Lista de Exercícios de Busca Adversária

Nome: _____

Busca AND-OR

1. Defina o que são os nós OR e AND.

> **Solution:**
>
> - Nós OR são nós as ações escolhidas pelo próprio agente num ambiente determinístico.
> - Nós AND são as saídas escolhidas pelo ambiente a partir de ações do agente.

2. Dado o algoritmo AND-OR-SEARCH (Figura 1), explique por que ele retorna *failure* quando encontra um estado idêntico a um estado presente no caminho até a raiz da árvore, evitando a criação de ciclos.

function AND-OR-GRAPH-SEARCH($problem$) **returns** *a conditional plan, or failure*
  OR-SEARCH($problem$.INITIAL-STATE, $problem$, [ ])

---

function OR-SEARCH($state, problem, path$) **returns** *a conditional plan, or failure*
  if $problem$.GOAL-TEST($state$) **then return** the empty plan
  if $state$ is on $path$ **then return** *failure*
  for each $action$ **in** $problem$.ACTIONS($state$) **do**
    $plan \leftarrow$ AND-SEARCH(RESULTS($state, action$), $problem$, [$state \mid path$])
    if $plan \neq failure$ **then return** [$action \mid plan$]
  return *failure*

---

function AND-SEARCH($states, problem, path$) **returns** *a conditional plan, or failure*
  for each s_i **in** $states$ **do**
    $plan_i \leftarrow$ OR-SEARCH($s_i, problem, path$)
    if $plan_i$ = *failure* **then return** *failure*
  return [**if** s_1 **then** $plan_1$ **else if** s_2 **then** $plan_2$ **else** ... **if** s_{n-1} **then** $plan_{n-1}$ **else** $plan_n$]

Figura 1: Algoritmo AND-OR-SEARCH

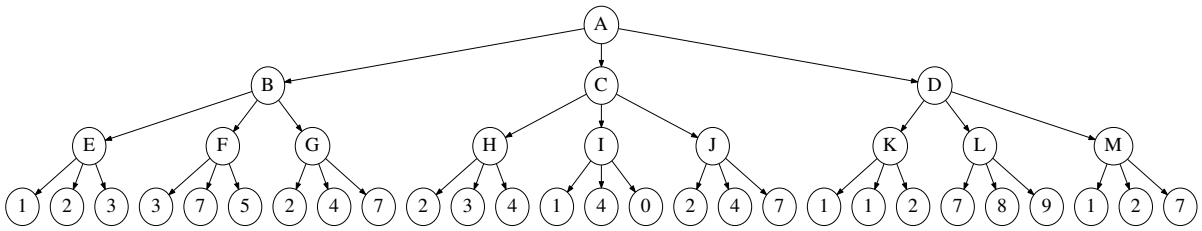
> **Solution:** Isto acontece porque o algoritmo garante que se existe uma solução acíclica, ela deve ser alcançada a partir do estado anterior ao atual. Assim, é garantido que o algoritmo termina em cada espaço de estado finito.

Algoritmo MINIMAX e Alpha-Beta Pruning

3. Diferencie os problemas de Busca Adversária dos problemas de busca com apenas um agente.

> **Solution:** Os problemas de busca vistos anteriormente são normalmente formados por um agente e um objetivo, sendo as ações do agente voltadas para alcançar esse objetivo. Na busca adversária há um ambiente competitivo no qual há mais de um agente e os agentes possuem objetivos conflitantes.

4. Considere a árvore abaixo:



(a) Utilizando o algoritmo MINIMAX (Figure 2), descubra os valores relacionados aos nós A-M.

> **Solution:** A-4, B-3, C-4, D-2, E-3, F-7, G-7, H-4, I-4, J-7, K-2, L-9, M-7

(b) É possível utilizar alpha-beta pruning e diminuir o número de nós visitados? Se sim, indique os nós folha que não seriam visitados usando a notação <nó pai> <nó filho>, como $H2$. Se não, explique por que.

> **Solution:** F7, F5, G7, I0, J7, DL, DM

(c) Quantos nós poderiam ser podados se houvesse uma ordenação dos nó-folhas?

> **Solution:** 16

5. Assinale verdadeiro ou falso para cada uma das afirmações abaixo, justificando a resposta no caso de a afirmação ser falsa.

(a) O algoritmo minimax (Figure 2) realiza uma busca em largura completa da árvore do jogo.

(a) **Falso**

(b) Alpha-beta search (Figura 3) retorna o mesmo movimento que o minimax normalmente traria.

(b) **Verdadeiro**

(c) No alpha-beta pruning, o α representa o menor valor encontrado dentre as escolhas feitas ao longo do caminho da árvore, enquanto o β representa o maior valor.

(c) **Falso**

(d) O algoritmo Minimax leva em consideração o valor minimax de cada nó, fazendo com que o algoritmo obtenha uma estratégia ótima dado uma árvore de jogo.

(d) **Verdadeiro**

Solution:

(a) O algoritmo minimax realiza uma busca em profundidade completa da árvore do jogo.

(c) α representa o maior valor dentre as escolhas feitas ao longo do caminho e β representa o menor valor.

```
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
```

Figura 2: Minimax algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) 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), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) 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) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Figura 3: Alpha-beta search