

Procura Adversária e Jogos

Capítulo 6

Enquadramento

- Ambientes **multi-agente**
 - Introduzem contingência
- Conflitos entre objetivos dos agentes
 - **Competições**, jogos!
- **Teoria dos jogos** em matemática / economia
 - Impacto de um agente é significativo para os outros agentes
- **Área tradicional** / importante em IA
 - Jogos *zero-sum* com informação perfeita
 - Determinísticos e completamente observáveis

Jogos vs. Problemas de Procura

- **Adversário "imprevisível"** → necessidade de tomar em consideração todas os movimentos que podem ser tomados pelo adversário
- **Pontuação com sinais opostos**
 - O que é bom para um jogador (vitória=+1) é mau para o outro (derrota=-1)
- **Limitação temporal** → tipicamente não é alcançado um objetivo mas antes uma aproximação

Sumário

- Jogos: conceitos básicos
- Decisões ótimas em jogos
 - Estratégias ótimas e o Minimax
 - Estratégias ótimas com múltiplos jogadores
- Procura α - β
- Procura α - β heurística
- Procura Monte Carlo
- Jogos estocásticos
- Jogos parcialmente observáveis
- Limitações

Ingredientes

- Vamos considerar que:
 - existem dois jogadores, o MAX e o MIN
 - o MAX joga primeiro
 - no fim do jogo o vencedor ganha pontos e o adversário é penalizado
- Qual a próxima (melhor) jogada?

Caraterização de um Jogo

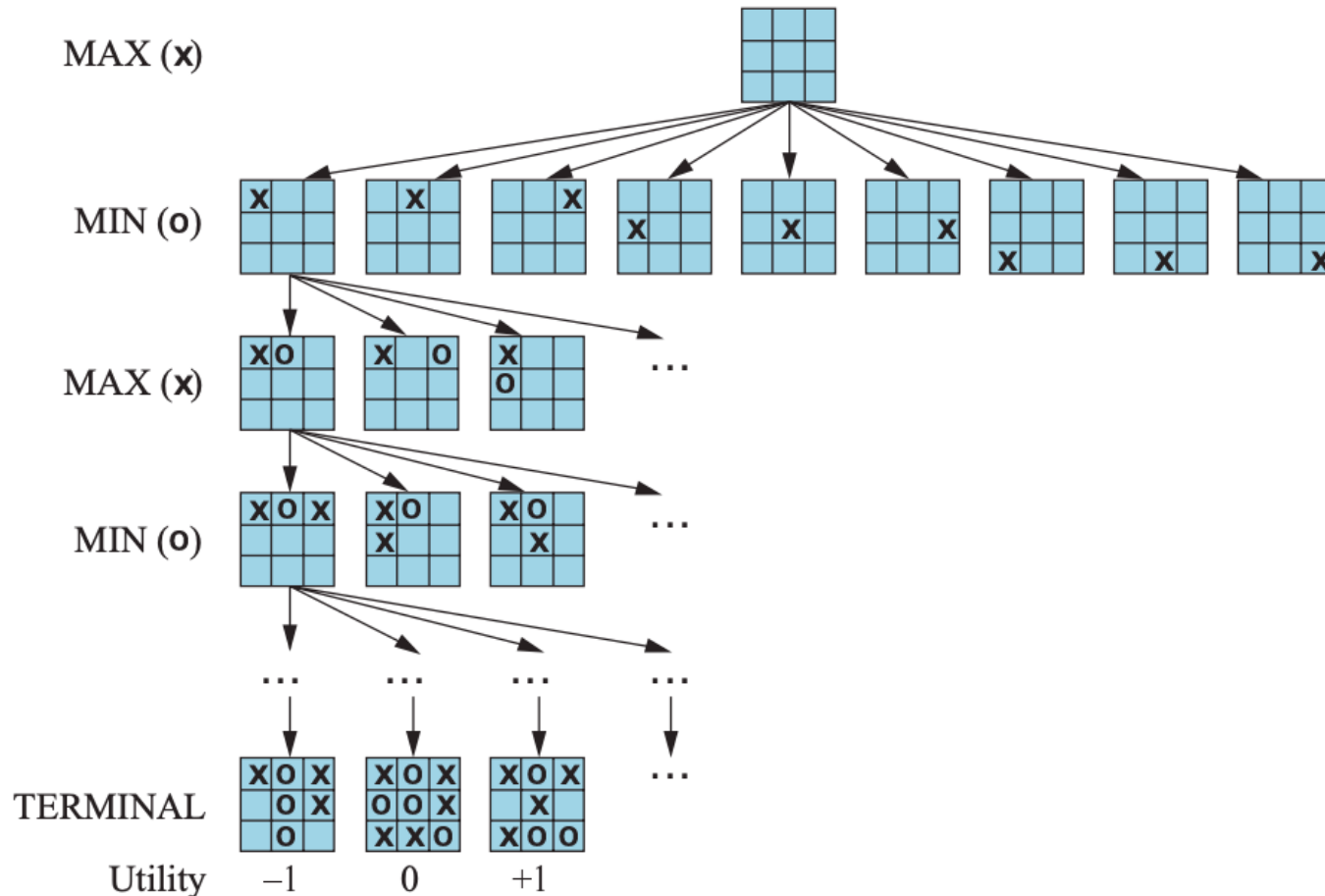
- S_0 : estado inicial
- $To-Move(s)$: qual dos jogadores joga no estado s
- $Actions(s)$: conjunto de movimentos possíveis num estado s
- $Result(s,a)$: modelo de transição que define resultado de ação a no estado s
- $Is-Terminal(s)$: teste para estados terminais
- $Utility(s,p)$: função de utilidade ou função objetivo que define valor numérico para um estado terminal s para um jogador p

Árvore do jogo

- O estado inicial e os movimentos possíveis para cada jogador definem a **árvore do jogo**.

Árvore para o jogo do galo

(2 jogadores, determinístico, alternado)



Sumário

- Jogos: Conceitos Básicos
- Decisões ótimas em jogos
 - Estratégias ótimas e o Minimax
 - Estratégias ótimas com múltiplos jogadores
- Procura α - β
- Procura α - β heurística
- Procura Monte Carlo
- Jogos estocásticos
- Jogos parcialmente observáveis
- Limitações

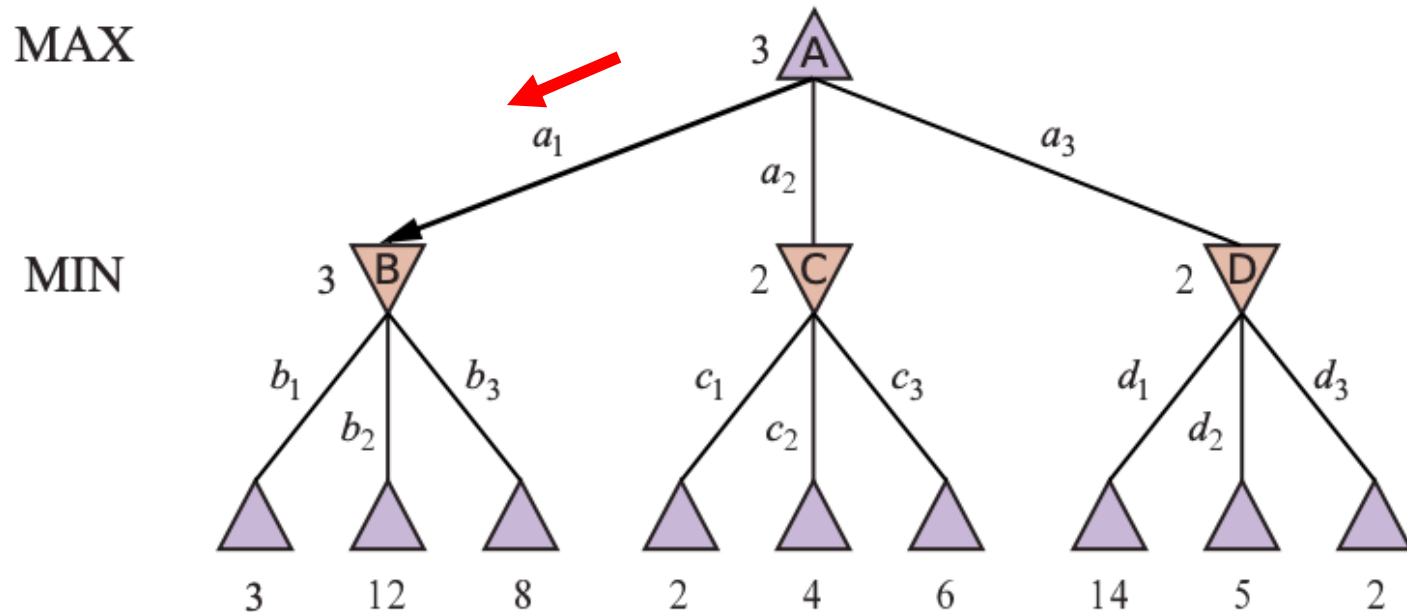
Minimax

- Estratégia mais adequada para jogos determinísticos
- Ideia: escolher jogada para o estado com o maior **valor minimax**
 - melhor valor para a função de utilidade contra as melhores jogadas do adversário

MINIMAX(s) =

$$\begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

Minimax: 2 jogadores



- Observações:
 - Formato dos nós em função do tipo de nó (MIN/MAX)
 - Valores dos estados terminais correspondem à função de utilidade para MAX (quanto vale cada um dos estados terminais)
 - Valores para os restantes estados obtidos a partir dos valores para os nós terminais através do cálculo do valor-minimax
 - Resultado do algoritmo: próxima jogada!

Algoritmo Minimax

function MINIMAX-SEARCH(*game, state*) **returns** *an action*

player \leftarrow *game*.TO-MOVE(*state*)

value, move \leftarrow MAX-VALUE(*game, state*)

return *move*

function MAX-VALUE(*game, state*) **returns** *a (utility, move) pair*

if *game*.IS-TERMINAL(*state*) **then return** *game*.UTILITY(*state, player*), *null*

v, move $\leftarrow -\infty$

for each *a* **in** *game*.ACTIONS(*state*) **do**

v2, a2 \leftarrow MIN-VALUE(*game, game*.RESULT(*state, a*))

if *v2* > *v* **then**

v, move \leftarrow *v2, a*

return *v, move*

function MIN-VALUE(*game, state*) **returns** *a (utility, move) pair*

if *game*.IS-TERMINAL(*state*) **then return** *game*.UTILITY(*state, player*), *null*

v, move $\leftarrow +\infty$

for each *a* **in** *game*.ACTIONS(*state*) **do**

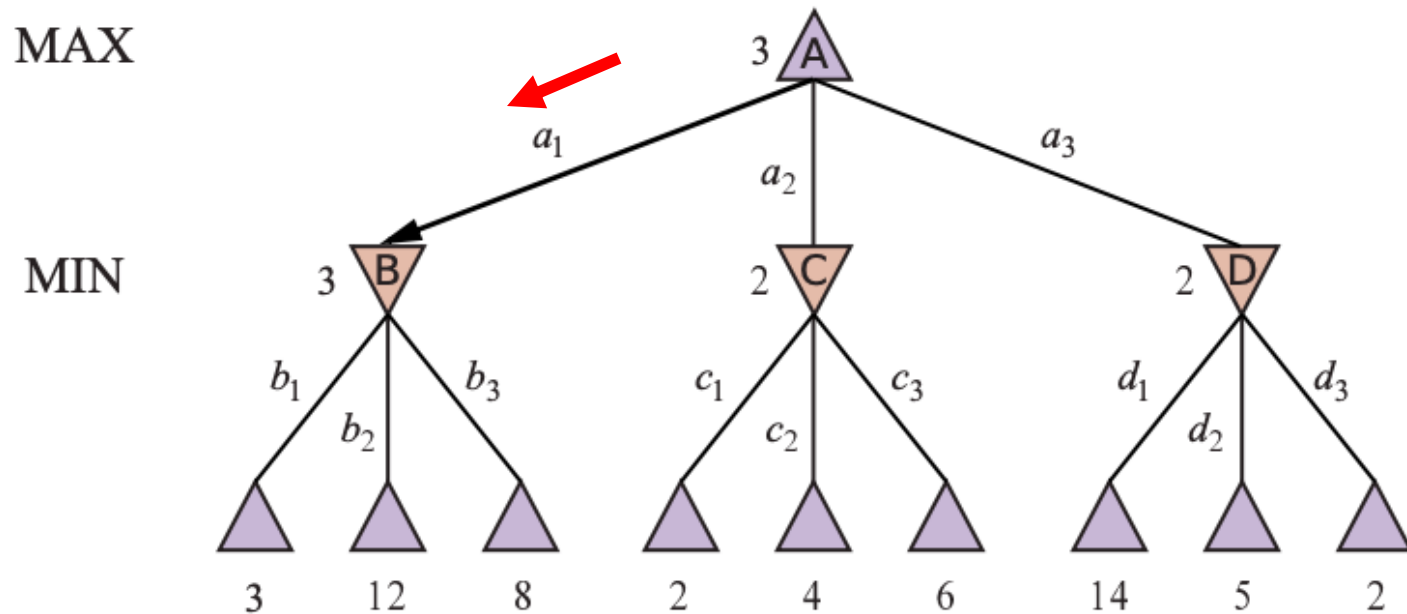
v2, a2 \leftarrow MAX-VALUE(*game, game*.RESULT(*state, a*))

if *v2* < *v* **then**

v, move \leftarrow *v2, a*

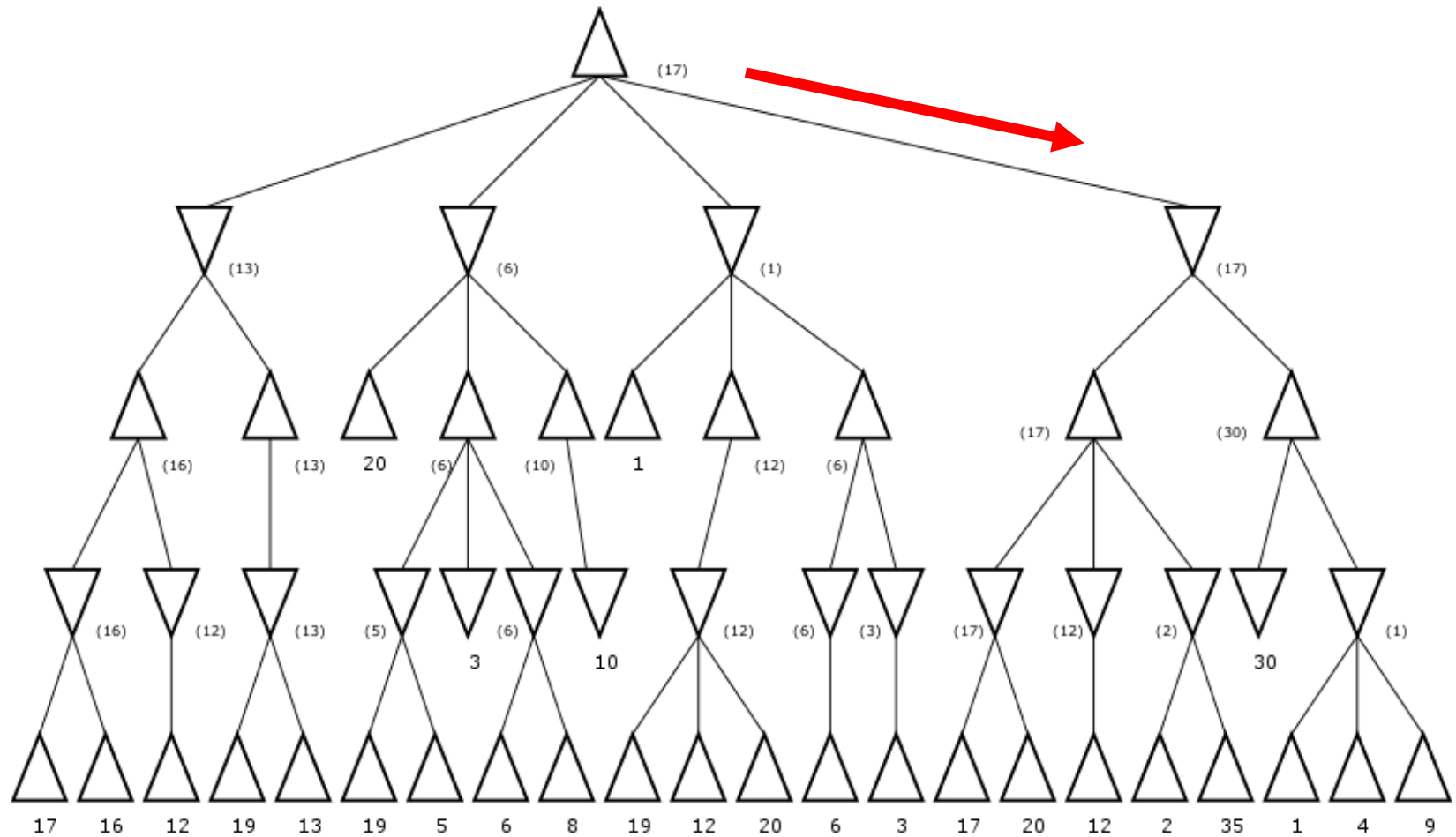
return *v, move*

Minimax: 2 jogadores



- Começa na raiz e a recursão conduz os cálculos até às folhas. Os valores minimax são depois usados quando termina a fase de expansão da recursão;
- Quando é o Min a jogar, escolhe a jogada que dá menos pontos ao MAX;
- Quando é o Max a jogar, escolhe a jogada que lhe dá mais pontos;
- A seta indica a escolha de MAX no fim da aplicação do algoritmo.

Minimax: outro exemplo



Propriedades do algoritmo minimax

O algoritmo faz uma procura em profundidade, explorando toda a árvore de jogo.

- É completo? Sim (se a árvore de procura é finita)
- É óptimo? Sim (contra um adversário ótimo)

Considerando:

d = profundidade máxima da árvore

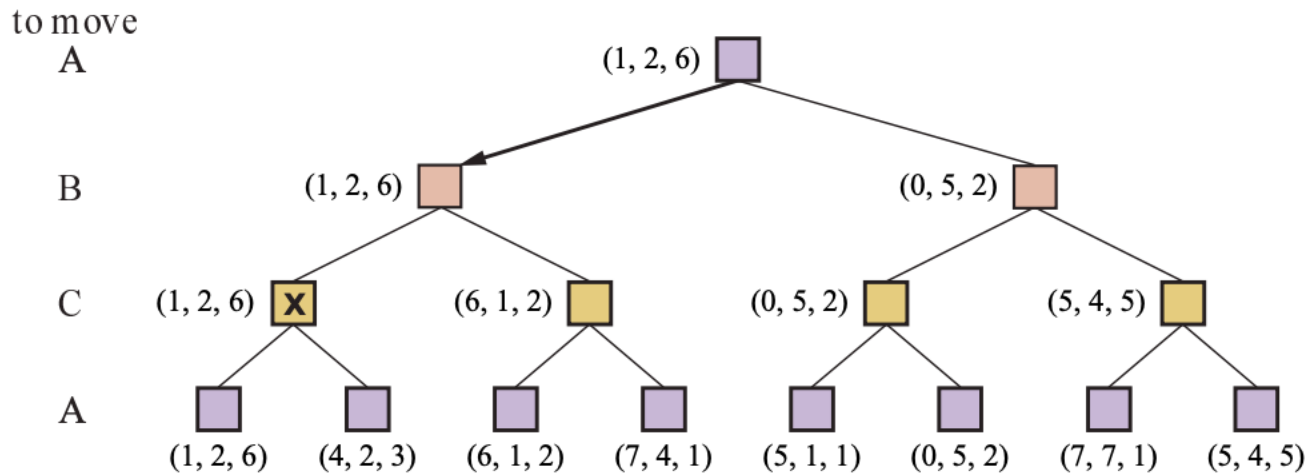
b = nº de movimentos legais em cada ponto

- Complexidade temporal?
 - $O(b^d)$
- Complexidade espacial?
 - $O(b \cdot d)$ para um algoritmo que gera todos os sucessores de uma vez
 - $O(d)$ para um algoritmo que gera os sucessores um a um
- Para xadrez, $b \approx 35$, $d \approx 100$ para um jogo padrão
→ impossível determinar a solução exata

Sumário

- Jogos: Conceitos Básicos
- Decisões ótimas em jogos
 - Estratégias ótimas e o Minimax
 - Estratégias ótimas com múltiplos jogadores
- Procura α - β
- Procura α - β heurística
- Procura Monte Carlo
- Jogos estocásticos
- Jogos parcialmente observáveis
- Limitações

Minimax: mais de 2 jogadores



- Função de utilidade devolve **vetor de valores** com utilidade do estado do ponto de vista de cada jogador
- Cada jogador procura **maximizar a sua utilidade** (ex: C em X escolhe a jogada que lhe dá mais pontos, i.e. 6)

Minimax: mais de 2 jogadores

- Tipicamente levam a alianças, formais ou informais, entre os jogadores (também podem existir alianças em jogos com 2 jogadores).
- Estas alianças podem ser uma consequência natural de estratégias ótimas para cada jogador num jogo multi-jogadores.

Sumário

- Jogos: Conceitos Básicos
- Decisões ótimas em jogos
 - Estratégias ótimas e o Minimax
 - Estratégias ótimas com múltiplos jogadores
- Procura α - β
- Procura α - β heurística
- Procura Monte Carlo
- Jogos estocásticos
- Jogos parcialmente observáveis
- Limitações

Procura com Cortes

- Jogos são muito mais difíceis do que os problemas de procura
- Fator de ramificação muito elevado - e.g. xadrez
 - fator de ramificação ≈ 35
 - 50 jogadas/jogador $\rightarrow 35^{100}$ nós (destes “apenas” 10^{40} são nós distintos)
- Cortes permitem eliminar partes da árvore de procura que são irrelevantes para o resultado final

Cortes α - β

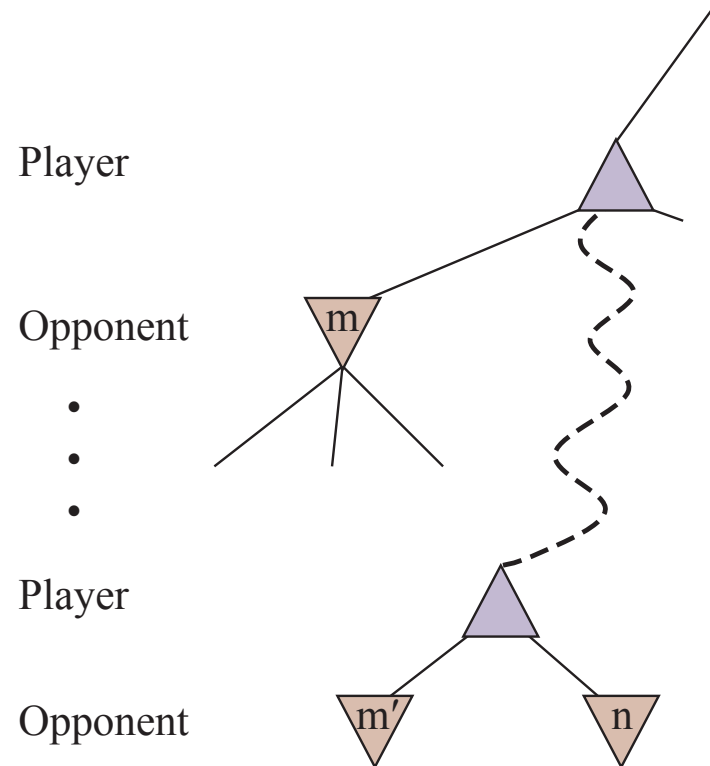
- Motivação:
 - Minimax: número de estados examinados é **exponencial** em função do número de jogadas
 - Não é possível eliminar o fator exponencial, mas podemos **reduzir o número de estados analisados** para metade!
 - É possível calcular **exatamente a mesma decisão** resultante do algoritmo Minimax sem ter de analisar todos os estados

Cortes α - β

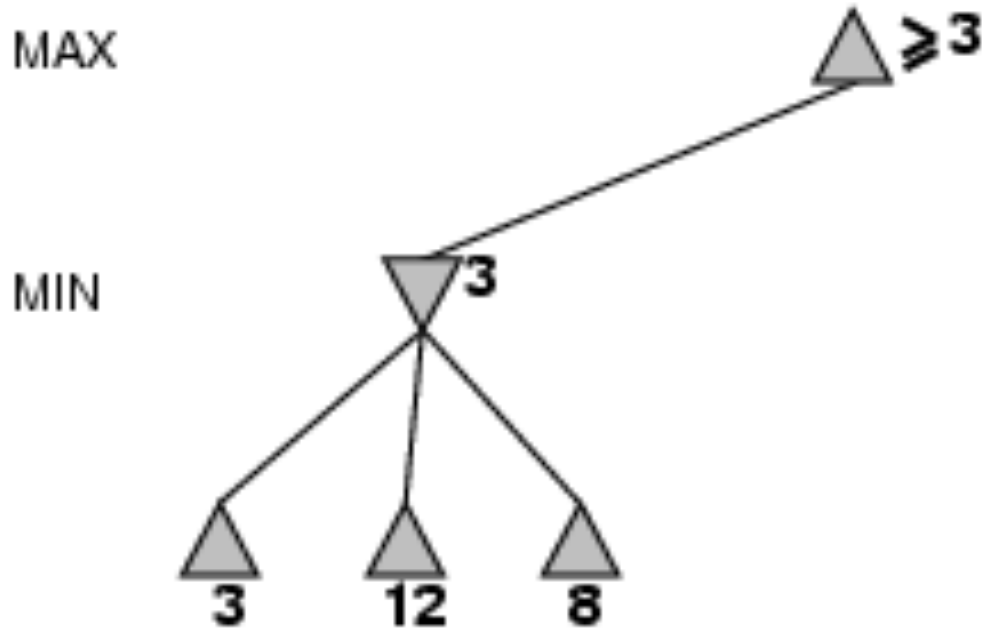
- α é o valor da **melhor escolha** (i.e., valor mais elevado) encontrada até ao momento em qualquer ponto de procura ao longo do caminho para ***max***
- β define-se para ***min*** de forma análoga

Cortes α - β

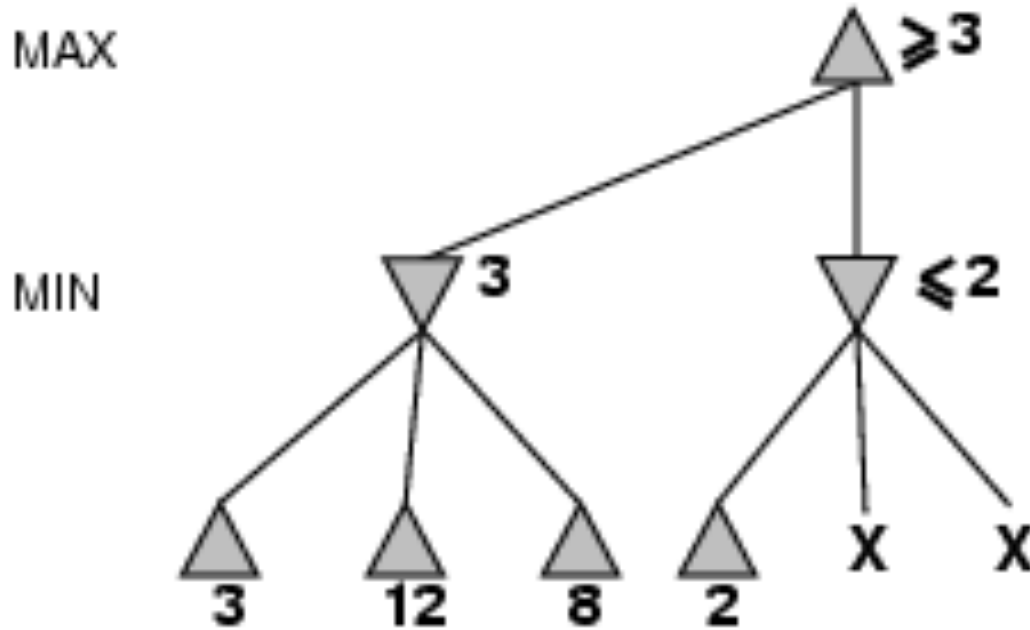
- Considere-se um nó n algures na árvore, tal que um jogador tem a hipótese de se mover para esse nó. Se o jogador tem uma hipótese melhor num nó qualquer acima do nó n , então esse n nunca vai ser atingido
- No ex^o, se m ou m' é melhor do que n , nunca chegaremos a n



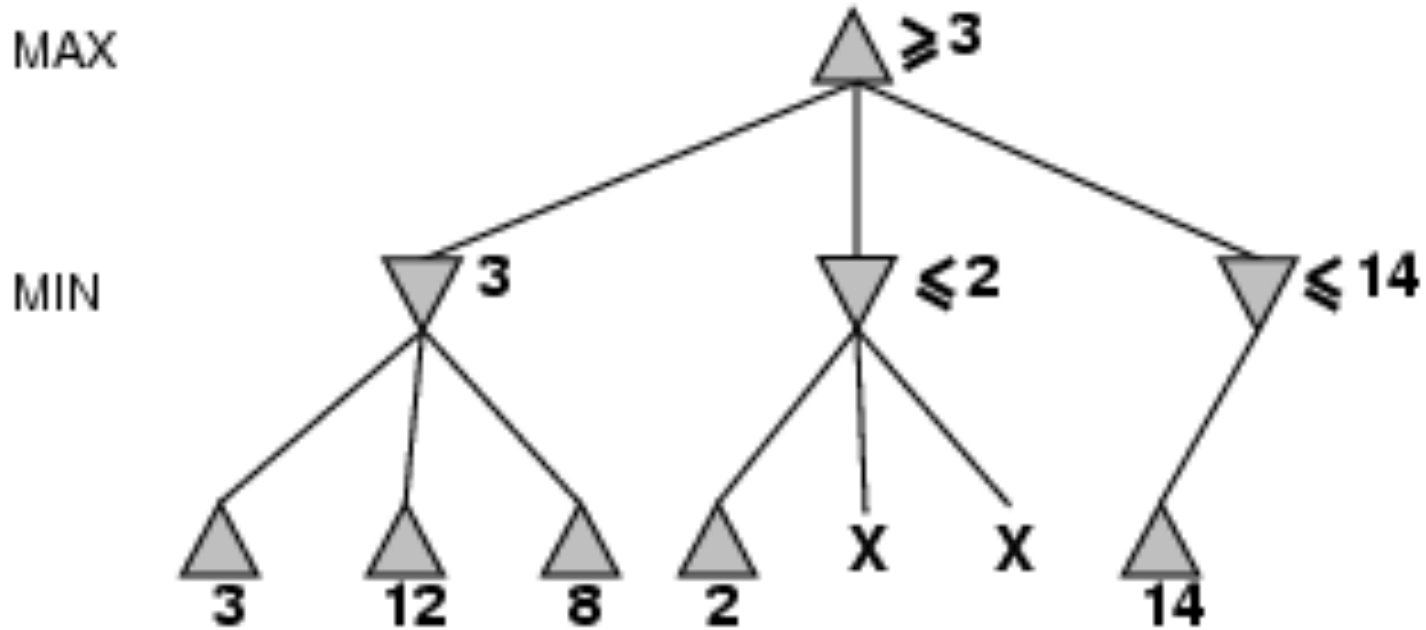
Cortes α - β : exemplo



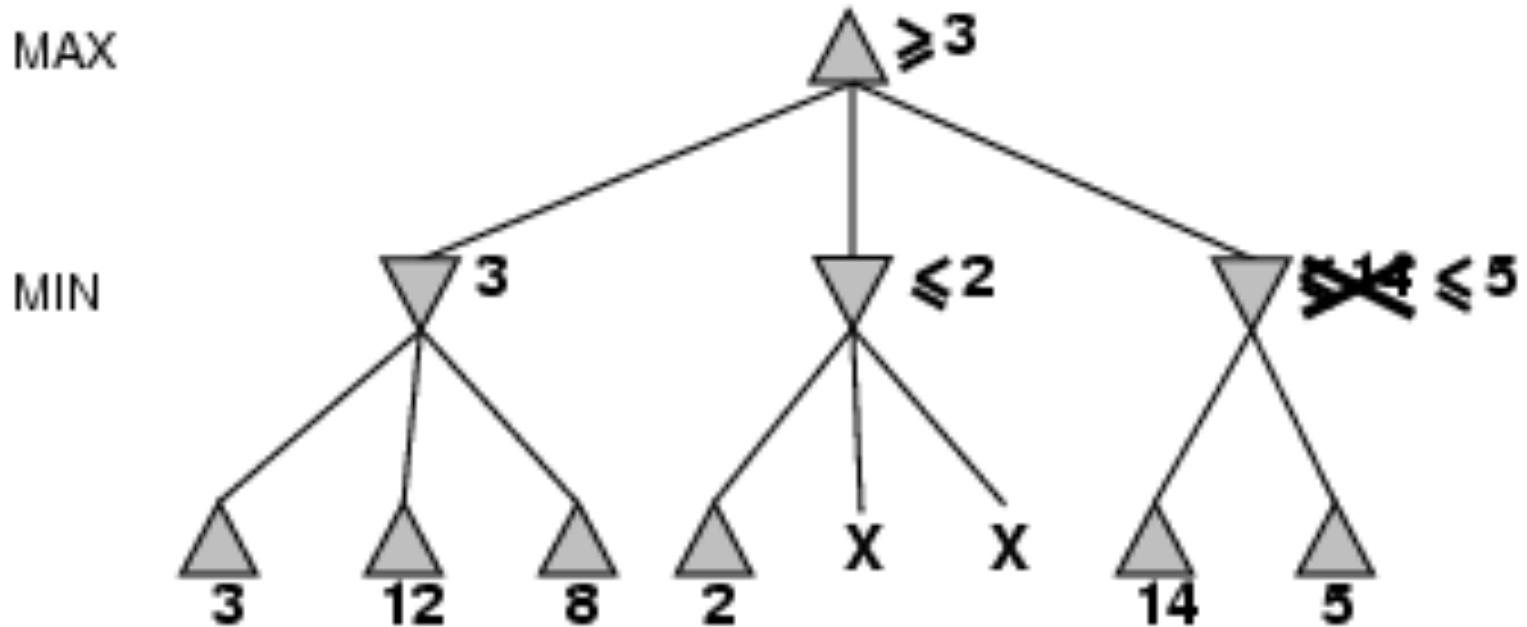
Cortes α - β : exemplo



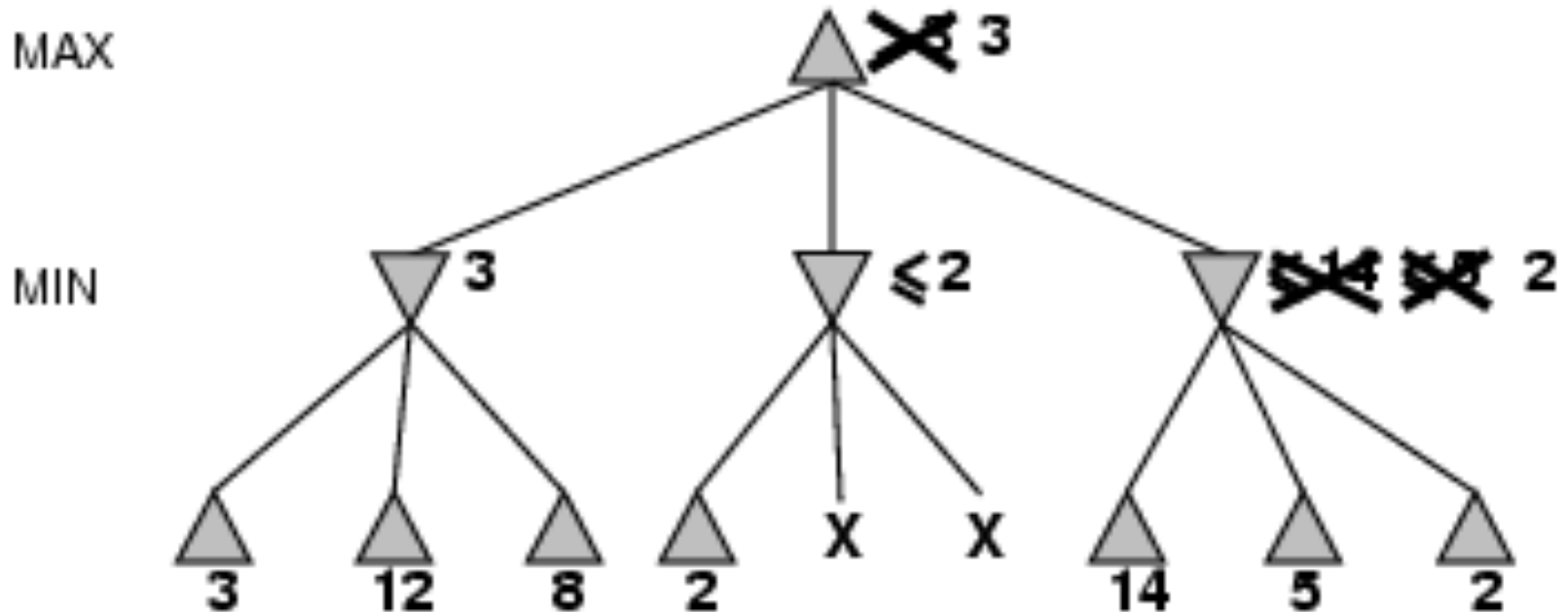
Cortes α - β : exemplo



Cortes α - β : exemplo



Cortes α - β : exemplo



- Os nós sucessores do primeiro nó a ser expandido em cada nível de profundidade nunca podem ser cortados

Algoritmo α - β

function ALPHA-BETA-SEARCH(*game*, *state*) **returns** an action
 $\text{player} \leftarrow \text{game.TO-MOVE}(\text{state})$
 $\text{value}, \text{move} \leftarrow \text{MAX-VALUE}(\text{game}, \text{state}, -\infty, +\infty)$
 return *move*

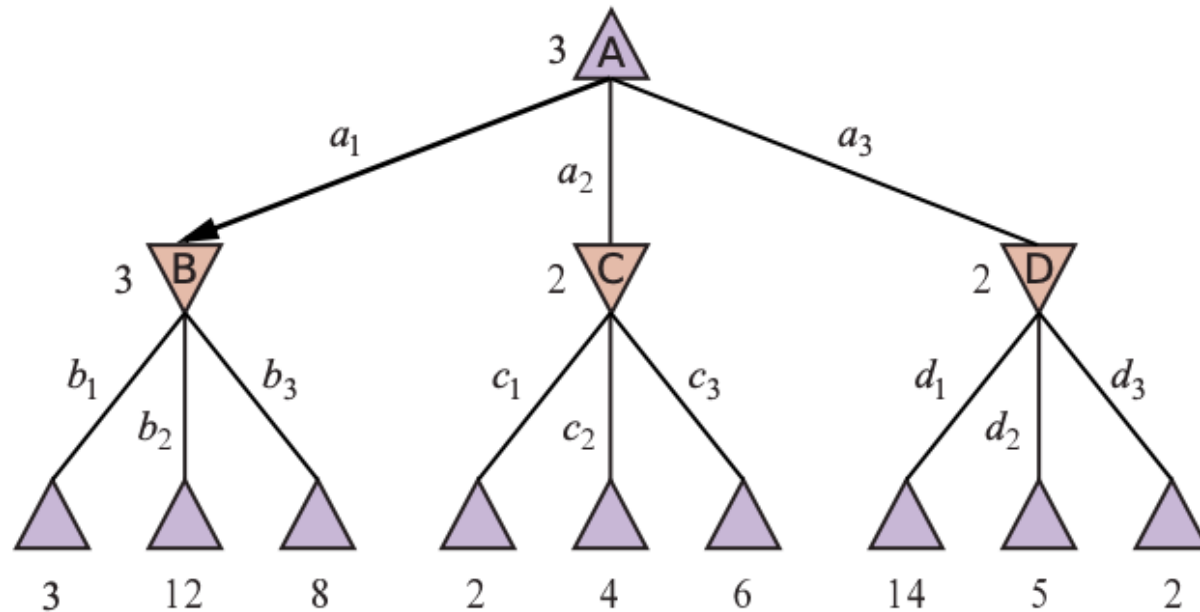
function MAX-VALUE(*game*, *state*, α , β) **returns** a (*utility*, *move*) pair
 if *game.IS-TERMINAL*(*state*) **then return** *game.UTILITY*(*state*, *player*), *null*
 $v \leftarrow -\infty$
 for each *a* **in** *game.ACTIONS*(*state*) **do**
 $v2, a2 \leftarrow \text{MIN-VALUE}(\text{game}, \text{game.RESULT}(\text{state}, a), \alpha, \beta)$
 if $v2 > v$ **then**
 $v, \text{move} \leftarrow v2, a$
 $\alpha \leftarrow \text{MAX}(\alpha, v)$
 if $v \geq \beta$ **then return** *v*, *move*
 return *v*, *move*

function MIN-VALUE(*game*, *state*, α , β) **returns** a (*utility*, *move*) pair
 if *game.IS-TERMINAL*(*state*) **then return** *game.UTILITY*(*state*, *player*), *null*
 $v \leftarrow +\infty$
 for each *a* **in** *game.ACTIONS*(*state*) **do**
 $v2, a2 \leftarrow \text{MAX-VALUE}(\text{game}, \text{game.RESULT}(\text{state}, a), \alpha, \beta)$
 if $v2 < v$ **then**
 $v, \text{move} \leftarrow v2, a$
 $\beta \leftarrow \text{MIN}(\beta, v)$
 if $v \leq \alpha$ **then return** *v*, *move*
 return *v*, *move*

Cortes α - β : intuição

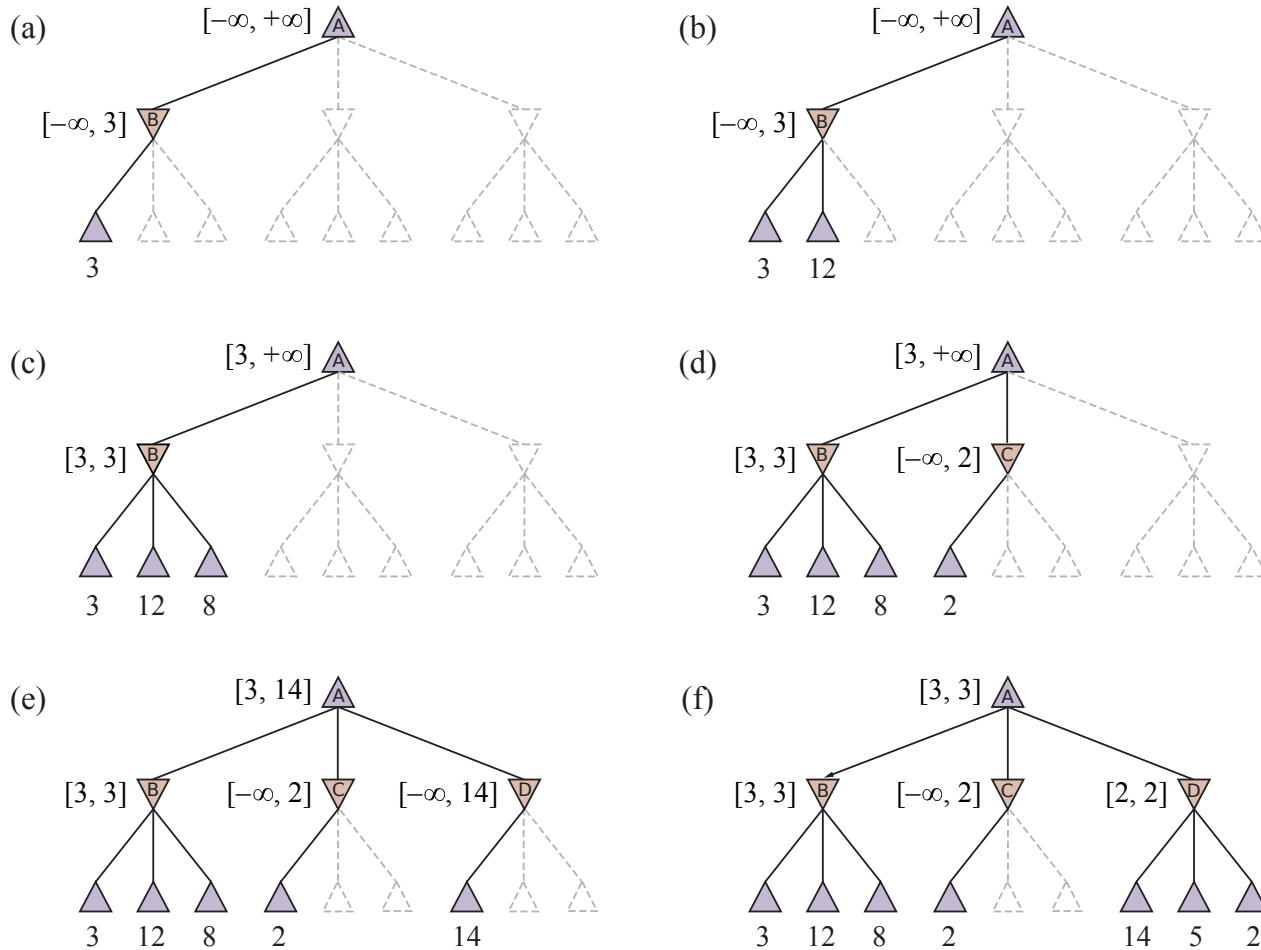
MAX

MIN

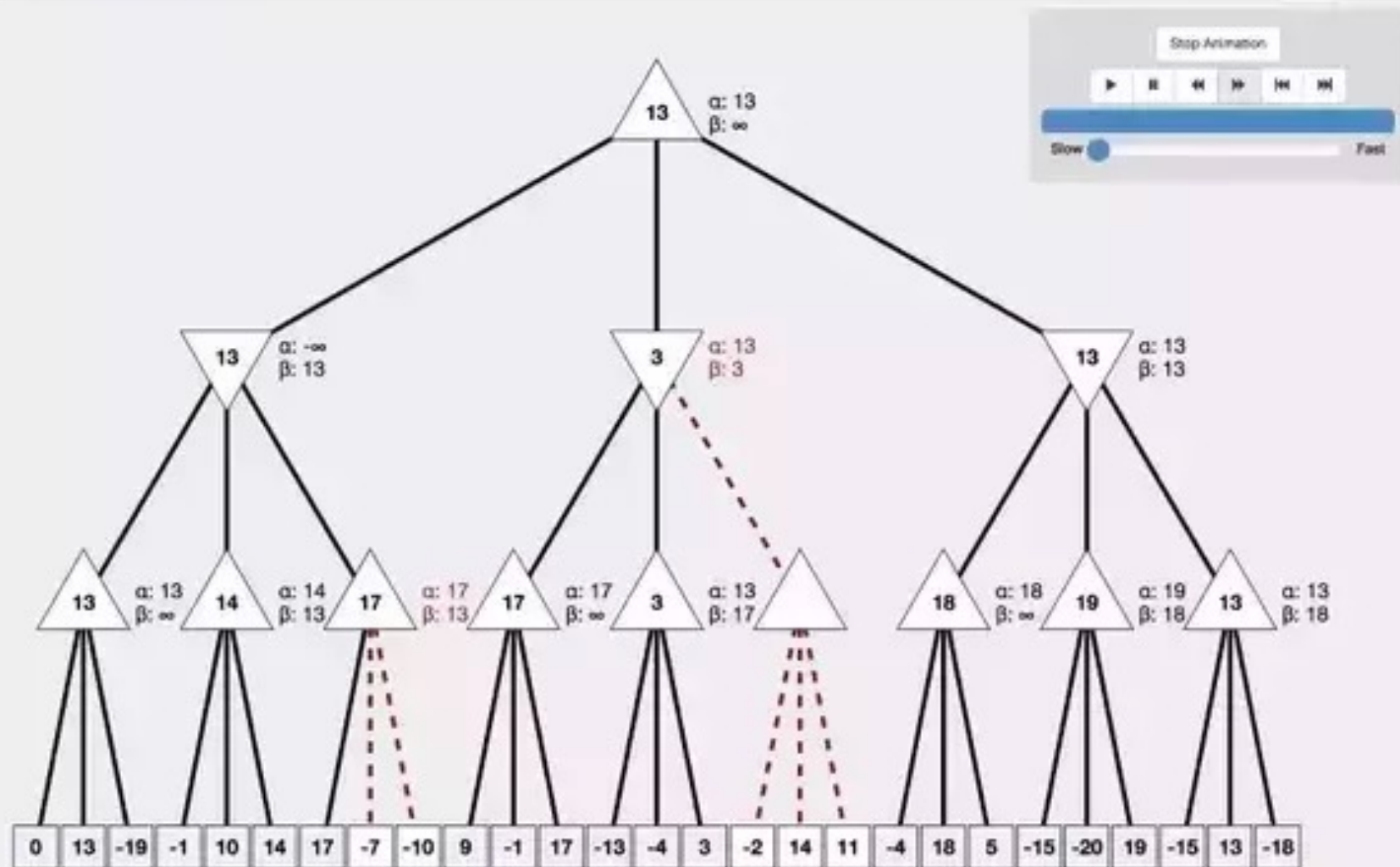


$$\begin{aligned}\text{MINIMAX}(\text{root}) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\ &= \max(3, \min(2, x, y), 2) \\ &= \max(3, z, 2) \quad \text{where } z = \min(2, x, y) \leq 2 \\ &= 3.\end{aligned}$$

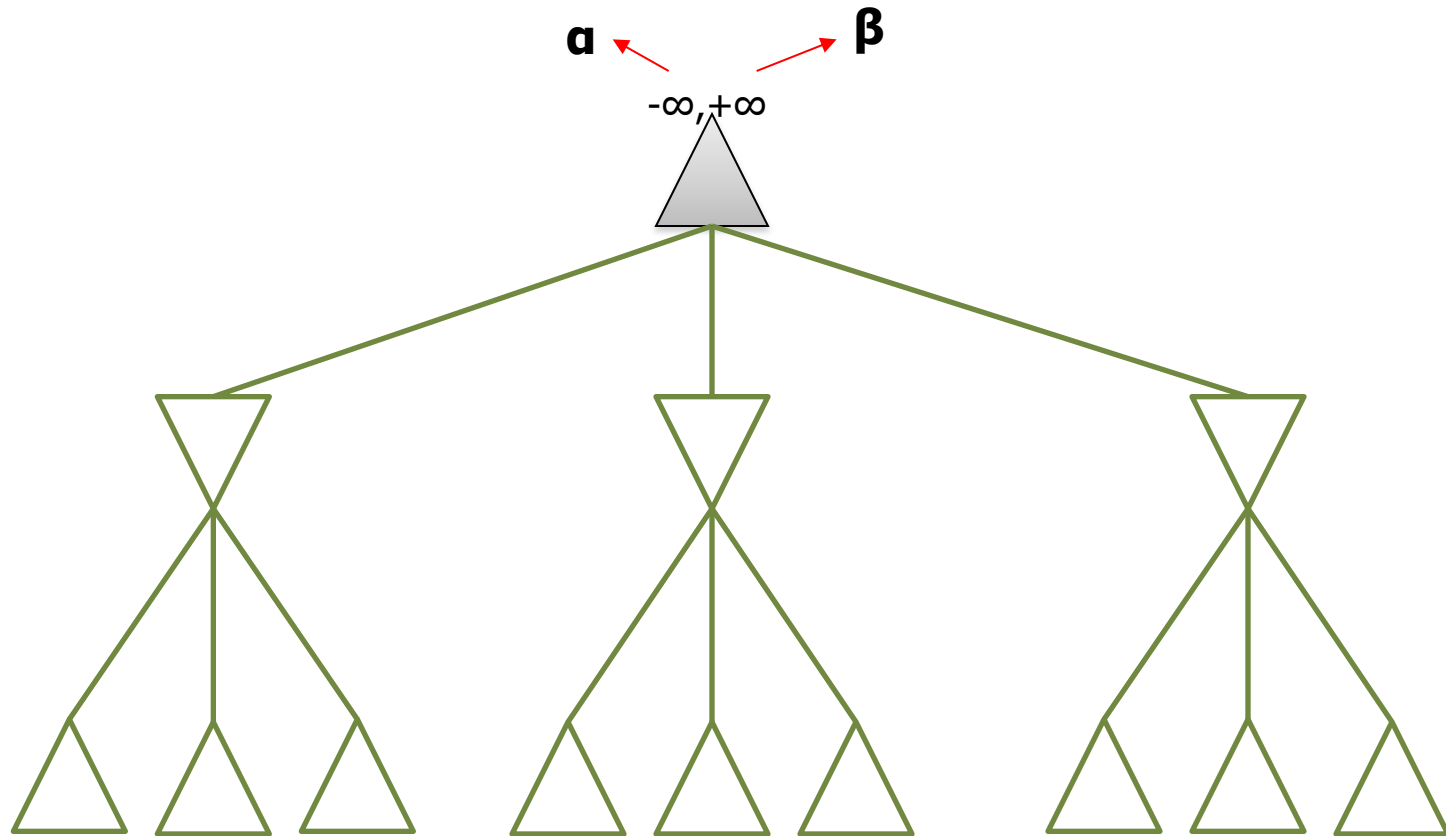
Cortes α - β : exemplo



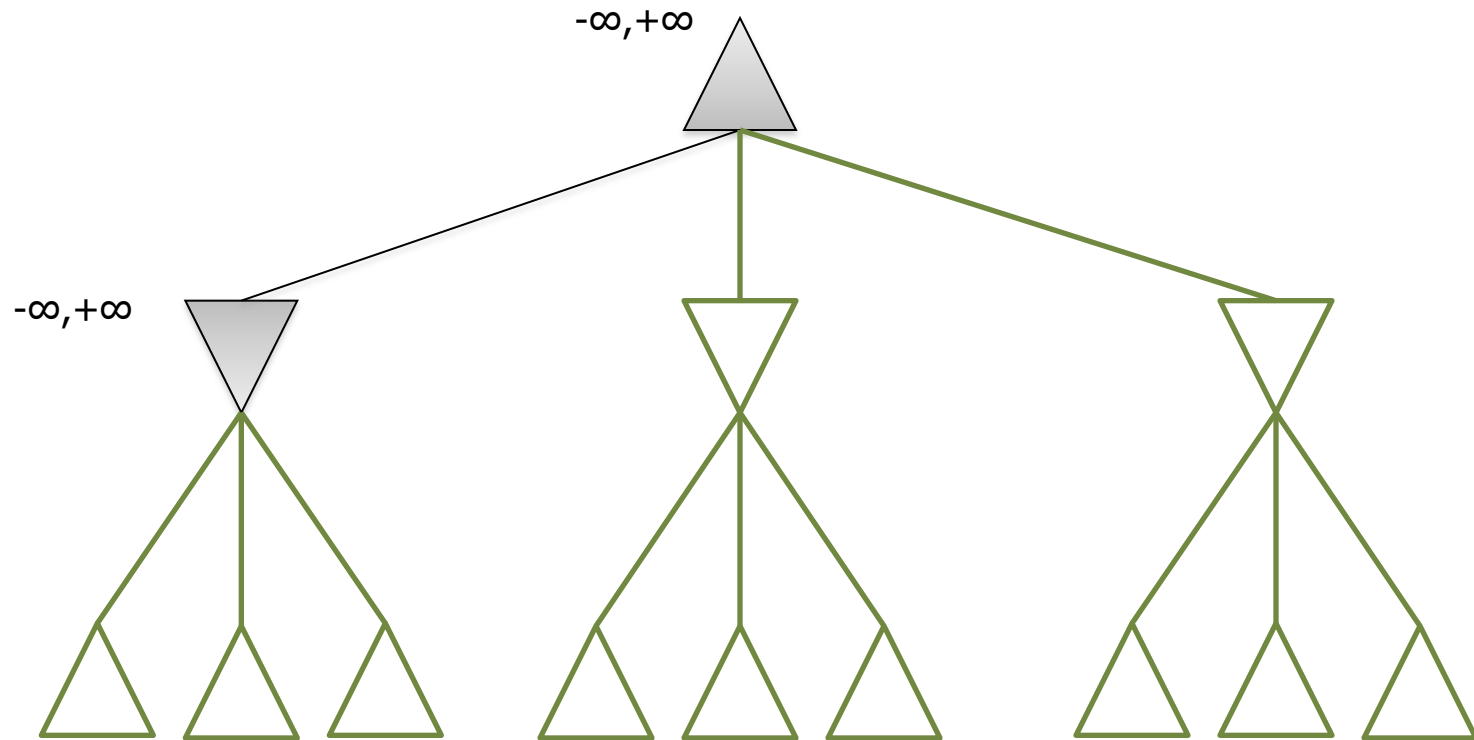
Para cada nó é apresentada a gama de valores possíveis



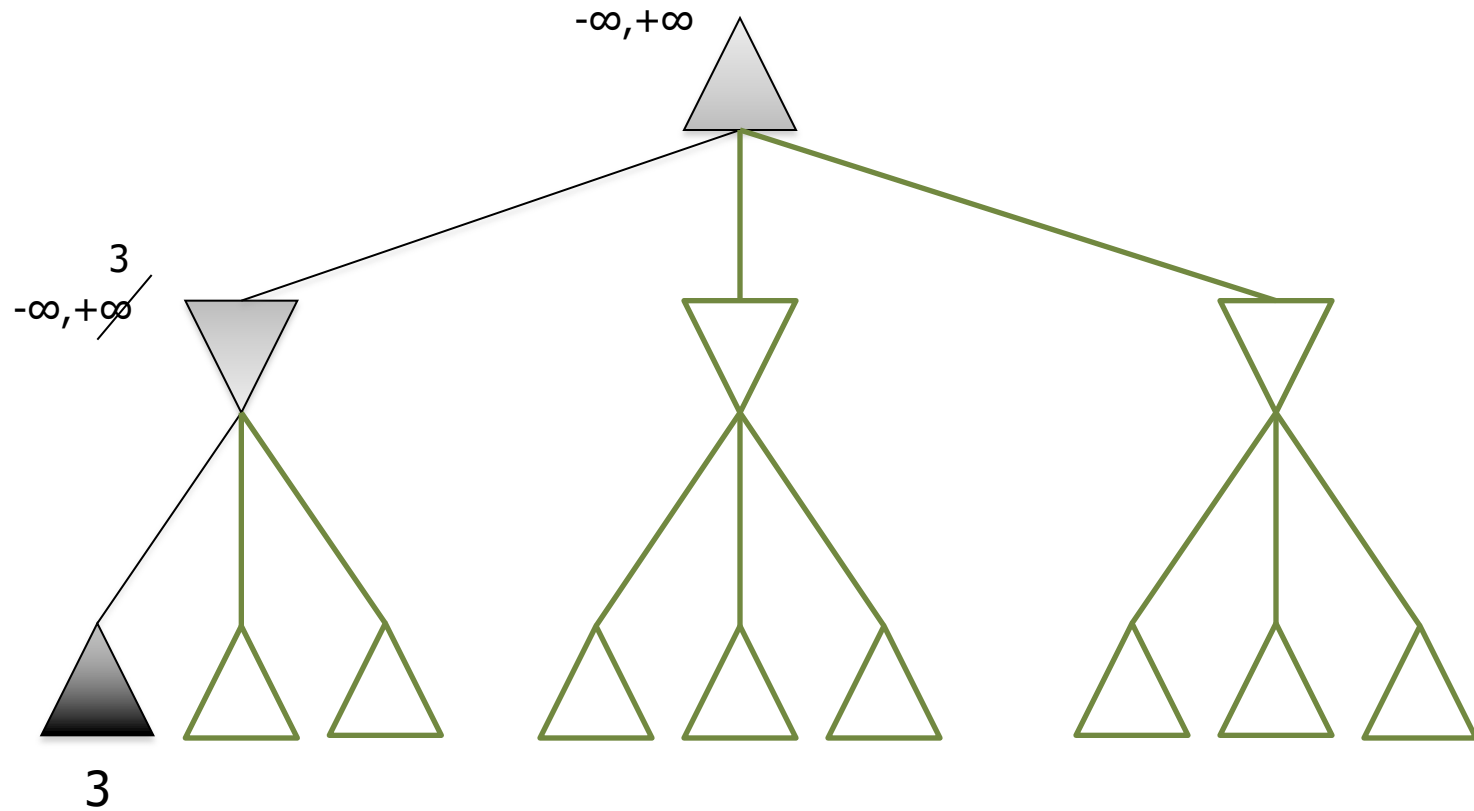
Cortes α - β : passo a passo



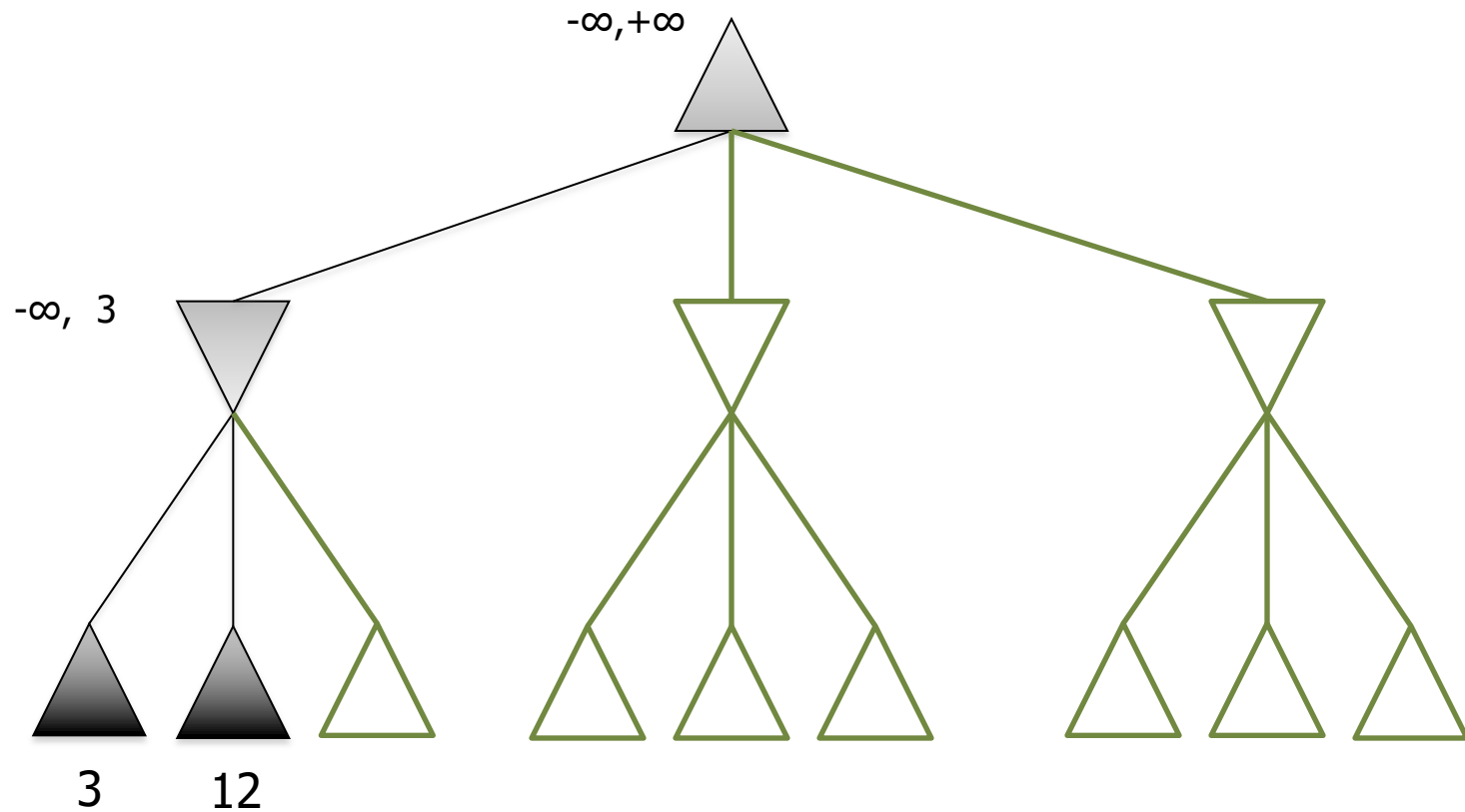
Cortes α - β : exemplo



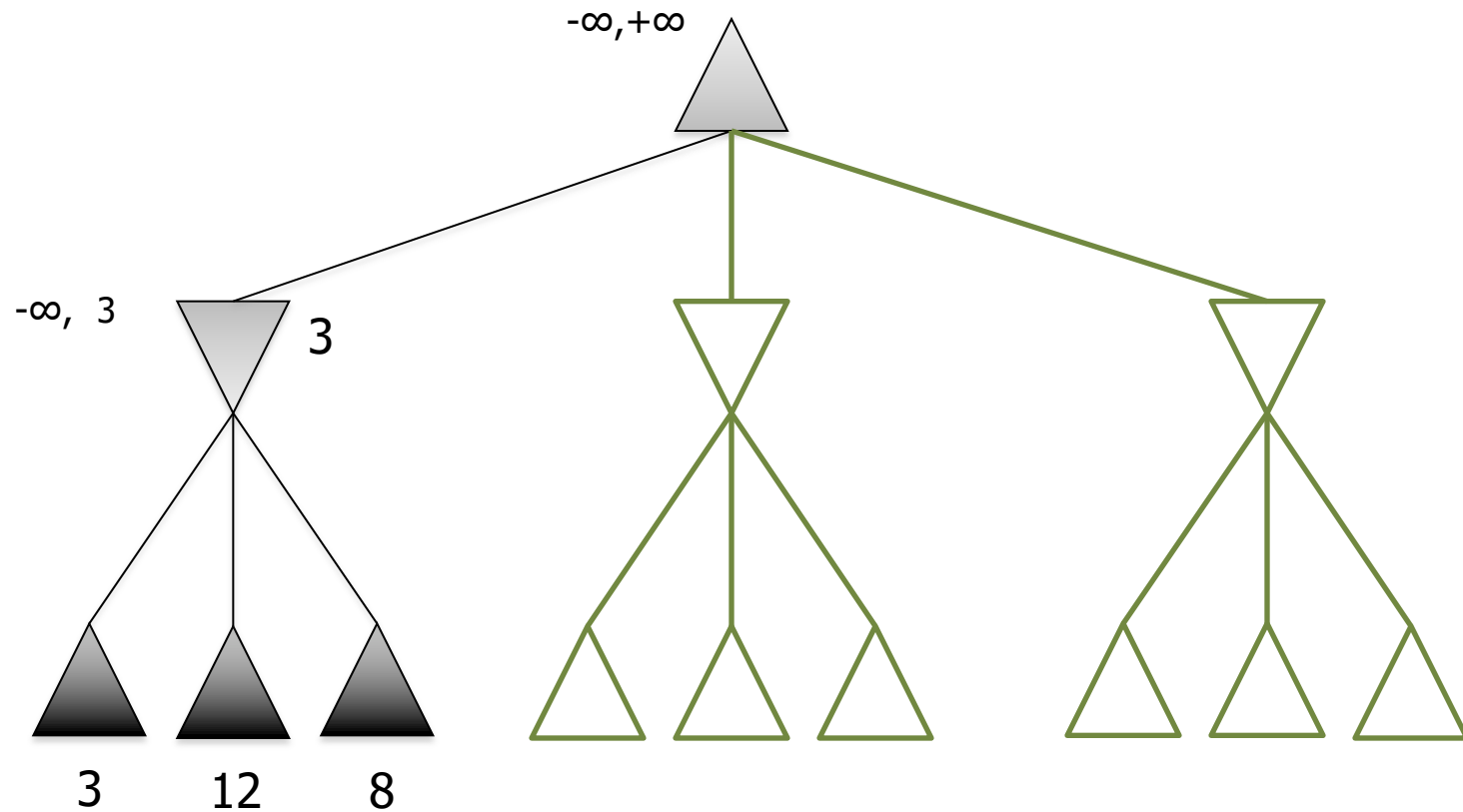
Cortes α - β : exemplo



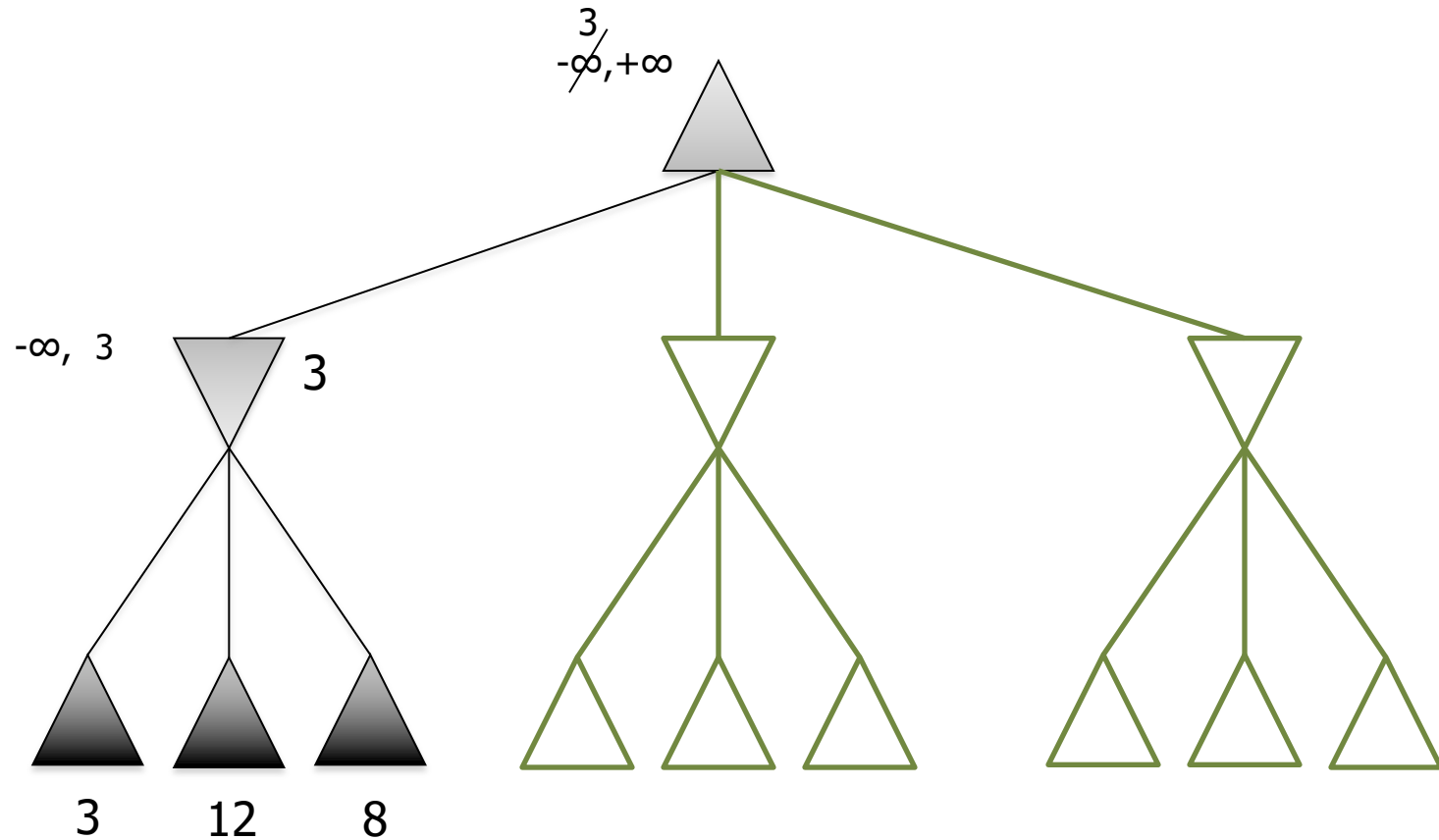
Cortes α - β : exemplo



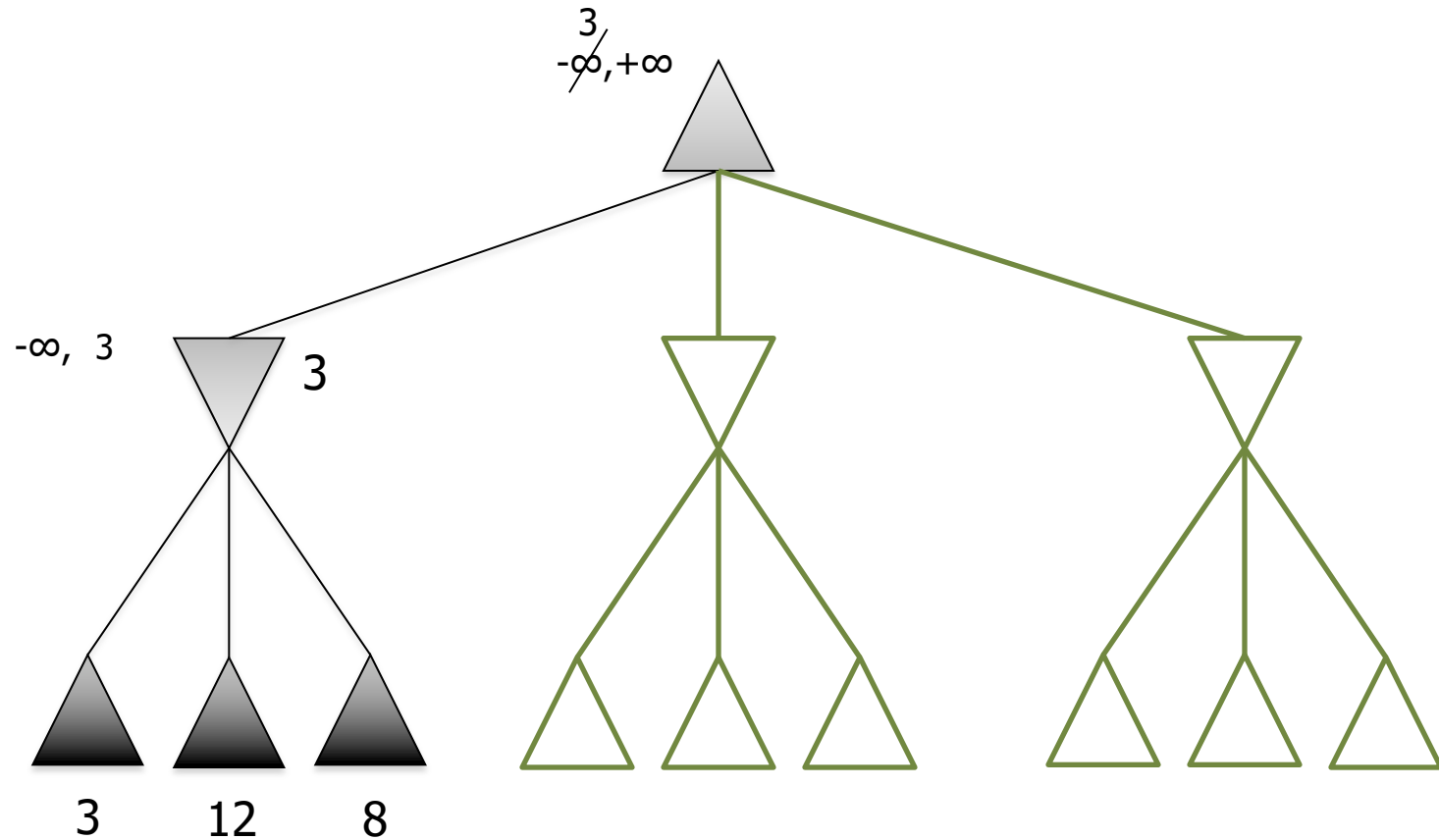
Cortes α - β : exemplo



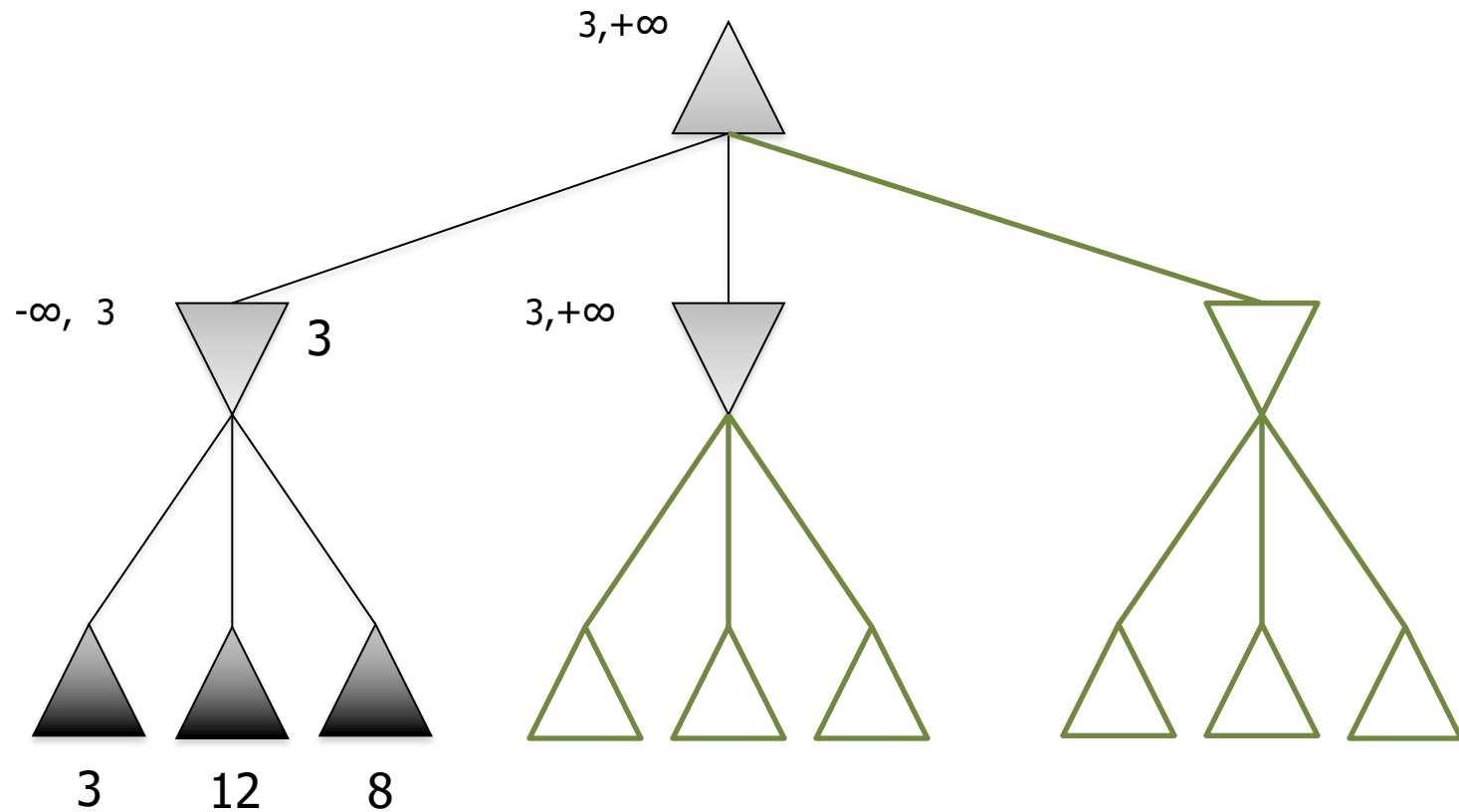
Cortes α - β : exemplo



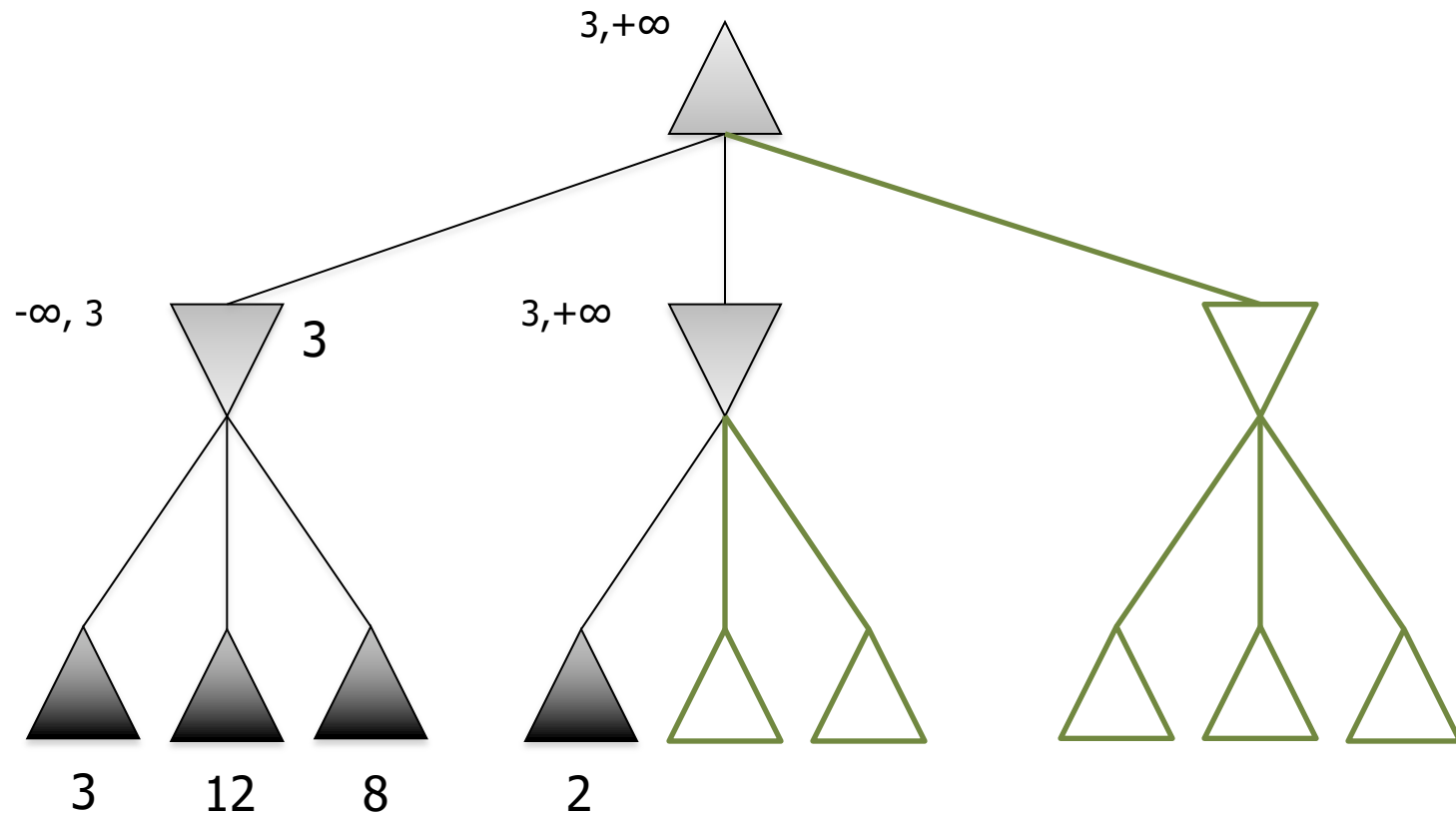
Cortes α - β : exemplo



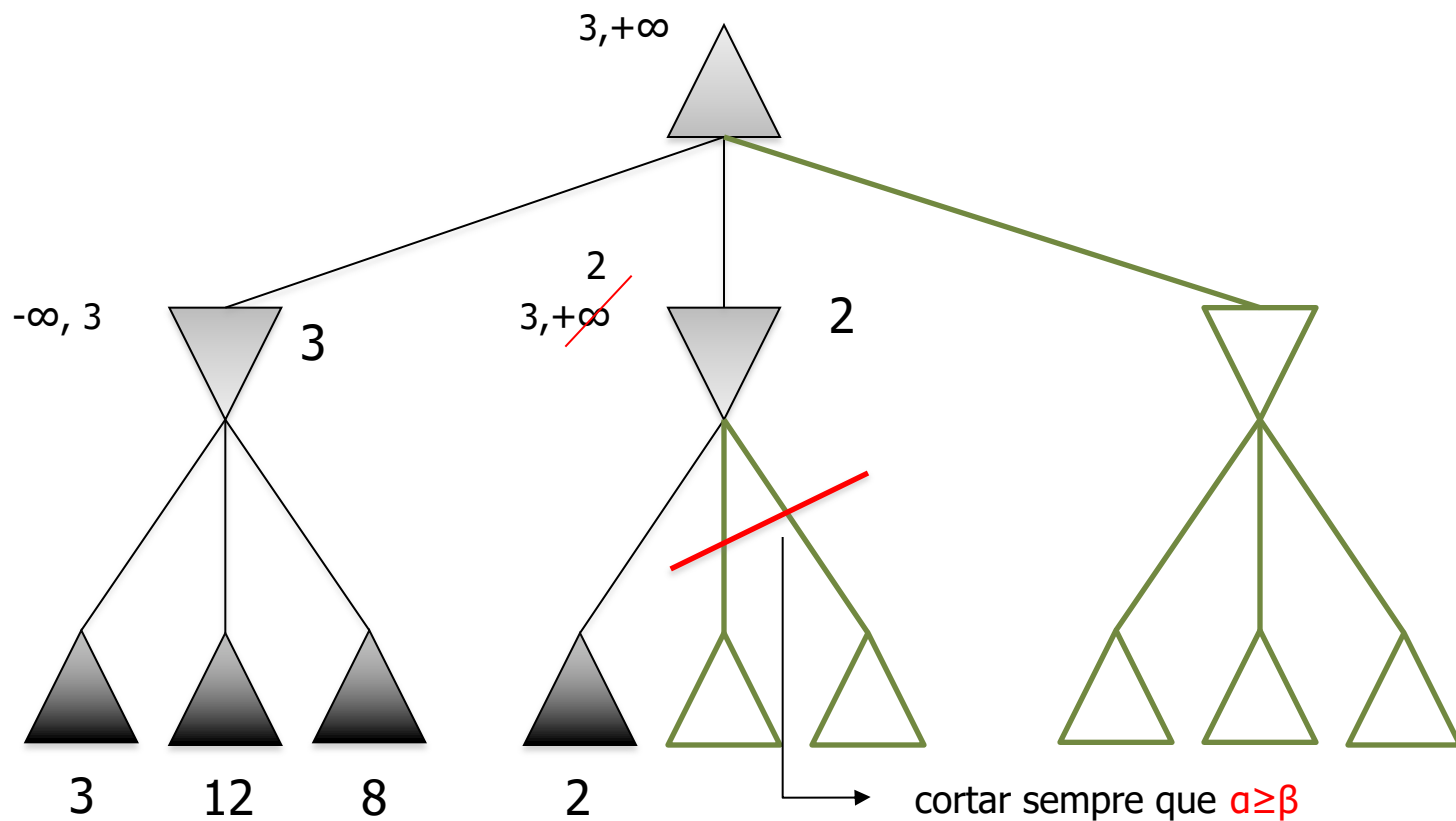
Cortes α - β : exemplo



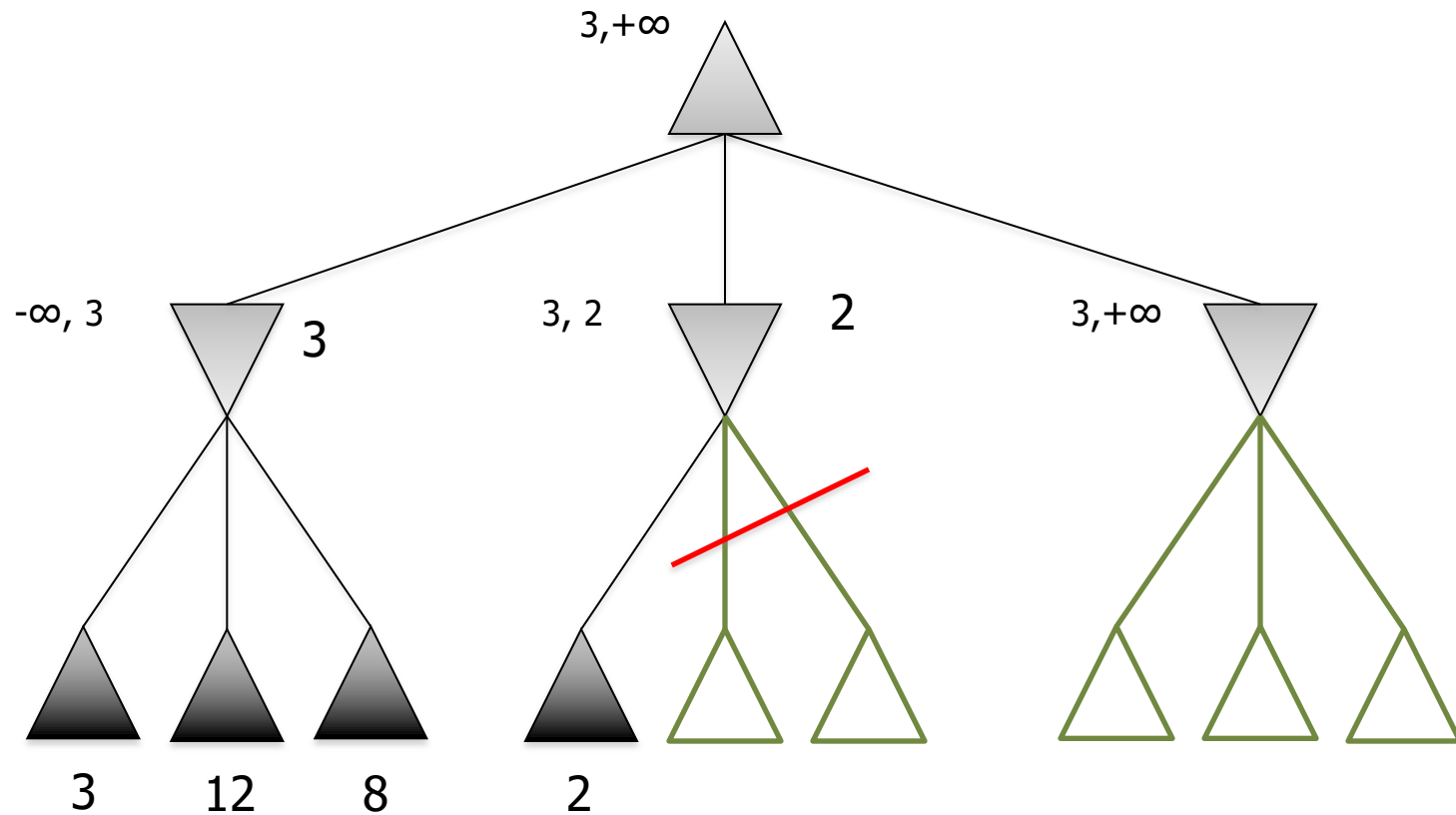
Cortes α - β : exemplo



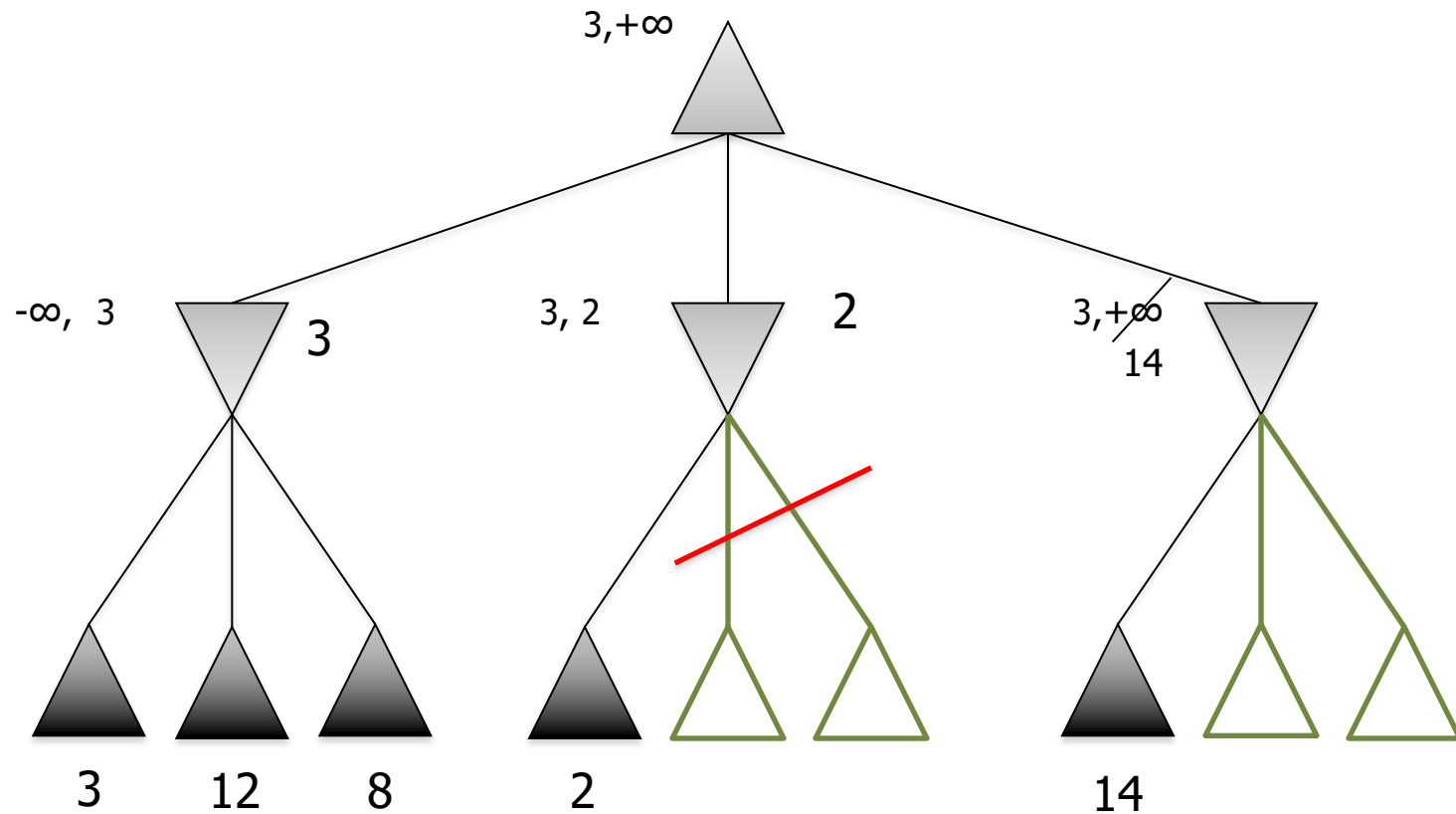
Cortes α - β : exemplo



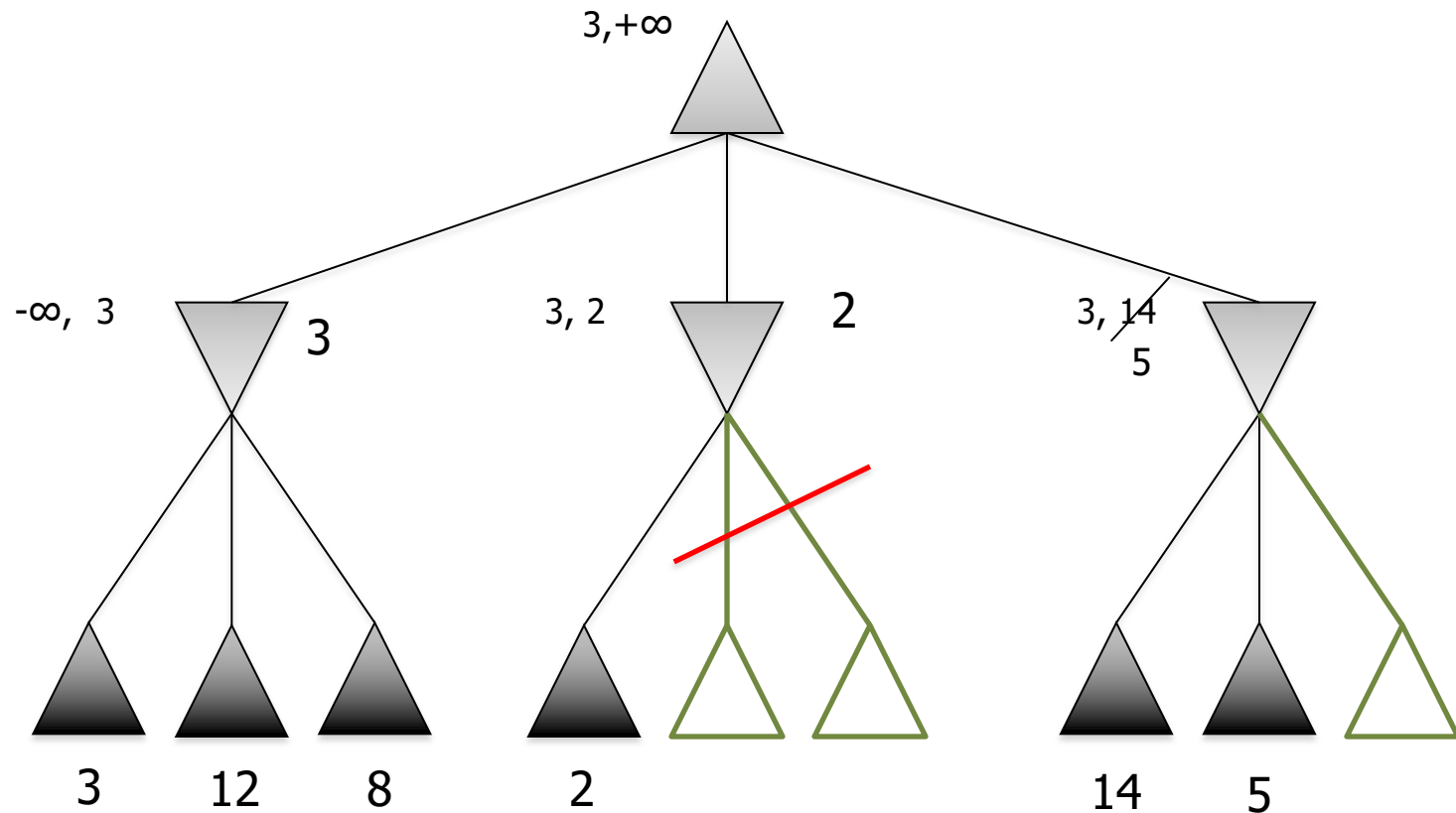
Cortes α - β : exemplo



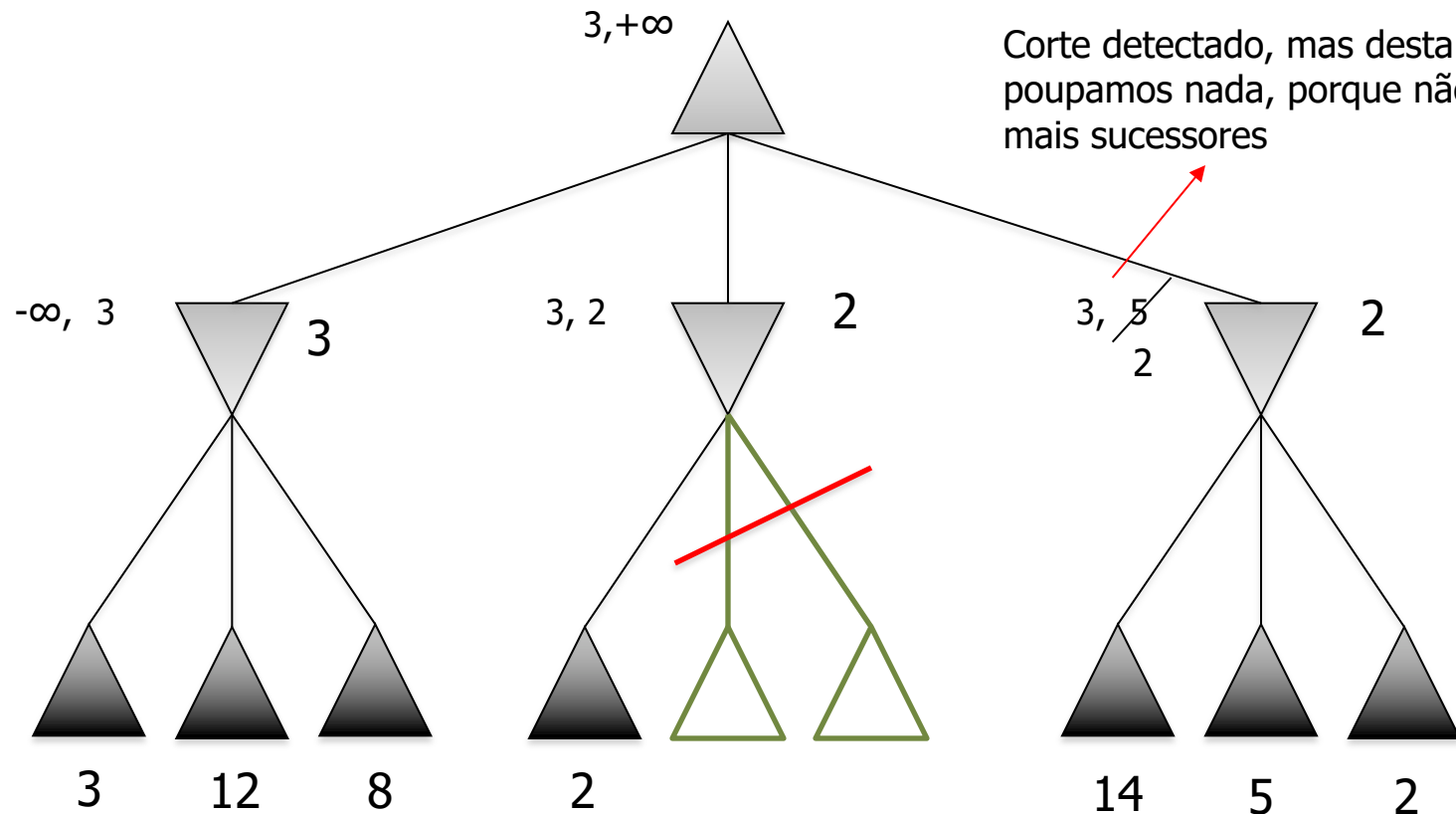
Cortes α - β : exemplo



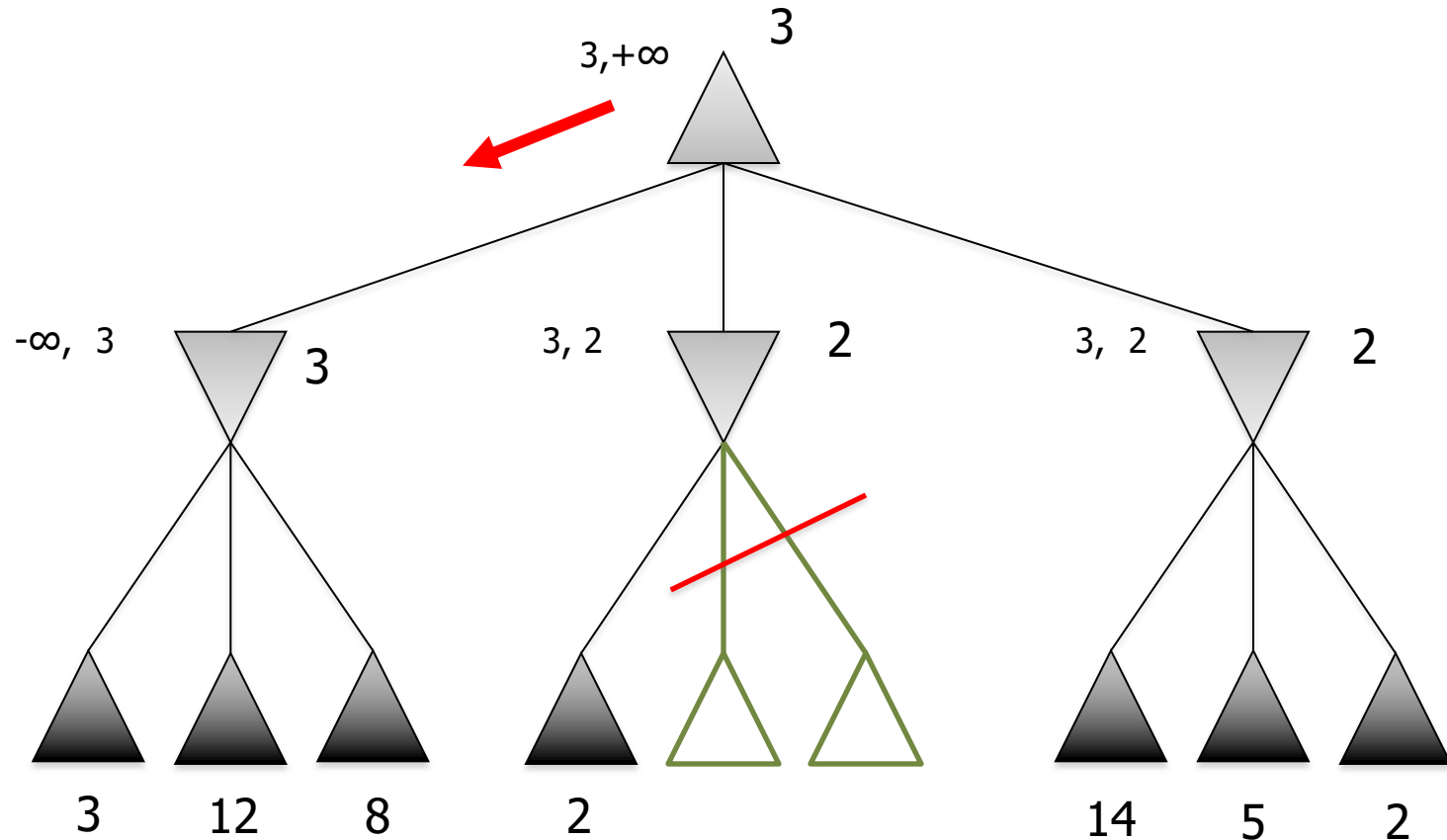
Cortes α - β : exemplo



Cortes α - β : exemplo



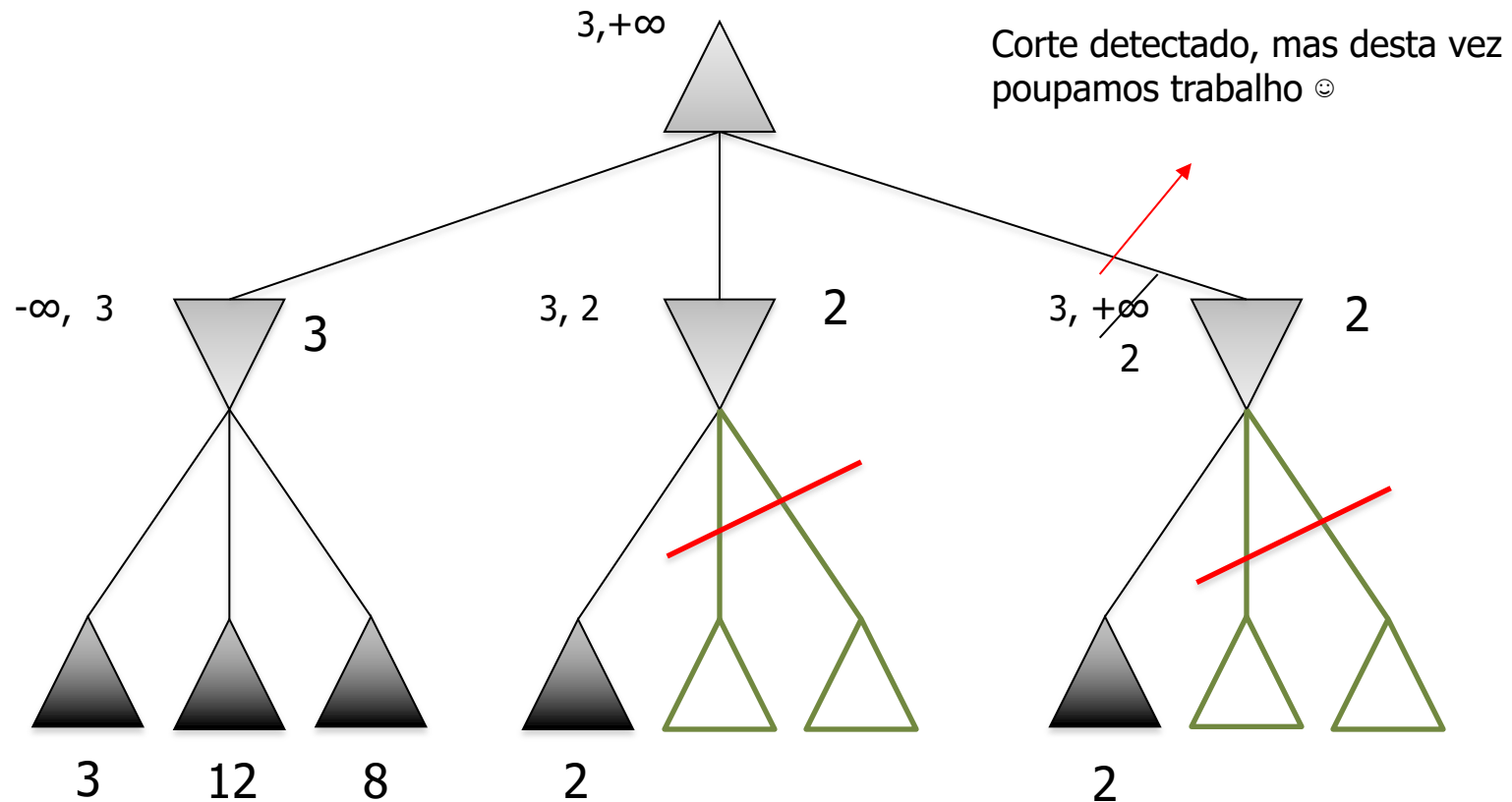
Cortes α - β : exemplo



Propriedades de α - β

- Cortes **não** afetam resultado final (= MINIMAX)
- Eficiência dos cortes depende da **ordenação** dos sucessores
 - Por exemplo, no caso anterior, se em vez do nó com valor 14 tivesse aparecido o nó com valor 2, não havia necessidade de gerar os outros nós
- Com uma “**ordenação perfeita**” a complexidade temporal fica reduzida a $O(b^{d/2})$
 - Nesta situação a procura α - β consegue atingir **o dobro da profundidade** da procura minimax no mesmo período de tempo

Cortes α - β : ordenação



Ordenação dos sucessores

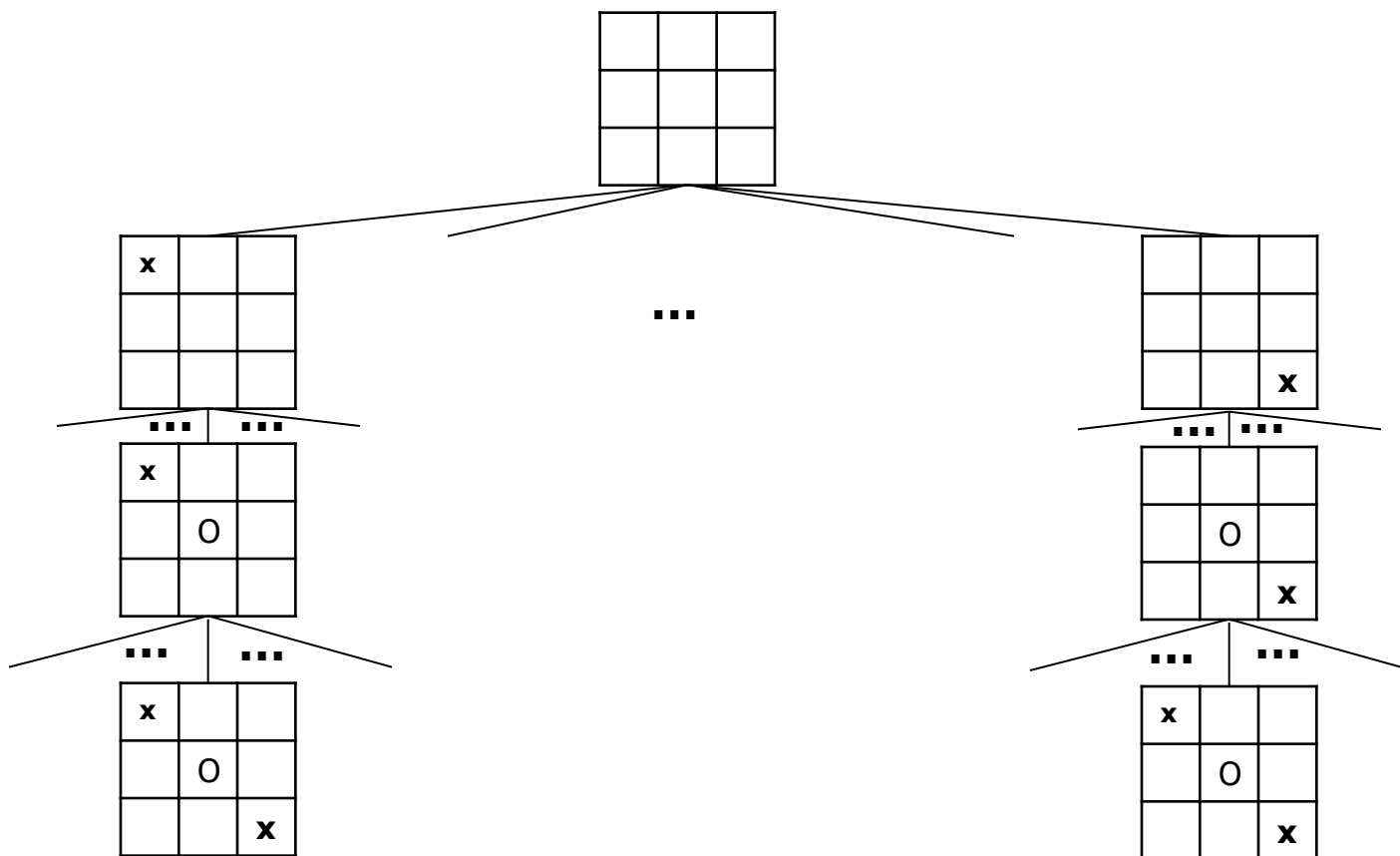
- Qual a melhor **ordenação** de modo a otimizar os cortes α - β ?
 - Cortes dependem do valor de α e β
 - Portanto vamos escolher primeiro os sucessores que atualizem α e β da maior maneira possível
 - **Nó Max: visitar primeiro o sucessor com maior valor minimax**
 - **Nó Min: visitar primeiro o sucessor com menor valor minimax**
- **Na realidade só interessa o primeiro sucessor visitado**
 - Ou existe corte logo no primeiro teste
 - Todos os restantes sucessores são cortados
 - Ou então não é possível haver corte

Ordenação dos sucessores

- Infelizmente, **não sabemos** o valor minimax de um nó antes de o visitar
- Então como escolher qual o melhor para visitar primeiro? (sem o visitar)
 - **Estimar o seu valor** com uma função de avaliação/heurística
 - Ou utilizar o seu valor minimax de iterações anteriores
 - Iterative Deepening :D
- **Killer Move heuristic**
 - Estratégia de visitar a melhor jogada primeiro
 - Melhor jogada – designada de killer move

Transposições

- Em jogos, ocorrem muitas vezes estados repetidos ou equivalentes (devido a permutações)



Transposições

- Pode valer a pena guardar o estado e o seu valor minimax calculado numa *hash table*
 - Guardado na 1.^a vez que é encontrado
 - Da próxima vez que o estado for encontrado em vez de calcular o seu valor
 - Usamos o valor guardado na hash table
- A esta tabela chamamos **tabela de transposição**
 - No entanto, se são avaliados milhões de nós por segundo não é viável guardar tantos nós numa tabela
 - existem várias estratégias para determinar quais os estados a guardar

Sumário

- Jogos: Conceitos Básicos
- Decisões ótimas em jogos
 - Estratégias ótimas e o Minimax
 - Estratégias ótimas com múltiplos jogadores
- Procura α - β
- Procura α - β heurística
- Procura Monte Carlo
- Jogos estocásticos
- Jogos parcialmente observáveis
- Limitações

Procura α - β heurística

Mesmo com cortes, a procura α - β tem de percorrer a árvore de jogo até à **profundidade máxima**

Decisões têm que ser tomadas em tempo real

- **não é possível analisar toda a árvore**
- é necessário cortar a árvore e acabar a procura mais cedo
- mas como saber qual o valor de utilidade de um nó sem chegar ao fim da árvore?

Procura α - β heurística

Função **EVAL** (IS-CUTOFF) substitui UTILITY (IS-TEST)

$$\text{H-MINIMAX}(s, d) = \begin{cases} \text{EVAL}(s, \text{MAX}) & \text{if IS-CUTOFF}(s, d) \\ \max_{a \in \text{Actions}(s)} \text{H-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{H-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if TO-MOVE}(s) = \text{MIN}. \end{cases}$$

Função heurística de avaliação (**EVAL**) devolve uma **estimativa** da utilidade do estado

Idealmente a **ordenação** resultante da função EVAL **é igual** à da função UTILITY

Funções de Avaliação

- O desempenho de um jogo depende da **qualidade da função de avaliação!**
- E como “escolher” uma boa função de avaliação?
 - Para nós terminais
 - deve ser capaz de ordenar os estados terminais do mesmo modo que a função de utilidade
 - Para nós não terminais
 - deve estar fortemente correlacionada com as hipóteses reais de ganhar
 - O seu cálculo não pode ser demorado!!!!

Funções de Avaliação

- Tipicamente são uma soma linear de **caraterísticas** do jogo (f), associadas a diferentes pesos (w)

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^n w_i f_i(s)$$

- Ex: Xadrez
 - rainha vale 9, bispo e cavaleiro 3, etc..
 - $w_1 = 9$ e $f_1(s) = n^\circ$ de rainhas brancas
 - $w_2 = 3$ e $f_2(s) = n^\circ$ de bispos
 - ...

Funções de Avaliação

- A soma de valores de diferentes características (features) é razoável embora seja discutível pois assume que as características são independentes umas das outras
- Atualmente a maioria dos programas usam combinações não lineares das “features”...

Cortar a procura

- Como cortar a procura para usar EVAL?
 - Tipicamente limite está na **profundidade**
- Função **teste-limite (cutoff-test)**
 - Recebe um estado e a profundidade desse estado
 - Decide se o estado é considerado final (mesmo não o sendo)
 - **Deve retornar T se o estado recebido for um estado terminal**

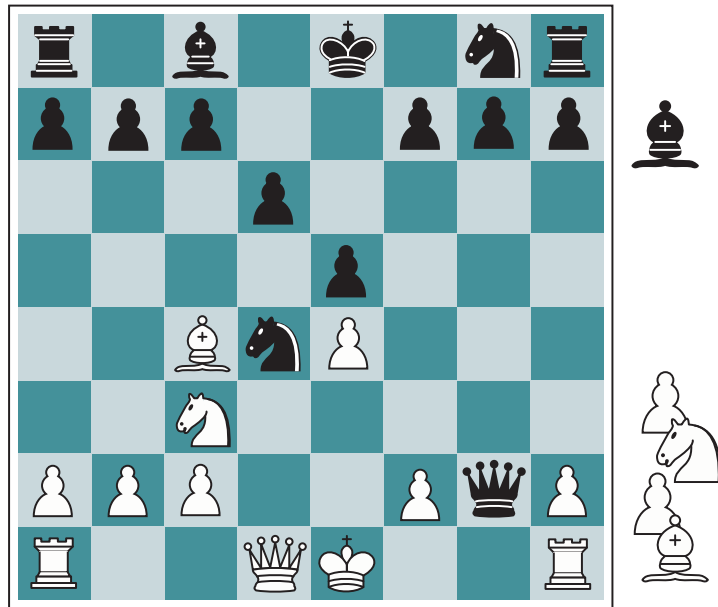
Caracterização de um Jogo

- Jogo passa a ser um problema de procura com:
 - Estado inicial
 - Jogador
 - Ações
 - Resultado
 - **Teste-limite (ou teste-corte)**
 - função que recebe um estado e e a profundidade d desse estado e determina se a procura deve parar nesse estado ou não
 - **Função-avaliação**
 - função que dado um estado terminal e e um jogador j retorna a **estimativa da utilidade esperada** a partir desse estado para o jogador (valor numérico)

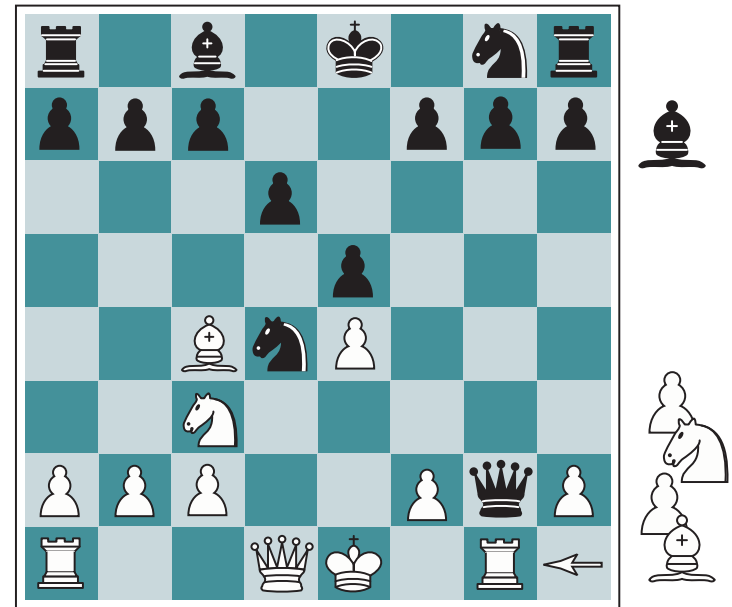
Teste Limite

- Problema da **aquiescência (estados inativos, parados...)**
 - Significado: *consentimento por condescendência*
 - Estados não aquiescentes no limite devem ser expandidos até que sejam gerados estados aquiescentes
 - por ex^o, um estado em que não foram feitas capturas
 - A esta procura adicional, chama-se procura aquiescente
 - Por vezes é aplicada apenas a algum tipo de jogadas
 - Quando rapidamente se pode perceber se são boas ou más
 - como capturas

Teste Limite



(a) White to move



(b) White to move

Figure 6.8 Two chess positions that differ only in the position of the rook at lower right. In (a), Black has an advantage of a knight and two pawns, which should be enough to win the game. In (b), White will capture the queen, giving it an advantage that should be strong enough to win.

Teste Limite

- Problema do efeito de horizonte
 - Procura com limite coloca eventuais problemas futuros para além do horizonte
 - ex: um movimento adversário que vai ter consequências desastrosas, mas que o jogador consegue adiar
 - O jogador consegue adiar, mas não evitar
 - A consequência é adiada para além do horizonte
 - Não se consegue perceber que esta é uma má jogada, por causa do limite

Efeito Horizonte

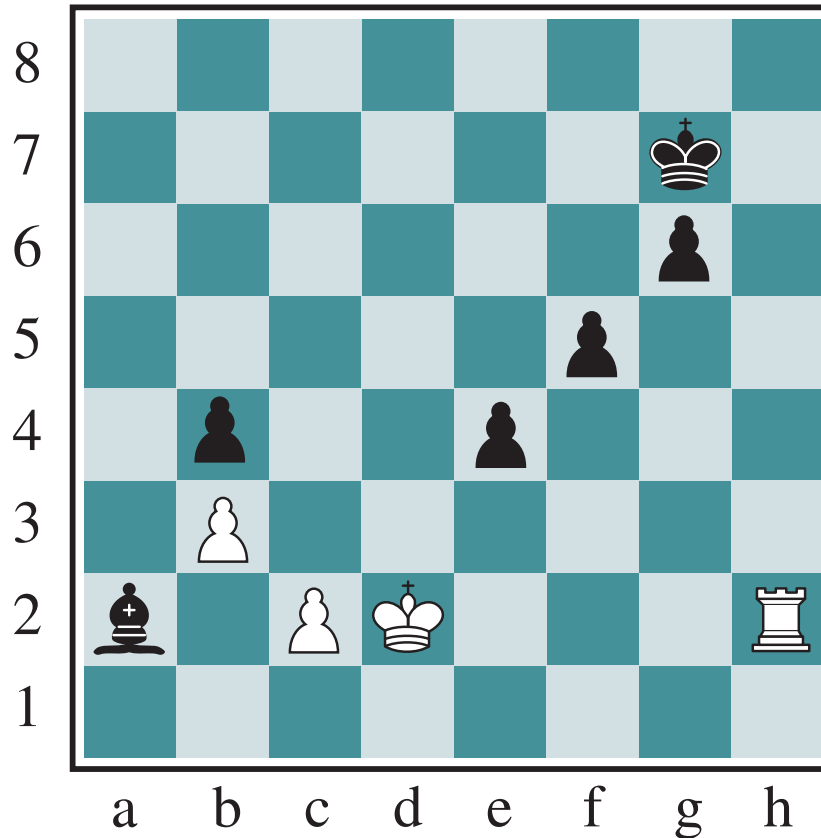


Figure 6.9 The horizon effect. With Black to move, the black bishop is surely doomed. But Black can forestall that event by checking the white king with its pawns, encouraging the king to capture the pawns. This pushes the inevitable loss of the bishop over the horizon, and thus the pawn sacrifices are seen by the search algorithm as good moves rather than bad ones.

Teste Limite

- Extensões singulares permitem mitigar o problema do efeito de horizonte
 - Quando é encontrada uma jogada considerada claramente melhor que as jogadas restantes para a mesma posição
 - Essa jogada é registada como jogada singular
 - Aumenta-se o limite de procura para os sucessores da jogada singular
- A profundidade da árvore aumenta...
 - Mas como existem poucas extensões singulares não são adicionados muitos nós à árvore

Cortes Progressivos

- Até agora vimos 2 tipos de corte
 - Cortes alfa-beta (não influenciam o resultado)
 - Cortes limite
- Existe um 3.º tipo de corte
 - Cortes progressivos
 - Cortar imediatamente algumas jogadas por cada posição analisada

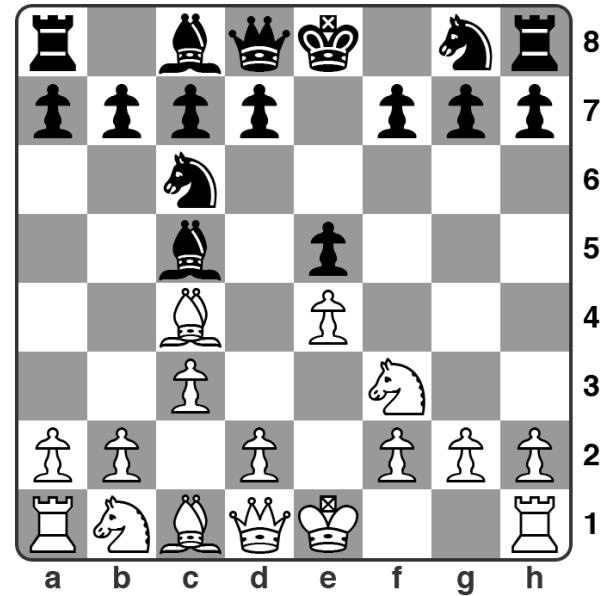
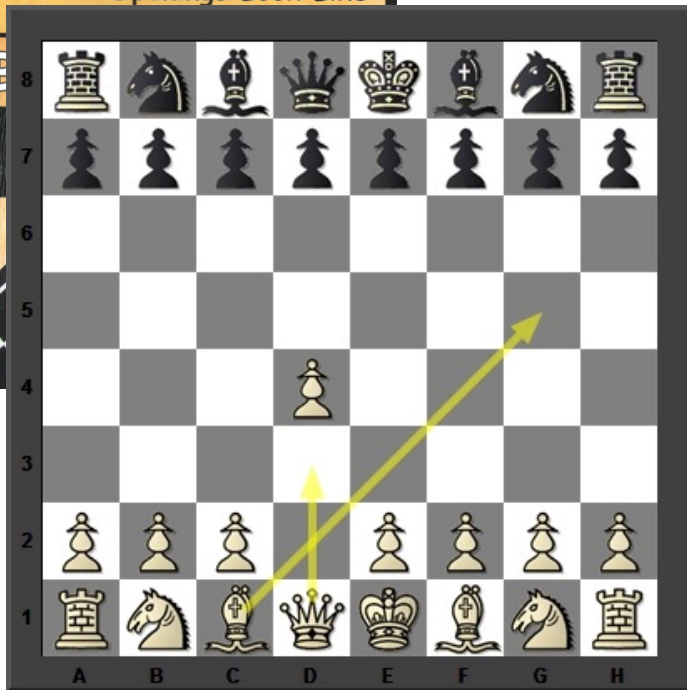
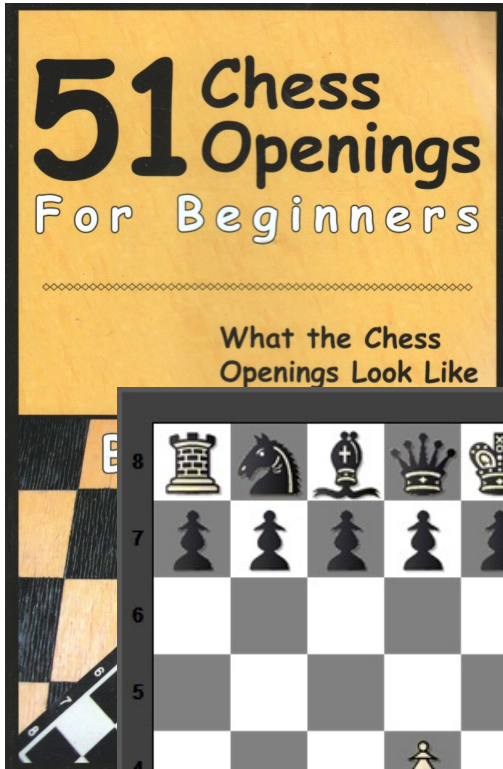
Cortes Progressivos

- Procura em banda (**beam search**)
 - Para cada posição considerar as n melhores jogadas (como os humanos fazem!)
 - Em vez de considerar todas as jogadas
 - Qualidade da jogada estimada com a função de avaliação
 - Perigoso:
 - Nada nos garante que a melhor jogada real não seja imediatamente cortada

Cortes Progressivos

- ProbCut (probabilistic cut)
 - Tenta diminuir a probabilidade da melhor jogada real ser cortada
 - A procura alfa-beta corta nós que temos a certeza de ter valor fora da janela (α - β)
 - ProbCut corta nós que **provavelmente** têm valor fora da janela (α - β)
 - Probabilidade calculada fazendo uma procura pouco profunda para calcular o valor minimax v de um nó
 - Usa-se experiência de procuras anteriores para determinar a probabilidade de um valor v à profundidade p estar fora da janela (α - β)
 - Se a probabilidade for maior que um limite, o nó é cortado

Aberturas e finais



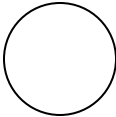
Tabelas de aberturas e finais

- Muitos programas para jogar jogos usam **tabelas de aberturas e finais**
- Em vez de usar procura para determinar melhor jogada
 - Usa-se tabela com melhores jogadas iniciais
 - Obtidas por conhecimento de perito
 - Ou por estatísticas
 - Quais as jogadas iniciais que levam mais vezes a vitórias?
 - O mesmo para o fim do jogo
 - Tabelas de estados perto do fim, a partir dos quais a jogada ótima está memorizada

Sumário

- Jogos: Conceitos Básicos
- Decisões ótimas em jogos
 - Estratégias ótimas e o Minimax
 - Estratégias ótimas com múltiplos jogadores
- Procura α - β
- Procura α - β heurística
- Procura Monte Carlo
- **Jogos estocásticos**
- Jogos parcialmente observáveis
- Limitações

Jogos Estocásticos

- Árvore de jogo com nós sorte  para além dos nós MIN e MAX
- Cada ramo de um nó sorte representa um resultado diferente nos dados
 - está associado a uma probabilidade

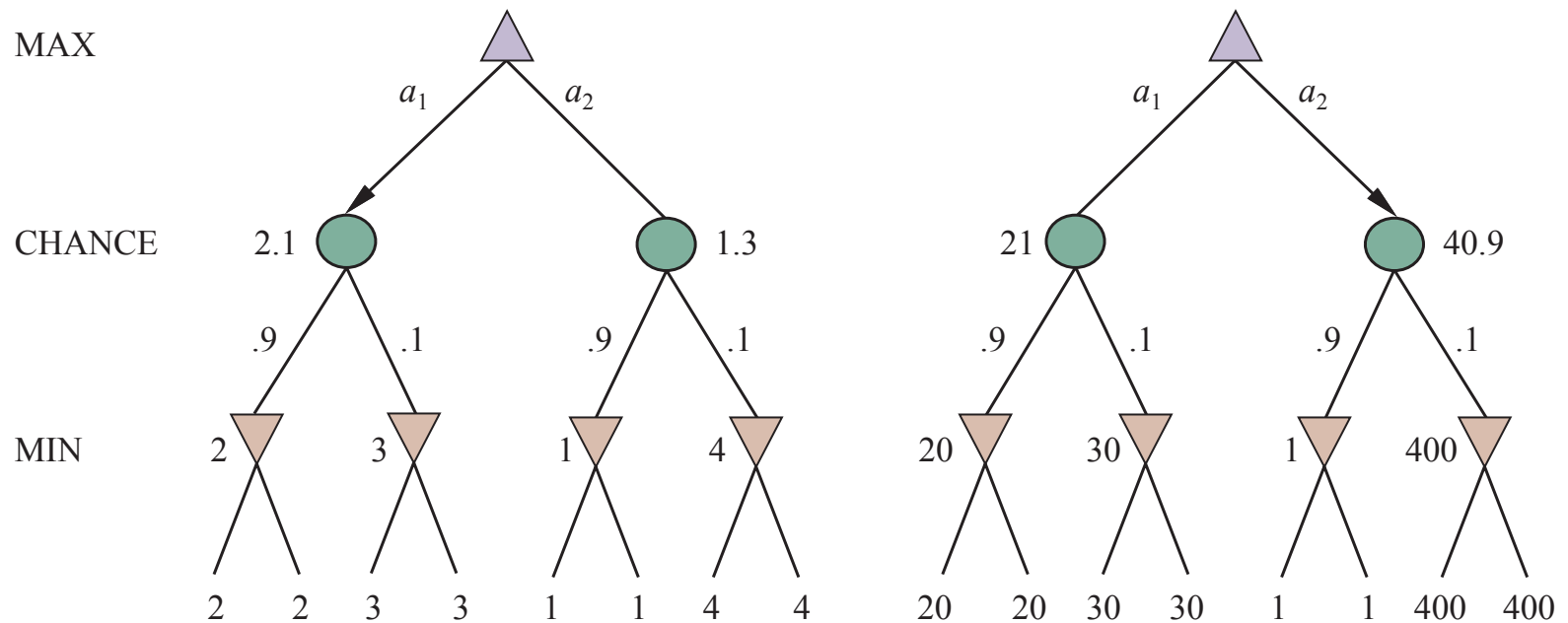
EXPECTIMINIMAX(s) =

$$\begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \\ \sum_r P(r) \text{EXPECTIMINIMAX}(\text{RESULT}(s, r)) & \text{if TO-MOVE}(s) = \text{CHANCE} \end{cases}$$

Jogos Estocásticos

- Funções de avaliação para jogos estocásticos
 - É preciso ter um cuidado adicional
 - ordenação resultante da função de avaliação deve ser igual à da função de utilidade
 - Em jogos estocásticos
 - isto não é suficiente

Jogos Estocásticos: exemplo



- Atenção: alteração de valores das folhas mantendo a mesma ordem relativa de valores resulta em decisões diferentes.

Jogos Estocásticos

- Para lidar com o problema
 - Função de avaliação deve ser:
 - Transformação linear positiva
 - Da probabilidade de vencer a partir de uma posição
 - Ou da utilidade esperada da posição

Jogos Estocásticos

- Complexidade Temporal
 - Se não tivéssemos lançamento de dados
 - $O(b^m)$
 - Com m = máxima profundidade
 - Mas se considerarmos lançamento de dados
 - $O(b^m n^m)$
 - Com n = nº de lançamentos distintos de dados
 - Ou seja, é um problema muito mais difícil
 - Em jogos estocásticos não se consegue normalmente atingir uma profundidade elevada

Jogos Estocásticos

- É possível usar cortes alfa-beta
 - Cortes nós max e min processam-se da mesma maneira
 - Cortes nó sorte
 - Calcular limites (máximo e mínimo) para valor do nó
 - Usar limites para decidir corte
 - Alternativamente
 - Avaliar uma posição usando simulação de Monte Carlo
 - Correr o algoritmo para jogar milhares de jogos a partir do estado inicial contra si próprio utilizando valores aleatórios para lançamentos de dados
 - A percentagem de vitórias de uma posição é uma boa estimativa do valor da posição
 - Usar valor estimado para decidir cortes

Vídeos e Ferramentas

- <https://www.youtube.com/watch?v=l-hh51ncgDI>
- <http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html>