Búsqueda con adversario

- **Uso:** Decidir mejor jugada en cada momento para cierto tipo de juegos
- Hay diferentes tipos de juegos según sus características:
  - Numero de jugadores, toda la información conocida por todos los jugadores, azar, indeterminismo, cooperación/competición, recursos limitados, ...
- Nos focalizaremos en juegos con:
  - 2 jugadores.
  - Movimientos alternos (jugador MAX, jugador MIN)
  - Información perfecta
  - Por ejemplo: ajedrez, damas, otello, go, ...

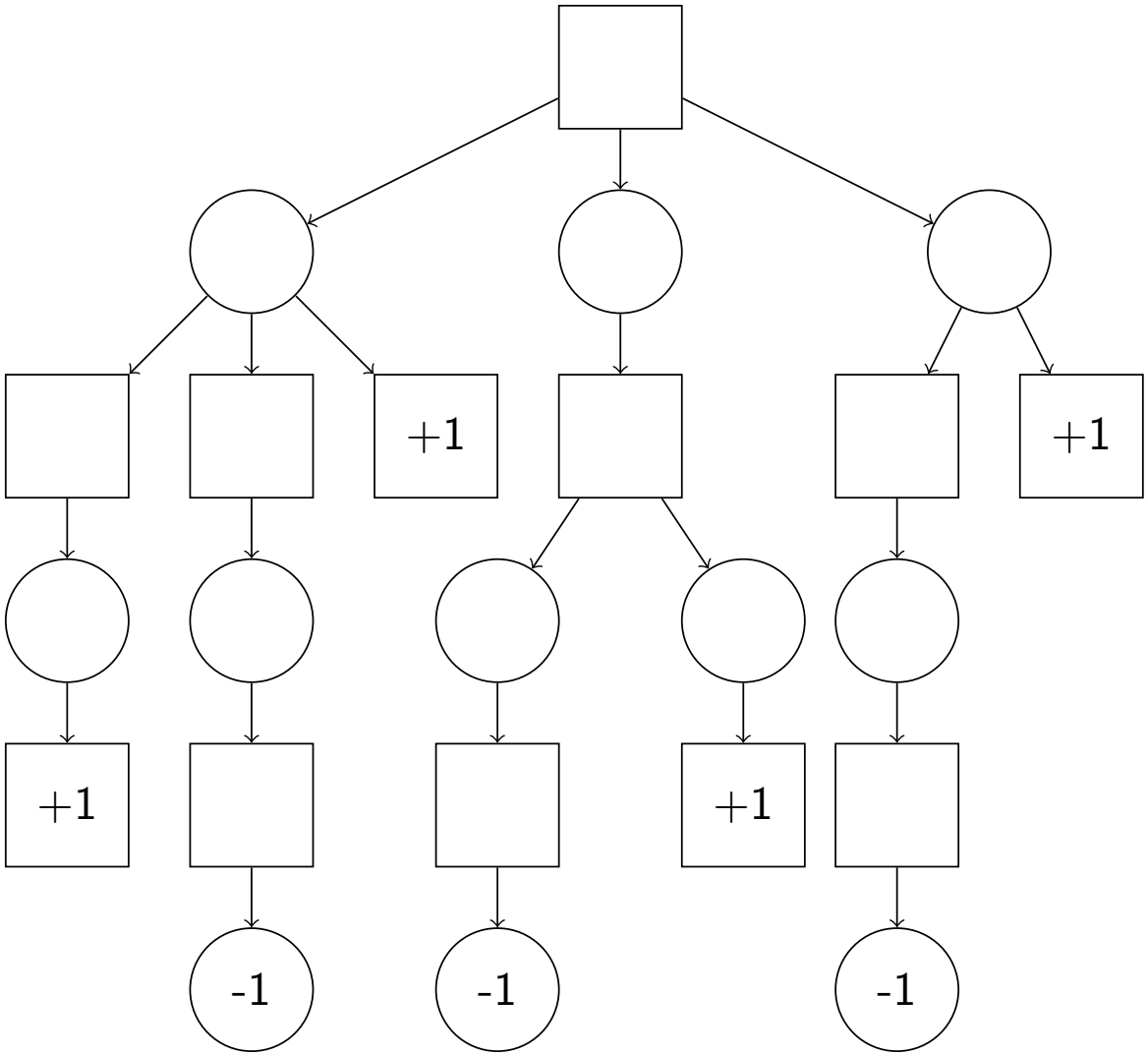
Representación del juego

- Puede ser definido como un problema de espacio de estados
  - Estado = Elementos del juego
  - Estados finales= Estados ganadores (Definidos por sus propiedades)
  - Acciones/operadores = Reglas del juego
- Son problemas con características especiales
  - La accesibilidad de los estados depende de las acciones elegidas por el contrario
  - Dos tipos de soluciones diferentes (una para cada jugador)
  - No hay nocion de optimalidad (todas las soluciones son iguales, no importa la longitud del camino)

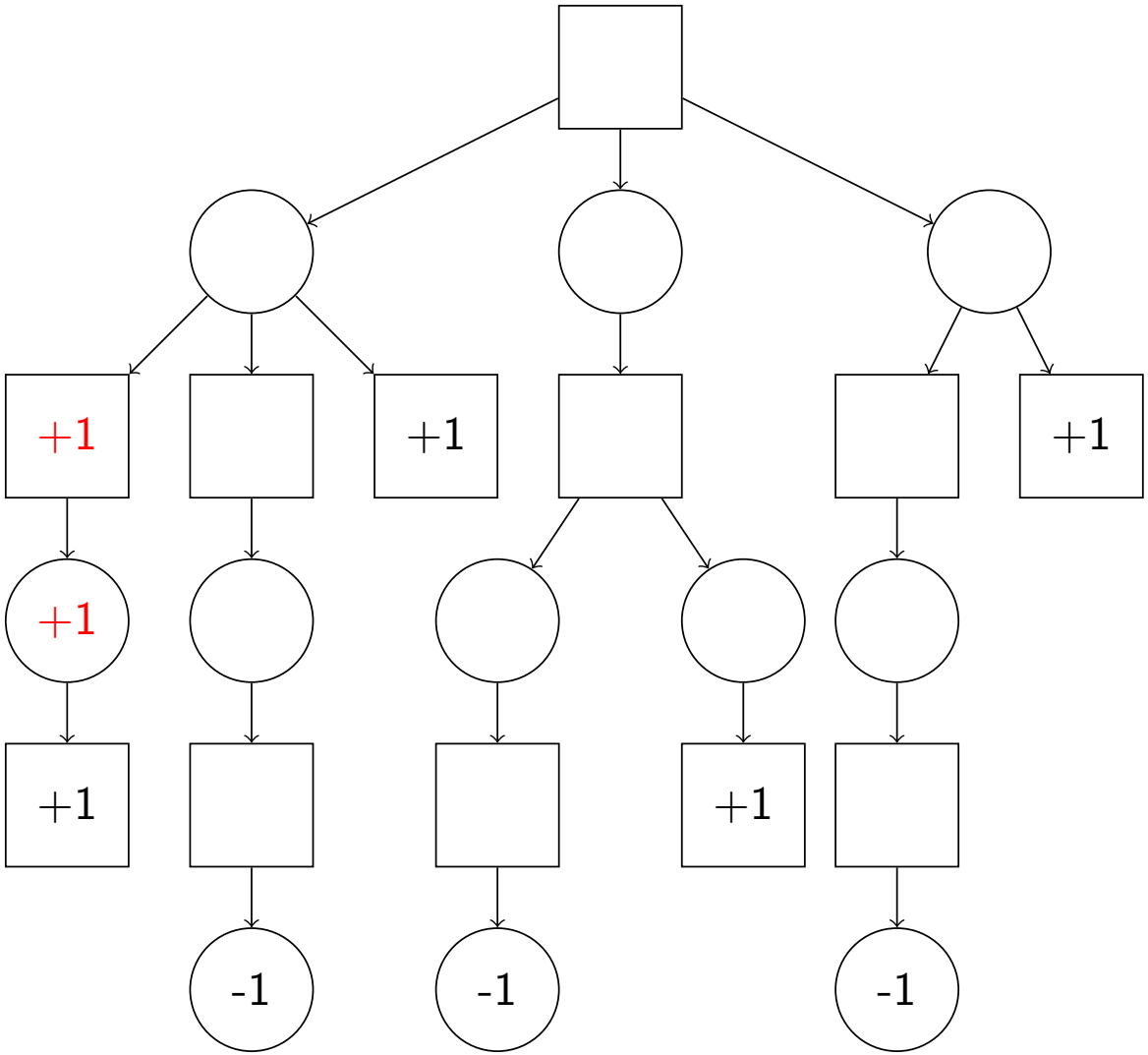
Búsqueda con adversario

- La aproximación trivial es generar todo el árbol de jugadas
- Etiquetamos las jugadas terminales dependiendo de si gana MAX o MIN (+1 o -1)
- El objetivo es encontrar un conjunto de movimientos accesible que de como ganador a MAX
- Se propagan los valores de las jugadas terminales de las hojas hasta la raíz, elegimos una rama de una hoja ganadora accesible
- Una búsqueda en profundidad minimiza el espacio
- En juegos mínimamente complejos esta búsqueda es impracticable (p.e.: ajedrez $O(2^{35})$, go $O(2^{300})$)

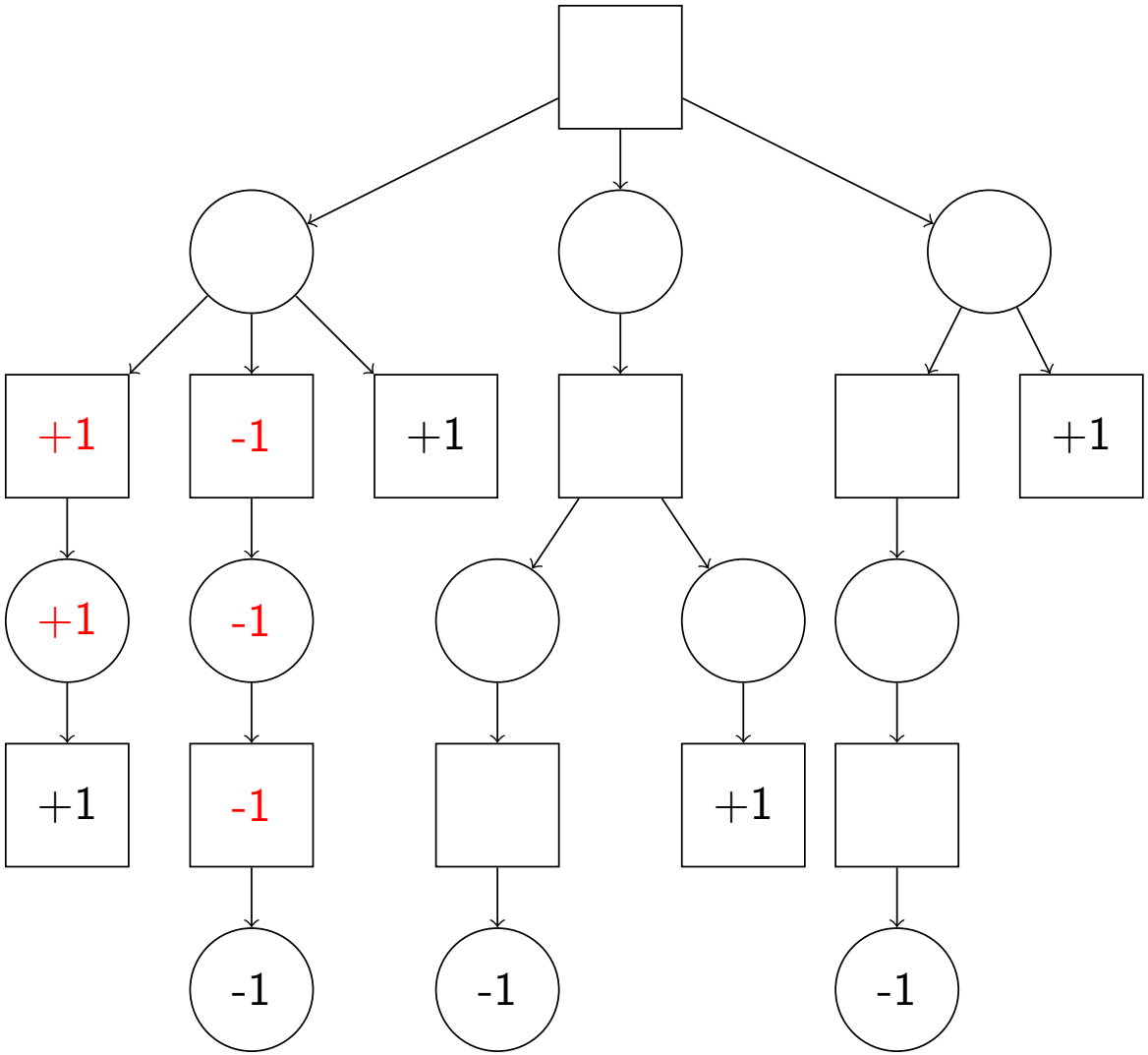
Búsqueda con adversario



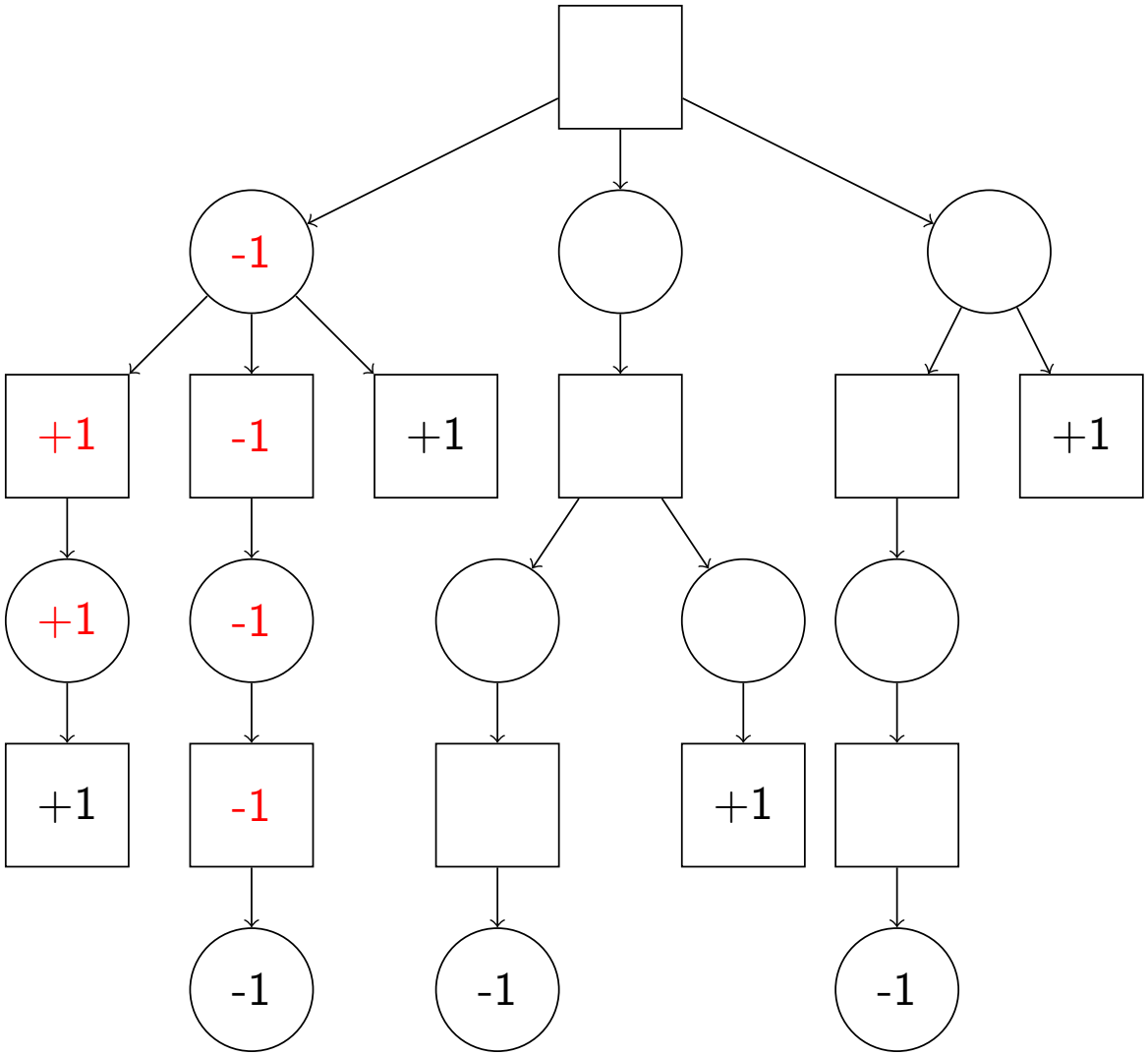
Búsqueda con adversario



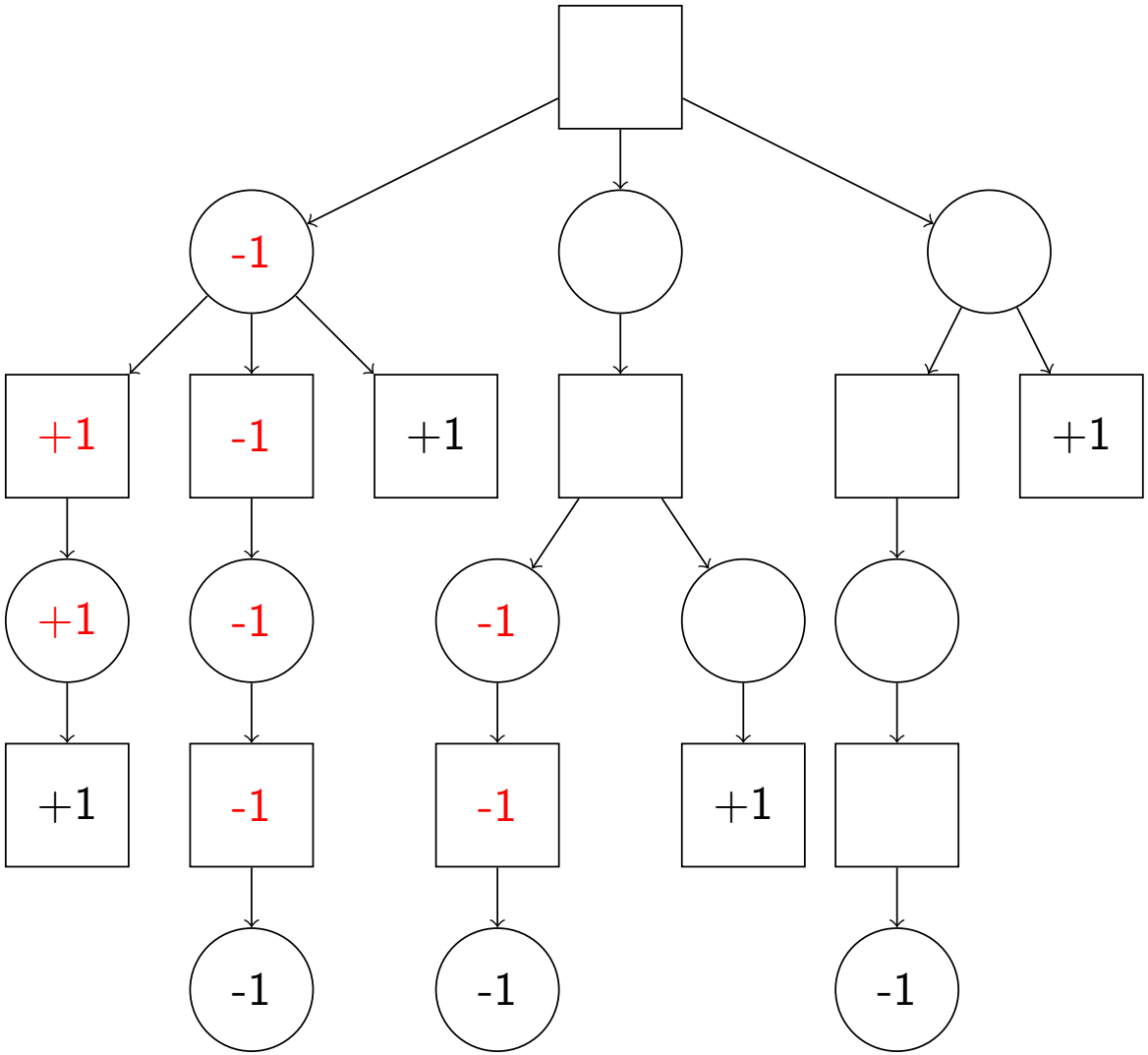
Búsqueda con adversario



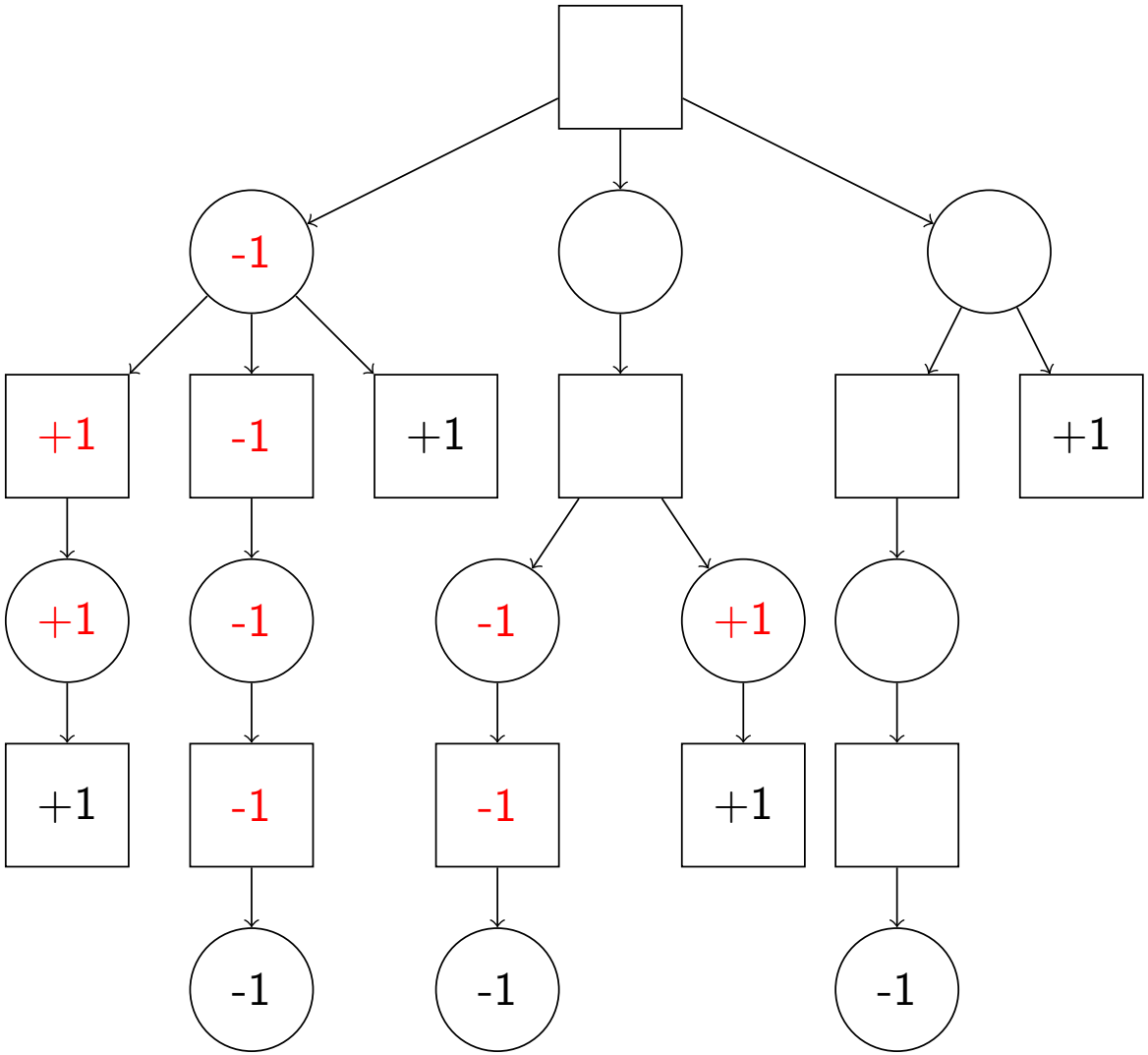
Búsqueda con adversario



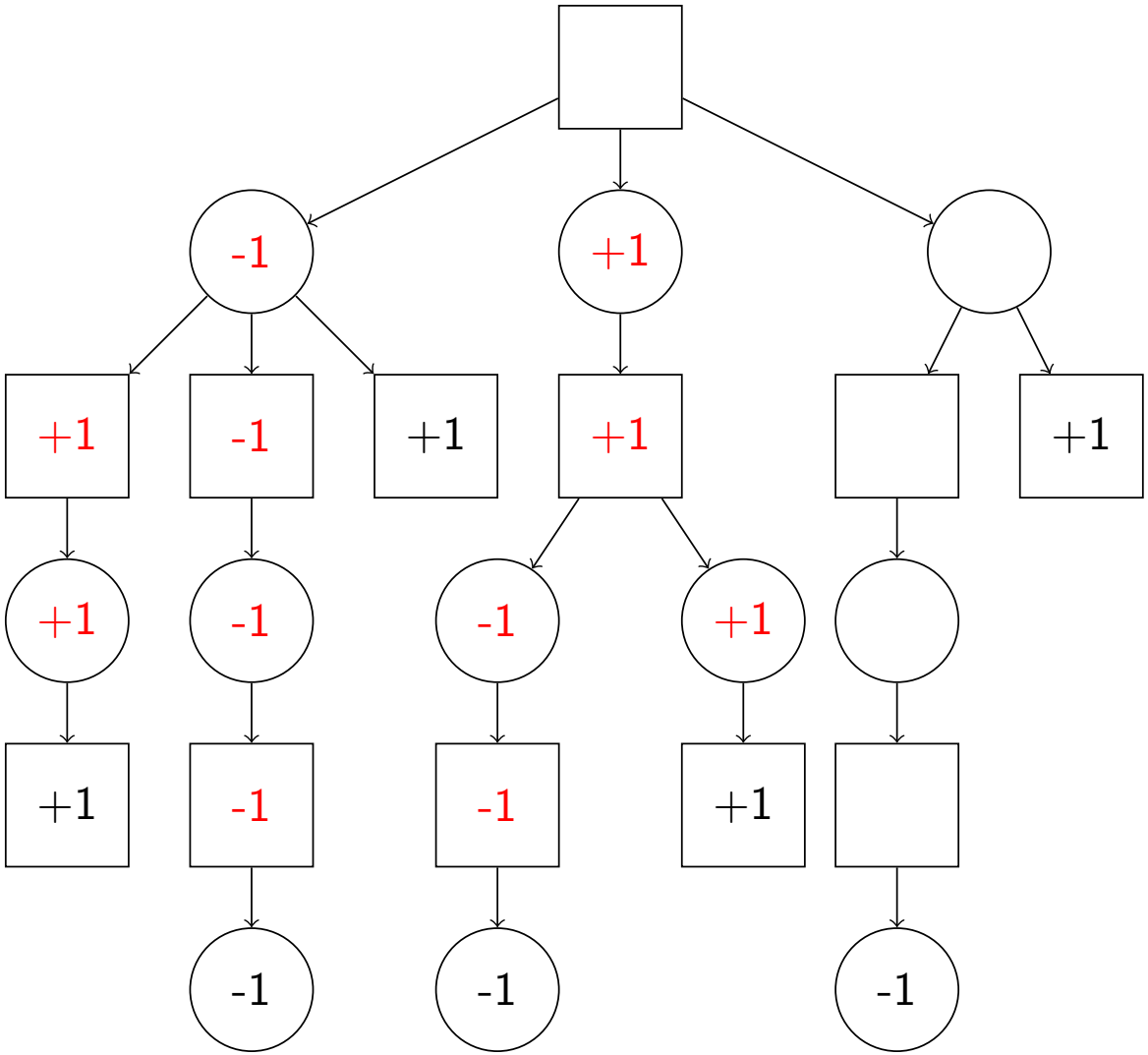
Búsqueda con adversario



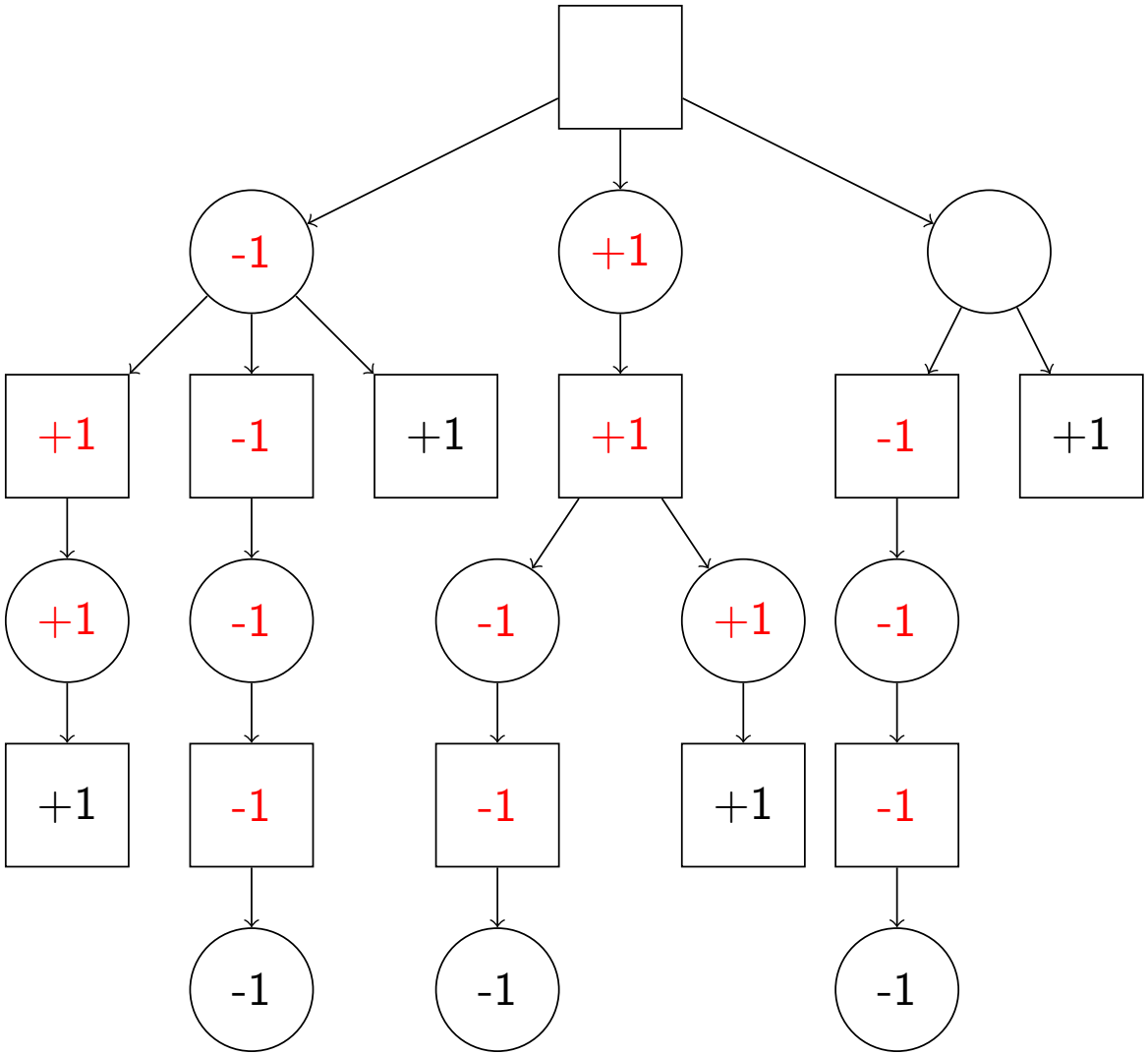
Búsqueda con adversario



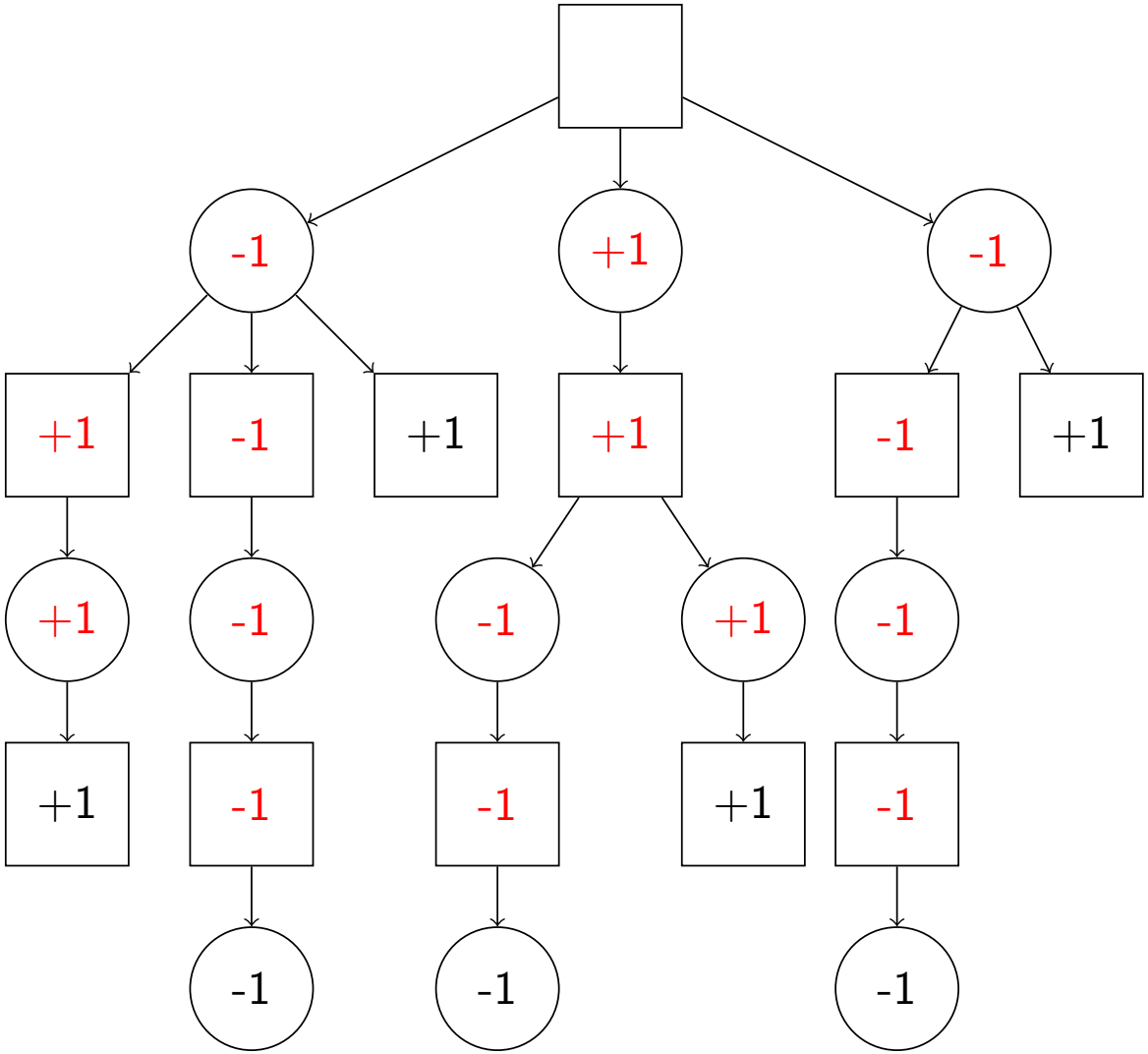
Búsqueda con adversario



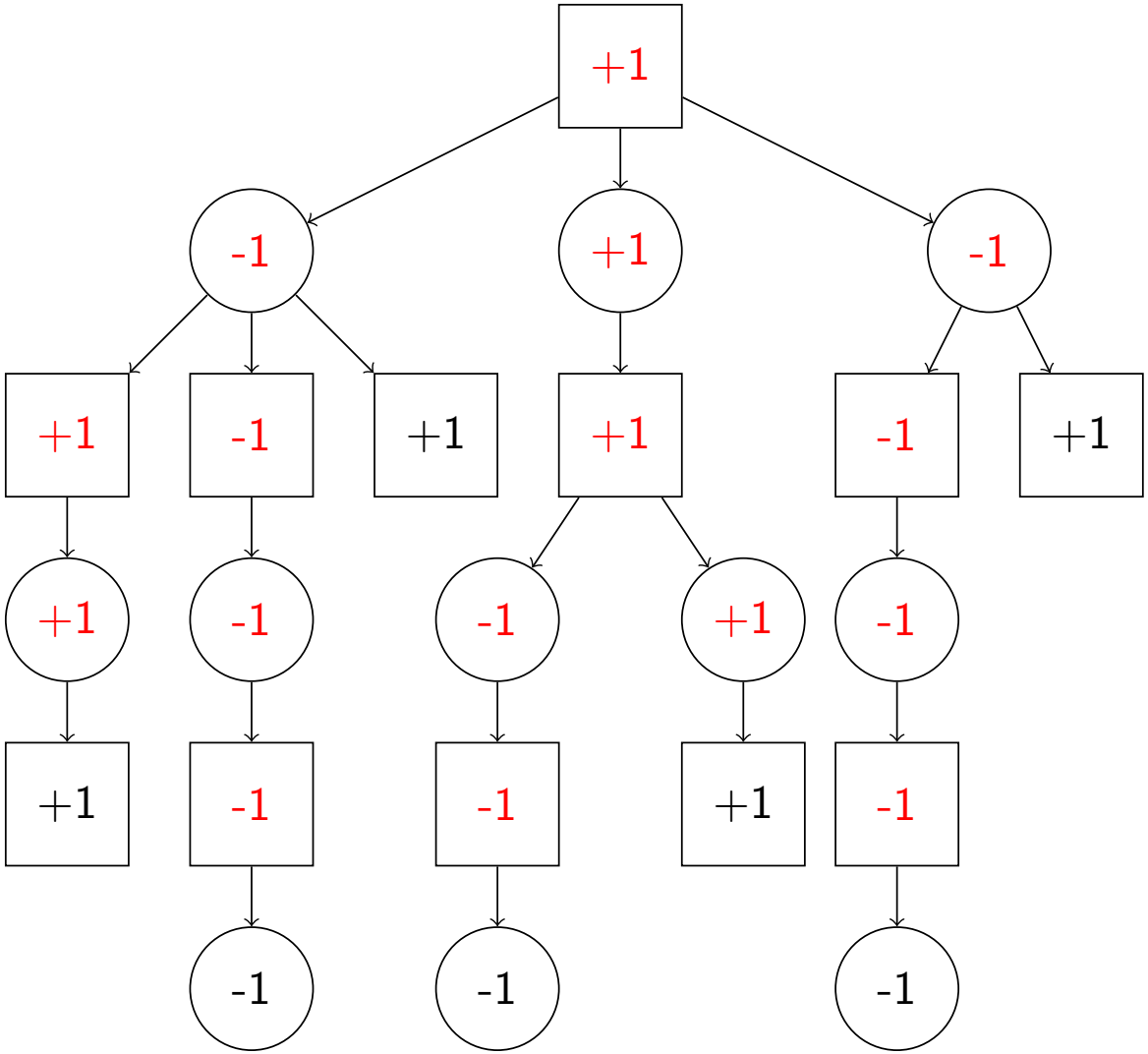
Búsqueda con adversario



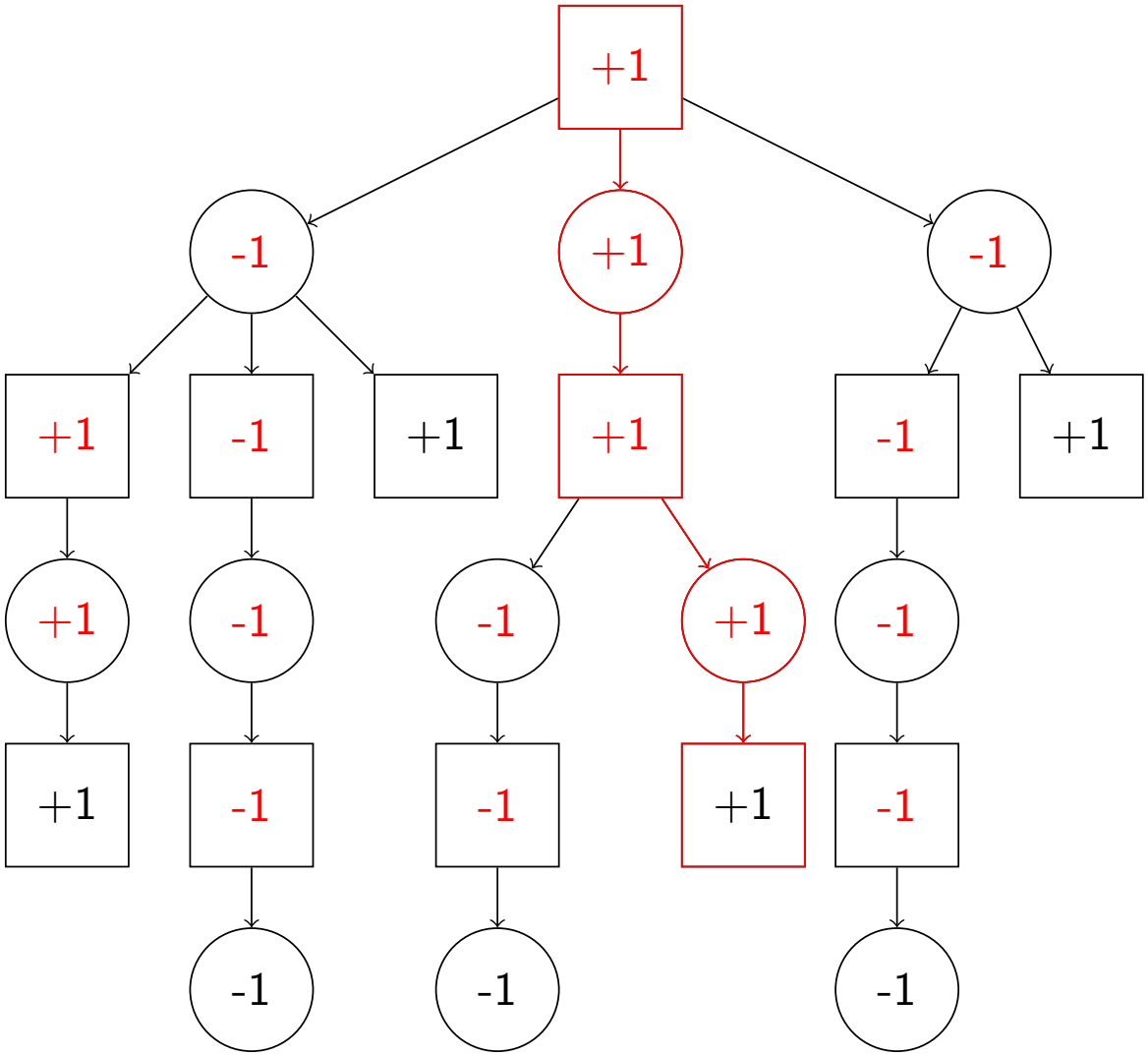
Búsqueda con adversario



Búsqueda con adversario



Búsqueda con adversario



Búsqueda con adversario

- Aproximación heurística: Definir una función que nos indique lo cerca que estamos de una jugada ganadora (o perdedora)
- En esta función intervendrá información del dominio
- Esta función no representa ningún coste ni es una distancia en pasos
- Por convención las jugadas ganadoras se evalúan a $+\infty$ y las perdedoras a $-\infty$
- El algoritmo busca con profundidad limitada y sólo decide la siguiente jugada a partir del nodo raíz
- Cada nueva decisión implicará repetir la búsqueda
- A mayor profundidad en la búsqueda mejor jugaremos

Algoritmo Minimax

Función: MiniMax (g)

```
movr:movimiento; max,maxc:entero
max ← −∞
para cada mov ∈ movs_posibles(g)
hacer
    cmax ← valor_min(aplicar(mov,g))
    si cmax > max entonces
        max ← cmax
        movr ← mov
    fin
fin
retorna movr
```

g: Representación del estado (posición de las piezas, profundidad máxima a explorar, turno actual, ...)
movs_posibles(g): genera la lista de todos los movimientos posibles en el estado actual
aplicar(mov,g): Genera el estado que se obtiene al aplicar el movimiento al estado actual

- Se inicia un recorrido en profundidad del árbol del juego hasta una profundidad máxima
- Dependiendo del nivel se llama a una función que obtiene el valor máximo o mínimo de la evaluación de los descendientes (valorMax, valorMin)
- El recorrido se inicia con la jugada del jugador MAX
- Asumimos que la función de evaluación es la misma para los dos jugadores

Algoritmo Minimax

Función: valorMax (g)

```
vmax:entero
si estado_terminal(g) entonces
    retorna valor(g)
sino
    vmax ← −∞
    para cada mov ∈ movs_posibles(g)
    hacer
        vmax ← max(vmax,
        valorMin(aplicar(mov,g))
    fin
    retorna vmax
fin
```

Función: valorMin (g)

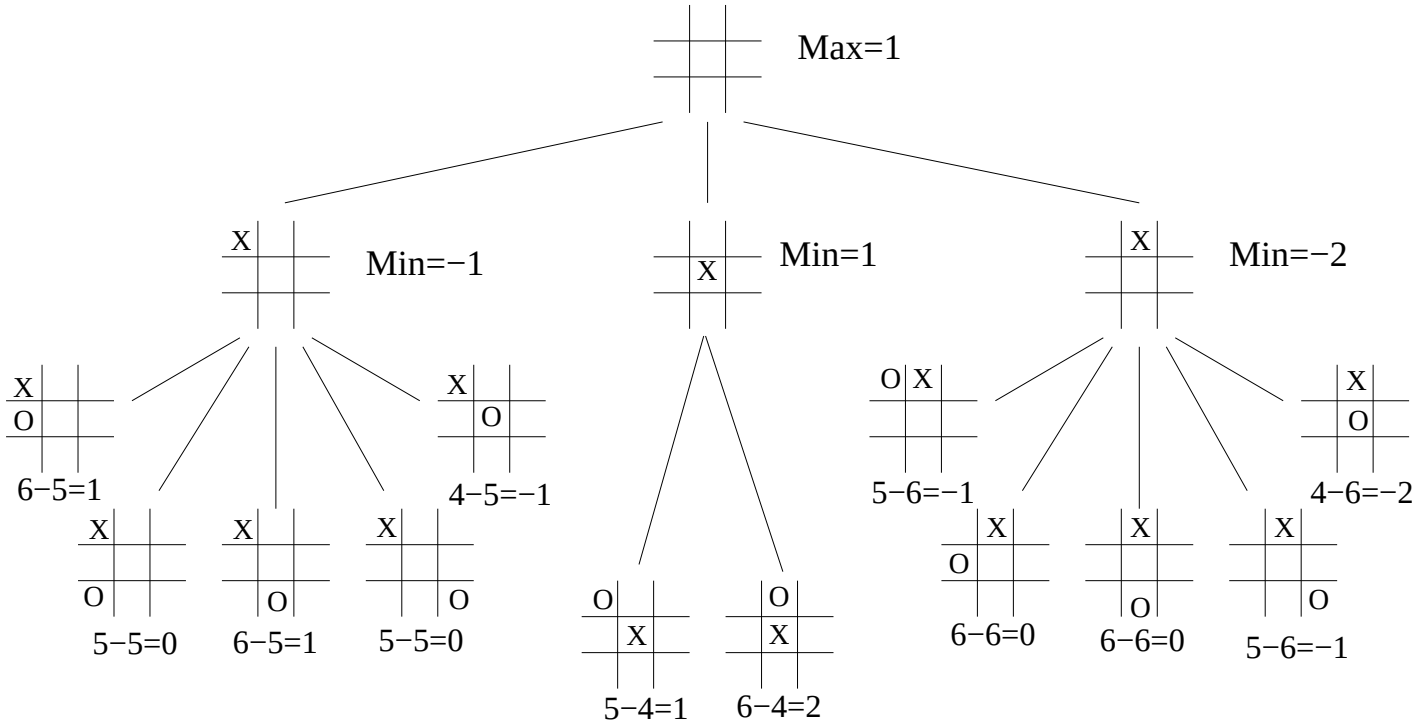
```
vmin:entero
si estado_terminal(g) entonces
    retorna value(g)
sino
    vmin ← +∞
    para cada  mov ∈ movs_posibles
    hacer
        vmin ← min(vmin,
        valorMax(aplicar(mov,g))
    fin
    retorna vmin
fin
```

estado_terminal(g): Determina si el estado actual es terminal (profundidad máxima, jugada ganadora)
evaluacion(g): Retorna el valor de la función de evaluación para el estado actual

Ejemplo: 3 en raya (1)

e = número de filas, columnas y diagonales completas disponibles para MAX - número de filas, columnas y diagonales completas disponibles para MIN
MAX juega con **X** y desea maximizar e, MIN juega con **O** y desea minimizar e

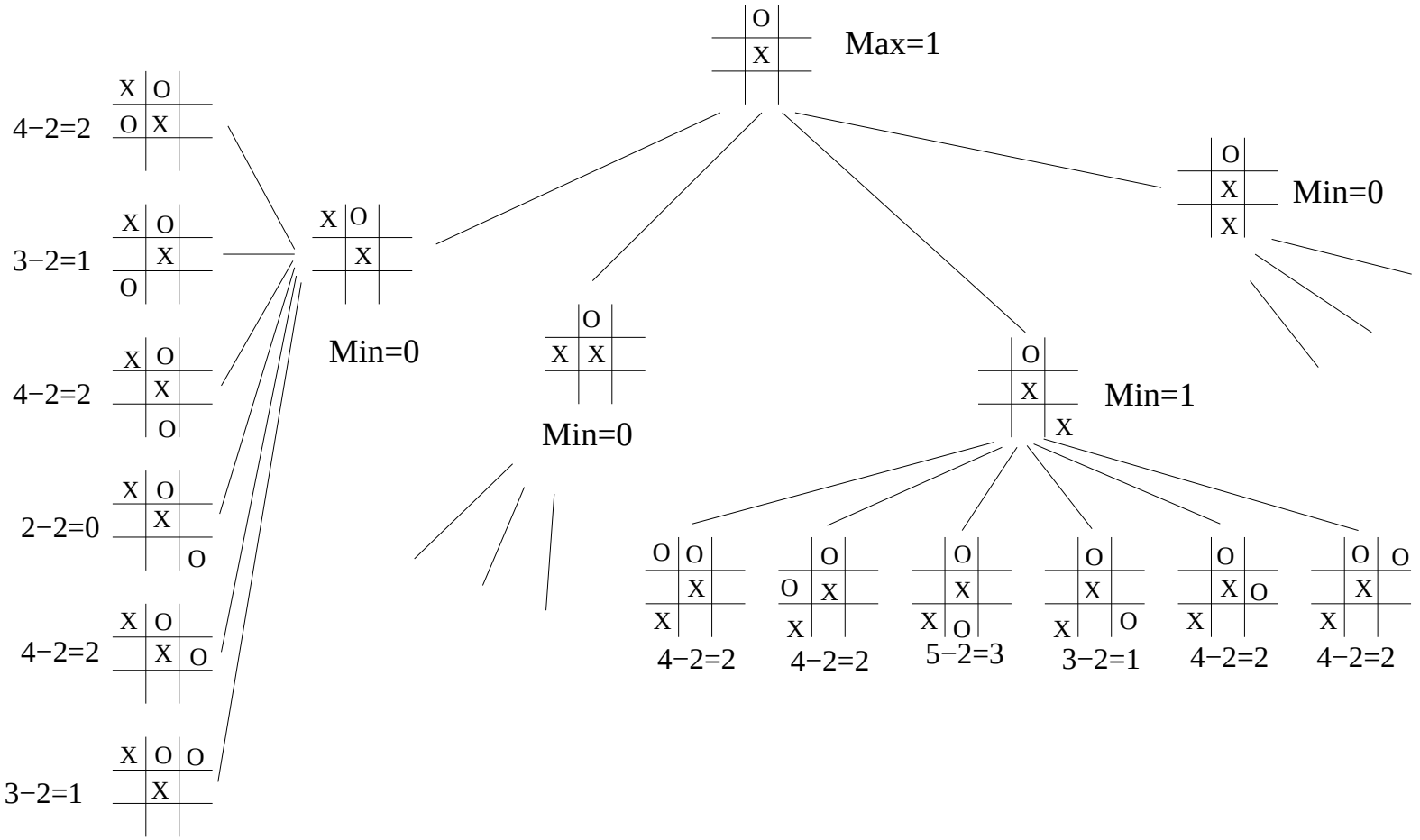
Los valores altos significan una buena posición para el que tiene que mover, Podemos controlar las simetrías, establecemos una profundidad de parada (en el ejemplo 2)



La mejor jugada es en la que MAX coloca su ficha en el centro

Ejemplo: 3 en raya (2)

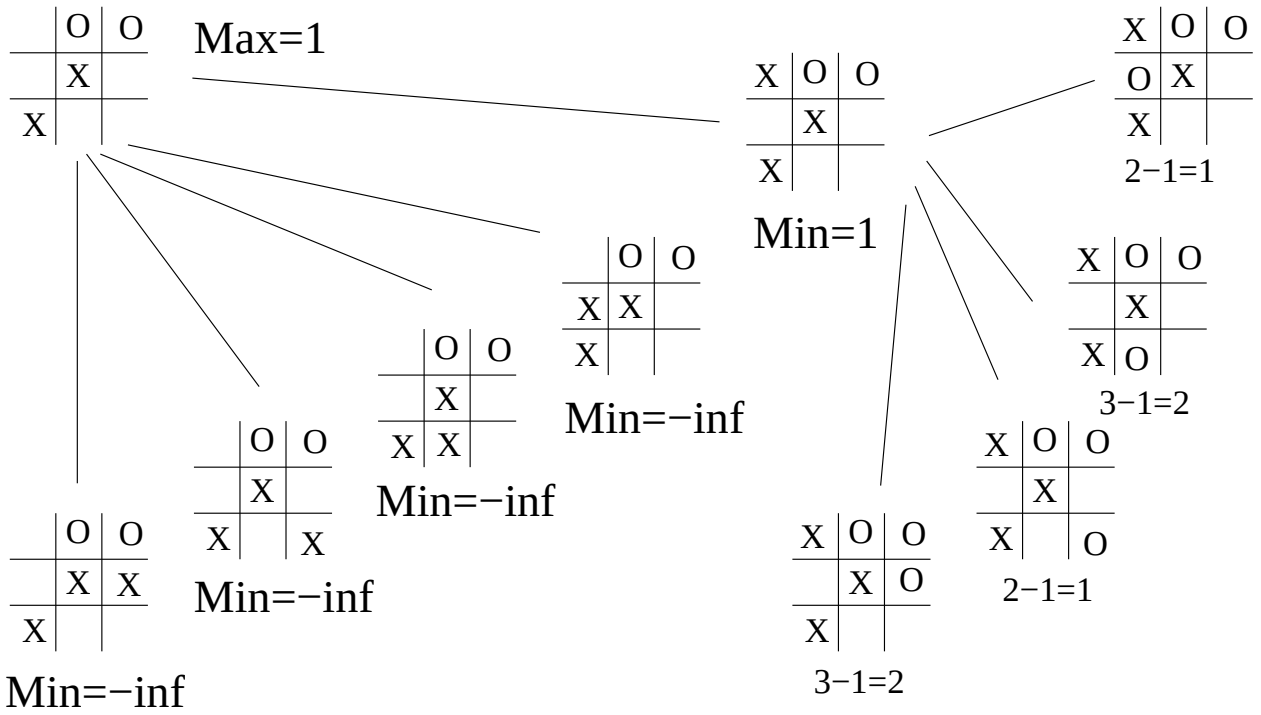
Suponiendo que MIN coloca su ficha en la posición superior de la columna central



La mejor jugada de MAX es colocar su ficha en la esquina inferior izquierda

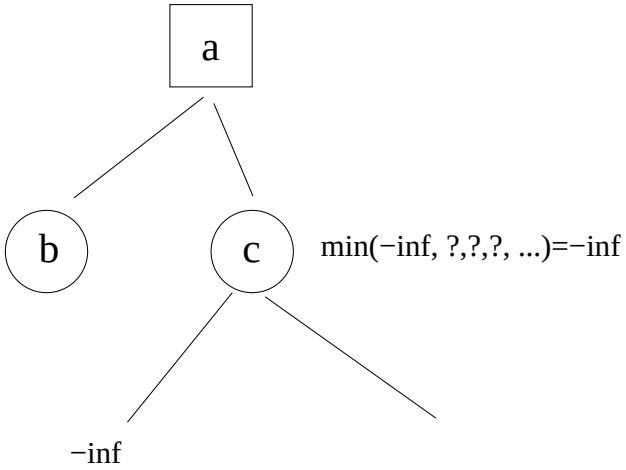
Ejemplo: 3 en raya (3)

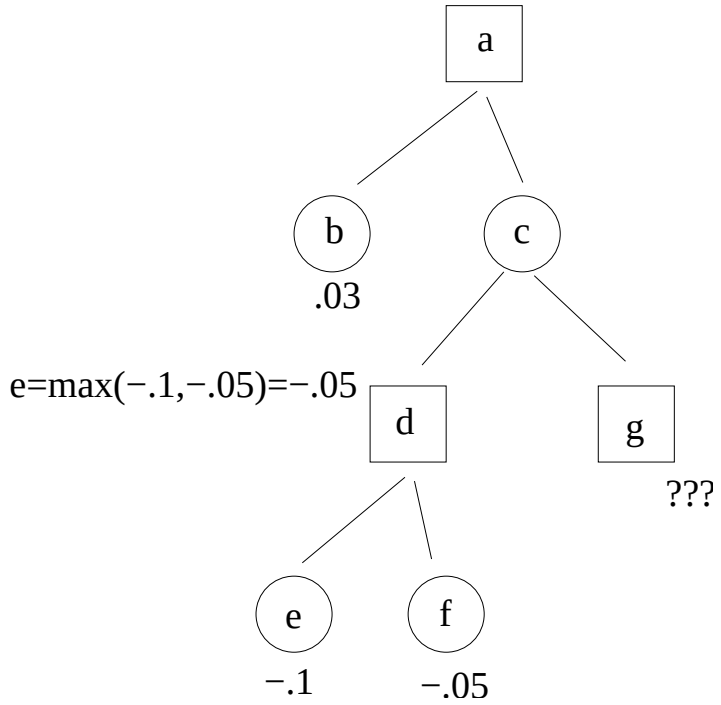
Suponiendo que MIN coloca su ficha en la esquina superior derecha



La mejor jugada de MAX es colocar su ficha en la esquina superior izquierda

Minimax con poda $\alpha\beta$

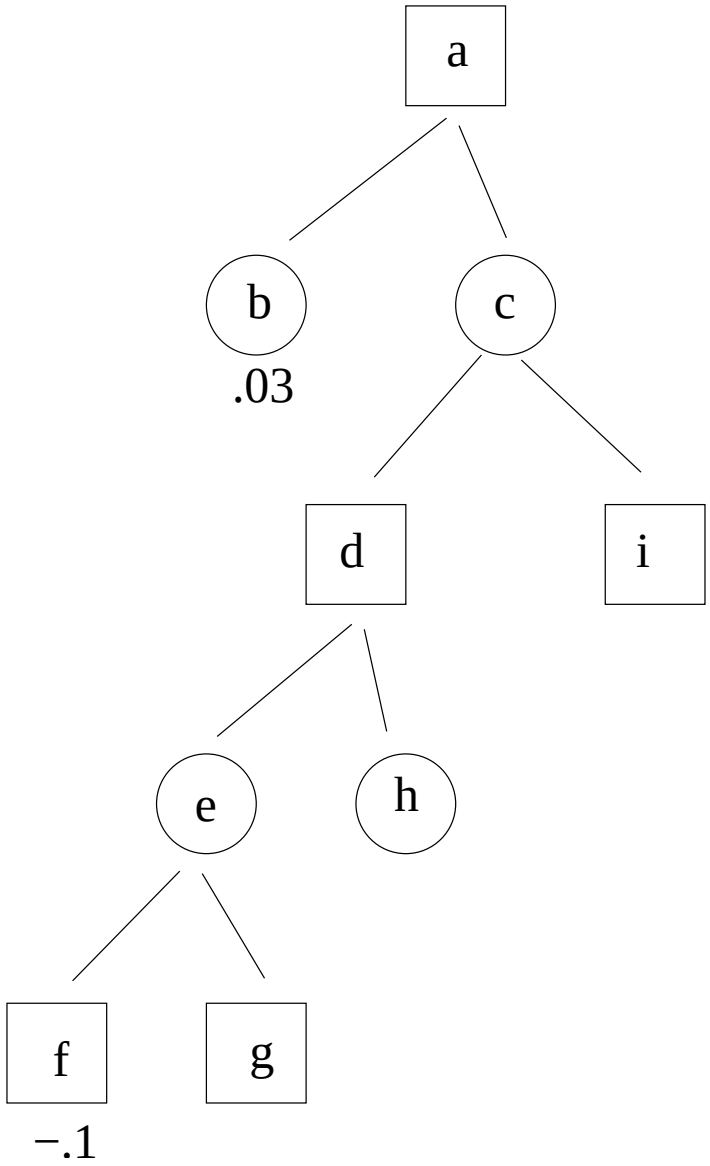




No tiene sentido seguir explorando sucesores de **c** ya que tenemos el mejor valor posible

En **c** tendremos $e = min(-.05, v_g)$, por lo tanto en **a** tendremos
$e = max(.03, min(-.05, v_g)) = .03$
Podemos pues podar todos los nodos de **g** ya que no aportan nada

Minimax con poda $\alpha\beta$



e(e) = min(-.1,g)

Como la rama b ya me da un .03 cualquier cosa peor no nos sirve ⇒ No hay que explorar g

e(d) = max(e(e), h) ⇒ Sí hay que explorar h

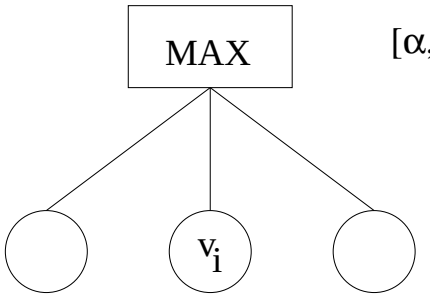
Al valor mínimo alcanzado hasta el momento para los nodos max le llamaremos cota α y nos da un límite inferior de e(n).

Al valor máximo alcanzado por los nodos min le llamaremos cota β y nos dará un límite superior de e(n)

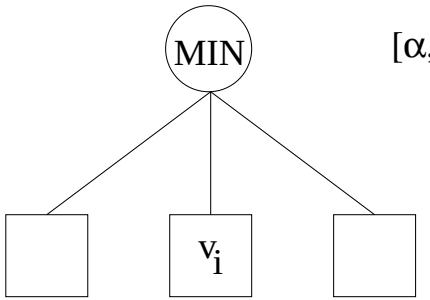
En el ejemplo el nodo a (max) tiene de momento un valor mínimo de .03 proporcionado por su hijo b

Fijémonos en que el valor que se puede asignar a un nodo max viene aportado por nodos min y viceversa

Minimax con poda $\alpha\beta$



Si $v_i > \alpha$ entonces modificar α
Si $v_i \geq \beta$ entonces poda β
Retornar α



Si $v_i < \beta$ entonces modificar β
Si $vi \leq \alpha$ entonces poda α
Retornar β

Las cotas α y β se transmiten de padres a hijos de 1 en 1 y en el orden de visita de los nodos.
α es la cota inferior de un nodo max - β es la cota superior de un nodo min
$\Longrightarrow$ La efectividad de la poda depende del orden de exploración de los descendientes

Algoritmo Minimax con poda

Función: valorMax (g,α,β)

```
si estado_terminal(g) entonces
    retorna valor(g)
sino
    para cada mov ∈ movs_posibles(g) hacer
        α ← max(α,valoeMin(aplicar(mov,g) ,α,β))
        si α ≥ β entonces
            retorna β
        fin
    fin
    retorna α
fin
```

Función: valorMin (g,α,β)

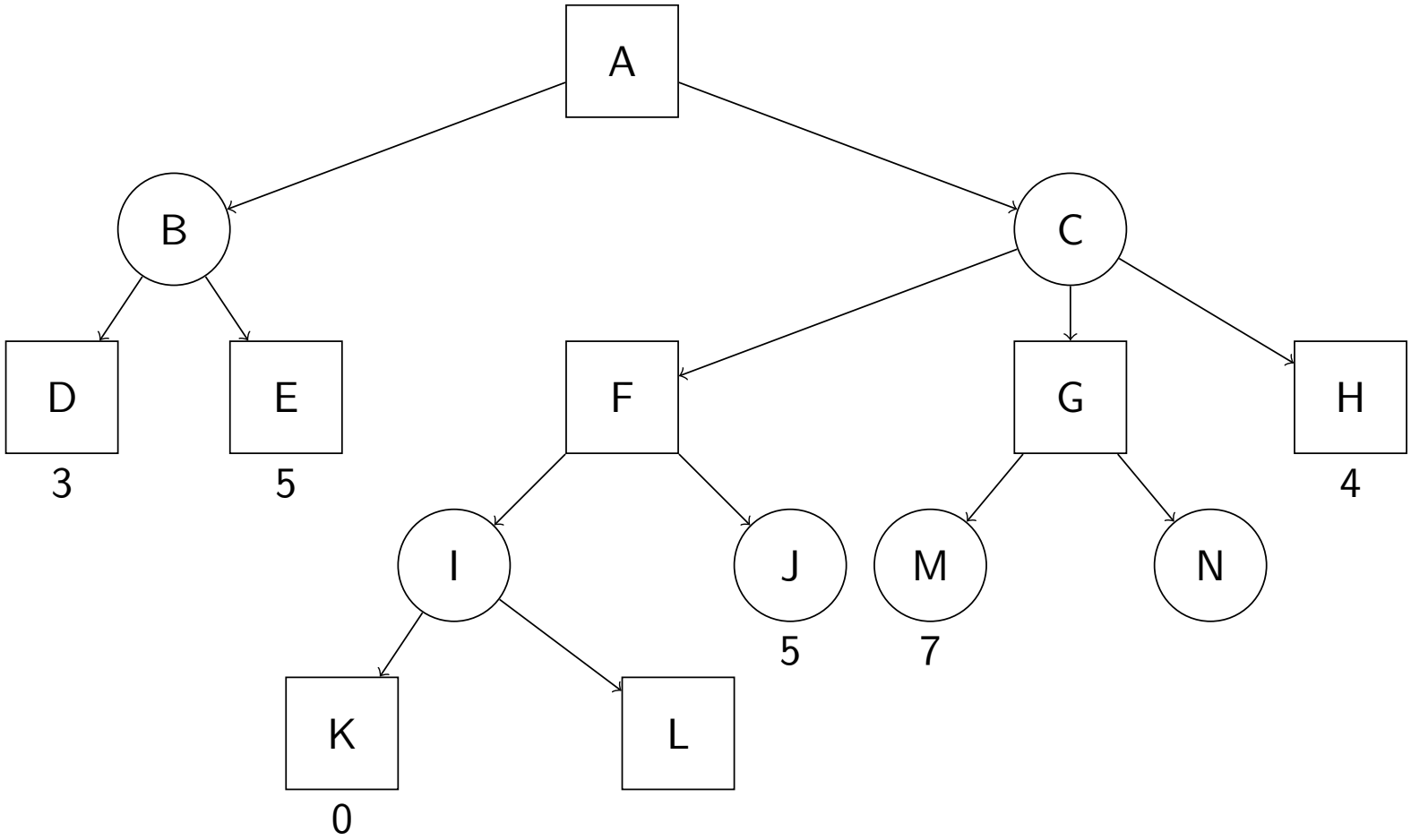
```
si estado_terminal(g) entonces
    retorna valor(g)
sino
    para cada mov ∈ movs_posibles(g) hacer
        β ← min(β,valorMax(apply(mov,g) ,α,β))
        si α ≥ β entonces
            retorna α
        fin
    fin
    retorna β
fin
```

El recorrido se inicia llamando a la función valorMax con $\alpha = -\infty$ $\beta = +\infty$

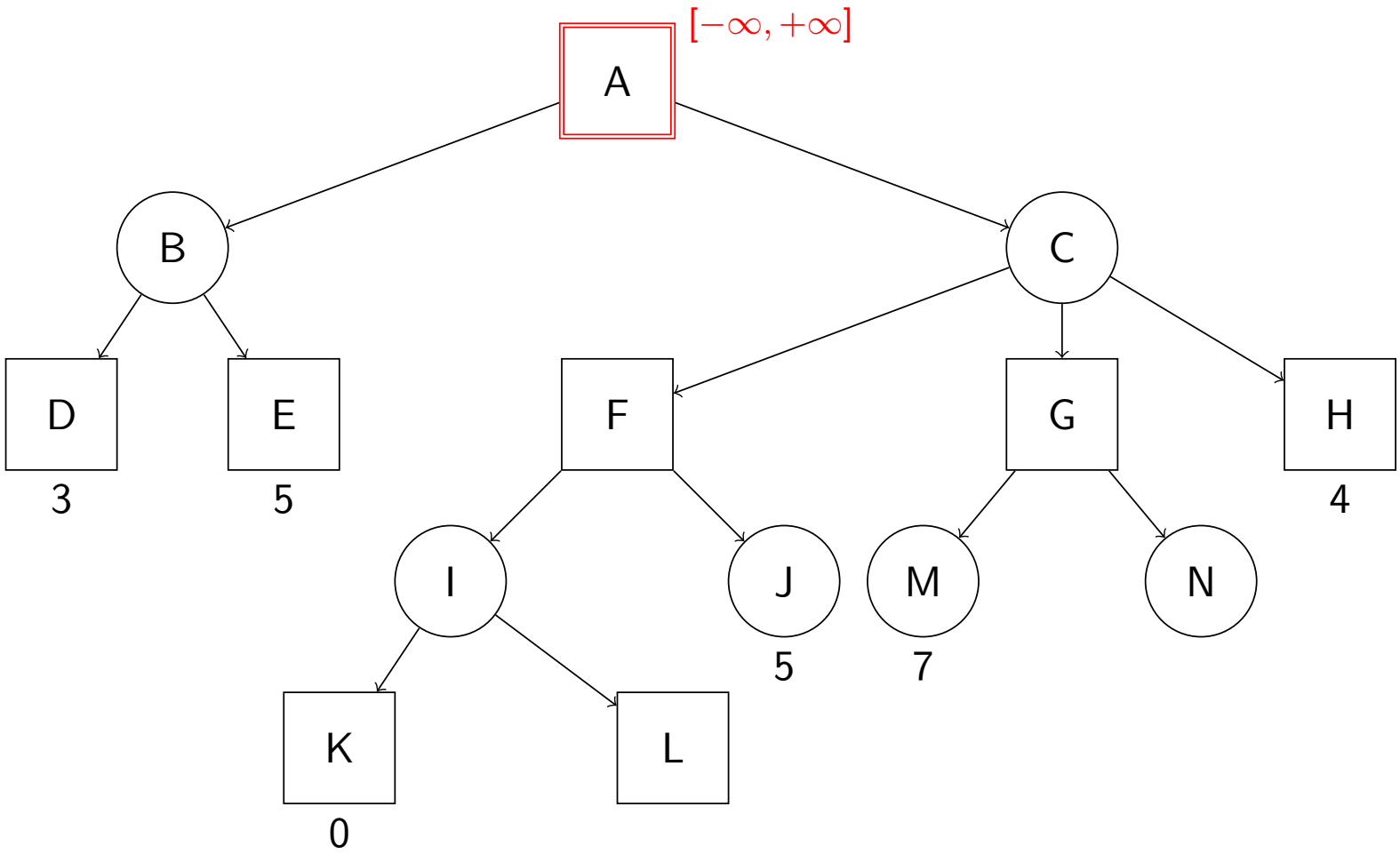
En la función valorMax α es el valor que se actualiza y β es el valor de la mejor jugada

En la función valorMin β es el valor que se actualiza y α es el valor de la mejor jugada

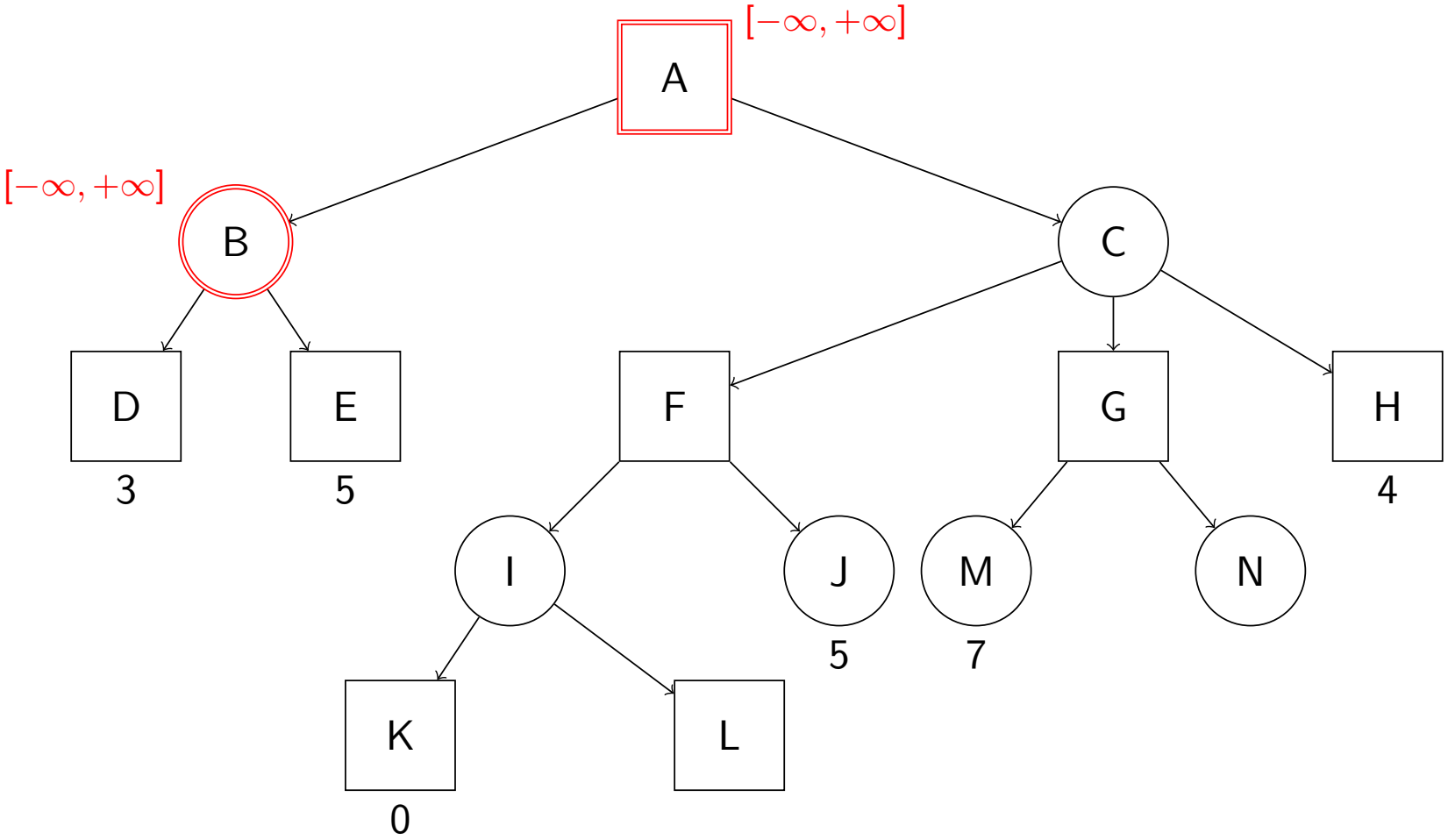
Poda $\alpha\beta$: Ejemplo



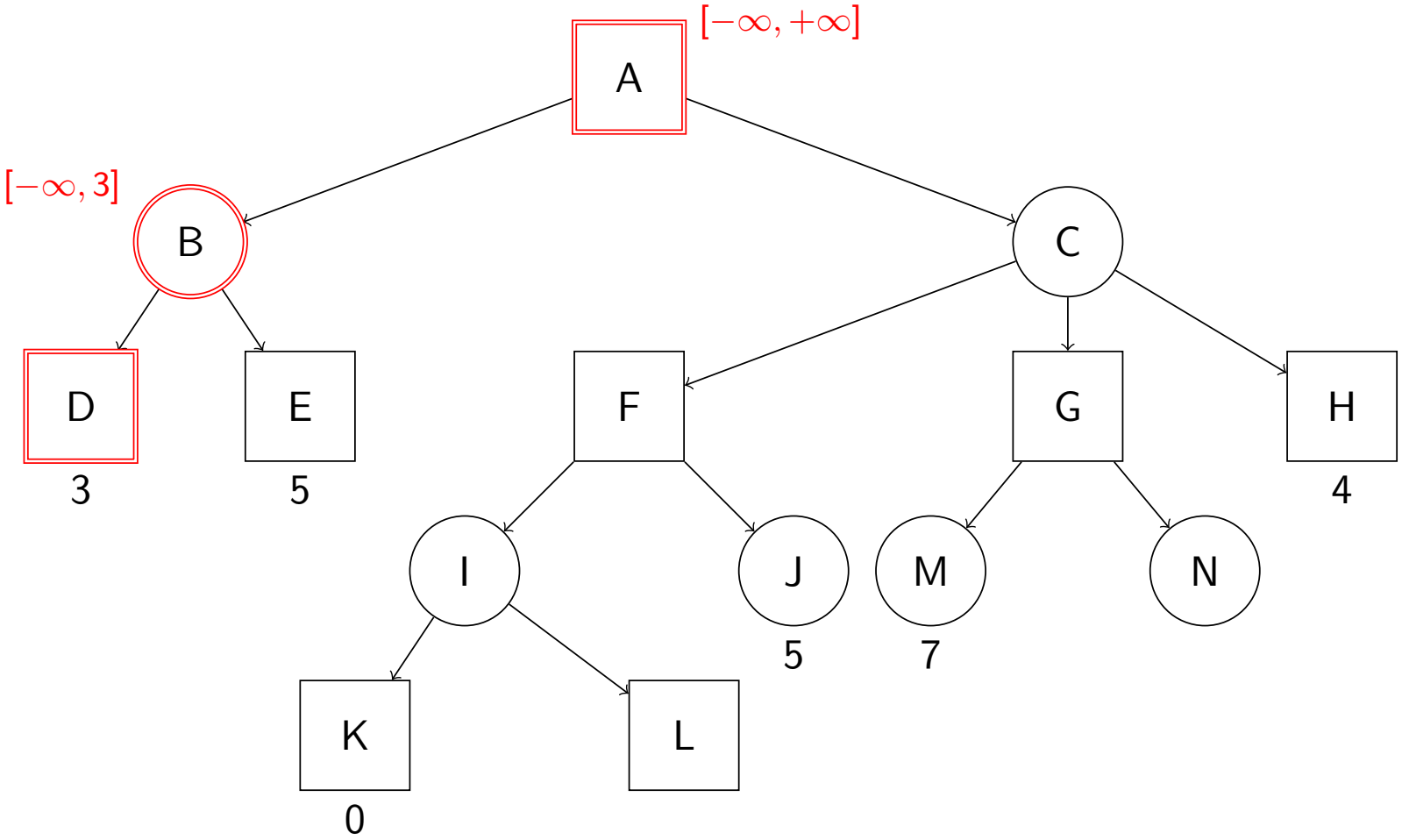
Poda $\alpha\beta$: Ejemplo



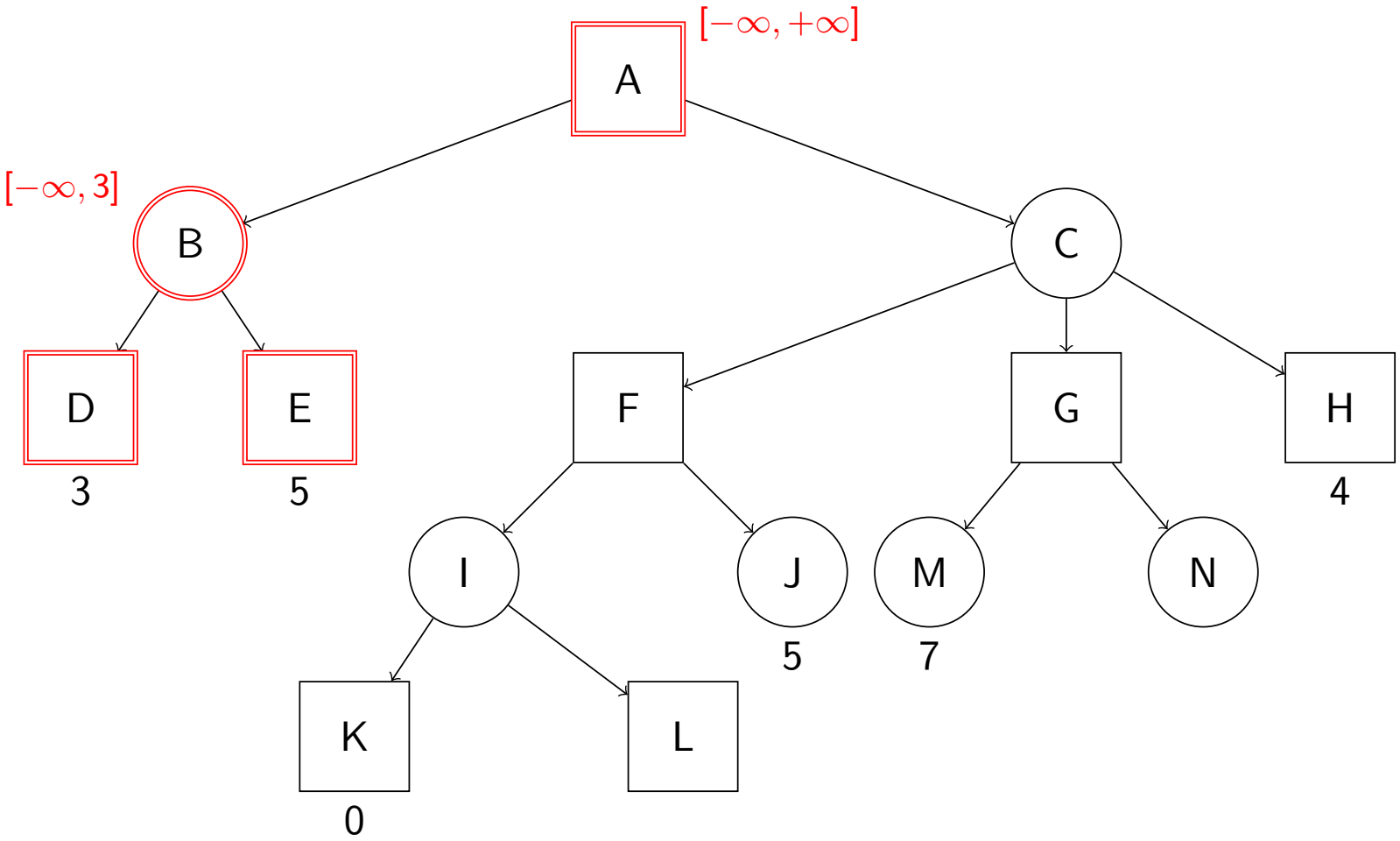
Poda $\alpha\beta$: Ejemplo



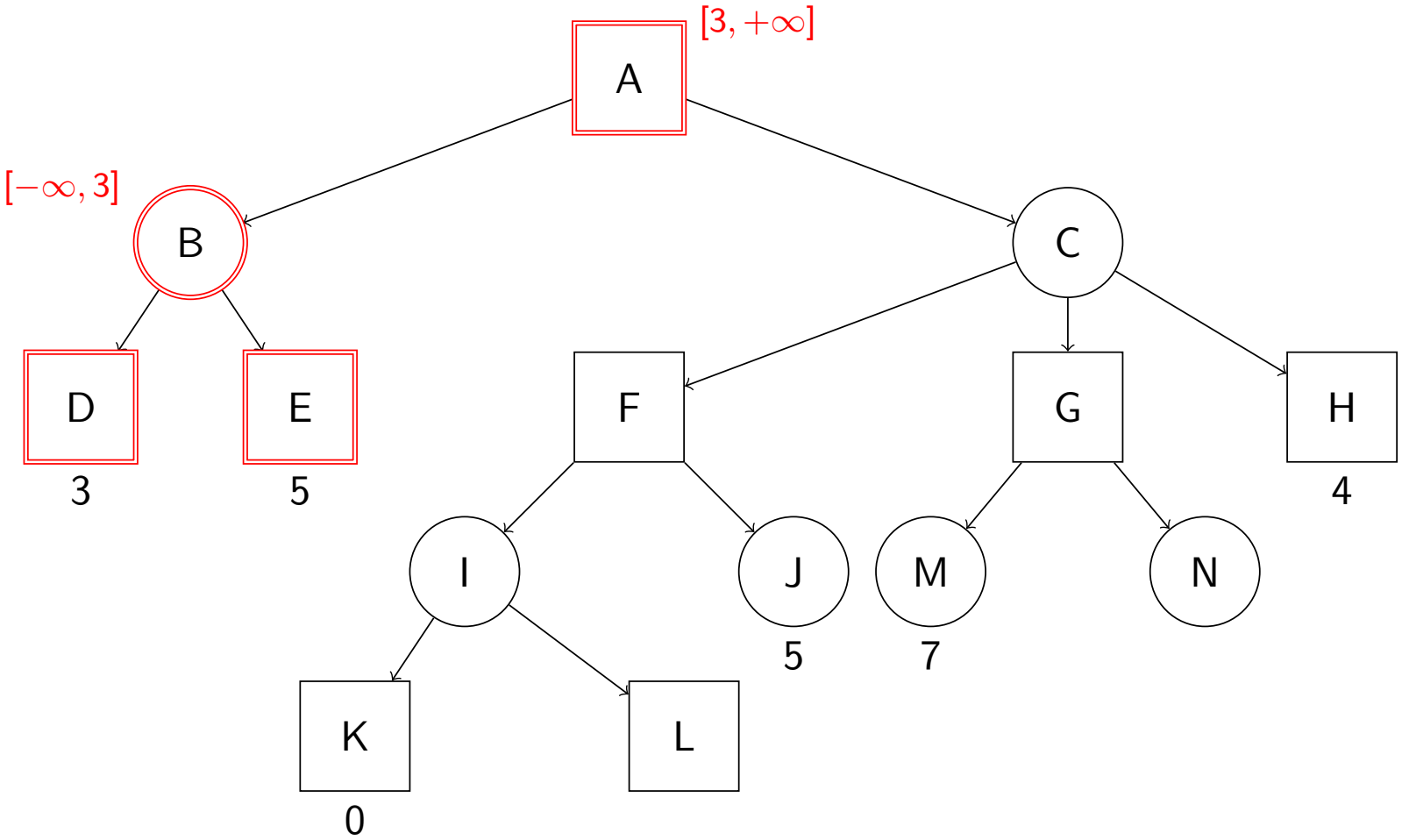
Poda $\alpha\beta$: Ejemplo



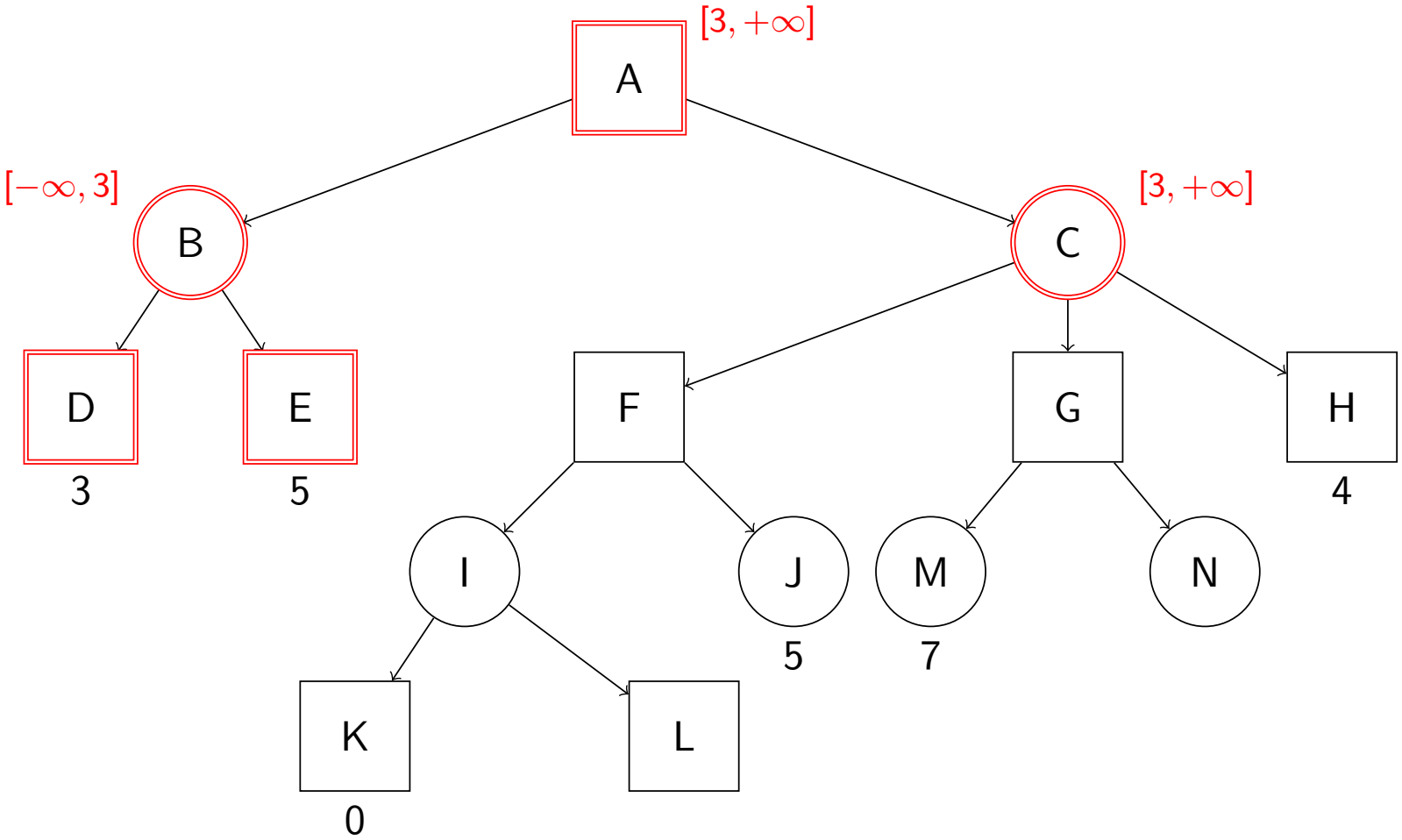
Poda $\alpha\beta$: Ejemplo



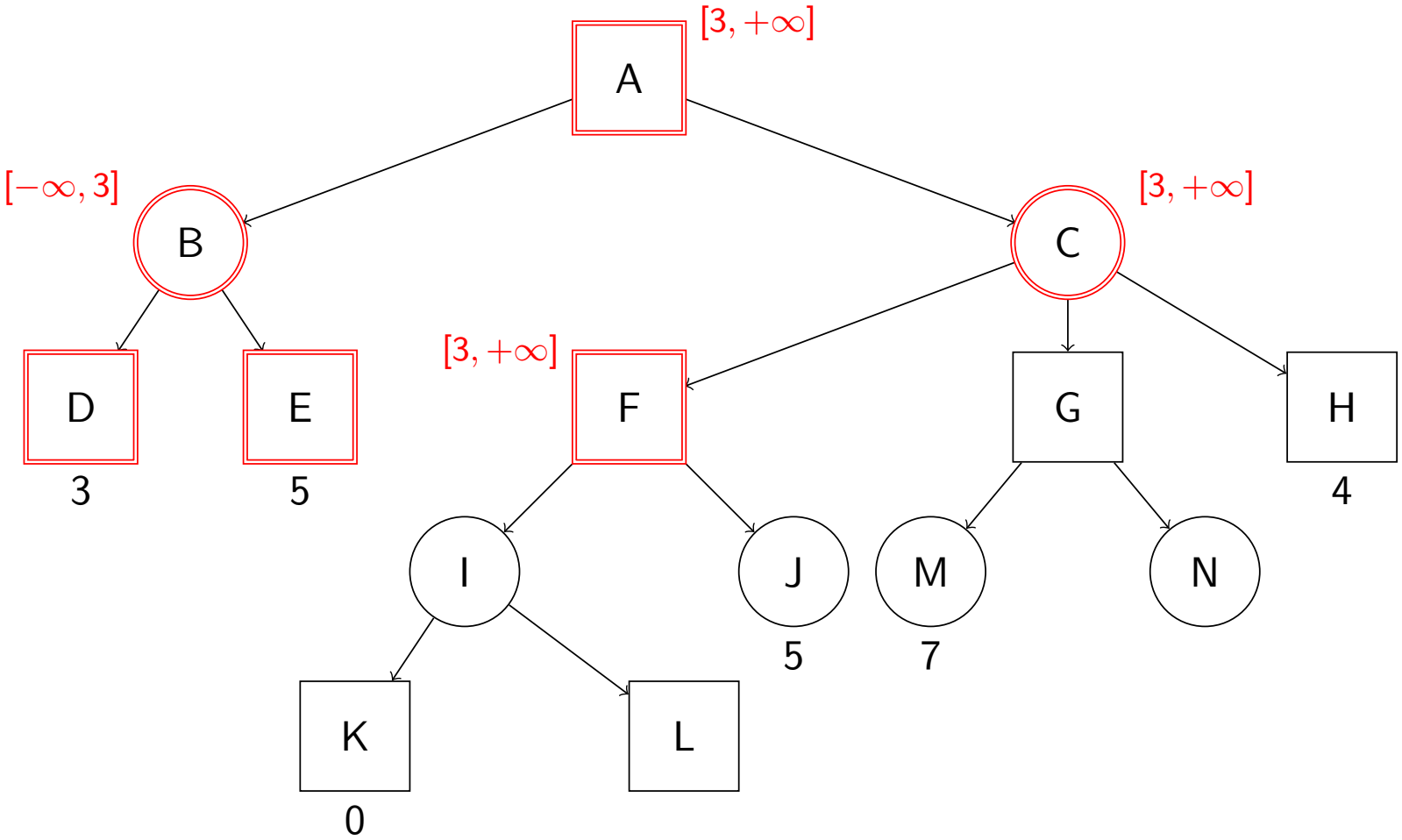
Poda $\alpha\beta$: Ejemplo



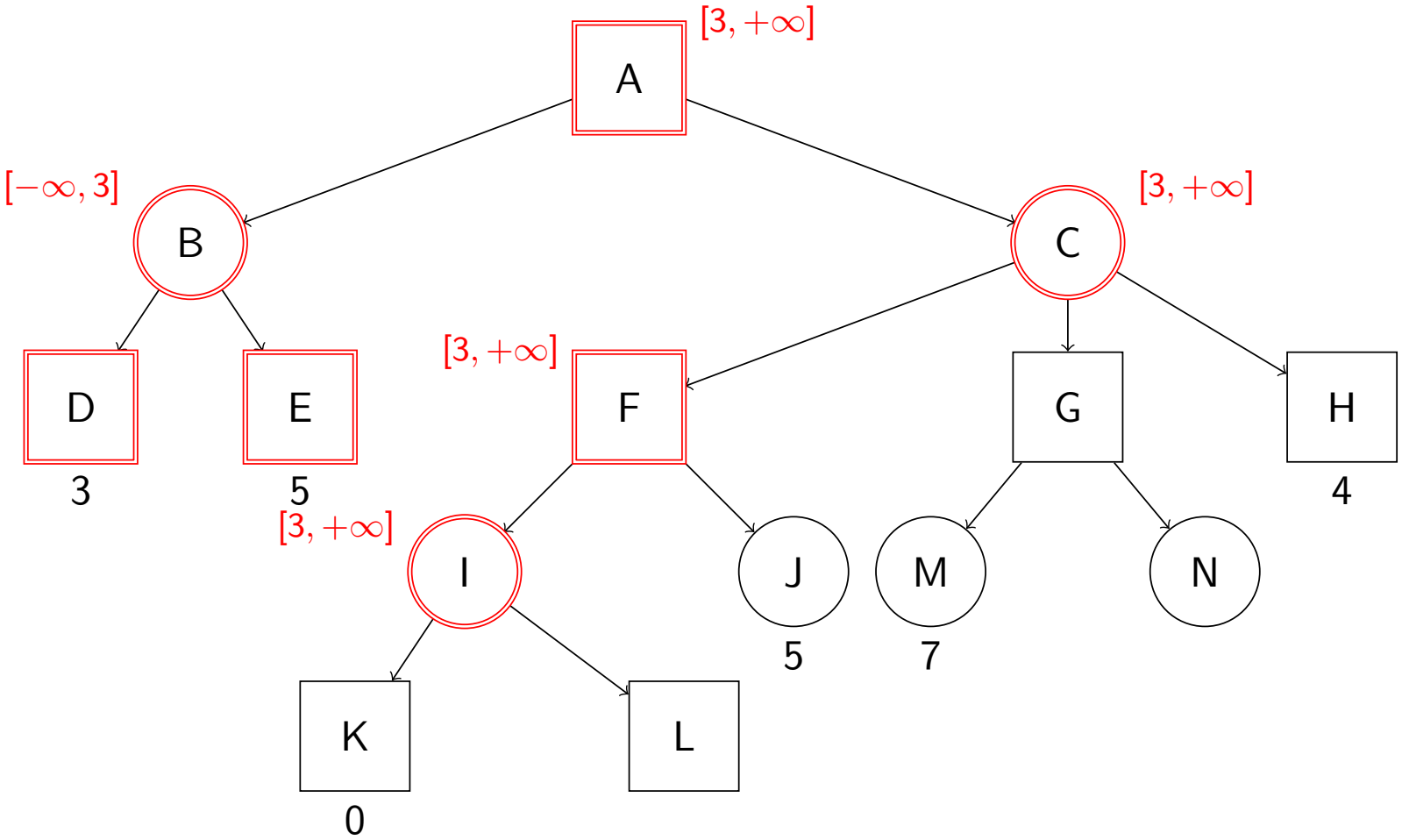
Poda $\alpha\beta$: Ejemplo



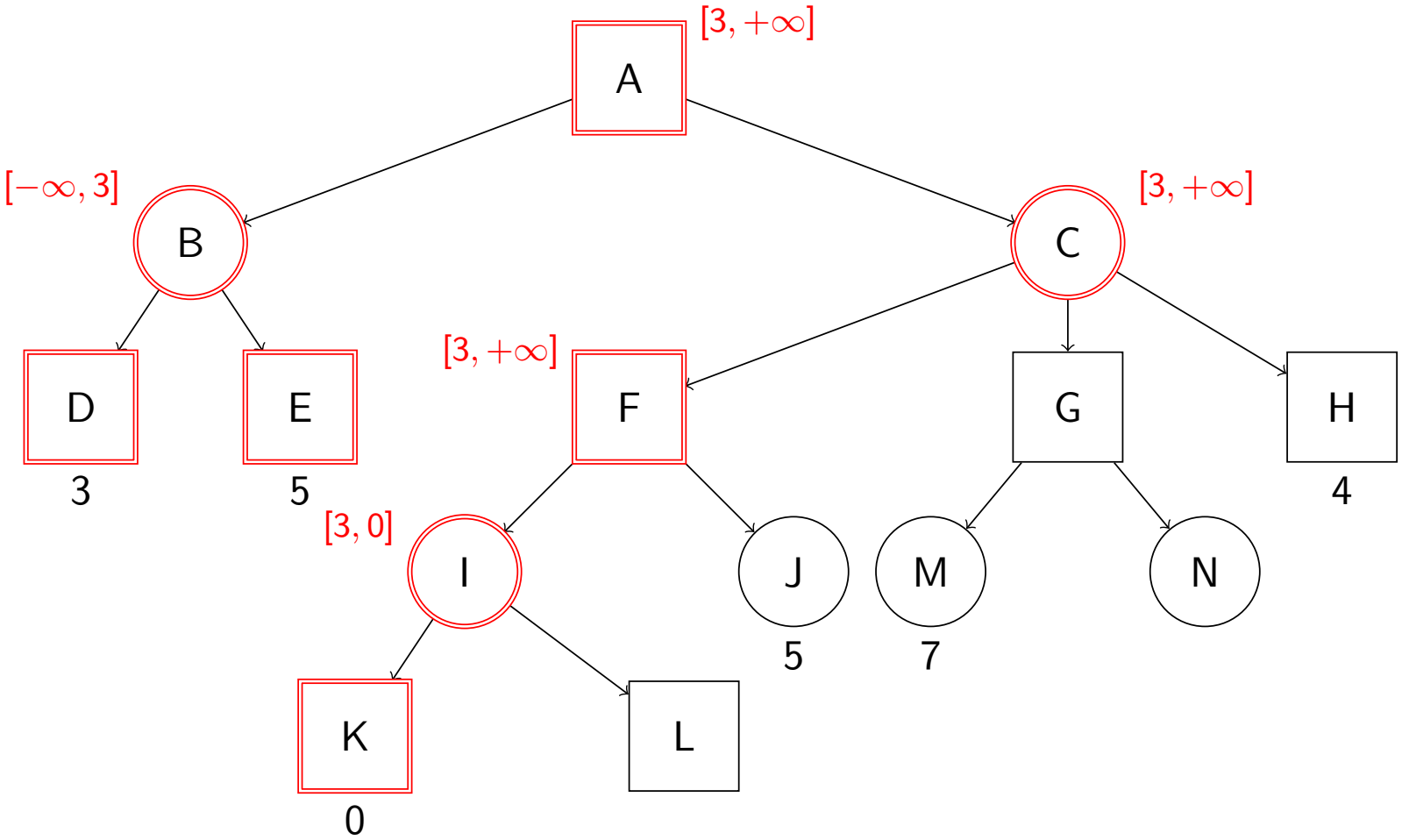
Poda $\alpha\beta$: Ejemplo



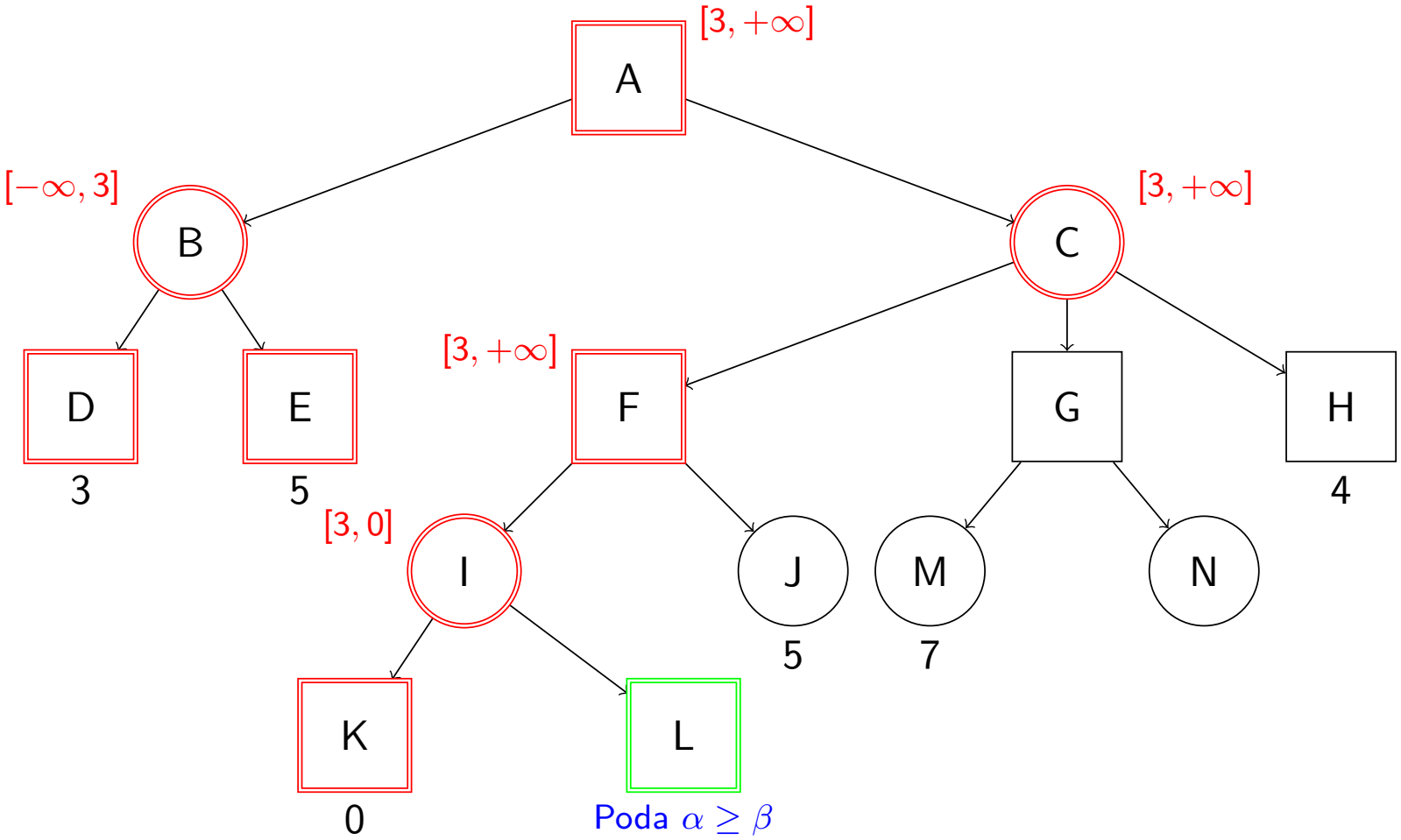
Poda $\alpha\beta$: Ejemplo



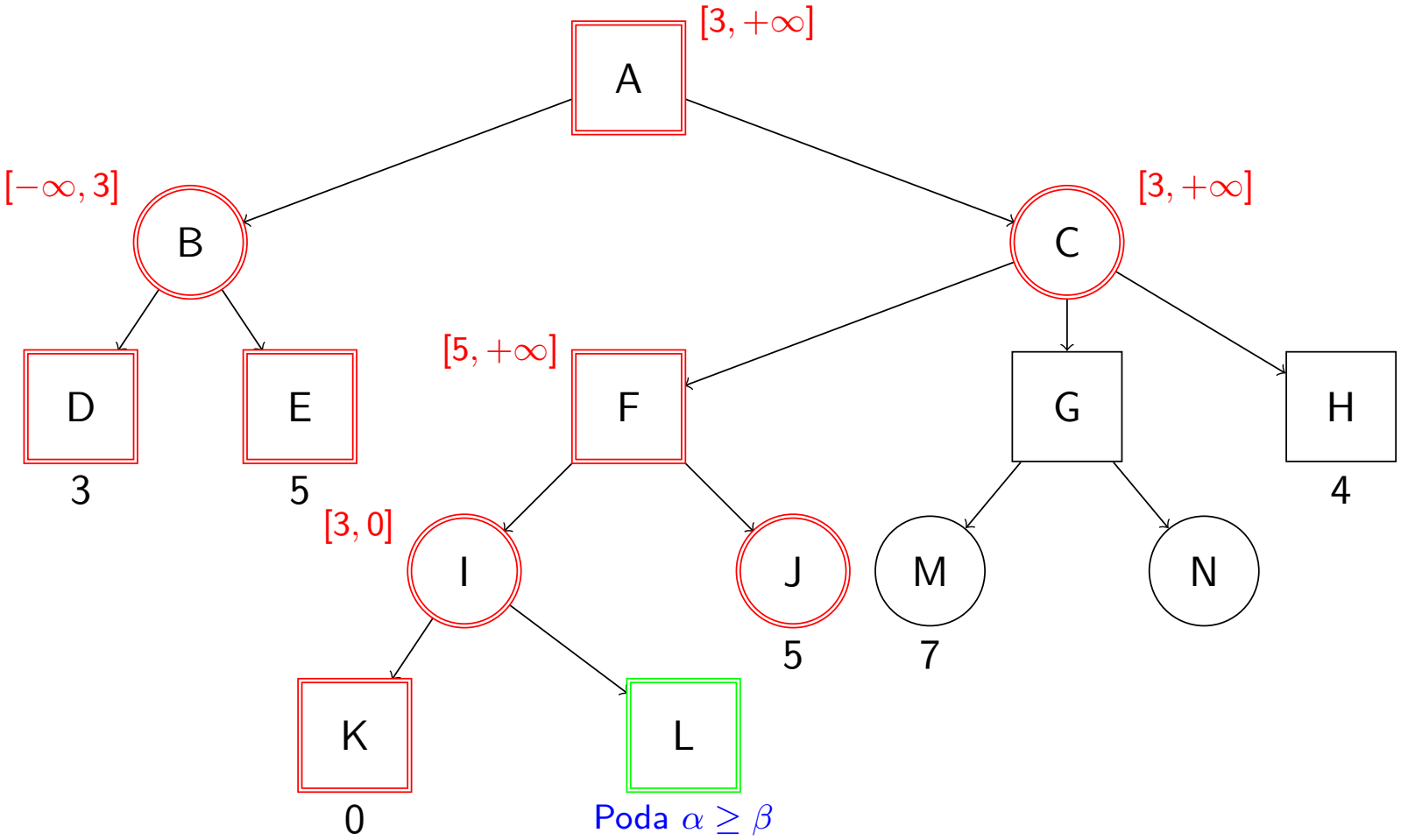
Poda $\alpha\beta$: Ejemplo



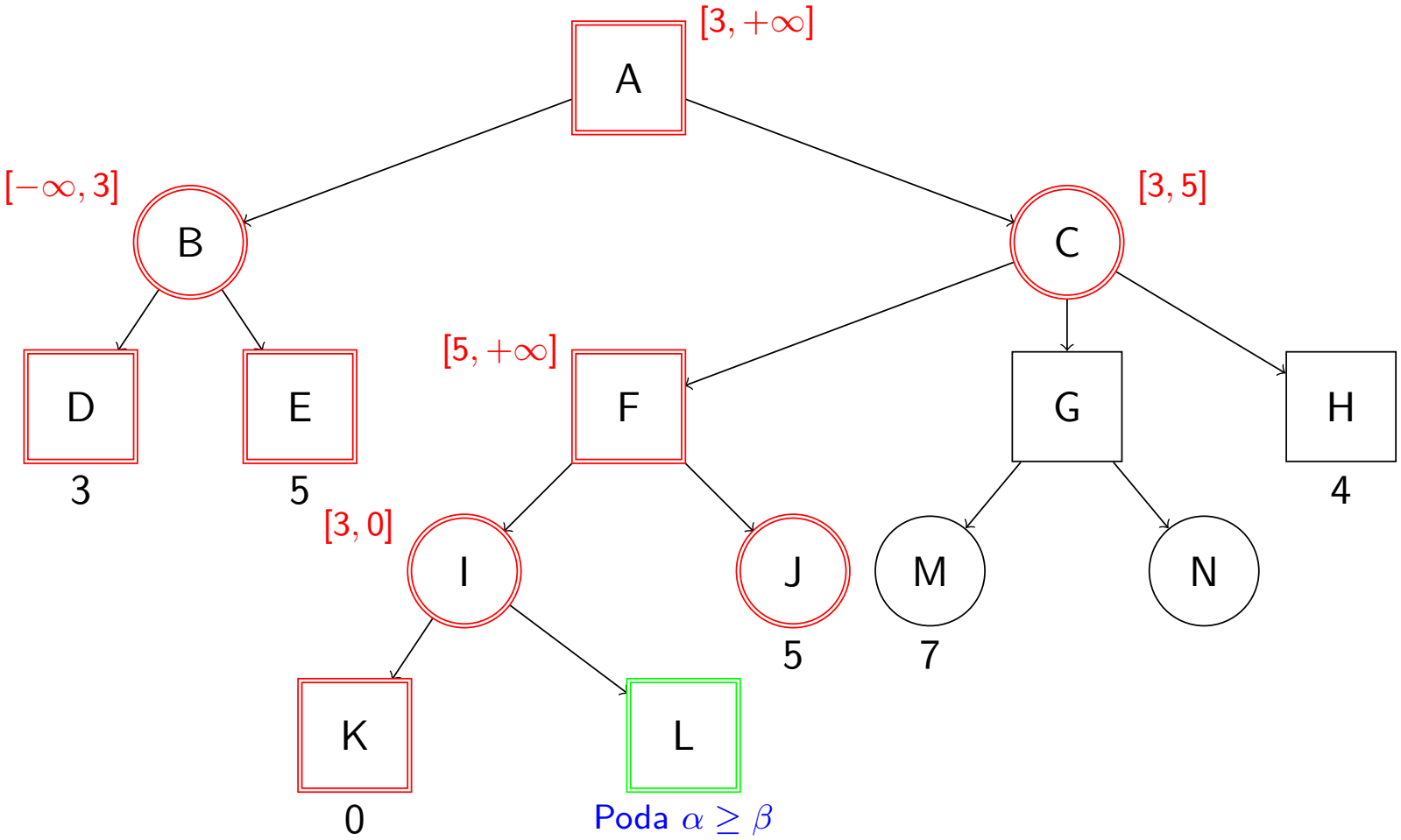
Poda $\alpha\beta$: Ejemplo



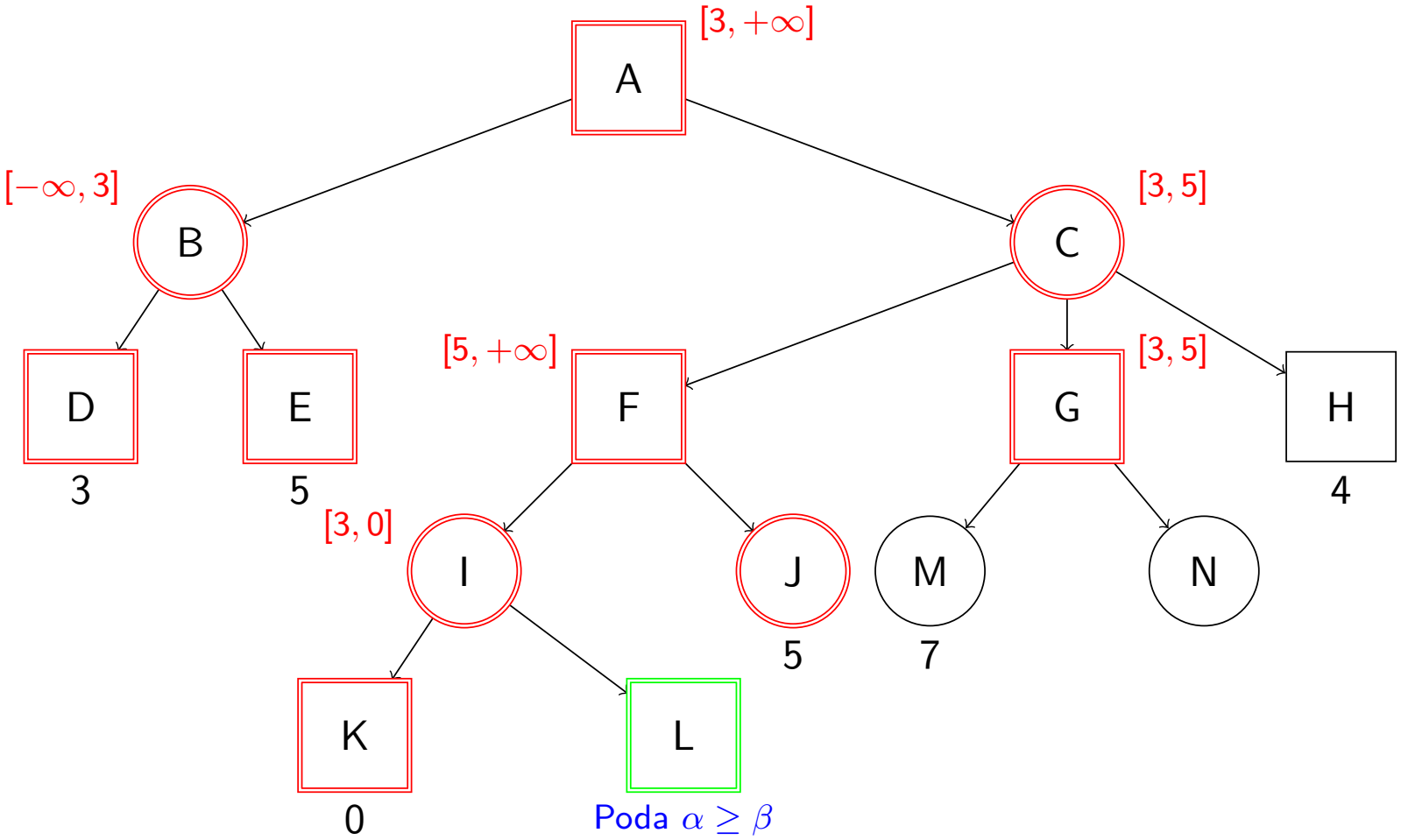
Poda $\alpha\beta$: Ejemplo



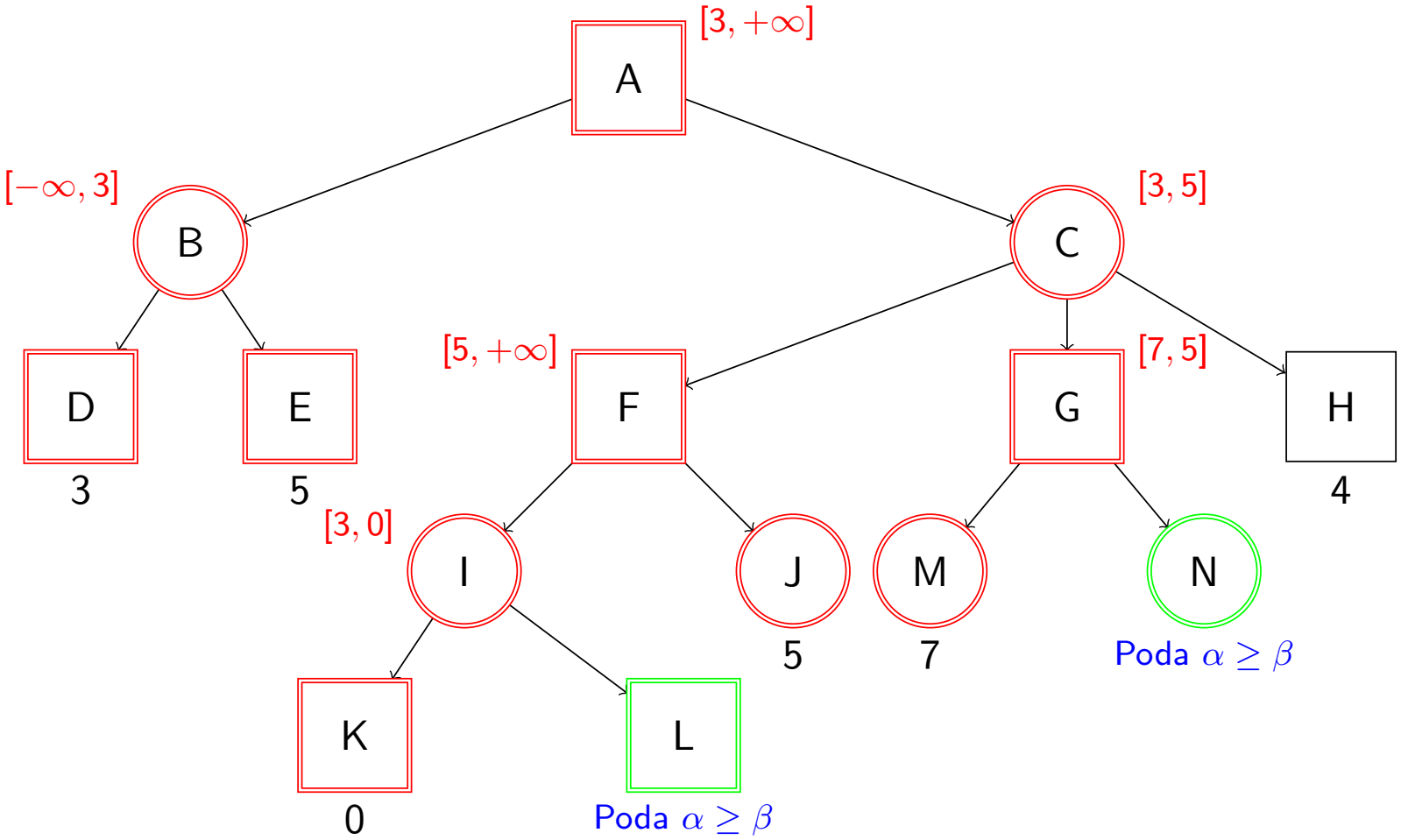
Poda $\alpha\beta$: Ejemplo



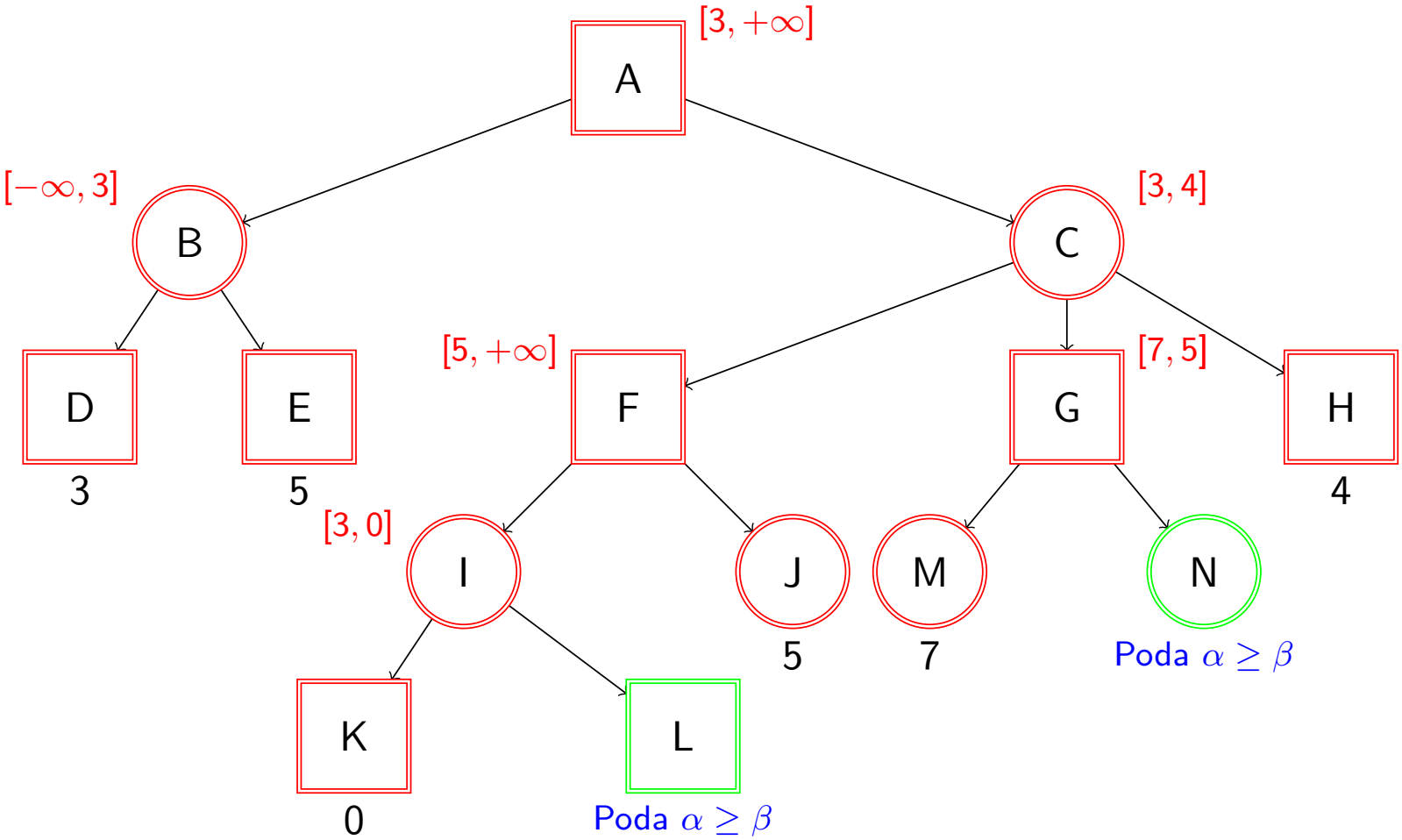
Poda $\alpha\beta$: Ejemplo



Poda $\alpha\beta$: Ejemplo



Poda $\alpha\beta$: Ejemplo



Poda $\alpha\beta$: Ejemplo