Busca competitiva

Inteligência Artificial

Profª. Solange O. Rezende

Até aqui...

- Problemas sem interação com outro agente;
- O agente possui total controle sobre suas ações e sobre o efeito de suas ações;
- Muitas vezes encontrar a solução ótima é factível.

Diferentemente da busca tradicional vista até agora, na qual a situação não troca durante a busca, a **busca com adversários** considera que há oponentes hostis em sua trajetória.

Jogos são o exemplo clássico.

IA e Jogos com adversário

- Em IA, jogos com adversários são de um tipo específico
  - exigem a tomada de decisões (muitas vezes em um curto intervalo de tempo)
  - Há interação e pode haver não determinismo
  - Consideram a alternância entre **dois jogadores** (turnos)
  - **Jogos determinísticos**: cada ação leva a um resultado conhecido
  - **Soma zero**: o ganho de um jogador é a perda do outro
  - **Informações perfeitas**: todos os jogadores conhecem o estado atual do jogo

Jogos vs. busca

- O oponente é “imprevisível”
  - levar em consideração todos os movimentos possíveis de oponente;
- Limite de tempo
  - tomar uma decisão, mesmo que não seja ótima.

Jogos

- Exemplo: **Xadrez**
- É complicado demais para ser resolvido exatamente:
  - Fator de ramificação médio de 35
  - Duram em média 50 movimentos por jogador
  - Árvore de busca tem tipicamente 35^{100} ou 10^{154} nós
  - Grafo possui 10^{40} nós distintos
- Por comparação:
  - 10^{154} é mais do que a quantidade de átomos do universo
  - 200 milhões posições/segundo, mais de 10^{100} anos de processamento
  - Universo tem 10^{10} anos

Jogos

- Deep Blue
  - Venceu Garry Kasparov, melhor enxadrista do mundo em 1997
  - Deep Blue possui inteligência para xadrez?
  - Deep Blue possui melhor inteligência para o xadrez que Kasparov?
  - **Como** Deep Blue conseguia **tomar decisões tão boas, em tempo real, em um espaço de busca tão grande**?

Garry Kasparov x IBM Deep Blue
Foto cortesia da IBM.

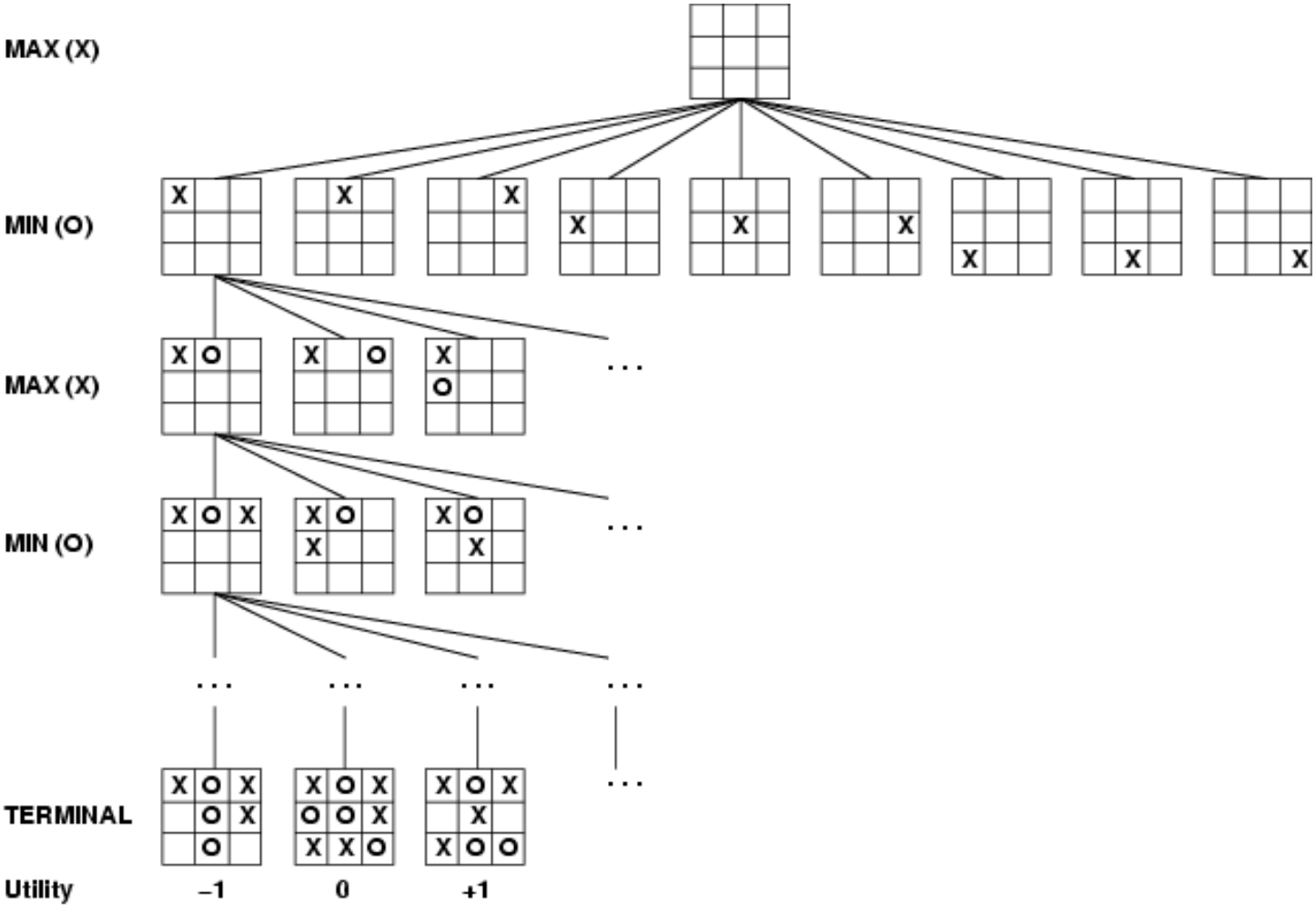
Problemas de busca em jogos com adversários

- **Tamanho** (espaço de busca) + **limitação de tempo**
- Adversário é imprevisível
  - Cada agente tem que levar em consideração todos os movimentos possíveis de oponente e ter um plano de contingência para eles
- Restrição sobre recursos
  - Difícil encontrar a meta
  - O agente tem que tomar uma decisão, mesmo que não seja ótima (decisão aproximada)

Busca em jogos com adversários

- Inicialmente jogos com dois jogadores:
  - 'MAX' e 'MIN';
- Um jogo pode ser definido como uma árvore de busca:
  - **Estado inicial**: posição do tabuleiro e identificar o jogador que fará o movimento
  - **Função sucessor**: indica um movimento válido e o estado resultante
  - **Teste de término**: verifica se o jogo terminou
  - **Função de utilidade**: dá um valor numérico aos estados terminais
    - Ex: Xadrez: (1, -1 ou 0); Gamão: [-192 a 192]

Árvore de jogo (2 jogadores)



Do ponto de vista de max, valores altos de utilidade são bons.
Jogo da velha: 9! = 362.880 nós terminais

Problemas de busca em jogos com adversários

- **Soma zero**: uma única **função de utilidade** é utilizada para determinar a qualidade de uma posição para ambos os jogadores
  - Um jogador tentará **MAXIMIZAR** a função de utilidade
  - O oponente tentará **MINIMIZAR** a função de utilidade

Minimax

- 1944 - John von Neumann propõe um **método de busca (Minimax) para jogos de soma zero** que maximiza a sua posição enquanto minimiza a de seu oponente.
- Para implementar esse método necessitamos de **medir**, de alguma maneira, o quanto é boa a nossa posição. Usamos para isso a **função de utilidade** (**utility function**).
- Inicialmente, será um valor que descreve exatamente a nossa posição**.**

Como jogar?

- **Uma maneira de jogar consiste em:**
  - **Considerar todos os movimentos** legais que podem ser realizados
  - **Computar a nova posição** resultante de cada movimento legal
  - Avaliar cada posição resultante, **decidir e executar** o melhor movimento
  - **Esperar pelo movimento** do **oponente** e repetir o processo
- MAX precisa de uma estratégia que especifica:
  - Movimento inicial (assumimos que MAX inicia o jogo)
  - Movimentos possíveis de MAX para cada movimento de MIN
- Por sua vez MIN precisa de:
  - Movimentos possíveis para cada possível movimento de MAX

Algoritmo Minimax

1 **Gerar a árvore do jogo**

2 **Calcular a função de utilidade de cada estado terminal**

3 **Propagar a utilidade** dos nós terminais para níveis superiores:

- se no nível superior é a vez de MIN jogar, escolher o menor valor
- se no nível superior é a vez de MAX jogar, escolher o maior valor

4 **No nó raiz, MAX escolhe o movimento que leva ao maior valor** (decisão Minimax)

Estratégias ótimas

- A solução ótima para MAX depende dos movimentos de MIN, logo:
  - **MAX deve encontrar uma *estratégia de contingência*** que especifique **o movimento de MAX no estado inicial**, e depois o movimento de MAX nos estados resultantes de cada movimento de MIN e assim por diante...

Estratégias ótimas

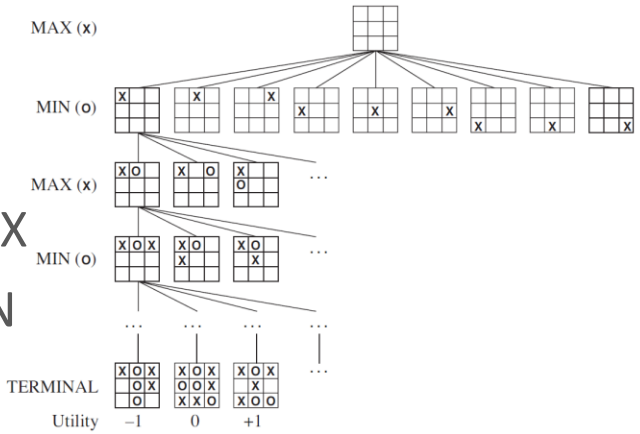
- Dada uma árvore de jogo, a estratégia ótima pode ser determinada a partir do valor ***minimax*** de cada nó.
- O valor **minimax** (para MAX) é a utilidade de MAX para cada estado, ***assumindo que MIN escolhe os estados mais vantajosos*** para ele mesmo (i.e. os estado com menor valor utilidade para MAX).

VALOR-MINIMAX(n)=

UTILIDADE(n) se n é terminal

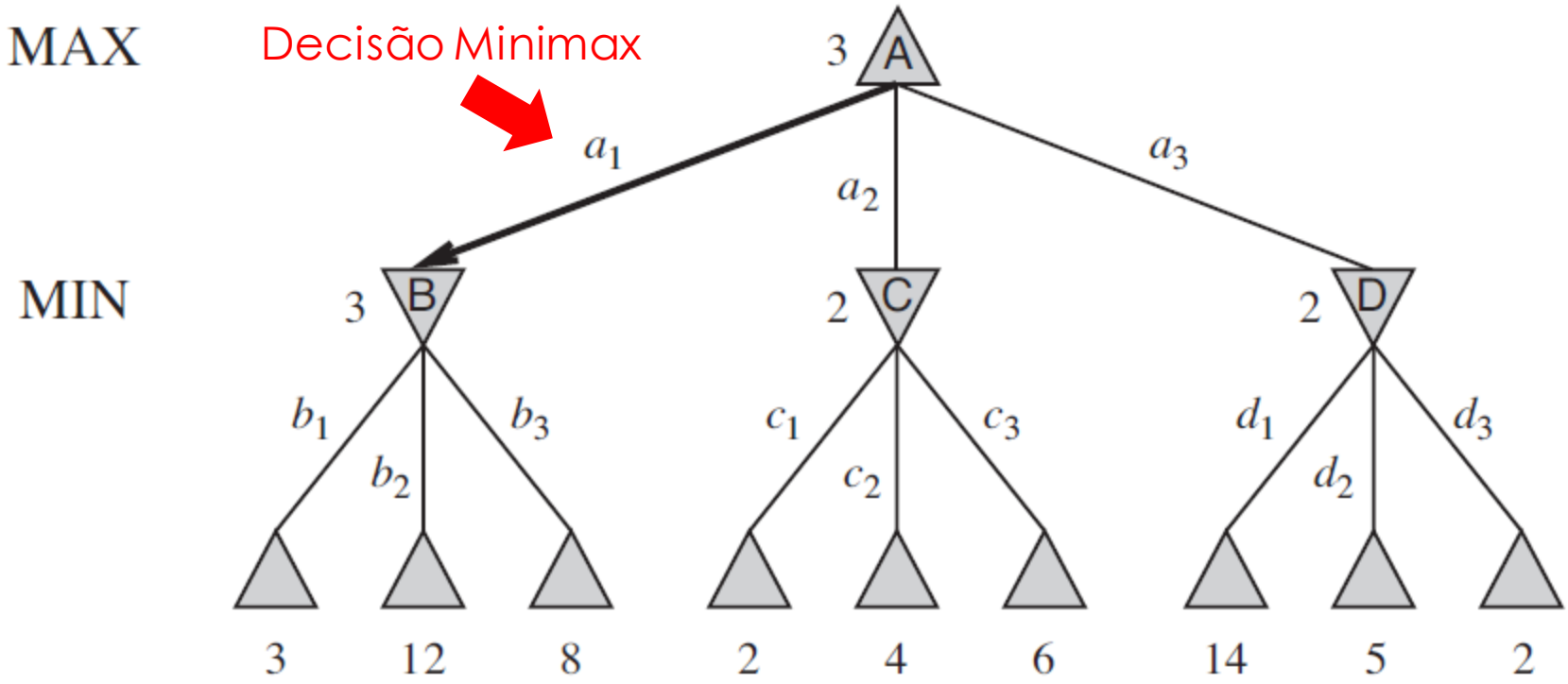
$\mathbf{max}_{x \in Succ(n)}$ Valor Minimax(x) se n é um nó de MAX

$\mathbf{min}_{x \in Succ(n)}$ Valor Minimax(x) se n é um nó de MIN



Árvore de Jogo

Vamos utilizar como exemplo para a melhor jogada para um jogo determinístico simples assumindo o melhor oponente.



A ação ***a1*** é a **escolha ótima para MAX**, porque leva ao sucessor com mais alto valor Minimax.

Valor Minimax

- Valor Minimax de cada nó assume que ambos os jogadores jogam de forma ótima
- Mas assumir que MIN joga otimamente é uma boa estratégia?
  - É uma análise de **pior caso**
  - Logo se MIN não joga otimamente, MAX vai ter resultados ainda melhores
  - Outras estratégias contra oponentes sub-ótimos pode dar melhores resultados
  - Mas, essas estratégias desempenham pior contra oponentes ótimos

Algoritmo Minimax (informal)

1. **Gerar a árvore** do jogo
2. **Calcular** a **função de utilidade** de cada estado terminal
3. **Propagar a utilidade** dos nós terminais para níveis superiores:
   - **se** no nível superior **é a vez de MIN jogar**, escolher o **menor valor**
   - **se** no nível superior **é a vez de MAX jogar**, escolher o **maior valor**
4. **No nodo raiz**, **MAX escolhe** o movimento que leva ao **maior valor** (decisão Minimax)

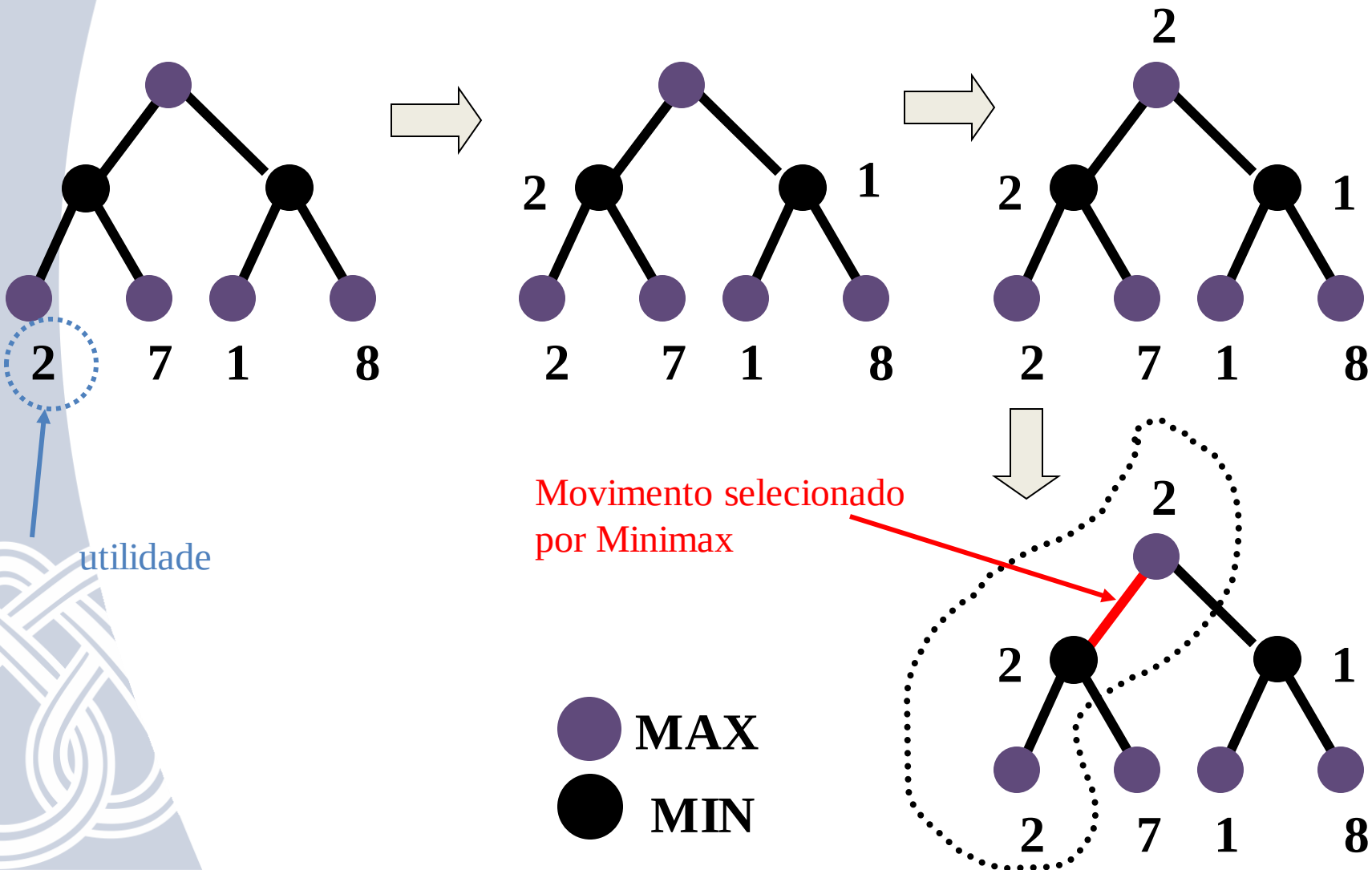
Algoritmo Minimax

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))
```

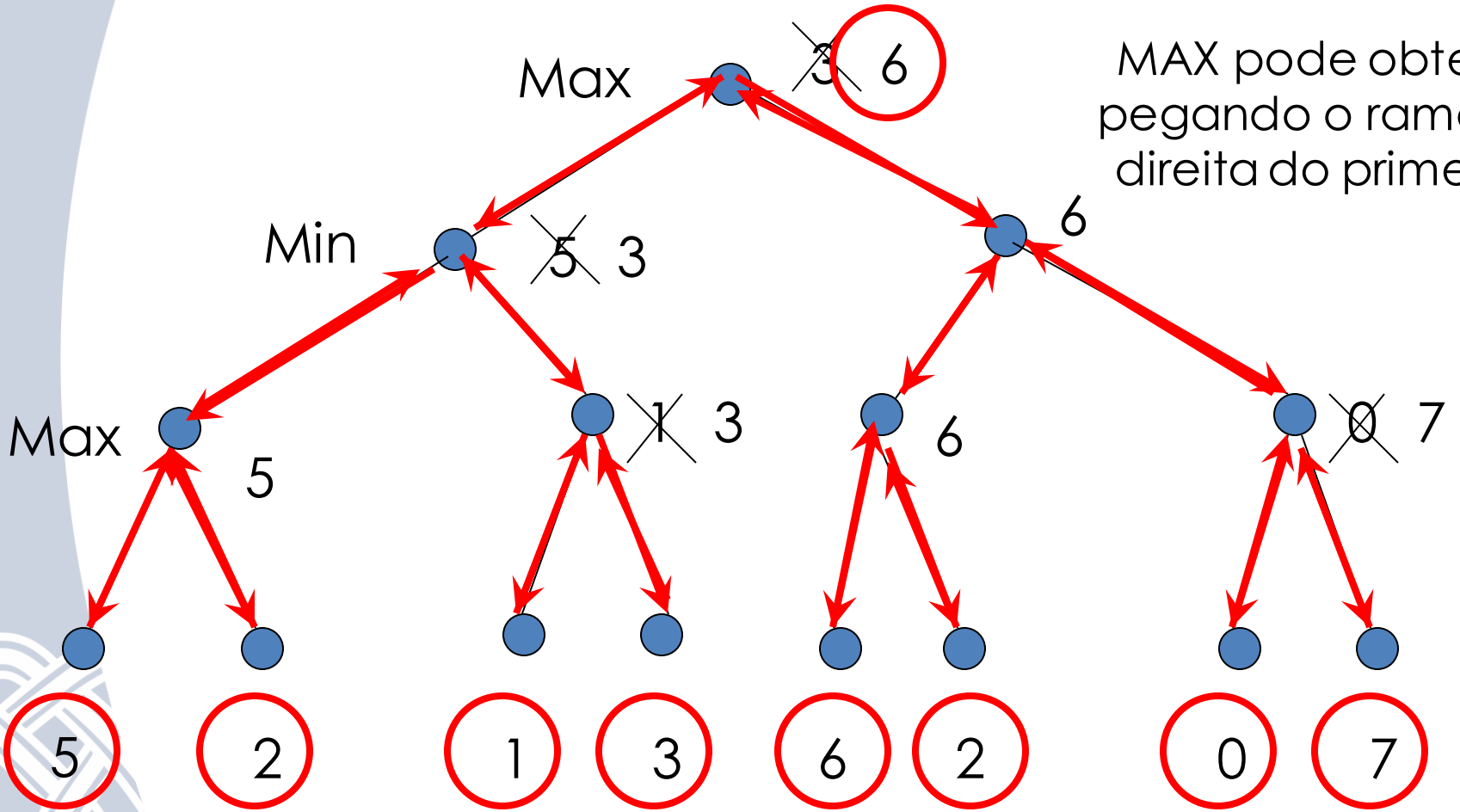
```
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v
```

```
function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

Algoritmo Minimax



Algoritmo Minimax

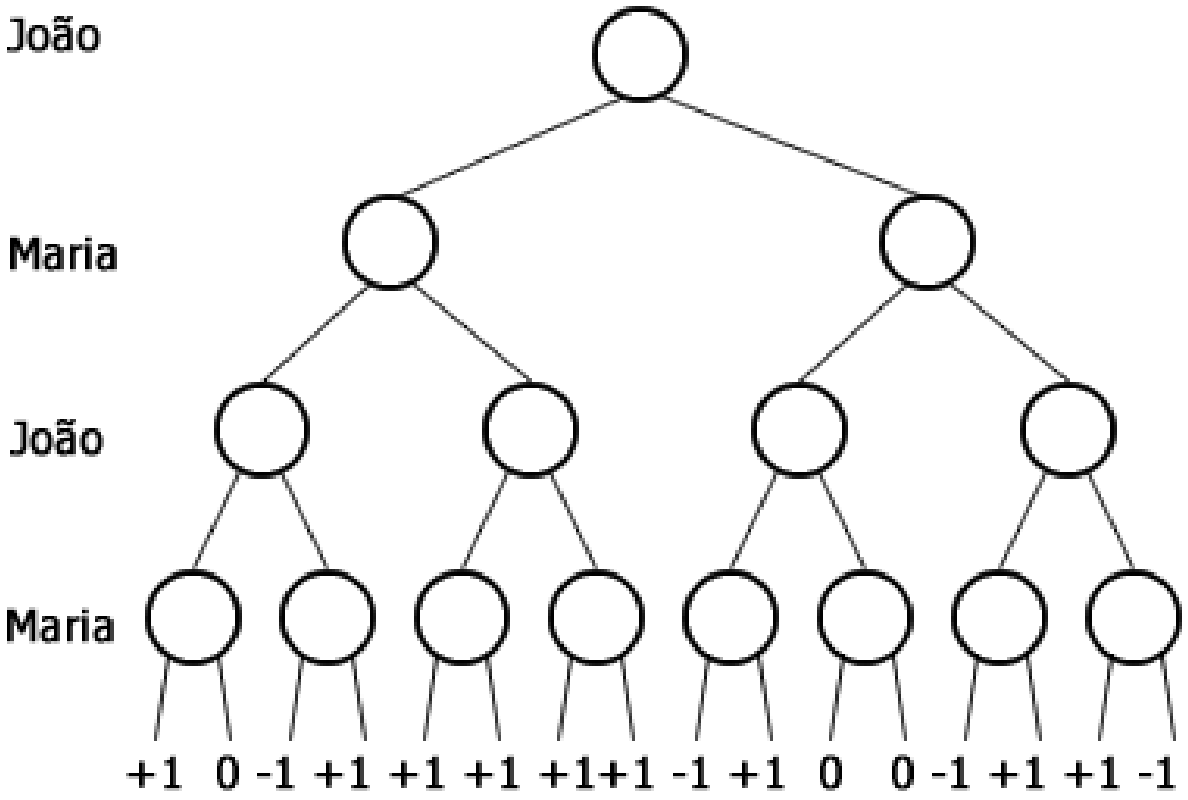


Algoritmo minimax

- Ótimo (para um oponente ótimo);
- Tempo: busca completa em profundidade na árvore do jogo: $O(b^m)$
  - m: profundidade
  - b: movimentos válidos em cada estado
- Espaço:
  - O(bm) se todos os sucessores são gerados
  - O(m) se gera um sucessor por vez
  - Para xadrez, b ≈ 35, m ≈100 para jogos "razoáveis"
    → solução exata não é possível

Exercício

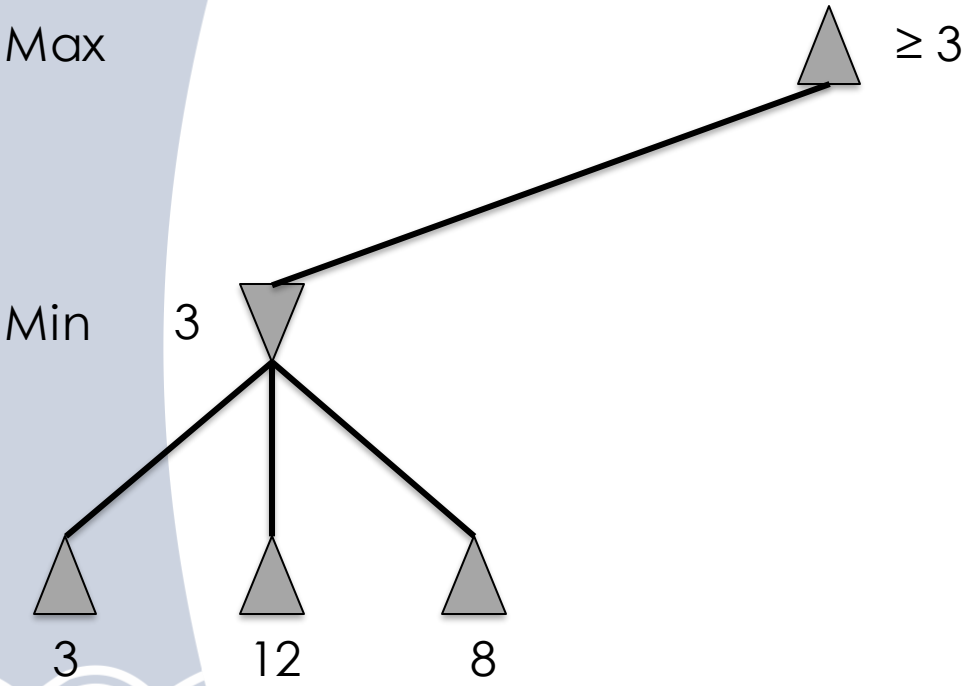
- Considere a árvore de jogo da figura abaixo. Os valores -1, 0 e +1 contidos nos nós folha significam vitória da Maria, empate e vitória do João, respectivamente. **Considerando que os dois jogadores jogam de forma ótima, quem vencerá o jogo?**



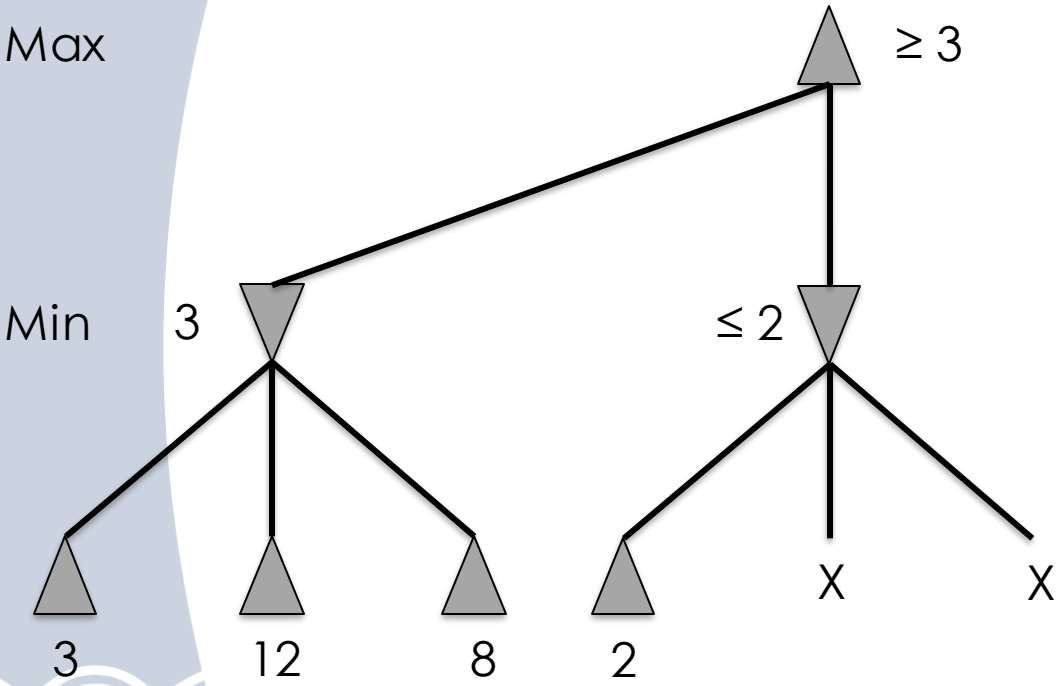
Poda αβ

- O algoritmo Minimax é feito em dois passos
  - **Descida em profundidade e aplicação da heurística**
  - **Retropropagação dos valores**
- O número de estados do jogo, no Minimax, é **exponencial** em relação do número de movimentos porém alguns ramos podem ser **podados**
- Poda α-β :
  - calcular a **decisão correta sem examinar todos os nós** da árvore;
  - **retorna o mesmo que minimax**, porém sem percorrer todos os estados.

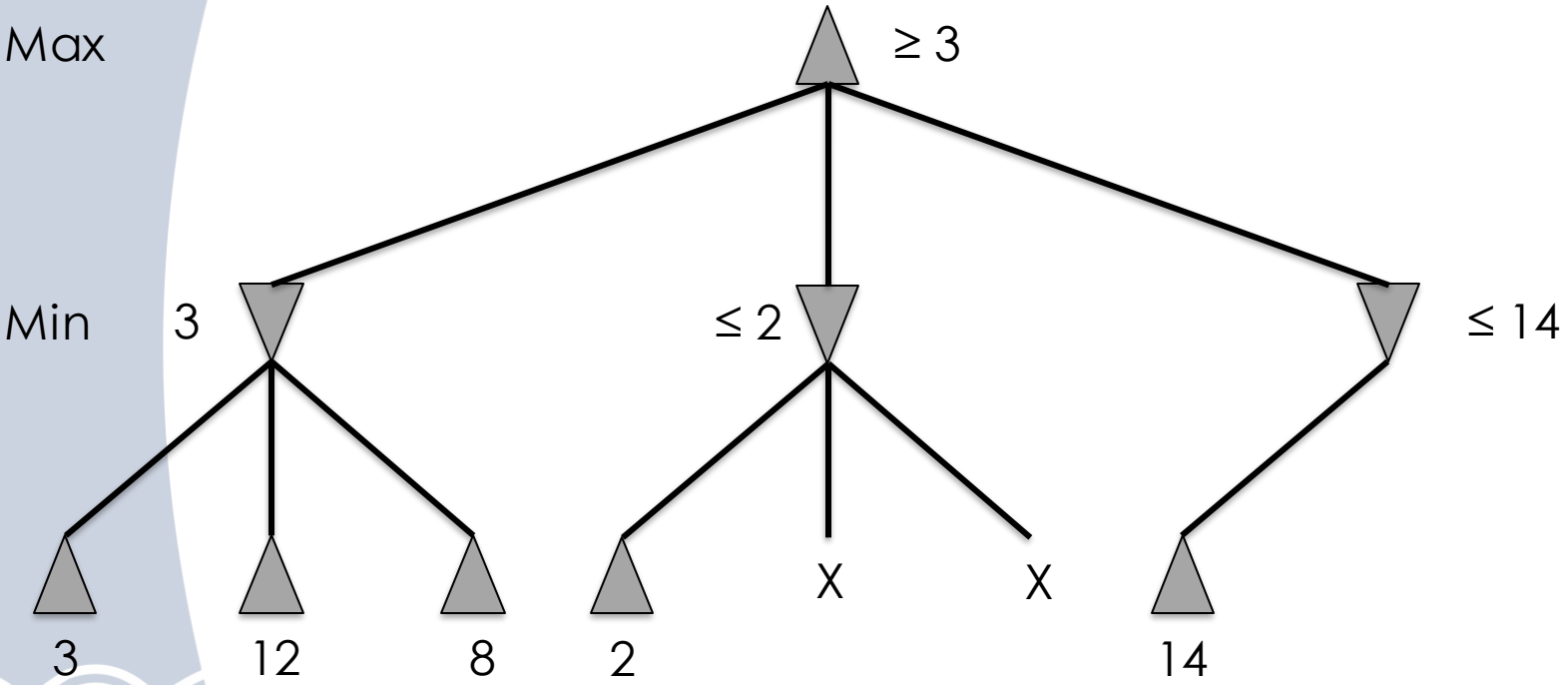
Poda αβ



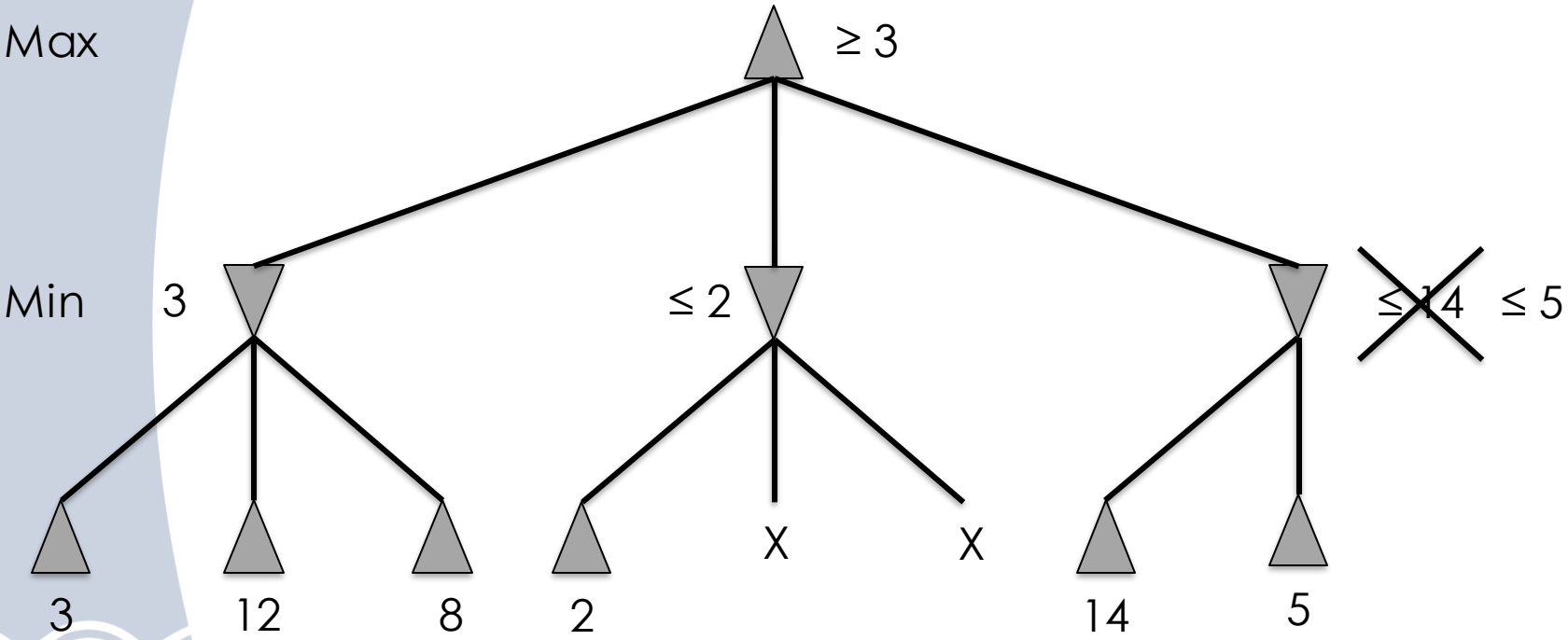
Poda αβ



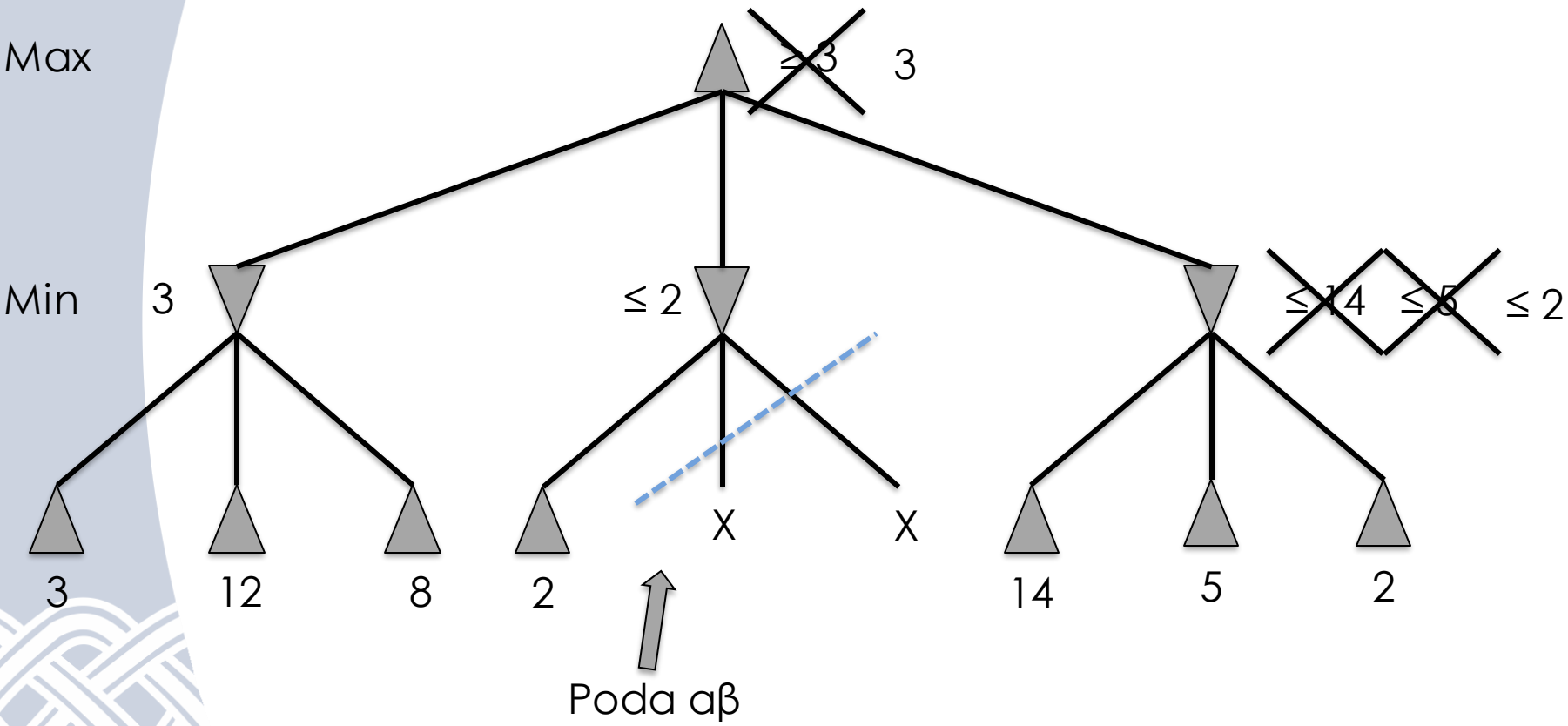
Poda αβ



Poda αβ



Poda αβ



A efetividade da **poda αβ** depende da ordem em que os sucessores são examinados

Poda αβ

- A esse tipo de poda, dá-se o nome de Poda αβ
- Ideia principal: não processar sub-árvores que não afetam o resultado
  - **Valor α**
    - Valor da **melhor escolha** encontrada até agora no caminho para **MAX** (**maior valor**) e **não pode decrescer**
    - **É usado nos nós MIN para decisão de poda**
  - **Valor β**
    - Valor da **melhor escolha** encontrada até agora no caminho para **MIN** (**menor valor**) e **não pode aumentar**
    - **É usado nos nós MAX para decisão de poda**

Poda αβ

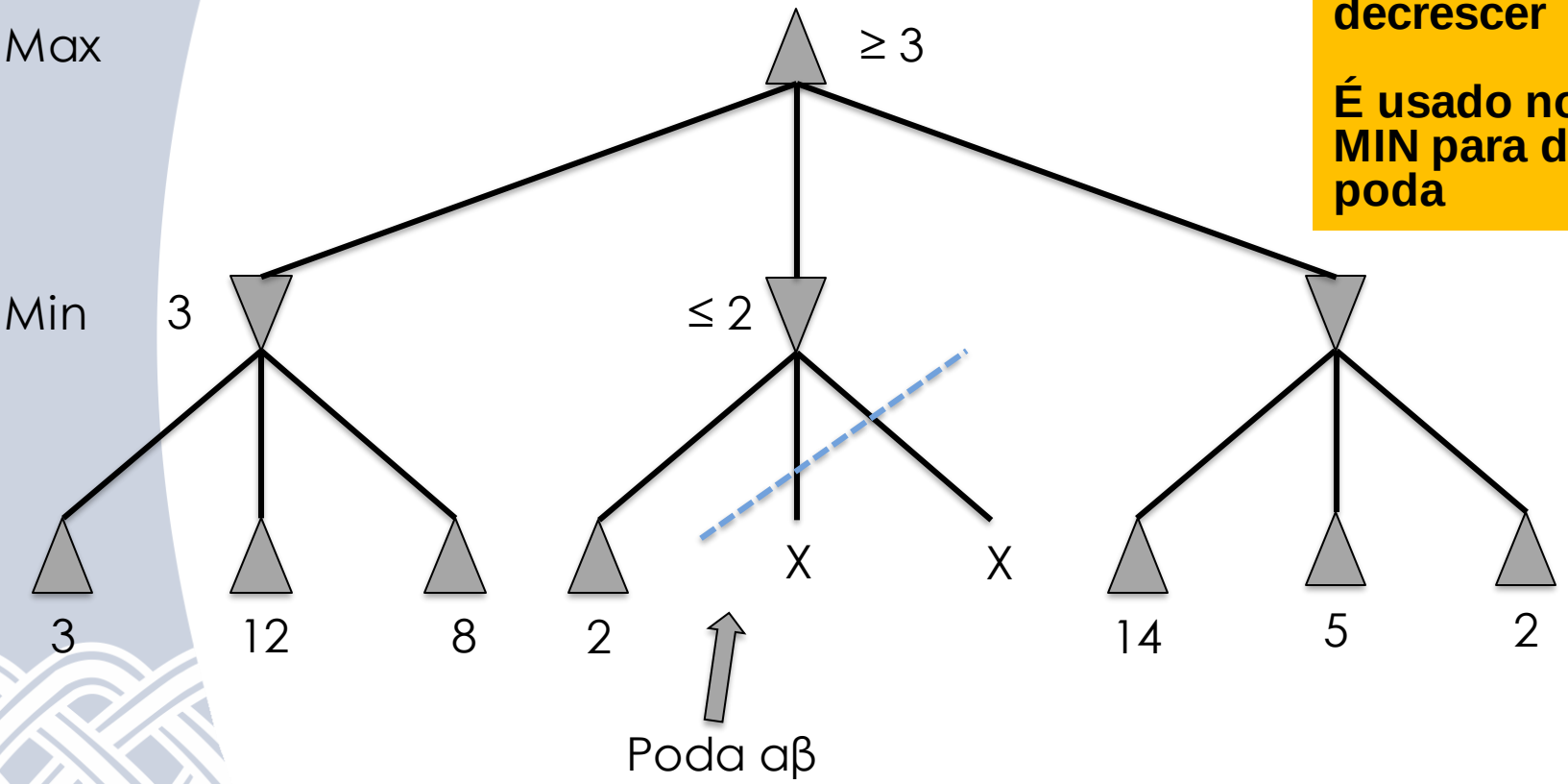
- Visite a árvore de busca em profundidade
- Para cada nó **n MAX**, **α(n)** = máximo valor até agora
- Para cada nó **n MIN**, **β(n)** = mínimo valor até agora
  - O valor de α começa em –∞ e somente incrementa
  - O valor de β começa em +∞ e somente decrece
- **Corte β**: dado um nó **n**, corte a busca após **n** MAX se **α(n) >= β(i)** para algum nó **i** MIN ancestral de **n**.
- **Corte α:** corte a busca abaixo de um nó **n** MIN se **β(n) <= α(i)** para algum nó **i** MAX ancestral de **n**.

Poda αβ

Valor α

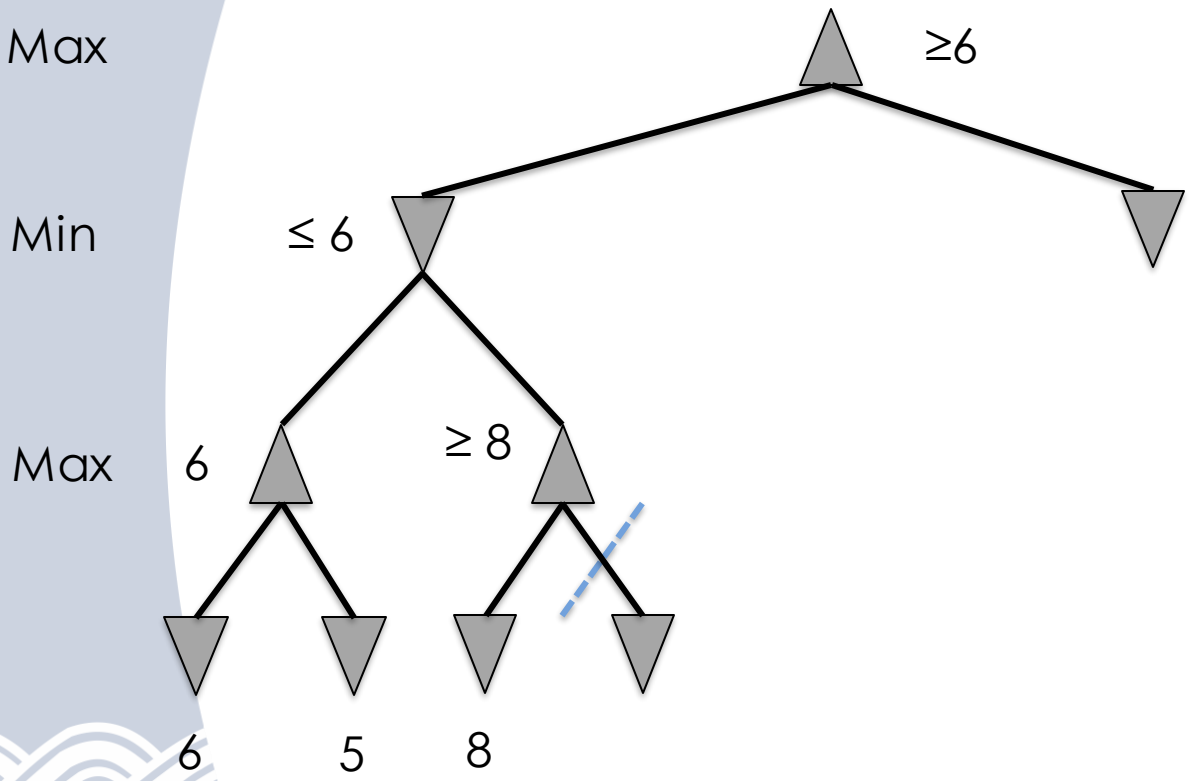
Valor da **melhor escolha** encontrada até agora no caminho para **MAX** (**maior valor**) e **não pode decrescer**

É usado nos nós MIN para decisão de poda



- **Corte α:** corte a busca abaixo de um nó **n** MIN se **β(n) <= α(i)** para algum nó **i** MAX ancestral de **n**.

Poda αβ



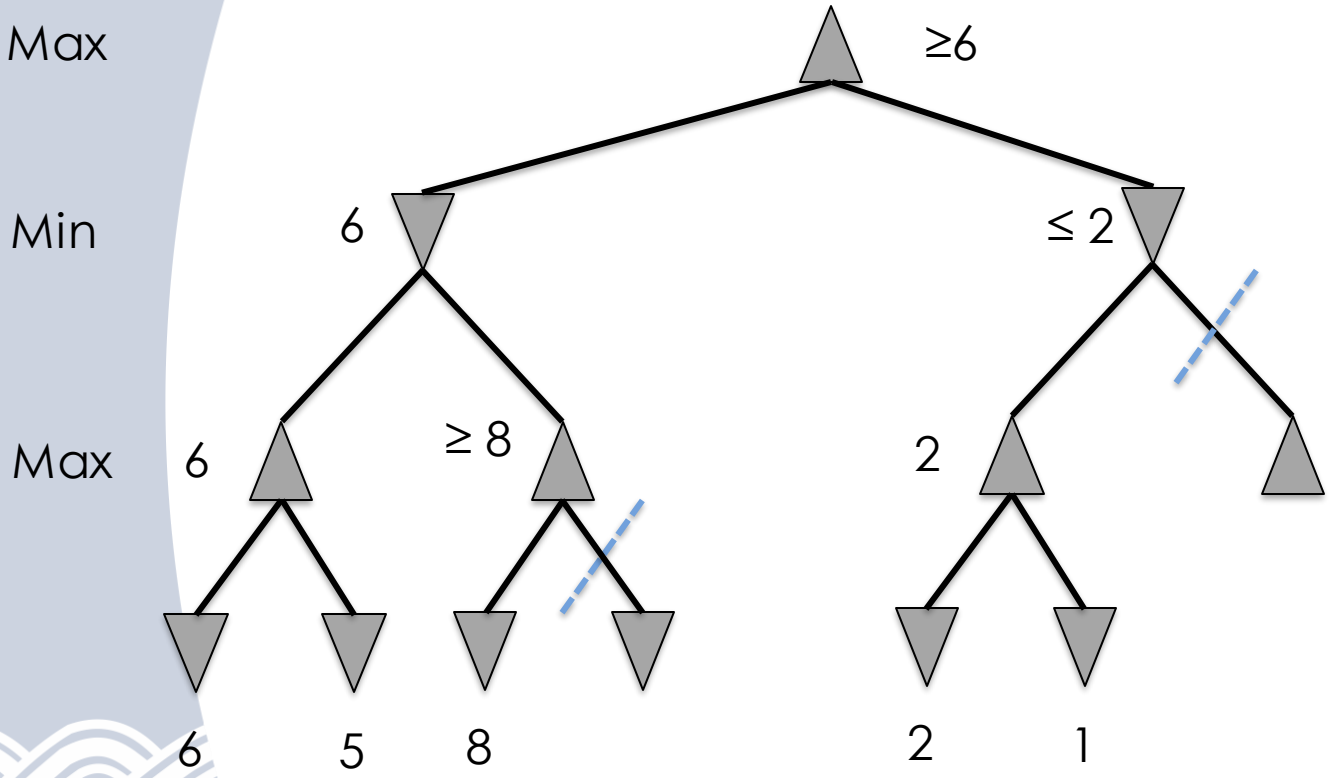
Valor β

Valor da **melhor escolha** encontrada até agora no caminho para **MIN** (**menor valor**) e **não pode aumentar**

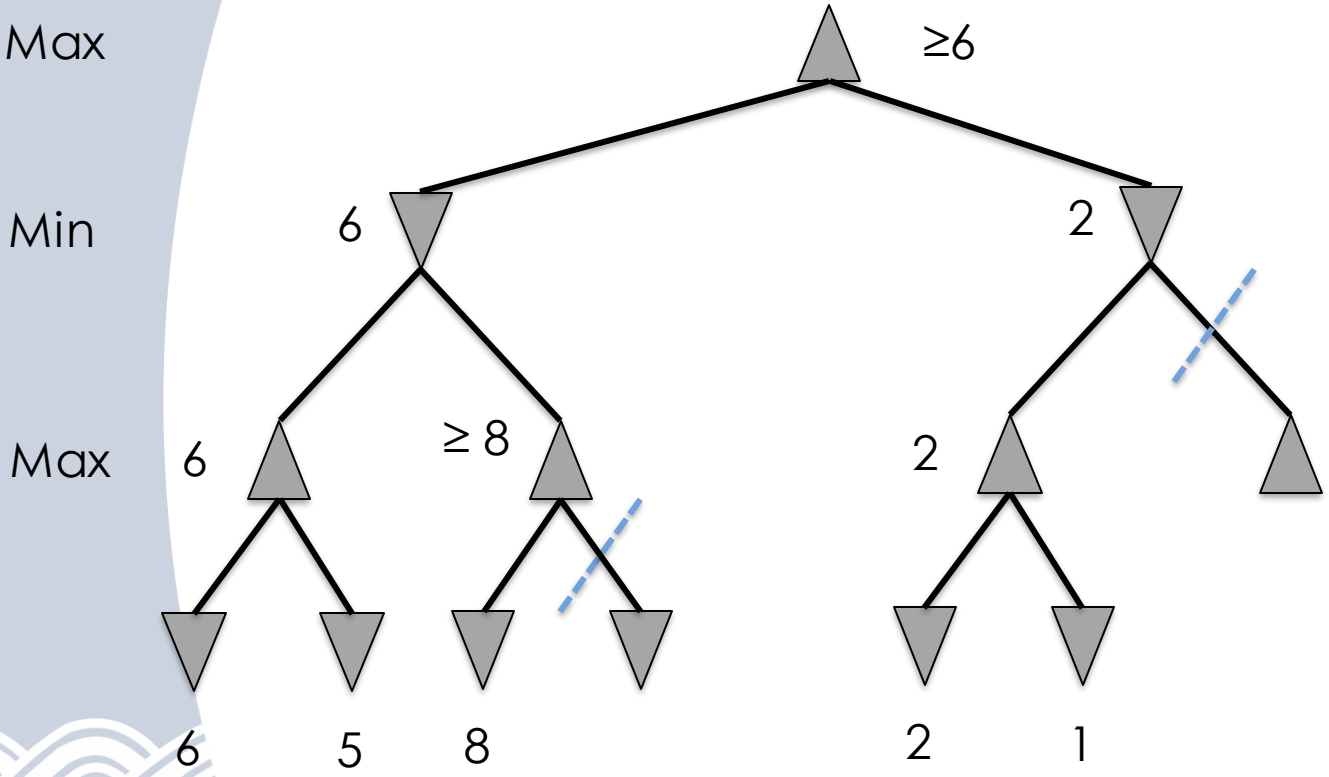
É usado nos nós MAX para decisão de poda

- **Corte β**: dado um nó **n**, corte a busca após **n** MAX se **α(n) >= β(i)** para algum nó **i** MIN ancestral de **n**.

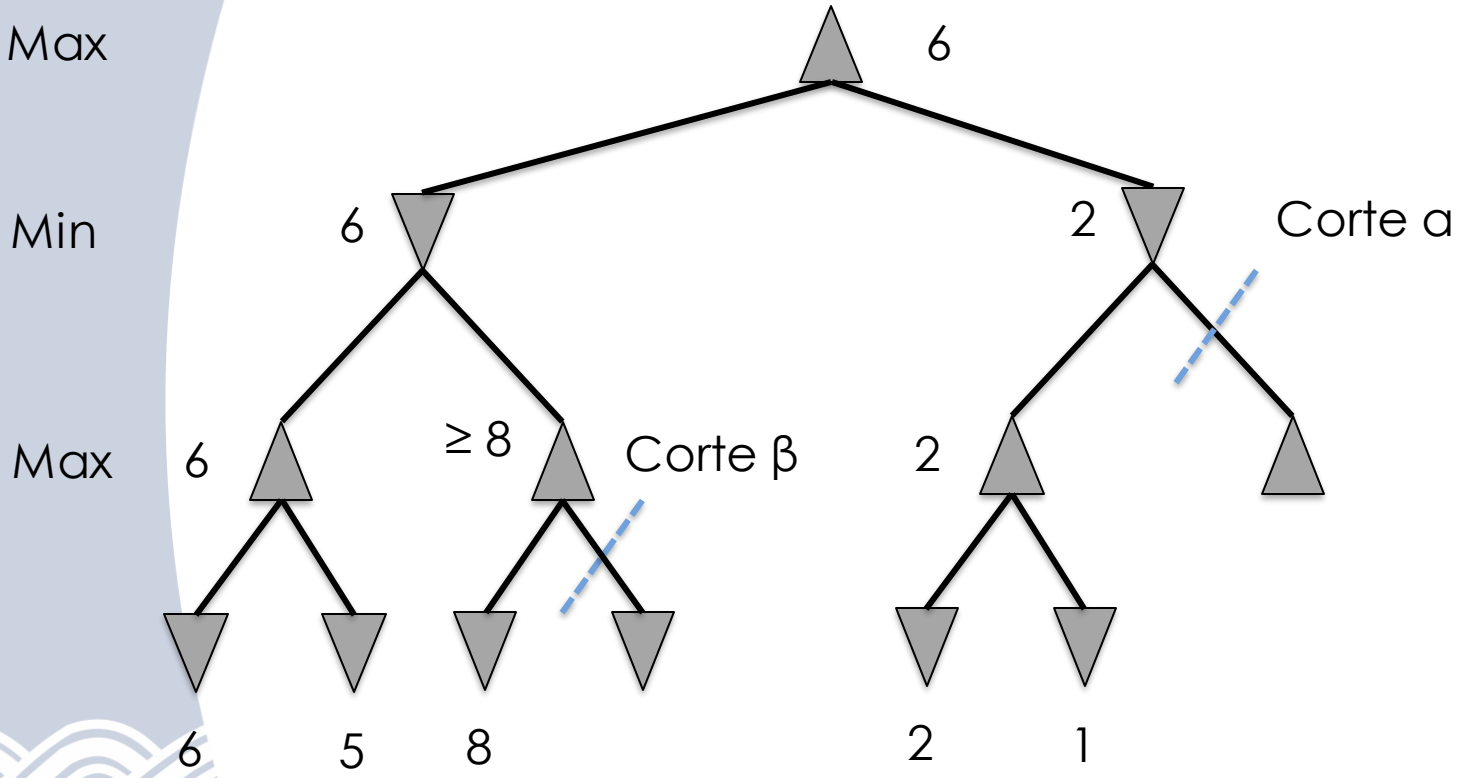
Poda αβ



Poda αβ



Poda αβ

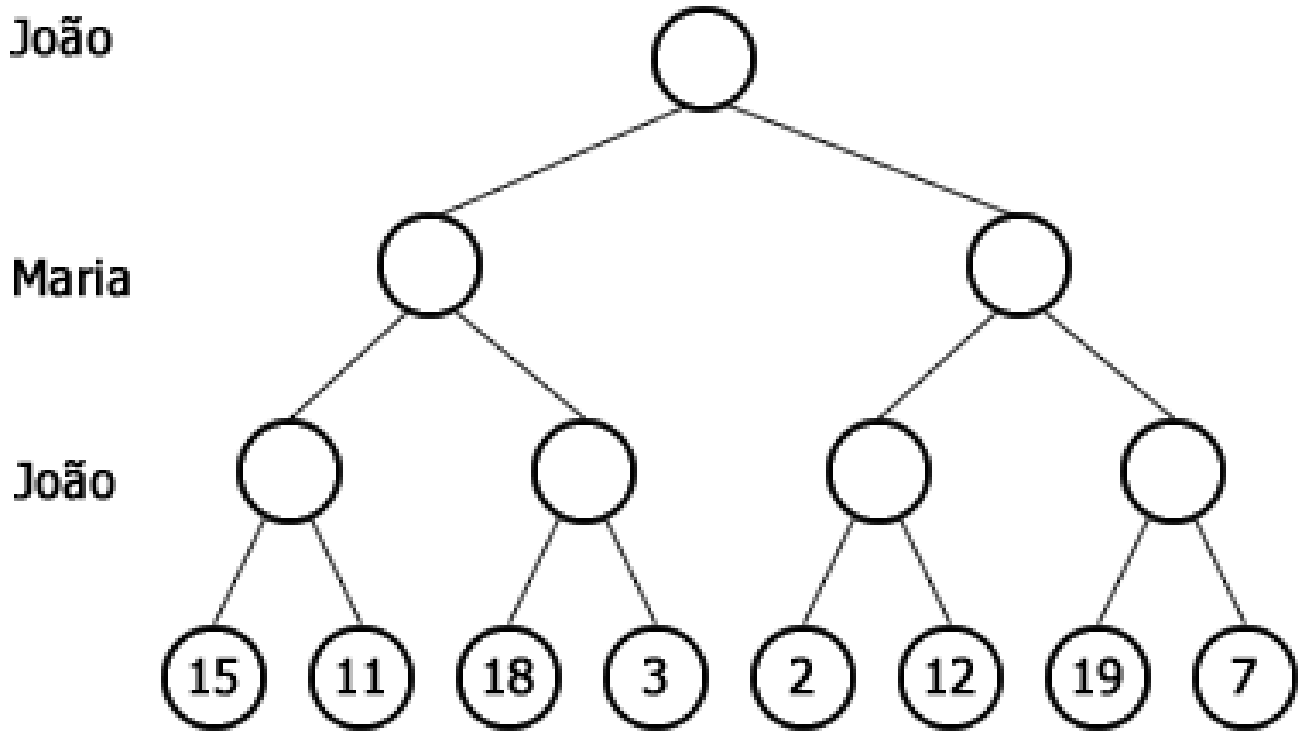


Poda αβ

- Calcula a decisão **correta** sem examinar todos os nós da árvore
- Retorna o **mesmo resultado** que o algoritmo Minimax, porém sem percorrer todos os estados
- A efetividade da Poda α β depende da ordem em que os sucessores são examinados
  - Com a melhor ordem possível, a complexidade de tempo = $O(b^{m/2})$

Exercício

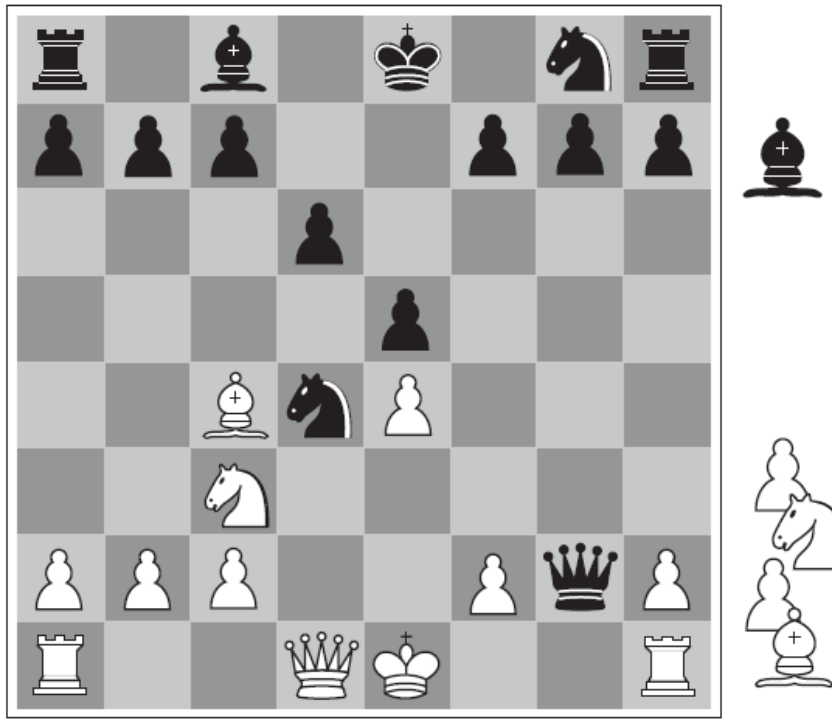
Considere a árvore de jogo da figura abaixo. A função de utilidade nesse jogo pode assumir um valor entre 0 e 20, sendo que João é beneficiado por valores mais altos, enquanto Maria é beneficiada por valores mais baixos. Considerando que os dois jogadores jogam de forma ótima, qual o valor de utilidade que João terá no fim do jogo? Se for utilizada a poda αβ, quantos nós deixarão de ser avaliados?



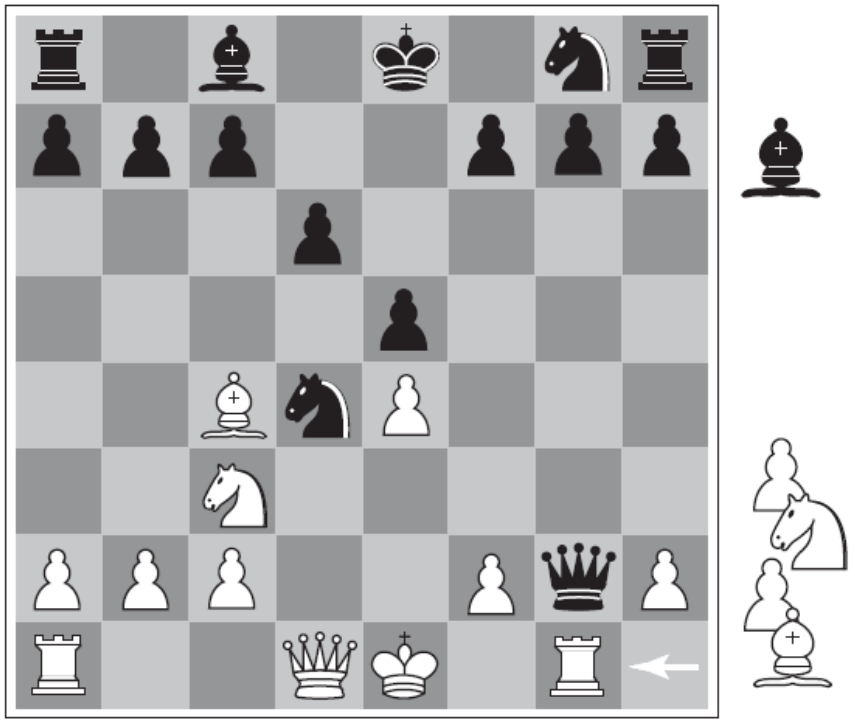
Função de Avaliação

- Exemplos da função de avaliação:
  - Diferença entre o total de pontos dos dois jogadores
  - Somas ponderadas de fatores (ex. Xadrez)
  - $utilidade(S) = w_1 f_1(S) + w_2 f_2(S) + \ldots + w_n f_n(S)$
  - $f_1(S)$ = (Num. de rainhas Br) – (Num. de rainhas Pr)
  - $f_2(S)$ = (Num. de torres Br) – (Num. de torres Pr)
  - ...

Função de Avaliação

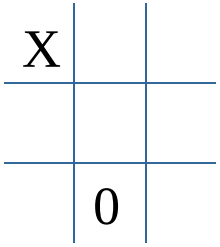


(a) White to move

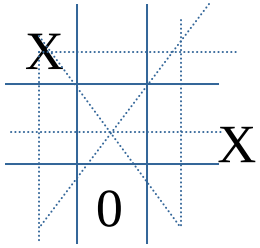


(b) White to move

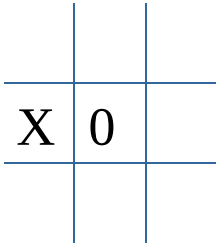
Função de Avaliação



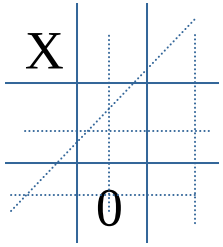
h = 6 - 5 = 1



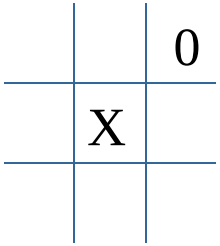
X tem 6 possibilidades



h = 4 - 6 = = -2

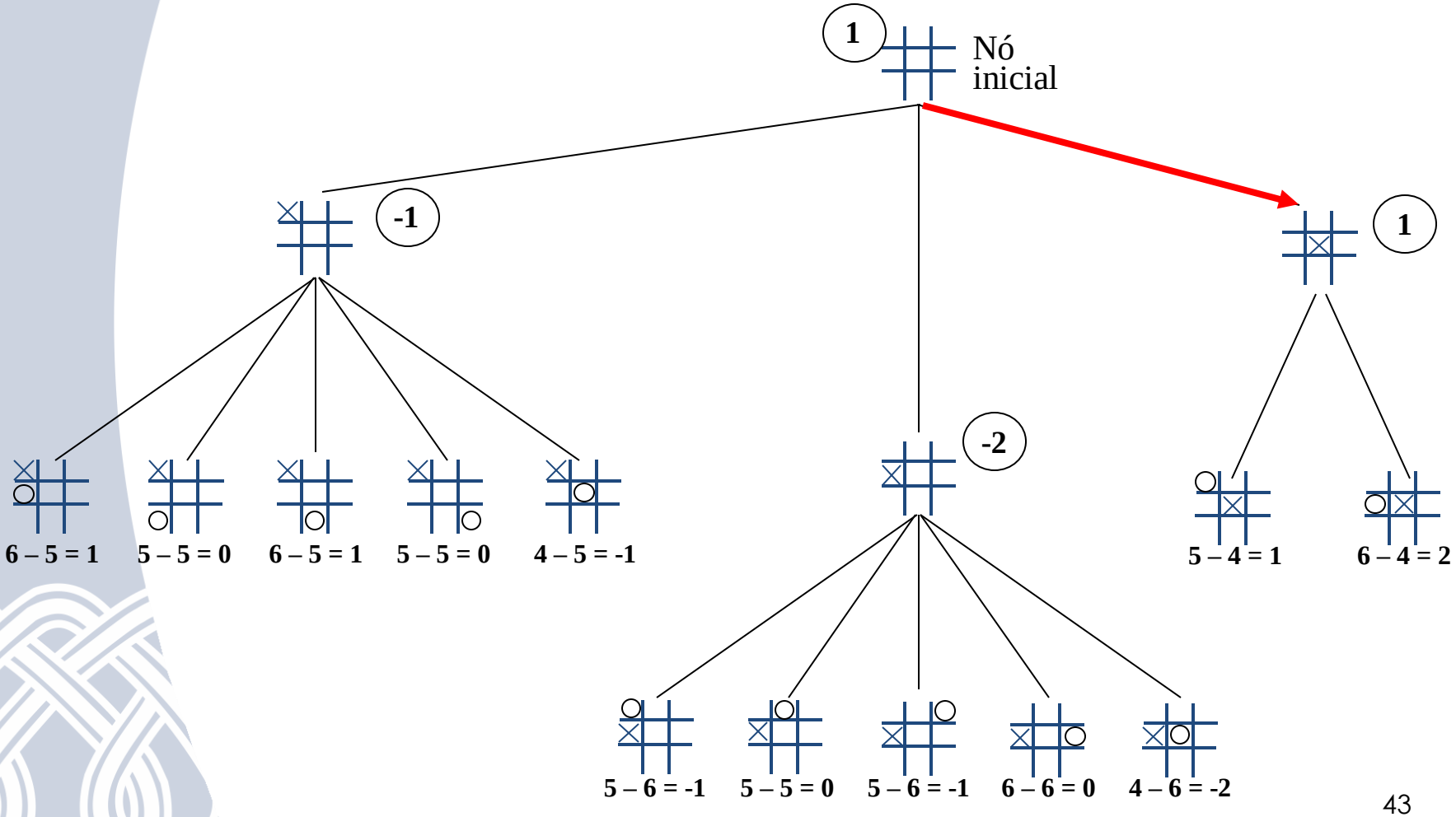


0 tem 5 possibilidades



h = 5 - 4 = 1

Função de Avaliação



43

Mecanismo de Parada

- Um mecanismo de parada simples é fazer uma busca com profundidade limitada, ajustando uma profundidade máxima *d* compatível com o tempo disponível para processamento
- Alternativamente, pode-se fazer uma busca em profundidade iterativa, aumentando *d* até que não se tenha mais tempo disponível.

Combinando as Técnicas

- Para um jogo de xadrez com as técnicas descritas sem poda alfa-beta (tabela de transposições, função avaliação heurística, busca limitada e processamento de 200 milhões de nós por turno) seria possível um lookahead de aproximadamente 5 jogadas
  - Nível de um jogador iniciante
- Com busca alfa-beta com ordenação dos nós seria possível atingir 10 jogadas
  - Nível de um jogador avançado

Exercício Busca competitiva

- Inicialmente, há 2 conjuntos de 2 fósforos. Em cada jogada um jogador remove qualquer número de fósforos de exatamente uma pilha. O vencedor é aquele que remove o último fósforo do jogo.
  - Desenhe a árvore minimax deste jogo.
  - Há como executar poda $\alpha\beta$?

Dados históricos

- Até as décadas de 1960 e 1970, os jogos eletrônicos não utilizavam técnicas de IA
- A indústria percebeu que a inclusão dessas técnicas poderia atrair um público maior, aumentando, assim, os lucros
- Havia também a necessidade da inclusão de elementos virtuais que imitassem o comportamento humano, para que os jogadores pudessem jogar sozinhos
- No meio acadêmico, já havia muitas técnicas capazes de oferecer a entidades virtuais características de autonomia e raciocínio, potencialmente úteis aos jogos
- Em 1974, com os jogos Pursuit e Qwak, os jogadores tinham que atirar em alvos móveis
- Em 1978, o jogo Space Invaders implantou as primeiras entidades inteligentes em jogos

Dados históricos

- Em 1980, Pac-man conta com movimentos padronizados dos inimigos, porém cada fantasma tem um modo diferente de caçar o jogador
- Em 1990, O primeiro jogo de estratégia em tempo real, Herzog Wei, é lançado. A busca de caminho apresentada nesse jogo era de baixa qualidade
- Em 1993, Doom é lançado como primeiro jogo de tiro em primeira pessoa
- Em 1996, BattleCruiser: 3000AD é o primeiro jogo a utilizar redes neurais comercialmente
- Em 1998, Half-Life é lançado como a melhor Game IA até o momento
- Em 2001, o jogo Black & White é alvo da mídia a respeito de como as criaturas aprendem com as decisões do jogador

IA acadêmica versus game IA

- O termo **game IA** surgiu para diferenciar os estudos em IA para jogos eletrônicos dos elaborados na academia
- A principal diferença entre a IA acadêmica e a game IA é que a primeira tem por objetivo a **solução de problemas difíceis**, como reconhecimento de padrões, enquanto a segunda tem por **objetivo a diversão dos jogadores**, seja pelo aumento do grau de verossimilhança dos jogos, ou pelo nível de desafio apresentado.
- Para isso, são utilizadas algumas das soluções pesquisadas e encontradas no meio acadêmico.
- Técnicas de game IA:
  - Algoritmos determinísticos e padrões de movimentos Máquinas de estados
  - Sistemas baseados em regras
  - Algoritmos de busca
  - Algoritmos genéticos

Encerramos busca em IA!

- Busca não informada: espaço de busca de +- 10^{25}
- Busca informada: espaço de busca de +- 10^{11}
- Busca competitiva: 10^{20} nós no xadrez, com corte
- Por quê busca é tão importante em IA?
  - Diversos problemas em IA são intratáveis. Busca é a única maneira de lidar com eles
  - Diversas aplicações de busca: Aprendizado / Planejamento / PLN / Visão

Sugestão de leitura

- Artificial Intelligence – A Modern Approach

  Stuart Russel and Peter Norvig. 2nd. ed.

  Cap. 6 – Seções 6.1 a 6.4

Agradecimentos:

Bruno Magalhães Nogueira

Gustavo Batista