Ayudantia 6

Búsqueda adversaria

Martín Castillo y Carlos Stappung

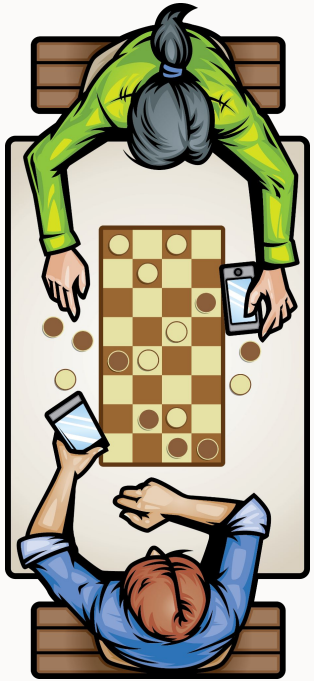
27 de septiembre

Problemas a desarrollar (Juegos)

Problemas de dos jugadores, de turnos, **información perfecta** y **suma cero**.

- **Información perfecta:** Conocemos toda la información del tablero de juego (estados).

- **Suma cero:** Todo lo que me beneficia a mi, perjudica a mi contrincante en igual medida y vice versa.



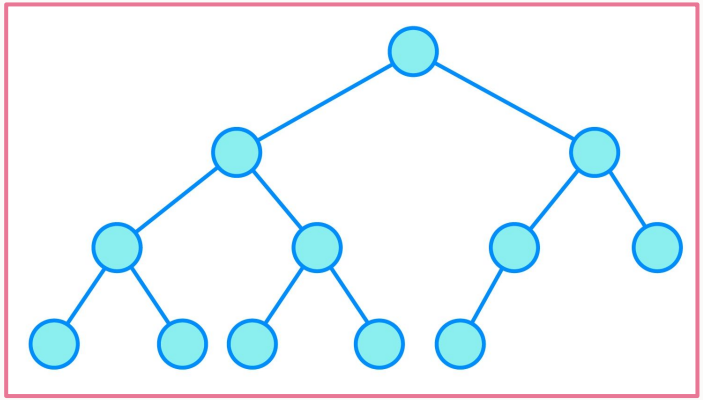
Búsqueda

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

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

El **objetivo** será 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

Definición

Yo **no sé** cómo va a jugar mi adversario, pero si **asumo que es un genio** (siempre juega lo que es mejor para él/peor para mí) estaré preparada para lo peor.

Luego si mi adversario toma decisiones subóptimas durante el juego, sólo es mejor para mí.

Algoritmo Minimax

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 **max**imizar el puntaje.

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

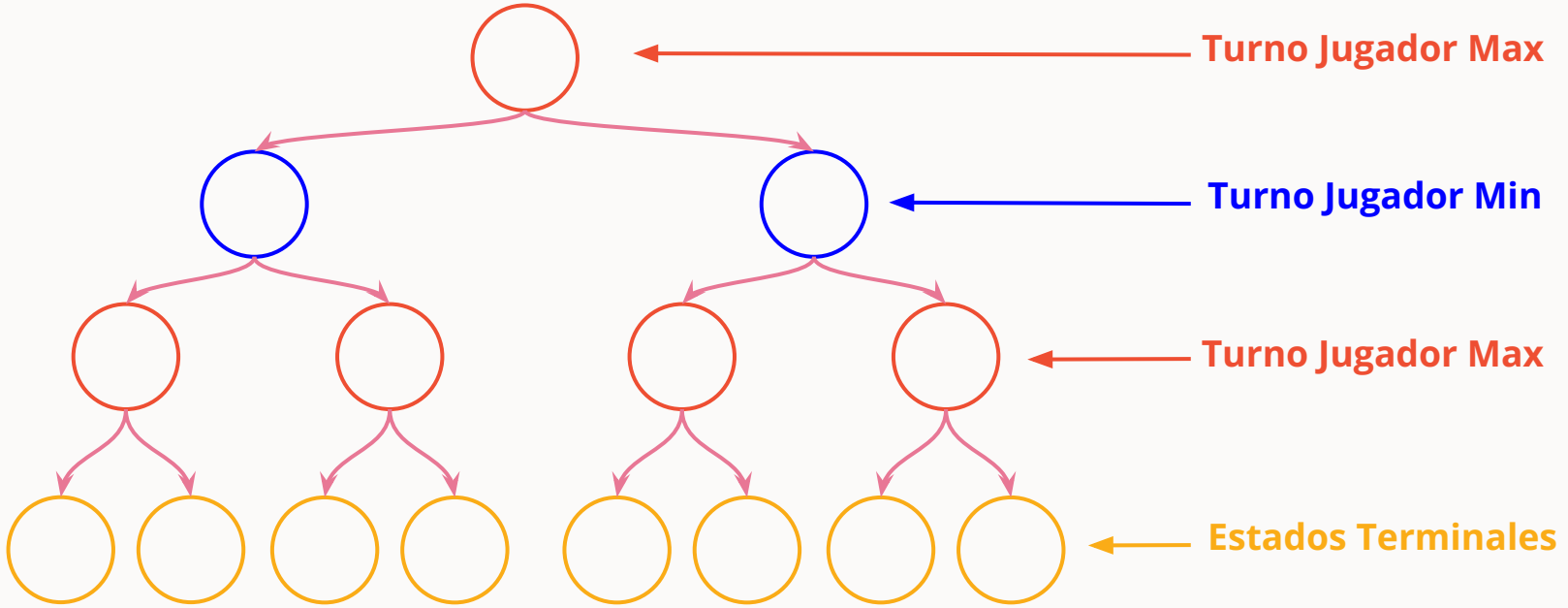
- Asumo que mi oponente siempre toma las decisiones óptimas (para él) entonces elige la opción que **min**imiza mi puntaje (la mejor jugada para mi oponente es la que me “hace peor”).

Búsqueda MiniMax

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

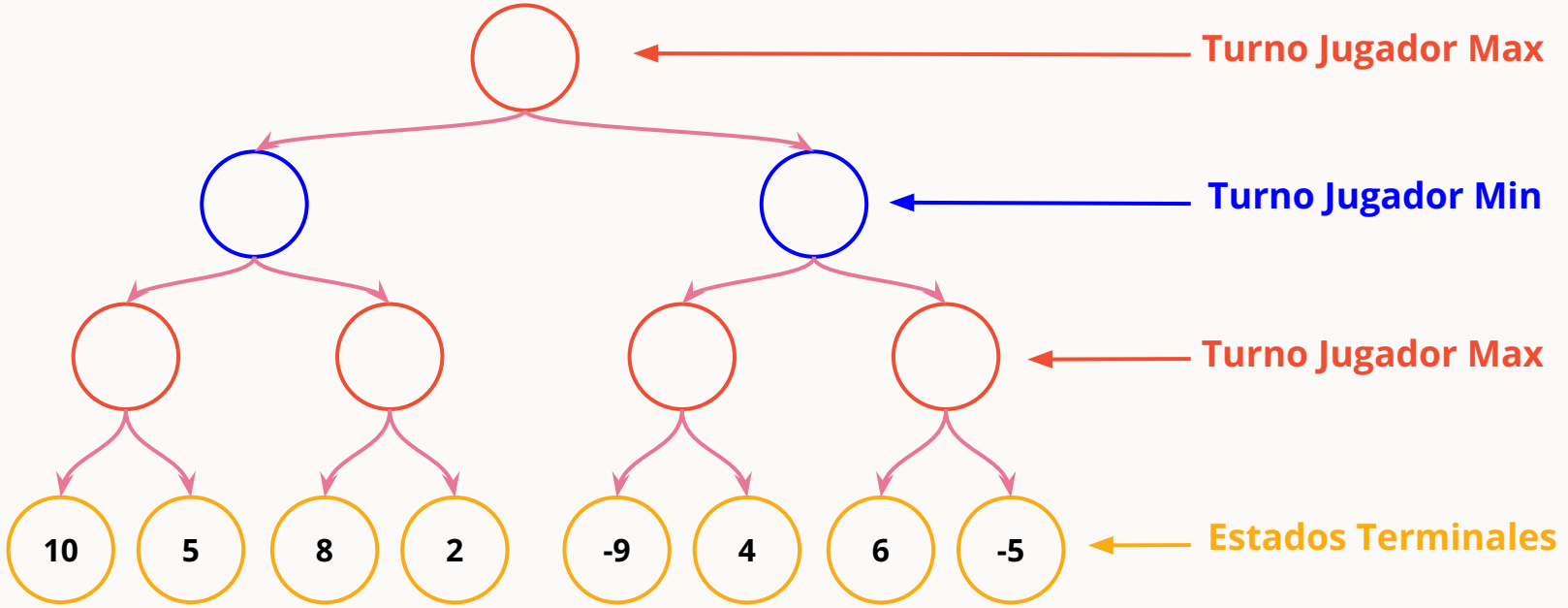
```
function MAX-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v, move ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
    if v2 > v then
      v, move ← v2, a
  return v, 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 ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
    if v2 < v then
      v, move ← v2, a
  return v, move
```

Cálculo del valor MiniMax



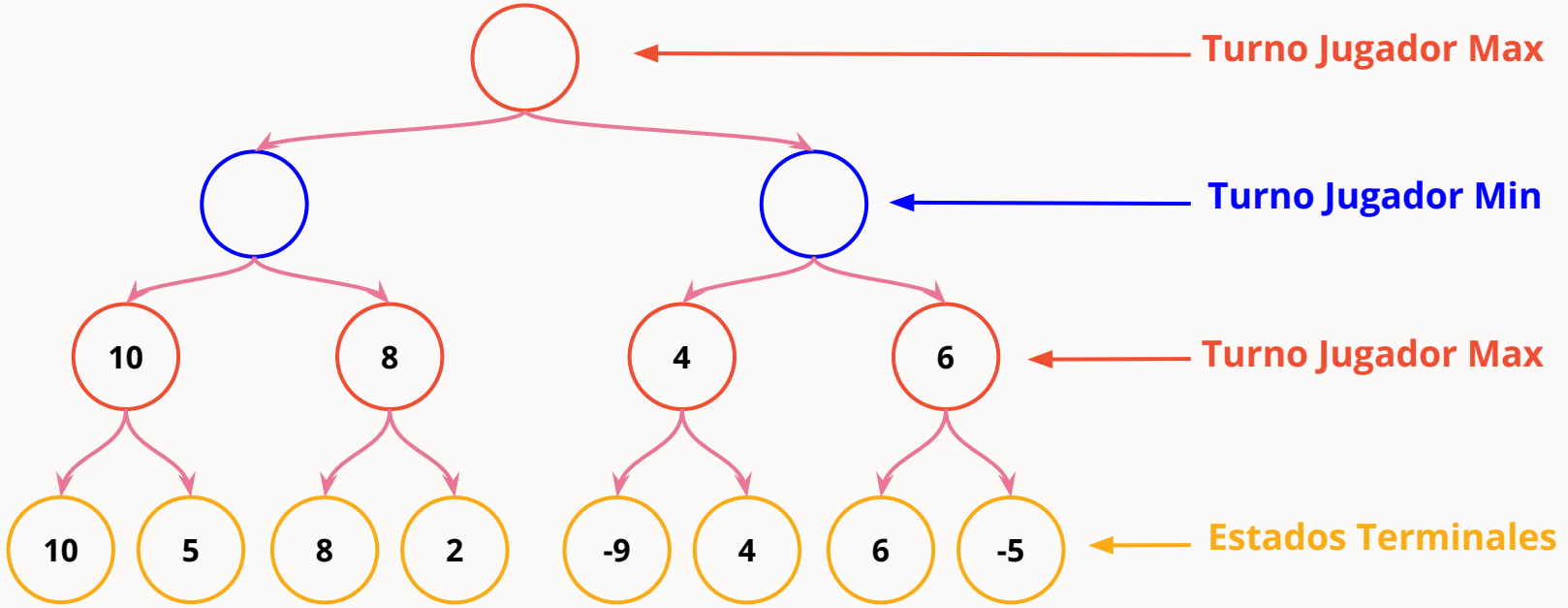
$$\text{MINIMAX(s)} = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \text{máx}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \text{mín}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s)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

Cálculo del valor MiniMax



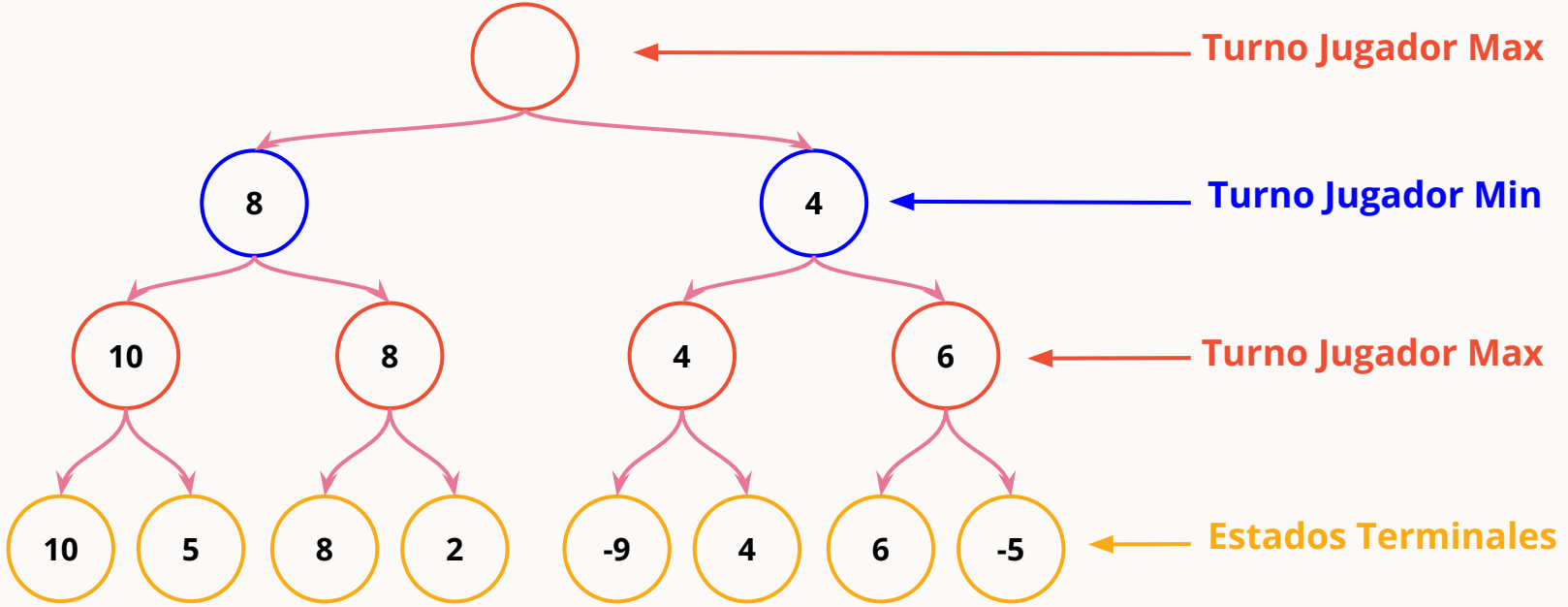
$$\text{MINIMAX(s)} = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \text{máx}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \text{mín}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s)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

Cálculo del valor MiniMax



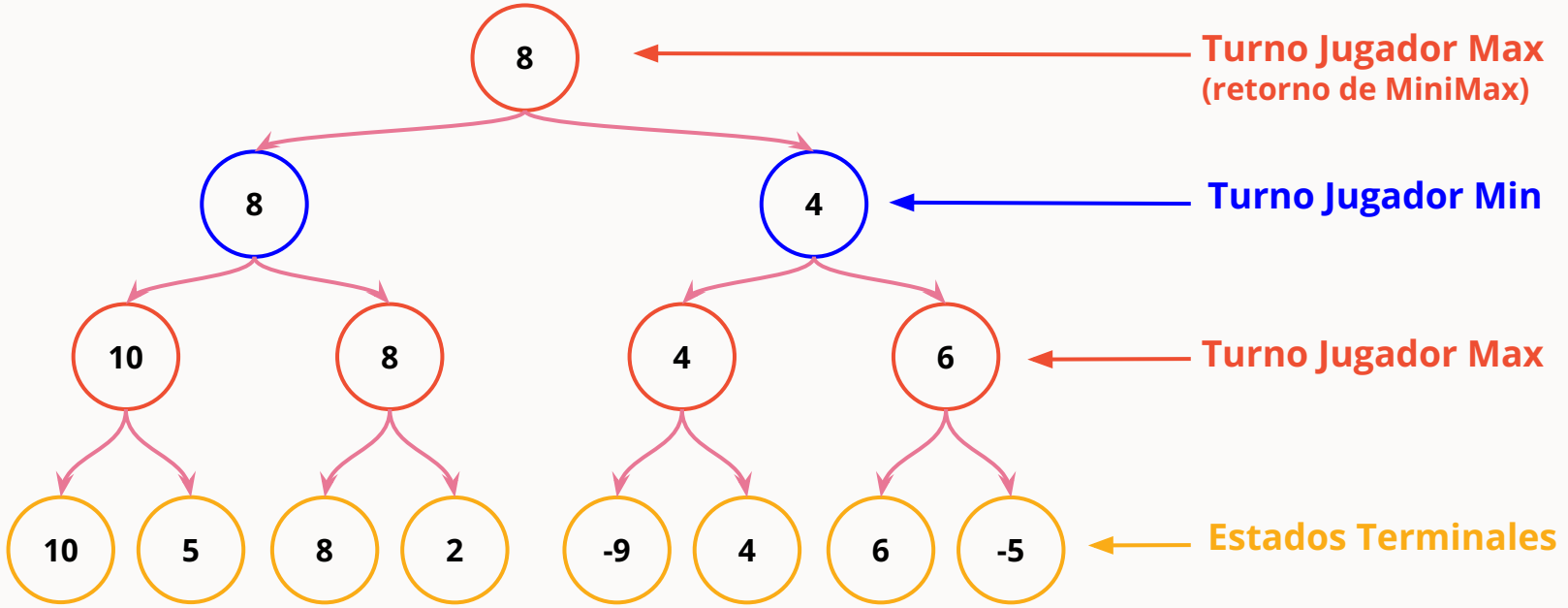
$$\text{MINIMAX(s)} = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \text{máx}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \text{mín}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s)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

Cálculo del valor MiniMax



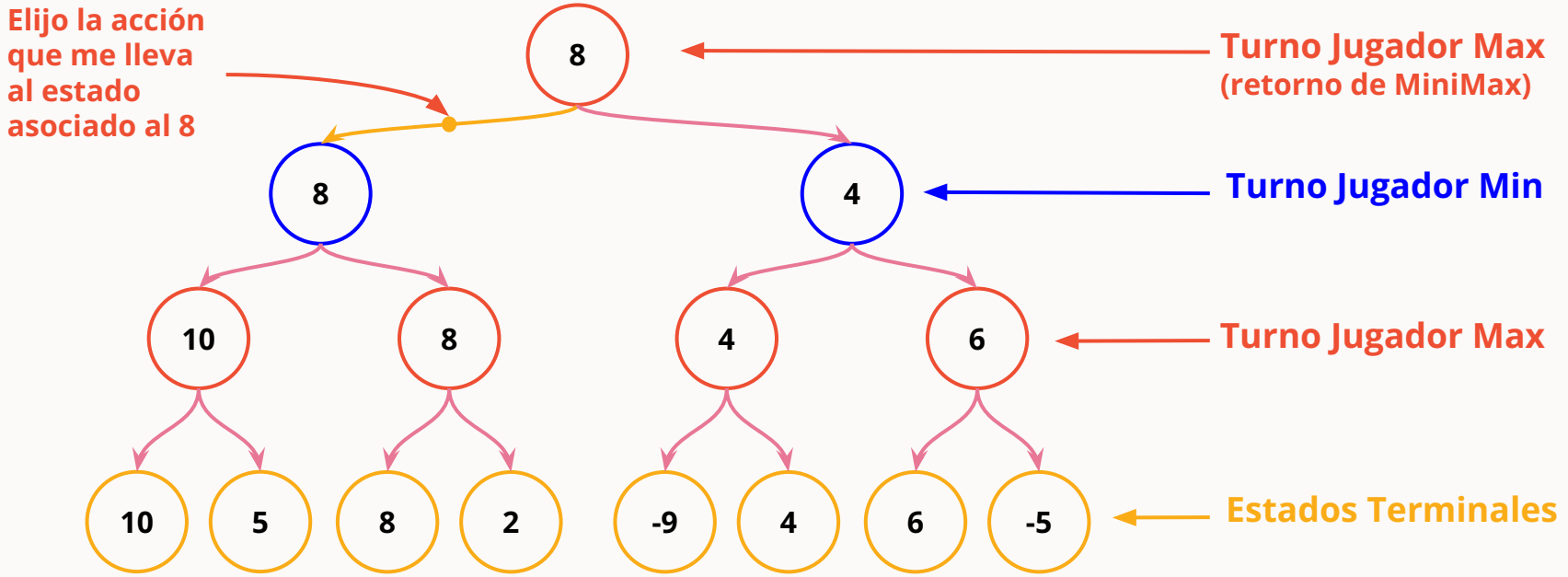
$$\text{MINIMAX(s)} = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \text{máx}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \text{mín}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s)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

Cálculo del valor MiniMax



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

Cálculo del valor MiniMax



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

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

→ Básicamente, hace una estimación sobre “quién va ganando”

Poda Alpha-Beta

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

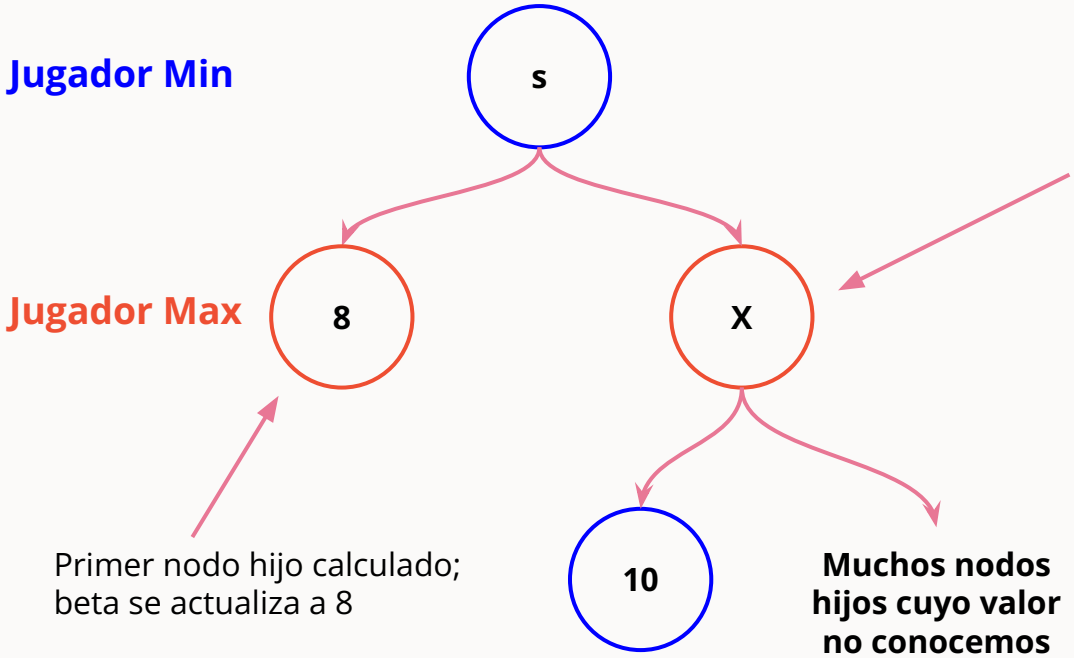
- Alpha: cota **inferior** del valor de un nodo
- Beta: cota **superior** del valor de un nodo

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

Poda Alpha-Beta

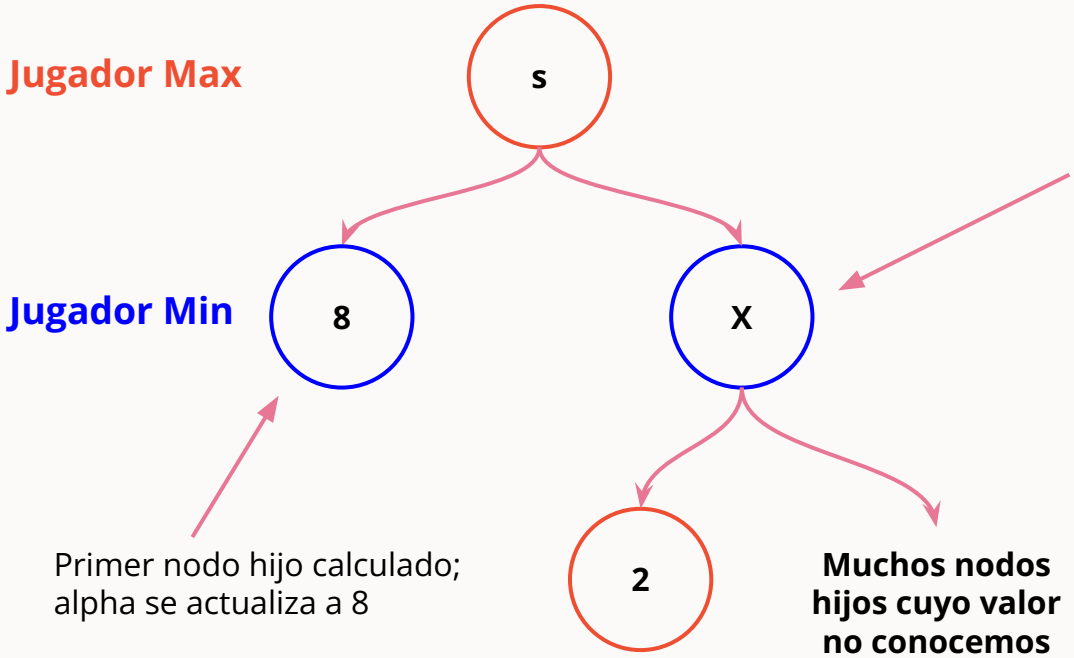
```
function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
    if v2 > v then
      v, move ← v2, a
      α ← MAX(α, v)
    if v ≥ β then return v, move
  return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
    if v2 < v then
      v, move ← v2, a
      β ← MIN(β, v)
    if v ≤ α then return v, move
  return v, move
```

Poda Alpha-Beta - Cota superior Beta



Sabemos que x ≥ 10 apenas verificamos el valor del primer nodo hijo. Como nuestro beta es igual a 8, sabemos que el nodo *s* **no** optará por el valor *x*. Luego podemos omitir el cálculo de los nodos hijos aún no calculados y llegar a la misma elección (podamos el sub-árbol)

Poda Alpha-Beta - Cota inferior Alpha



Sabemos que x ≤ 2 apenas verificamos el valor del primer nodo hijo. Como nuestro alpha es igual a 8, sabemos que el nodo *s* **no** optará por el valor *x*. Luego podemos omitir el cálculo de los nodos hijos aún no calculados y llegar a la misma elección (podamos el sub-árbol)

Menti!

Repaso!

Como bien sabes, un número natural n es un número primo si n es divisible solo por 1 y por n. El objetivo de esta pregunta es que escribas un programa para encontrar todos los primos menores o iguales a `m`. Suponiendo que tu programa ya tiene una regla:

```
numero(1..m).
```

1. Escribe el predicado `compuesto`, tal que `compuesto(N)` se satisface si existe un número `M` mayor que 1 y menor que `N`, que divide a `N`. *Ayuda:* Usa la expresión `L\R` para calcular el módulo de la división entre `L` y `R`. (`L\R` es el equivalente de `L % R` en Python.) (1 pto.)

Como bien sabes, un número natural n es un número primo si n es divisible solo por 1 y por n. El objetivo de esta pregunta es que escribas un programa para encontrar todos los primos menores o iguales a `m`. Suponiendo que tu programa ya tiene una regla:

```
numero(1..m).
```

1. Escribe el predicado `compuesto`, tal que `compuesto(N)` se satisface si existe un número `M` mayor que 1 y menor que `N`, que divide a `N`. *Ayuda:* Usa la expresión `L\R` para calcular el módulo de la división entre `L` y `R`. (`L\R` es el equivalente de `L % R` en Python.) (1 pto.)

   ```
   compuesto(N):- 1 < M, M < N, N \ M = 0, numero(N), numero(M).
   ```

Como bien sabes, un número natural n es un número primo si n es divisible solo por 1 y por n. El objetivo de esta pregunta es que escribas un programa para encontrar todos los primos menores o iguales a `m`. Suponiendo que tu programa ya tiene una regla:

```
numero(1..m).
```

2. Escribe el predicado `primo`, tal que `primo(N)` se satisface si `N` es primo. (1 pto.)

Como bien sabes, un número natural n es un número primo si n es divisible solo por 1 y por n. El objetivo de esta pregunta es que escribas un programa para encontrar todos los primos menores o iguales a `m`. Suponiendo que tu programa ya tiene una regla:

```
numero(1..m).
```

2. Escribe el predicado `primo`, tal que `primo(N)` se satisface si `N` es primo. (1 pto.)

```
primo(N):- not compuesto(N), numero(N).
```

8. El problema de planificar los movimientos de un robots al interior de un edificio generalmente se reduce al de encontrar un camino en una grilla. A su vez, el problema de encontrar un camino en una grilla se puede representar como un problema de búsqueda.

   La Figura 1 muestra un estado inicial, en donde el robot está en la posición $(1, 1)$. El robot debe llegar a $(3, 2)$. Las acciones posibles son *arriba*, *abajo*, *izquierda*, *derecha*. El robot puede ejecutar cualquier acción que lo lleve a una celda adyacente sin atravesar un muro. Los muros representados por la líneas gruesas entre celdas.

   a) **(0,5 puntos)** Diga cómo representaría un estado del espacio de búsqueda de este problema (por ejemplo, en Python).

   b) **(2 puntos)** Diga (o dibuje) qué estados quedan en Closed y en Open justo antes de que retorne una ejecución DFS. Suponga que DFS agrega estados a la pila (*stack*) Open en el siguiente orden: primero el que resulta de ejecutar *arriba*, segundo el que resulta de ejecutar *derecha*, tercero el que resulta de ejecutar *abajo*, cuarto el que resulta de ejecutar *izquierda*. (Por favor, vuelva a leer la oración anterior y asegúrese que la entiende antes de continuar.)

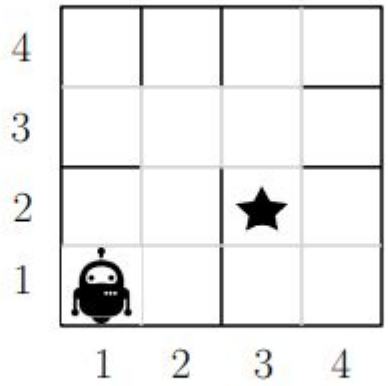


Figura 1: Situación inicial del mundo de navegación usada en preguntas a), b)

Indica si la afirmación es verdadera o falsa y justifica tu respuesta con claridad.

1. Si h_1 y h_2 son heurísticas admisibles, entonces $(h_1 + h_2)/2$ es una heurística admisible. (1 pto.)

Indica si la afirmación es verdadera o falsa y justifica tu respuesta con claridad.

1. Si h_1 y h_2 son heurísticas admisibles, entonces $(h_1 + h_2)/2$ es una heurística admisible. (1 pto.)

   Verdadero. Si $C_r(A.B)$ es el costo real de llegar desde A hasta B, tenemos que

$$h1(A, B) \leq C_r(A, B)$$
$$h2(A, B) \leq C_r(A, B)$$

$$h1(A, B) + h2(A, B) \leq C_r(A, B) + C_r(A, B)$$
$$\frac{h1(A, B) + h2(A, B)}{2} \leq C_r(A, B)$$

2. Si N es el número de caminos que hay desde el estado inicial hasta cierto estado s del espacio de búsqueda, entonces A^* podría expandir s N veces. (1 pto.)

2. Si N es el número de caminos que hay desde el estado inicial hasta cierto estado s del espacio de búsqueda, entonces A^* podría expandir s N veces. (1 pto.)

   Falso. Un nodo es expandido solo una vez dentro de la búsqueda de A*, además, el algoritmo nos garantiza que su expansión será llevada a cabo solo cuando el costo mínimo de llegar a aquel nodo ha sido alcanzado.

14. Una heurística si dice *consistente* si y solo si:

    i. $h(s) \leq c(s,t) + h(t)$ para todo estado s y todo estado t que es sucesor de s, donde $c(s,t)$ representa el costo de la acción que lleva desde s a t, y

    ii. $h(s_g) = 0$ para todo estado objetivo s_g

    Conteste las siguientes preguntas:

    a) (2/3 del puntaje) Demuestre que si h es consistente, cada vez que A* expade un nodo s, todo t que es sucesor de s y que es agregado a Open en esa misma iteración es tal que $f(t) \geq f(s)$.

14. Una heurística si dice *consistente* si y solo si:

i. $h(s) \leq c(s,t) + h(t)$ para todo estado s y todo estado t que es sucesor de s, donde $c(s,t)$ representa el costo de la acción que lleva desde s a t, y

ii. $h(s_g) = 0$ para todo estado objetivo s_g

Conteste las siguientes preguntas:

a) (2/3 del puntaje) Demuestre que si h es consistente, cada vez que A* expade un nodo s, todo t que es sucesor de s y que es agregado a Open en esa misma iteración es tal que $f(t) \geq f(s)$.

Respuesta: Sumando $g(s)$ a ambos lados obtenemos:

$$g(s) + h(s) \leq g(s) + c(s,t) + h(t)$$

Si t es agregado a Open entonces $g(t) = g(s) + c(s,t)$ por lo que podemos reemplazar arriba y obtener $g(s) + g(s) \leq g(t) + h(t)$ que es equivalente a lo que se quiere mostrar.

b) (1/3 del puntaje) Diga por qué esta relación podría no cumplirse con algunos sucesores de s que ya estaban en Open.

Respuesta: Cuando un sucesor de s, digamos t, no se agrega a Open, entonces se da que $g(s) + c(s,t) \geq g(t)$, por esta razón la desigualdad de arriba podría perfectamente invalidarse de hacer la sustitución que hicimos arriba.