Inteligência Artificial - Notas de aula

Raoni F. S. Teixeira

Aula 4 - Busca Competitiva

1 Introdução

Essa aula trata do algoritmo de busca Minmax em jogos. Um jogo é um ambiente competitivo em que o agente não sabe de antemão o que irá acontecer e por isso não pode operar usando os algoritmos das aulas anteriores.

2 Jogo de dois jogadores

O jogo ocorre em um ambiente determinístico e observável, como xadrez ou damas. Os jogadores alternam jogadas até o fim da partida; o vencedor recebe recompensas e o perdedor, penalidades. Em caso de empate, os dois jogadores ganham a mesma quantidade de pontos.

Formalmente, seja $\mathcal{S}$ um conjunto de estados do ambiente e $\mathcal{A}$ um conjunto de jogadas (ações permitidas), um jogo é um objeto com os 6 (seis) componentes a seguir:

1. um estado inicial $s_i \in \mathcal{S}$ que especifica como o jogo começa;
2. uma função J, $J(s) \mapsto [\text{jogador}_1, \text{jogador}_2]$, de controle de alternância que especifica para cada estado s qual jogador deve jogar;
3. uma função A, $A(s) \mapsto \mathcal{A}$, que devolve as jogadas válidas em um estado s;
4. um modelo de transição T, $T(s, a) \mapsto s'$, que especifica o estado alcançado (s') após a jogada a ser realizada em um estado s;
5. uma função F, $F(s) \mapsto [\text{Verdadeiro}, \text{Falso}]$, que verifica se o jogo terminou ou não. Os estados em que o jogo acaba são chamados de terminais. A função devolve o valor Verdadeiro apenas se o estado s é terminal e

6. uma função U, $U(s) \mapsto [-\infty, \infty]$, que avalia um estado terminal s e atribui um valor númerico considerando o desempenho de um jogador referência. Os valores atribuídos aos jogadores são iguais e opostos (a recompensa é positiva e a penalidade é negativa). No jogo da velha, por exemplo, a função devolve 1 para indicar que, por exemplo, o primeiro jogador venceu e -1 para indicar que seu adversário venceu. O valor 0 (zero) indica empate.

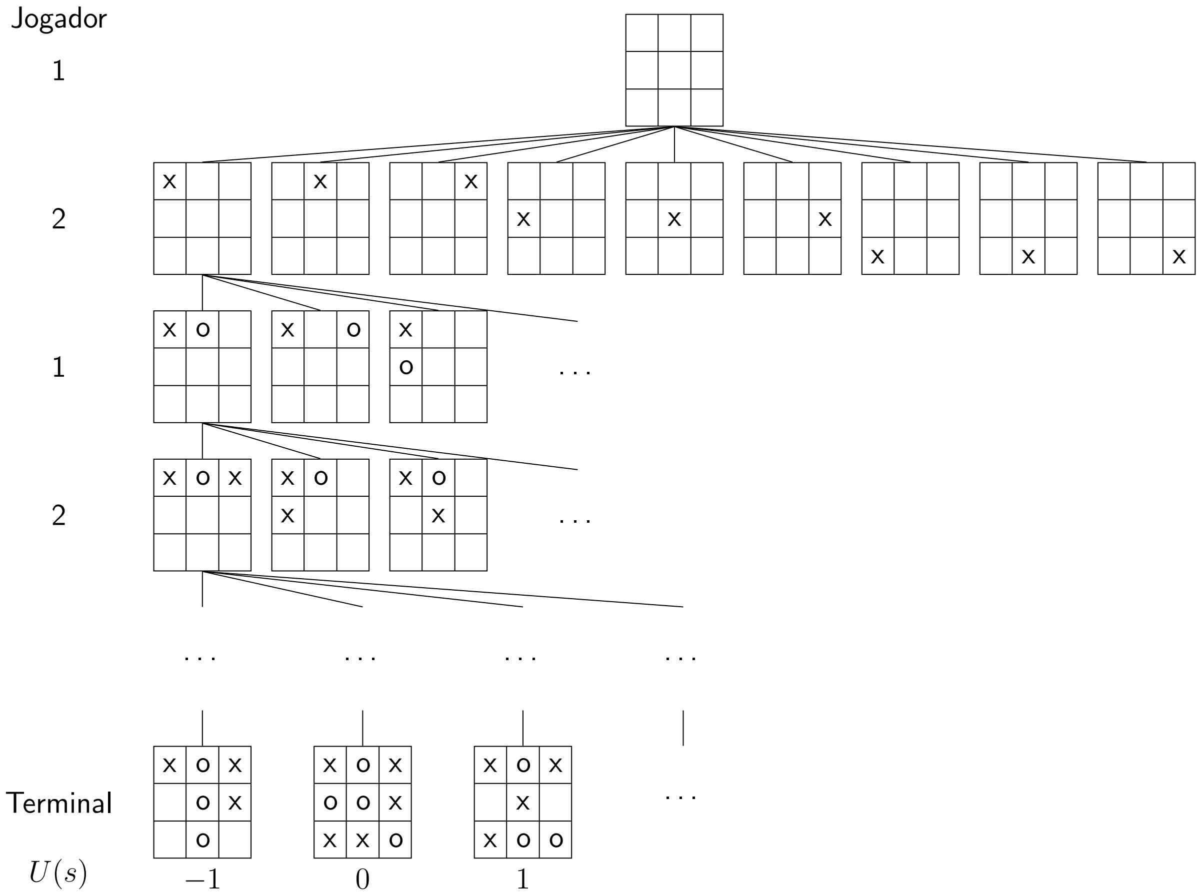


Figura 1: Árvore de partidas do jogo da velha. Os nós são os estados do jogo e cada caminho da raiz à folha é uma partida.

A Figura 1 mostra a árvore de partidas do jogo da velha, com o estado inicial na raiz e os estados terminais nas folhas. Cada caminho da raiz a uma folha é uma sequência completa de jogadas. A função U atribui valores aos nós terminais para indicar vitória (1), derrota (-1) ou empate (0).

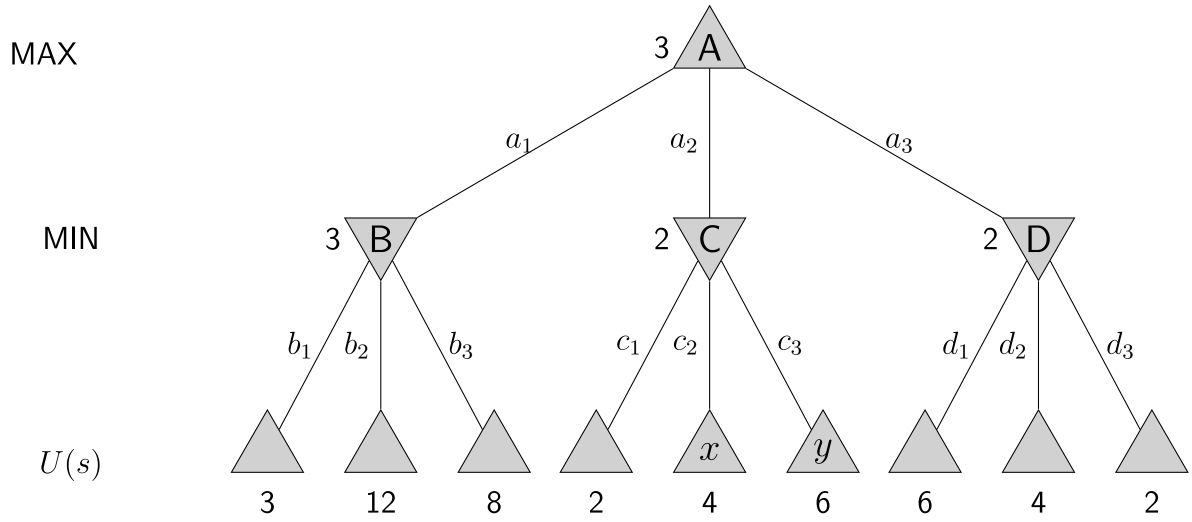


Figura 2: Árvore de um jogo de dois jogadores. $\triangle$ e $\triangledown$ indicam o primeiro jogador (agente) e seu adversário. Os números nas folhas são os valores da função $U(s)$.

3 Minmax

Nesse ambiente, o agente deve escolher suas jogadas levando em conta que seu adversário sempre faz a jogada que minimiza U. Este é o problema de busca Minmax em que agente e o adversário se alternam maximizando e minimizando a função U:

$$\mathrm{Minmax}(s) = \begin{cases} U(s), & \text{se } F(s) \text{ é Verdadeiro.} \\ \max\limits_{a \in A(s)} \mathrm{Minmax}(T(s,a)), & \text{se } J(s) \text{ é o agente.} \\ \min\limits_{a \in A(s)} \mathrm{Minmax}(T(s,a)), & \text{se } J(s) \text{ é o adversário.} \end{cases} \quad (1)$$

A função Minmax avalia todas as combinações possíveis e devolve a jogada com maior valor de U, considerando a resposta do adversário.

A Figura 2 apresenta a árvore de decisões de um jogo de dois jogadores. Os estados não terminais (A, B, C e D) têm três ações possíveis cada. Os valores nas folhas representam a função U(s): o jogador $\triangle$ busca maximizar U, enquanto o jogador $\triangledown$ tenta minimizá-lo. O jogador $\triangle$ assume que seu adversário irá escolher as ações b_1, c_1 e d_3 que minimizam U e escolhe a ação a_1 que possui o maior valor de minmax.

O número de estados do jogo é exponencial na profundidade da árvore minmax. Na prática, 2 (duas) estratégias são utilizadas para reduzir o tempo do minmax: a) reduzir a profundidade e b) cortar/podar partes da árvore que não afetam o resultado.

O controle de profundidade depende de uma função heurística que avalia estados não-terminais do jogo. O algoritmo a seguir é uma implementação da Equação 1 em uma única função com controle de profundidade. Eval(s) é uma função heurística de avaliação de estados que estima quão bom é o estado s para o agente. O algoritmo recebe um estado s e um nível de profundidade p e devolve o valor minmax da jogada ótima.

Minmax(s, p):

1 **if** $F(s) = \text{Verdadeiro}$ *ou* $p \leq 0$ **then**
2 | **return** Eval(s)
3 $v \leftarrow -\infty$
4 **foreach** *jogada* $a \in A(s)$ **do**
5 | $v \leftarrow$ max($v, -1 *$ Minmax($T(s, a)$, $p - 1$))
6 **return** v

Algoritmo 1: Minmax com controle de profundidade.

A implementação utiliza a mudança de sinal (multiplicação por -1 na linha 5) para alterar a perspectiva dos jogadores min e max (min $\{A\} = -$max $\{-A\}$).

Embora o Minimax seja eficaz, a enumeração de todas as jogadas é cara computacionalmente. A poda alfa-beta resolve esse problema, como veremos a seguir.

4 Poda alpha-beta

A poda alfa-beta (α-β) otimiza o Minimax ao eliminar subárvores que não influenciam o resultado final e reduzir o número de nós avaliados.

Este é o caso das jogadas c_2 e c_3 na sub-árvore C da Figura 2 que não alteram o valor de Minmax:

$$\begin{aligned}
\text{Minmax}(\mathsf{A}) &= \mathsf{max}(\mathsf{min}(\mathsf{B}), \mathsf{min}(\mathsf{C}), \mathsf{min}(\mathsf{D})) \\
&= \mathsf{max}(\mathsf{min}(3, 12, 8), \mathsf{min}(2, x, y), \mathsf{min}(6, 4, 2)) \\
&= \mathsf{max}(3, \mathsf{min}(2, x, y), 2) \\
&= 3 && (\forall \text{ valores de x, y})
\end{aligned}$$

A ideia da poda é armazenar o maior e o menor valor encontrados ao longo dos caminhos da árvore em duas variáveis α e β e utilizá-los para interromper a busca. A Figura 3 ilustra esse princípio para um nó k qualquer. Se o Jogador tiver uma escolha melhor que k no mesmo nível (por exemplo, n) ou em qualquer ponto acima na árvore (por exemplo, m), então o Jogador nunca passará para k. Assim, uma vez que tenhamos descoberto o suficiente sobre a árvore, podemos podar o nó k.

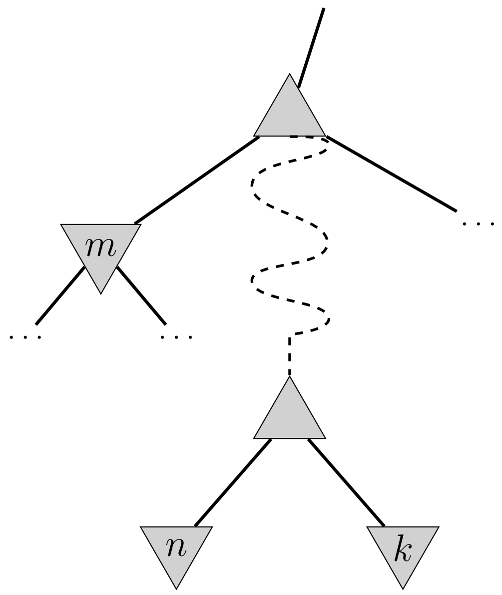


Figura 3: Poda alpha-beta. Se m ou n são melhores que k, então k pode ser podado.

A Figura 4 detalha as etapas da poda alfa-beta, com valores de α e β indicados entre colchetes. Inicialmente, $\alpha = -\infty$ e $\beta = \infty$. A sequência de diagramas (a a f) mostra como a poda elimina subárvores ao comparar valores mínimos e máximos, interrompendo a busca em ramos irrelevantes

A Figura 4a mostra a árvore após a descoberta da primeira folha da sub-árvore de B, cujo valor é 3. Nesse ponto, β vale 3, pois é o menor valor em B. A Figura 4b mostra a descoberta da segunda folha de B, que tem valor 12 — como o jogador adversário (min) evitaria esse movimento, o valor de β em B continua sendo 3. A Figura 4c mostra a descoberta da terceira folha de B cujo valor é 8 — como já conhecemos todos as folhas de B, então o valor de α é também 3.

A Figura 4d mostra a árvore após a descoberta da primeira folha da sub-árvore C cujo o valor é 2. O valor β em C é 2. Como B vale 3, então o jogador max nunca escolheria C. Assim, não faz sentido olhar para os outros estados sucessores de C e os estados seguintes são podados.

A Figura 4e mostra a árvore após a descoberta da primeira folha da sub-árvore D cujo valor é 6 — β em D vale 6. Como 6 é maior que a melhor alternativa de min 3, então precisamos continuar explorando os estados sucessores de D. Observe também que agora temos limites para todos os sucessores da raiz, então o valor da raiz também é no máximo 6. A Figura 4f mostra segundo sucessor de D vale 4, então novamente é preciso continuar explorando. O terceiro sucessor vale 2, então agora D vale exatamente 2. A decisão ótima do jogador max

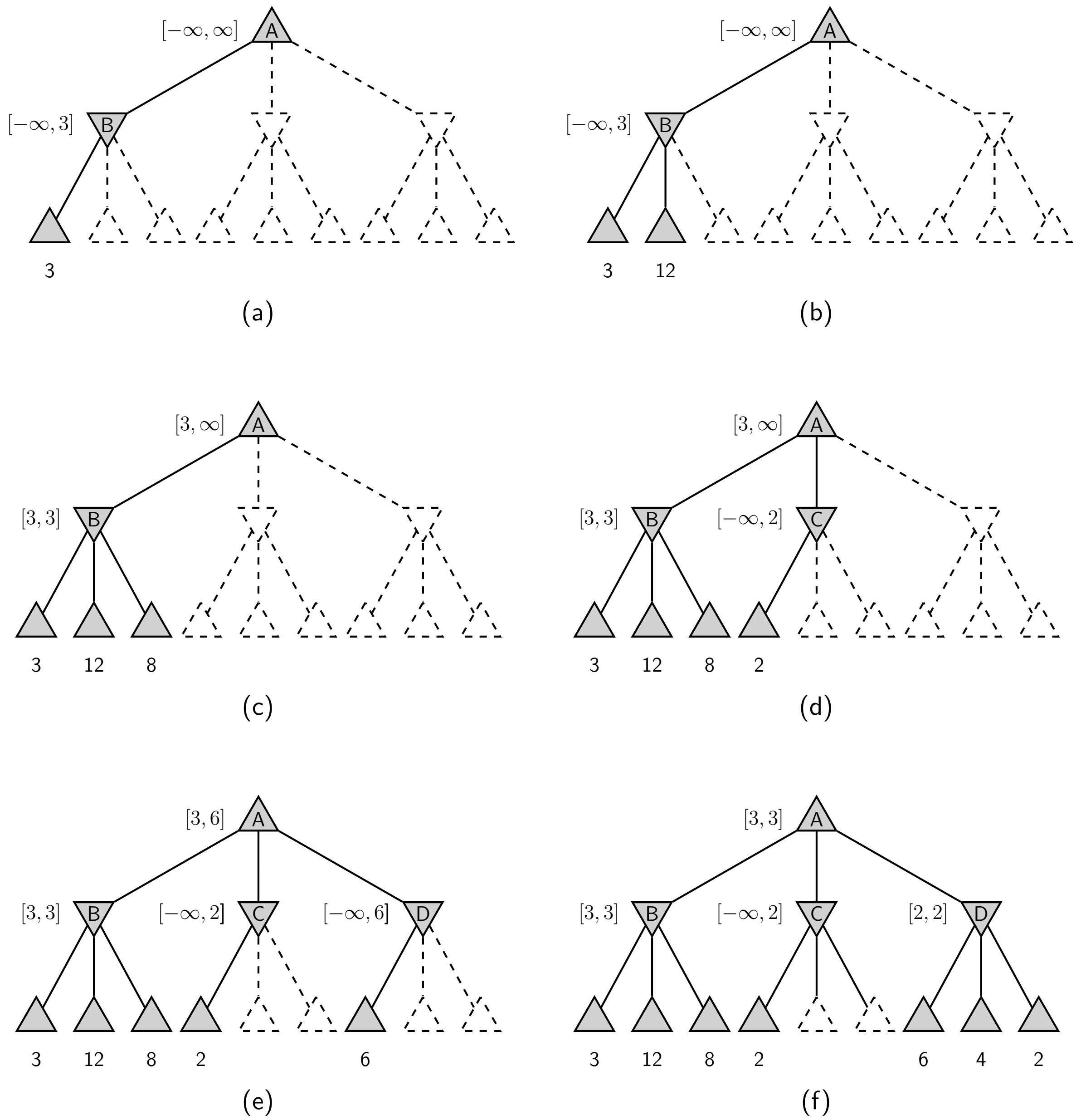


Figura 4: Simulação da poda alpha-beta para árvore da Figura 1. Os valores de alfa e beta estão indicados entre colchetes e são inicialmente $-\infty$ e ∞. A busca é interrompida na sub-árvore C — os estados tracejados na parte (f) foram podados.

em A é escolher a jogada que leva para sub-árvore B (jogada a_1 na Figura 2).

As Figuras 4d, 4e e 4f mostram que a ordem da descoberta de jogadas afeta a quantidade de nós podados pelo método alfa-beta. Na Figura 4d, a ordem de descoberta é 2, 4 e 6. Como o menor valor foi descoberto primeiro, foi possível interromper a busca. O mesmo não acontece nas Figuras 4e e 4f cuja ordem é 6, 4 e 2.

Os algoritmos 2 e 9 apresentam uma implementação da poda alfa-beta. Novamente, Eval é uma função heurística que avalia estados sobre a perspectiva do agente. Para usar o algoritmo, basta chamar a função **MAXPoda**: **MaxPoda**($s_0, 10, -\infty, \infty$).

MaxPoda(s**,** p, α, β**):**

if $F(s) = \text{Verdadeiro}$ *or* $p \leq 0$ **then**
    return Eval(s)
$v \leftarrow -\infty$;
foreach *jogada* $a \in A(s)$ **do**
    $v \leftarrow$ max(v, MinPoda($T(s, a)$, $p - 1$, α, β));
    $\alpha \leftarrow$ max(v, α);
    if $\alpha \geq \beta$ **then**
        return α;
return v

Algoritmo 2: Max com controle de profundidade e poda α-β.

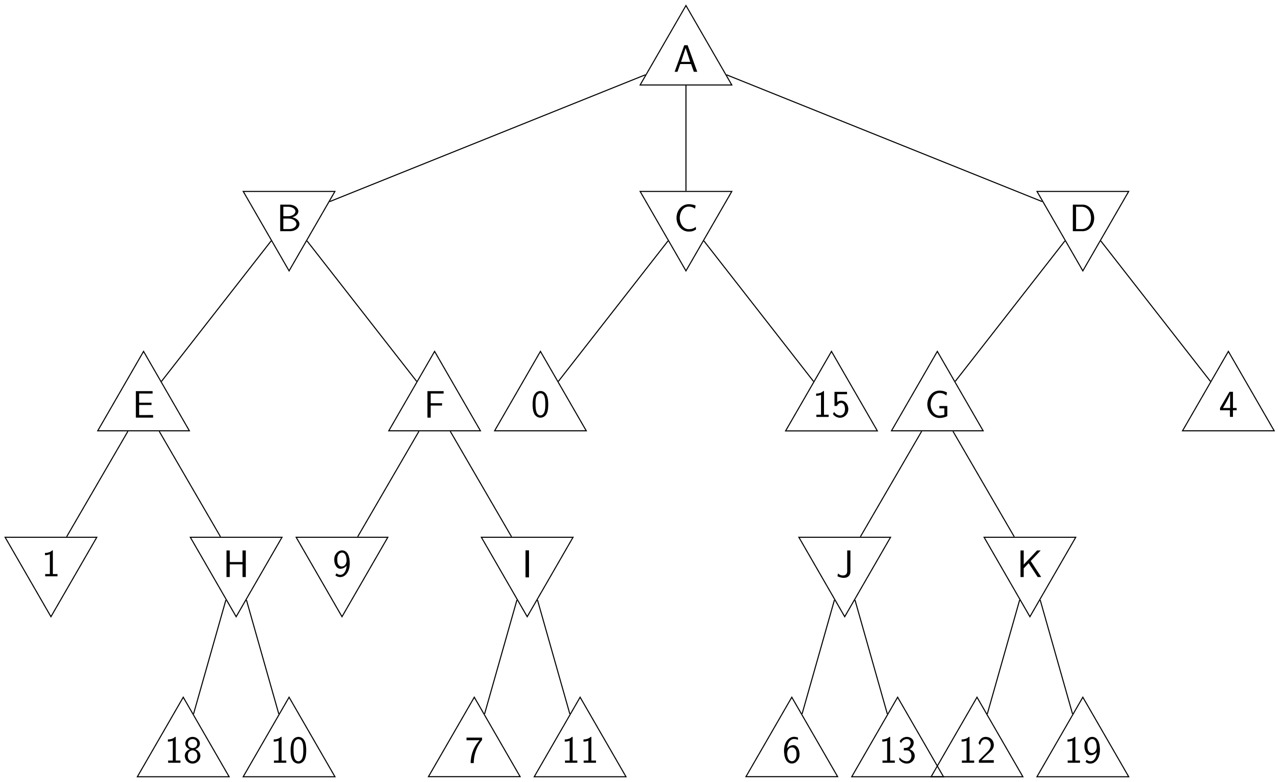
MinPoda(s**,** p, α, β**):**

if $F(s) = \text{Verdadeiro}$ *or* $p \leq 0$ **then**
    return Eval(s)
$v \leftarrow +\infty$;
foreach *jogada* $a \in A(s)$ **do**
    $v \leftarrow$ min(v, MaxPoda($T(s, a)$, $p - 1$, α, β));
    $\beta \leftarrow$ min(v, β);
    if $\alpha \geq \beta$ **then**
        return β;
return v

Algoritmo 3: Min com controle de profundidade e poda α-β.

5 Exercícios

1. Como a busca minimax modela a competição em jogos de dois jogadores?
2. Calcule o valor de Minmax de cada nó árvore a seguir e indique os nós podados pela poda alfa-beta.
3. De que forma a poda alfa-beta melhora a eficiência da busca minimax?


A
B
C
D
E
F
0
15
G
4
1
H
9
I
J
K
18
10
7
11
6
13
12
19