Intelligenza Artificiale

Anno Accademico 2022 - 2023

Giochi a Due Avversari

Norme di utilizzo dei materiali didattici

La visione e l'utilizzo del presente materiale didattico è riservato agli utenti iscritti al corso al solo fine di studio e approfondimento didattico.

L'utente che accede alla sezione e-learning e che ne consulta i contenuti è tenuto a rispettare le disposizioni legislative che tutelano il diritto d'autore e pertanto è fatto espresso divieto di riprodurre, pubblicare o distribuire i materiali tratti dal presente sito, anche in forma parziale, fatta salva la possibilità di realizzare un'unica copia del materiale, in formato cartaceo e/o digitale, all'esclusivo fine di studio.

L'utente è responsabile per l'uso improprio o non autorizzato del materiale pubblicato effettuato in violazione delle suddette disposizioni.

Sommario

- Giochi a due avversari
- Algoritmo Mini-Max
- Potatura Alfa-Beta

Giochi a due avversari (1)

1950: Shannon e Turing hanno realizzato i primi programmi per giocare a scacchi.

La presenza di un **avversario** rende il problema più complicato, rispetto ai problemi di search, poichè si inserisce un fattore di **incertezza** dovuto alla mancanza di conoscenza sulle mosse che farà l'avversario.

Problema della contingenza ma l'incertezza non è casuale

Giochi a due avversari (2)

La causa principale di incertezza è dovuta alla

Dimensione dei problemi : impossibilità di esaminare tutto l'albero. Necessità di scegliere la mossa migliore senza conoscerne tutte le possibili conseguenze.

Scacchi:
b=35; numero mosse=50 $\Rightarrow$ 35^{100} nodi (10^{40} posizioni legali diverse)

Come usare al meglio il tempo a disposizione quando non è possibile prendere decisioni ottime?

Giochi a due avversari (3)

Un gioco può essere formalmente definito come un problema di ricerca:

Stato Iniziale : posizione sulla scacchiera, turno.

Operatori : realizzano le mosse legali.

Goal-Test : determina la fine del gioco (stati terminali).

Funzione di utilità : attribuisce un valore numerico agli stati terminali.

Algoritmo MINIMAX per decisioni perfette (1)

Trovare la mossa teoricamente migliore, senza considerare limitazioni di tempo.

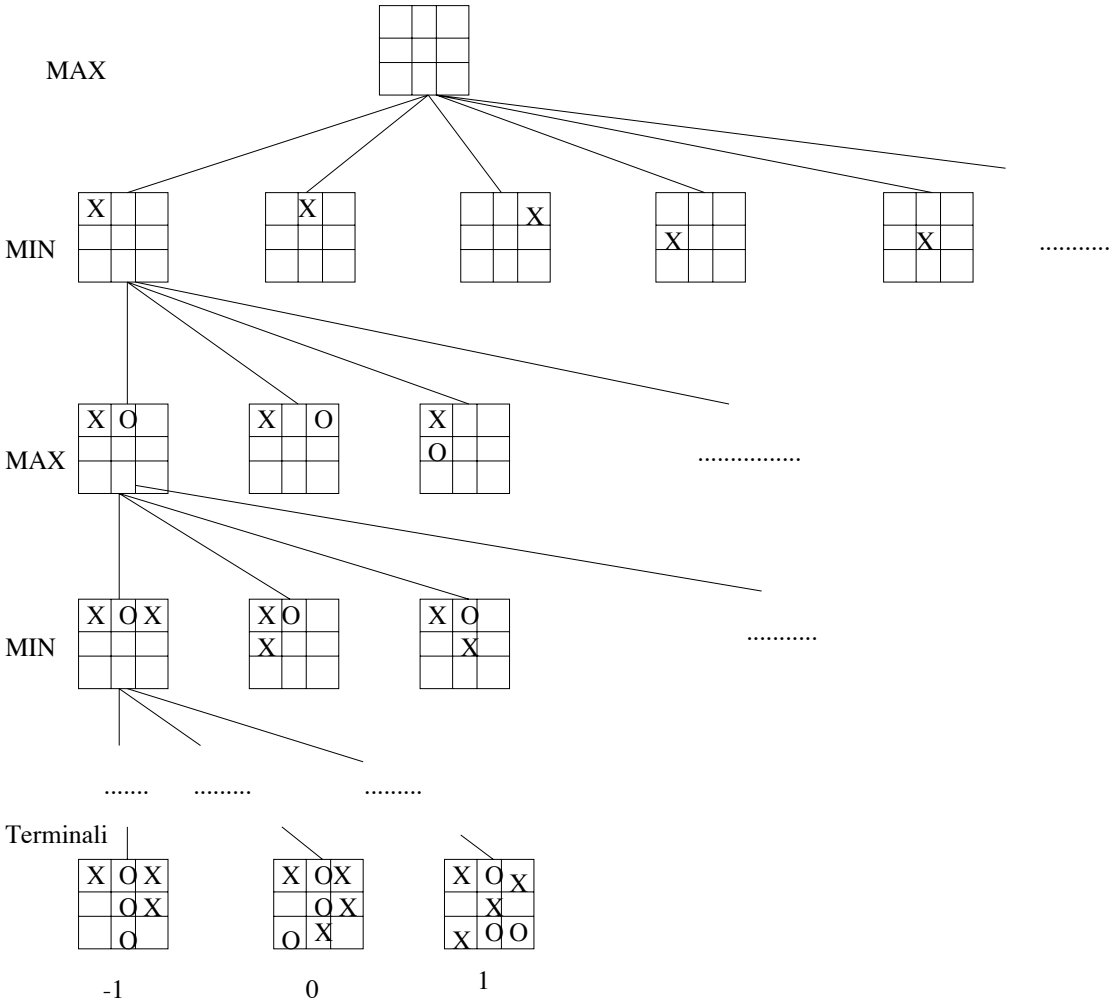
Massimizzare la funzione di utilità, sotto l'ipotesi che l'avversario giochi perfettamente cercando di minimizzarla.

Supponiamo di avere due giocatori: Max e MinMax muove per primo.

La ricerca della mossa migliore da parte di Max deve tener conto delle possibili contromosse di Min.

Max deve trovare una **strategia** che permetterà la vittoria indipendentemente dalle mosse di Min: un algoritmo che dia la mossa migliore di Max per ciascuna mossa possibile di Min.

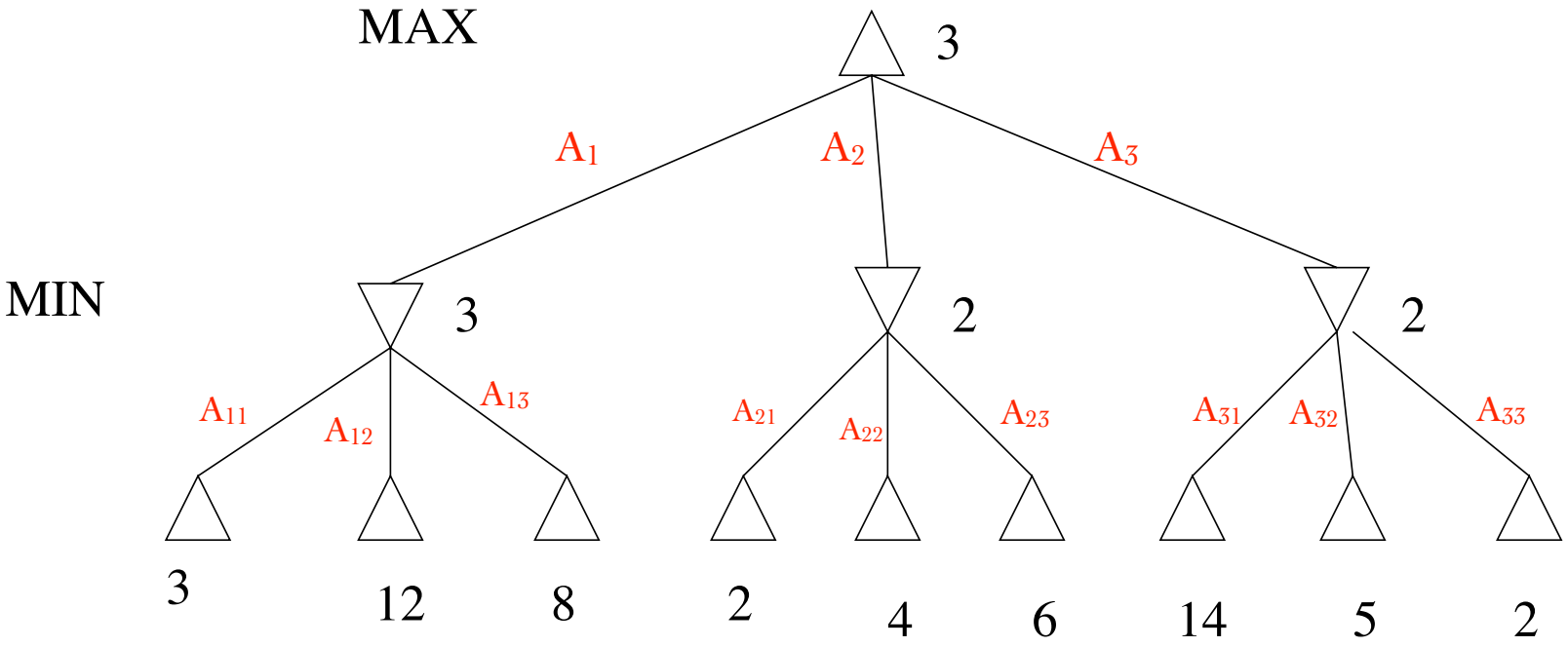
Algoritmo MINIMAX per decisioni perfette (2)



Procedura del MINIMAX su un albero completo (1)

- generare tutto l'albero di gioco
- determinare il valore delle foglie applicando la funzione di utilità
- propagare i valori ai genitori:
  - il valore di un nodo MIN (turno di MIN) è il minimo tra i valori dei figli
  - il valore di un nodo MAX (turno di MAX) è il massimo tra i valori dei figli
- la radice corrisponde a un turno di MAX: sceglie la mossa che porta al valore più alto

Procedura del MINIMAX su un albero completo (2)



Algoritmo Mini-Max (1)

```
function MINIMAX-DECISION(game) returns an operator
for each op in OPERATORS(game) do
  VALUE[op] ← MINIMAX-VALUE(APPLY(op, game), game)
end
return op with max VALUE[op]
end

function MINIMAX-VALUE(state, game) returns a utility value
if TERMINAL-TEST[game](state) then return UTILITY[game](state)
else if MAX is to move in state
      then return the highest MINIMAX-VALUE of SUCCESSORS(state)
      else return the lowest MINIMAX-VALUE of SUCCESSORS(state)
end
```

Algoritmo Mini-Max (2)

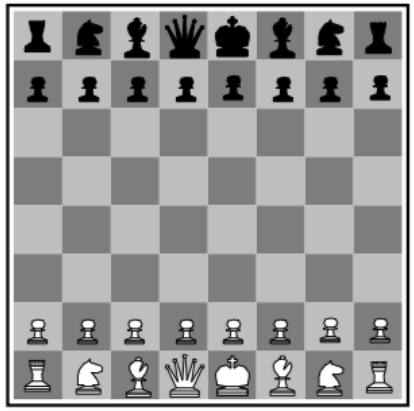
Complessità in tempo : $O(b^m)$
Profondità = m; Fattore di ramificazione = b.

Complessità in spazio : $O(bm)$ perchè la procedura si sviluppa in profondità.

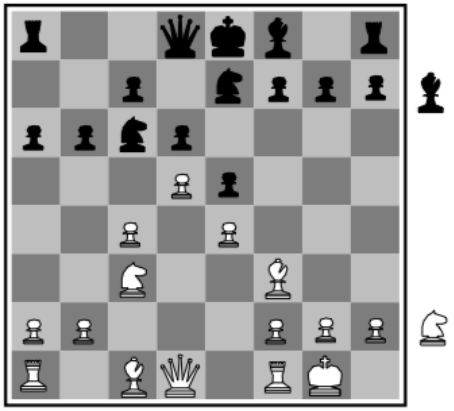
Algoritmo Mini-Max per decisioni imperfette (1)

- Anziché costruire l'albero fino agli stati terminali, si costruisce fino a una profondità fissata:
  al posto del Terminal-Test: Cut-Off Test
- Per assegnare un valore alle foglie dell'albero, al posto dell funzione di utilità Utility, si utilizza una funzione di valutazione **euristica** Eval

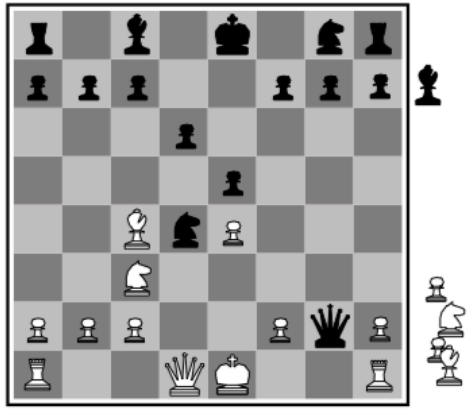
Esempio per gli scacchi



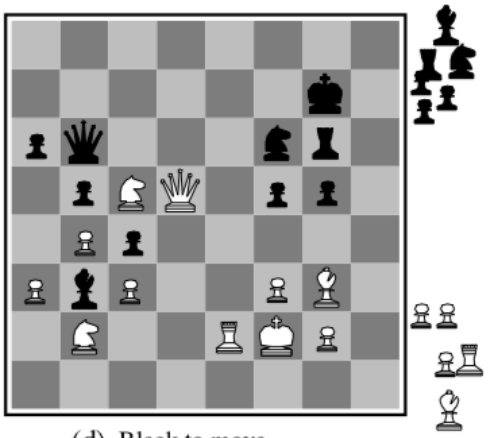
(a) White to move
Fairly even



(b) Black to move
White slightly better



(c) White to move
Black winning



(d) Black to move
White about to lose

Algoritmo Mini-Max per decisioni imperfette (2)

Scelta di Eval

Eval fornisce una stima dell'utilità di una certa posizione di gioco:

- Deve riflettere l'effettiva possibilità di vittoria
- Il suo calcolo non deve essere troppo costoso
- Eval(S)=Utility(S) se S è uno stato terminale

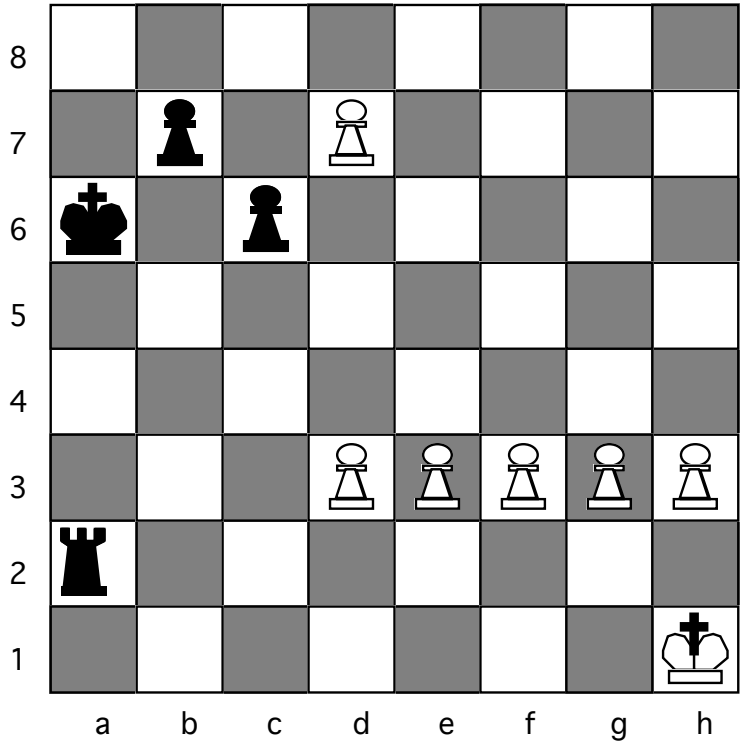
Esempi:

- Negli scacchi si può usare la somma dei "valori materiali" dei pezzi (pedoni: 1, alfiere: 3, torre: 5,...) + buona struttura dei pedoni, ecc.
- Nel filetto: se p non è vincente, allora $eval(p)$ = (numero di righe, colonne e diagonali aperte per MAX) - (numero di righe, colonne e diagonali aperte per MIN); altrimenti $+\infty$ o $-\infty$

Problemi nel tagliare la ricerca

- Posizioni "**non quiescenti**"
  - La funzione di valutazione dovrebbe essere applicata solo alle posizioni che sono **quiescenti**, ossia che è improbabile che manifestino delle notevoli oscillazioni di valore nel futuro immediato.
- Problema dell'**orizzonte**
  - Questo problema è più difficile da eliminare. Si presenta quando il programma si trova di fronte a una mossa dell'avversario che causa seri danni ed è del tutto inevitabile.

Il problema dell'Orizzonte



Algoritmo Mini-Max per decisioni imperfette (3)

Scelta di Cut-Off

- Scegliere una profondità fissata tale da usare al meglio il tempo a disposizione
- Applicare la tecnica degli approfondimenti iterativi. Efficacia a volte disastrosa!!! (es.: scacchi)
- **Ricerca di quiescenza**: applicare la funzione di valutazione solo a posizioni **quiescenti** (che probabilmente non avranno notevoli oscillazioni di valore).
  Le posizioni non quiescenti vengono invece espanse.

Potatura di rami (1)

Scacchi: tempo=150s; efficienza=1000 pos/s; $\Rightarrow$ 150.000 posizioni
fattore di ramificazione $\simeq$ 35; $\Rightarrow$ 3-4 strati $\Rightarrow$ Principiante

Se la valutazione delle posizioni avviene contemporaneamente alla costruzione dell'albero, è possibile potare dei rami: calcolare la decisione MiniMax senza esaminare tutti i nodi dell'albero (**pruning**).

Mentre si esegue la ricerca in profondità ad ogni nodo viene assegnato un valore provvisorio VP, prima di aver generato ed esaminato tutti i suoi sottoalberi.

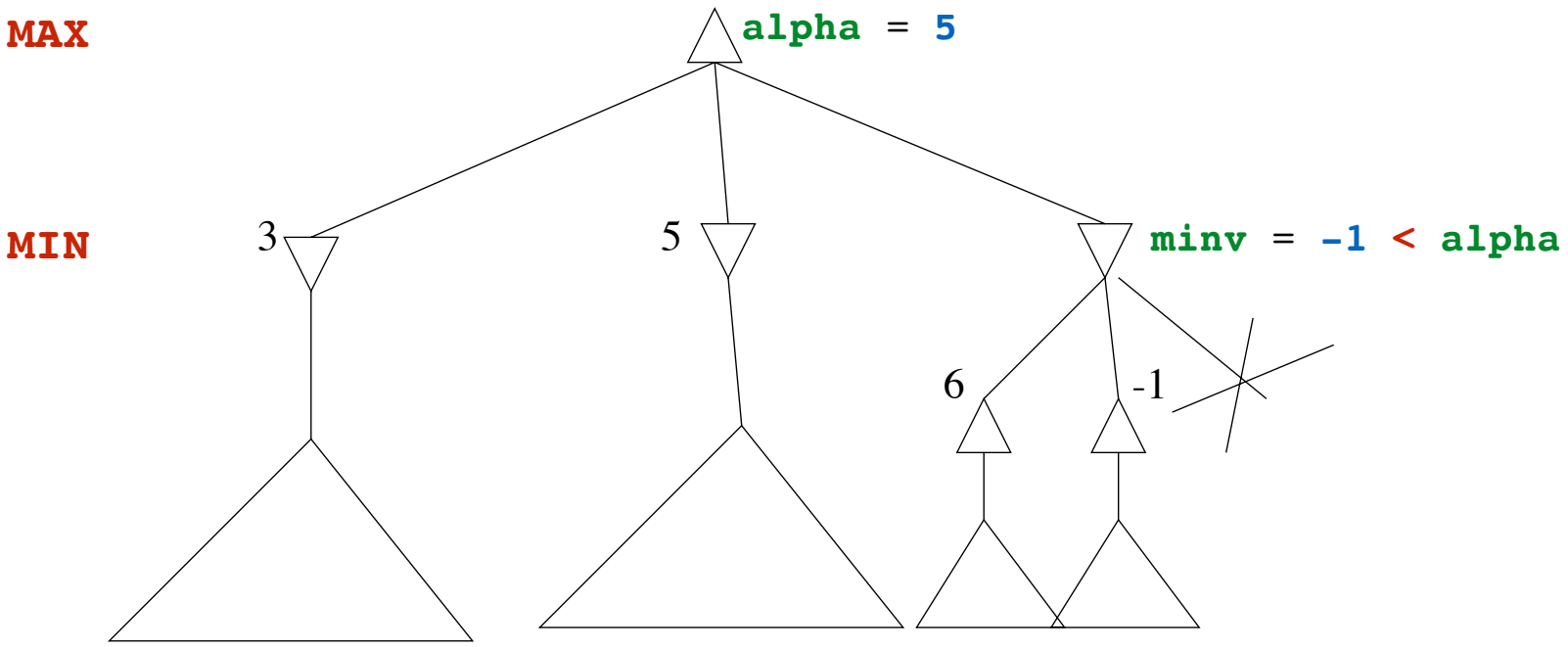
Potatura di rami (2)

- Il VP è tale che:
  - Il valore provvisorio di un nodo **MAX** non può mai diminuire man mano che si considerano i valori **MINIMAX** dei propri nodi figli.
  - Il valore provvisorio di un nodo **MIN** non può mai aumentare man mano che si considerano i valori **MINIMAX** dei propri nodi figli.

Potatura di rami (3)

- Il punto chiave del metodo è quello di mantenere le seguenti due variabili durante la ricerca della decisione MiniMax:
  - **`alpha`**: migliore scelta (quella con valore più alto) già esplorata per **`Max`** lungo il cammino.
  - **`beta`**: migliore scelta (quella con valore più basso) già esplorata per **`Min`** lungo il cammino.

Potatura Alfa-Beta (4)



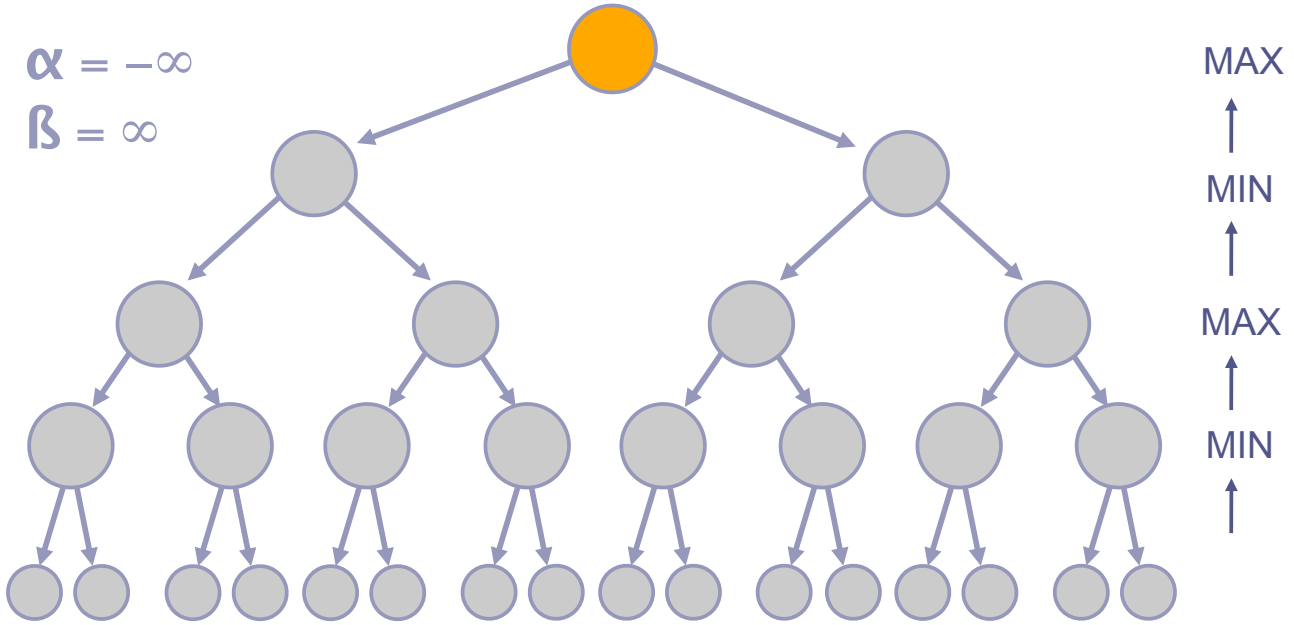
Potatura di rami (5)

- Non appena si determina nell'albero un nodo **MIN** con valore provvisorio $\leq$ **alpha**, non vengono generati altri sottoalberi di tale nodo.

- Non appena si determina nell'albero un nodo **MAX** con valore provvisorio $\geq$ **beta**, non vengono generati altri sottoalberi di tale nodo.

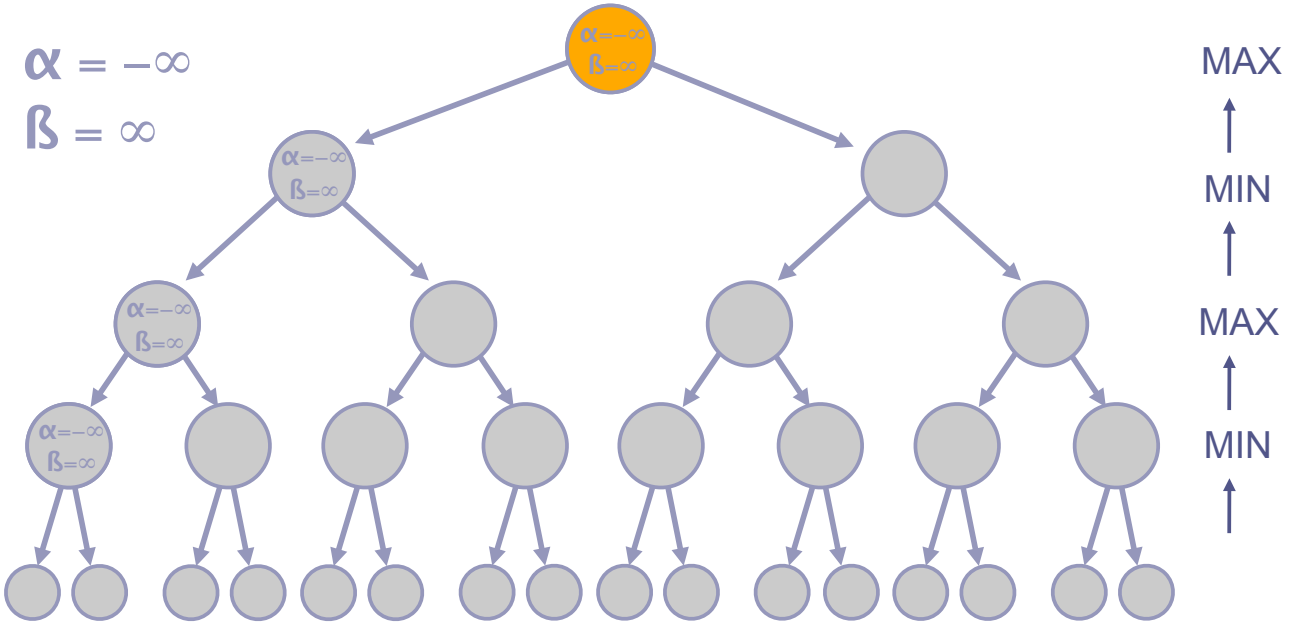
La potatura Alfa-Beta

esempio



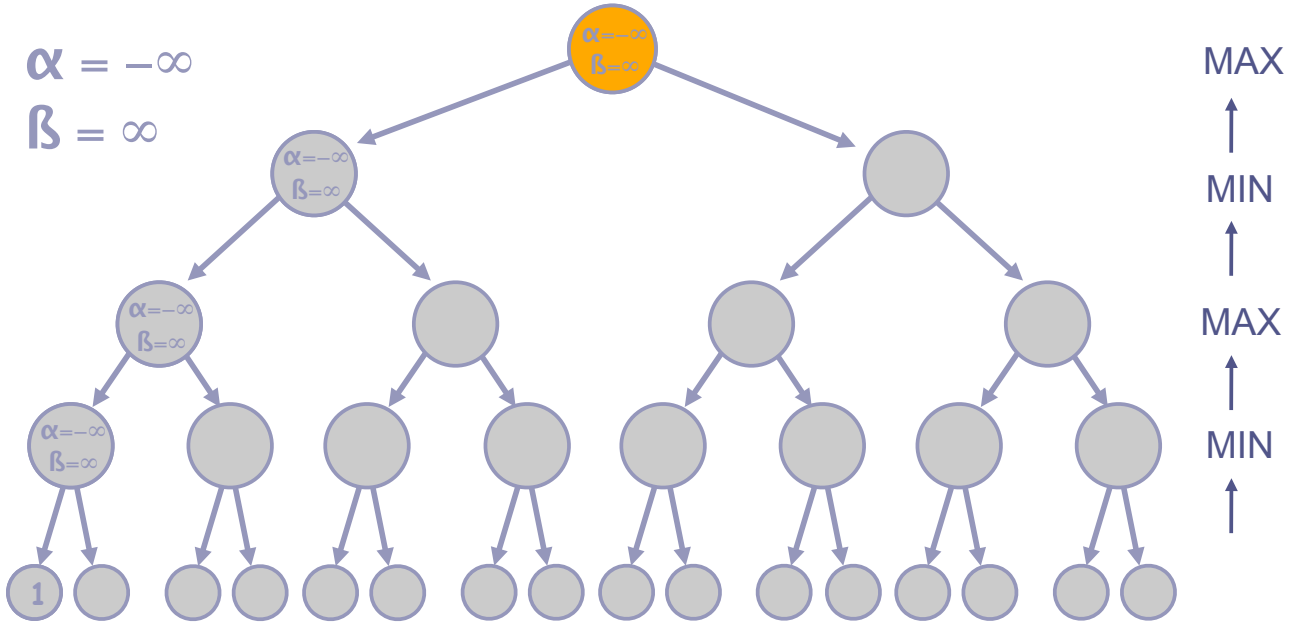
La potatura Alfa-Beta

esempio



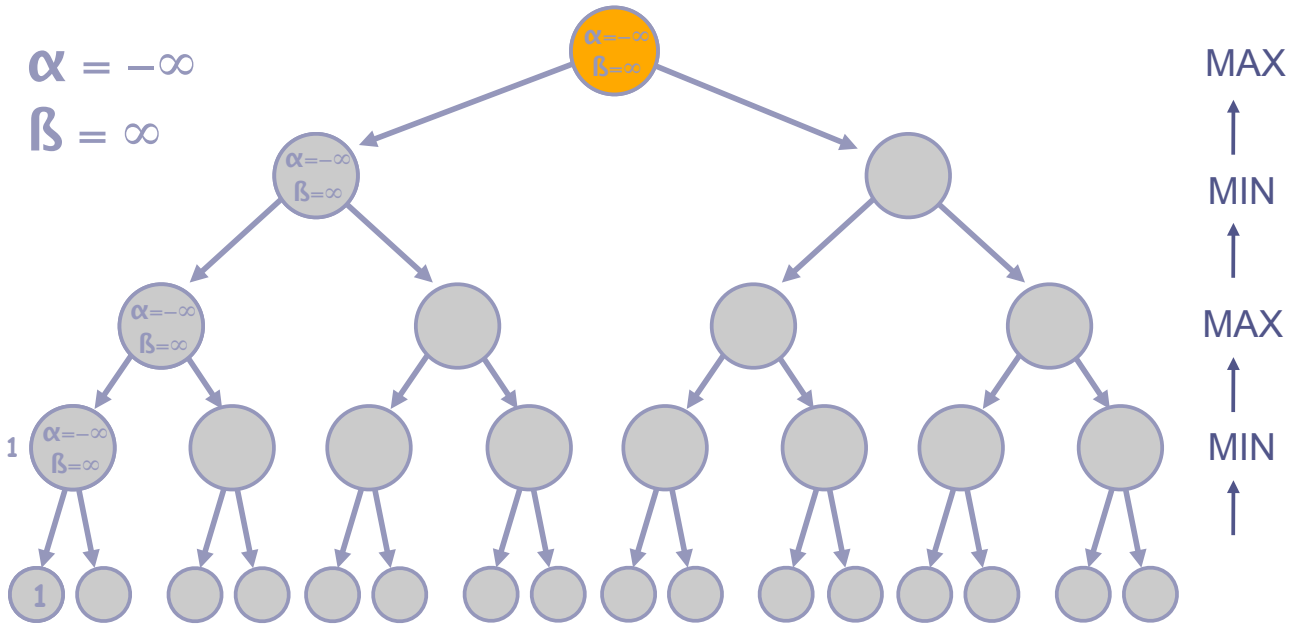
La potatura Alfa-Beta

esempio



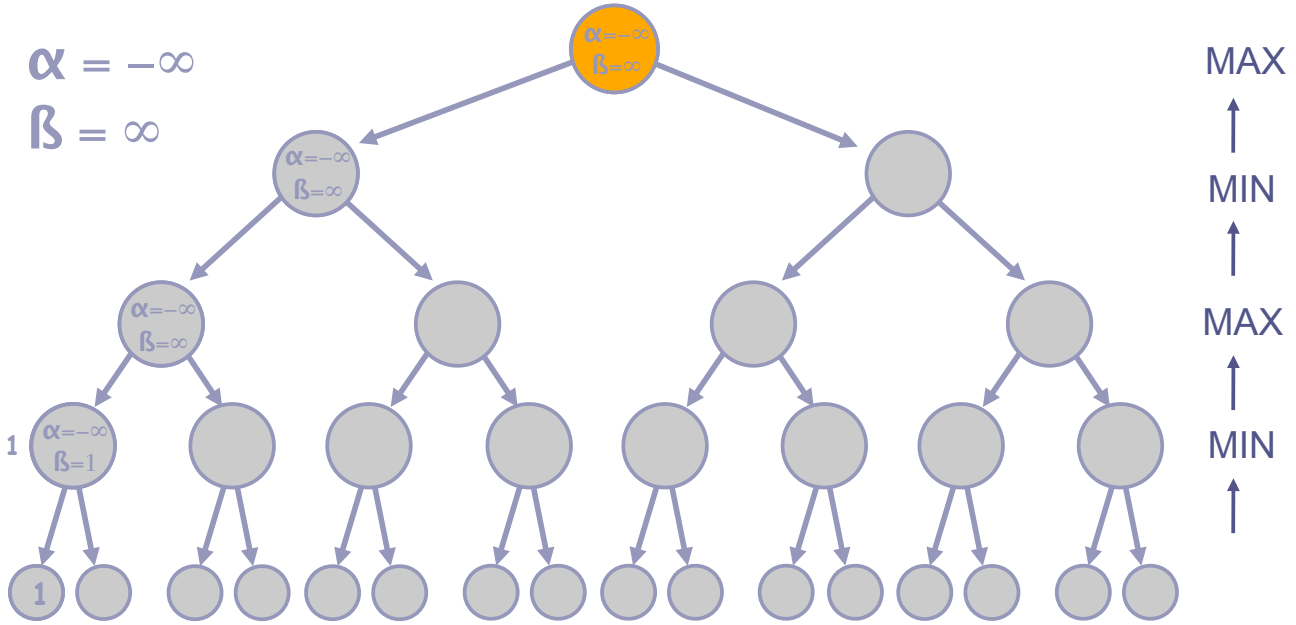
La potatura Alfa-Beta

esempio



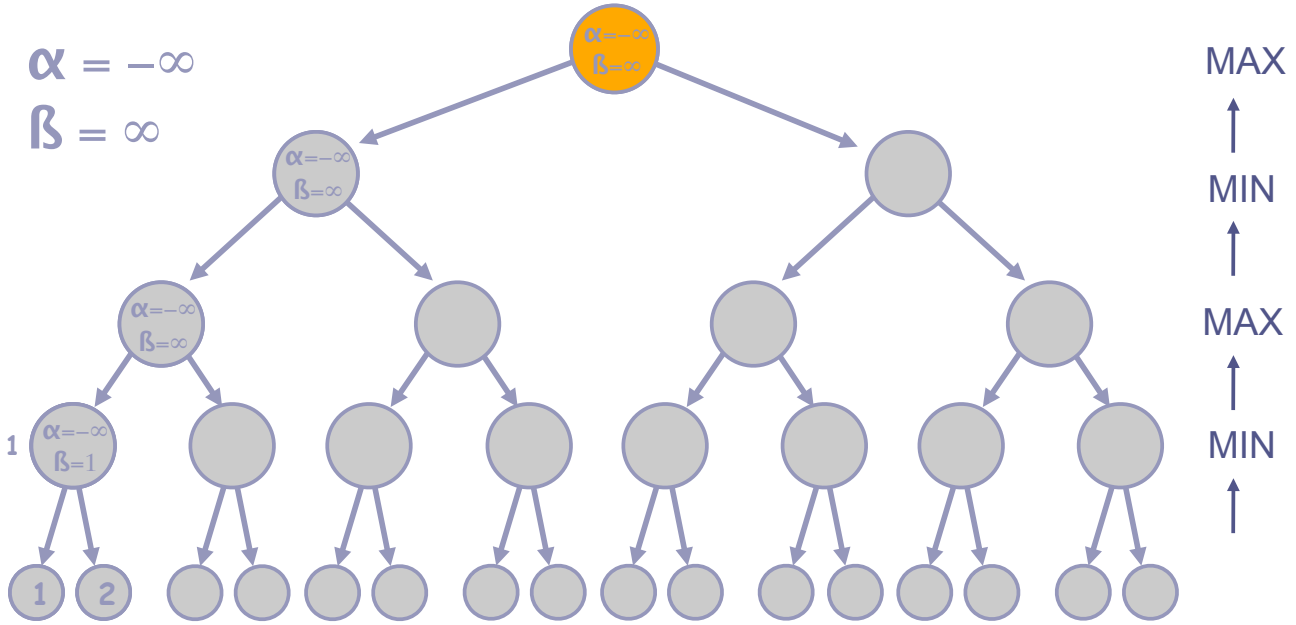
La potatura Alfa-Beta

esempio



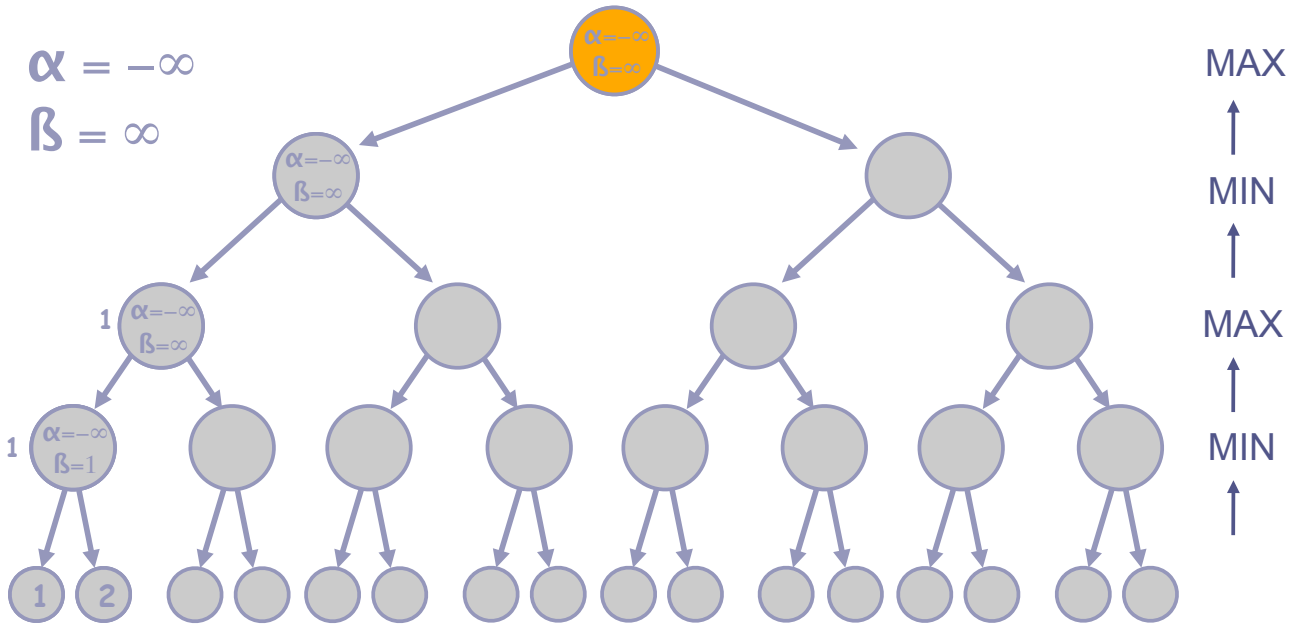
La potatura Alfa-Beta

esempio



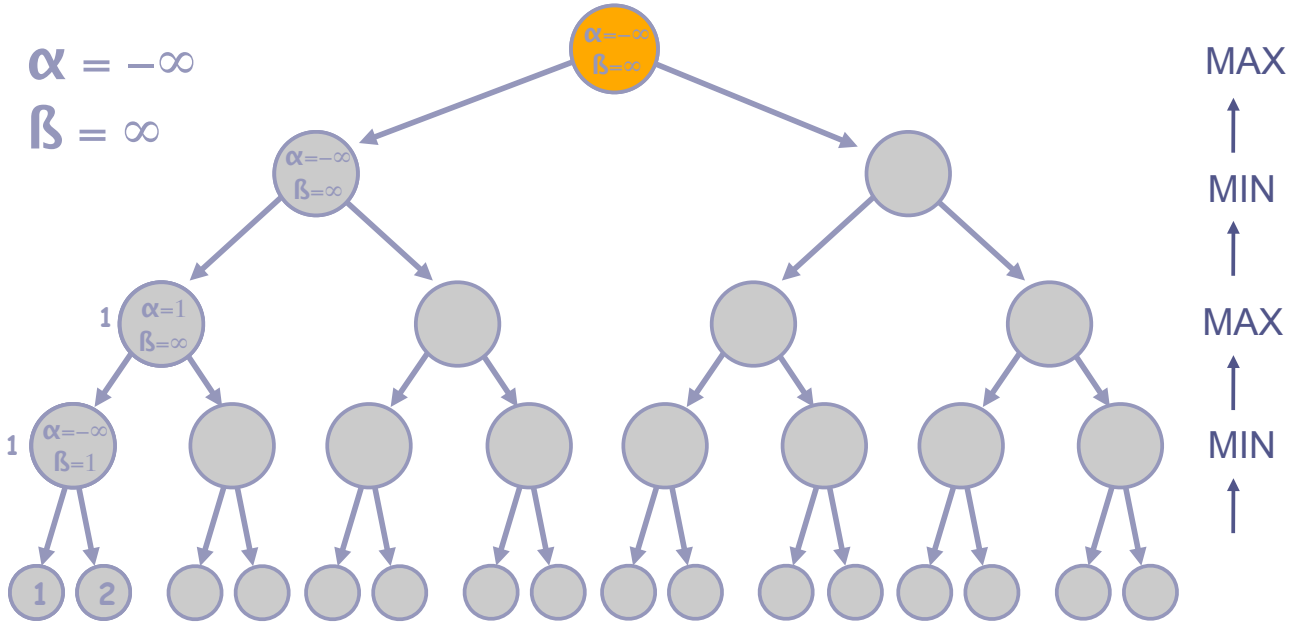
La potatura Alfa-Beta

esempio



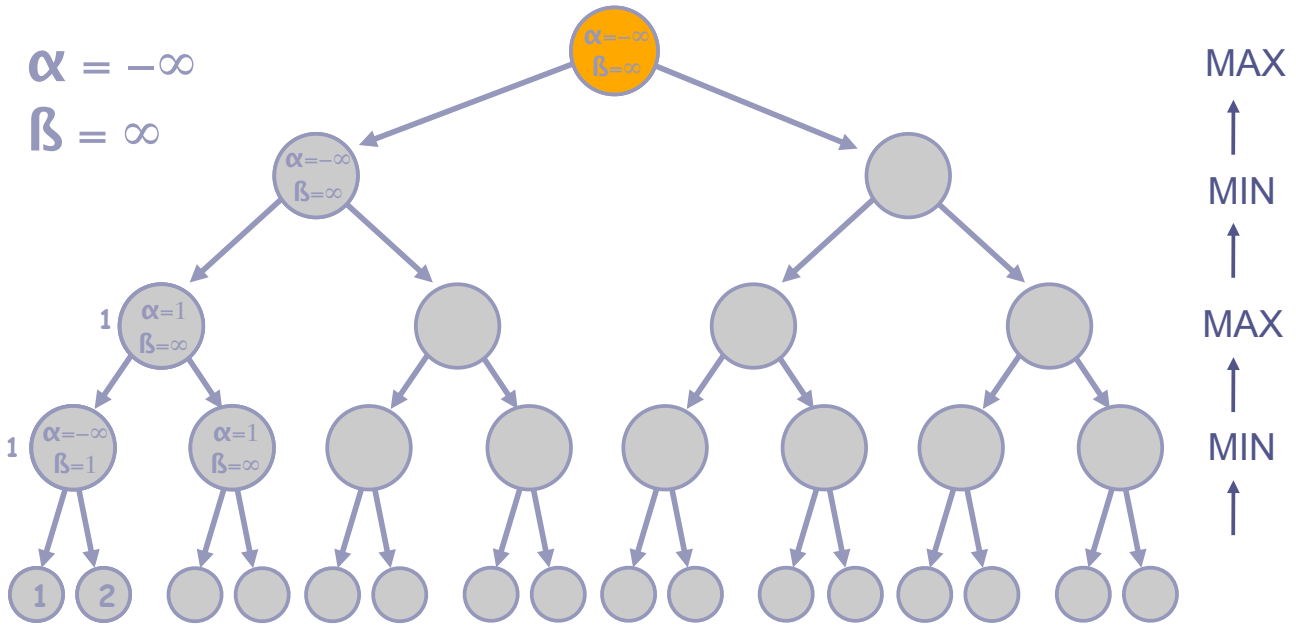
La potatura Alfa-Beta

esempio



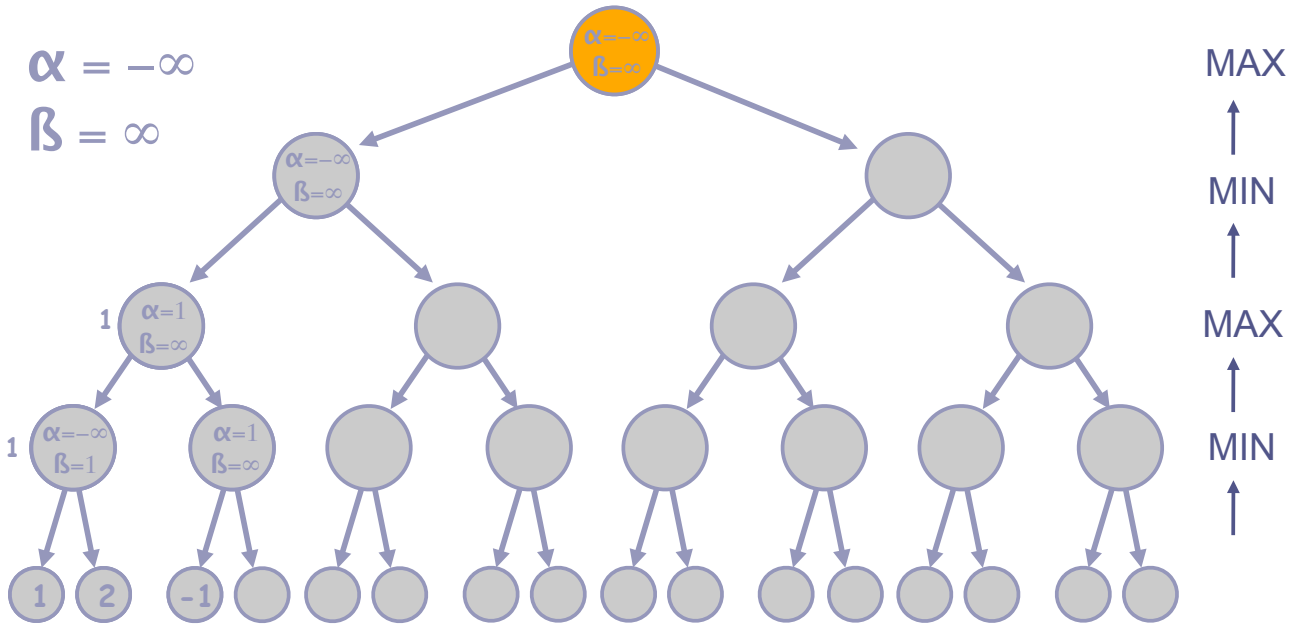
La potatura Alfa-Beta

esempio



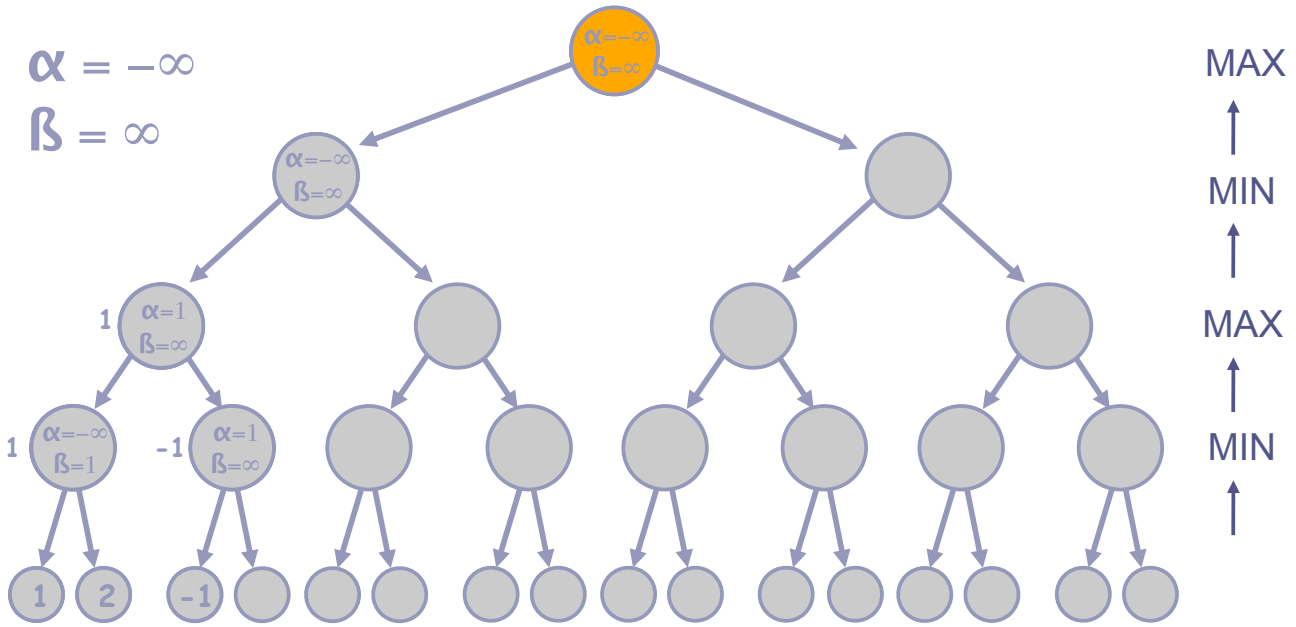
La potatura Alfa-Beta

esempio



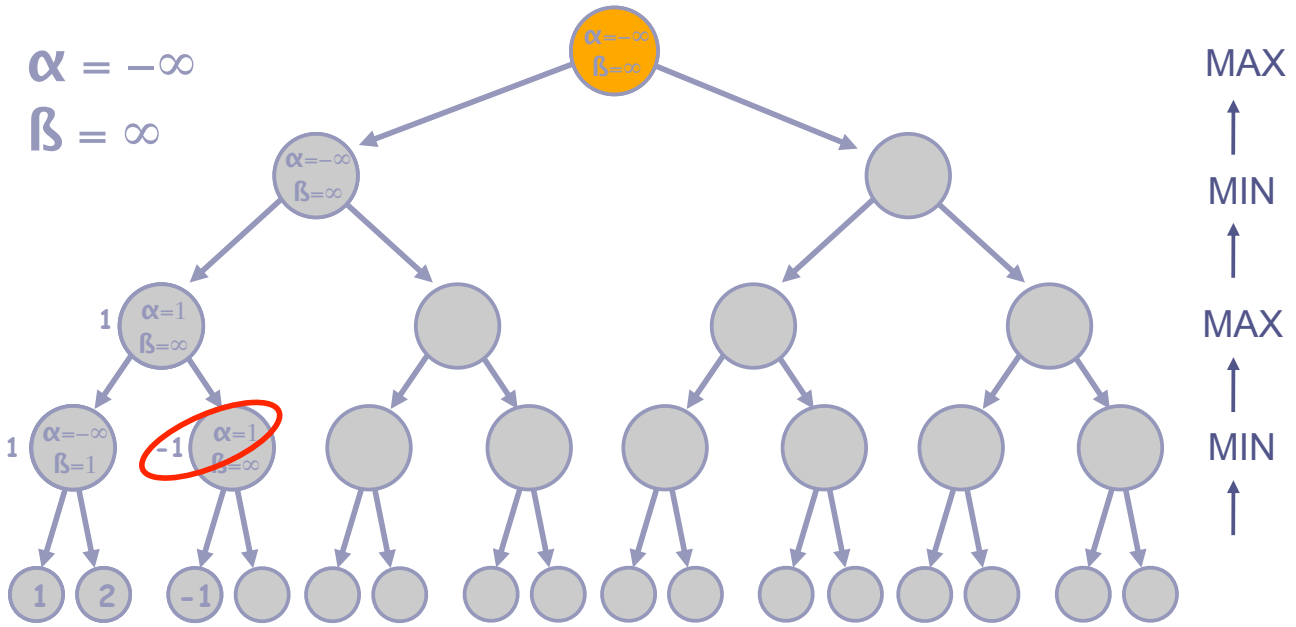
La potatura Alfa-Beta

esempio



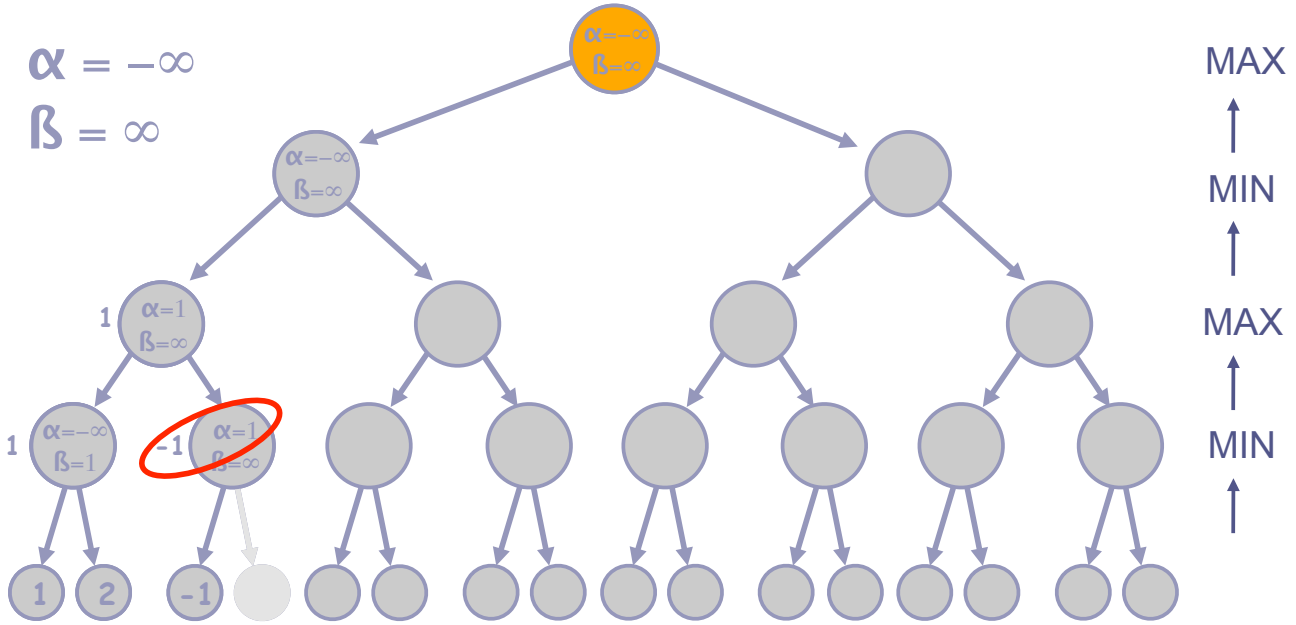
La potatura Alfa-Beta

esempio



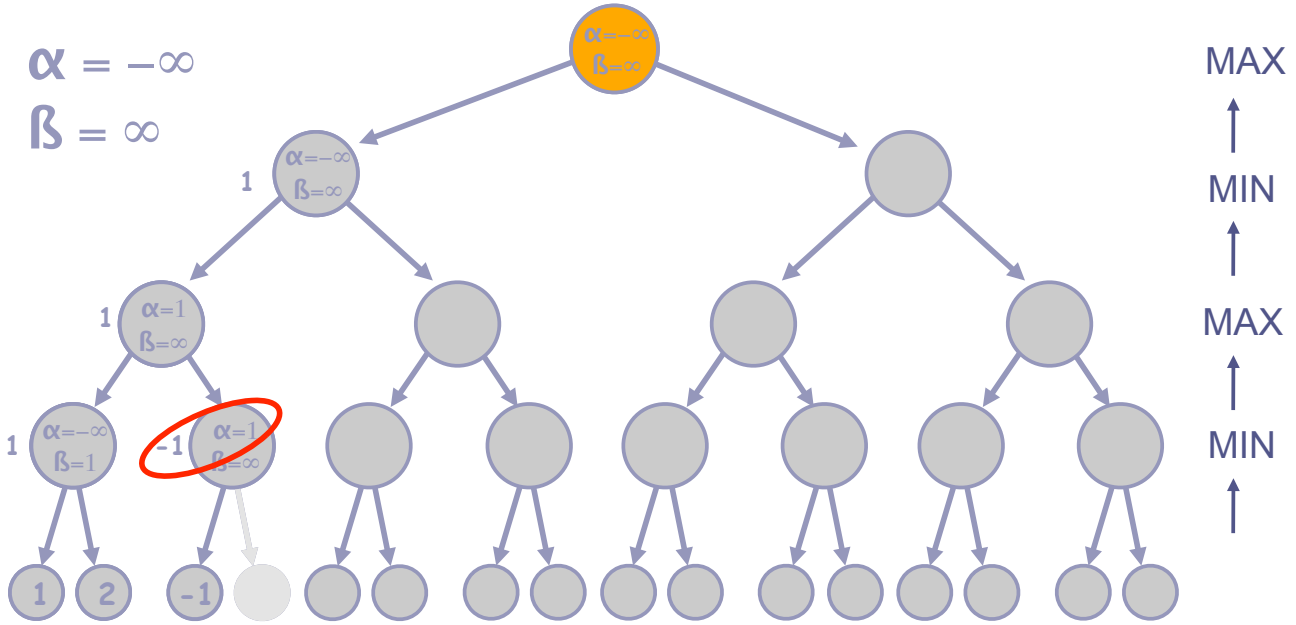
La potatura Alfa-Beta

esempio



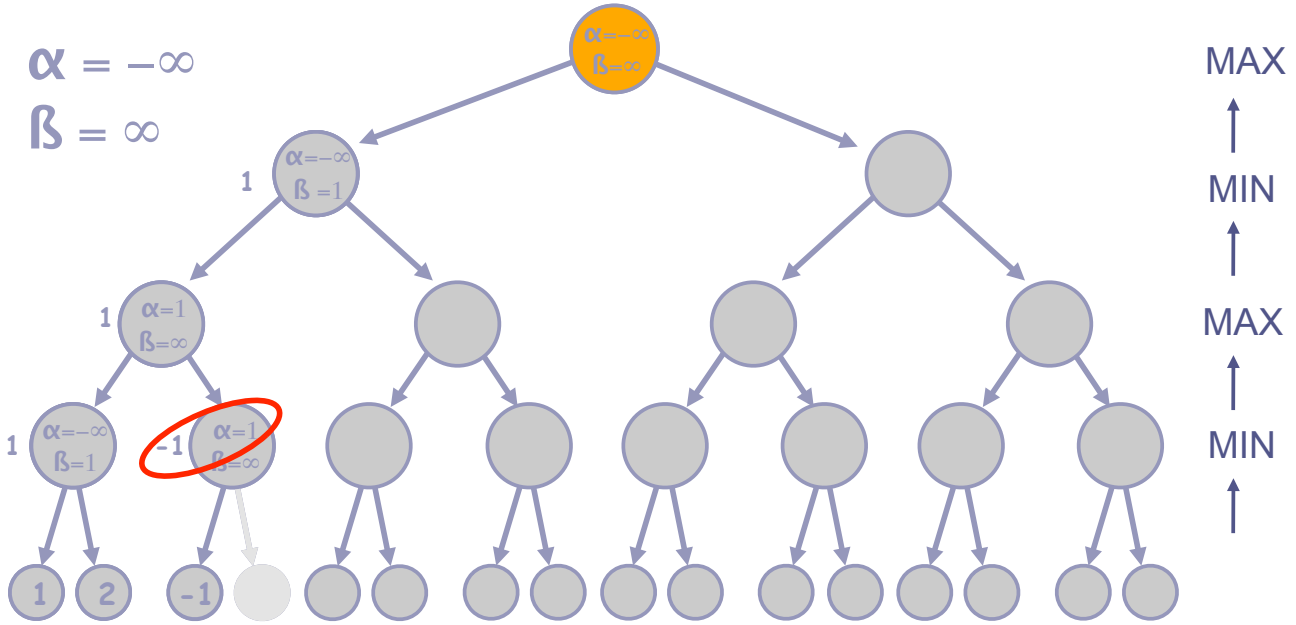
La potatura Alfa-Beta

esempio



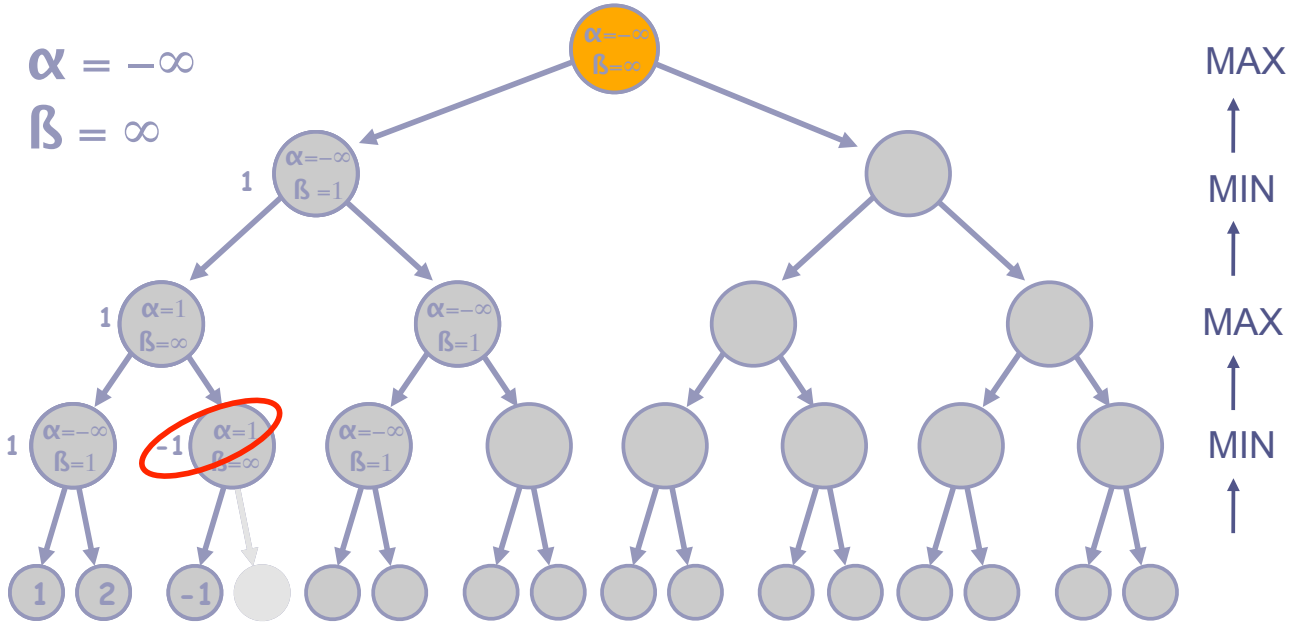
La potatura Alfa-Beta

esempio



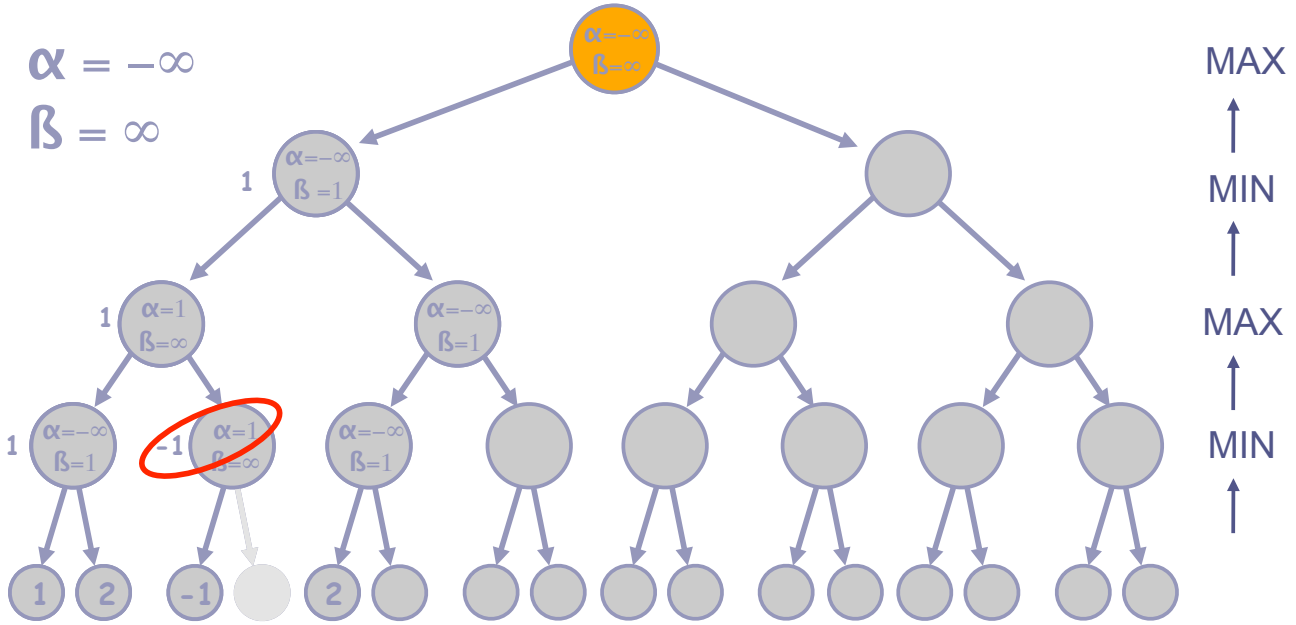
La potatura Alfa-Beta

esempio



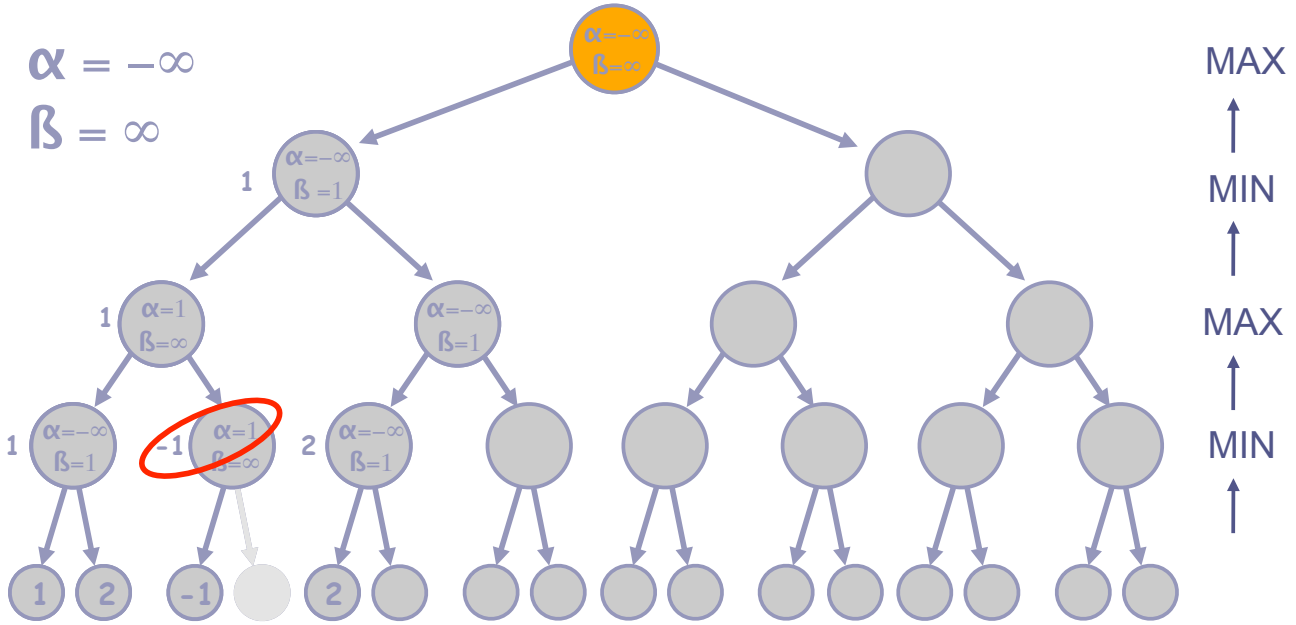
La potatura Alfa-Beta

esempio



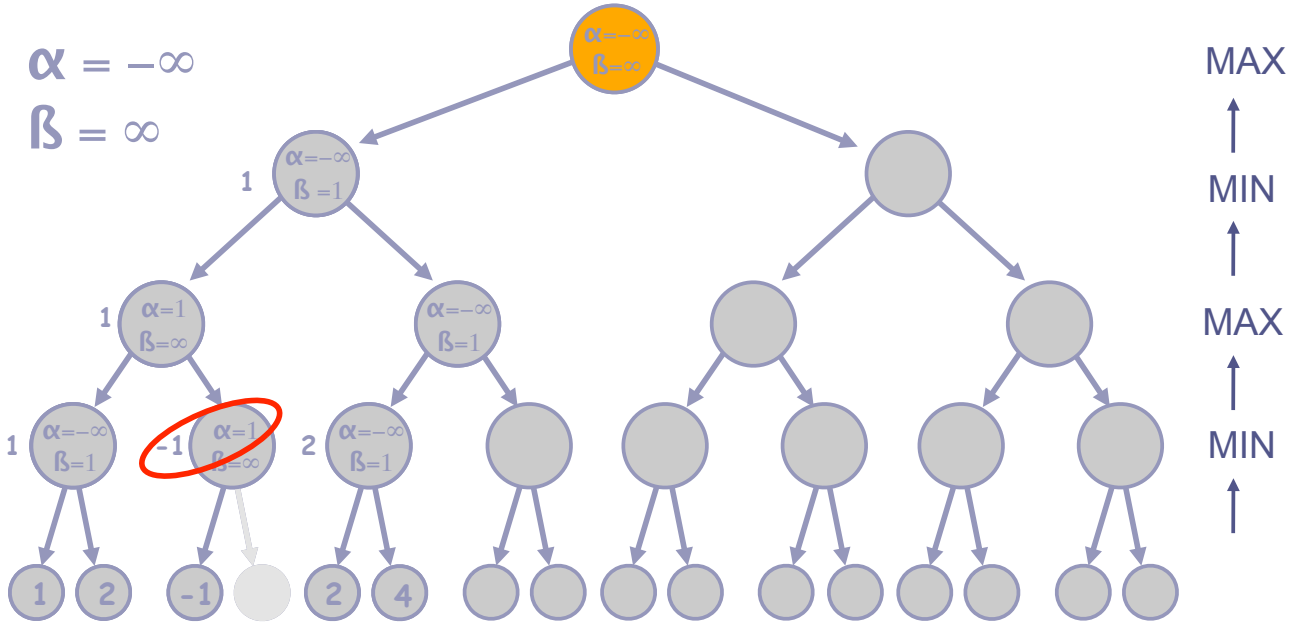
La potatura Alfa-Beta

esempio



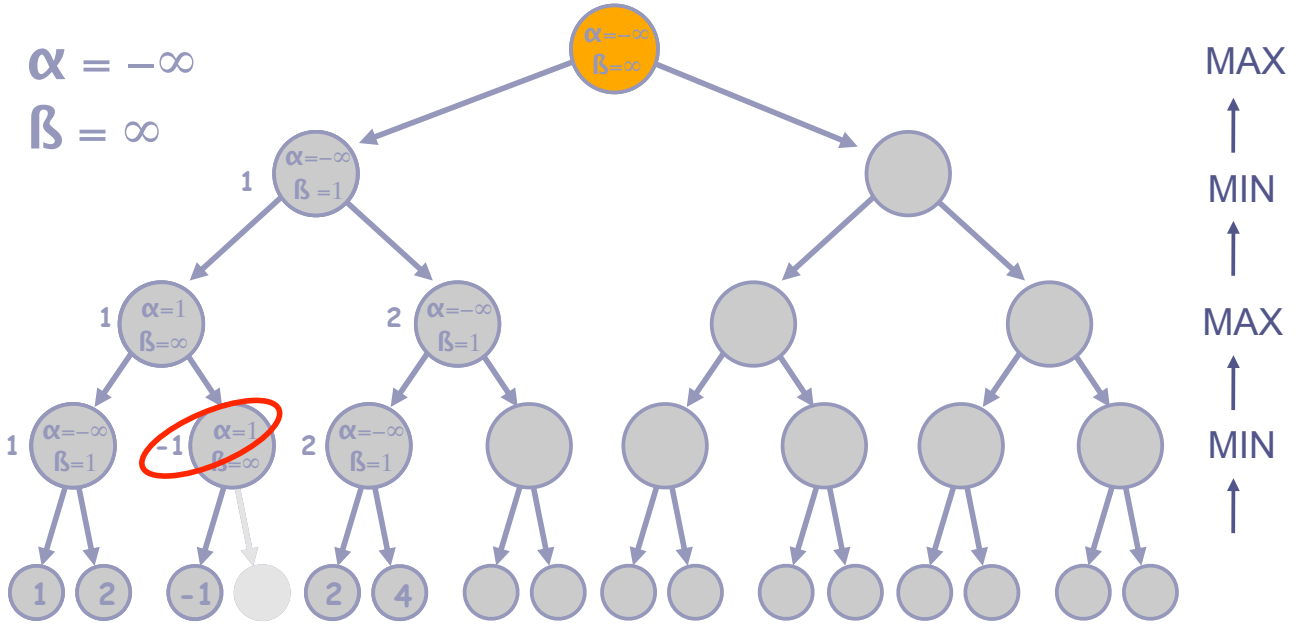
La potatura Alfa-Beta

esempio



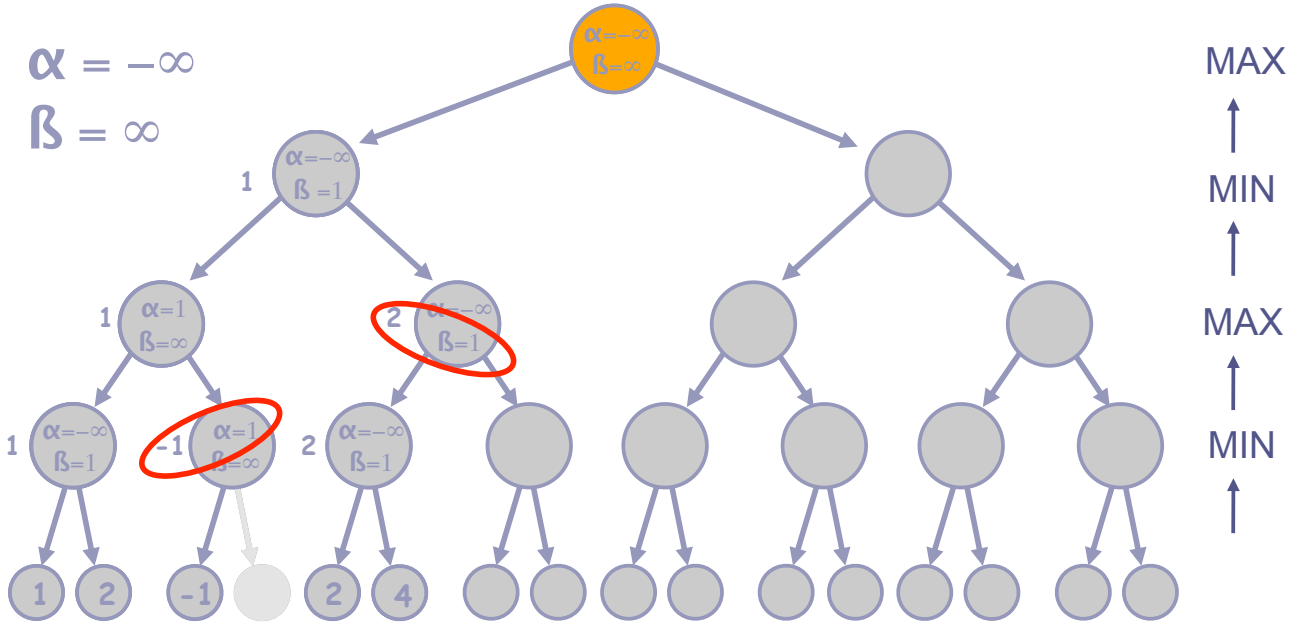
La potatura Alfa-Beta

esempio



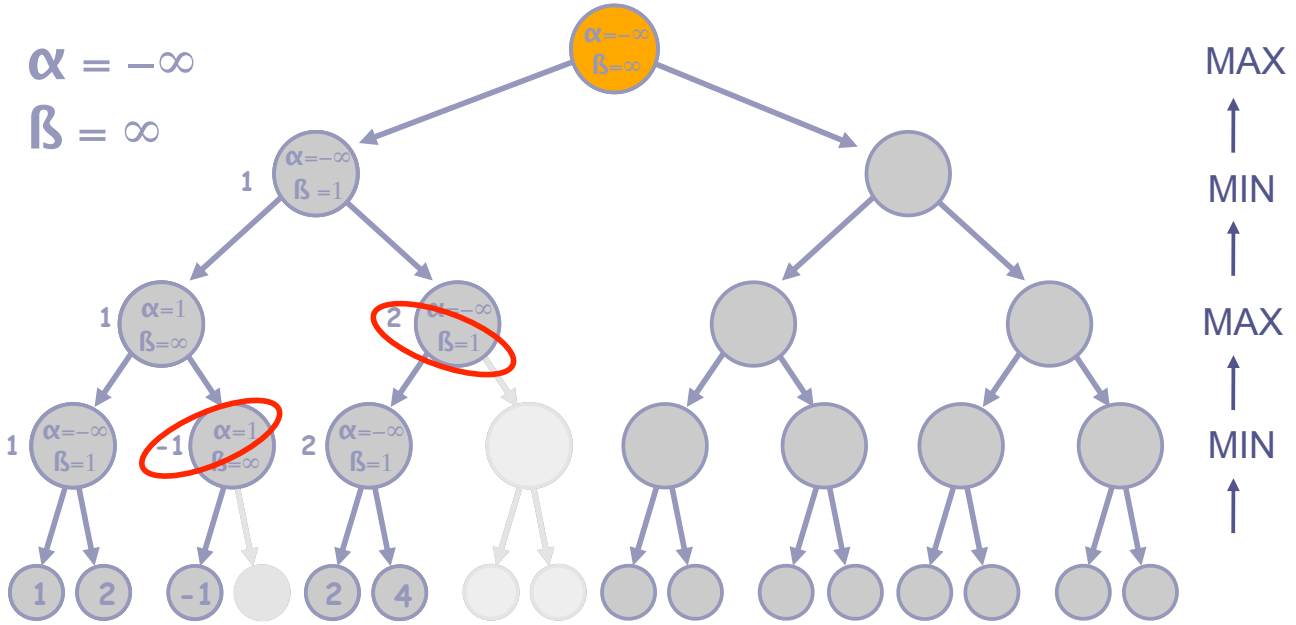
La potatura Alfa-Beta

esempio



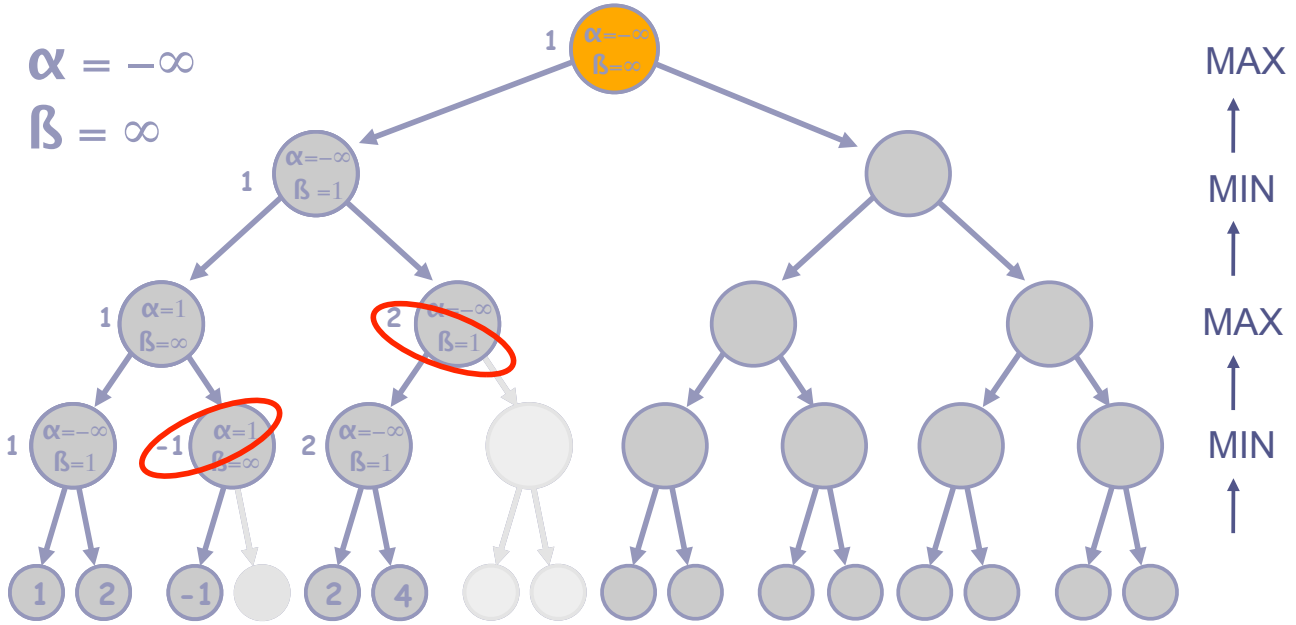
La potatura Alfa-Beta

esempio



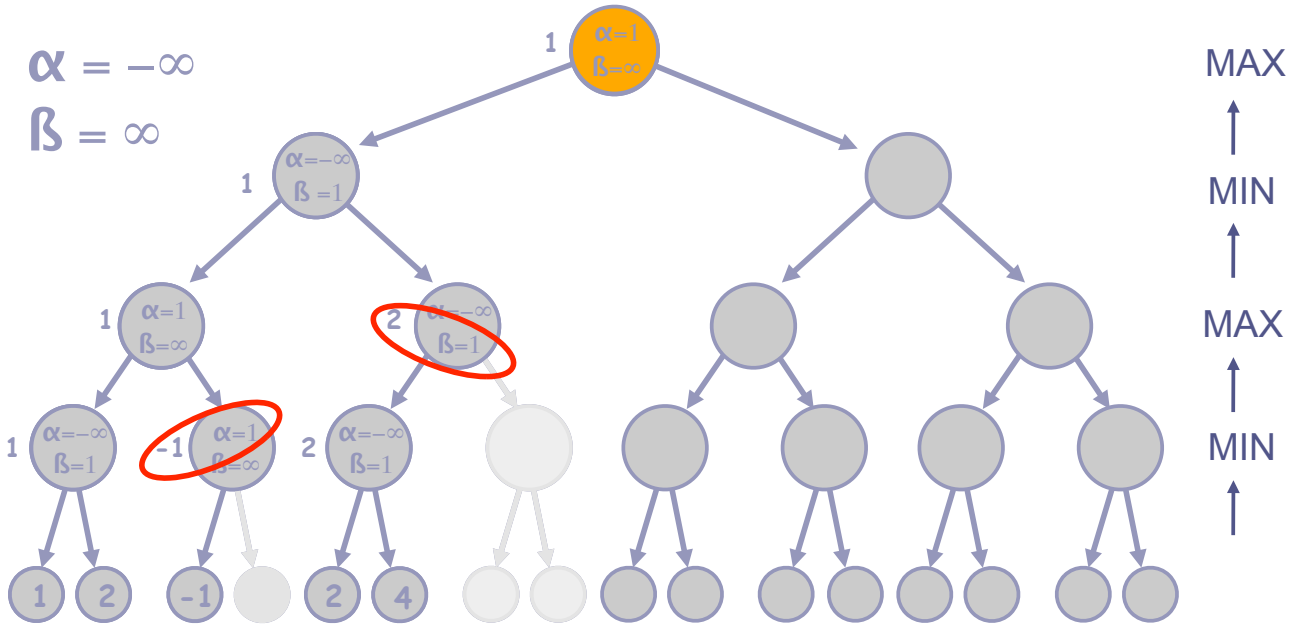
La potatura Alfa-Beta

esempio



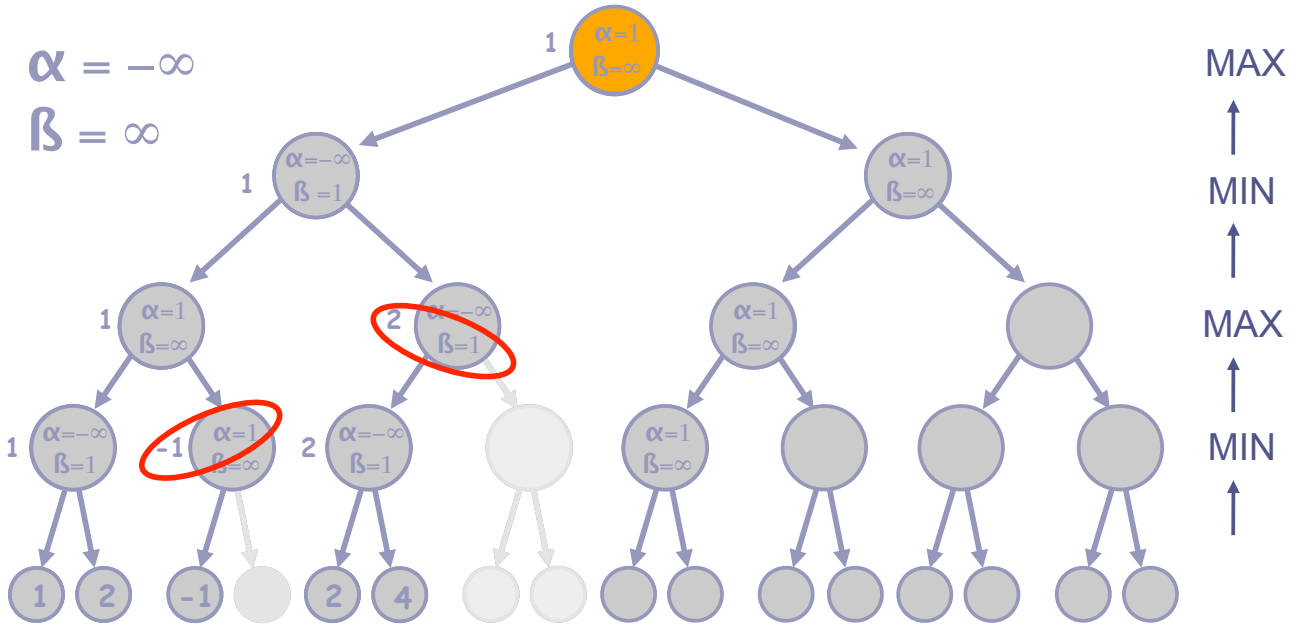
La potatura Alfa-Beta

esempio



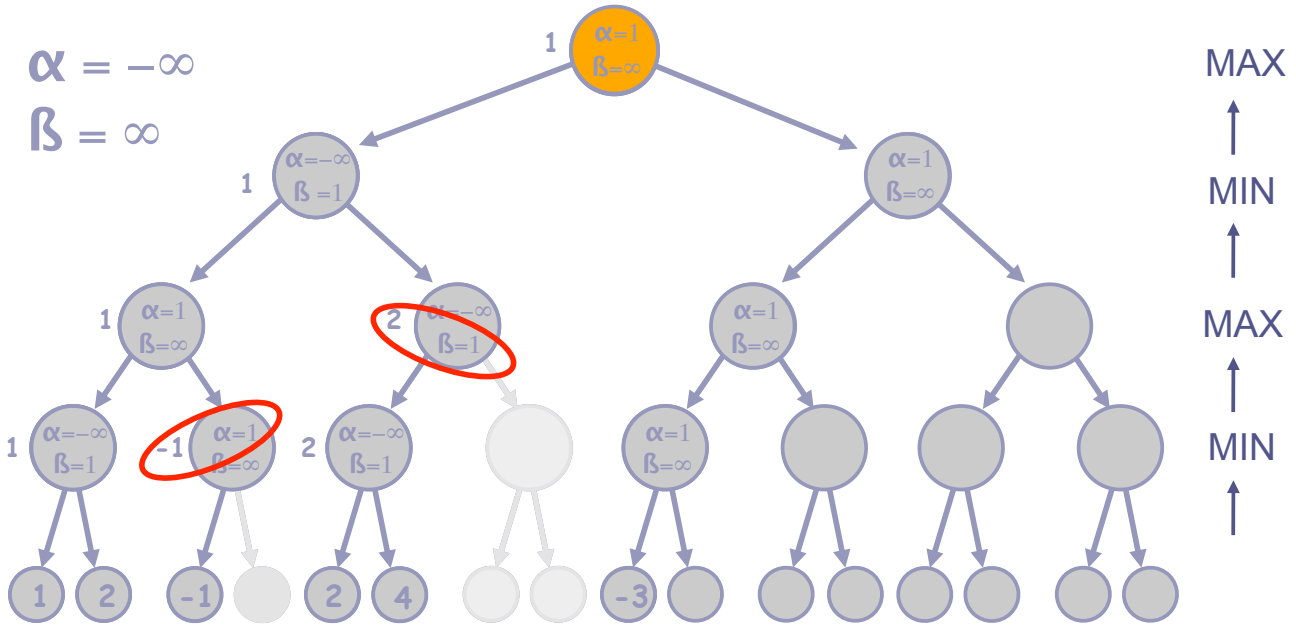
La potatura Alfa-Beta

esempio



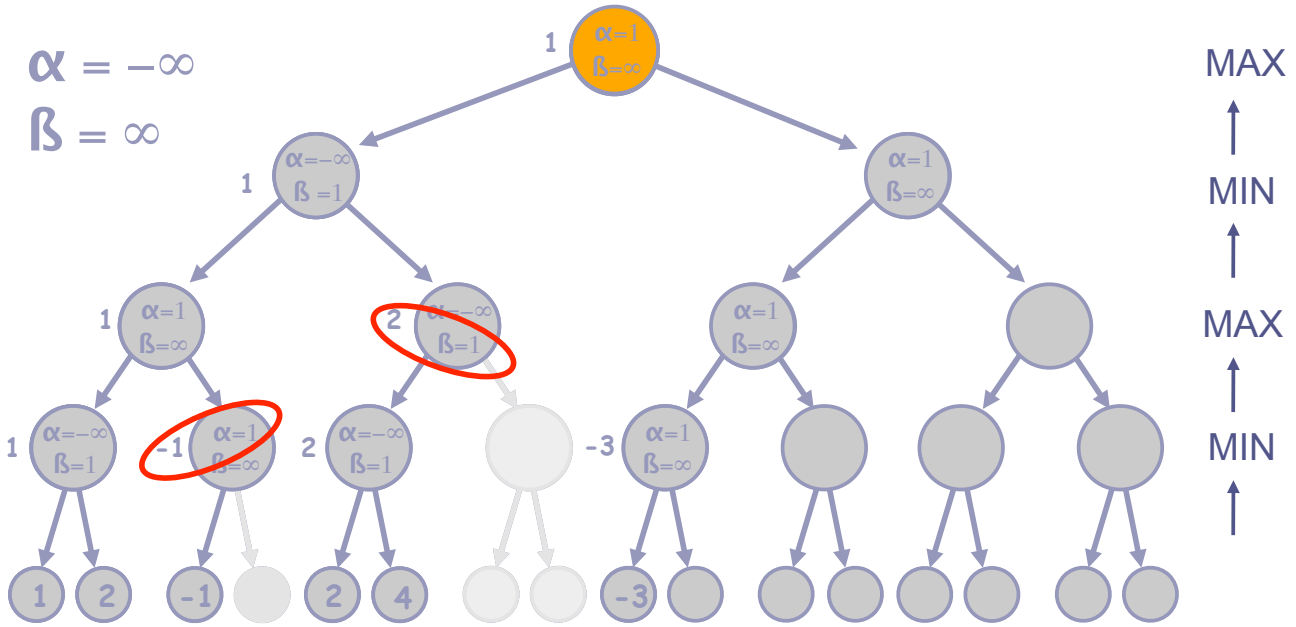
La potatura Alfa-Beta

esempio



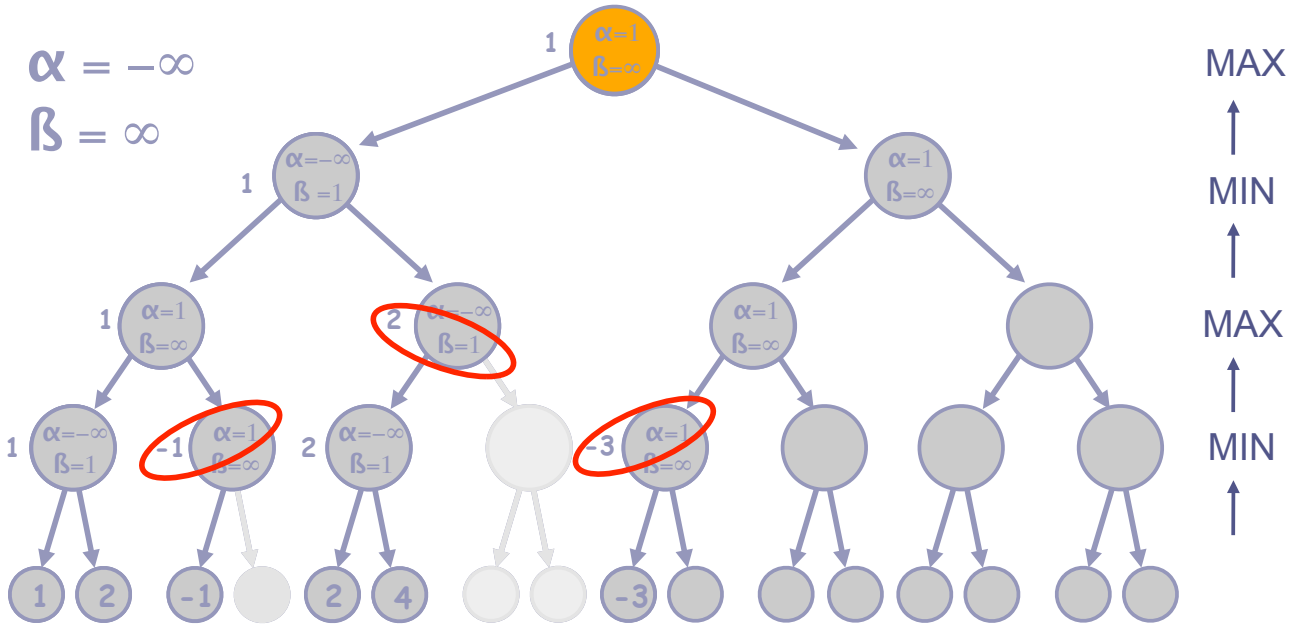
La potatura Alfa-Beta

esempio



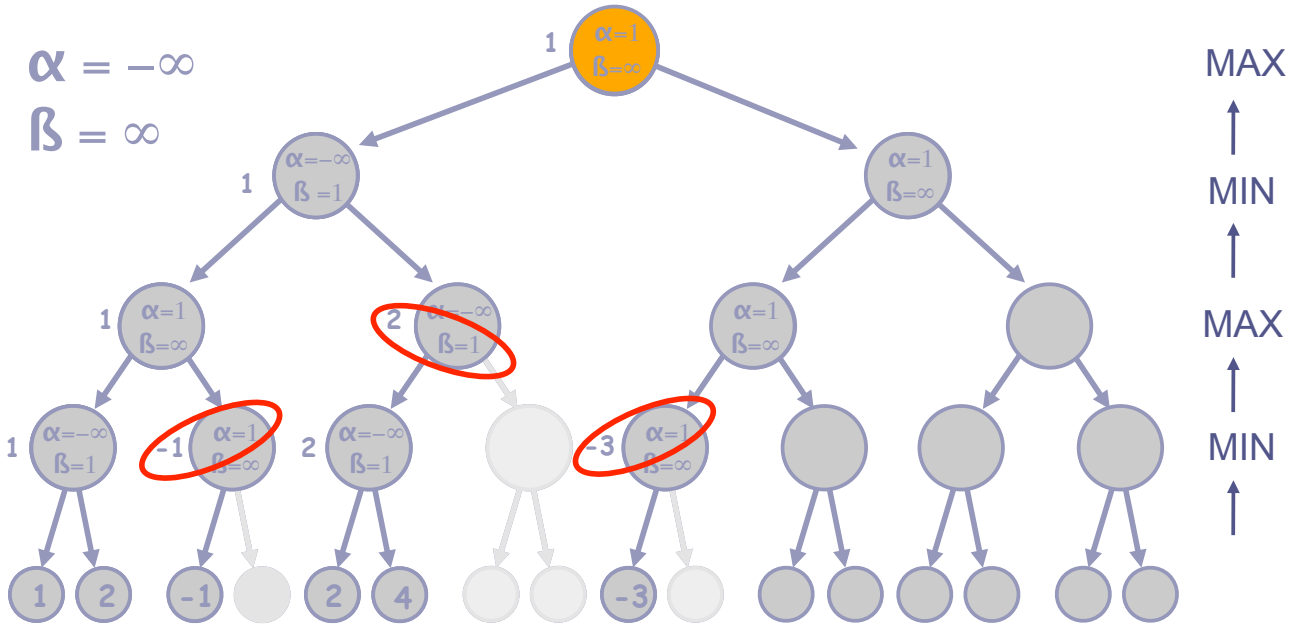
La potatura Alfa-Beta

esempio



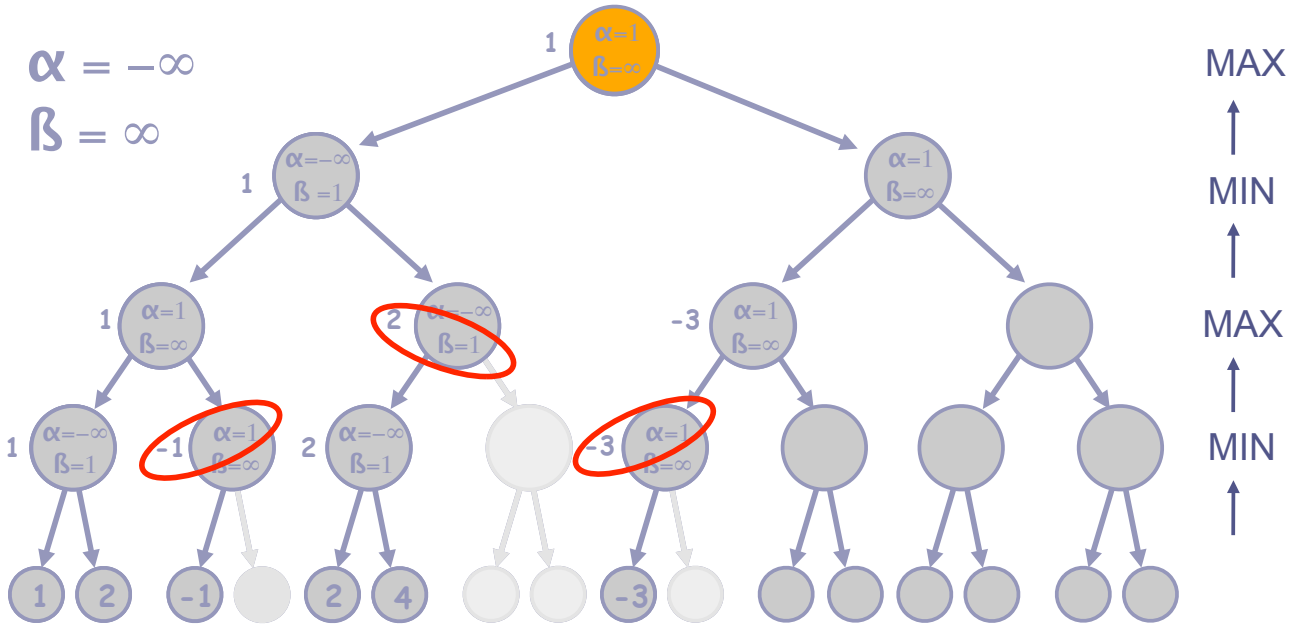
La potatura Alfa-Beta

esempio



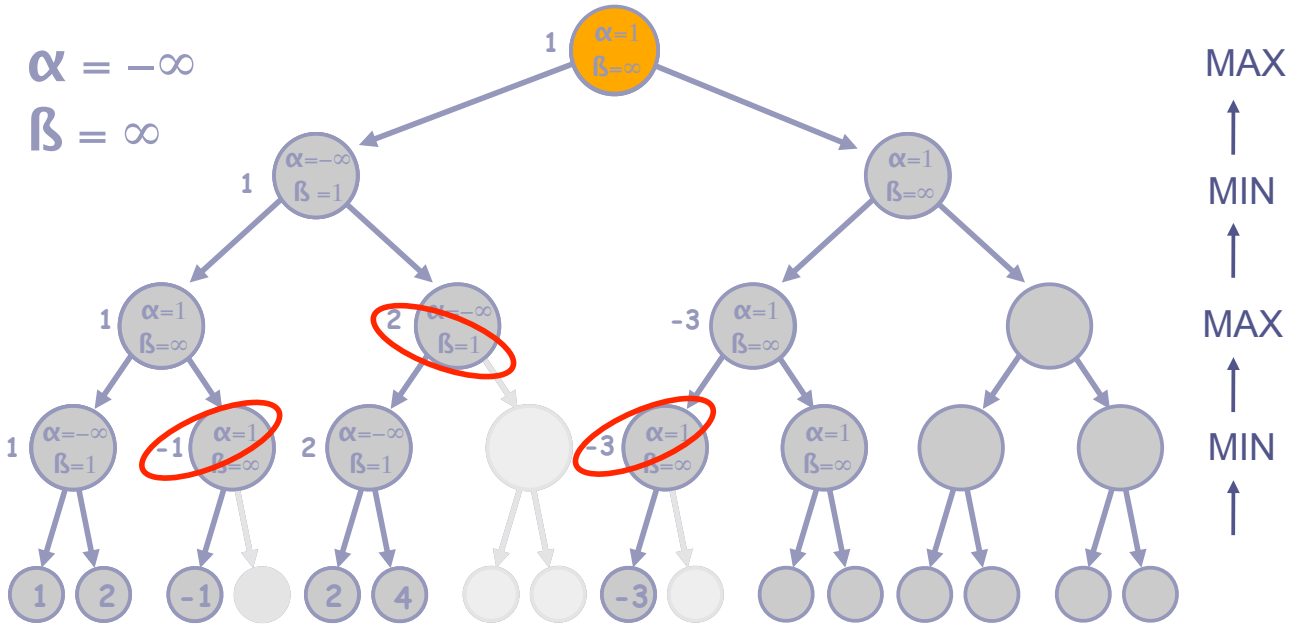
La potatura Alfa-Beta

esempio



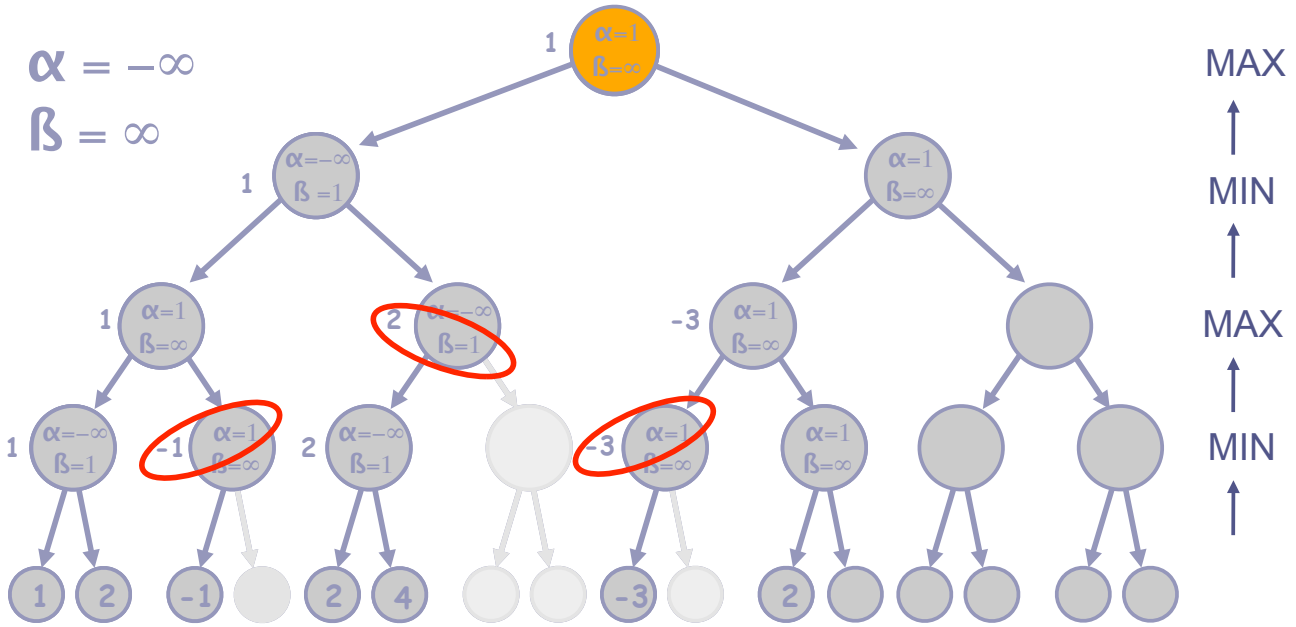
La potatura Alfa-Beta

esempio



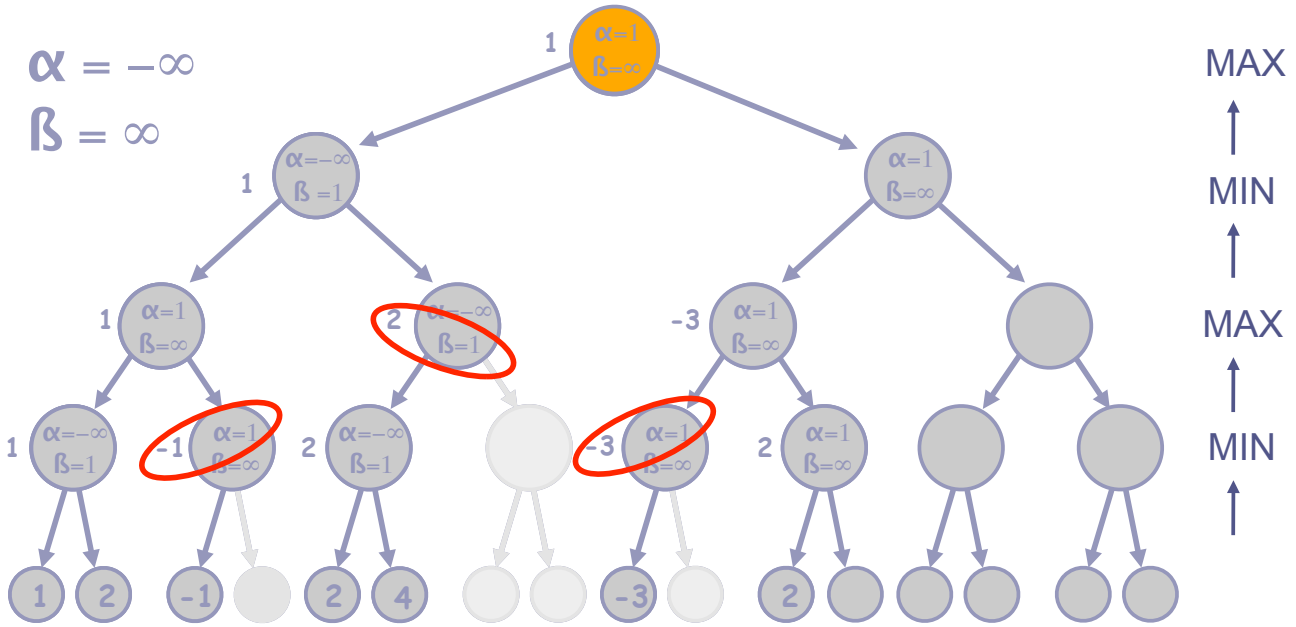
La potatura Alfa-Beta

esempio



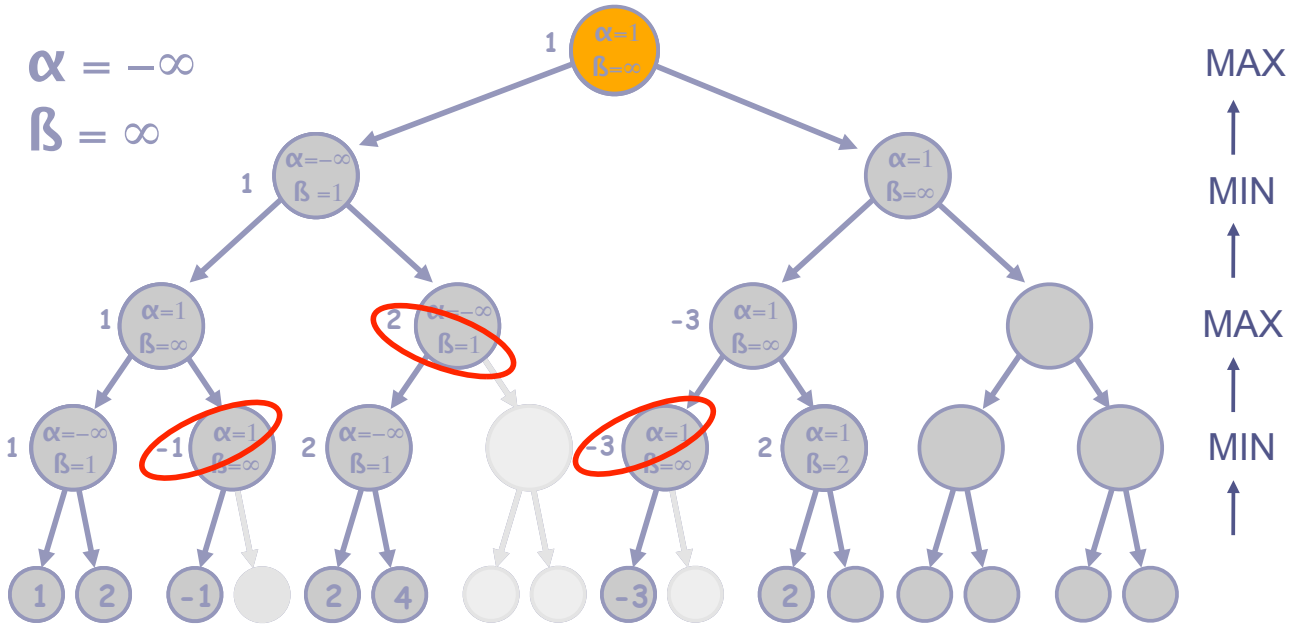
La potatura Alfa-Beta

esempio



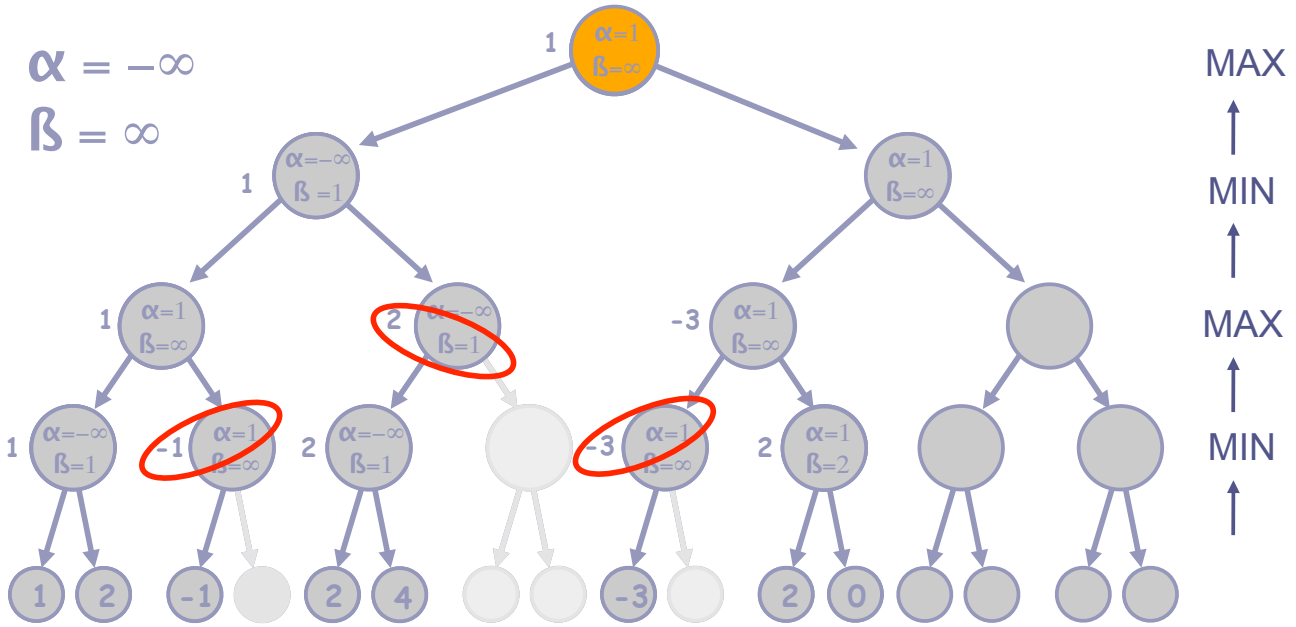
La potatura Alfa-Beta

esempio



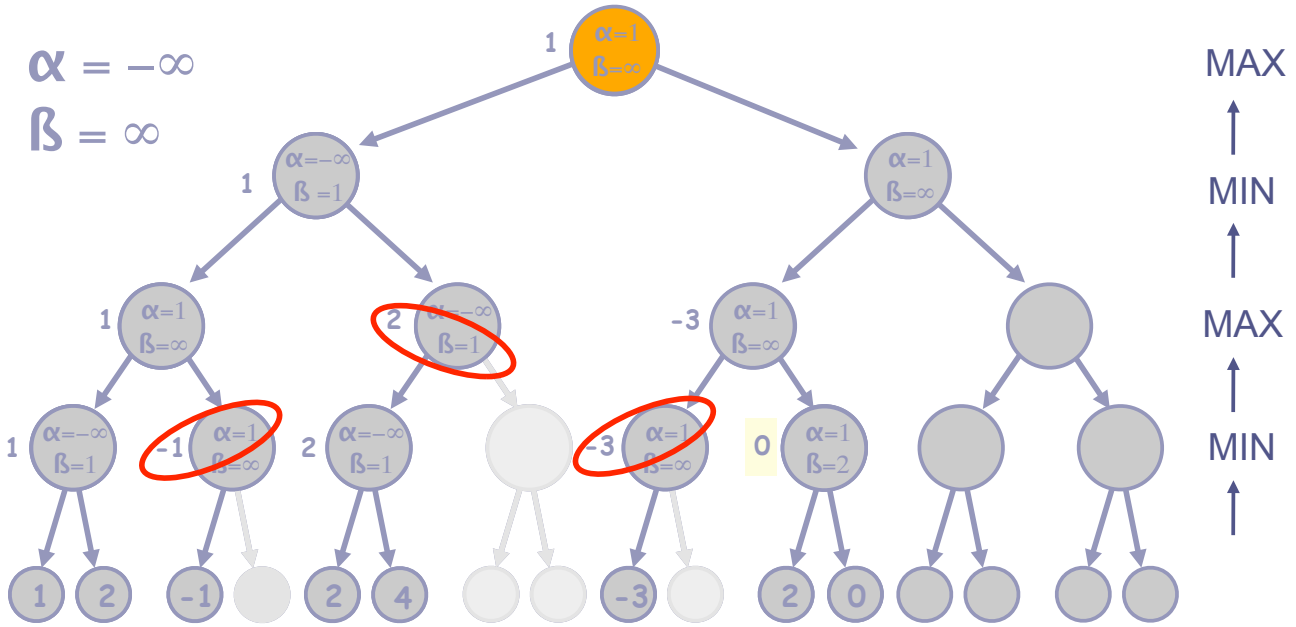
La potatura Alfa-Beta

esempio



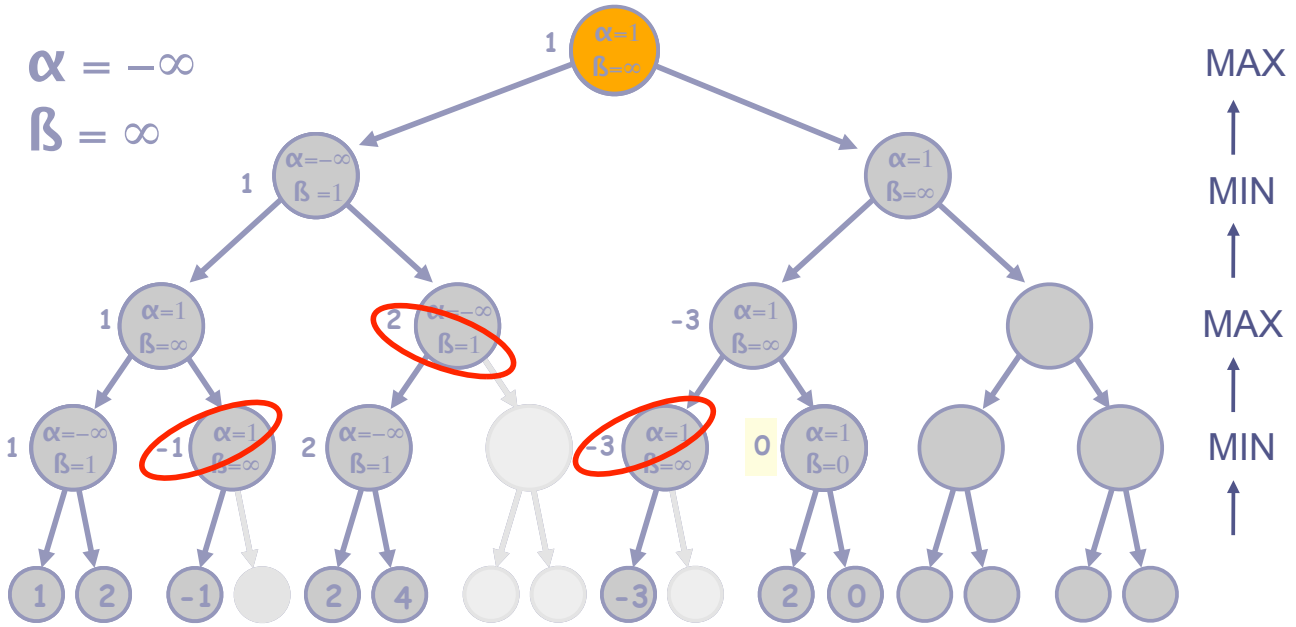
La potatura Alfa-Beta

esempio



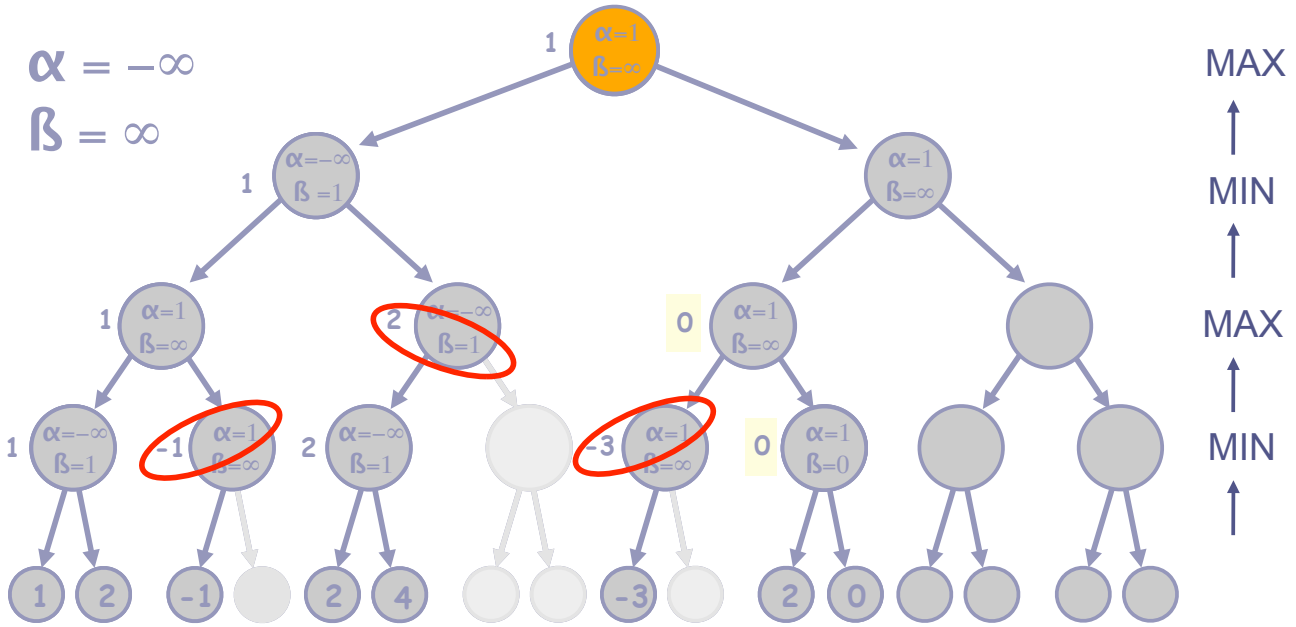
La potatura Alfa-Beta

esempio



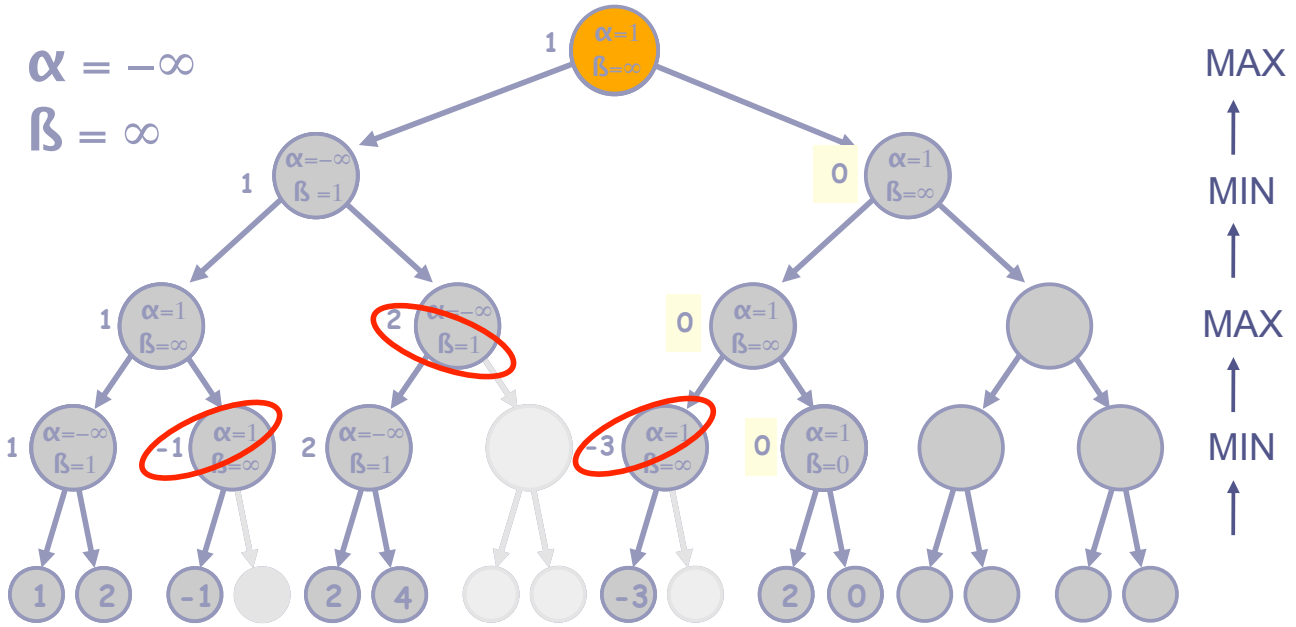
La potatura Alfa-Beta

esempio



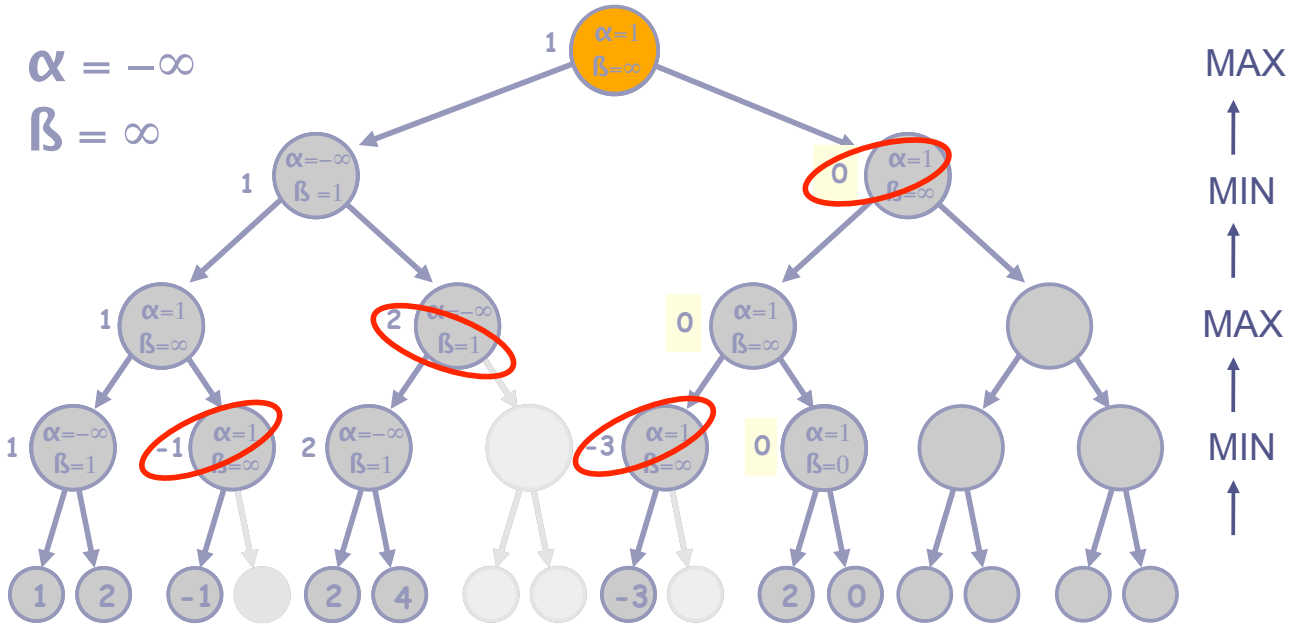
La potatura Alfa-Beta

esempio



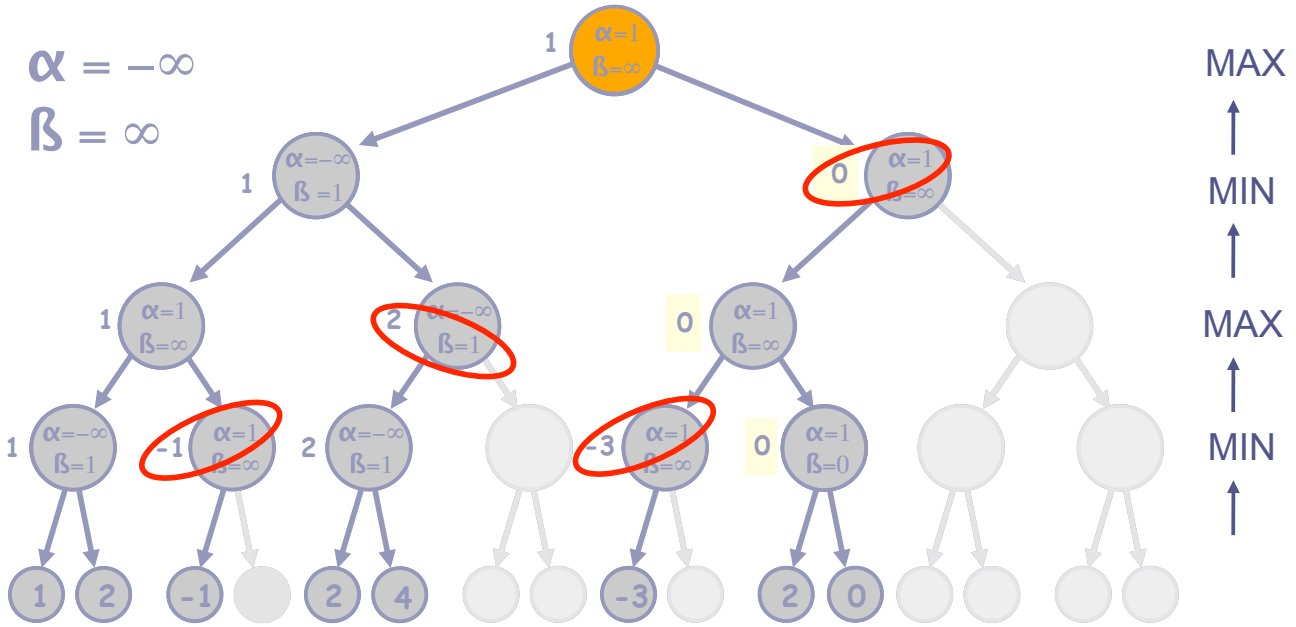
La potatura Alfa-Beta

esempio



La potatura Alfa-Beta

esempio



Sintesi degli Argomenti Trattati nella Lezione

- I **giochi a due avversari**, ad esempio i giochi con scacchiera come gli Scacchi o il Go, sono in qualche misura interessanti perché offrono una pura competizione astratta. E' questa astrazione che rende i giochi un obiettivo attraente di ricerca in AI. E' facile rappresentare lo stato di un gioco e di solito gli agenti vengono limitati ad effettuare un numero abbastanza piccolo di azioni ben definite. Ciò rende un gioco una idealizzazione di un mondo in cui agenti ostili agiscono in modo da diminuire il benessere di qualcuno.

- Un **gioco** può essere definito dallo stato iniziale (che dice come è disposta la scacchiera), dagli operatori (che definiscono le mosse lecite), da un test di terminazione (che dice quando è finito il gioco) e da una **funzione di utilità** o **guadagno** (che dice chi vince e di quanto).

- Nei giochi a due giocatori con informazioni perfette, l'algoritmo **minimax** può determinare la mossa migliore per un giocatore (assumendo che l'avversario giochi perfettamente) enumerando l'intero albero di gioco.

- L'algoritmo **alfa-beta** esegue gli stessi calcoli di minimax, ma è più efficiente perché pota i rami dell'albero di ricerca che riesce a dimostrare irrilevanti per il risultato finale.

- Di solito, non è possibile considerare l'intero albero di gioco (anche con alfa-beta), perciò è necessario interrompere la ricerca in qualche punto e applicare una **funzione di valutazione** che dia una stima dell'utilità di uno stato.

Riferimenti

Russell, S., Norvig, P. *Artificial Intelligence - a Modern Approach*, fourth edition, Pearson, 2021.

Nilsson, N.J. *Artificial Intelligence - a New Synthesis*, Morgan Kaufman, 1998.