

Ayudantía 6

Heurísticas y búsqueda adversaria

Maximiliano Torres - Blanca Romero

Heurísticas

¿Qué es una heurística?

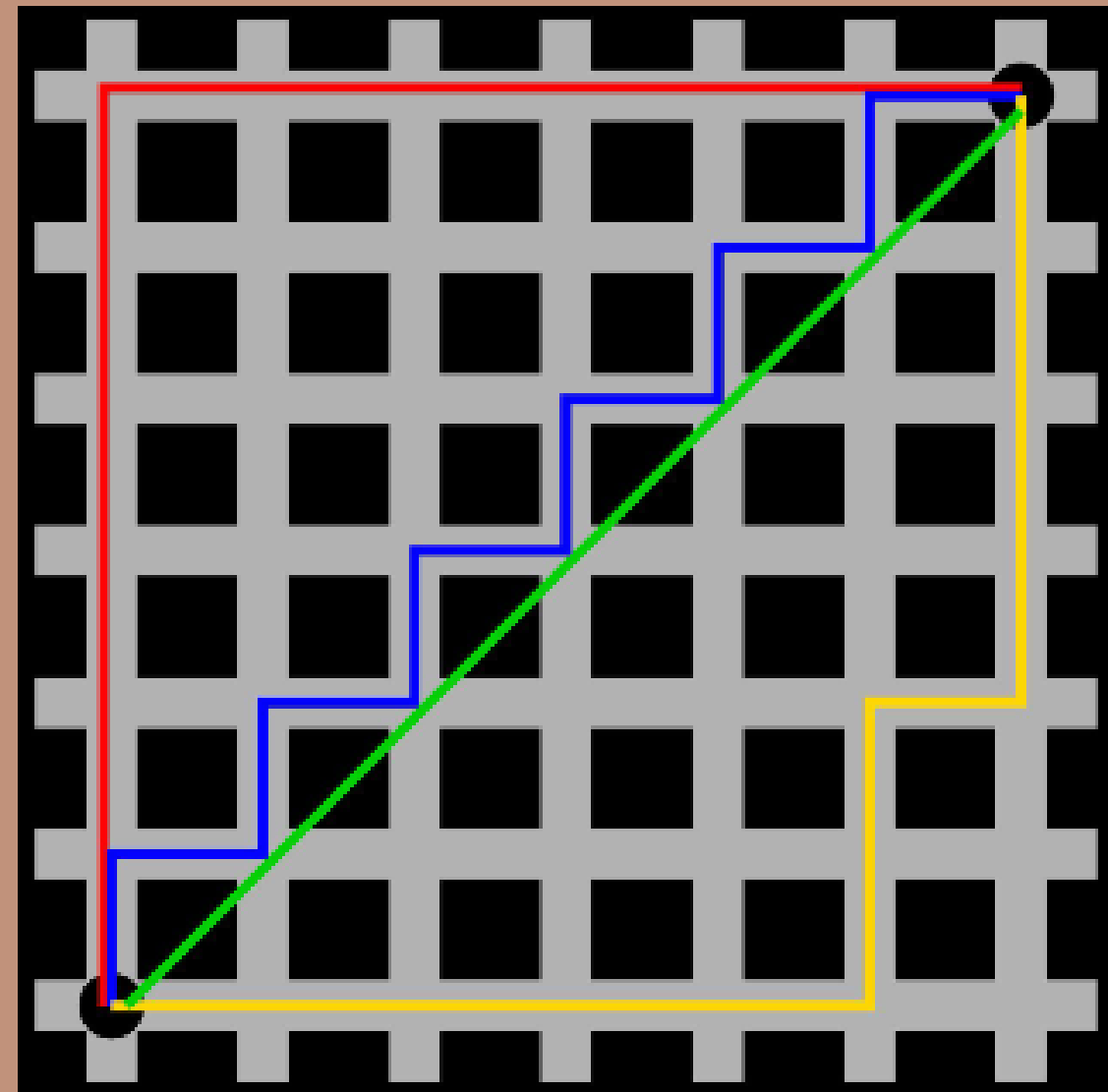
Las heurísticas usan **conocimiento previo** para discriminar cuáles estados son "mejores" o "peores".

Así se **reduce el tiempo de búsqueda** (pues evitamos irnos por caminos "malos").

La heurística es una **estimación** de la cercanía de cierto estado a la solución.

Ejemplos de heurísticas

- No comprar un producto con mala calificación.
- Llevar paraguas si el día está nublado.
- Elegir un producto porque su envase se ve de buena calidad.
- Distancia Euclidiana.
- Distancia Manhattan.



Formalmente...

Si se tiene un problema de búsqueda **(S, A, s_init, G)**

S = conjunto de estados posibles

A = conjunto de acciones posibles

s_init = estado inicial

G = conjunto de estados objetivo (puede ser sólo uno...)

Una heurística es una función h que recibe como entrada un estado s y entrega un valor real.

La idea es que el valor $h(s)$ sea una medida de qué tan cerca está el estado s de alguna solución g .

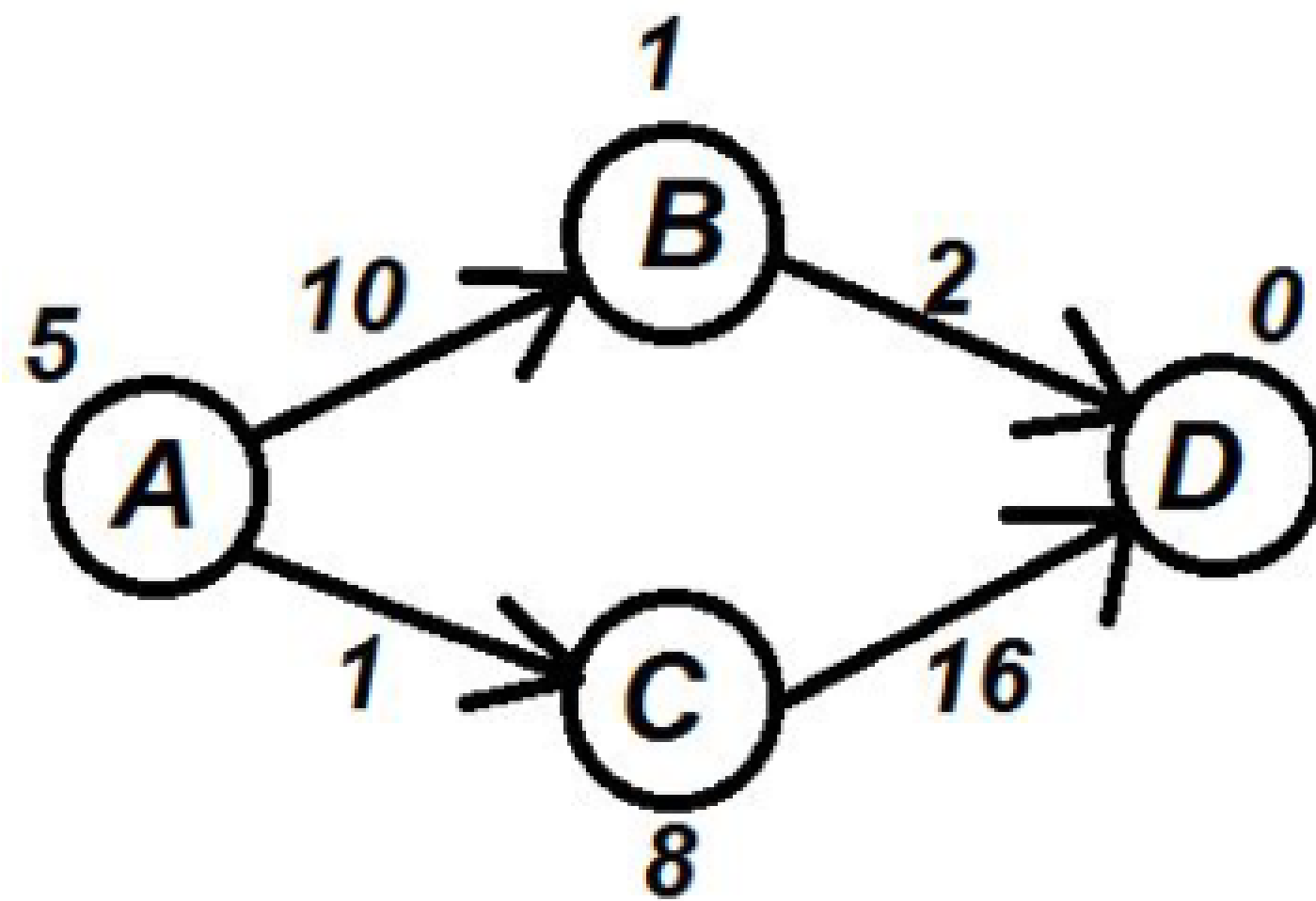
Admisibilidad

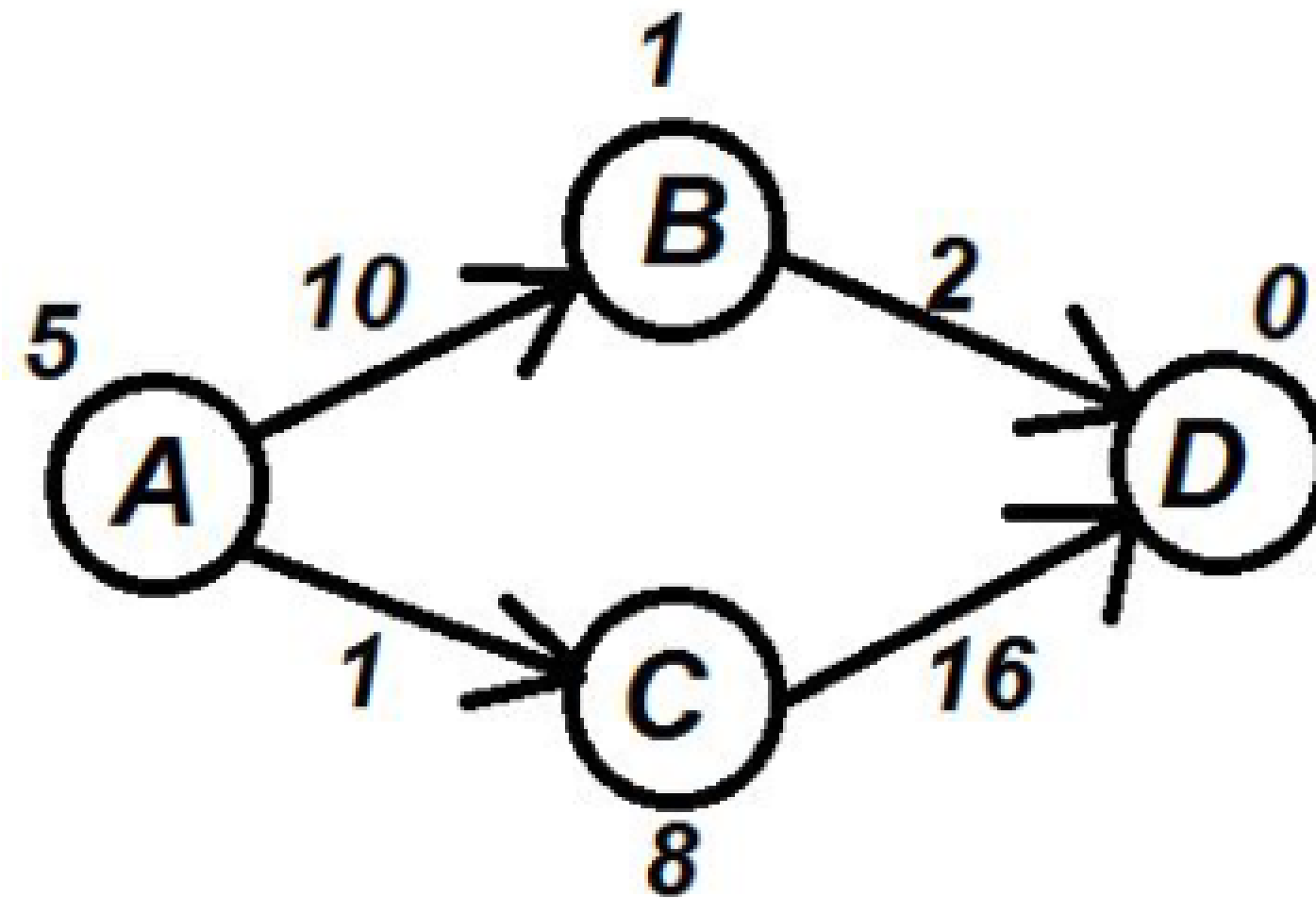
Una heurística h se dice **admisible** si y solo si **nunca sobreestima respecto a un camino óptimo**. Es decir:

$$h(s) \leq h^*(s)$$

para **todo estado s** .

$h^*(s)$ = costo de un camino óptimo desde s hasta un objetivo





$$h(A) \leq h^*(A)$$

$$h(B) \leq h^*(B)$$

$$h(C) \leq h^*(C)$$



$$5 \leq 12 \quad \checkmark$$

$$1 \leq 2 \quad \checkmark$$

$$8 \leq 16 \quad \checkmark$$



La heurística es
admisibile!

Consistencia

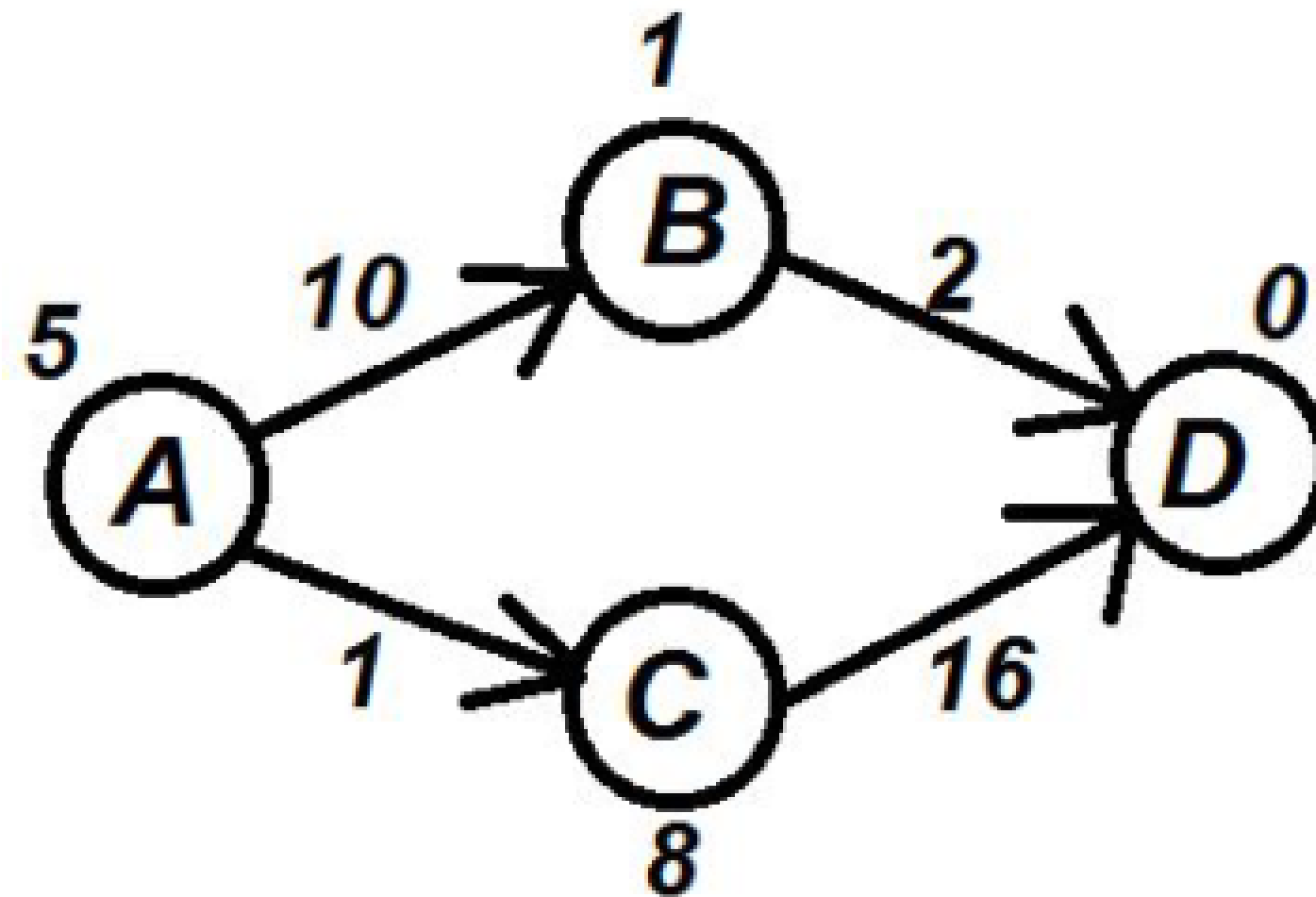
Una heurística h se dice **consistente** si y solo si

- $h(s) = 0$ para todo $s \in G$

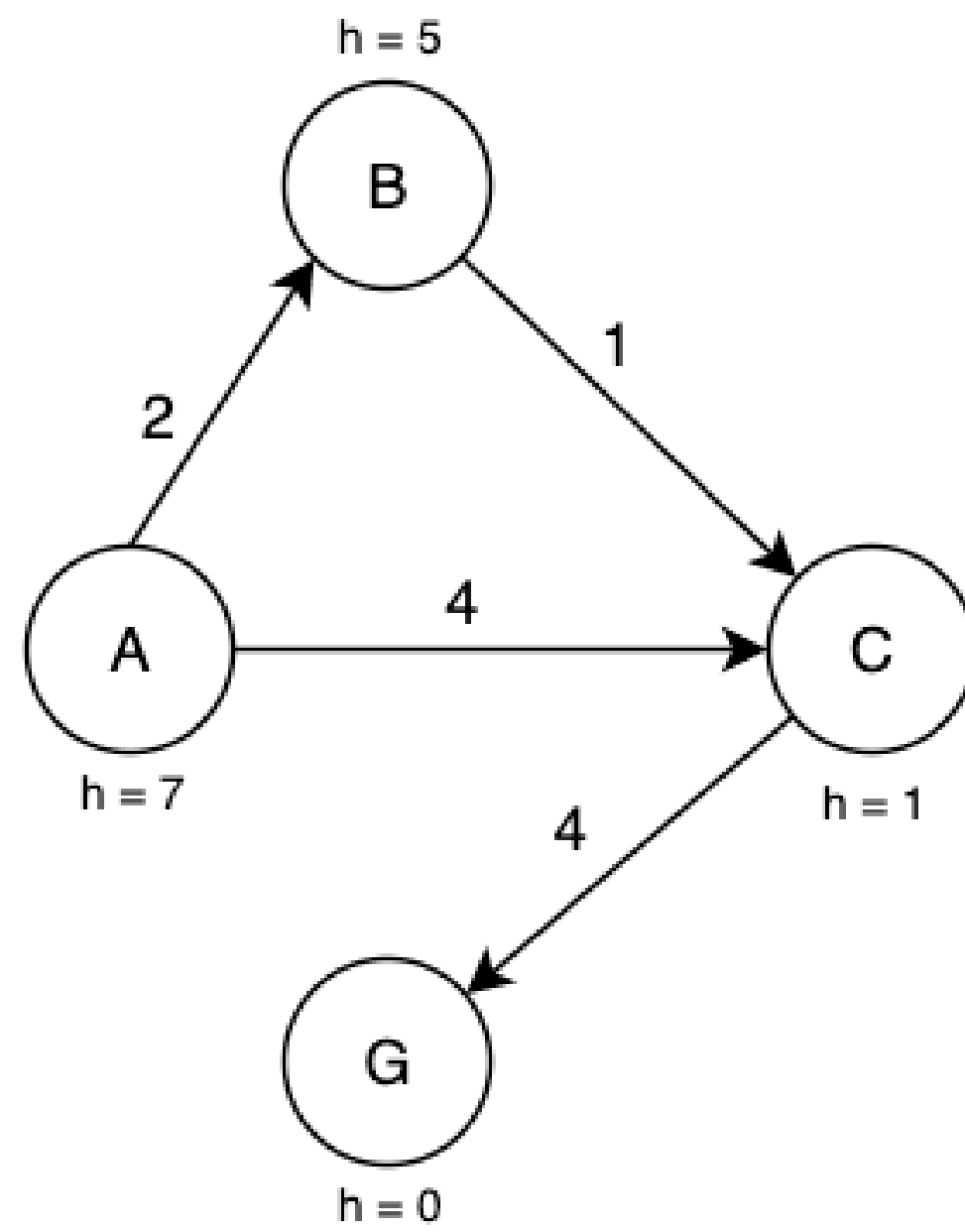
No se sobre estima
en los objetivos

- $h(s) \leq c(s, s') + h(s')$ para todo vecino s' de s

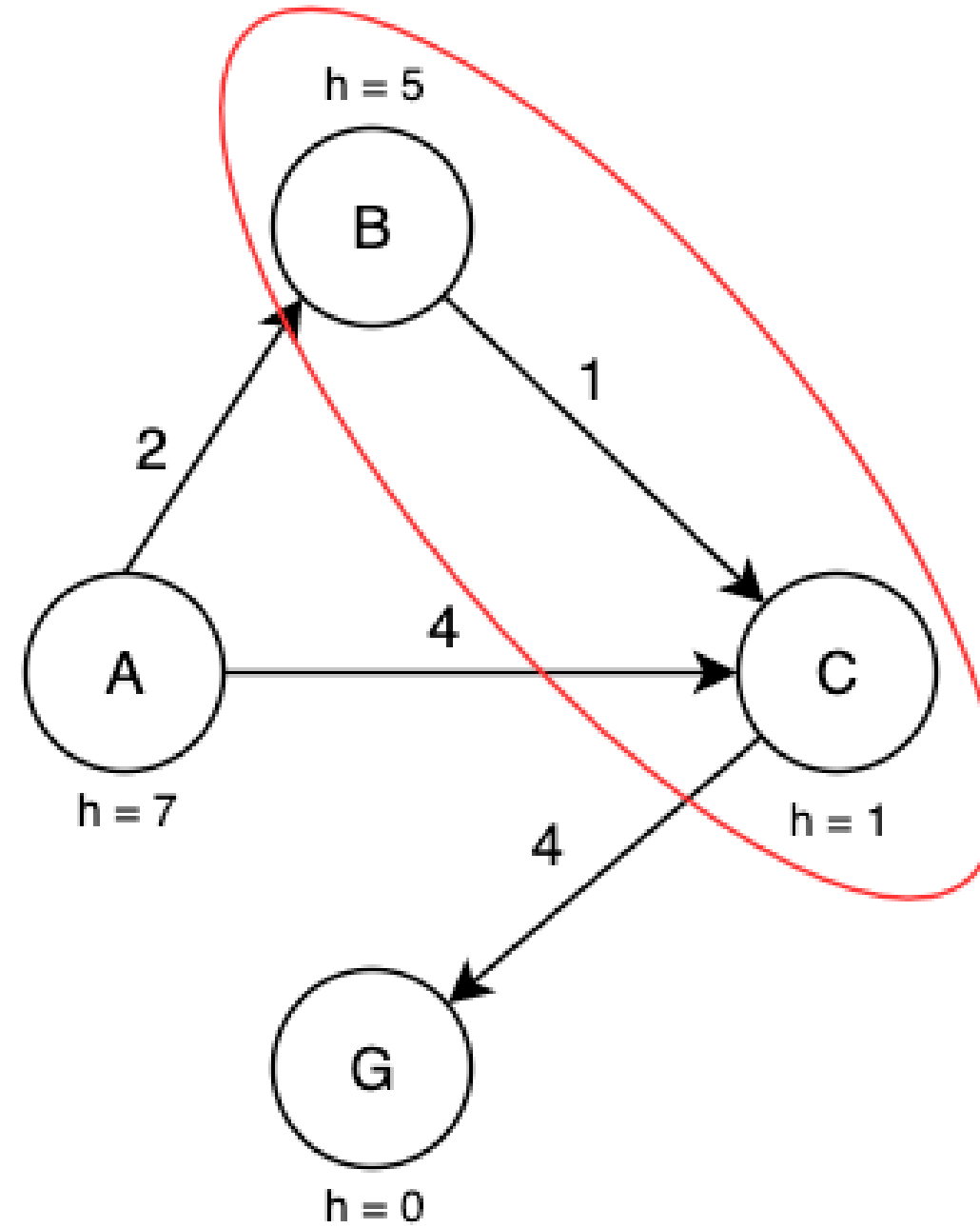
La variación del h de dos
estados no es mayor que el
costo de ir de uno a otro



La heurística también
es **consistente!**



la heurística es **inconsistente**



$$h(B) \leq c(B, C) + h(C)$$



$$5 \leq 1 + 1 \quad \text{X}$$

IMPORTANTE

Si h es **consistente**, entonces h es **admisibile**.

El algoritmo A* siempre entrega soluciones óptimas si se usa con una heurística admisible.

IMPORTANTE

Si h es **consistente**, entonces h es **admisibile**.

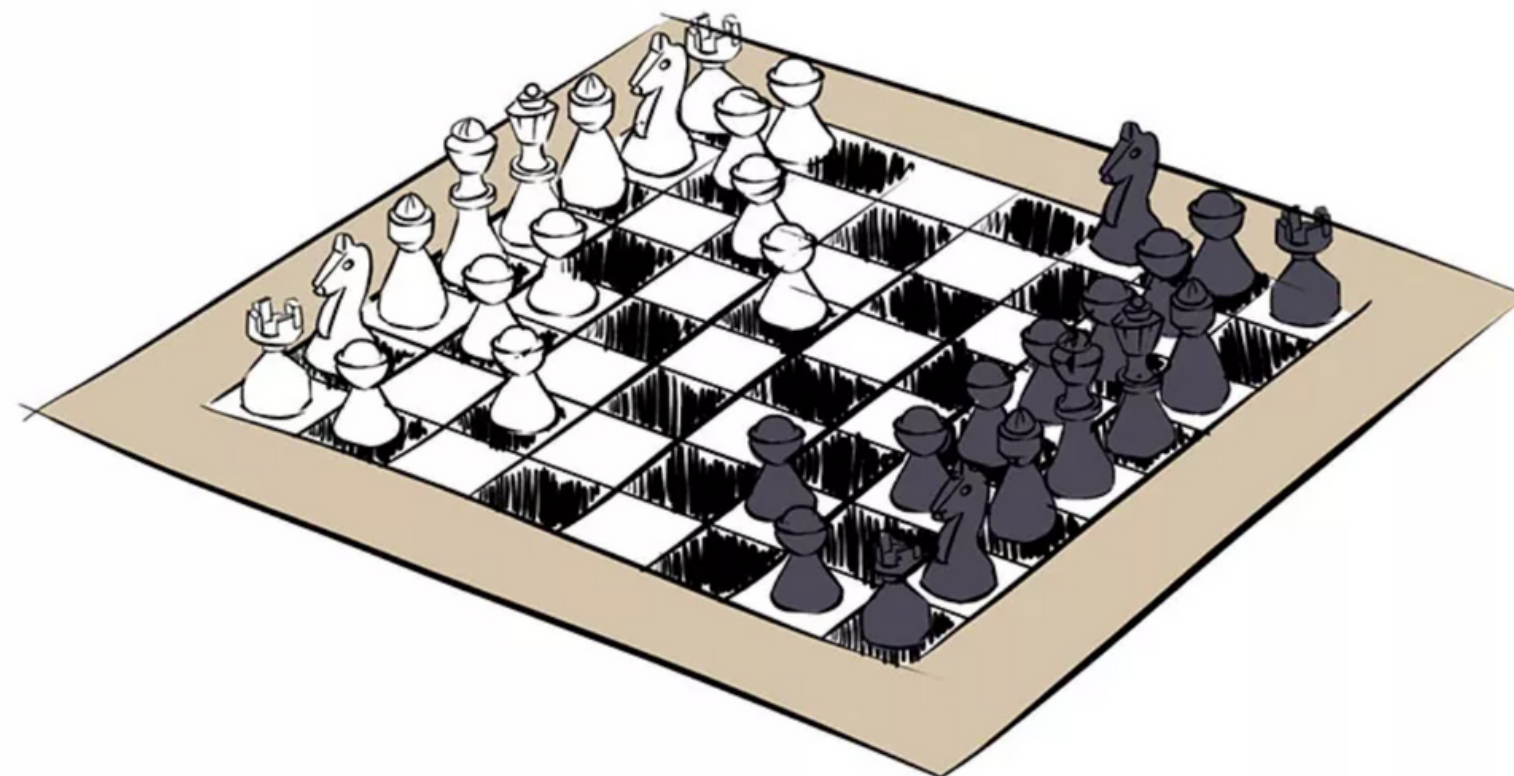
El algoritmo A* siempre entrega soluciones óptimas si se usa con una heurística admisible.

Búsqueda adversaria

Tipo de problemas (Juegos)

Veremos juegos de **dos jugadores**, con **turnos**, **información perfecta** y **suma cero**:

- **Información perfecta:** Tenemos toda la información del tablero de juego
- **Suma cero:** Lo que es bueno para mí, es malo para mi oponente (en igual medida)



Búsqueda

Podemos considerar un juego como un **espacio de búsqueda**:

- **Tablero** = nodo/estado
- **Jugadas** = conecciones/acciones

El **objetivo** será llegar a un **tablero donde ganamos** (estado objetivo):

- Gato: tres fichas (nuestras) en línea
- Conecta-4: cuatro fichas (nuestras) en línea
- Ajedrez: Jaque-mate al rey del oponente

Adversario

Para cada estado que revisemos en la búsqueda, tenemos que considerar si es **mi turno, o el turno de mi adversario**.

La búsqueda puede presentarse como una simulación del juego entre dos jugadores: Max y Min:

Si es mi turno: **Max**

- Elijo la mejor jugada que tengo disponible, para **maximizar** el puntaje.

Si es el turno de mi oponente: **Min**


- Asumo que mi oponente toma siempre las decisiones óptimas, entonces elige la opción que **minimiza** mi puntaje (la mejor jugada para mi oponente es la que me "hace peor")

¿Cómo defino la mejor jugada desde un tablero?

Puntuamos el tablero en base al valor minimax

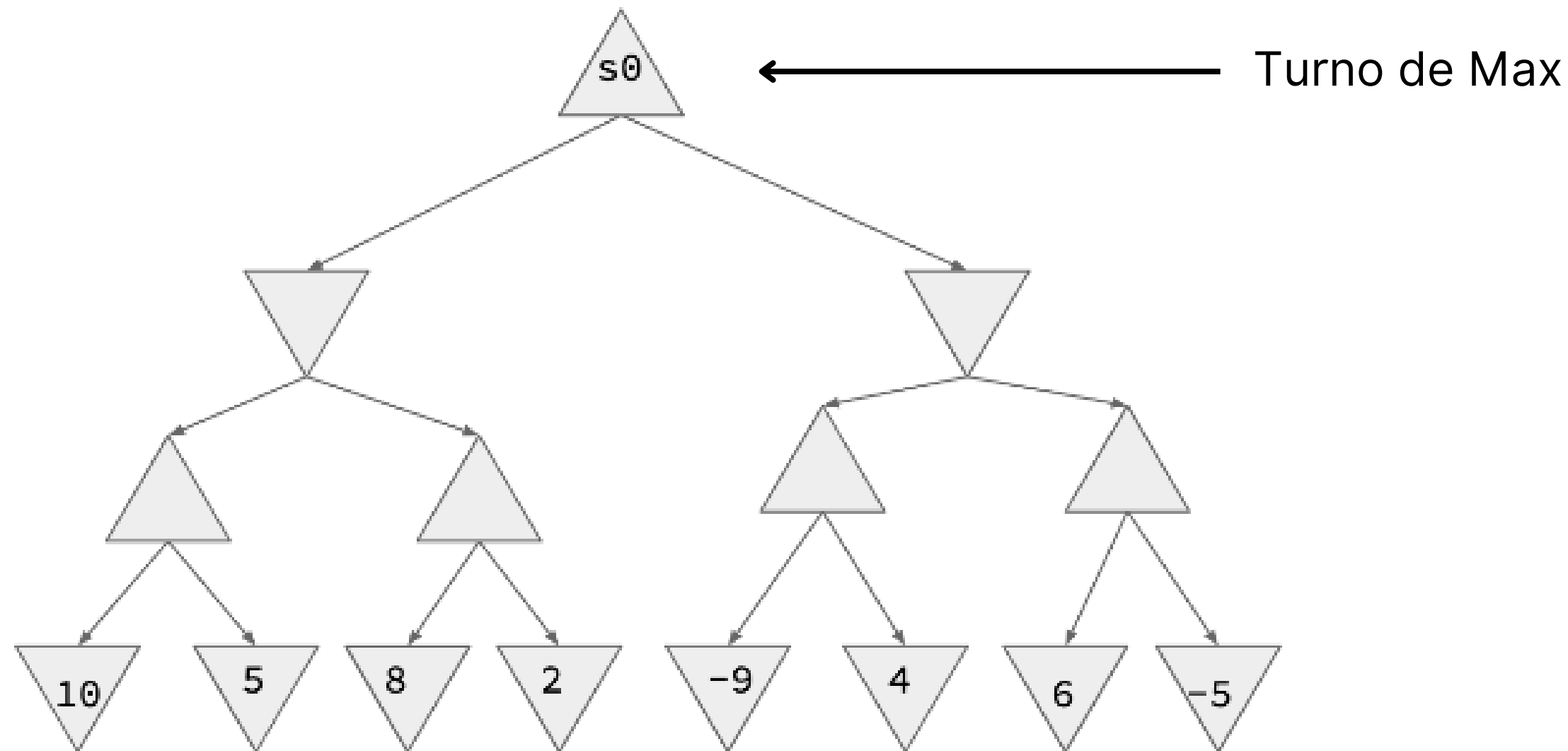
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

valor del tablero para Max



Para jugar de manera óptima, elegimos la acción asociada al valor minimax

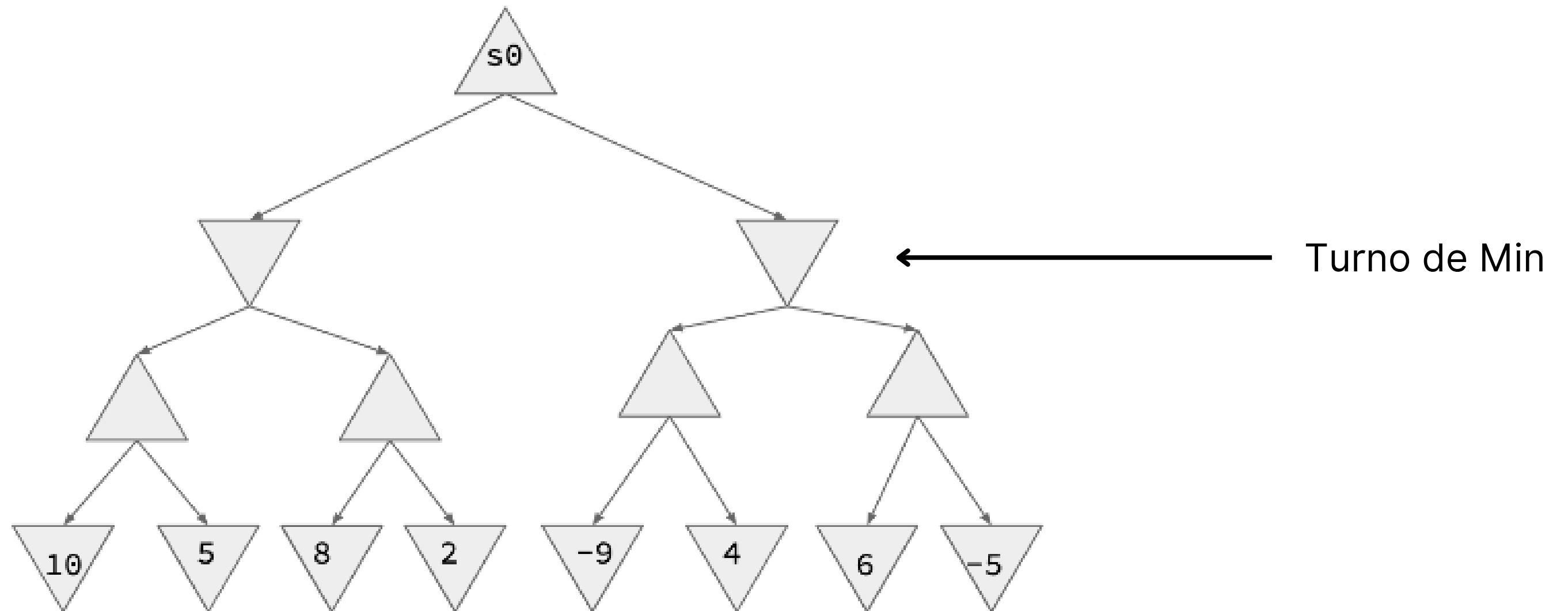
Cálculo del valor MiniMax



MINIMAX(s) =

$$\begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

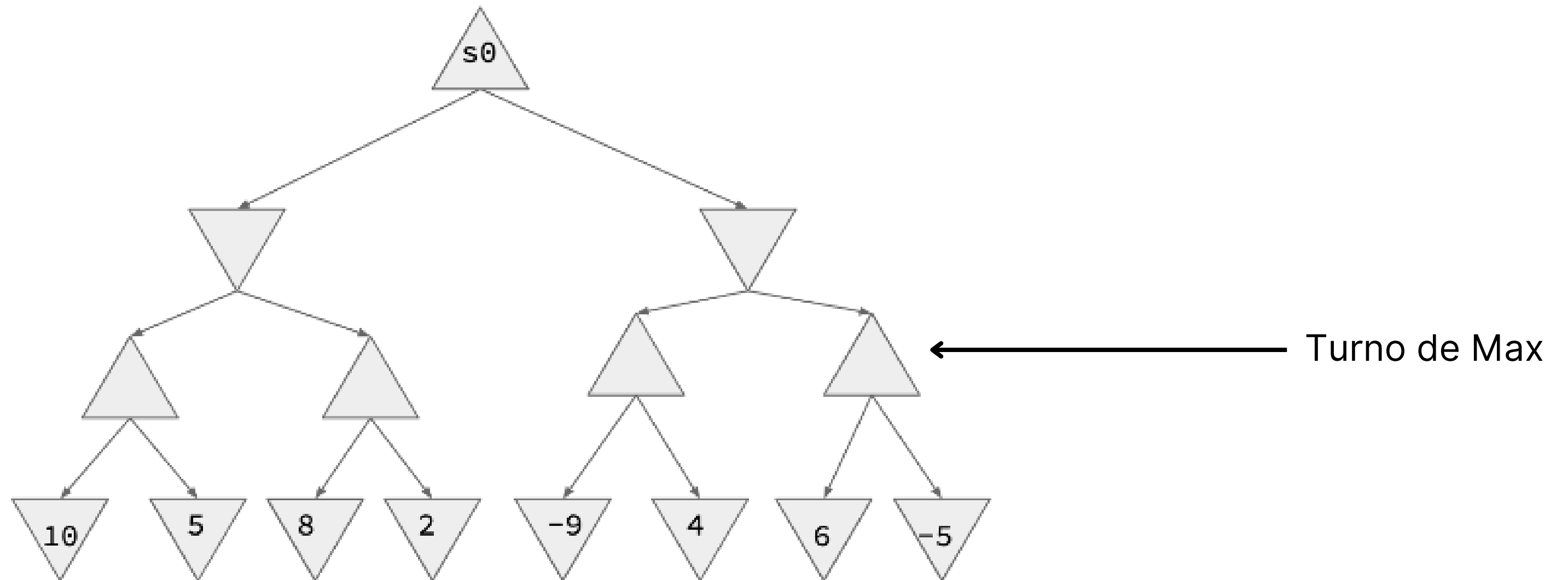
Cálculo del valor MiniMax



MINIMAX(s) =

$$\begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

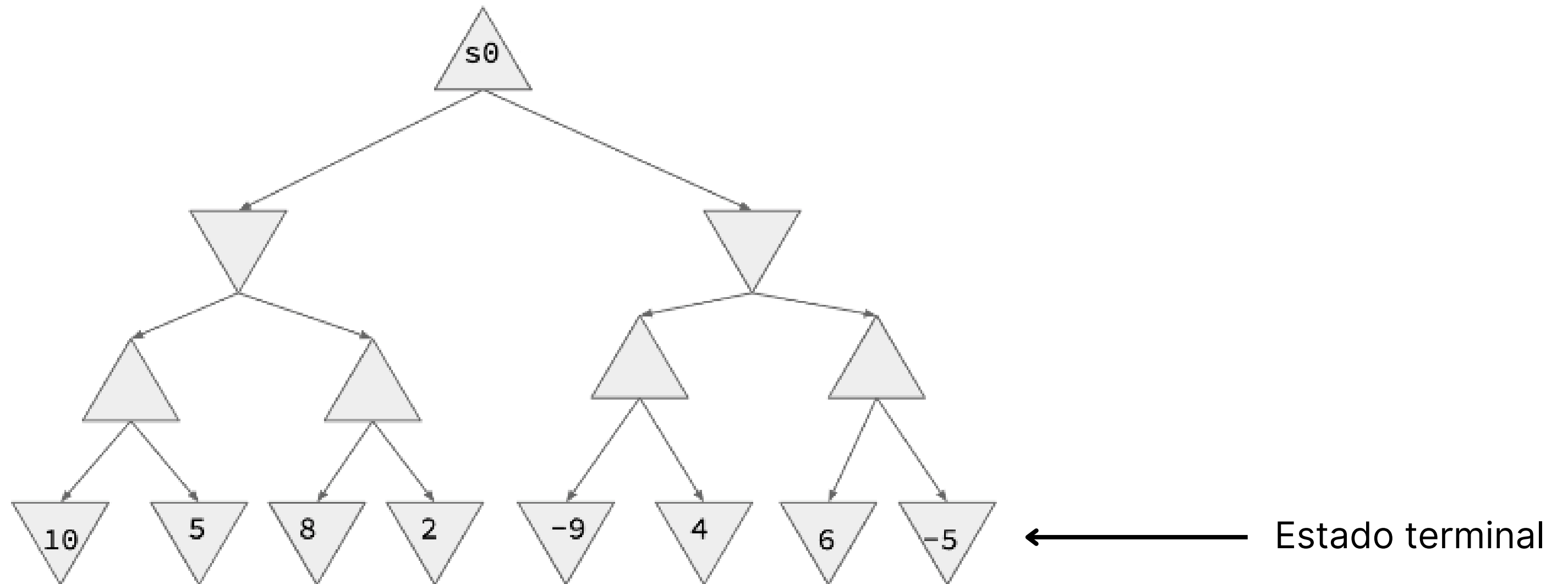
Cálculo del valor MiniMax



MINIMAX(s) =

$$\begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

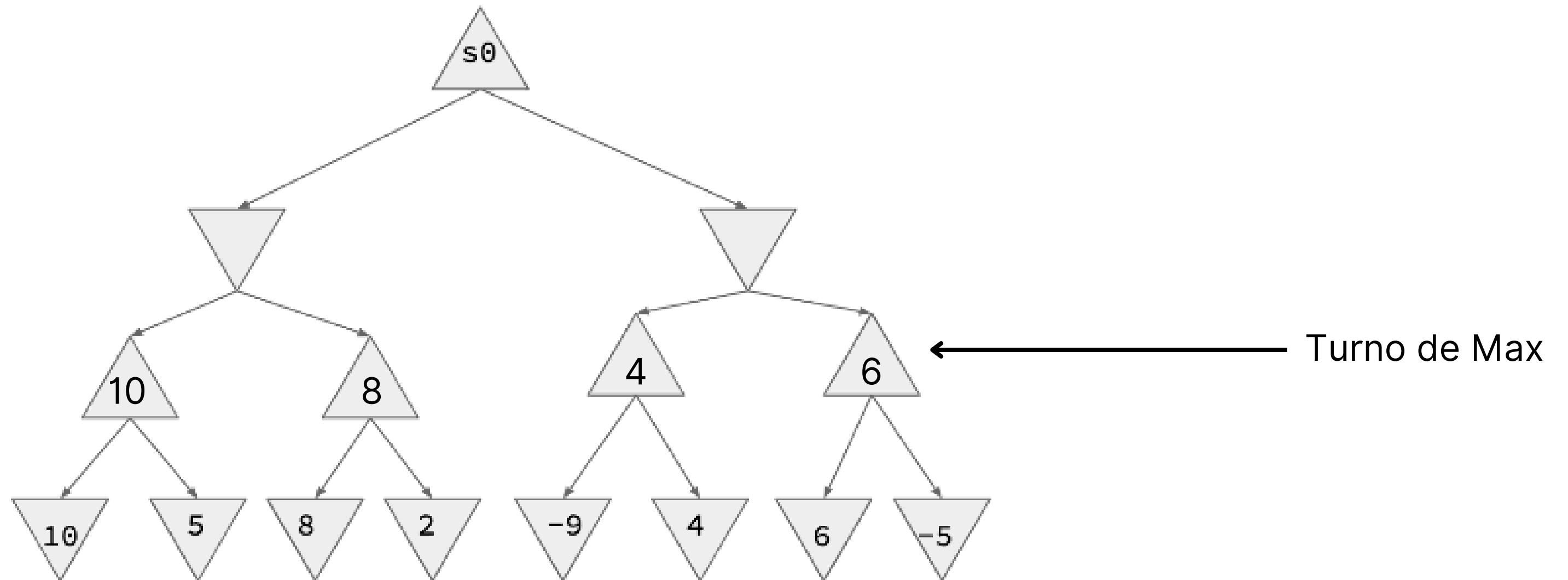
Cálculo del valor MiniMax



MINIMAX(s) =

$$\begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

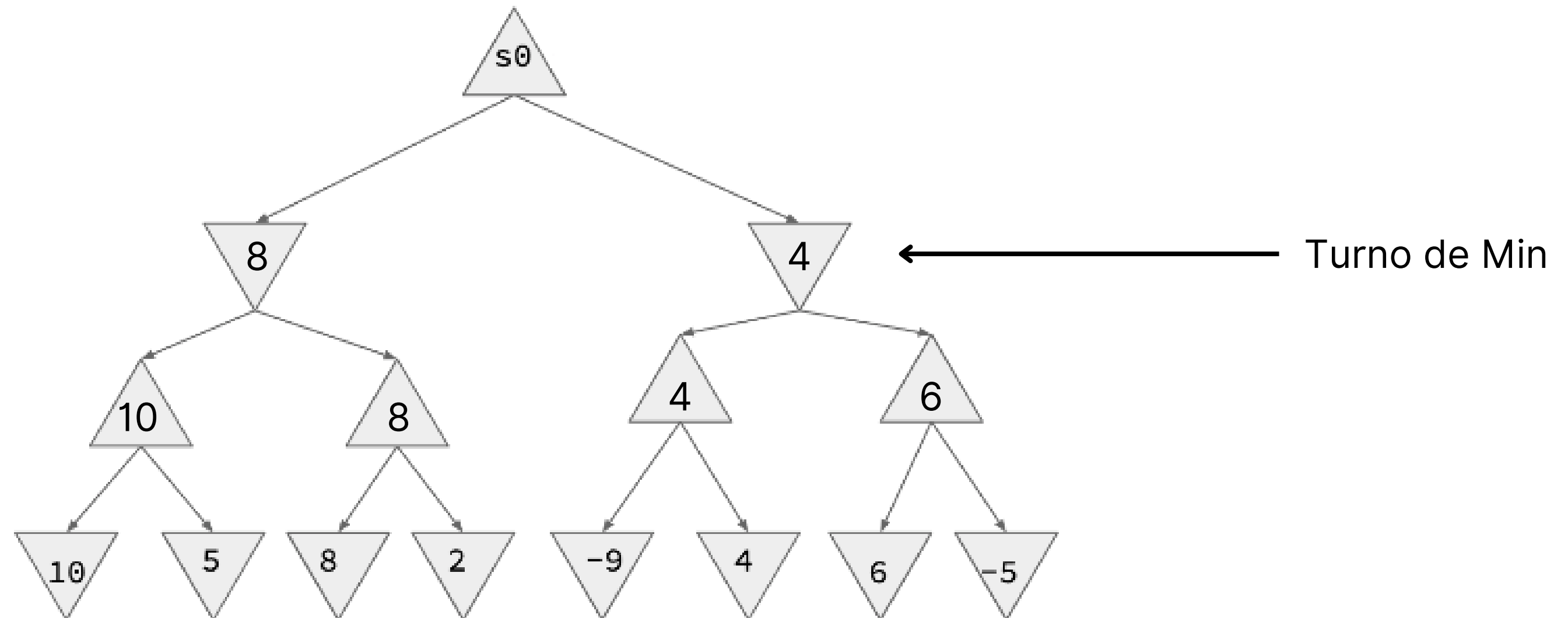
Cálculo del valor MiniMax



MINIMAX(s) =

$$\begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

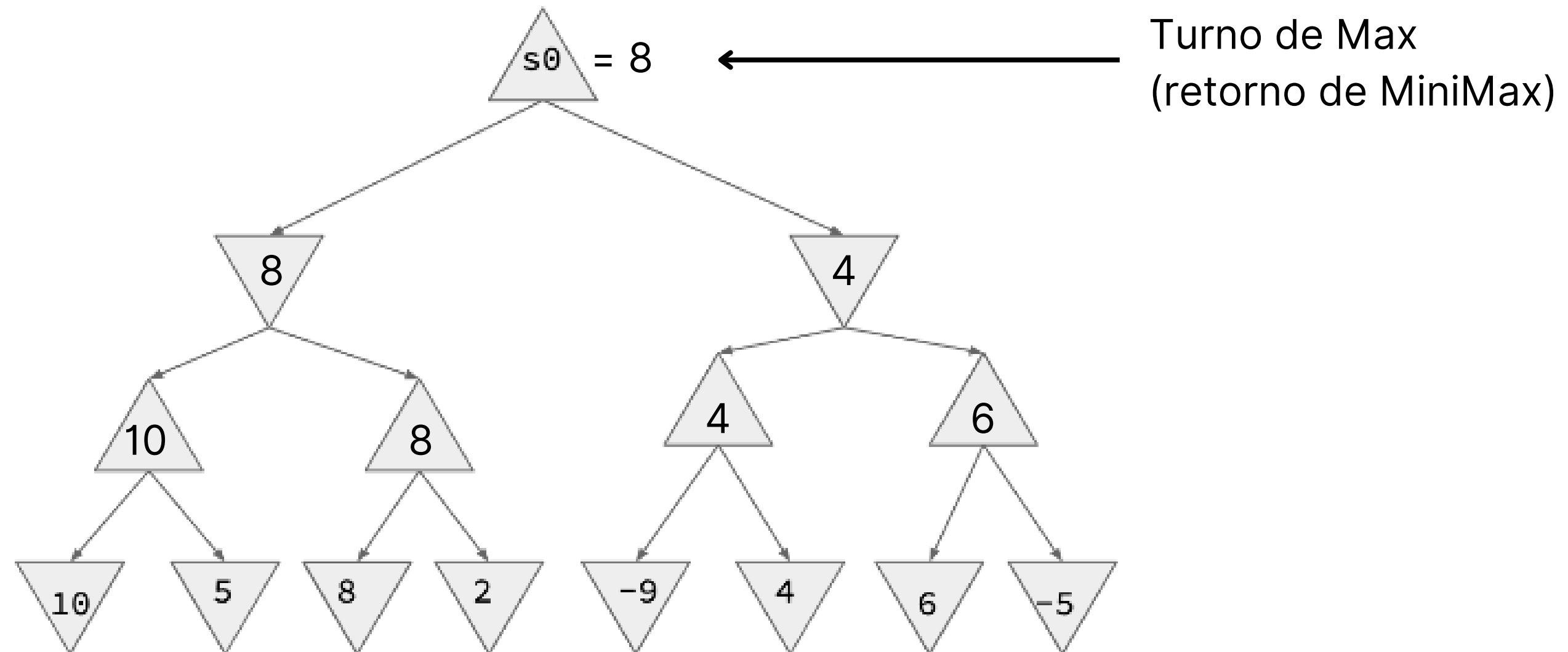
Cálculo del valor MiniMax



MINIMAX(s) =

$$\begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

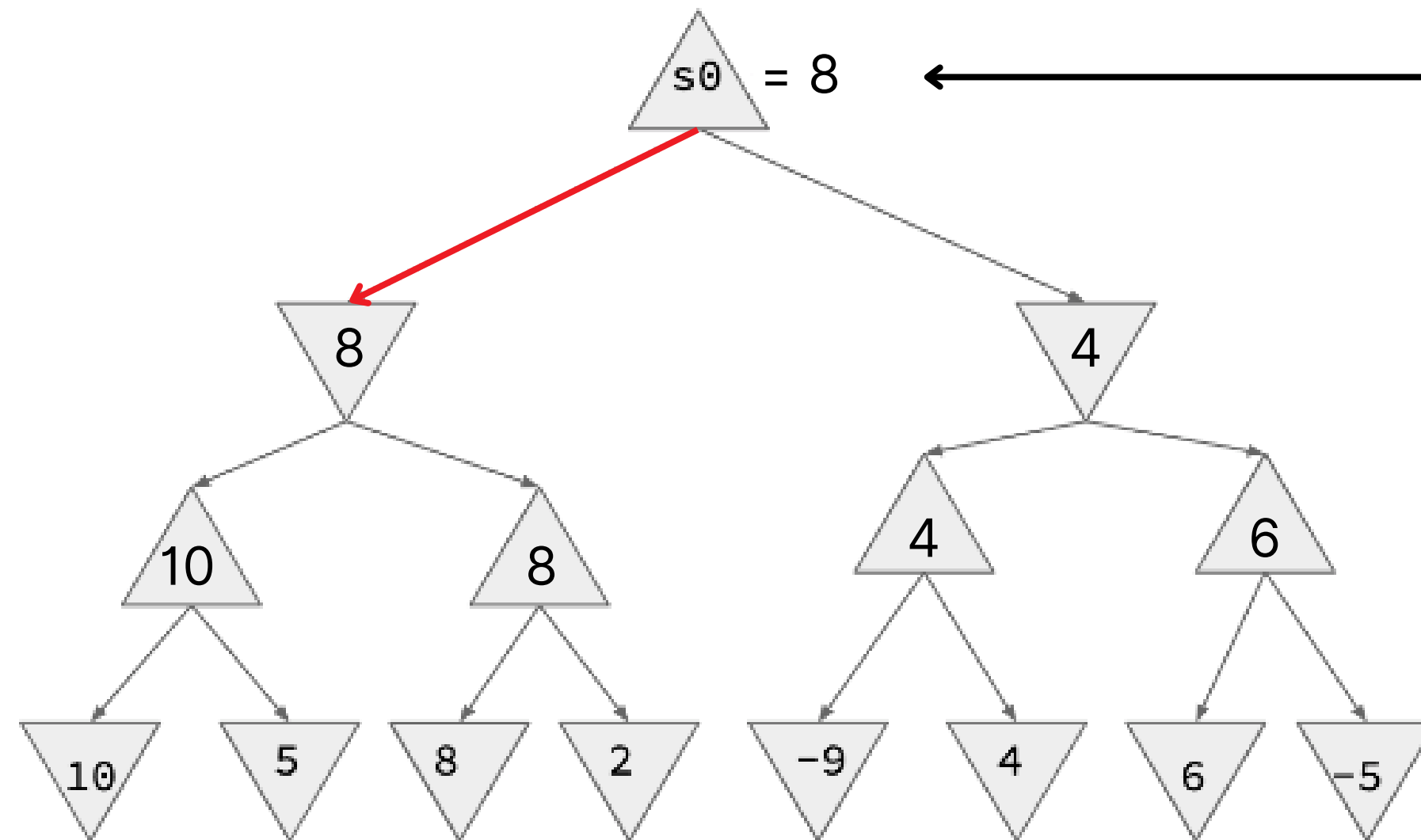
Cálculo del valor MiniMax



MINIMAX(s) =

$$\begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \leftarrow \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

Cálculo del valor MiniMax



Me traslado al estado asociado al 8 (hijo izquierdo)

MINIMAX(s) =

$$\begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$



Búsqueda MiniMax

Es un algoritmo que recibe un tablero y retorna la acción asociada al valor minimax de dicho tablero:

```
function MAX-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
   $v, move \leftarrow -\infty$ 
  for each a in game.ACTIONS(state) do
     $v2, a2 \leftarrow \text{MIN-VALUE}(\text{game}, \text{game.RESULT}(\text{state}, a))$ 
    if  $v2 > v$  then
       $v, move \leftarrow v2, a$ 
  return  $v, move$ 
```

```
function MIN-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
   $v, move \leftarrow +\infty$ 
  for each a in game.ACTIONS(state) do
     $v2, a2 \leftarrow \text{MAX-VALUE}(\text{game}, \text{game.RESULT}(\text{state}, a))$ 
    if  $v2 < v$  then
       $v, move \leftarrow v2, a$ 
  return  $v, move$ 
```

Rendimiento MiniMax

La búsqueda MiniMax corre DFS **sobre el árbol de estados completo:**

> Para el ajedrez, esto es explorar 10^{44} estados

Para mejorar el rendimiento podemos:

- Acotar la profundidad del árbol de búsqueda.
- Podar ramas del árbol de búsqueda.

Acotar profundidad del árbol de búsqueda

Función de evaluación

- Es como una **heurística** pero en el contexto de búsqueda adversaria
- **Asigna puntajes a estados no terminales**, lo que permite limitar a un máximo la altura de un árbol

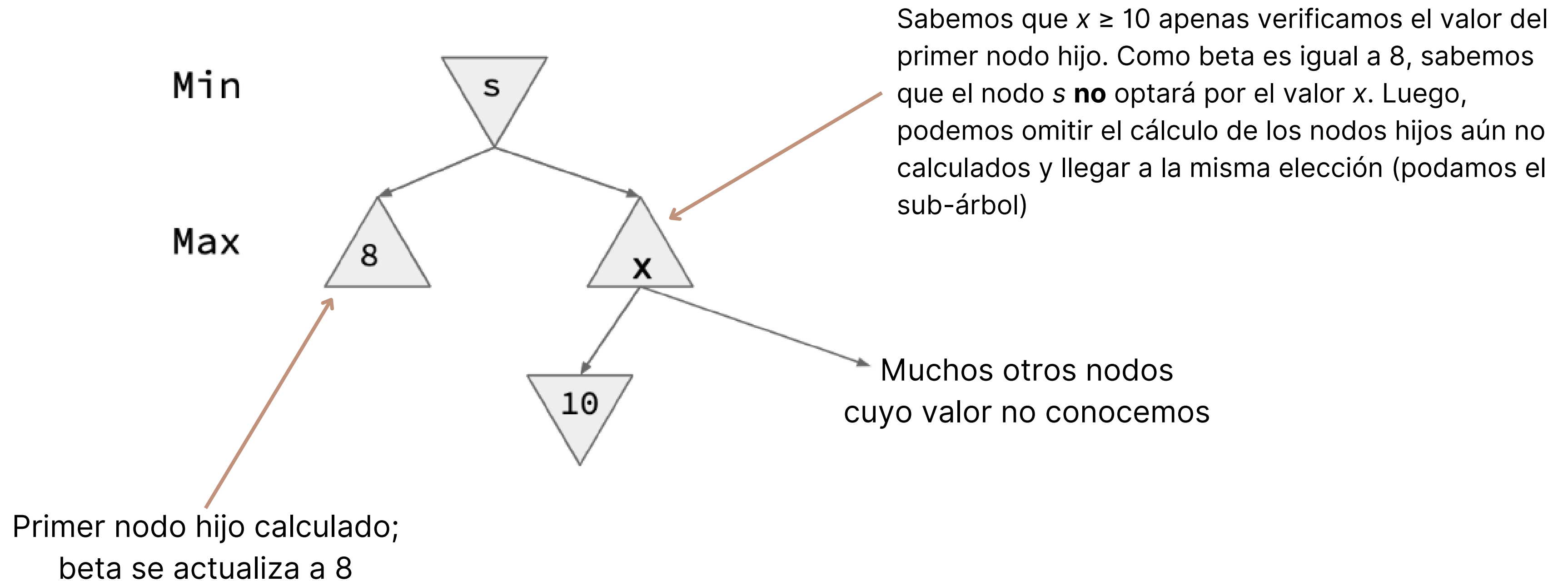
Poda Alpha-Beta

Podemos podar buena parte del árbol si guardamos dos parámetros adicionales en cada llamada a la búsqueda MiniMax

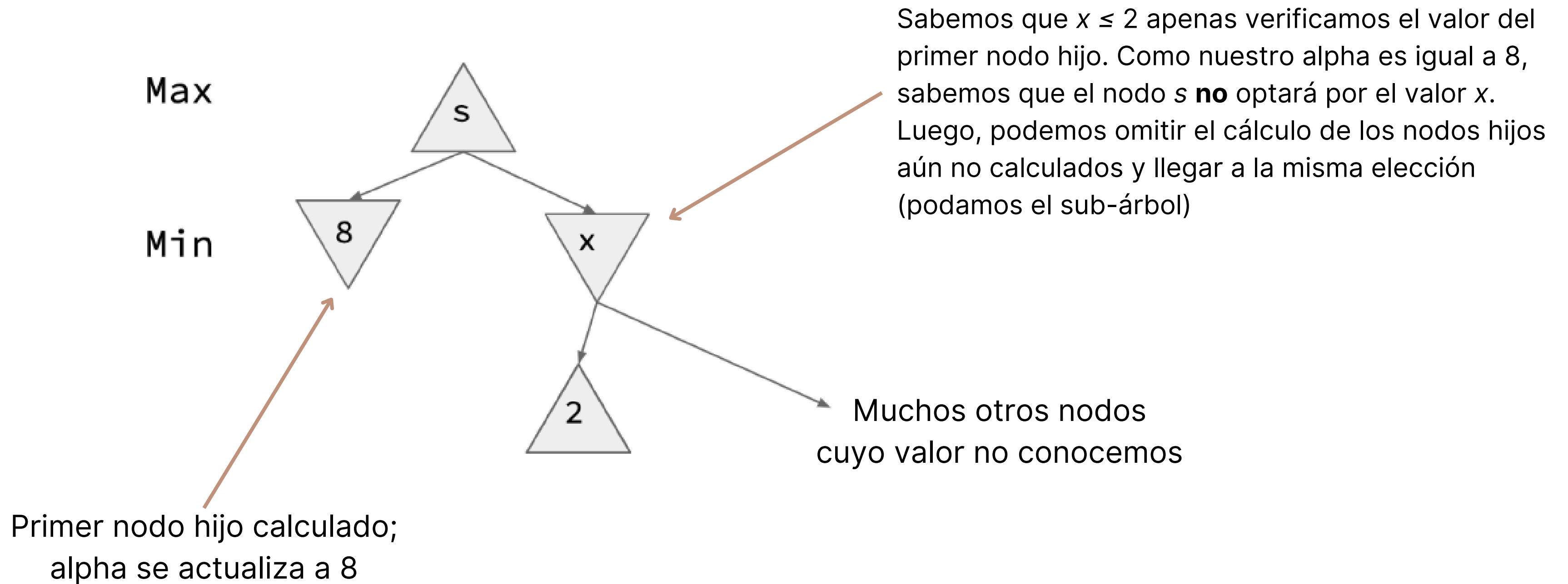
- alpha: cota **inferior** del valor de un nodo
- beta: cota **superior** del valor de un nodo

Tener estas cotas nos indica si es necesario calcular un nodo.

Poda Alpha-Beta - cota superior beta



Poda Alpha-Beta - cota inferior alpha



Poda Alpha-Beta

```
function MAX-VALUE(game, state,  $\alpha$ ,  $\beta$ ) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
   $v \leftarrow -\infty$ 
  for each a in game.ACTIONS(state) do
     $v2, a2 \leftarrow$  MIN-VALUE(game, game.RESULT(state, a),  $\alpha$ ,  $\beta$ )
    if  $v2 > v$  then
       $v, move \leftarrow v2, a$ 
       $\alpha \leftarrow$  MAX( $\alpha$ ,  $v$ )
    if  $v \geq \beta$  then return  $v, move$ 
  return  $v, move$ 
```

```
function MIN-VALUE(game, state,  $\alpha$ ,  $\beta$ ) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
   $v \leftarrow +\infty$ 
  for each a in game.ACTIONS(state) do
     $v2, a2 \leftarrow$  MAX-VALUE(game, game.RESULT(state, a),  $\alpha$ ,  $\beta$ )
    if  $v2 < v$  then
       $v, move \leftarrow v2, a$ 
       $\beta \leftarrow$  MIN( $\beta$ ,  $v$ )
    if  $v \leq \alpha$  then return  $v, move$ 
  return  $v, move$ 
```

FIN

Nos pueden preguntar cualquier duda xd