Departamento de Ciência de Computadores - FCUP
Exame de Inteligência Artificial / Sistemas Inteligentes

**Duração: 1 hora** **ÉPOCA NORMAL** **PARTE 1**

Data: 1 de Junho de 2017

**1)** Ao comparar algoritmos de pesquisa em árvores, normalmente mede-se o número de nós expandidos. No pior caso, quantos nós são expandidos para cada estratégia abaixo, assumindo que o fator de ramificação é $b$ com solução em profundidade $d$? (Não use a notação "big-O": O(...). Utilize $b$ e $d$ para contar o número de nós)

(a) Busca em largura (BFS)

(b) Busca em profundidade (DFS)

(c) Busca limitada em profundidade (limite = d)

(d) Busca iterativa limitada em profundidade (IDFS)

**2)** Qual é a definição de "heurística admissível"?

**3)** O algoritmo guloso ("greedy") que usa heurística é completo? Justifique.

**4)** Quais são as consequências de se implementar o algoritmo A* removendo estados já visitados?

**5)** Seja o grafo da Figura 1. Cada nó é representado por uma letra e sua respectiva heurística em relação ao nó G.

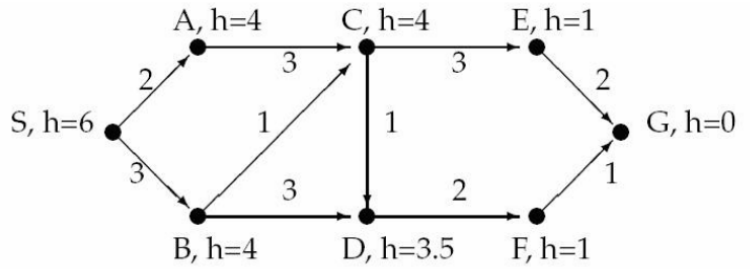


Figura 1: Figura para a pergunta 5

(a) Encontre o menor caminho entre S e G e preencha o quadro abaixo, de acordo com a ordem dos nós expandidos, anotando seus custos.

| iteração | nó expandido | Lista de nós ao final desta iteração |
|---|---|---|
| 0 | | S |
| 1 | S | A=..., B=... (preencher custos) |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |

(b) Faça este exemplo ser inadmissível mudando a heurística em algum dos nós do grafo. Qual é o nó que escolheu e qual foi o valor de heurística que escolheu?

(c) A busca continua completa?

(d) Qual é o efeito da mudança que fez?

**6)** A busca IDA* usa a função de custo (g+h) para determinar se vai continuar a explorar um determinado caminho na árvore. Qual é o novo valor da função de custo utilizado pelo IDA* a cada iteração?

**7)** A cada iteração, o "simulated annealing" avalia somente um movimento dentre os possíveis.

(a) Como é que o próximo movimento é escolhido?

(b) Responda Falso ou Verdadeiro: O movimento avaliado é sempre descartado se for pior do que o estado corrente e outro movimento é escolhido.

(c) É possível ficar preso num máximo local usando o algoritmo "simulated annealing"?

**8)** Na árvore da Figura 2, responda:

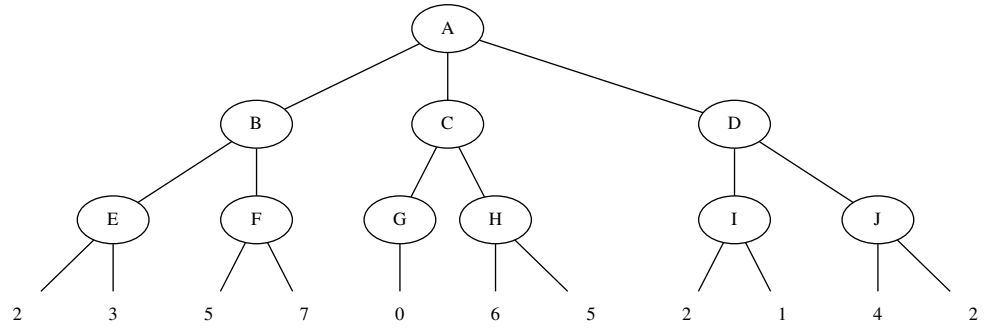


Figura 2: Figura para a pergunta 8

(a) Qual é o valor retornado em C pelo algoritmo minimax?

(b) Quais são os nós cortados pela poda alfa-beta?

(c) Reorganize os nós desta árvore de forma a obter o corte máximo.

**As duas próximas perguntas correspondem à parte prática do teste. Esta parte deve ser respondida apenas por aqueles que não entregaram os dois primeiros trabalhos.**

**(PRÁTICA 1)**
Dado o grafo da Figura 3, explique porque a heurística do nó E não é apropriada para ser utilizada pelo algoritmo A*.

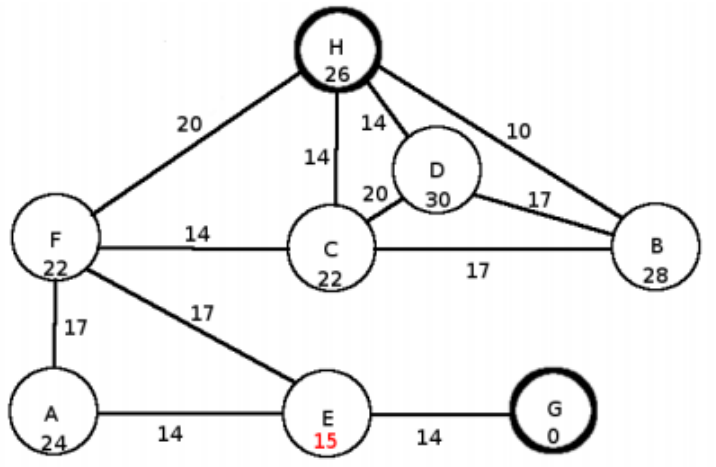


Figura 3: Figura para a pergunta prática 1

**(PRÁTICA 2)**
Sejam os dois algoritmos mostrados nas Figuras 4 e 5. Responda:

**(A)** Qual é a diferença principal entre estes dois algoritmos?

**(B)** Se estes algoritmos forem implementados utilizando a estratégia A*, qual seria a diferença em termos de qualidade da solução? Ilustre com um exemplo.

**(PRÁTICA 3)**
Usando a sua linguagem de programação favorita, escreva o código que verifica se o jogo dos 15 tem solução. Assuma que podemos dar qualquer configuração inicial e qualquer configuração final.

```
function GENERAL_SEARCH(problem, QUEUEING_FN) retorna solucao ou falha
     nodes := MAKE_QUEUE(MAKE_NODE(INITIAL_STATE(problem)))
     loop
        if EMPTY?(nodes) then return falha
        node := REMOVE_FRONT(nodes)
        if STATE(node) = GOAL_TEST[problem] then return node
        successors = empty list
        foreach s = successor(node, Operators)
           add(s,successors)
        end
        nodes := QUEUEING_FN(nodes,successors)
     end
```

Figura 4: Figura para a pergunta (PRÁTICA 2)

```
function GENERAL_SEARCH(problem, QUEUEING_FN) retorna solucao ou falha
     nodes := MAKE_QUEUE(MAKE_NODE(INITIAL_STATE(problem)))
     loop
        if EMPTY?(nodes) then return falha
        node := REMOVE_FRONT(nodes)
        successors = empty list
        foreach s = successor(node, Operators)
           if STATE(s) = GOAL_TEST[problem] then return s
           add(s,successors)
        end
        nodes := QUEUEING_FN(nodes,successors)
     end
```

Figura 5: Figura para a pergunta (PRÁTICA 2)