

Ricerca con avversari: GIOCHI

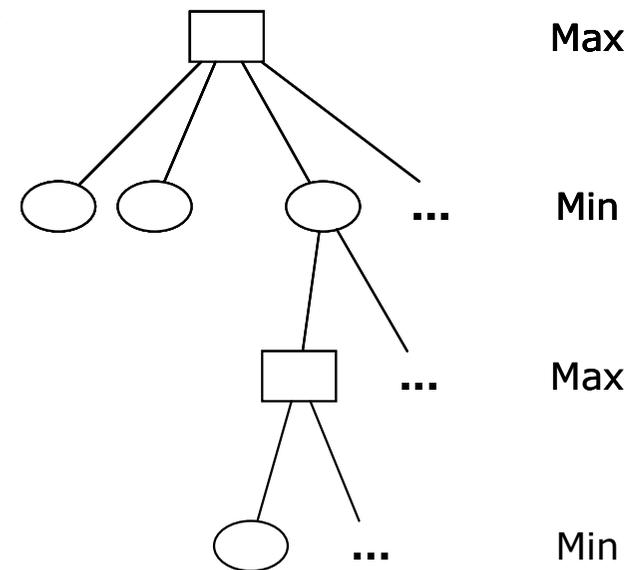
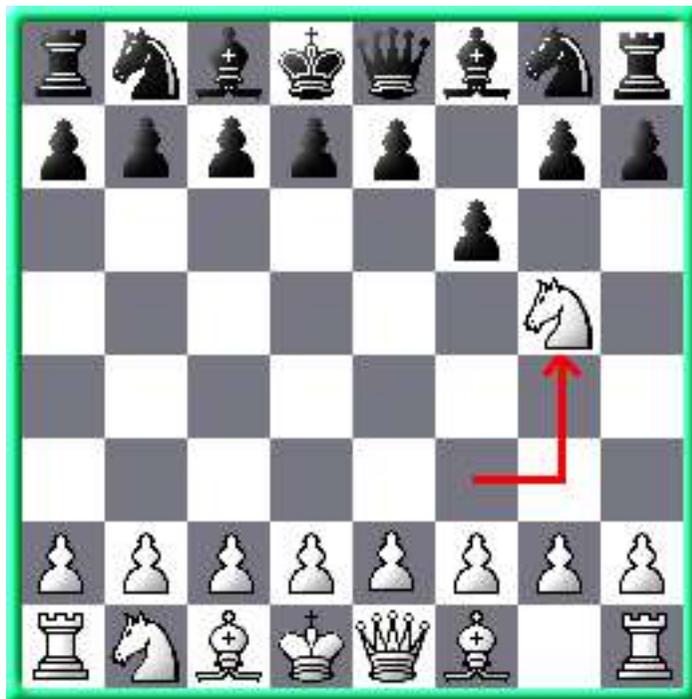
- Ambiente multi-agente che deve tenere conto della presenza di un “avversario”
- Teoria dei giochi → branca dell’economia
- Giochi formali (piu’ che reali), anche se esiste una competizione di calcio fra robot
- Attualmente le macchine hanno superato gli esseri umani in molti giochi quali Othello, Dama, Scacchi, Backgammon e GO.
- Go, battuto recentemente il campione europeo e mondiale (marzo 2016).

GIOCHI

- L'intelligenza artificiale considera giochi con le seguenti proprietà:
 - 1) Sono giochi a due giocatori (min e max) in cui le mosse sono alternate e le funzioni di utilità complementari (vince e perde);
 - 2) Sono giochi con conoscenza perfetta in cui i giocatori hanno la stessa informazione (non tipicamente i giochi di carte quali poker, bridge ecc).
- Lo svolgersi del gioco si può interpretare come un albero in cui la radice è la posizione di partenza e le foglie le posizioni finali.

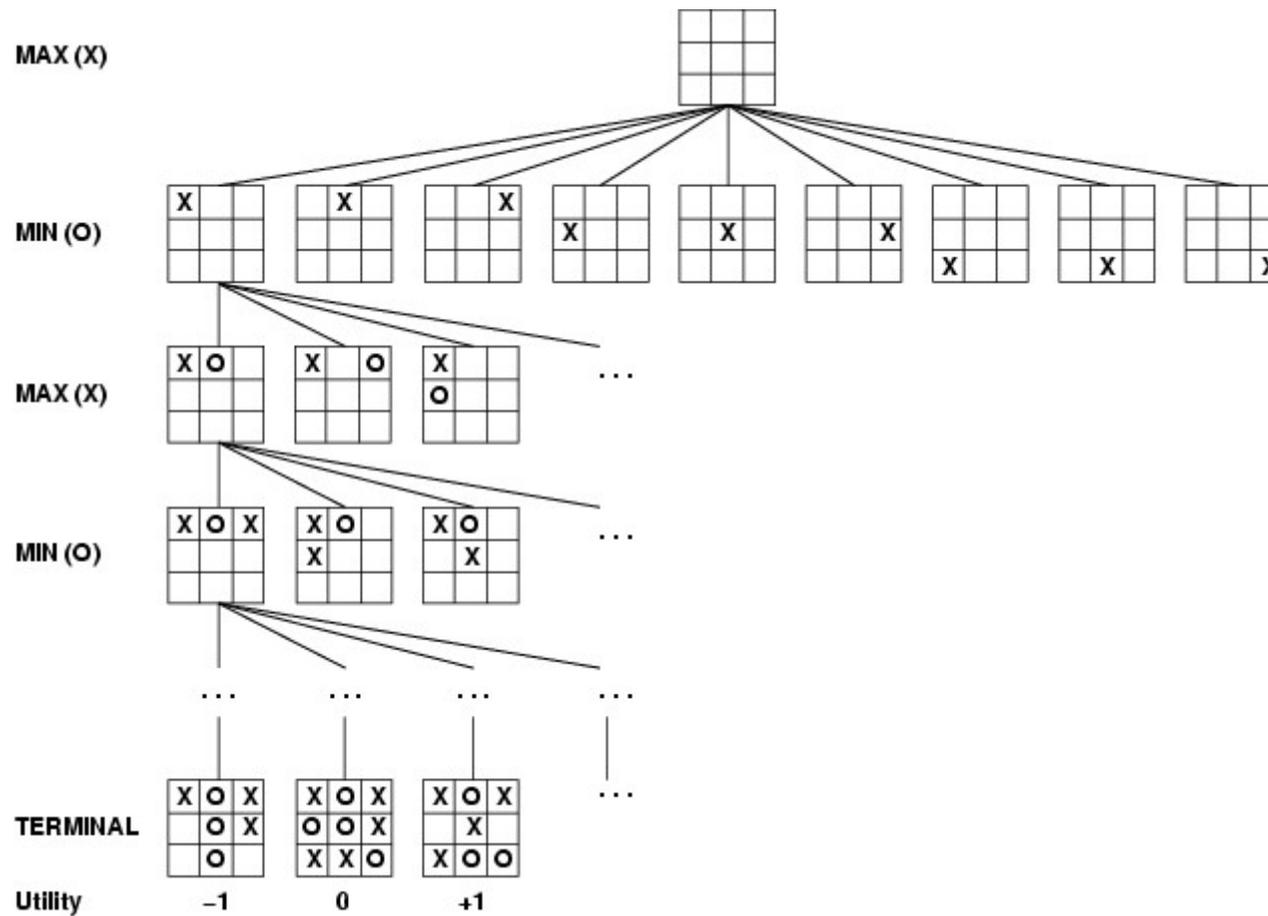


GIOCHI IN IA



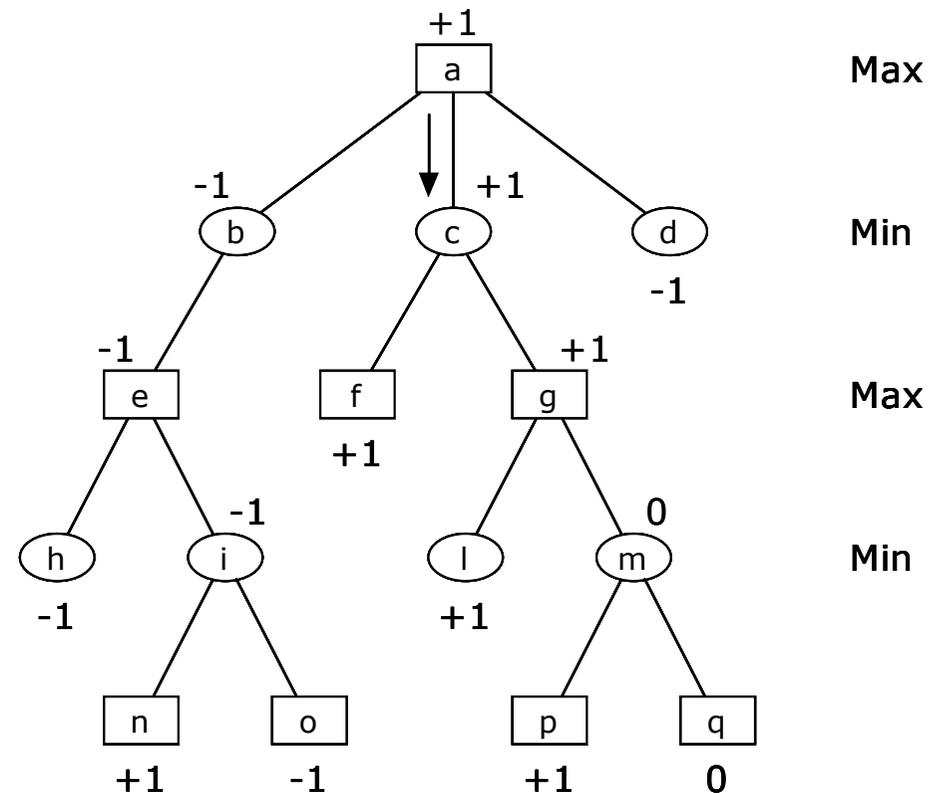
Albero di gioco

Albero di gioco (2-giocatori, deterministico, giocano alternandosi)



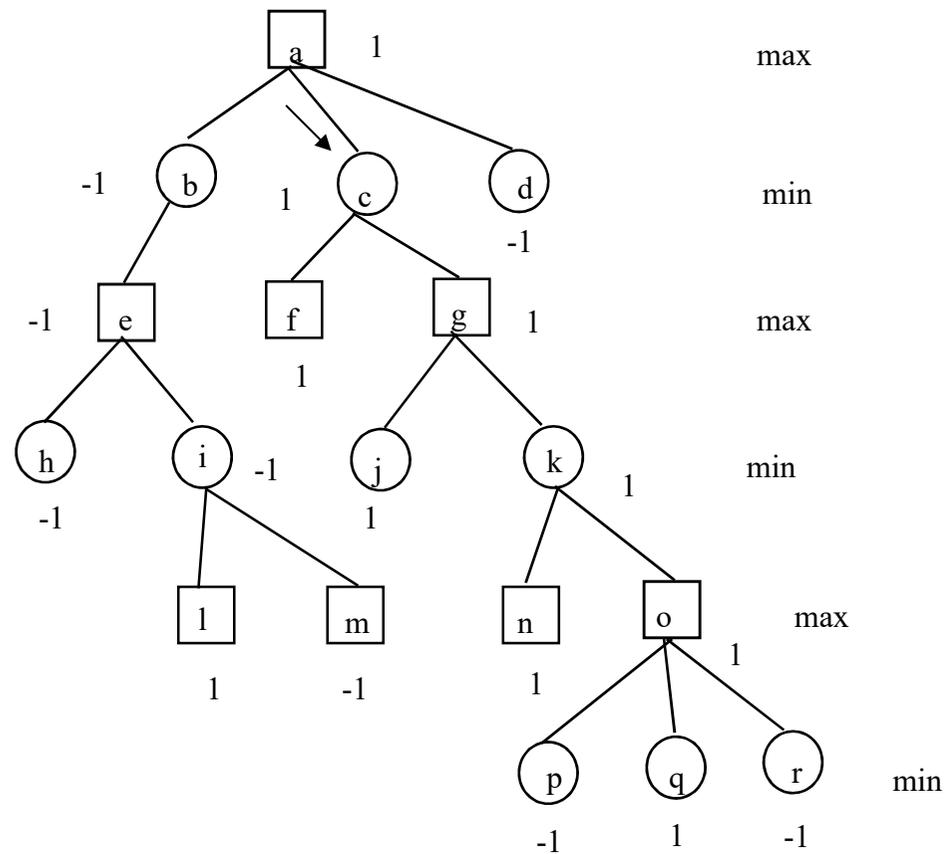
ALGORITMO MIN-MAX

- ❑ L' algoritmo minmax è progettato per determinare la strategia ottimale per "Max" e per suggerirgli, di conseguenza, la prima mossa migliore da compiere; per fare questo, ipotizza che "Min" faccia la scelta a lui più favorevole.
- ❑ Non e' interessante la "strada", ma solo la prossima mossa



MIN-MAX

- Sono etichettate con 1 e -1. Un giocatore cerca di arrivare a -1 (minimizzatore), l'altro a +1 (massimizzatore).



MIN-MAX

- Per i quadri tocca muovere al max, per i cerchi al min.
- Consideriamo il nodo o.
 - Deve muovere il max. Il gioco termina comunque. Può muovere p ed r e perdere, oppure q e vincere. Supponiamo che muova q.
 - o è quindi una posizione vincente. (+1)
- Consideriamo il nodo k.
 - Comunque muova min, perde. Quindi l'etichetta è (+1).
 - Consideriamo il nodo i. Min ha un'opzione vincente dunque (-1).
- Quindi:
 - Un nodo con max che deve muovere ha come label il massimo delle labels dei figli. Viceversa per min.

ALGORITMO MIN-MAX

- Per valutare un nodo n:
 - 1) Espandi l'intero albero sotto n;
 - 2) Valuta le foglie come vincenti per max o min;
 - 3) Seleziona un nodo n' senza etichetta i cui figli sono etichettati. Se non esiste ritorna il valore assegnato ad n;
 - 4) Se n' è un nodo in cui deve muovere min assegna ad esso il valore minimo dei figli, se deve muovere max assegna il valore massimo dei figli. Ritorna a 3.
- Patta: si assegna il valore 0.
- Si possono assegnare dei valori ai nodi che poi vengono aggiornati quando si espandono i figli.
- Complessità in tempo e spazio = b^d

ALGORITMO MIN-MAX (rivisto-> in profondita`)

- Per valutare un nodo n in un albero di gioco:
 - 1) Metti in $L = (n)$ i nodi non ancora espansi.
 - 2) Sia x il primo nodo in L . Se $x = n$ e c'è un valore assegnato a esso ritorna questo valore.
 - 3) Altrimenti se x ha un valore assegnato V_x , sia p il padre di x e V_p il valore provvisorio a esso assegnato.
 - Se p è un nodo min, $V_p = \min(V_p, V_x)$, altrimenti $V_p = \max(V_p, V_x)$. Rimuovi x da L e torna allo step 2.
 - 4) Se ad x non è assegnato alcun valore ed è un nodo terminale, assegnagli $0, -1,$ o 0 . Lascia x in L perchè si dovranno aggiornare gli antenati e ritorna al passo 2.
 - 5) Se a x non è stato assegnato un valore e non è un nodo terminale, assegna a $V_x = -\text{infinito}$ se X è un max e $V_x = +\text{infinito}$ se è un min. Aggiungi i figli di x a L **in testa** e ritorna allo step 2.
 - Complessita` in spazio bd.

Proprietà di Min-Max

- Completo? Sì (se l'albero è finito)
-
- Ottimale? Sì (contro un avversario che gioca al meglio)
-
- Complessità Temporale? $O(b^m)$
-
- Complessità spaziale? $O(bm)$ (depth-first)
-
- Per gli scacchi , $b \approx 35$, $m \approx 100$ per partite “ragionevoli”
→ impensabile tale soluzione!!!
- DOBBIAMO POTARE L' “ALBERO”!!!

ALGORITMO MIN-MAX RIVISTO

- Se devo sviluppare tutto l'albero la procedura è molto inefficiente (esponenziale).
- Se b è il fattore di ramificazione e d sono i livelli allora il numero dei nodi diventa b^d .
- La soluzione (Shannon, 1949): si guarda avanti solo per un po' e si valutano le mosse fino ad un nodo non terminale ritenuto di successo. In pratica si applica minimax fino ad una certa profondità.
- Utilizzo una certa funzione di valutazione per stimare la bontà di un certo nodo.

$e(n) = -1$ sicuramente vincente per min;

$e(n) = +1$ sicuramente vincente per max;

$e(n) = 0$ circa le stesse probabilità;

Poi valori intermedi per $e(n)$.

ESEMPIO

- Negli scacchi sommare i valori dei pezzi che ogni giocatore ha e normalizzare il risultato in modo da avere un valore da +1 o -1.
- Ad esempio somma pesata di valori (lineare)

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

- e.g., $w_1 = 9$ con
 $f_1(s) = (\text{numero di regine bianche}) - (\text{numero di regine nere})$, etc.
potrebbe essere più raffinata tenendo conto delle posizioni relative: il re è difeso? Il pedone protegge un altro pezzo? ecc.
- Trade-off fra ricerca e funzione di valutazione.
- Supponiamo comunque di avere selezionato una funzione di valutazione $e(n)$.

ALGORITMO MIN-MAX RIVISTO II

Per valutare un nodo n in un albero di gioco:

- 1) Metti in $L = (n)$ i nodi non ancora espansi.
- 2) Sia x il primo nodo in L . Se $x = n$ e c'è un valore assegnato a esso ritorna questo valore.
- 3) Altrimenti se x ha un valore assegnato V_x , sia p il padre di x e V_p il valore provvisorio a esso assegnato.
- Se p è un nodo min, $V_p = \min(V_p, V_x)$, altrimenti $V_p = \max(V_p, V_x)$. Rimuovi x da L e torna allo step 2.
- 4) Se ad x non è assegnato alcun valore ed è un nodo terminale, **oppure decidiamo di non espandere l'albero ulteriormente, assegnagli il valore utilizzando la funzione di valutazione $e(x)$** . Lascia x in L perchè si dovranno aggiornare gli antenati e ritorna al passo 2.
- 5) Se a x non è stato assegnato un valore e non è un nodo terminale, assegna a $V_x = -\infty$ se X è un max e $V_x = +\infty$ se è un min. Aggiungi i figli di X a L e ritorna allo step 2.

Algoritmo MIN-MAX versione ricorsiva:

```
function MINIMAX-DECISION(state) returns an action
```

```
  v ← MAX-VALUE(state)
```

```
  return the action in SUCCESSORS(state) with value v
```

```
function MAX-VALUE(state) returns a utility value
```

```
  if TERMINAL-TEST(state) then return UTILITY(state)
```

```
  v ←  $-\infty$ 
```

```
  for a, s in SUCCESSORS(state) do
```

```
    v ← MAX(v, MIN-VALUE(s))
```

```
  return v
```

```
function MIN-VALUE(state) returns a utility value
```

```
  if TERMINAL-TEST(state) then return UTILITY(state)
```

```
  v ←  $\infty$ 
```

```
  for a, s in SUCCESSORS(state) do
```

```
    v ← MIN(v, MAX-VALUE(s))
```

```
  return v
```

Nota: con eval rimpiazza TERMINAL-TEST con:
if CUTOFF-TEST(*state*,*depth*) **then return** EVAL(*state*)
Inoltre aggiorna *depth* ad ogni chiamata ricorsiva

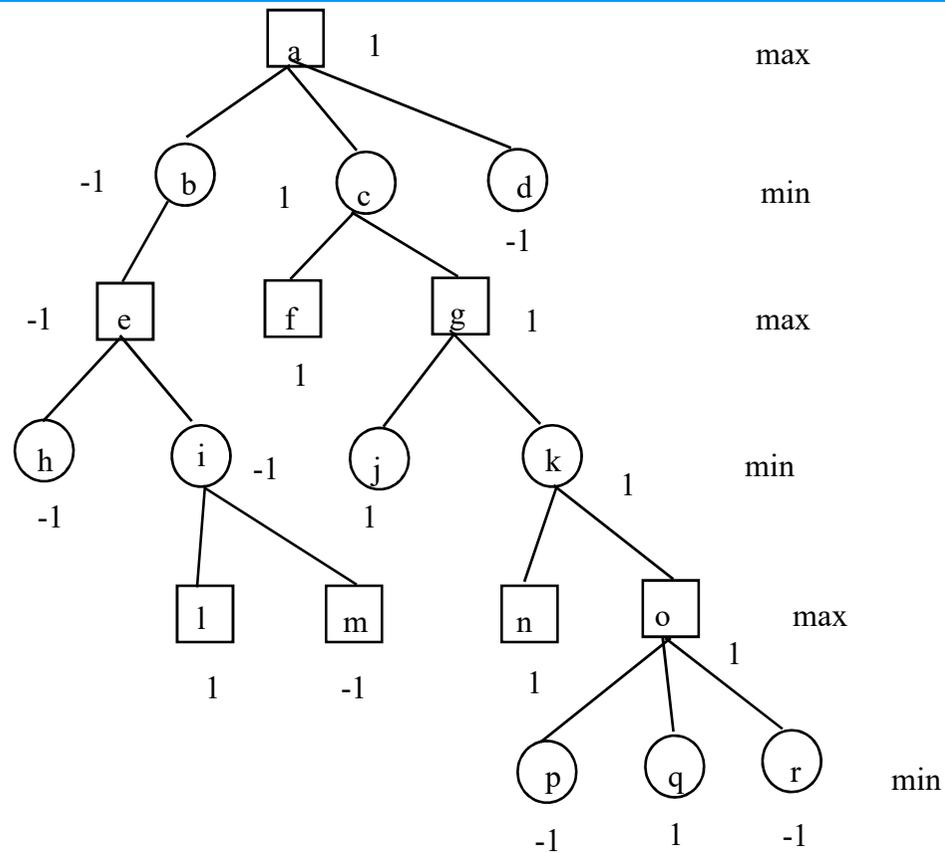
PROBLEMA

- Come decido che non voglio espandere ulteriormente l'albero?
- Nota: se $e(n)$ fosse perfetta non avrei questo problema. Espanderei solo i figli della radice per decidere cosa fare.
- Soluzione possibile e semplice anche dal punto di vista computazionale: espando sempre fino ad una certa profondità p .
- Problemi:
 - Mosse tatticamente più complicate (con valori che si modificano più ampiamente per $e(n)$) dovrebbero essere valutate con più profondità fino alla quiescenza (valori di $e(n)$ che cambiano molto lentamente).
- Effetto orizzonte:
 - Con mosse non particolarmente utili, allungo la profondità dell'albero di ricerca oltre p , se p è la profondità massima, per cui le mosse essenziali non vengono in realtà prese in considerazione.
- Soluzione: a volte conviene fare una ricerca secondaria, mirata sulla mossa scelta.

TAGLI ALFA BETA

- Da tutto quello detto fino ad ora risulta che i computer che giocano semplicemente cercano in alberi secondo certe proprietà matematiche.
- Perciò considerano anche mosse e nodi che non si verificheranno mai.
- Si deve cercare di ridurre lo spazio di ricerca.
- La tecnica più conosciuta è quella del taglio alfa-beta.

ESEMPIO

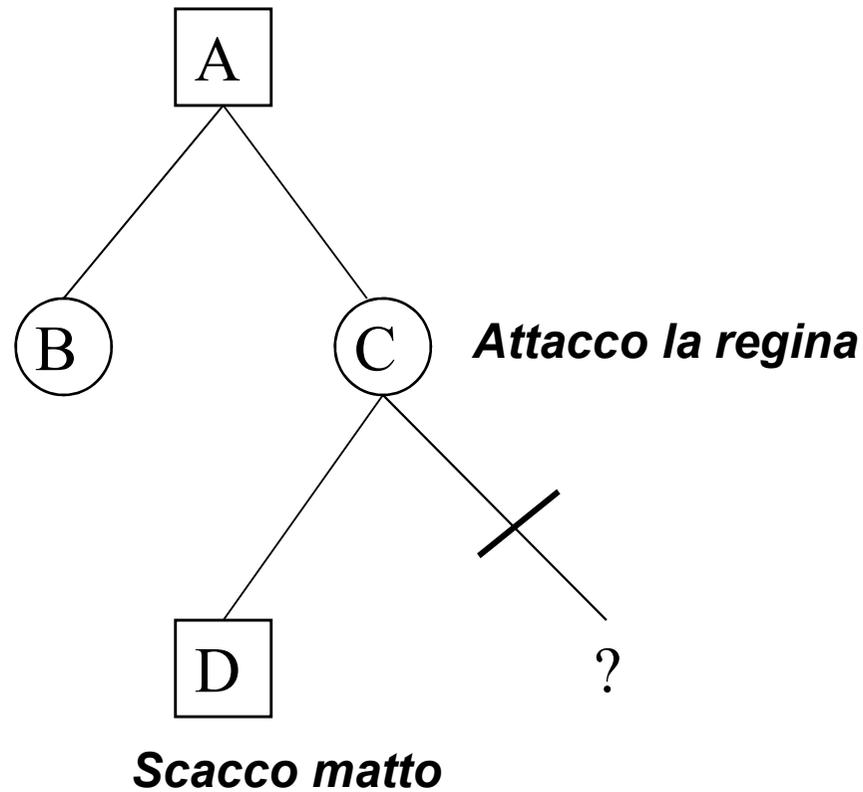


- Quando ho scoperto che la mossa verso c è vincente, non mi interessa espandere i nodi di b e d.
- I nodi sotto b non andranno mai ad influenzare la scelta.

PRINCIPIO GENERALE DEI TAGLI ALFA-BETA

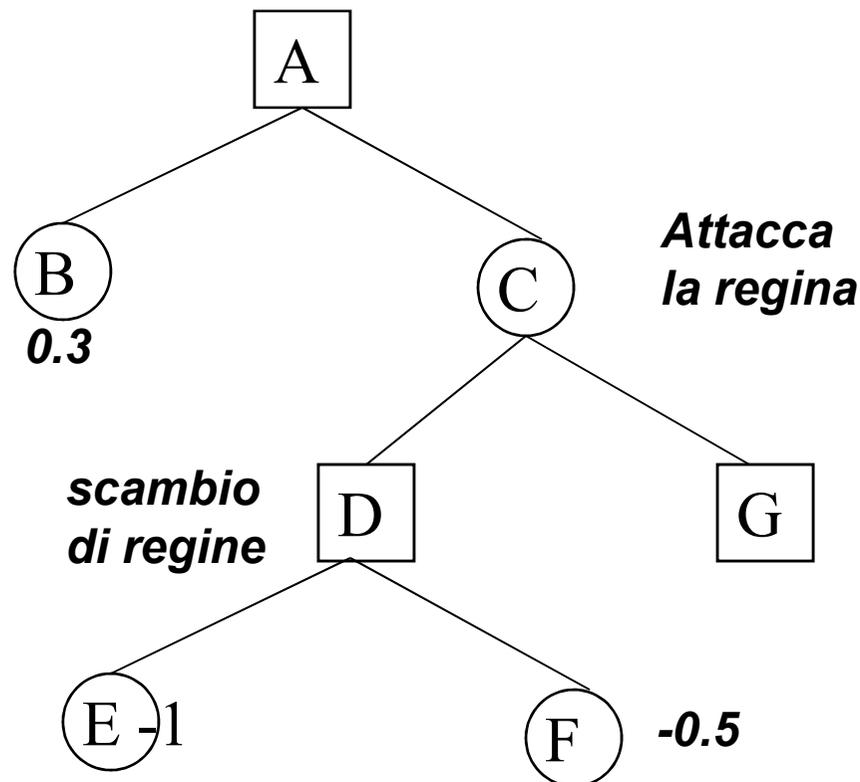
- Si consideri un nodo N nell' albero. Il giocatore si muoverà verso quel nodo?
- Se il giocatore ha una scelta migliore M a livello del nodo genitore od in un qualunque punto di scelta precedente, N non sarà mai selezionato. Se raggiungiamo questa conclusione possiamo eliminare N .
- Sia $ALFA$ il valore della scelta migliore trovata sulla strada di MAX (il più alto) e $BETA$ il valore della scelta migliore trovata sulla strada di MIN (il più basso).
- L' algoritmo aggiorna $ALFA$ e $BETA$ e taglia quando trova valori peggiori.

ESEMPIO



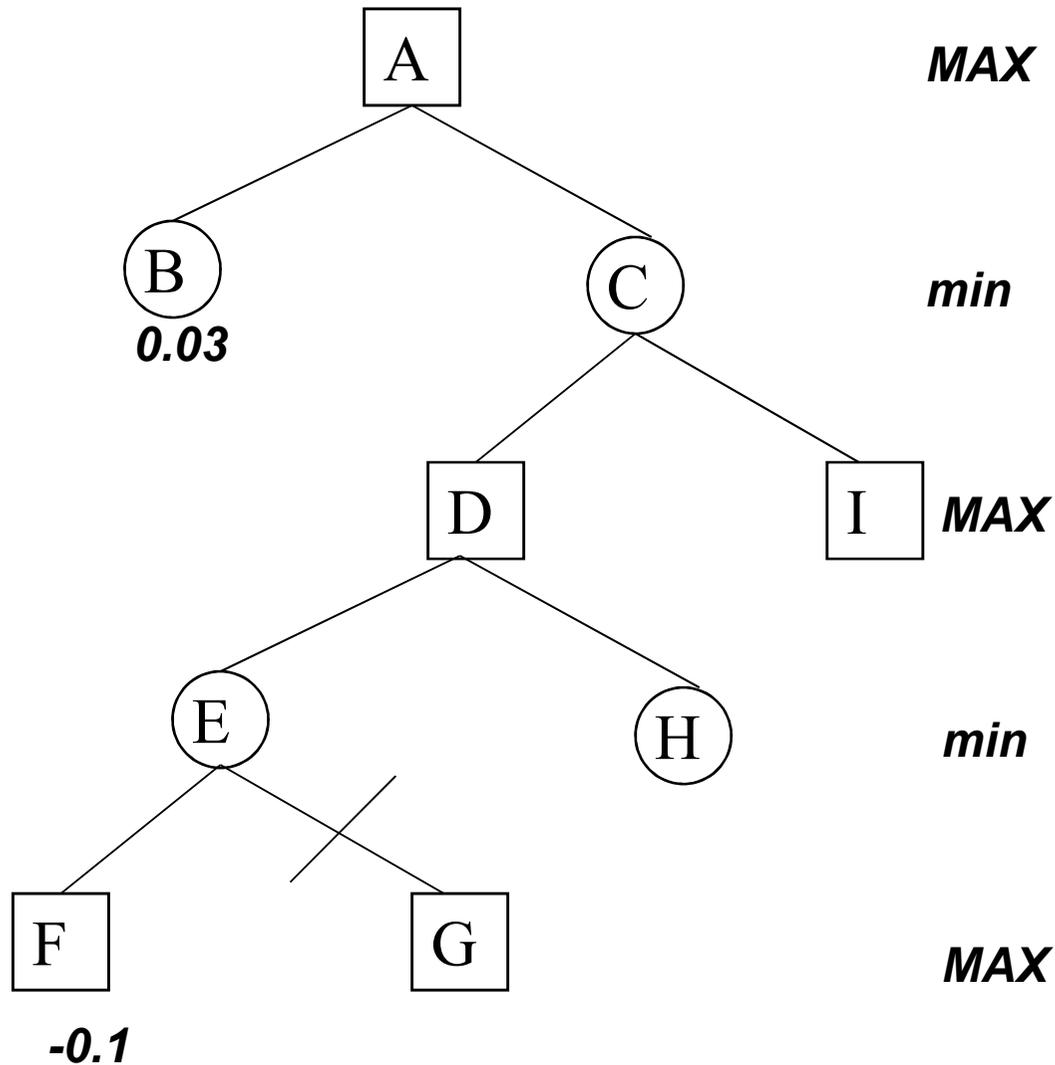
- Non importa che valuti gli altri figli di C! (ho già capito che non mi conviene fare la mossa C).

ALTRO ESEMPIO



- Evito C, perché B è comunque meglio. Al più - 0,5 per max
- Il sottoalbero in G può essere tagliato poiché non mi conviene comunque selezionare C.
Infatti: $C = \min(-0.5, g)$;
 $A = \max(0.3, \min(-0.5, g)) = 0.3$
poiché A è indipendente da G, l'albero sotto G può essere tagliato.

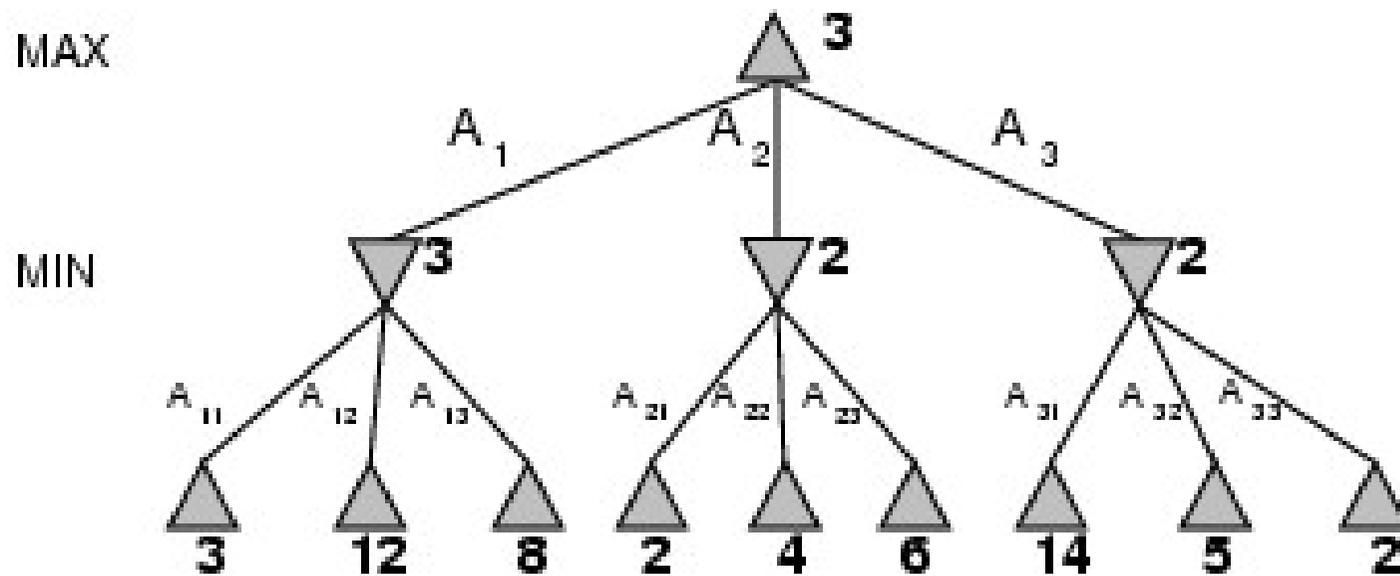
ALTRO ESEMPIO



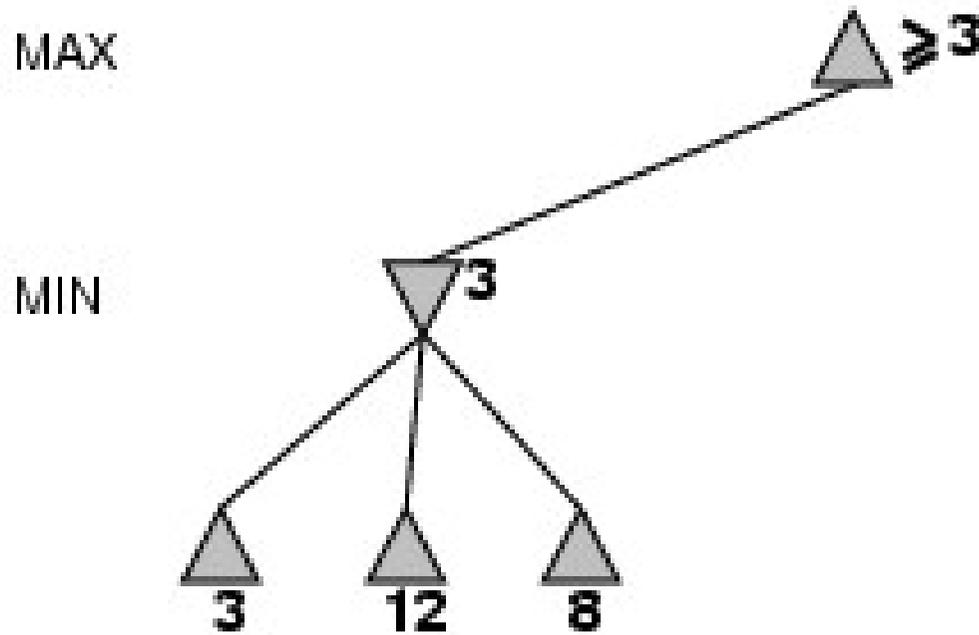
- **G è sulla linea di ricerca che sarà sviluppata?**

- Se G è sulla linea di ricerca, allora anche E lo è. Da E min può sempre ottenere -0.1 che è peggio di .03 per max. Quindi g non può essere nella corrente linea di ricerca.

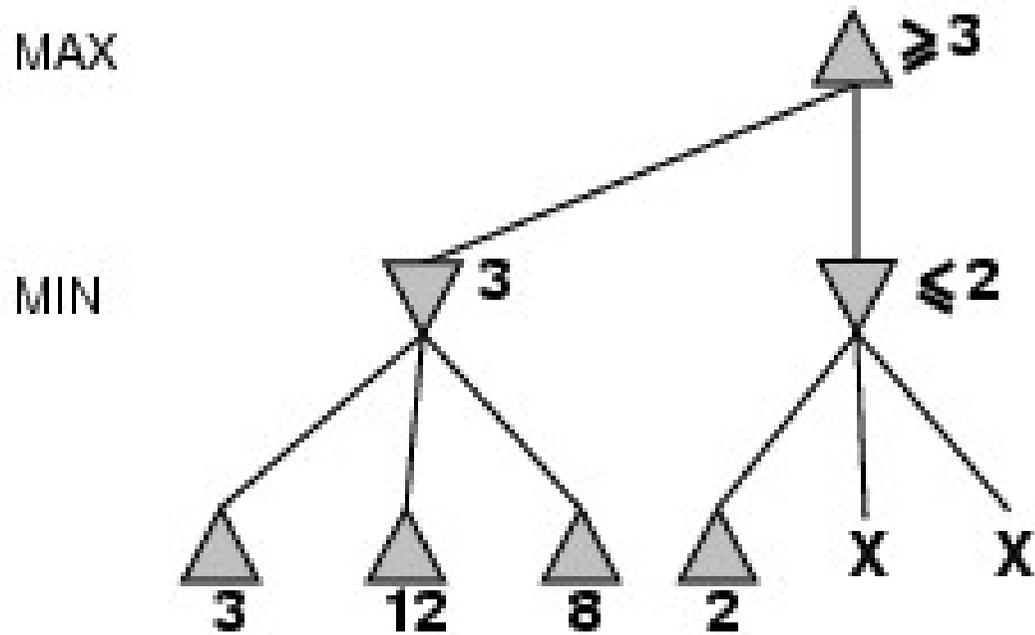
Minimax



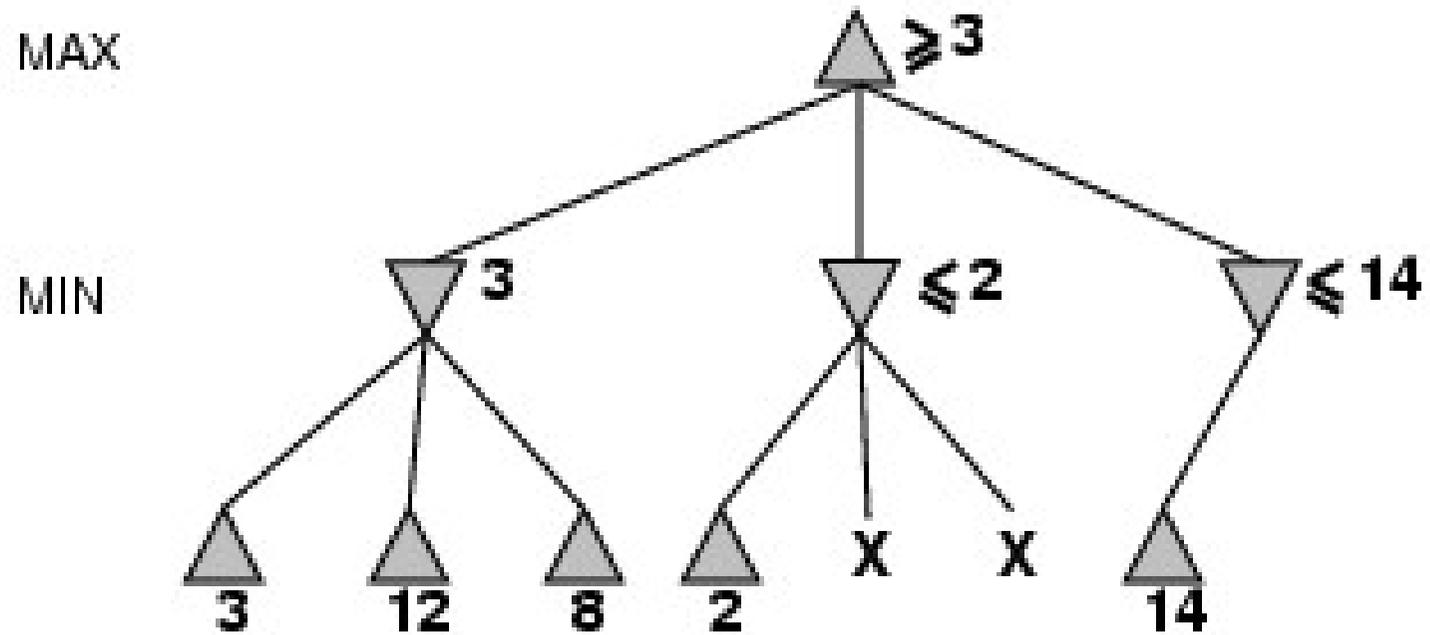
Esempio Tagli α - β



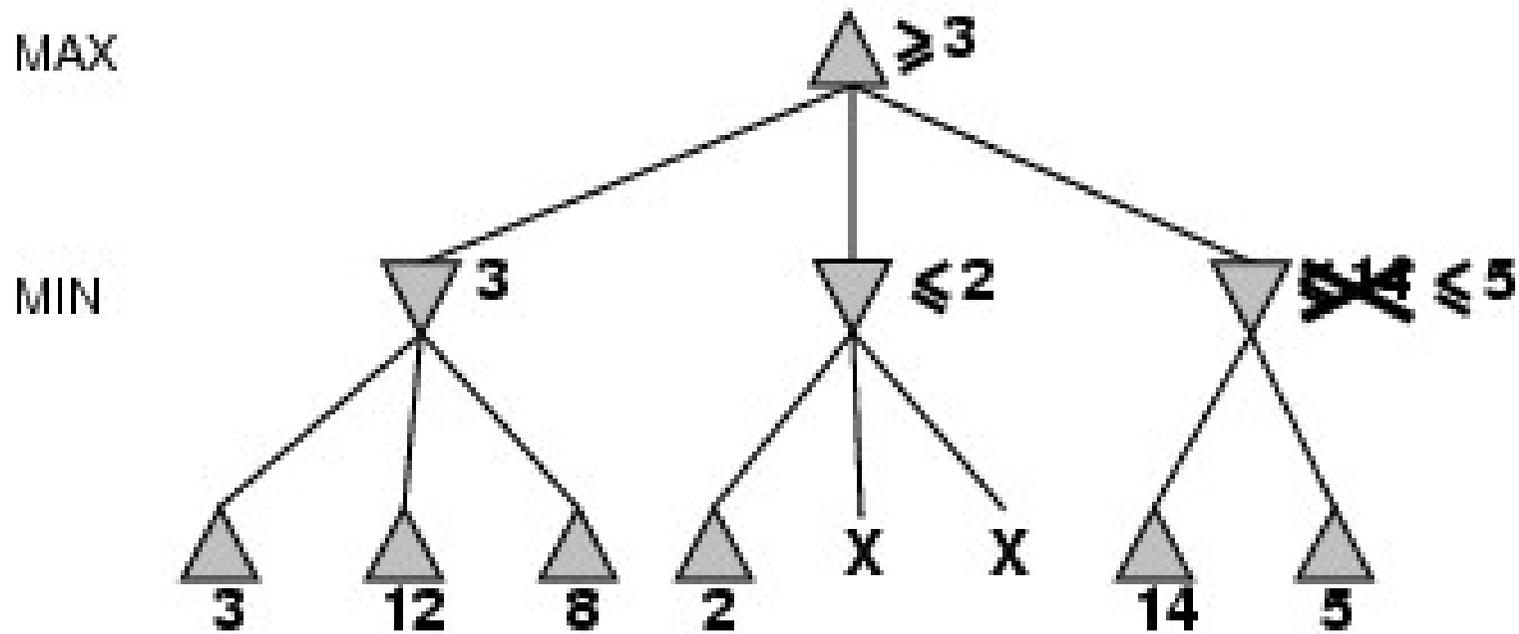
Esempio Tagli α - β



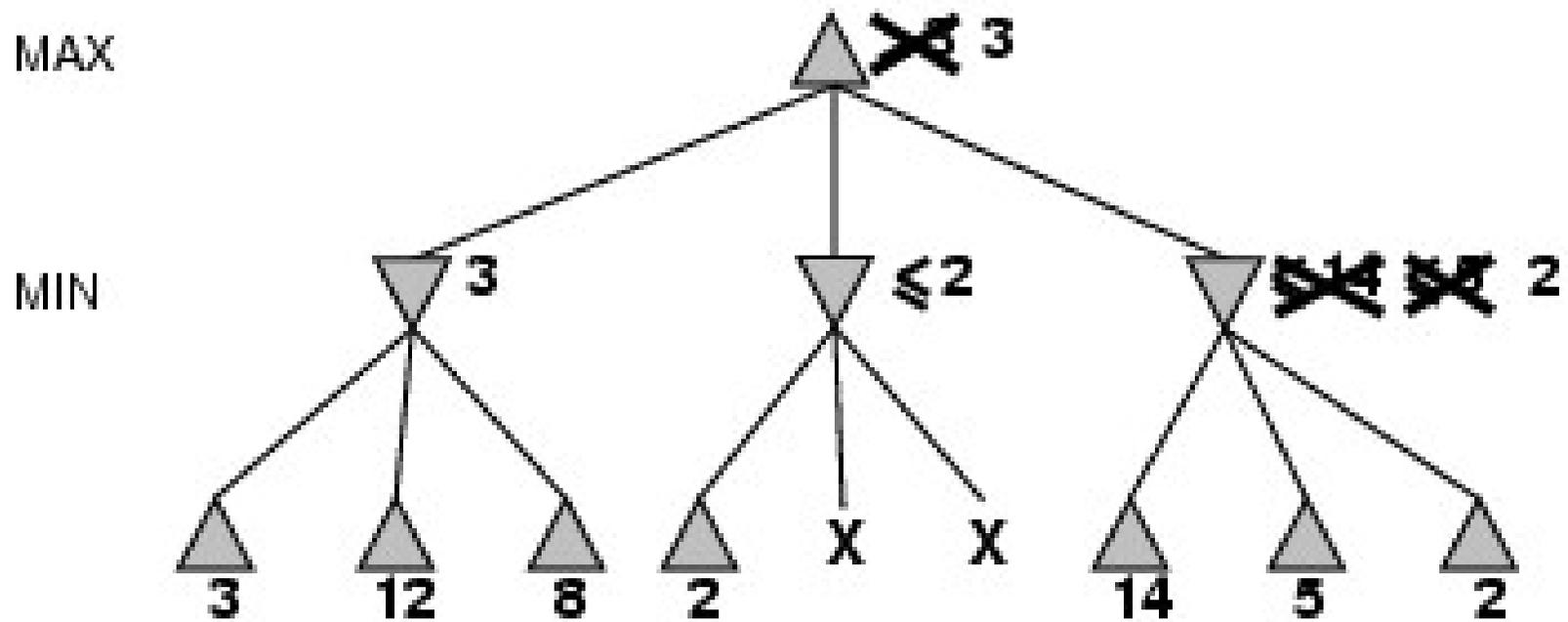
Esempio Tagli α - β



Esempio Tagli α - β

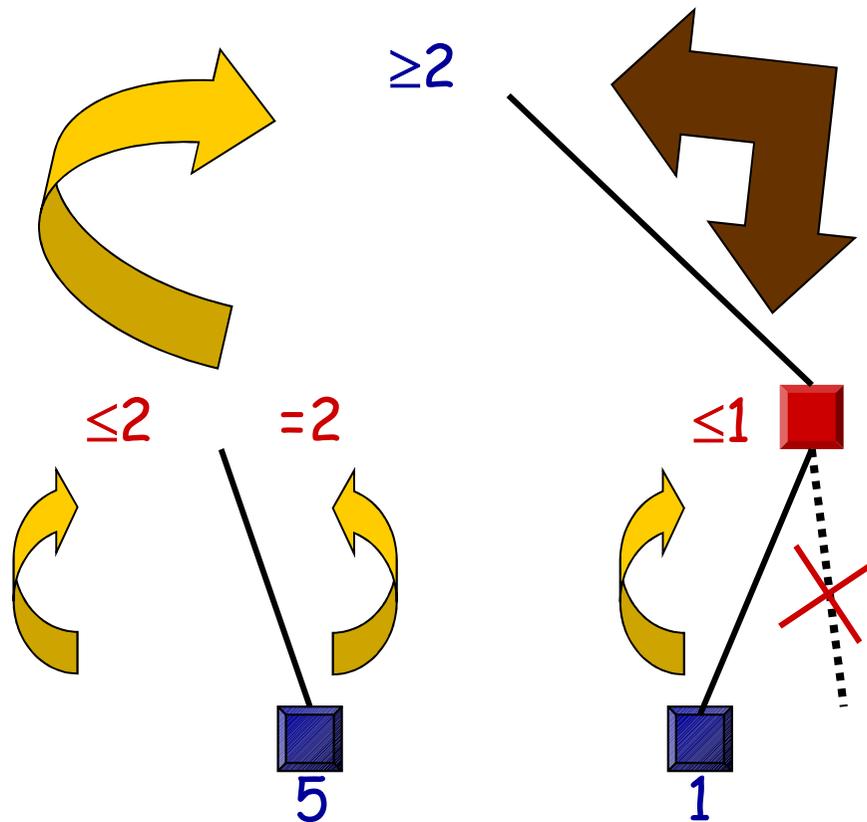


Esempio Tagli α - β



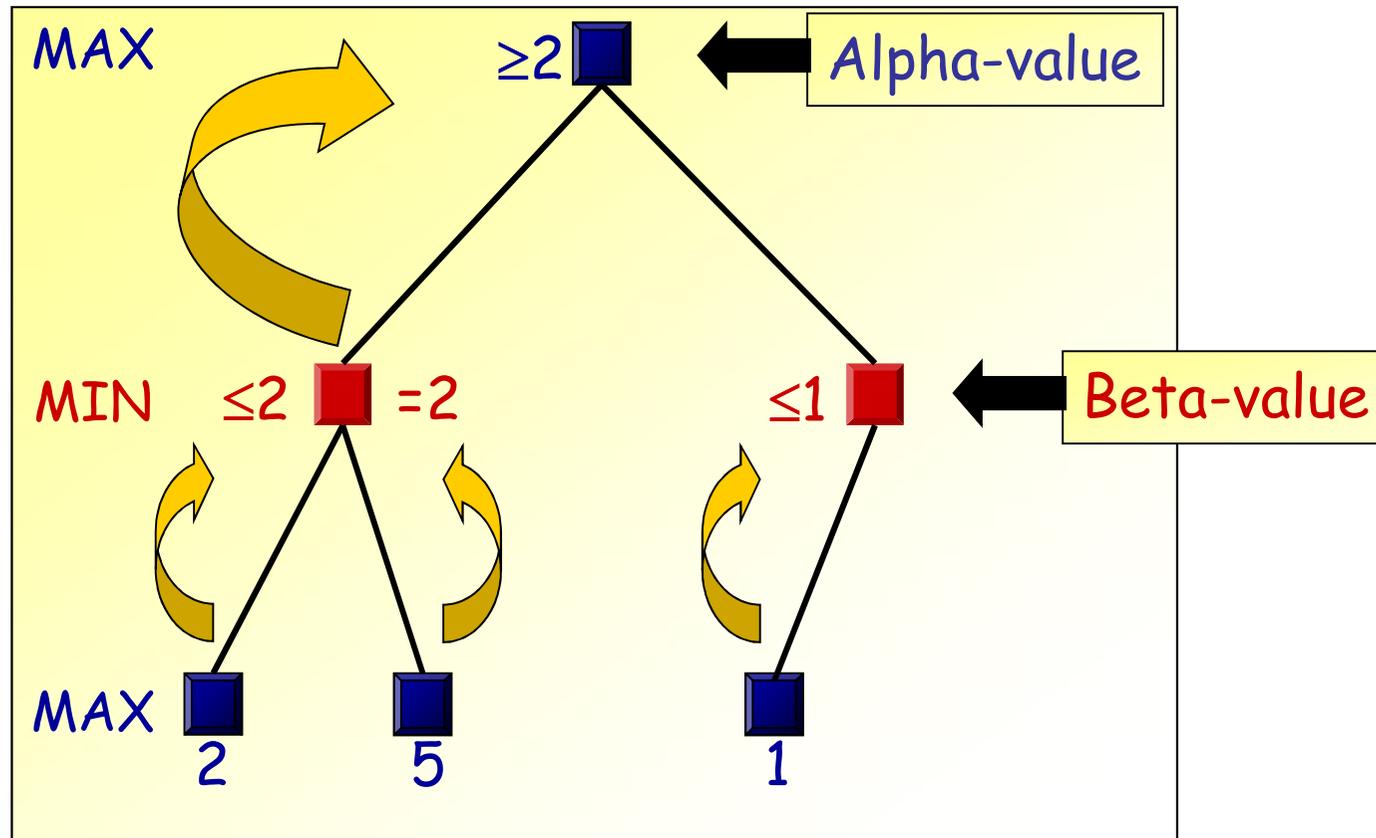
Alpha-Beta idea:

- Principi:
 - si genera l'albero depth-first, left-to-right
 - si propagano i valori (stimati) a partire dalle foglie



Terminologia:

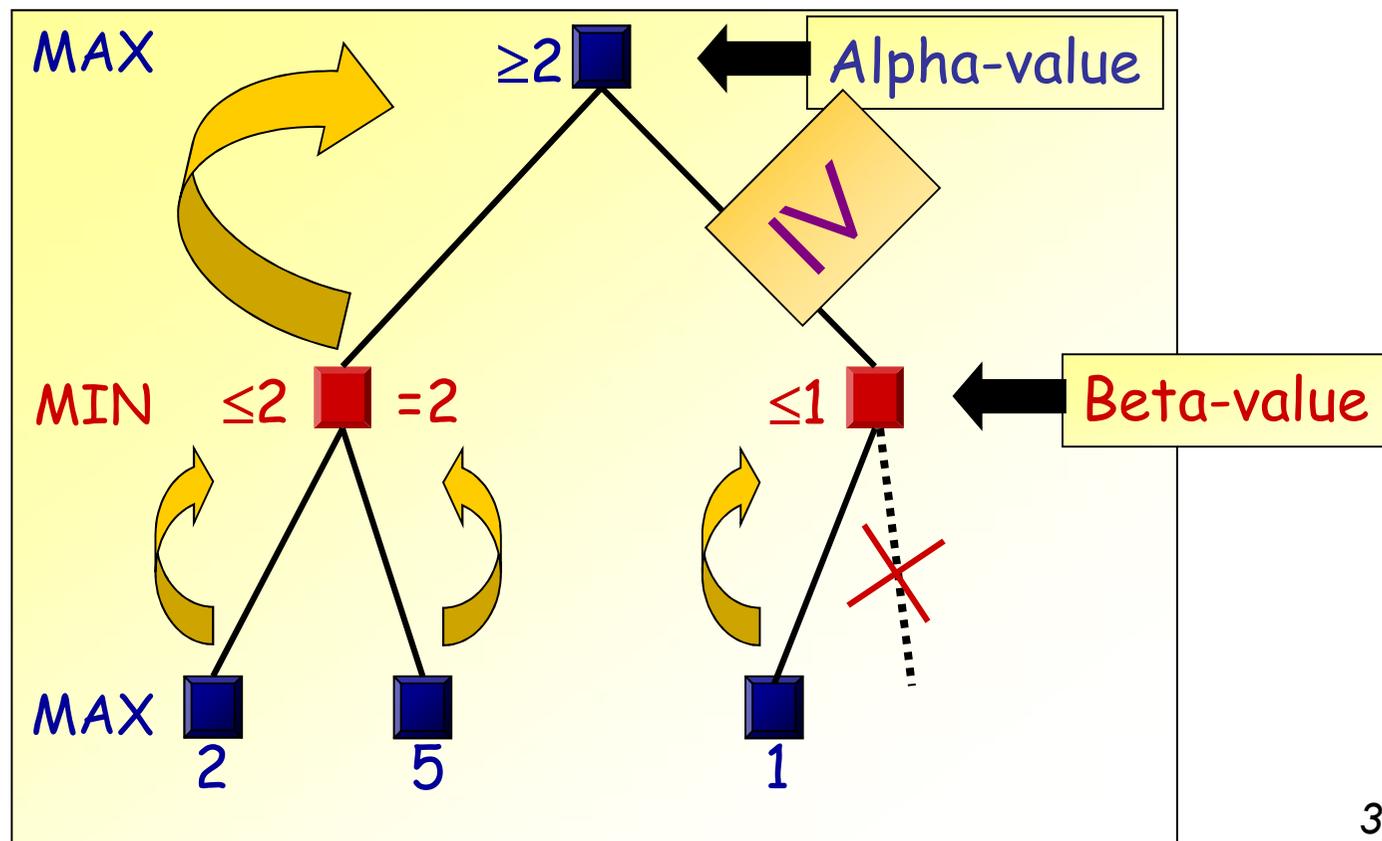
- I (temporanei) valori nei **MAX**-nodes sono **ALPHA-values**
- I (temporanei) valori nei **MIN**-nodes sono **BETA-values**



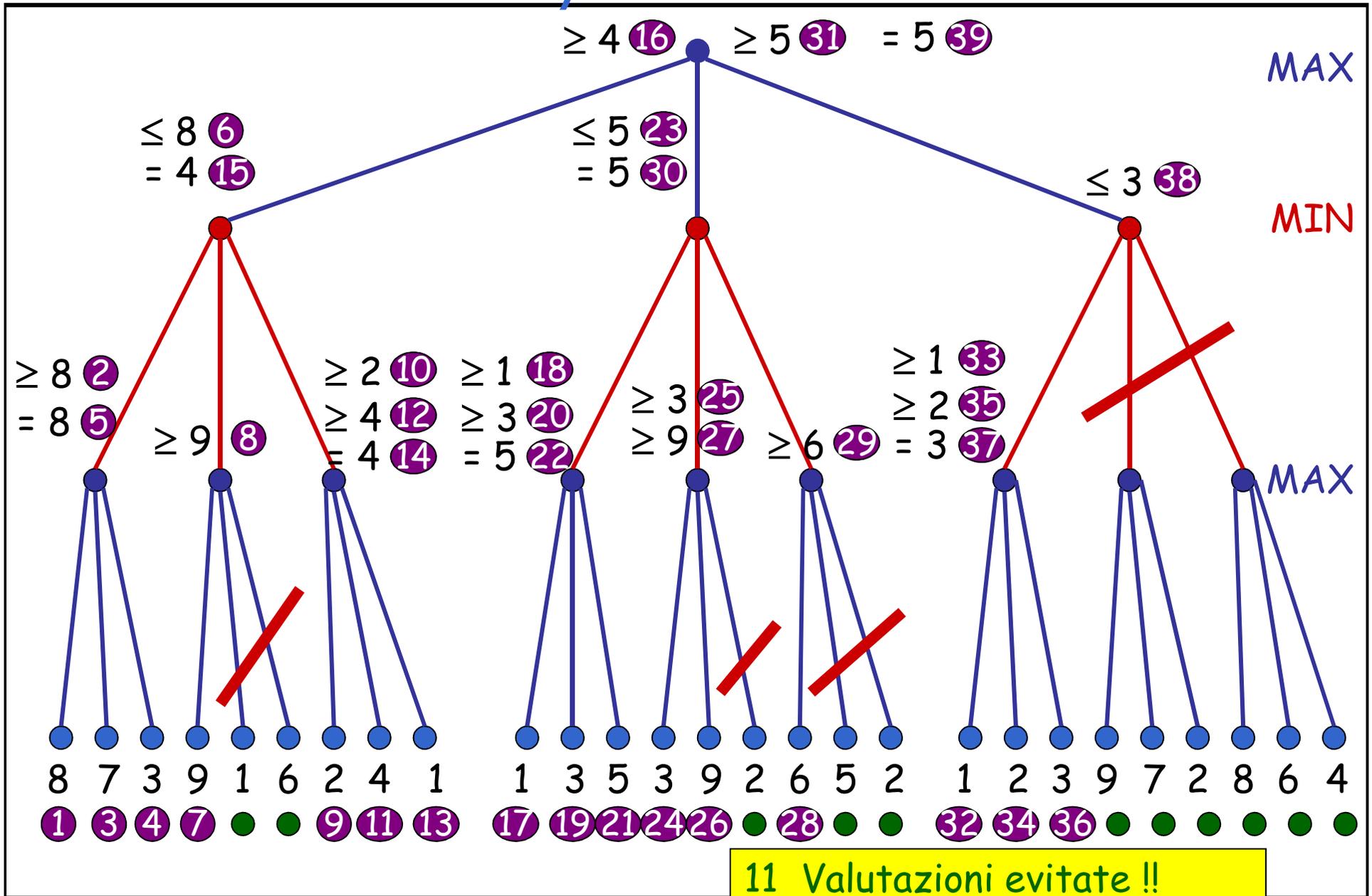
Il principio Alpha-Beta:

- Se un ALPHA-value e maggiore od uguale di un Beta-value di un nodo discendente: stop alla generazione di figli del discendente!

- Se un Beta-value e' piu' piccolo od uguale ad un Alpha-value di un nodo discendente: stop alla generazione dei figli del discendente



Mini-Max con $\alpha-\beta$:



ALGORITMO ALFA-BETA

- Per valutare un nodo n in un albero di gioco:
 - 1) Metti in $L = (n)$ i nodi non ancora espansi.
 - 2) Sia x il primo nodo in L . Se $x = n$ e c'è un valore assegnato a esso ritorna questo valore.
 - 3) Altrimenti se x ha un valore assegnato V_x , sia p il padre di x . Se a x non è assegnato un valore vai al passo 5.
 - Determiniamo se p ed i suoi figli possono essere eliminati dall'albero. Se p è un nodo min, sia α il massimo di tutti i correnti valori assegnati ai fratelli di p e dei nodi min che sono antenati di p .
 - Se non ci sono questi valori $\alpha = -\infty$.
 - Se $V_x \leq \alpha$ rimuovi p e tutti i suoi discendenti da L (dualmente se p è un max).

ALGORITMO ALFA-BETA

- 4) Se p non può essere eliminato, sia V_p il suo valore corrente. Se p è un nodo min, $V_p = \min(V_p, V_x)$, altrimenti $V_p = \max(V_p, V_x)$. Rimuovi x da L e torna allo step 2.
- 5) Se a x non è assegnato alcun valore ed è un nodo terminale, oppure decidiamo di non espandere l'albero ulteriormente, assegnagli il valore utilizzando la funzione di valutazione $e(x)$. Lascia x in L perché si dovranno aggiornare gli antenati e ritorna al passo 2.
- 6) Se a x non è stato assegnato un valore e non è un nodo terminale, assegna a $V_x = -\infty$ se X è un max e $V_x = +\infty$ se è un min. Aggiungi i figli di X ad L e ritorna allo step 2.

ALGORITMO ALFA-BETA

- Il valore che corrisponde ad alfa per gli antenati max è chiamato beta
 - Determiniamo se p ed i suoi figli possono essere eliminati dall'albero. Se p è un nodo max, sia β il massimo di tutti i correnti valori assegnati ai fratelli di p e dei nodi max che sono antenati di p .
 - Se non ci sono questi valori $\beta = +\infty$.
 - Se $V_x \geq \beta$ rimuovi p e tutti i suoi discendenti da L .

Perche' e' chiamata α - β ?

- α e' il valore migliore (i.e., piu' alto) trovato in ogni punto di scelta per max (valore di un nodo min)
- Se v e' peggio di α , *max* lo evitera`
 - taglia quel ramo non appena avrai raggiunto tale conclusione
 - Nel caso di v nodo min, se uno dei suoi figli ha valore minore o uguale di α
- β e' definito in modo simile per *min*

MAX

MIN

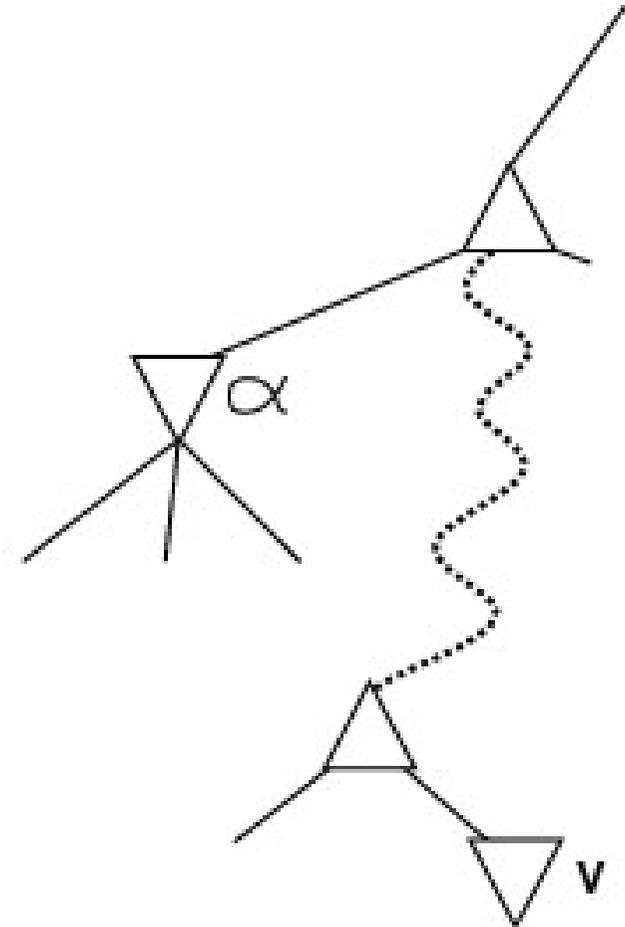
..

..

..

MAX

MIN



The α - β algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action  
  inputs: state, current state in game  
   $v \leftarrow \text{MAX-VALUE}(\textit{state}, -\infty, +\infty)$   
  return the action in SUCCESSORS(state) with value  $v$ 
```

```
function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value  
  inputs: state, current state in game  
            $\alpha$ , the value of the best alternative for MAX along the path to state  
            $\beta$ , the value of the best alternative for MIN along the path to state  
  if TERMINAL-TEST(state) then return UTILITY(state)  
   $v \leftarrow -\infty$   
  for  $a, s$  in SUCCESSORS(state) do  
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$   
    if  $v \geq \beta$  then return  $v$   
     $\alpha \leftarrow \text{MAX}(\alpha, v)$   
  return  $v$ 
```

The α - β algorithm

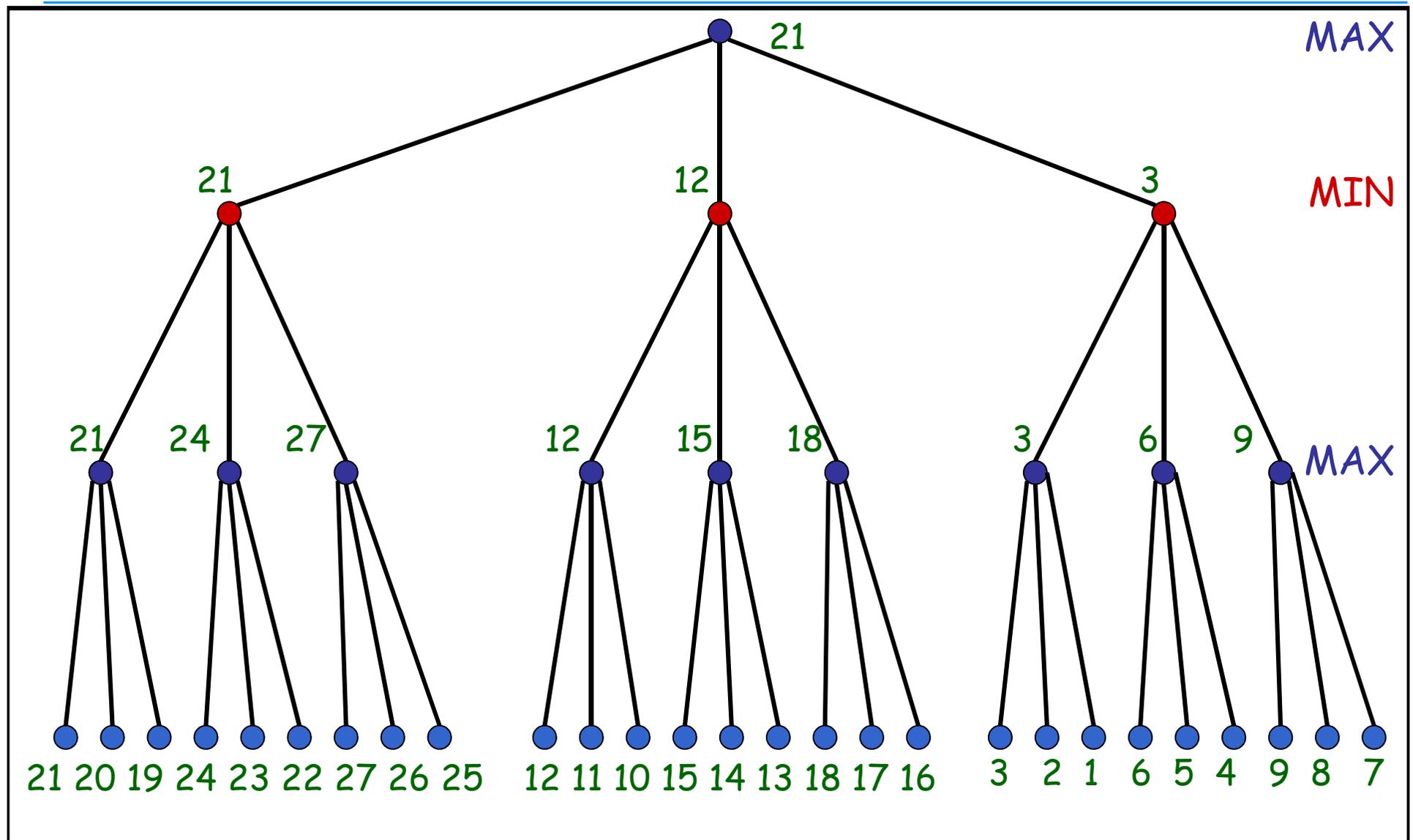
```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  inputs: state, current state in game
            $\alpha$ , the value of the best alternative for MAX along the path to state
            $\beta$ , the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow +\infty$ 
  for  $a, s$  in SUCCESSORS(state) do
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta))$ 
    if  $v \leq \alpha$  then return  $v$ 
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return  $v$ 
```

EFFICACIA DEI TAGLI

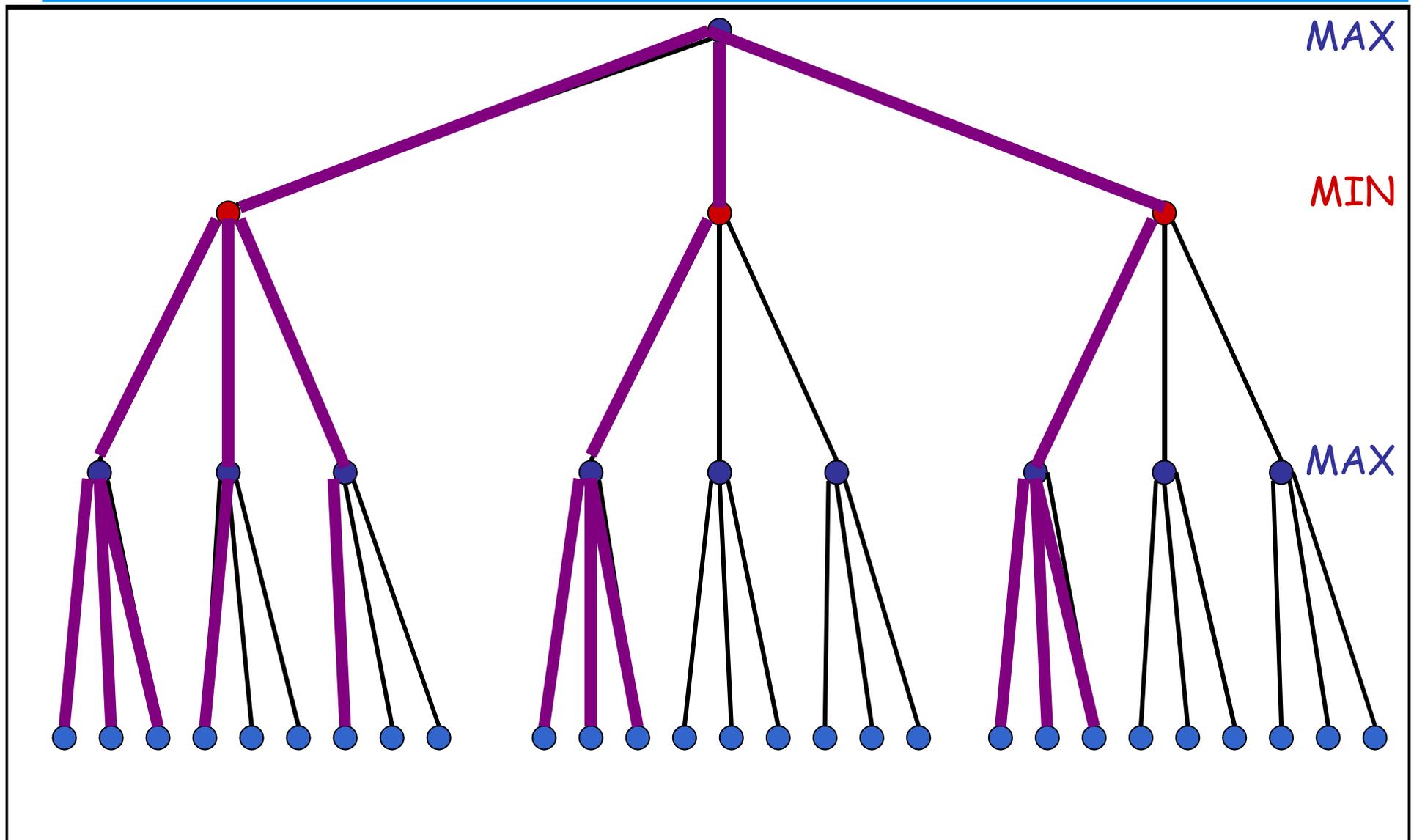
- Ovviamente se valutiamo sempre i nodi peggiori, i nodi valutati successivamente risultano sempre nella linea corrente di ricerca e non c'è nessun taglio.
- Il caso migliore è quando i nodi migliori sono valutati per primi. I restanti vengono sempre tagliati (ovviamente è del tutto teorico).
- In questo caso si va a ridurre il numero dei nodi da b^d a circa $b^{d/2}$. (in pratica, si riduce della radice quadrata il fattore di ramificazione, ovvero si può guardare due volte più avanti nello stesso tempo).
- Giocatore Principiante → Giocatore Esperto
- Nel caso medio con distribuzione casuale dei valori ai nodi, il numero di nodi diventa circa $b^{3d/4}$.
- Quindi è importante ordinare bene i figli di un nodo.
- Si noti inoltre che tutti i risultati qui citati sono per un albero di gioco “ideale” con profondità e ramificazione fissati per tutti i rami e nodi.
- Stati ripetuti, lista dei nodi chiusi (vedi graph search).

Esempio perfetto!



Il caso migliore:

- Ad ogni livello il nodo migliore e' a sinistra.



IL GIOCO DEGLI SCACCHI

- La dimensione del problema è enorme. Solo all'inizio partita le mosse possibili sono 400, diventano più di 144.000 alla seconda
- In particolare, gli scacchi hanno un fattore di ramificazione ~ 35 e ~ 50 mosse per ciascun giocatore. Quindi avremmo 35^{100} nodi. (in realtà le mosse lecite sono 10^{40}).
- Occorre quindi una funzione di valutazione. In particolare, si darà un peso a ciascun pezzo, ma questo non è sufficiente.
- Si deve tener conto anche della posizione relativa dei pezzi (due torri incolonnate valgono a volte più della stessa regina).

ESEMPIO: VALUTAZIONE DI UN CAVALLO

- Il valore materiale di un cavallo è 350 punti. Il principale aggiustamento a tale valore base è dato da un bonus che premia le posizioni centrali, da 0 punti negli angoli, a 100 punti al centro.
- Un altro bonus viene assegnato a quei cavalli che si trovano entro le due case di distanza da un pezzo nemico. Tale bonus varia con l'avanzamento della partita valendo al massimo 4 punti verso la fine del gioco.
- Un terzo bonus viene assegnato a quei cavalli in posizione favorevole rispetto a quella dei pedoni avversari.
- Viene invece inflitta una penalità in base alla distanza da ciascun re, pari ad un punto per ciascuna casa di distanza.

ESEMPIO: VALUTAZIONE DI UN CAVALLO

- Anche il momento della partita è importante. Ad esempio i cavalli sono importanti nel centro partita ma non lo sono in un finale di partita con pochi pezzi.
- Ma anche dare un peso a tutte queste componenti non è sufficiente, occorre anche una funzione che leghi al meglio tutti questi parametri.
- L'altra scelta è di quanto scendere in profondità nell'albero delle soluzioni. Ci si aspetta che la macchina risponda in un tempo paragonabile a quello di un giocatore umano.
- Un computer medio elabora circa 1000 posizioni al secondo (ma può arrivare anche a 2.500).
- Ogni mossa richiede al massimo 150 secondi. Quindi un computer elabora circa 150.000 mosse possibili che corrispondono a circa 3-4 livelli giocando ad un livello da principiante).

TAGLI ALFA BETA: diventano essenziali

- Gli attuali programmi scendono circa di 7 livelli ed elaborano circa 250.000 posizioni per volta ma in particolari condizioni possono arrivare fino a 20 livelli e 700.000 posizioni.
- Inoltre quasi tutti i programmi utilizzano il tempo che il giocatore umano impiega per scegliere la sua mossa per esplorare altre strade.
- Il giocatore umano, in realtà sembra non scenda mai per più di 5 livelli, e con tagli notevoli. Non utilizza poi una funzione di valutazione definita in modo metodico (usa il "colpo d'occhio").

TAGLI ALFA BETA: diventano essenziali

- Il computer non è in grado di adottare una strategia globale, ma questa limitazione è spesso compensata da non commettere sviste o dimenticanze.
- Tutti i programmi di scacchi, inoltre, consultano la libreria delle aperture (ci sono un centinaio di aperture ormai completamente esplorate e che possono condizionare tutta la partita).
- Mentre il computer è fortissimo nel centro partita, il giocatore umano è più abile nel finale, dove la strategia posizionale è meno importante. Ma oggi i programmi di scacchi, proprio per ovviare a questo inconveniente, tendono a utilizzare librerie ed algoritmi specializzati per il finale.

DEEPBLUE

- Il 10/5/1997, a New York, una macchina ha battuto in un match di sei partite il campione del mondo (match DeepBlue – Kasparov – 2-1 e tre patte).
- Esiste un sistema di valutazione della forza di gioco (Elo) capace di misurare il progresso dei giocatori umani ed artificiali.
- I punti si guadagnano in tornei ufficiali:
 - Giocatore principiante: 500 punti Elo
 - Maestro: 2.200
 - Campione del Mondo: 2.800
 - Deep Blue: 3.000
- I giocatori artificiali possono classificarsi in due categorie in base al fatto che utilizzino hardware generico (PC) o hardware speciale (e' il caso di Deep Blue).
- In particolare, Deep Blue utilizza una macchina parallela general-purpose a 32 processori più 512 chip specializzati per la generazione di mosse e valutazione.

DEEPBLUE (continua)

- I grandi successi dei giocatori artificiali si sono verificati a partire dagli anni 80 di pari passo con i progressi delle tecnologie VLSI (la tecnica e' circa la stessa degli anni 60', ma e' aumentata la potenza di calcolo).
- L' approccio "forza bruta" si è rivelato il più pagante.
- Deep Blue arriva a esplorare alberi profondi 12/14 semimosse ($\sim 10^{11}$ posizioni) in circa 3 minuti. L' esplorazione piu' conveniente e' iterative deepening.
- Si calcola che ogni semimossa in più ci fa guadagnare circa 50/100 punti Elo.
- Nel futuro i giocatori artificiali giocheranno sempre meglio...
- Quindi gli scacchi si può considerare un sistema quasi risolto.

Altri approcci: deep learning e metodi sub-simbolici

- DeepMind, piccola azienda di Londra di cui è stato cofondatore, nel 2011, il britannico Demis Hassabis, bambino prodigio degli scacchi, designer di videogame e neuroscienziato computazionale.
- Nel 2014 DeepMind è stata acquistata per centinaia di milioni di dollari da Google.
- Algoritmo basato su deep learning (reti neurali) e apprendimento che impara da solo a giocare (anche ai videogiochi), e spesso molto meglio dei giocatori umani.
- Non si utilizzano tecniche ad hoc e specifiche come per gli scacchi e Watson.
- L'algoritmo non impara un solo gioco, e lo hanno allenato a ben 49 diversi giochi per Atari 2600, tutti sviluppati per generazioni di adolescenti.
- Ultimo caso, è riuscito a battere al gioco GO il campione europeo ad Ottobre 2015.

REPUBBLICA 9 Marzo 2016

Supercomputer di Google batte il campione del mondo di Go

Il primo round della storica sfida tra intelligenza artificiale e uomo è andato al programma AlphaGo della Google Deepmind, che ha sgominato il sudcoreano Lee Sedol, considerato il più bravo nel millenario gioco da tavola cinese.

Go è un gioco da tavolo nato in Cina oltre 2500 anni fa e molto diffuso in Asia orientale. Giocato da più di 40 milioni di persone in tutto il mondo, ha regole semplici: i giocatori, a turno, devono posizionare le pietre bianche o nere su un tavolo, cercando di catturare pietre dell'avversario o dominare gli spazi vuoti per conquistare il territorio. Ma la semplicità delle regole nasconde un gioco di profonda complessità: le possibili posizioni sono superiori al numero di atomi dell'universo

GO: un gioco antichissimo e molto complesso



La sfida: AlphaGo batte il campione mondiale

