MiniMax Algorithm

Daniel Nogueira

dnogueira@ipca.pt

MiniMax Algorithm

História

- John Von Neumann – 1928 ("Zur Theorie der Gesellshaftsspiele“ – “Sobre a teoria dos jogos de tabuleiro”)
- Emile Borel publicou, entre 1921 e 1927, quatro notas introduzindo os conceitos de estratégias puras e mistas e a solução MiniMax
- Borel considerou que o teorema MiniMax era em geral falso, apesar de tê-lo comprovado para casos especiais.
- Von Neumann provou o teorema para condições gerais e ainda criou a teoria dos jogos com mais de dois jogadores.

MiniMax Algorithm

O que é?

- Minimax é um algoritmo que minimiza a perda possível para um cenário de pior caso (perda máxima).

MiniMax Algorithm

O que é?

- Minimax é um algoritmo que minimiza a perda possível para um cenário de pior caso (perda máxima).

- MinMax é um algoritmo que identifica qual caminho seguir para vencer o jogo para que o inimigo não interfira.

MiniMax Algorithm

O que é?

- Minimax é um algoritmo que minimiza a perda possível para um cenário de pior caso (perda máxima).

- MinMax é um algoritmo que identifica qual caminho seguir para vencer o jogo para que o inimigo não interfira.

- O jogador assume que a decisão do oponente será desfavorável (o pior cenário é esperado antes que o oponente se mova).

MiniMax Algorithm

O que é?

- Minimax é um algoritmo que minimiza a perda possível para um cenário de pior caso (perda máxima).
- MinMax é um algoritmo que identifica qual caminho seguir para vencer o jogo para que o inimigo não interfira.
- O jogador assume que a decisão do oponente será desfavorável (o pior cenário é esperado antes que o oponente se mova).
- O algoritmo Minimax é como tomar a melhor decisão assumindo que o outro jogador escolherá o pior cenário para você.

MiniMax Algorithm

O que é?

- Minimax é um algoritmo que minimiza a perda possível para um cenário de pior caso (perda máxima).
- MinMax é um algoritmo que identifica qual caminho seguir para vencer o jogo para que o inimigo não interfira.
- O jogador assume que a decisão do oponente será desfavorável (o pior cenário é esperado antes que o oponente se mova).
- O algoritmo Minimax é como tomar a melhor decisão assumindo que o outro jogador escolherá o pior cenário para você.
- É aplicável em um jogo de dois jogadores que não é cooperativo (um jogo de soma zero).

MiniMax Algorithm

O que é?

- Minimax é um algoritmo que minimiza a perda possível para um cenário de pior caso (perda máxima).
- MinMax é um algoritmo que identifica qual caminho seguir para vencer o jogo para que o inimigo não interfira.
- O jogador assume que a decisão do oponente será desfavorável (o pior cenário é esperado antes que o oponente se mova).
- O algoritmo Minimax é como tomar a melhor decisão assumindo que o outro jogador escolherá o pior cenário para você.
- É aplicável em um jogo de dois jogadores que não é cooperativo (um jogo de soma zero).
- Isso significa que se um jogador ganha, o outro perde (cada agente estará interessado em maximizar sua utilidade, mesmo que isso prejudique o outro).

MiniMax Algorithm

Representação

Estado inicial

posição no tabuleiro do jogo e qual jogador inicia

MiniMax Algorithm

Representação

Estado inicial

posição no tabuleiro do jogo e qual jogador inicia

Função Sucessor

lista de pares (movimento possível, estado resultante)

MiniMax Algorithm

Representação

Estado inicial → **posição no tabuleiro do jogo e qual jogador inicia**

Função Sucessor → **lista de pares (movimento possível, estado resultante)**

Teste Objetivo → **determina quando o jogo termina (estados terminais)**

MiniMax Algorithm

Representação

Estado inicial

posição no tabuleiro do jogo e qual jogador inicia

Função Sucessor

lista de pares (movimento possível, estado resultante)

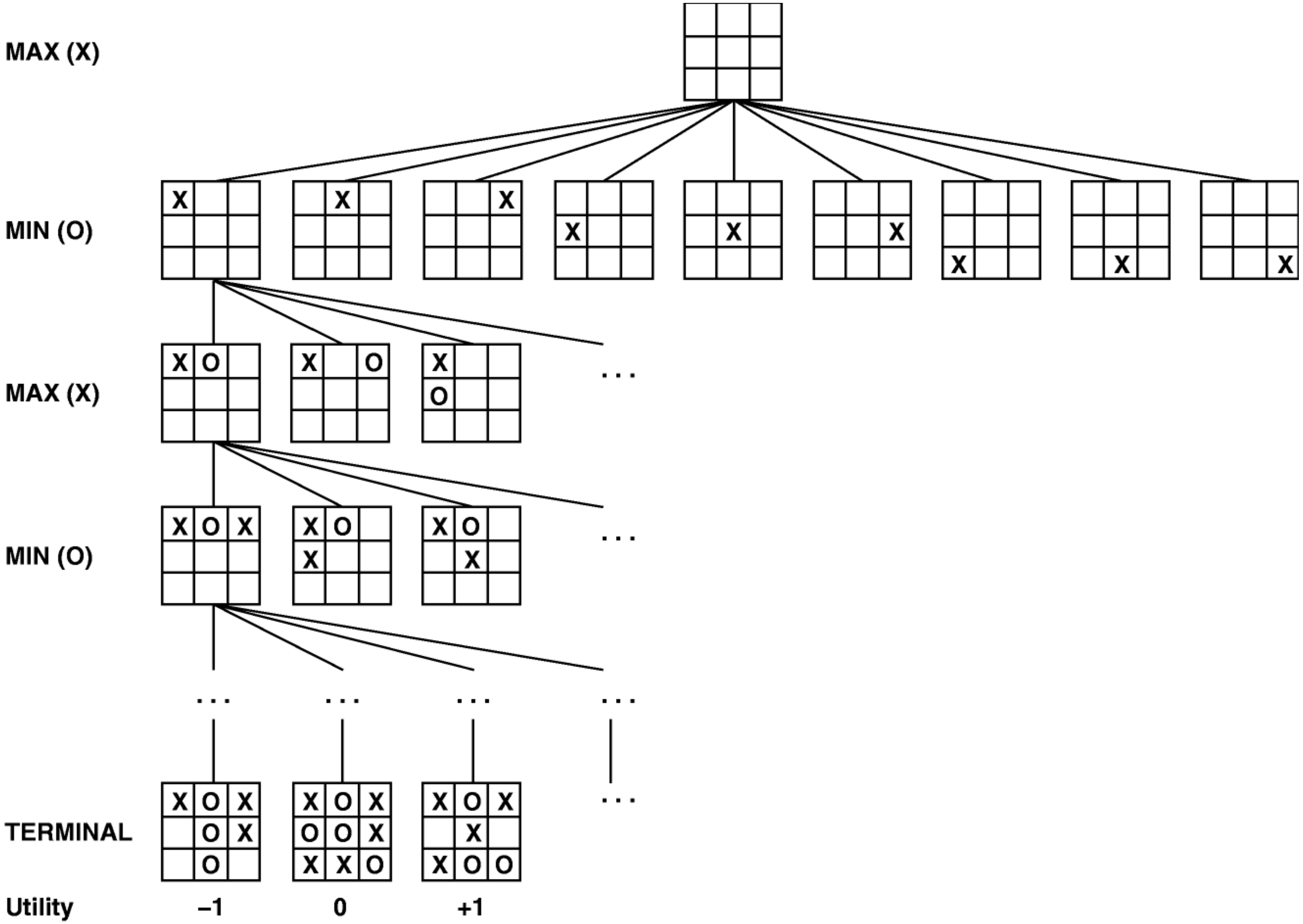
Teste Objetivo

determina quando o jogo termina (estados terminais)

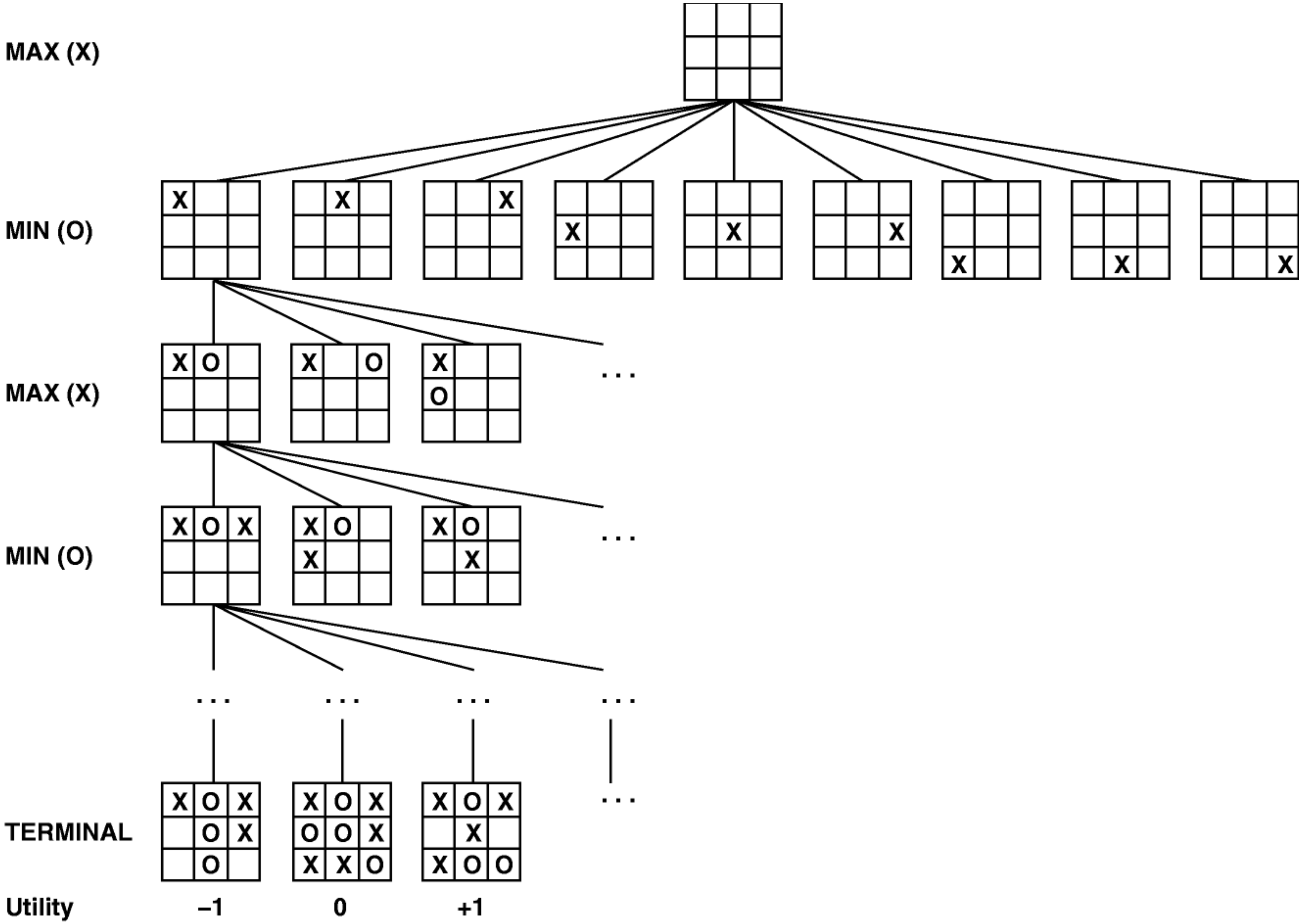
Função utilidade (objetivo)

determina valores numéricos aos estados terminais

MiniMax Algorithm



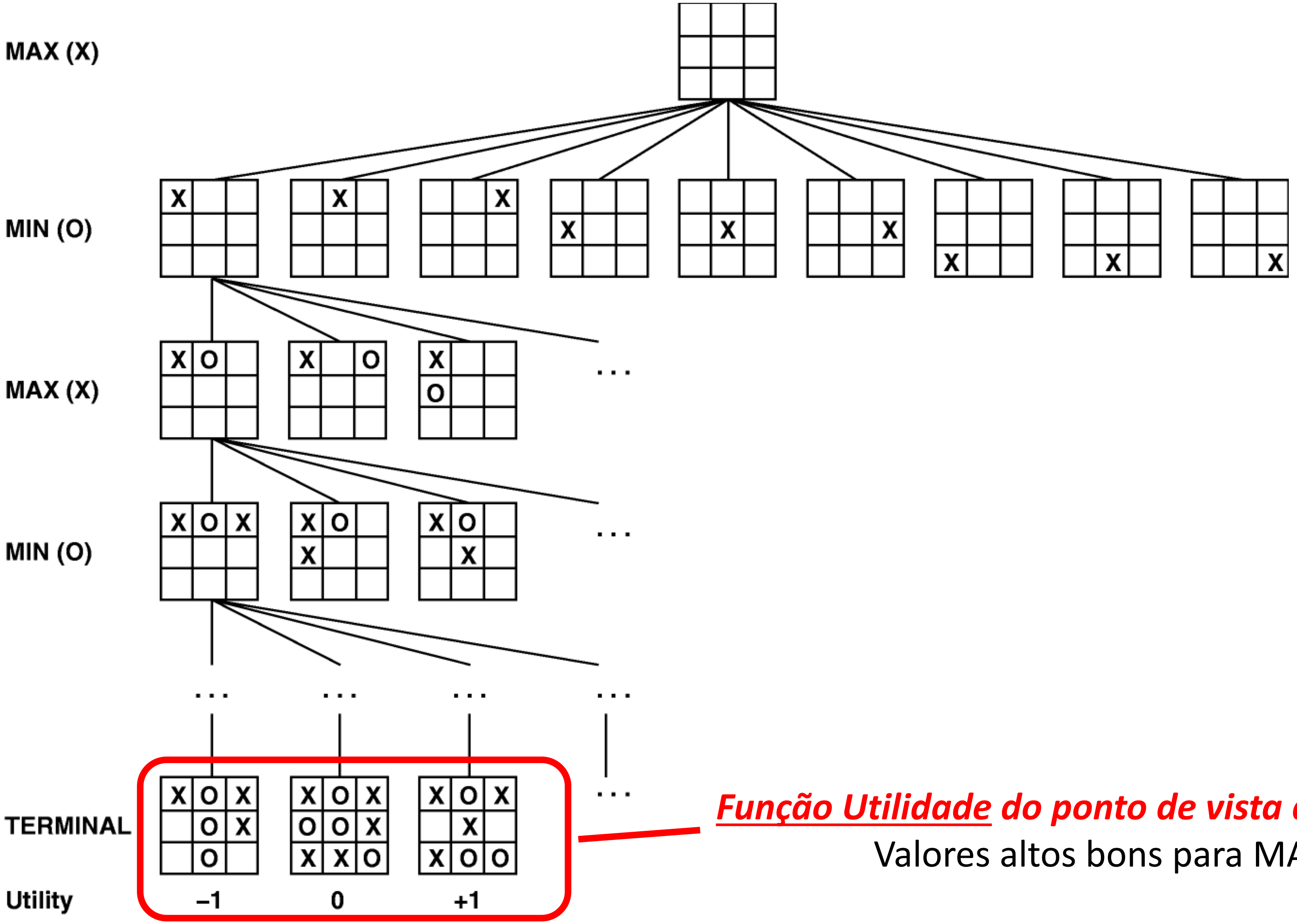
MiniMax Algorithm



2 jogadores, alternados em turnos (determinístico)

MAX = X (jogador)
MIN = O (adversário)

MiniMax Algorithm



2 jogadores, alternados em turnos (determinístico)

MAX = X (jogador)
MIN = O (adversário)

<u>Função Utilidade</u> do ponto de vista de MAX

Valores altos bons para MAX e ruins para MIN

MiniMax Algorithm

Estratégica de Jogo

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...

MiniMax Algorithm

Estratégica de Jogo

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

MiniMax Algorithm

Estratégica de Jogo

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

MiniMax Algorithm

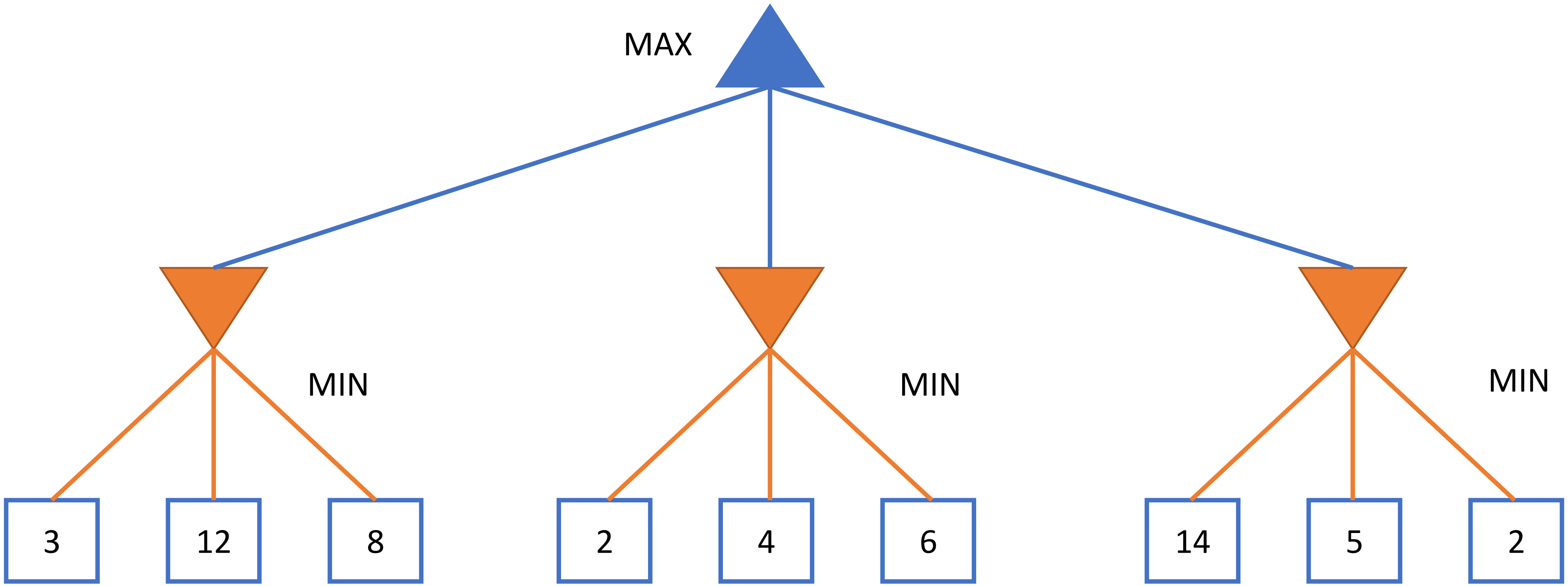
Estratégica de Jogo

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

MiniMax Algorithm

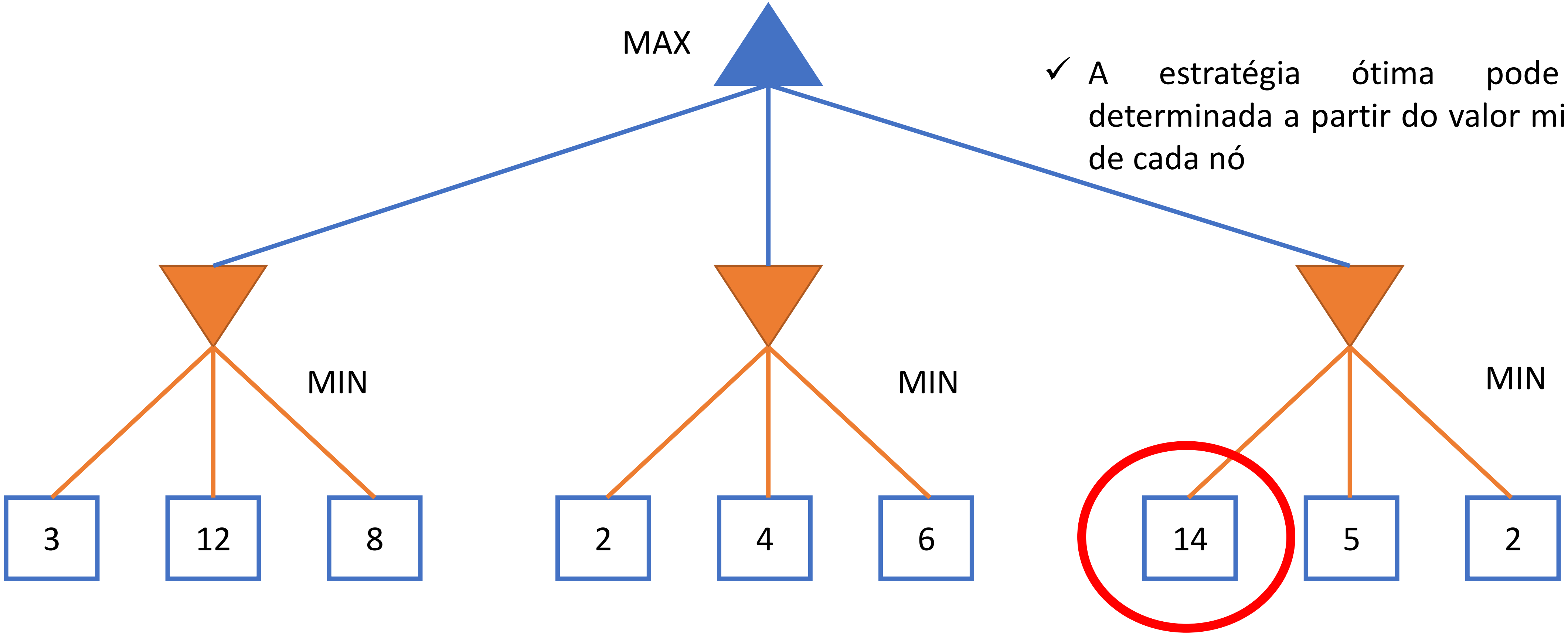
Estratégica de Jogo



MiniMax Algorithm

Estratégica de Jogo

- ✓ MAX prefere mover para estado de minimax máximo
- ✓ MIN prefere valor mínimo

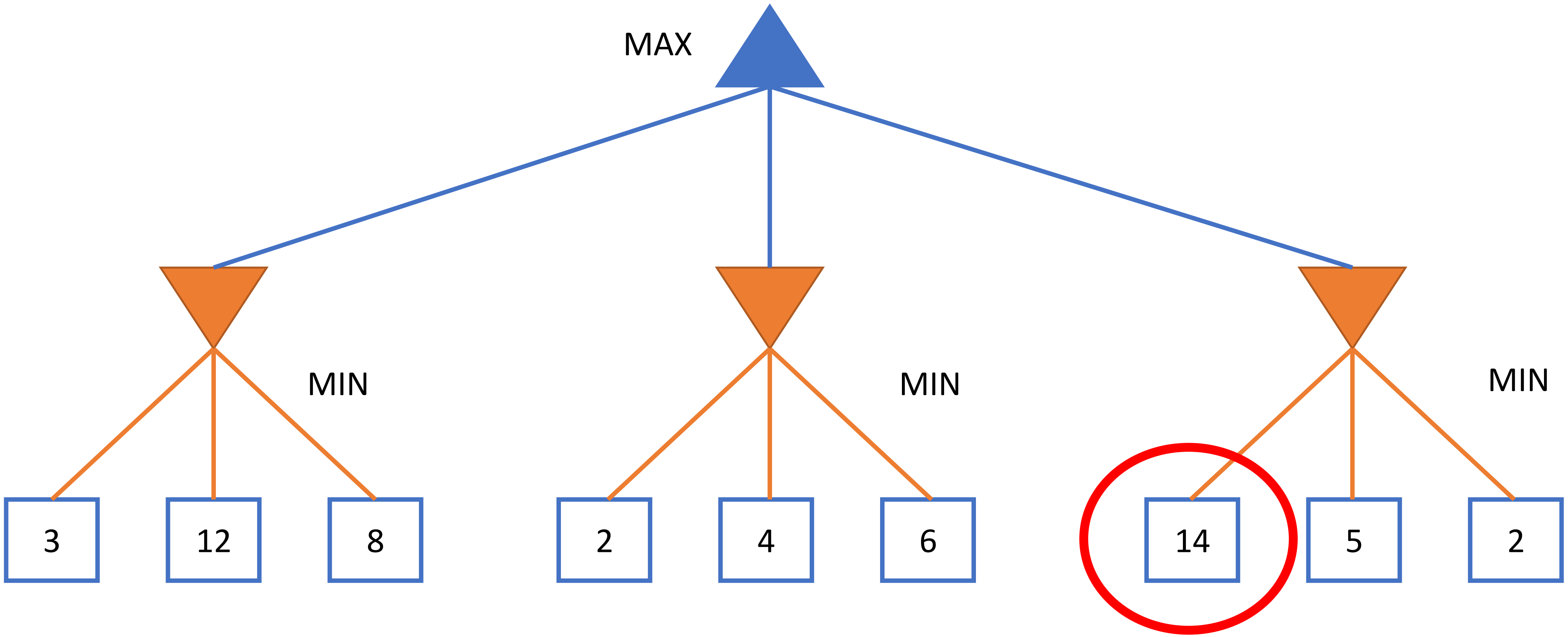


- ✓ A estratégia ótima pode ser determinada a partir do valor minimax de cada nó

MiniMax Algorithm

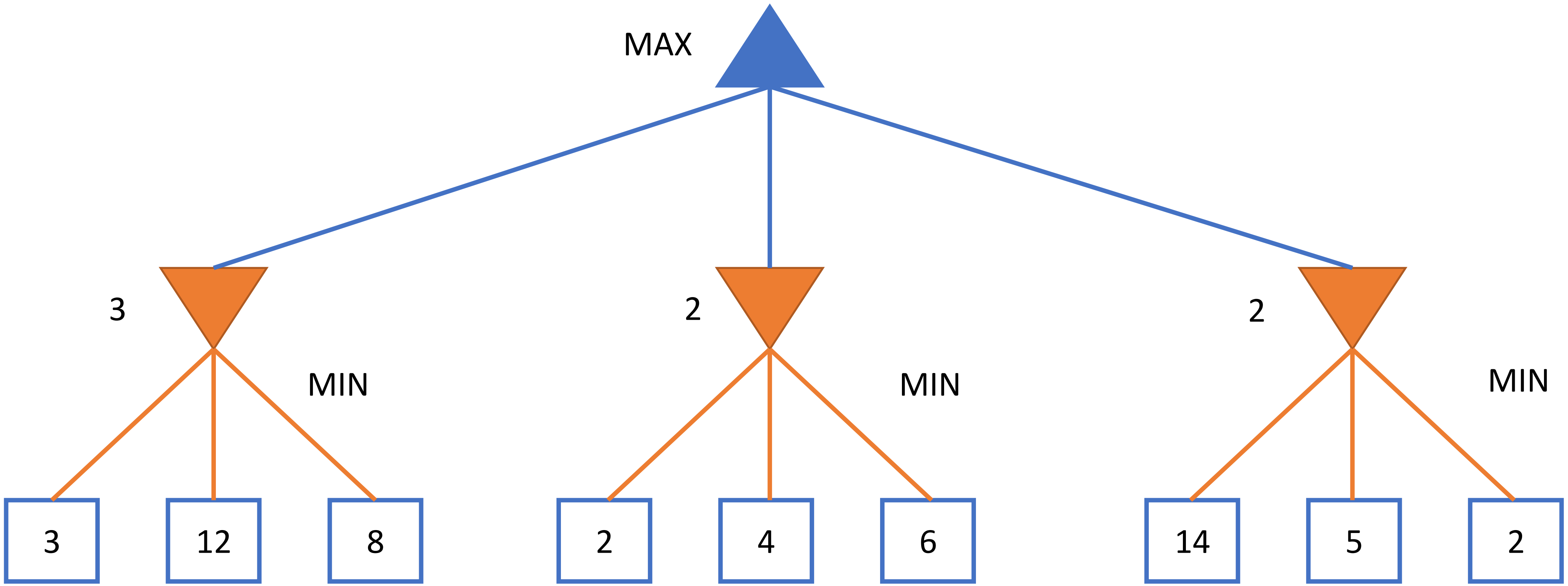
Estratégica de Jogo

- ✓ A estratégia ótima pode ser determinada a partir do valor minimax de cada nó



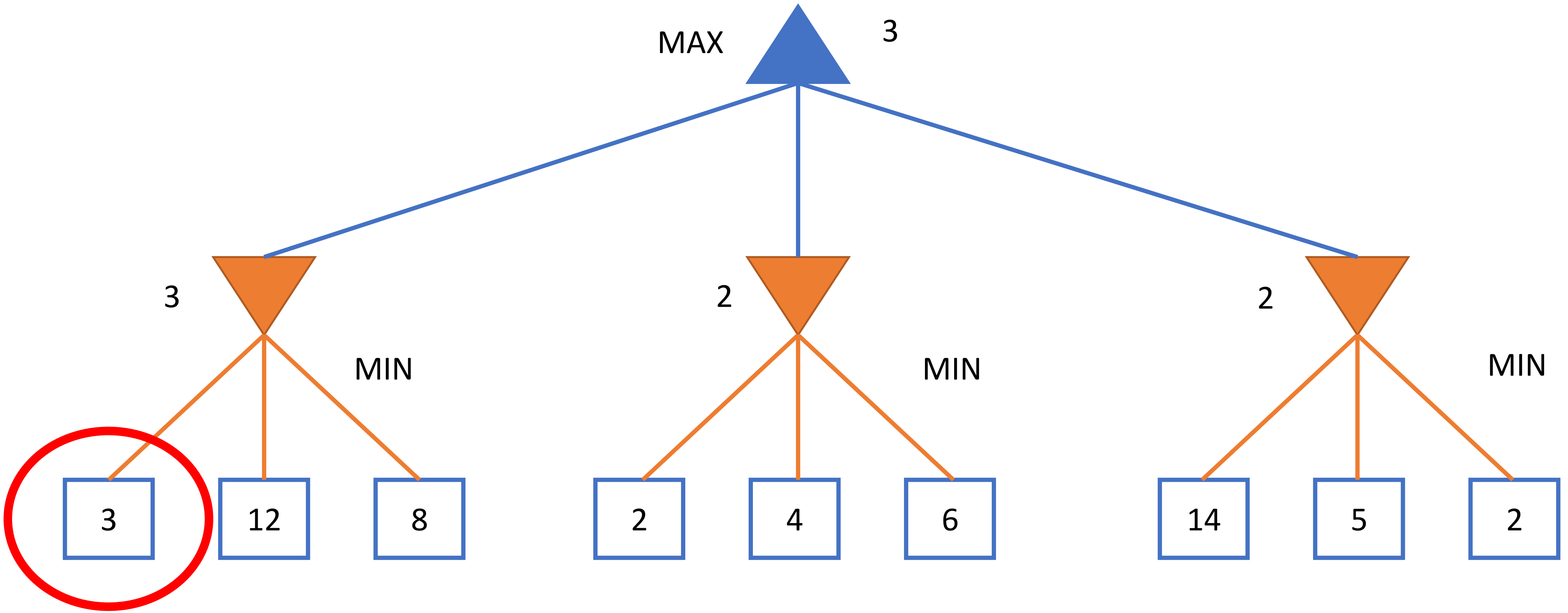
MiniMax Algorithm

Estratégica de Jogo



MiniMax Algorithm

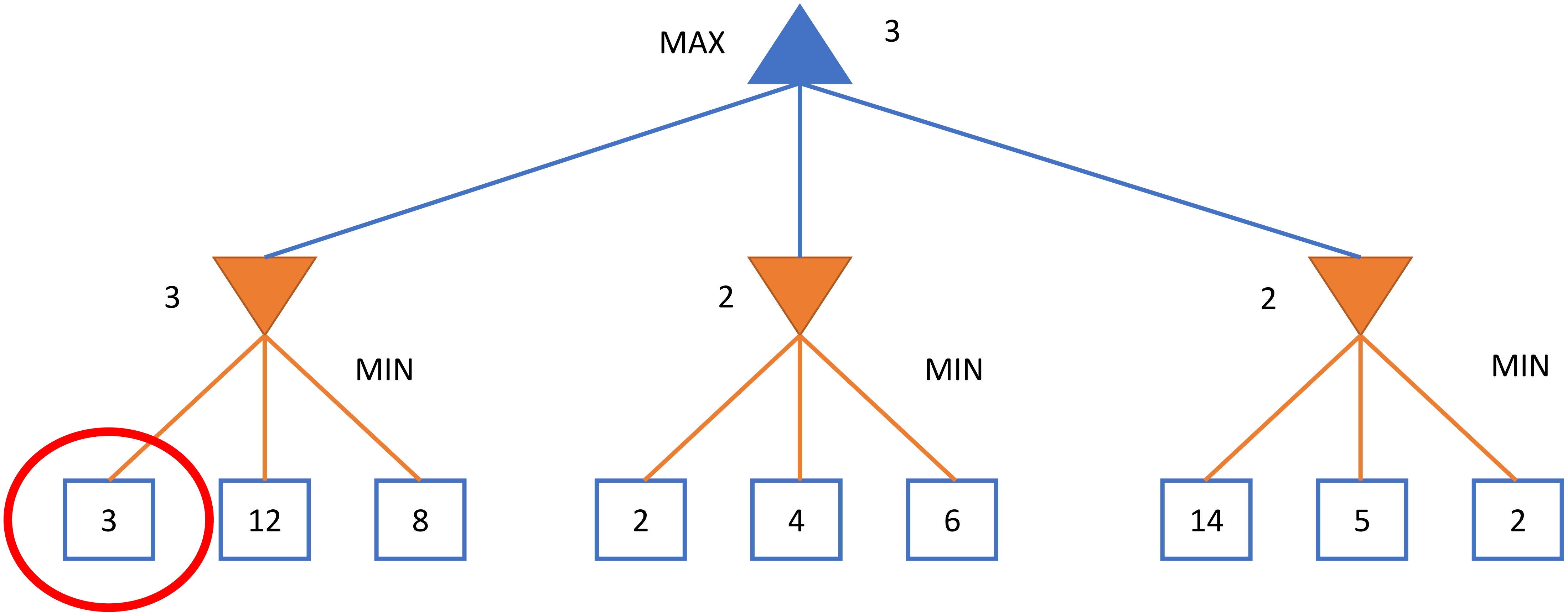
Estratégica de Jogo



MiniMax Algorithm

Estratégica de Jogo

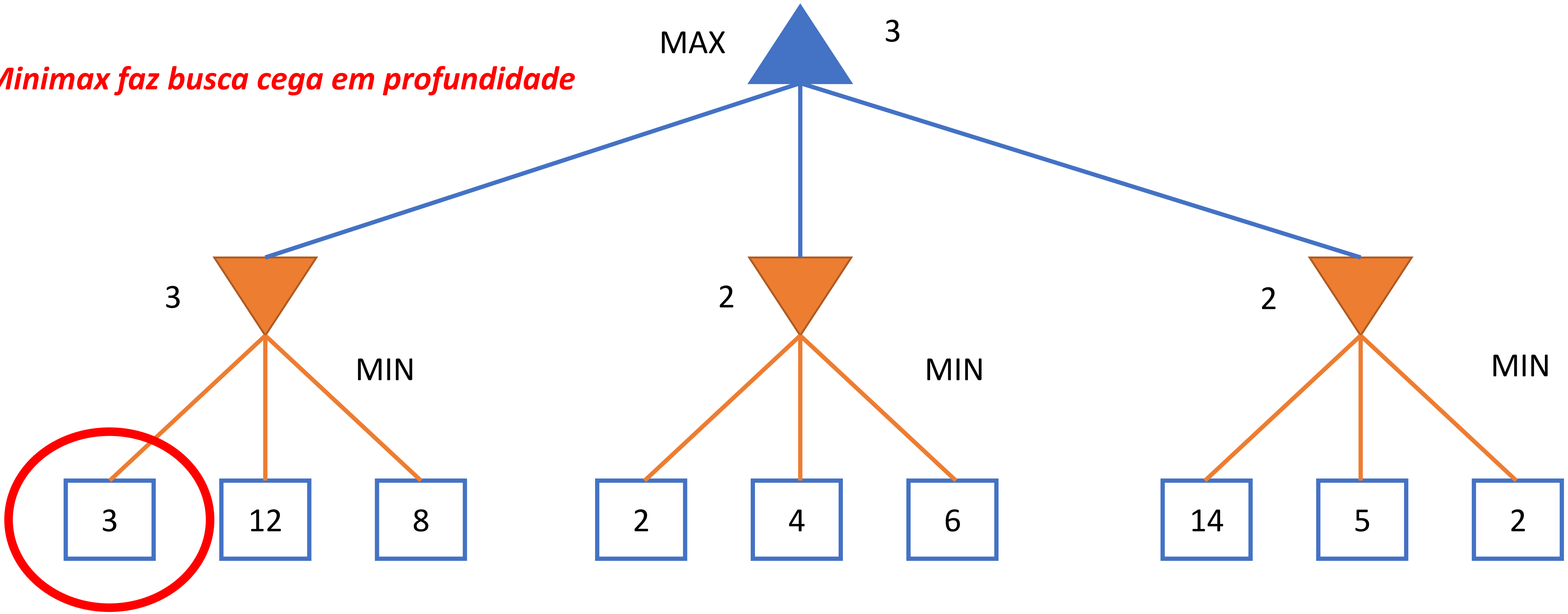
maximizar a utilidade (ganho) supondo que o adversário vai tentar minimizá-la.



MiniMax Algorithm

Estratégica de Jogo maximizar a utilidade (ganho) supondo que o adversário vai tentar minimizá-la.

Minimax faz busca cega em profundidade



MiniMax Algorithm

Valor MINIMAX	Valor MINIMAX de um nó é a utilidade de MAX no estado correspondente

MiniMax Algorithm

Valor MINIMAX

Valor MINIMAX de um nó é a utilidade de MAX no estado correspondente

Valor_minimax(n) = { utilidade(n) → se n é estado terminal

MiniMax Algorithm

Valor MINIMAX

Valor MINIMAX de um nó é a utilidade de MAX no estado correspondente

Valor_minimax(n) =

- utilidade(n) → se n é estado terminal
- Max [Valor_minimax(n)] dos Sucessores → se n é um nó MAX

MiniMax Algorithm

Valor MINIMAX — Valor MINIMAX de um nó é a utilidade de MAX no estado correspondente

Valor_minimax(n) =

- utilidade(n) → se n é estado terminal
- Max [Valor_minimax(n)] dos Sucessores → se n é um nó MAX
- Min [Valor_minimax(n)] dos Sucessores → se n é um nó MIN

MiniMax Algorithm

Valor MINIMAX — Valor MINIMAX de um nó é a utilidade de MAX no estado correspondente

Valor_minimax(n) =

- utilidade(n) → se n é estado terminal
- Max [Valor_minimax(n)] dos Sucessores → se n é um nó MAX
- Min [Valor_minimax(n)] dos Sucessores → se n é um nó MIN

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

MiniMax Algorithm

Implementação

```
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
```

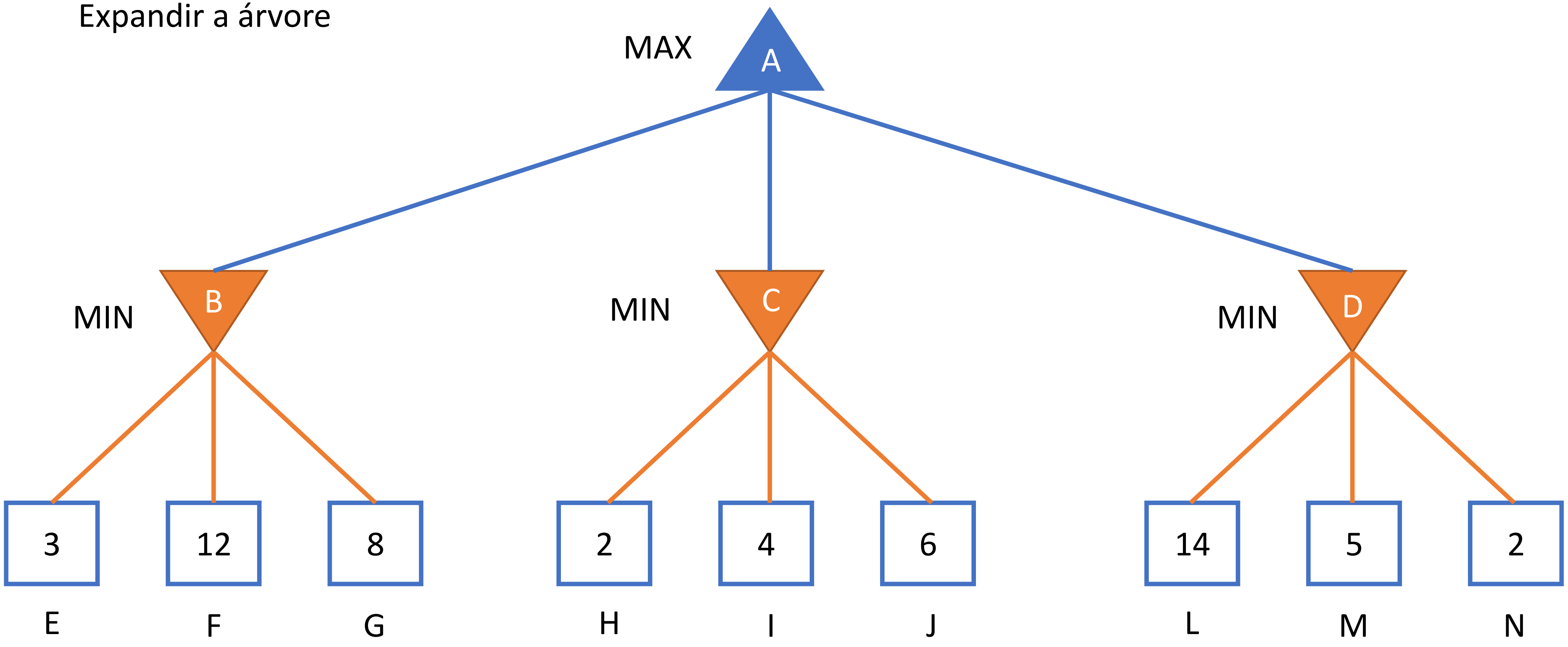
MiniMax Algorithm

Implementação

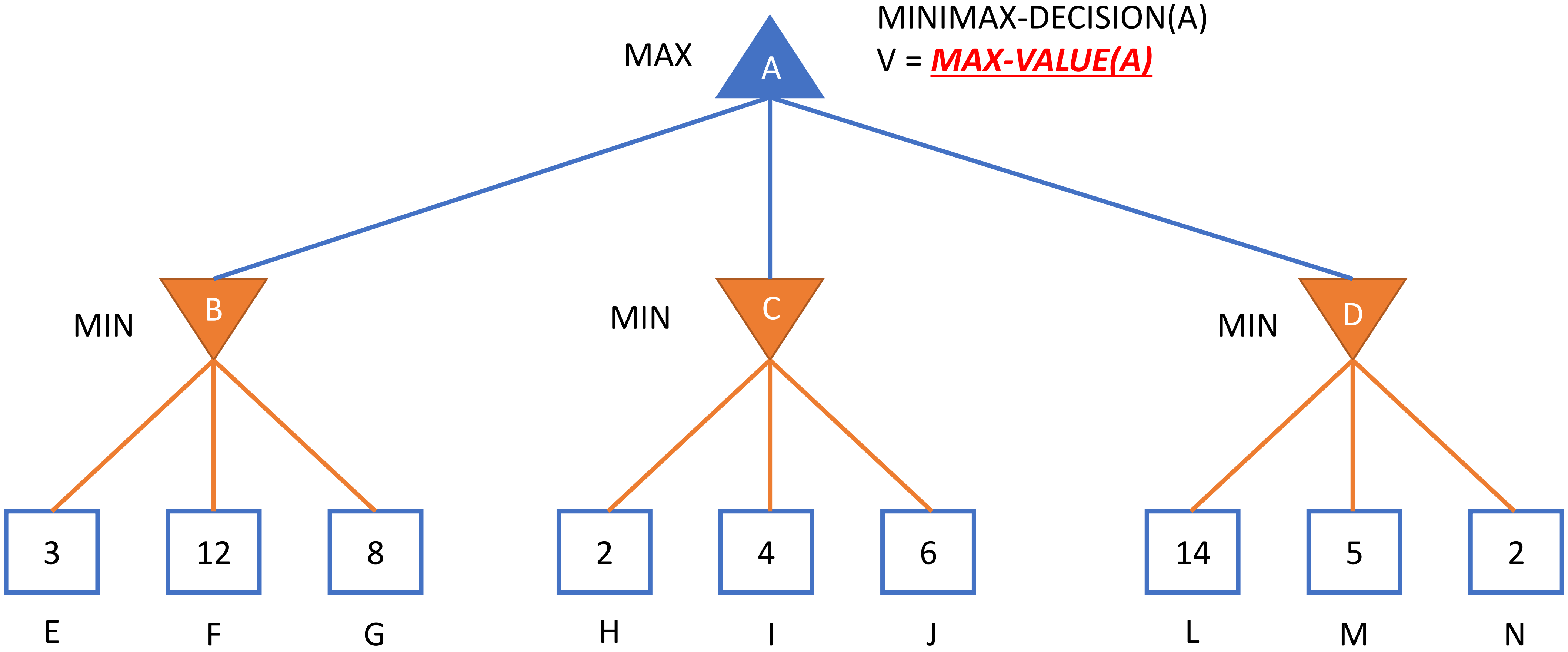
1. Expandir a árvore inteira abaixo da raiz
2. Avaliar os nós terminais como ganhos/perdas para o MAX
3. Selecionar um nó sem utilidade, n, que tenha todos os filhos já com valor. Se não há um nó desses, a busca terminou: retornar o valor da raiz.
4. Se n é movimento MIN, atribuí-lo um valor que é o mínimo dos valores de seus filhos.
   Se n é MAX, atribuí-lo um valor que é o máximo dos valores dos seus filhos.
5. Retornar ao Passo 3.

MiniMax Algorithm

Expandir a árvore



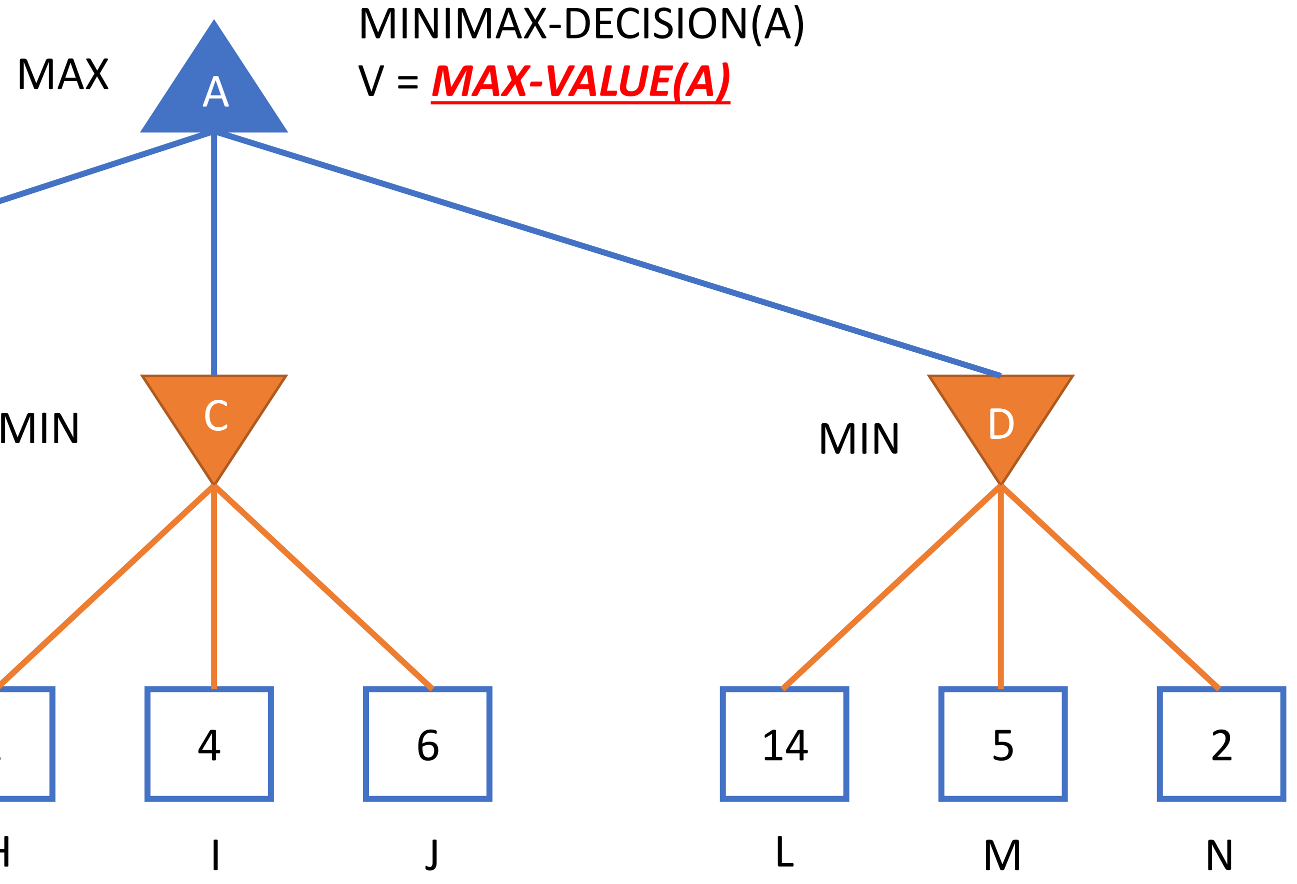
MiniMax Algorithm



MiniMax Algorithm

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

$v \leftarrow$ MAX-VALUE(*state*)
return the *action* in SUCCESSORS(*state*) with value v



MiniMax Algorithm

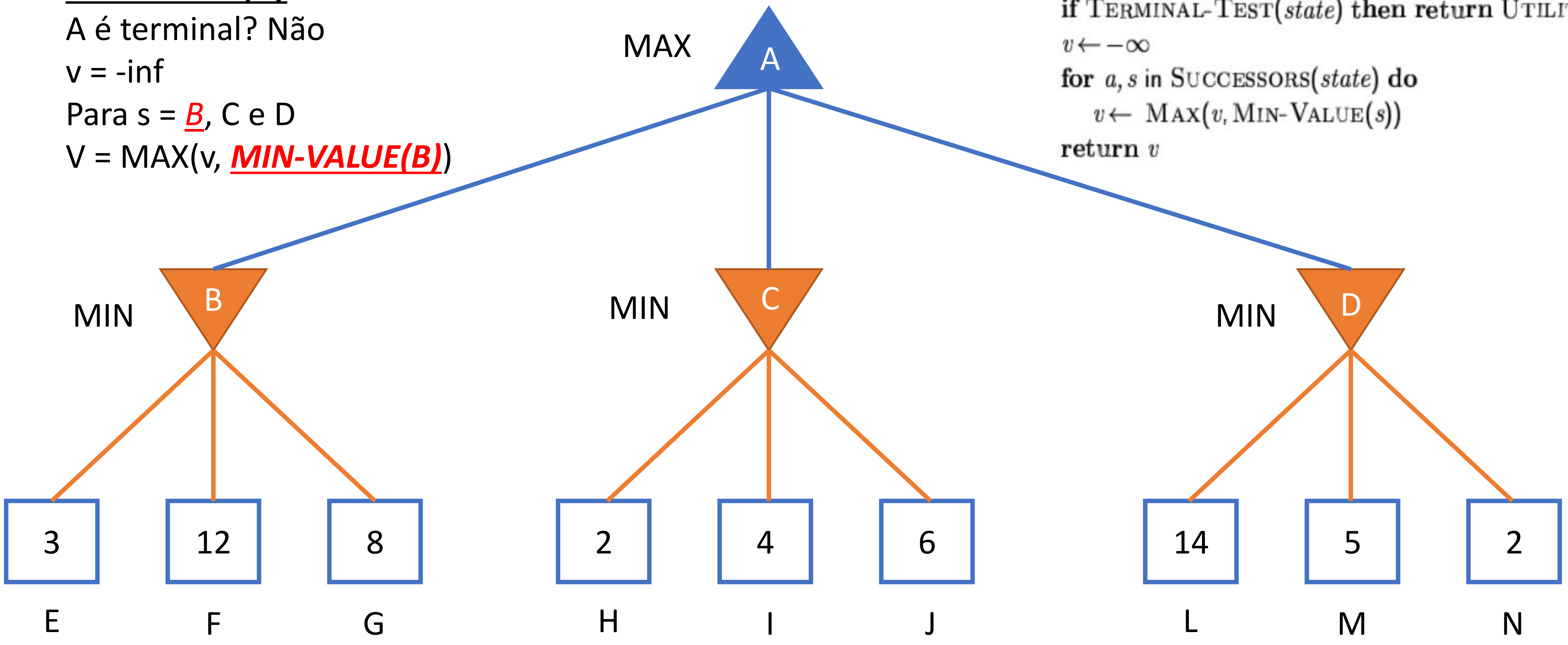
MAX-VALUE(A)

A é terminal? Não

v = -inf

Para s = *B*, C e D

V = MAX(v, ***MIN-VALUE(B)***)



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**
    $v \leftarrow$ MAX(v, MIN-VALUE(s))
  return v

MiniMax Algorithm

MIN-VALUE(B)

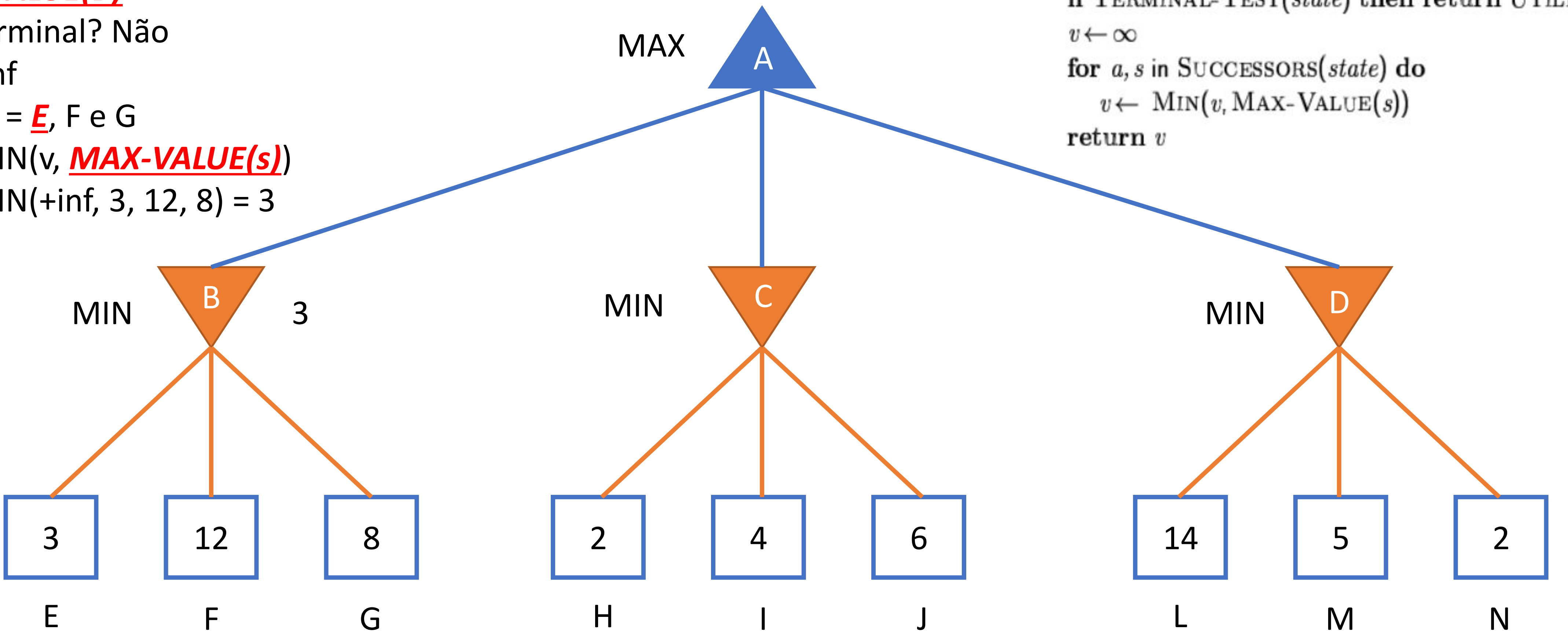
B é terminal? Não

v = +inf

Para s = ***E***, F e G

V = MIN(v, ***MAX-VALUE(s)***)

V = MIN(+inf, 3, 12, 8) = 3



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

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

  $v \leftarrow \infty$

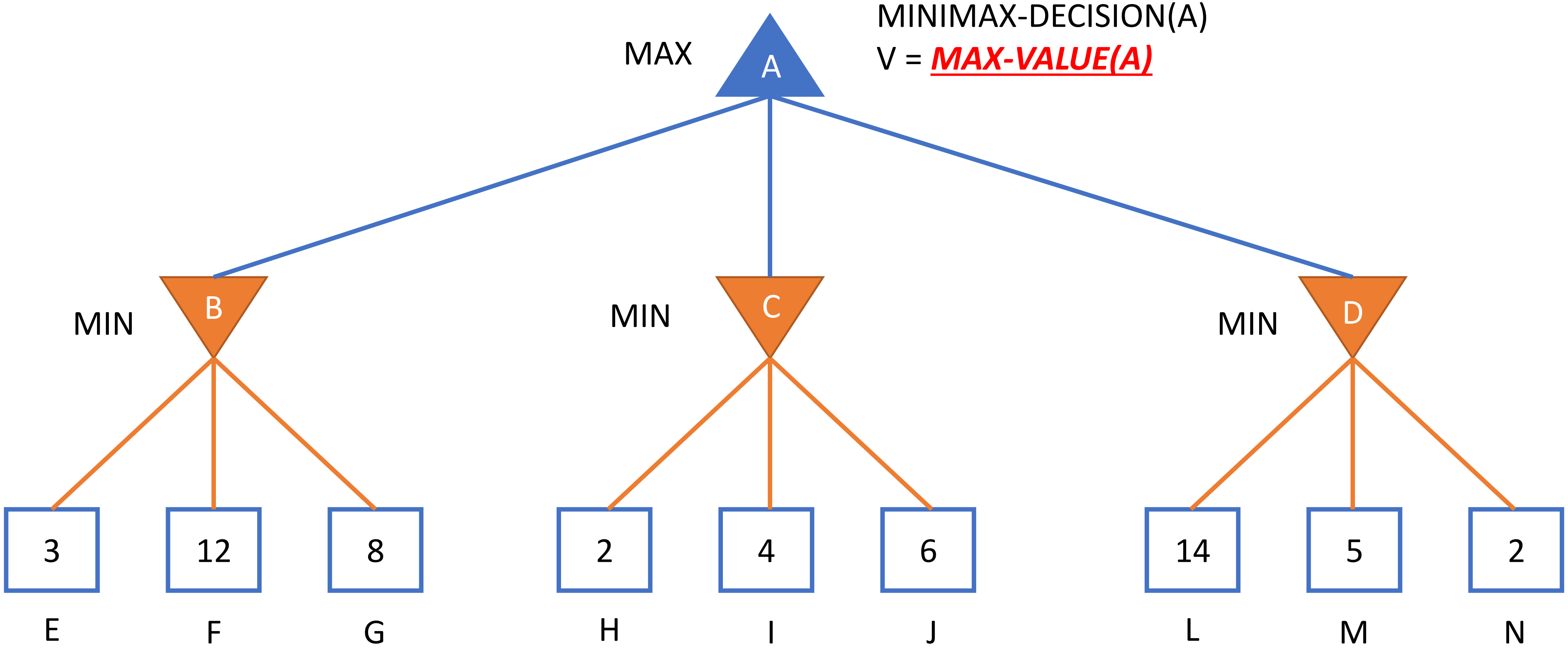
  for a, s in SUCCESSORS(*state*) **do**

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

  return v

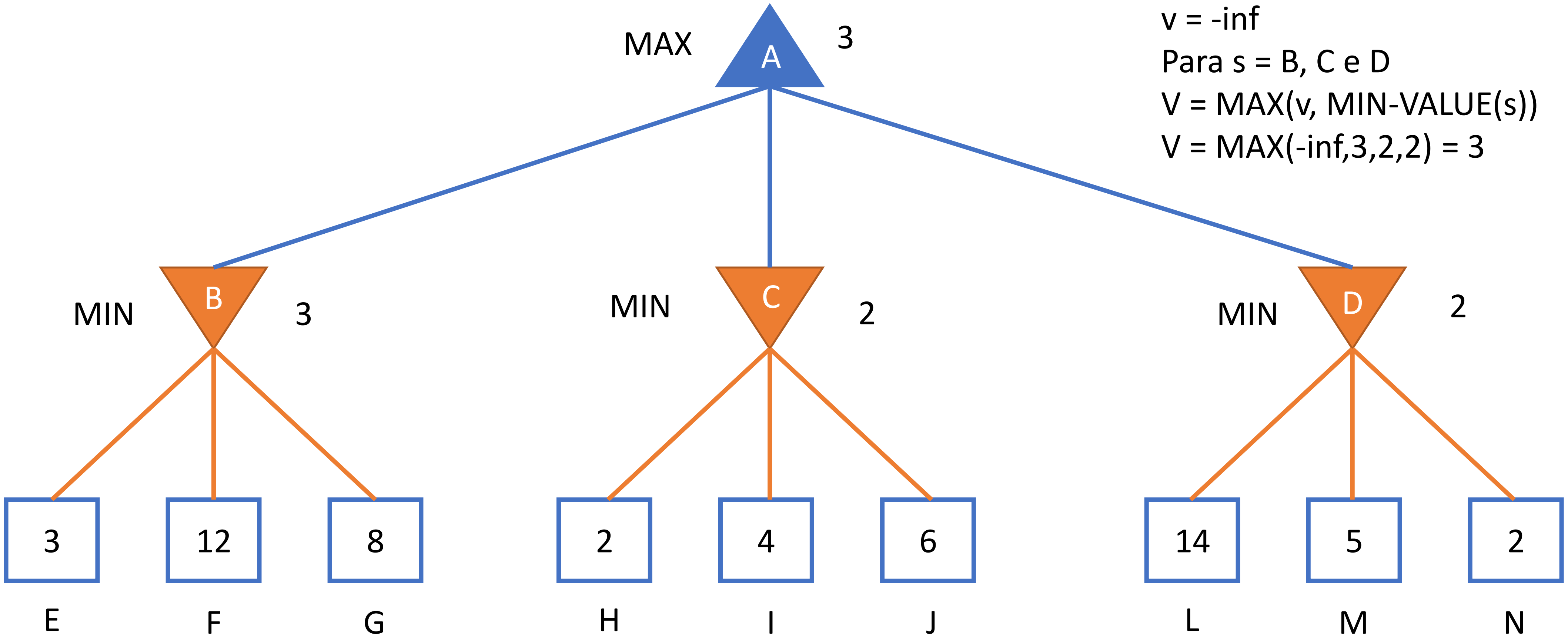
MiniMax Algorithm

MINIMAX-DECISION(A)
V = ***MAX-VALUE(A)***



MAX-VALUE(E)
E é terminal? Sim
v = UTILITY(E) = 3

MiniMax Algorithm



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

MiniMax Algorithm

Desvantagem

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 = $O(b*m)$

$$Complexidade\ Tempo = O(3^3)$$
$$Complexidade\ Espaço = O(3*3)$$

m = 100

b = 35

$$Complexidade\ Tempo = O(35^{100})$$
$$Complexidade\ Espaço = O(35*100)$$

MiniMax Algorithm

Poda (Pruning) α-β

- ✓ Deixar de considerar grandes partes da árvore de jogo
- ✓ Podar 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)
- ❑ Retornar o mesmo que MINIMAX, porém sem percorrer todos os estados.

MiniMax Algorithm

Poda (Pruning) α-β

- Recebe o nome de dois parâmetros.
- Eles descrevem limites nos valores que aparecem em qualquer lugar ao longo do caminho em consideração:

✓ α = o valor da melhor escolha (ou seja, valor mais alto) encontrada até agora ao longo do caminho para MAX

✓ β = o valor da melhor escolha (ou seja, valor mais baixo) encontrada até agora ao longo do caminho para MIN

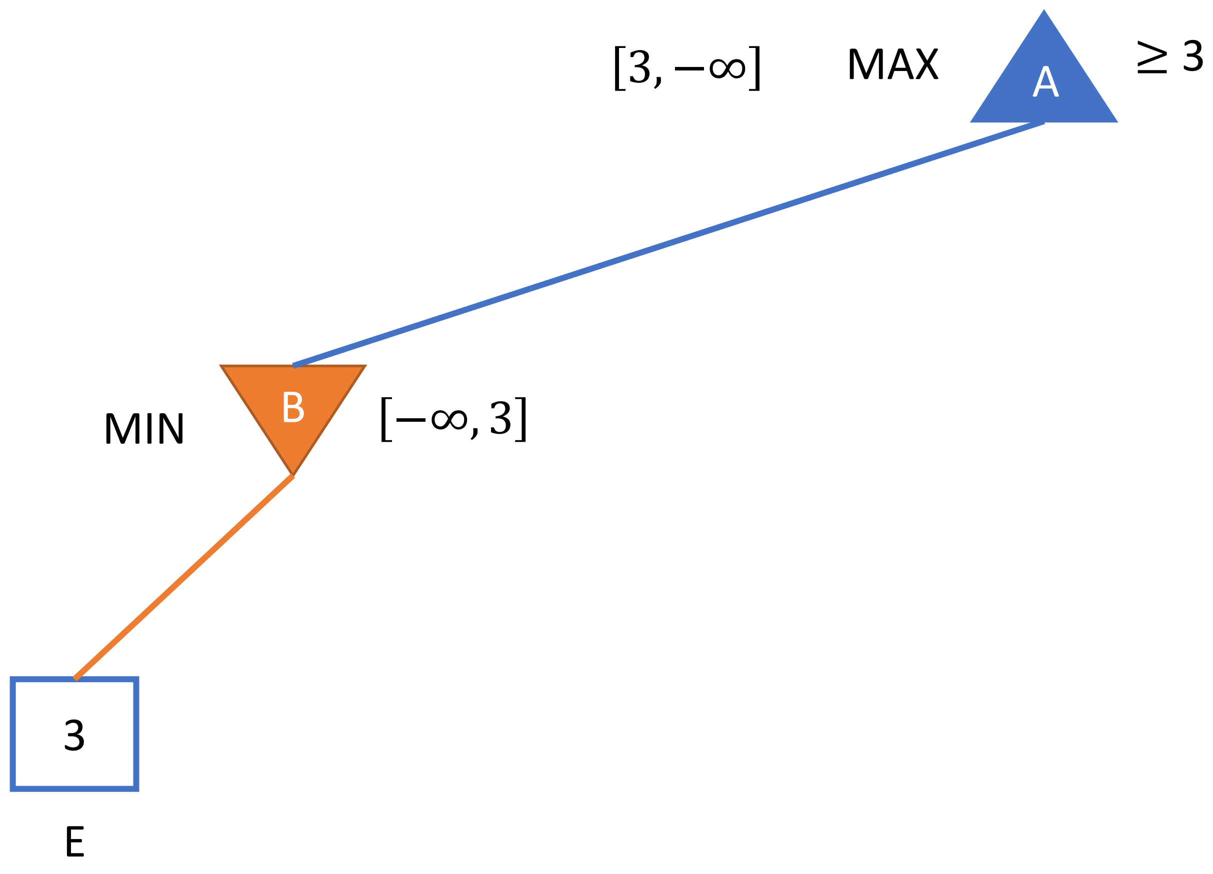
MiniMax Algorithm

[α, β] MAX

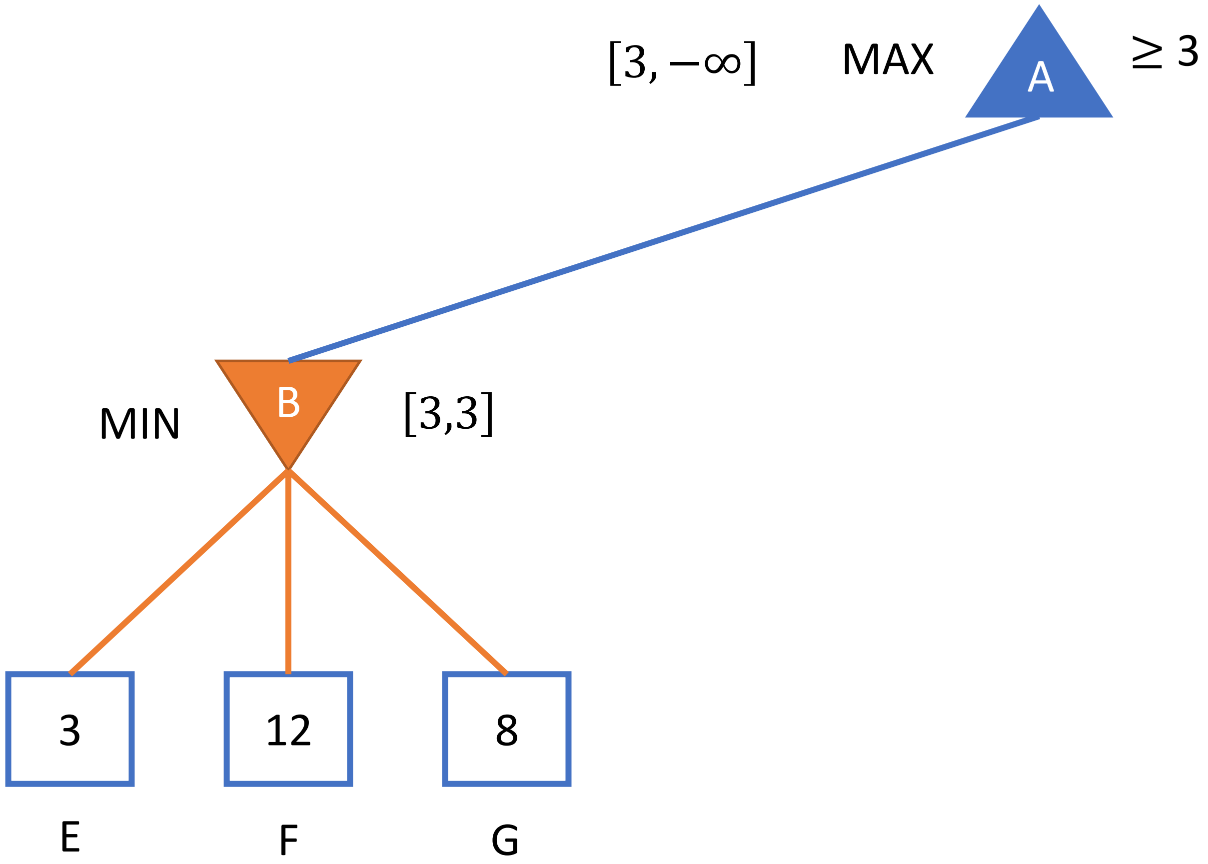
MiniMax Algorithm

$[+\infty, -\infty]$ MAX

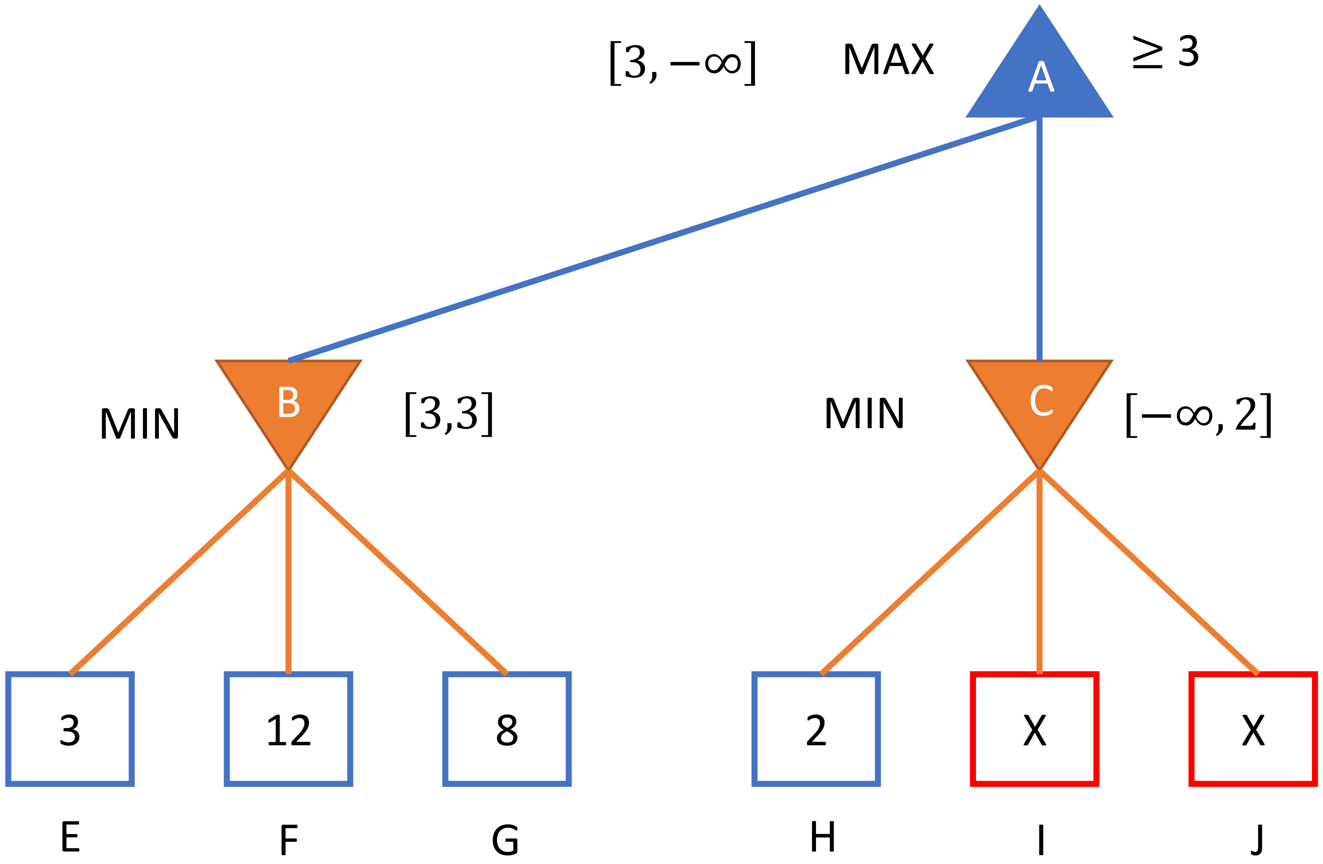
MiniMax Algorithm



MiniMax Algorithm

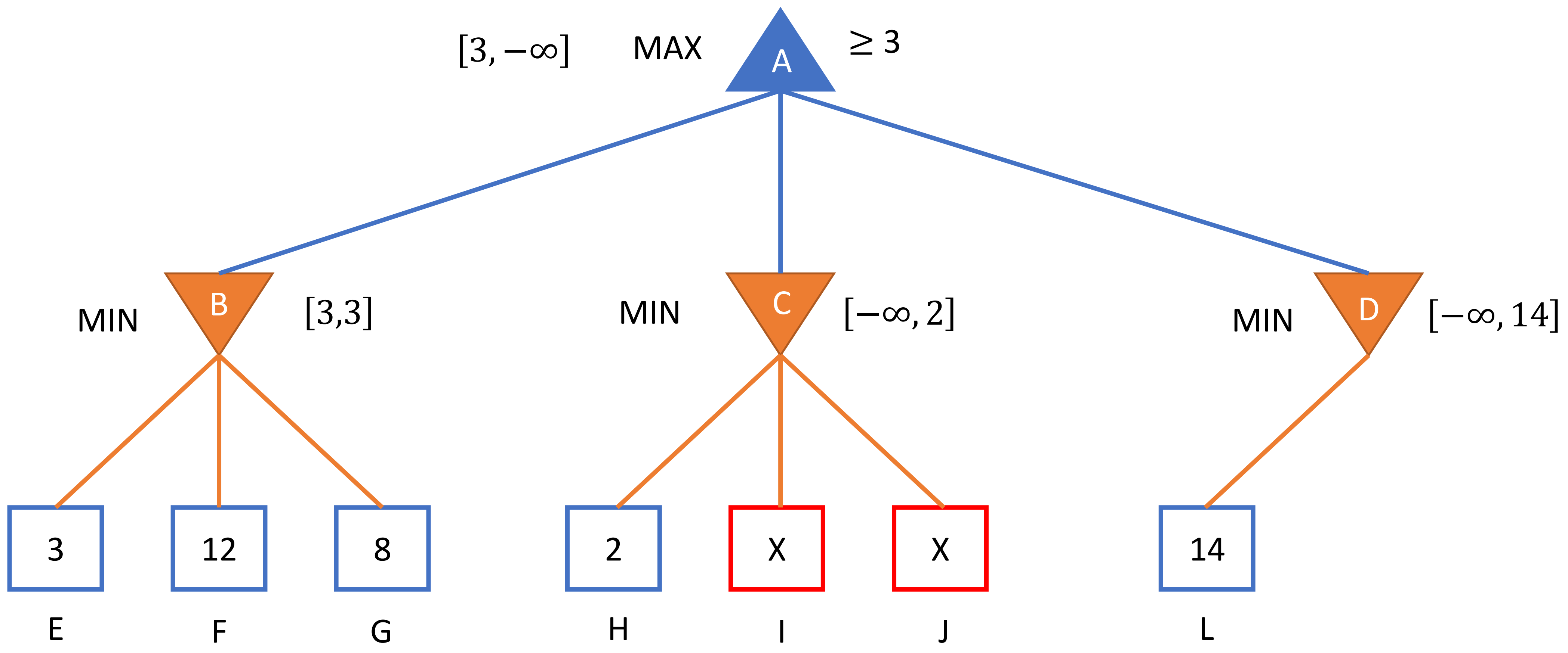


MiniMax Algorithm

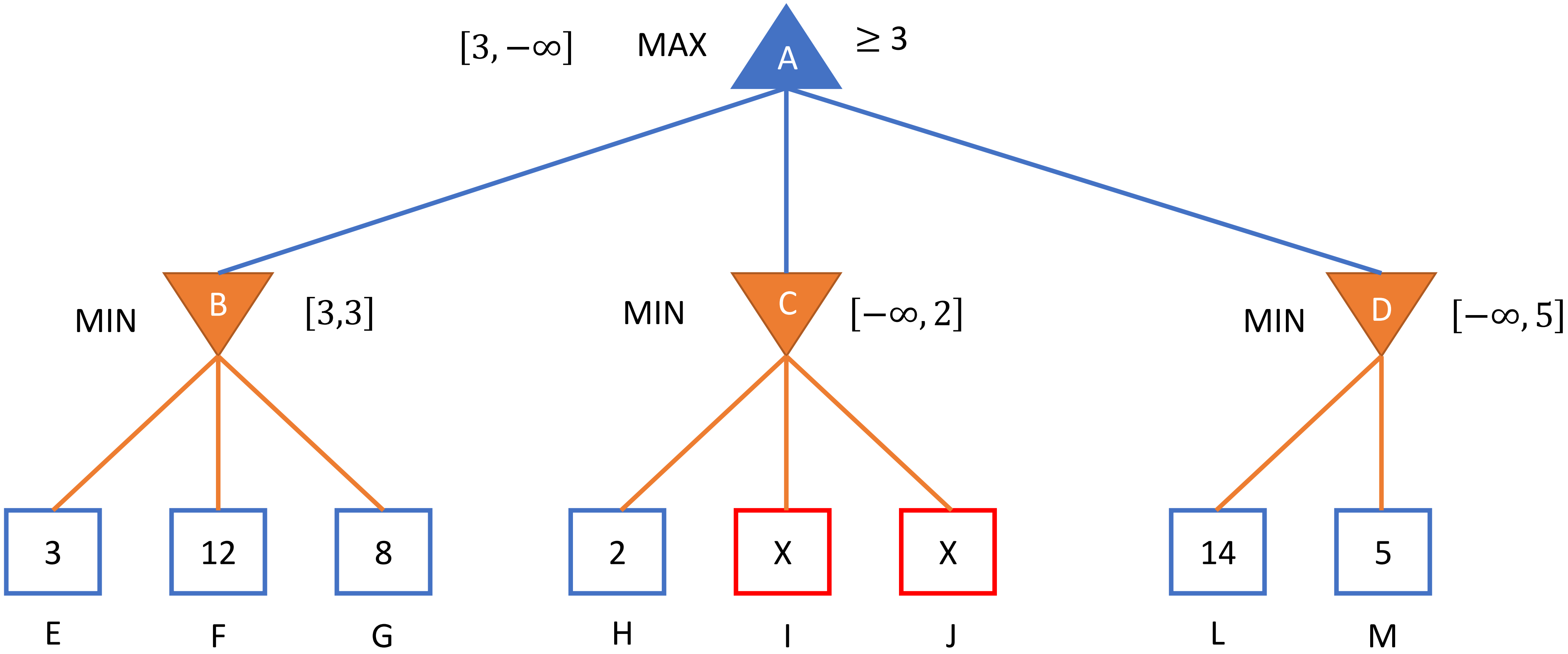


Não olha as outras folhas => $2 < 3$

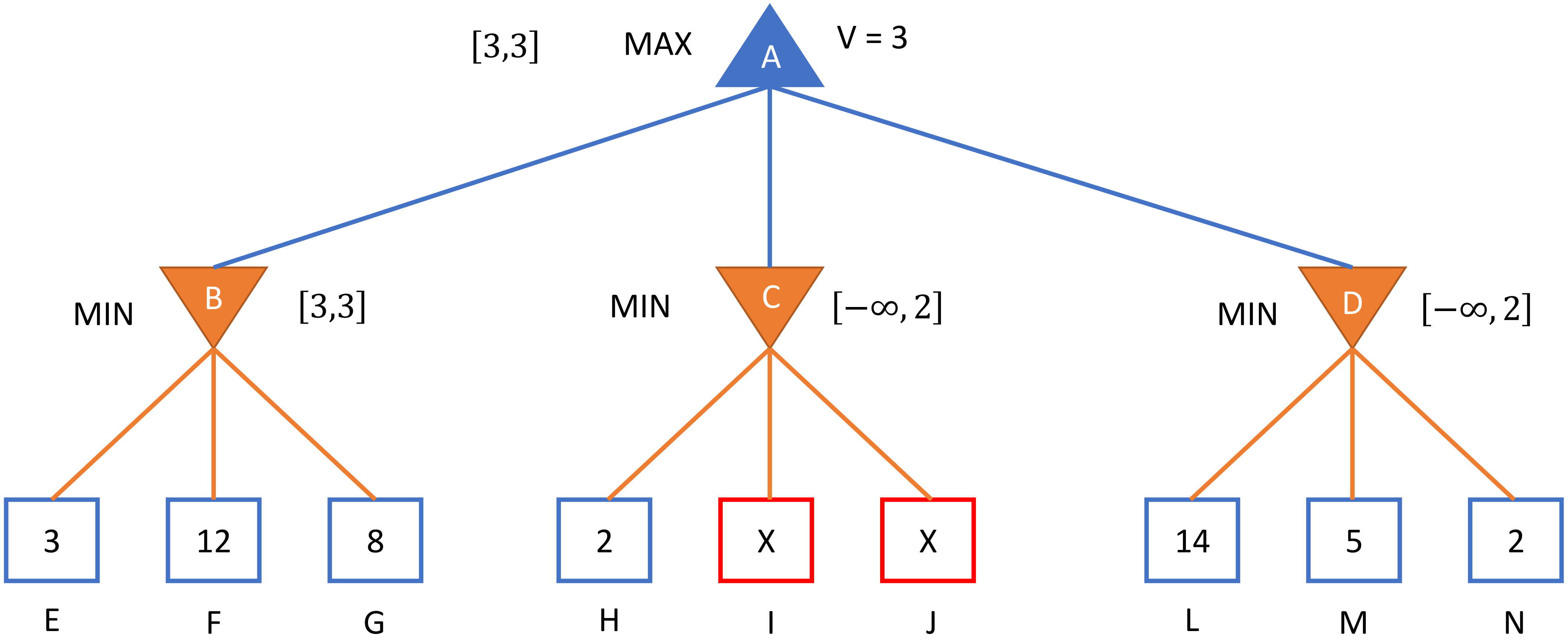
MiniMax Algorithm



MiniMax Algorithm



MiniMax Algorithm



MiniMax Algorithm

Poda (Pruning) α-β

```
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
```

MiniMax Algorithm

Poda (Pruning) α-β

function MIN-VALUE($state, \alpha, \beta$) **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**
    $v \leftarrow$ MIN(v, MAX-VALUE(s, α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

MiniMax Algorithm

Poda (Pruning) α-β

- Supondo que utiliza melhor ordem

Alfa-beta: examina $\boldsymbol{O(b^{\frac{m}{2}})}$ nós para escolher melhor movimento

- Examinando em ordem aleatória

Alfa-beta: examina $\boldsymbol{O(b^{\frac{3m}{4}})}$ nós para escolher melhor movimento

MiniMax Algorithm

Poda (Pruning) α-β

- Supondo que utiliza melhor ordem

Alfa-beta: examina $\boldsymbol{O}(\boldsymbol{b}^{\frac{m}{2}})$ nós para escolher melhor movimento

- Examinando em ordem aleatória

Alfa-beta: examina $\boldsymbol{O}(\boldsymbol{b}^{\frac{3m}{4}})$ nós para escolher melhor movimento

MiniMax: examina $\boldsymbol{O}(\boldsymbol{b}^{m})$ nós para escolher melhor movimento

MiniMax Algorithm

Poda (Pruning) α-β

Xadrez

Minimax:

- 5 jogadas à frente
- Humano médio: 6 a 8

Alfa-beta:

- 10 jogadas à frente
- Desempenho de especialista

MiniMax Algorithm

Poda (Pruning) α-β

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

MiniMax Algorithm

Poda (Pruning) α-β

- 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!

MiniMax Algorithm

Decisões Imperfeitas

- 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

MiniMax Algorithm

Decisões Imperfeitas

- 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: Shannon (1950)
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

MiniMax Algorithm

Decisões Imperfeitas

- Substituir a <u>função Utilidade</u> por uma <u>***função Heuristica***</u> e <u>teste Objetivo</u> por <u>***teste de Corte***</u>;
- A função de avaliação retorna uma ***estimativa*** da utilidade esperada;
- Nós Não-Terminais transformam-se em nós Terminais para a MINIMAX ou Poda α-β

MiniMax Algorithm

Decisões Imperfeitas

- Função Heuristica deve ordenar os estados terminais da mesma forma que a Função Utilidade:

  ex. 1-vitorias 2-Empates 3- derrotas

- A computação não deve demorar tempo demais

- Deve estar fortemente relacionada com as chances reais de vitória

MiniMax Algorithm

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) = w1*f1+w2*f2+...+wn*fn

  Ex. Os pesos *w* no xadrez poderiam ser o tipo de pedra do xadrez (Peão-1, ..., Rainha-9)

  Os valores de *f* poderiam ser o número de cada peça no tabuleiro.
- Função Não-Linear de Peso de propriedade do nó:

  Ex. O valor de um Par de Bispo ser maior que o dobro do valor do um Bispo

MiniMax Algorithm

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) = w1*f1+w2*f2+...+wn*fn
  Ex. Os pesos *w* no xadrez poderiam ser o tipo de pedra do xadrez (Peão-1, ..., Rainha-9)
  Os valores de *f* poderiam ser o número de cada peça no tabuleiro.
- Função Não-Linear de Peso de propriedade do nó:
  Ex. O valor de um Par de Bispo ser maior que o dobro do valor do um Bispo

Escolha crucial: compromisso entre precisão e eficiência

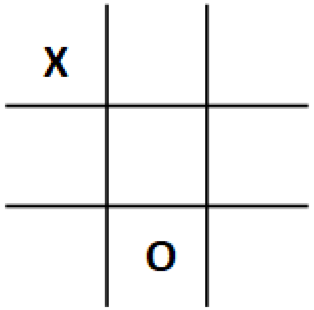
MiniMax Algorithm

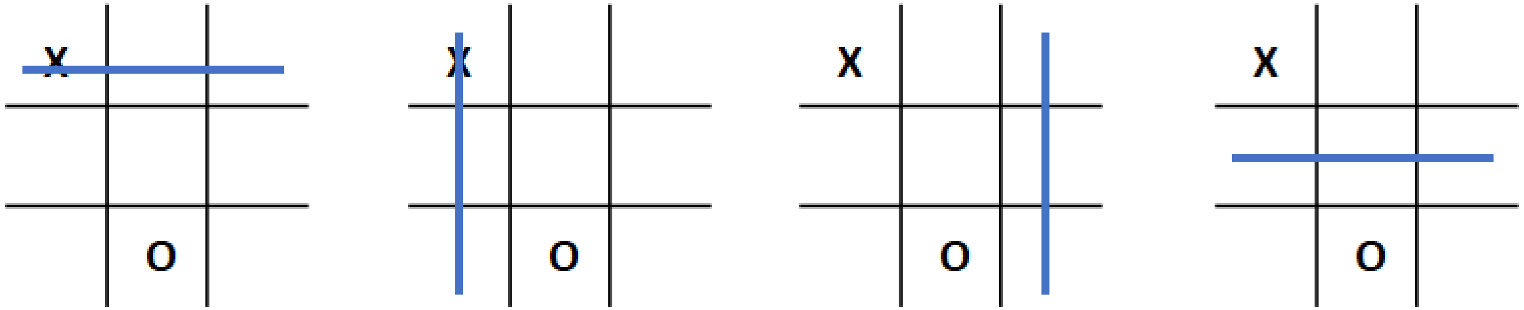
Funções de Avaliação

- Caracteristicas e Pesos não fazem parte das regras do jogo
- Foram aprendidos ao longo dos anos
- Pesos podem ser aprendidos utilizando técnicas de aprendizado automáticas

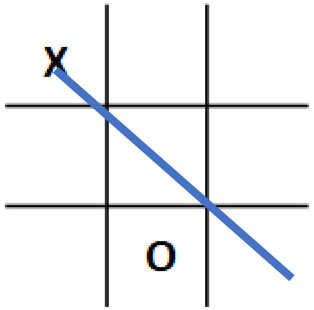
MiniMax Algorithm

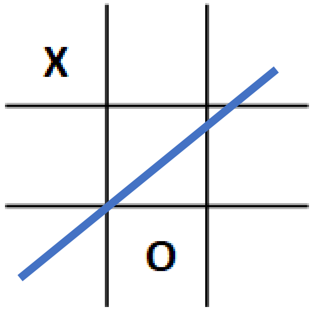
Funções de Avaliação





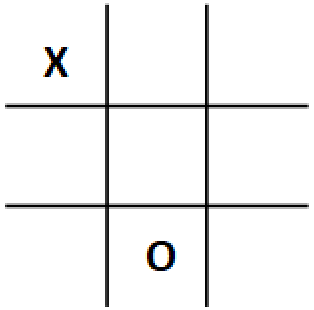
X tem 6 possibilidades

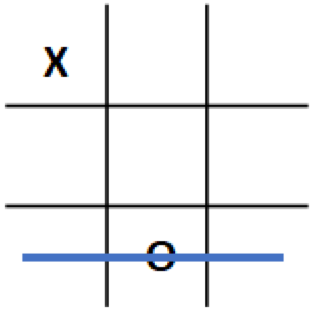


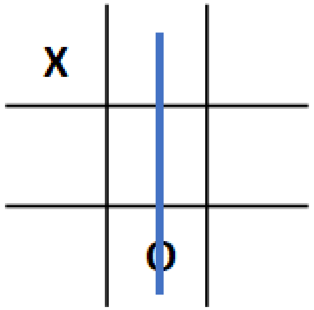


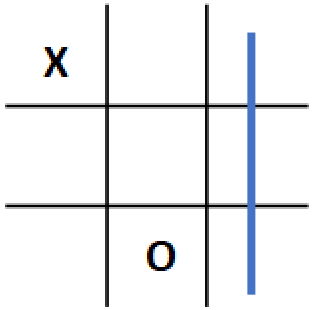
MiniMax Algorithm

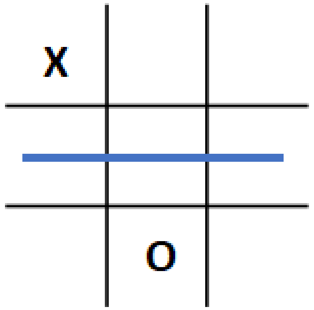
Funções de Avaliação

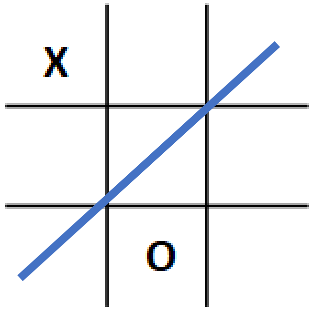








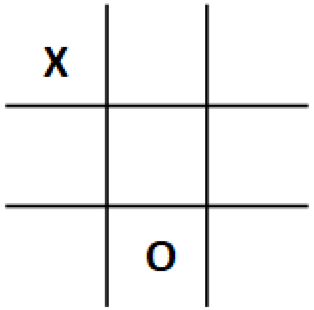




O tem 5 possibilidades

MiniMax Algorithm

Funções de Avaliação



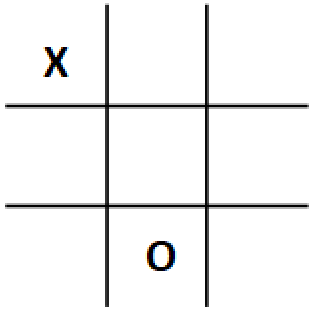
X tem 6 possibilidades

O tem 5 possibilidades

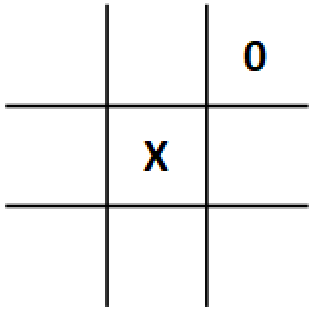
H= 6 -5 = 1

MiniMax Algorithm

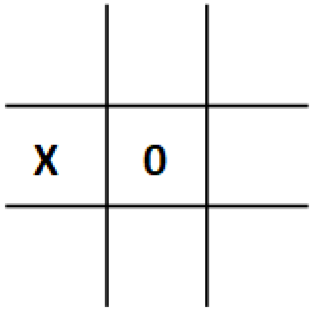
Funções de Avaliação



H= 6 - 5 = 1

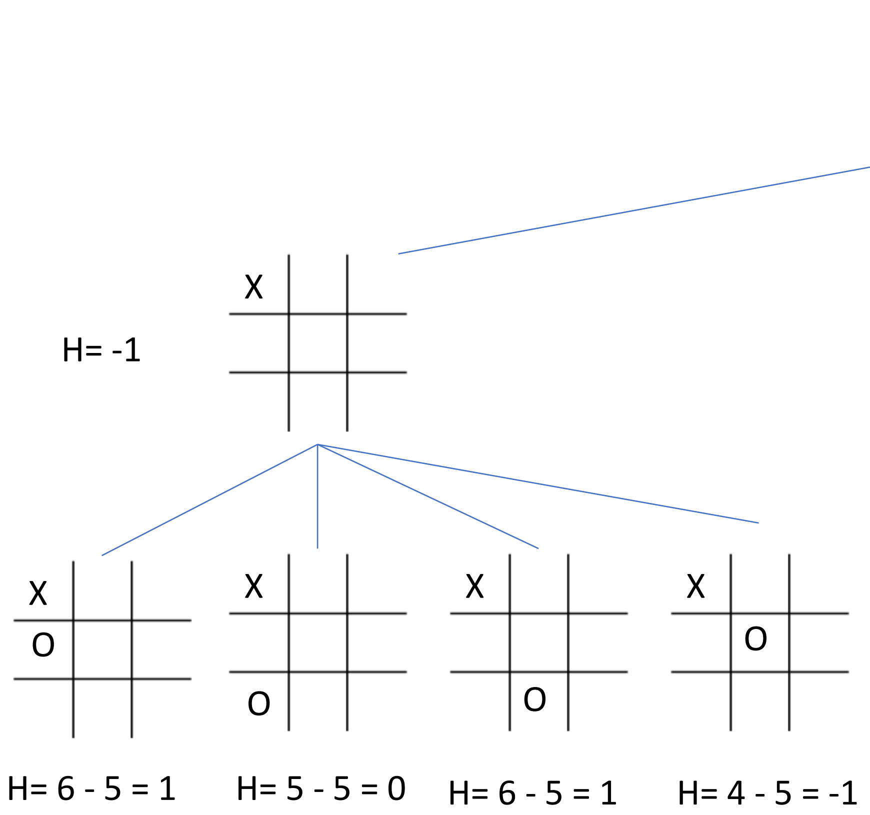


H= 5 - 4 = 1

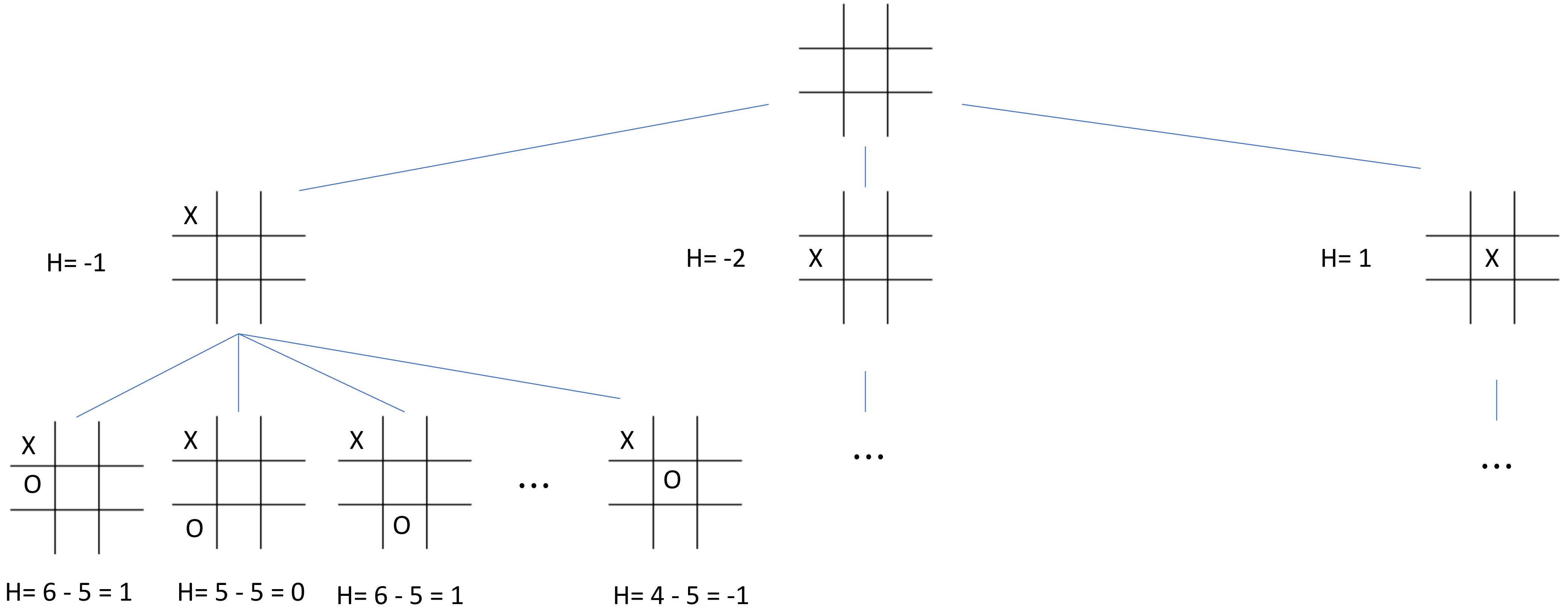


H= 4 - 6 = -2

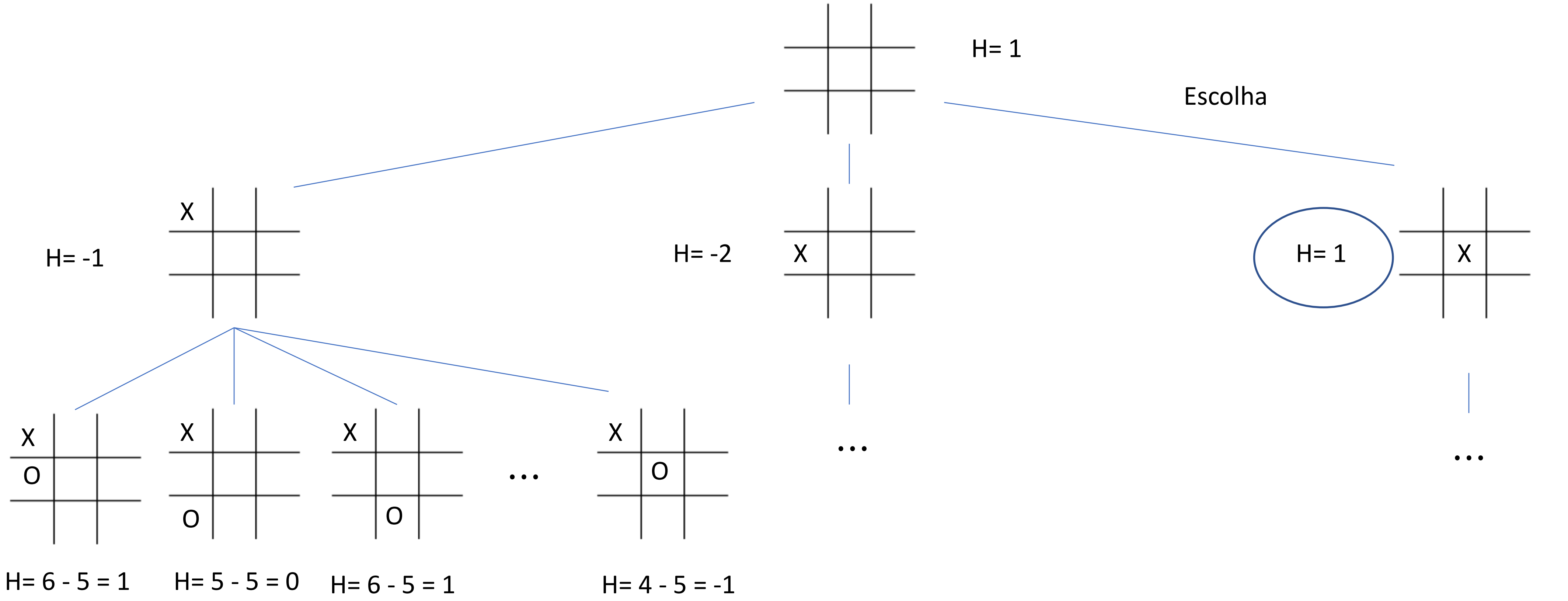
MiniMax Algorithm



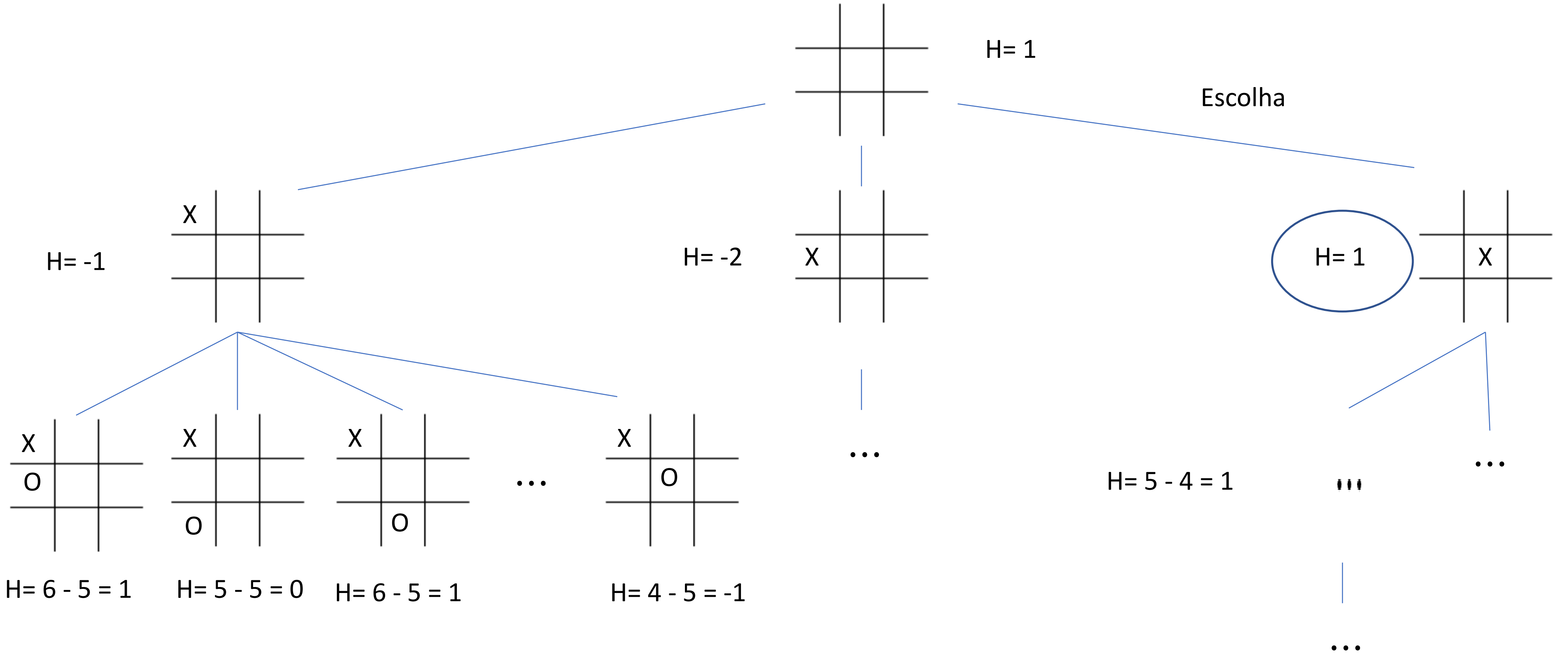
MiniMax Algorithm



MiniMax Algorithm



MiniMax Algorithm



MiniMax Algorithm

Implementação

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

Naïve Bayes

Teorema de Naive Bayes

- Acurária do exame - 90%

Pessoas que tem a doença - 90% o resultado é SIM

Pessoas que não tem a doença - 90% o resultado é NÃO

Se o resultado do exame for SIM – Qual a probabilidade da pessoa ter, realmente, a doença????

1% da população tem a doença

1000 pessoas

Pessoas que tem a doença - 10 pessoas

9 verdadeiros positivos

1 falso positivo

Pessoas que não tem a doença - 990 pessoas

891 verdadeiros negativos

99 falso negativo

108 pessoas – resultado TEM a doença

$$\frac{9}{108} = 8{,}3\%$$

Daniel Nogueira

dnogueira@ipca.pt