ÁRBOLES DE JUEGO

UNIVERSIDAD
COMPLUTENSE
MADRID

ALBERTO VERDEJO

Motivación

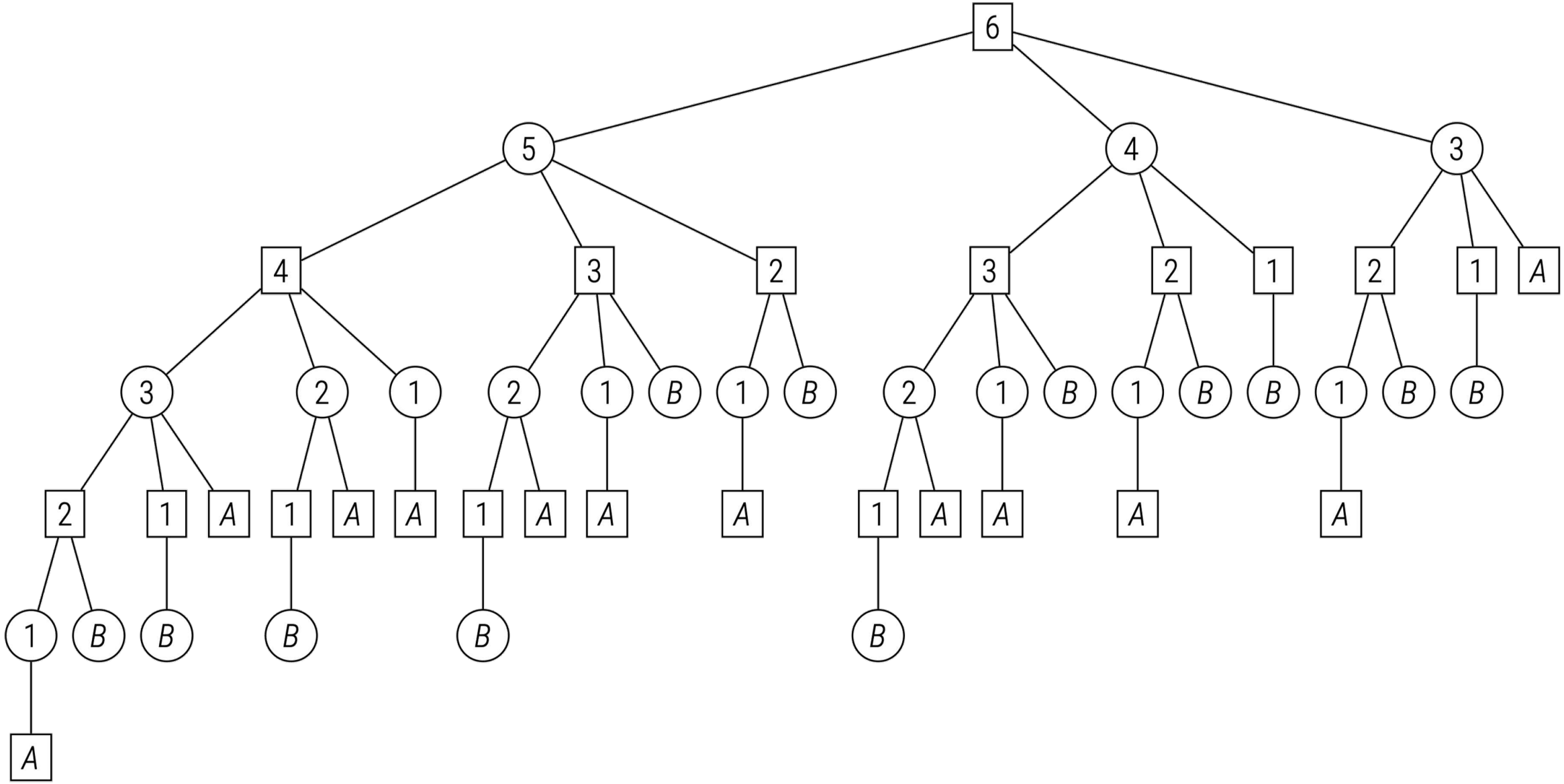
- Queremos un programa que sepa jugar, lo haga de la mejor forma posible y en un tiempo razonable.

	Determinista	No determinista (aleatoriedad)
Totalmente observable	Ajedrez Damas Go Othello	Backgammon Monopoly
Parcialmente observable	Hundir la flota	Juegos de cartas

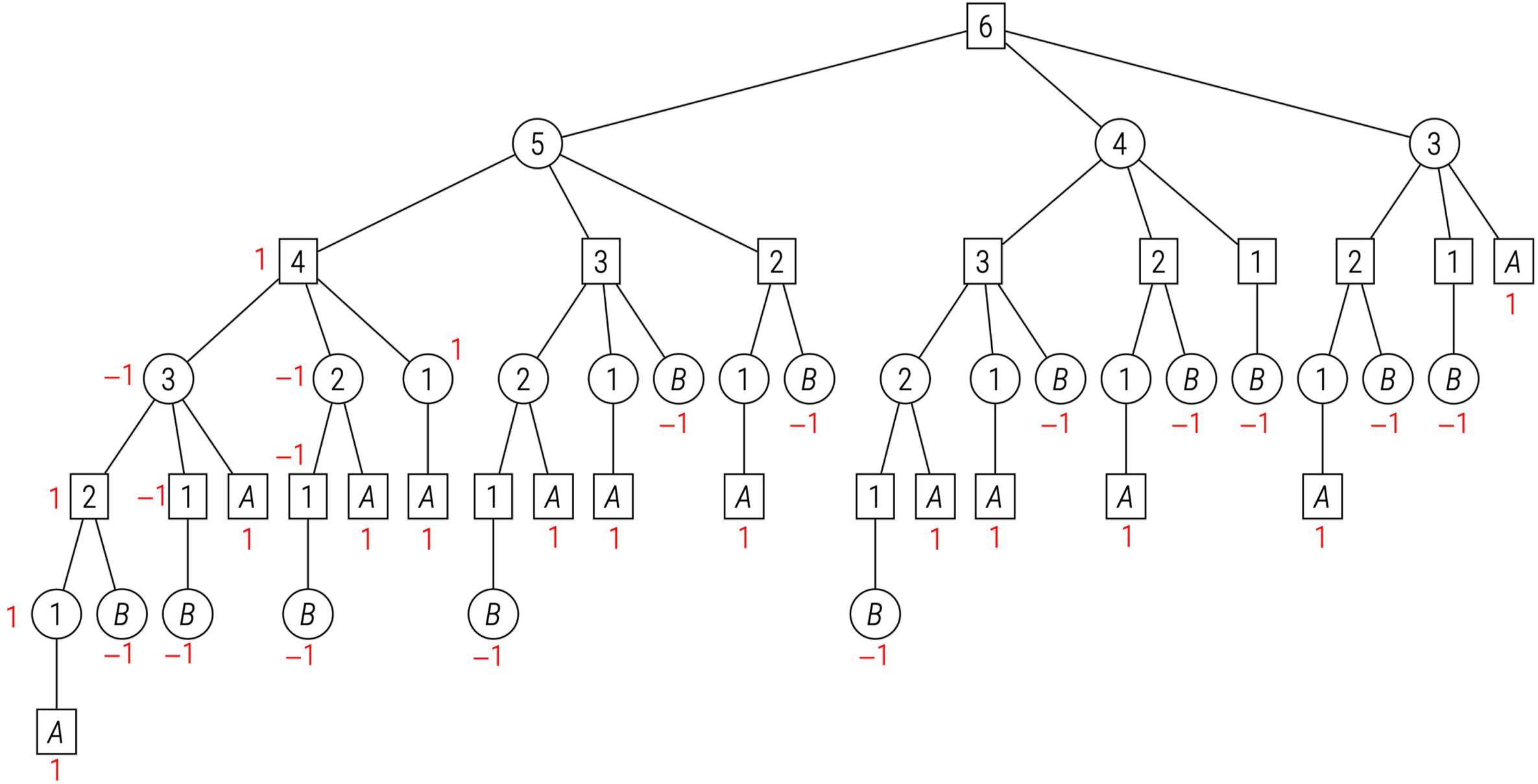
Juego de NIM

- Dos jugadores *A* y *B* y un tablero que inicialmente contiene *n* palillos.
- Los jugadores se alternan comenzando *A*.
- Un movimiento legal consiste en eliminar 1, 2 o 3 palillos.
- Pierde el jugador que elimina el último palillo y el otro gana.

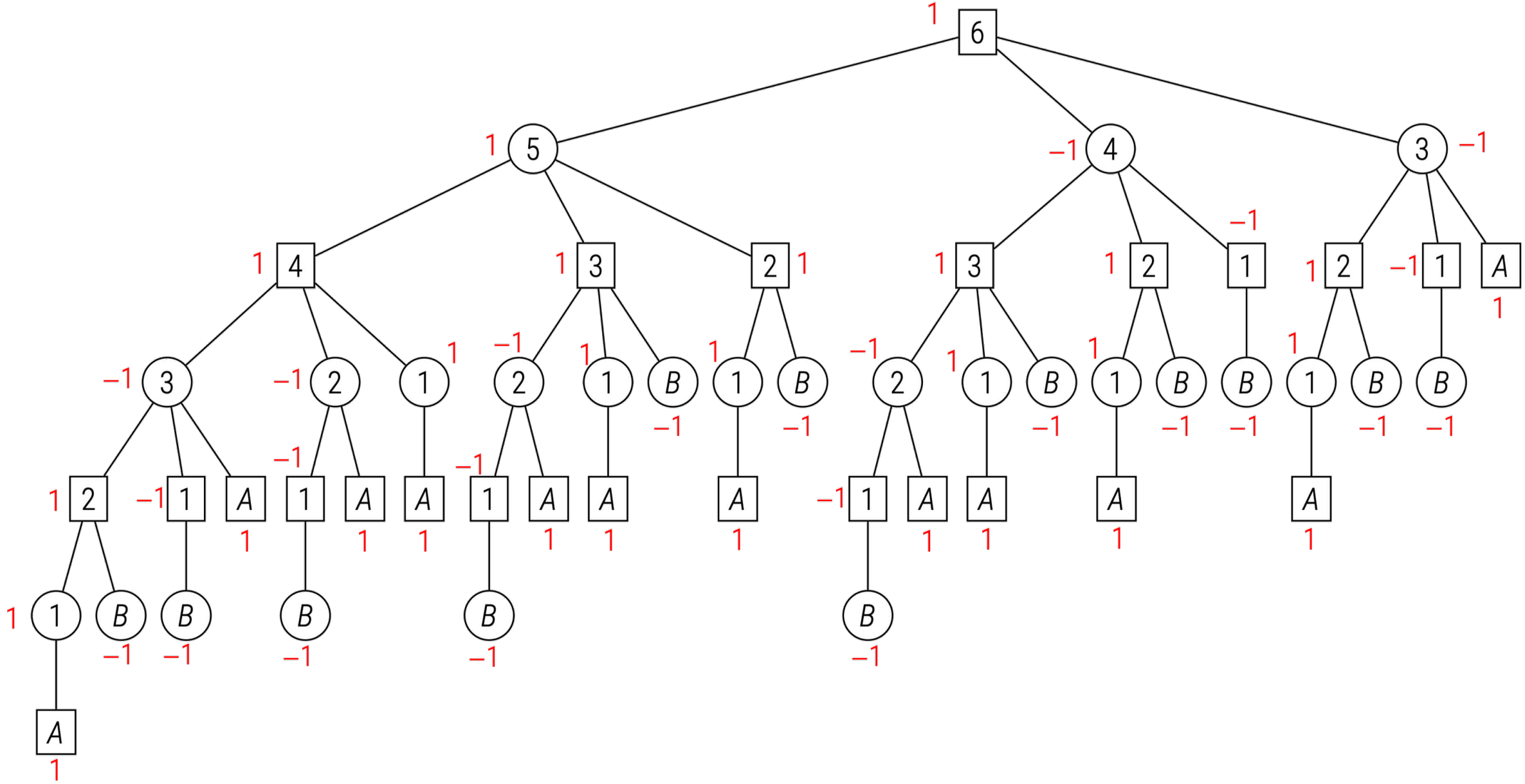
Juego de NIM con 6 palillos



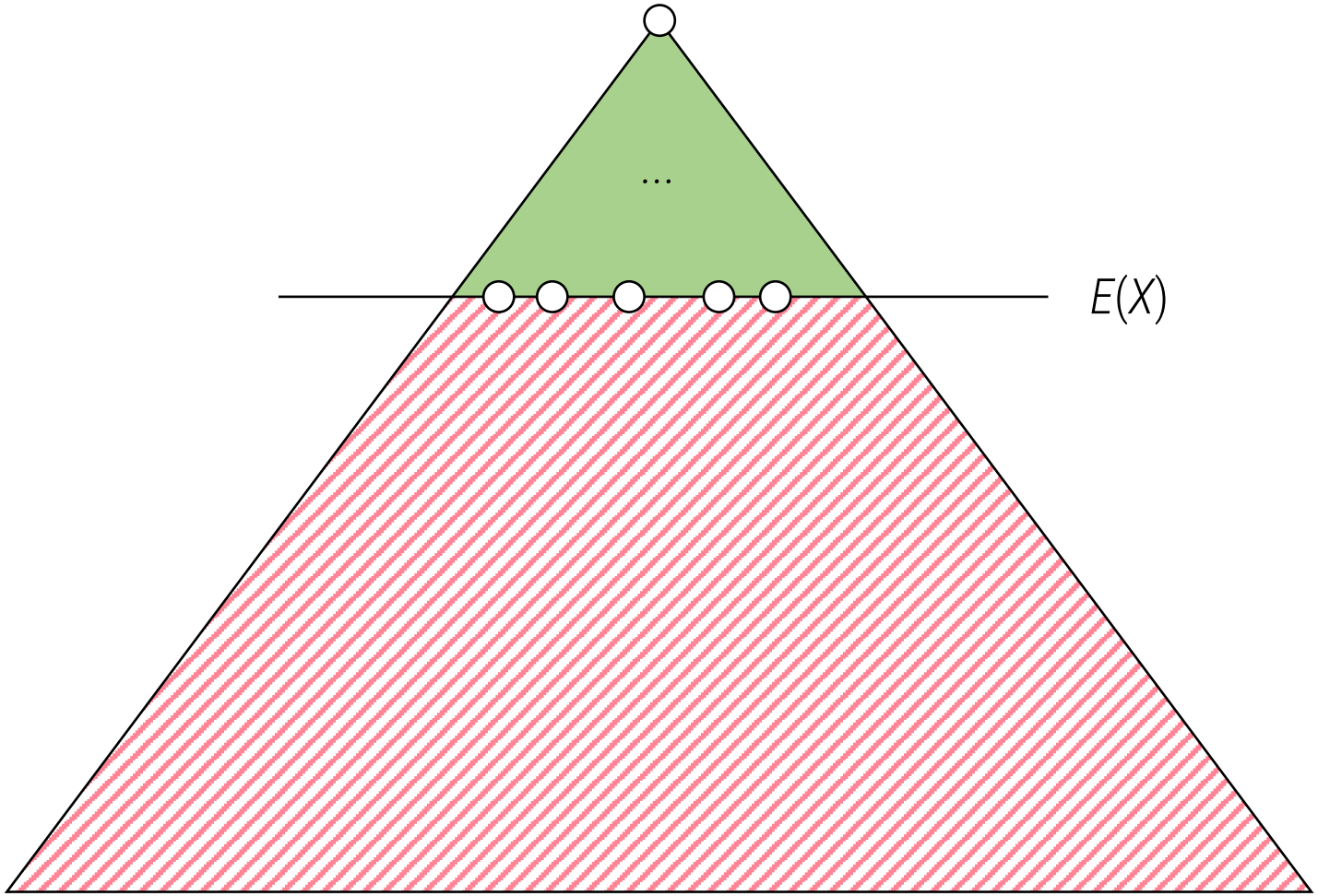
Juego de NIM con 6 palillos



Juego de NIM con 6 palillos



Construcción parcial del árbol de juego



Algoritmo de minimax

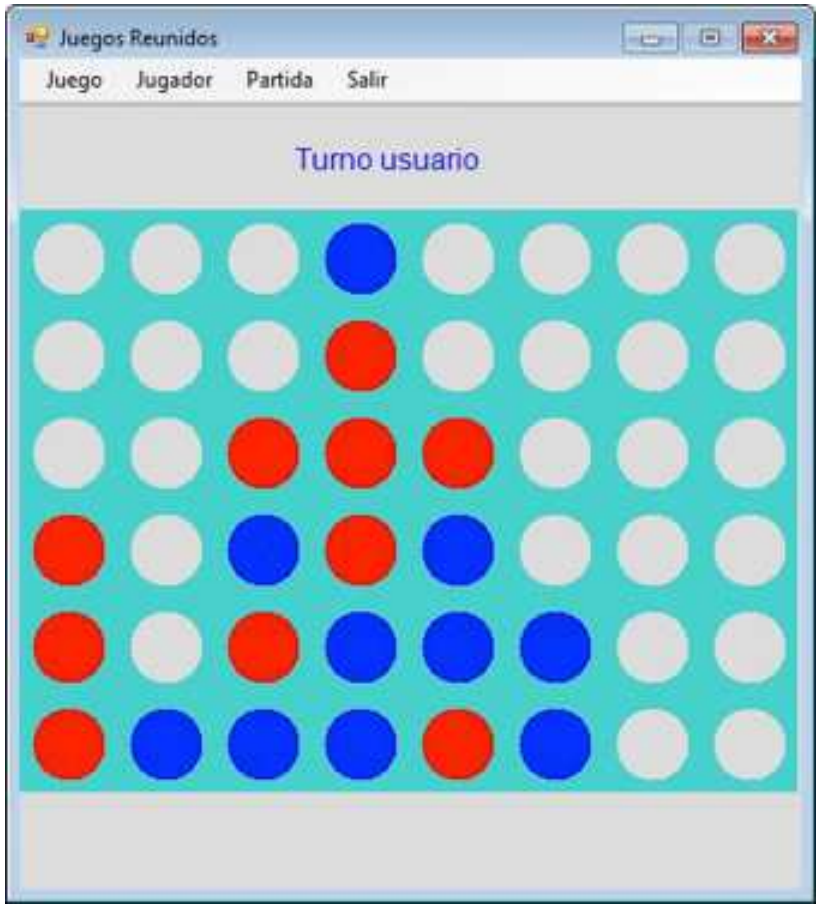
```
jugada Jugador::juega(Juego const& EJ) {
    jugada mejorC;
    int mejorV = -∞;
    for (jugada c : EJ.jugadasDesde()) {
        int v = valoraMin(EJ.aplica(c), EJ.turno(), nivel - 1);
        if (v > mejorV) {
            mejorV = v;
            mejorC = c;
        }
    }
    return mejorC;
}
```

Algoritmo de minimax

```
int Jugador::valoraMin(Juego const& EJ, Turno t_orig, int n) {
   if (EJ.terminal() || n == 0)
      return Heuristica(EJ, t_orig);
   else {
      int mejorV = +∞;
      for (jugada c : EJ.jugadasDesde())
         mejorV = min(valoraMax(EJ.aplica(c), t_orig, n - 1), mejorV);
      return mejorV;
   }
}

int Jugador::valoraMax(Juego const& EJ, Turno t_orig, int n) {
  // igual que valoraMin pero calculando un máximo
}
```

Conecta 4



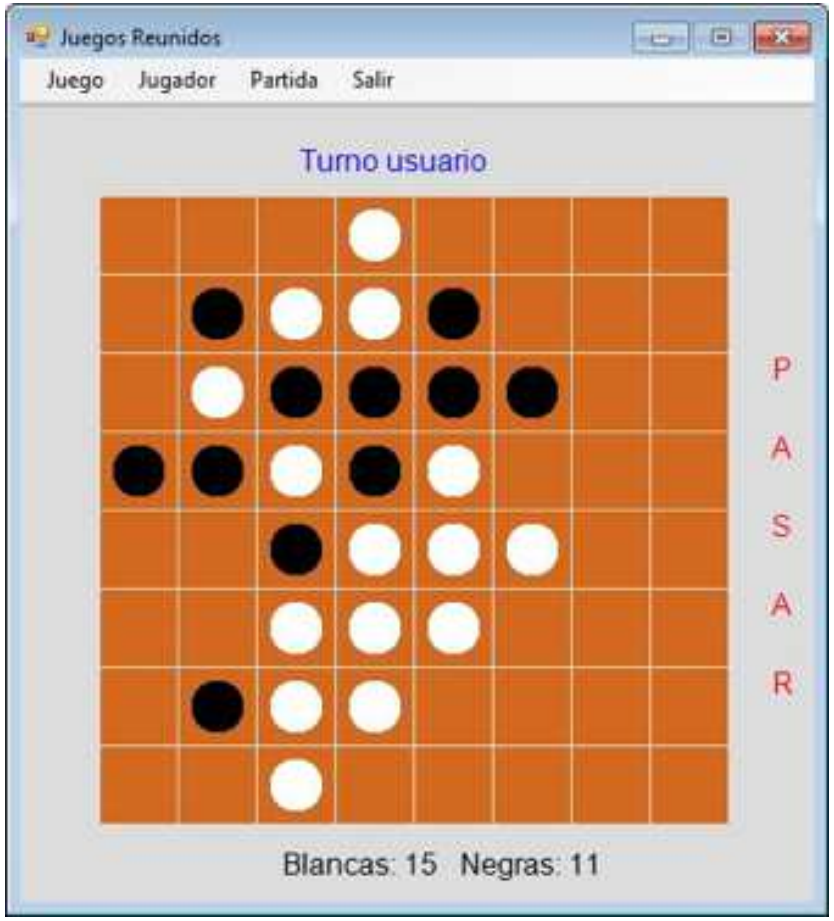
La heurística suma el valor de todas las posibles líneas de 4 posiciones del jugador y le resta la suma de los valores de las del contrincante:

$$\sum_{línea4} \text{valor}_{jug}(línea4) - \sum_{línea4} \text{valor}_{contr}(línea4)$$

donde

$$\text{valor}_A(línea) = \begin{cases} 0 & \text{si hay fichas de } B \\ 10^k & \text{si no hay fichas de } B \\ & \text{y hay } k \text{ fichas de } A \end{cases}$$

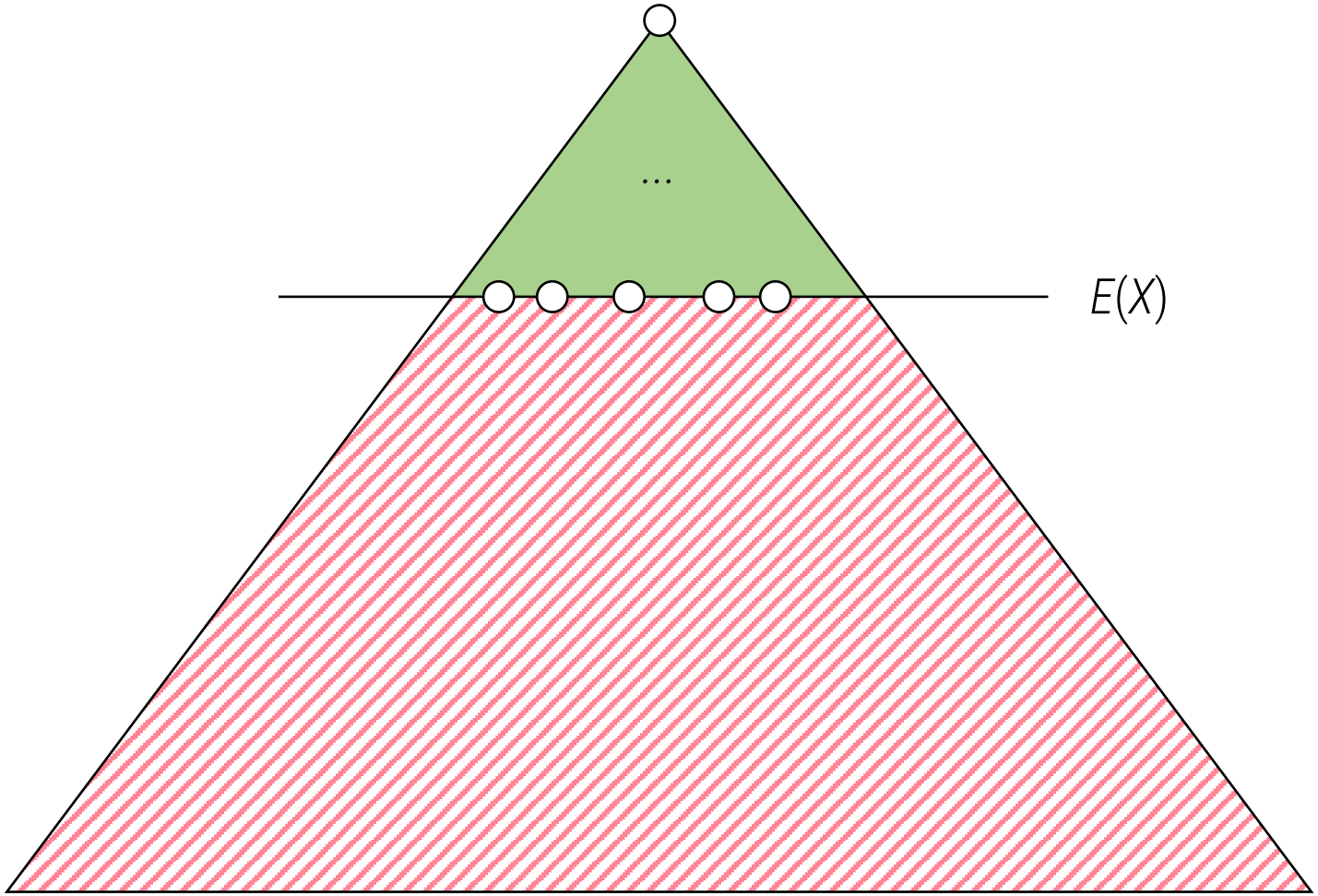
Othello



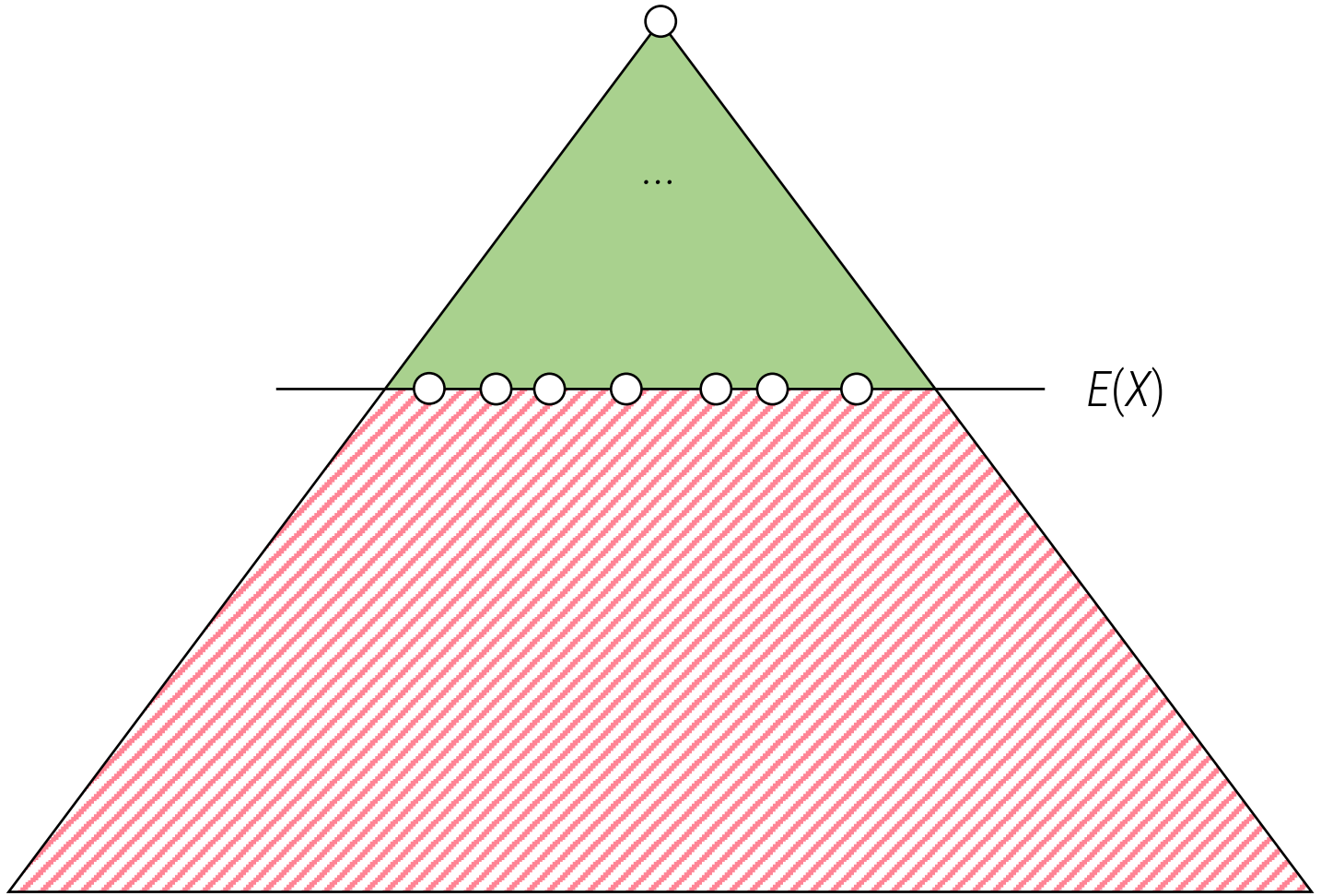
La heurística calcula la diferencia entre el número de fichas del jugador (ponderadas por una matriz de pesos) y el número de fichas del rival.

```
const int pesos[8][8] = {
      {50, -1,5,2,2,5, -1,50},
      {-1,-10,1,1,1,1,-10,-1},
      { 5,  1,1,1,1,1,  1, 5},
      { 2,  1,1,0,0,1,  1, 2},
      { 2,  1,1,0,0,1,  1, 2},
      { 5,  1,1,1,1,1,  1, 5},
      {-1,-10,1,1,1,1,-10,-1},
      {50, -1,5,2,2,5, -1,50}
};
```

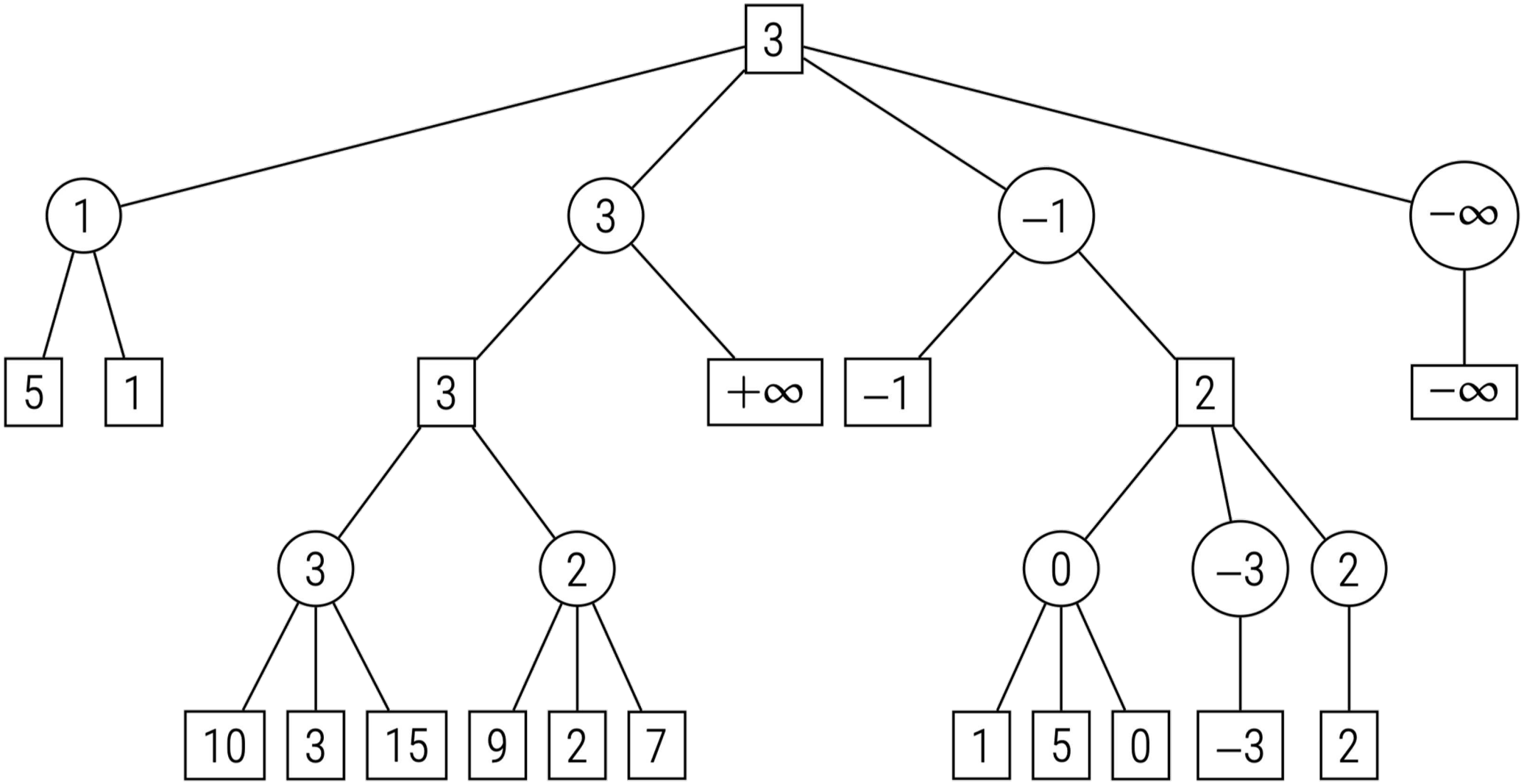
La calidad del juego



La calidad del juego



Podas α y β



Algoritmo de minimax con podas α y β

```
jugada Jugador::juega(Juego const& EJ) {
    jugada mejorC;
    int mejorV = -∞;
    for (jugada c : EJ.jugadasDesde()) {
        int v = valoraMin(EJ.aplica(c), EJ.turno(), nivel-1, mejorV, +∞);
        if (v > mejorV) {
            mejorV = v;
            mejorC = c;
        }
    }
    return mejorC;
}
```

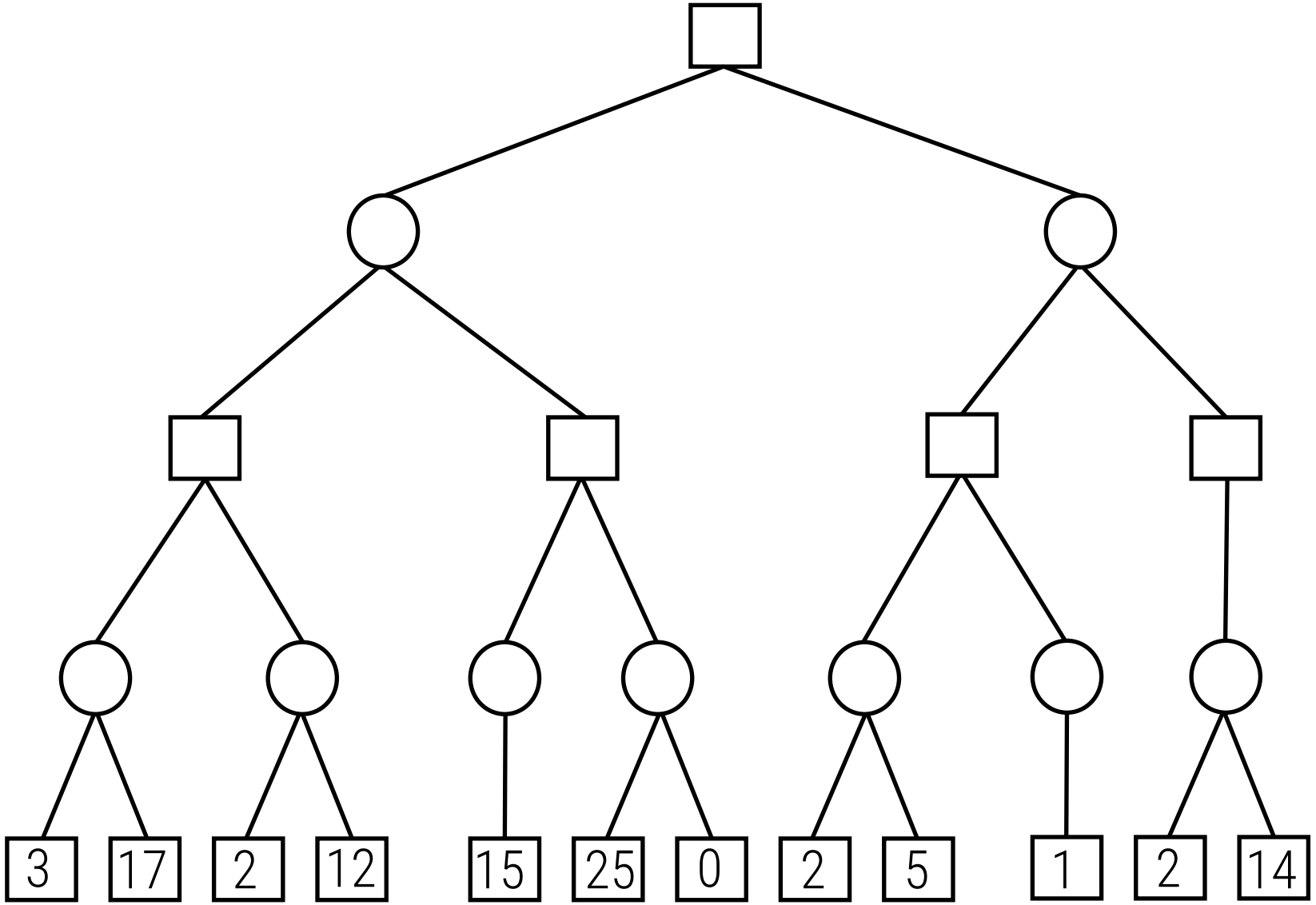
Algoritmo de minimax con podas α y β

```
int Jugador::valoraMin(Juego const& EJ, Turno t, int n, int α, int β) {
  if (EJ.terminal() || n == 0)
     return Heuristica(EJ, t)
  else {
     for (jugada c : EJ.jugadasDesde()) {
        β = min(valoraMax(EJ.aplica(c), t, n - 1, α, β), β);
        if (α >= β) break;
     }
     return β;
  }
}
```

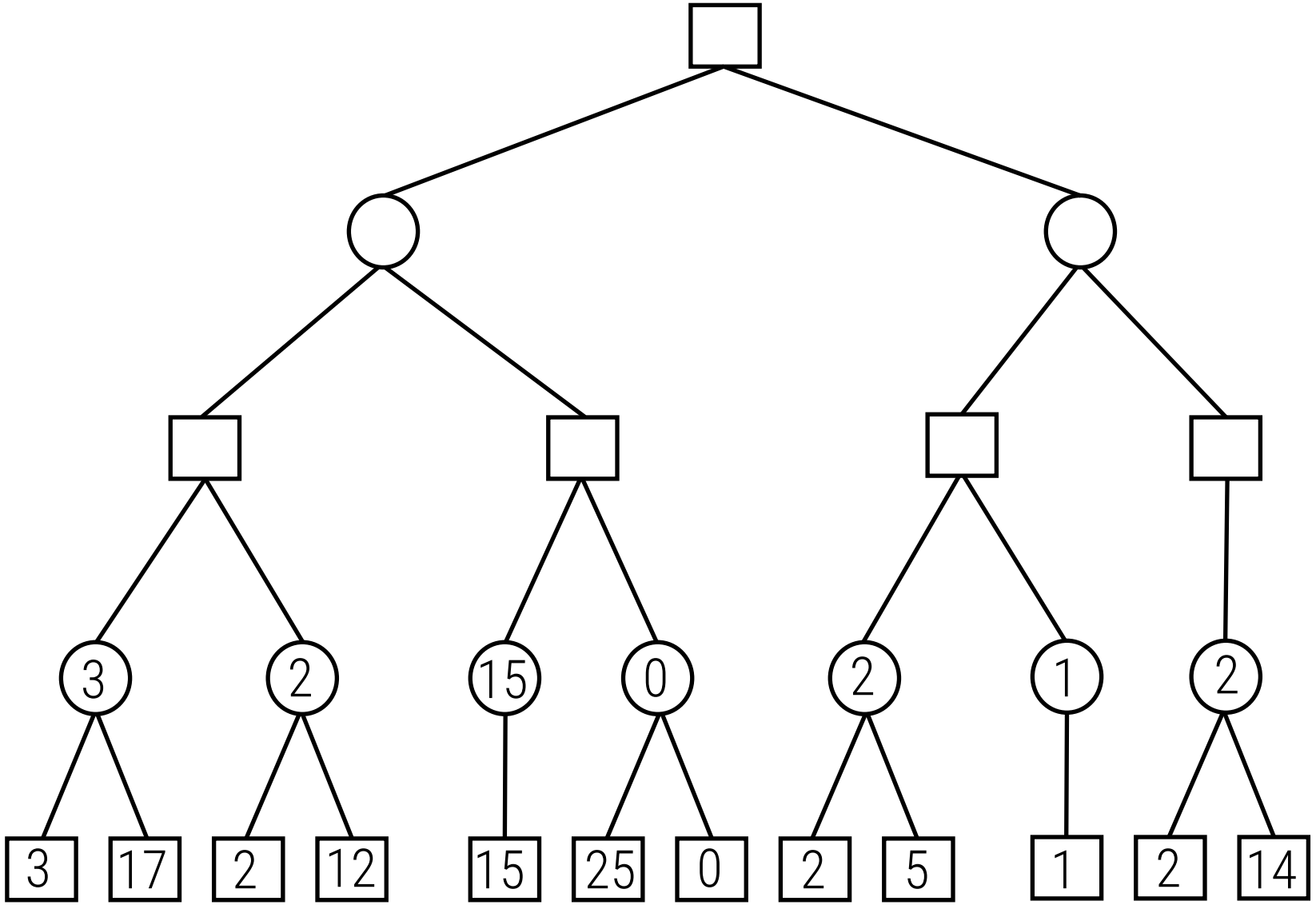
Algoritmo de minimax con podas α y β

```
int Jugador::valoraMax(Juego const& EJ, Turno t, int n, int α, int β) {
  if (EJ.terminal() || n == 0)
     return Heuristica(EJ, t)
  else {
     for (jugada c : EJ.jugadasDesde()) {
        α = max(valoraMin(EJ.aplica(c), t, n - 1, α, β), α);
        if (α >= β) break;
     }
     return α;
  }
}
```

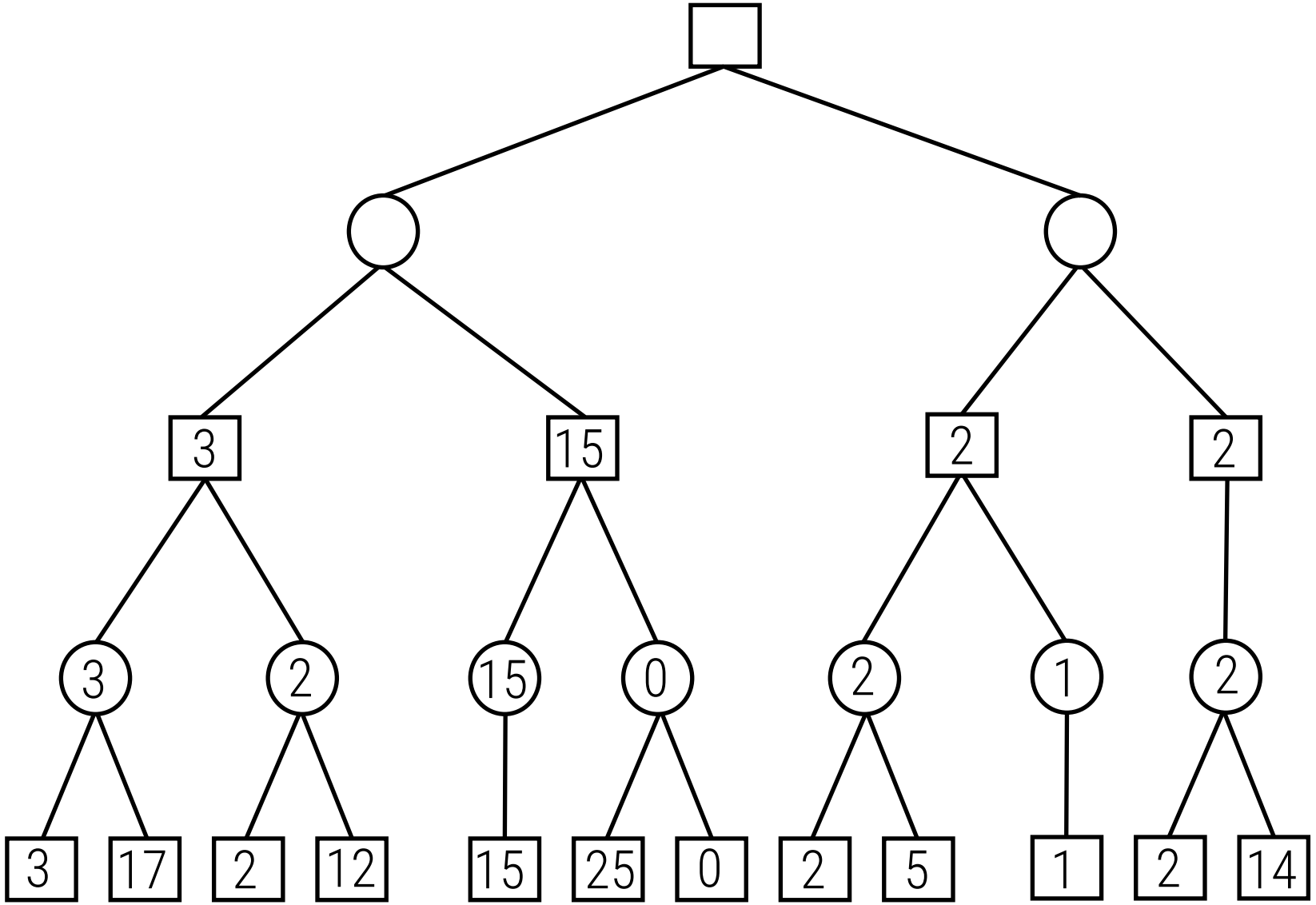
Ejemplo: Algoritmo de minimax



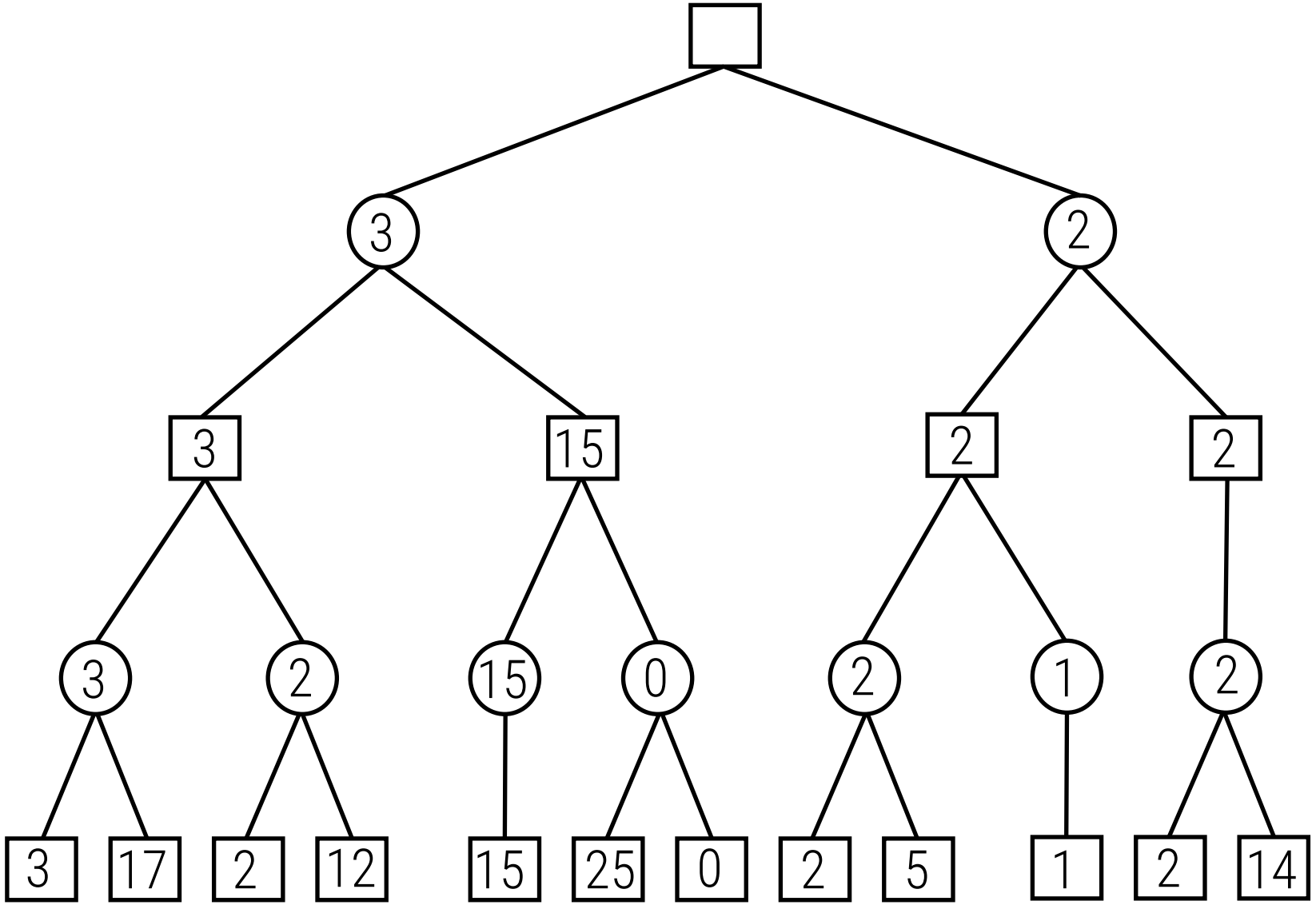
Ejemplo: Algoritmo de minimax



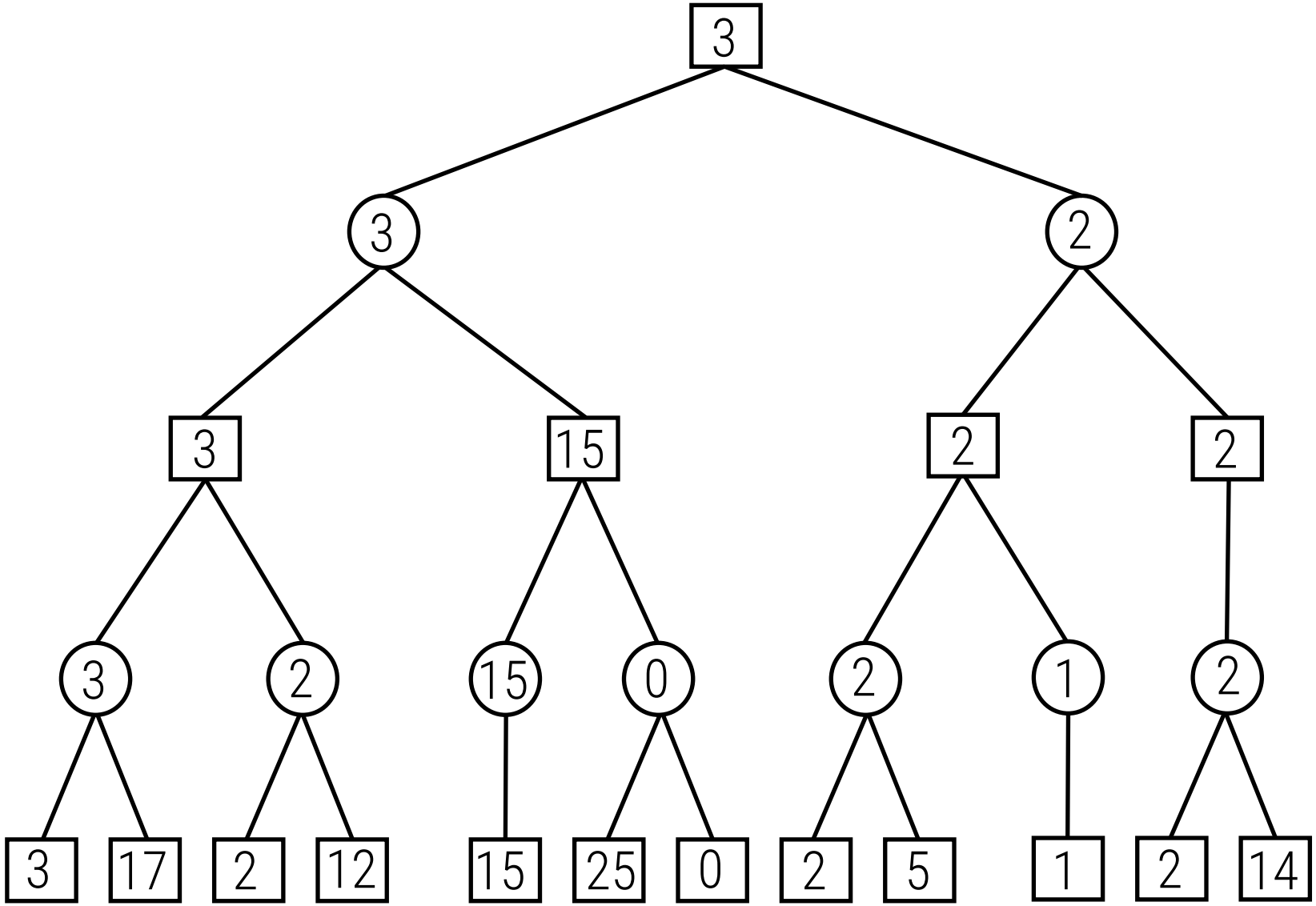
Ejemplo: Algoritmo de minimax



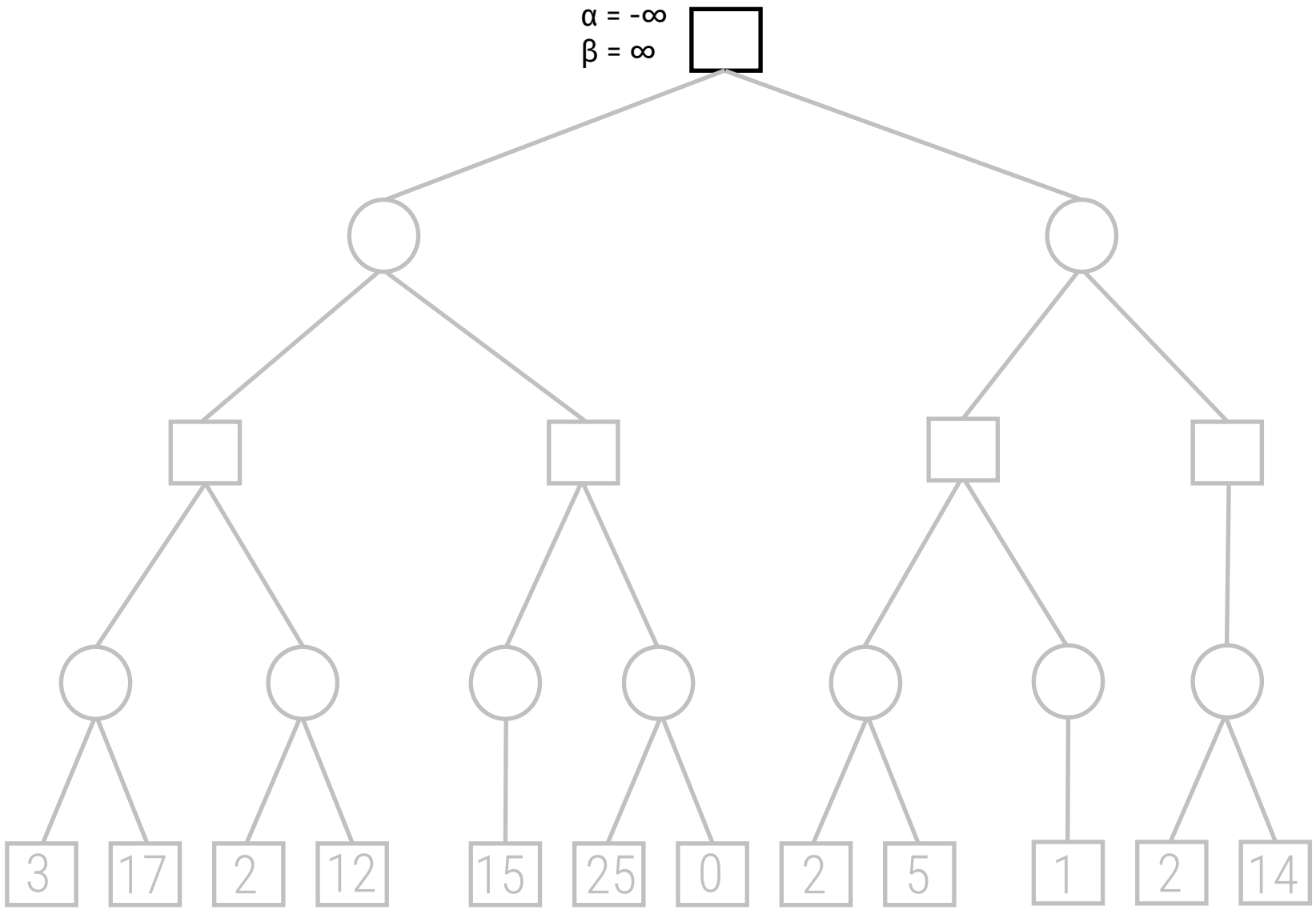
Ejemplo: Algoritmo de minimax



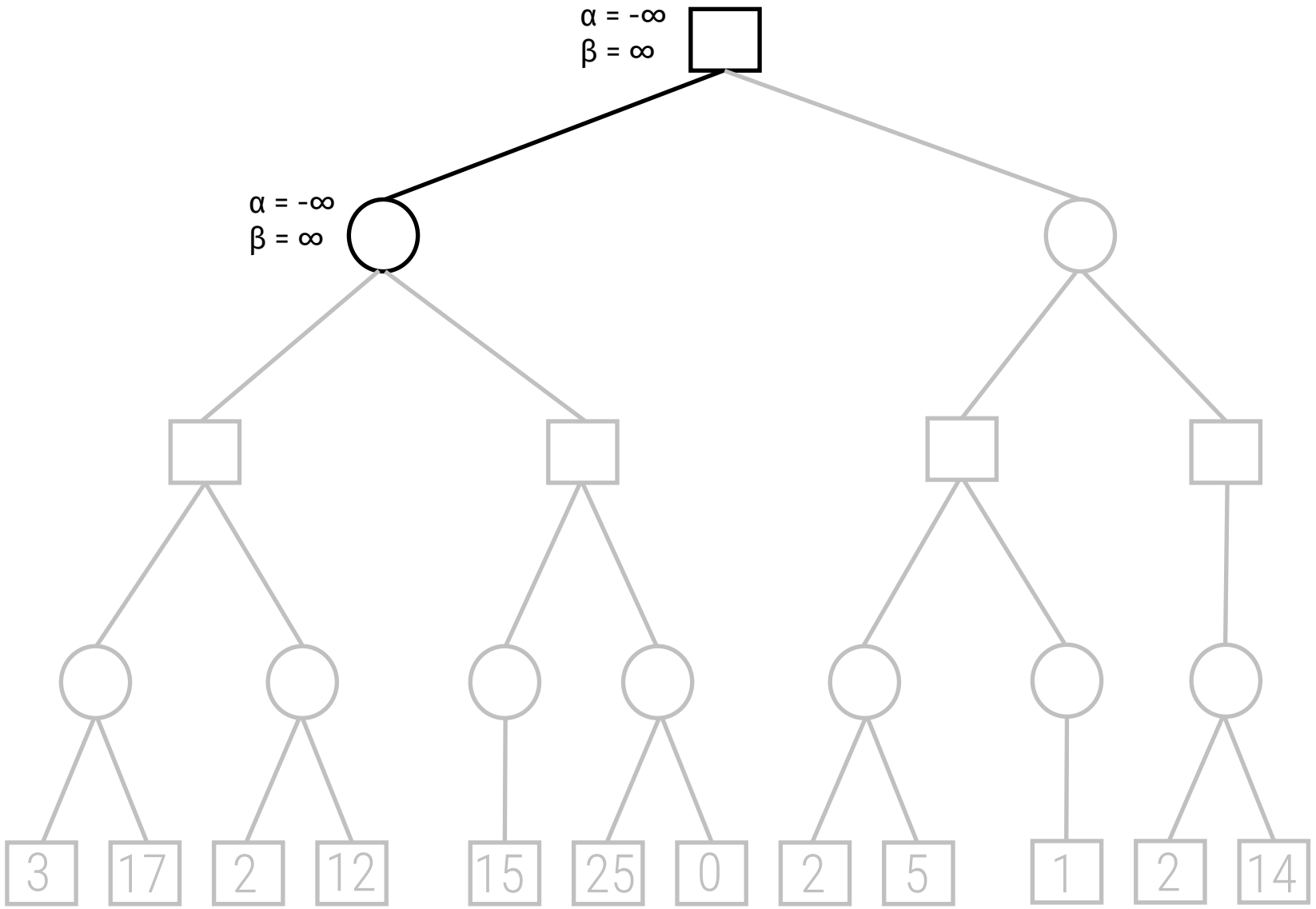
Ejemplo: Algoritmo de minimax



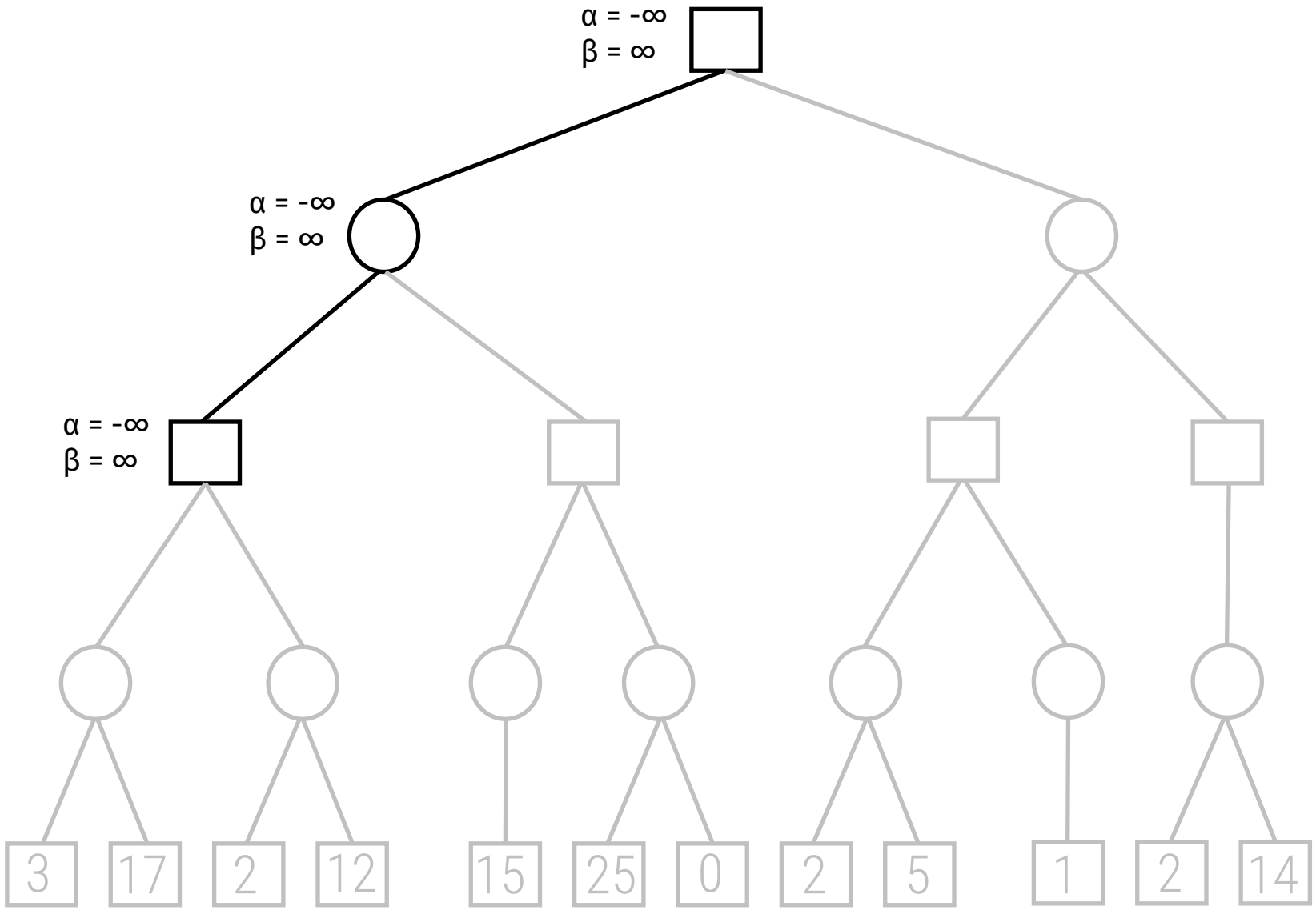
Ejemplo: Algoritmo de minimax con podas α y β



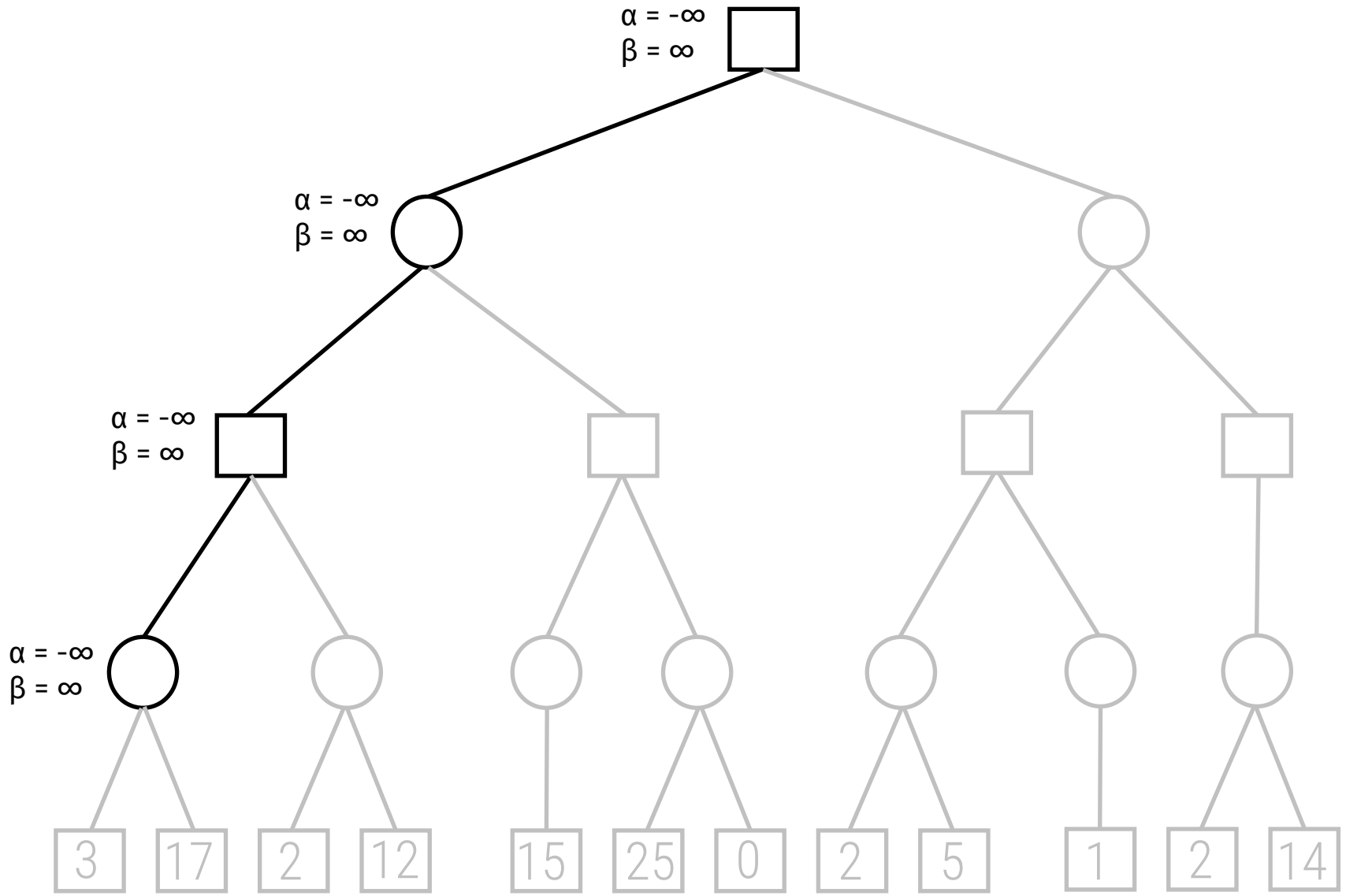
Ejemplo: Algoritmo de minimax con podas α y β



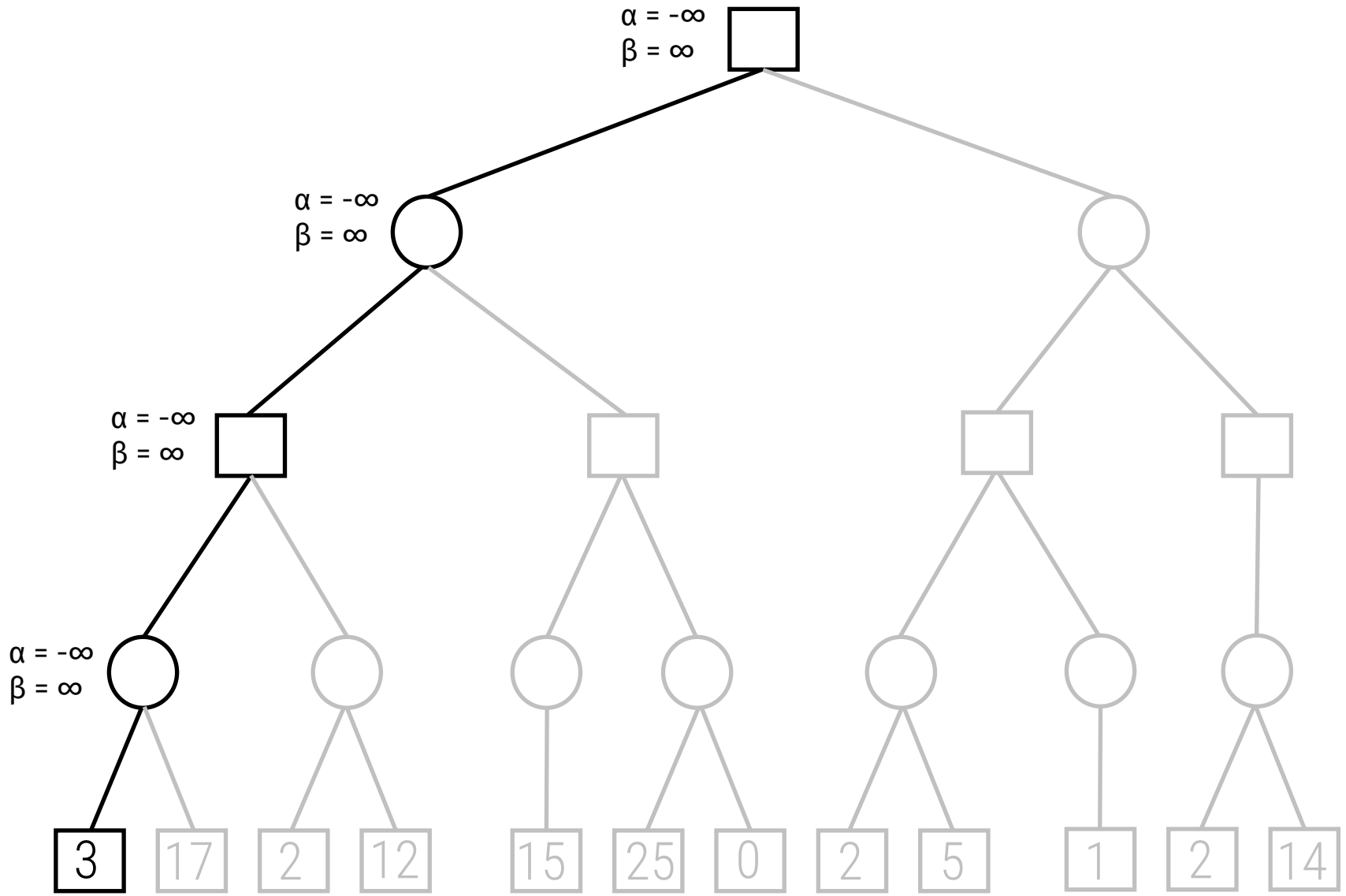
Ejemplo: Algoritmo de minimax con podas α y β



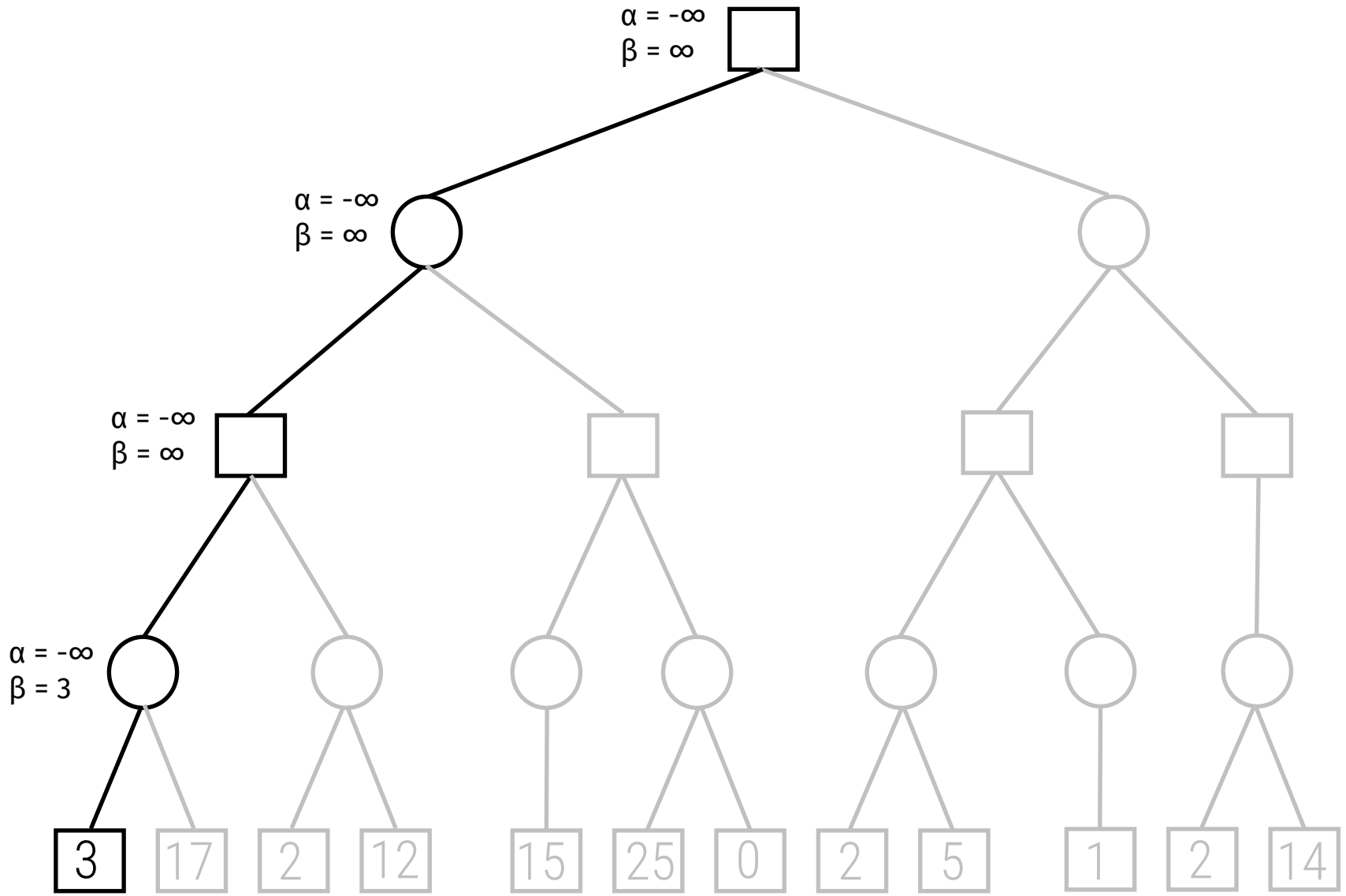
Ejemplo: Algoritmo de minimax con podas α y β



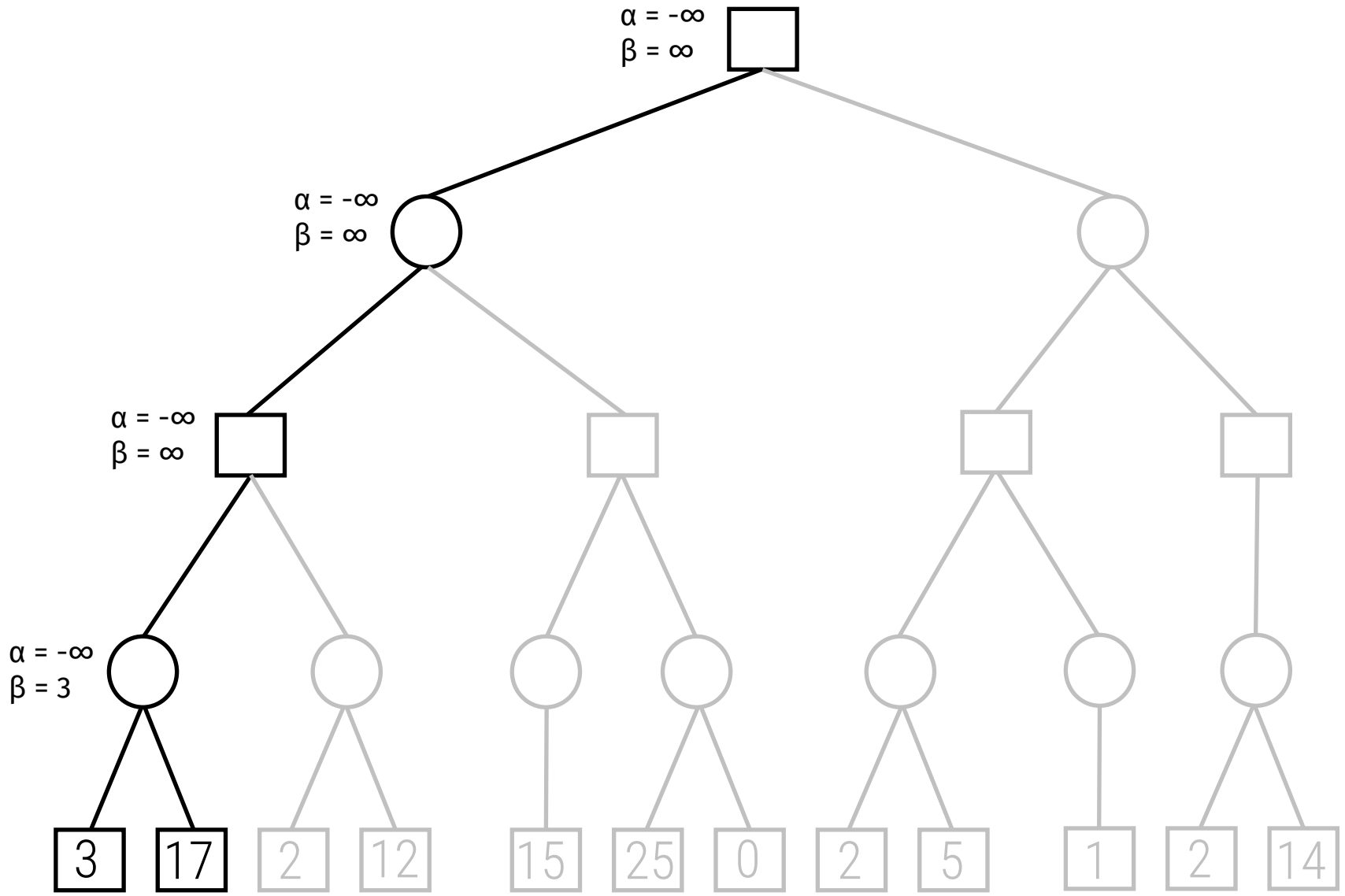
Ejemplo: Algoritmo de minimax con podas α y β



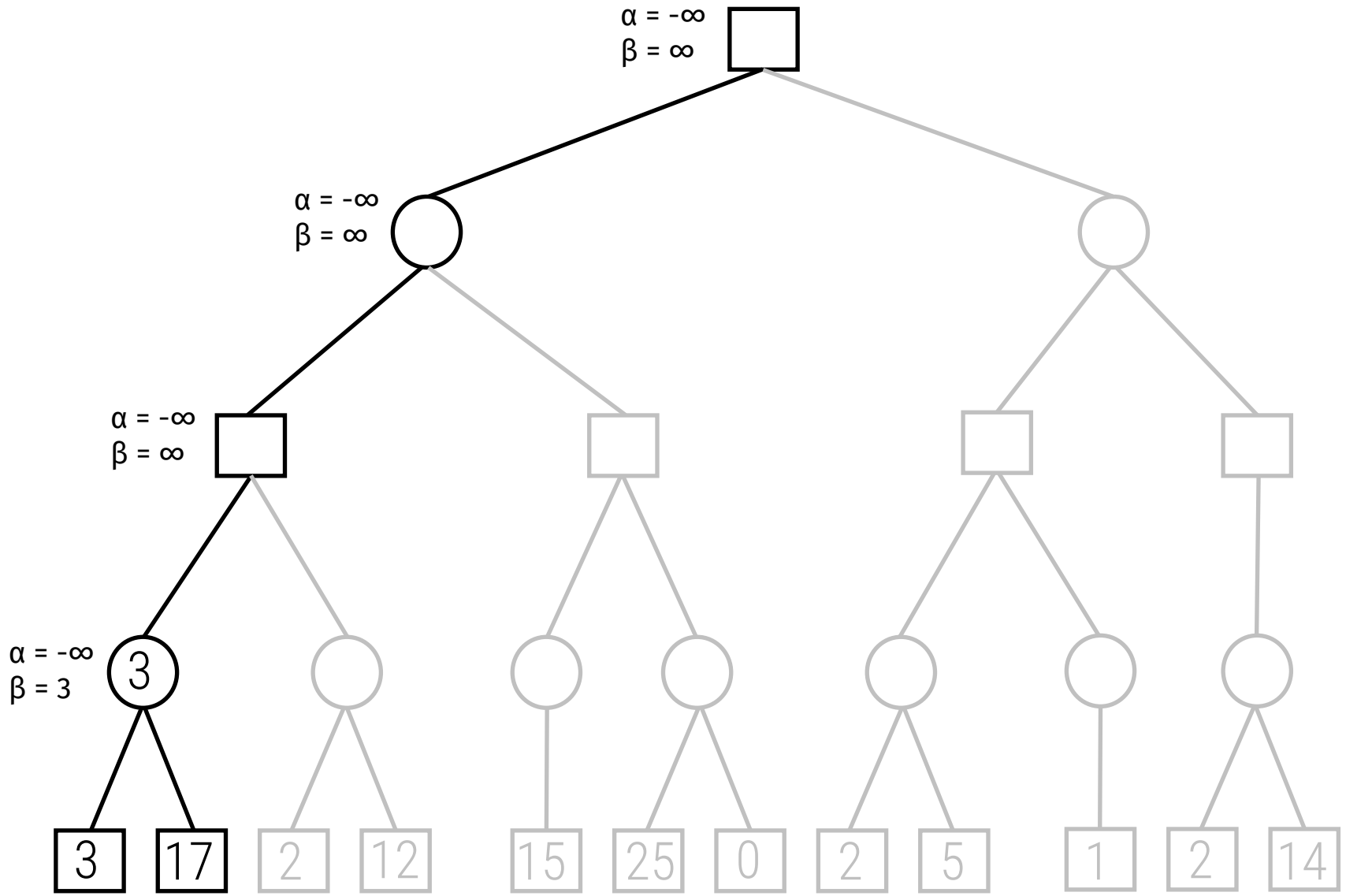
Ejemplo: Algoritmo de minimax con podas α y β



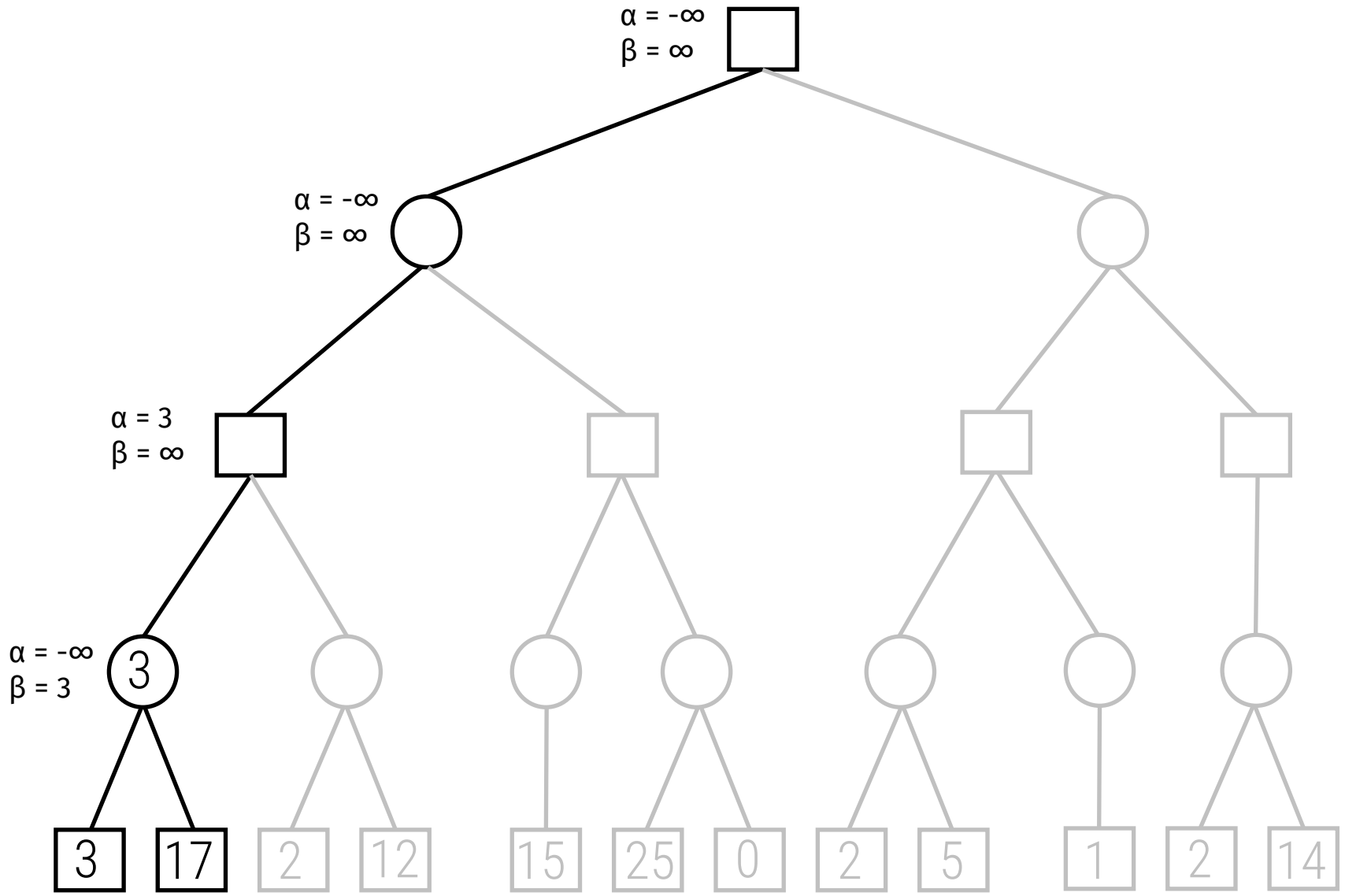
Ejemplo: Algoritmo de minimax con podas α y β



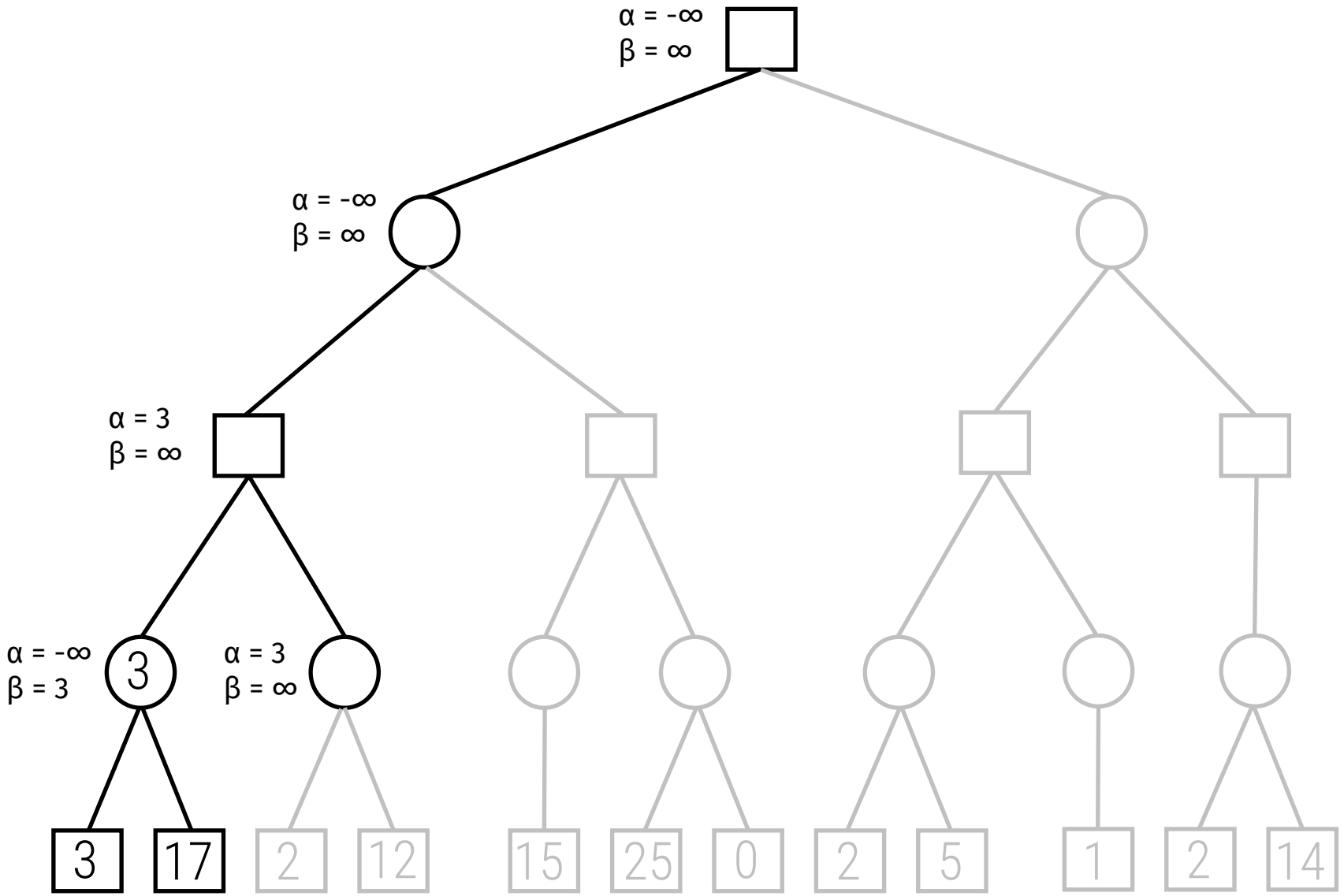
Ejemplo: Algoritmo de minimax con podas α y β



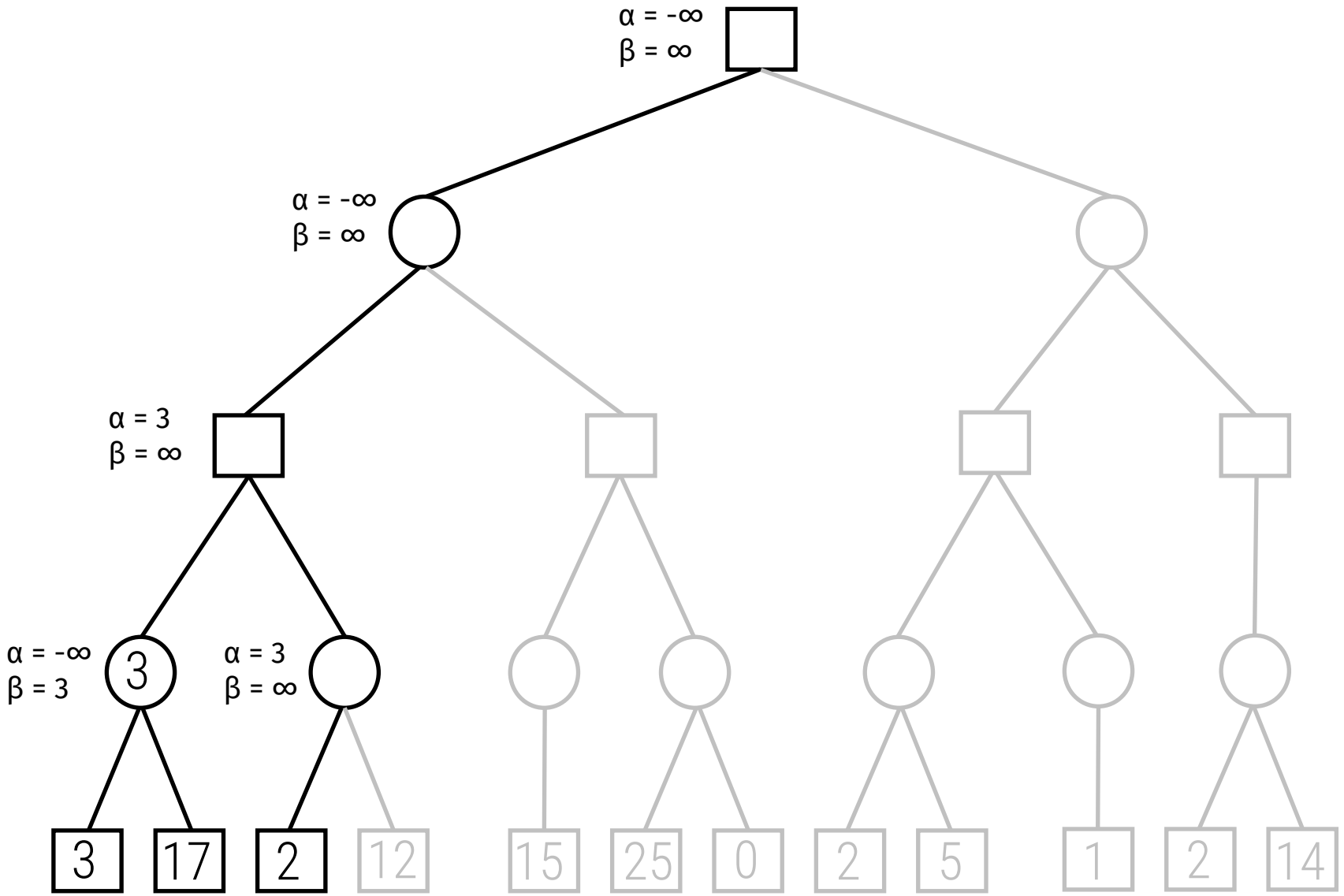
Ejemplo: Algoritmo de minimax con podas α y β



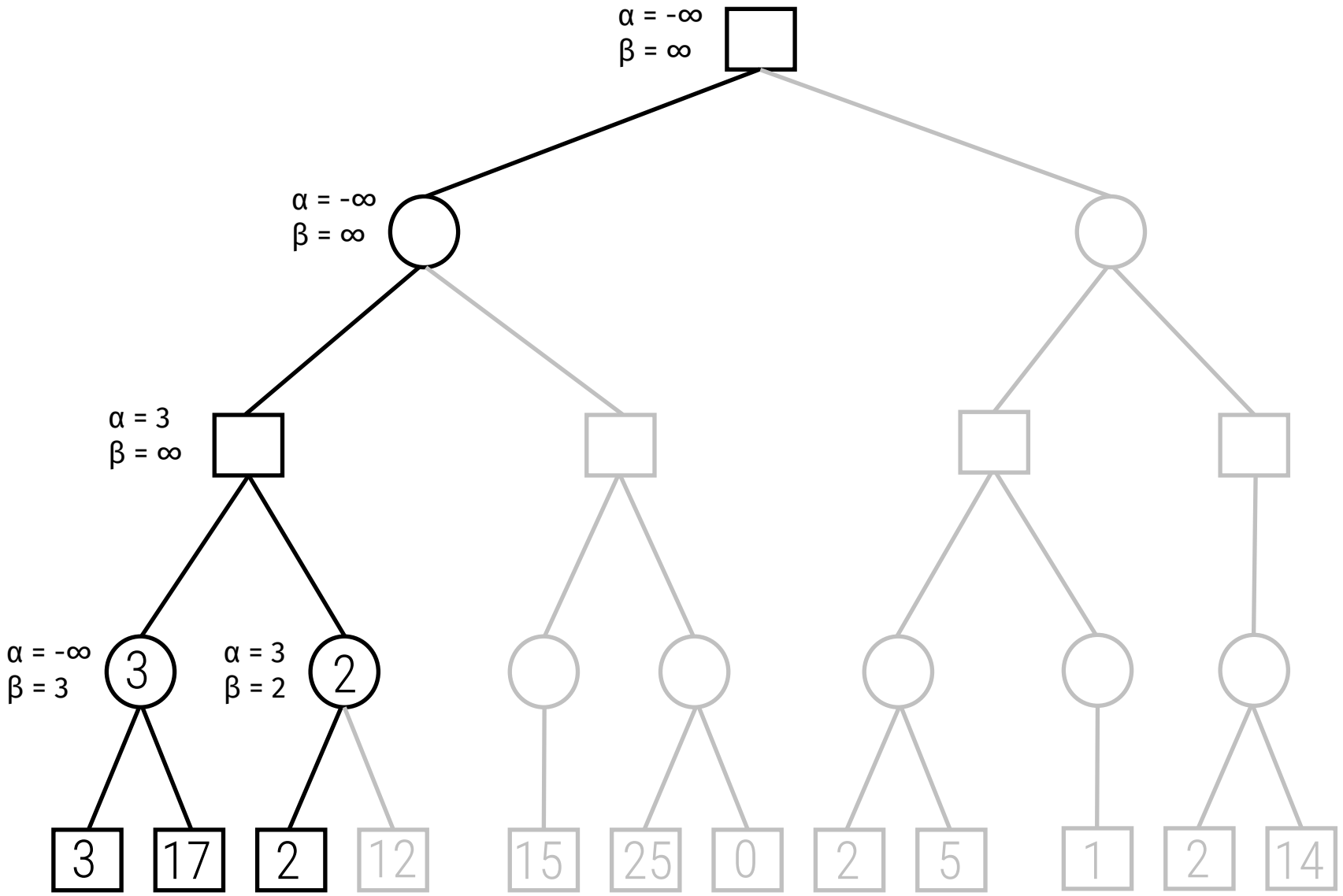
Ejemplo: Algoritmo de minimax con podas α y β



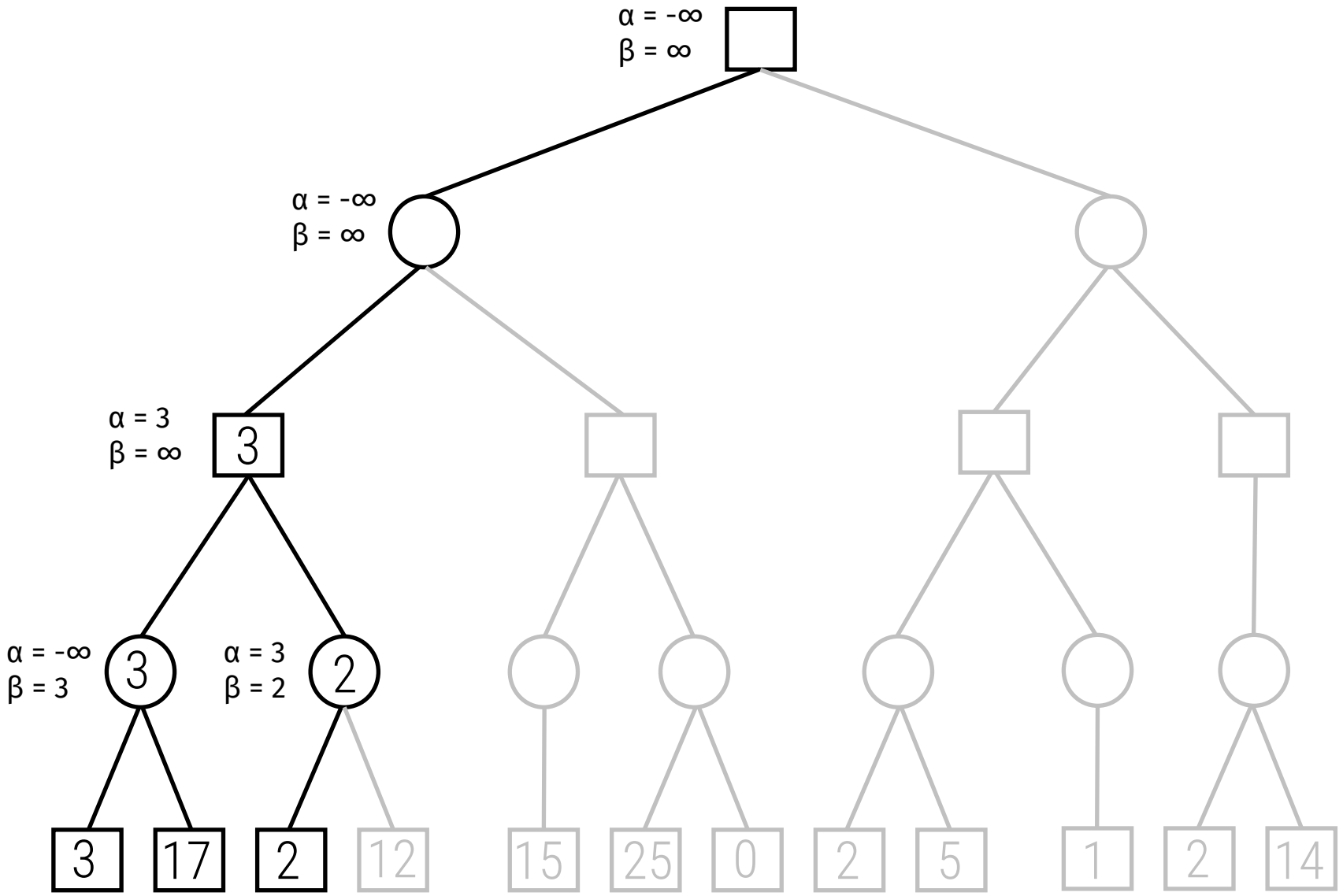
Ejemplo: Algoritmo de minimax con podas α y β



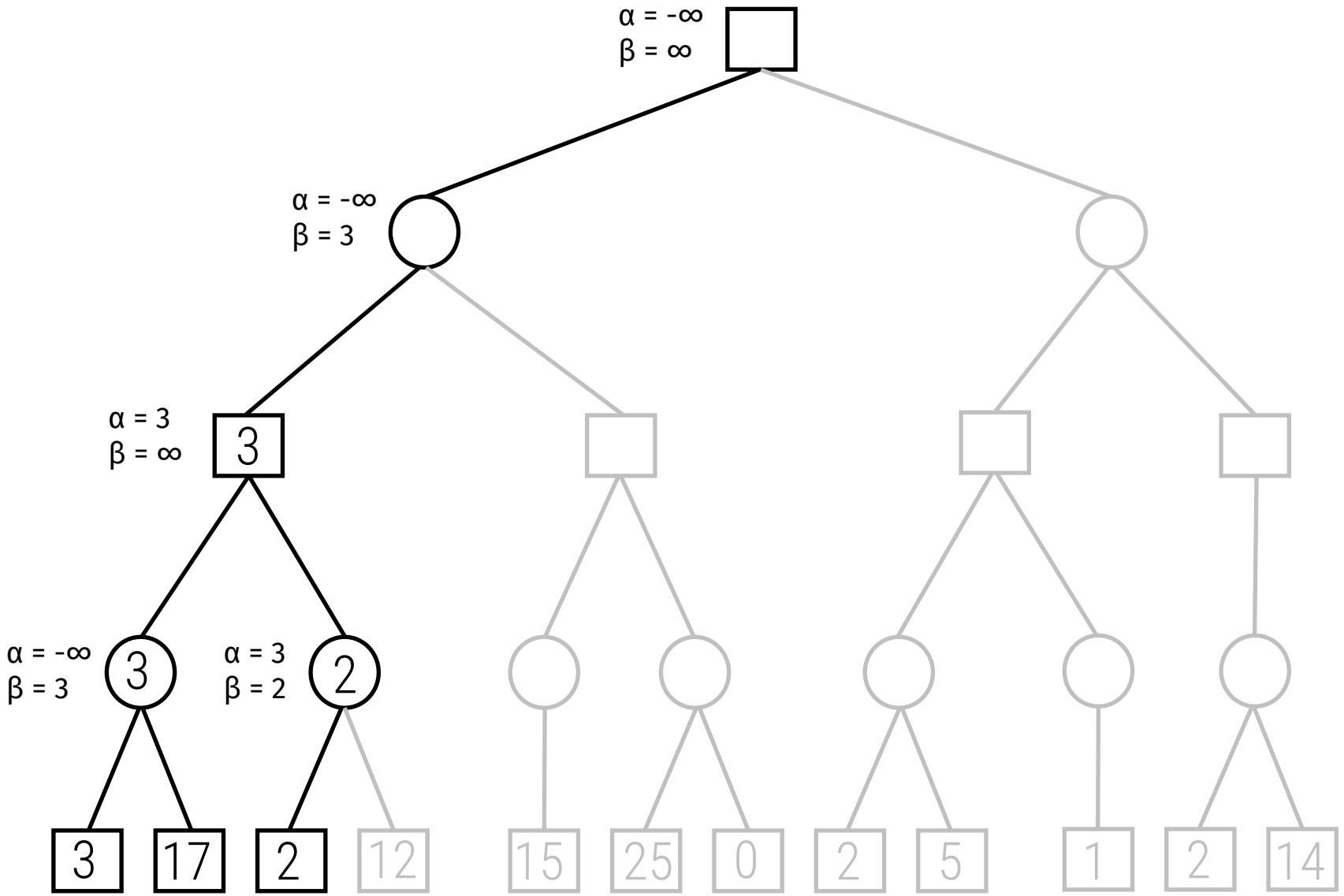
Ejemplo: Algoritmo de minimax con podas α y β



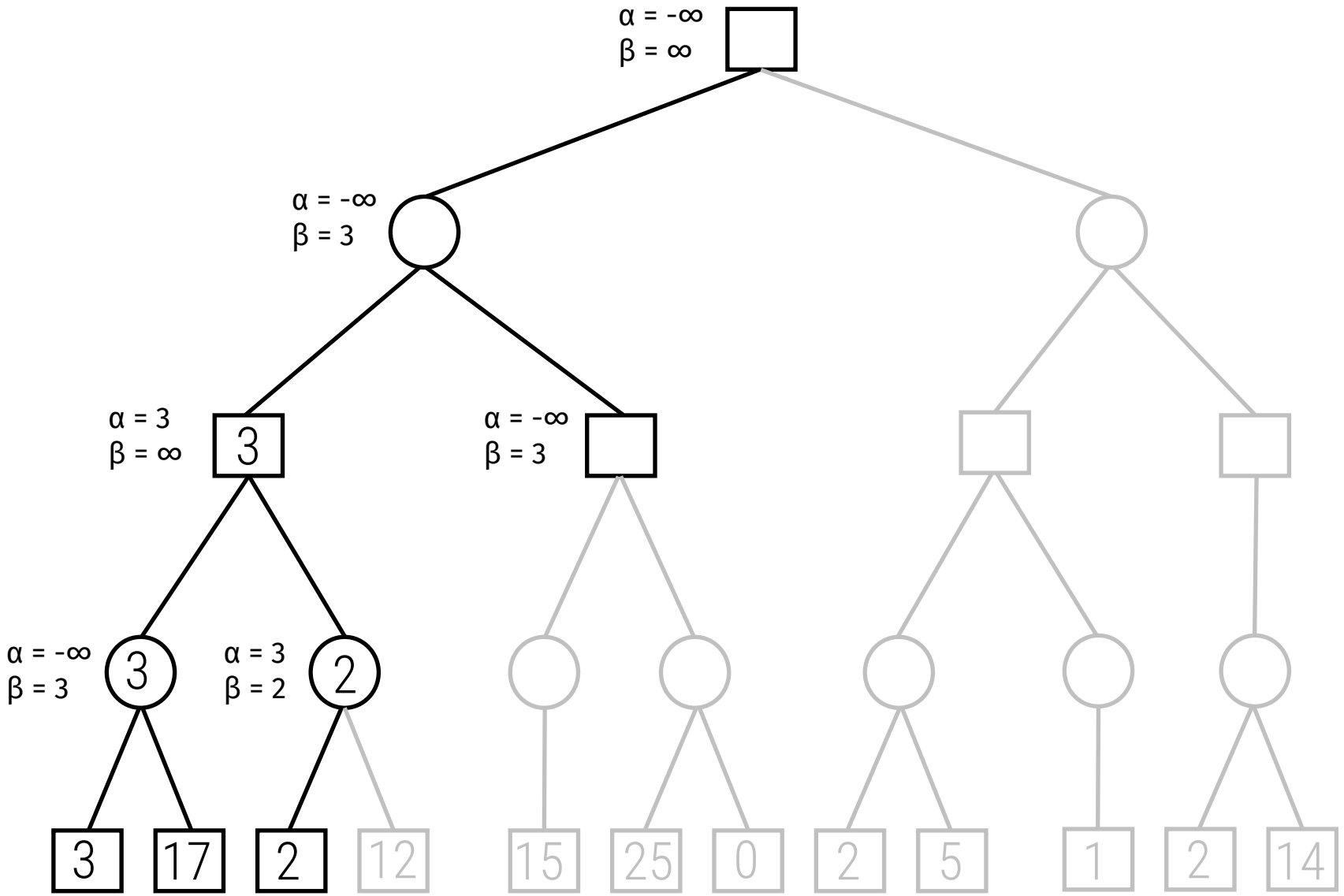
Ejemplo: Algoritmo de minimax con podas α y β



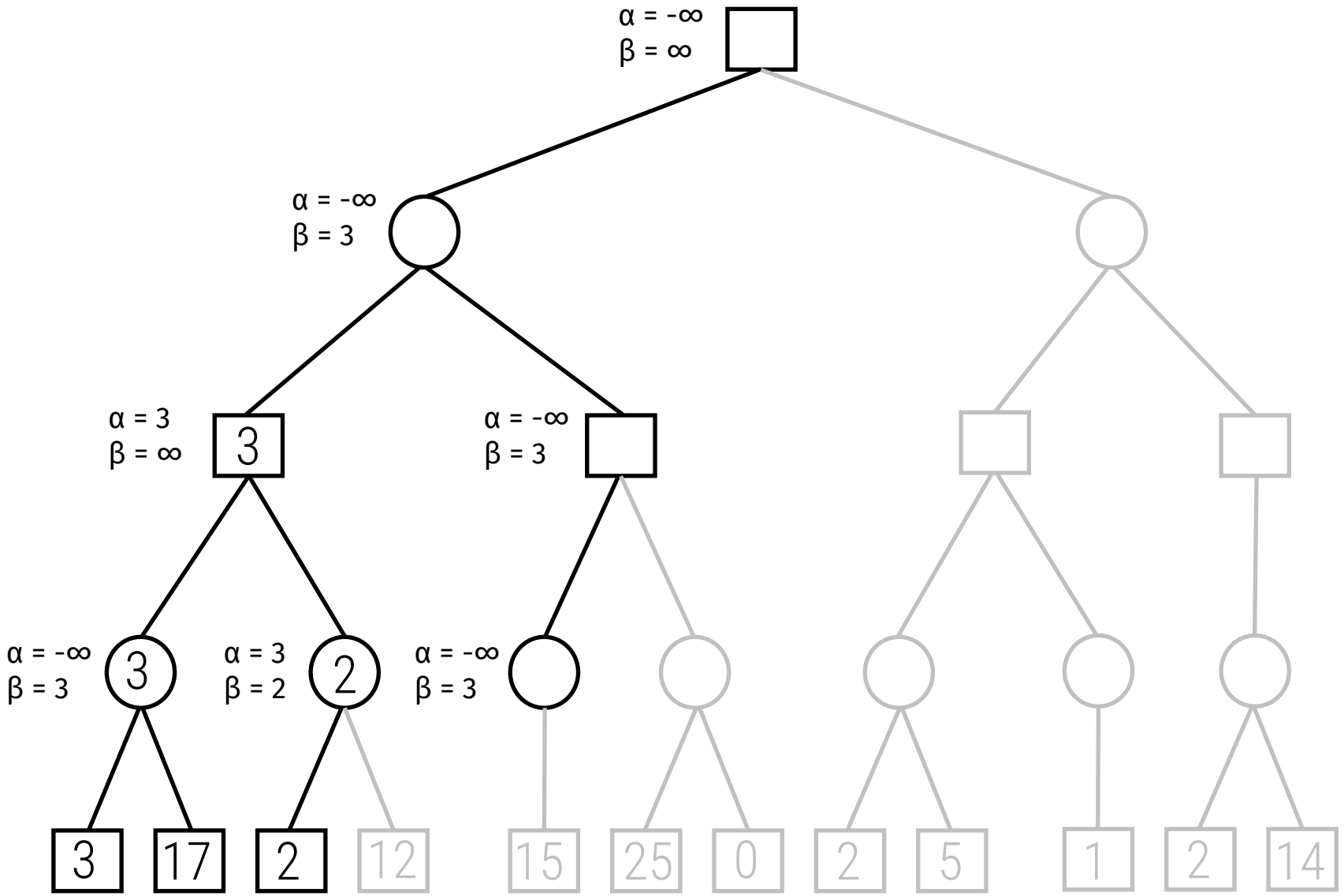
Ejemplo: Algoritmo de minimax con podas α y β



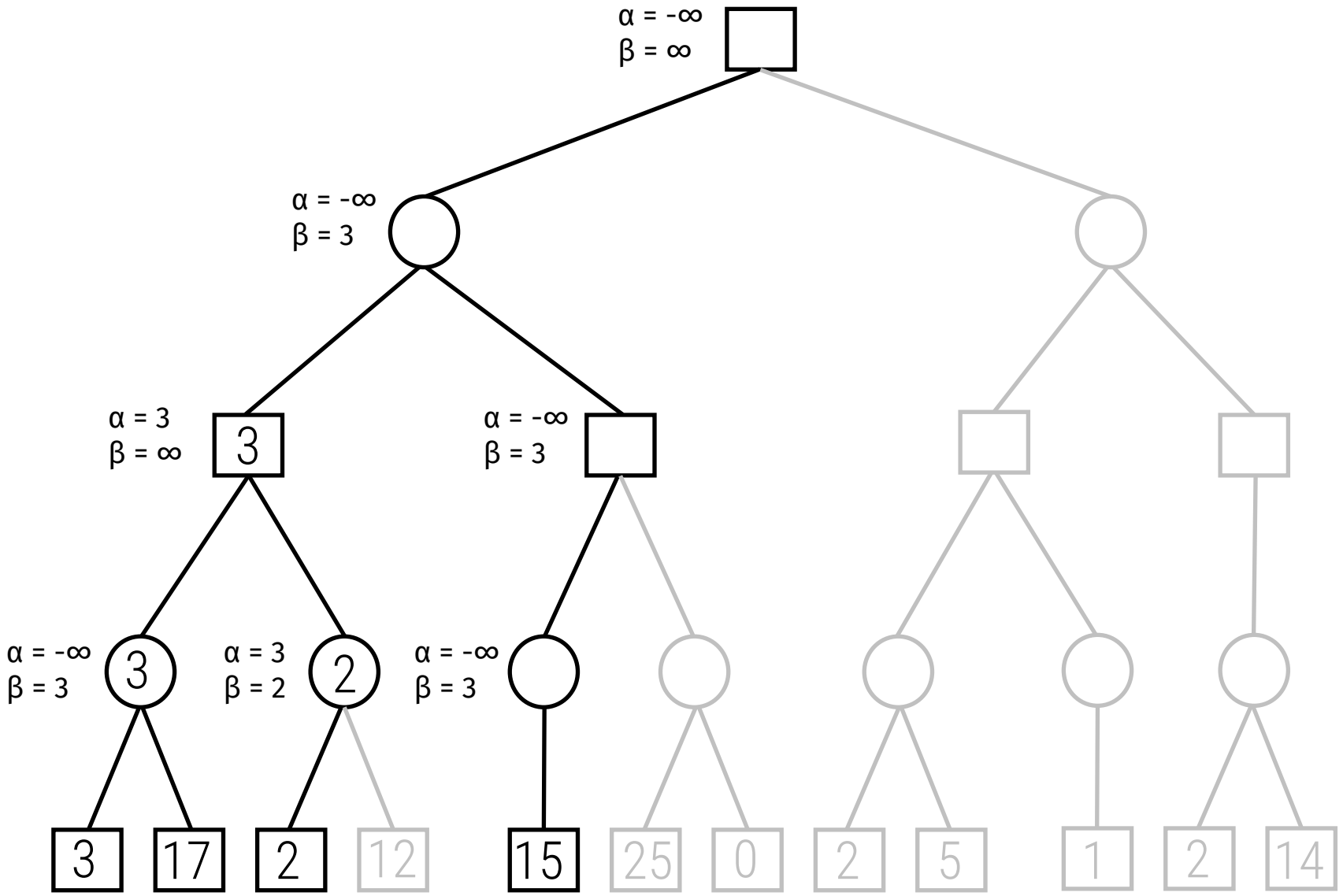
Ejemplo: Algoritmo de minimax con podas α y β



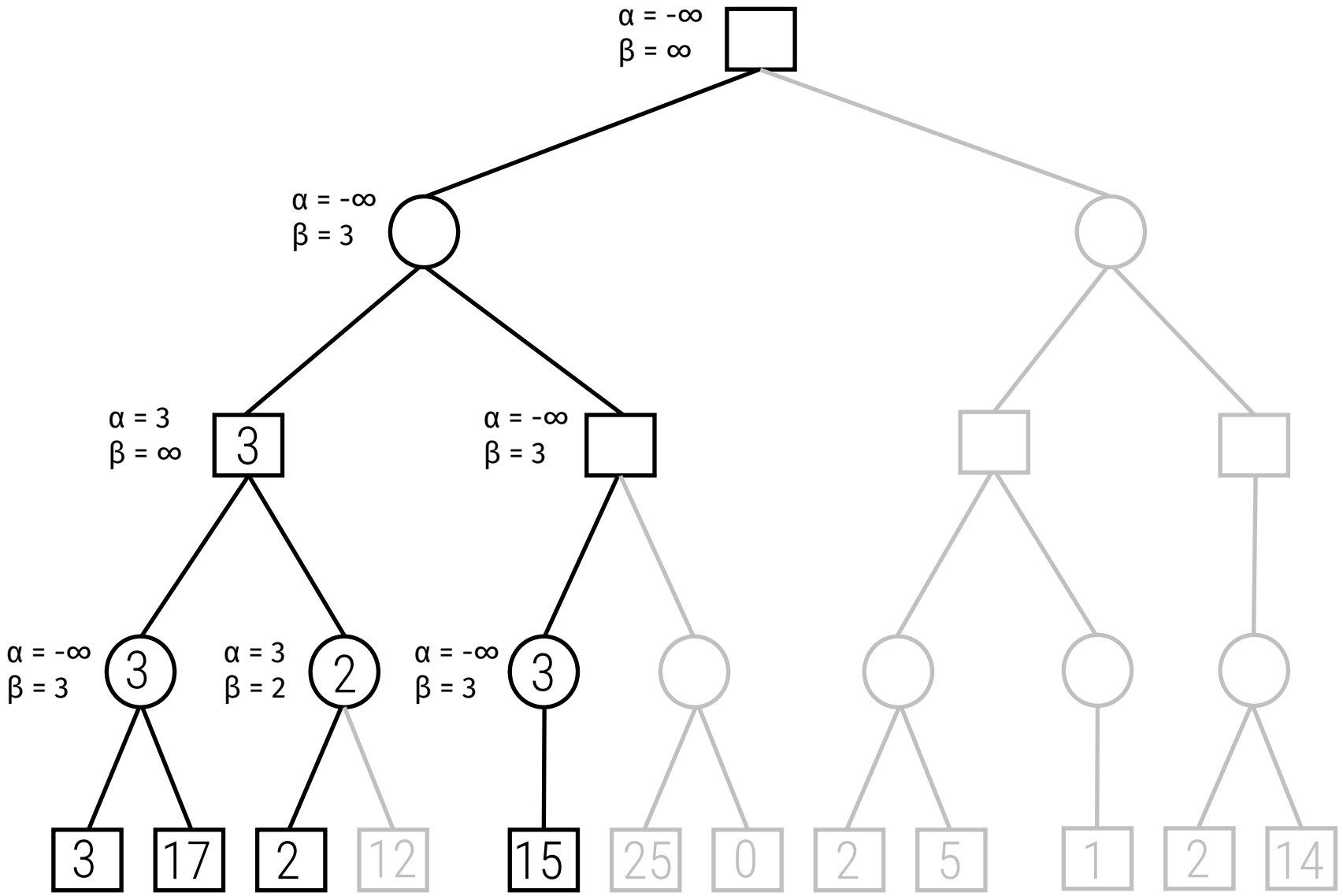
Ejemplo: Algoritmo de minimax con podas α y β



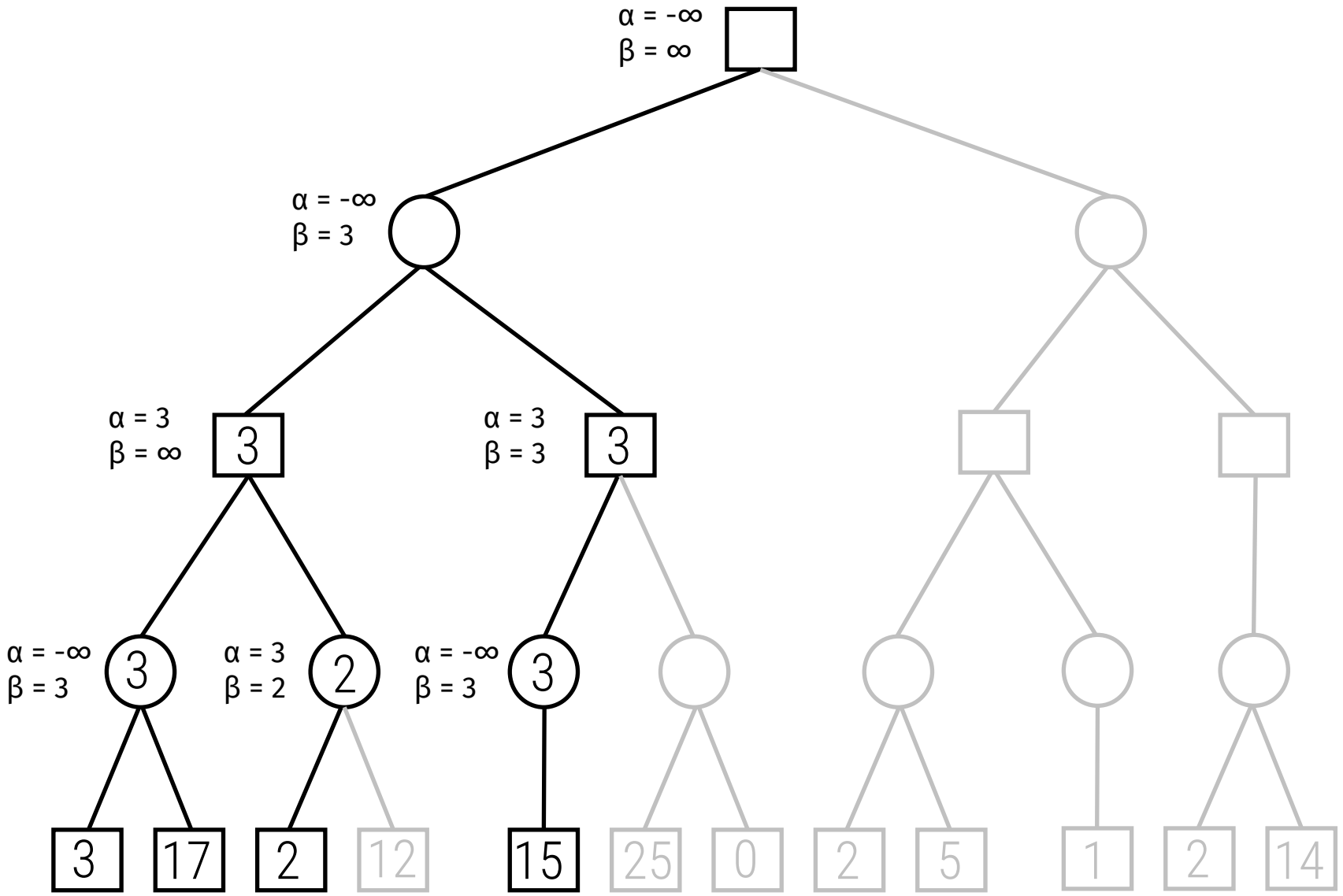
Ejemplo: Algoritmo de minimax con podas α y β



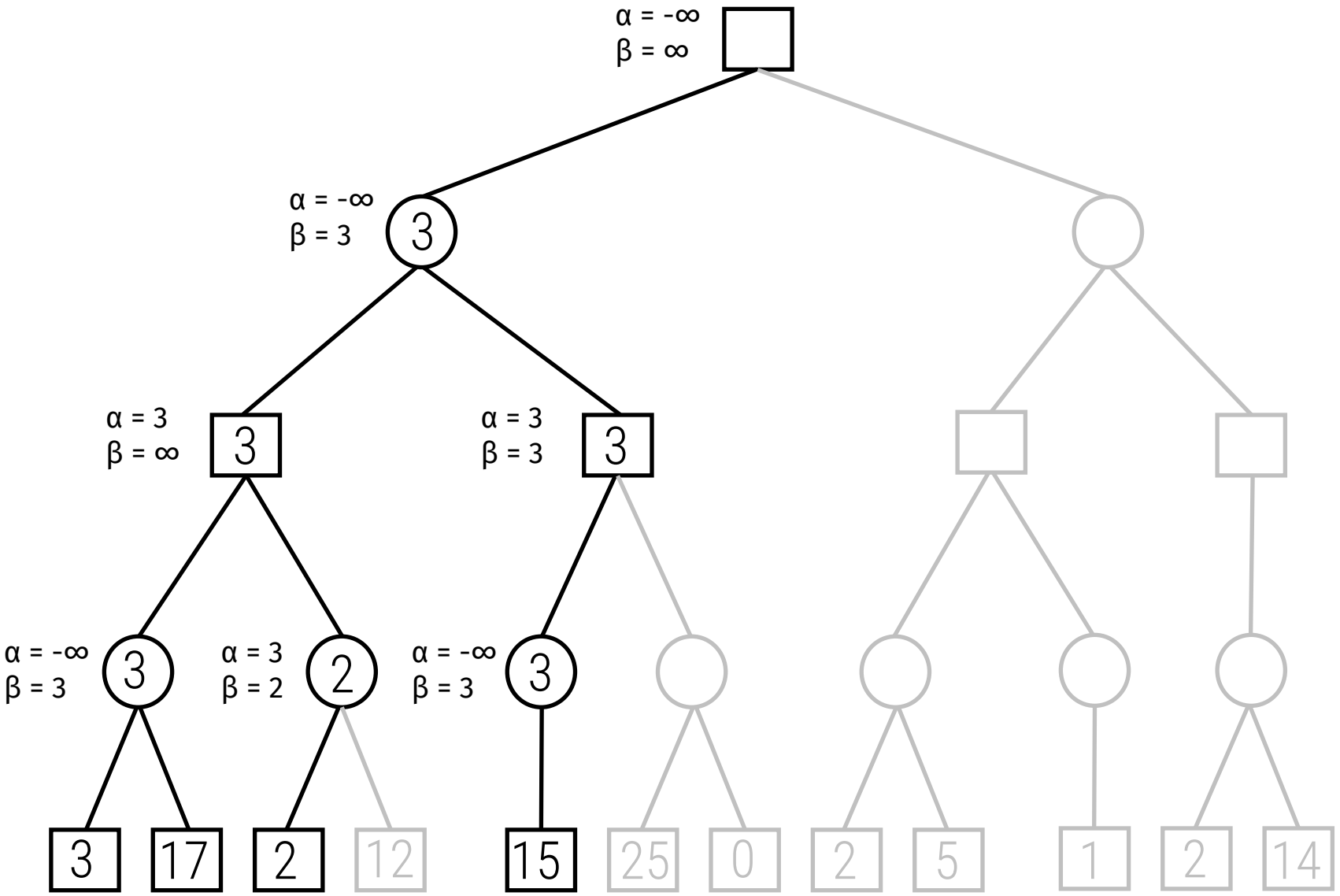
Ejemplo: Algoritmo de minimax con podas α y β



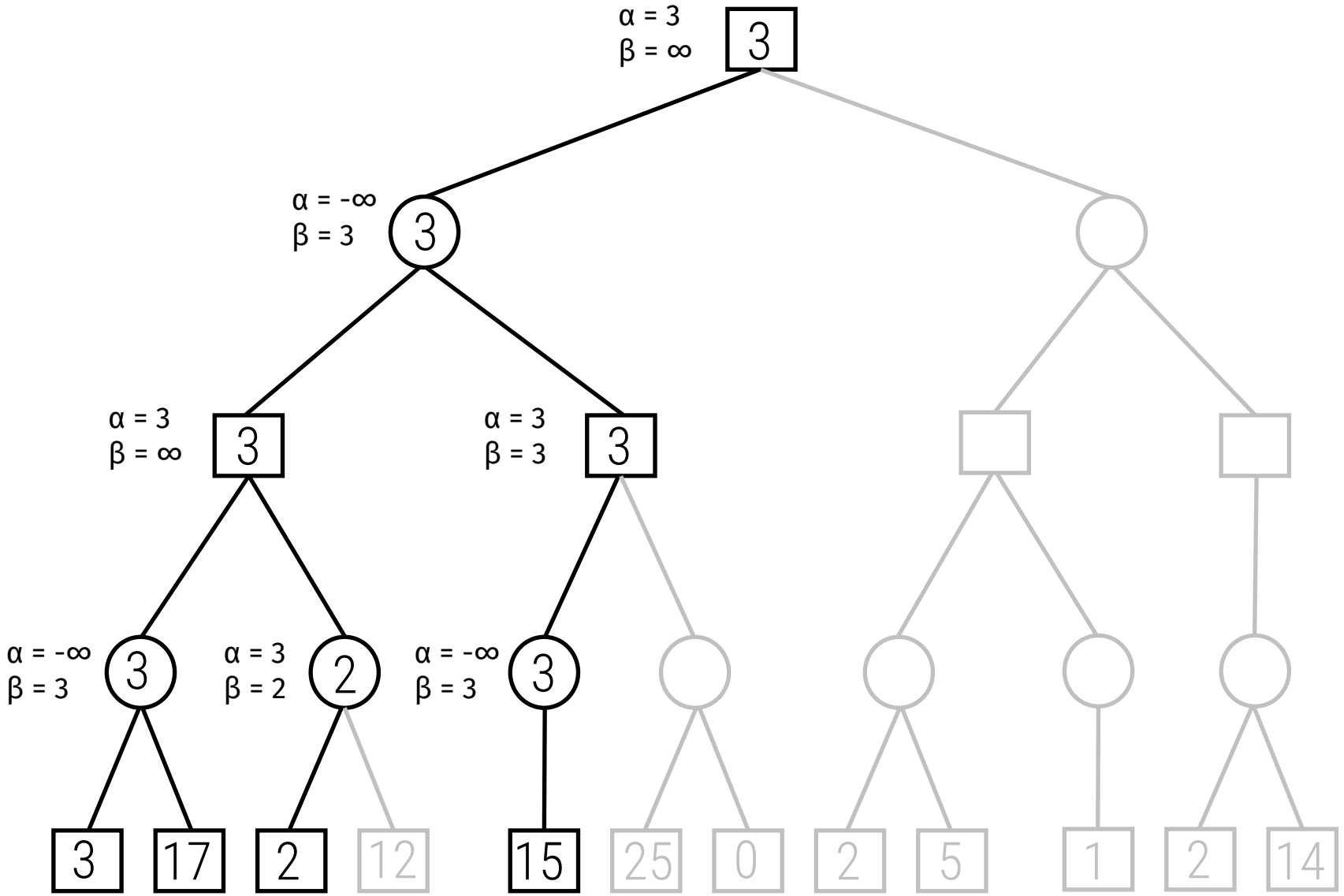
Ejemplo: Algoritmo de minimax con podas α y β



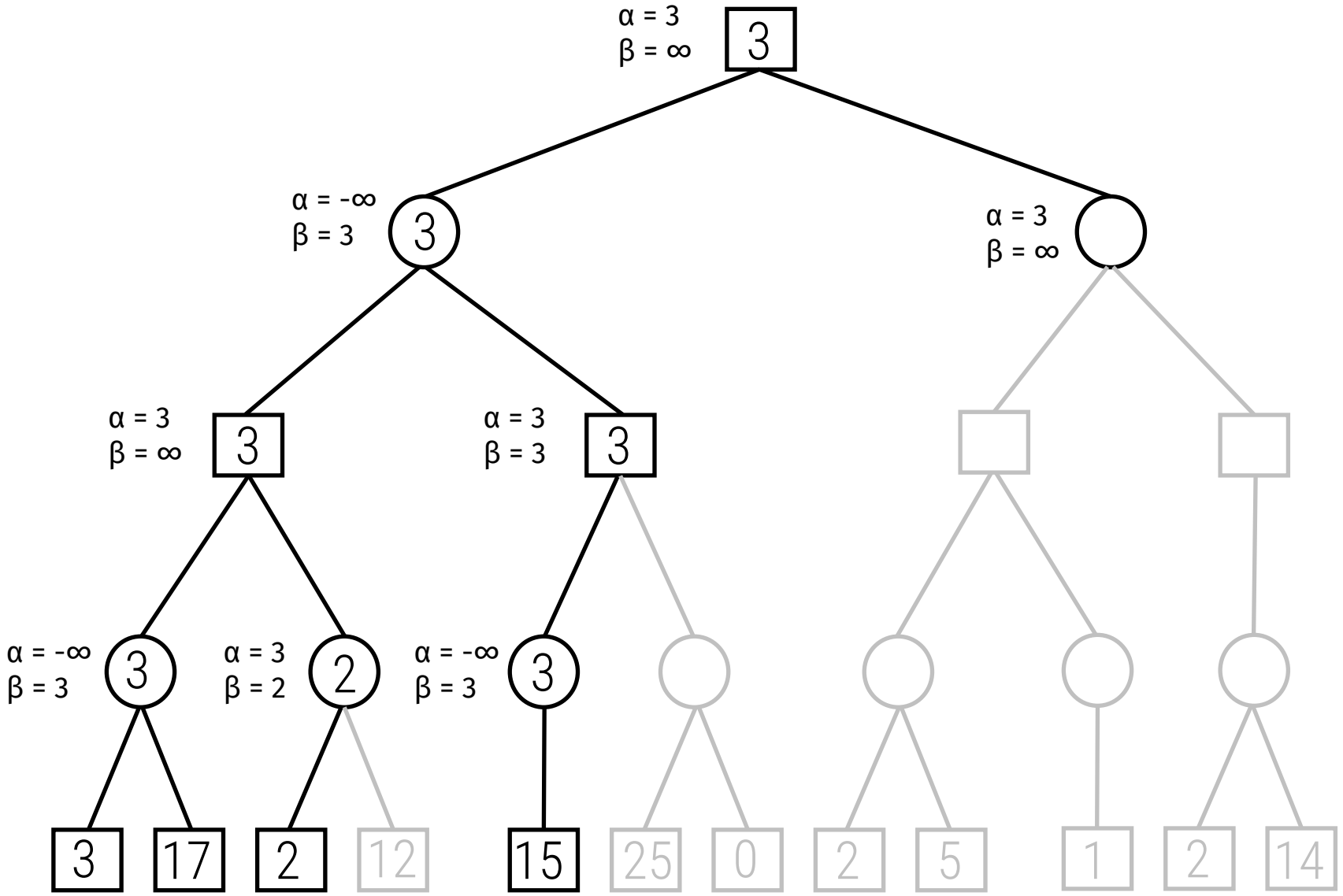
Ejemplo: Algoritmo de minimax con podas α y β



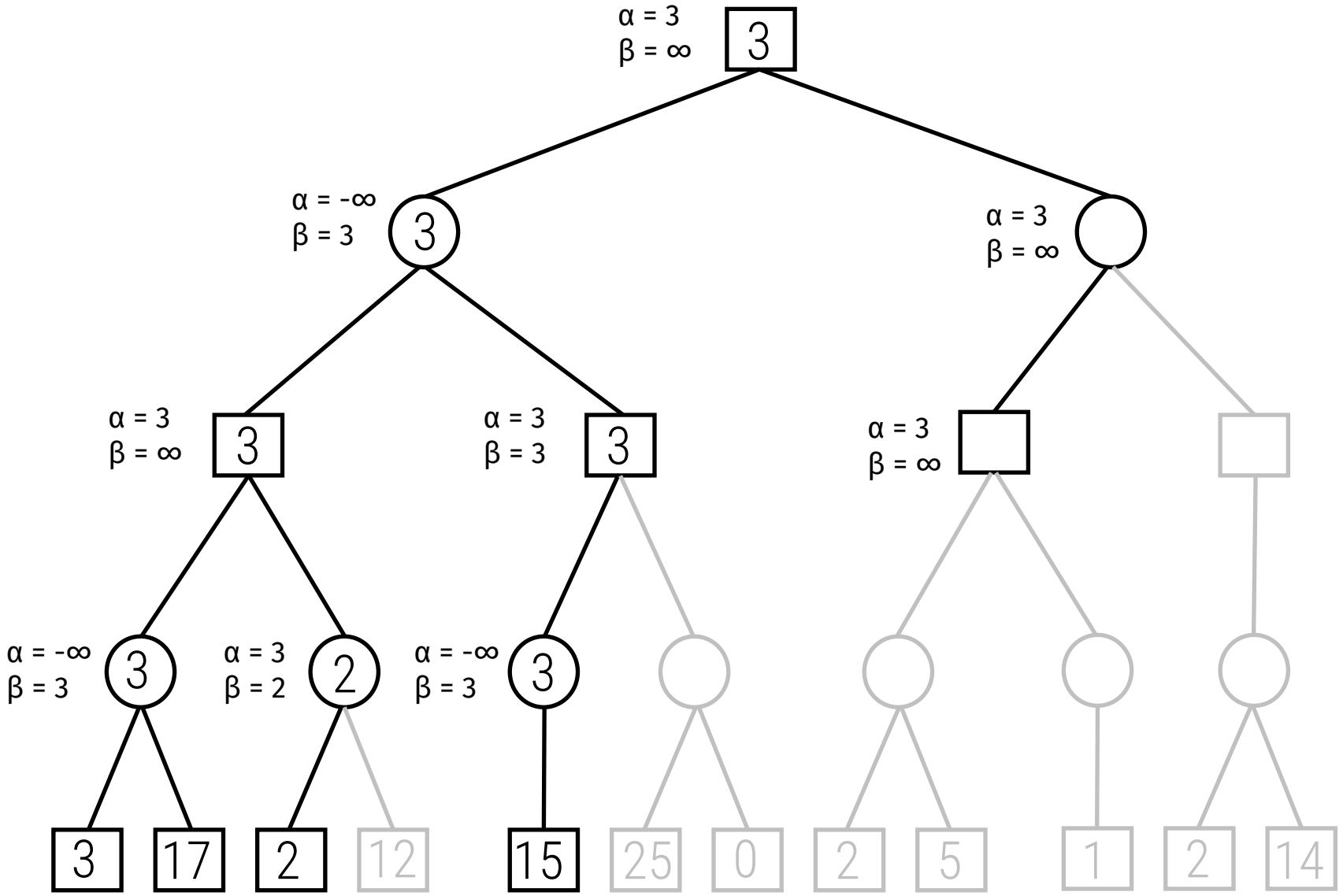
Ejemplo: Algoritmo de minimax con podas α y β



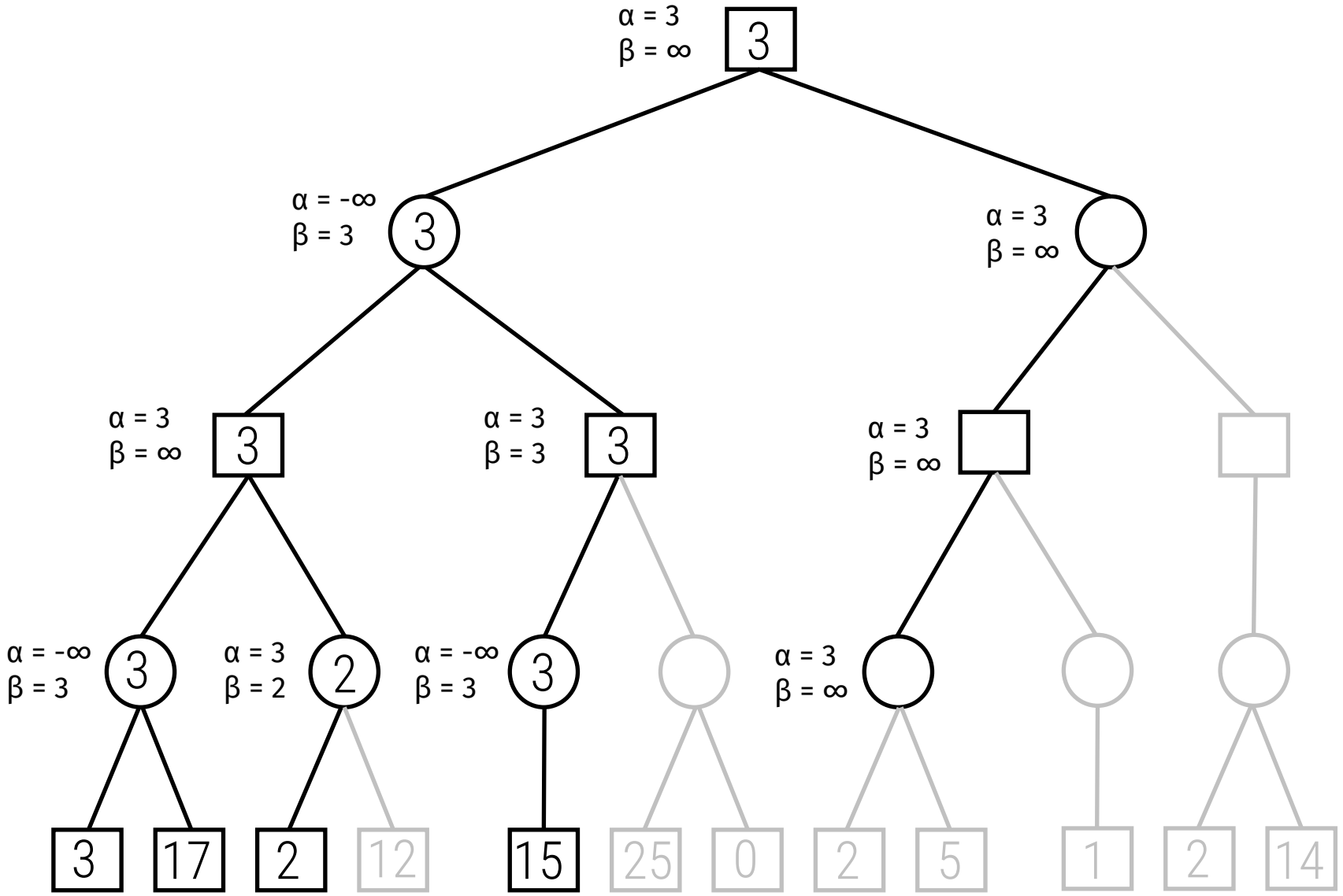
Ejemplo: Algoritmo de minimax con podas α y β



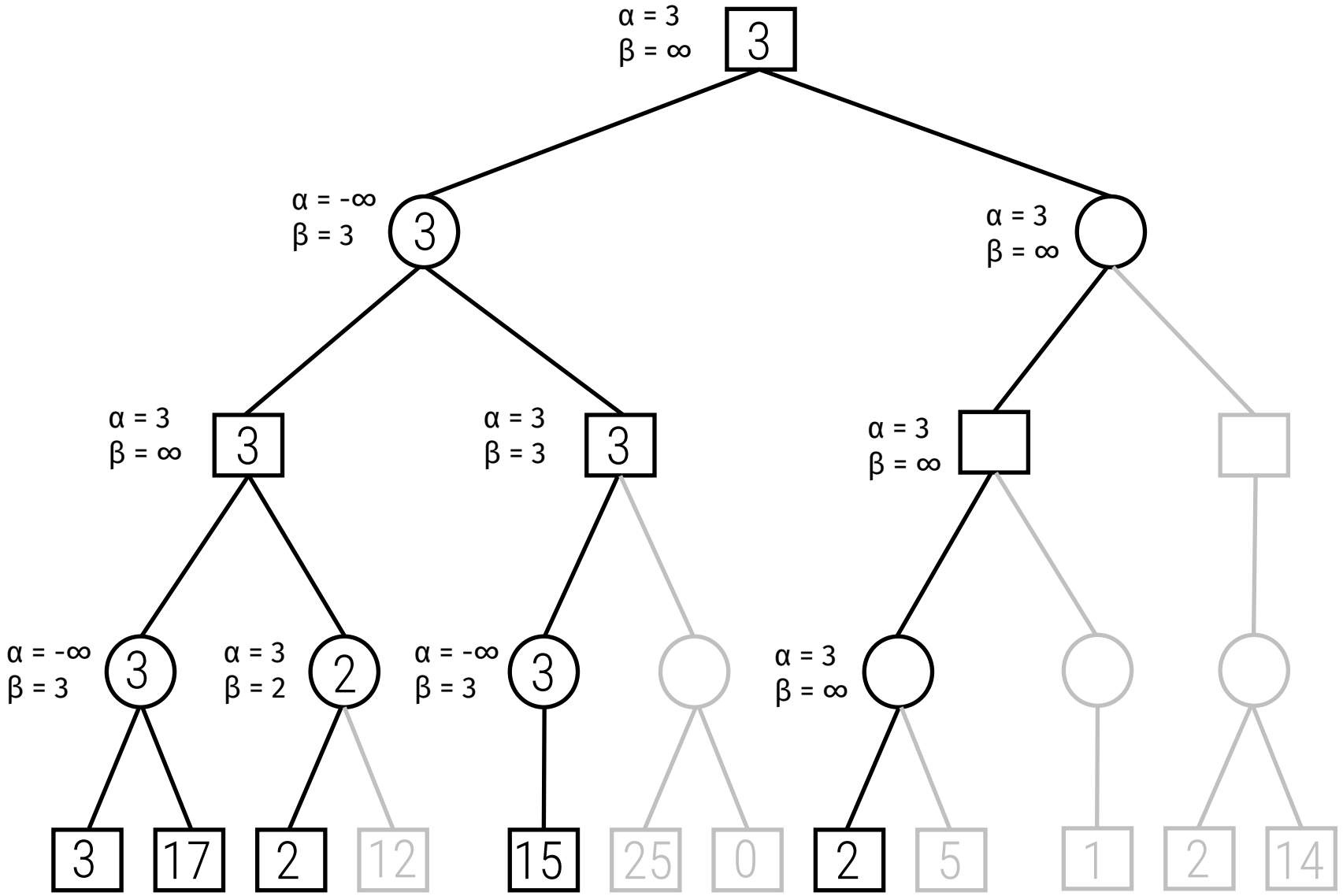
Ejemplo: Algoritmo de minimax con podas α y β



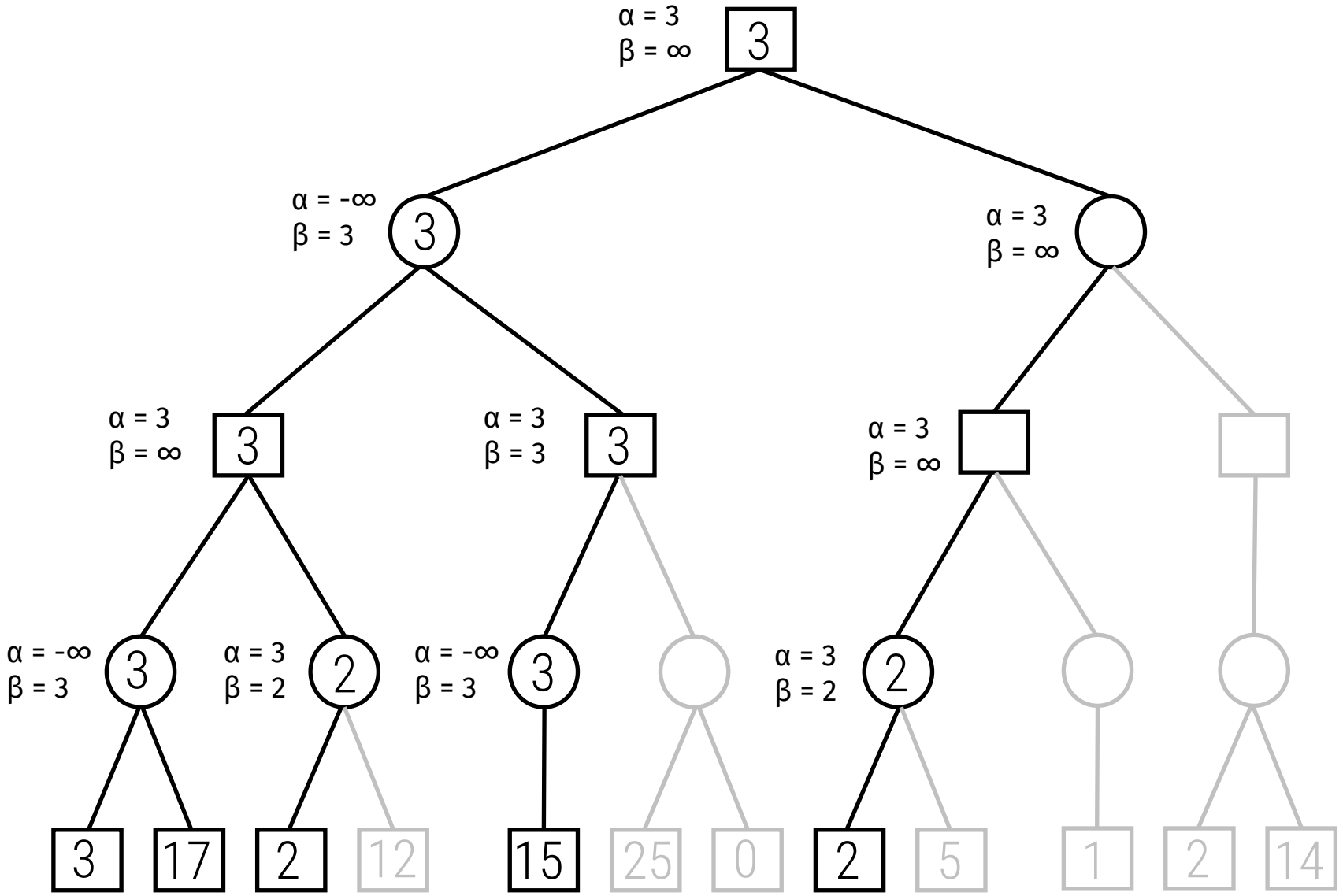
Ejemplo: Algoritmo de minimax con podas α y β



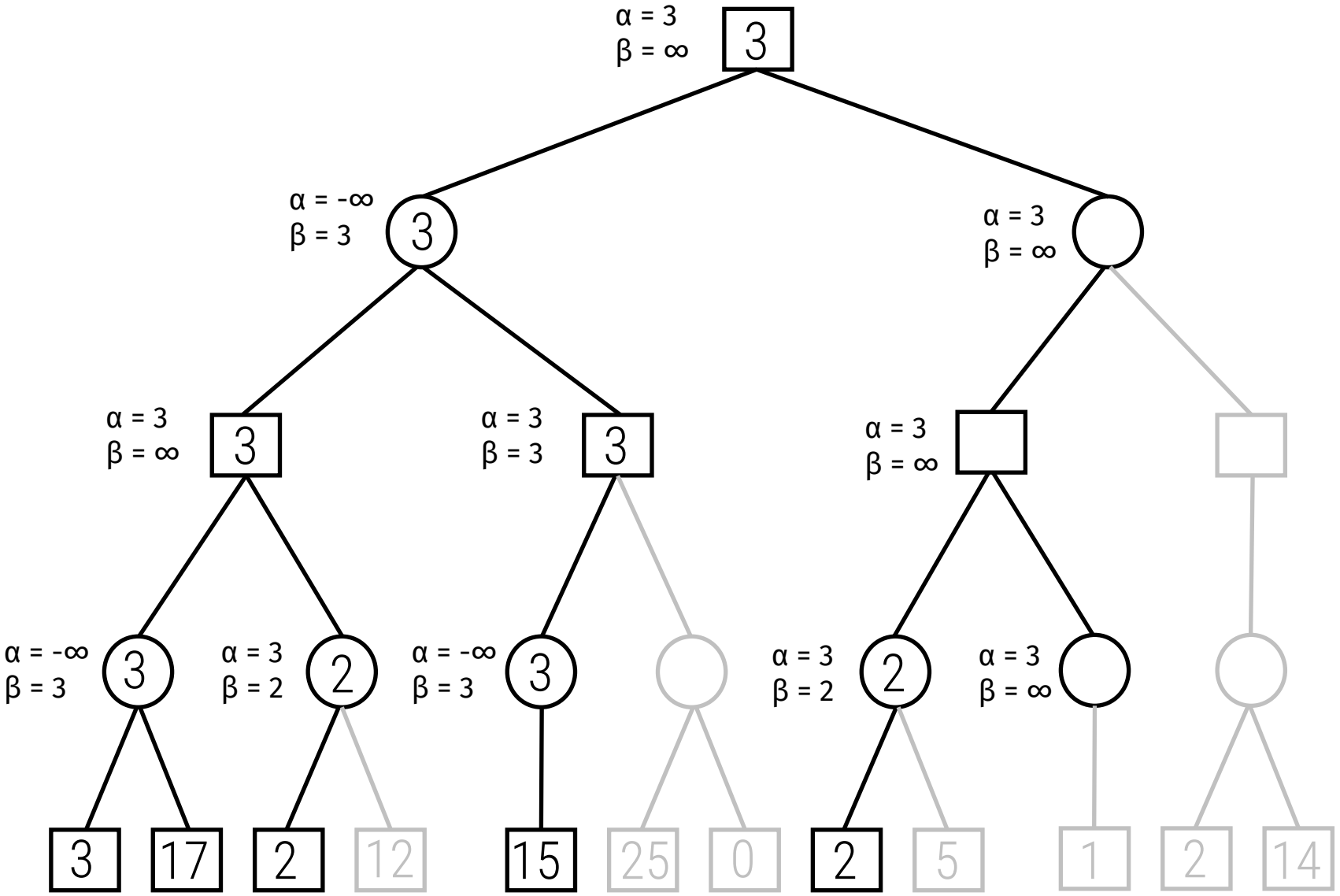
Ejemplo: Algoritmo de minimax con podas α y β



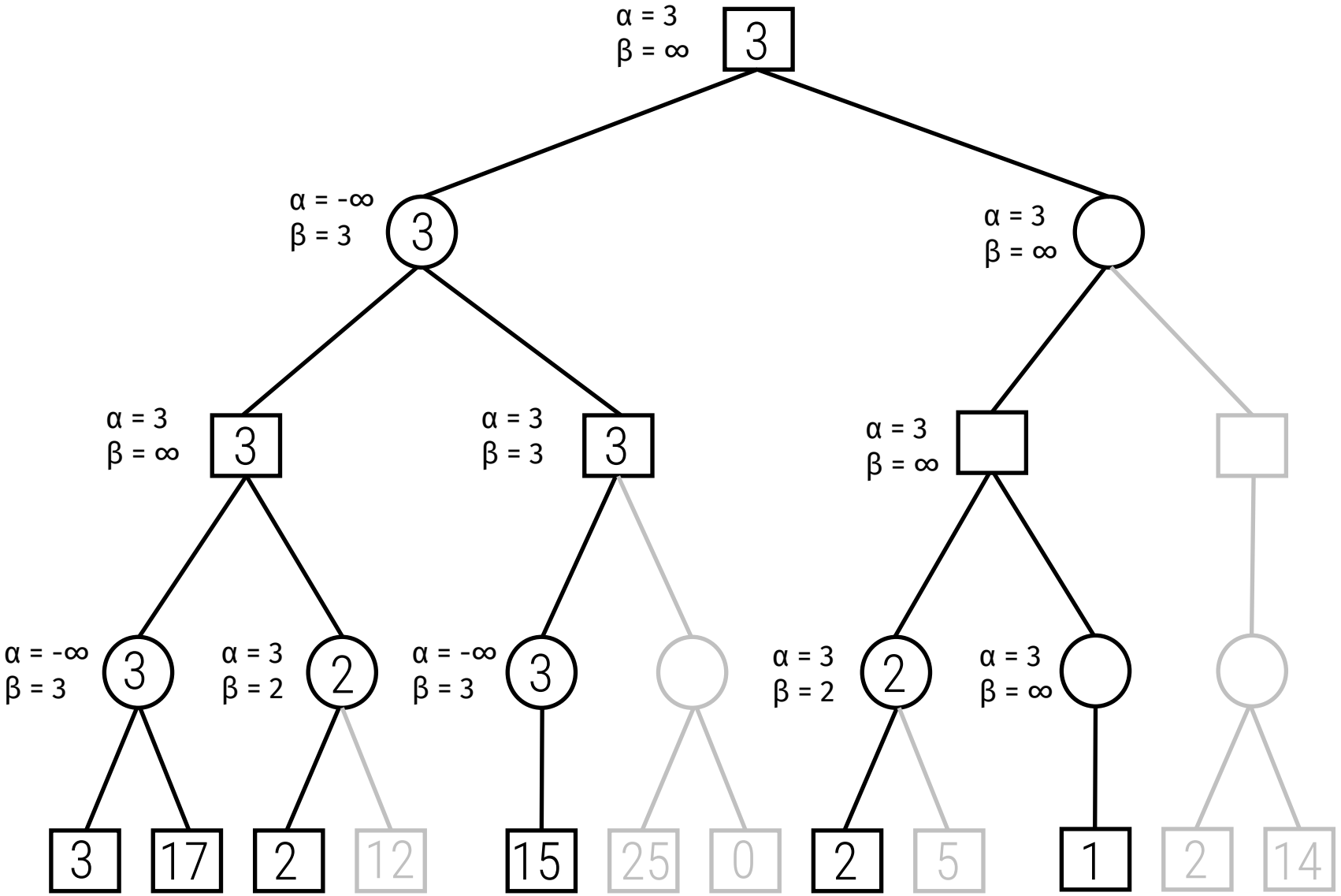
Ejemplo: Algoritmo de minimax con podas α y β



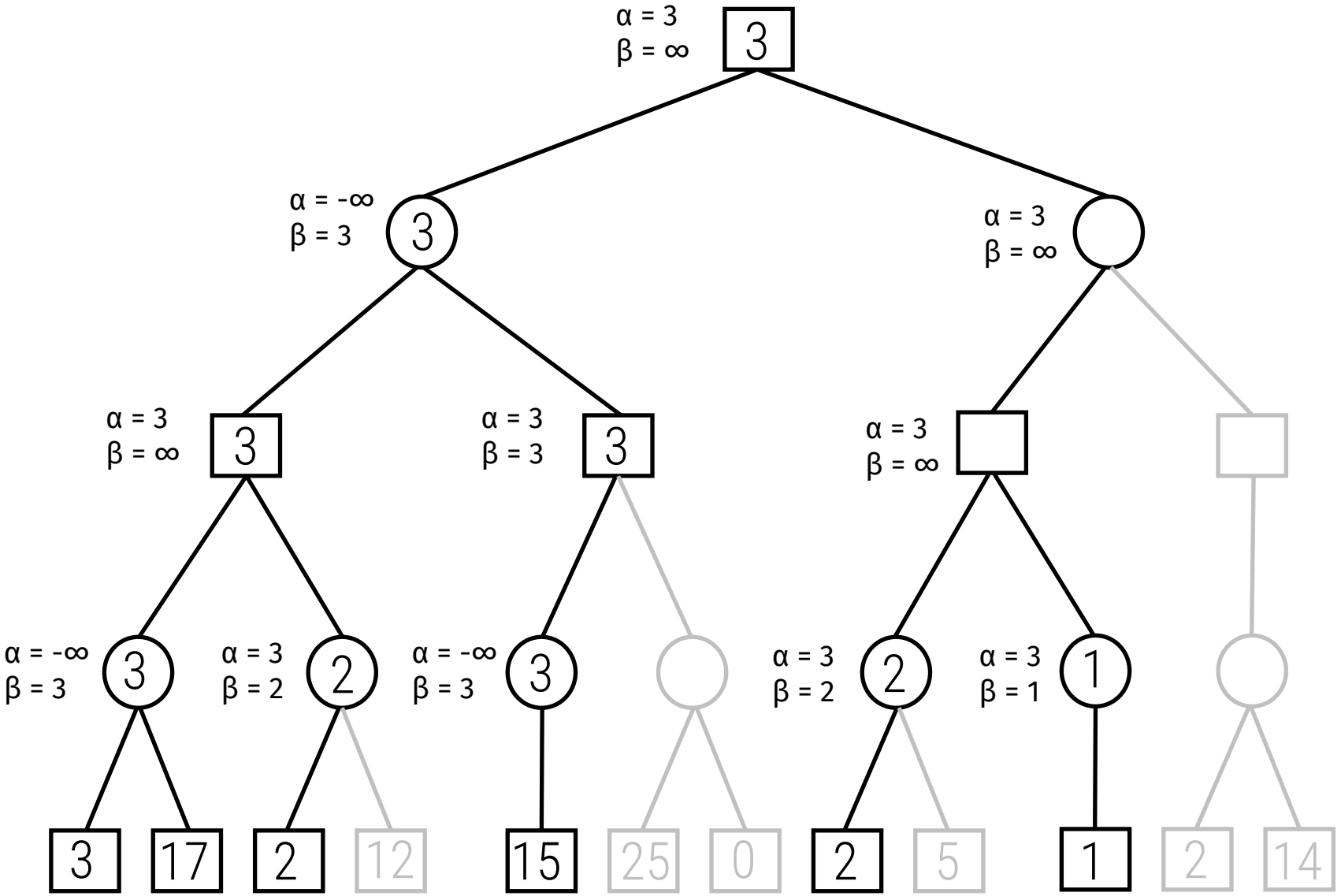
Ejemplo: Algoritmo de minimax con podas α y β



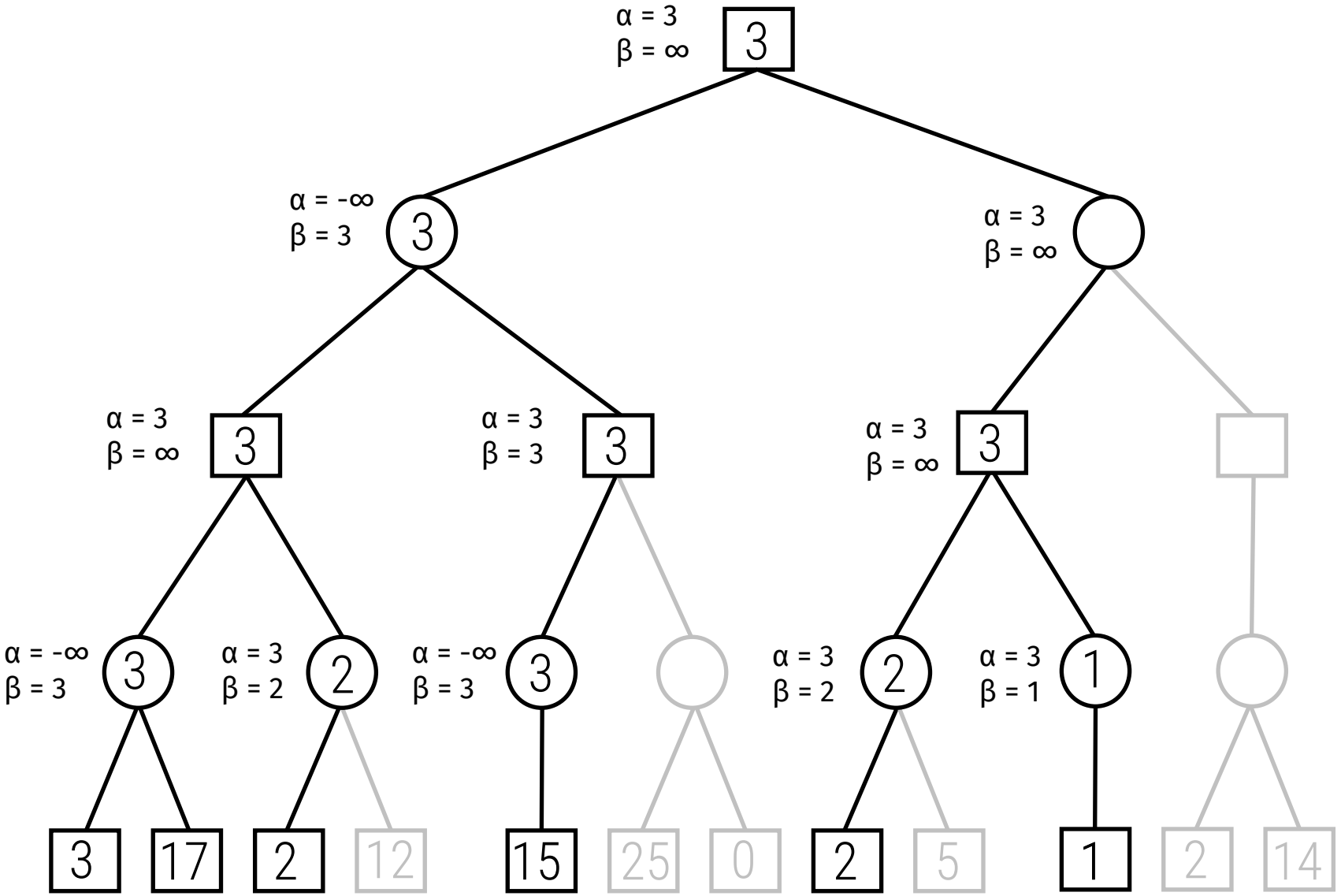
Ejemplo: Algoritmo de minimax con podas α y β



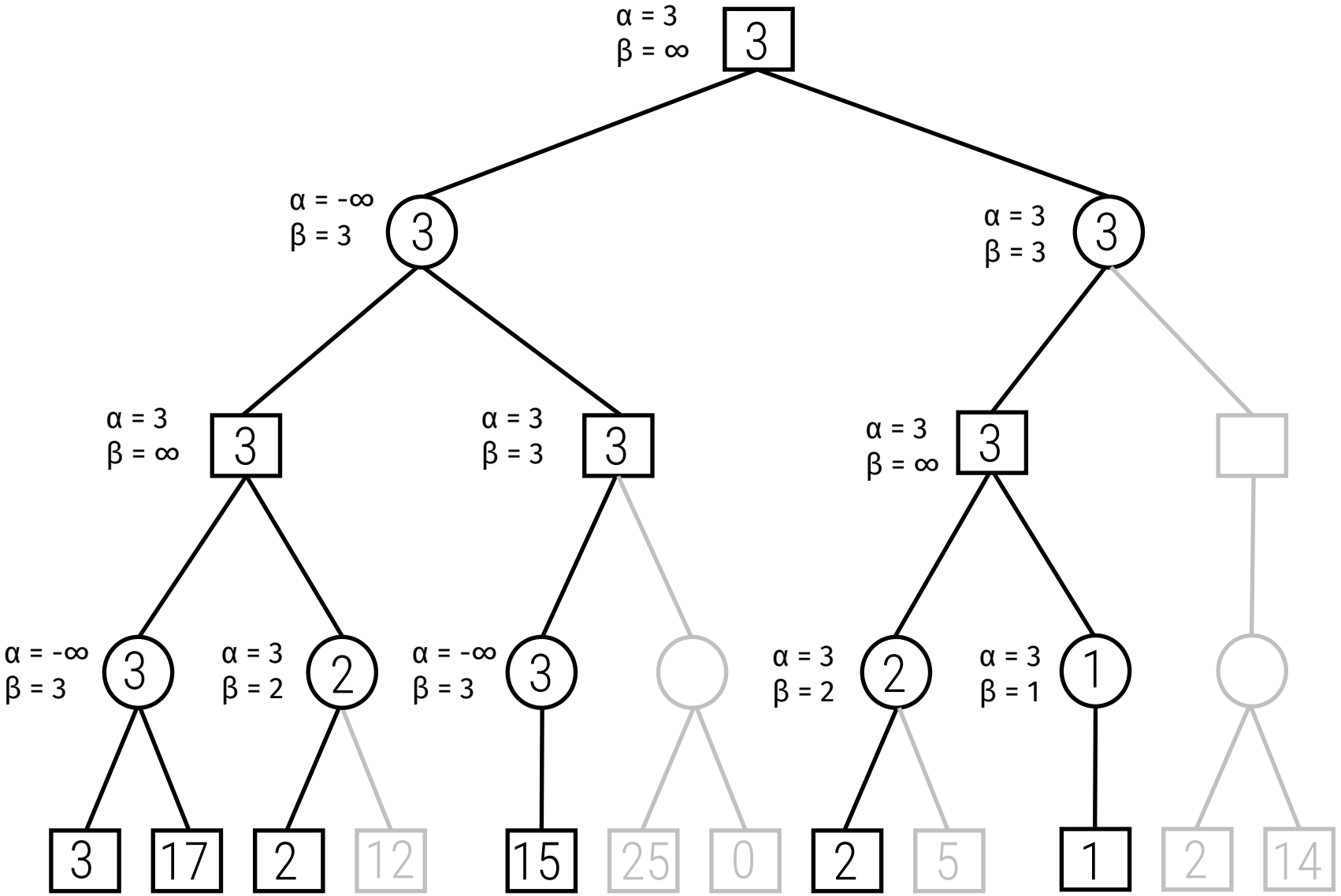
Ejemplo: Algoritmo de minimax con podas α y β



Ejemplo: Algoritmo de minimax con podas α y β



Ejemplo: Algoritmo de minimax con podas α y β



Ejemplo: Algoritmo de minimax con podas α y β