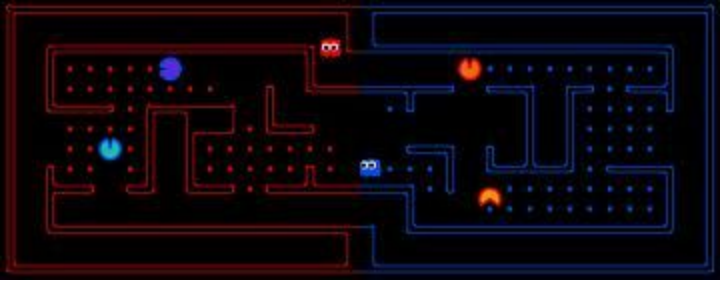


Inteligência Artificial
Ciência da Computação

Prof. Aline Paes / alinepaes@ic.uff.br

Busca com Agentes Adversários - Jogos
RN 5; ER 12

Universidade Federal Fluminense

Instituto de Computação

História do xadrez no computador

- Shannon e Turing escreveram sobre o xadrez em 1950
  - sugeriram a base dos algoritmos usados hoje (min-max e poda)
- Em 1956 temos o primeiro jogo de xadrez no computador (tabuleiro 6X6)
- Em 1968 David Levy aposta que nenhum programa de computador o venceria antes de 1978

Aposta de David Levy

- 1973: "*Clearly, I shall win my ... bet in 1978, and I would still win if the period were to be extended for another ten years. Prompted by the lack of conceptual progress over more than two decades, I am tempted to speculate that a computer program will not gain the title of International Master before the turn of the century and that the idea of an electronic world champion belongs only in the pages of a science fiction book.*"

Aposta de David Levy

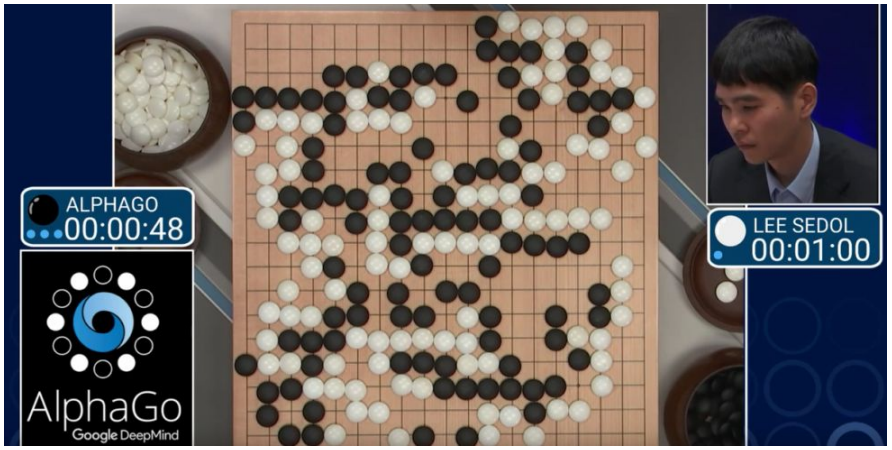
- 1978: “*I had proved that my 1968 assessment had been correct, but on the other hand my opponent in this match was very, very much stronger than I had thought possible when I started the bet.*”

Deep Blue

- Venceu Gary Kasparov em 1997
  - Baseado na técnica de poda que veremos daqui a pouco
  - Hardware mostruoso!
    - Busca por 30 bilhões de posições a cada jogada
    - Alcançava profundidade 15 rotineiramente
    - Função de avaliação com 8000 atributos
    - Banco de dados de 700.000 jogos de grandmasters e incluindo todas as posições de 5 peças

AlphaGo e AlphaZero

- Desenvolvido pela Google DeepMind para jogar Go
- Em março / 2016 venceu Lee Sedol, campeão Sul-coreano
- Utiliza
  - MCTS
  - Reinforcement Learning
  - Neural Networks
  - TPUs



- Alguns alegam que ele marca o fim de uma era...

Jogos e IA

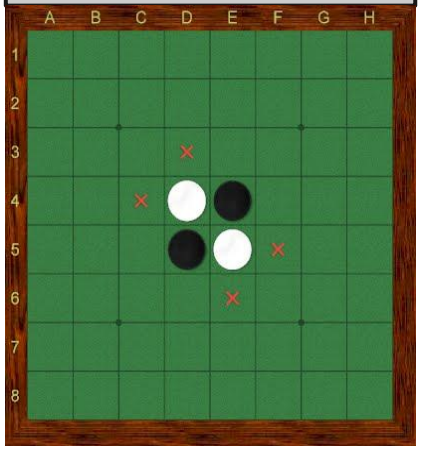
- Por que estudar jogos em IA?
  - não são triviais
  - é necessário "inteligência" para jogar e principalmente para vencer
  - Podem ser bastante complexos, considerando principalmente o espaço de busca
  - requerem tomada de decisão dentro de um tempo limitado
  - bem definidos

Tipos de jogos

- Zero-sum: a vitória de um jogador é a derrota do outro
- Discreto: estados e decisões têm valores discretos
- Finito: número finito de estados e decisões
- Determinístico: não envolvem chance (lançar dados, moedas, etc)
- Informação perfeita: cada jogador pode ver o estado completo do jogo. Sem decisões simultâneas

Classificação de Jogos

Othello

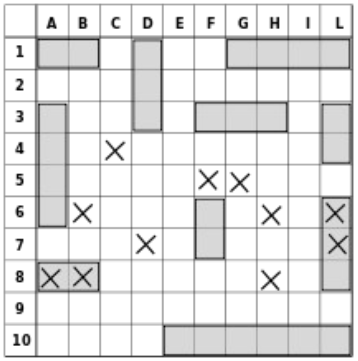


Poker

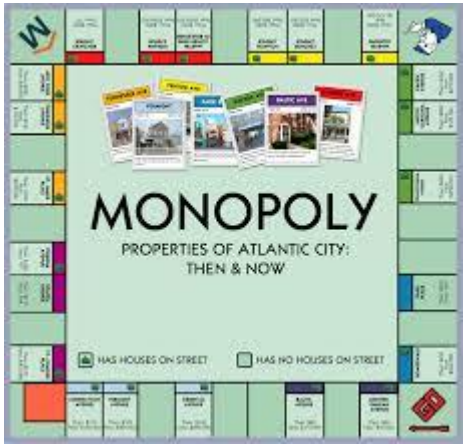
Backgammon

[How to play]

Battle ship



Bridge



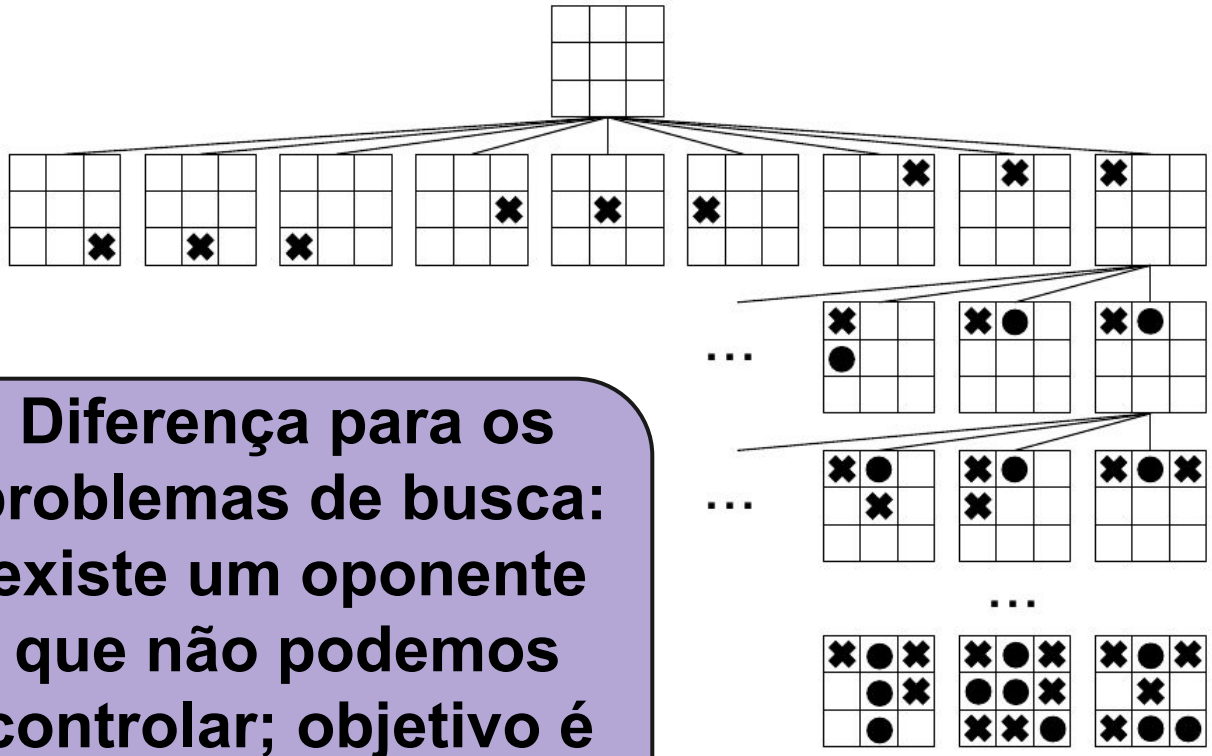
Tipos de jogos do ponto de vista de IA

	Determinístico	Estocástico
Completamente observável	Damas, xadrez, Go, Othello, Tic-Tac-Toe	Backgammon, Monopoly
Parcialmente observável	Battleship	Bridge, Poker, Scrabble

Jogo e busca

- Considere jogos de tabuleiro com dois jogadores, informação perfeita e zero-sum
  - xadrez, damas, jogo da velha, etc
- Configuração do tabuleiro: disposição única de peças
- Representação como um problema de busca
  - estados ?
  - ações ?
  - estado inicial ?
  - meta ?

Exemplo: jogo da velha

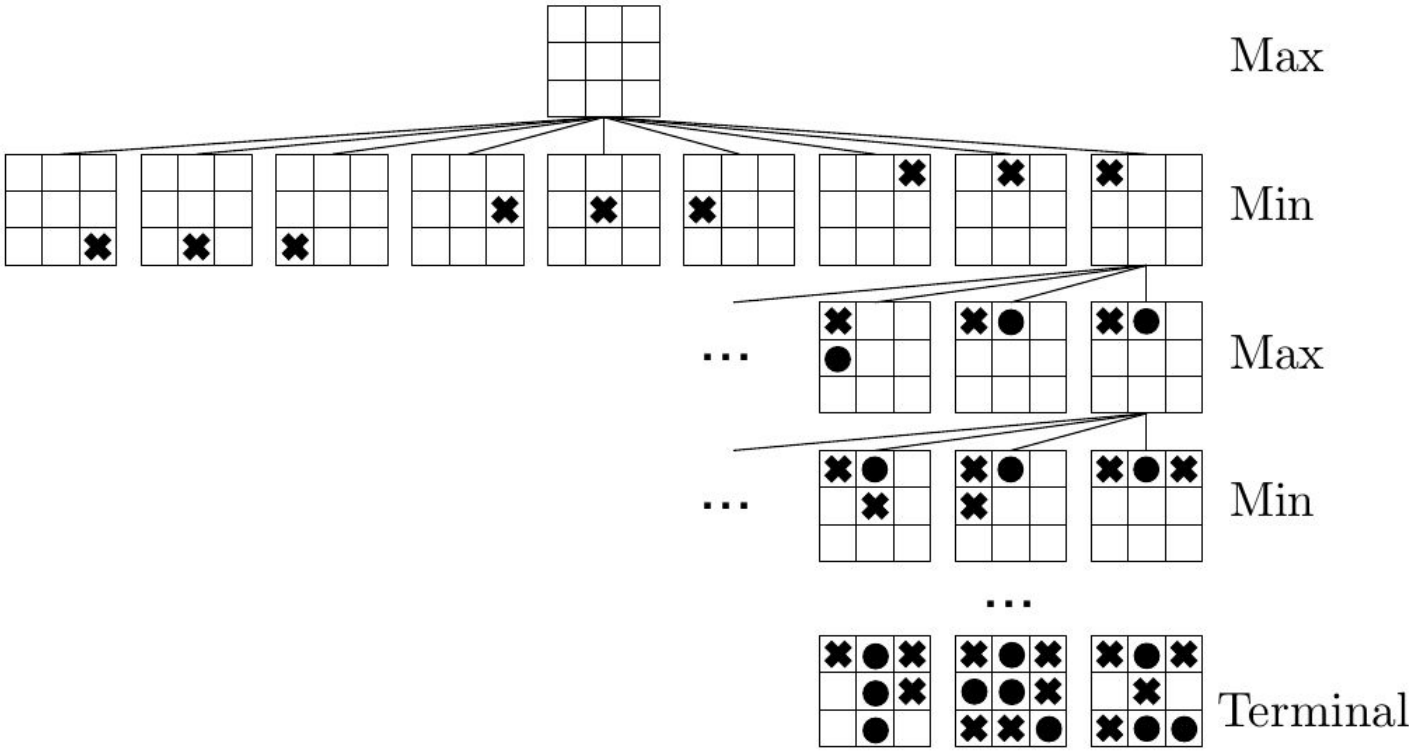


Diferença para os problemas de busca: existe um oponente que não podemos controlar; objetivo é vencer

Formalização da busca

- Estado inicial: configuração inicial do Jogo
- Actions(s): conjunto de jogadas permitidas em um estado s
- Result(s,a): modelo de transição, define o resultado de uma jogada
- Terminal-Teste(s): verdadeiro quando o jogo acabou
- **Dois jogadores**: *min e max*
- **Player(s)**: qual jogador pode jogar no estado *s*
- **utility(s,p)**: define um valor numérico final para um jogo que termina em um estado *s*, para o jogador *p*

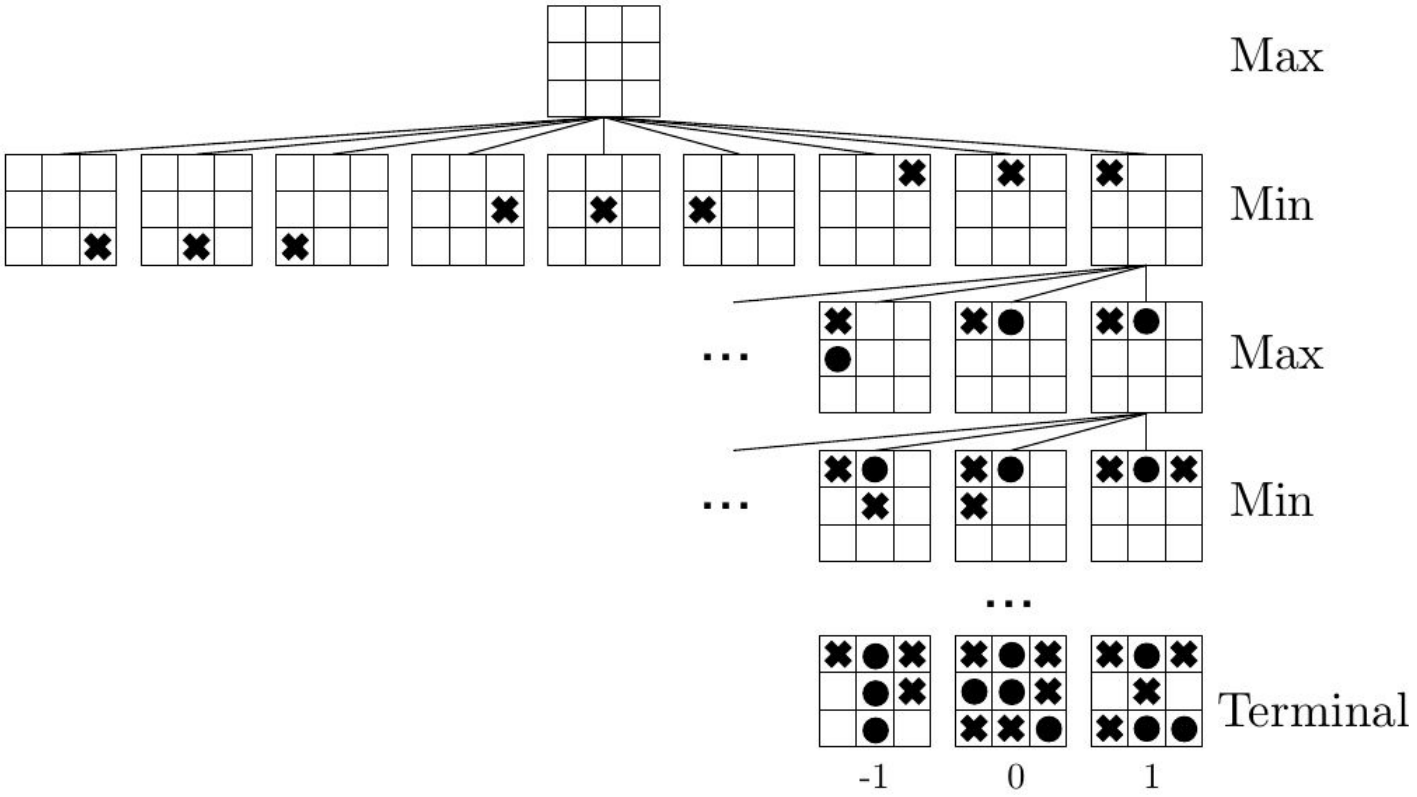
Exemplo: jogo da velha



Função de utilidade

- Função de utilidade: mapeia cada estado final do tabuleiro para um valor de pontuação, indicando o valor da saída para o jogador
  - Vitória: valores positivos
  - Derrota: valores negativos
  - Empate: **0**

Exemplo: jogo da velha



Estratégia

- Busca tradicional
  - encontrar um caminho até a meta
  - encontrar um estado de solução

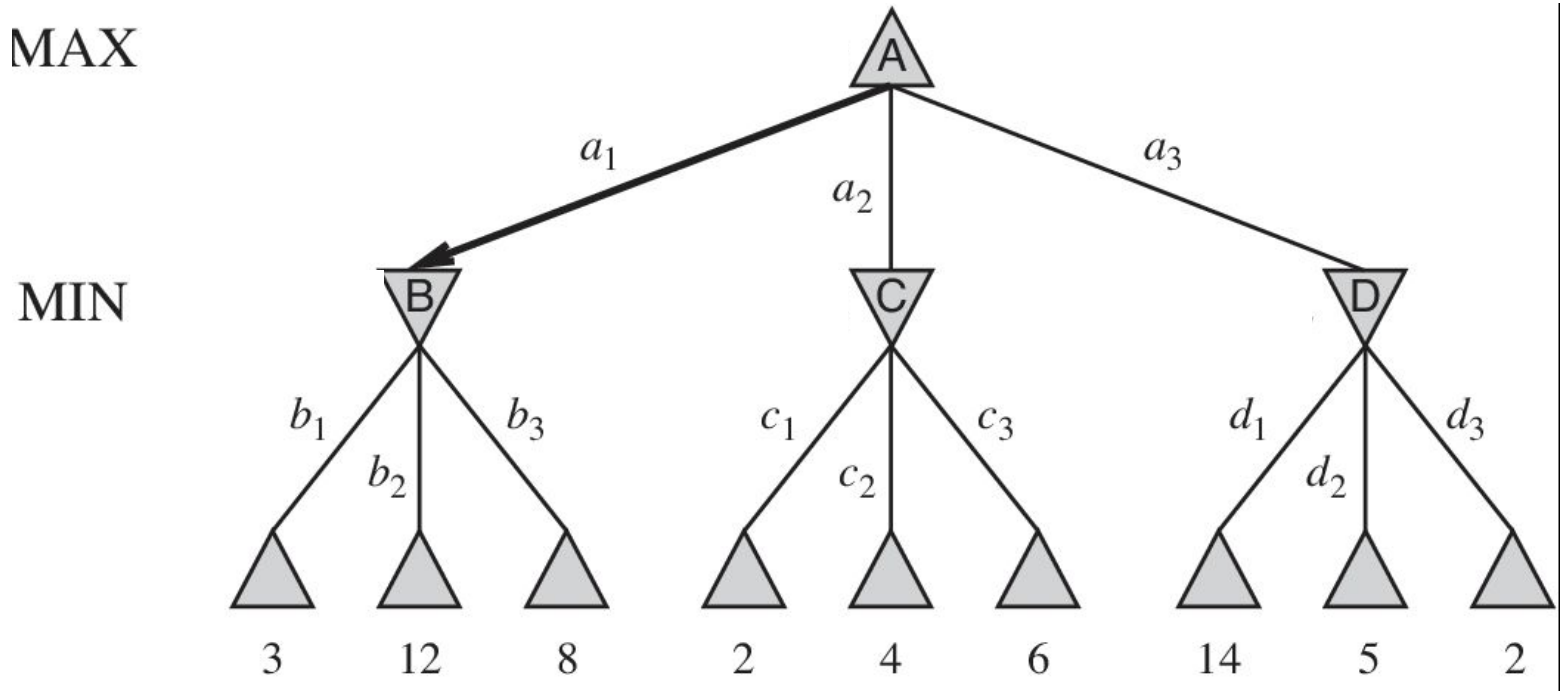
- Busca em jogos
  - estratégia que
    - tente chegar em uma vitória e/ou
    - não deixe o adversário ganhar

Estratégia

- Busca em jogos
  - estratégia que
    - tenta chegar em uma vitória e/ou
    - não deixe o adversário ganhar

- Considerando dois jogadores, MAX e MIN
  - solução ótima para MAX depende das jogadas de MIN
  - supõe que MIN está jogando o melhor que ele pode

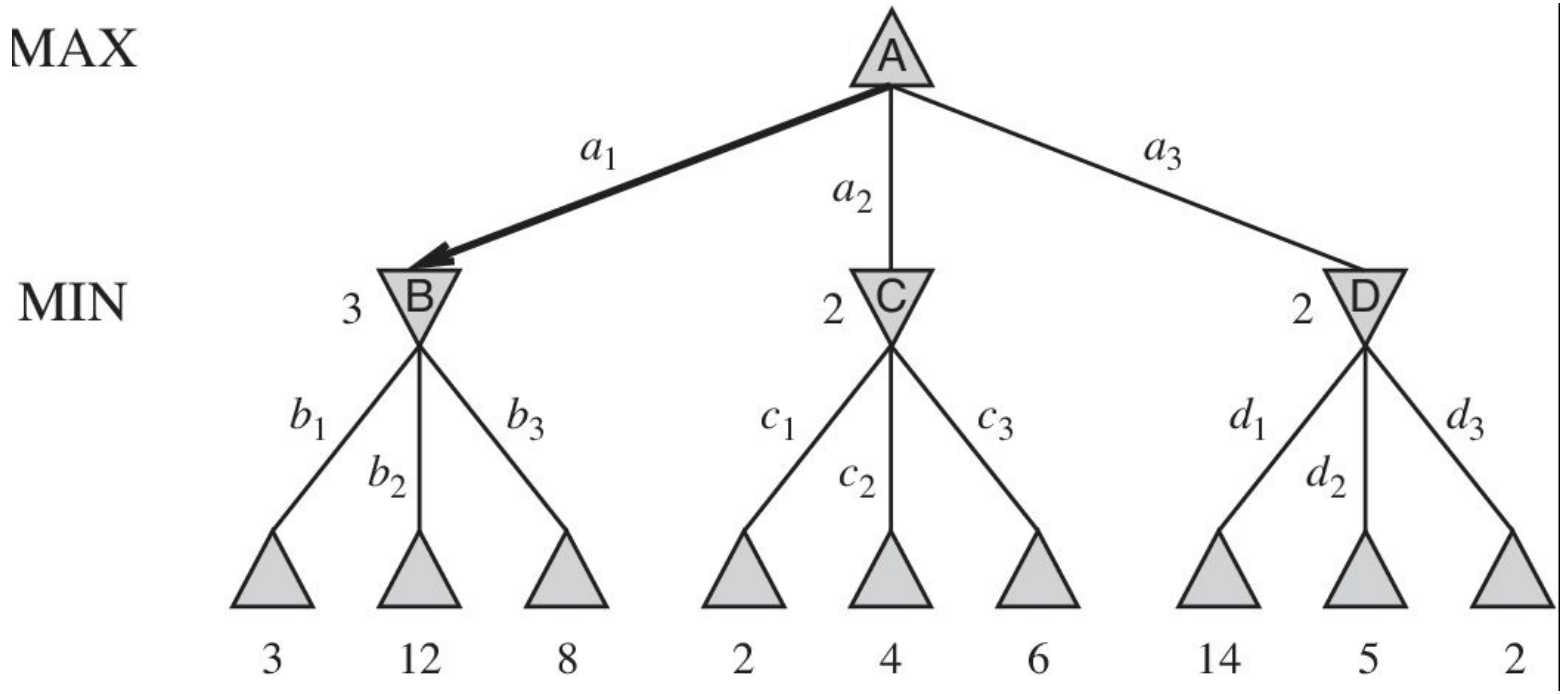
Árvore de jogo



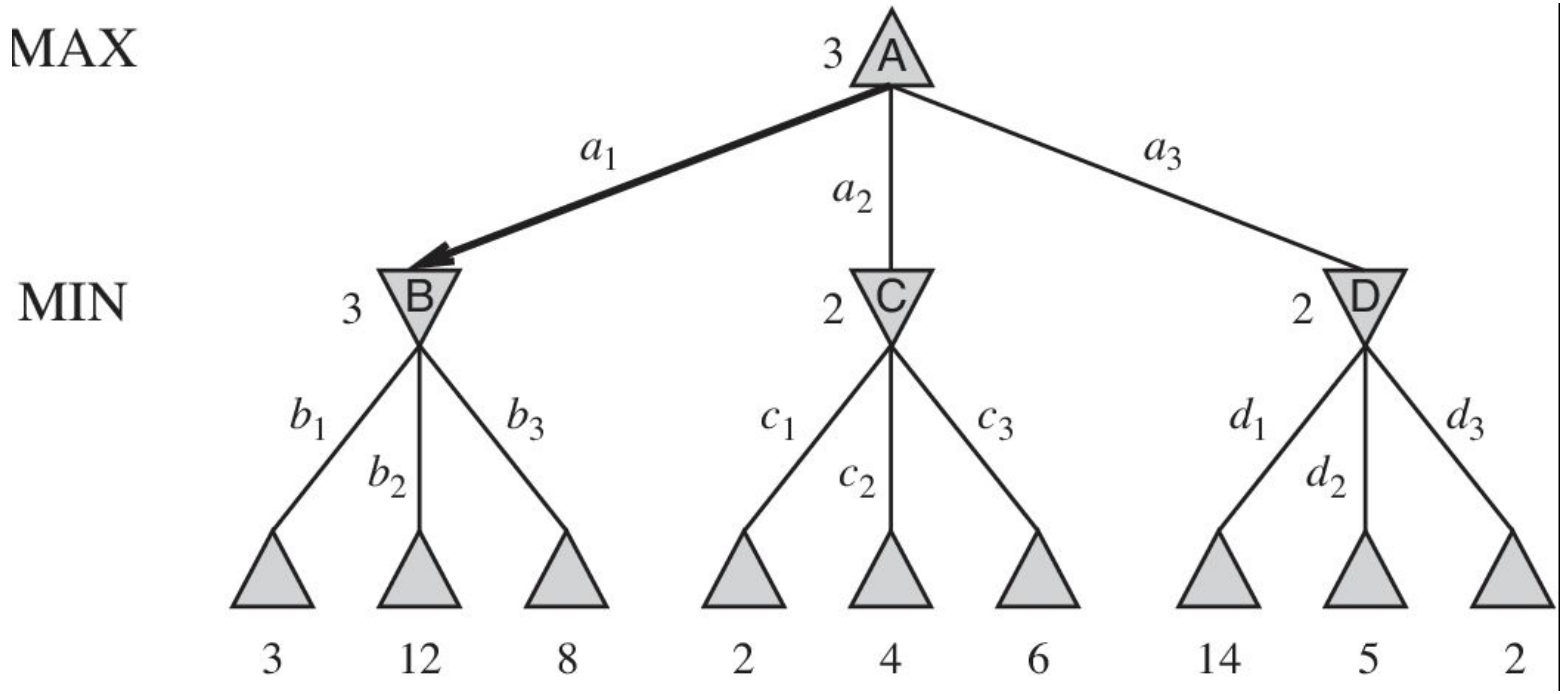
MINIMAX

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

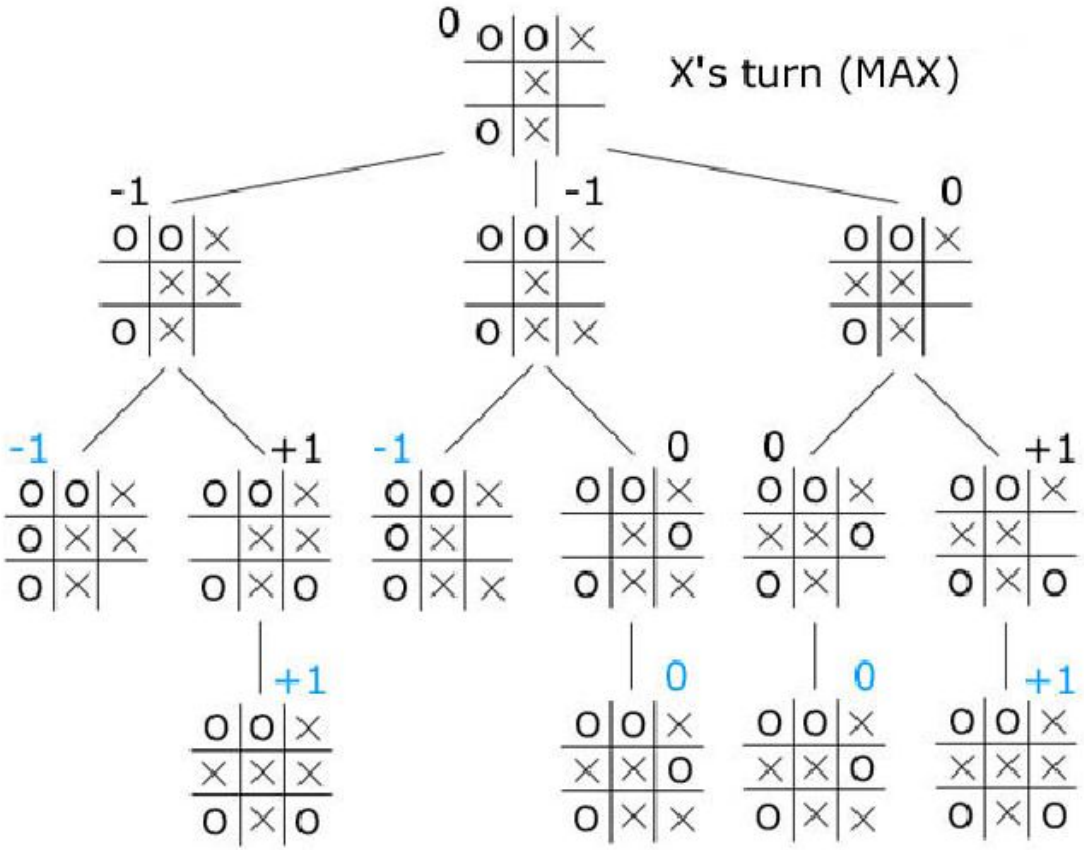
MINIMAX



Decisão MINIMAX



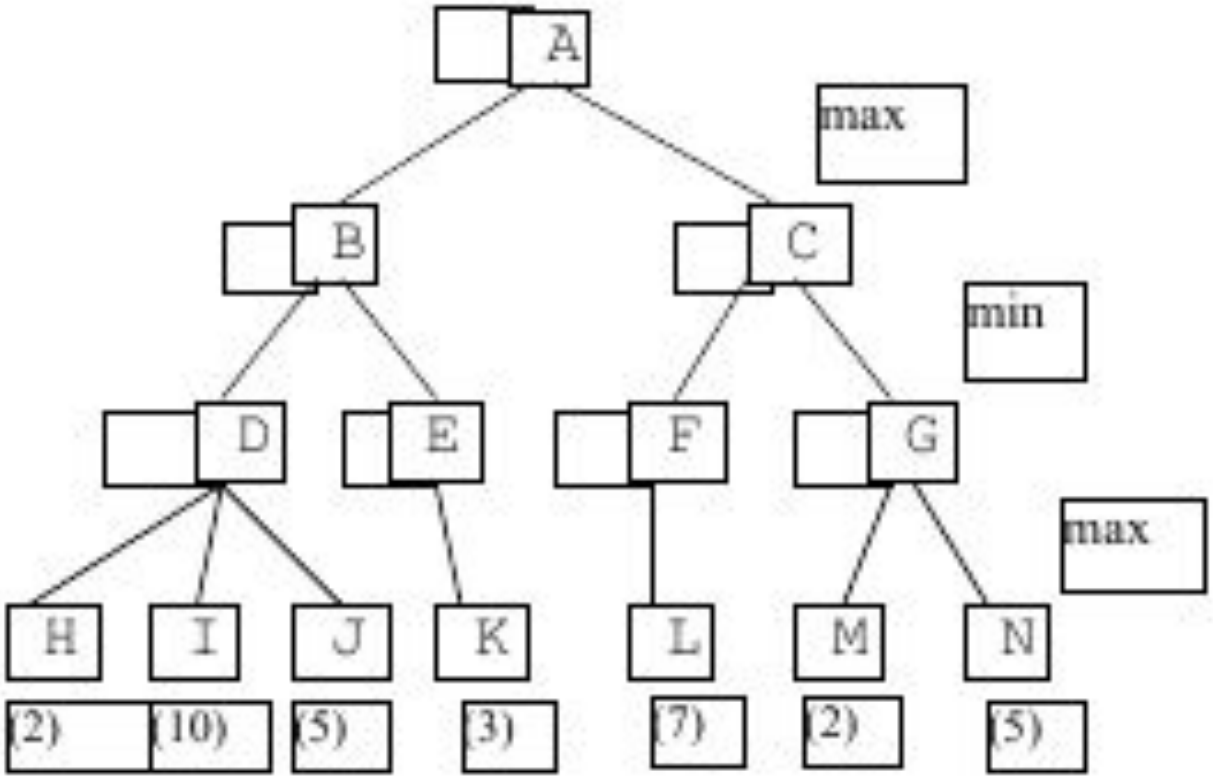
Exemplo - MINIMAX no meio de um jogo da velha



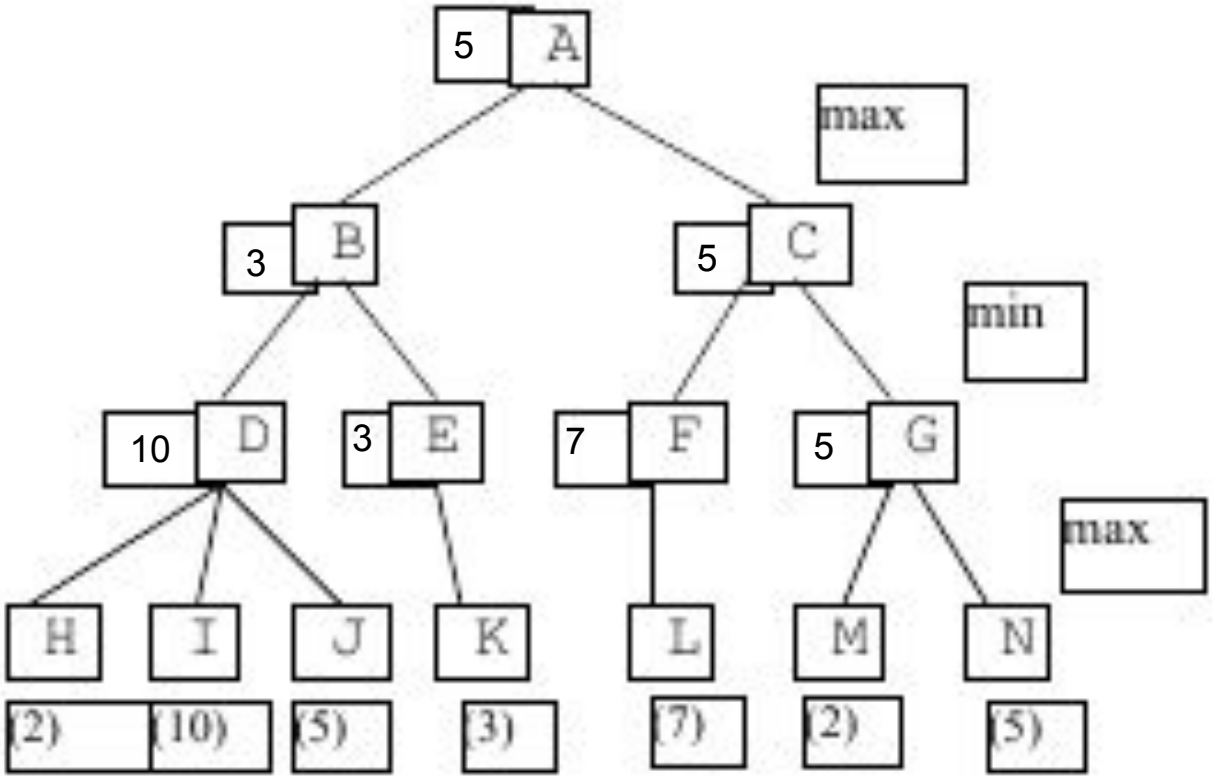
Algoritmo MINIMAX

```
function MINIMAX-DECISION(estado) retorna uma ação
    return arg max a in ACTIONS ( s)  MIN-VALUE(RESULT(state, a))

---

function MAX-VALUE( estado) retorna um  valor de utilidade
    if  TERMINAL-TEST(estado) then return UTILITY(estado)
    v =  - 00
    for each a in ACTIONS(estado) do
        v <-  MAX(v, MIN-VALUE(RESULT(s, a)))
    return v

---

function MIN-VALUE( estado) retorna um valor de utilidade
    if  TERMINAL-TEST(estado) then return UTILITY(estado)
    v = 00
    for each a in ACTIONS(estado) do
        v <-  MIN(v, MAX-VALUE(RESULT(s, a)))
    return v
```

MINIMAX



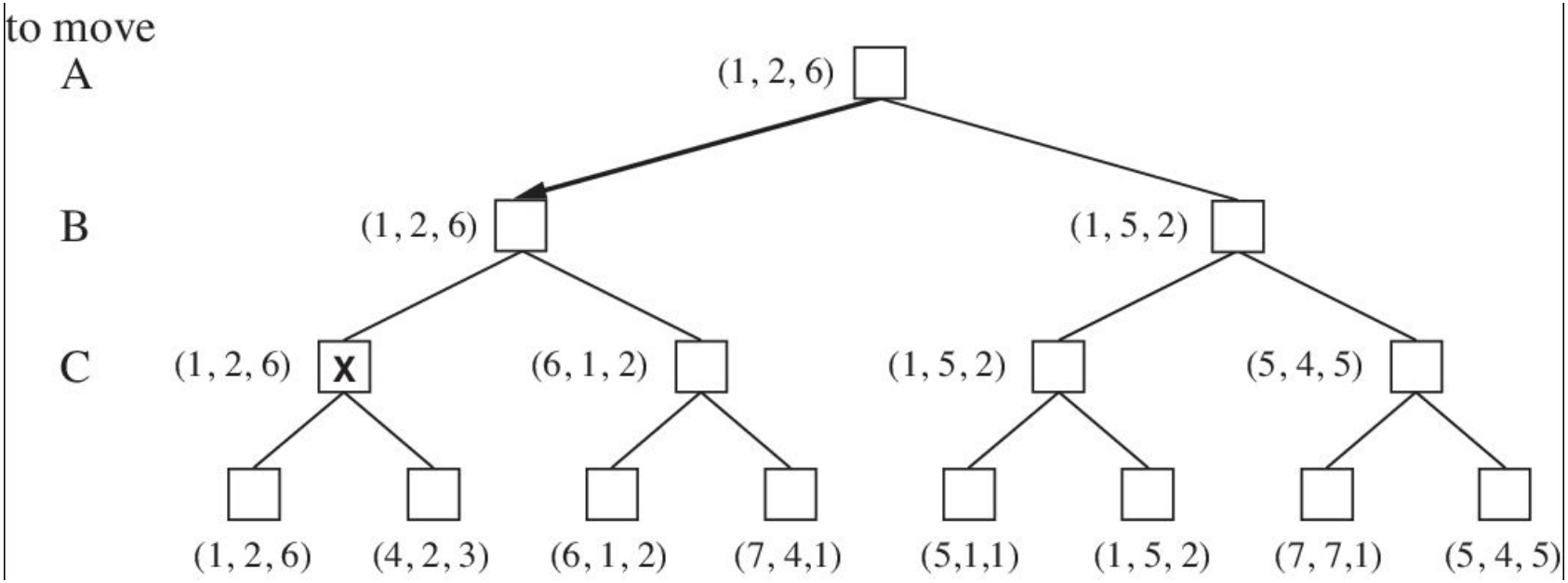
MINIMAX



Jogos com mais de dois jogadores

- Valor de utilidade -> vetor de utilidades
- Utilidade do estado de acordo com o ponto de vista de cada jogador
- Propagação dos valores para os nós acima dependerão de quem estiver jogando

Exemplo - 3 jogadores



Alianças

- Jogos com múltiplos jogadores podem requerer alianças
  - são estabelecidas e quebradas durante o jogo
  - A, B em posições ruins e C em uma boa posição
    - A e B escolhem atacar C ao invés de atacarem um ao outro

MINIMAX

- Busca equivale a profundidade
  - m: profundidade máxima da árvore
  - b: movimentos válidos em cada estado

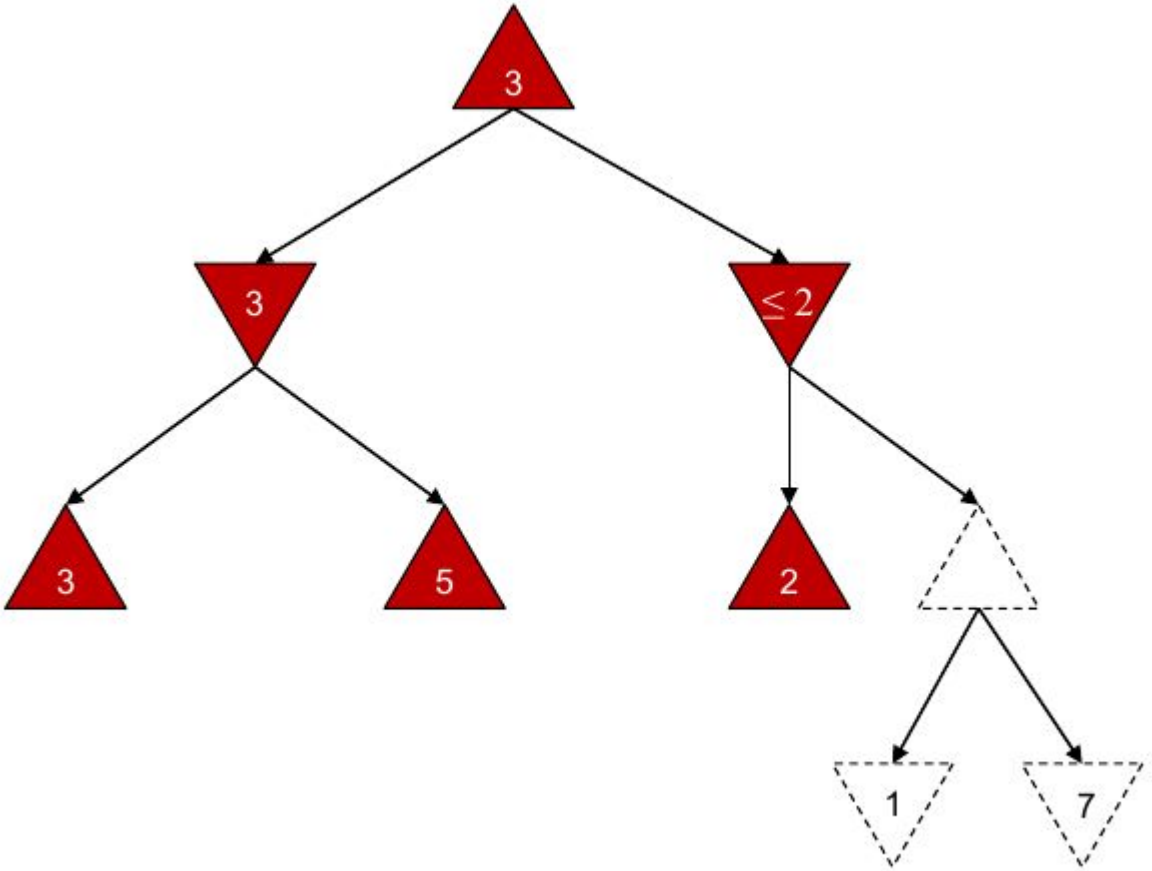
- Complexidade
  - Tempo: $O(b^m)$
  - Espaço? O(bm)

Poda Alfa-Beta

- Objetivo:
  - calcular a decisão correta sem examinar todos os nós da árvore,
  - não precisa avaliar nós que serão irrelevantes para a decisão final

- Alfa - o valor da melhor escolha até agora (maior valor) ao longo do caminho de MAX
- Beta - o valor da melhor escolha até agora (menor valor) ao longo do caminho de MIN

Exemplo: poda alfa-beta

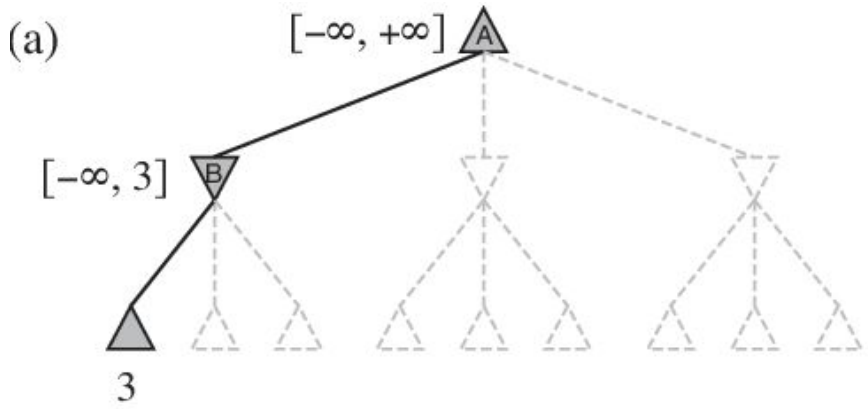


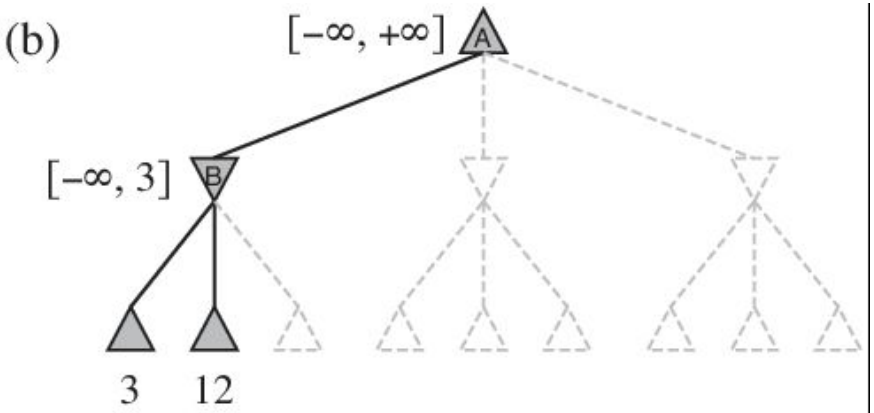
Algoritmo Poda Alfa Beta

```
função  ALPHA-BETA-SEARCH(estado) retorna uma ação
        v = MAX- VALUE(estado, -oo, +oo)
        retorna a ação em  ACTIONS(estado) com valor v

função MAX-VALUE(estado,alfa,beta) retorna um valor de utilidade
        if  TERMINAL-TEST(estado) then retorna UTILITY(estado)
        v =  - 00
        for each a in ACTIONS(estado) do
                v = MAX(v, MIN-VALUE(RESULT(s,a),alfa,beta))
                if v >= beta then retorna v
                alfa =  MAX(alfa, v)
        retorna  v

função MIN-VALUE(estado,alfa,beta) retorna um valor de utilidade
        if  TERMINAL-TEST(estado) then retorna UTILITY(estado)
        v =  + 00
        for each a in ACTIONS(estado) do
                v = MIN(v, MAX-VALUE(RESULT(s,a),alfa,beta))
                if v <= alfa then retorna v
                beta =  MIN(beta, v)
        retorna  v
```

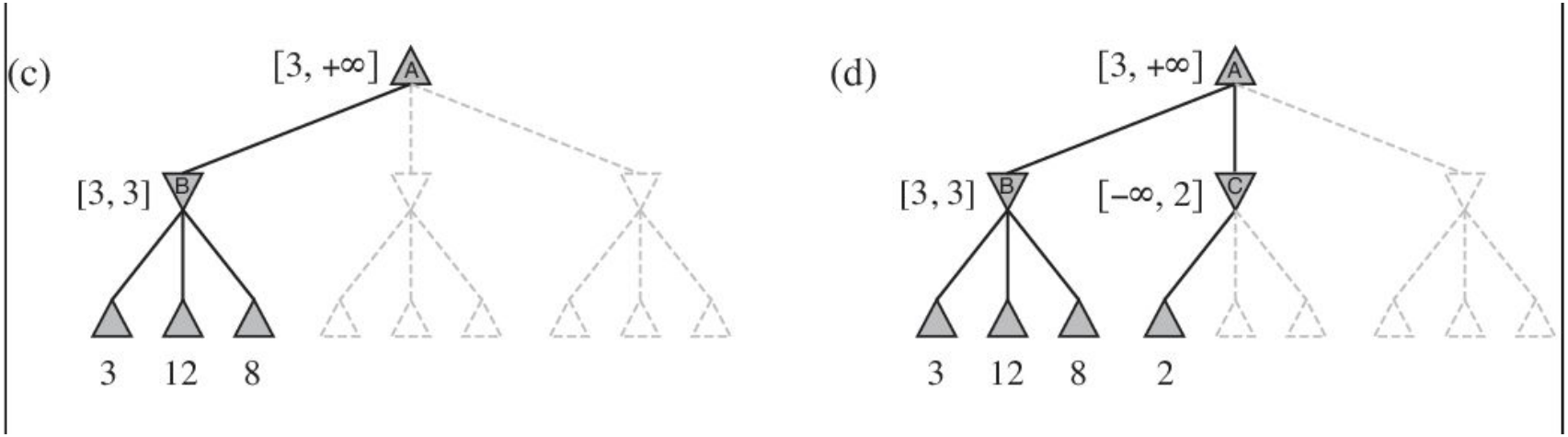
Poda Alfa-Beta

MAX: atualiza alfa
MIN: atualiza beta

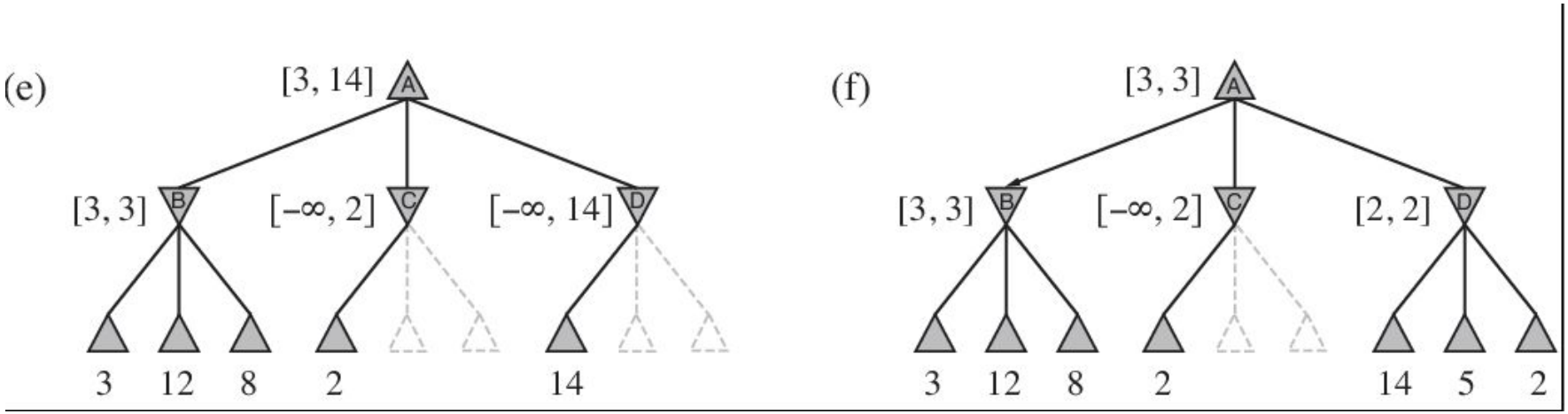




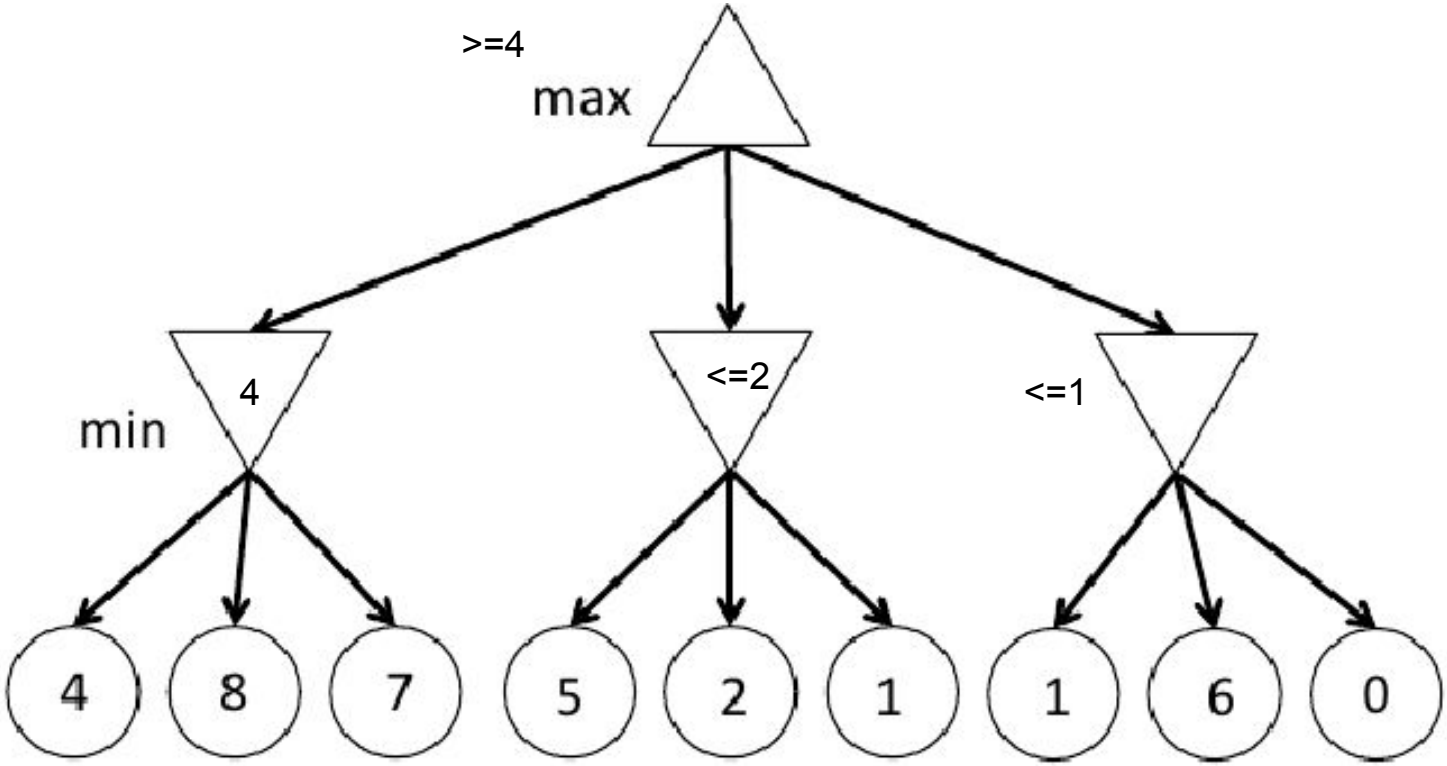
Poda Alfa-Beta



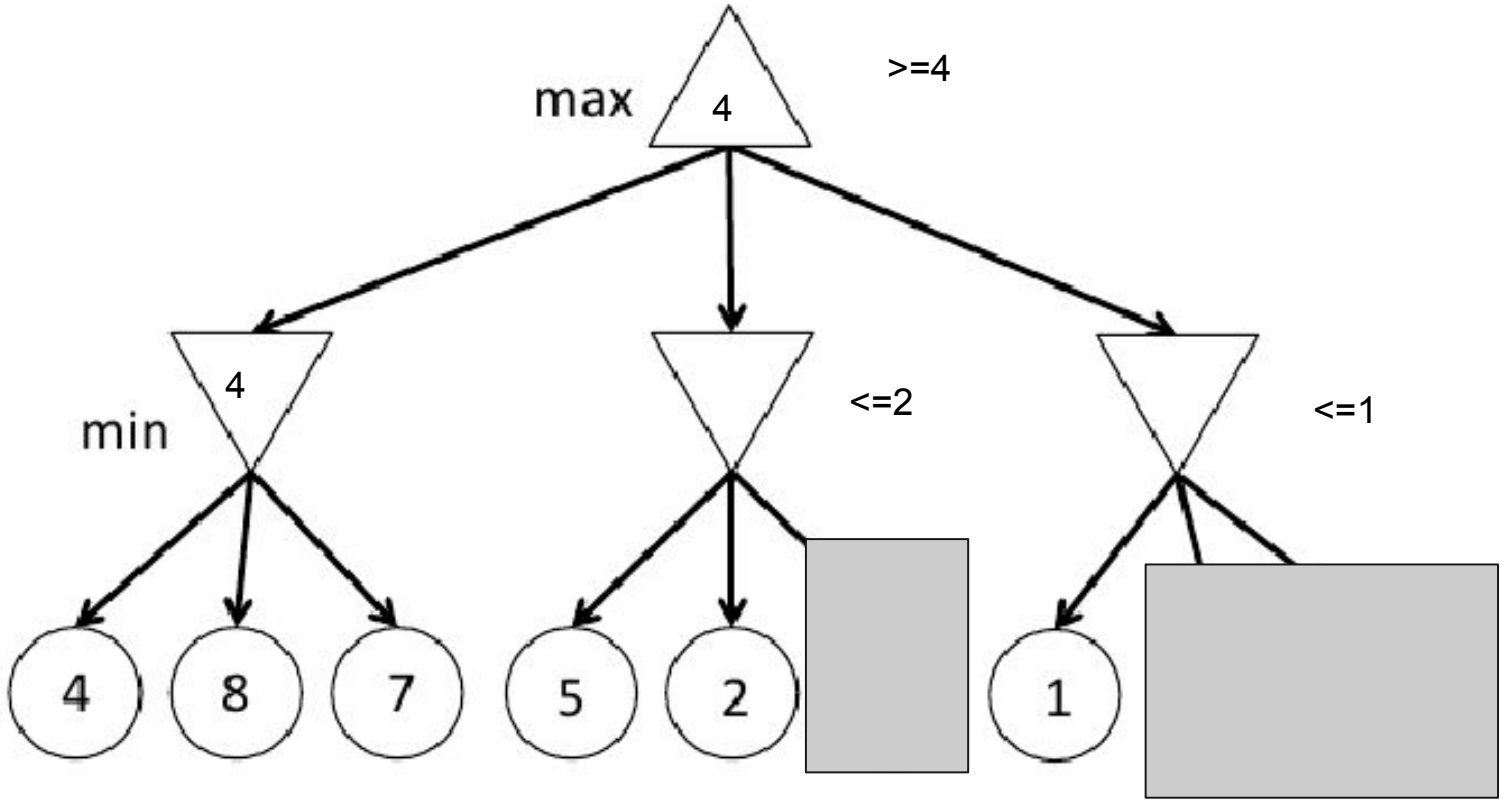
Poda Alfa-Beta



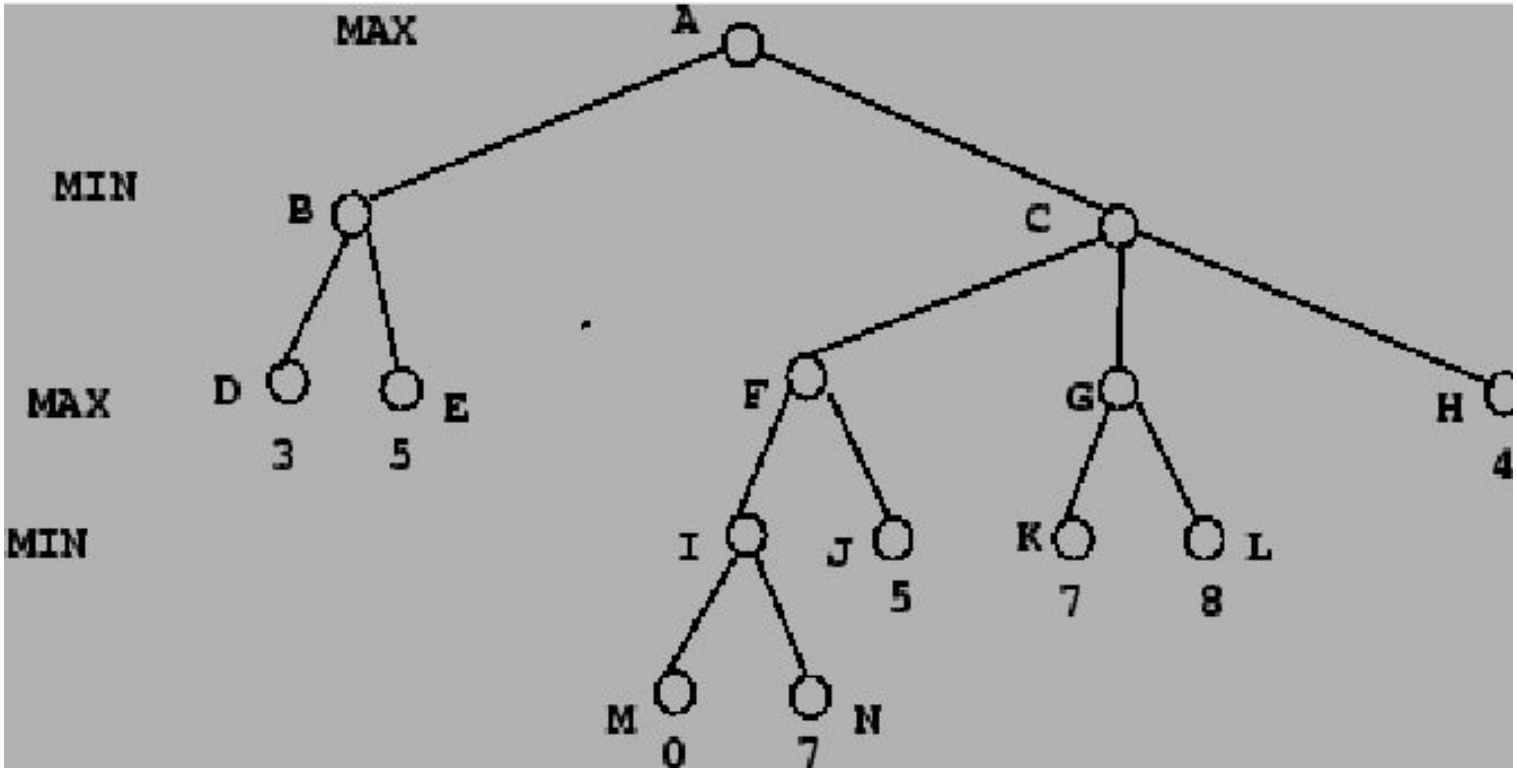
Quais nós podem ser podados?



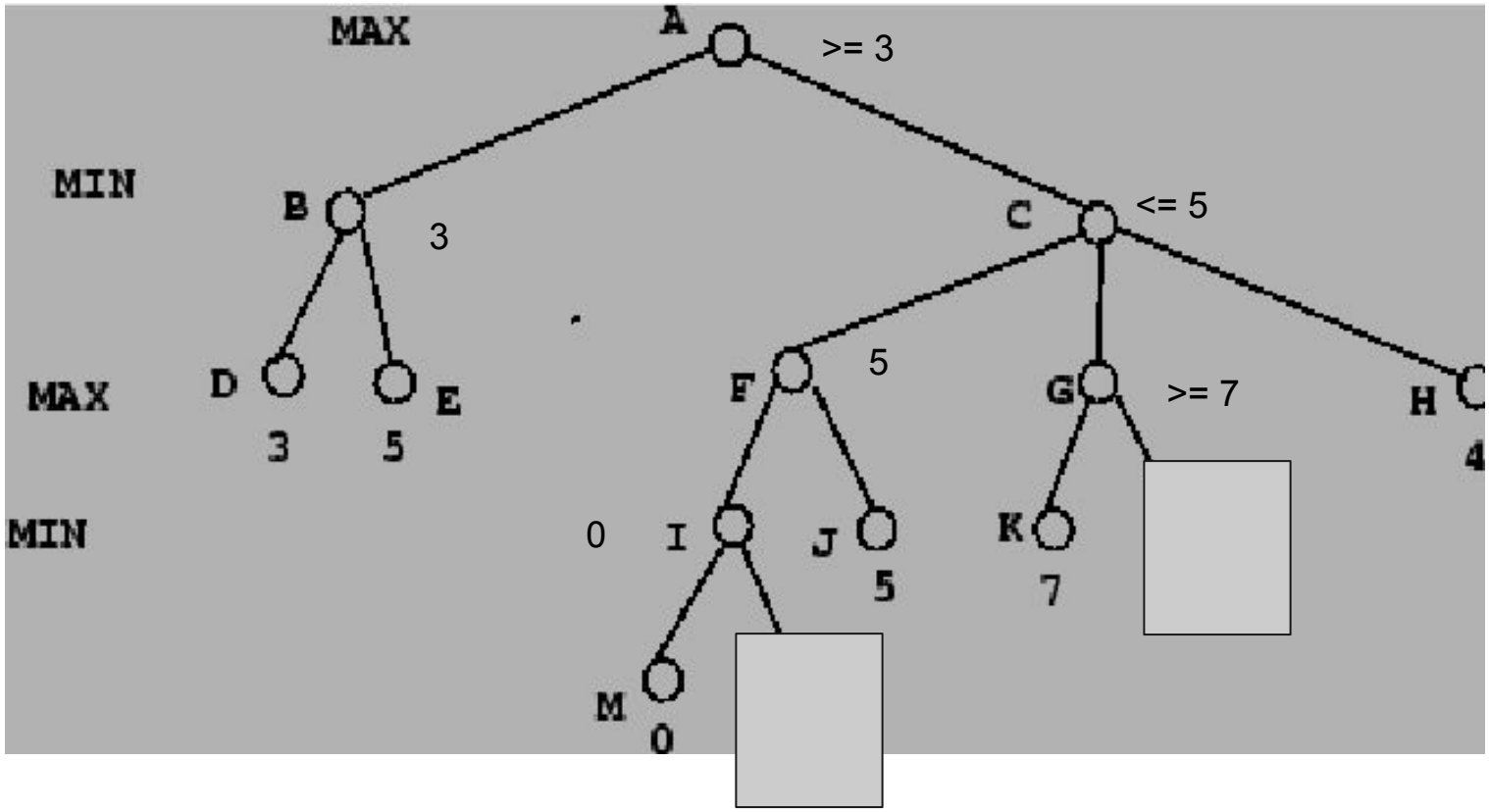
Quais nós podem ser podados?



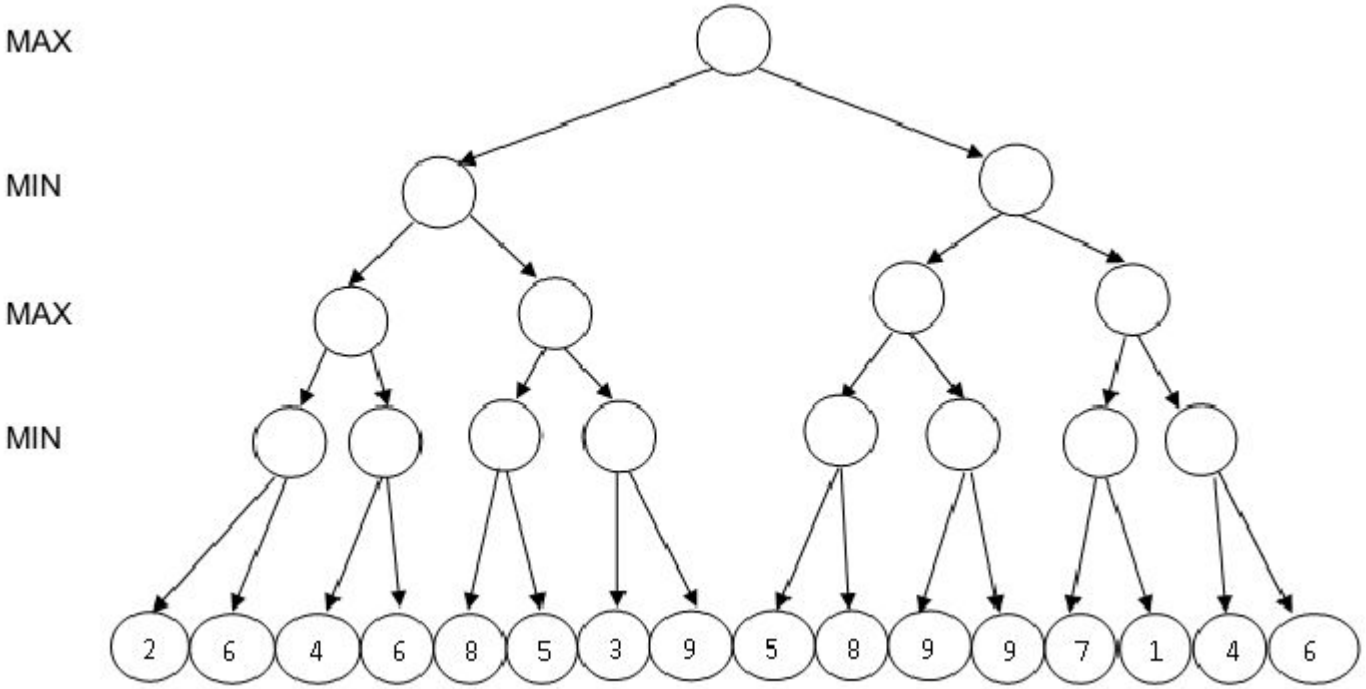
Quais nós podem ser podados?



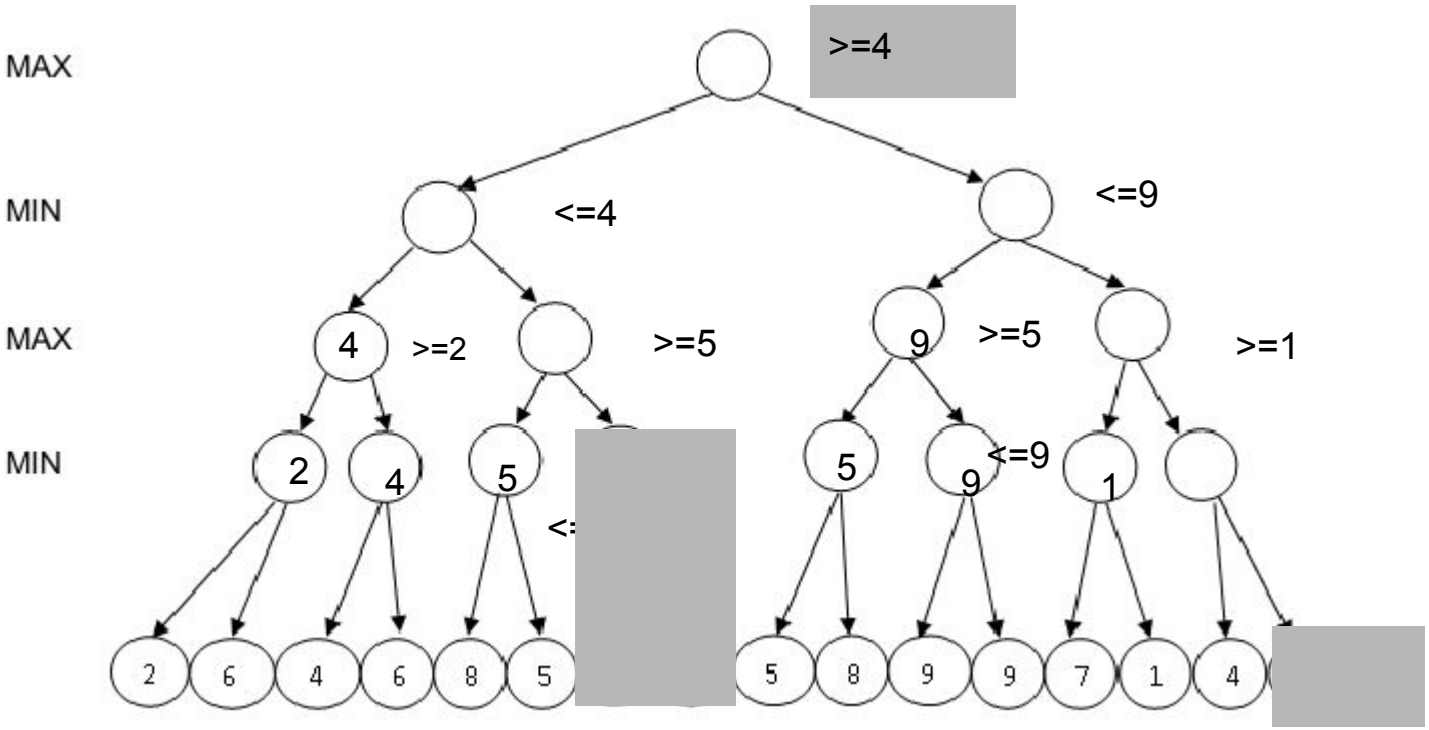
Quais nós podem ser podados?



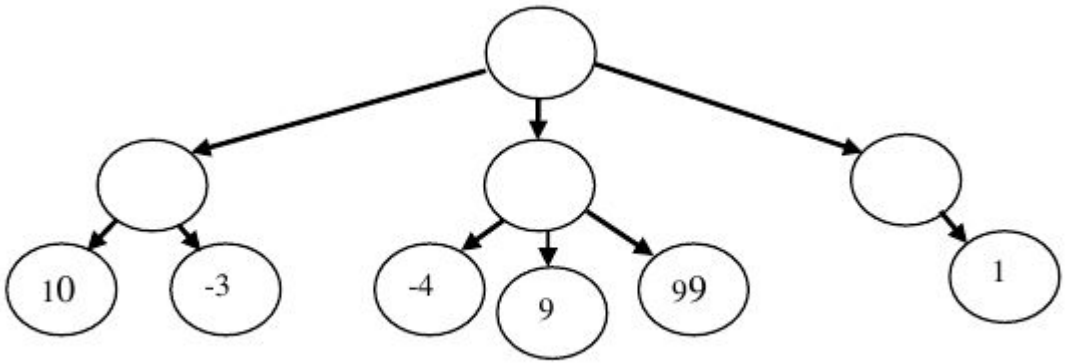
Exercícios: Minimax e poda

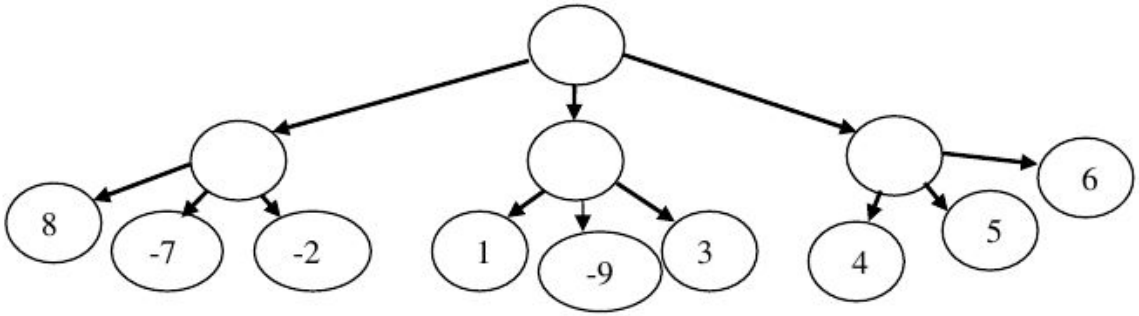


Exercícios: Minimax e poda



Minimax e Alfa-Beta





Minimax e Alfa-Beta