

# Bloque 1 – Representación del conocimiento y búsqueda



**Tema 6:  
Búsqueda con adversario**

# Tema 6- Índice

1. Juegos: definición del problema
2. Algoritmo MINIMAX
3. Poda Alfa-beta
4. Refinamientos adicionales

## Bibliografía:

- S. Russell, P. Norvig. *Artificial Intelligence . A modern approach.* Prentice Hall, 3rd edition, 2010 (Tema 5) <http://aima.cs.berkeley.edu/>
  - N. J. Nilsson. *Artificial Intelligence: A New Synthesis.* Morgan Kaufmann Publishers, 1998 (Tema 1)
- Alternativamente:
- S. Russell, P. Norvig. *Inteligencia Artificial. Una aproximación moderna.* Prentice Hall, 2nd edition, 2004 (Tema 6) <http://aima.cs.berkeley.edu/2nd-ed/>

## 1. Juegos: definición del problema



Un juego es una situación conflictiva en la que uno debe tomar una decisión sabiendo que los demás también toman decisiones, y que el resultado del conflicto se determina, de algún modo, a partir de todas las decisiones realizadas. (John von Neumann )

Siempre existe una forma racional de actuar en juegos de dos participantes, si los intereses que los gobiernan son completamente opuestos. (John von Neumann )

La demostración es la Teoría Minimax ( 1926).

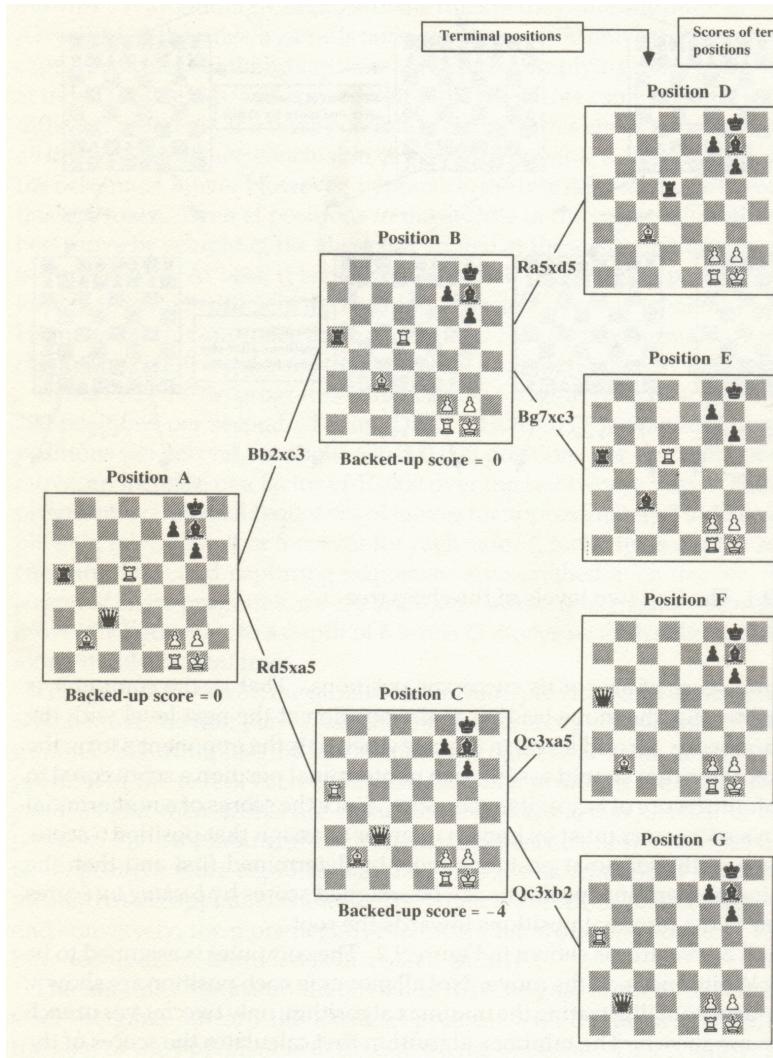
## 1. Juegos: definición del problema (2)



La mayoría de juegos que se estudian en IA son:

- **Deterministas** (pero estratégicos): la suerte no interviene
- Se juegan alternativamente **por turnos**
- **Dos jugadores**: persona-persona, persona-ordenador, ordenador-ordenador
- **Suma cero**: la ganancia (o pérdida) de un participante es exactamente la pérdida (o ganancia) del oponente. Esta oposición define los juegos como problemas de búsqueda con adversario.
- **Información perfecta**: los juegos son conocidos y limitados: cada jugador conoce perfectamente la evolución del juego y los movimientos que puede hacer su oponente (observable)

# 1. Juegos: definición del problema(3)



El número de movimientos de los jugadores está generalmente restringido a un pequeño número de acciones

Los resultados de las acciones se definen con reglas precisas

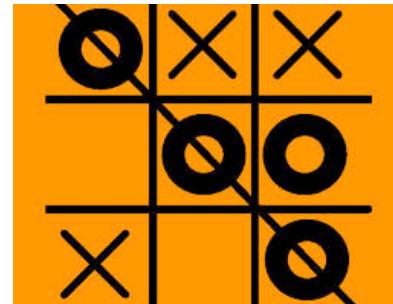
Los juegos son problemas complejos. Por ejemplo, el ajedrez tiene un factor de ramificación medio de aproximadamente 35, y en un juego, cada jugador hace una media de 50 movimientos, así que el árbol de búsqueda del juego tendría unos  $35^{100}$  nodos ...

Los juegos son problemas que requieren tomar una decisión aunque la decisión óptima sea inviable

## 1. Juegos: definición del problema(4)

Dos jugadores: **MAX** y **MIN**. MAX mueve primero y juegan alternativamente por turnos hasta que el juego finaliza.

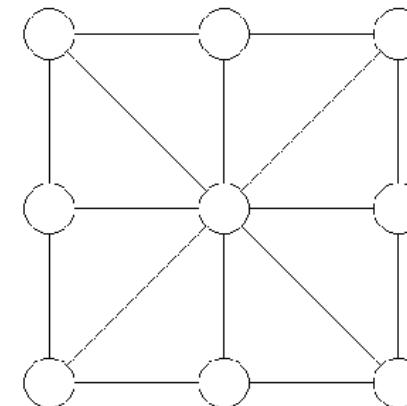
Ejemplo: tic-tac-toe (Tres en raya )



**Estado inicial:** incluye la posición del tablero e identificación de los dos jugadores.

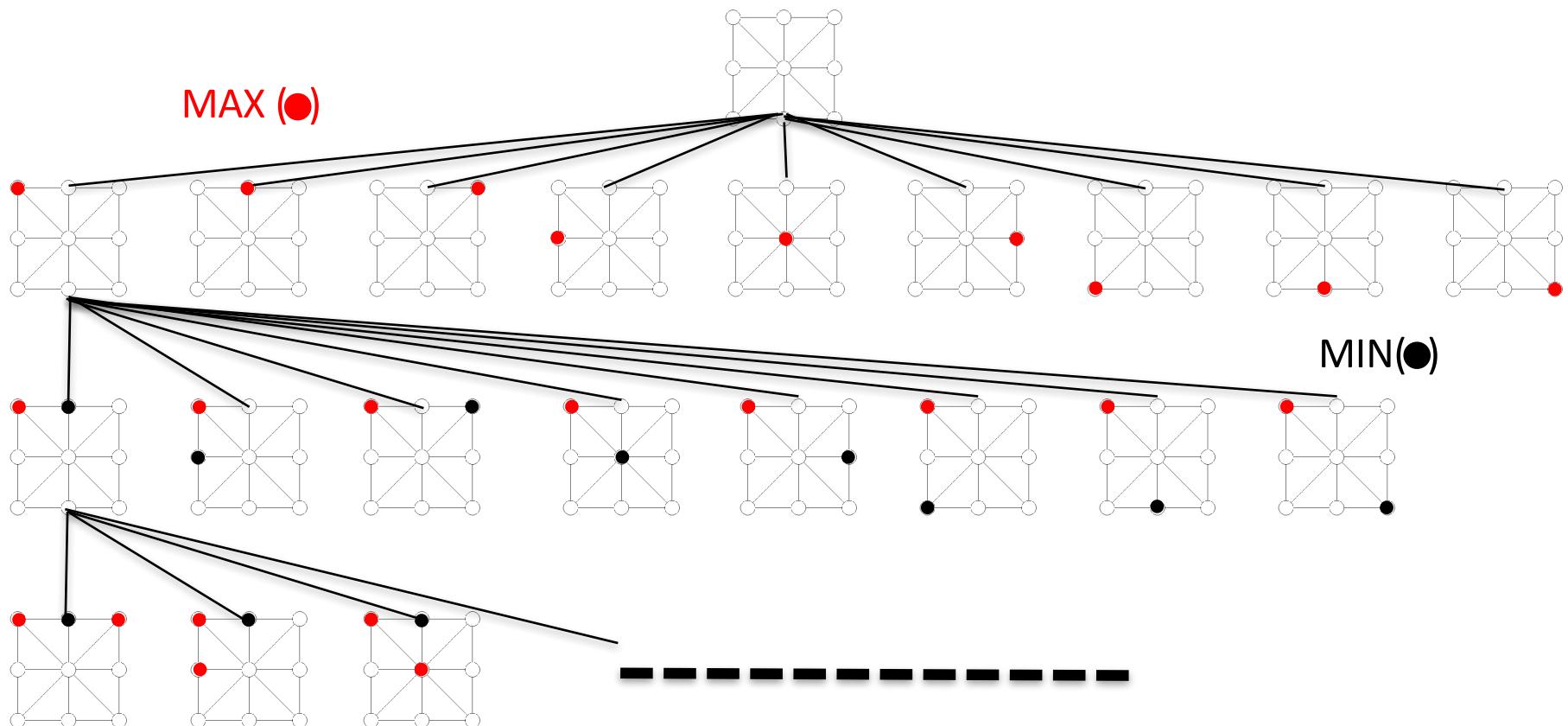
MAX (●), 1er turno

MIN (○), 2º turno

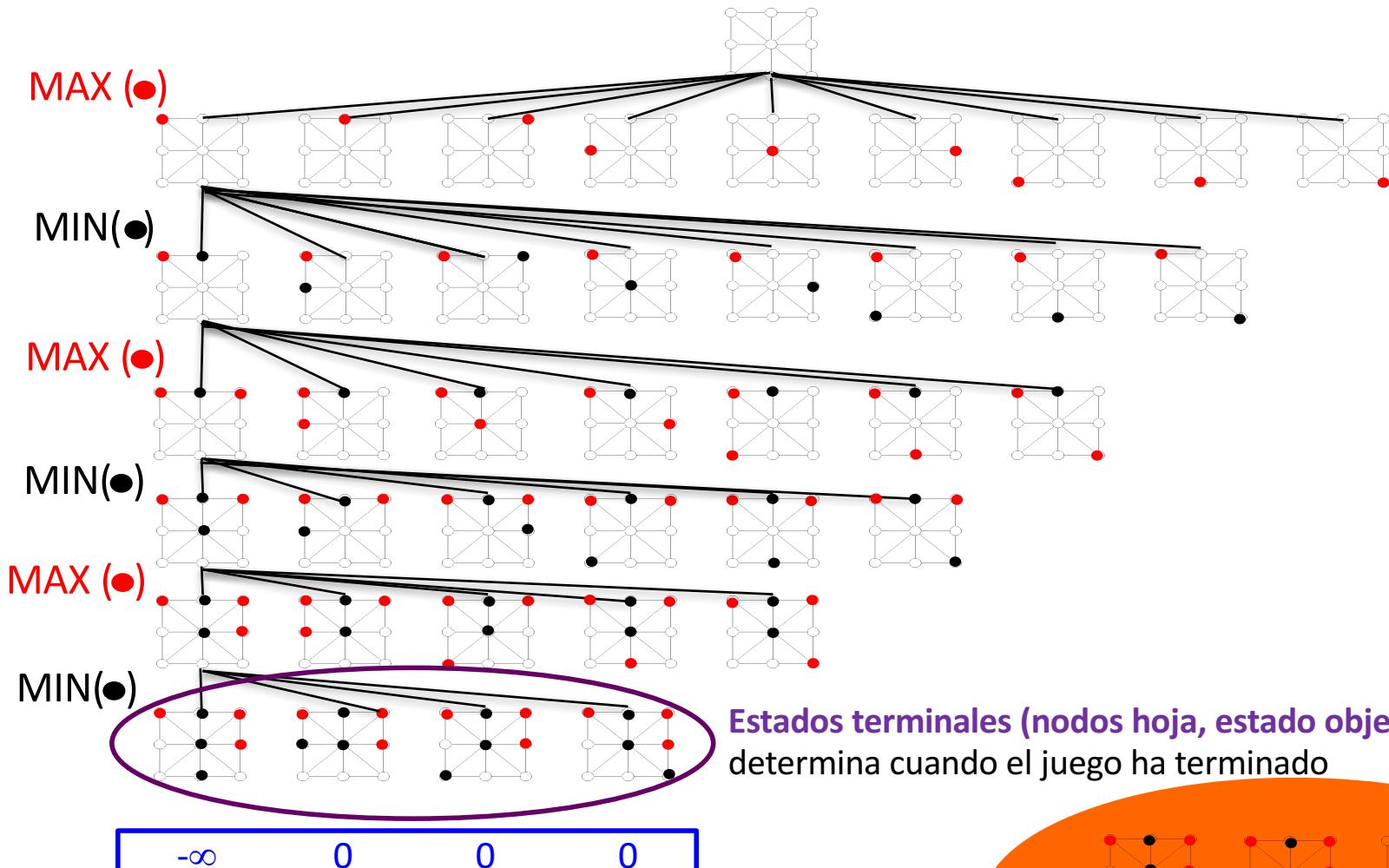


## 1. Juegos: definición del problema(5)

**Acciones**: movimientos legales que puede hacer cada jugador; una acción da lugar a un nuevo estado (nuevo estado del tablero). Los estados sucesores representan los estados que un jugador puede alcanzar en un turno.



# 1. Juegos: definición del problema(6)

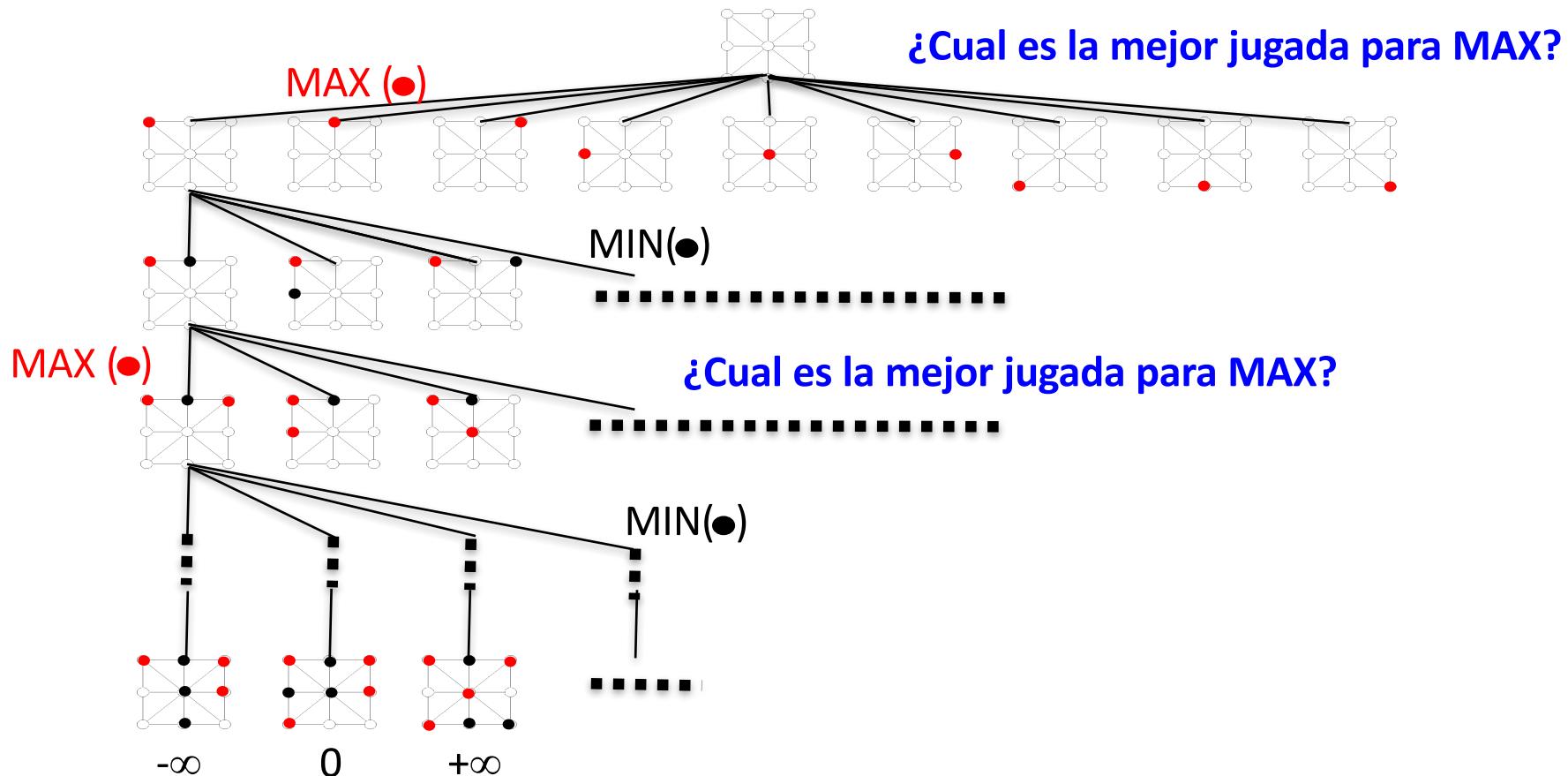


**Estados terminales (nodos hoja, estado objetivo):**  
determina cuando el juego ha terminado

**Función de utilidad:** también llamada función objetivo o función de recompensa; da un valor numérico a los estados terminales. Los valores son ganar ( $+\infty$ ), perder ( $-\infty$ ) o empate (0)

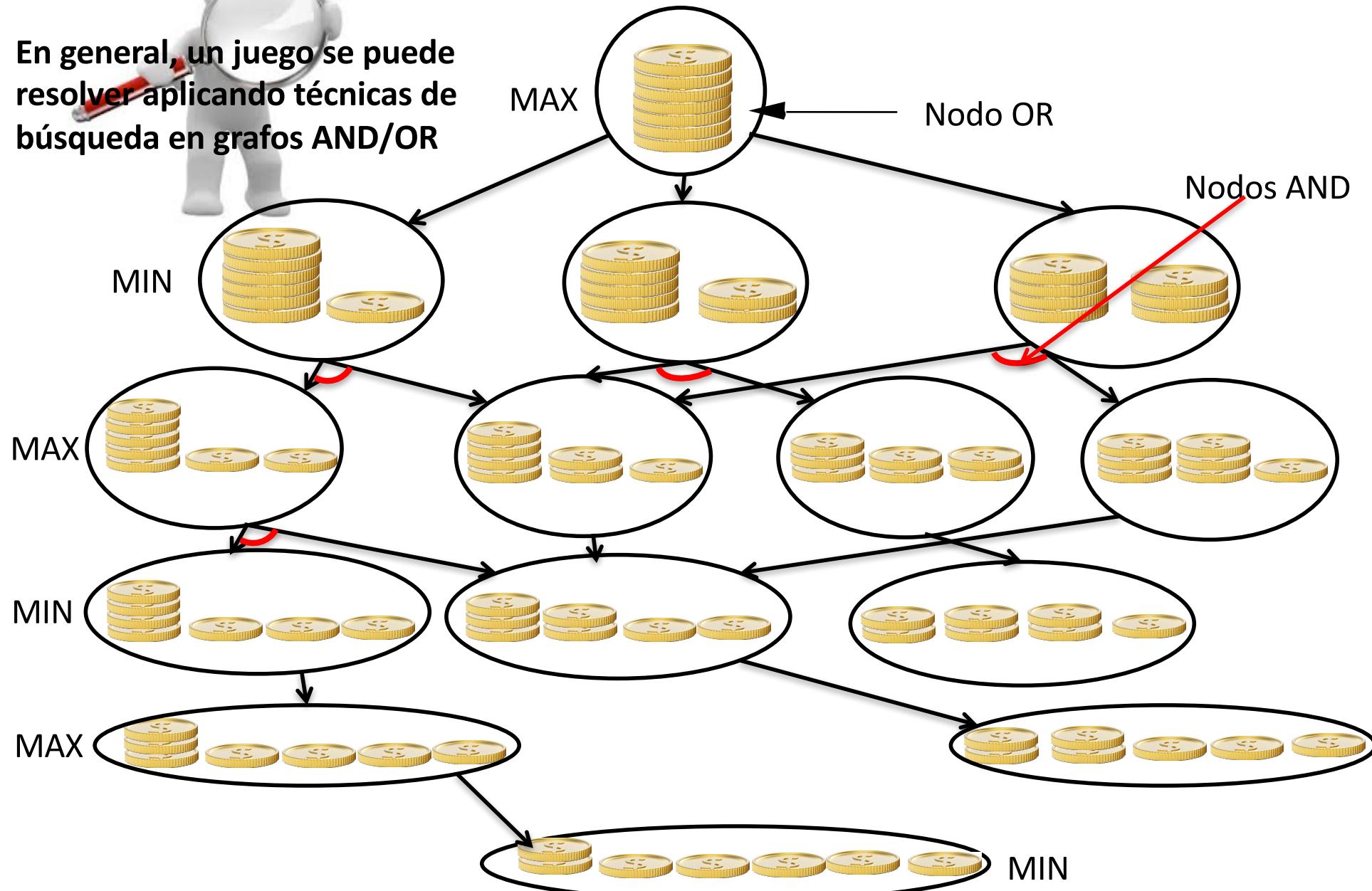
## 1. Juegos: definición del problema(7)

- El estado inicial y los movimientos (reglas) del juego definen el árbol del juego
- MAX es el jugador inicial. Los valores de los nodos hoja indican el **valor de utilidad del estado terminal desde el punto de vista de MAX**. Valores altos son buenos para MAX y malos para MIN.
- Objetivo: usar el árbol de búsqueda del juego (particularmente la utilidad de los nodos terminales) para **determinar el mejor movimiento de MAX**.



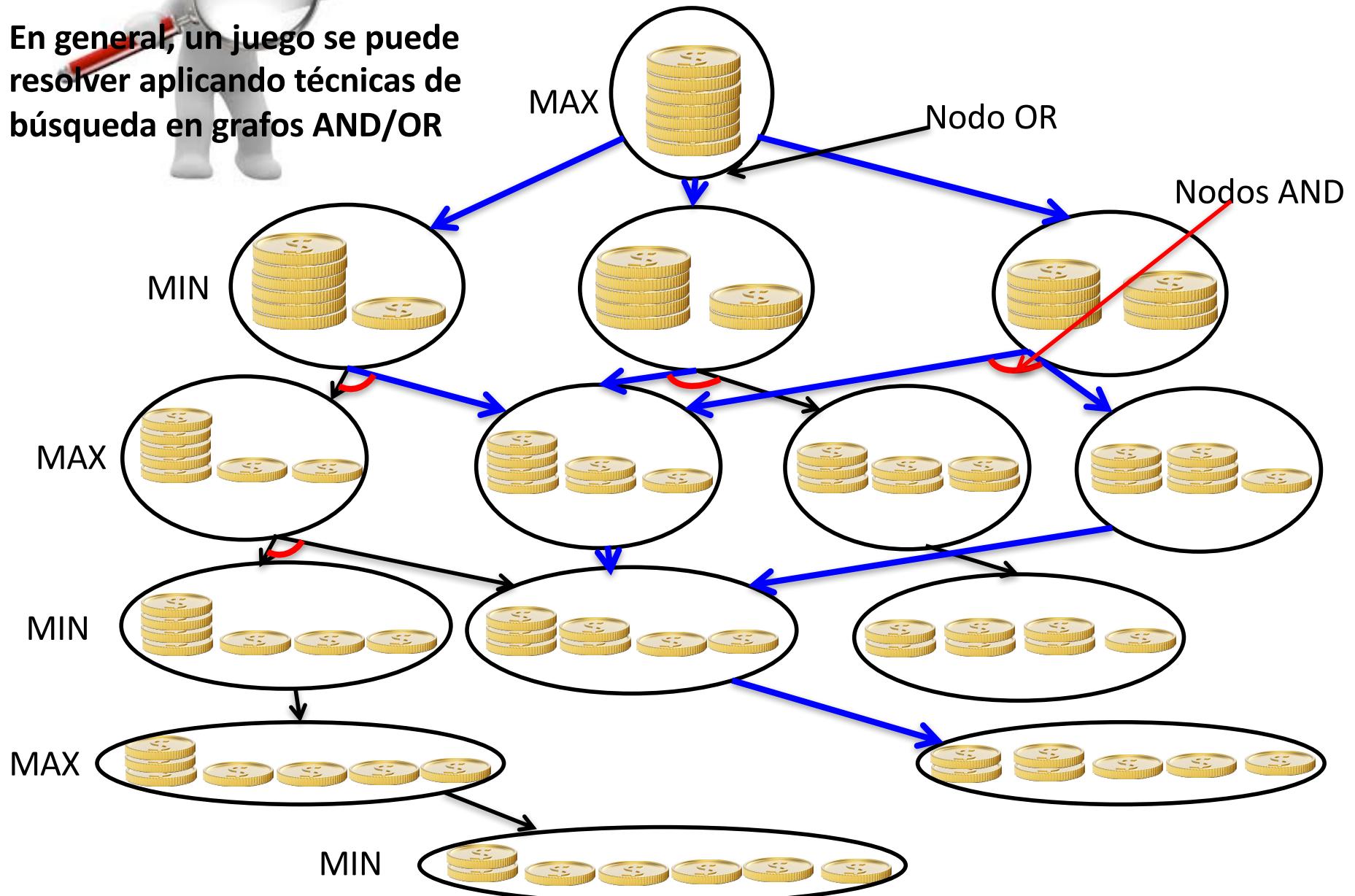
# 1. Juegos: definición del problema (8). Juego de Grundy

En general, un juego se puede resolver aplicando técnicas de búsqueda en grafos AND/OR

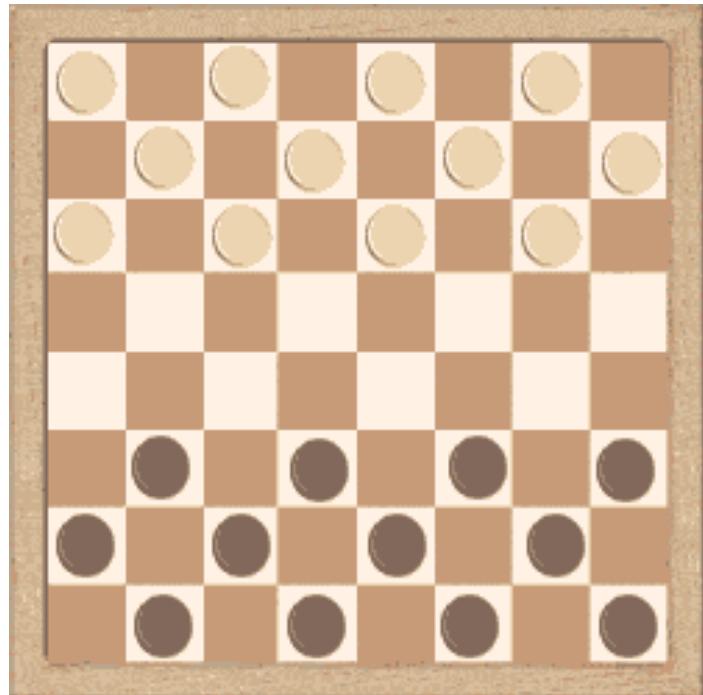


# 1. Juegos: definición del problema (9). Juego de Grundy

En general, un juego se puede resolver aplicando técnicas de búsqueda en grafos AND/OR



# 1. Juegos: definición del problema (10)



## Problemas:

- Incluso para juegos sencillos, realizar una búsqueda completa es inviable

## Soluciones:

- Incrementar la información heurística del método aún a costa de perder admisibilidad. El objetivo es aproximar  $b^* \approx 1$ . (Esto no es viable en algunos juegos ya que se dispone de poca información sobre el problema).
- Podar el árbol de búsqueda en ciertos puntos y usar métodos para seleccionar el mejor movimiento:

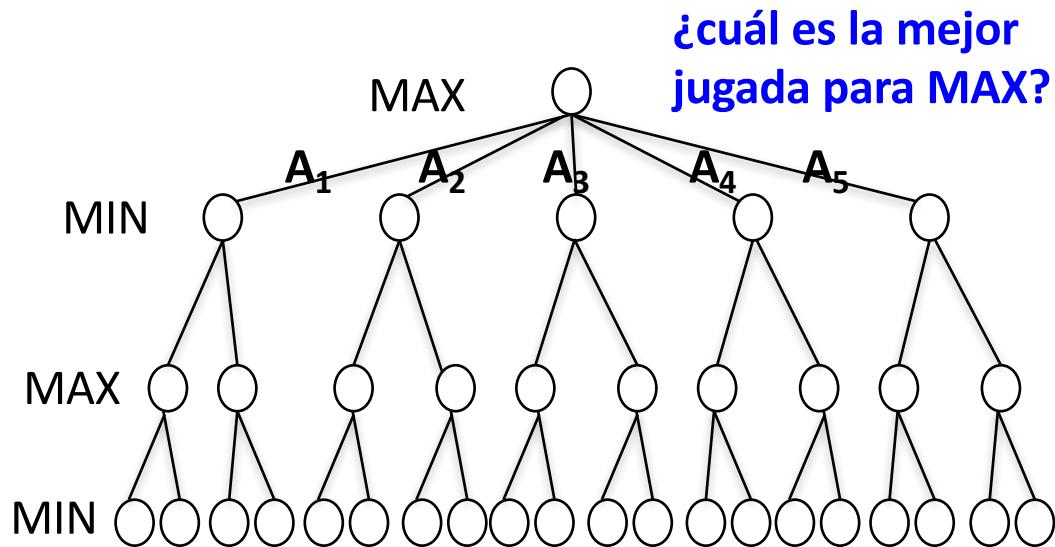
**MINIMAX   y    $\alpha-\beta$**

## 2. Algoritmo Minimax (John von Neumann )

¿En el estado de desarrollo del juego donde tenga que jugar MAX (estado inicial) cuál es su mejor jugada?

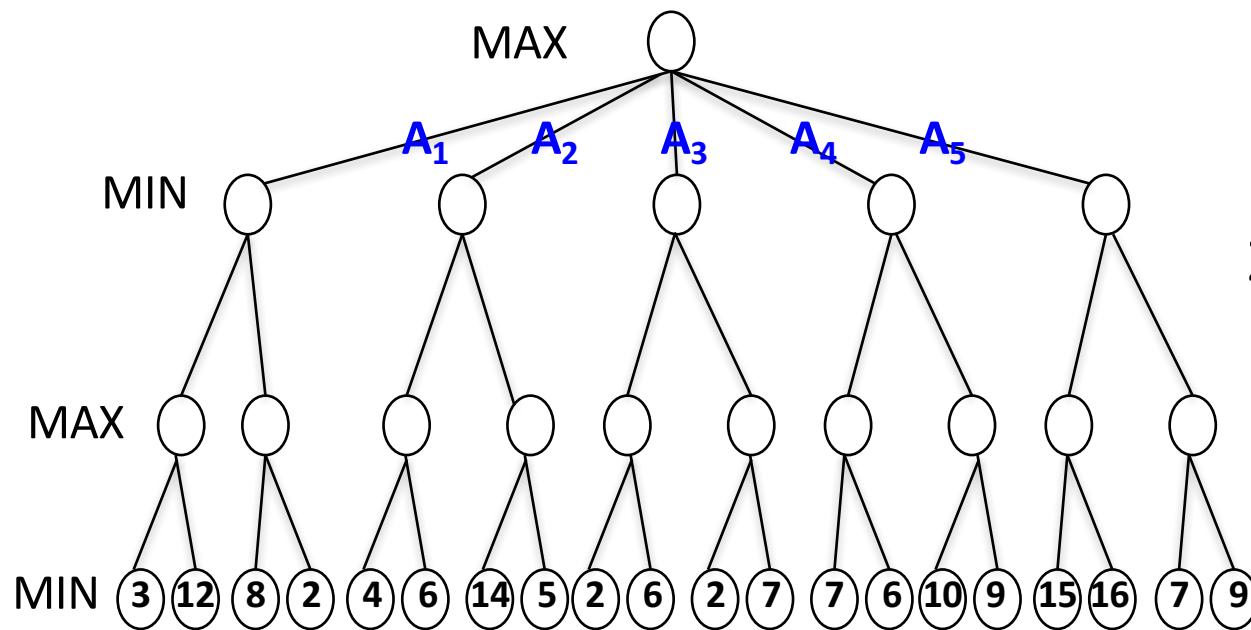
Max debe encontrar una estrategia que especifique el movimiento de MAX en el estado inicial considerando los posibles movimientos de MAX y de MIN en los estados que resultan de desarrollar el juego hasta un cierto nivel.

En términos generales, una estrategia óptima conduce a resultados al menos tan buenos como cualquier otra estrategia cuando uno juega con un oponente infalible.



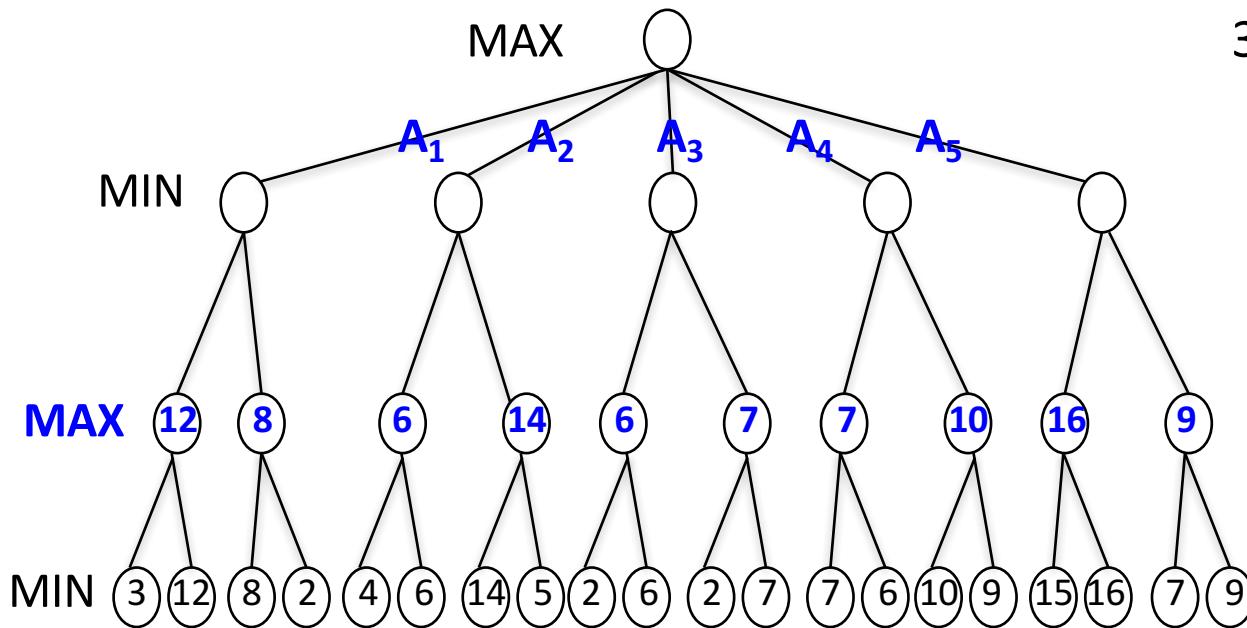
1. Generar el árbol del juego hasta un cierto nivel de profundidad en anchura hasta alcanzar los nodos terminales en el nivel máximo de profundidad.

## 2. Algoritmo Minimax (2)



2. Aplicar la función de utilidad a cada nodo terminal

## 2. Algoritmo Minimax (3)

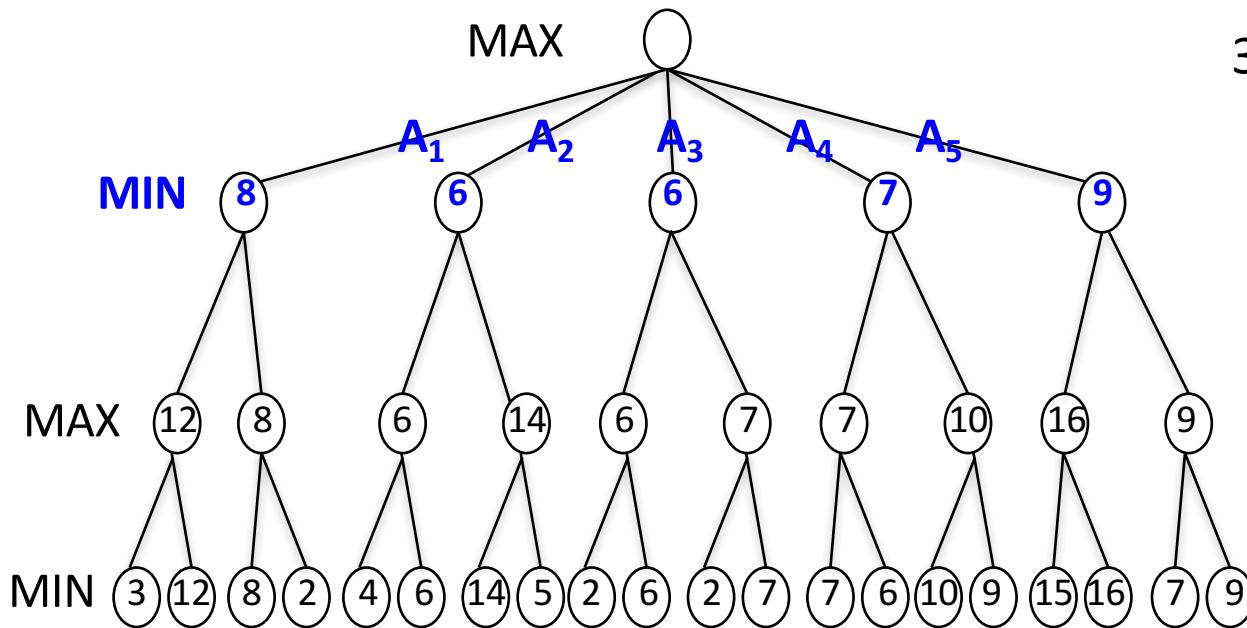


De las distintas jugadas que puede realizar un nodo MAX la mejor será la que 'le lleve' al nodo hijo con mayor valor de la función de utilidad

**Valor volcado nodo MAX** = máximo valor de función de utilidad de sus hijos

3. Usar los valores de utilidad de los nodos terminales para determinar la utilidad de los nodos del nivel superior. **Volcar** los valores de utilidad desde los nodos hoja hasta la raíz, nivel por nivel (**profundidad**). En los nodos MIN se vuelca el menor valor de utilidad de sus sucesores; en los nodos MAX se vuelca el mayor valor de utilidad de sus sucesores.

## 2. Algoritmo Minimax (4)



De las distintas jugadas que puede realizar un nodo MIN la mejor será la que '*le lleve*' al nodo hijo con **menor** valor de la función de utilidad

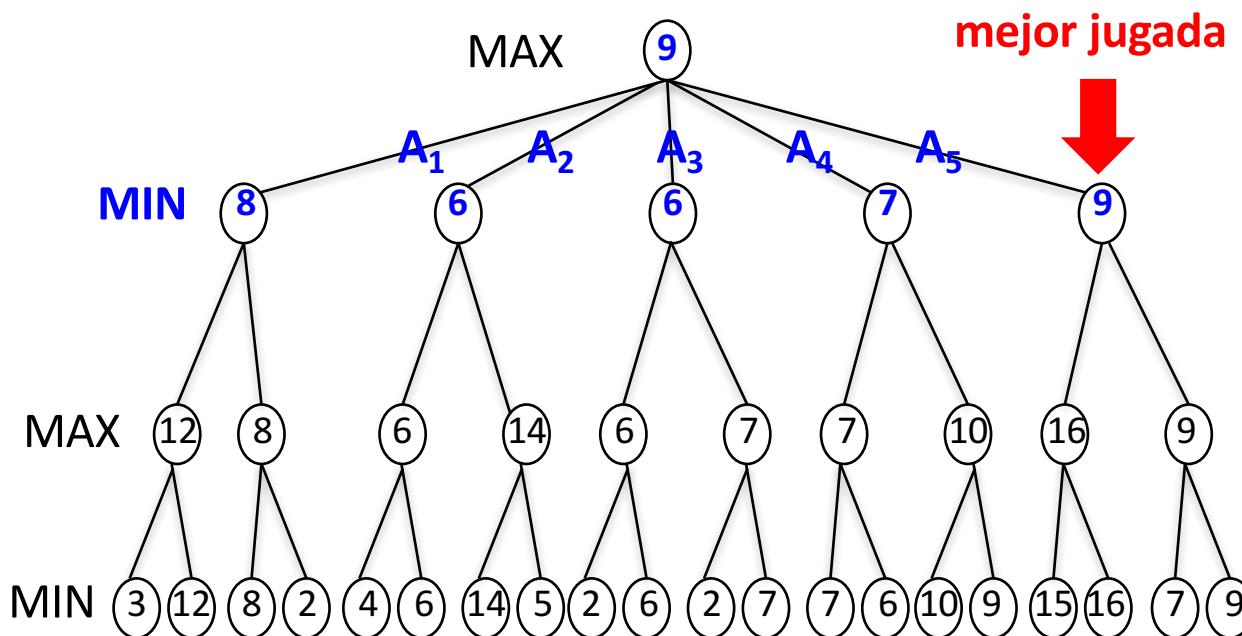
**Valor volcado nodo MIN** = mínimo valor de función de utilidad de sus hijos

3. Usar los valores de utilidad de los nodos terminales para determinar la utilidad de los nodos del nivel superior. **Volcar** los valores de utilidad desde los nodos hoja hasta la raíz, nivel por nivel. En los nodos MIN se vuelca el menor valor de utilidad de sus sucesores; en los nodos MAX se vuelca el mayor valor de utilidad de sus sucesores.

## 2. Algoritmo Minimax (5)

¿Cuál es la mejor jugada para el nodo raíz (MAX)?

Aquella con mayor valor volcado, ya que realizando la jugada que se corresponda con el máximo valor volcado MAX tiene garantizado alcanzar un estado del juego con una utilidad mínima (en el ejemplo '9'). Observar que si MIN no '*jugase bien*' MAX podría alcanzar un estado con utilidad '16'



4. Eventualmente, el nodo raíz adquiere un valor volcado; en este punto, MAX escoge el movimiento que le conduce al estado de máxima utilidad.

En el caso de producirse empates entre los máximos valores de utilidad de algunos hijos del nodo raíz da igual cuál de ellos elijamos como mejor jugada. Cuando se produce uno de estos empates decimos que se produce una **meseta**.

## 2. Algoritmo Minimax (6)

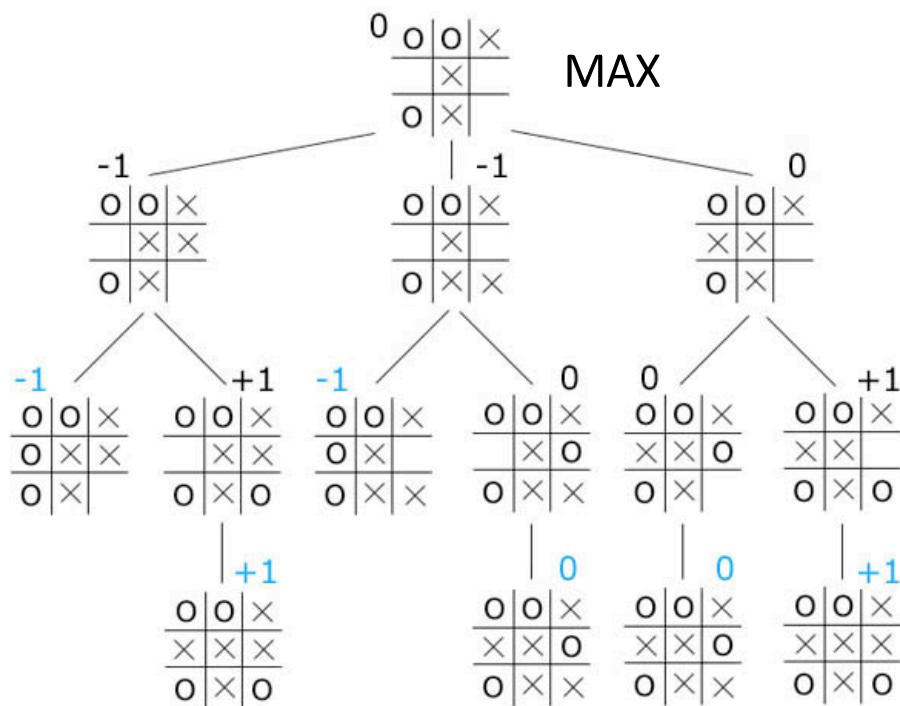
1. Generar el árbol del juego hasta un cierto nivel de profundidad en anchura hasta alcanzar los nodos terminales en el nivel máximo de profundidad.
2. Aplicar la función de utilidad a cada nodo terminal
3. Usar los valores de utilidad de los nodos terminales para determinar la utilidad de los nodos del nivel superior. **Volcar** los valores de utilidad desde los nodos hoja hasta la raíz, nivel por nivel (**profundidad**). En los nodos MIN se vuelca el menor valor de utilidad de sus sucesores; en los nodos MAX se vuelca el mayor valor de utilidad de sus sucesores.

**Valor\_Minimax(n)=**

$$\left\{ \begin{array}{ll} \text{Utilidad}(n) & \text{si } n \text{ es un nodo terminal} \\ \max_{s \in \text{Sucesor}(n)} \text{Valor_Minimax}(s) & \text{si } n \text{ es un nodo MAX} \\ \min_{s \in \text{Sucesor}(n)} \text{Valor_Minimax}(s) & \text{si } n \text{ es un nodo MIN} \end{array} \right.$$

4. Eventualmente, el nodo raíz adquiere un valor volcado; en este punto, MAX escoge el movimiento que le conduce al estado de máxima utilidad.

## 2. Algoritmo Minimax (7)



### Utilidad del Minimax:

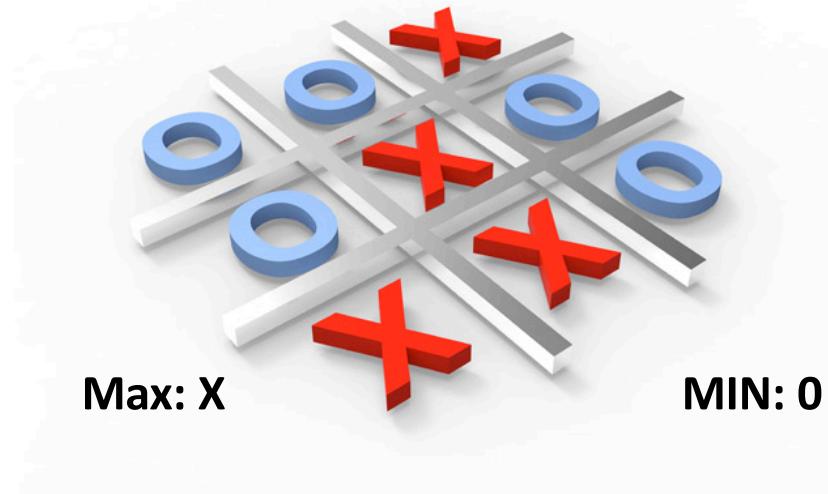
La selección del mejor movimiento se realiza teniendo en cuenta la evolución del juego en niveles más profundos: el valor volcado de un nodo es un valor mucho más informado que el resultado de aplicar directamente la función de utilidad sobre dicho nodo.

### Eficiencia del Minimax:

Nivel de profundidad en el árbol de juego

Función de utilidad

### 3. Algoritmo Minimax. Ejemplo: tic-tac-toe (tres en raya)



Función de utilidad: incorpora conocimiento del problema

$f(n) = (\text{número de filas, columnas o diagonales abiertas para MAX}) - (\text{número de filas, columnas o diagonales abiertas para MIN})$

$f(n) = +\infty$ , si es una jugada ganadora para MAX

$f(n) = -\infty$ , si es una jugada ganadora para MIN

$$f(n) = 5 - 4 = 1$$

### 3. Algoritmo Minimax. Ejemplo: tic-tac-toe (tres en raya) (2)

#### Árbol de búsqueda:

- Profundidad: 2 niveles
- No se representan posiciones simétricas

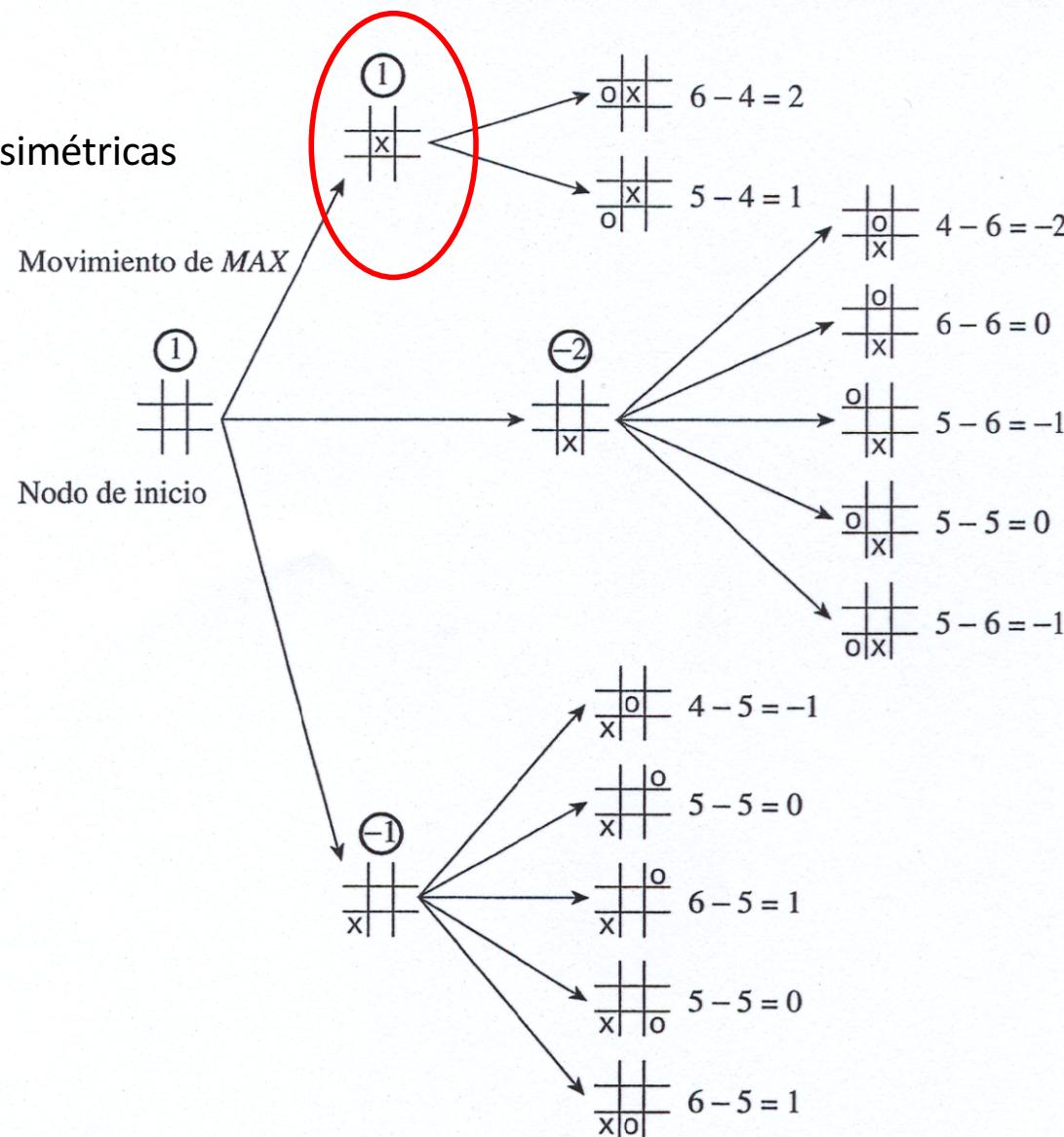
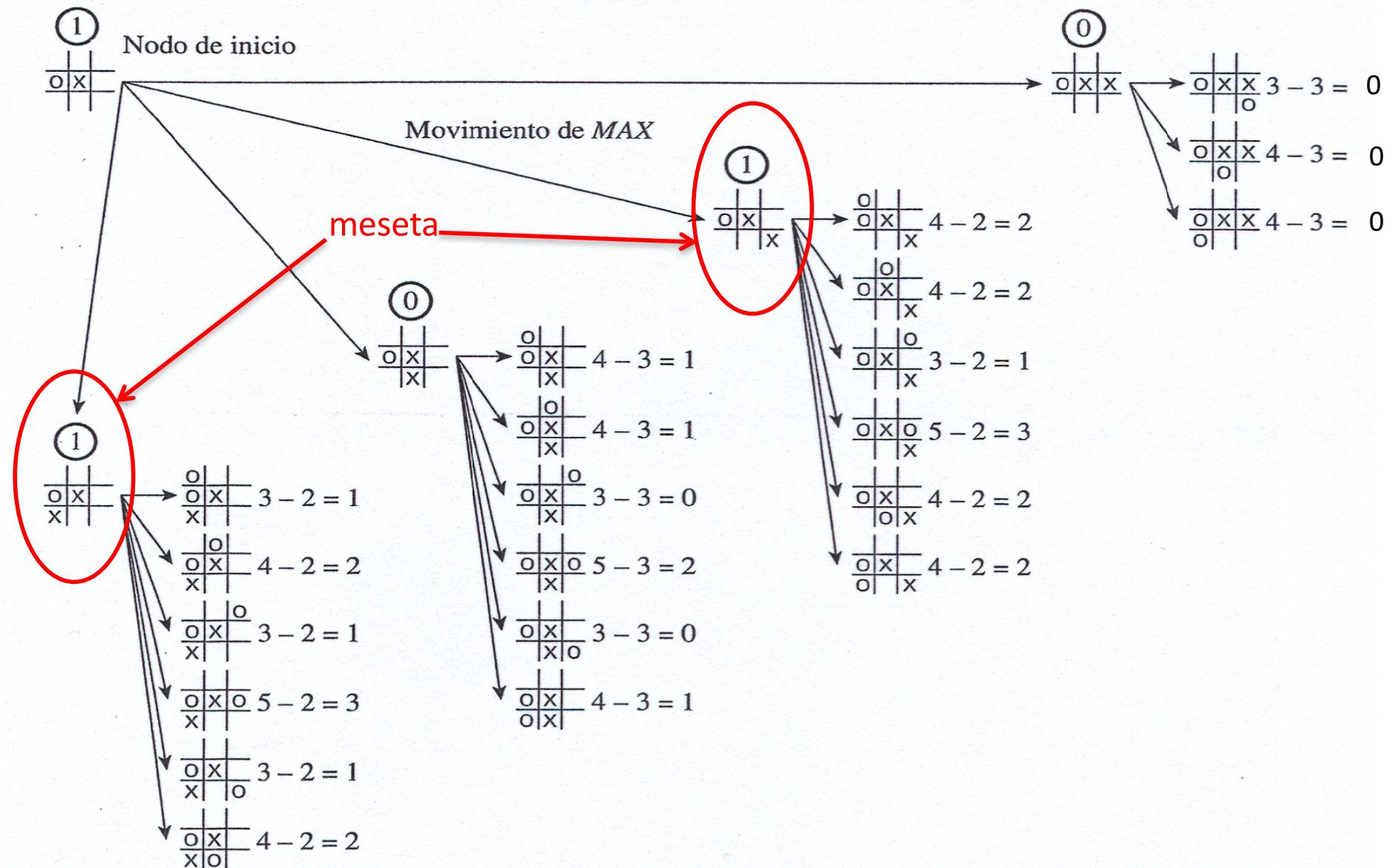
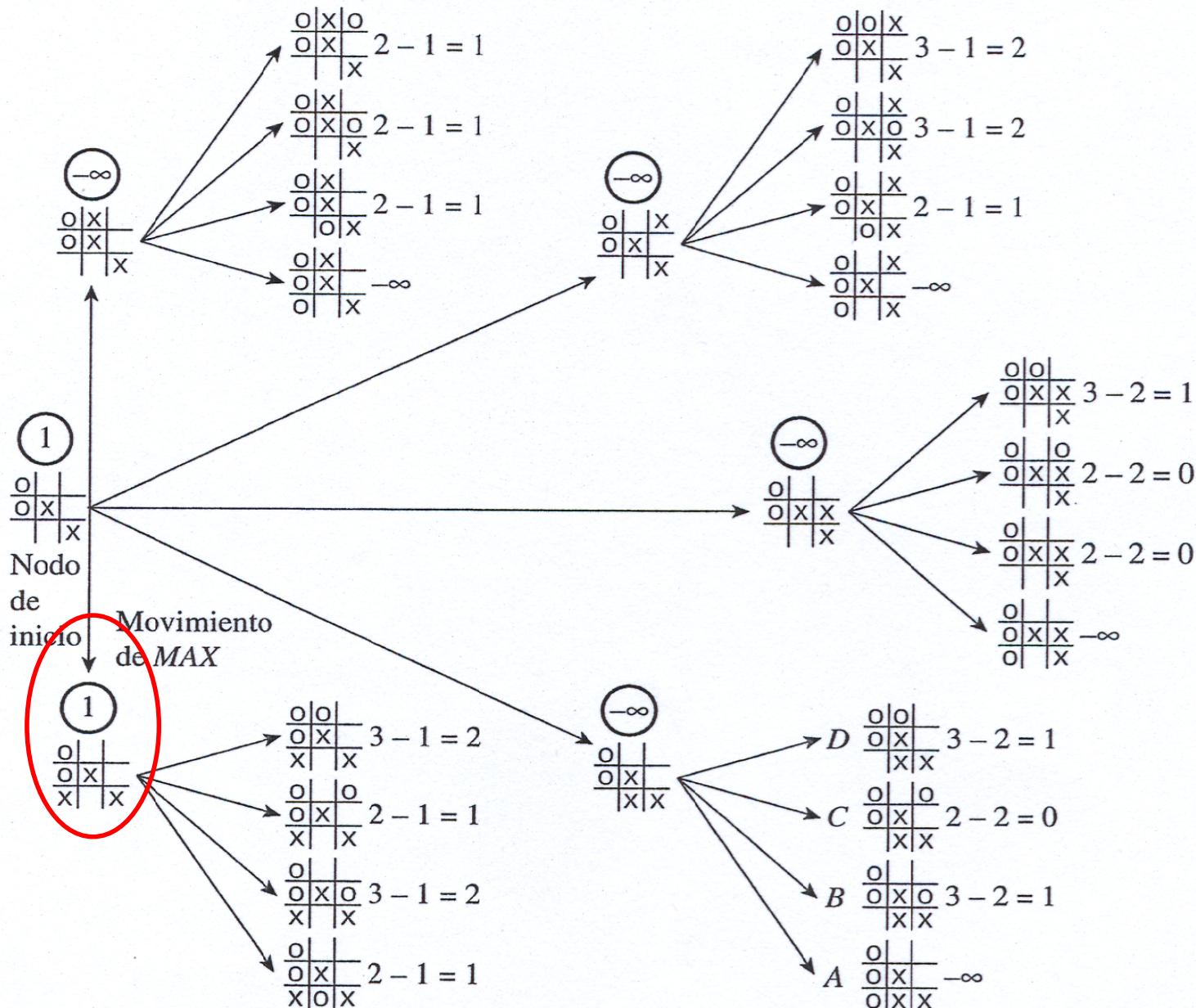


Figura 12.3.

### 3. Algoritmo Minimax. Ejemplo: tic-tac-toe (tres en raya) (3)



### 3. Algoritmo Minimax. Ejemplo: tic-tac-toe (tres en raya) (4)

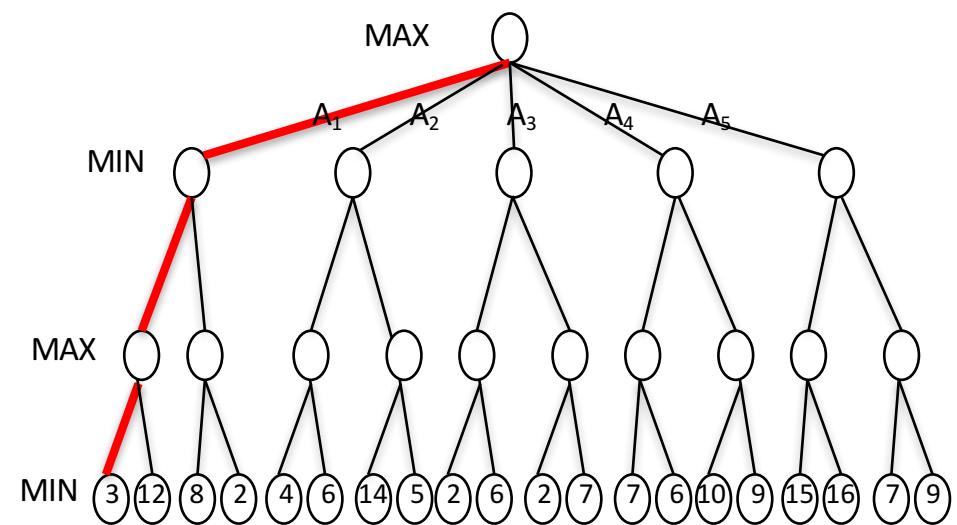
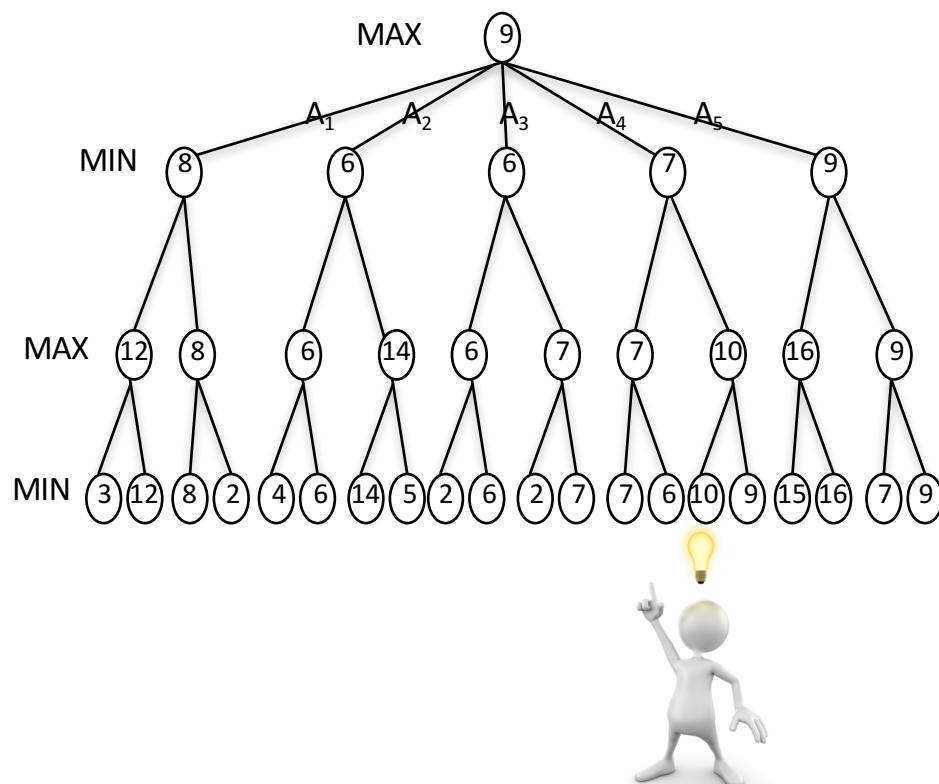


## 4. Algoritmo $\alpha$ - $\beta$

Mini-Max: número de nodos en el árbol de juego es exponencial con el número de movimientos.

¿Se puede calcular la decisión Mini-Max correcta sin tener que examinar todos los nodos del árbol de búsqueda? ¿Es decir, se puede eliminar (podar) grandes partes del árbol?

**Mini-Max → búsqueda en amplitud  
hasta un cierto nivel de profundidad**



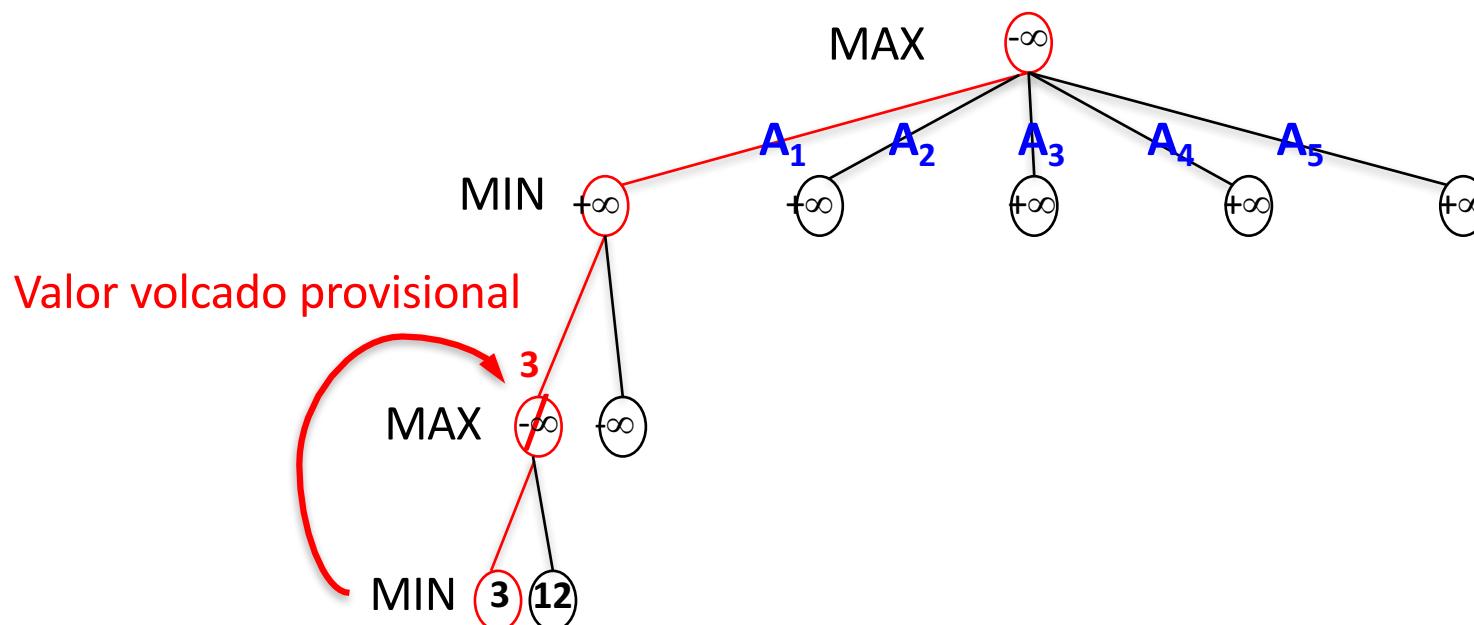
**¿Desarrollar, hasta generar el mismo árbol de búsqueda en el mismo nivel de profundidad, una búsqueda en profundidad?**

## 4. Algoritmo $\alpha$ - $\beta$ (2)

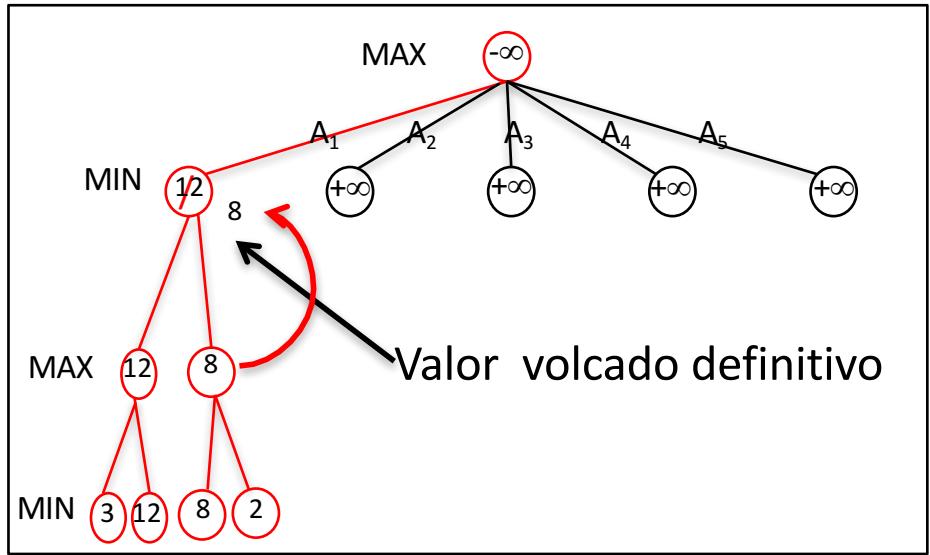
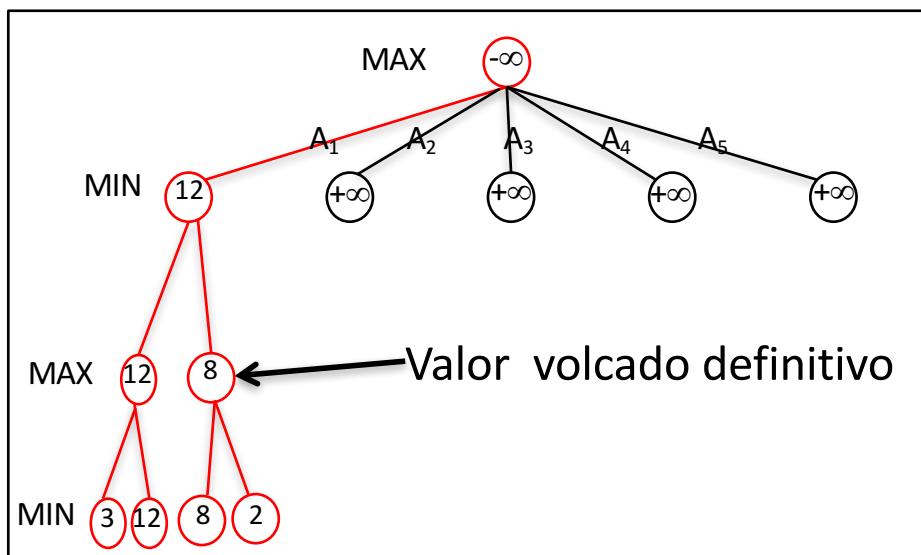
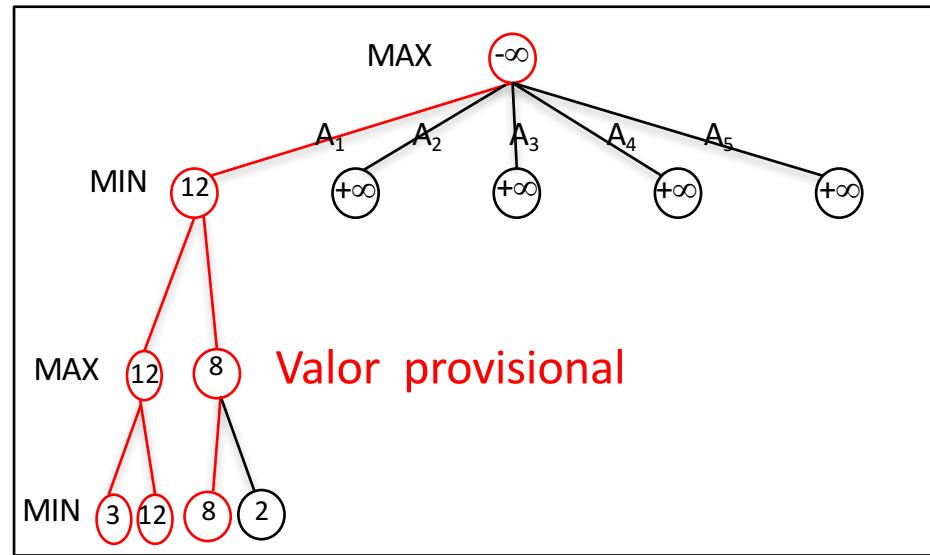
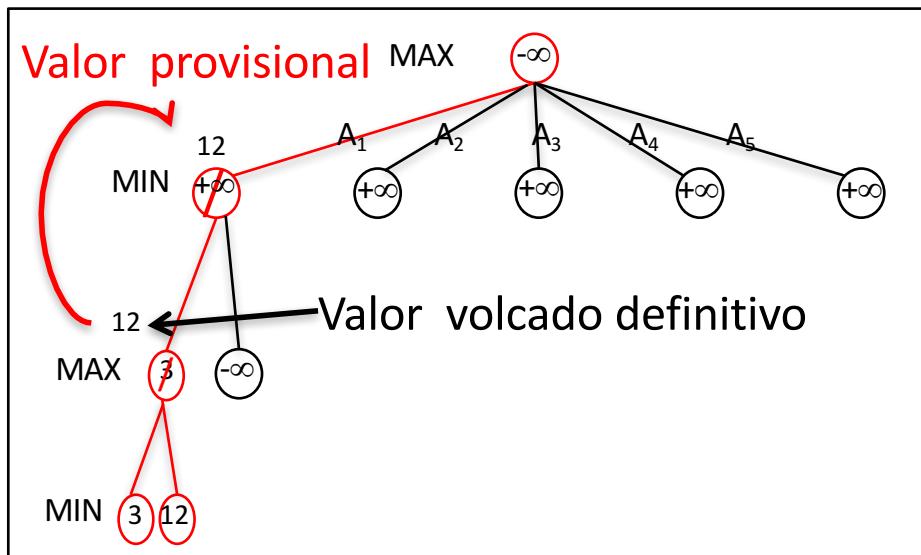
Desarrollamos la búsqueda en profundidad

¿Como calculamos los valores volcados?

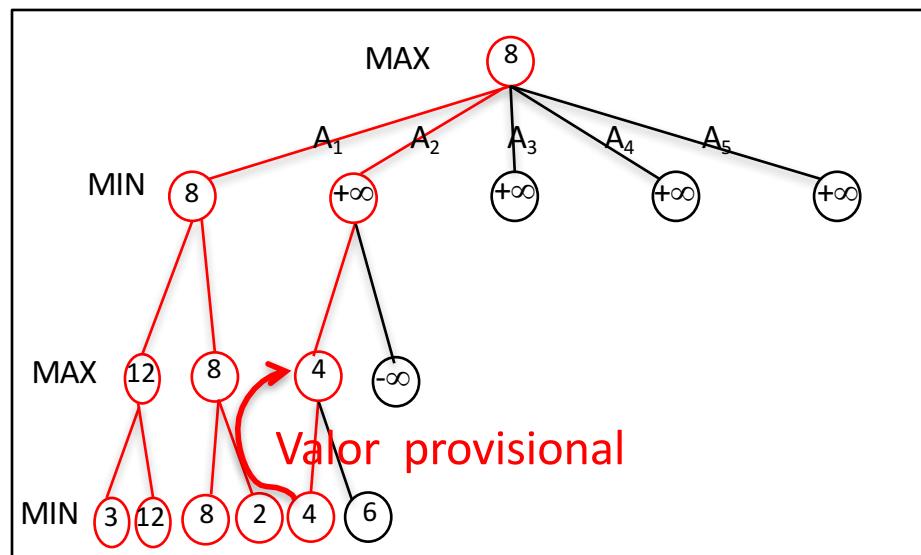
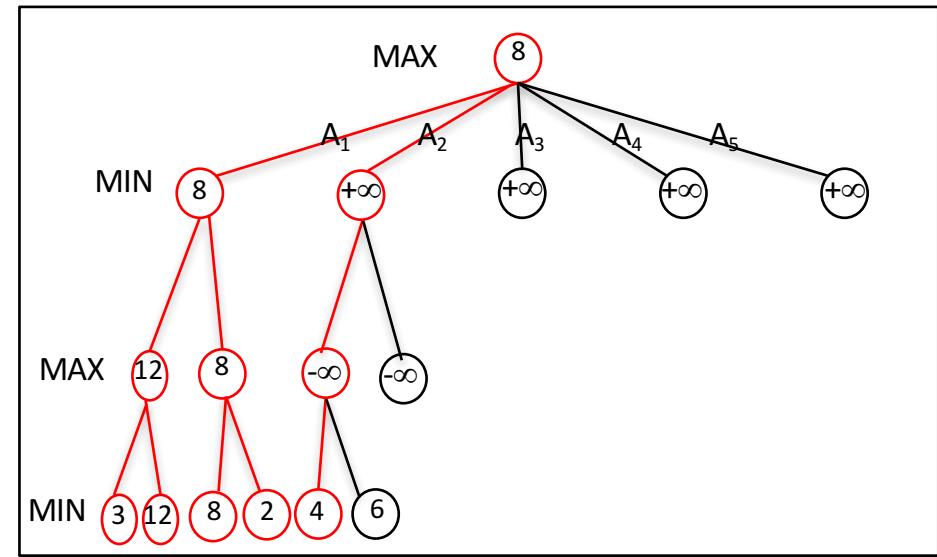
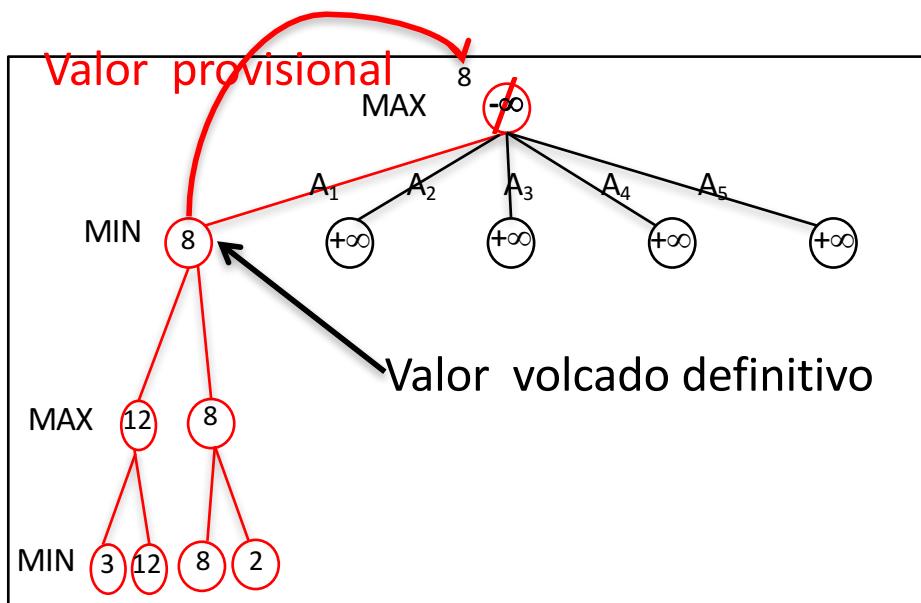
- Asumimos que los nodos MAX tienen un valor volcado inicial de  $-\infty$  y los nodos MIN de  $+\infty$
- Cuando un hijo de un nodo tenga un valor definitivo (porque es un nodo hoja o porque ya han sido explorados todos sus descendientes) volcamos su valor definitivo a su padre de la siguiente forma:
  - Si es un nodo MAX su nuevo valor será el máximo del valor que tiene y el de su hijo.
  - Si es un nodo MIN su nuevo valor será el mínimo del valor que tiene y el de su hijo.



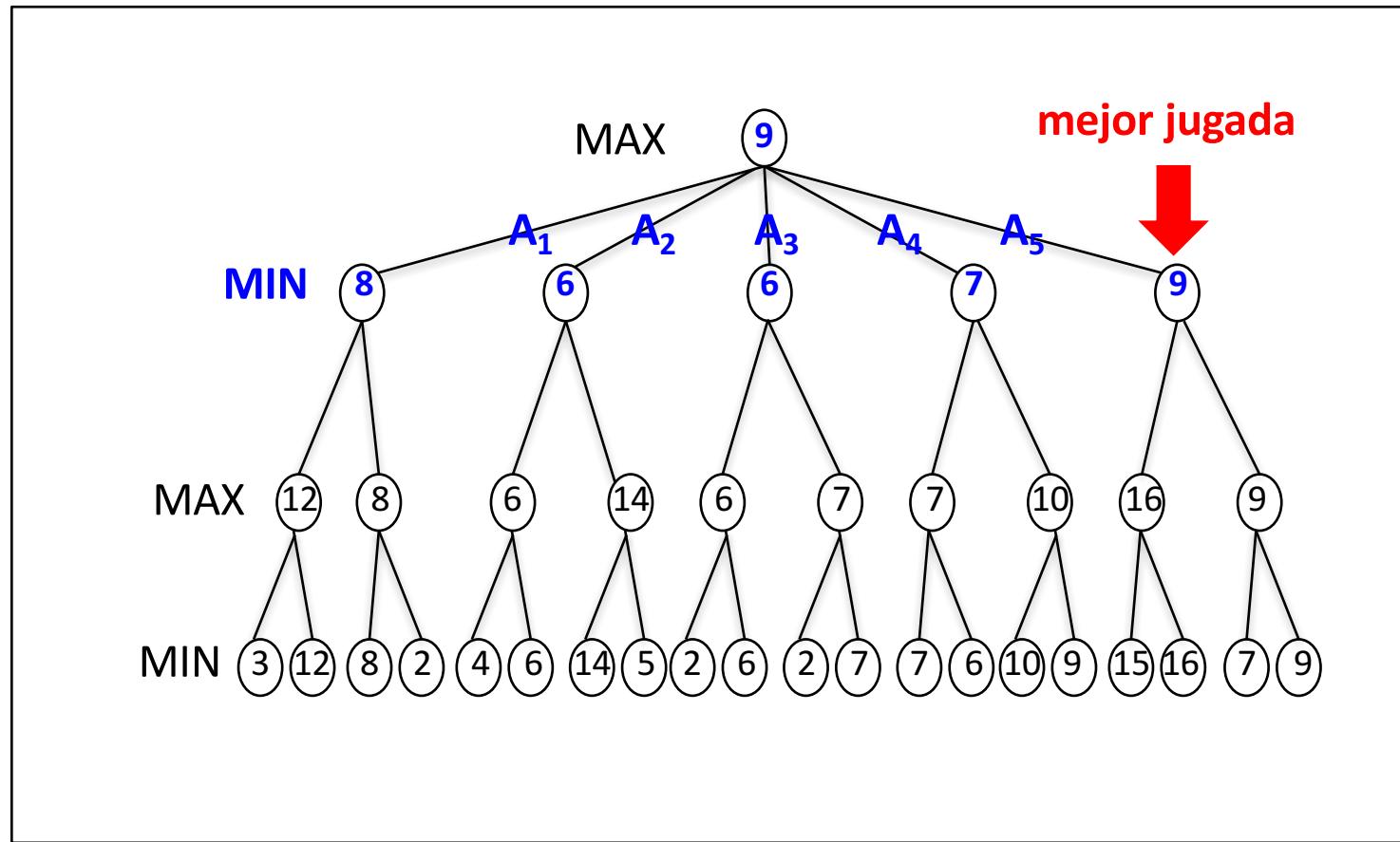
## 4. Algoritmo $\alpha$ - $\beta$ (3)



## 4. Algoritmo $\alpha$ - $\beta$ (4)



## 4. Algoritmo $\alpha$ - $\beta$ (5)



El espacio de búsqueda generado al desarrollar la búsqueda en profundidad es el mismo que desarrolla Mini-Max (amplitud), como ya sabíamos, y por lo tanto el resultado será el mismo. Hasta ahora lo único diferente es como hemos desarrollado la búsqueda y el método de cálculo de los valores volcados, pero los resultados son los mismos.

## 4. Algoritmo $\alpha$ - $\beta$ (6)

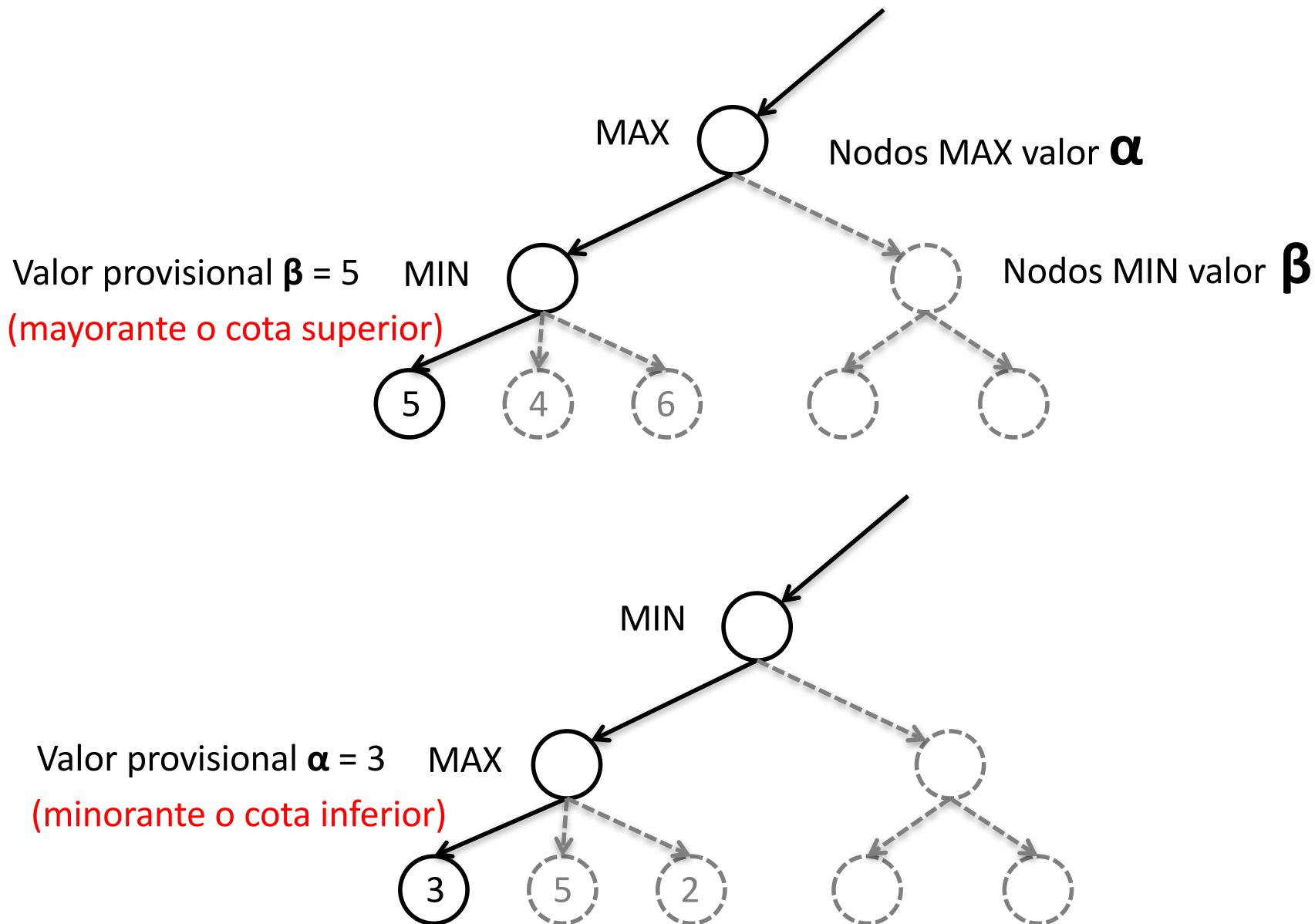
¿Se puede calcular la decisión Mini-Max correcta sin tener que examinar todos los nodos del árbol de búsqueda? ¿Es decir, se puede eliminar (podar) grandes partes del árbol?

El procedimiento MINIMAX separa completamente el proceso de generación del árbol del proceso de evaluación.

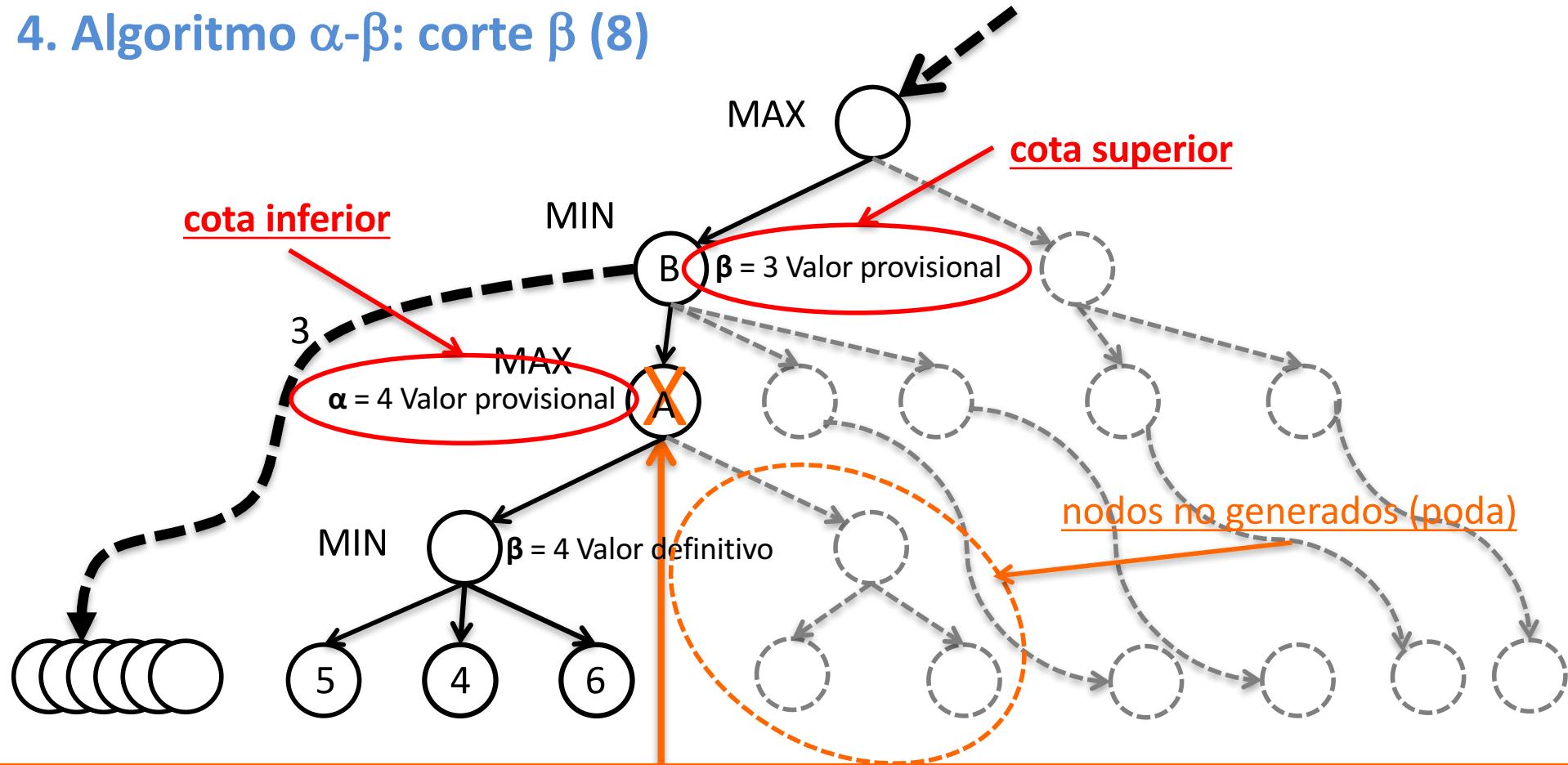
El método en profundidad que acabamos de ver genera un nodo terminal, y, simultáneamente, lo evalúa y vuelve el valor a su nodo padre antecesor (**generación y exploración en profundidad**)

**¿Tenemos algún criterio de poda en el método en profundidad que nos garantice calcular la decisión Mini-Max correcta?**

#### 4. Algoritmo $\alpha$ - $\beta$ : Poda (7)



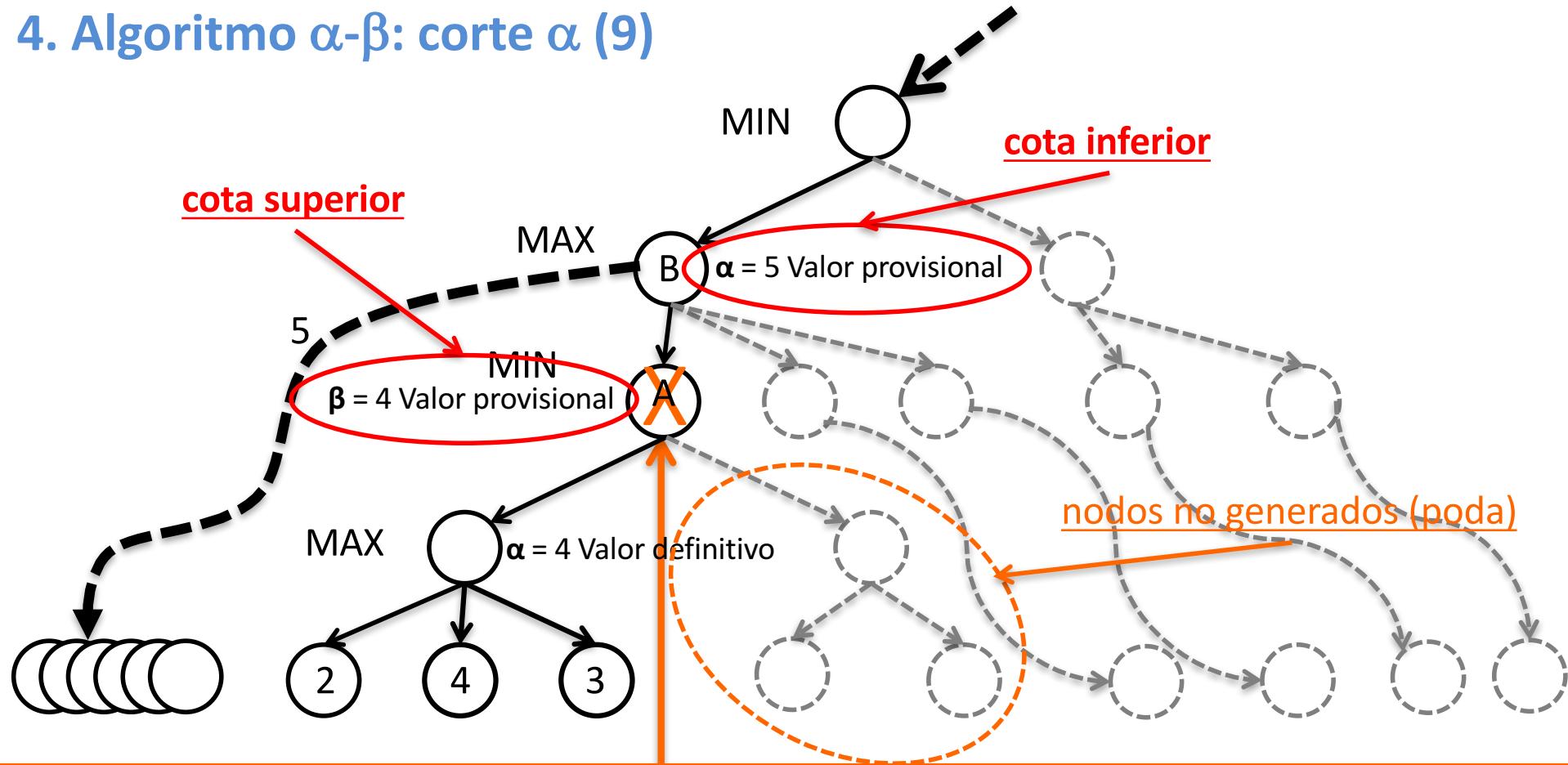
#### 4. Algoritmo $\alpha$ - $\beta$ : corte $\beta$ (8)



Si desarrollamos la búsqueda por debajo del nodo A podemos obtener un valor  $\alpha$  mayor o igual que el valor provisional actual, su nodo MIN padre (B) tiene un valor provisional  $\beta = 3$  y su valor definitivo será el menor entre su valor provisional y los valores volcados que puedan llegar de sus hijos, por lo tanto desarrollar la búsqueda por debajo de A podría dar lugar a un valor  $\alpha$ , mayor o igual que el actual, que no cambiará el valor provisional  $\beta$  actual del nodo B → podemos podar la búsqueda por debajo de A

**Corte  $\beta$ :**  $\alpha$  provisional  $\geq \beta$  predecessor (no sólo padre)

#### 4. Algoritmo $\alpha$ - $\beta$ : corte $\alpha$ (9)



Si desarrollamos la búsqueda por debajo del nodo A podremos obtener un valor  $\beta$  menor o igual que el valor provisional actual, su nodo MAX padre (B) tiene un valor provisional  $\alpha = 5$  y su valor definitivo será el mayor entre su valor provisional y los valores volcados que puedan llegar de sus hijos, por lo tanto desarrollar la búsqueda por debajo de A podría dar lugar a un valor  $\beta$ , menor o igual que el actual, que no cambiará el valor provisional  $\alpha$  actual del nodo B → podemos podar la búsqueda por debajo de A

**Corte  $\alpha$ :**  $\beta$  provisional  $\leq \alpha$  predecessor (no sólo padre)

## 4. Algoritmo $\alpha$ - $\beta$ (10)

1) Inicialmente:

Nodos MAX: valores  $\alpha$  (inicialmente  $\alpha = -\infty$ )

Nodos MIN: valores  $\beta$  (inicialmente  $\beta = +\infty$ )

2) Generar un nodo terminal en profundidad. Calcular su valor de utilidad.

3) Volcar el valor al nodo padre y mantener el mejor valor para el padre.

4) Pregunta de corte. Cuando se vuelca un valor a un nodo (MAX o MIN), este valor se compara con los valores de todos los nodos predecesores contrarios con objeto de ver si se produce un corte  $\alpha$  o  $\beta$ .

– Si se produce un corte, ir a 3

5) Valor provisional/definitivo:

– Si es un valor provisional, ir a 2  
– Si es un valor definitivo, ir a 3

## 4. Poda $\alpha$ - $\beta$ : características (11)

Los nodos MAX tienen valores  $\alpha$ :

- Tienen garantizado conseguir, a partir de ese nodo, un estado con un valor  $\geq \alpha$ .
- Los valores  $\alpha$  provisionales de un nodo MAX son cotas inferiores del nodo y nunca pueden disminuir al desarrollar la búsqueda.
- El valor  $\alpha$  de un nodo no terminal es el mayor valor entre los valores de sus nodos sucesores.

Los nodos MIN tienen valores  $\beta$ :

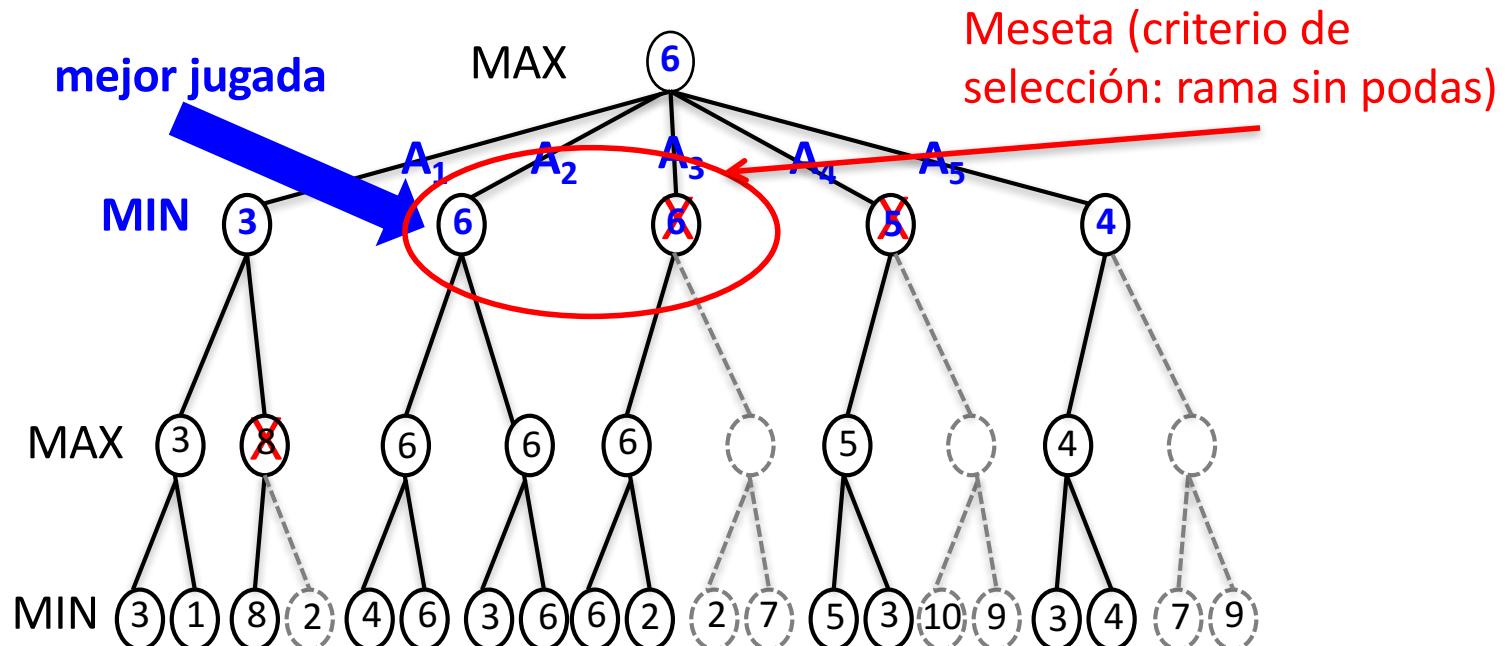
- Tienen garantizado conseguir, a partir de ese nodo, un estado con un valor  $\leq \beta$ .
- Los valores  $\beta$  provisionales de un nodo MIN son cotas superiores del nodo, y nunca pueden aumentar al desarrollar la búsqueda.
- El valor  $\beta$  de un nodo no terminal es el menor valor entre los valores de sus nodos sucesores.

## 4. Poda $\alpha$ - $\beta$ : características (12)

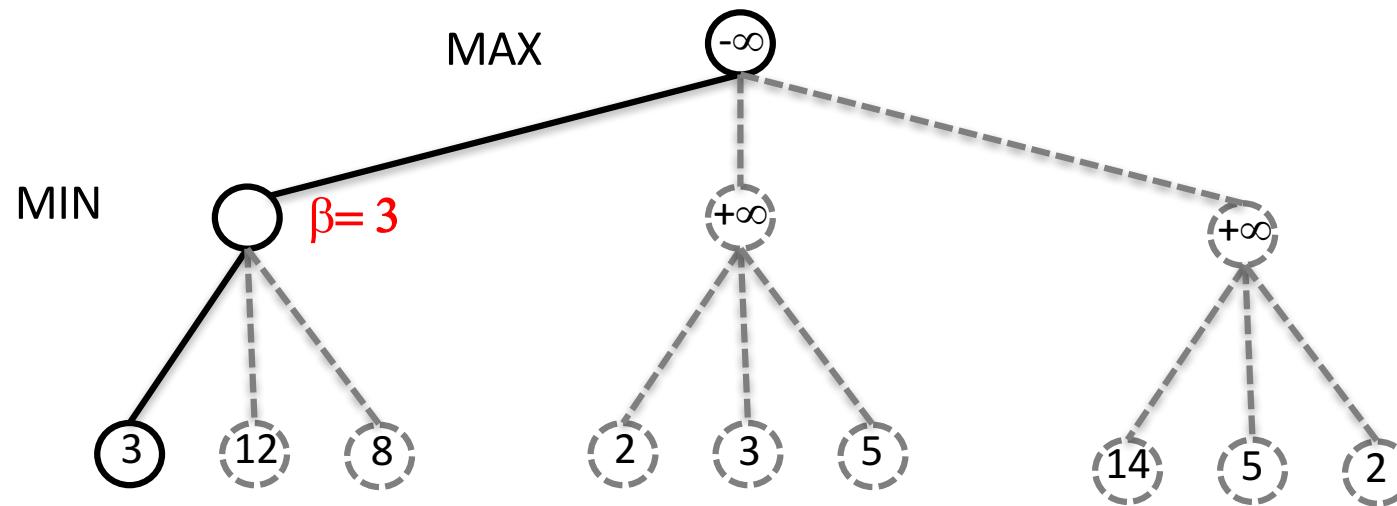
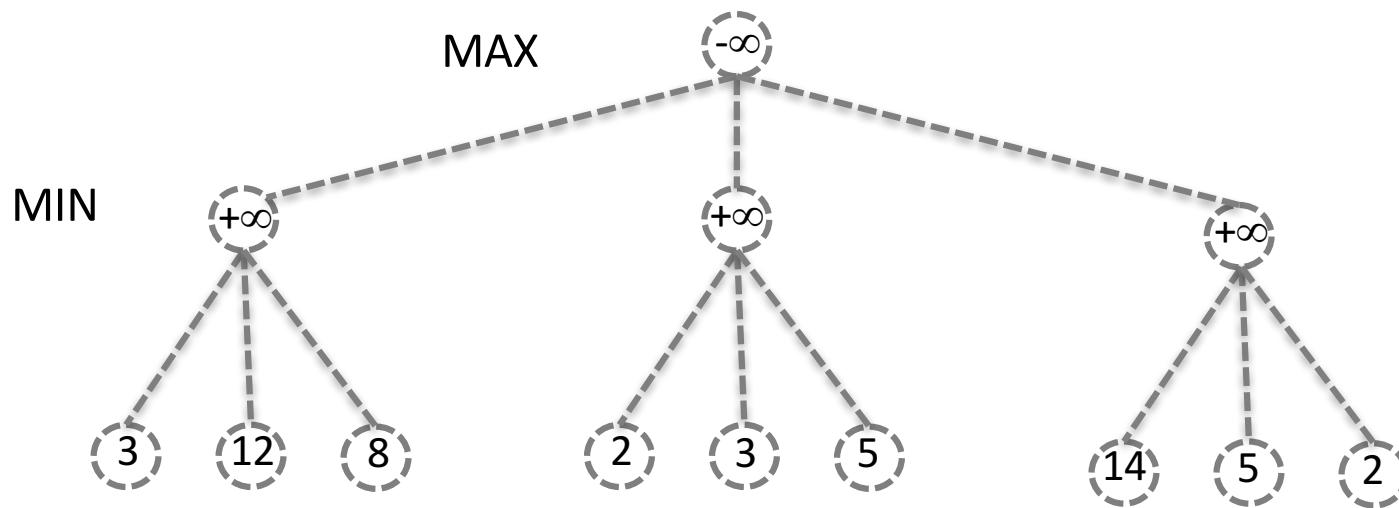
La búsqueda puede detenerse (corte) en cualquier nodo que:

- Cualquier nodo MIN que tenga un valor  $\beta \leq$  que el valor  $\alpha$  de un nodo MAX predecesor: corte  $\alpha$ .
- Cualquier nodo MAX que tenga un valor  $\alpha \geq$  que el valor  $\beta$  de un nodo MIN predecesor: corte  $\beta$

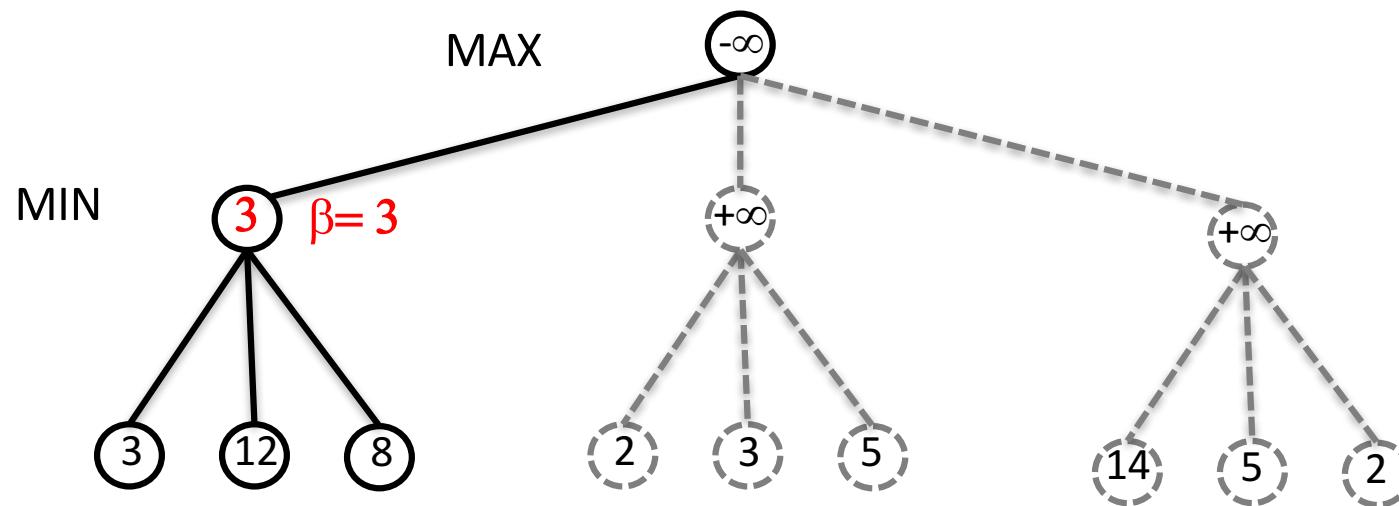
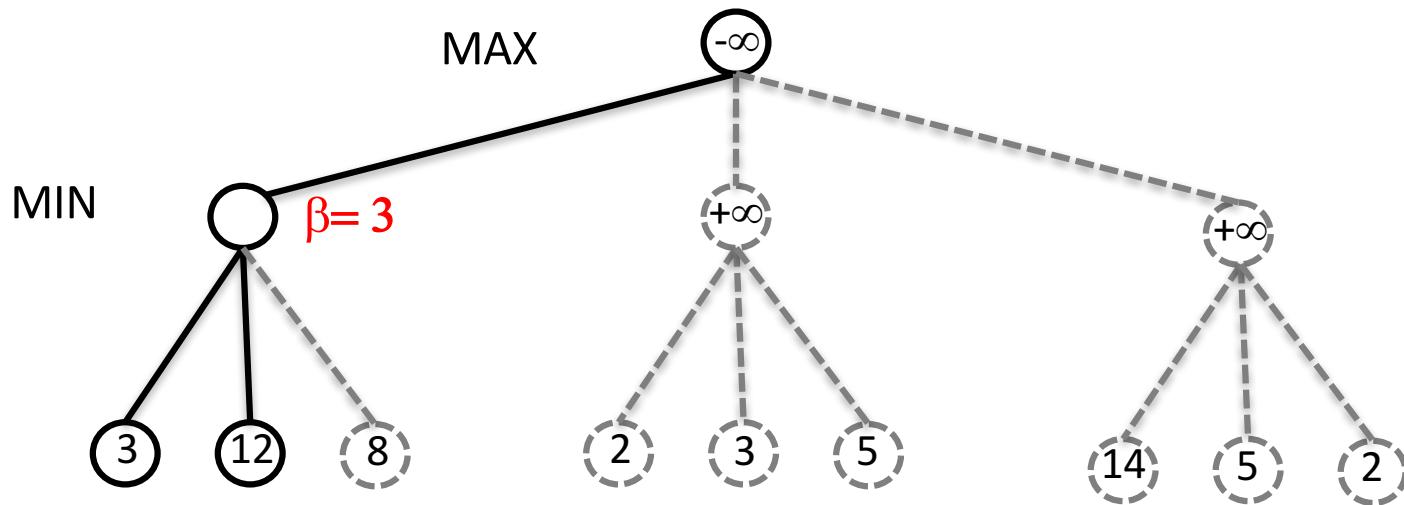
Los valores finales de los nodos en el procedimiento  $\alpha$ - $\beta$  son iguales que en el procedimiento MINIMAX, excepto aquellos nodos cuyos valores provengan de cortes  $\alpha$  (podría ser menor) o  $\beta$  (podría ser mayor).



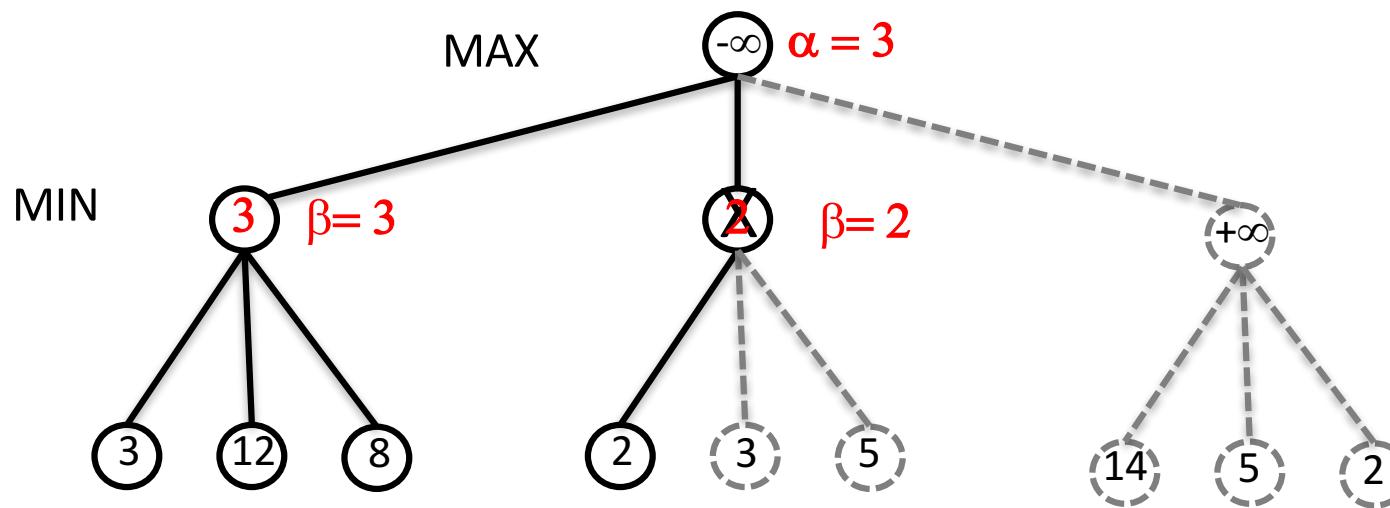
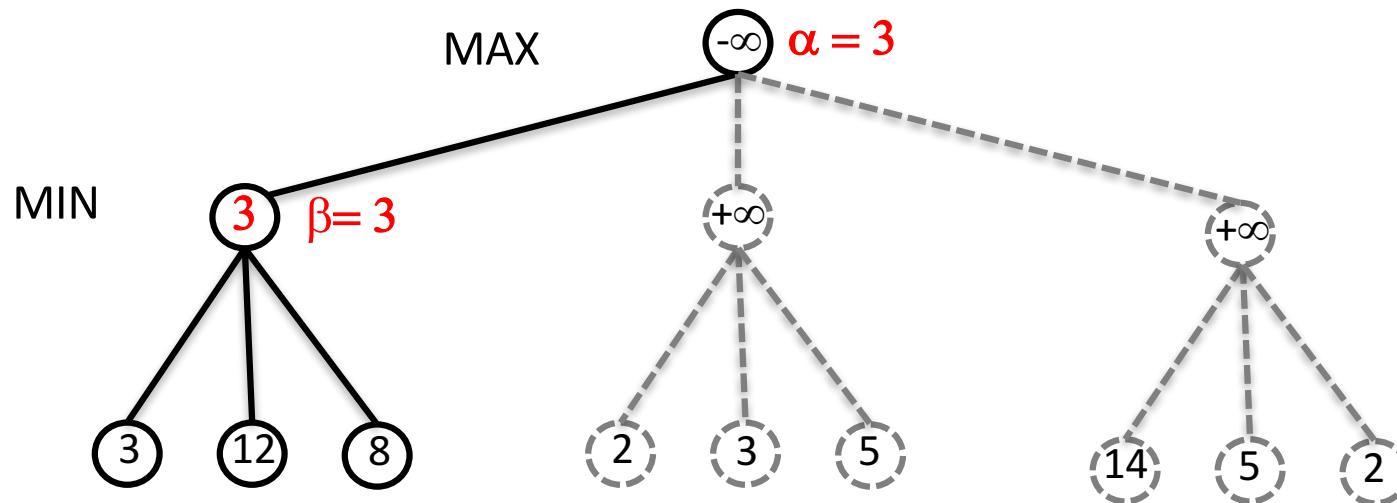
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 1



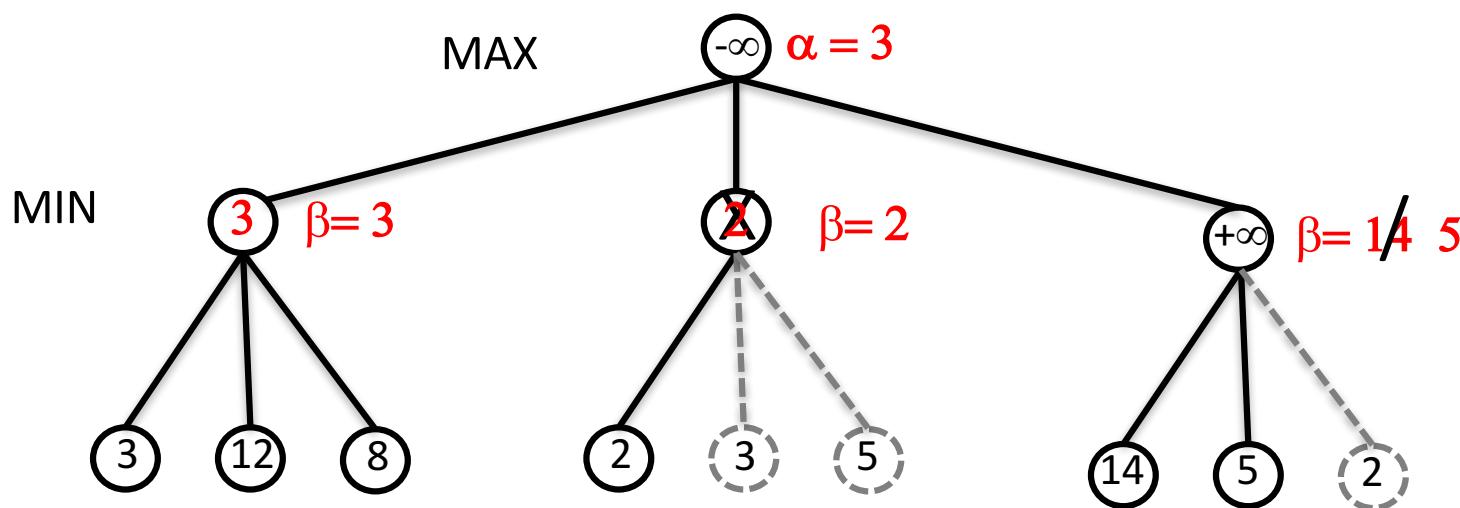
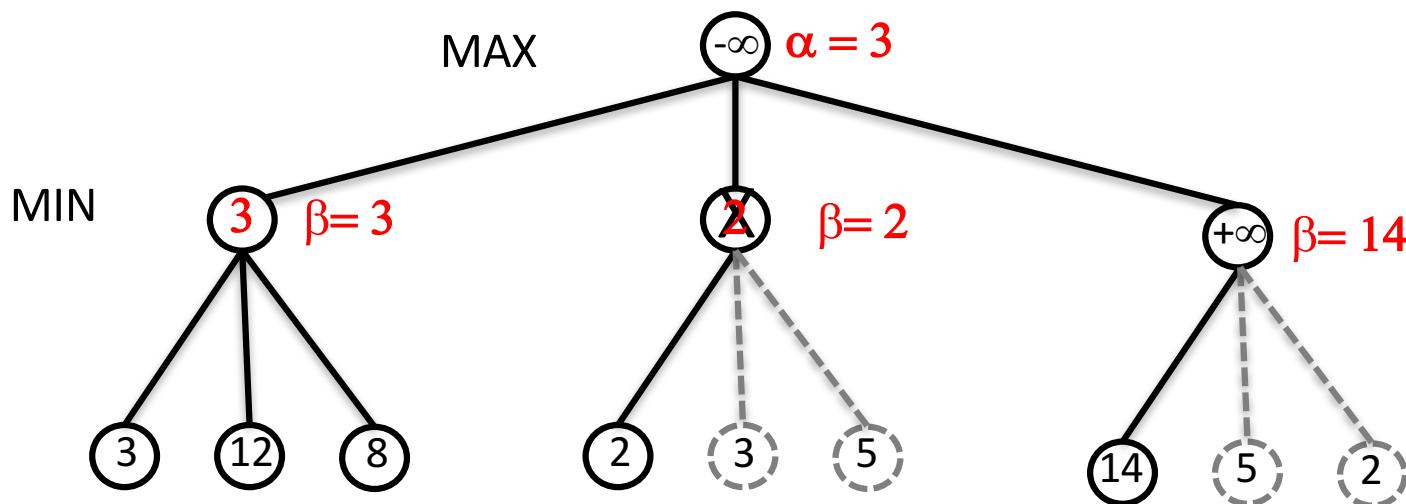
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 1 (2)



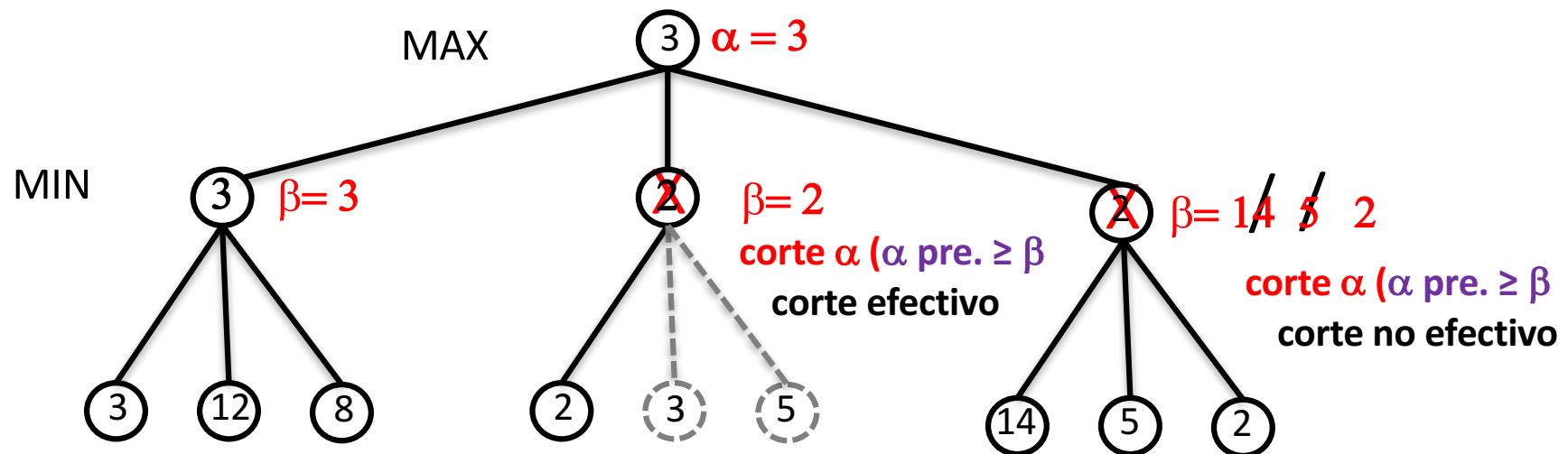
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 1 (3)



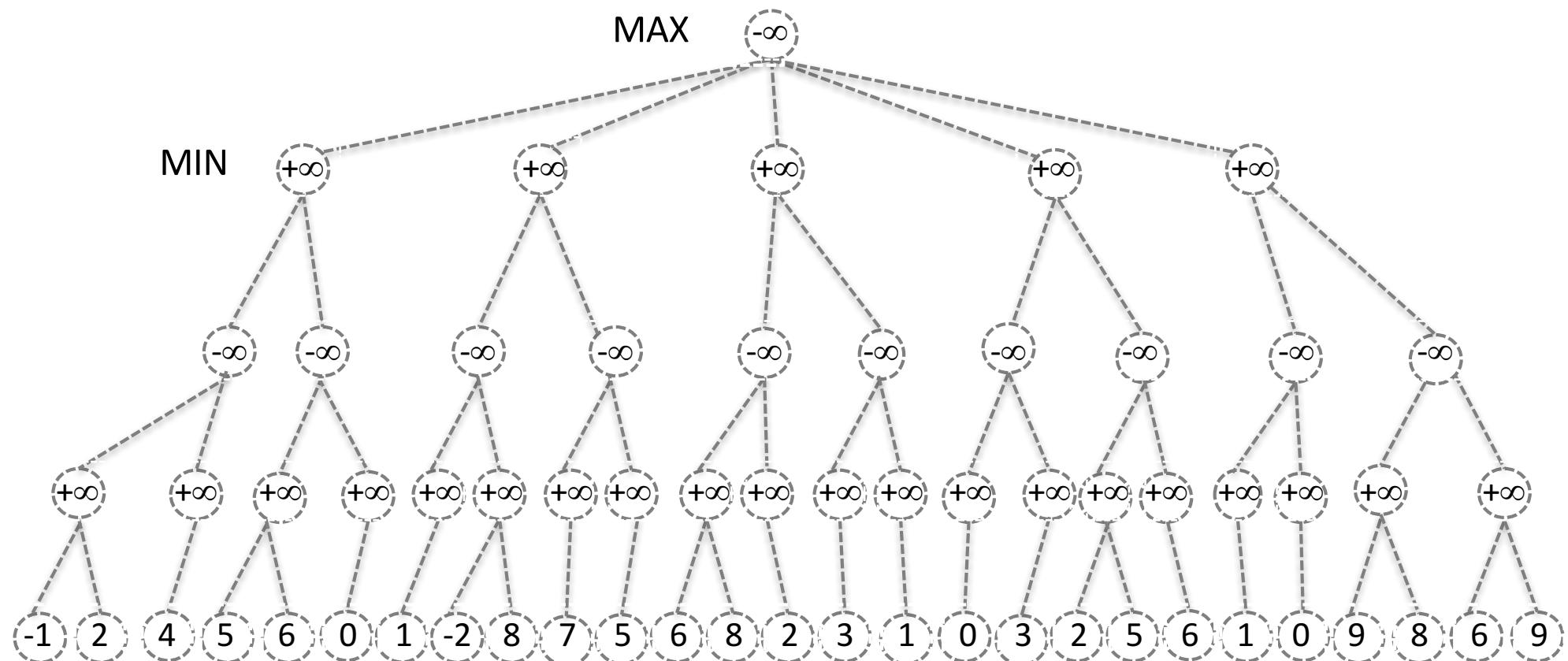
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 1 (4)



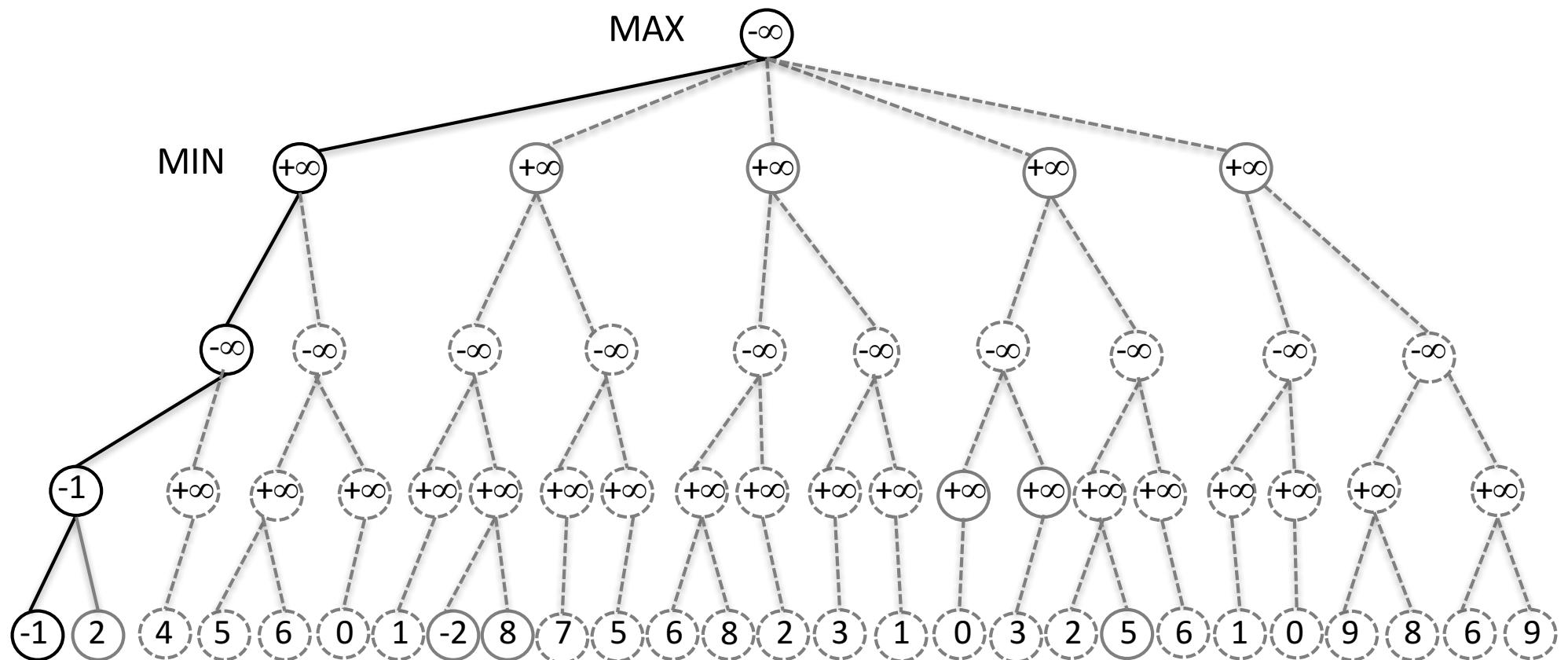
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 1 (5)



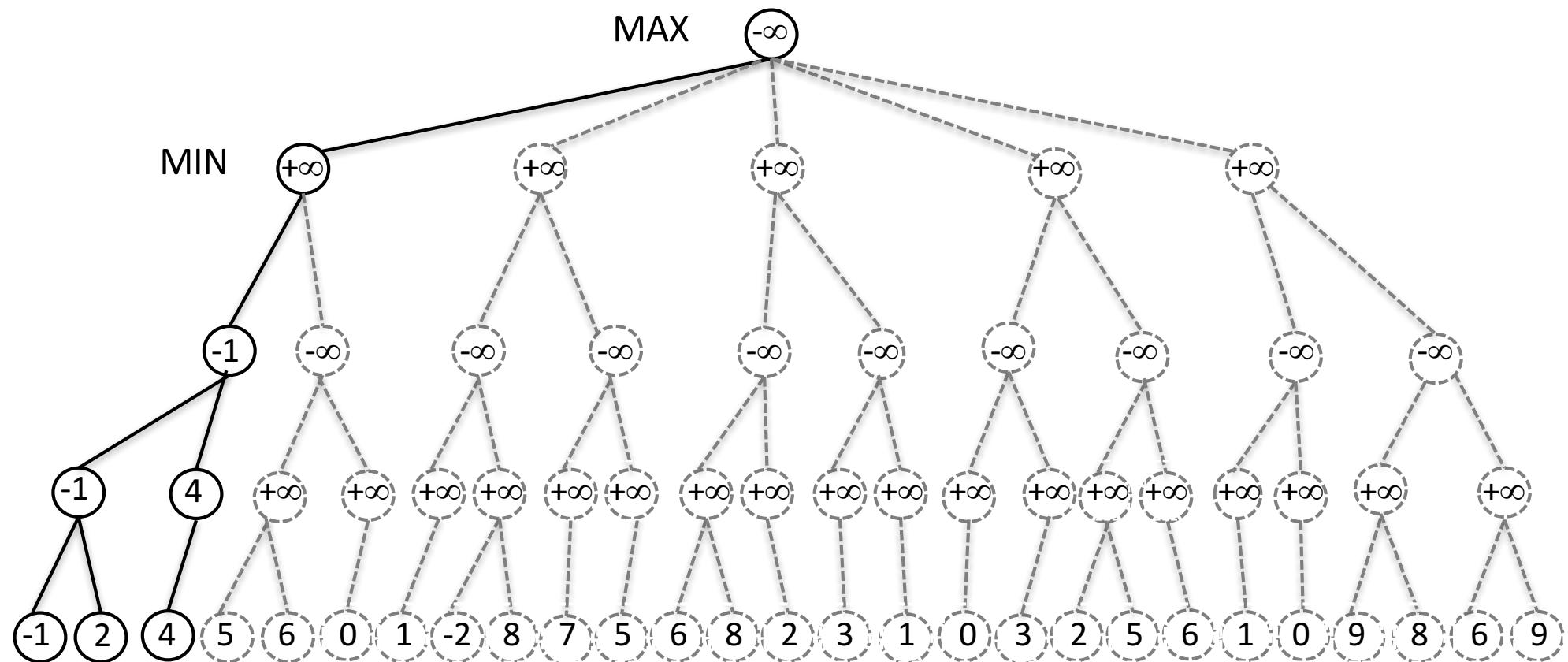
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



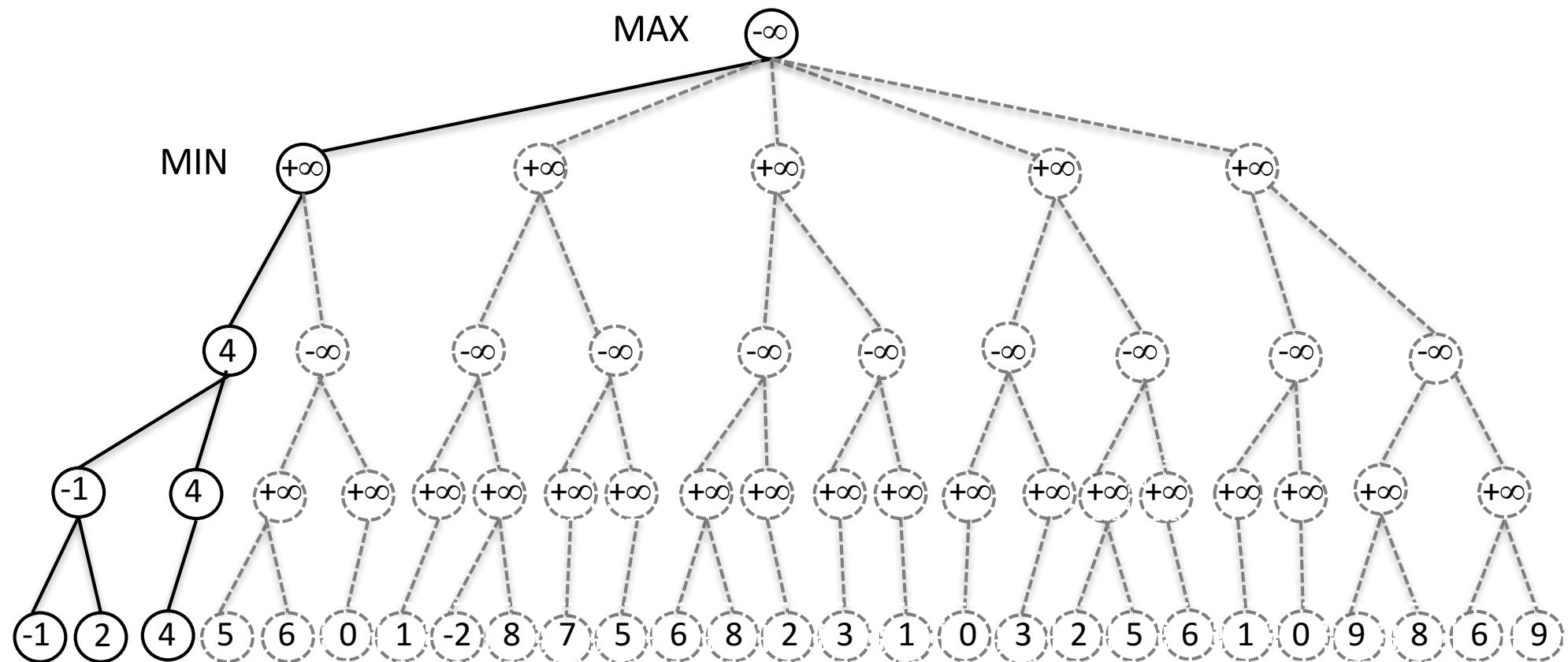
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



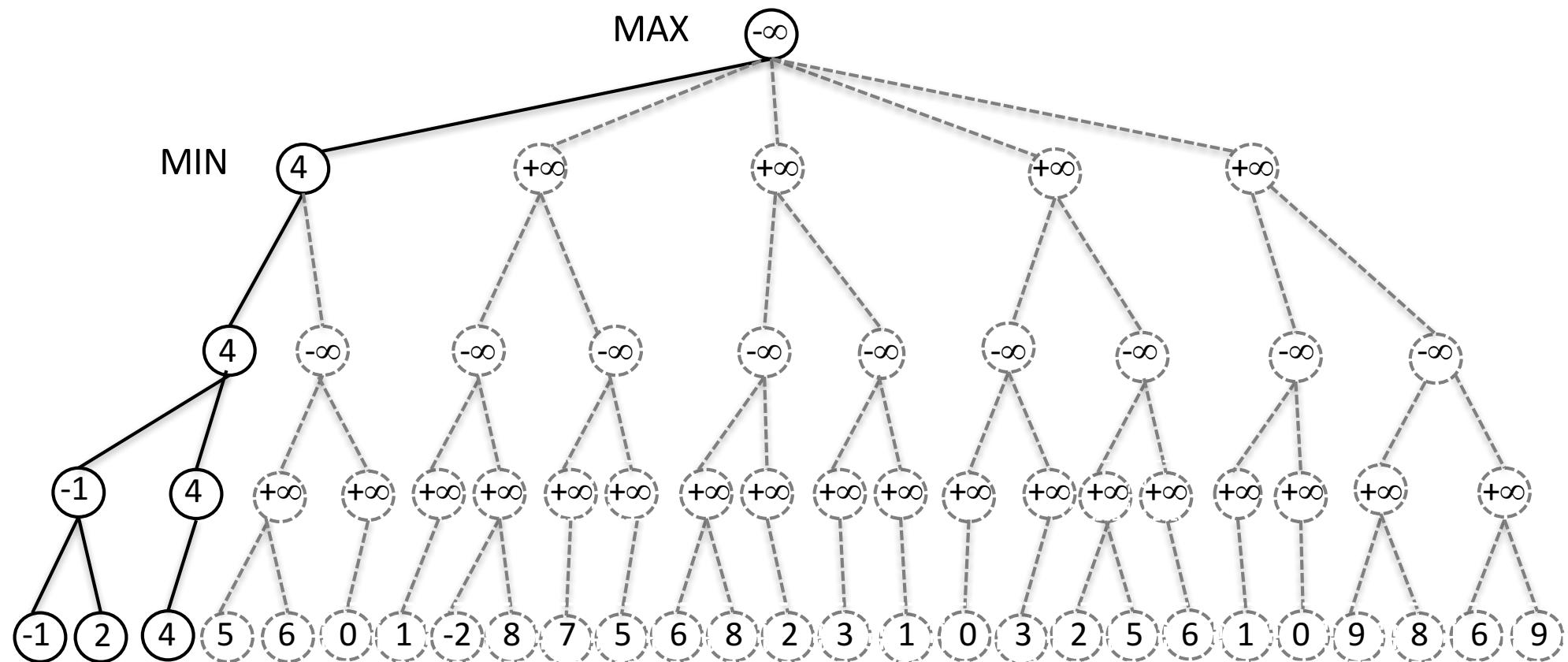
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



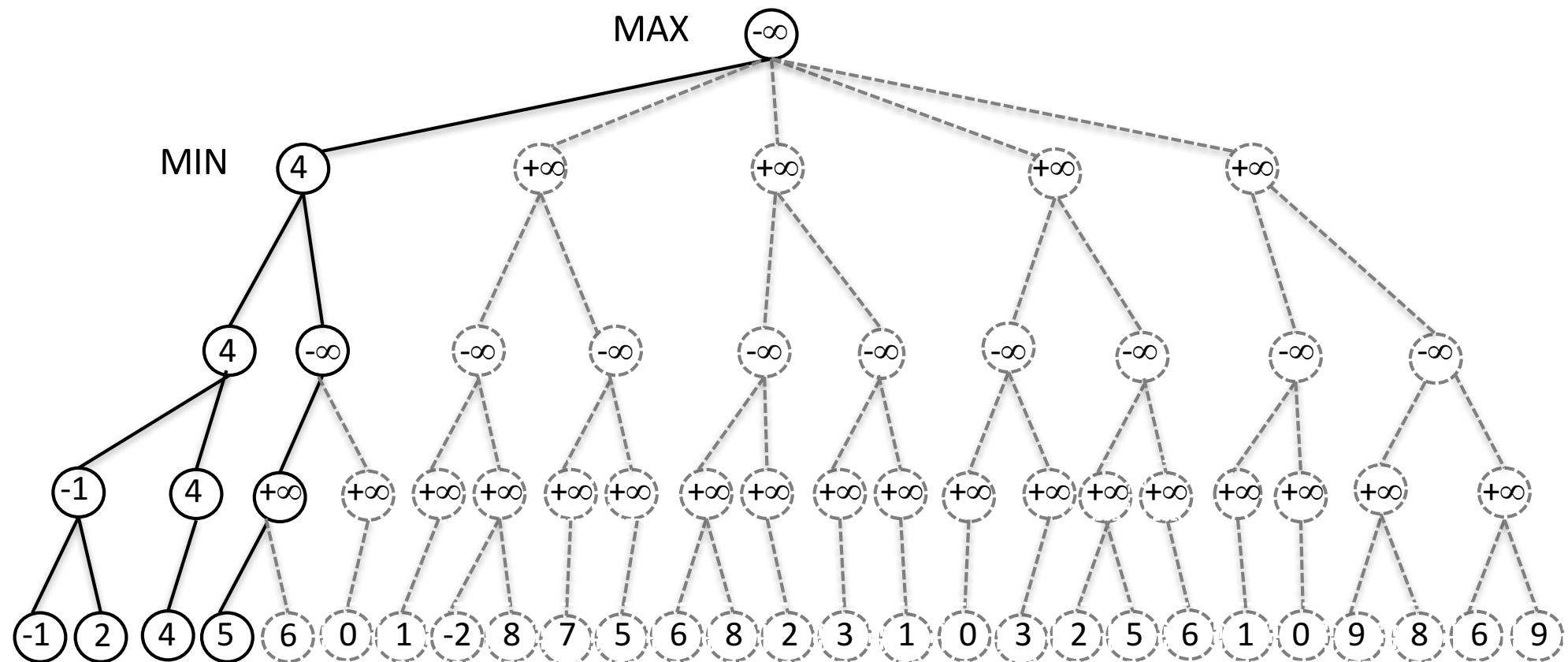
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



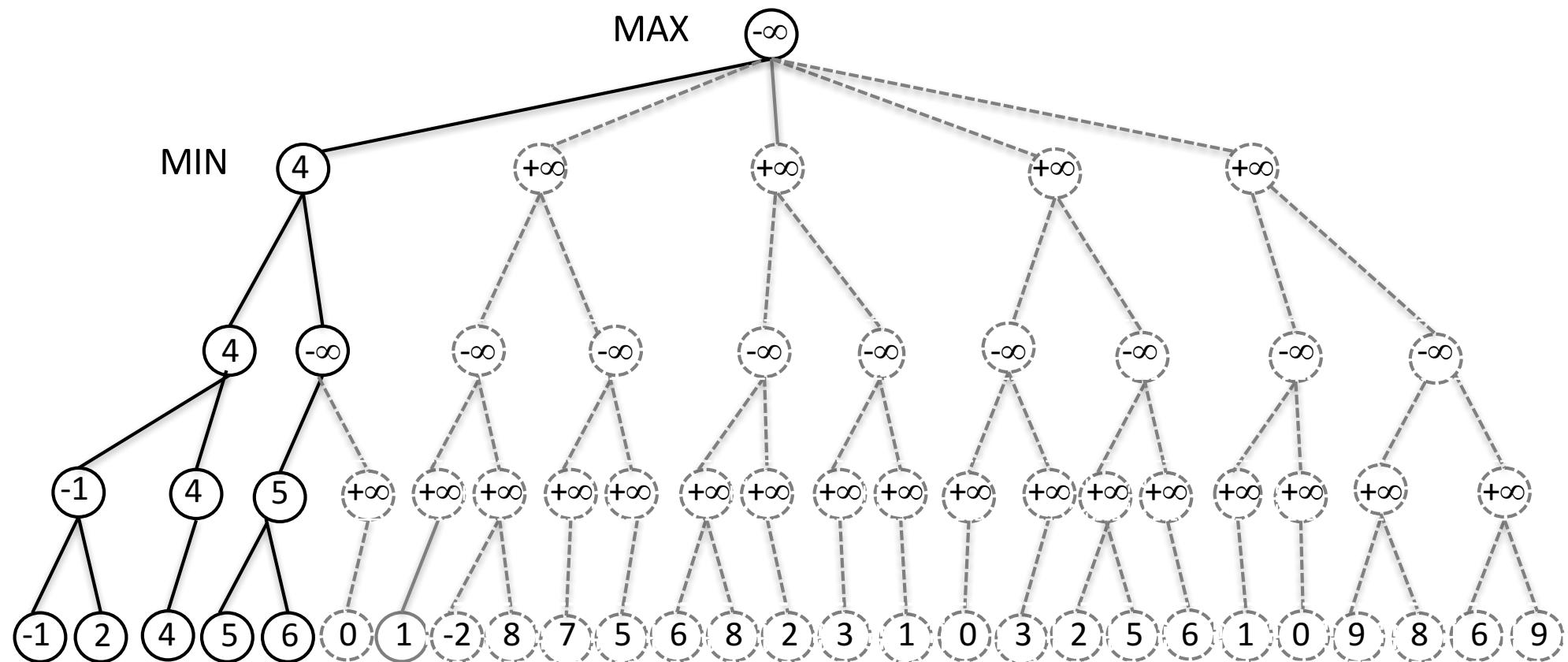
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



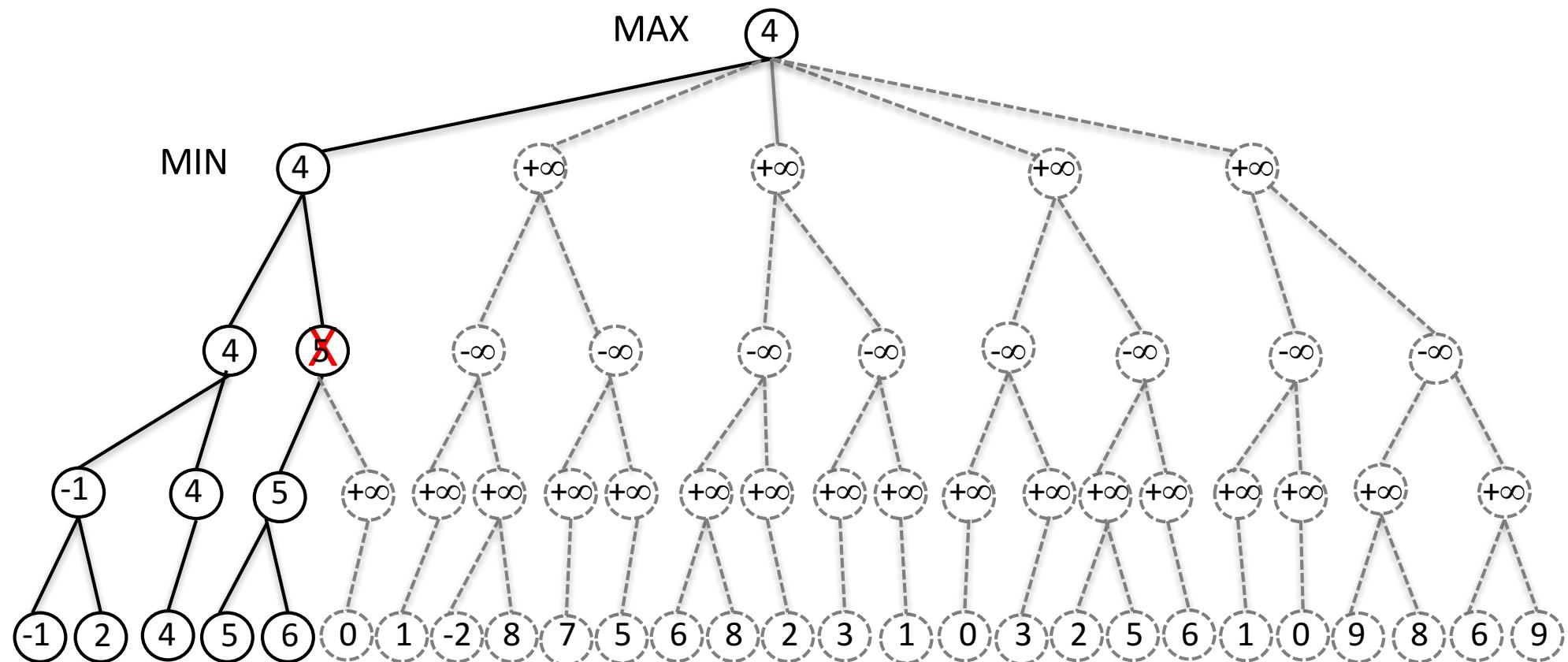
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



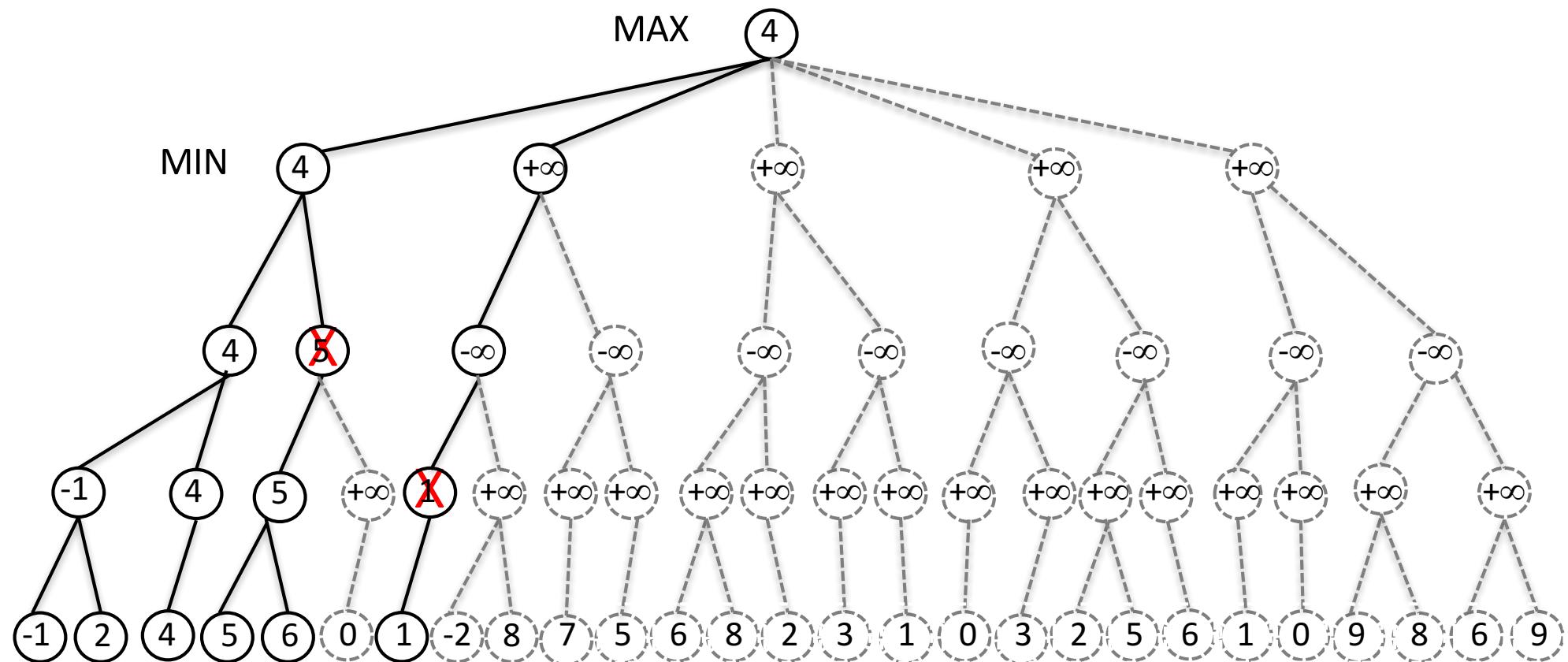
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



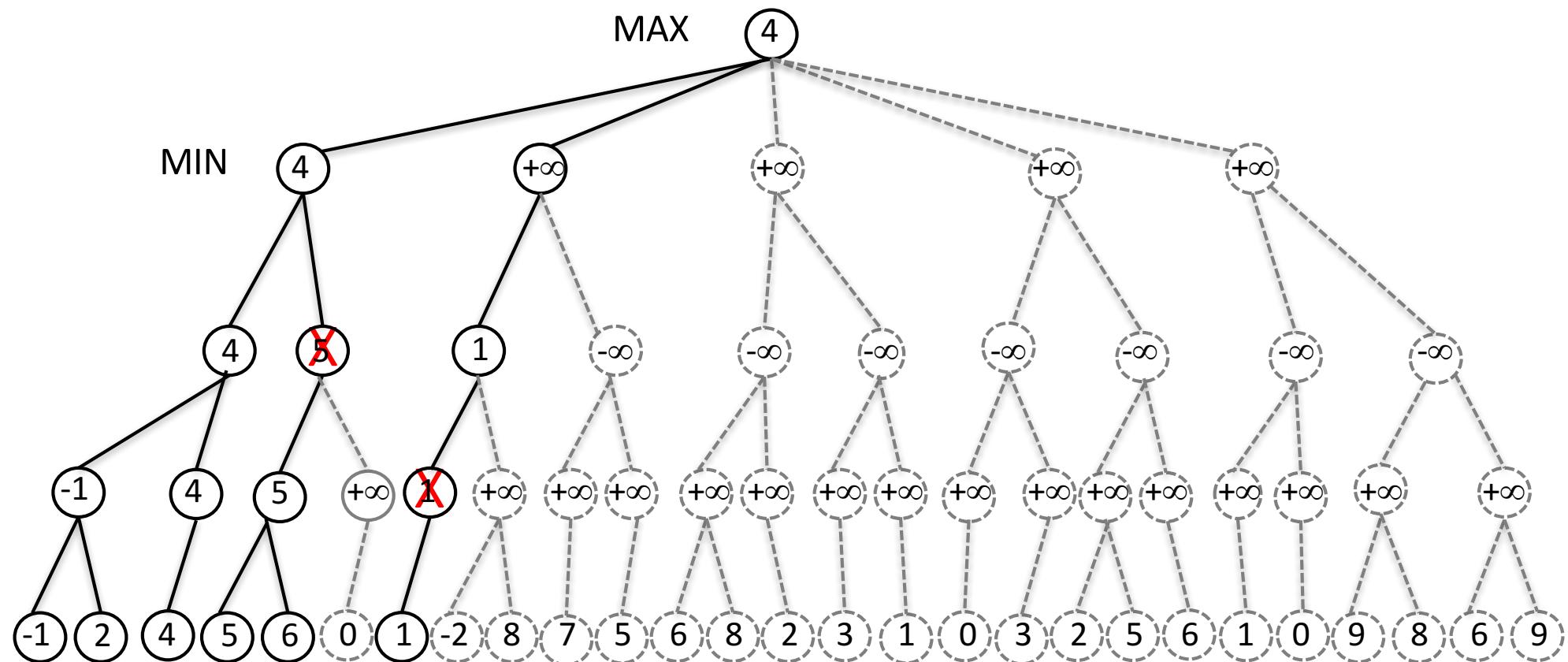
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



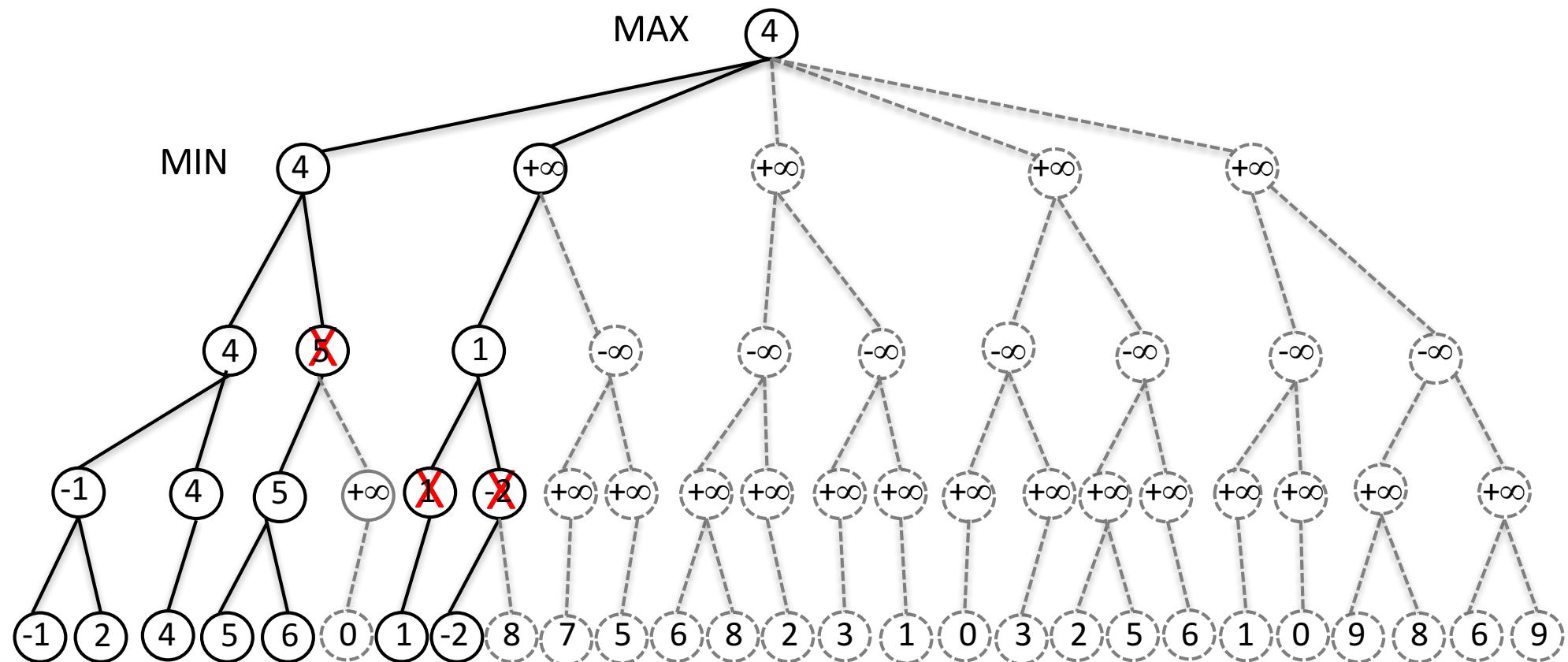
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



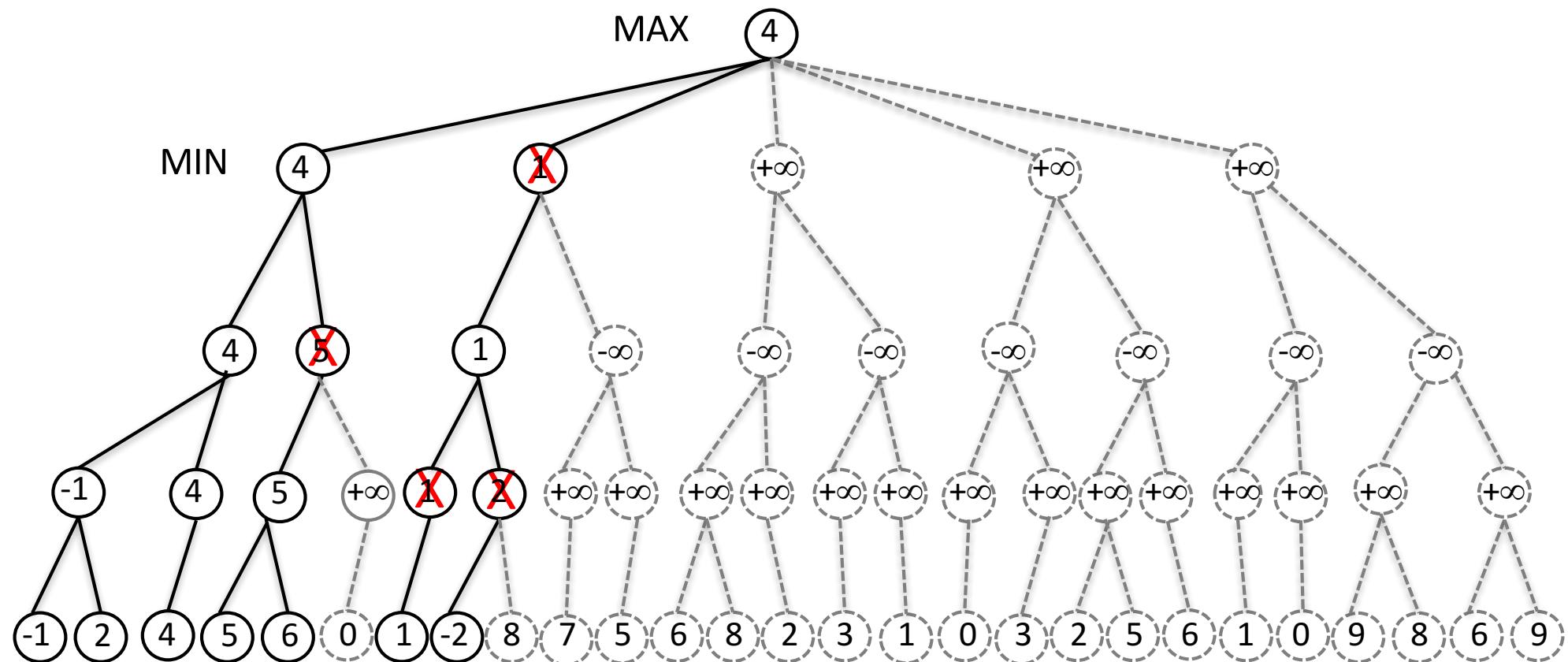
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



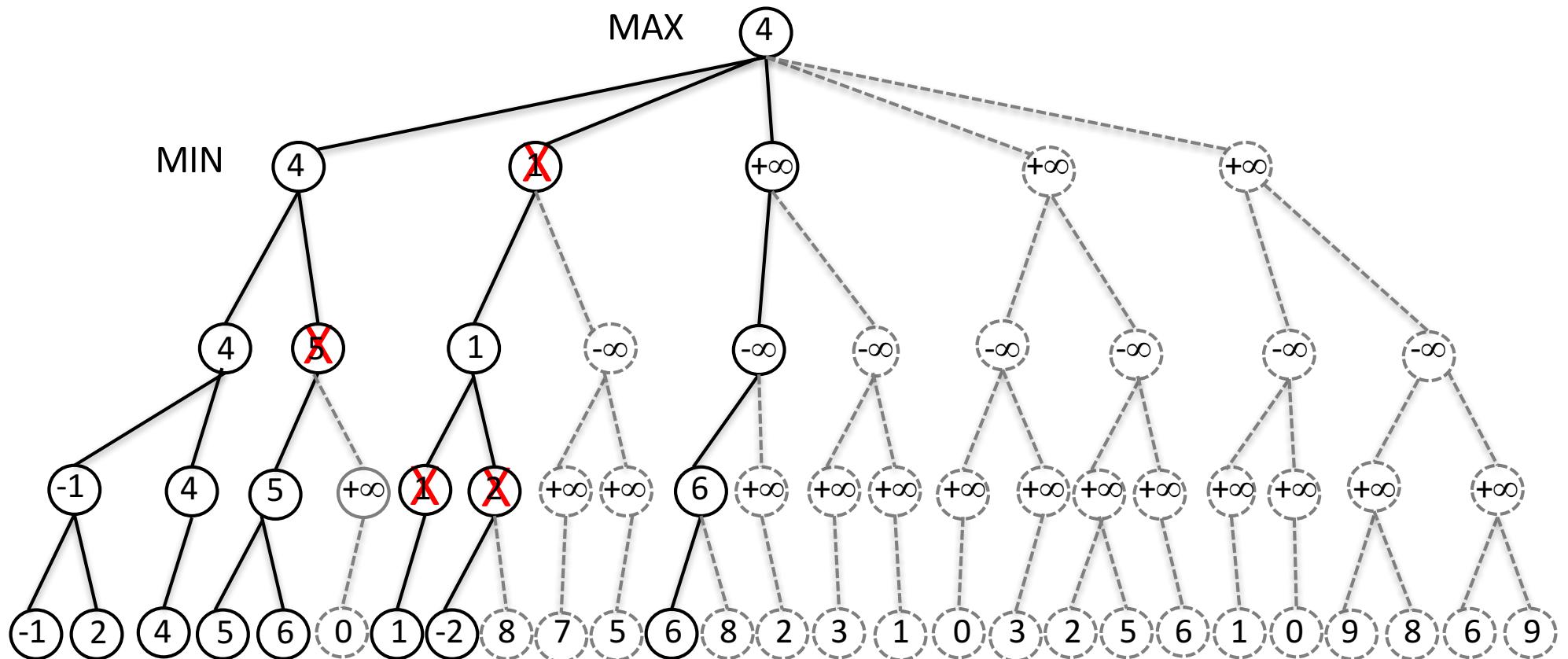
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



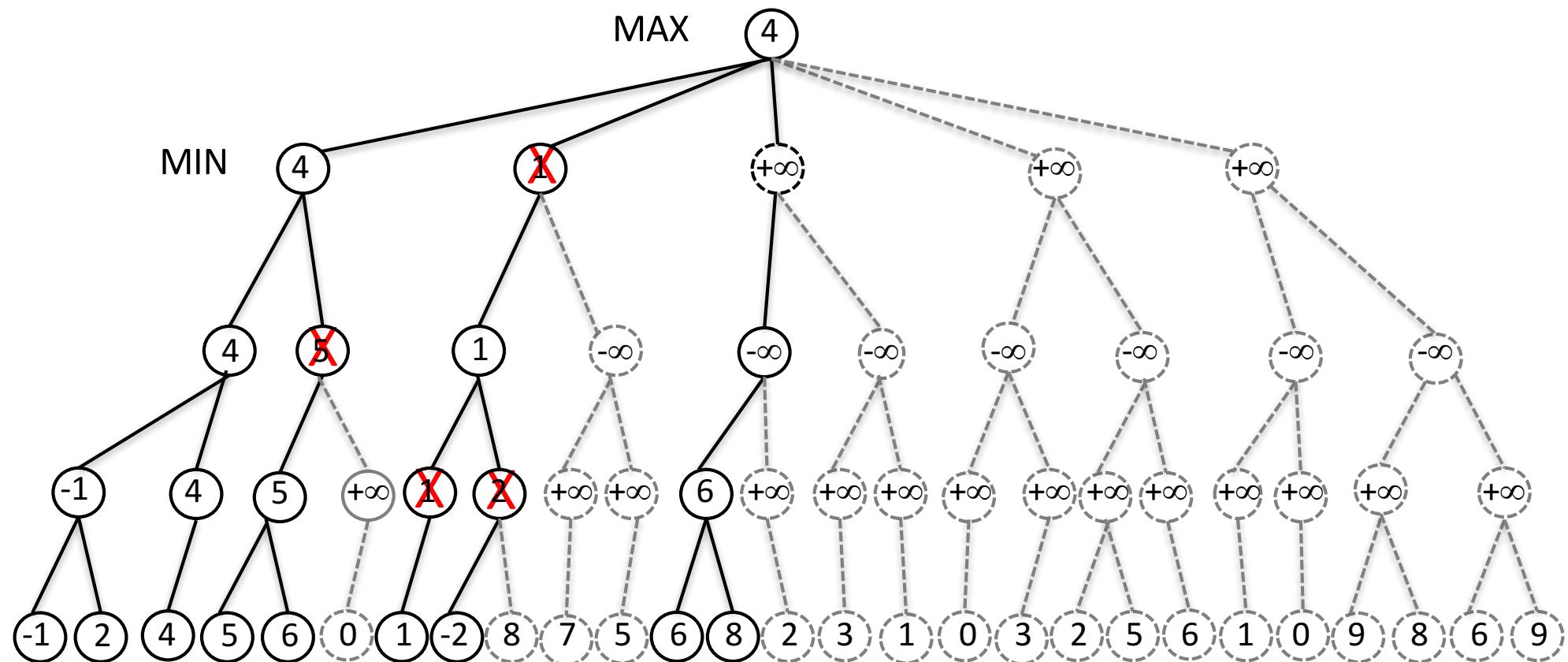
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



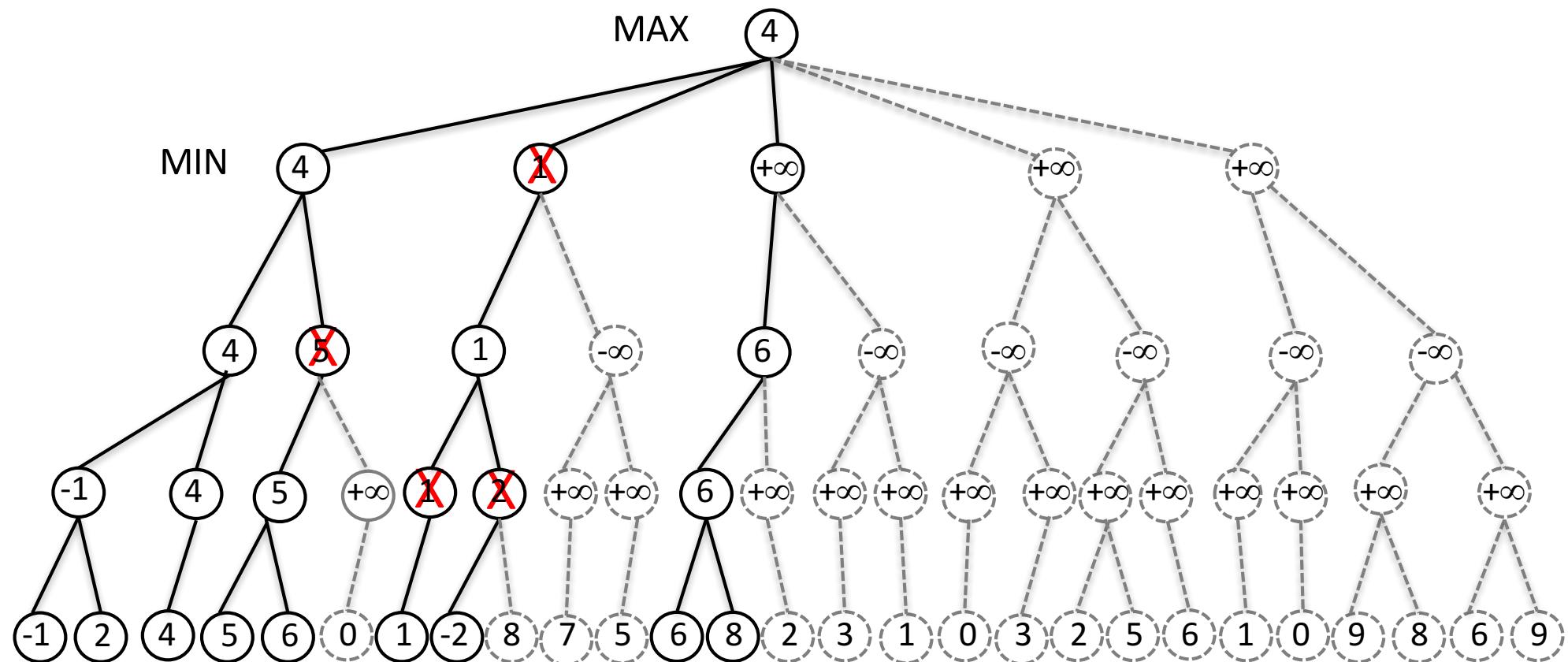
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



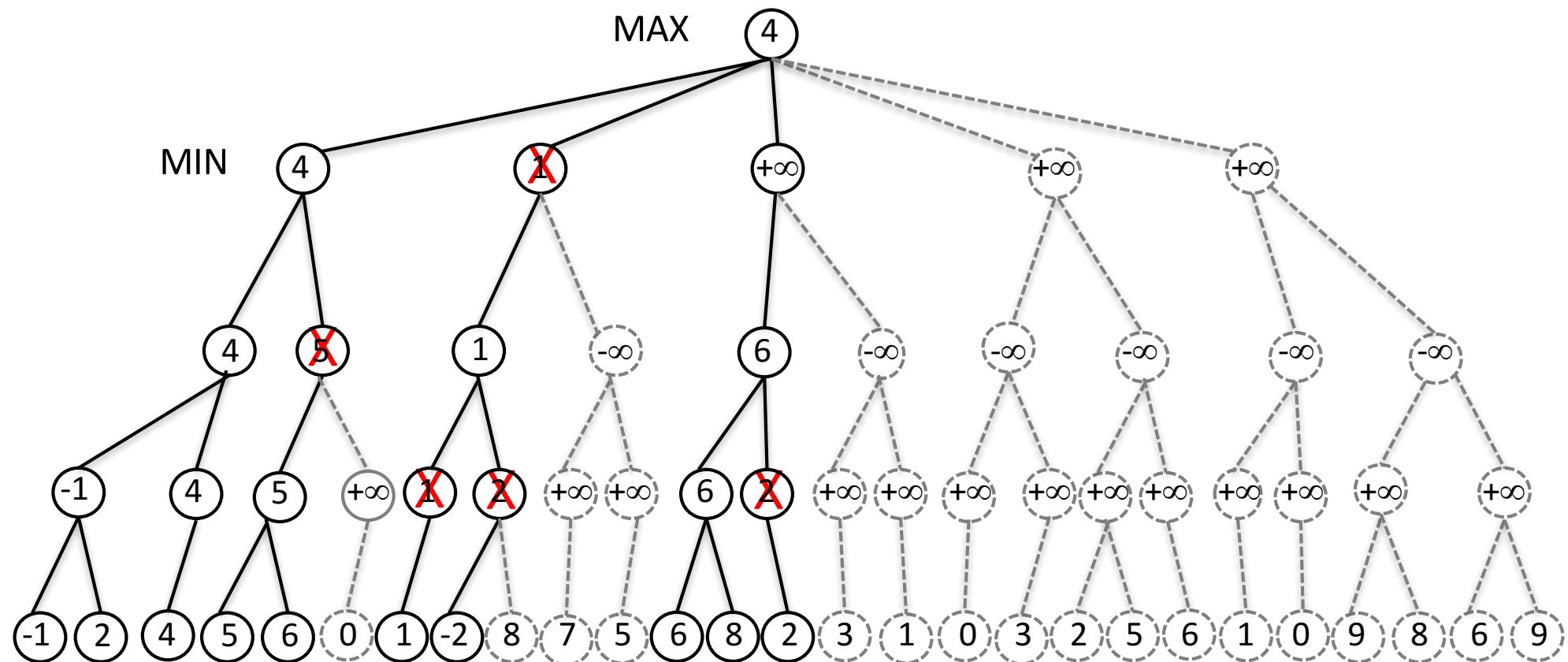
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



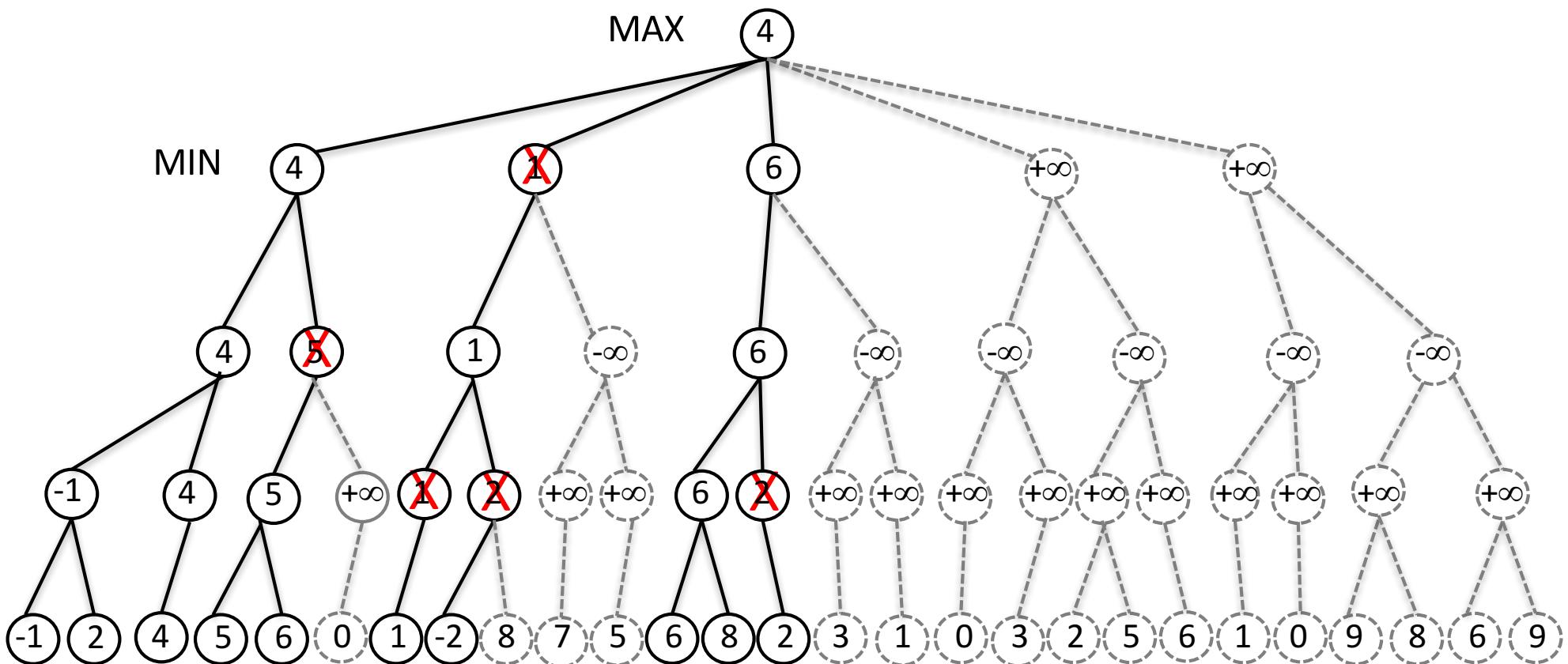
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



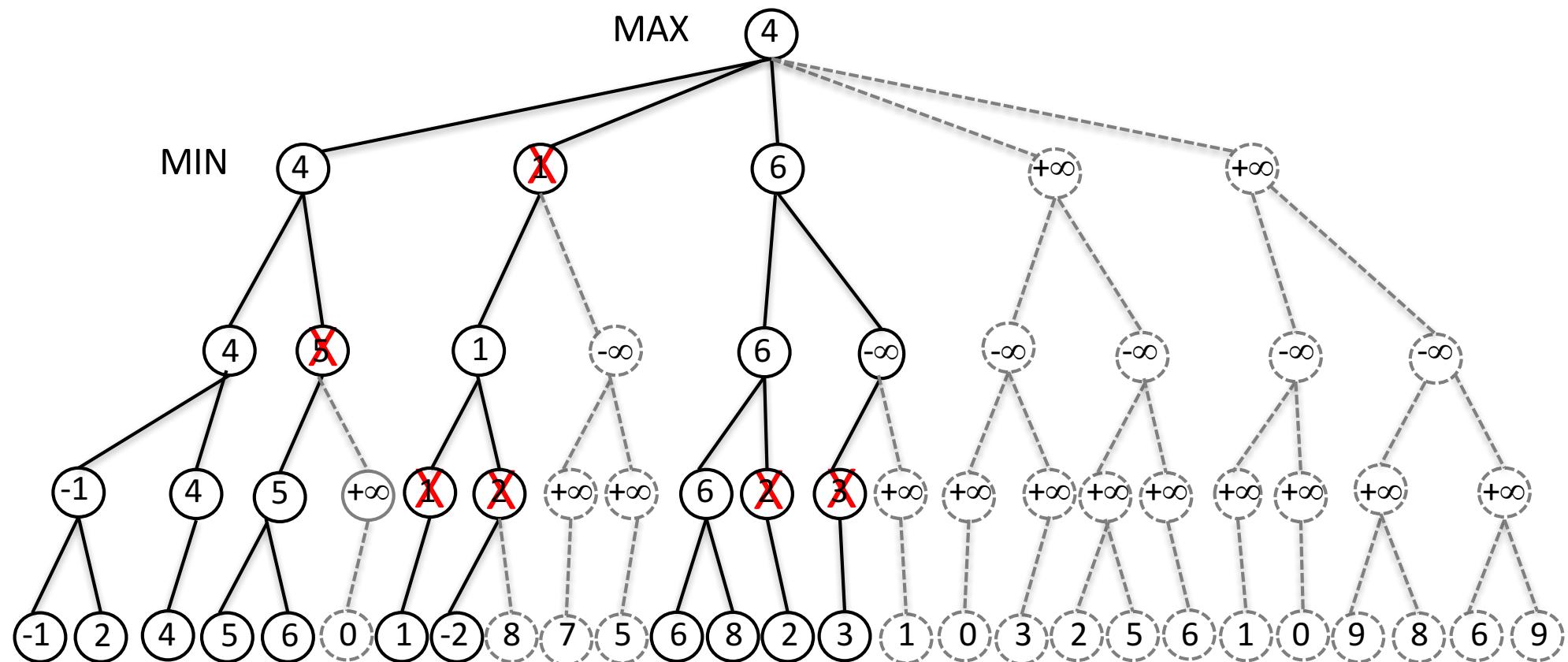
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



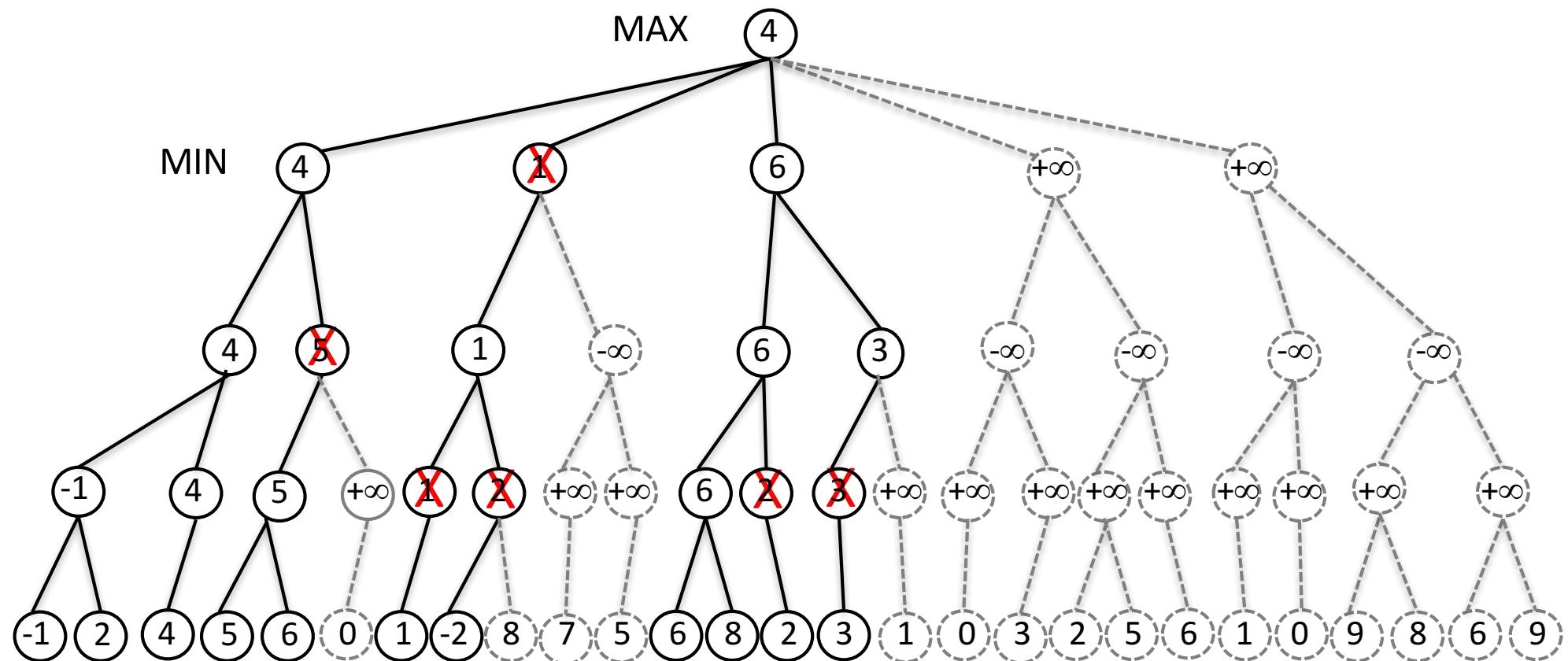
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



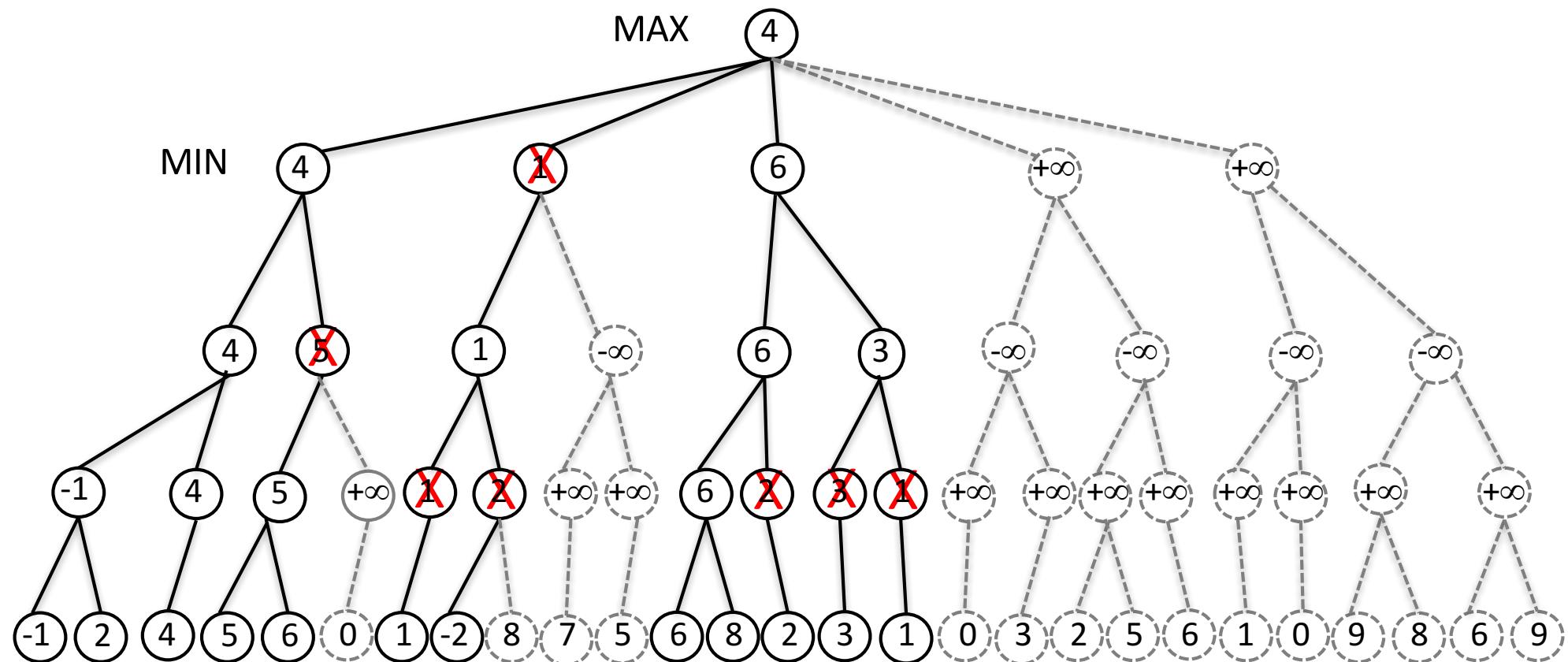
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



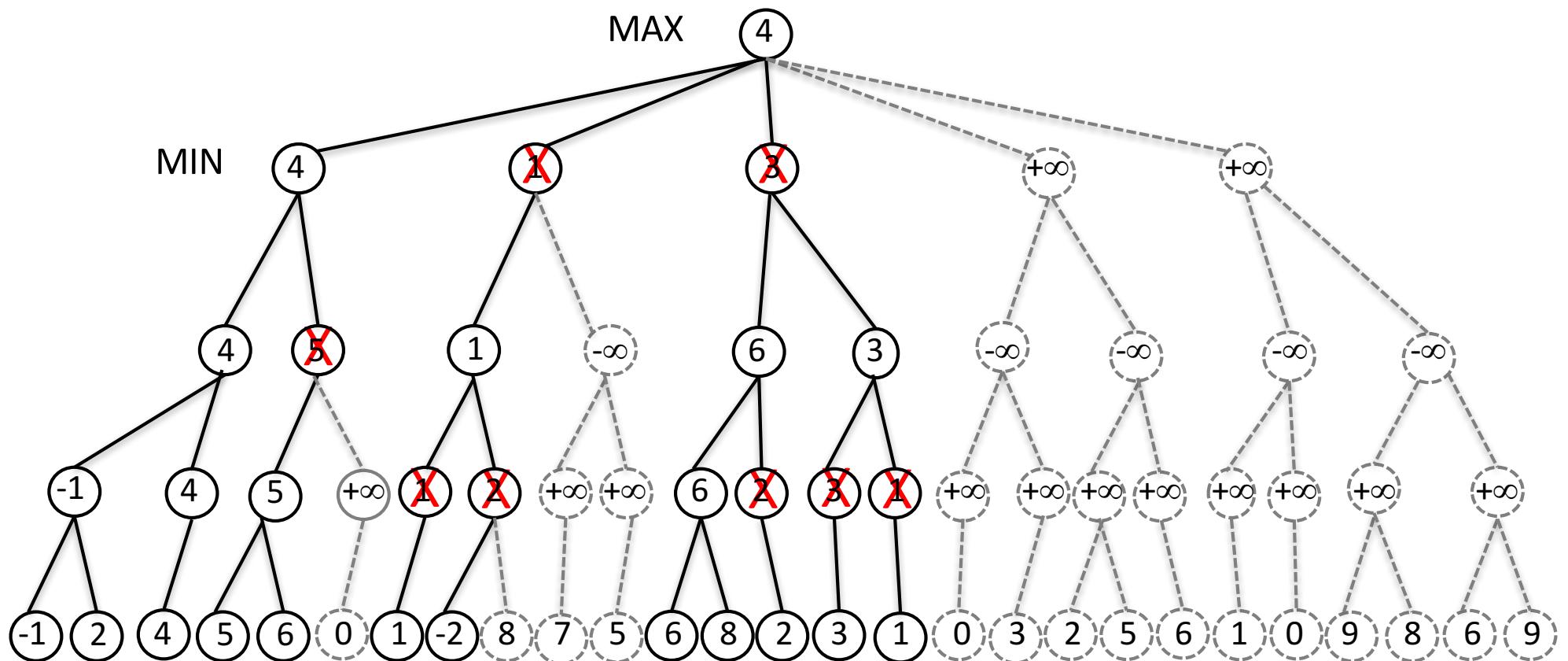
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



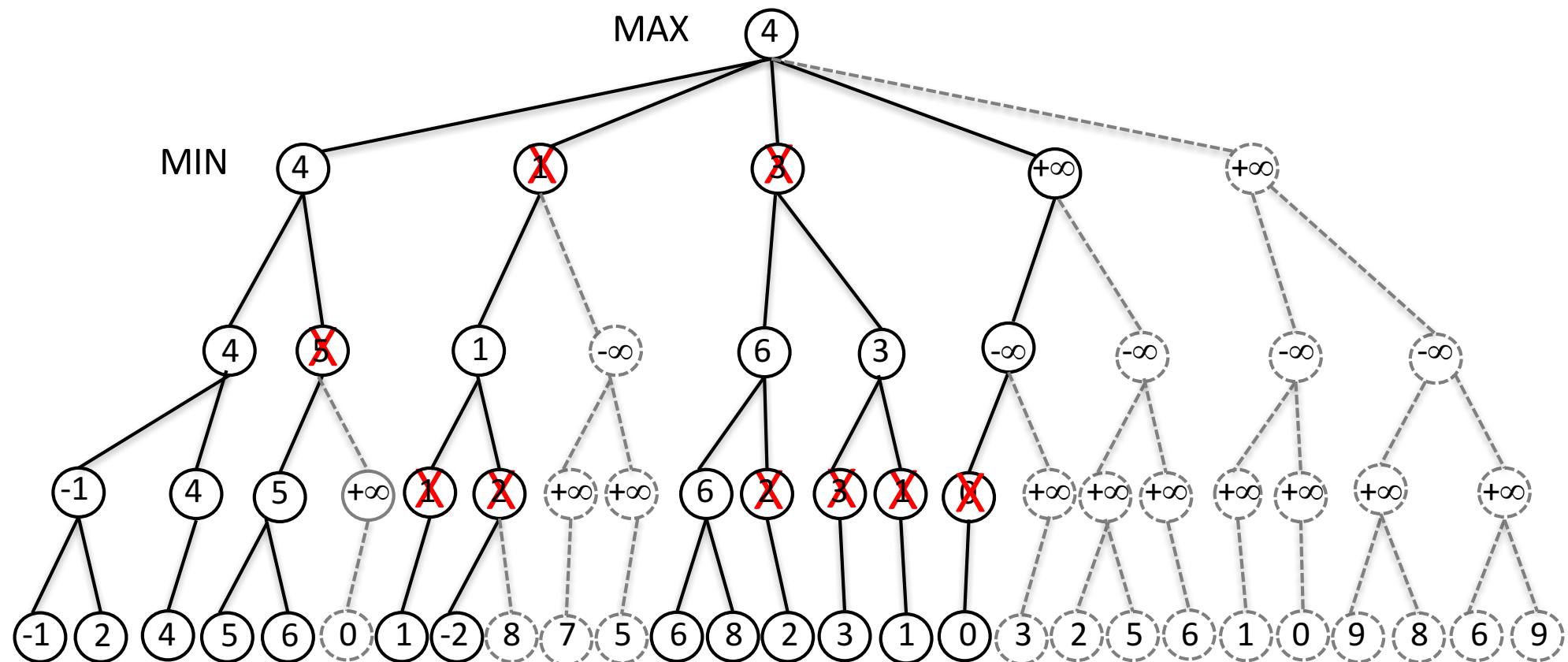
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



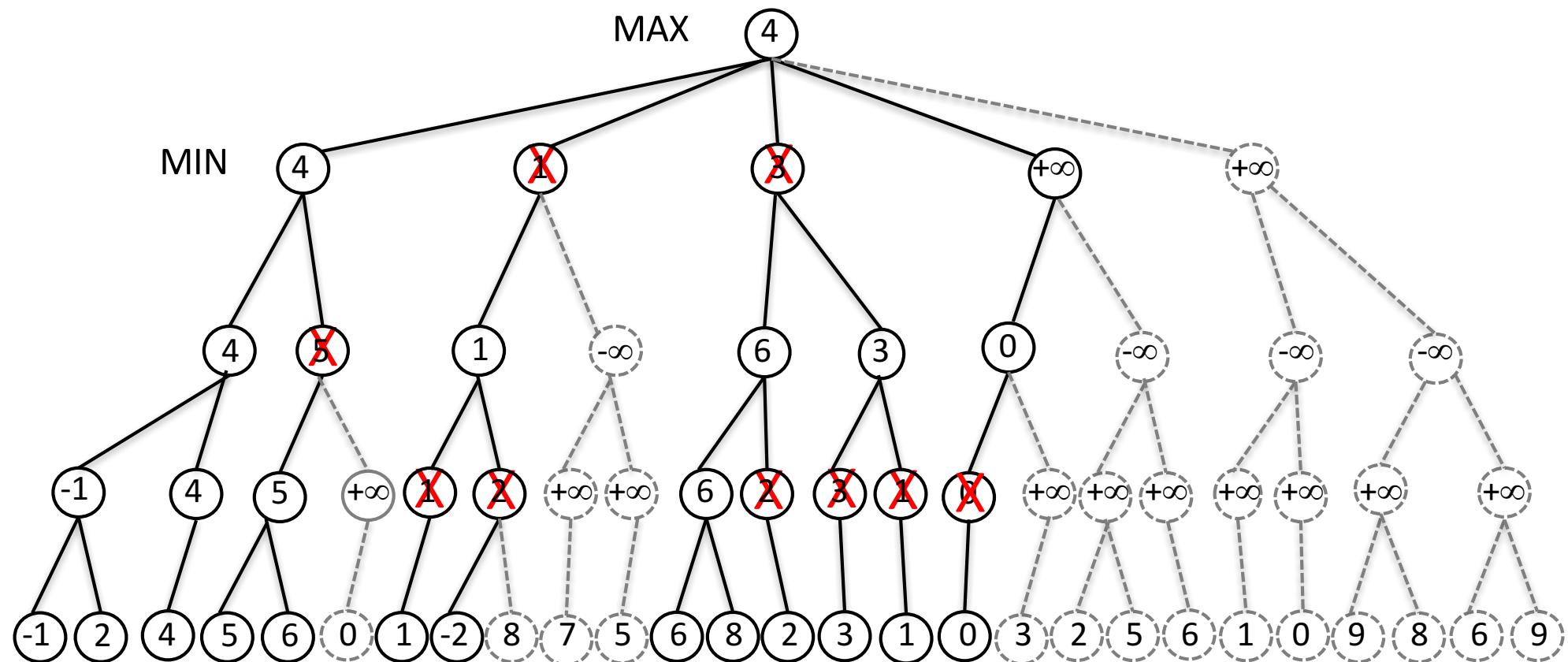
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



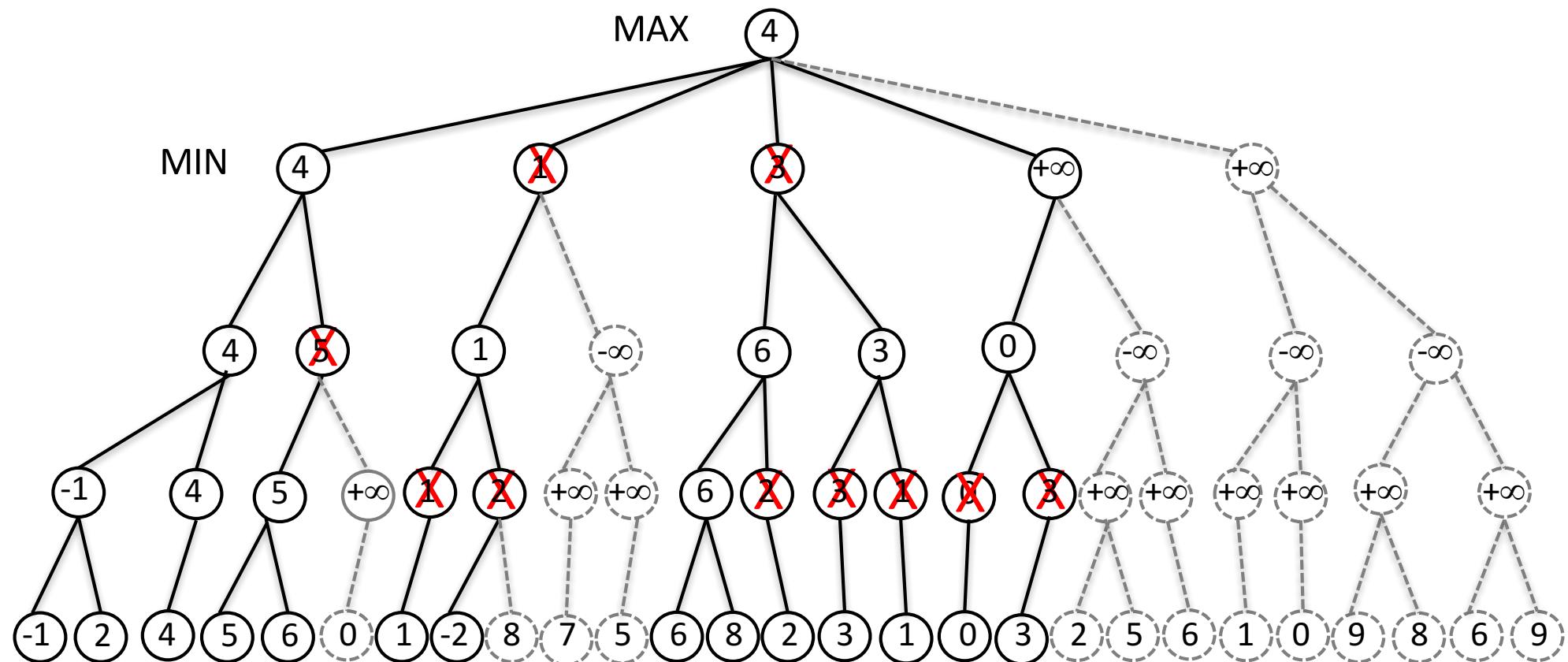
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



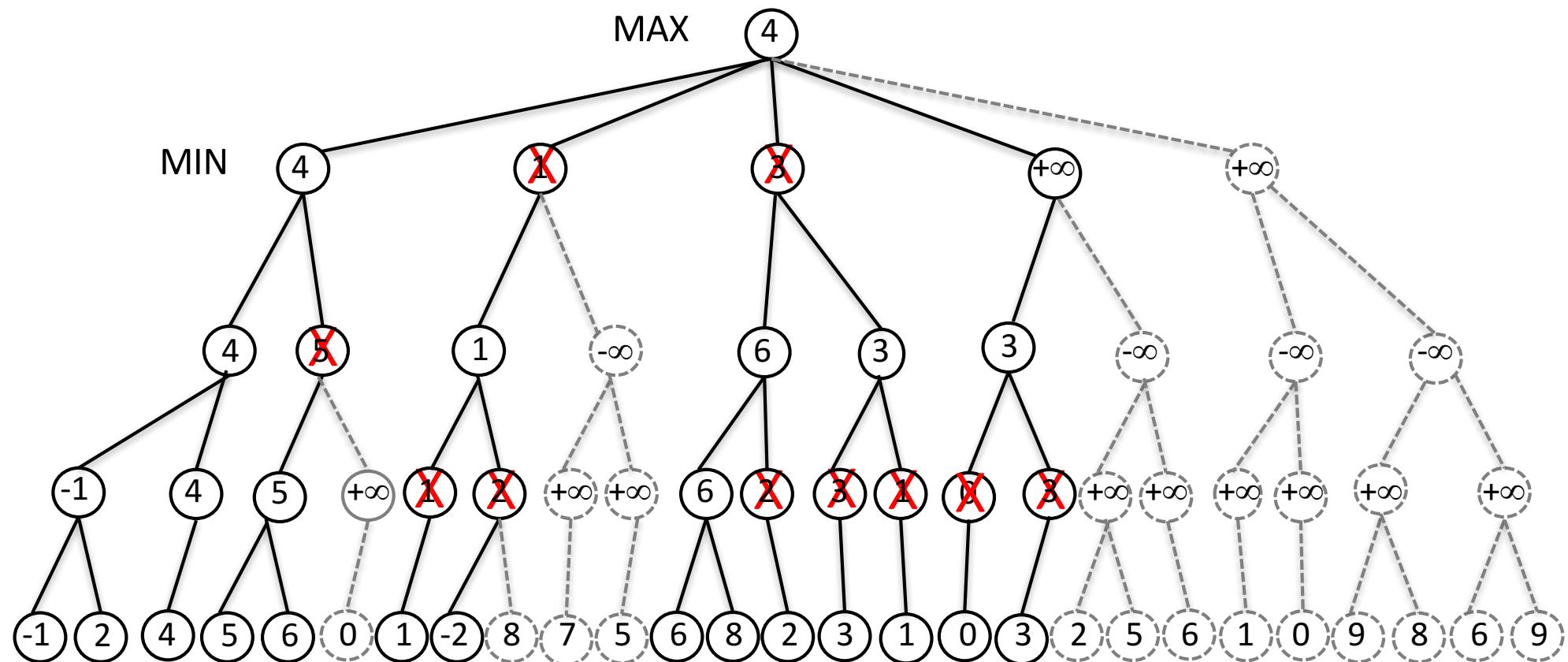
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



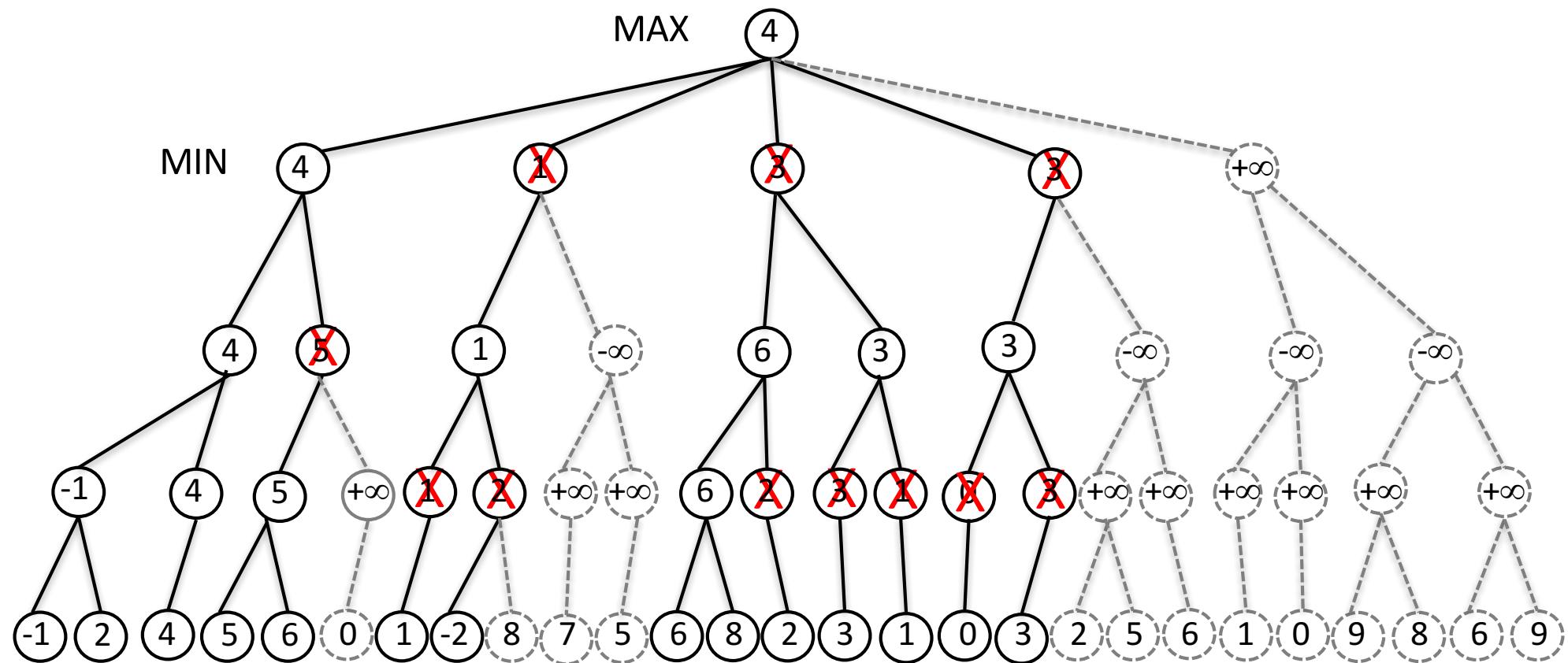
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



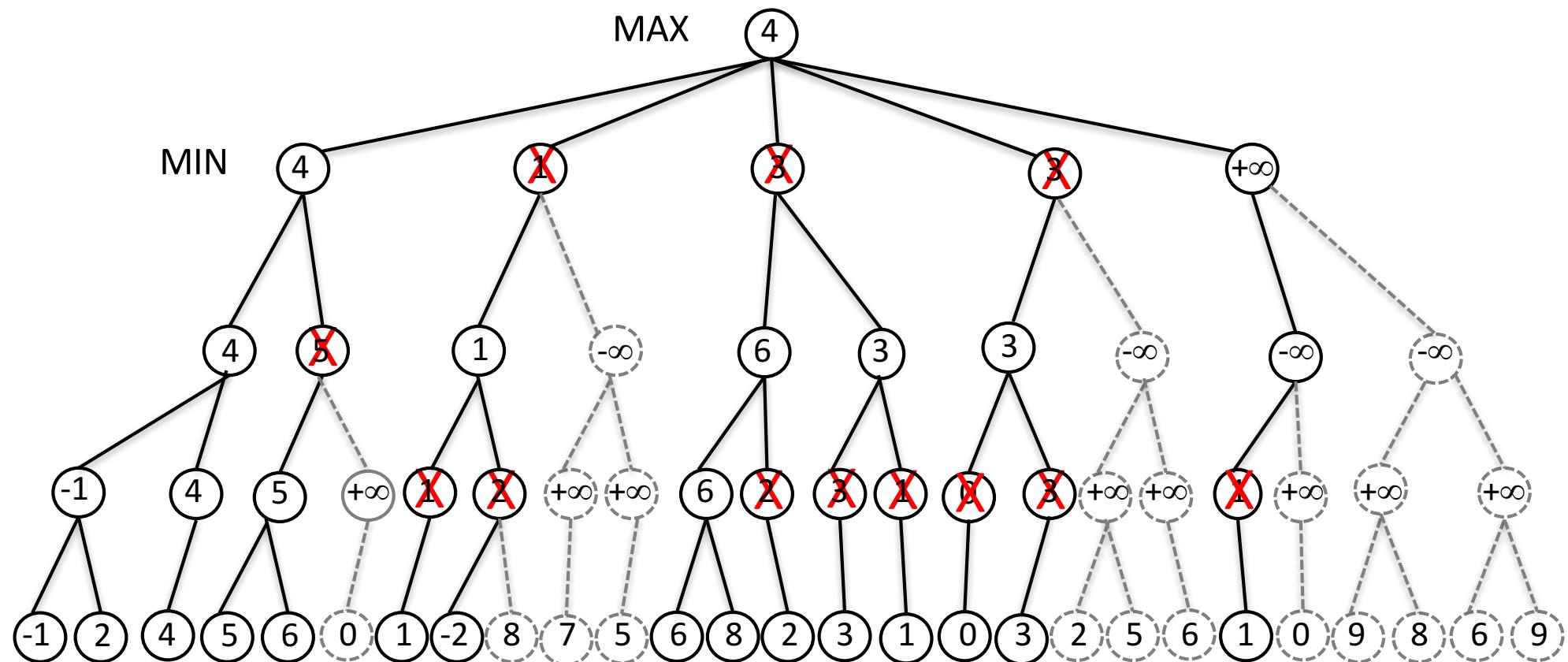
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



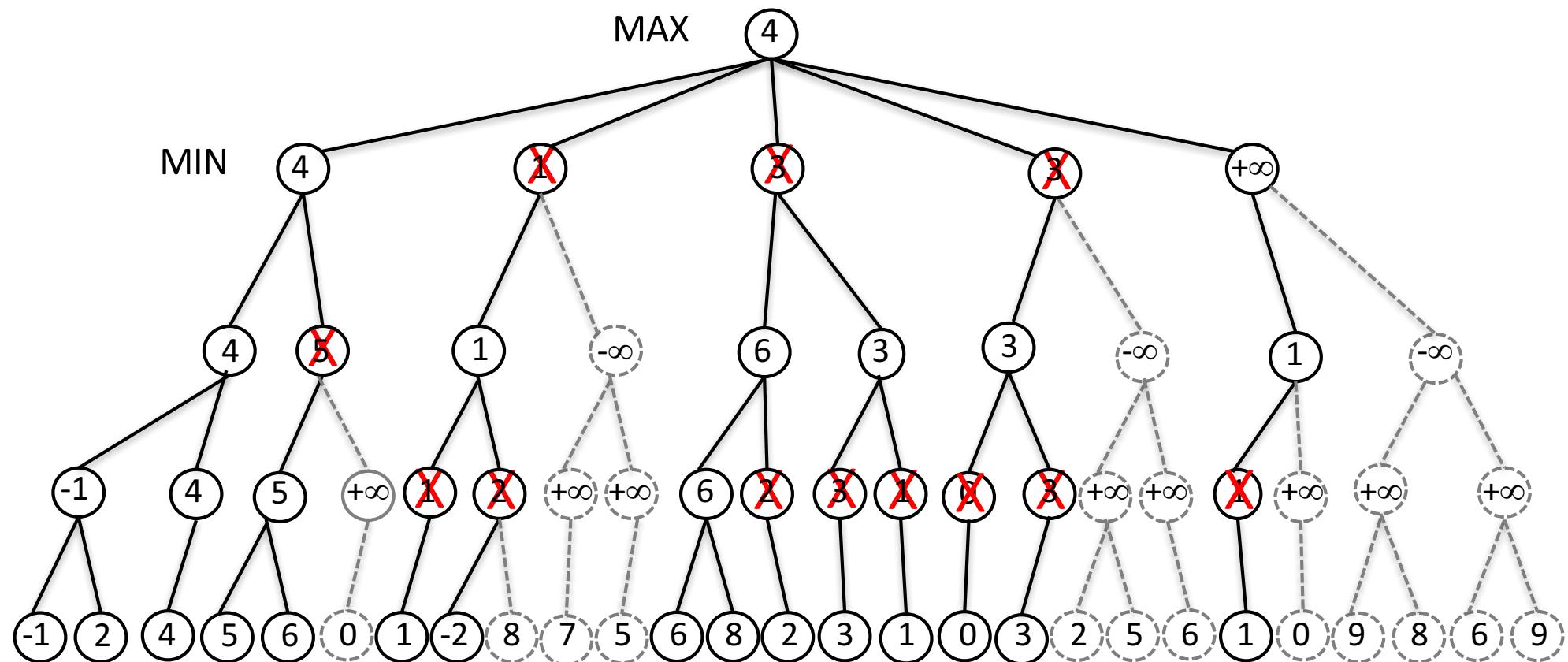
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



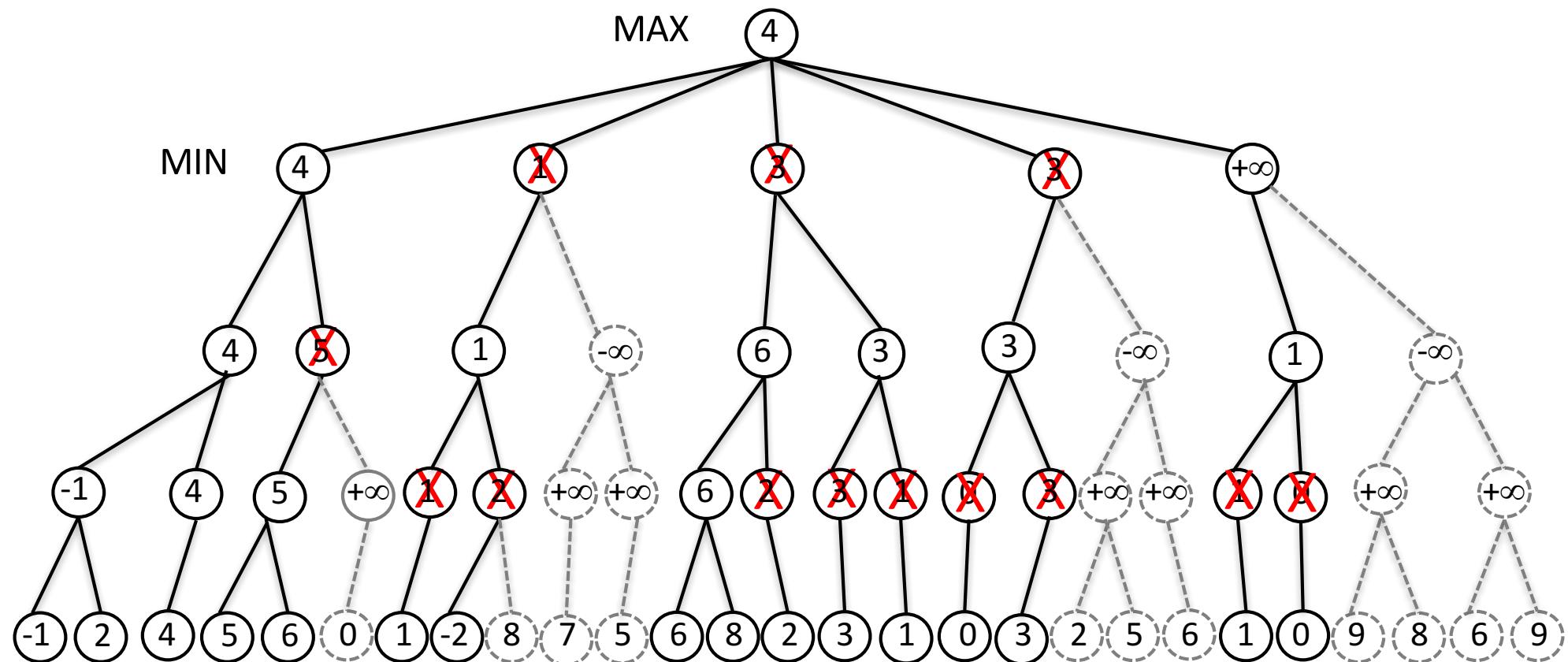
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



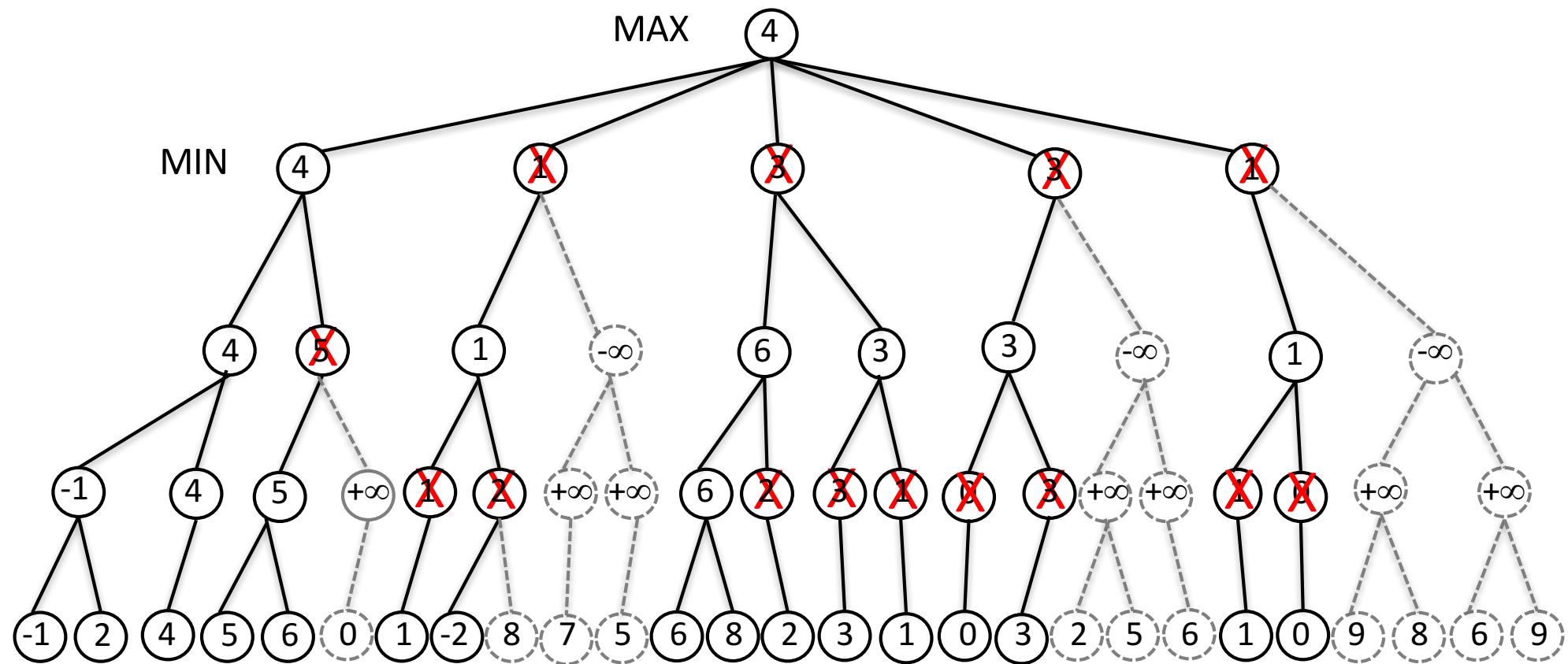
## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



## 5. Algoritmo $\alpha$ - $\beta$ : ejemplo 2



## 6. Algoritmo $\alpha$ - $\beta$ : eficiencia, refinamientos

La eficiencia del procedimiento  $\alpha$ - $\beta$  depende del nº de cortes que se realizan  $O(b^{3d/4})$  para un orden de expansión aleatorio.

El número de cortes que se pueden hacer depende del orden en que aparezcan los valores  $\alpha/\beta$  que permiten un corte, el algoritmo  $\alpha$ - $\beta$  será más eficiente si examinamos en primer lugar los sucesores que probablemente sean los mejores.

El valor final asumido por el nodo inicial es igual al valor de la evaluación de un nodo terminal (nodo óptimo en el nivel obtenido): si este nodo aparece pronto, en la búsqueda en profundidad, el nº de cortes será máximo, generando el mínimo árbol de búsqueda.

En un  $\alpha$ - $\beta$  óptimo se generarán  $O(b^{d/2})$  mientras que en Mini-Max se generan  $O(b^d)$ , luego el factor de ramificación eficaz es  $\sqrt{b}$  en lugar de  $b$ , por lo tanto el número de nodos terminales que se generan en un árbol con profundidad  $n$ , es igual al número de nodos terminales que se generarían en un procedimiento Minimax de profundidad  $n/2$ . Es decir, dada una limitación espacio/tiempo, el procedimiento  $\alpha$ - $\beta$  permite una profundidad doble que el procedimiento Minimax.

## 6. Algoritmo $\alpha$ - $\beta$ : eficiencia, refinamientos (2)

Diferentes variantes del  $\alpha$ - $\beta$  para encontrar el mejor nodo terminal lo antes posible y maximizar el número de cortes:

**A) Expansión ordenada a cada nivel**

**B) Orden preliminar**

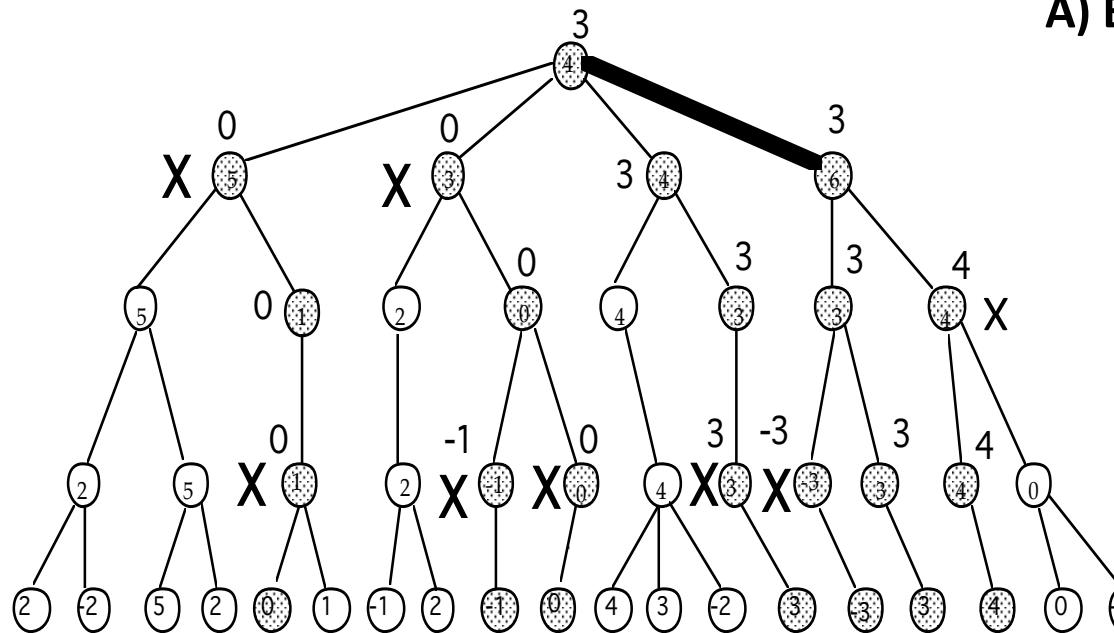
En Juegos es necesario:

- Utilizar funciones heurísticas para podar el árbol
- Utilizar procedimientos más inteligentes para conseguir un comportamiento dinámico

**Ejemplos:**

- Tener una serie de movimientos clásicos (biblioteca de movimientos, salidas, preferencia de piezas, zonas de tablero, ...)
- Tener patrones de tablero y mejores movimientos para esas posiciones
- Tácticas y estrategias de juego, hábitos del oponente, fallos (movimientos erróneos), etc.
- Aprendizaje de la función de evaluación.

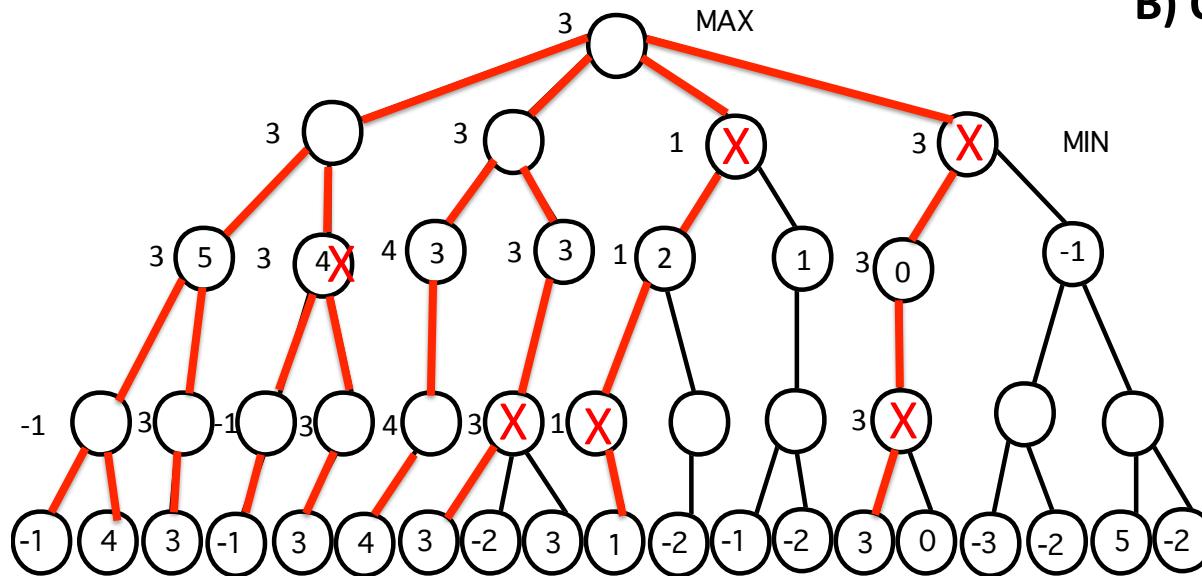
## 6. Algoritmo $\alpha$ - $\beta$ : eficiencia, refinamientos (3)



#### A) Expansión ordenada a cada nivel:

- Se expanden todos los sucesores del nodo expandido (a partir del nodo inicial).
  - Se evalúan los nodos sucesores expandidos.
  - Se expande aquel nodo que:
    - tenga un valor máximo, si es sucesor de un nodo max
    - tenga un valor mínimo, si es sucesor de un nodo min
  - Este método es una combinación anchura<sup>1</sup>/ profundidad, y supone no elegir los sucesores a expandir de cada nodo de forma arbitraria.

## 6. Algoritmo $\alpha$ - $\beta$ : eficiencia, refinamientos (4)



## B) Orden preliminar:

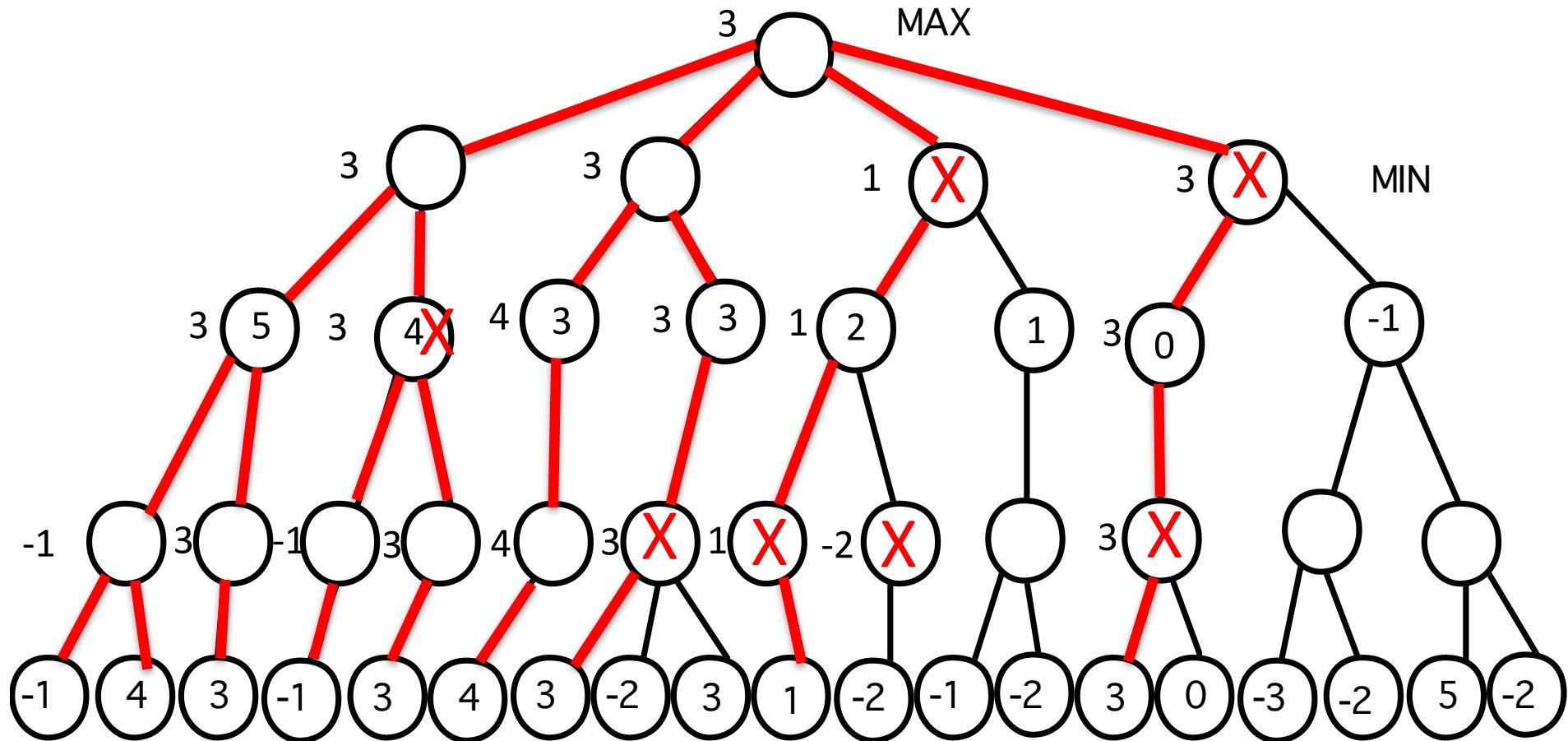
- Expandir el árbol en anchura hasta un nivel  $m$ .
  - Evaluamos los nodos terminales del nivel  $m$ .
  - Ordenar los nodos del nivel  $m$  en función de su mejor posición para el nodo inicial.
  - Empezamos un procedimiento  $\alpha-\beta$  a partir de ese nivel hasta el deseado, pero expandiendo los nodos en orden a su mejor promesa.

# Bloque 1 – Representación del conocimiento y búsqueda

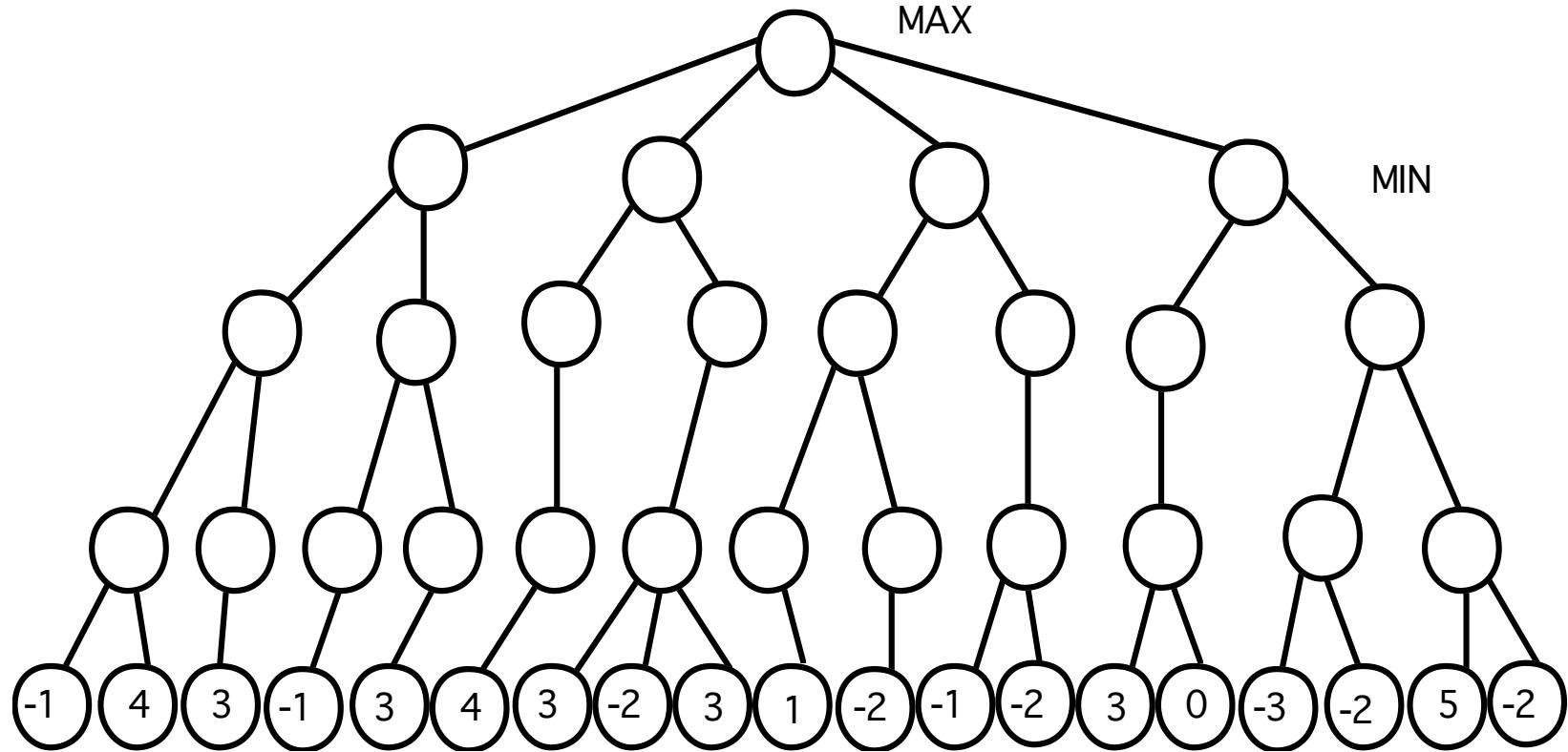


**Tema 7:  
Búsqueda con adversario**

### 3. Poda $\alpha$ - $\beta$ : ejemplo 2



## Poda $\alpha$ - $\beta$ : ejemplo



## Poda $\alpha$ - $\beta$ : ejemplo

