Inteligência Computacional

Busca competitiva
(jogos adversariais)

Prof. Fabio Augusto Faria

Material adaptado de Profa. Ana Carolina Lorena e livro
“Inteligência Artificial, S. Russell e P. Norving”

2º semestre 2021

Jogos

- Entre as primeiras tarefas empreendidas em IA
  - Máquinas:
    - Ultrapassaram humanos em:
      - Damas
      - Othello
    - Derrotaram humanos algumas vezes em:
      - Xadrez
      - Gamão

Jogos

- São domínios clássicos em IA
  - Estruturados: fáceis de formalizar e representar
    - Ações bem definidas
  - Clara definição de sucesso e fracasso
  - Podem ter sua complexidade reduzida ou aumentada
  - Exigem **tomada de decisões**
    - Muitas vezes em um curto intervalo de **tempo**
  - Há interação e pode haver não determinismo (ser estocástico – e.g., gamão)

Exemplo

- Xadrez
  - Fator médio de ramificação = 35
  - Média de 50 movimentos por jogador

  ⇒ árvore de busca com $\mathbf{10^{154}}$ nós
  ⇒ exige habilidade em tomadas de decisão

Jogos x busca clássica

- O oponente é “imprevisível”
  - Dificuldade de levar em consideração todos os movimentos possíveis do oponente
- Limites de **tempo**
  - tomar uma decisão, mesmo que não seja ótima

Decisões em jogos

- Jogos com dois jogadores
  - MIN e MAX (MAX faz inicia o movimento e eles se revezam)
- Jogos
  - Metas em conflito
  - Jogos de revezamento de dois jogadores
    - Soma 0
      - Ex.: jogador que ganha xadrez ⇒ +1
        - » outro necessariamente perde ⇒ -1
      - Gera situação de competição

Busca competitiva

Modelar um jogo como busca

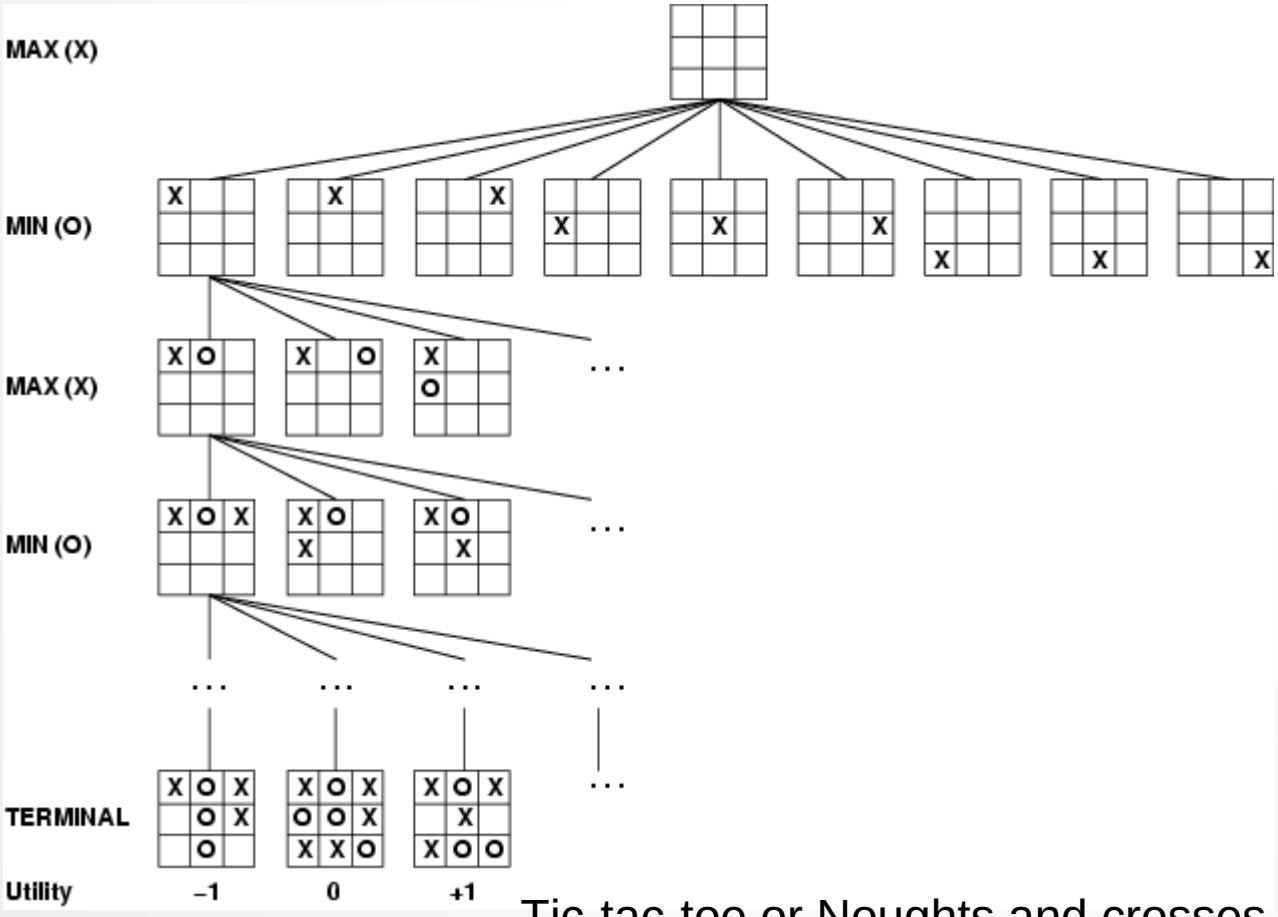
- Formulação:
  - ***Estado inicial*:**
    - Posição no tabuleiro do jogo e qual jogador inicia
  - ***Gerando sucessores*:**
    - Lista de pares (movimento possível, estado resultante)
  - ***Teste de término*:**
    - Determina quando o jogo termina (estados terminais)
  - ***Função de utilidade*** *(objetivo ou compensação)*:
    - Dá valor numérico aos estados terminais
      - Exemplos:
        - Xadrez: vitória = +1; derrota = -1; empate = 0
        - Gamão: 192 a -192

Jogos de soma zero

Aquela que a compensação total é
a mesma para cada instância do jogo

Árvore de jogo

- Árvore de busca de jogo
  - 2 jogadores, determinístico, turnos
  - Mostra todas as possibilidades do jogo



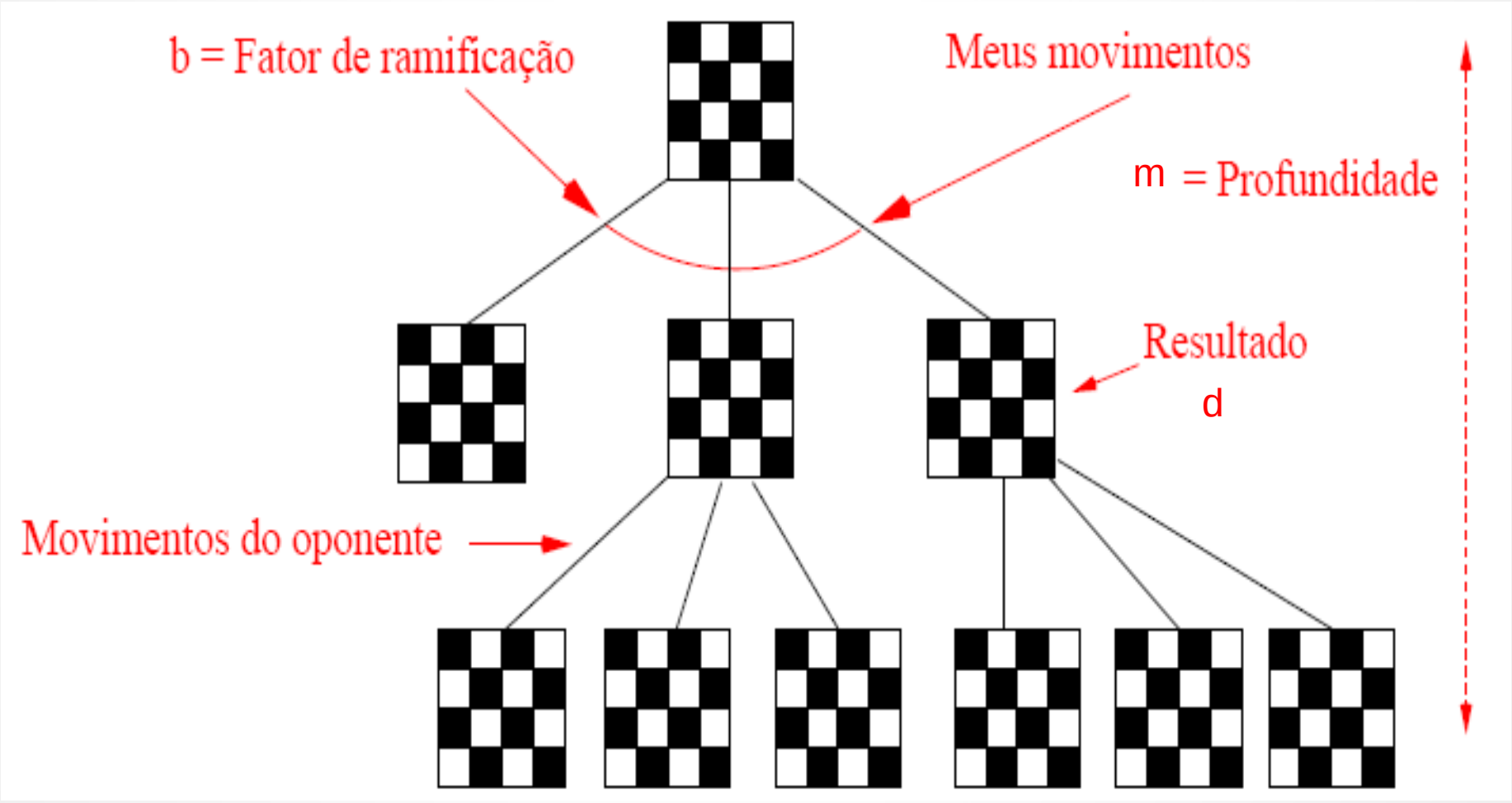
Tic-tac-toe or Noughts and crosses

- MAX = X (jogador)
- MIN = O (adversário)
- Utilidade do ponto de vista de MAX
- Valores altos bons para MAX e ruins para MIN

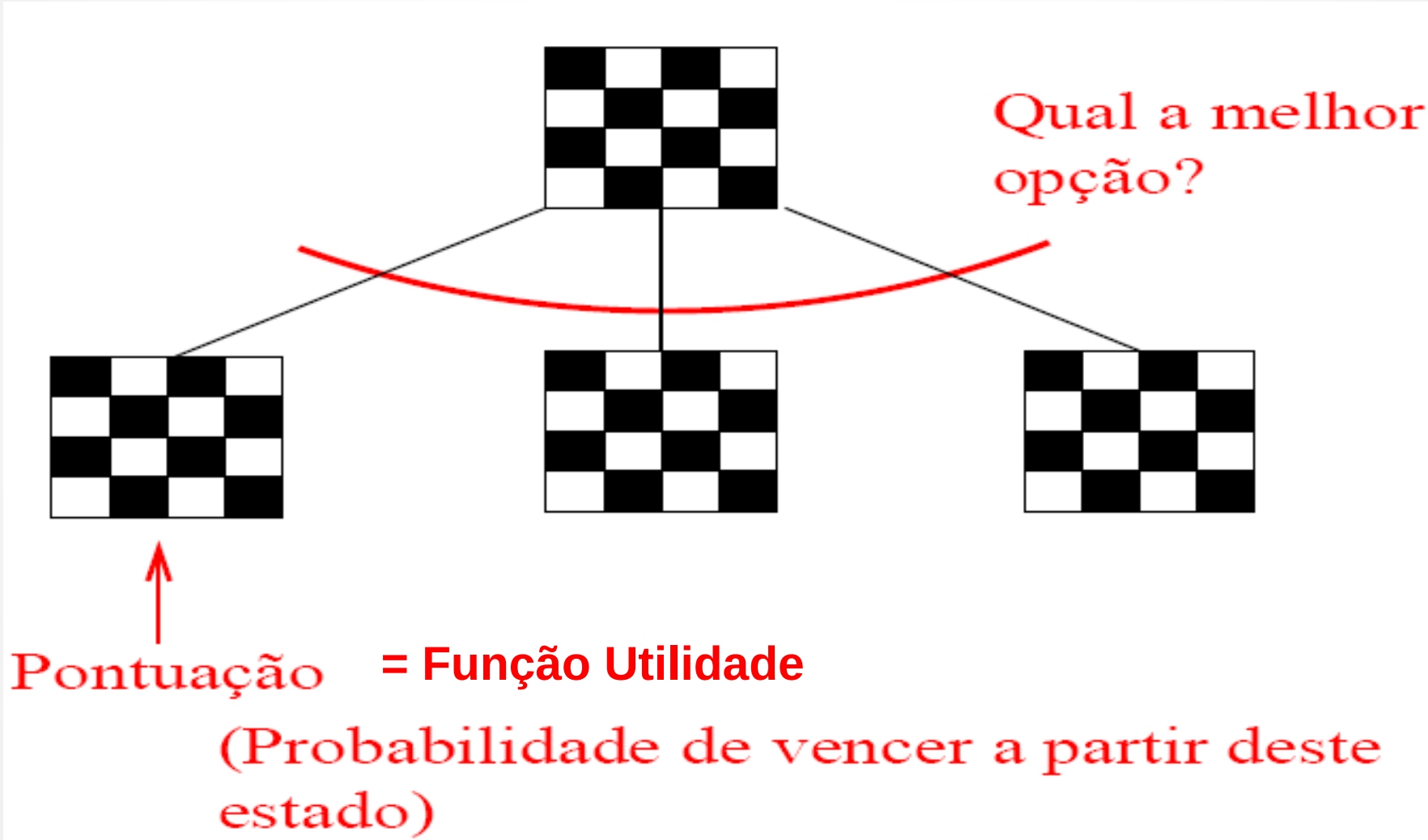
Estratégias de busca

- A solução ótima para MAX depende dos movimentos de MIN, logo:
  - MAX deve encontrar uma ***estratégia*** 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...
  - Procura-se pelo próximo movimento
  - Espera-se que leve à vitória
  - Melhores movimentos dependem dos movimentos do adversário

Jogos

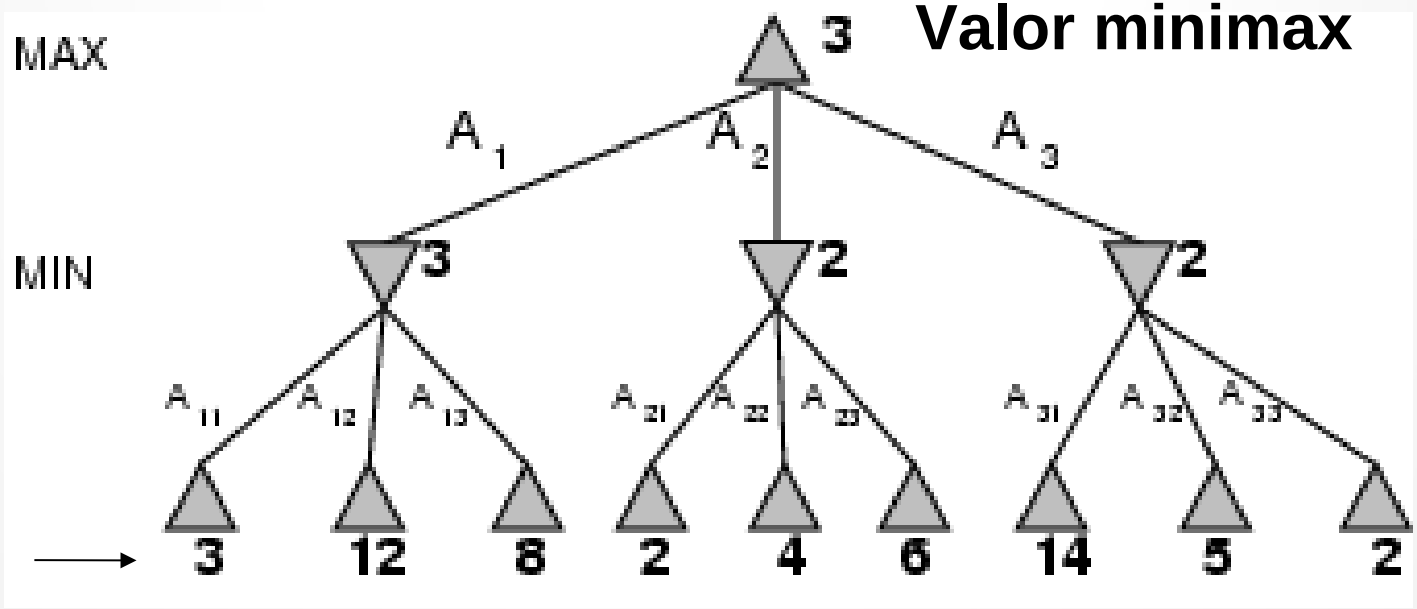

b = Fator de ramificação
Meus movimentos
m = Profundidade
Resultado
d
Movimentos do oponente

Jogos


Qual a melhor opção?
Pontuação
= Função Utilidade
(Probabilidade de vencer a partir deste estado)

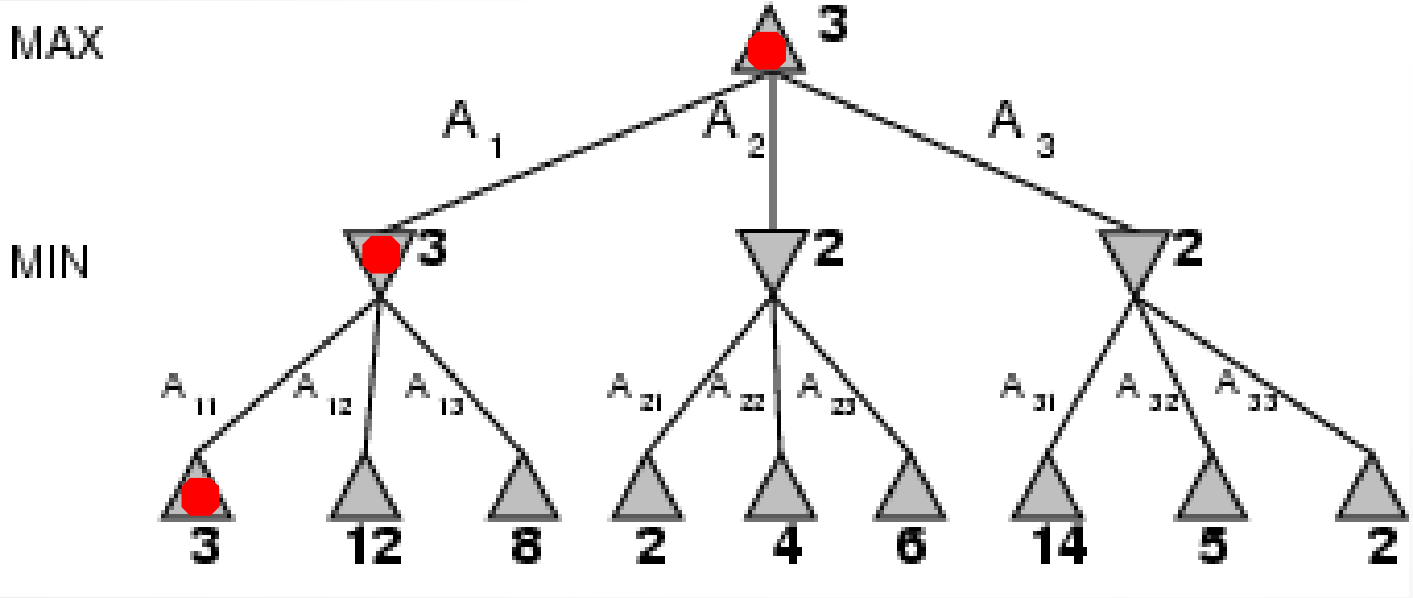
Algoritmo Minimax

- Jogo
  - Deve-se levar em conta o caráter competitivo
    - O que MAX fará é determinado também por MIN
  - Supor o jogo:



Algoritmo Minimax

- **MAX** prefere mover para estado de minimax máximo
- **MIN** prefere valor mínimo
  - Dada uma árvore de jogo, a estratégia ótima pode ser determinada a partir do valor ***minimax*** de cada nó



Ideia: maximizar a utilidade (ganho) supondo que o adversário vai tentar minimizá-la. **Minimax faz busca cega em profundidade**

Valor minimax

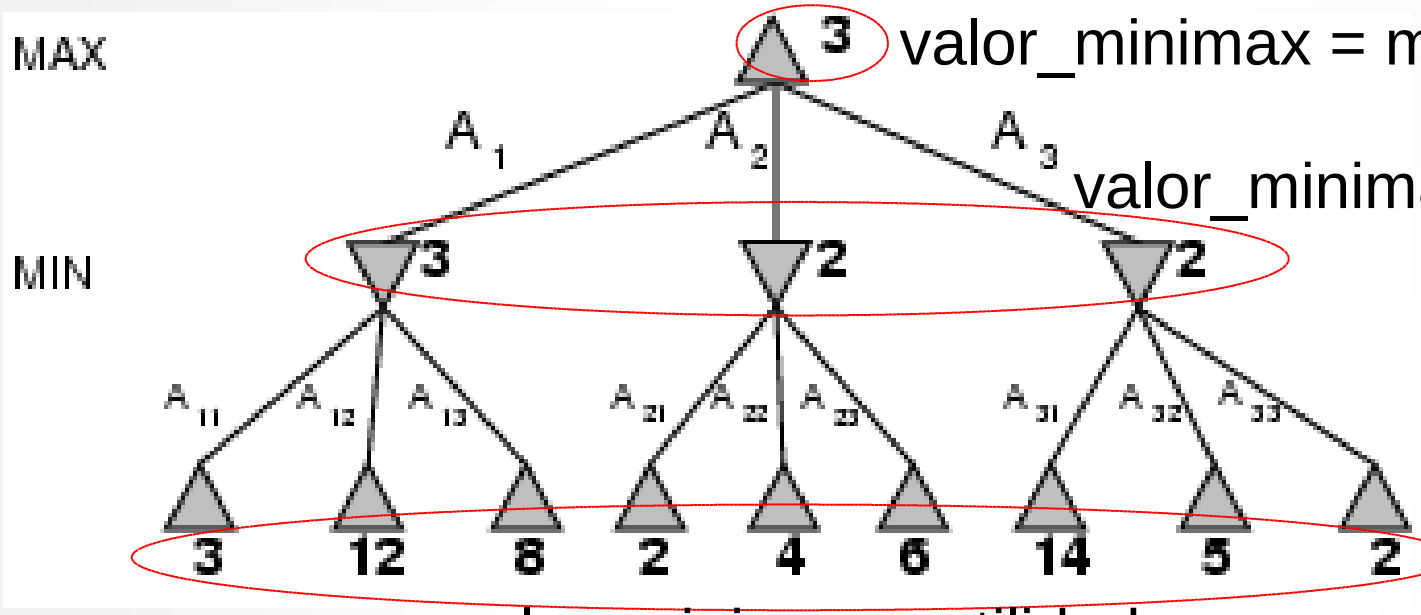
- Valor minimax de um nó é a utilidade de MAX no estado correspondente

Supondo que ambos jogadores têm desempenho ótimo do estado atual até estado terminal (fim do jogo)

$$
\text{Valor\_minimax}(n) = \begin{cases} \text{utilidade}(n) & \text{se } n \text{ é estado terminal} \\ \max_{S \in \text{Sucessor}(n)} \text{Valor\_minimax}(n) & \text{se } n \text{ é um nó MAX} \\ \min_{S \in \text{Sucessor}(n)} \text{Valor\_minimax}(n) & \text{se } n \text{ é um nó MIN} \end{cases}
$$

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 estados com menor valor utilidade para MAX)

Valor minimax



valor_minimax = max sucessores

valor_minimax = min sucessores

valor_minimax = utilidade

A ação *A1* é a escolha ótima para MAX, porque leva ao sucessor com mais alto valor minimax

É a melhor jogada para um jogo determinístico assumindo o melhor oponente

Algoritmo Minimax

- Exploração completa em profundidade da árvore de jogo
  - Calcula recursivamente valores de utilidade
  - E toma decisão com base nesses valores

Seja **m** a profundidade máxima da árvore
b o número de movimentos possíveis em cada ponto

Complexidade de tempo **= $O(b^m)$**
Complexidade de espaço **= O(bm)**
(ou O(m) se gera um sucessor por vez)

Algoritmo Minimax

Passos:

1. Gera a árvore inteira até os estados terminais
2. Aplica a função de utilidade nas folhas
3. Propaga os valores dessa função subindo a árvore através do minimax
4. Determinar qual valor que será escolhido por MAX

- Jogada perfeita para jogos determinísticos
- Ideia: escolher movimento para posição com máximo valor minimax
  - melhor alcançável contra melhor jogador

Algoritmo Minimax

```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

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

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

Ideia simplificada:

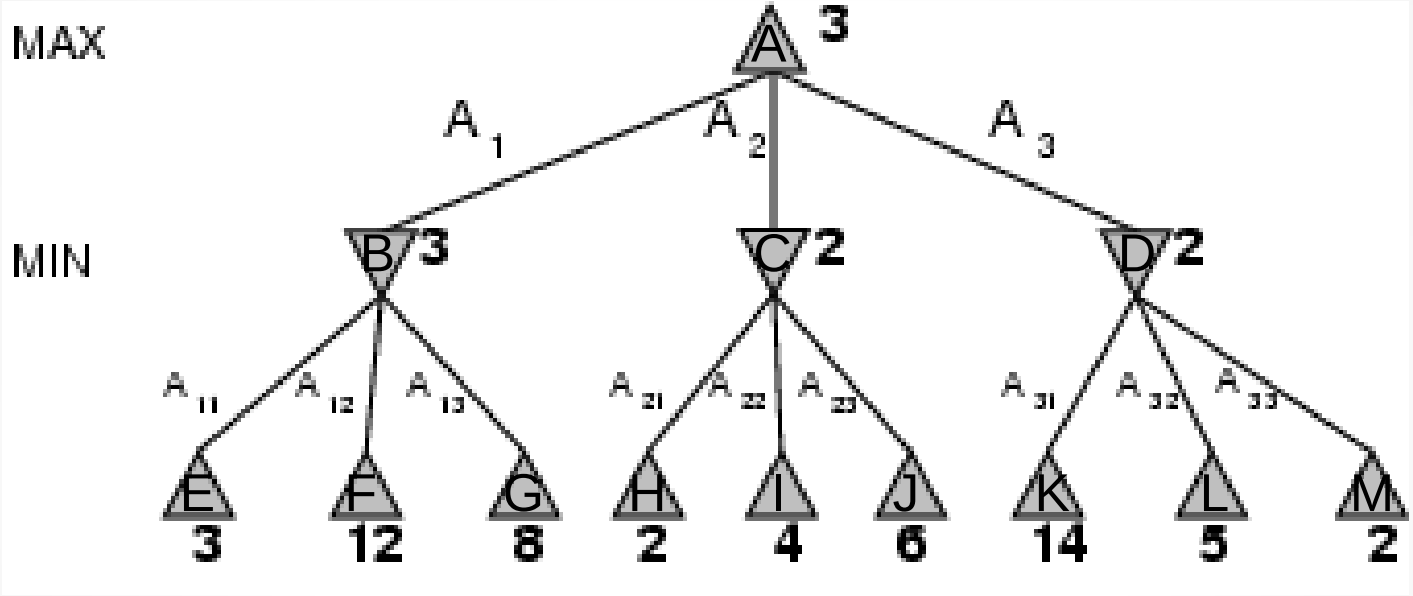
1. Expandir a árvore inteira abaixo da raiz

2. Avaliar os nós terminais como ganhos/ perdas para o MAX (maximizer) – utilidade

3. Selecionar um nó sem utilidade, *n*, que tenha todos os filhos já com valor. Se não há um nó desses, a busca terminou: retornar o valor da raiz.

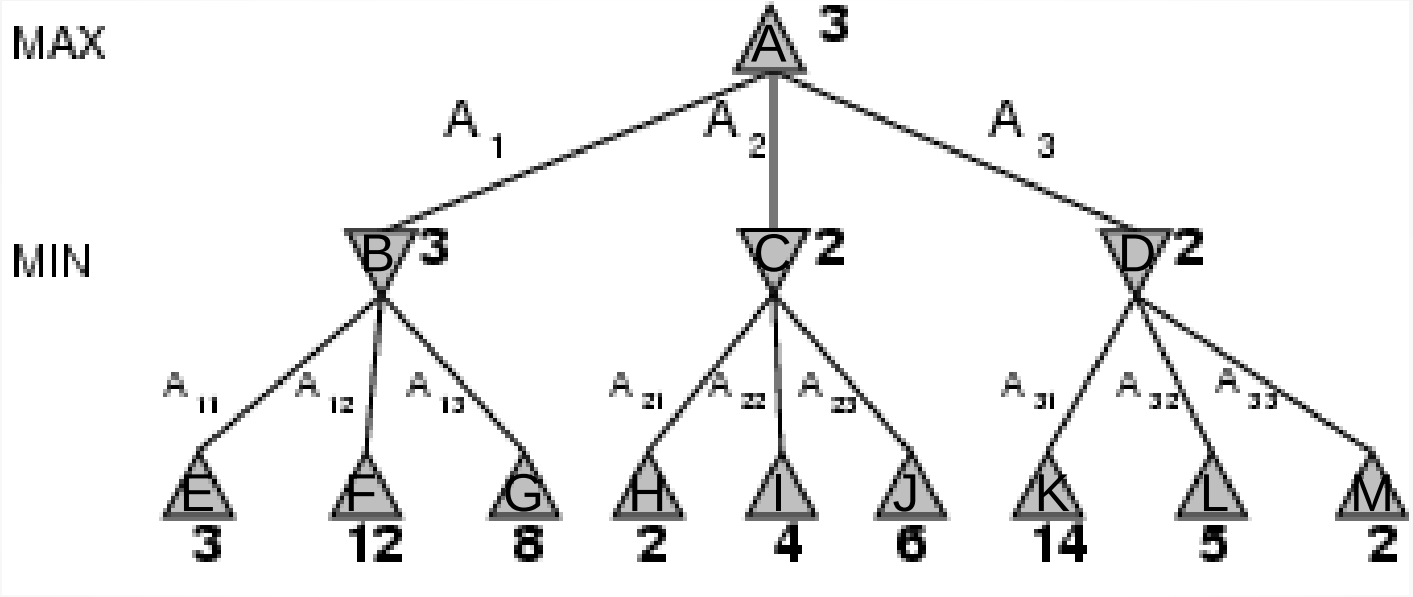
4. Se *n* é movimento *MIN*, atribuí-lo um valor que é o mínimo dos valores de seus filhos.
Se *n* é MAX, atribuí-lo um valor que é o máximo dos valores dos seus filhos.
Retornar ao Passo 3.

Algoritmo Minimax



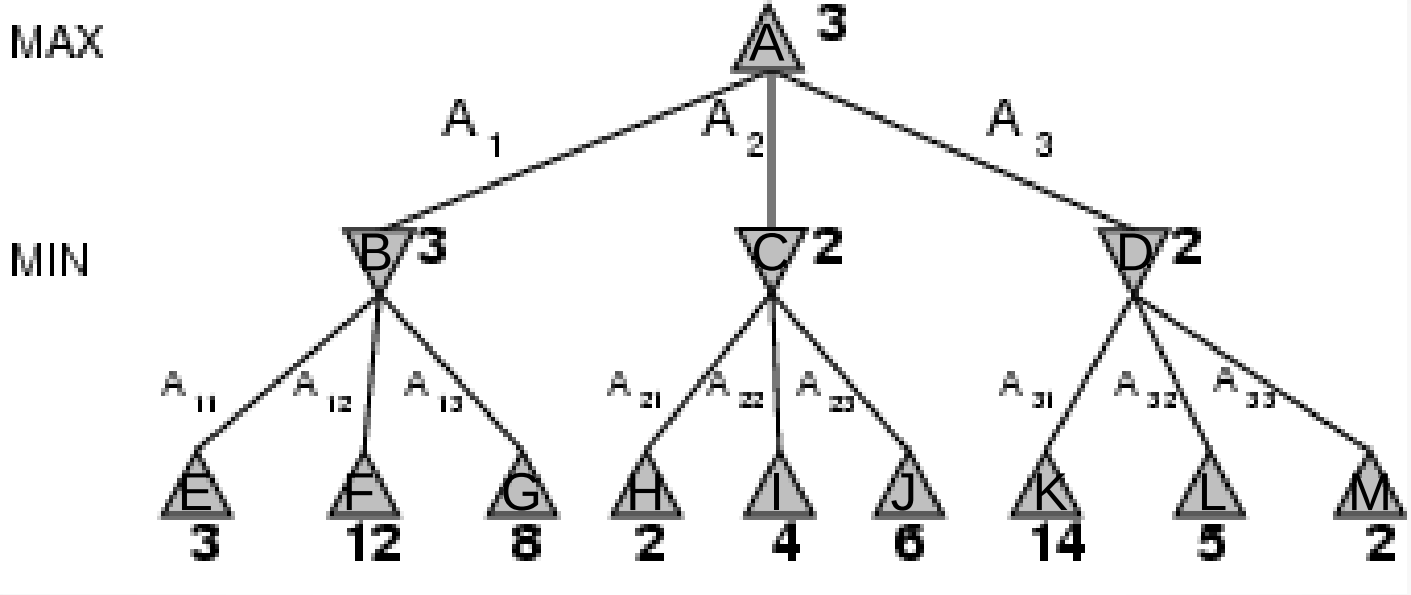
- MINIMAX-DECISION(A)
  - V = MAX-VALUE(A)
- MAX-VALUE(A)
  - A é terminal? Não ⇒ v = -inf
  - Para s = B, C e D
    - V = MAX(v, MIN-VALUE(s))
- MIN-VALUE(B)
  - B é terminal? Não ⇒ v = +inf
  - Para s = E, F e G
    - V = MIN(v, MAX-VALUE(s))
    - V = MIN(+inf, 3, 12, 8) = 3
- MAX-VALUE(E)
  - E é terminal? Sim
    - v = UTILITY(E) = 3
- MAX-VALUE(F)
  - F é terminal? Sim
    - v = UTILITY(F) = 12
- MAX-VALUE(G)
  - G é terminal? Sim
    - v = UTILITY(G) = 8

Algoritmo Minimax



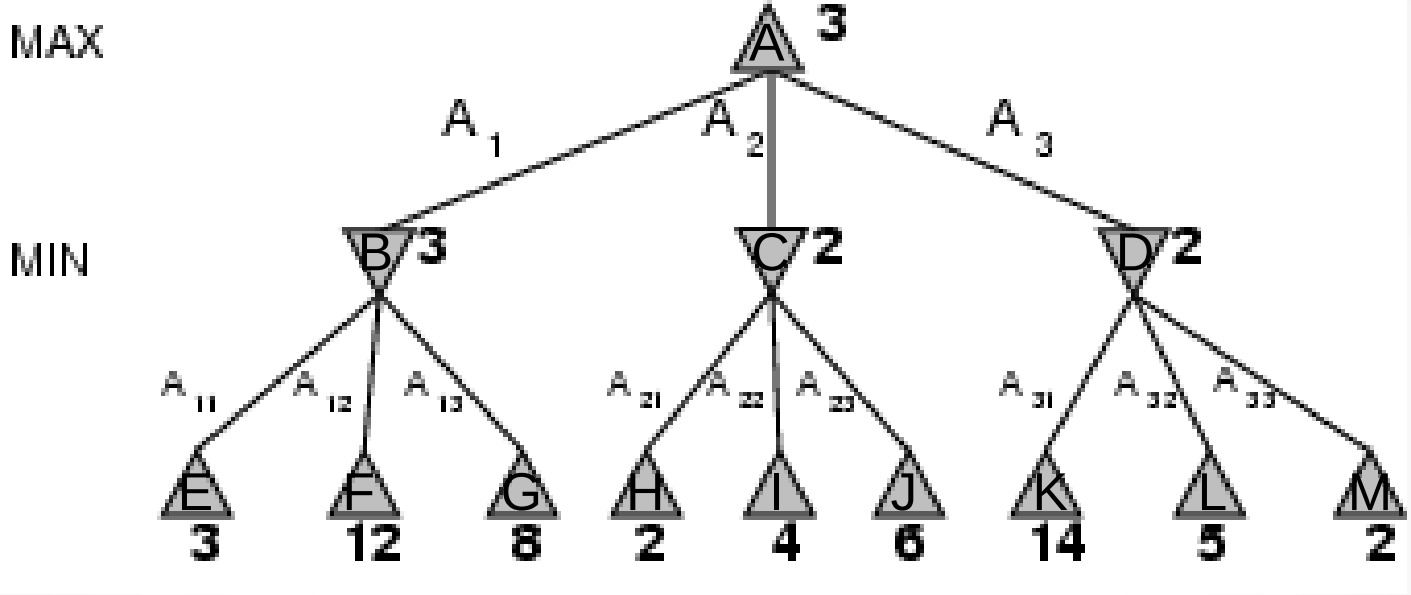
- MINIMAX-DECISION(A)
  - V = MAX-VALUE(A)
- MAX-VALUE(A)
  - A é terminal? Não ⇒ v = -inf
  - Para s = B, C e D
    - V = MAX(v, MIN-VALUE(s))
- MIN-VALUE(C)
  - C é terminal? Não ⇒ v = +inf
  - Para s = H, I e J
    - V = MIN(v, MAX-VALUE(s))
    - V = MIN(+inf,2, 4, 6) = 2
- MAX-VALUE(H)
  - H é terminal? Sim
    - v = UTILITY(H) = 2
- MAX-VALUE(I)
  - I é terminal? Sim
    - v = UTILITY(I) = 4
- MAX-VALUE(J)
  - J é terminal? Sim
    - v = UTILITY(J) = 6

Algoritmo Minimax



- MINIMAX-DECISION(A)
  - V = MAX-VALUE(A)
- MAX-VALUE(A)
  - A é terminal? Não ⇒ v = -inf
  - Para s = B, C e D
    - V = MAX(v, MIN-VALUE(s))
- MIN-VALUE(D)
  - D é terminal? Não ⇒ v = +inf
  - Para s = K, L e M
    - V = MIN(v, MAX-VALUE(s))
    - V = MIN(+inf,14, 5, 2) = 2
- MAX-VALUE(K)
  - K é terminal? Sim
    - v = UTILITY(K) = 14
- MAX-VALUE(L)
  - L é terminal? Sim
    - v = UTILITY(L) = 5
- MAX-VALUE(M)
  - M é terminal? Sim
    - v = UTILITY(M) = 2

Algoritmo Minimax



- MINIMAX-DECISION(A)
  - V = MAX-VALUE(A) = 3
  - RETORNA A1
- MAX-VALUE(A)
  - A é terminal? Não ⇒ v = -inf
  - Para s = B, C e D
    - V = MAX(v, MIN-VALUE(s))
    - V = MAX(-inf,3,2,2) = 3

Desempenho Minimax

- Completo? Sim
  - Se árvore é finita
- Ótimo? Sim
  - Contra um oponente ótimo
- Complexidade de tempo? $O(b^m)$
  - m = profundidade máxima
  - b = movimentos válidos em cada estado
- Complexidade de espaço? O(bm)
  - Com busca em profundidade
  - Ou O(m) se gerar um sucessor por vez

Para xadrez, b ≈ 35, m ≈100
em jogos “razoáveis”
⇒ Solução exata é inviável

Poda alfa-beta

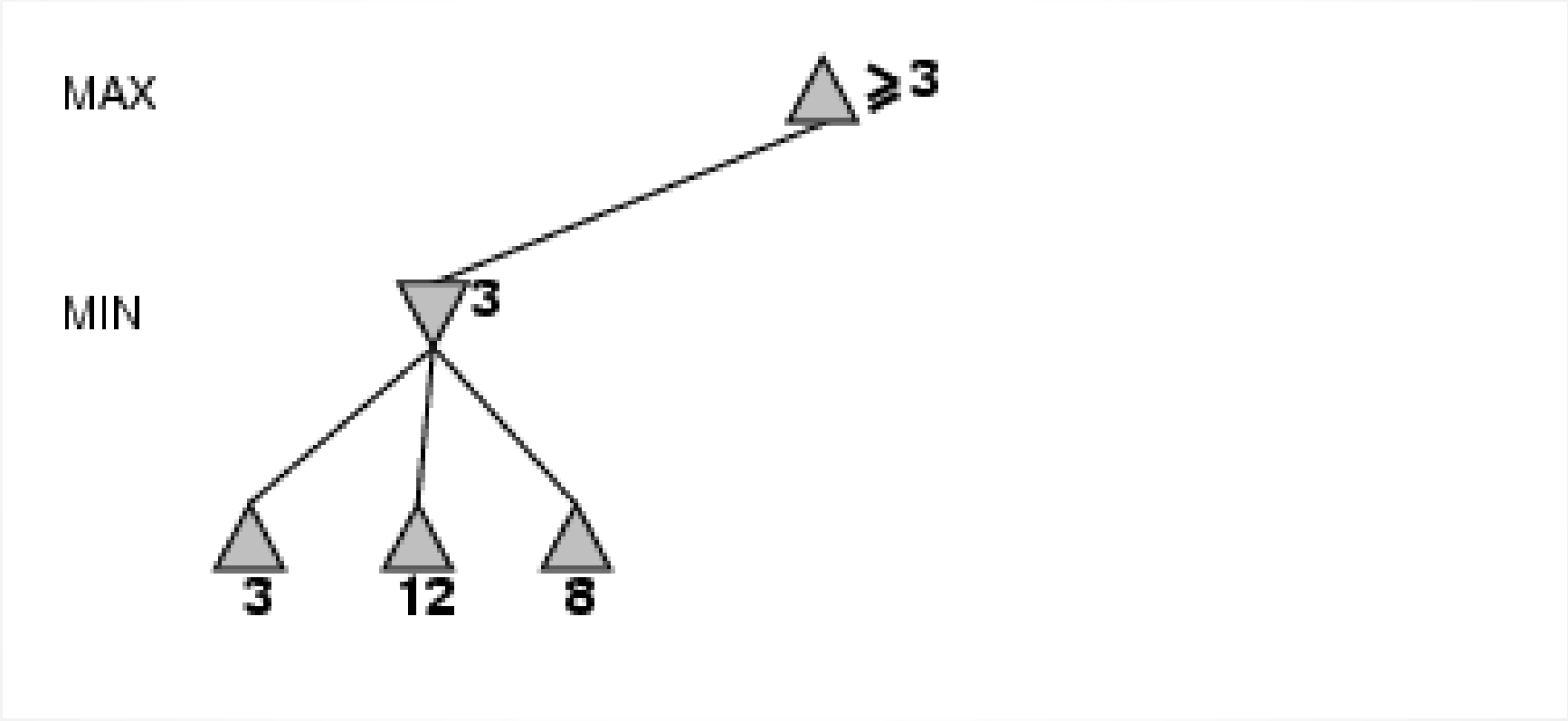
- Minimax é impraticável para muitos jogos
  - Número de estados do jogo a examinar é exponencial
  - É possível reduzir expoente

- **Poda (*Pruning*)**
  - Deixar de considerar grandes partes da árvore de jogo
  - Podando ramificações que não influenciam a decisão final

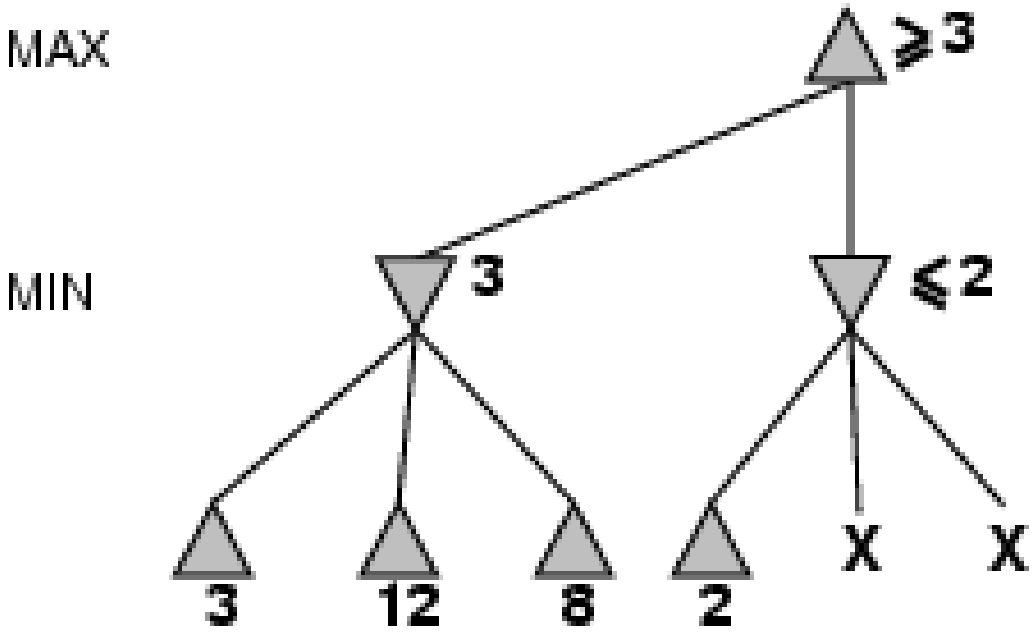
Calcular a decisão correta sem examinar todos os nós da árvore (evitar gerar toda a árvore, analisando que subárvores não influenciam na decisão)

Retorna o mesmo que minimax, porém sem percorrer todos os estados.

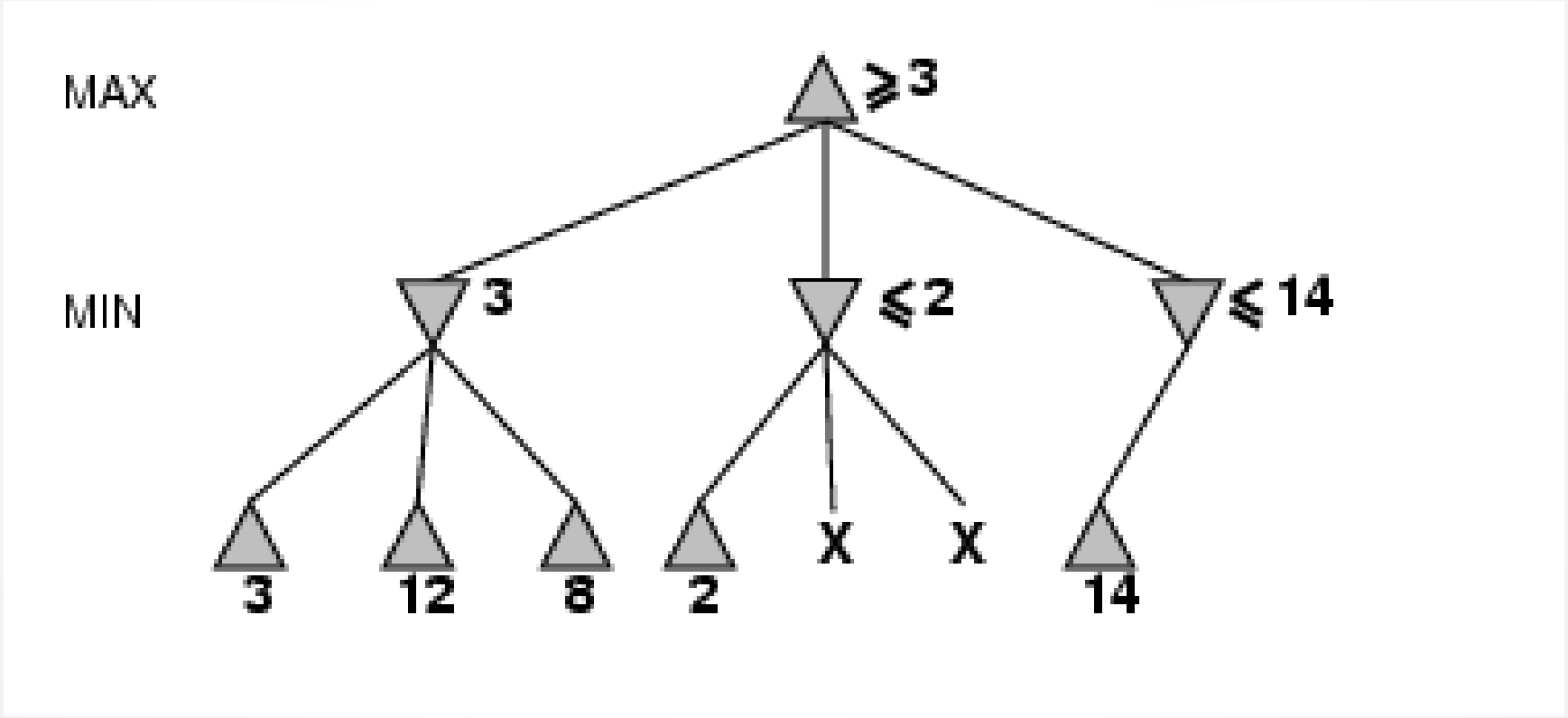
Poda alfa-beta



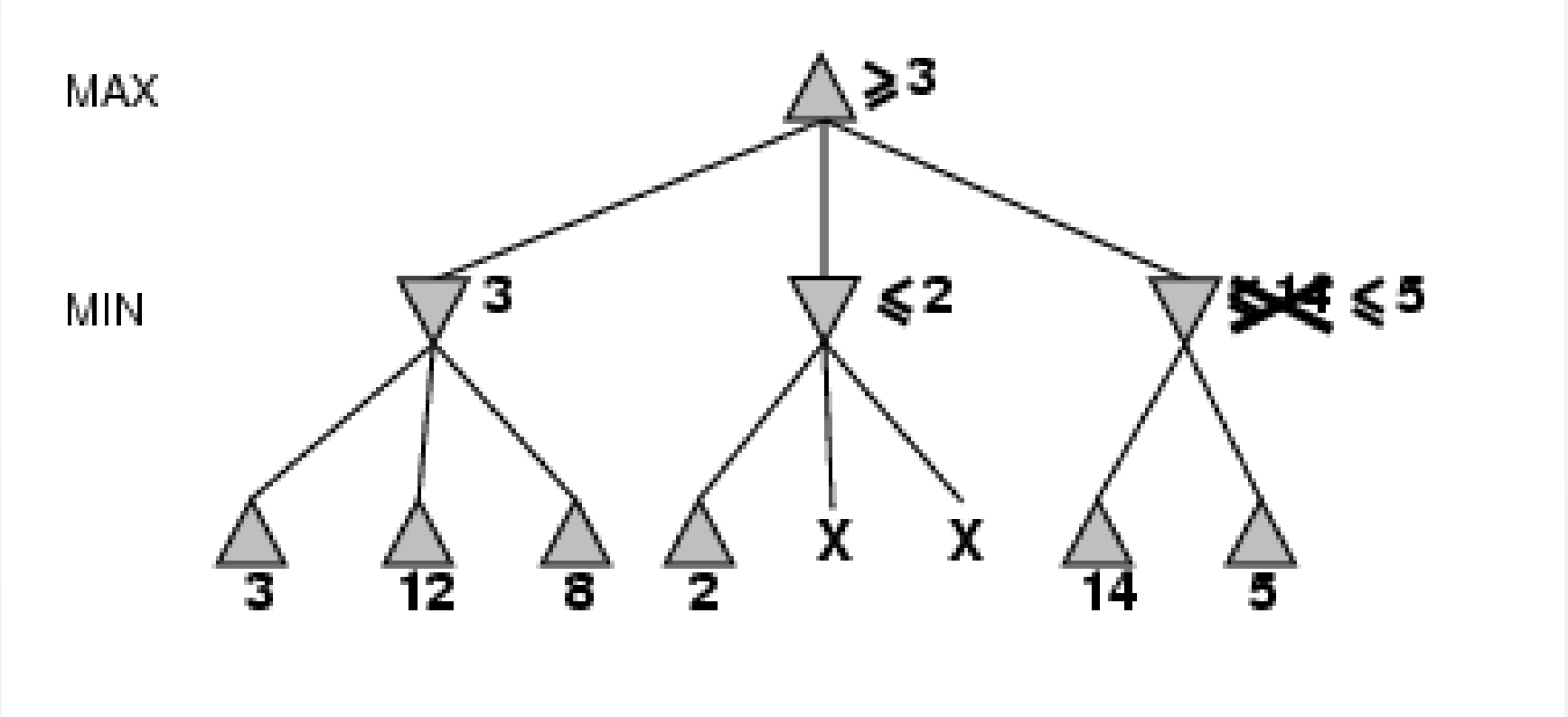
Poda alfa-beta



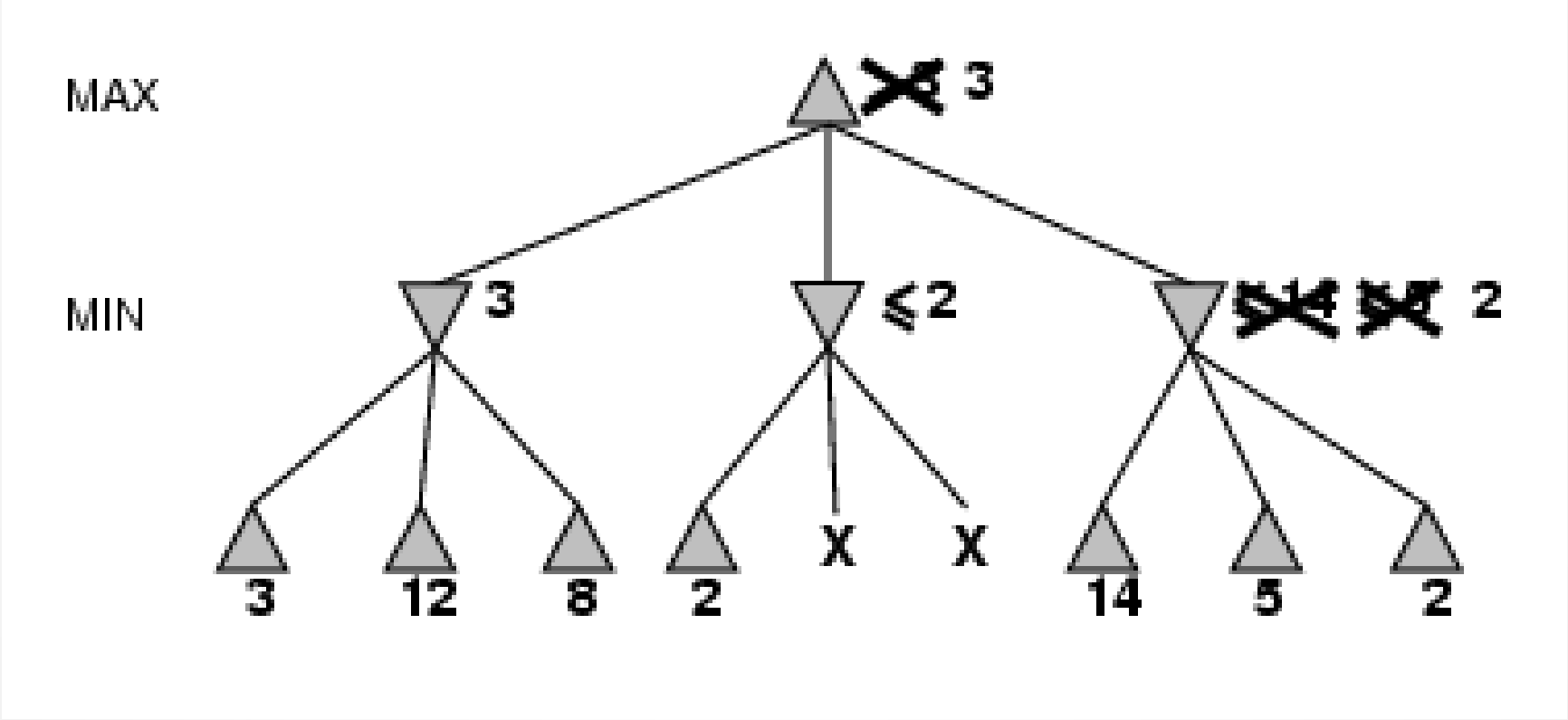
Poda alfa-beta



Poda alfa-beta



Poda alfa-beta



Poda alfa-beta

- Alfa
  - Valor da melhor escolha encontrado em qualquer ponto ao longo do caminho de busca para MAX
    - Valor mais alto
- Beta
  - Valor da melhor escolha encontrado em qualquer ponto de escolha do caminho para MIN
    - Valor mais baixo

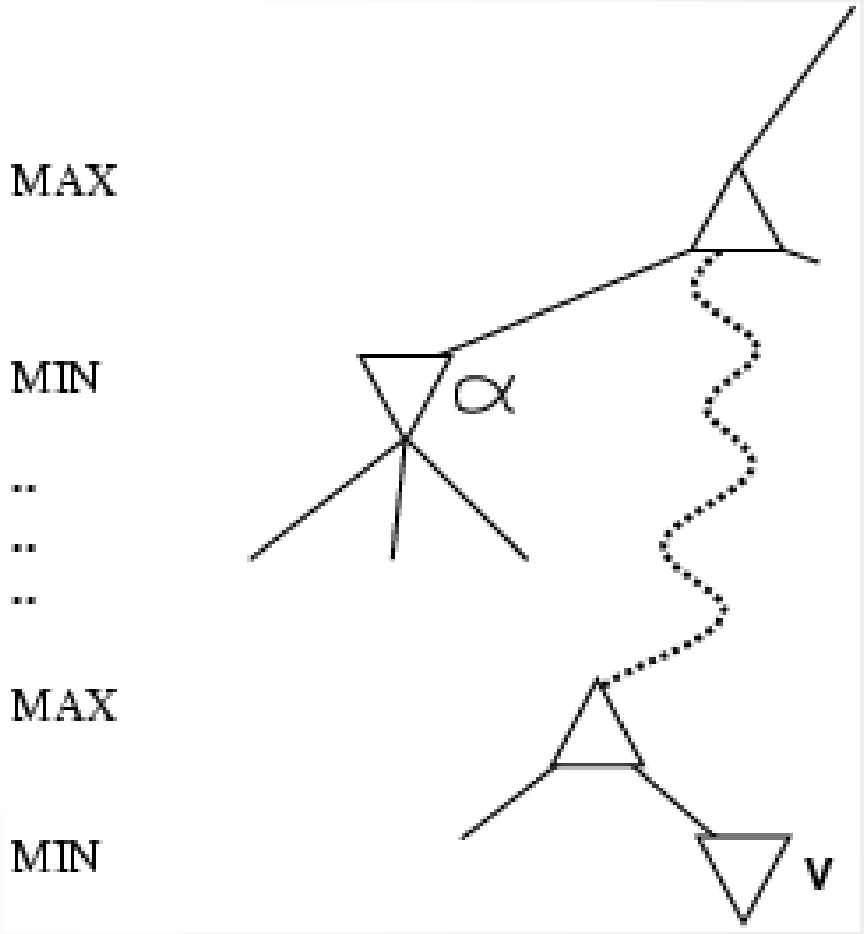
Poda alfa-beta

- **α** é o valor da melhor escolha encontrado até então para qualquer ponto de escolha de ***MAX***
  - Se ***v*** é pior que **α**, ***MAX*** irá evitá-lo
  - → poda o ramo e não percorre este caminho mais

Se **α** é melhor que **v** para o jogador, nunca chegará à **v**

MAX = jogador; **MIN** = adversário

<u>Se já achou algo melhor, por que piorar?</u>



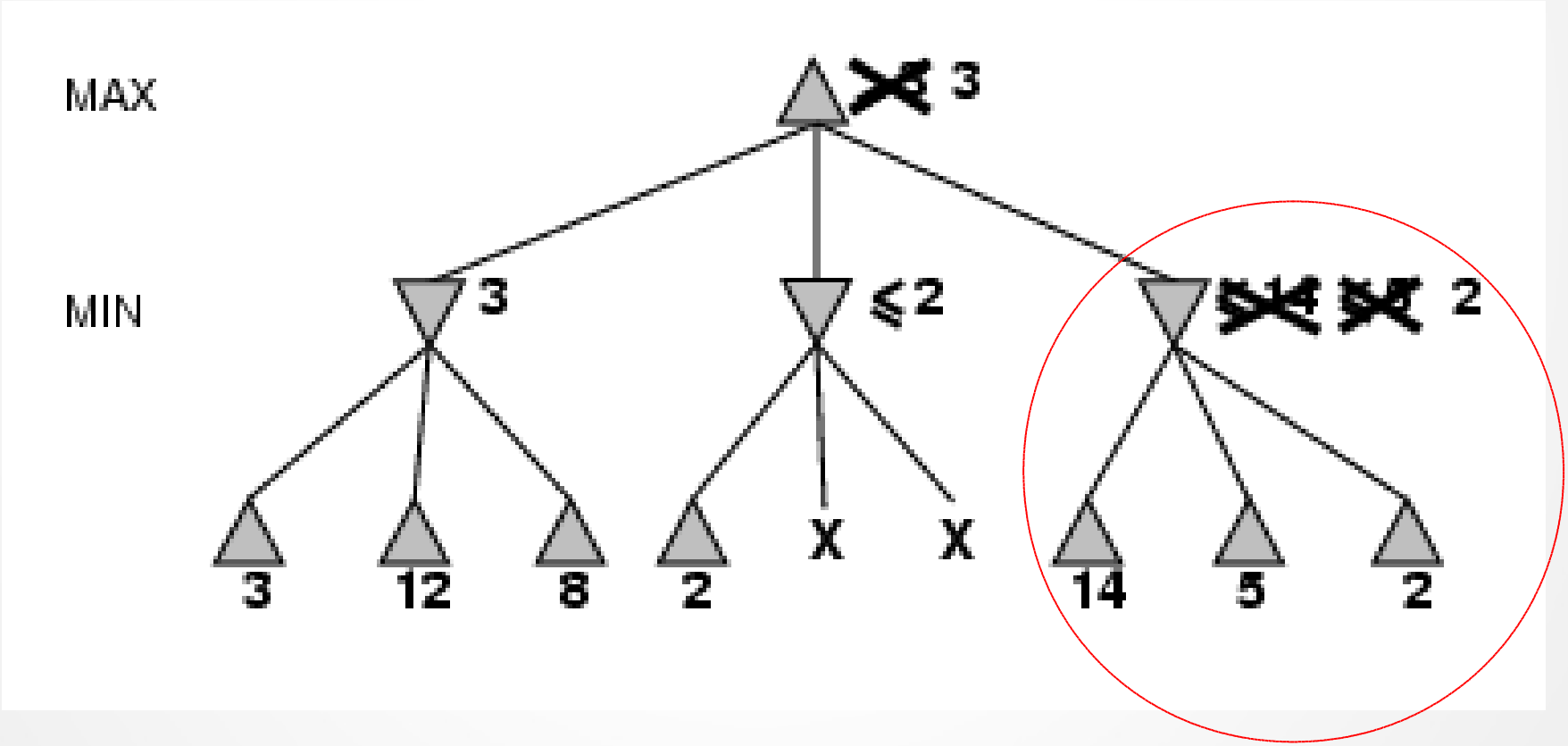
- β é definido de maneira similar para *MIN*

Poda alfa-beta

- Atualiza valores de alfa e beta à medida que prossegue em profundidade
  - **Poda** ramificações tão logo sabe que o valor de um nó corrente é **pior** que o valor corrente de **alfa** ou **beta** para MAX ou MIN

Poda alfa-beta

- Efetividade depende da ordem em que sucessores são examinados



Se terceiro sucessor tivesse sido gerado primeiro, outros dois poderiam ter sido podados

Poda alfa-beta

```
function ALPHA-BETA-SEARCH(state) returns an action
    inputs: state, current state in game
    v ← MAX-VALUE(state, −∞, +∞)
    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 ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

α = limite inferior no resultado de MAX; inicialmente $-\infty$

β = limite superior no resultado de MIN; inicialmente $+\infty$

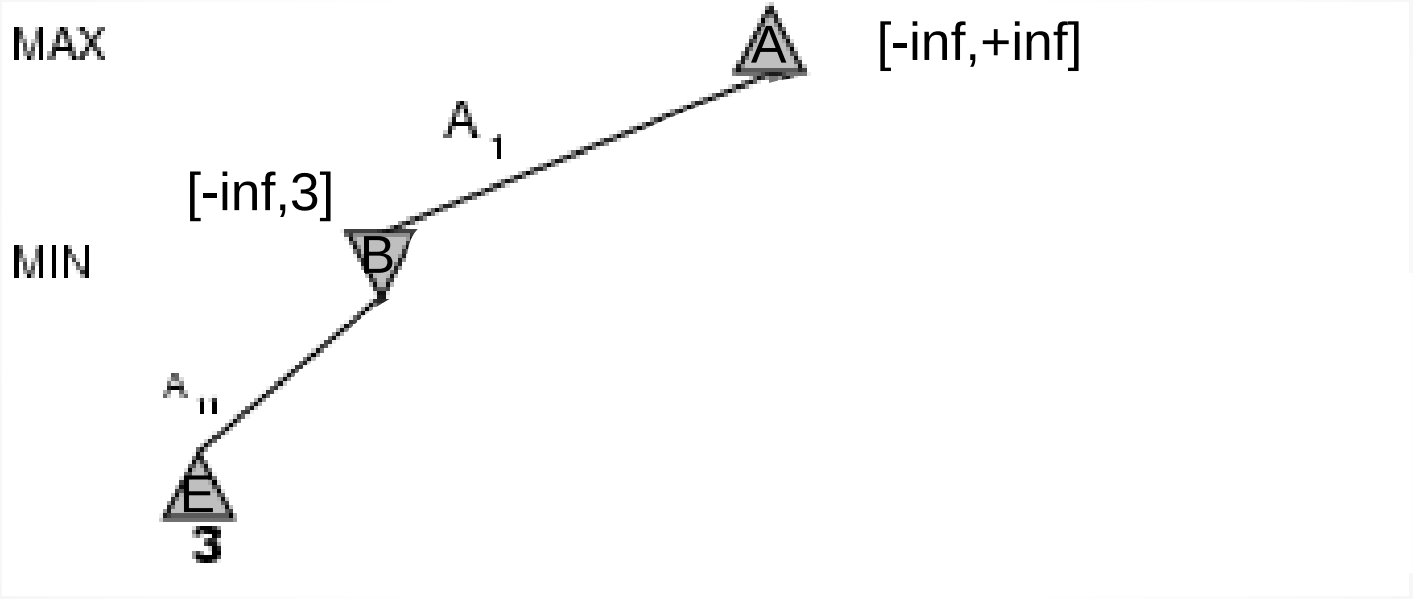
Chama-se busca alfa-beta recursivamente com intervalos cada vez mais estreitos

Poda alfa-beta

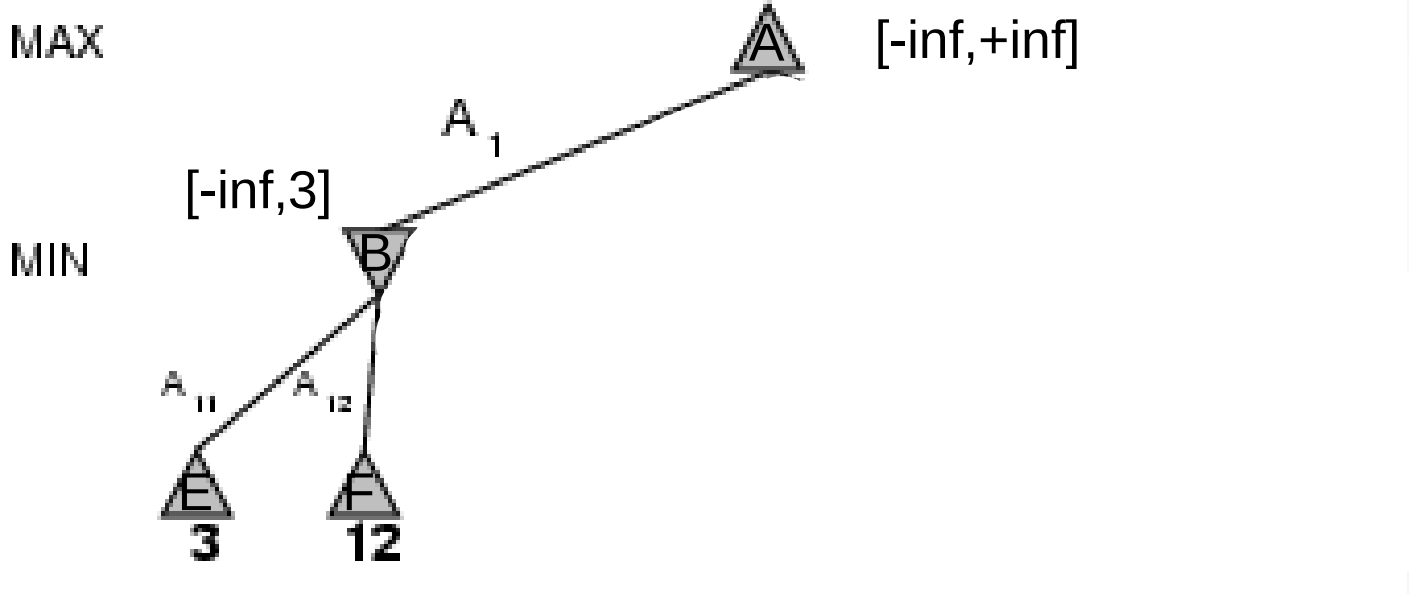
```
function MIN-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state

    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s, α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

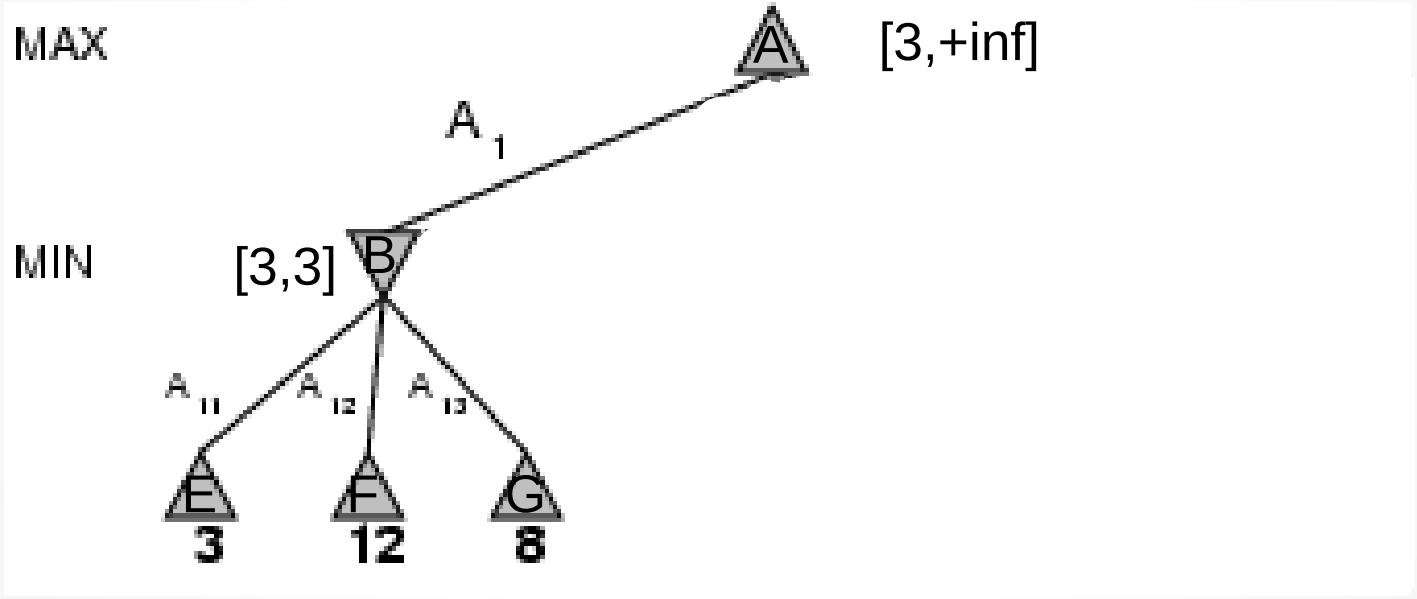
Poda alfa-beta



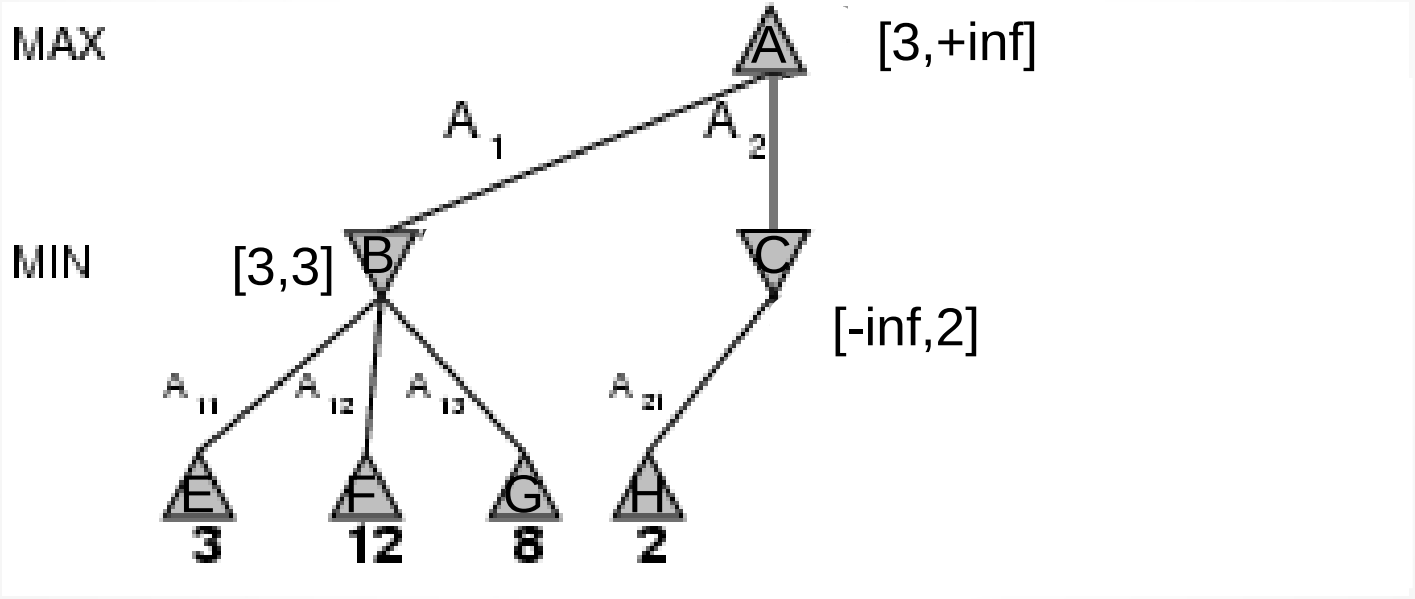
Poda alfa-beta



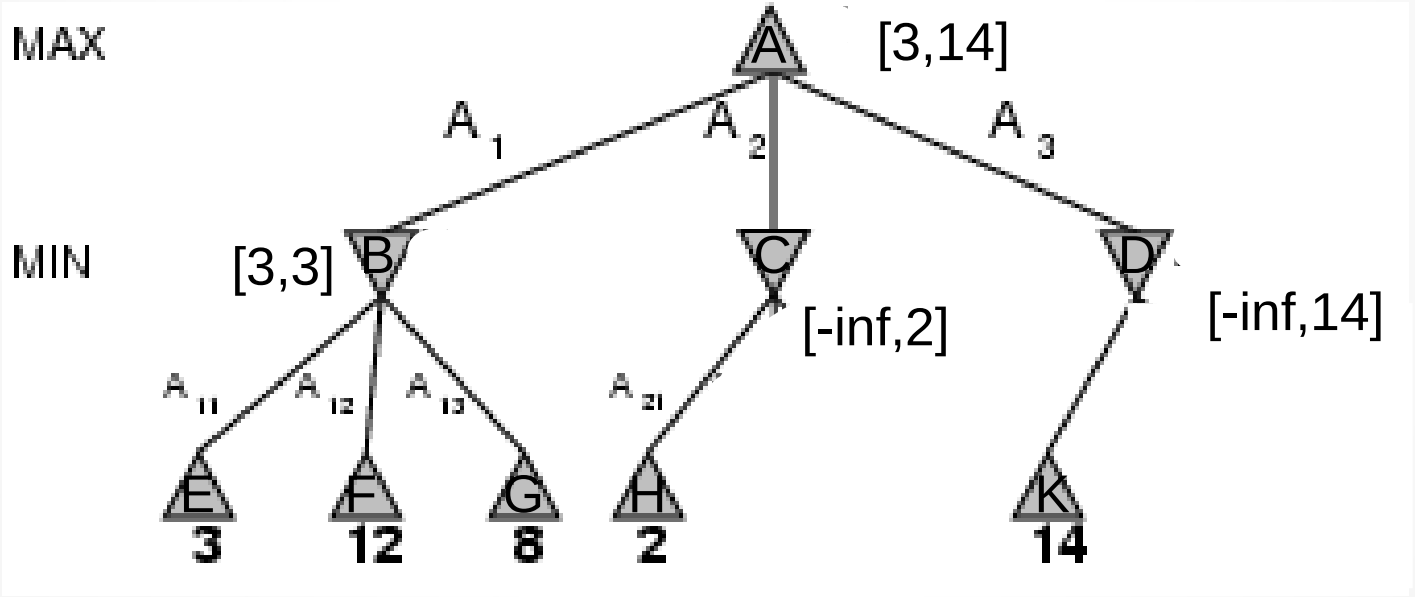
Poda alfa-beta



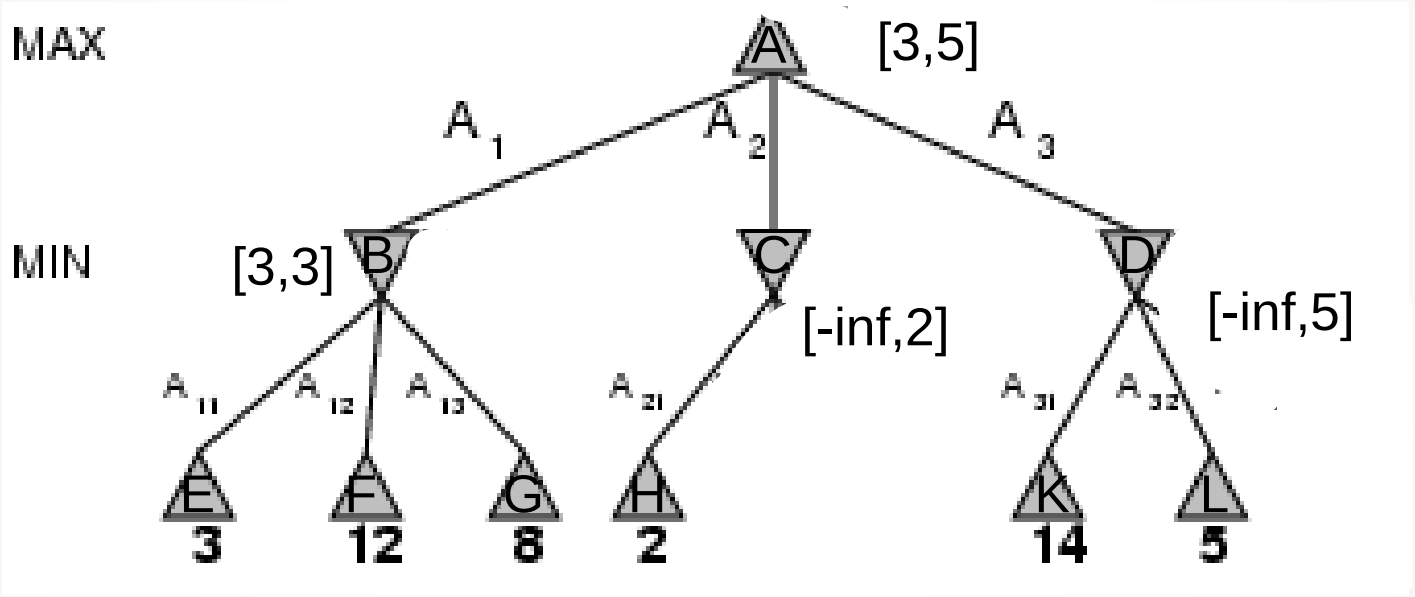
Poda alfa-beta



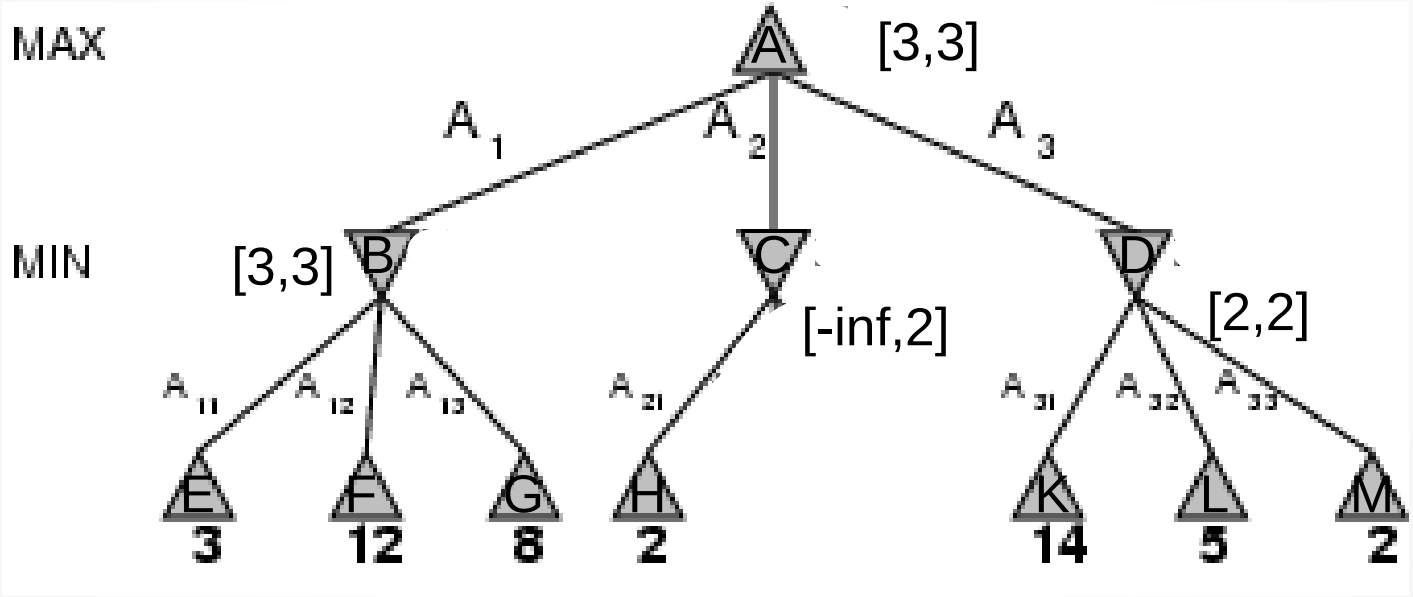
Poda alfa-beta



Poda alfa-beta



Poda alfa-beta



Poda alfa-beta

- Supondo que utiliza melhor ordem
  - Alfa-beta examina $O(b^{m/2})$ nós para escolher melhor movimento
    - Contra $O(b^{m})$ do minimax
    - Pode efetuar exame antecipado a uma distância aproximadamente duas vezes maior no mesmo tempo
- Examinando em ordem aleatória
  - $O(b^{3m/4})$

Propriedades alfa-beta

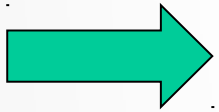
- Poda não afeta resultado final
- Boa ordem de movimento melhora efetividade da poda
- Com “ordem perfeita”, complexidade de tempo = $O(b^{m/2})$
  - Dobra profundidade da busca
- Exemplo simples do valor de raciocinar sobre que computações são relevantes
  - Forma de meta-raciocínio

Exemplo

- Xadrez
  - Minimax: 5 jogadas à frente
    - Humano médio: 6 a 8
  - Alfa-beta: 10 jogadas à frente
    - Desempenho de especialista

Buscas adversariais

- Minimax gera o espaço de busca todo;
- Poda α-β ainda tem que chegar até os estados terminais



São ineficientes para jogos que possuam muitos passos para os estados terminais...
I.e., quase todos os jogos interessantes!

Decisões Imperfeitas em Tempo Real

- **Ambos** algoritmos precisam realizar a busca em toda distância até os nós terminais (folhas)
- Nem sempre o melhor movimento é feito pelo adversário (**Decisões imperfeitas**)
- Precisam ser realizados em período de **tempo razoável**
- **Solução:** substituir a função utilidade por uma função de avaliação (heurística), a qual fornece uma **estimativa da utilidade** esperada da posição e o teste de término

Funções de Avaliação

“O desempenho do algoritmo está diretamente relacionado com a função de avaliação projetada.”

- Deve ordenar os estados terminais
  ex. 1-vitorias 2-Empates 3- derrotas
- A computação não deve demorar tempo demais
- Deve estar fortemente relacionada com as chances reais de vitória

Funções de Avaliação

- Reflete as chances de ganhar: baseada no valor material

  ex. valor de uma peça independentemente da posição das outras

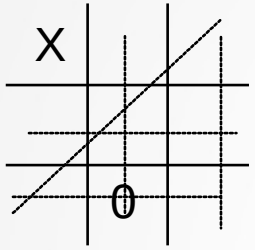
- Função Linear de Peso de propriedade do nó:

$$AVAL(s) = w_1f_1+w_2f_2+...+w_nf_n$$

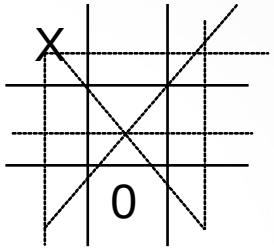
Ex. Os pesos (w) no xadrez poderiam ser o tipo de pedra do xadrez (Peão-1, ..., Rainha-9) e os (f) poderiam ser o número de cada peça no tabuleiro.

- Escolha crucial: compromisso entre precisão e eficiência

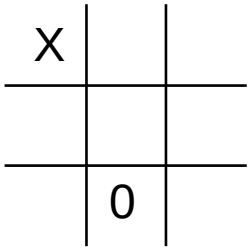
Função de avaliação (h) para o jogo da velha:



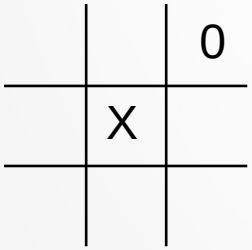
0 tem 5 possibilidades



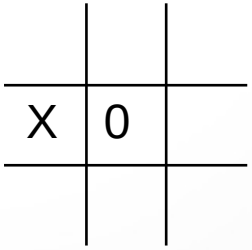
X tem 6 possibilidades



h = 6 - 5 = 1

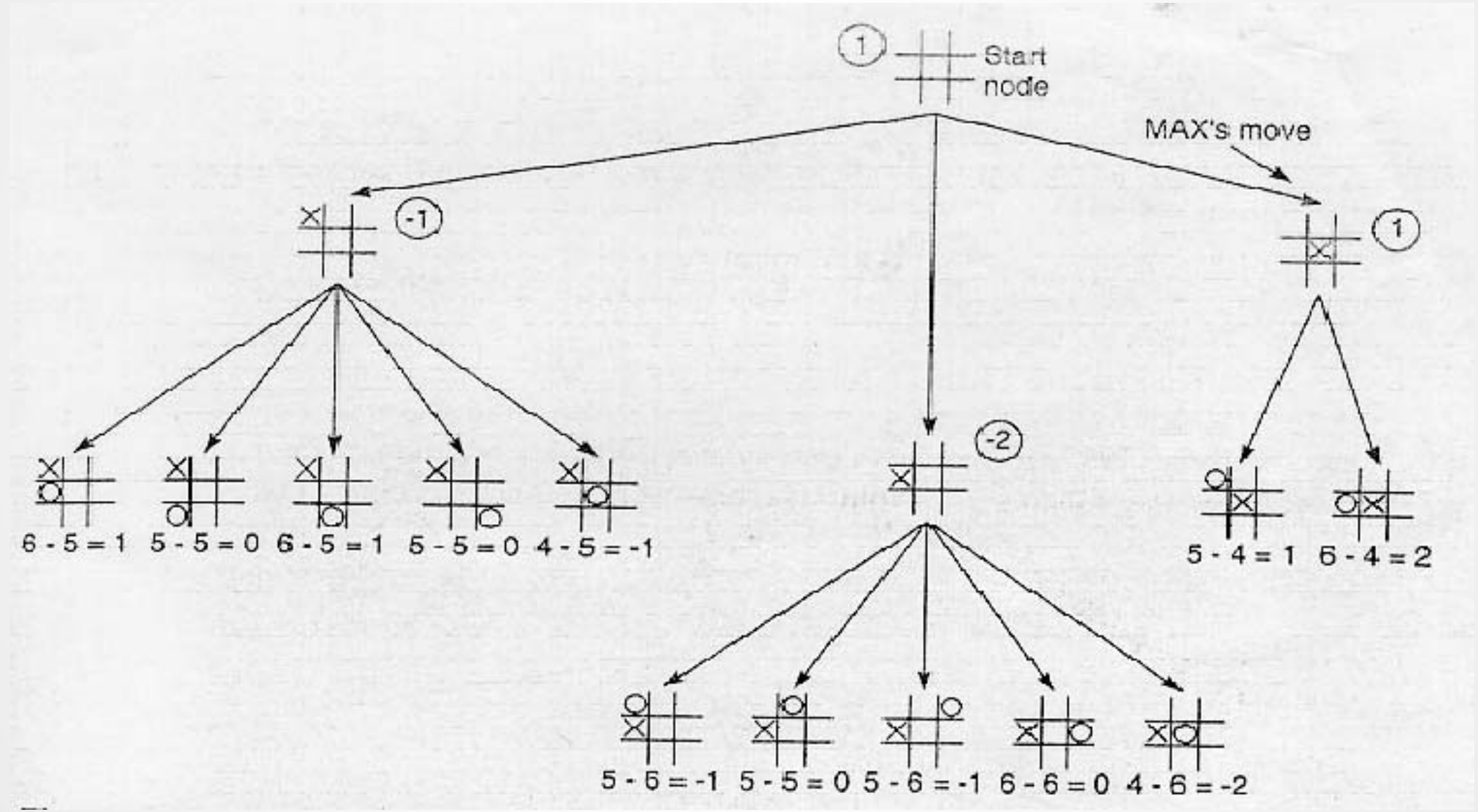


h = 5 - 4 = 1



h = 4 - 6 = = -2

Uso da Funções de Avaliação



Minimax de duas jogadas (two-play) aplicado à abertura do jogo da velha

Referências

- Cap 5 Livro Russel e Norvig
- Material UFPE

- Material complementar:

https://www.youtube.com/watch?v=STjW3eH0Cik

https://www.youtube.com/watch?v=zDskcx8FStA

https://www.youtube.com/watch?v=Eychv62adsI

https://www.youtube.com/watch?v=6kFKnB6JtcY