

# AYUDANTÍA 6: HEURÍSTICAS Y BÚSQUEDA ADVERSARIA

**Benjamín Pizarro – Tomás Couso**

HEURÍSTICAS

# ¿QUÉ ES UNA HEURÍSTICA?



## Algoritmo A\*

**Input:** Un problema de búsqueda ( $S, A, s_0, G$ )

**Output:** Un nodo objetivo

- 1 **for each**  $s \in S$  **do**  $g(s) \leftarrow \infty$
- 2  $Open \leftarrow \{s_0\}$
- 3  $g(s_0) \leftarrow 0$ ;  $f(s_0) \leftarrow h(s_0)$
- 4 **while**  $Open \neq \emptyset$
- 5   Extrae un  $u$  desde  $Open$  con menor valor- $f$
- 6   **if**  $u$  es objetivo **return**  $u$
- 7   **for each**  $v \in Succ(u)$  **do**
- 8     Insertar  $v$

### Insertar $v$ en $Open$

- 1  $cost_v = g(u) + c(u, v)$                    // el costo de llegar a  $v$  por  $u$
- 2 **if**  $cost_v \geq g(v)$  **return**               // seguimos solo si  $cost_v < g(v)$
- 3  $parent(v) \leftarrow u$
- 4  $g(v) \leftarrow cost_v$
- 5  $f(v) \leftarrow g(v) + h(v)$
- 6 **if**  $v \in Open$  **then** Reordenar  $Open$    // depende de la impl.
- 7 **else** Insertar  $v$  en  $Open$

# ¿QUÉ ES UNA HEURÍSTICA?

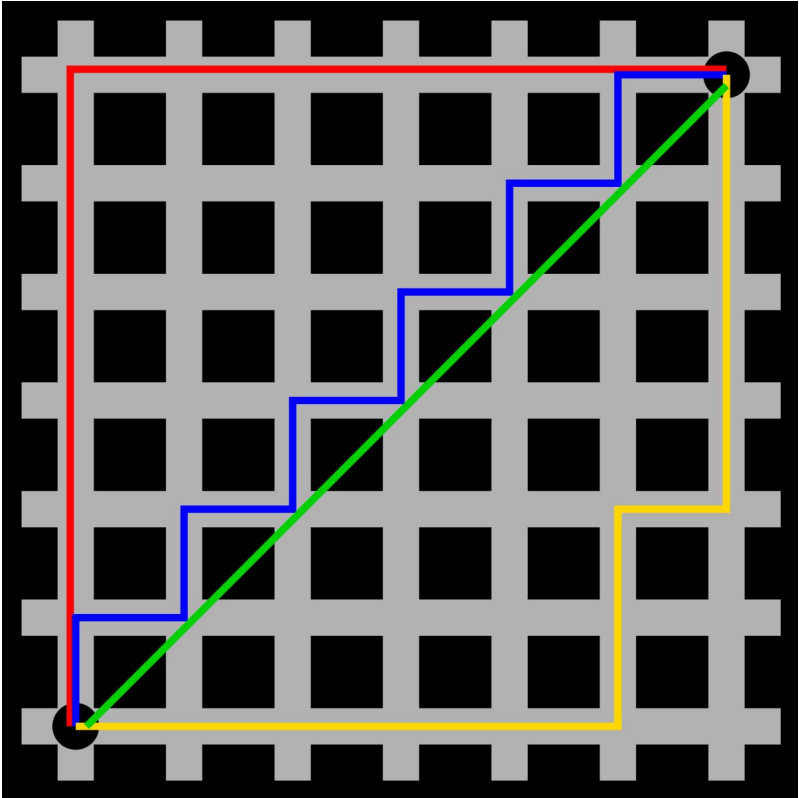
- La idea es utilizar **conocimiento previo** para discriminar qué estados son “mejores” o “peores”.
- Así, se busca **reducir el tiempo de búsqueda** y no irse por caminos que probablemente no son buenos.
- Se busca **estimar** qué tan cerca estamos de la solución.

# Y FORMALMENTE...

- Dado un problema de búsqueda  $(S, A, s\_INIT, G)$
- Una heurística es una función  $h$ , que recibe como entrada un nodo  $(s)$  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  $g$ .

# EJEMPLOS DE HEURÍSTICAS

- No comprar un producto del que escuchamos malas opiniones.
- Llevar paraguas cuando el día está nublado.
- Elegir un producto porque su envase se ve de mejor calidad.
- Distancia Euleriana
- Distancia de Manhattan



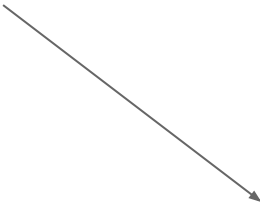


# ADMISIBILIDAD

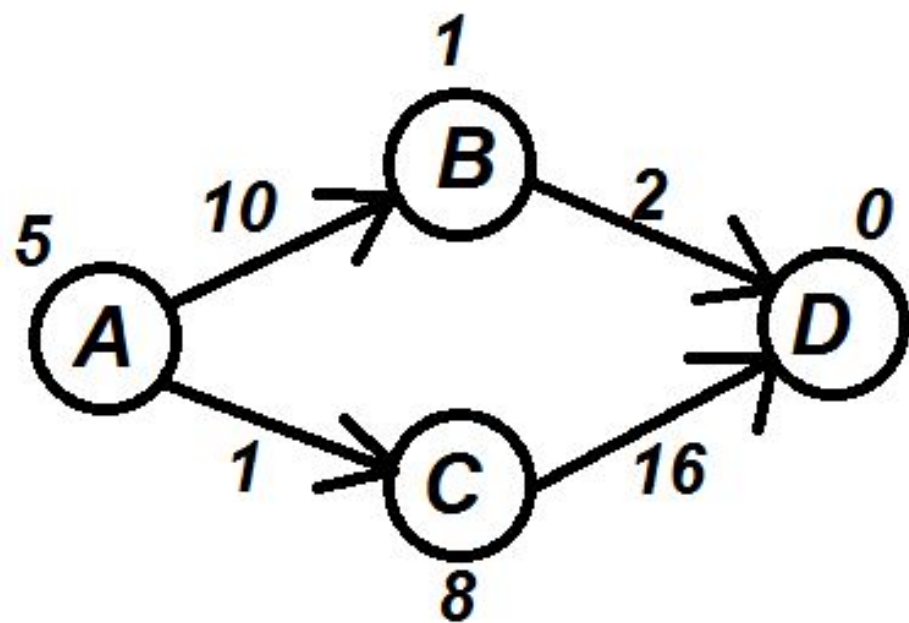
Una heurística  $h$  se dice **admisible** si y solo si **nunca sobre-estima respecto a un camino óptimo**. Es decir:

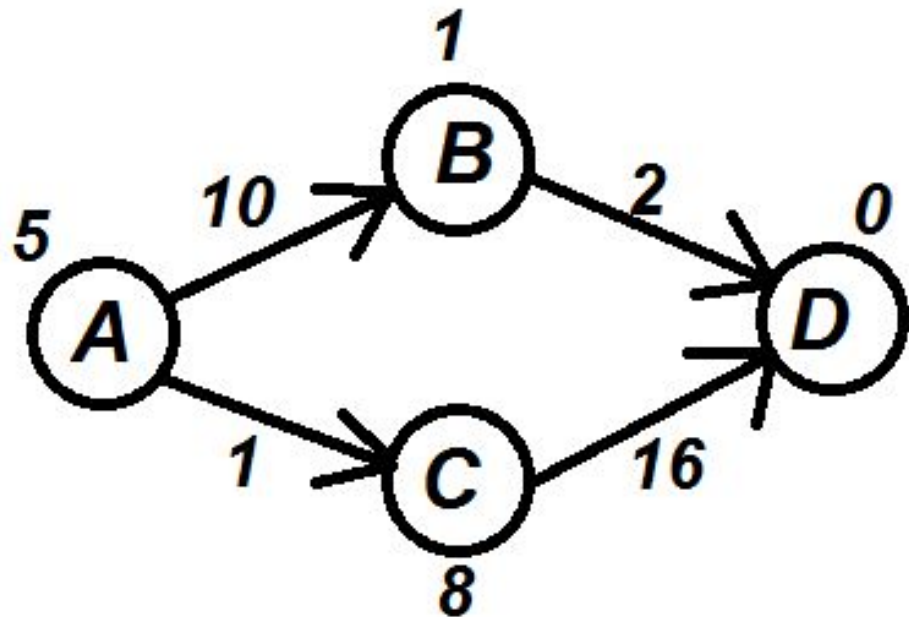
$$h(s) \leq h^*(s)$$

**para todo estado  $s$ .**



costo de un camino  
óptimo desde  $s$  a una  
solución





Admissible:

$$h(A) \leq h^*(A) = 12$$

$$h(B) \leq 2$$

$$h(C) \leq 16$$

# EJEMPLO

Distancia Manhattan en el puzzle de 8.

2	8	3
1	6	4
7		5

# CONSISTENCIA

Una heurística  $h$  se dice consistente si y sólo si:

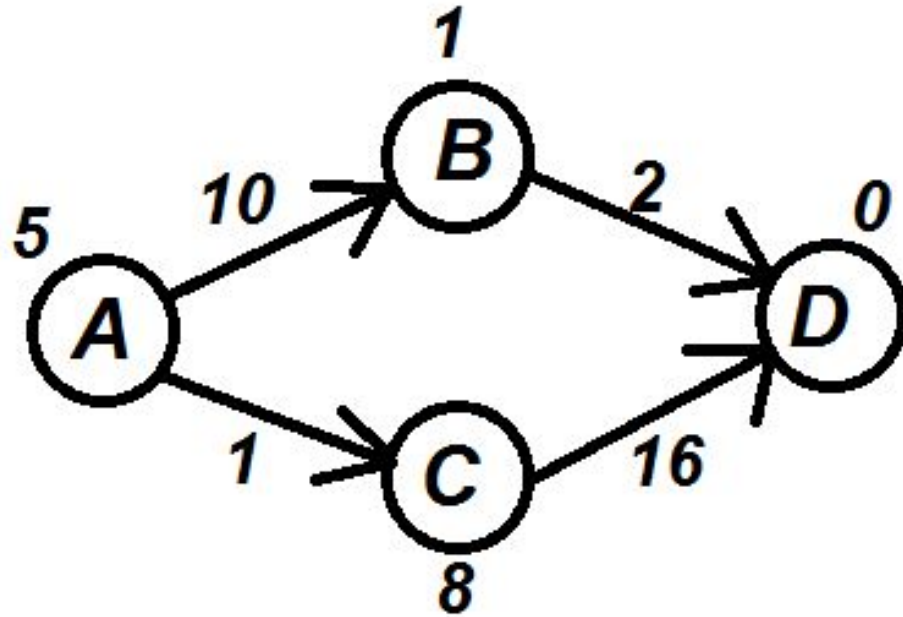
- $h(s) = 0$ , para todo  $s \in G$

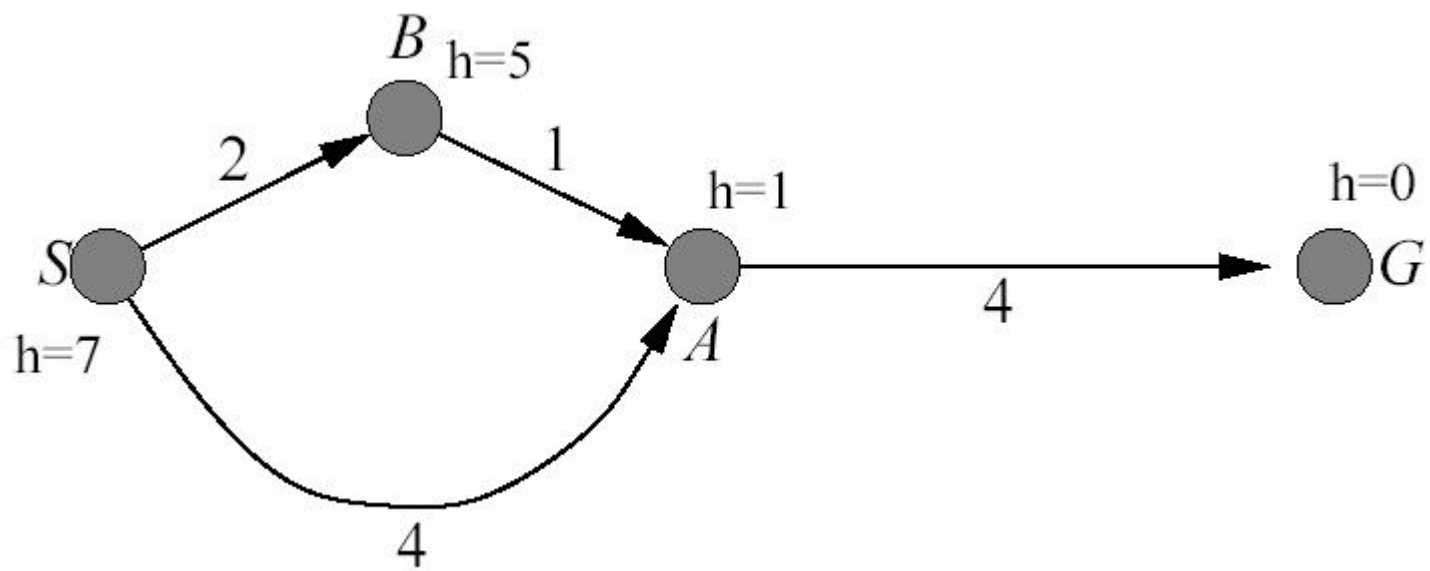
**No se sobre-estima  
en los objetivos**

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

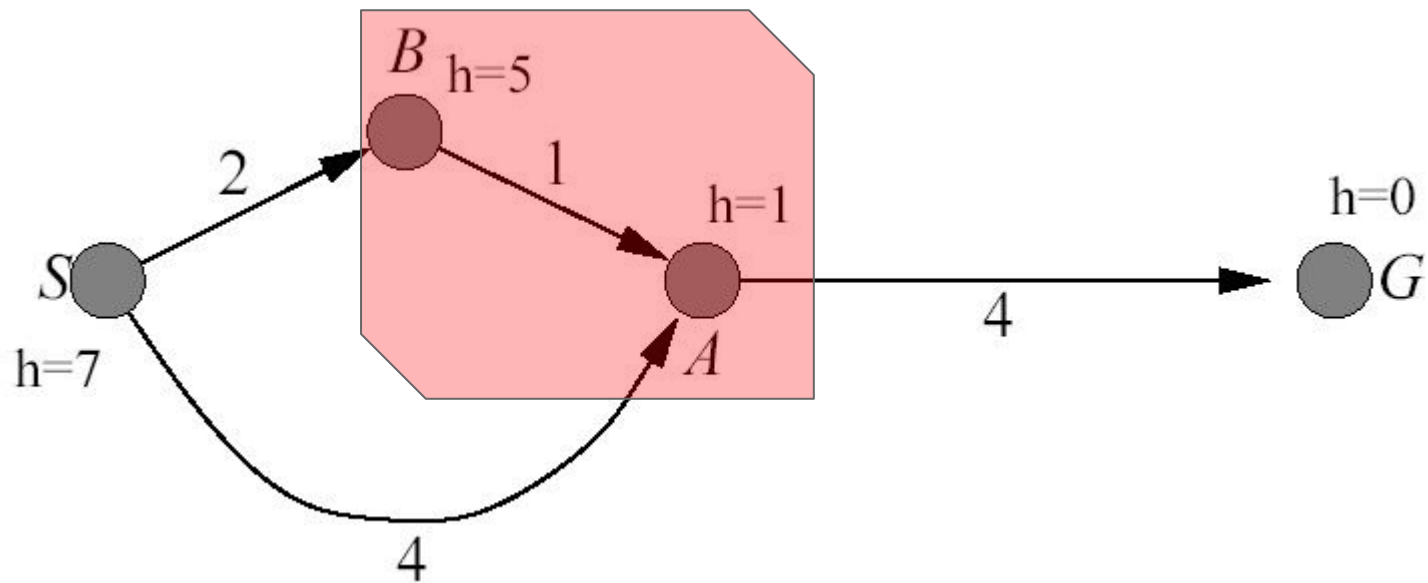
**La variación del  $h$  de  
dos estados no es  
mayor que el costo  
de ir de uno al otro**

CONSISTENTE





# INCONSISTENTE





# IMPORTANTE

- Una heurística **consistente implica que es admisible** (¿por qué?).
- El algoritmo **A\*** siempre entrega soluciones óptimas si se usa con una heurística admisible.

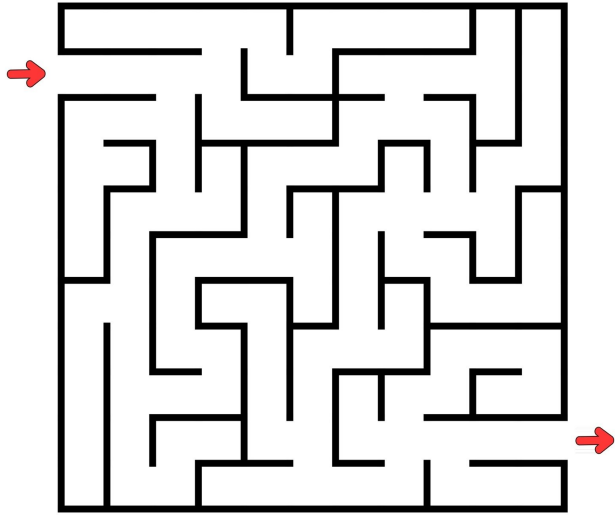
EL ALGORITMO A\*  
SIEMPRE ENTREGA  
SOLUCIONES ÓPTIMAS  
SI SE USA CON UNA  
HEURÍSTICA ADMISIBLE.

# UNA TÉCNICA: RELAJACIÓN DE PROBLEMAS

- Obtener un supergrafo de búsqueda que contenga al original, y encontrar el **costo óptimo en este supergrafo**.
- Ese costo óptimo nos servirá como **heurística del problema original**.

2	8	3
1	6	4
7		5

# HEURÍSTICA DE RELAJACIÓN



Si se pudieran atravesar las paredes, el costo óptimo estaría dado por la distancia de Manhattan

VEAMOS UN POQUITO  
DE CÓDIGO

# BÚSQUEDA CON ADVERSARIO

# JUEGOS

Tratamos con juegos de dos jugadores, turnos, información perfecta y suma cero.

- Información perfecta: Tenemos visualización completa del tablero de juego
- Suma cero: Lo que es bueno para mí es en igual medida malo para mi oponente



# BÚSQUEDA

- Podemos considerar un juego como un espacio de búsqueda:
  - Tablero: nodo
  - Jugadas: conecciones
  -
- El objetivo será llegar a un tablero donde ganemos:
  - 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 de mi adversario.

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

- Si es mi turno: **Max**
  - Elijo la mejor jugada que tengo disponible, de modo que **maximizamos** el puntaje.
- Si es el turno de mi oponente: **Min**
  - Asumimos que el oponente tiene decisiones óptimas, y elegimos la peor jugada que tenemos disponible, de modo que **minimizamos**.

# ¿CÓMO DEFINIMOS LA MEJOR JUGADA DESDE UN TABLERO?

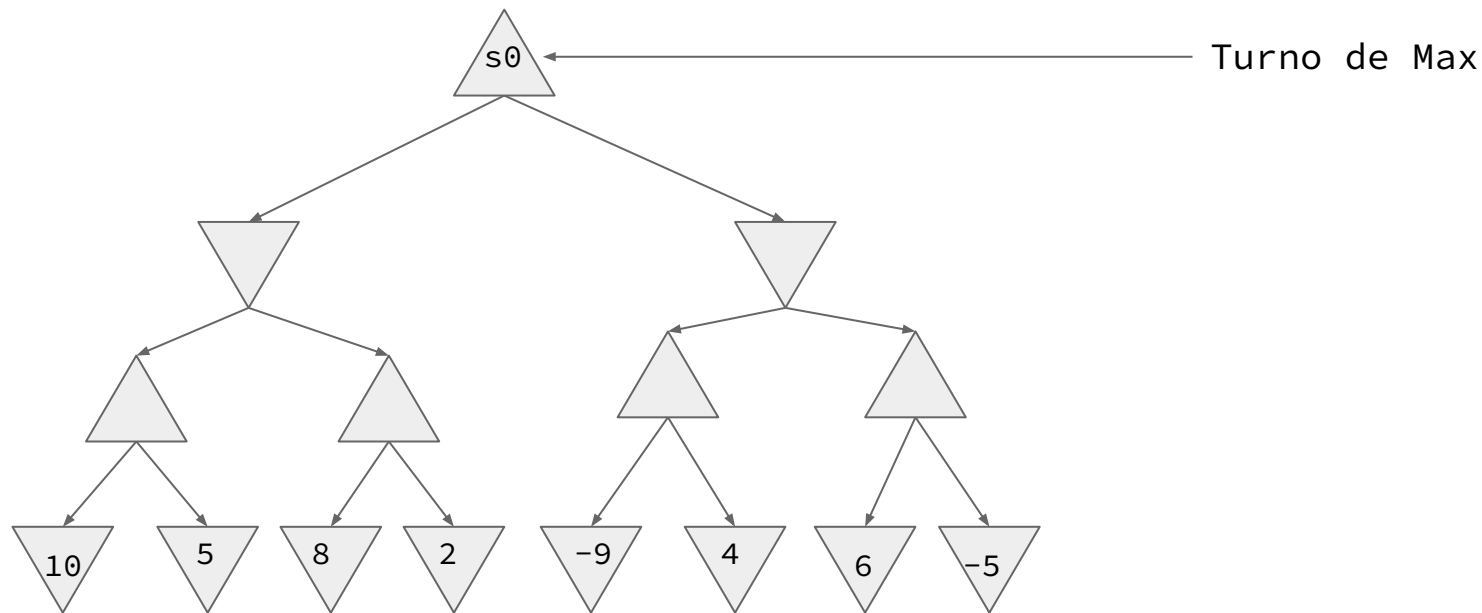
- Puntuamos tablero en base a valor minimax

valor del tablero para Max

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

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

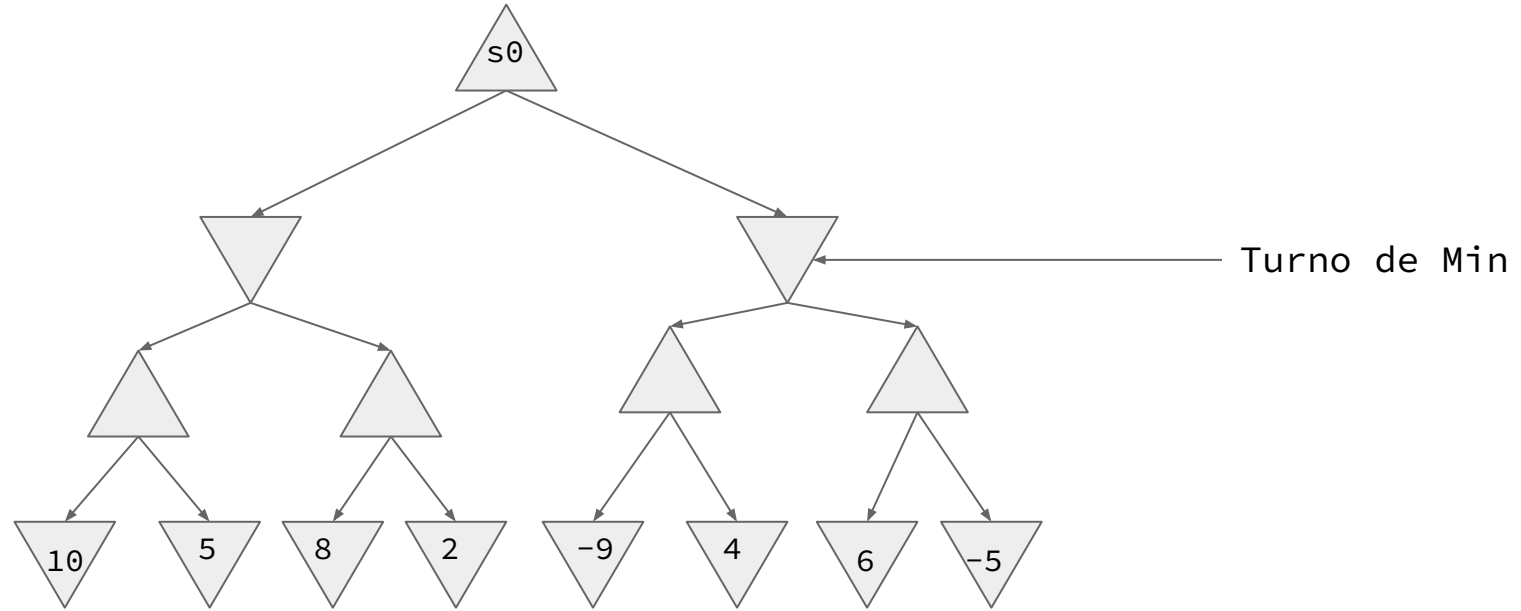
# CÁLCULO DE VALOR MINIMAX



$\text{MINIMAX}(s) =$

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

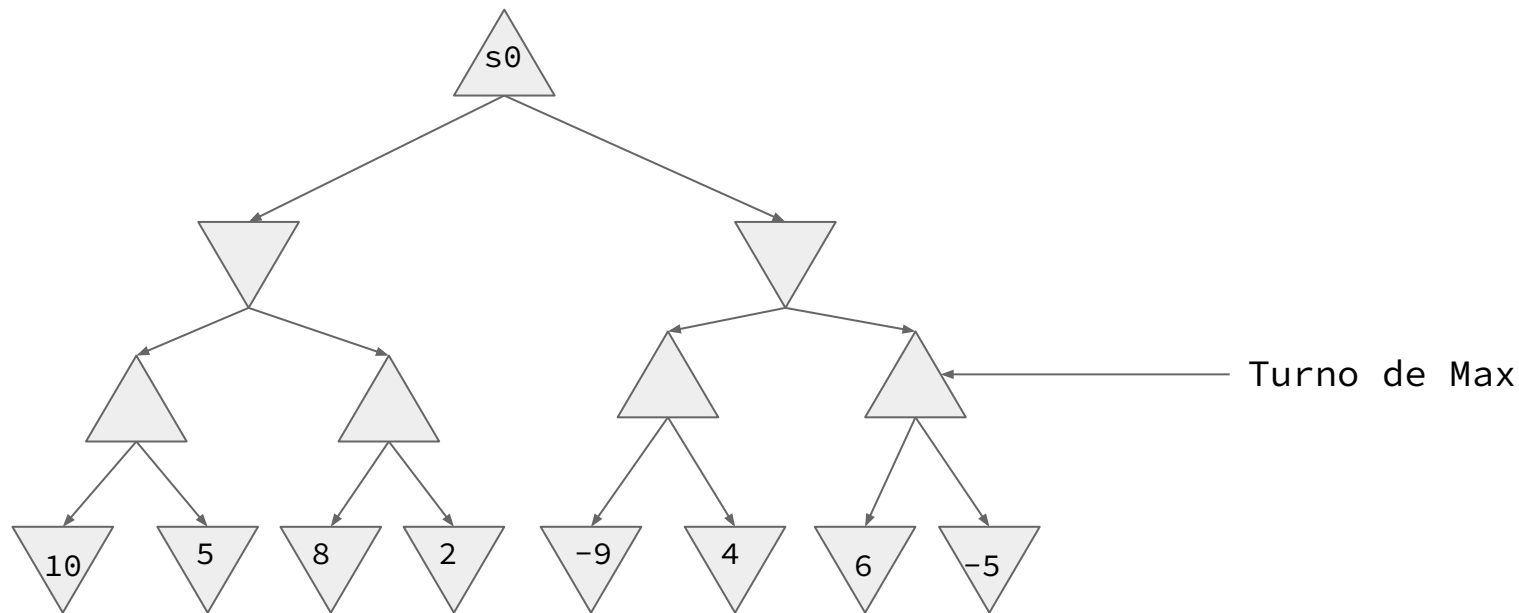
# CÁLCULO DE VALOR MINIMAX



$\text{MINIMAX}(s) =$

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

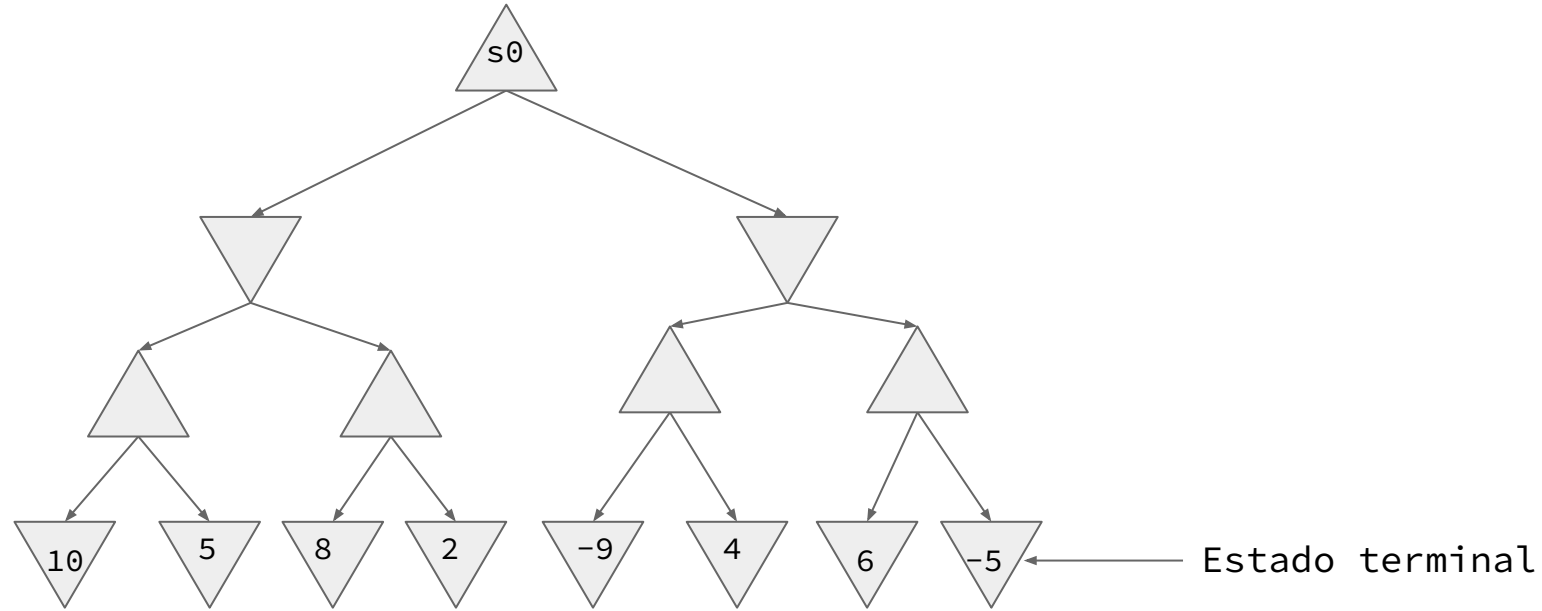
# CÁLCULO DE VALOR MINIMAX



MINIMAX( $s$ ) =

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

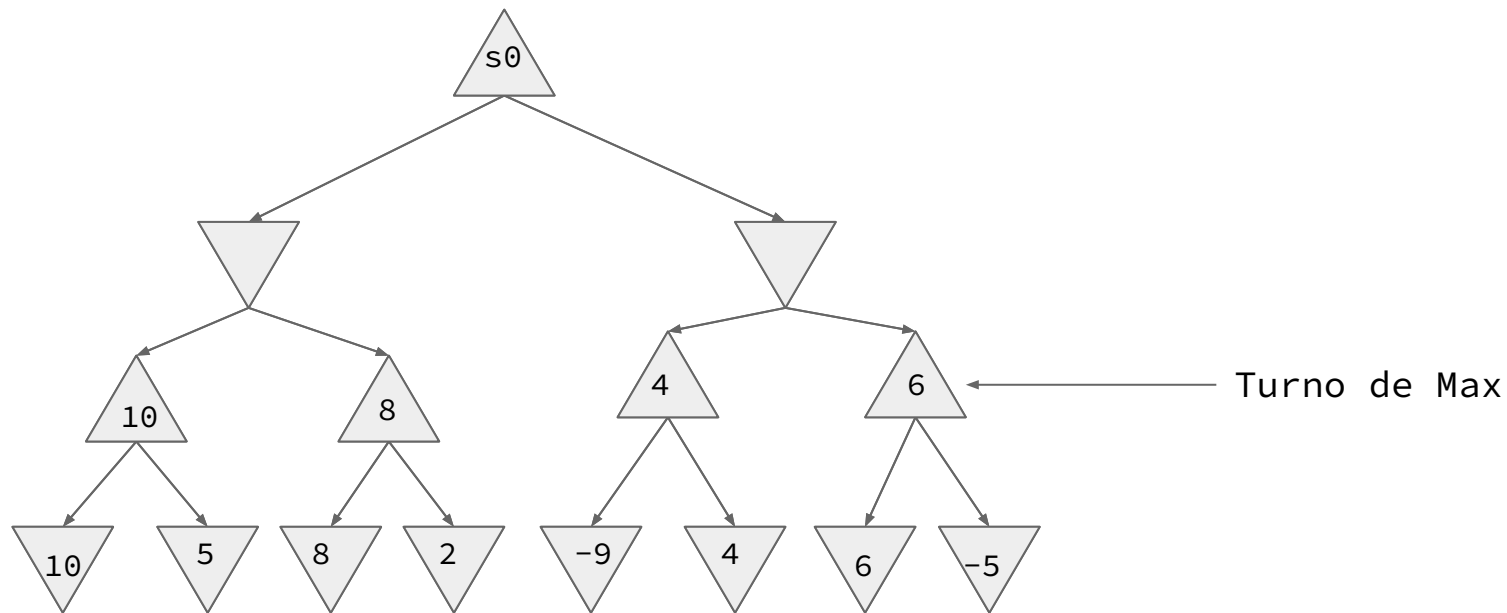
# CÁLCULO DE VALOR MINIMAX



MINIMAX( $s$ ) =

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

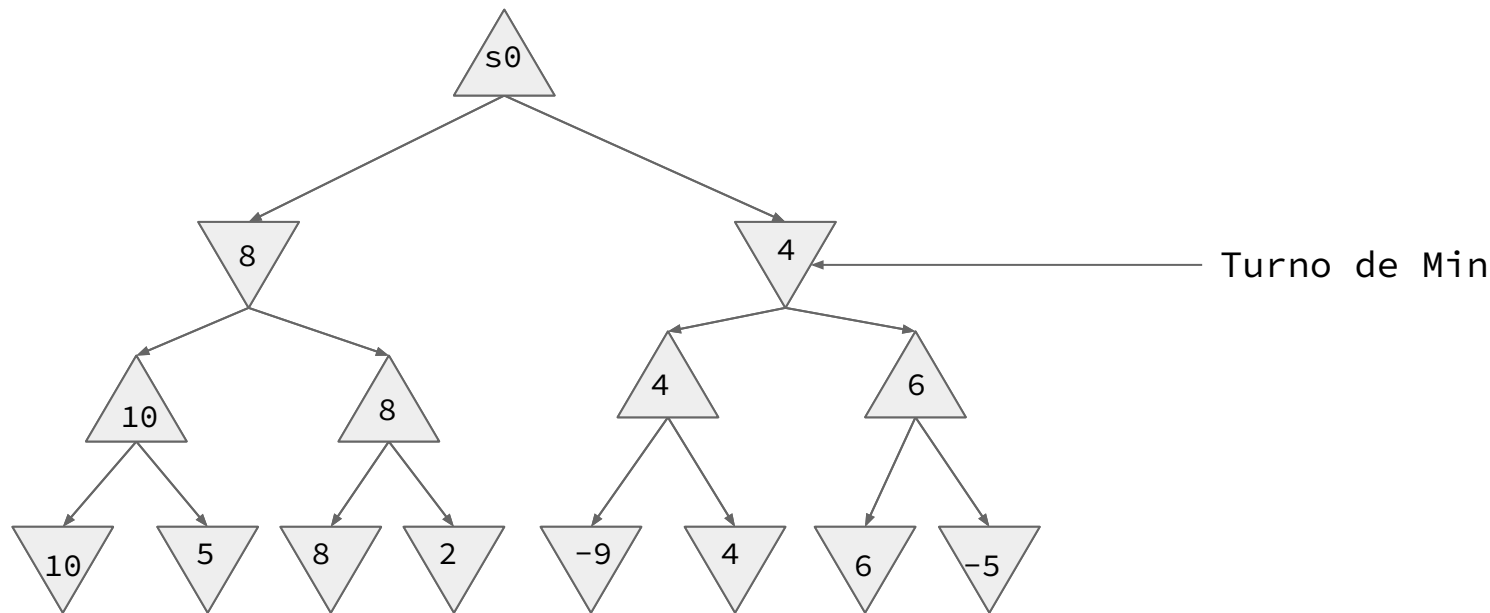
# CÁLCULO DE VALOR MINIMAX



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 DE VALOR MINIMAX

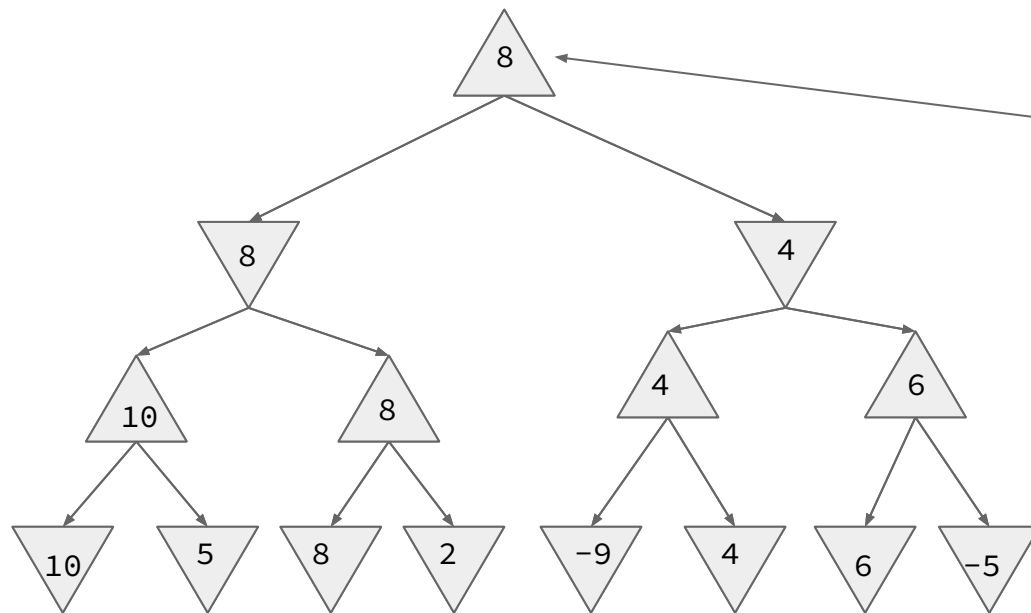


MINIMAX( $s$ ) =

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



# CÁLCULO DE VALOR MINIMAX



Retorno de minimax:  
Nos trasladamos al  
estado asociado al  
8 (hijo izquierdo)

MINIMAX( $s$ ) =

$$\begin{cases} \text{UTILITY}(s, \text{MAX}) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) \end{cases}$$

if IS-TERMINAL( $s$ )

if TO-MOVE( $s$ ) = MAX ←

if TO-MOVE( $s$ ) = MIN

# BÚSQUEDA MINIMAX

- 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  $\leftarrow$  MIN-VALUE(game, game.RESULT(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  $\leftarrow$  MAX-VALUE(game, game.RESULT(state, a))
    if v2 < v then
      v, move  $\leftarrow$  v2, a
  return v, move
```

# ¿CÓMO MEJORAMOS EL RENDIMIENTO DE ESTO?

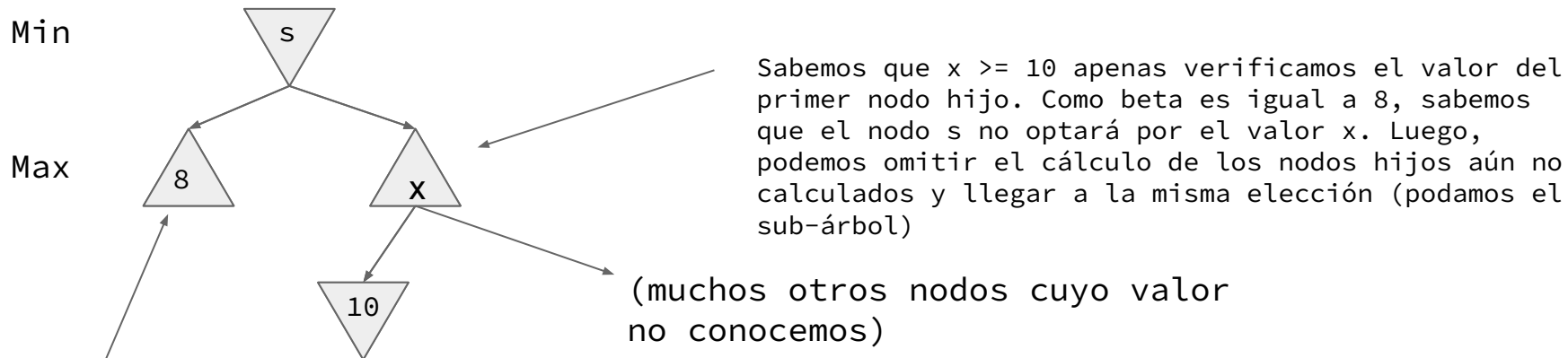
- Búsqueda minimax corre DFS sobre el árbol de estados completo:
  - Para el ajedrez, esto es explorar  $10^{44}$  estados
- Una opción es podar ramas del árbol de búsqueda
- Otra es acotar la profundidad del árbol de búsqueda

# PODA ALPHA BETA

- Podemos podar buena parte del árbol si guardamos dos parámetros adicionales en cada llamada a búsqueda minimax
  - alpha: Cota inferior del valor de un nodo.
  - beta: Cota superior del valor de un nodo.
- Tener cotas nos indica si es necesario calcular un nodo

# EJEMPLO PODA ALFA - BETA: COTA SUPERIOR

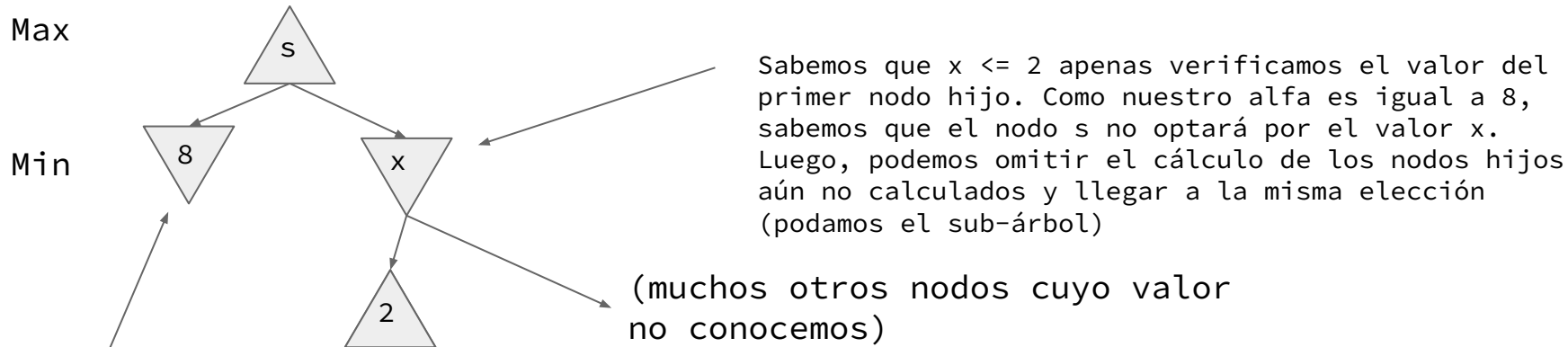
Cota superior beta



Primer nodo hijo  
calculado; beta se  
actualiza a 8

# EJEMPLO PODA ALFA - BETA: COTA INFERIOR

Cota superior beta



Primer nodo hijo  
calculado; alfa se  
actualiza a 8

# 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  $v, move$ 
```

# FUNCIÓN DE EVALUACIÓN

- 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



VEAMOS UN POQUITO  
DE CÓDIGO