

Árboles de juego

Alberto Verdejo

Dpto. de Sistemas Informáticos y Computación

Universidad Complutense de Madrid

Diciembre 2012

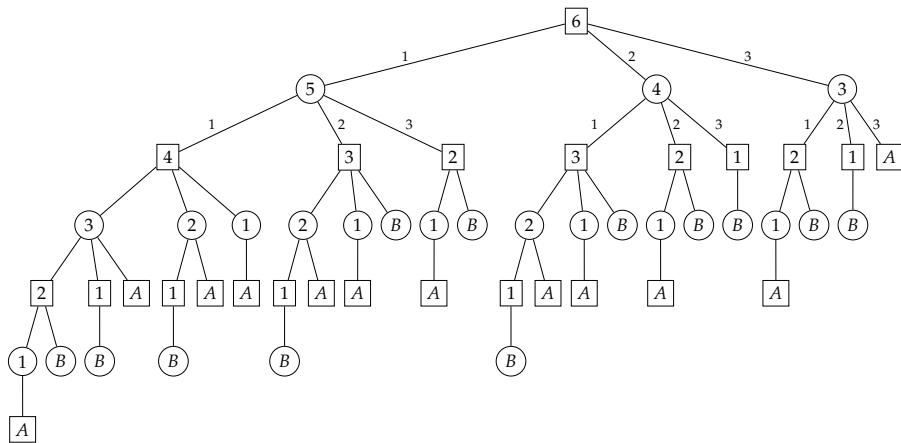
Motivación

- Queremos un programa que sepa jugar, lo haga de forma óptima y en un tiempo razonable.
- Tipos de juegos:

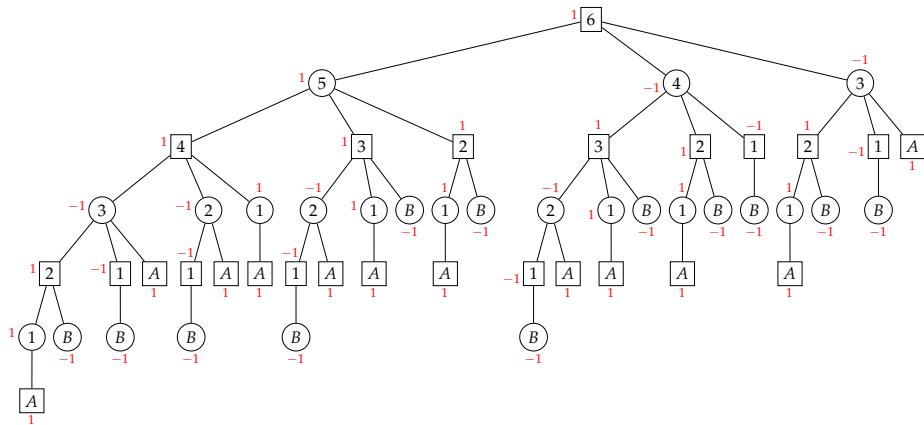
	Determinista	No determinista (aleatoriedad)
Totalmente observable	Ajedrez Damas Go Othello	Backgammon Monopoly
Parcialmente observable	Hundir la flota	Juegos de cartas

Juego de NIM con 6 palillos

Dos jugadores A y B y un tablero que inicialmente contiene n ($= 6$) palillos. Los jugadores se alternan comenzando A . Un *movimiento legal* consiste en eliminar 1, 2 o 3 palillos (pero no más de los que haya). Pierde el jugador que elimina el último palillo y el otro gana.



Juego de NIM con 6 palillos, nodos valorados



Algoritmo de minimax

```

jugada Jugador::juega(Juego EJ) {
    jugada mejorC;
    float mejorV =  $-\infty$ ;
    for (jugada c : jugadasDesde(EJ)) {
        float v = valoraMin(EJ.aplica(c), nivel-1);
        if (v > mejorV){ mejorV = v; mejorC = c; }
    }
    return mejorC;
}

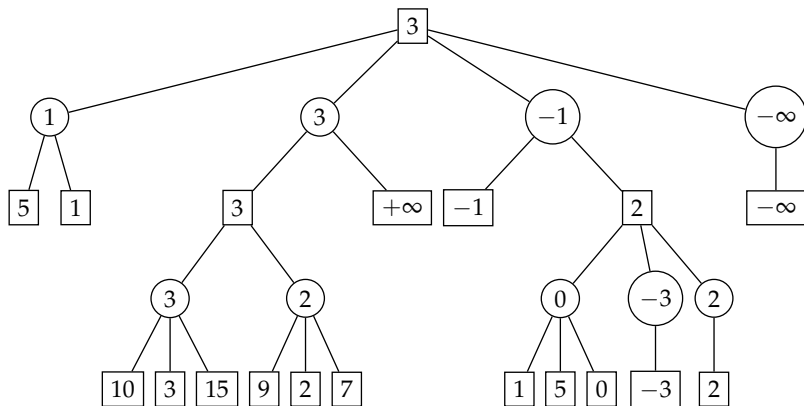
float Jugador::valoraMin(Juego EJ, int n) {
    if (terminal(EJ) || n==0) return Heuristica(EJ)
    else { float mejorV =  $+\infty$ ;
        for (jugada c : jugadasDesde(EJ))
            mejorV = min(valoraMax(EJ.aplica(c), n-1), mejorV);
        return mejorV; }
}

float Jugador::valoraMax(Juego EJ, int n) {
    // igual que valoraMin pero haciendo un máximo
}

```

Podas α y β

Evitan el procesamiento de subárboles que no afectan a la decisión sobre cuál es la mejor jugada.



Algoritmo de poda alfa-beta

```

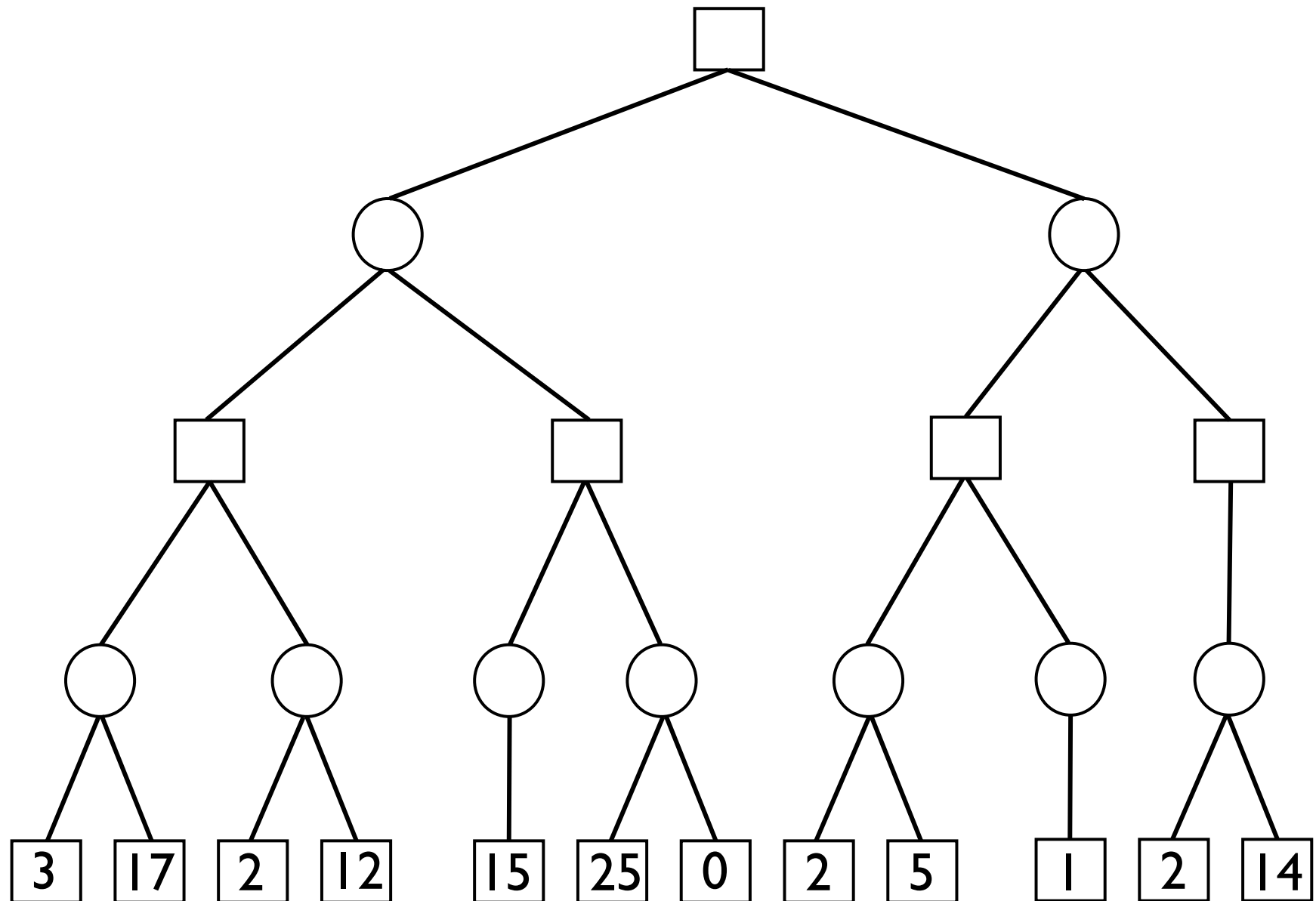
jugada Jugador::juega(Juego EJ) {
    jugada mejorC;
    float mejorV =  $-\infty$ ;
    for (jugada c : jugadasDesde(EJ)) {
        float v = valoraMin(EJ.aplica(c), nivel-1, mejorV,  $+\infty$ );
        if (v > mejorV){ mejorV = v; mejorC = c; }
    }
    return mejorC;
}

float Jugador::valoraMin(Juego EJ, int n, float  $\alpha$ , float  $\beta$ ) {
    if (terminal(EJ) || n==0) return Heuristica(EJ)
    else { for (jugada c : jugadasDesde(EJ)) {
         $\beta$  = min(valoraMax(EJ.aplica(c), n-1,  $\alpha$ ,  $\beta$ ),  $\beta$ );
        if ( $\alpha$  >=  $\beta$ ) return  $\beta$ ;
    }
    return  $\beta$ ; }
}

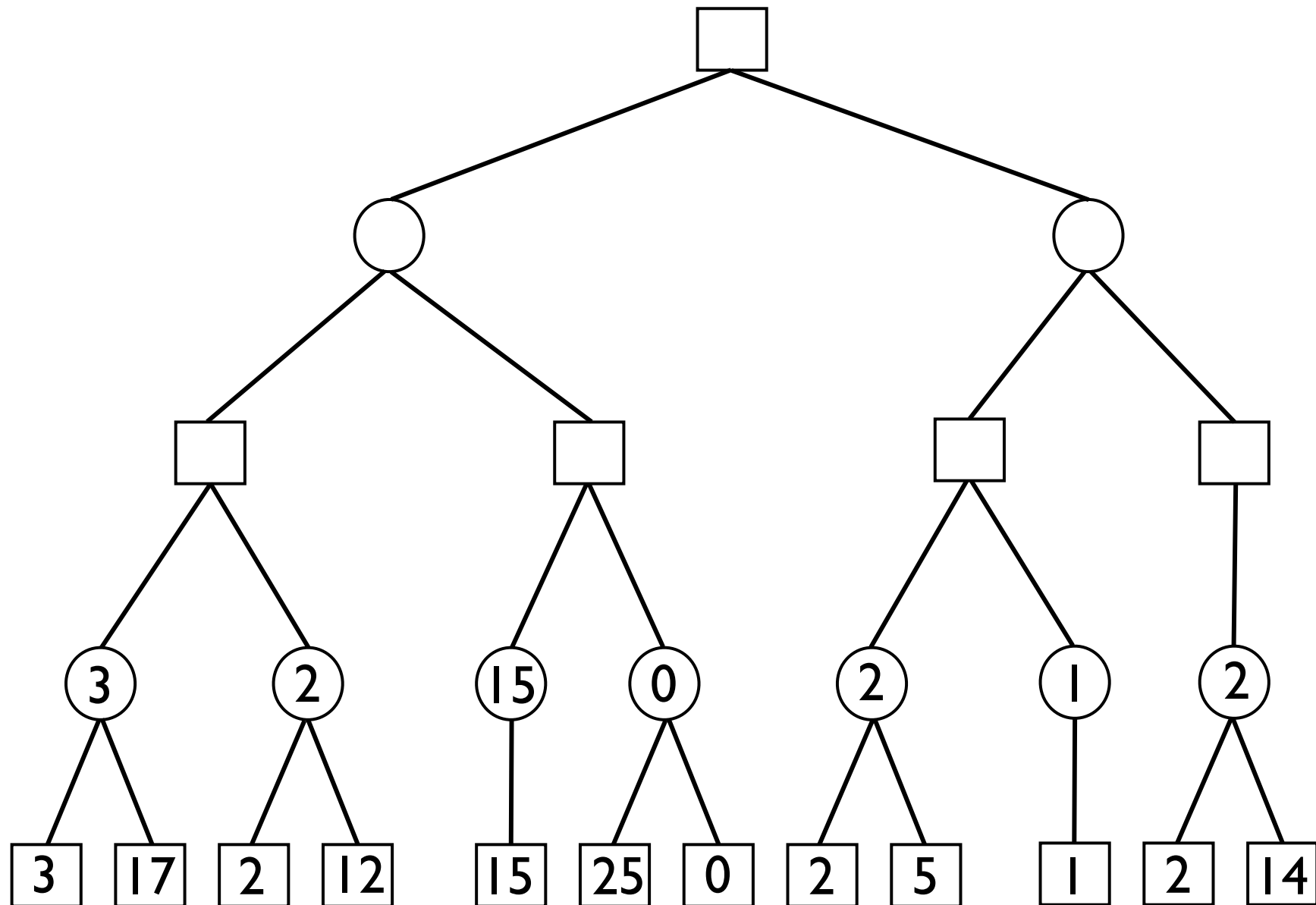
float Jugador::valoraMax(Juego EJ, int n, float  $\alpha$ , float  $\beta$ ) {
    // similar a valoraMin pero haciendo un máximo e intercambiando
    // los papeles de  $\alpha$  y  $\beta$ 
}

```

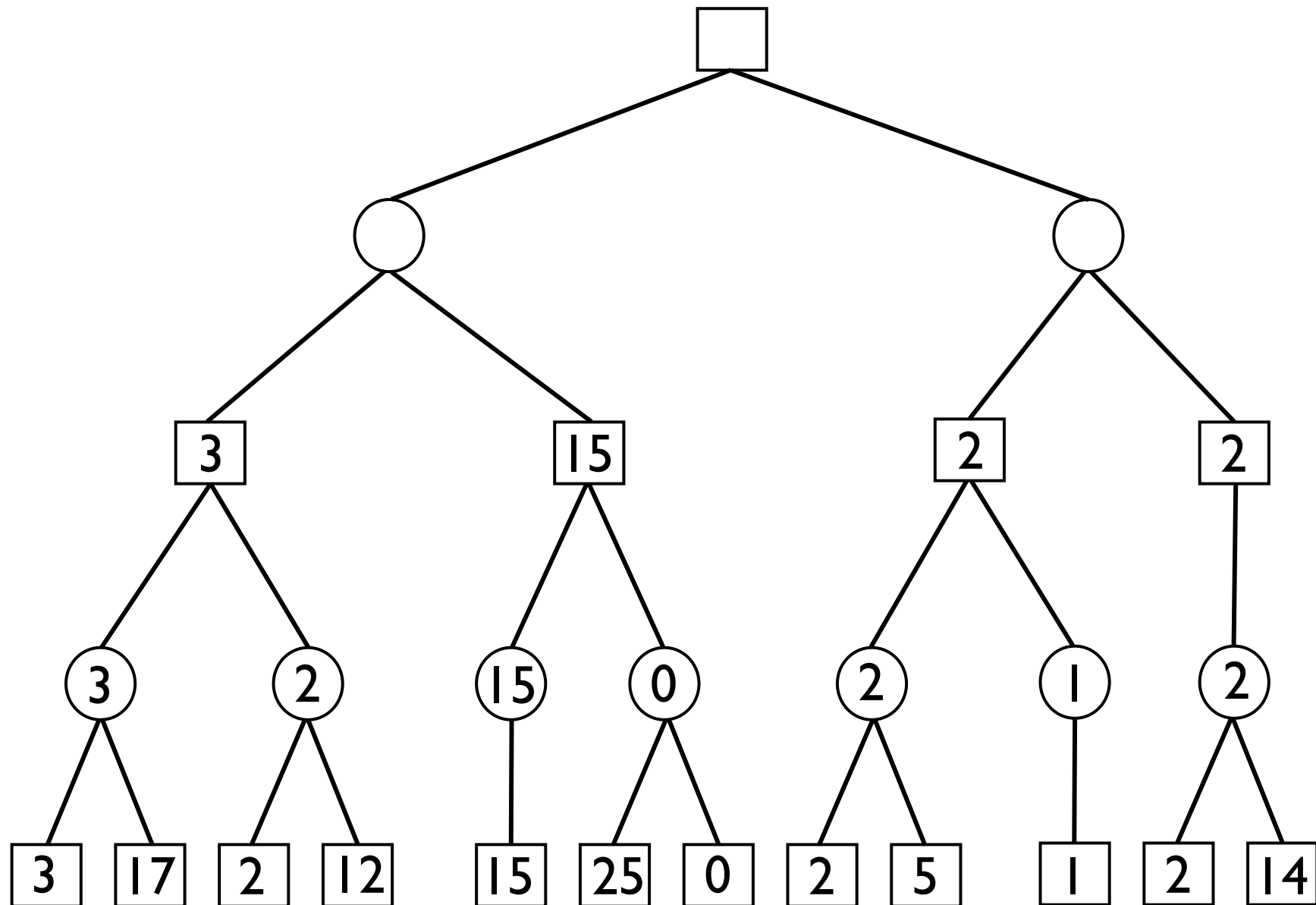
Algoritmo minimax



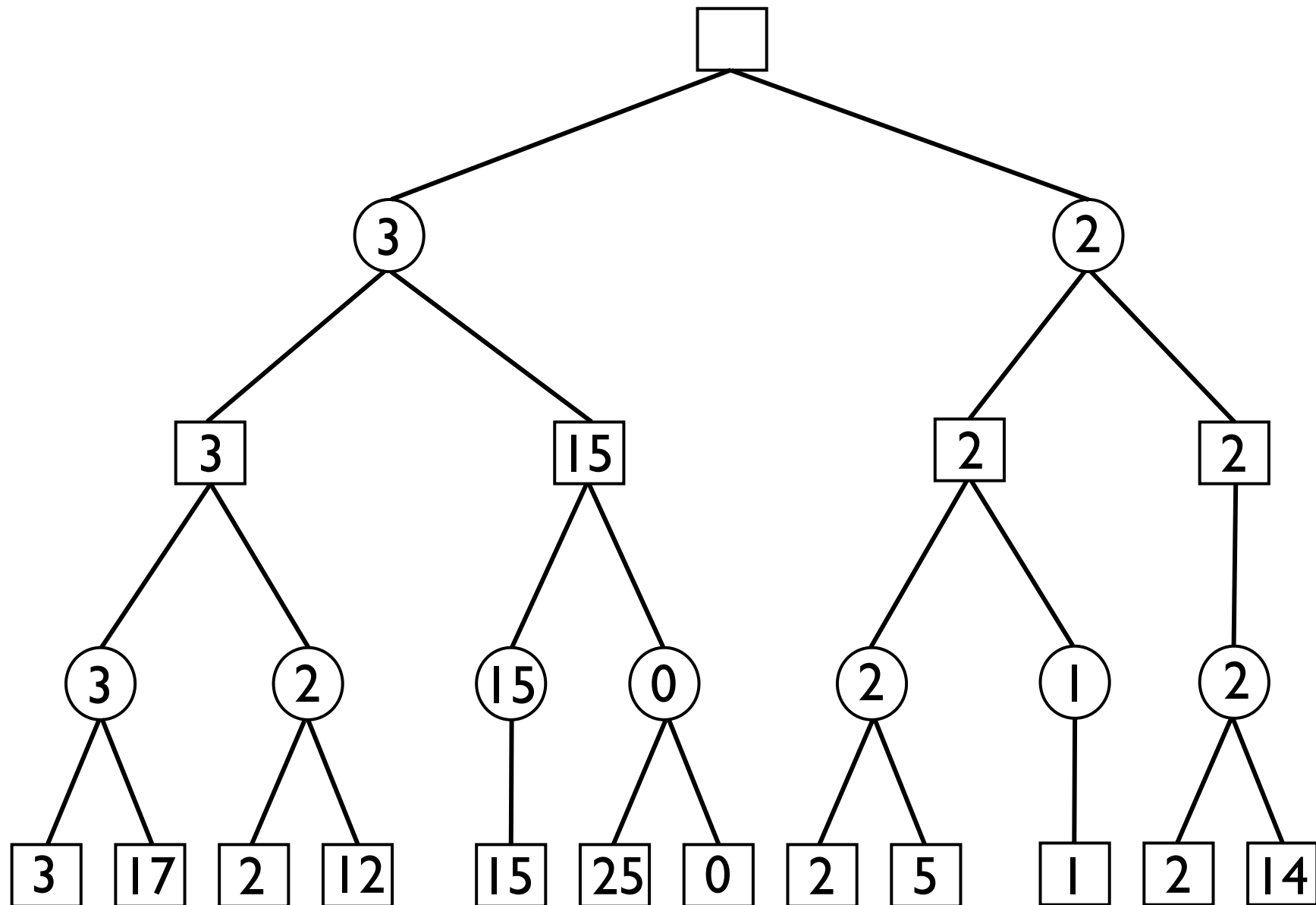
Algoritmo minimax



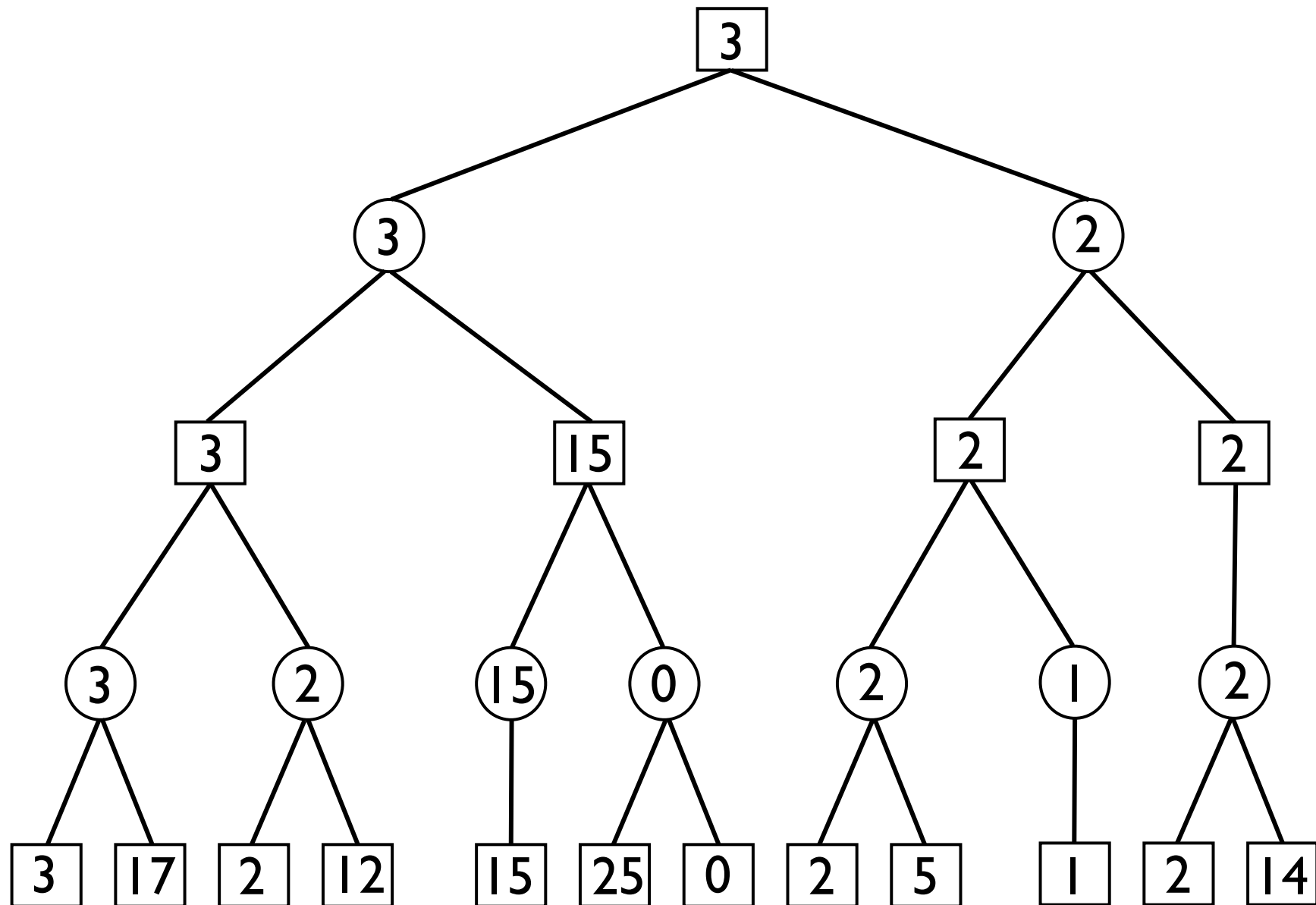
Algoritmo minimax



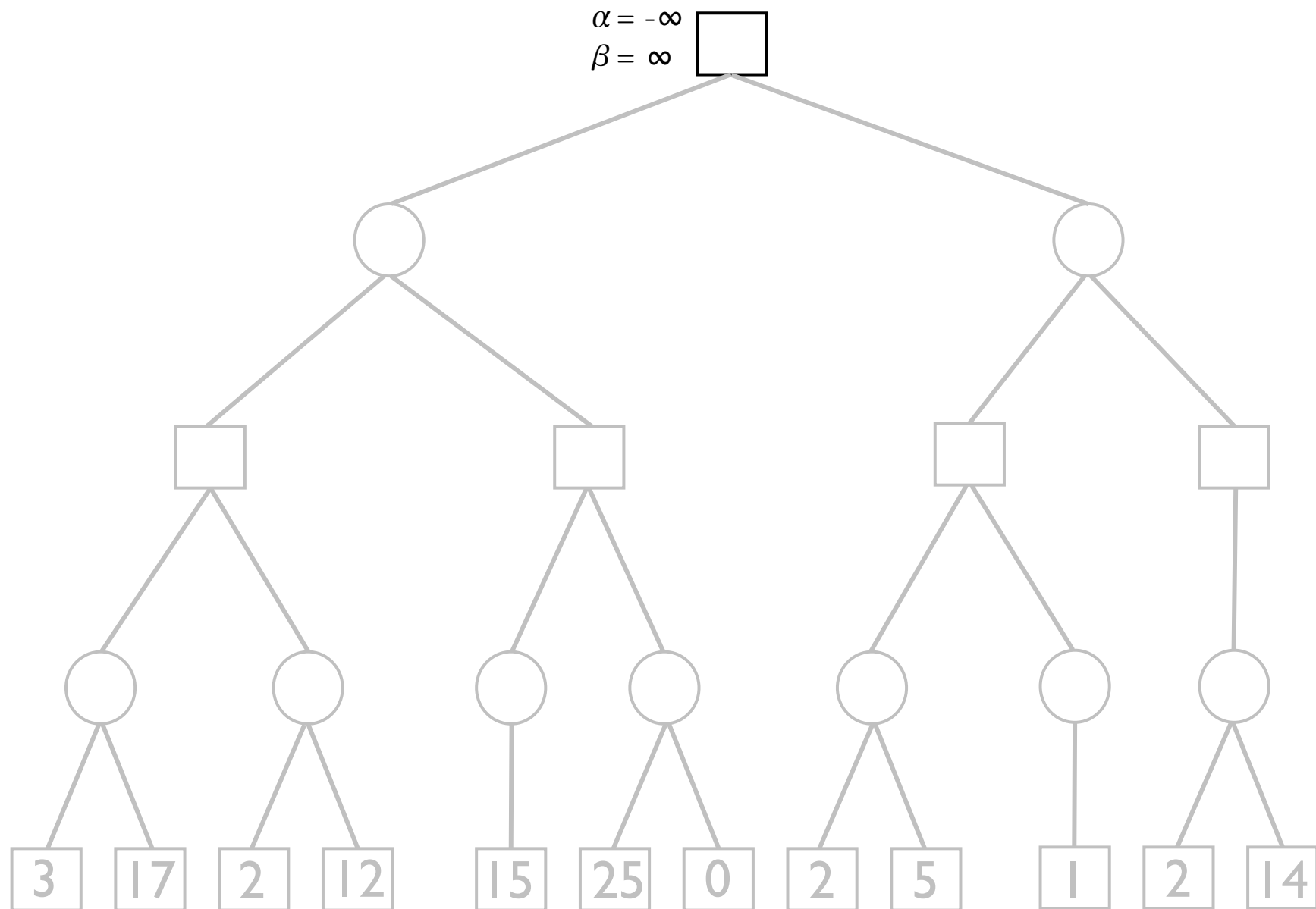
Algoritmo minimax



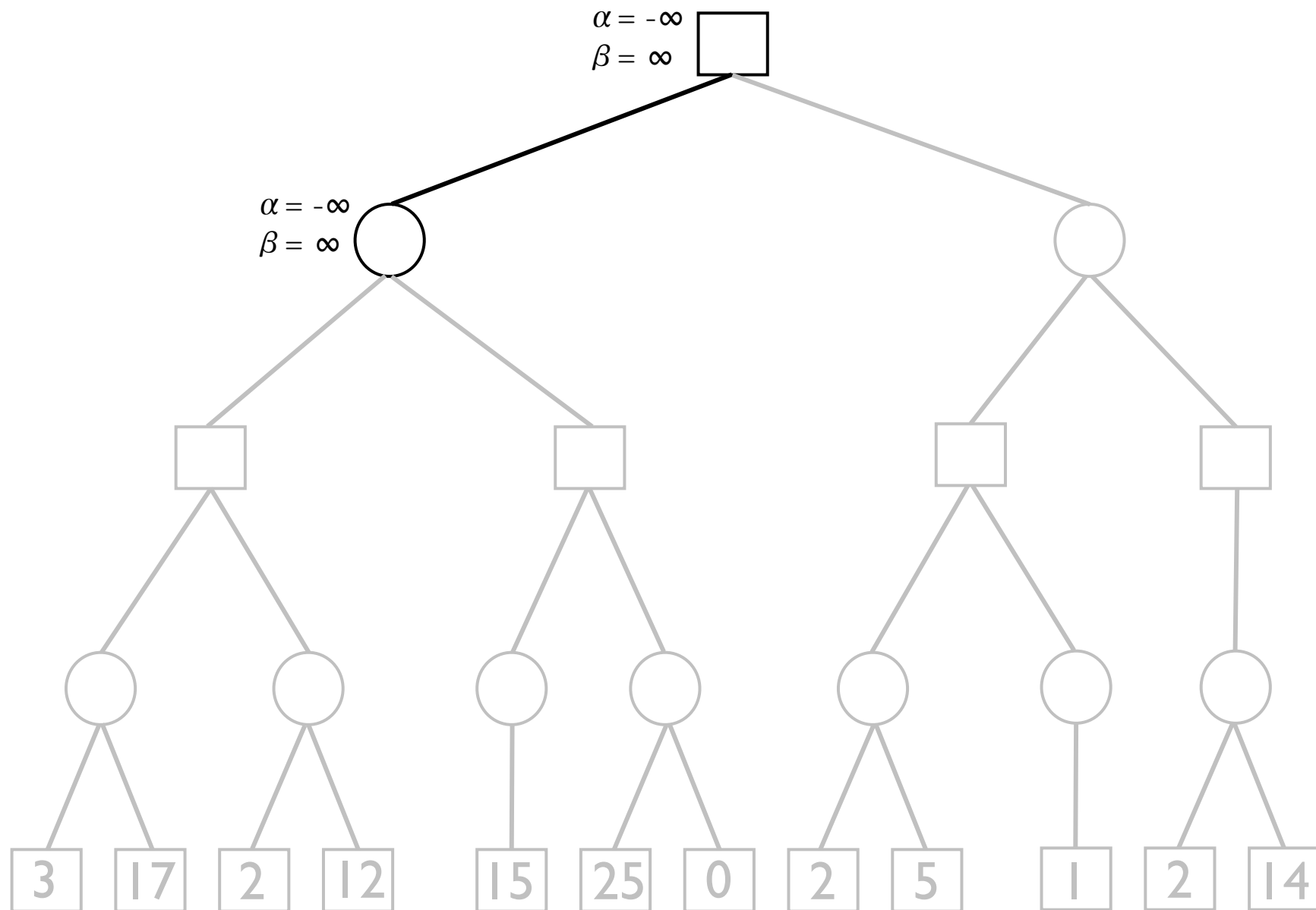
Algoritmo minimax



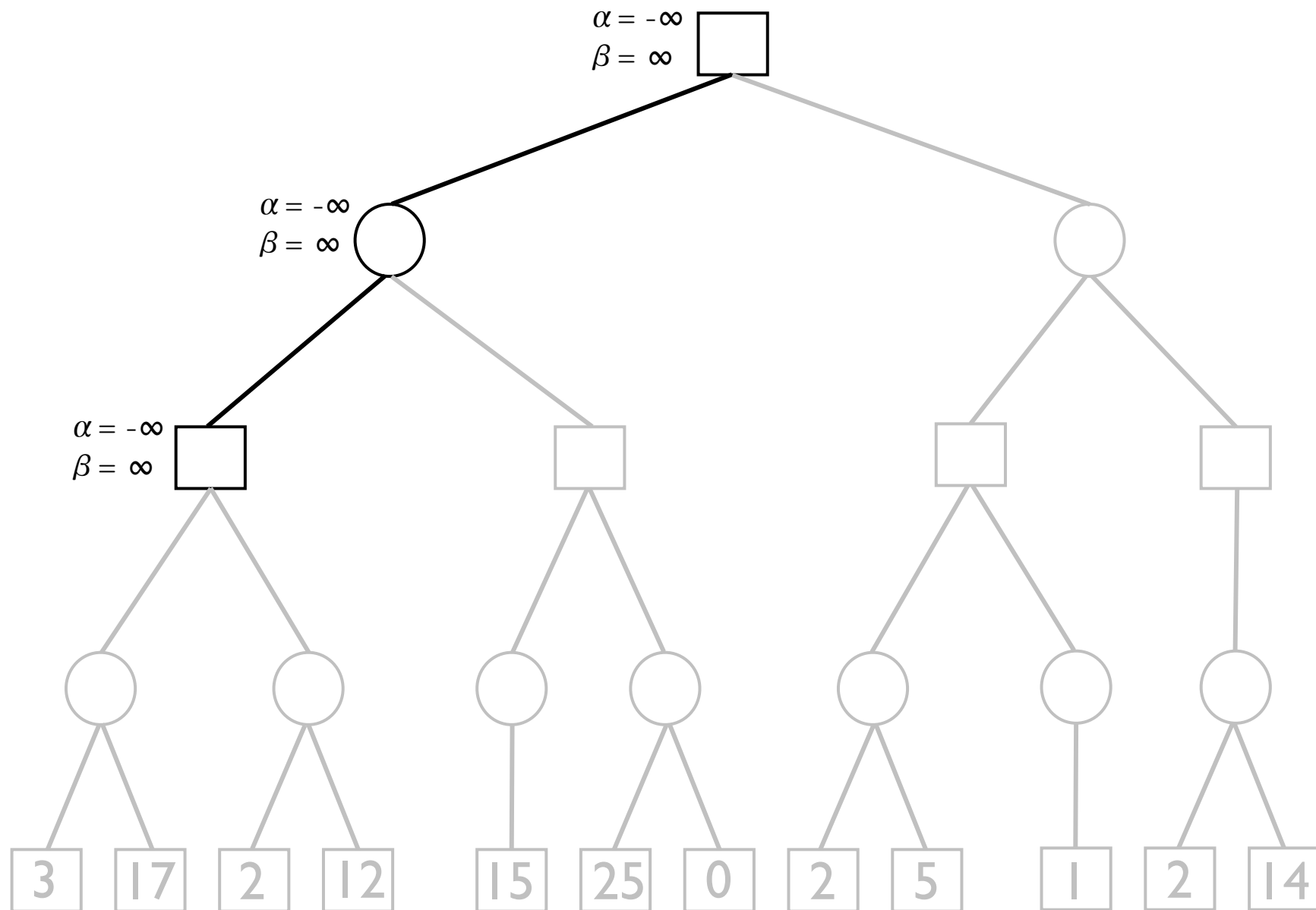
Poda alfa-beta



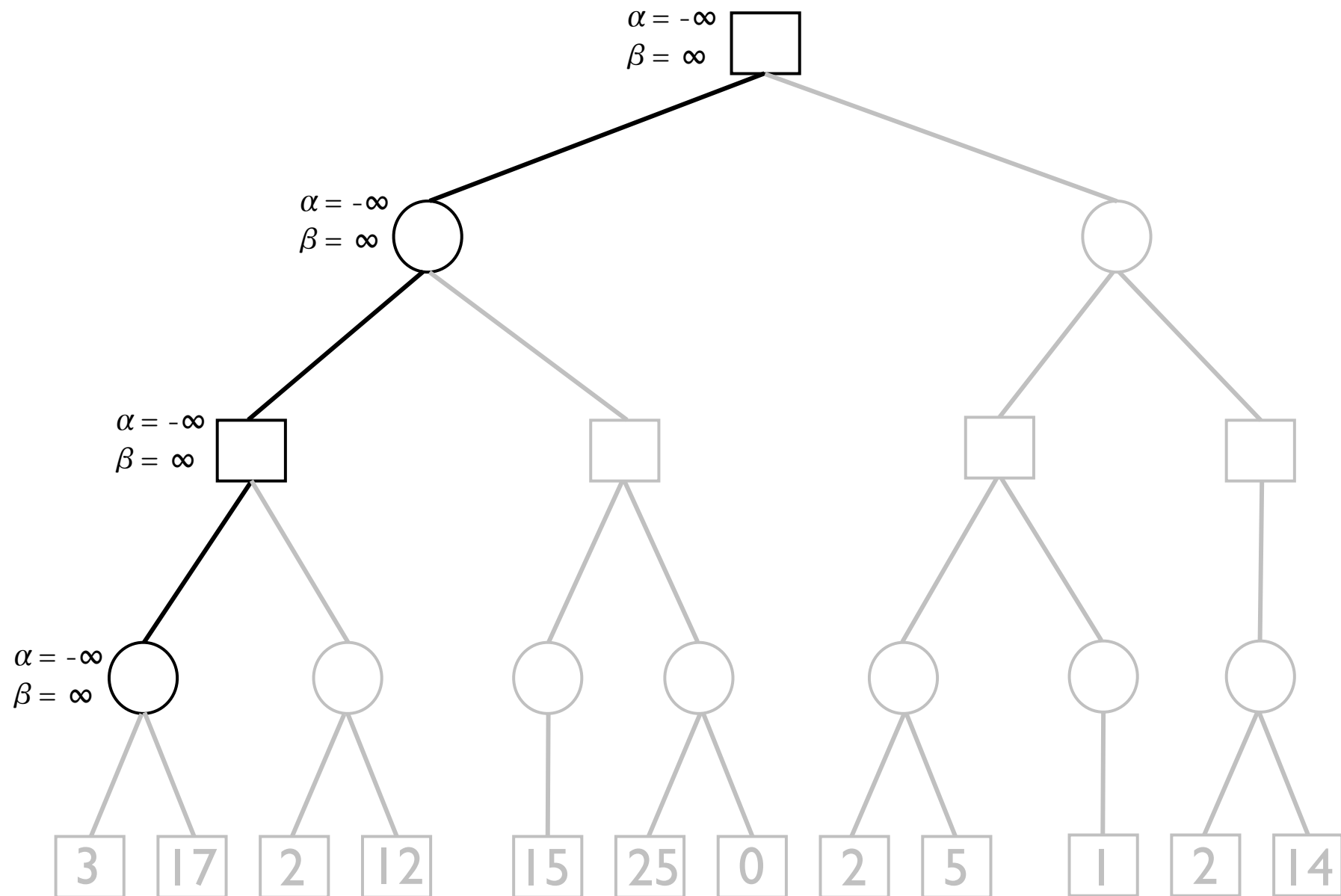
Poda alfa-beta



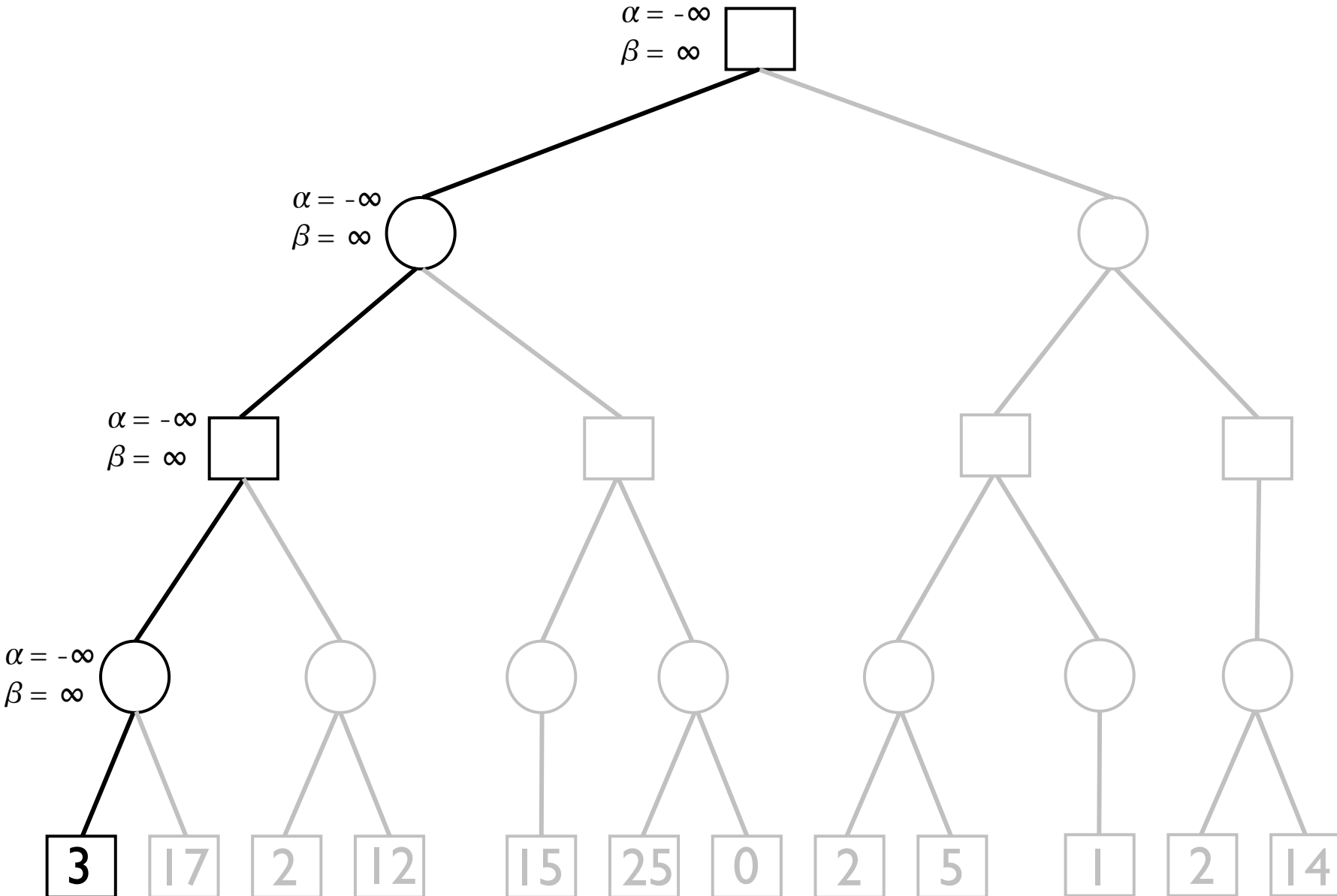
Poda alfa-beta



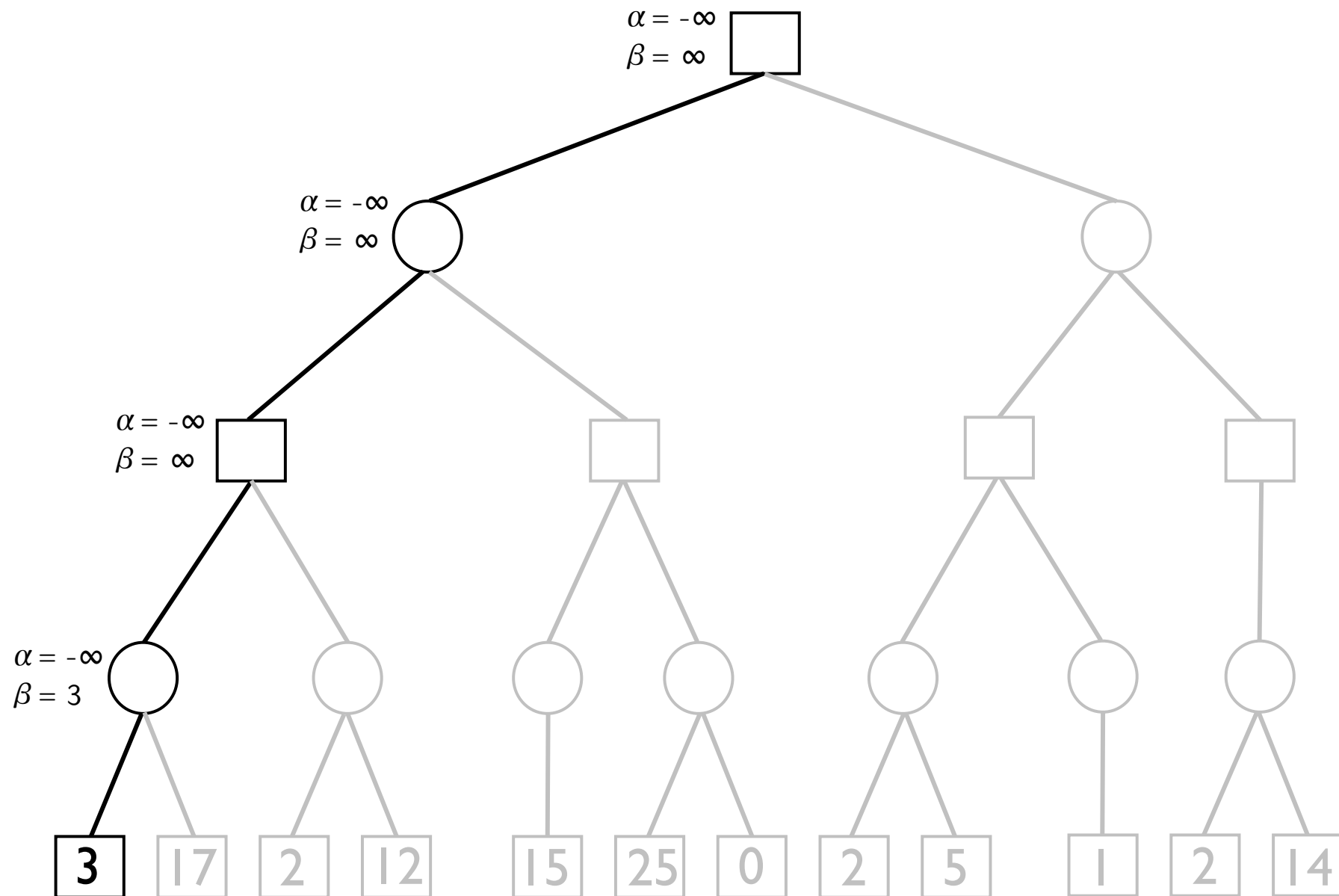
Poda alfa-beta



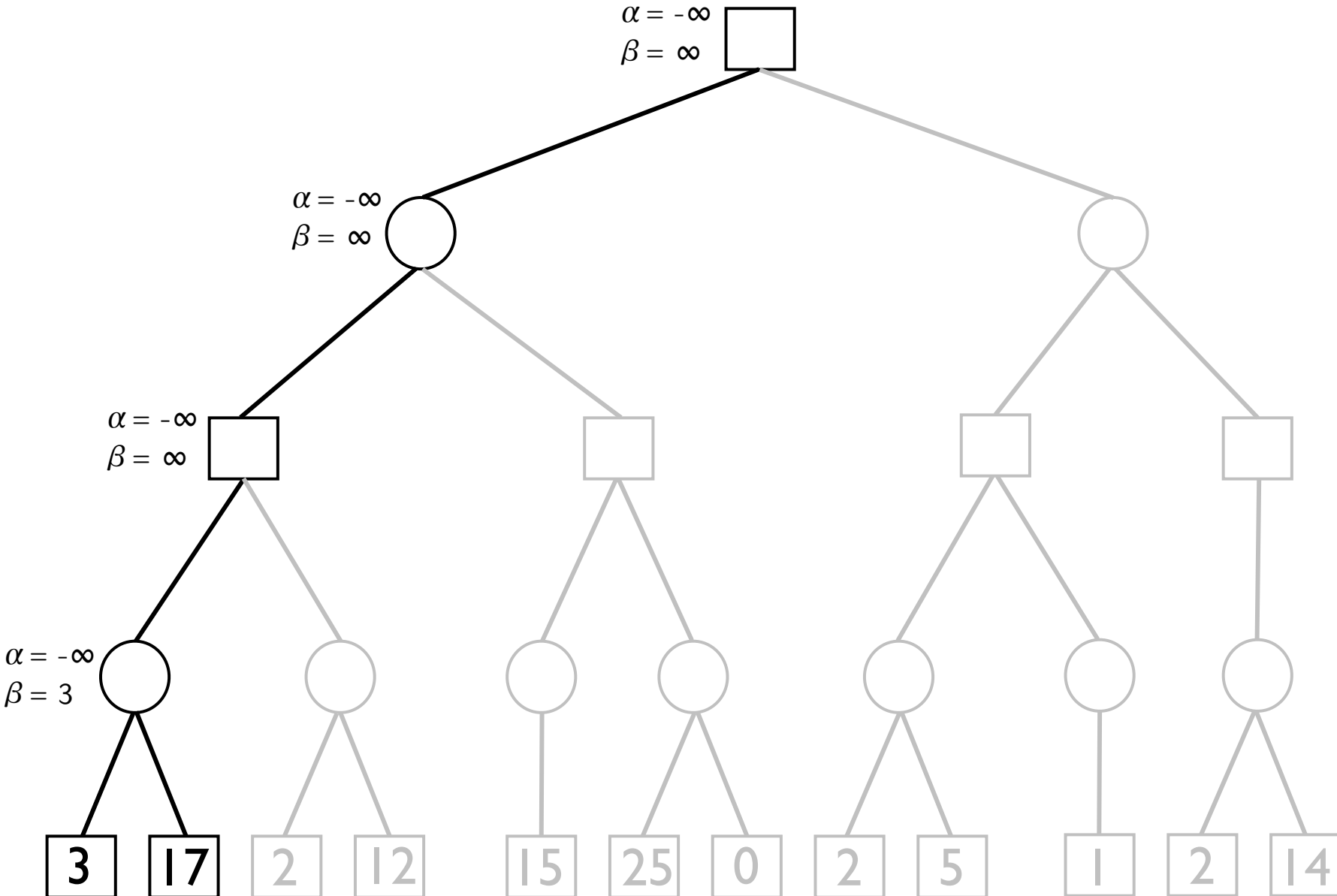
Poda alfa-beta



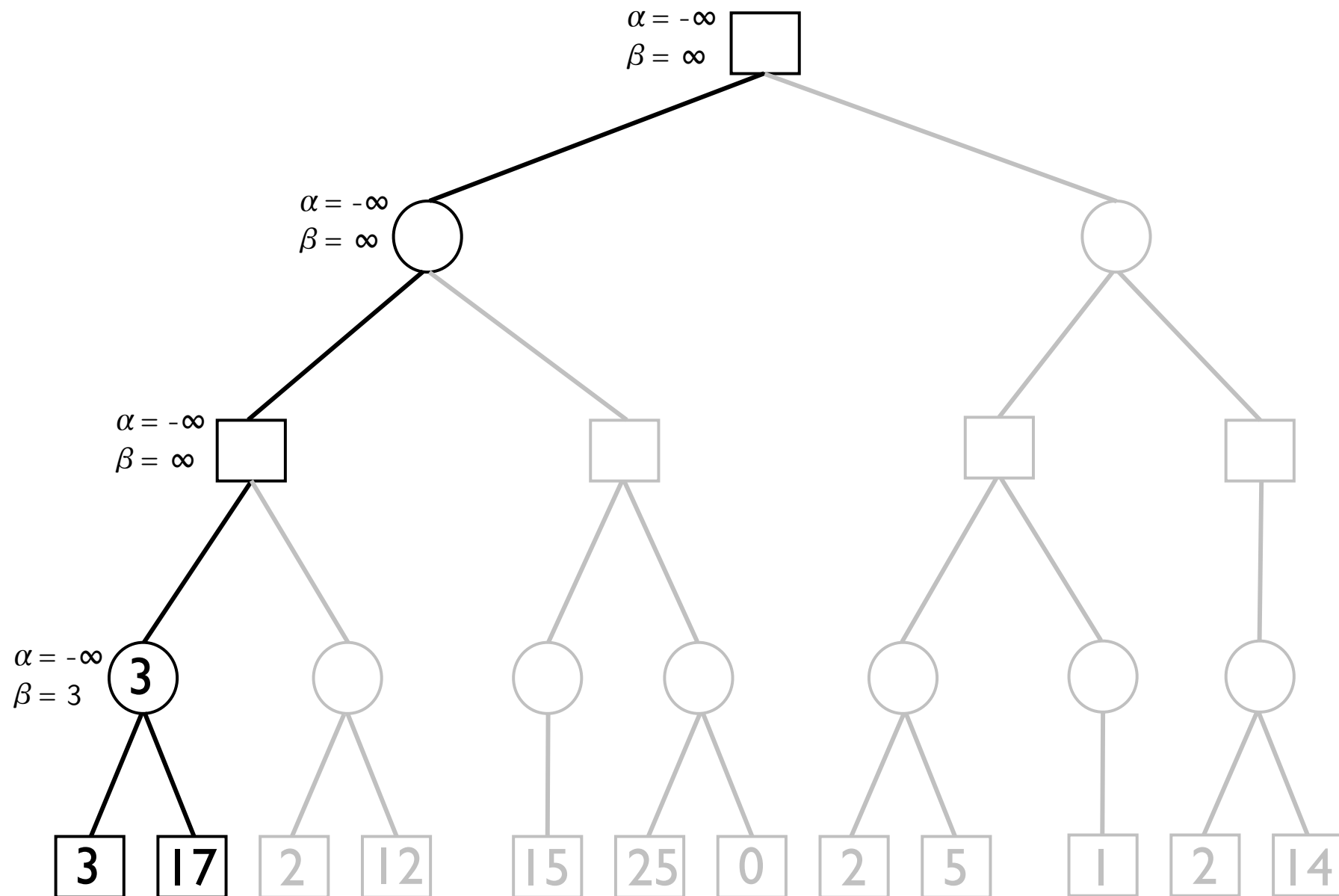
Poda alfa-beta



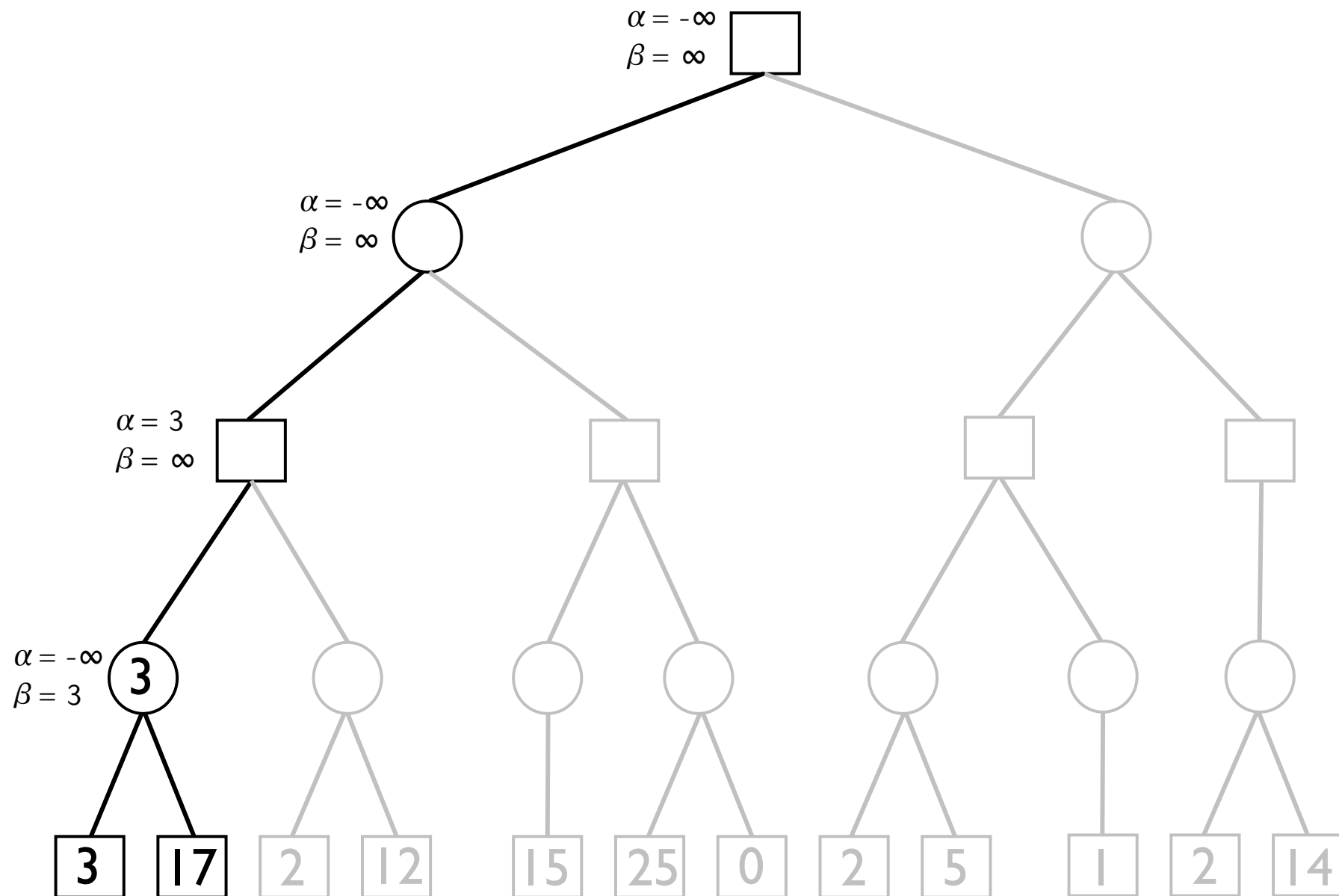
Poda alfa-beta



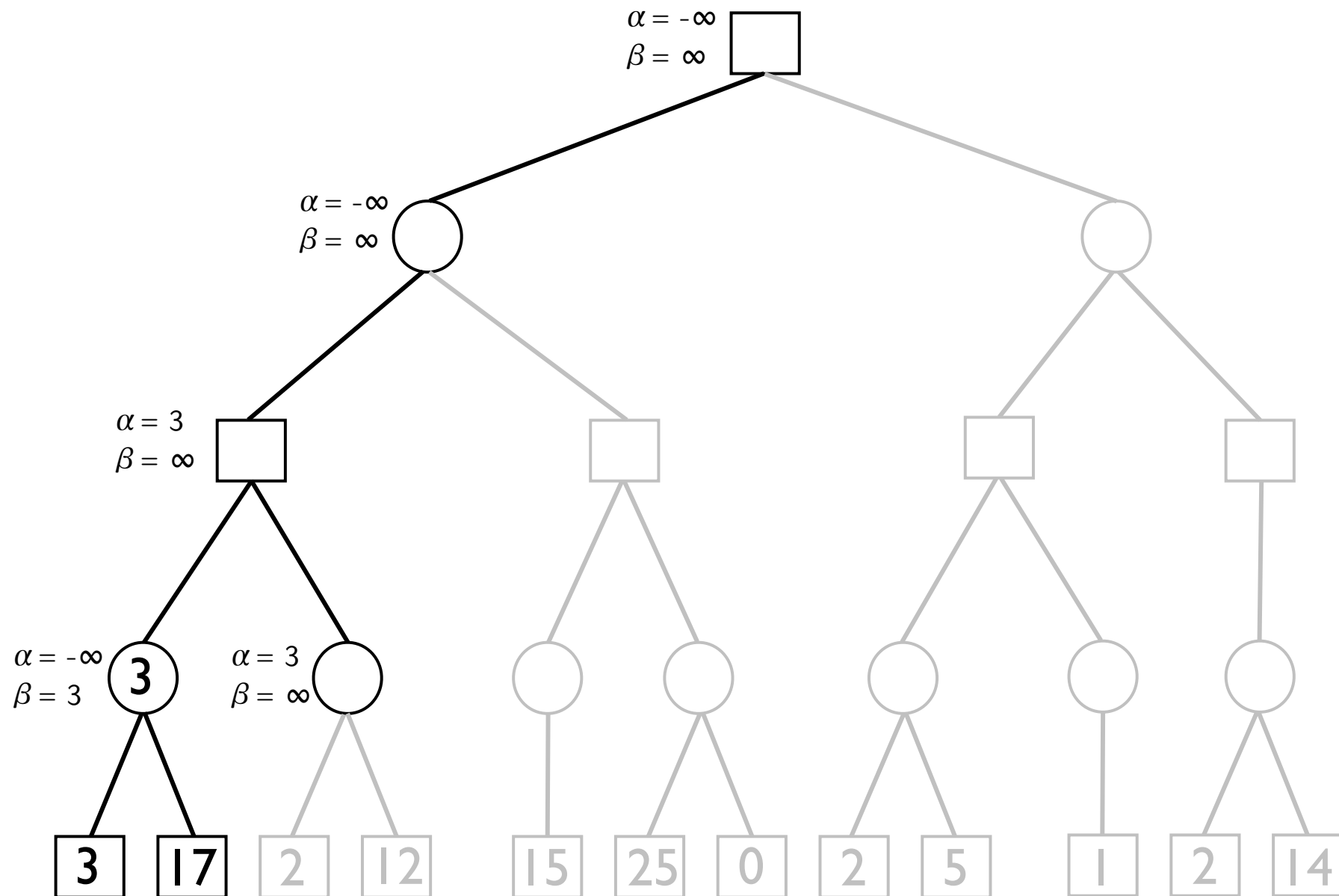
Poda alfa-beta



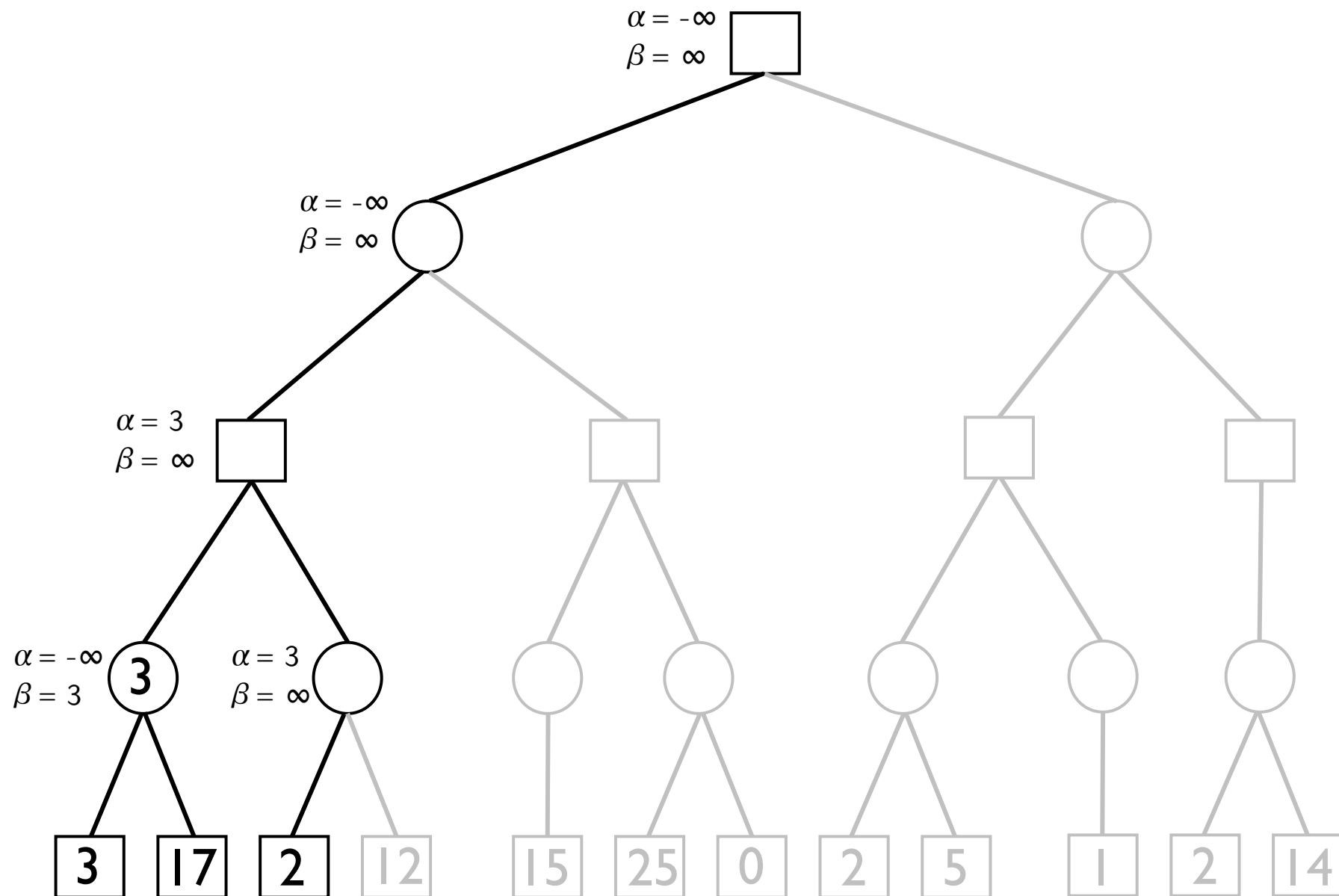
Poda alfa-beta



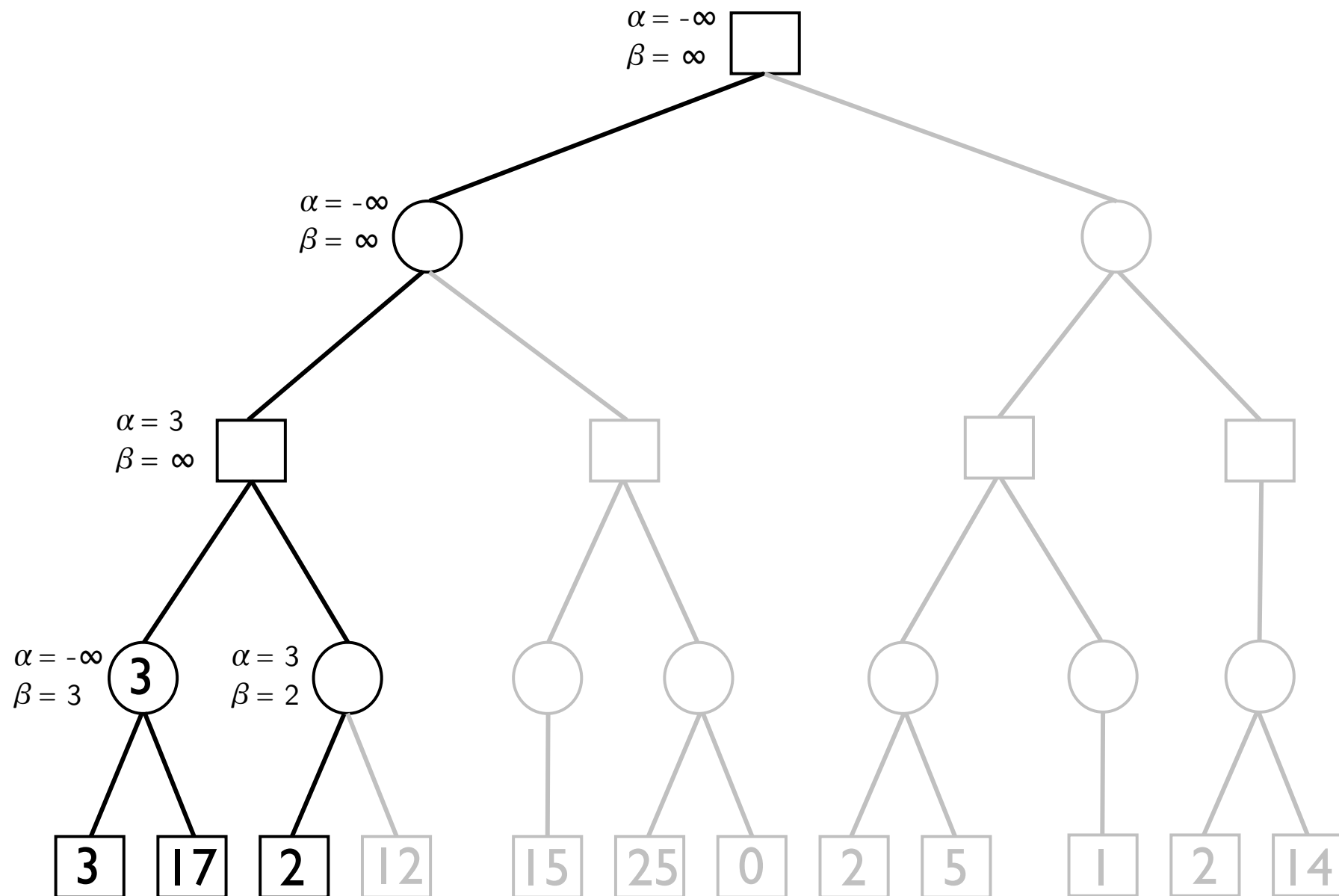
Poda alfa-beta



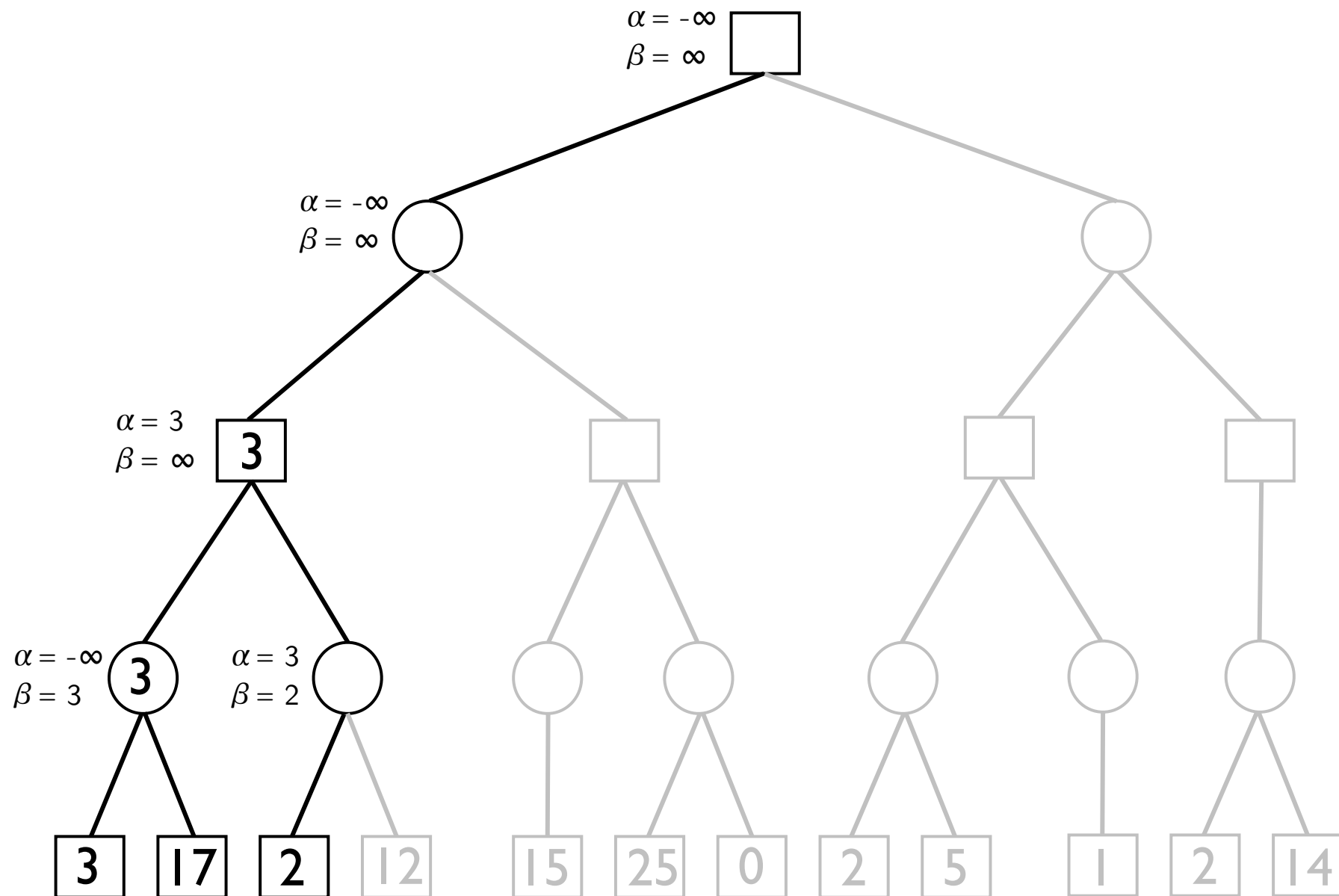
Poda alfa-beta



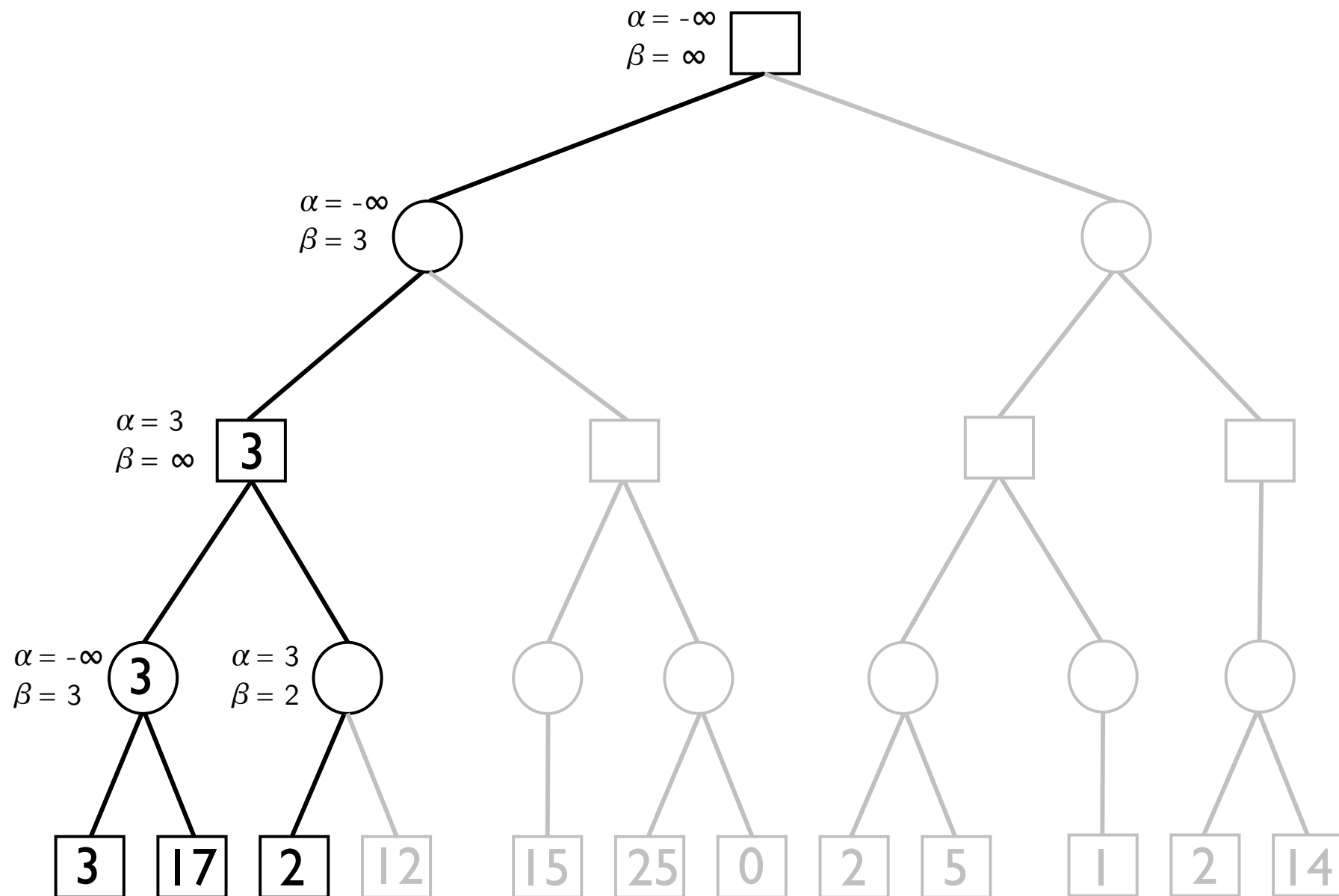
Poda alfa-beta



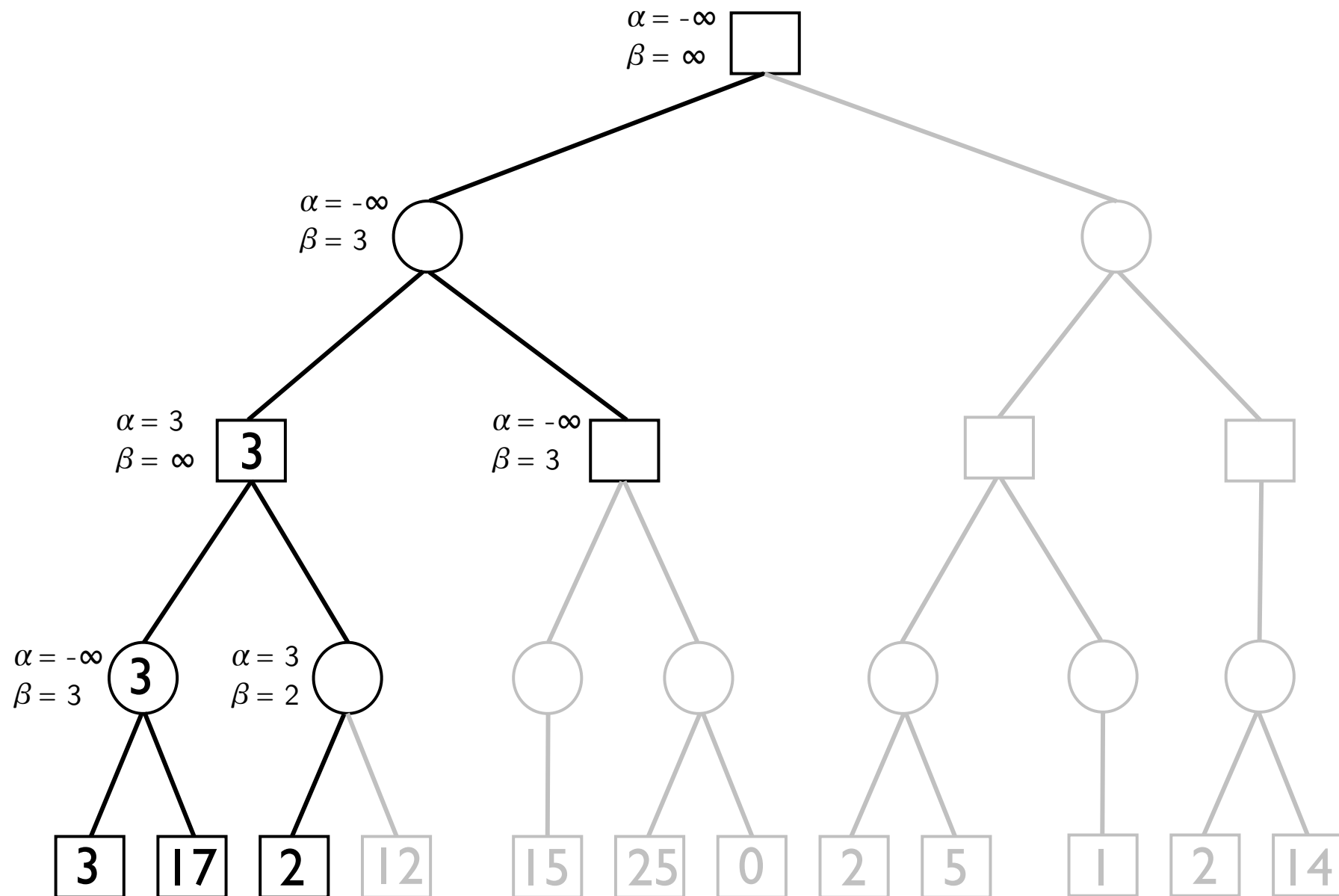
Poda alfa-beta



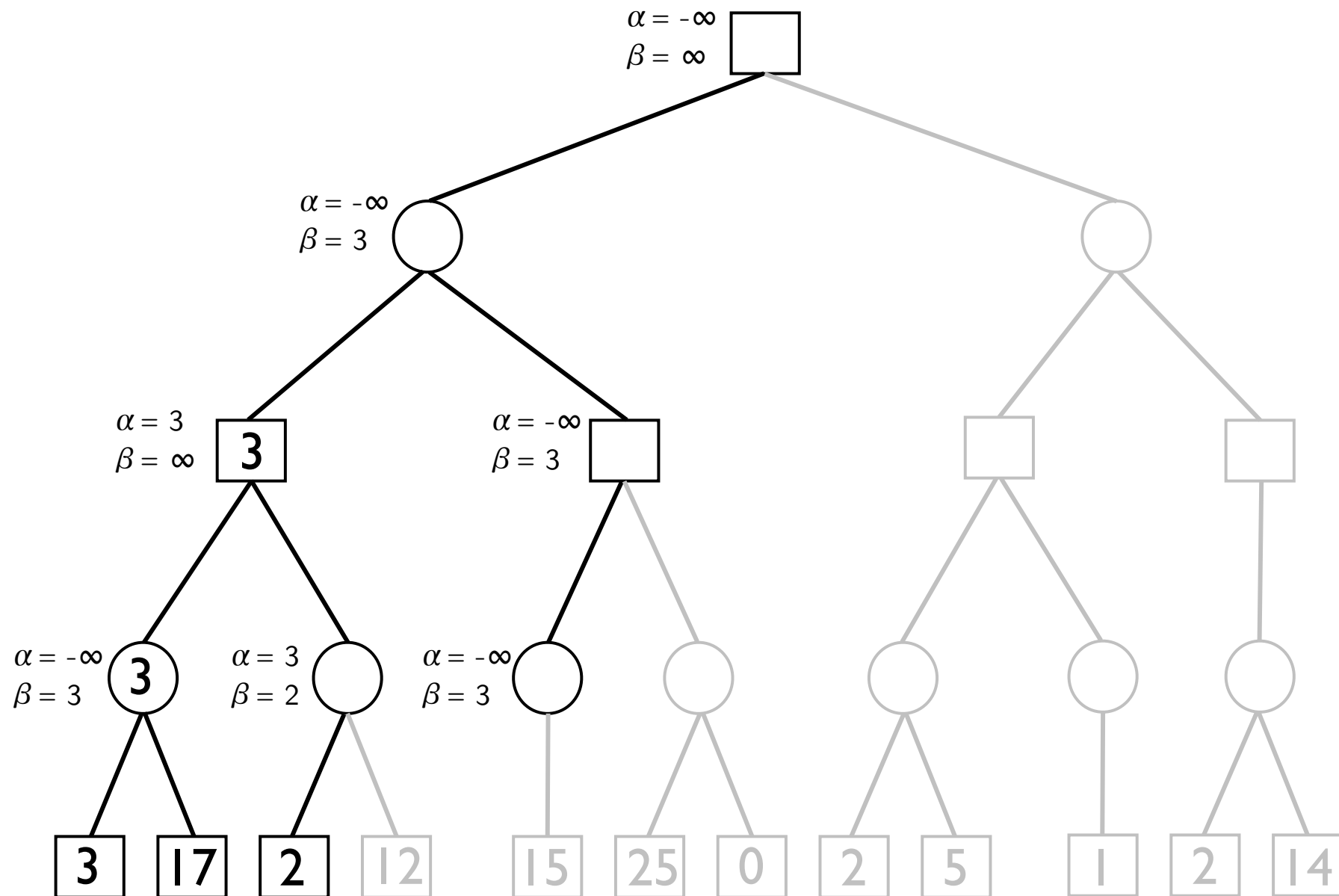
Poda alfa-beta



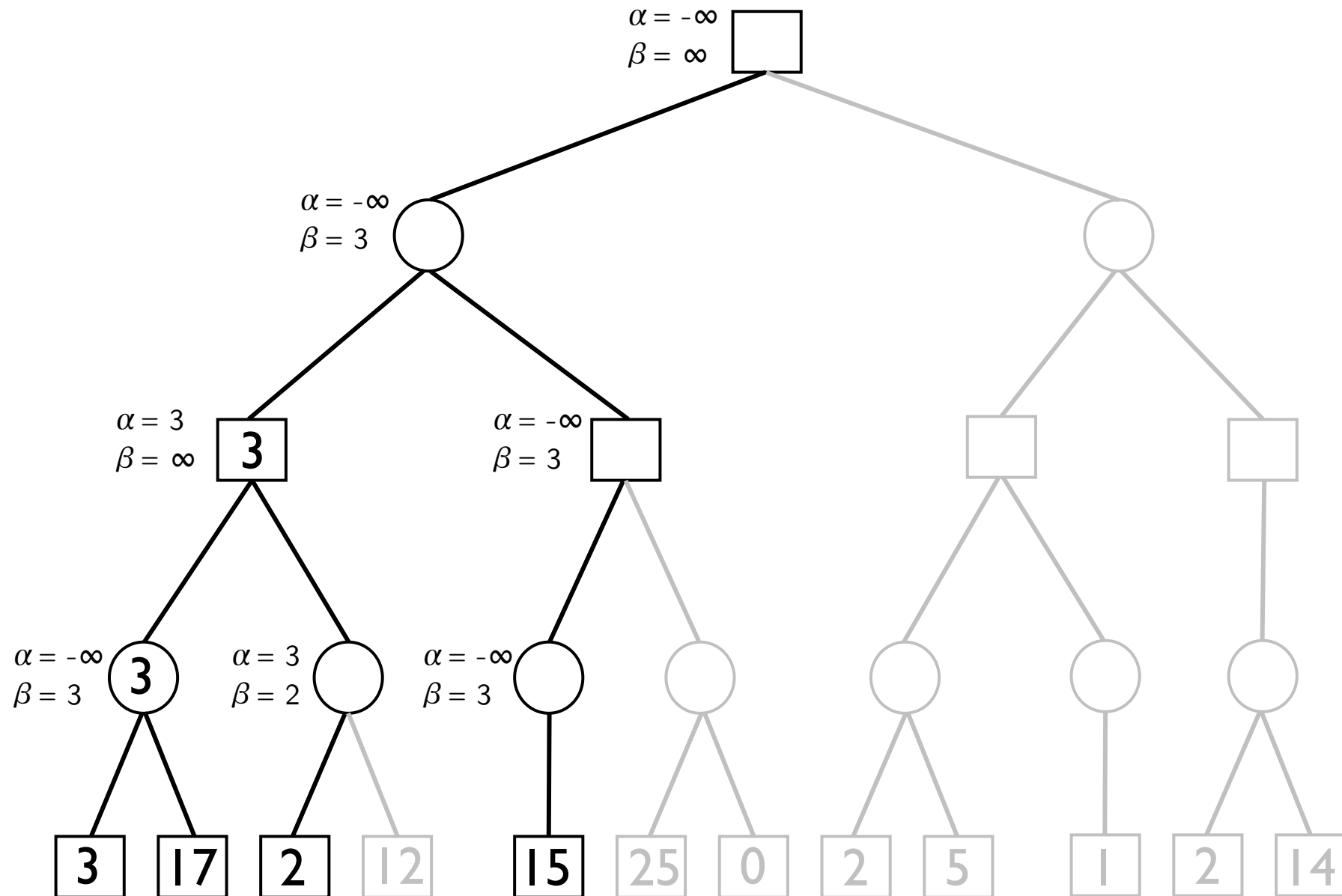
Poda alfa-beta



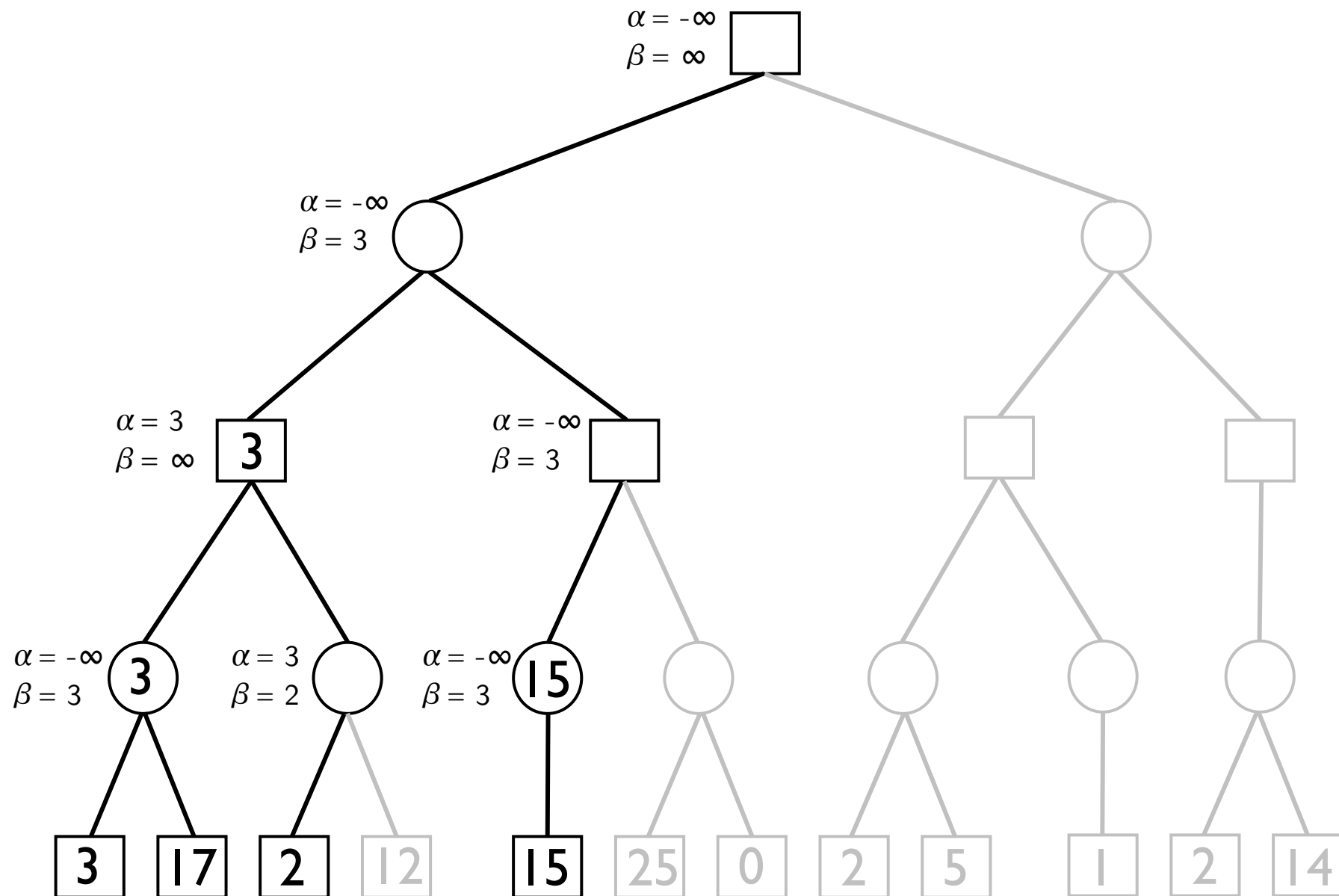
Poda alfa-beta



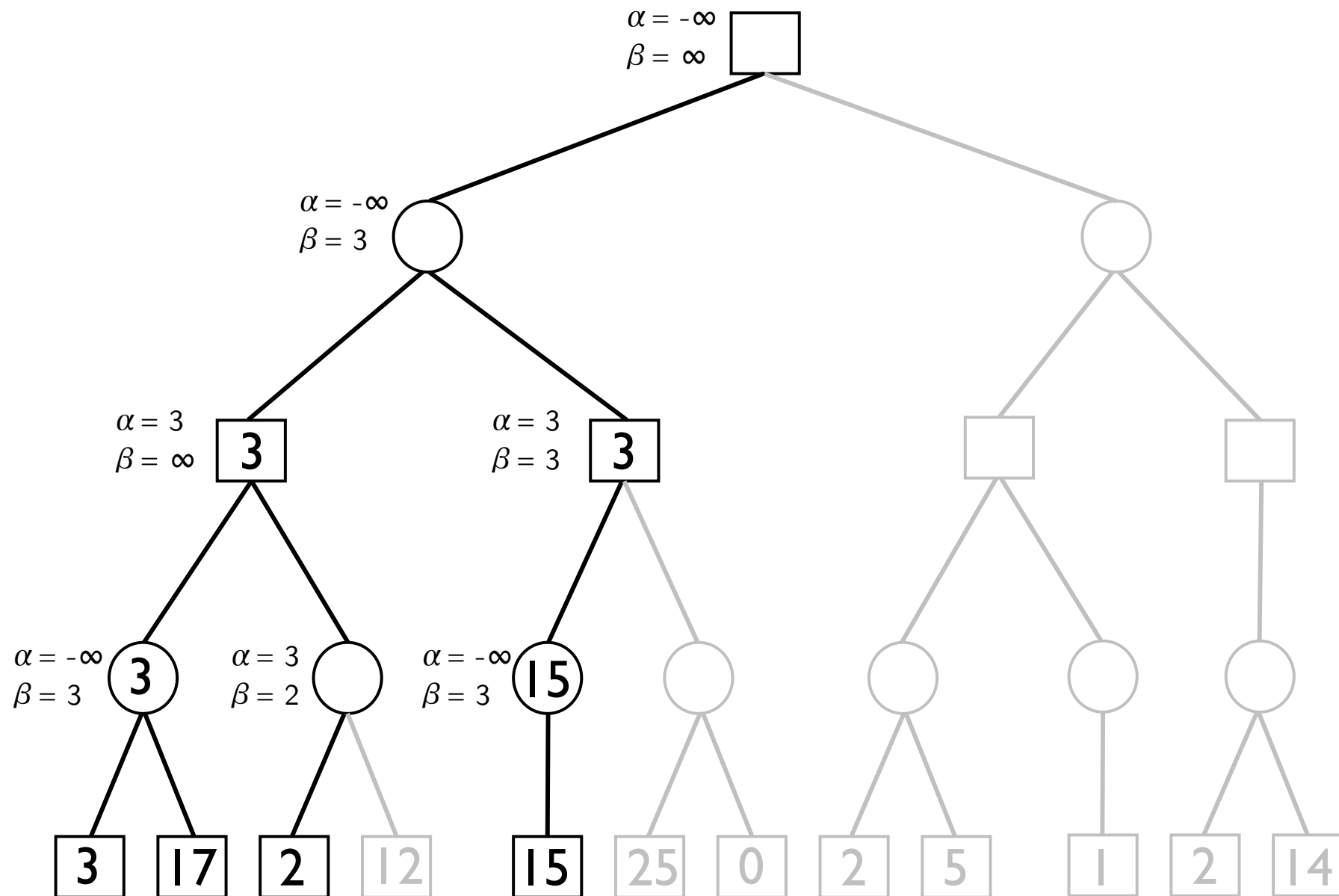
Poda alfa-beta



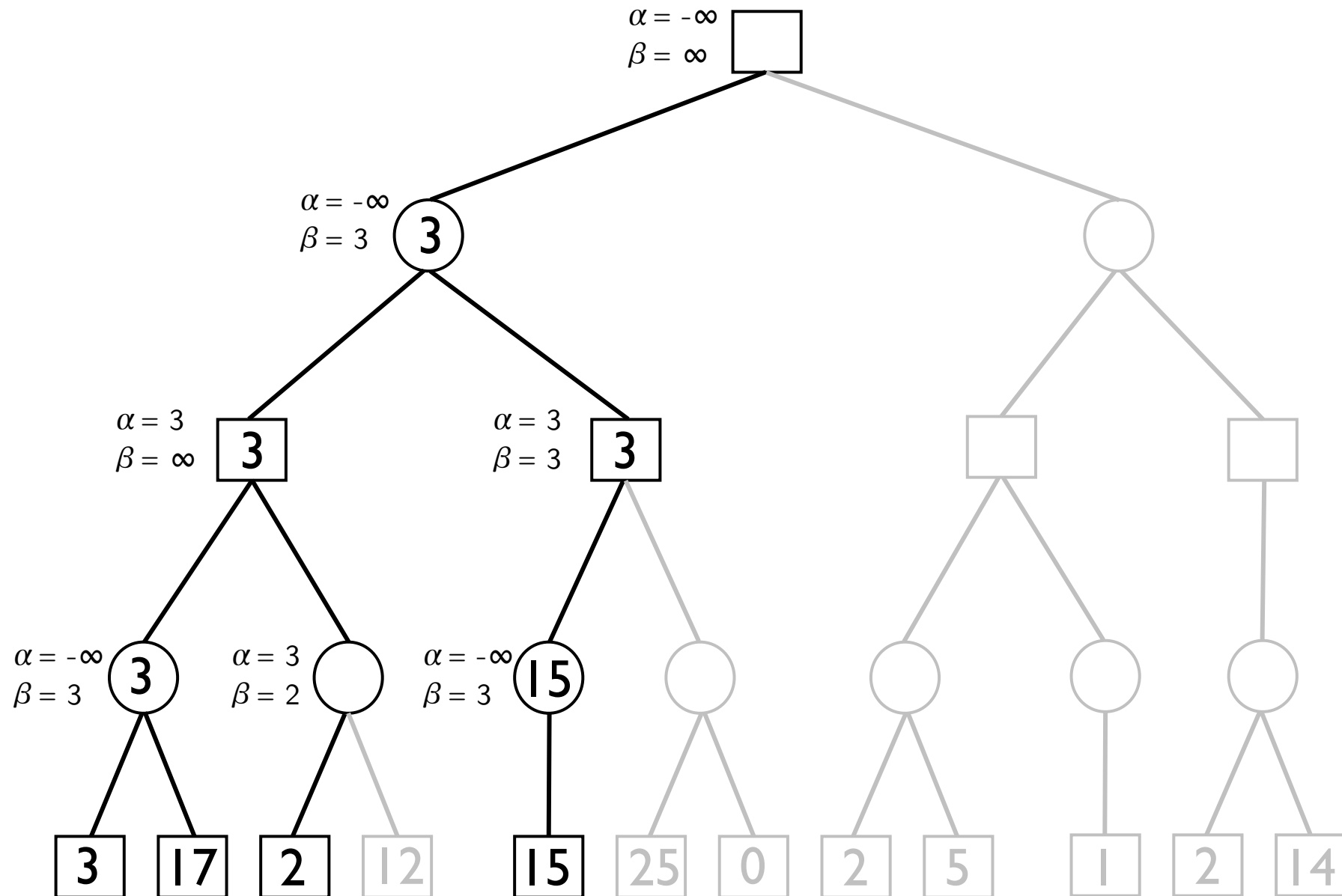
Poda alfa-beta



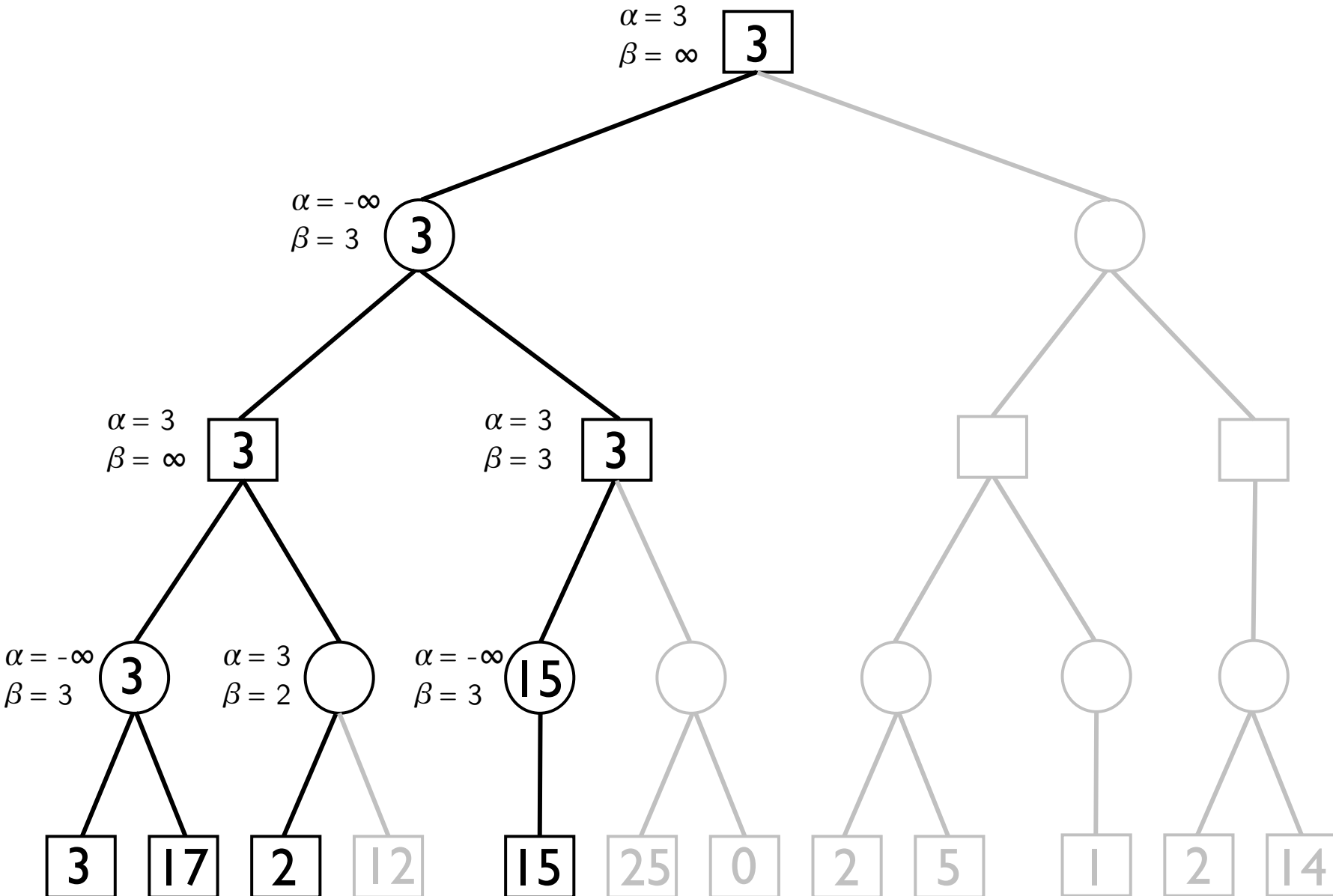
Poda alfa-beta



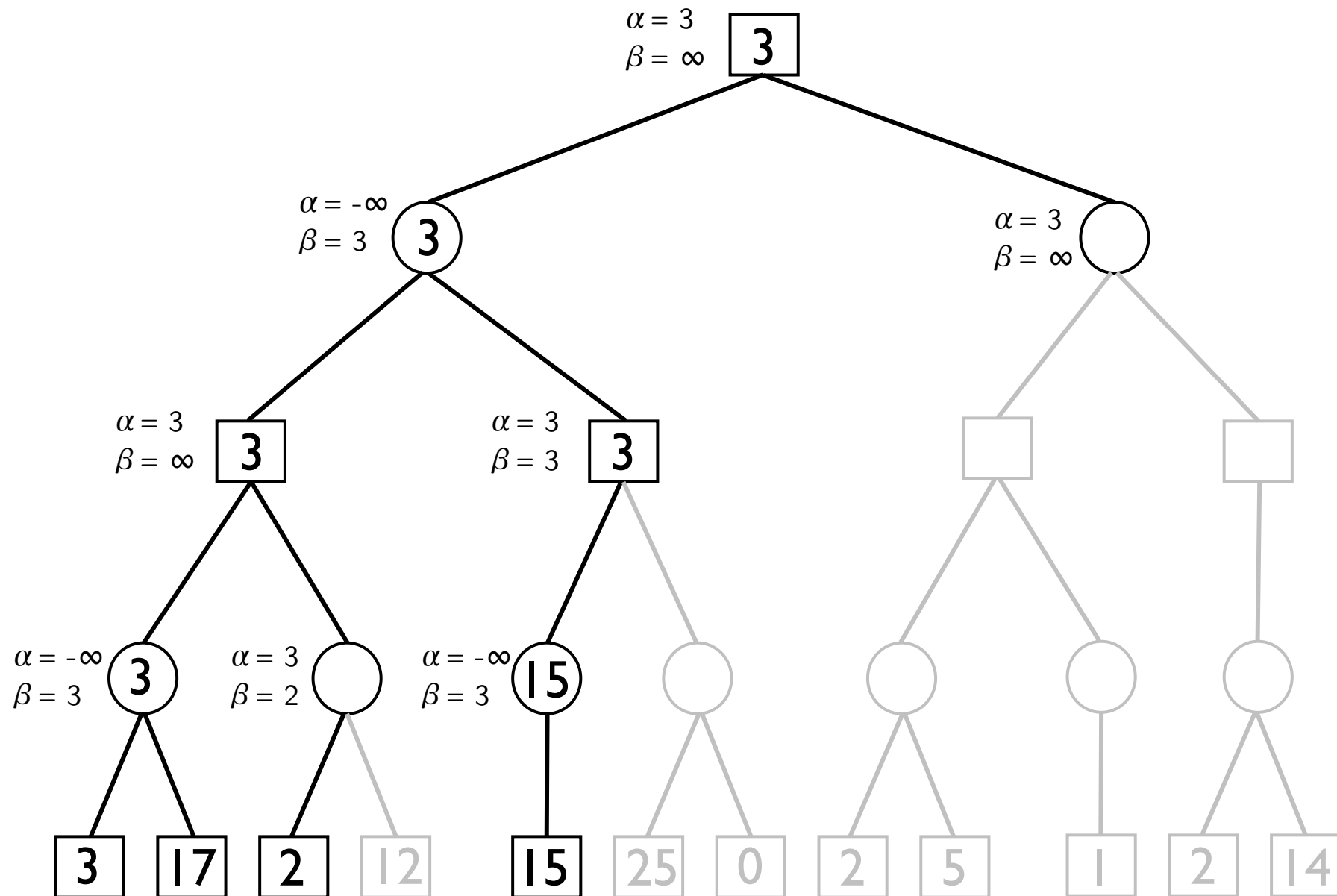
Poda alfa-beta



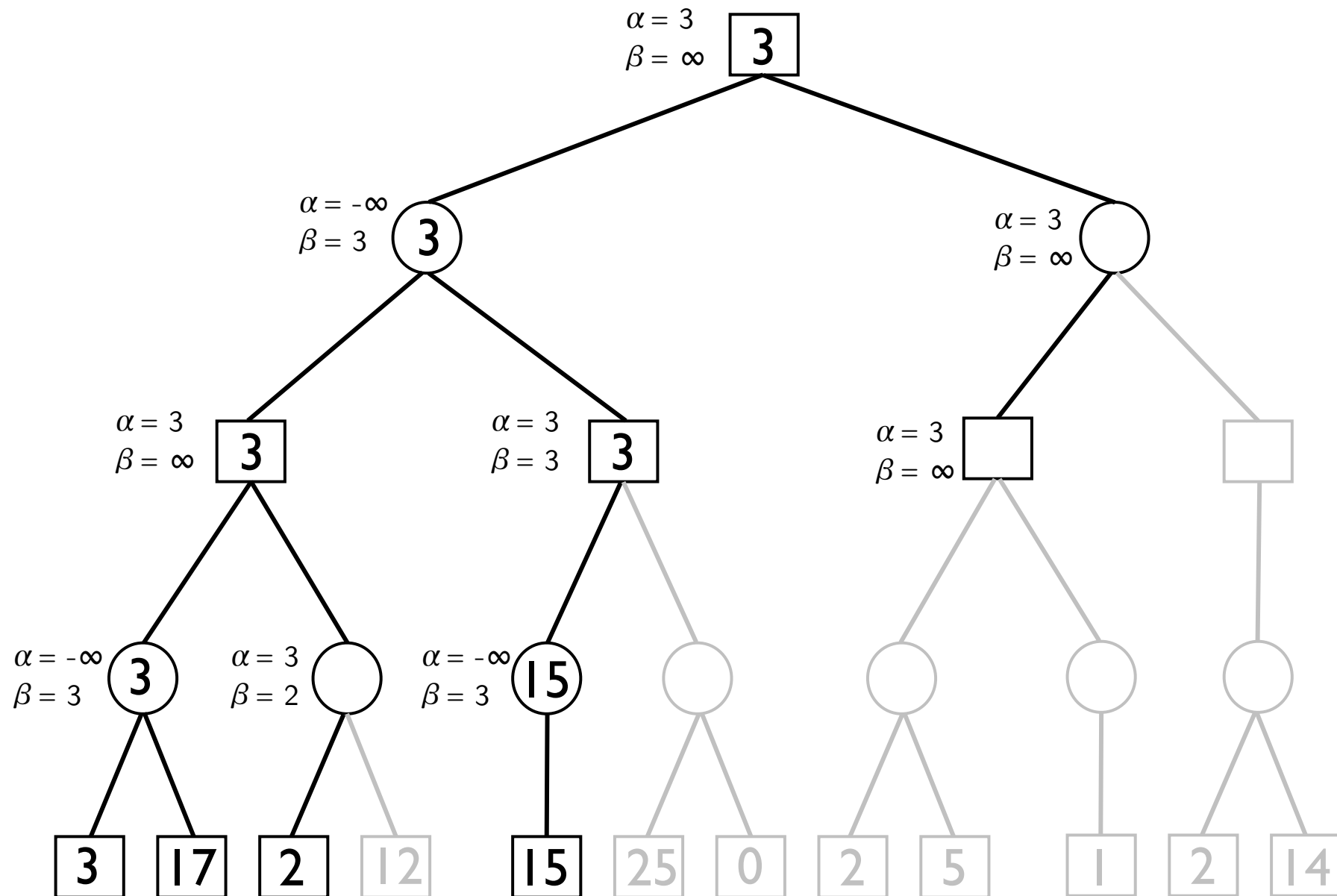
Poda alfa-beta



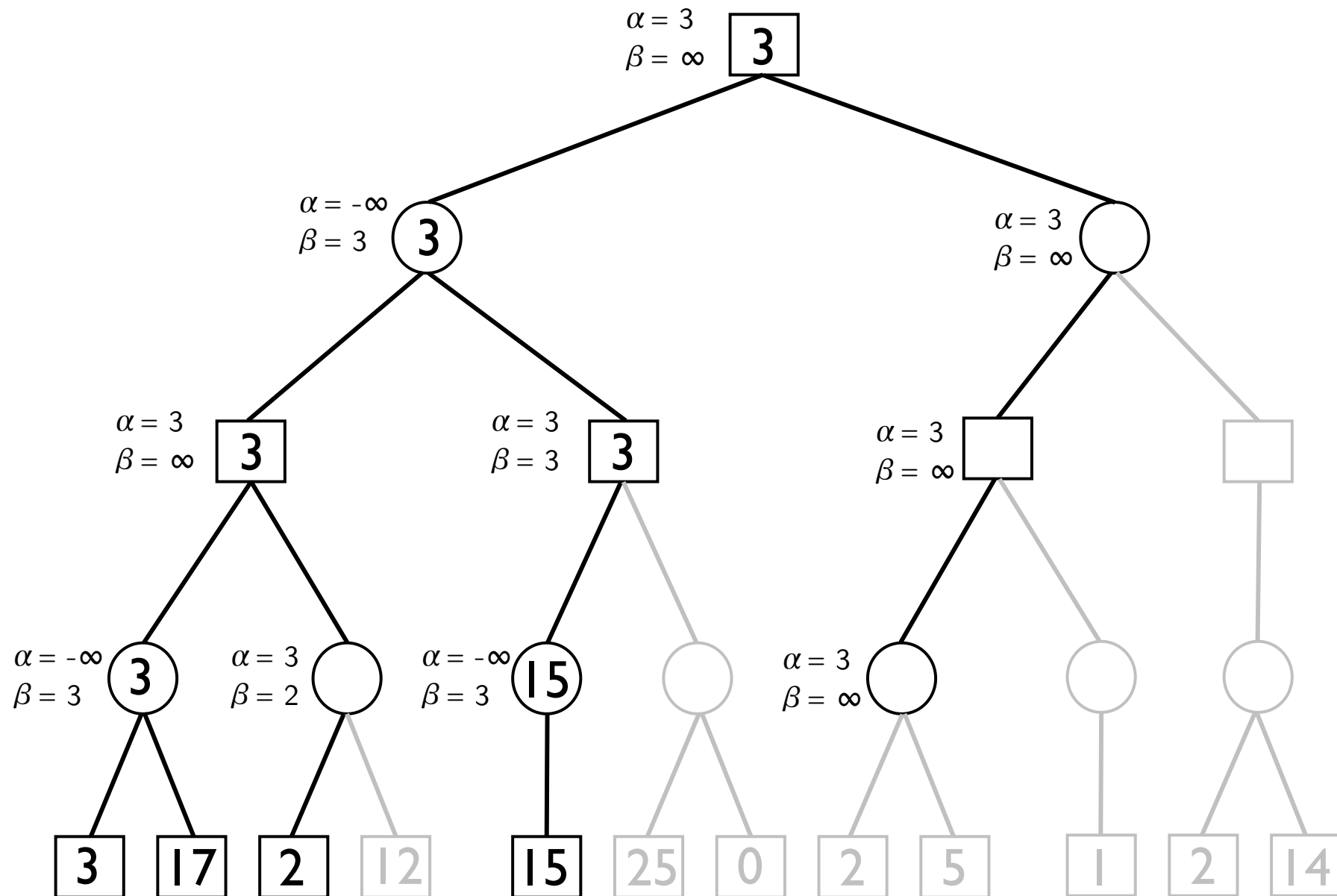
Poda alfa-beta



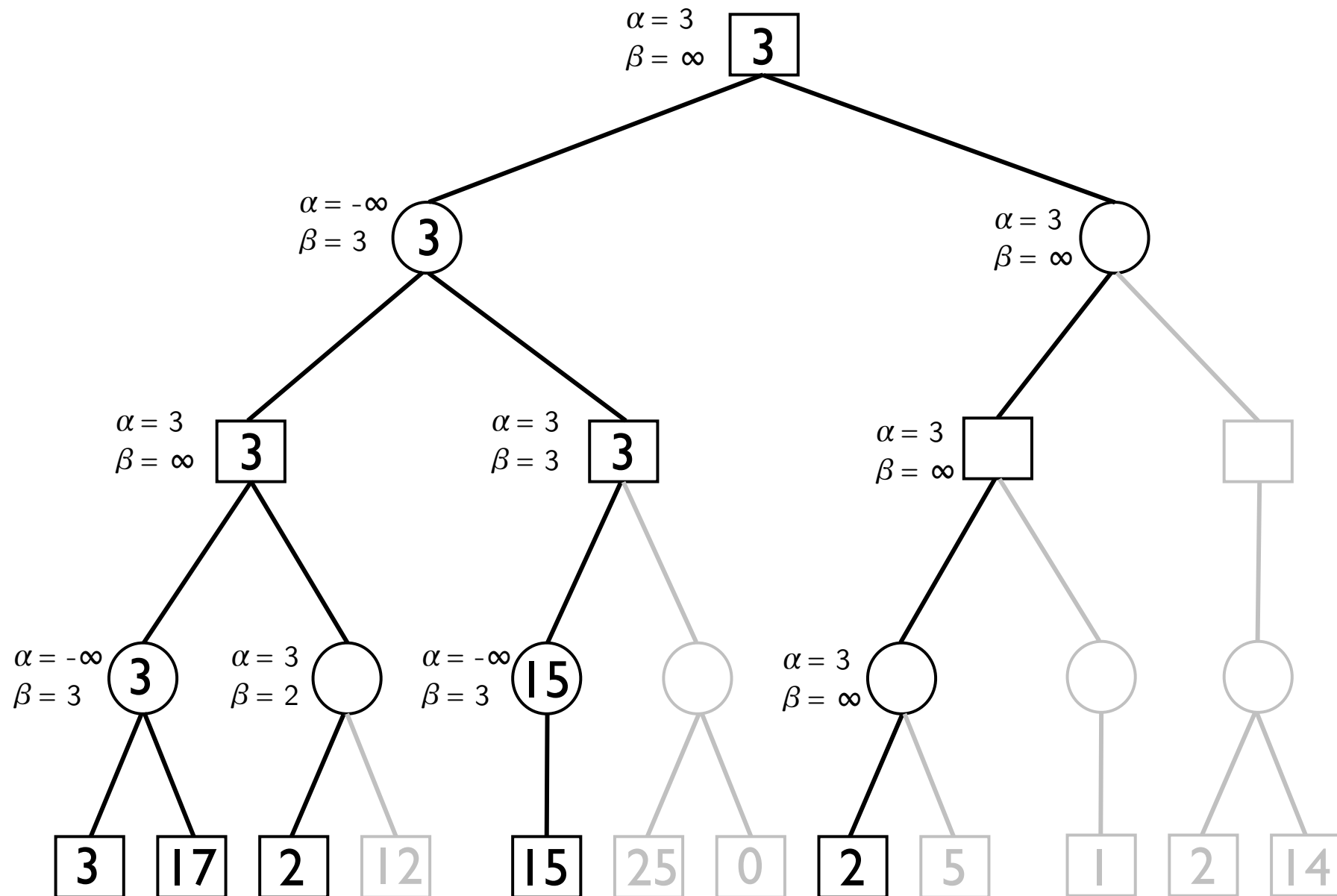
Poda alfa-beta



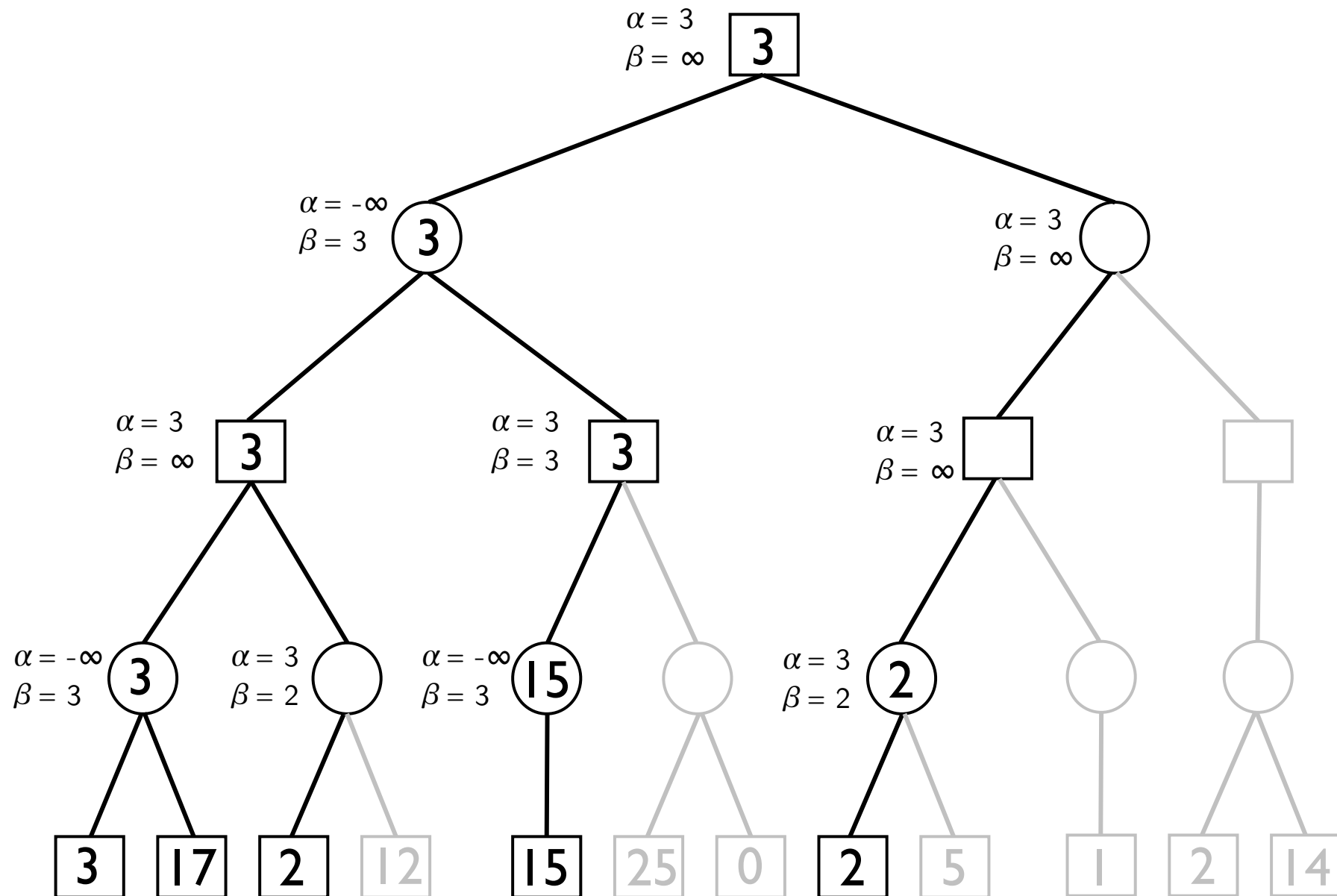
Poda alfa-beta



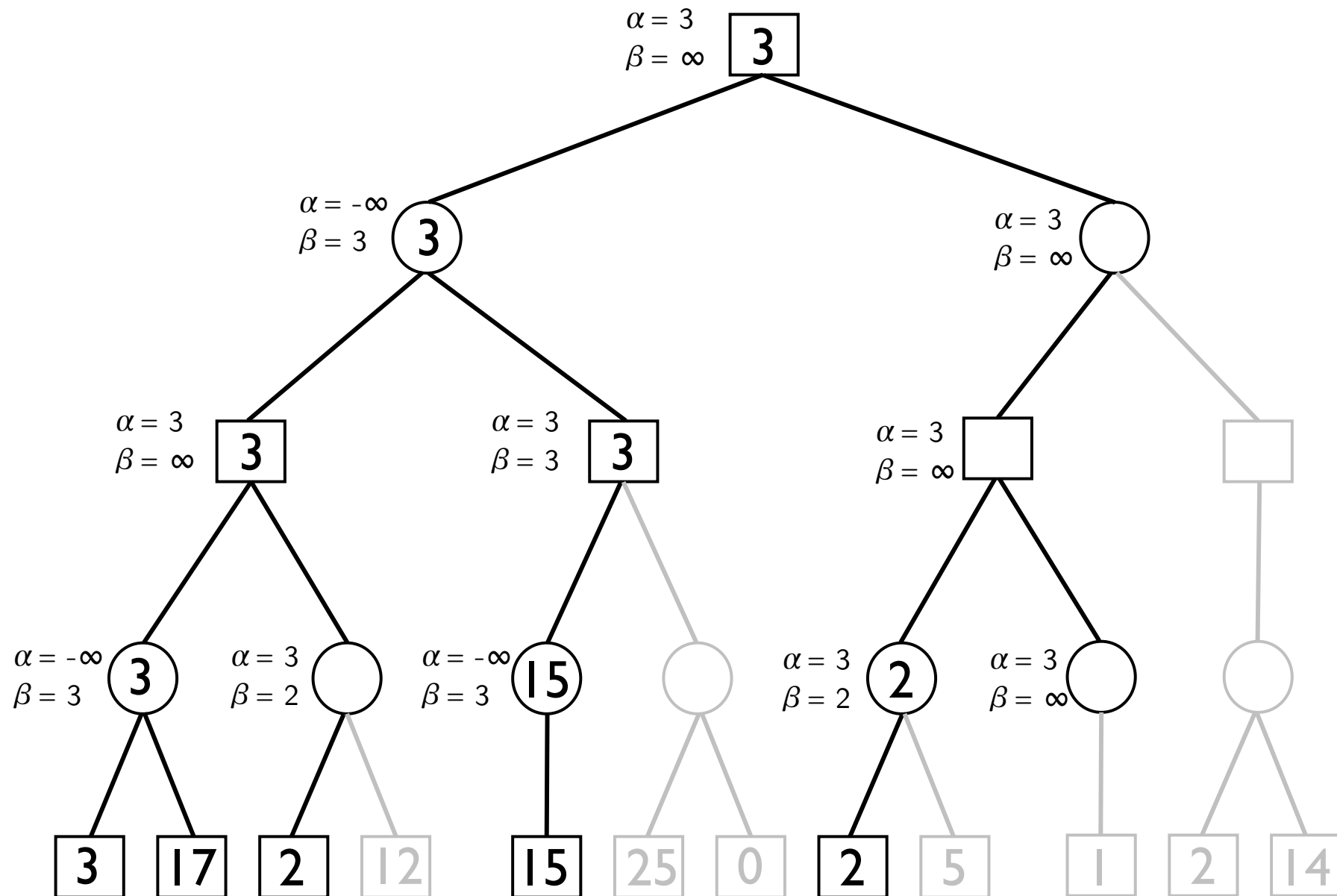
Poda alfa-beta



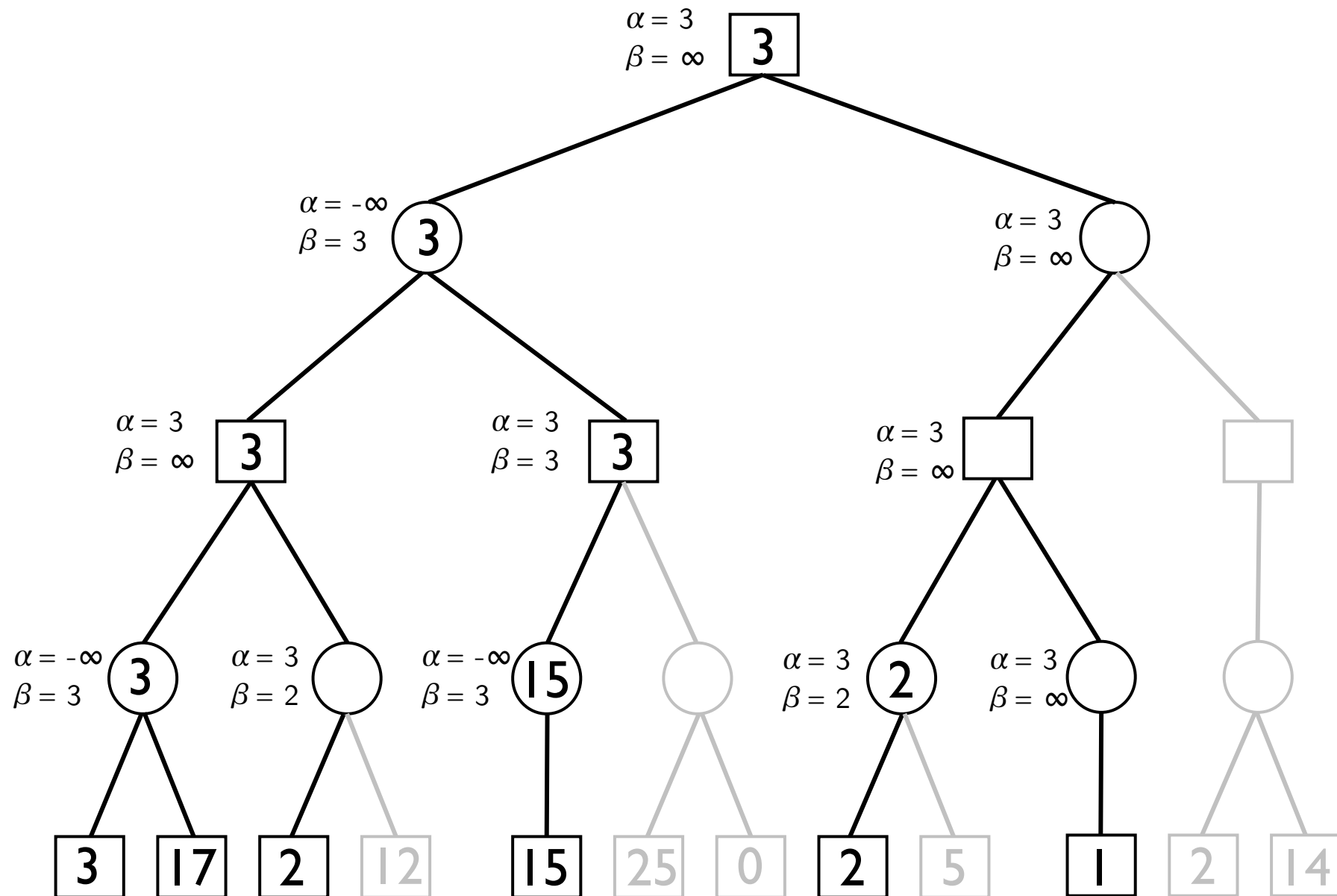
Poda alfa-beta



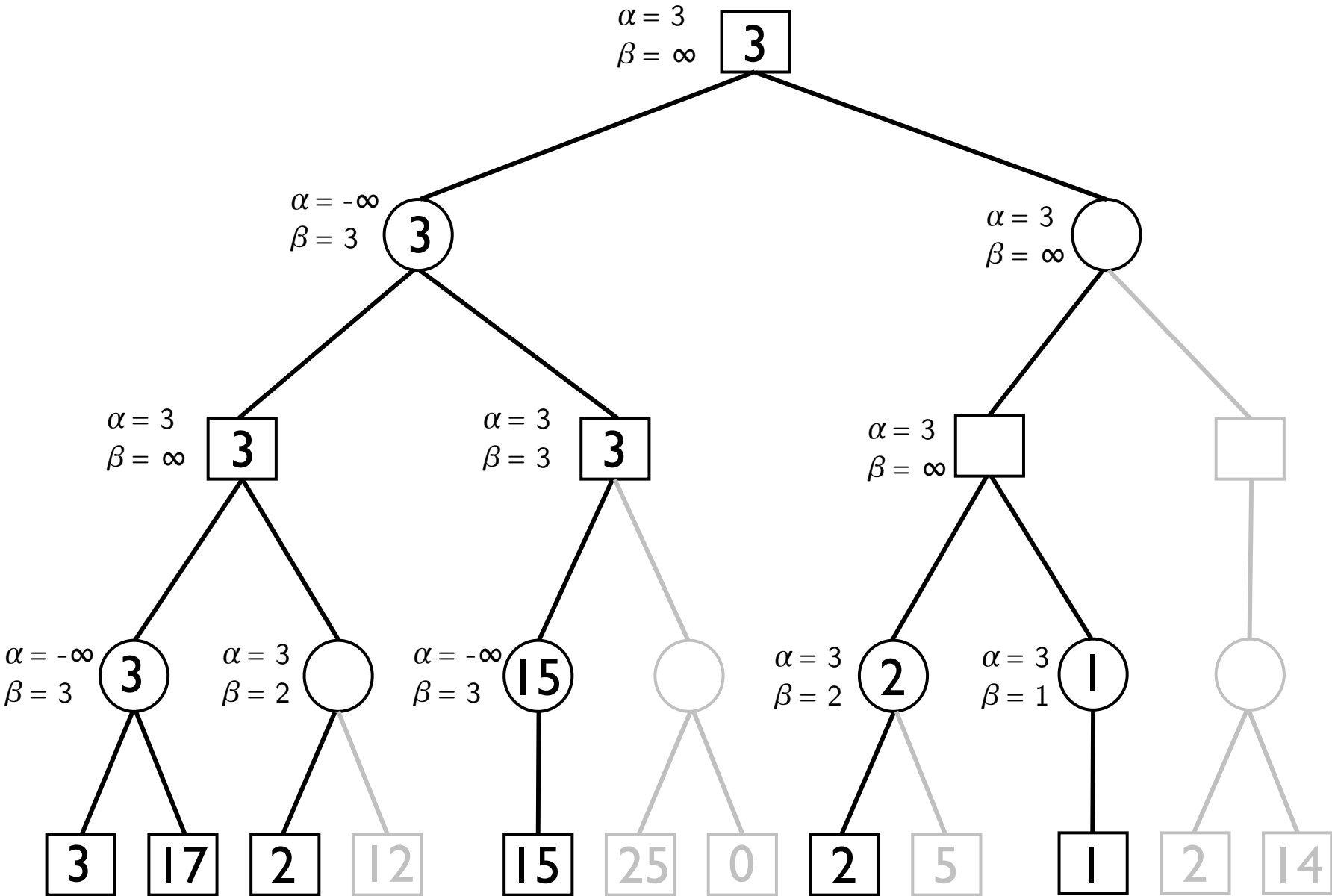
Poda alfa-beta

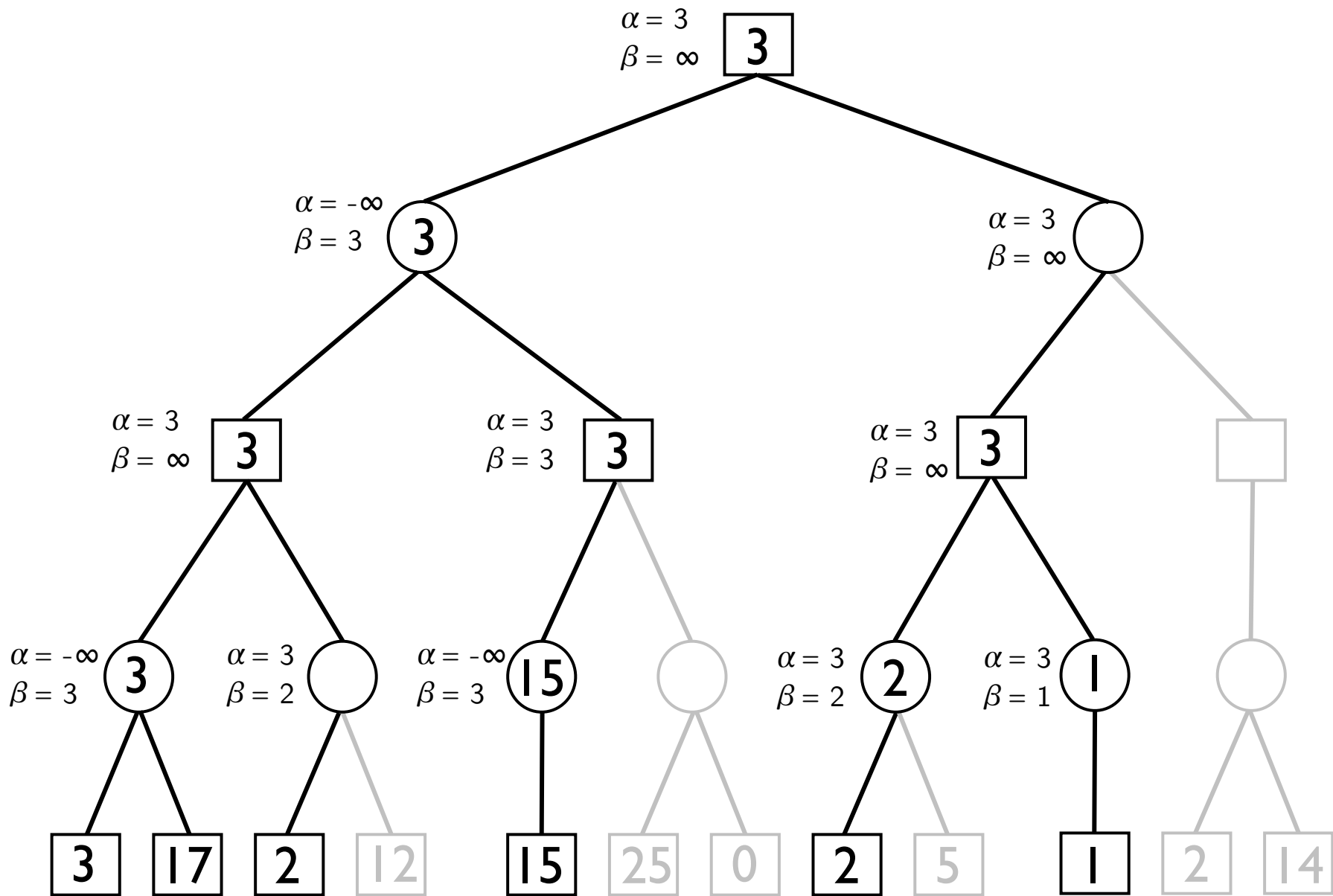


Poda alfa-beta

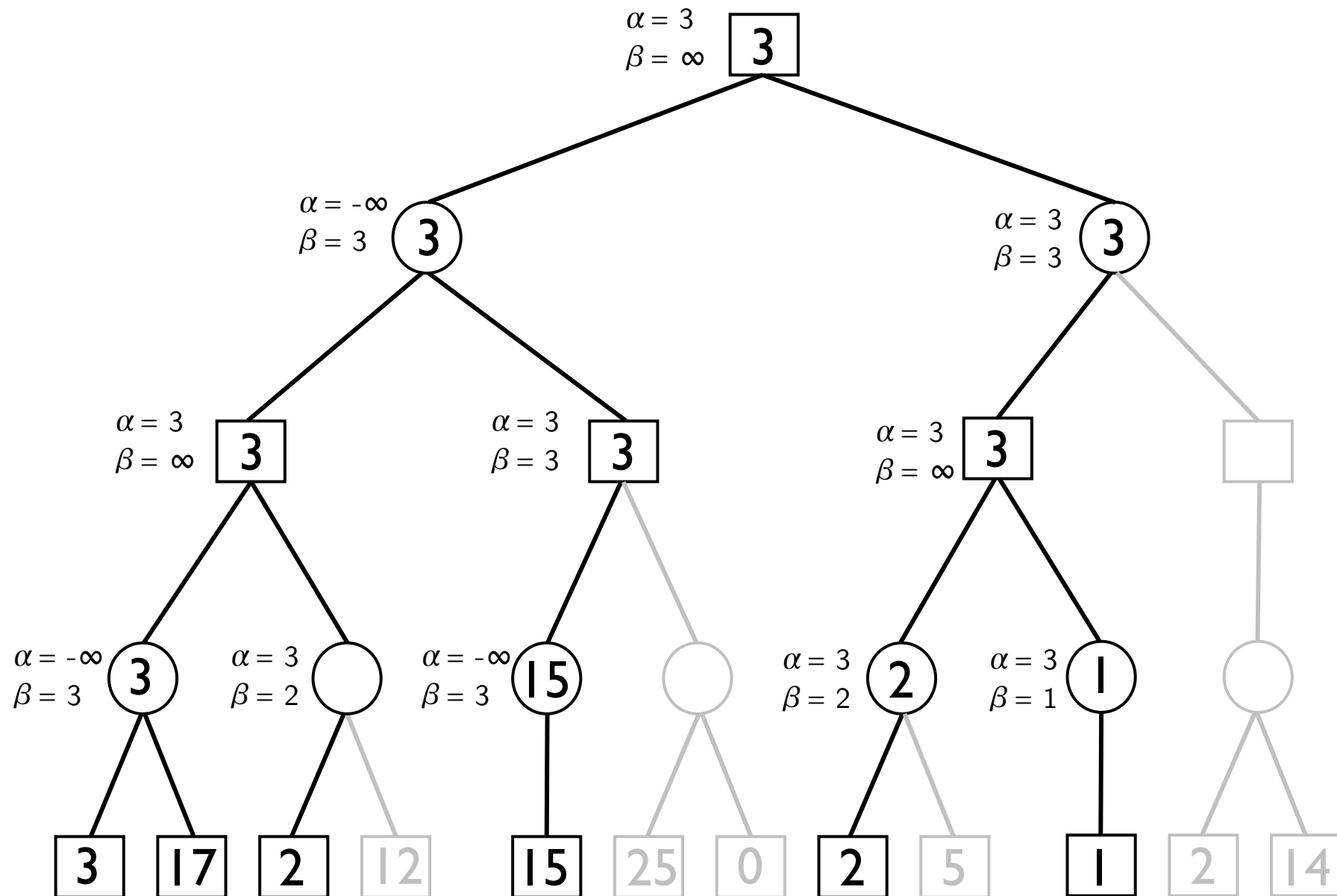


Poda alfa-beta





Poda alfa-beta



Poda alfa-beta

