

Strategie di ricerca con avversario

Studieremo alcune strategie che tengono conto della presenza di altri agenti che possono manipolare l'ambiente in cui operiamo. Tali agenti sono supposti avere obiettivi che contrastano con quelli che vorremmo conseguire. Le loro azioni non saranno quindi cooperative ma competitive.

Ambienti competitivi

- Ambienti **multi-agente**
- Ogni agente è guidato da **obiettivi**
- Gli obiettivi di agenti diversi sono **conflittuali**, cioè il conseguimento degli obiettivi di un agente impedisce il conseguimento degli obiettivi degli altri agenti
- I problemi di ricerca con avversario sono anche detti **giochi**



I giochi nell'AI e in altre discipline

Marvin Minsky (1968):

*“i giochi non vengono scelti perché sono chiari e semplici,
ma perché ci danno la massima complessità con le minime
strutture iniziali”*

Pungolo Scientifico

- **Matematica**: teoria dei grafi e complessità
- **Computer Science**: AI, database, calcolo parallelo, etc.
- **Economia**: teoria dei giochi, eco. cognitiva/sperim.
- **Psicologia**: fiducia, rischio, etc..

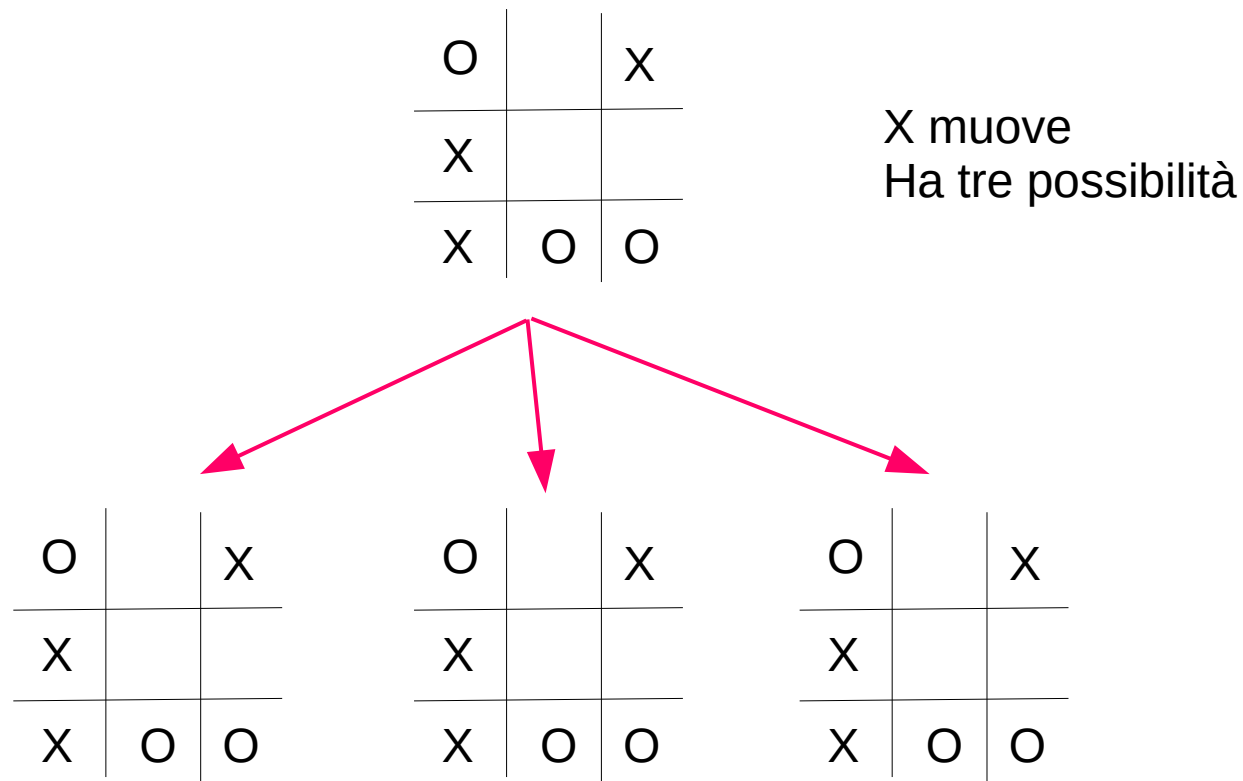
Tipologie di giochi

- **Condizioni di scelta**
 - **Ad informazione “perfetta”**: gli stati del gioco sono totalmente espliciti per tutti gli agenti
 - **Ad informazione “imperfetta”**: gli stati del gioco sono solo solo parzialmente esplicitati
- **Effetti della scelta**
 - **Deterministici**: gli stati sono determinati unicamente dalle azioni degli agenti
 - **Stocastici**: gli stati sono determinati anche da fattori esterni (es: dadi)

Esempi

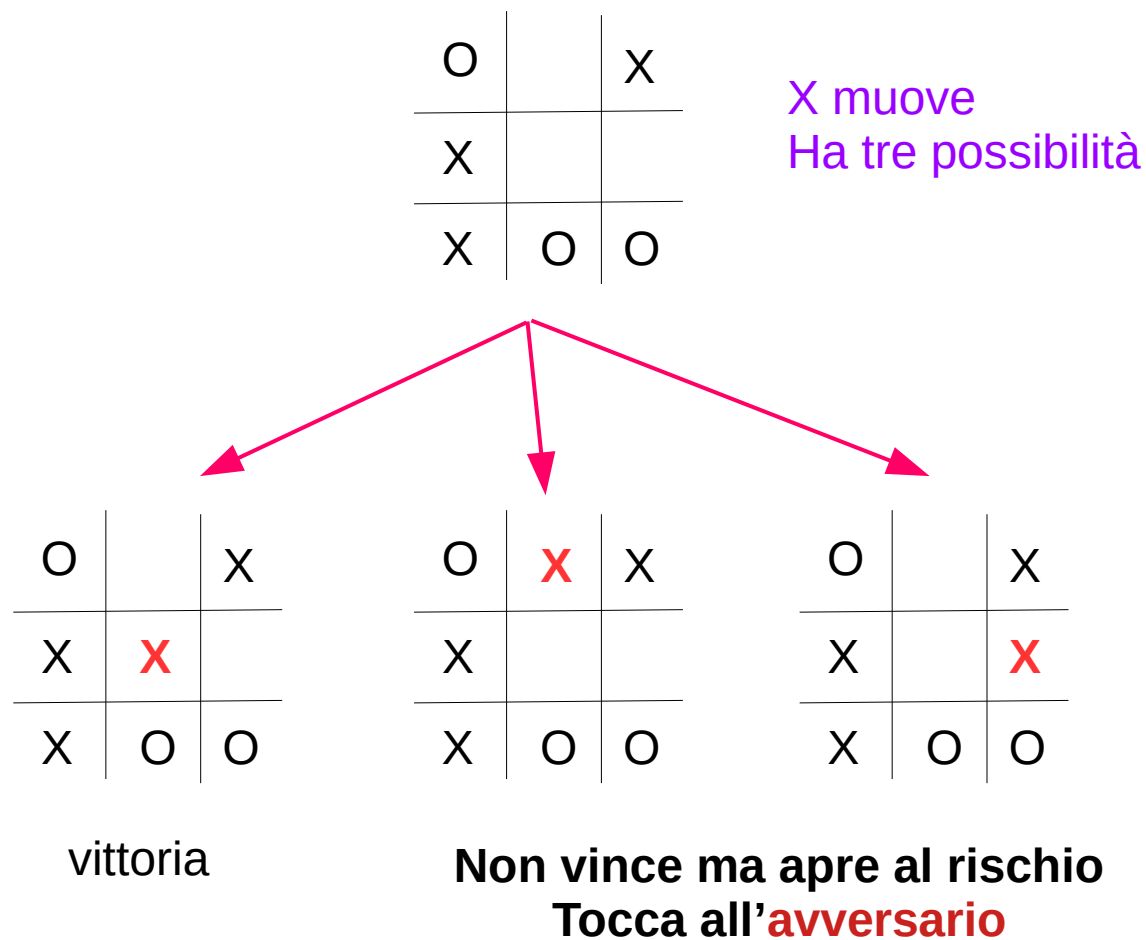
	Informazione Perfetta	Informazione Imperfetta
Giochi deterministici	Scacchi, Go, Dama, Otello, Forza4, tris	MasterMind
Giochi stocastici	Backgammon, Monopoli	Scarabeo, Bridge, Poker... (giochi di carte) Risiko

Esempio: gioco del tris



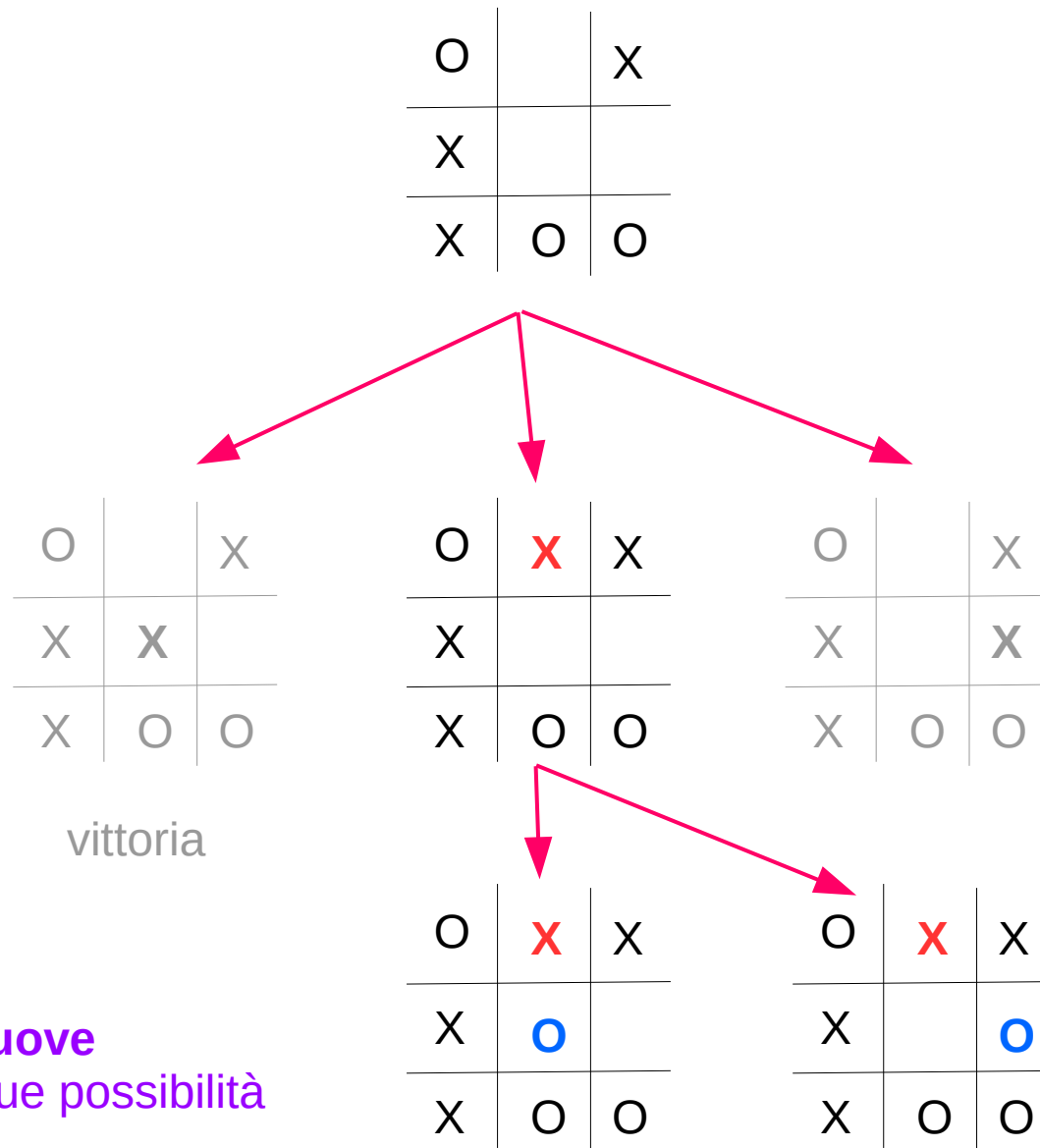
L'esplorazione pare simile a quanto già visto nella risoluzione automatica di problemi

Esempio: gioco del tris



L'ambiente è multi-agente!
L'evoluzione dello stato è solo parzialmente controllabile

Esempio: gioco del tris

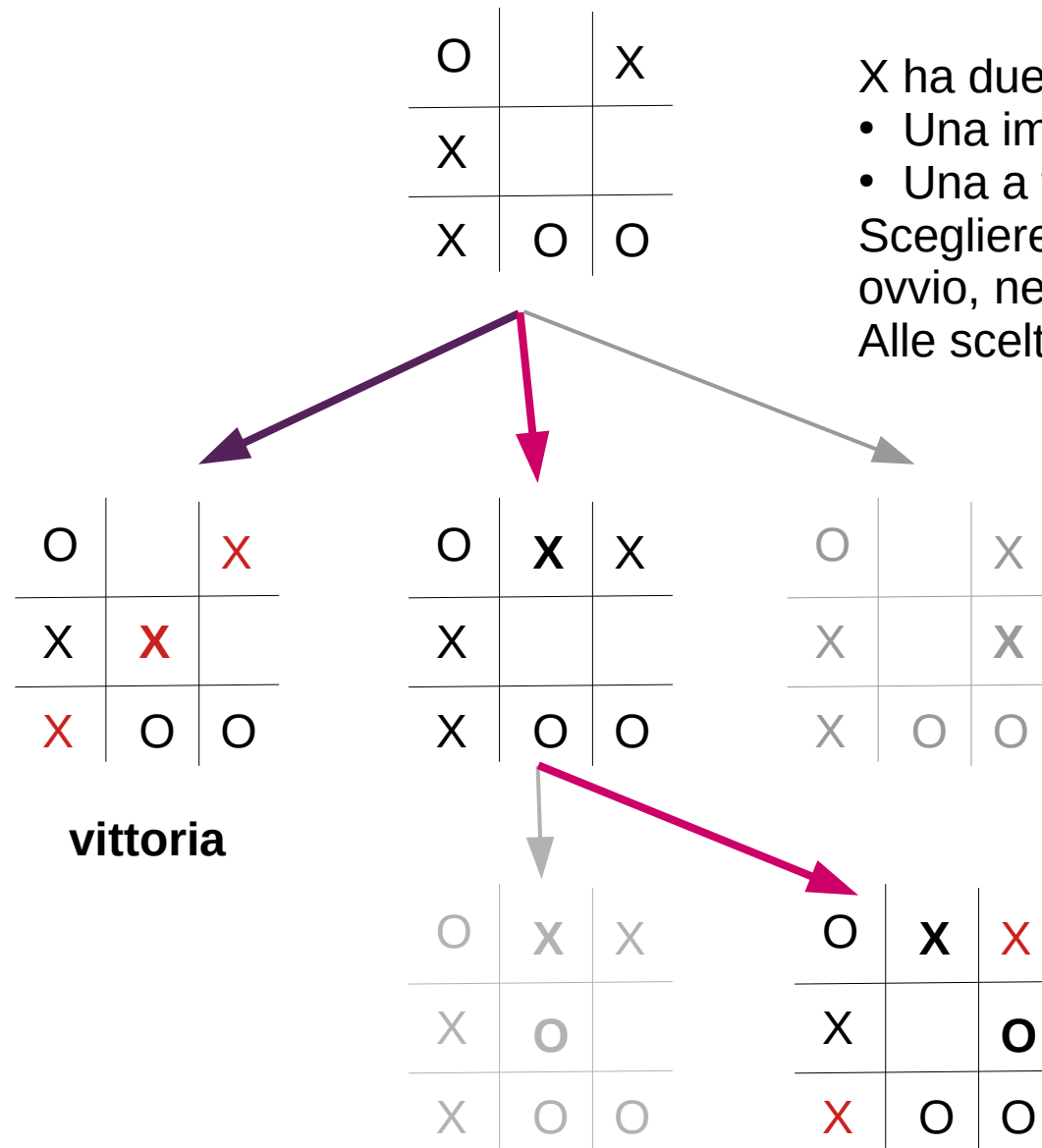


O muove
Ha due possibilità

Vittoria di O

**La vittoria sarà di X
al prossimo turno**

Esempio: due strategie per la vittoria

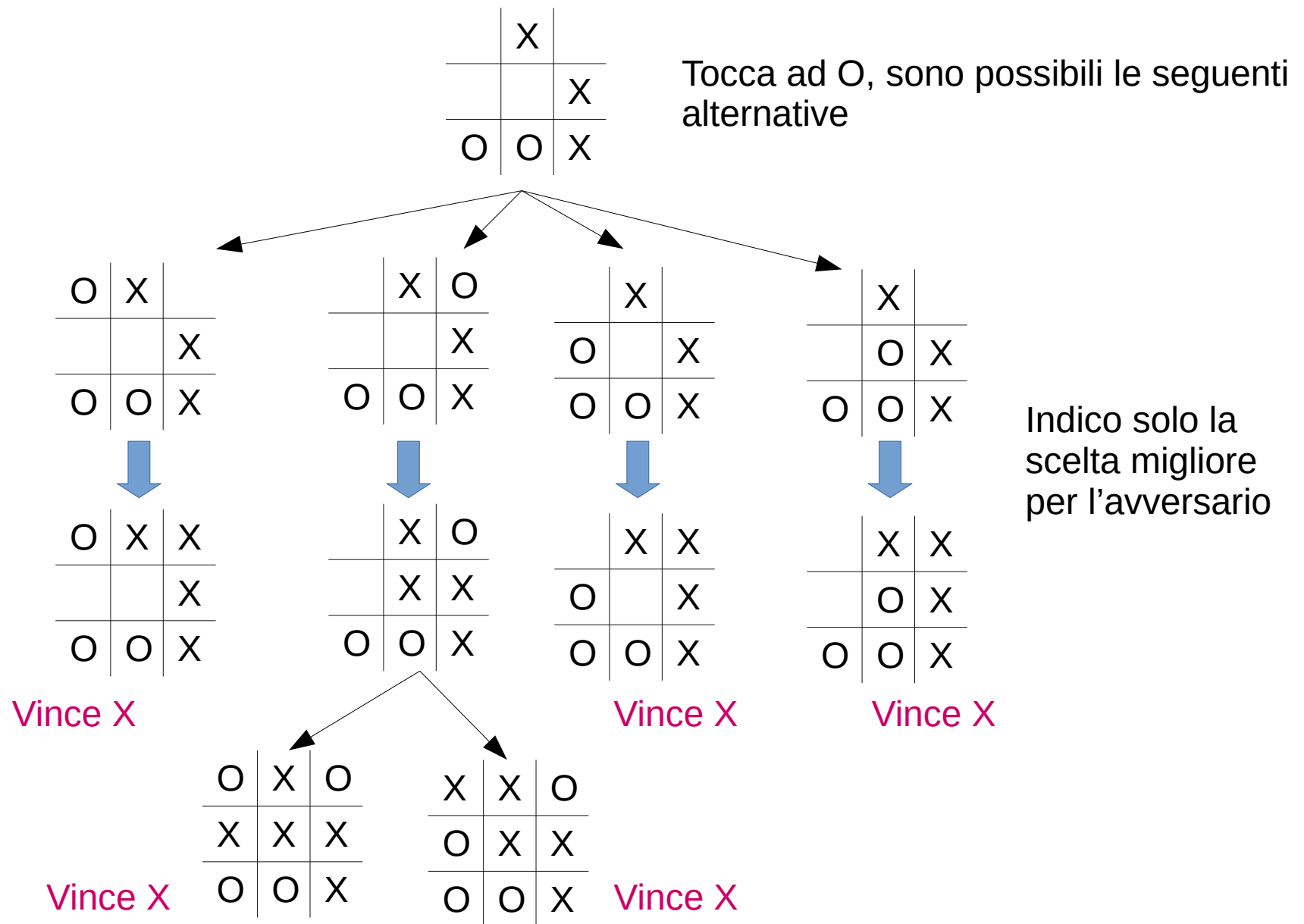


X ha due strategie per la vittoria:

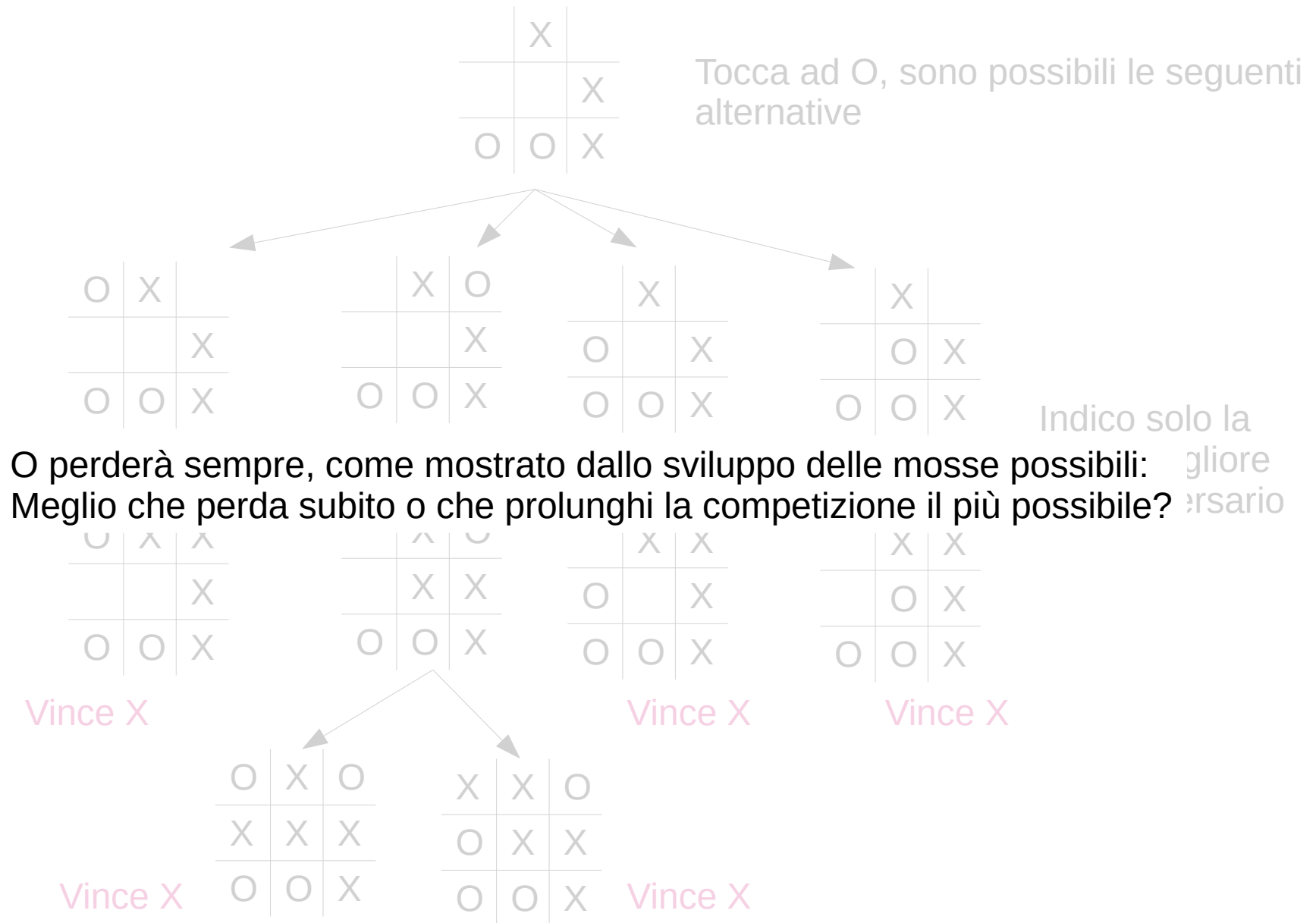
- Una immediata
- Una a tre passi

Scegliere quella immediata sembra ovvio, nell'altro caso ci esponiamo Alle scelte del secondo giocatore

Esempio: e in caso di sconfitta?



Esempio: e in caso di perdita?



Differenze con ricerca (non) informata

- L'obiettivo del giocatore è **determinare una strategia** (sequenza di mosse) che porti alla **vittoria**
- **$g(x)$** non si usa perché guadagno/perdita non dipendono dal costo delle azioni, che possiamo immaginare uniforme
- I **nodi terminali** sono categorizzabili in: **vittoria, sconfitta, parità**
- **Anche l'avversario muove!** Quindi la scelta del nodo **successore non è sempre controllabile**

Giochi a somma zero

- **Un gioco a somma zero** è un contesto di interazione multiagente in cui la perdita (o il guadagno) di un agente è compensata dal guadagno (perdita) degli altri
- **Esempio: le fette di torta**
quando le fette sono irregolari la parte in più di chi ha la fetta più grande è compensata dalla parte in meno che gli altri non ottengono



Approcci maximax, maximin, minimax regret (nella teoria delle decisioni)

- Approccio maximax (ottimistico)
- Approccio maximin (pessimistico o conservativo)
- Approccio minimax regret (“pentimento” minimax)
- Esempio: occorre scegliere quale investimento fare. È data una tabella dei payoff

alternative	sale	stabile	scende
fondo1	40	45	5
fondo2	70	30	-20
azioni	55	40	-10

Approcci maximax, maximin, minimax regret (teoria delle decisioni)

Andamenti: non sono controllabili,
dipendono da dinamiche esterne

alternative	sale	stabile	scende
fondo1	40	45	5
fondo2	70	30	-20
azioni	55	40	-10

Scelte, se ne può fare
una sola

Payoff: possono essere guadagno o
perdite ma anche riferirsi ad altre misure
(esempio: tempo, risorse). Sono specifici
del problema

Approccio maximax

Guarda i payoff più alti per ogni possibile scelta e fa la scelta che promette di più in assoluto (fondo2 dà 70): **il massimo dei massimi**. E' ottimistica perché non ha garanzie che le dinamiche esterne faranno salire il fondo scelto (nell'esempio).

alternative	sale	stabile	scende
fondo1	40	45	5
fondo2	70	30	-20
azioni	55	40	-10

Approccio maximin

Questo approccio guarda le perdite maggiori legate a ciascuna scelta e poi esegue l'azione che minimizza le perdite (**il massimo dei minimi**). Nell'esempio sceglierà fondo 1 che è l'unica opzione a non andare in perdita.

alternative	sale	stabile	scende
fondo1	40	45	5
fondo2	70	30	-20
azioni	55	40	-10

Approccio minimax regret: regret

Best regret = Best payoff – Real payoff

Se il mercato sale il best payoff è 70. Se abbiamo investito in fondo1 il real regret è 70-40 Cioè 30. Se abbiamo investito in azioni è 15 e se abbiamo investito in fondo2 è 0.



alternative	sale	stabile	scende
fondo1	40	45	5
fondo2	70	30	-20
azioni	55	40	-10

Approccio minimax regret: regret

Best regret = Best payoff – Real payoff

Stesso genere di conti viene fatto per gli altri casi: mercato stabile o in discesa.

alternative	sale	stabile	scende
fondo1	40	45	5
fondo2	70	30	-20
azioni	55	40	-10



Si calcola una **regret table** ...

Approccio minimax regret: regret table

Si calcolano i massimi regret per ogni scelta alternativa e poi si seleziona l'alternativa che porta al regret minimo, nell'esempio acquistare azioni che dà un pentimento di 15.

alternative	sale	stabile	scende
fondo1	30	0	0
fondo2	0	15	25
azioni	15	5	15

Leggendo le righe: fondo1 dà regret massimo pari a 30, fondo2 dà 25, azioni dà 15.

Guadagno e giochi con avversario

- Nel nostro caso le dinamiche dell'ambiente sono costituite dall'avversario:
 - Come non sappiamo come evolverà l'ambiente, non sappiamo quale scelta farà l'avversario
 - Dobbiamo far bastare la *conoscenza dello stato corrente* e delle *mosse a disposizione*

Caratteristiche del gioco

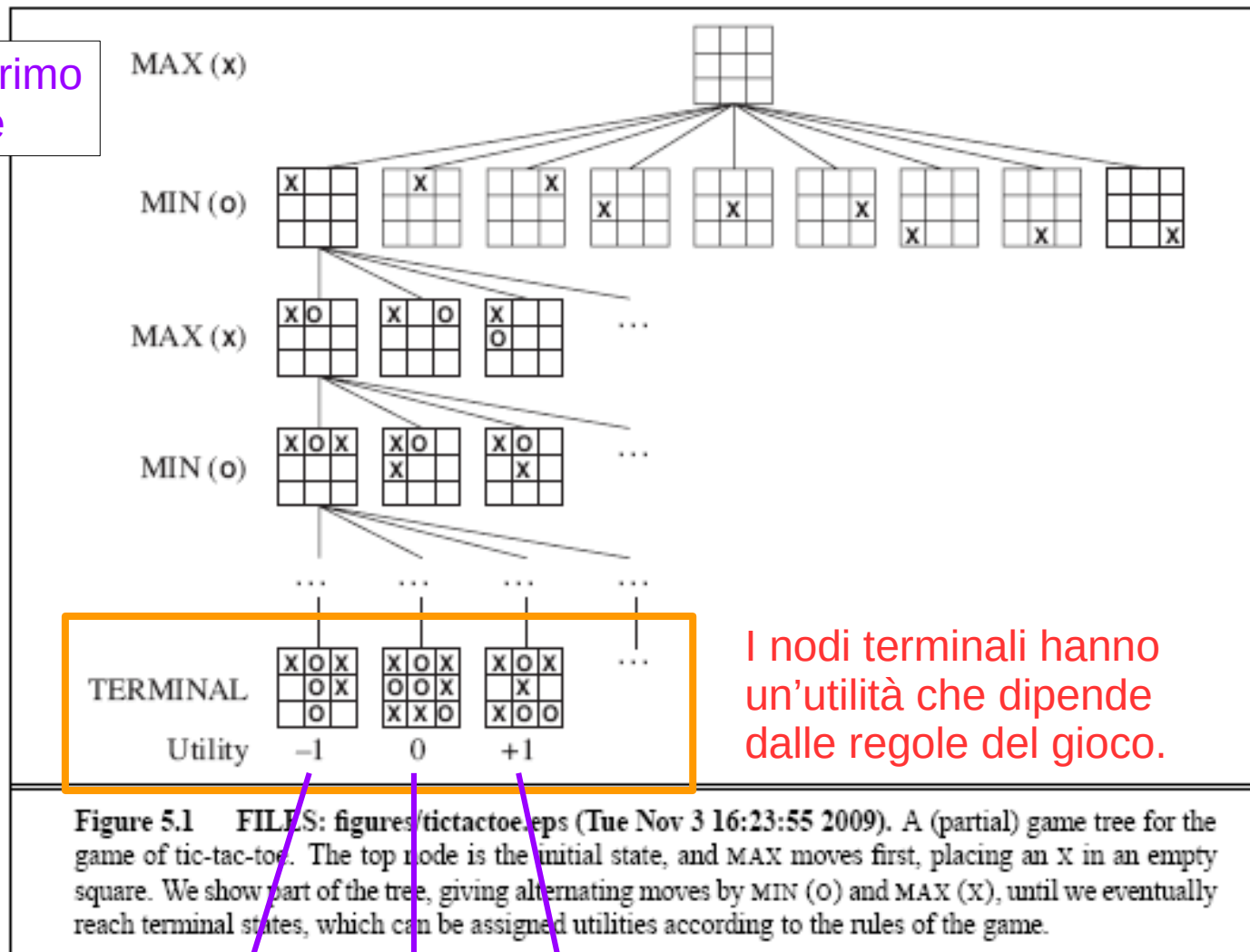
- **Due giocatori**, può essere comodo chiamarli MAX e MIN con **MAX che muove per primo**
- Ciascun giocatore non sa quali mosse farà l'altro ma le mosse possibili sono note e sono calcolabili i **successori** che produrranno una volta applicate a qualche stato
- **Osservabilità:**
 - È **totale** nel caso di giochi con turno: i giocatori conoscono i risultati delle mosse precedenti
 - È **parziale** nel caso di giochi ad azione simultanea: i giocatori non conoscono le mosse che i giocatori eseguono simultaneamente alla loro
- Partendo da uno **stato iniziale** è possibile sviluppare un **albero di possibili evoluzioni (albero di gioco)**, applicando le azioni eseguibili e calcolando così gli stati successivi
- Alcuni stati sono **terminali**, quando uno di essi è raggiunto la partita termina

Caratteristiche del gioco

- I giocatori si avvalgono del calcolo dell'**utilità degli stati**
- I giocatori devono **tener conto dell'avversario** quindi il calcolo dell'utilità comprende una valutazione del punto di vista dell'avversario
- **Giocatori pessimisti**: suppongono che l'avversario faccia sempre la mossa che gli porta il guadagno maggiore
- **Giochi con turno** (2-ply)

Utilità soggettiva nel caso del tris

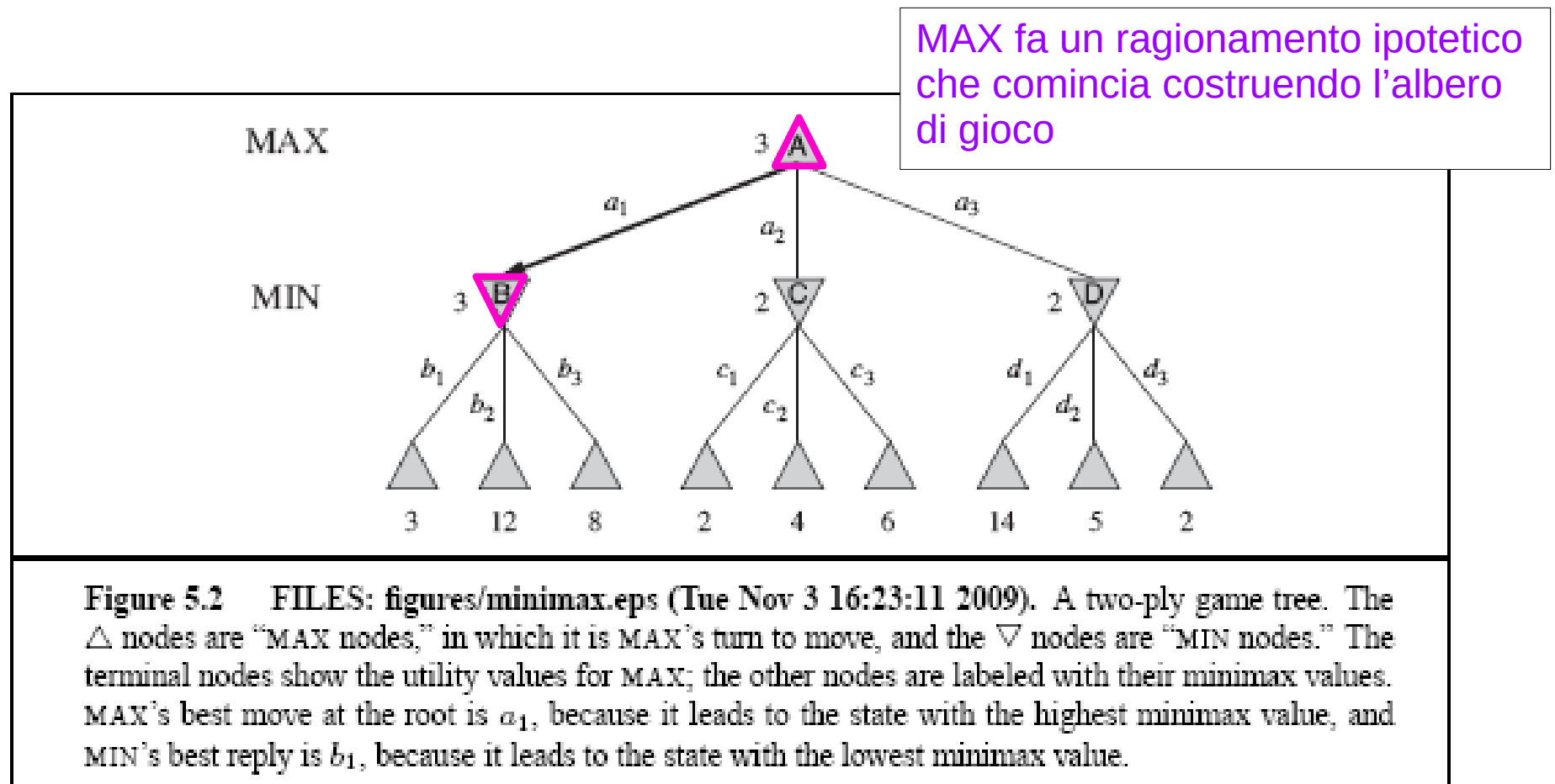
MAX è il primo a muovere



Strategia ottima per un agente

- Sequenza di mosse che porta a uno stato terminale corrispondente alla vittoria dell'agente
- L'agente non sa come muoverà l'altro, può solo "immedesimarsi"

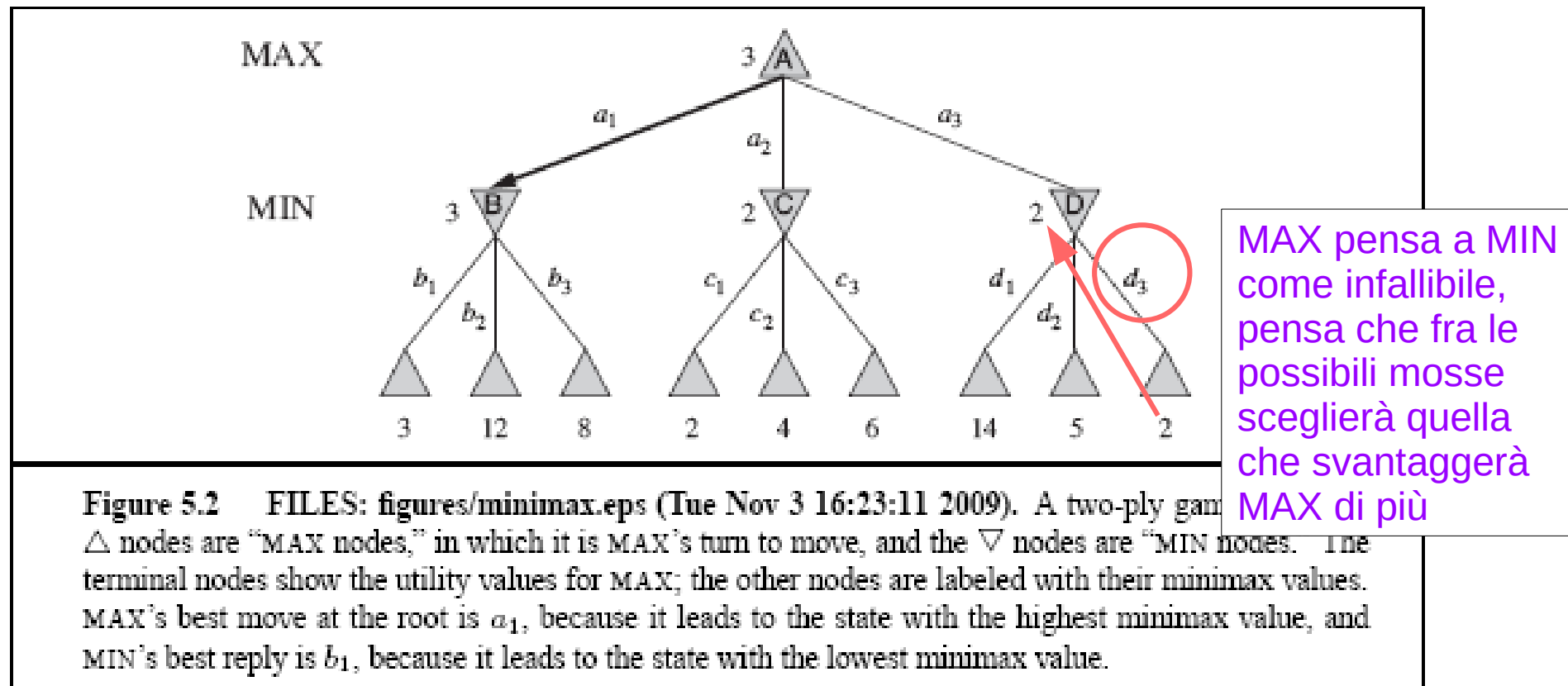
Ricerca strategia ottima, esempio



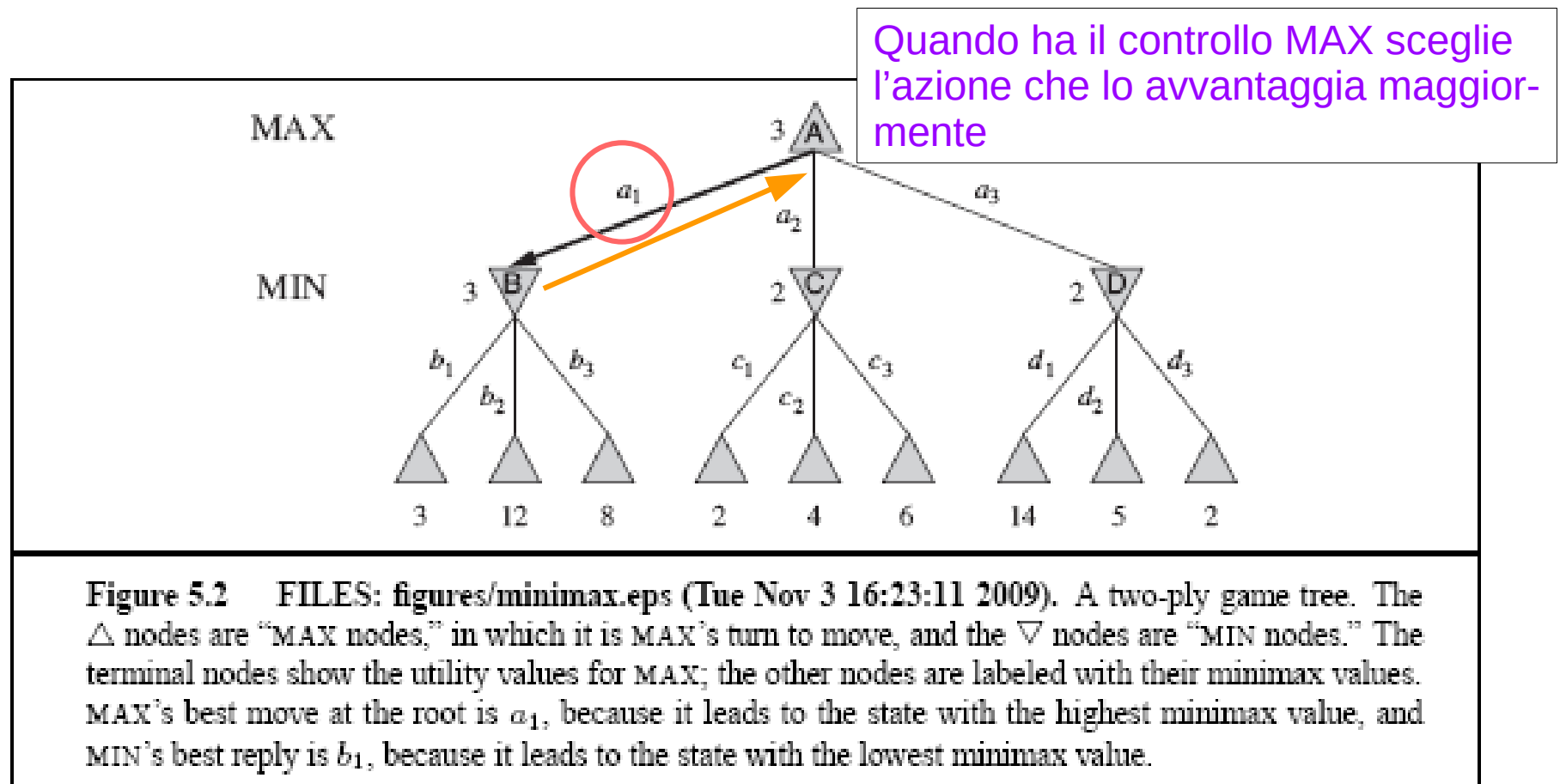
\triangle Nodi MAX

∇ Nodi MIN

Ricerca strategia ottima, esempio



Ricerca strategia ottima, esempio



Valore-minimax(n)

$$\text{Valore-minimax}(n) = \begin{cases} \text{utilità}(n) & \text{se } n \text{ terminale} \\ \max_{s \in \text{succ}(n)} \text{valore-minimax}(s) & \text{se } n \text{ è un nodo MAX} \\ \min_{s \in \text{succ}(n)} \text{valore-minimax}(s) & \text{se } n \text{ è un nodo MIN} \end{cases}$$

È un valore calcolato per ogni nodo. Permette all'agente la scelta della mossa da eseguire. I nodi MAX e MIN rappresentano due tipi di scelte: MAX è una **scelta interna** dell'agente mentre MIN è una **scelta esterna**, cioè tale che l'agente non ha controllo su di essa

Il nome minimax deriva dal fatto che ogni giocatore, minimizzando il guadagno massimo dell'altro minimizza la propria perdita

Nei **giochi a due** l'utilità per un giocatore è uguale a meno l'utilità per l'altro giocatore. Basta ricordare uno di questi valori per avere automaticamente anche l'altro

Algoritmo minimax ricorsivo

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

inputs: *state*, current state in game

$v \leftarrow \text{MAX-VALUE}(\textit{state})$

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*)

$v \leftarrow -\infty$

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

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$

return *v*

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

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

$v \leftarrow \infty$

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

$v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$

return *v*

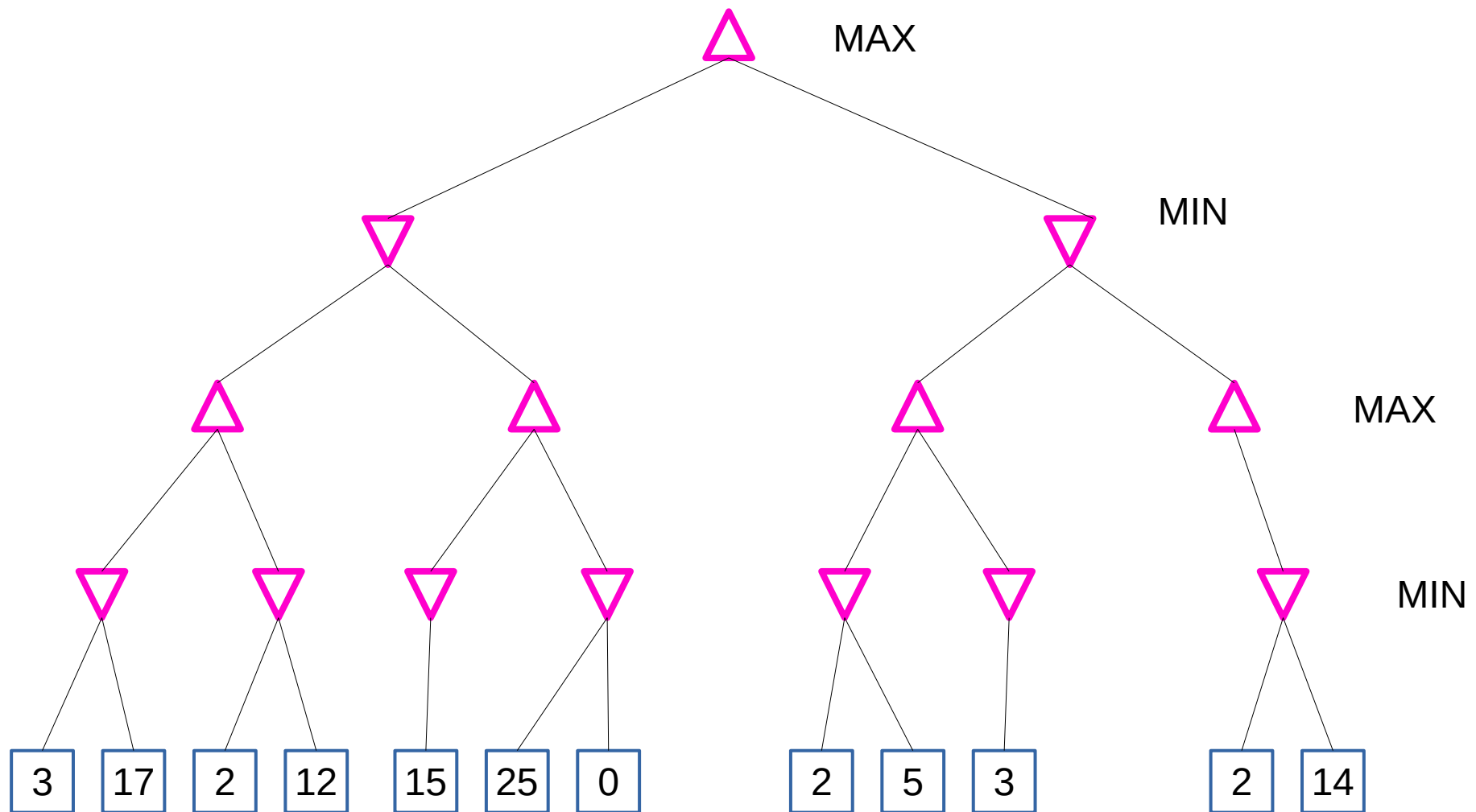
Visita completa in profondità dell'albero di gioco

Complessità temporale: $O(b^m)$
con b = branching factor ed
 m = profondità massima

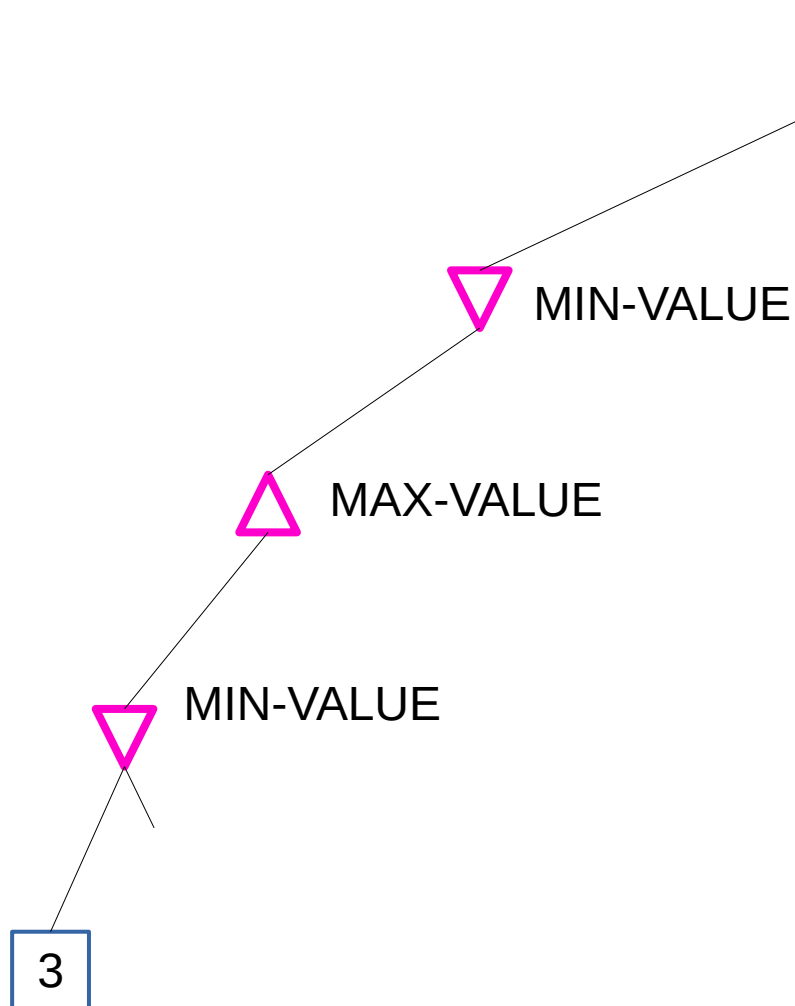
Complessità spaziale: $O(bm)$
se i successori sono generati
contemporaneamente

Figure 6.3 An algorithm for calculating minimax decisions. It returns the action corresponding to the best possible move, that is, the move that leads to the outcome with the best utility, under the assumption that the opponent plays to minimize utility. The functions MAX-VALUE and MIN-VALUE go through the whole game tree, all the way to the leaves, to determine the backed-up value of a state.

Esempio: albero di gioco completo



Esempio: albero di gioco completo



MAX-VALUE

MIN-VALUE

MAX-VALUE

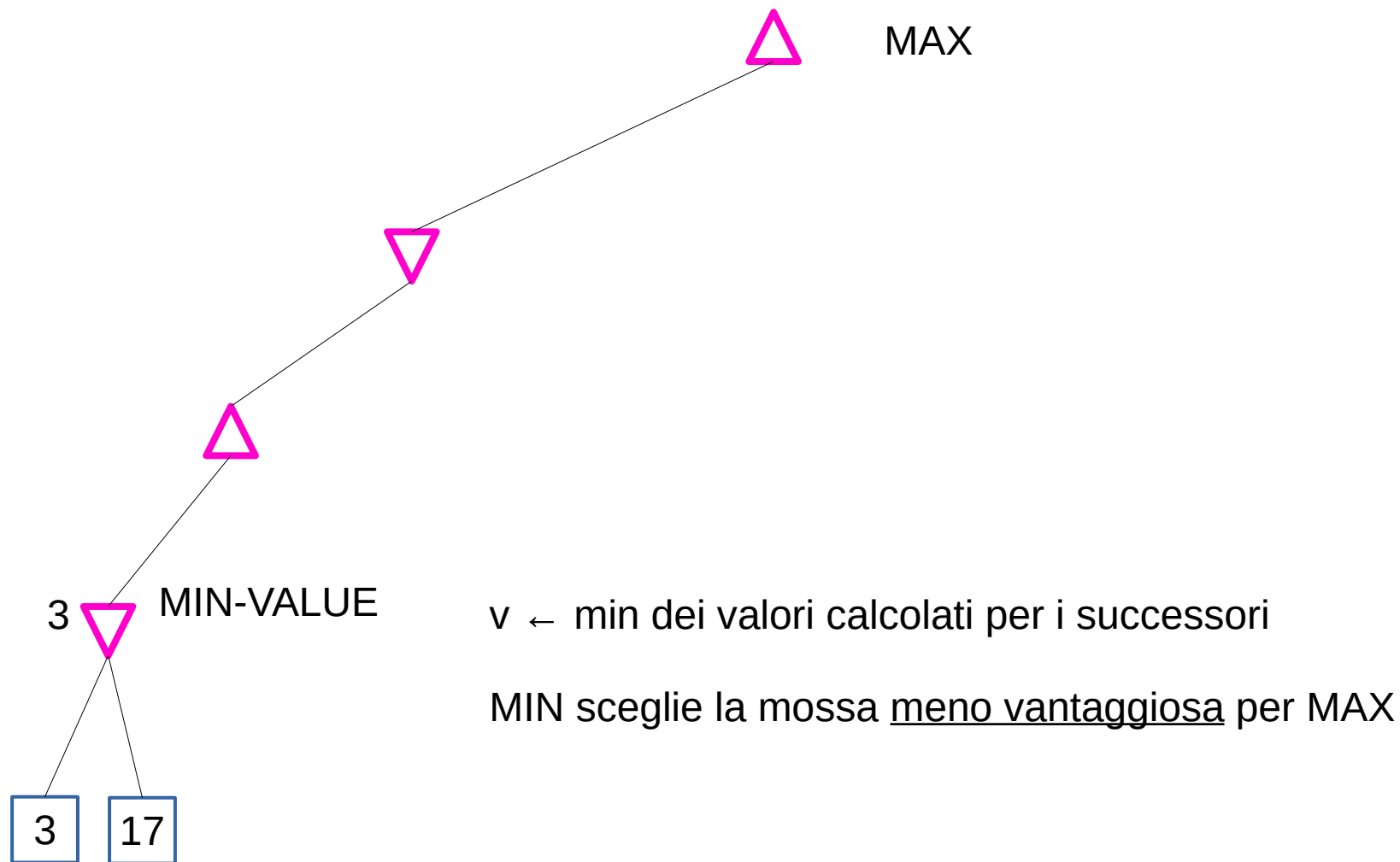
MIN-VALUE

Esplorazione ricorsiva dell'intero albero con chiamate alternate a max-value e a min-value.

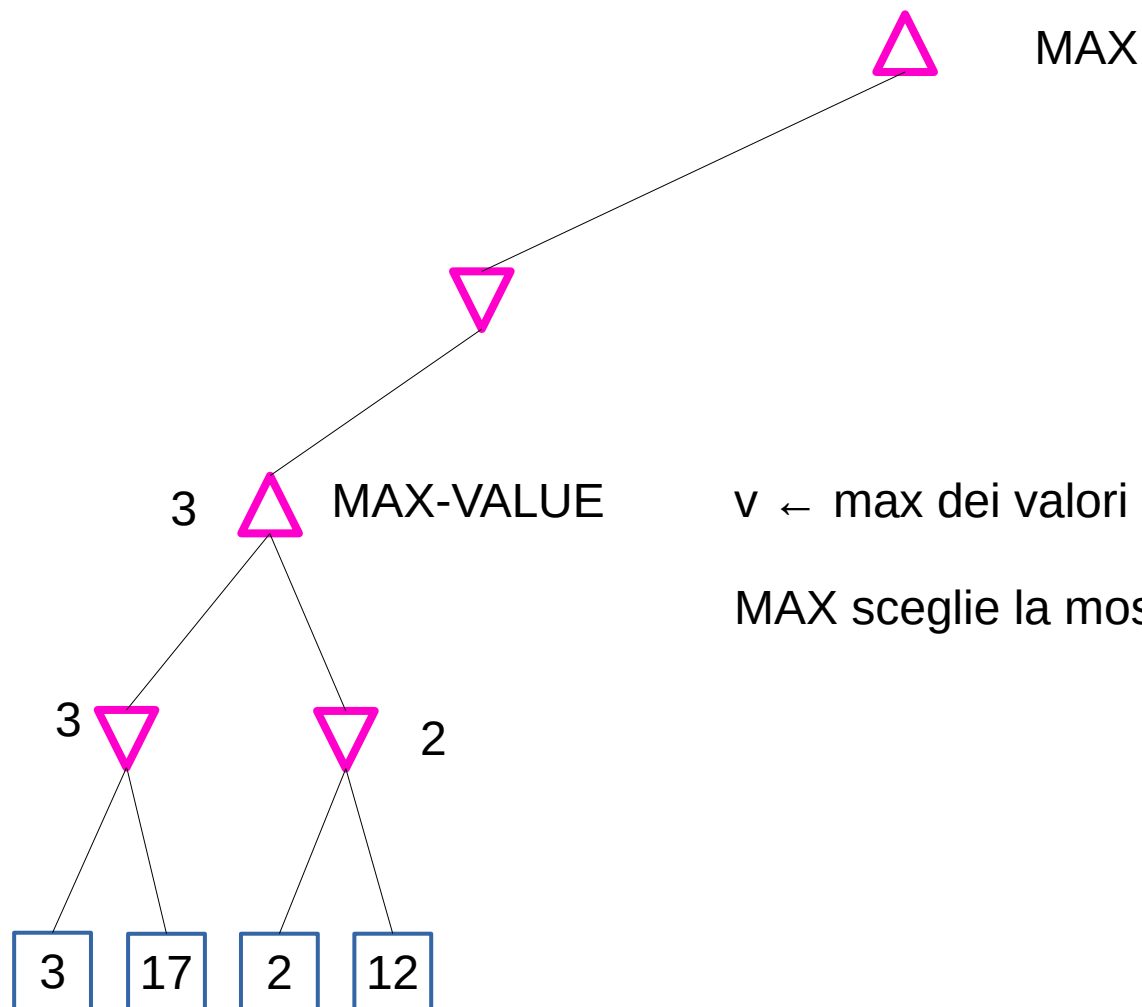
I nodi assumeranno un valore alla chiusura delle rispettive chiamate ricorsive.

I nodi terminali hanno come valore la loro utilità.

Esempio: albero di gioco completo



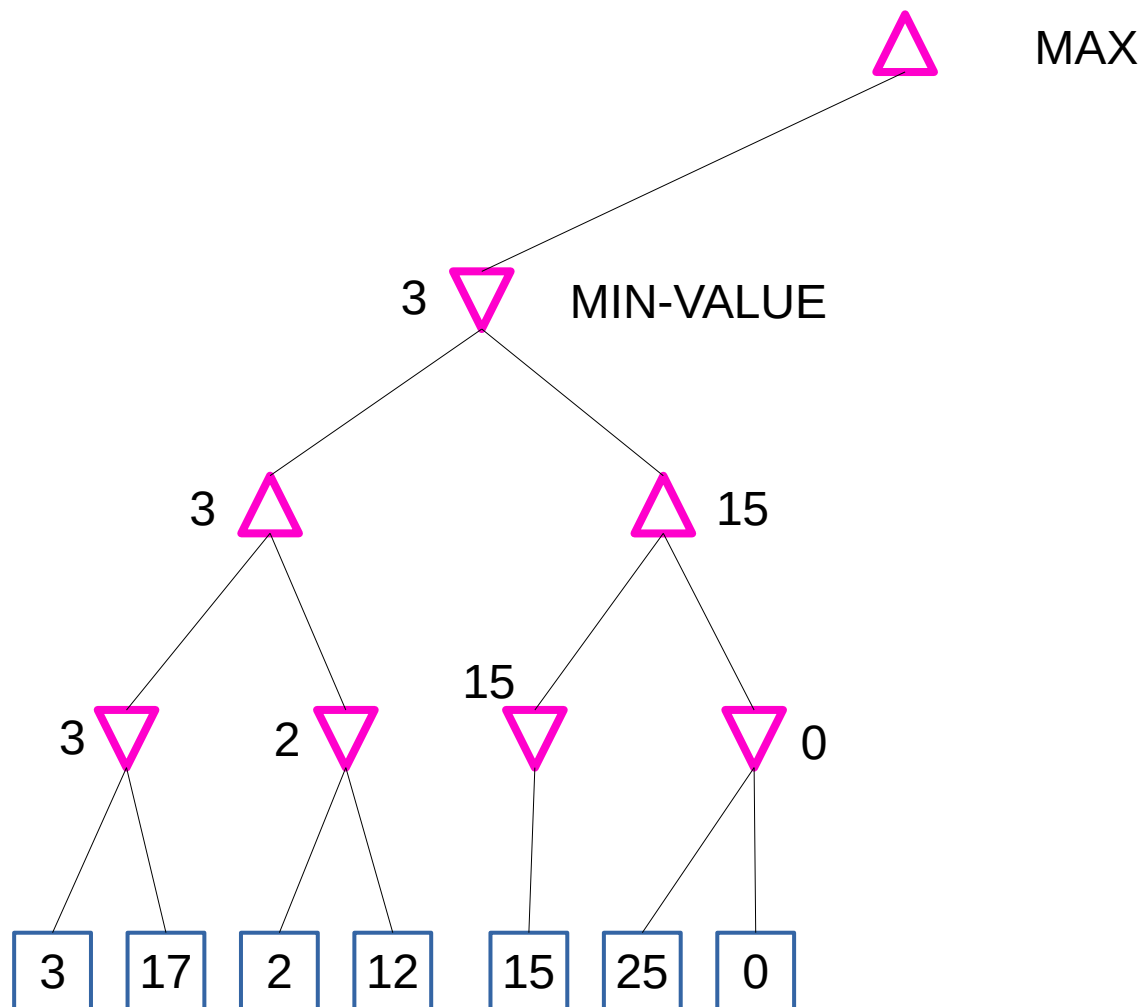
Esempio: albero di gioco completo



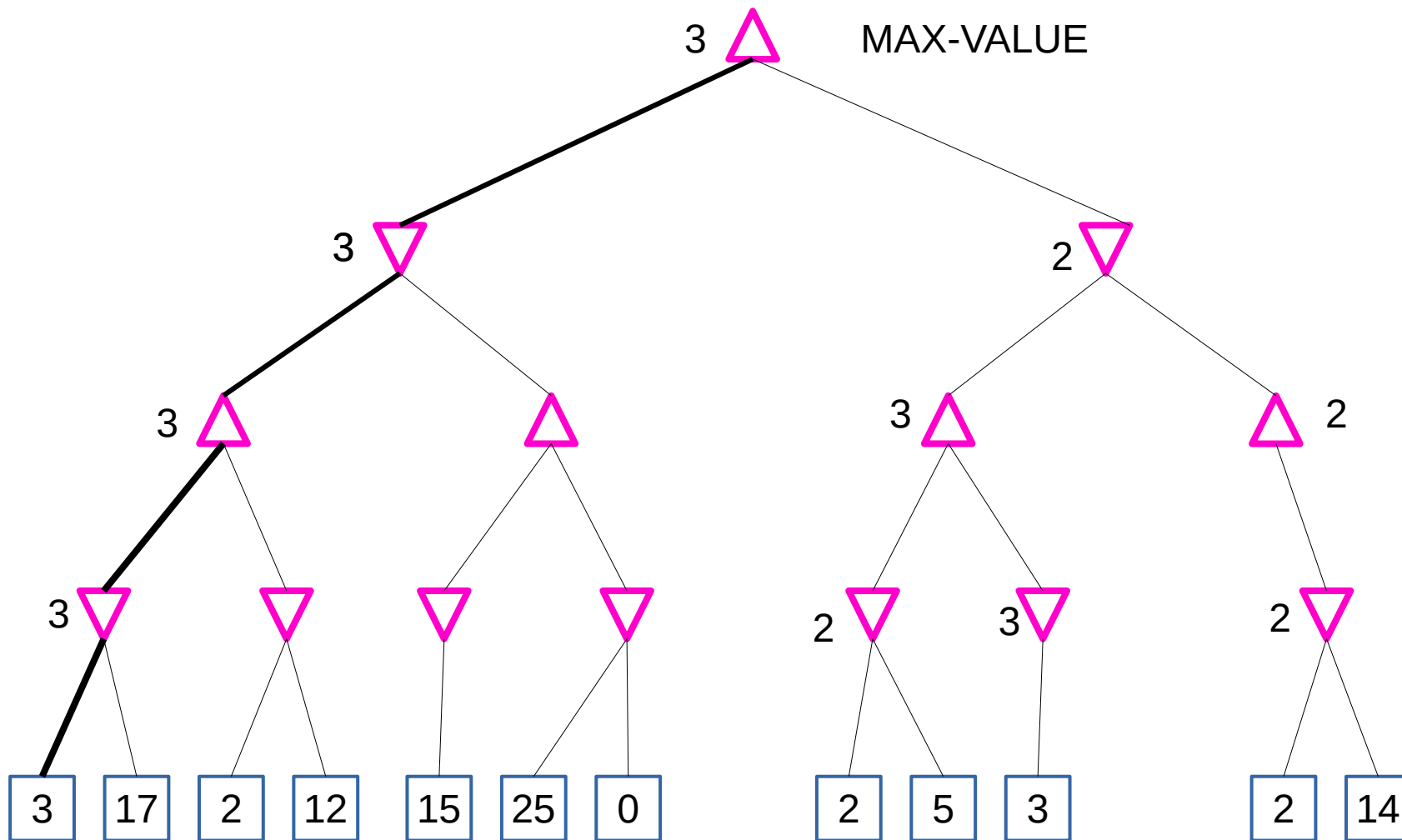
$v \leftarrow \max$ dei valori calcolati per i successori

MAX sceglie la mossa più vantaggiosa per sè

Esempio: albero di gioco completo



Esempio: albero di gioco completo



Giochi con più di due giocatori

- Minimax può essere esteso a **più giocatori**
- Utilità di ogni nodo data da un vettore $\langle u_1, \dots, u_k \rangle$ dove u_i indica l'utilità del nodo per il giocatore i
- Un aspetto dei giochi a più di due giocatori è la formazione di alleanze

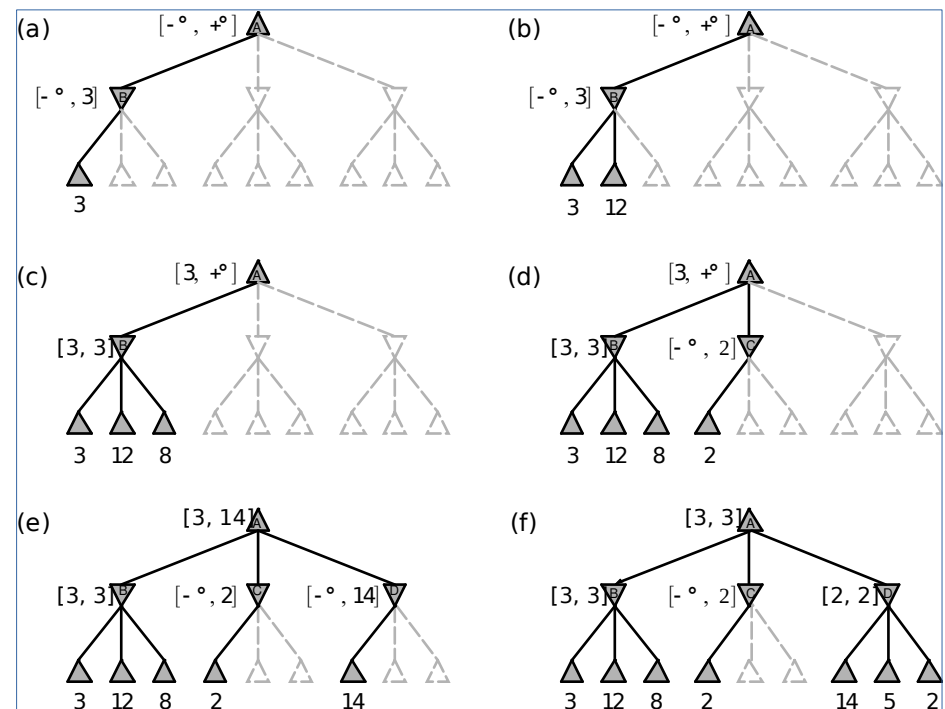
Valutazione

- Effettua una visita in profondità completa quindi la complessità temporale è esponenziale e quella spaziale è lineare
- È completo in grafi finiti
- È ottimale se MAX e MIN giocano in modo ottimale

Minimax con potatura Alfa-Beta

- In termini di tempo, minimax è molto costoso
- Per ridurre i tempi di ricerca: potare i rami meno promettenti senza seguirli
- Alfa-beta è un algoritmo di potatura

nell'esempio le foglie figlie del nodo centrale non espanso hanno valutazioni più alte di 2. È inutile considerarle perché darebbero un vantaggio all'avversario.



Potatura alfa-beta

- Ogni nodo ha associato un valore N che cambia man mano che l'esplorazione dei suoi sottoalberi procede
- L'esplorazione si porta dietro (e aggiorna) gli estremi di un intervallo $[\alpha, \beta]$:
 - α = **massimo lower bound** delle soluzioni possibili (è posto dai nodi MAX)
= valore della scelta migliore per MAX trovata in qualsiasi punto di scelta lungo
il cammino (inizialmente $-\infty$)
 - β = **minimo upper bound** delle soluzioni possibili (è posto dai nodi MIN)
= valore della scelta migliore per MIN trovata in qualsiasi punto di scelta lungo
il cammino (inizialmente ∞)
- Ha senso esplorare un nodo se e solo se il suo valore stimato N è compreso fra α e β

Potatura alfa-beta

- I valori α e β sono aggiornati man mano che la ricerca procede
- Idealmente α e β sono gli estremi di un intervallo che si restringe con la ricerca:

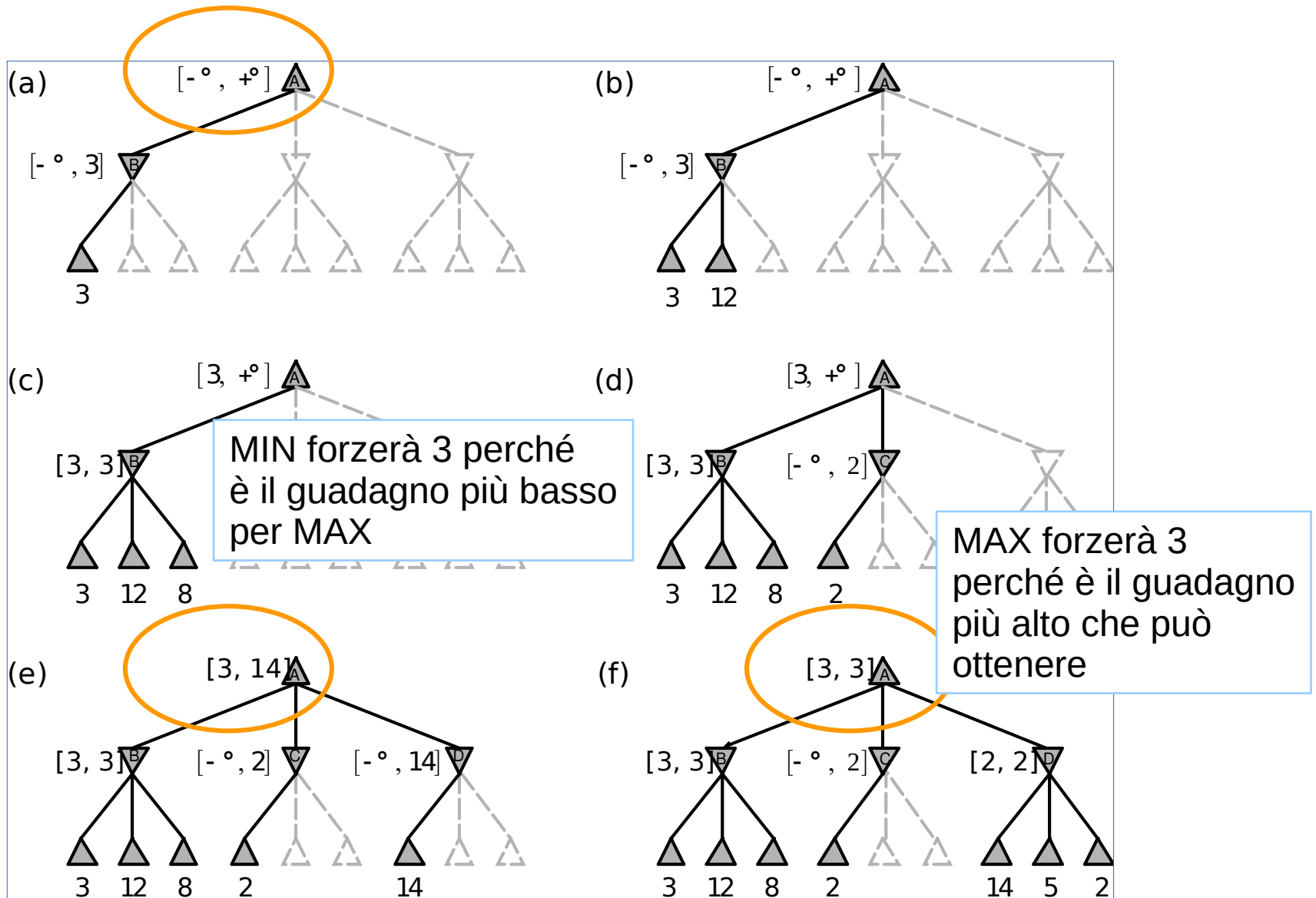


- I nodi che hanno valutazioni fuori dall'intervallo sono esclusi
- Se a un certo punto per un certo nodo i due estremi si invertono, si può evitare di condurre l'esplorazione dei sottoalberi rimanenti di quel nodo



- Nessun valore potrà infatti essere minore di β e maggiore di α

Esempio



Minimax con potatura alfa-beta: algoritmo

function ALPHA-BETA-SEARCH(*state*) **returns** an action
inputs: *state*, current state in game

$v \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty)$

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

function MAX-VALUE(*state*, α , β) **returns** a utility value

inputs: *state*, current state in game

α , the value of the best alternative for MAX along the path to *state*

β , 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*

MAX-VALUE modifica il
lower bound

function MIN-VALUE(*state*, α , β) **returns** a utility value

inputs: *state*, current state in game

α , the value of the best alternative for MAX along the path to *state*

β , 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**

$v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta))$

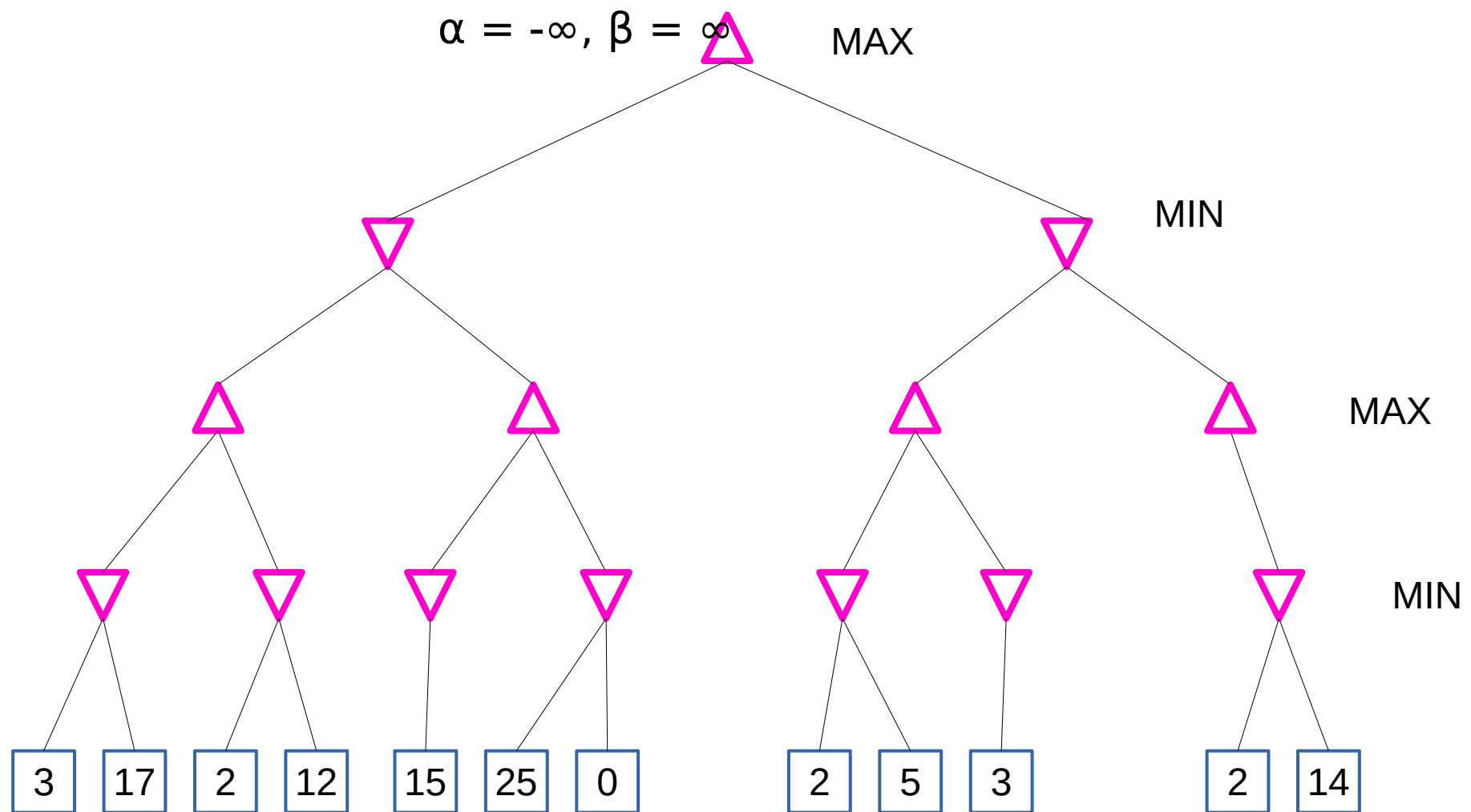
if $v \leq \alpha$ **then return** *v*

$\beta \leftarrow \text{MIN}(\beta, v)$

return *v*

MIN-VALUE modifica
l'upper bound

Esempio: albero di gioco completo



Esempio: parto dalla radice

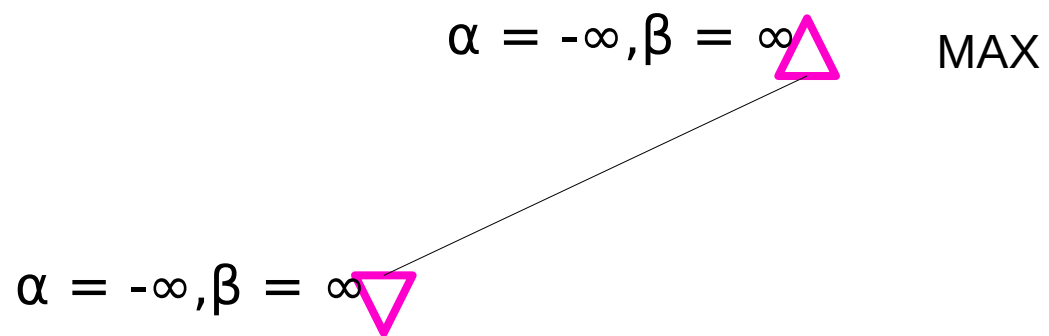
$$\alpha = -\infty, \beta = \infty \triangle \text{ MAX}$$



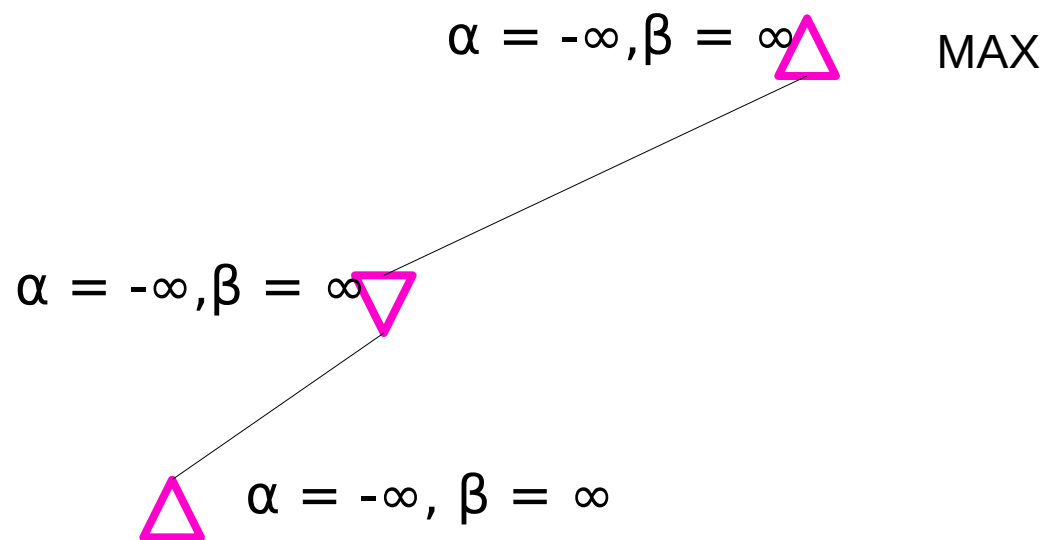
I nodi MIN che produrremo restringono l'upper bound α
I nodi MAX invece restringono il lower bound β

<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>

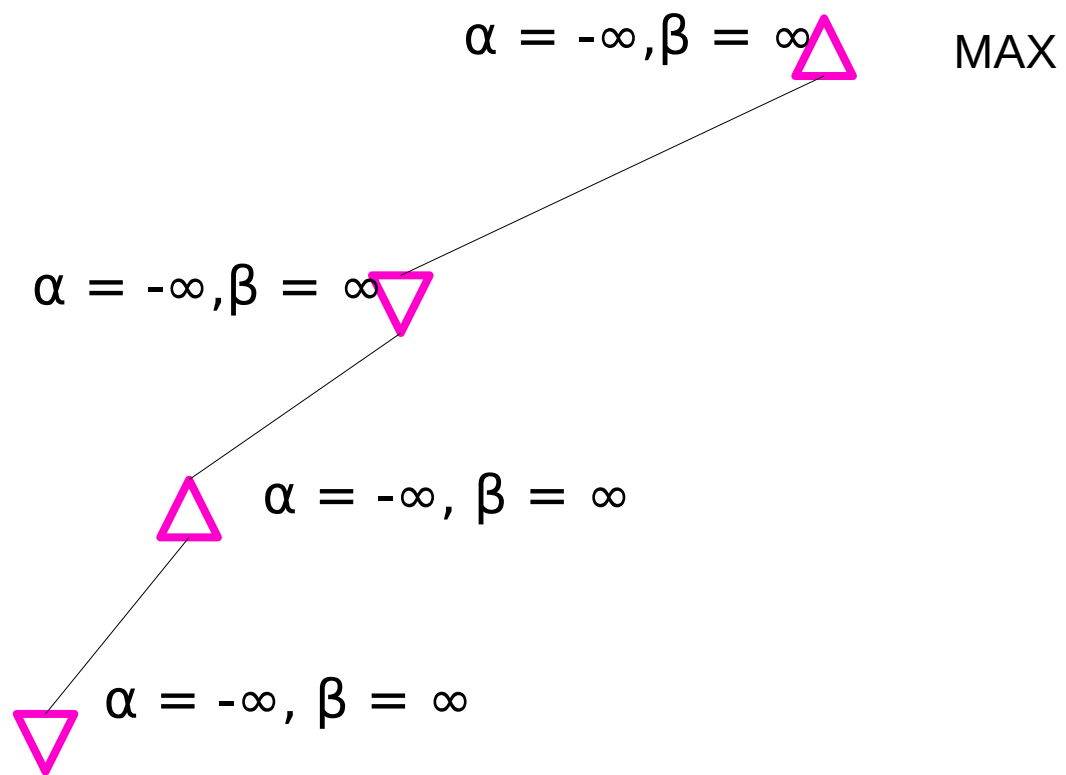
Esempio: costruisco il primo successore



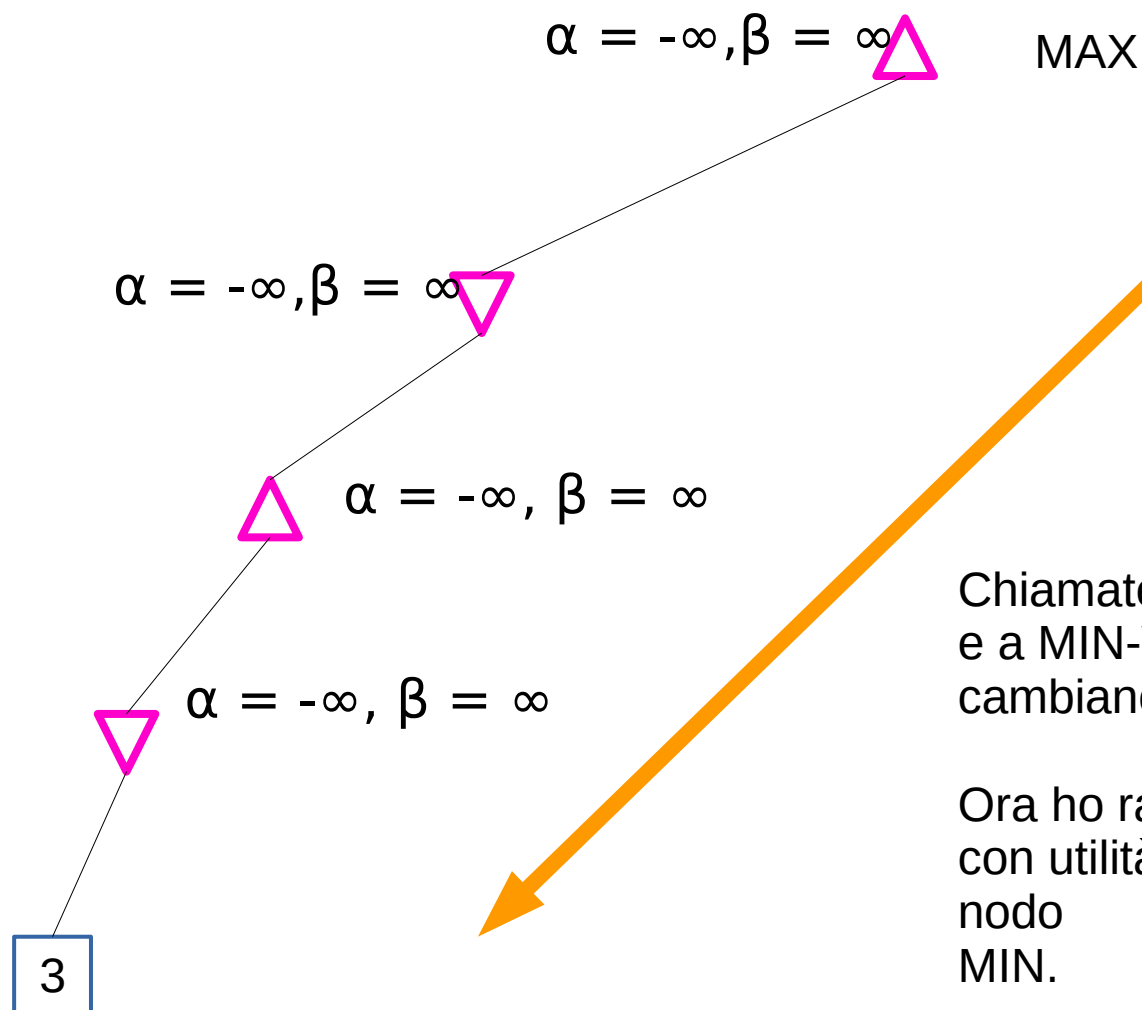
Esempio: continuo



Esempio: continuo



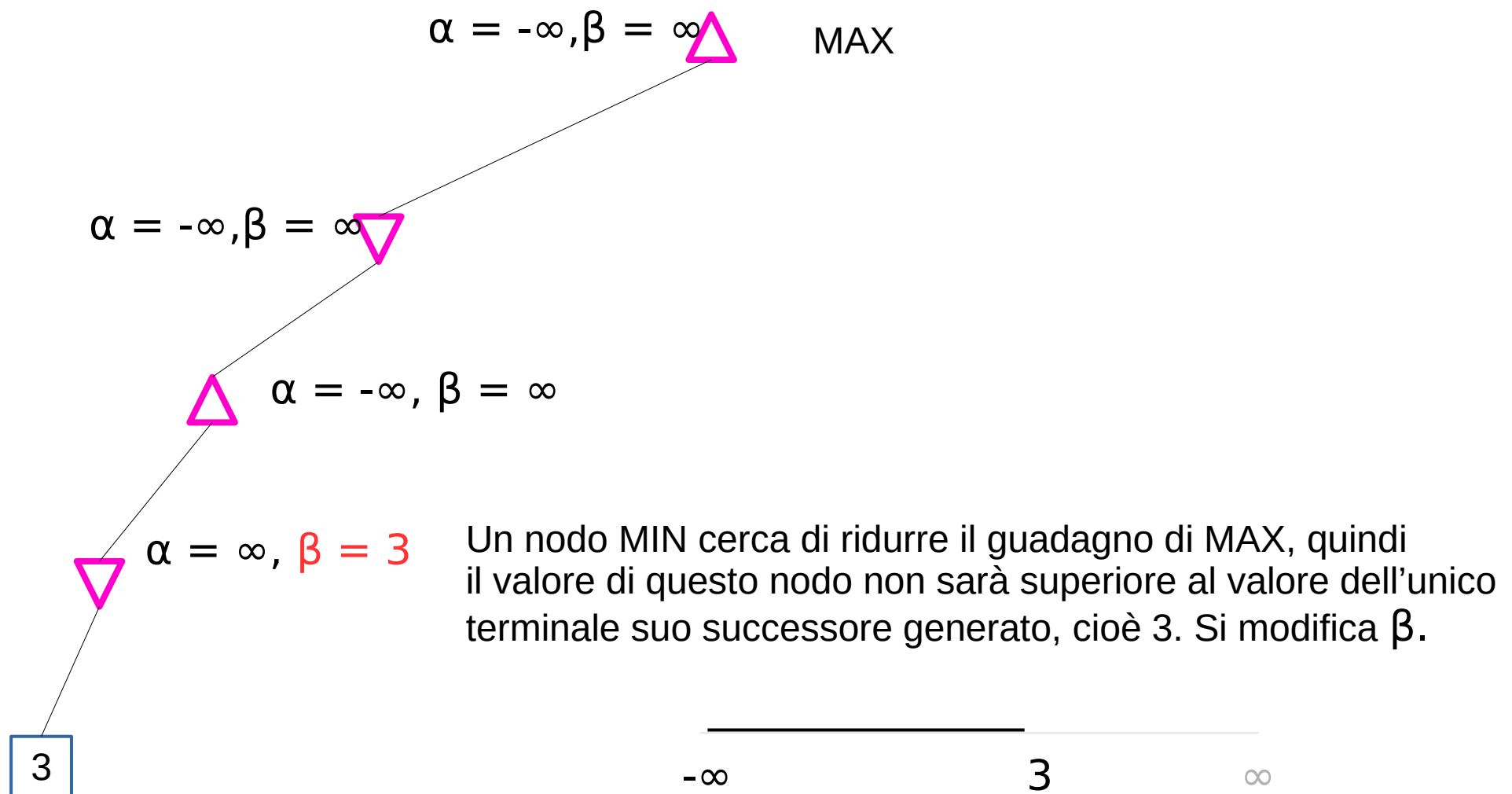
Esempio: rsggiungo il primo terminale



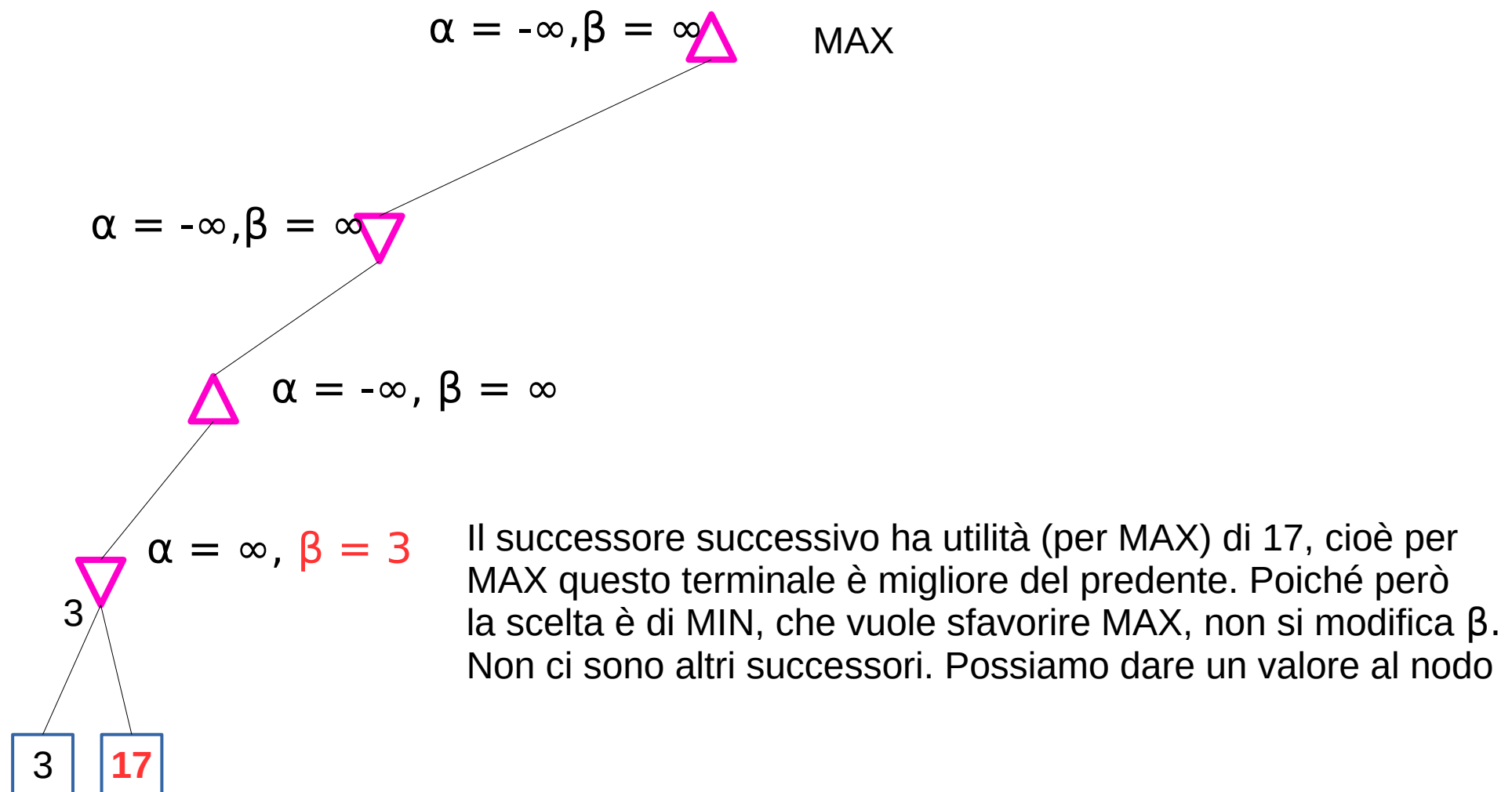
Chiamate ricorsive a MAX-VALUE e a MIN-VALUE. I due estremi non cambiano scendendo ricorsivamente.

Ora ho raggiunto un nodo terminale con utilità 3. L'ho raggiunto da un nodo MIN.

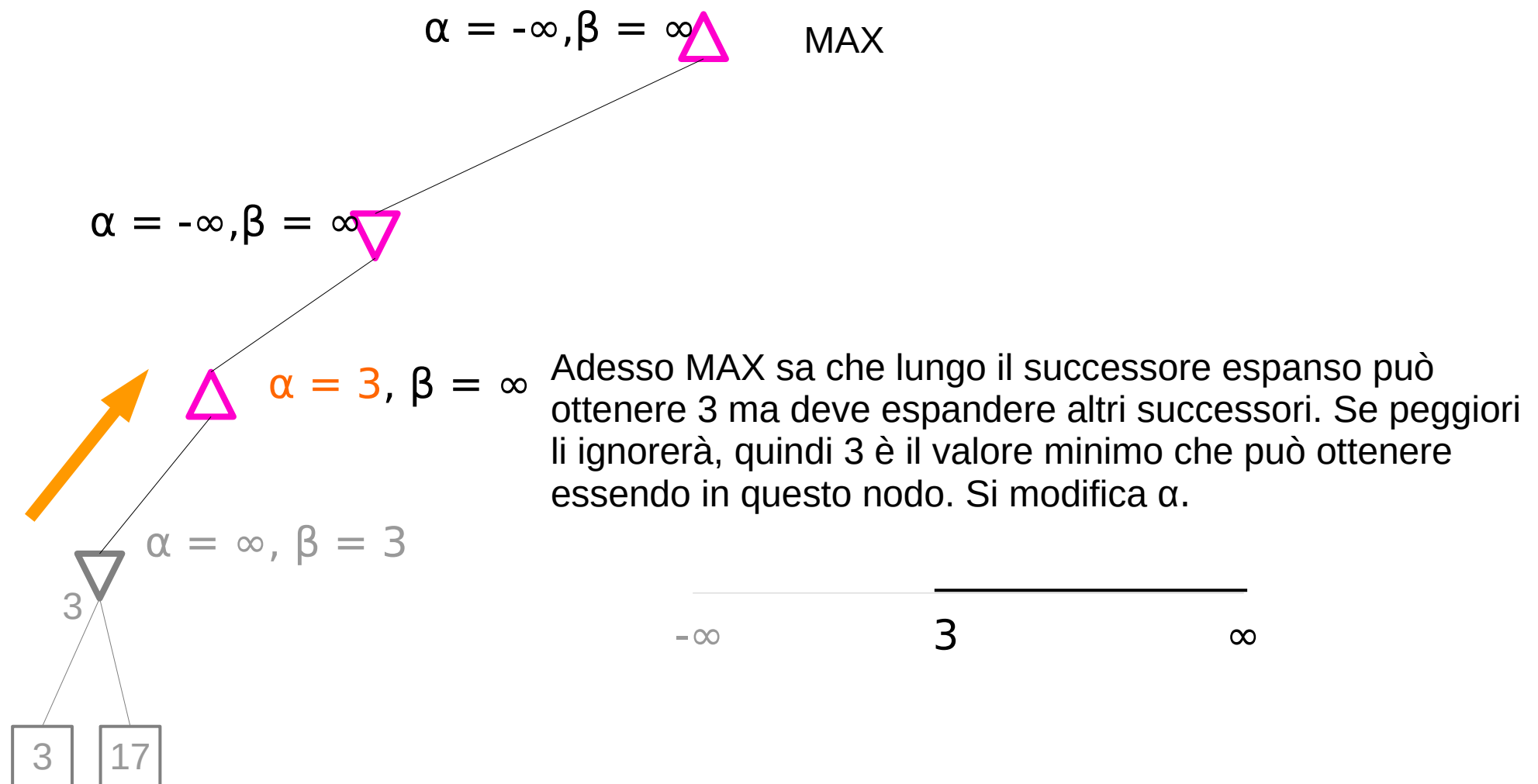
Esempio: comincio a retropropagare



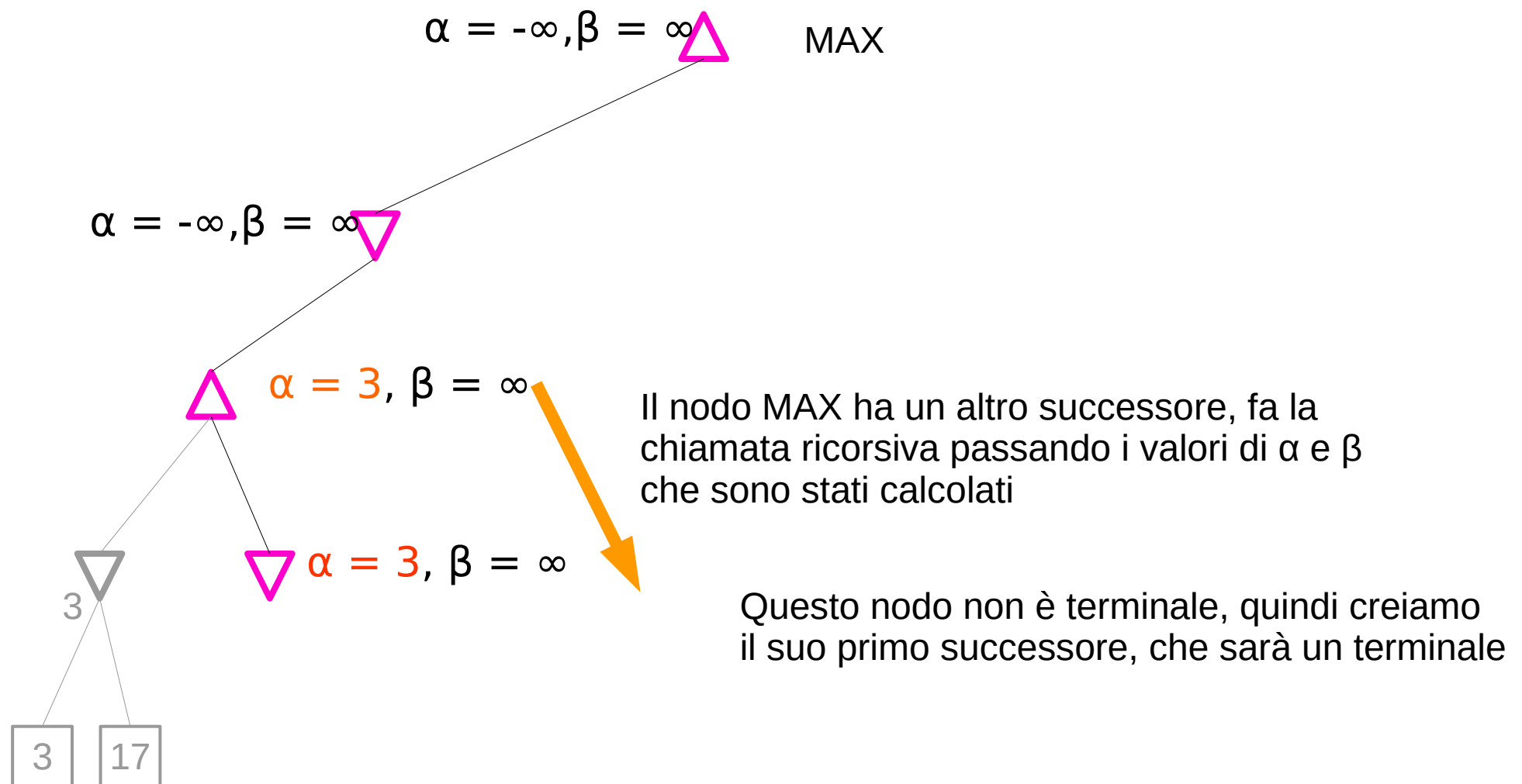
Esempio: passo all'altro successore



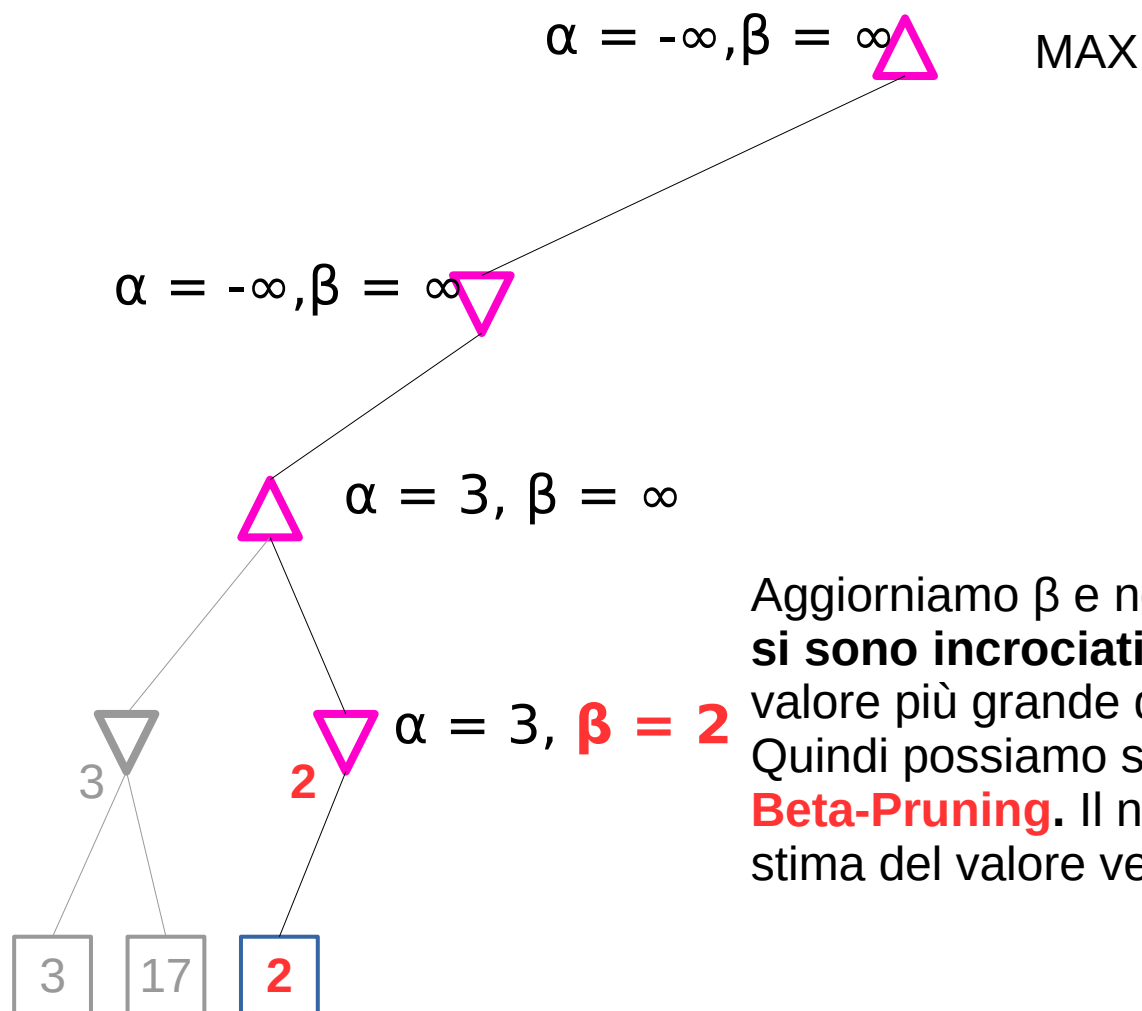
Esempio: retropropago



Esempio: scendo lungo un altro cammino

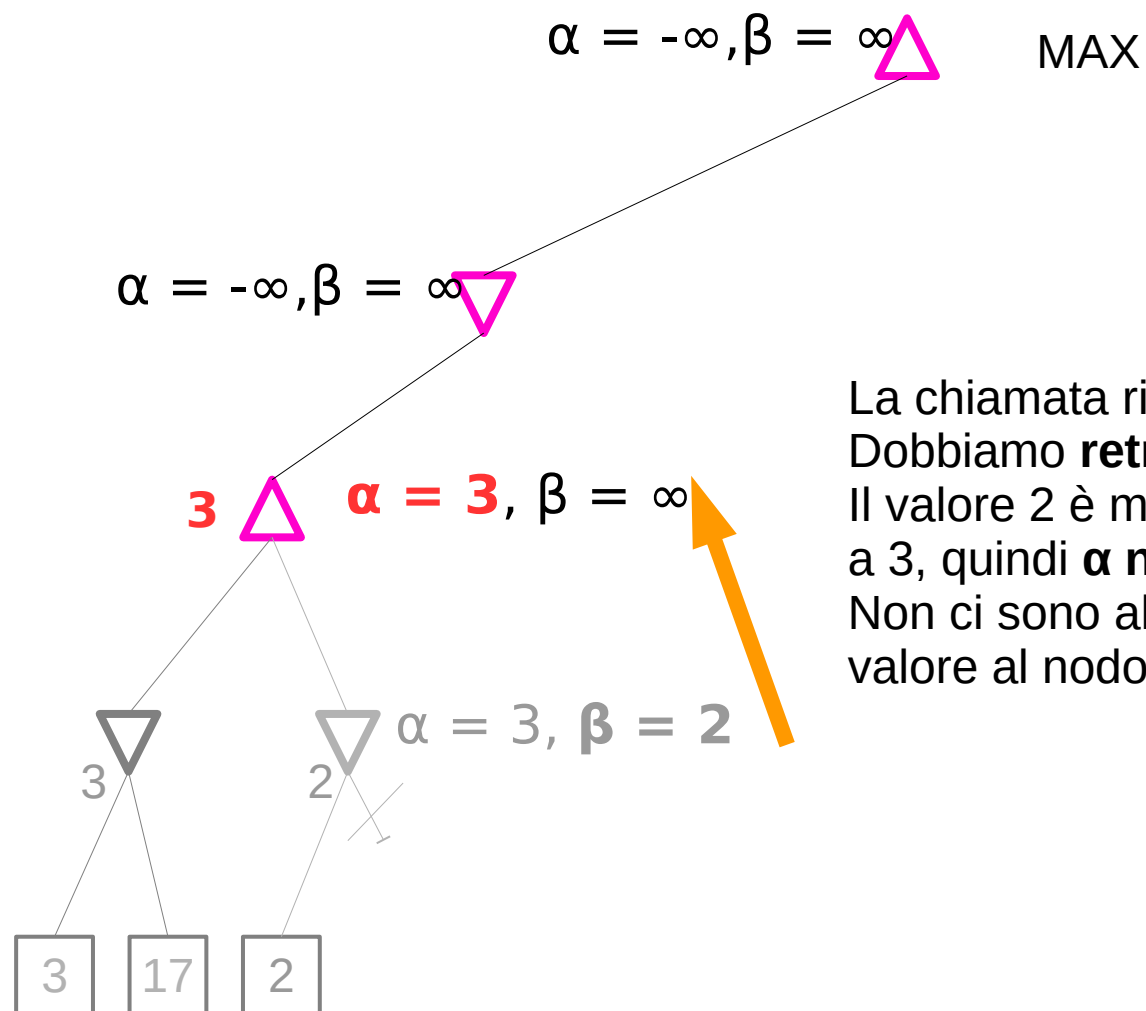


Esempio: raggiungo un terminale



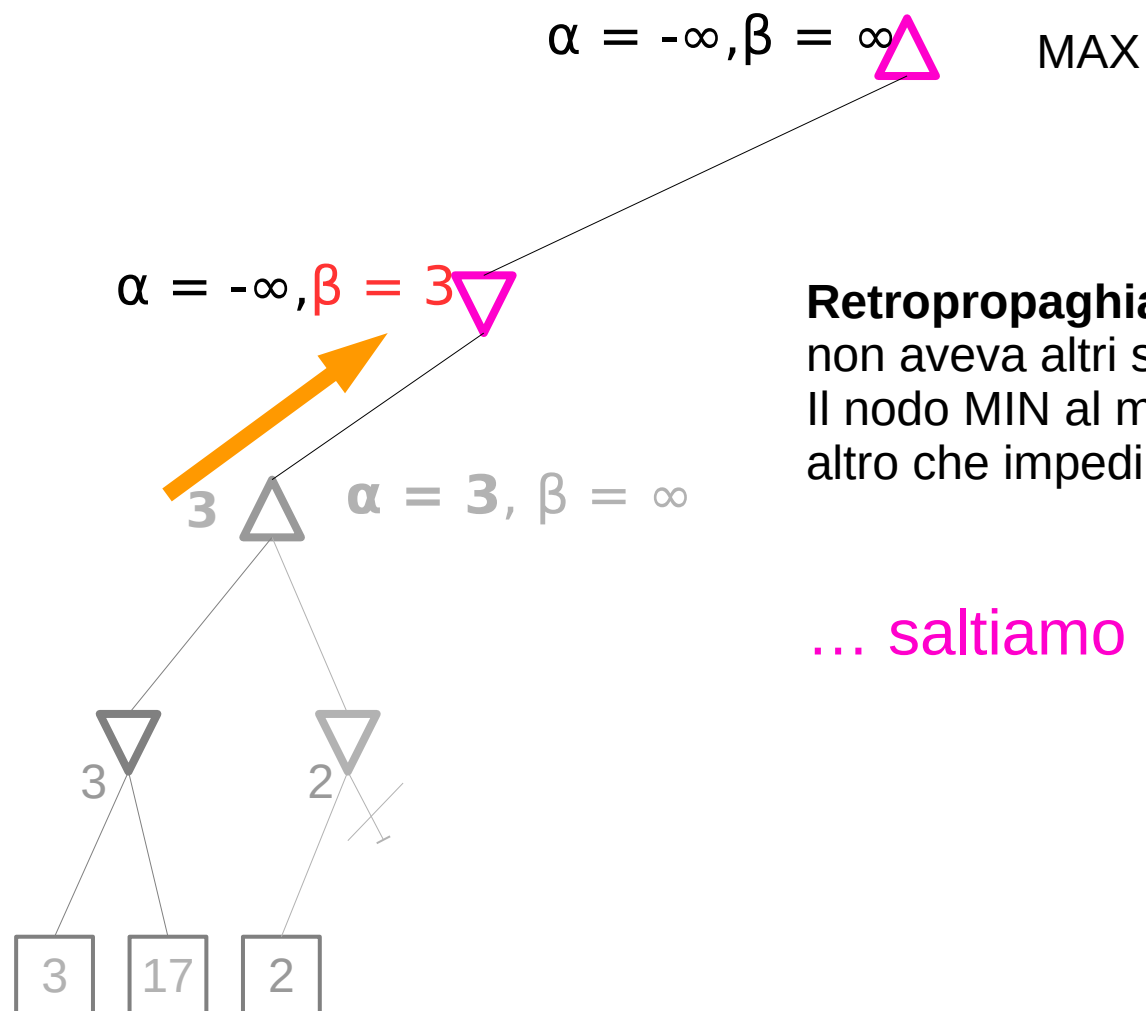
Aggiorniamo β e notiamo che **i due estremi dell'intervallo si sono incrociati**. Questo nodo MIN dovrebbe avere un valore più grande di 3 ma più piccolo di 2. **È impossibile!!** Quindi possiamo smettere di espanderlo. Si parla di **Beta-Pruning**. Il nodo prende valore 2, che è solo una stima del valore vero (abbiamo interrotto la ricerca)

Esempio: retropropago



La chiamata ricorsiva si chiude.
Dobbiamo **retropropagare** i valori calcolati.
Il valore 2 è meno buono per MAX rispetto a 3, quindi **α non cambia**.
Non ci sono altri figli, quindi possiamo dare valore al nodo (il valore 3)

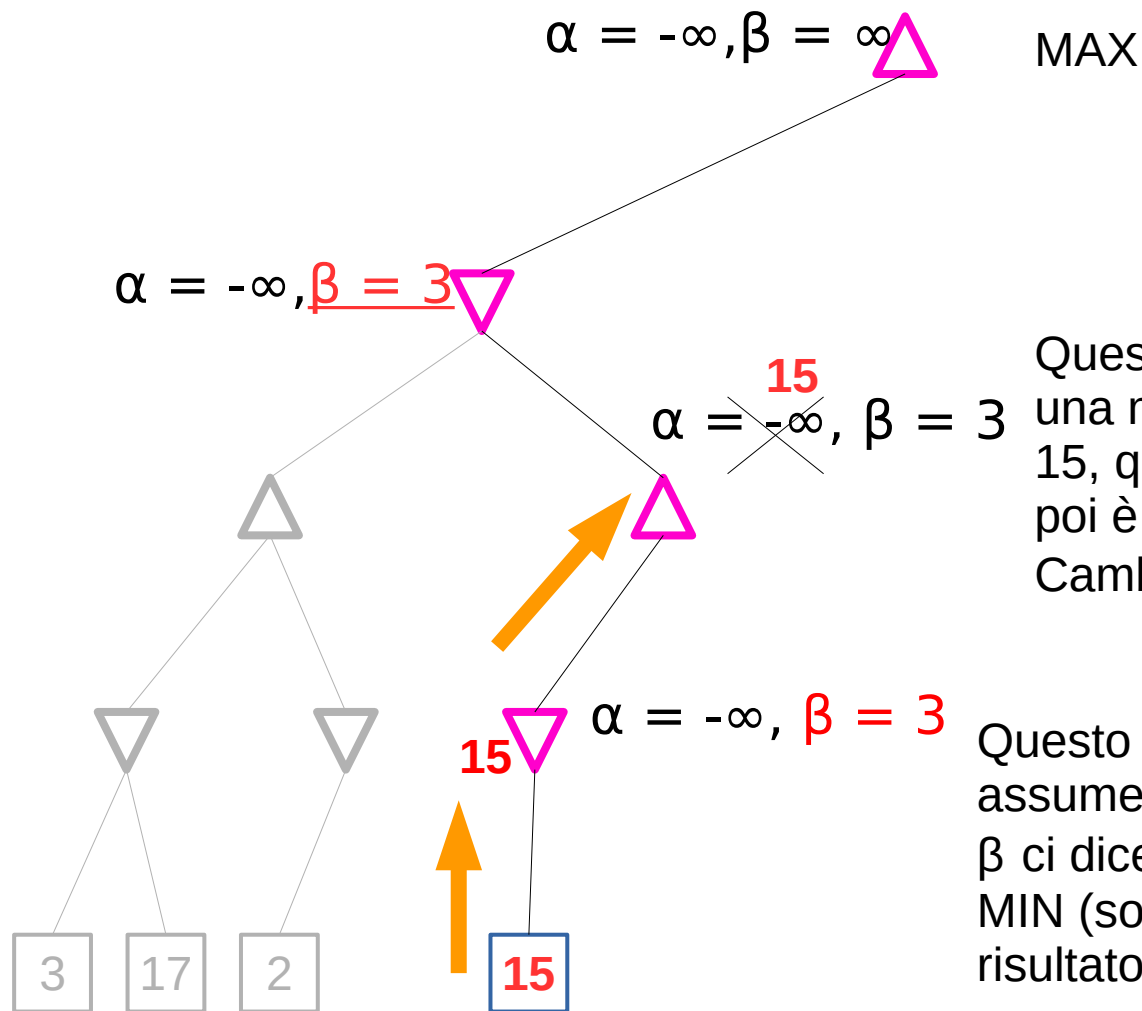
Esempio: retropropago



Retropropaghiamo ancora perché il nodo non aveva altri successori.
Il nodo MIN al momento non può garantire altro che impedire a MAX di fare meglio di 3

... saltiamo qualche mossa ...

Esempio: propagazione dei valori verso l'alto

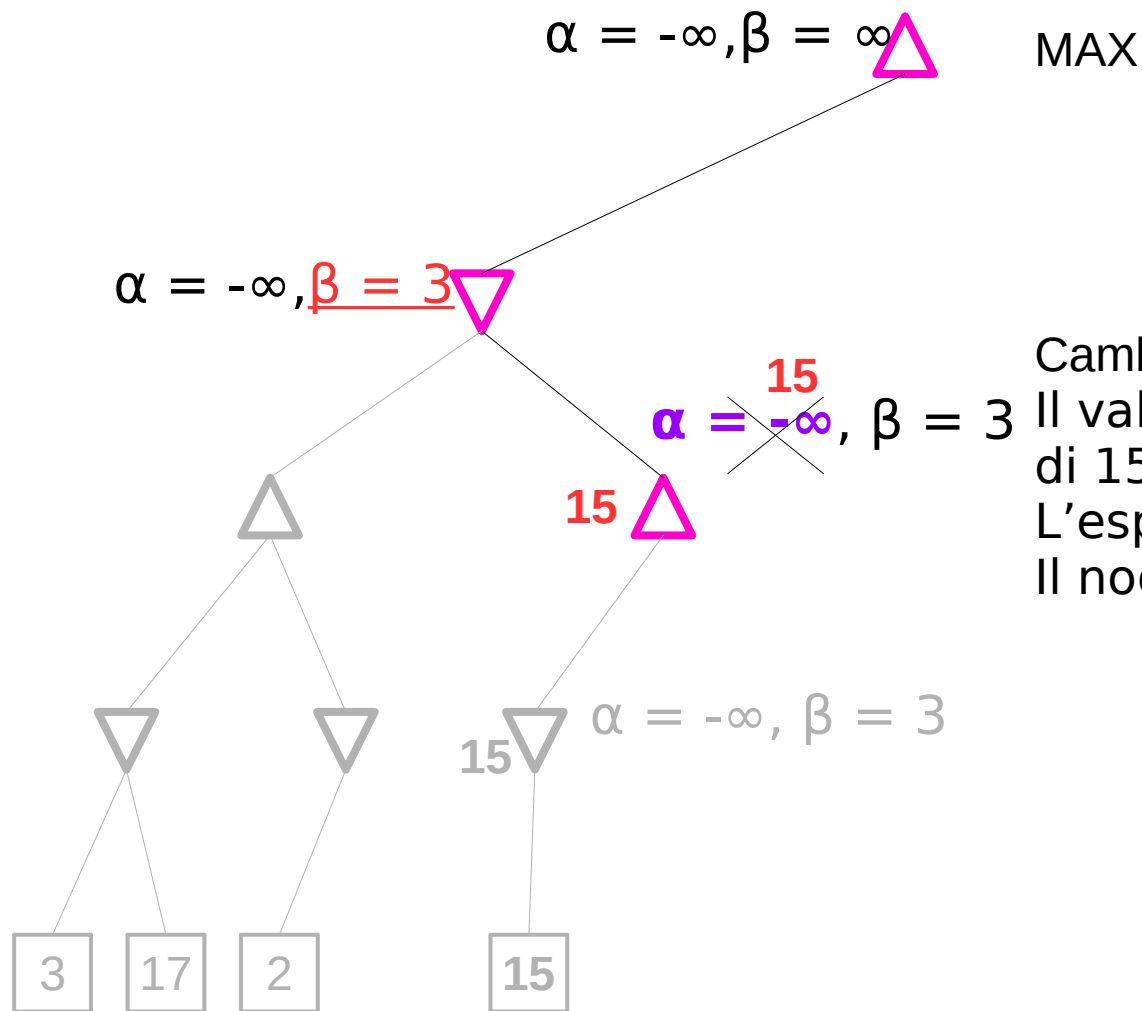


Questo è un nodo MAX, MAX qui può fare una mossa che lo porta a un nodo di valore 15, quindi il suo guadagno minimo di qui in poi è almeno 15.

Cambiamo α

Questo nodo ha un successore solo (terminale)
assume come valore quello del terminale.
 β ci dice che lungo il percorso c'è un alto nodo
MIN (sottolineato) che può forzare MAX a un
risultato peggiore quindi non cambiamo β

Esempio: propagazione dei valori verso l'alto

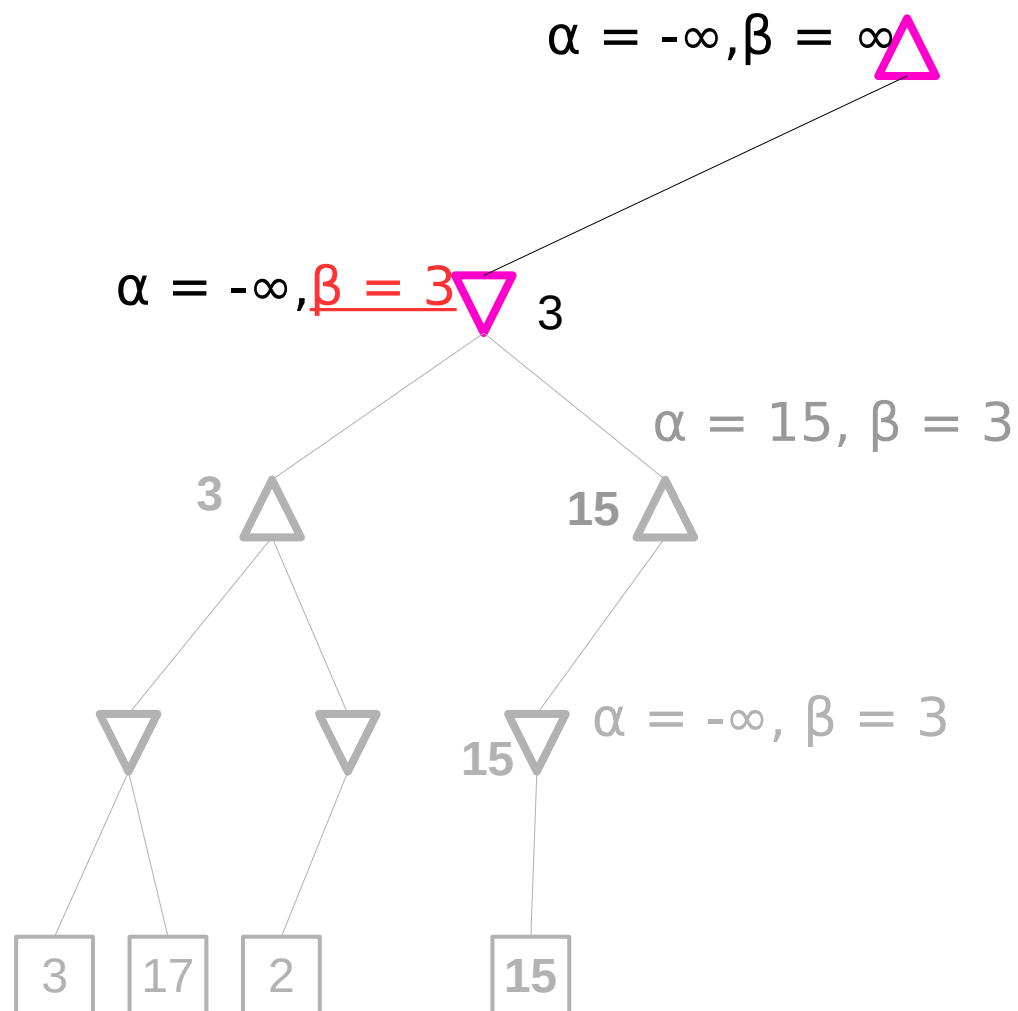


Cambiamo α

Il valore del nodo deve essere maggiore di 15 e minore di 3. Impossibile!!

L'esplorazione si ferma (**alfa pruning**).
Il nodo assume il valore stimato 15.

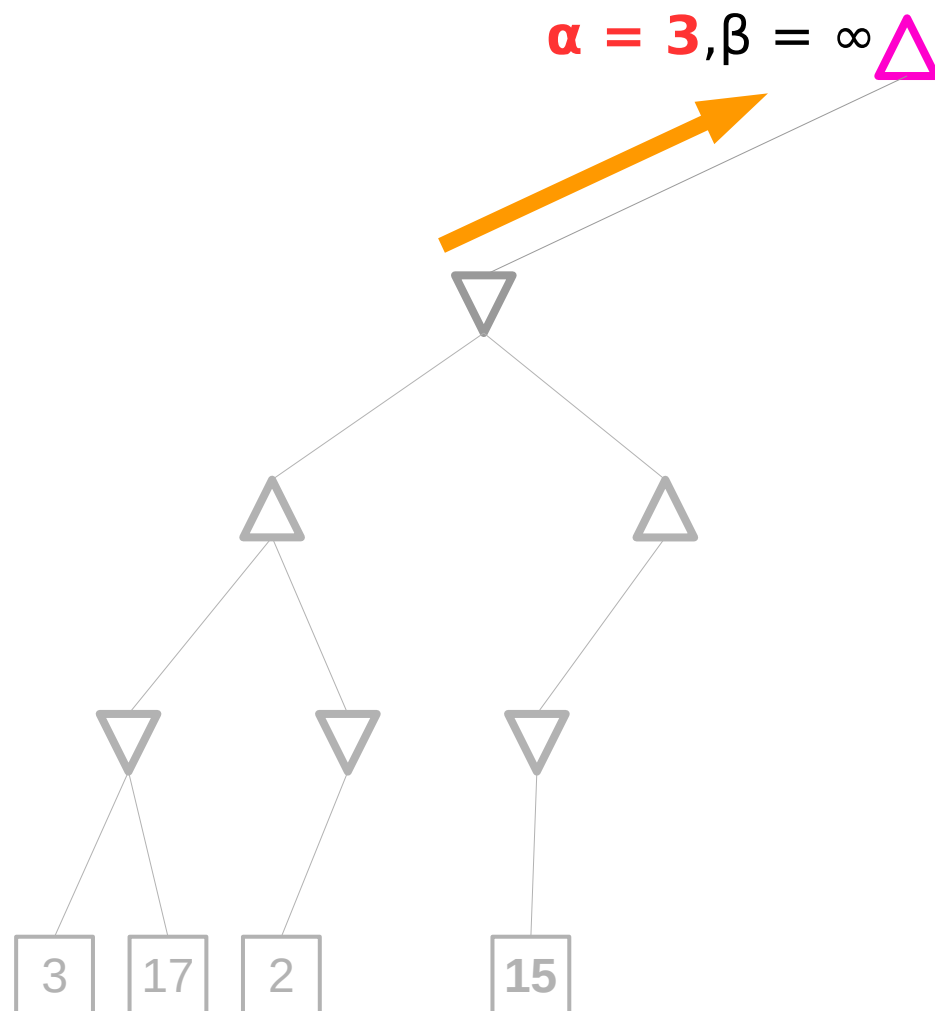
Esempio: propagazione dei valori verso l'alto



MAX

Il nodo MIN non ha altri successori. Ha il controllo su far guadagnare a MAX 3 oppure 15. Volendolo svantaggiare non modifica β

Esempio: propagazione dei valori verso l'alto



Esempio: albero di gioco completo

$\alpha = 3, \beta = \infty$

NB: il pruning si ha solo se
min-value esegue
If ($v \leq \alpha$) return value
anziché
If ($v < \alpha$) return value

$\alpha = 3, \beta = 3$

$\alpha = 3, \beta = \infty$

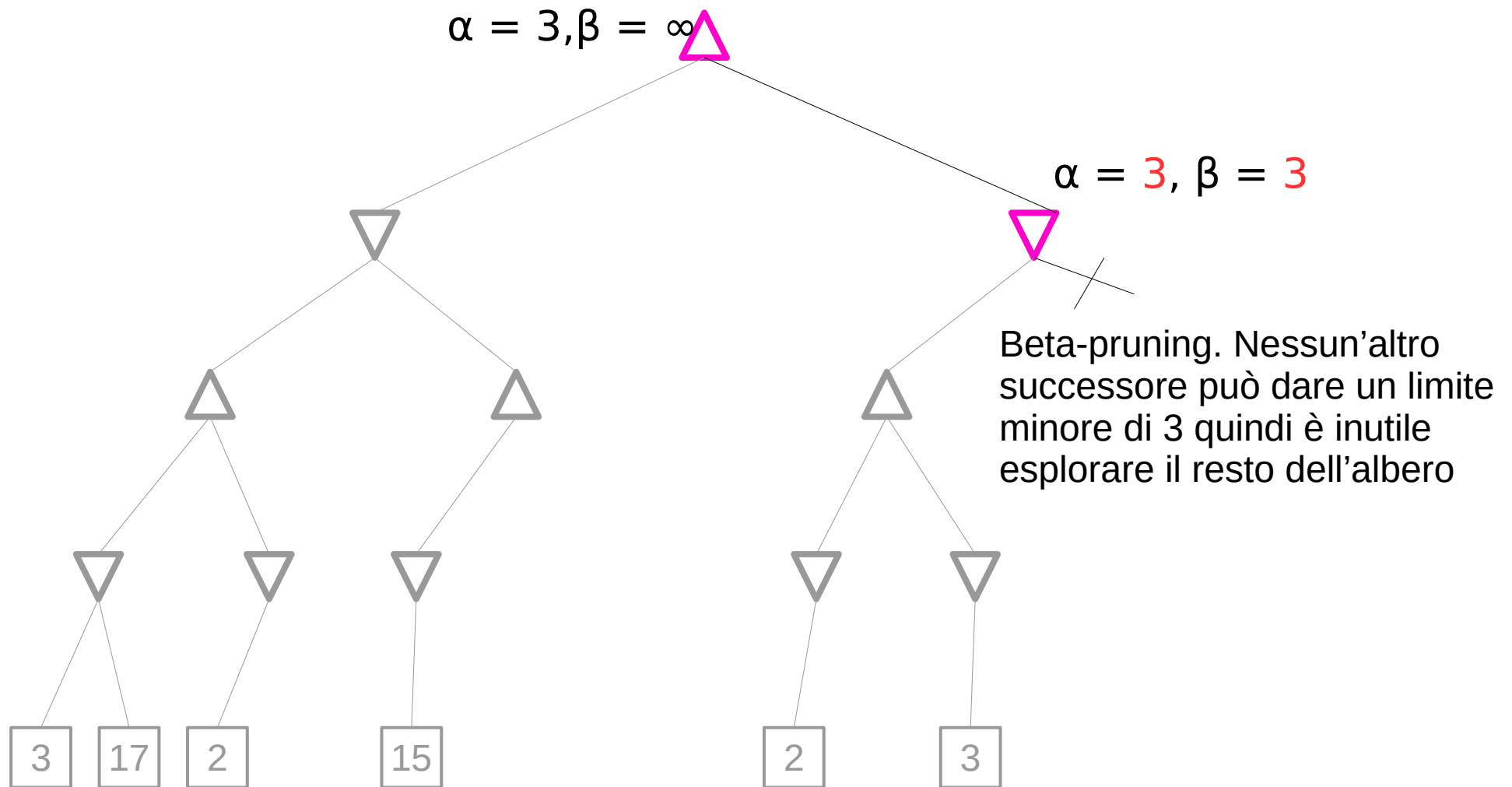
$\alpha = 3, \beta = 2$

$\alpha = 3, \beta = 3$

3 17 2 15

2 3

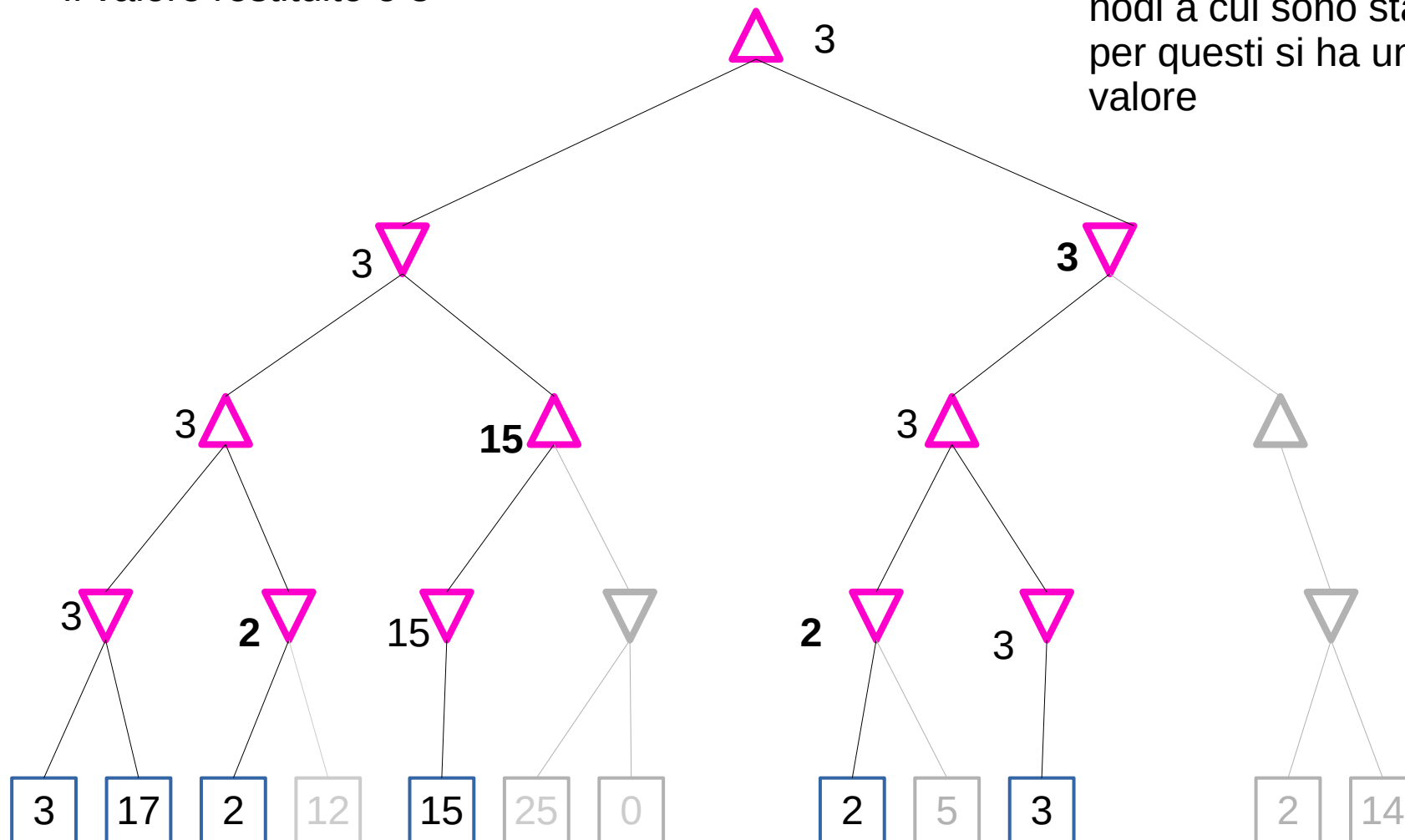
Esempio: albero di gioco completo



Esempio: albero di gioco completo

Esaminati 11 su 20 nodi
Il valore restituito è 3

I valori calcolati per i nodi sono
gli stessi di minimax tranne per i
nodi a cui sono stati potati figli,
per questi si ha una stima del
valore



Confronto con minimax

- Gli algoritmi **alpha-beta pruning** e **minimax** sono equivalenti nel senso che:
 - entrambi trovano la stessa mossa ottima per il nodo radice
 - ed attribuiscono al nodo radice la stessa valutazione
- Alfa-beta raggiunge questo risultato espandendo molti meno nodi, la complessità temporale può essere **$O(b^{m/2})$** contro **$O(b^m)$**
 - Piccolo esempio intuitivo: 2^4 contro 2^8 significa 16 contro 256 nodi espansi



Attenzione!

- Alpha-beta pruning è più o meno efficace a seconda dell'ordine con cui i successori di ciascun nodo sono considerati:
- se il successore più promettente è l'ultimo a essere considerato non è possibile evitare di esplorare i sottoalberi dei suoi fratelli
- La complessità vista nella slide precedente vale quando è possibile espandere i figli più promettenti (**killer move**) per primi (ordinamento dei successori)
- In questo caso il branching factor diventa circa la radice quadrata del branching factor del problema (esempio: negli scacchi il branching factor è 35, con ordinamento e alpha-beta pruning diventa 6)
- **Quando l'ordinamento dei successori non è possibile, la complessità diventa $O(b^{3m/4})$**

Come trovare le killer moves?

- Tramite **apprendimento**: il sistema “ricorda” le esperienze passate e usa questa esperienza per scegliere la mossa più promettente (efficace ma richiede tempo per imparare)
- Combinazione della potatura alfa-beta con iterative deepening
- Uso di **tabelle di trasposizione** (talvolta in aggiunta ad alfa-beta + iterative deepening)

Trasposizione

- In alcuni problemi, eseguendo un certo insieme di mosse, è possibile ottenere sempre uno stesso risultato anche se le mosse sono ordinate differentemente. I diversi ordinamenti sono detti **trasposizioni**

M1 M2 M3 M4

Mi mosse di un gioco

M2 M3 M1 M4

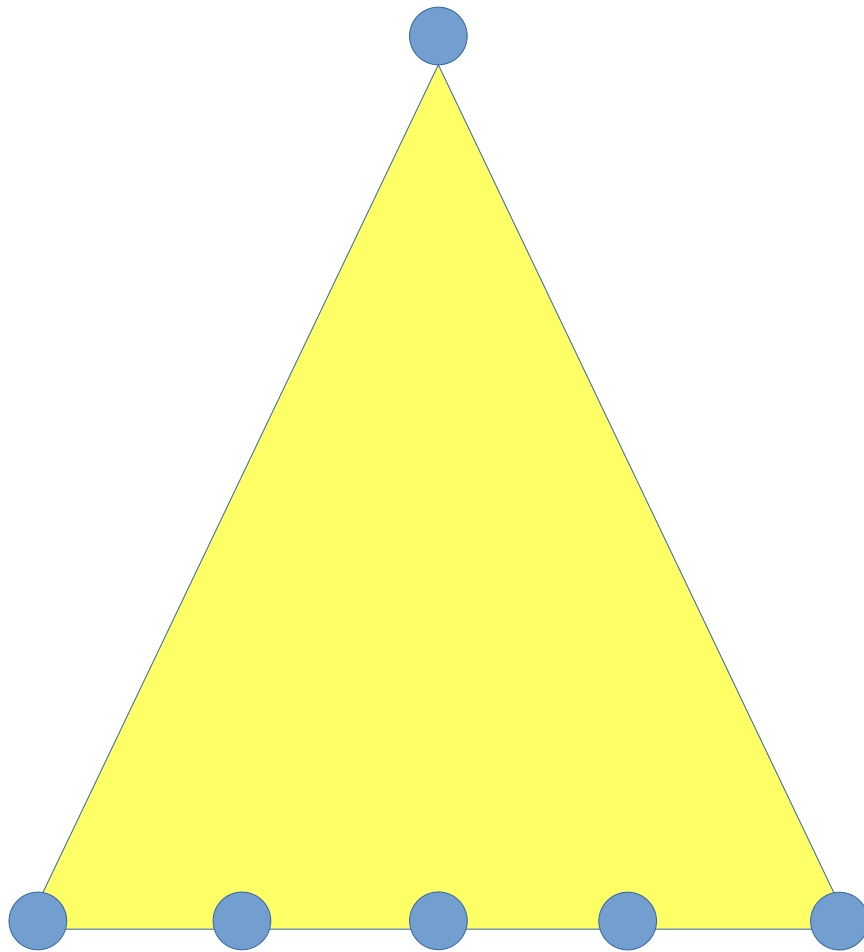
I due ordinamenti conducono a
uno stesso stato

- Quando lo spazio degli stati è grande riconoscere le trasposizioni è importante per evitare di esplorare più volte gli stessi stati.
- A questo fine le trasposizioni sono conservate in una hash table (**tabella delle trasposizioni**) e ogni volta che si genera un nuovo stato si controlla se corrisponde a uno stato già generato da una trasposizione. Se sì non viene esplorato
- Quando lo spazio degli stati supera le capacità di memorizzazione, è talvolta necessario mantenere nella tabella delle trasposizioni soltanto quelle usate più di frequente o più di recente

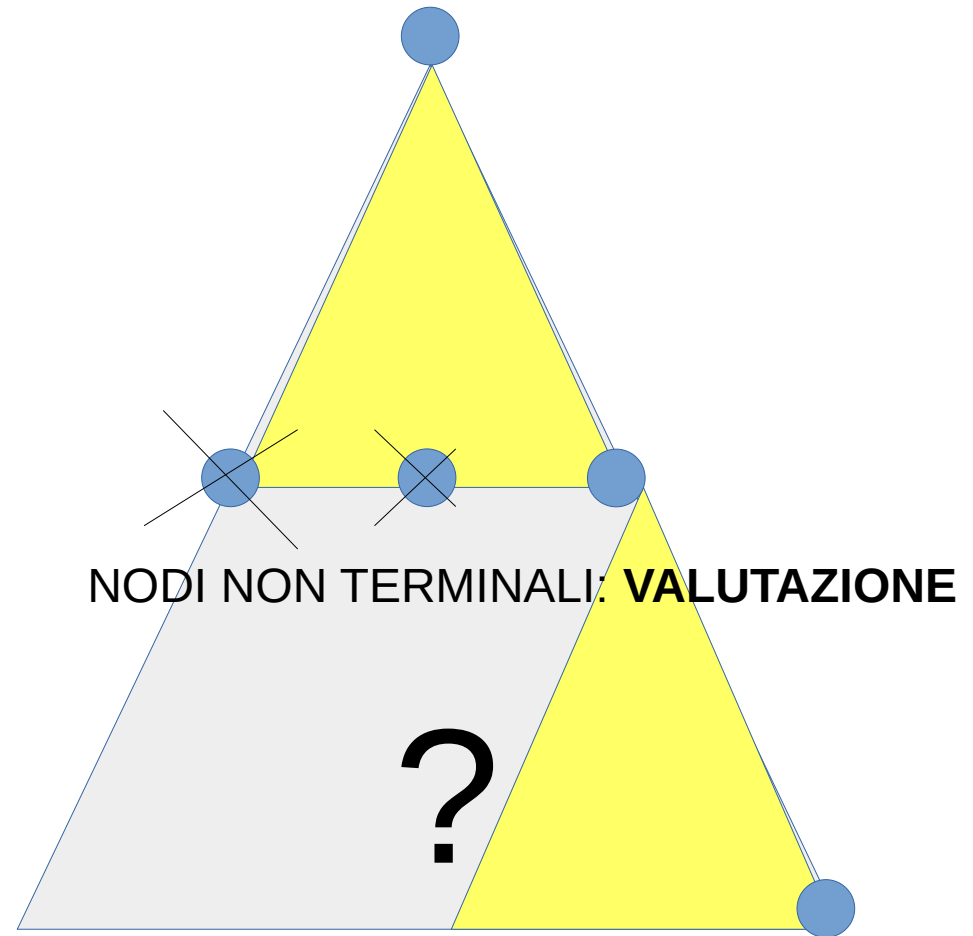
Uso di alfa-beta in contesti real-time

- Alfa-beta concentra la ricerca su una porzione limitata dello spazio degli stati ma deve comunque arrivare agli stati terminali
- Può diventare **troppo lento** nel produrre la risposta quando il nodo terminale è situato a grande profondità (esempio: scacchi)
- *Questo è un problema quando il problema di ricerca è **in real-time**, cioè tale da porre un vincolo sui tempi di risposta*
- In questo caso è necessario introdurre dei test di “**cutoff**” per produrre una decisione prima di raggiungere il nodo terminale

Cutoff basato su funzioni di valutazione



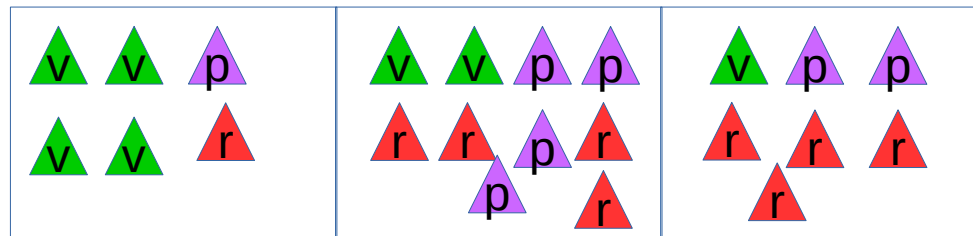
NODI TERMINALI: **UTILITÀ**



La funzione di valutazione stima il guadagno atteso essendo in un certo stato, E' usata per interrompere la ricerca lungo cammini dove la probabilità di vincere è bassa

Fz. di valutazione e categorizzazione degli stati

- In molti problemi di ricerca è possibile partizionare l'insieme degli stati basandosi su **caratteristiche** degli stati stessi
- Ogni insieme di stati prodotto conterrà stati che portano alla **vittoria**, altri che portano al **pareggio**, altri alla **sconfitta**



- Una **funzione di valutazione** fa una stima della bontà di uno stato intesa come percentuale di presenza degli stati che portano alla vittoria rispetto agli altri
- In altri termini calcola la probabilità di essere in uno stato che porta a vittoria conoscendo solo la classe di appartenenza dello stato

Troppe classi!

- Purtroppo nei problemi di interesse reale il numero delle classi che è necessario identificare perché la valutazione sia significativa è troppo alto
- **Alternativa:**
 - Identificate le caratteristiche
 - Attribuire un peso a ciascuna di esse
 - Usare una funzione lineare che combina tali caratteristiche (i loro valori o la loro presenza)
 - $\text{eval}(s) = \sum_{i \in [1,k]} w_i * f_i$
- **Esempio:** negli scacchi le f_i potrebbero essere il numero di pedoni, e via via degli altri pezzi ancora a disposizione del giocatore
- **NB:** quella lineare è la forma più semplice di combinazione che si possa realizzare.
Ricordatevene quando parleremo di reti neurali

Alfa-beta modificato

- Data la funzione di valutazione si può modificare l'algoritmo sostituendo all'istruzione "if (TEST-TERMINALE (...)) ..." l'istruzione:

if [TEST-TAGLIO(stato, profondità)] **then** return eval(stato)

- **Quando tagliare? Tante possibilità, esempi:**
 - 1) Raggiunta una profondità massima predefinita
 - 2) Iterative deepening:
 - quando devo muovere, uso tutto il tempo disponibile per la mossa per fare cercare la mossa migliore per approfondimenti crescenti.
 - Allo scadere del tempo restituisco la mossa migliore che ho trovato

Problema dell'orizzonte

- Gli algoritmi che introducono tagli artificiali incorrono nel **problema dell'orizzonte**:
 - l'algoritmo non vede oltre il punto di taglio ma ...
 - ... in alcune fasi il gioco si può capovolgere il fretta o, in altri termini, la funzione di valutazione è instabile
 - Tagliare in questi punti è prematuro perché rischioso: l'avversario potrebbe successivamente forzare un forte cambiamento della valutazione

- Ne consegue che: “The decision as to whether to terminate the search at a node or continue, has to be a function of the information that exists at that node and how this relates to the **quiescence** of each and every term in the evaluation function”(tratto da: Some necessary conditions for a master chess program, Hans J. Berliner, 3rd Int. Joint Conf. On AI, 1973)
- La nozione di quiescenza concerne la **permanenza della negatività (o positività) della valutazione**
- Si taglieranno nodi la cui valutazione è quiescente mentre quelli non quiescenti richiederanno un po' di esplorazione ulteriore dei sottoalberi che li vedono come radici
- Berliner è stato il realizzatore del primo programma che abbia mai battuto un maestro in un qualsiasi gioco, nel suo caso il backgammon

Alcuni programmi che giocano

- Checkers (Samuel, Chinook)
- Othello (Logistello)
- Backgammon (TD-gammon)
- Go (AlphaGo)
- Bridge (Bridge Baron, GIB)
- Chess (DeepBlue)

- Primo calcolatore a vincere una partita a scacchi contro un Campione del Mondo in carica, Garry Kasparov, con cadenza di tempo da torneo (10 febbraio 1996)
- Grande potenza computazionale:
 - un computer a parallelismo massivo a **30 nodi** basato su RS/6000, supportato da **480 processori** specifici VLSI progettati per il gioco degli scacchi
 - Sistema operativo: **AIX**
 - L'algoritmo per il gioco degli scacchi è implementato in **linguaggio C**
 - È capace di calcolare **200 milioni di posizioni al secondo**

- **Algoritmo:**

Iterative-deepening, alpha-beta search con tabella delle trasposizioni

- **Chiave del successo:** generare estensioni oltre il limite di profondità della ricerca per posizioni ritenute interessanti
- Di routine la ricerca raggiunge livello **14**, in certi casi **40**
- **Funzione di valutazione:**
 - Aveva oltre **8000 features**, inizializzate manualmente e raffinate automaticamente
 - Utilizzava un **database con 700.000 partite** di gran maestri e un ampio **database di finali di partita** (tutte quelle con 5 pezzi rimanenti e molte di quelle con 6 pezzi)
 - Kasparov ebbe il sospetto che alcune mosse fossero (scorrettamente) state suggerite da un umano

- Sviluppato da google, primo programma che ha battuto senza handicap a **go** un maestro umano, su un goban di dimensioni standard (anno 2015)
- Nelle **500 partite** disputate con altri programmi per il go ha vinto:
 - **Tutte le partite meno una** quando eseguito su un solo computer
 - **Tutte** le partite quando eseguito su di un cluster che impiegava 1202 CPU e 176 GPU, circa 25 volte in più rispetto all'hardware del computer singolo
 - La versione cluster ha battuto la versione su singolo computer nel **77%** delle partite
- **Utilizza deep learning neural networks e ricerca su alberi.** Le reti neurali sono state addestrate su un **dataset di 30.000.000 di mosse** e poi raffinate giocando contro se stesse

E gli esseri umani?

- Abbiamo studiato il modo in cui diversi algoritmi di ricerca mantengono informazione sull'albero (grafo) di ricerca per determinare una soluzione
- Studio basato sul numero di nodi “da ricordare”
- **Quanti elementi è in grado di tenere contemporaneamente a mente la memoria di lavoro (o a breve termine) umana?**

Legge di Miller (1956)

- 7 ± 2

Cowan (2001)

- 4 ± 1

Cowan, N. (2001). "The magical number 4 in short-term memory: A reconsideration of mental storage capacity". Behavioral and Brain Sciences. 24: 97-185.