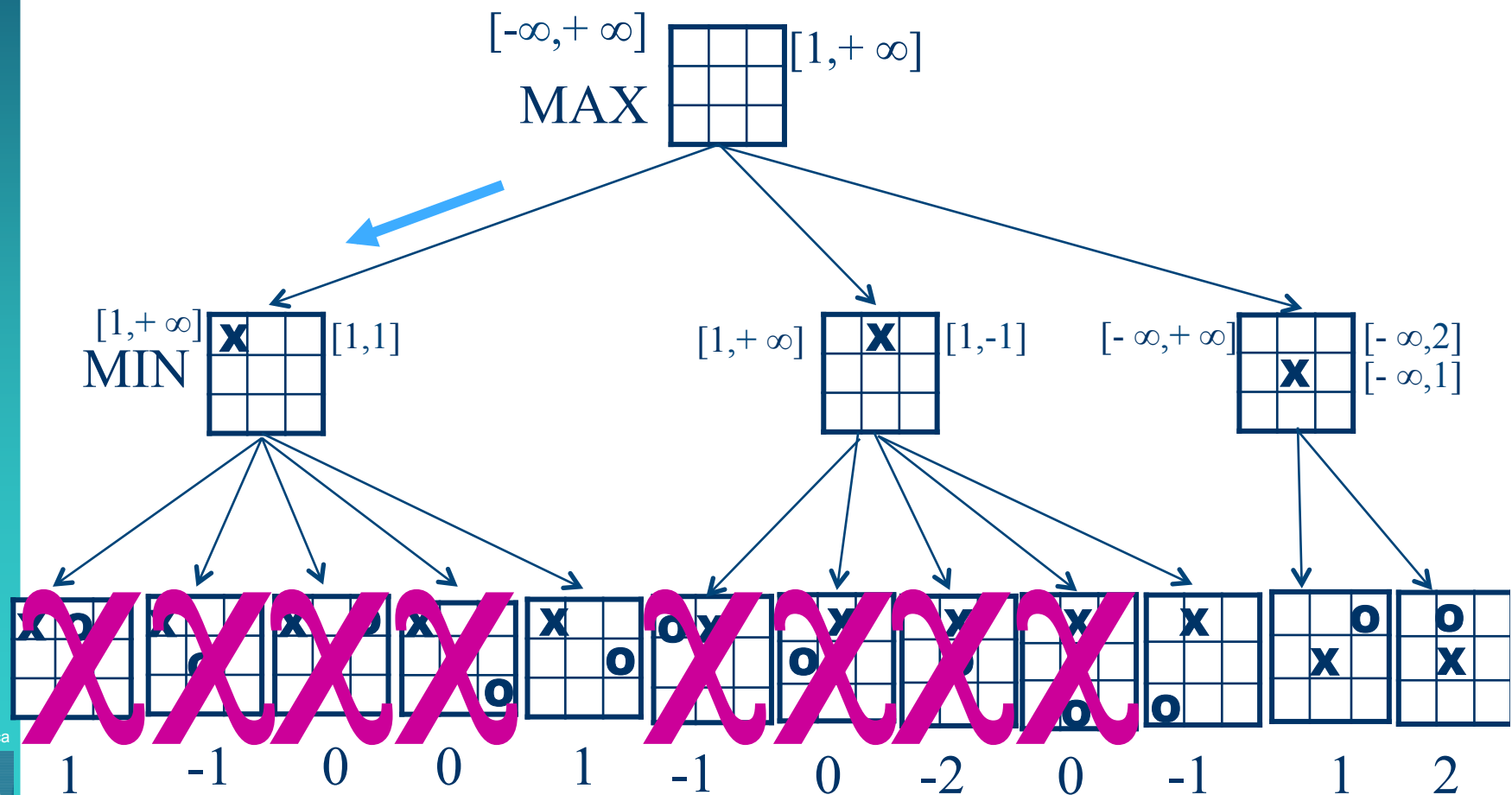
Ejercicio

- Poda alfa-beta de derecha a izquierda:



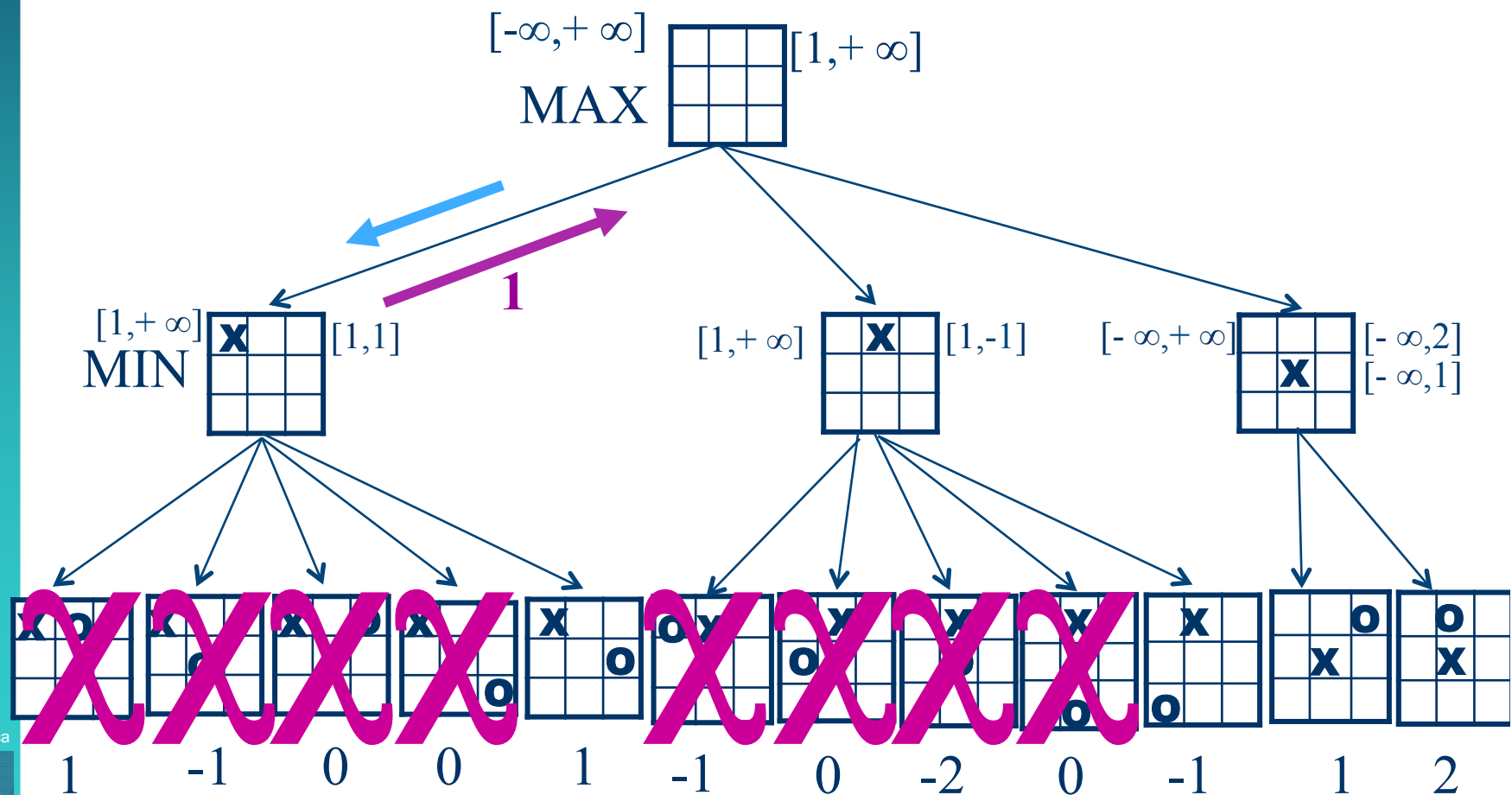
Ejercicio

- Poda alfa-beta de derecha a izquierda:



Ejercicio

- Poda alfa-beta de derecha a izquierda:



Referencias

Las 2 primeras referencias ofrecen buenos textos para repasar los tópicos vistos en esta clase y también son muy útiles para ampliar conocimientos sobre otras variedades de técnicas y juegos (con más adversarios, con azar, etc.)

- 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 puede 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