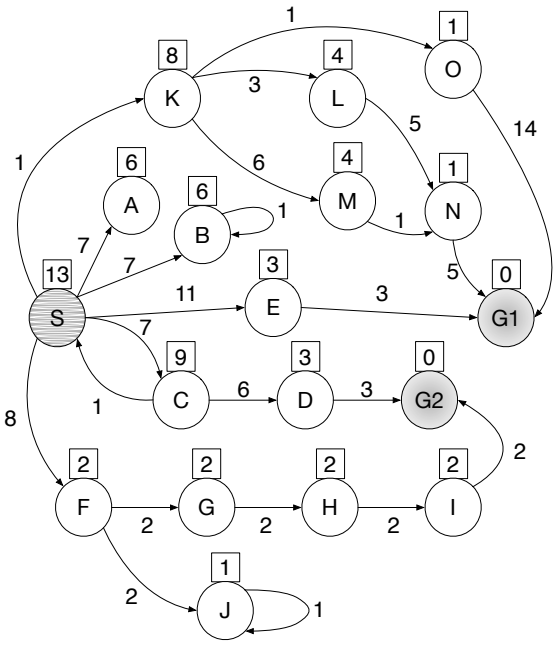
Inteligência Artificial – 2024/25

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

19/Outubro/2024–14:00h – Duração: 2:00h

1º Teste (Sem consulta)
Modelo A

I) [5val] Considere o seguinte grafo de estados de um problema de procura. Os valores apresentados nos arcos correspondem ao custo do operador respectivo. Os valores nos rectângulos correspondem ao valor de uma heurística (estimativa do custo de chegar desse estado ao objectivo). Não se representam os nomes dos operadores, correspondendo cada arco a um operador distinto. Em todos os algoritmos cegos e em caso de empate nos algoritmos informados, assuma que os sucessores de um nó são sempre colocados na fronteira por ordem lexicográfica dos seus nomes, de forma a que o nó mais próximo do início do alfabeto seja selecionado antes dos seus irmãos. Pretende-se encontrar um caminho desde o estado S até G1 ou G2.



a) Indique o caminho encontrado por cada um dos seguintes algoritmos: i) procura em profundidade primeiro (em árvores), ii) procura em profundidade primeiro (em grafos), iii) procura em largura primeiro (optimizada), iv) procura de custo uniforme (em grafos), v) procura sôfrega (em árvores), vi) procura sôfrega (em grafos) e vii) A* (em grafos, na versão que apenas garante a solução óptima para heurísticas consistentes).

b) Considere um espaço de estados infinitamente profundo, com ciclos, factor de ramificação finito, múltiplos estados objectivo e com custos por operador não negativos, possivelmente diferentes entre si. Para cada um dos seguintes algoritmos de procura, indique (S/N) se são completos (C) e se são óptimais (O): i) profundidade primeiro (em árvores), ii) profundidade primeiro (em grafos), iii) largura primeiro (optimizada), iv) custo uniforme (em árvores), v) profundidade limitada (em árvores), vi) aprofundamento progressivo (em árvores). Nesta alínea, cada resposta incorrecta desconta o equivalente a uma resposta correcta, tendo a alínea uma cotação mínima de 0 valores.

II) [3val] Considere a charada-8, que consiste numa moldura com nove posições, oito das quais ocupadas por peças numeradas e a nona vazia. O objetivo é organizar as peças por ordem numérica, como ilustrado na imagem da esquerda. Um operador neste espaço de procura corresponde a empurrar uma peça adjacente à posição vazia, passando a peça a ocupar a posição anteriormente vazia, e ficando vazia a posição anteriormente ocupada pela peça empurrada, como ilustrado nas imagens da direita. Um estado neste espaço é qualquer arranjo de peças na moldura.

a) Qual o tamanho do espaço de estados?

b) Assuma que o custo de cada operador é unitário. Seja (x_i, y_i) a posição actual da peça i, e seja (x_i^g, y_i^g) a sua posição objectivo. Para cada uma das seguintes funções heurísticas, indique se são (A) Apenas admissíveis; (B) Apenas Consistentes; (C) Consistentes e Admissíveis; (D) Nem Consistentes nem Admissíveis. Nesta alínea, cada resposta incorrecta desconta o equivalente a uma resposta correcta, tendo a alínea uma cotação mínima de 0 valores.
   i. h_1 = número de peças correctamente colocadas i.e., na posição objetivo.
   ii. h_2 = número de peças incorrectamente colocadas i.e., que não estão na posição objetivo.
   iii. h_3 = soma das distâncias de Manhattan da posição de cada peça até à sua posição objectivo i.e., $\sum_i(|x_i - x_i^g| + |y_i - y_i^g|)$.
   iv. $h_4 = max(h_1, h_3)$.
   v. $h_5 = min(h_1, h_3)$.
   vi. $h_6 = h_1 + h_2$.

c) Qual ou quais das heurísticas da alínea anterior $(h_1 \ldots h_6)$ seriam as melhores escolhas para usar com o algoritmo A* (assinale com um X na folha de respostas)?

d) Qual das seguintes condições correctamente caracteriza o fator de ramificação b para a charada-8?

   i. $b < 2$ ii. $2 \leq b \leq 4$ iii. $4 \leq b \leq 6$ iv. $6 < b$

III) [4val] Está a chegar a altura de preparar o exame de admissão de matemática. O exame vai ter 6 perguntas, cada uma cobrindo um dos seguintes tópicos:

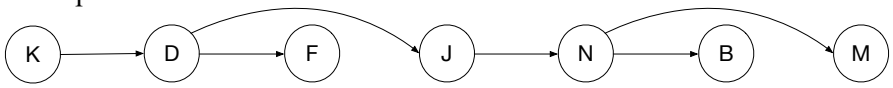
- Q1: Números Complexos
- Q2: Funções Reais
- Q3: Geometria Analítica
- Q4: Trigonometria
- Q5: Álgebra
- Q6: Probabilidades

Existem 7 pessoas na equipa que vai auxiliar o coordenador a preparar o exame: Bernardo, Daniela, Fernando, Joana, Kate, Miguel e Nuno. Cada um deles será responsável por auxiliar o coordenador do grupo na elaboração de uma pergunta. (Mas uma pergunta poderá ter mais do que um membro a auxiliar o coordenador, ou nenhum dos membros da equipa, sendo preparada exclusivamente pelo coordenador). Acontece que os membros da equipa são muito esquisitos e querem que as seguintes restrições sejam satisfeitas:

i. A Daniela (D) não trabalhará numa questão junto com Joana (J).
ii. A Kate (K) deve trabalhar em Números Complexos, Funções Reais ou Geometria Analítica.
iii. O Miguel (M) é muito estranho, por isso só pode contribuir para uma questão de número ímpar.
iv. O Nuno (N) deve trabalhar numa questão que seja antes da questão do Miguel (M).
v. A Kate (K) deve trabalhar numa questão que seja antes da questão da Daniela (D).
vi. O Bernardo (B) deve trabalhar em Álgebra.
vii. A Joana (J) deve trabalhar numa questão que seja depois da questão do Nuno (N).
viii. Se o Bernardo (B) tiver de trabalhar com alguém, não pode ser com o Nuno (N).
ix. O Nuno (N) não pode trabalhar na questão 6.
x. O Fernando (F) não pode trabalhar nas questões 4, 5 ou 6.
xi. A Daniela (D) não pode trabalhar na questão 5.
xii. A Daniela (D) deve trabalhar numa questão que seja antes da questão do Fernando (F).

Iremos tratar deste problema como um problema de satisfação de restrições. As variáveis correspondem a cada um dos membros da equipa – B, D, F, J, K, M N – e os seus domínios são as perguntas – 1, 2, 3, 4, 5, 6.

a) Depois de aplicar as restrições unárias, quais os domínios resultantes para cada variável? Elimine (na folha de resposta) os valores que seriam descartados.
b) Que variável seria escolhida de acordo com a heurística da variável mais constrangida?
c) Normalmente, procederíamos com a variável encontrada na alínea (b), mas, para separar esta questão da anterior (e evitar que erros potenciais se propaguem), vamos começar por atribuir um valor a Miguel. Para a ordenação dos valores, vamos usar a heurística do Valor Menos Restritivo (LCV), em que usamos Verificação para a Frente para calcular o número de valores restantes nos domínios das outras variáveis. Qual é a ordem de valores prescrita pela heurística LCV? Apresente os valores por ordem decrescente de escolha i.e., o valor que seria escolhido pela heurística LCV primeiro.
d) Percebendo que este é um CSP arbóreo, decidimos não executar a pesquisa com retrocesso e, em vez disso, usar o algoritmo eficiente de duas passagens para resolver CSPs arbóreos. Executaremos este algoritmo de duas passagens após aplicar as restrições unárias da alínea (a). Abaixo está a versão linearizada do grafo CSP em estrutura de árvore para ser usado:



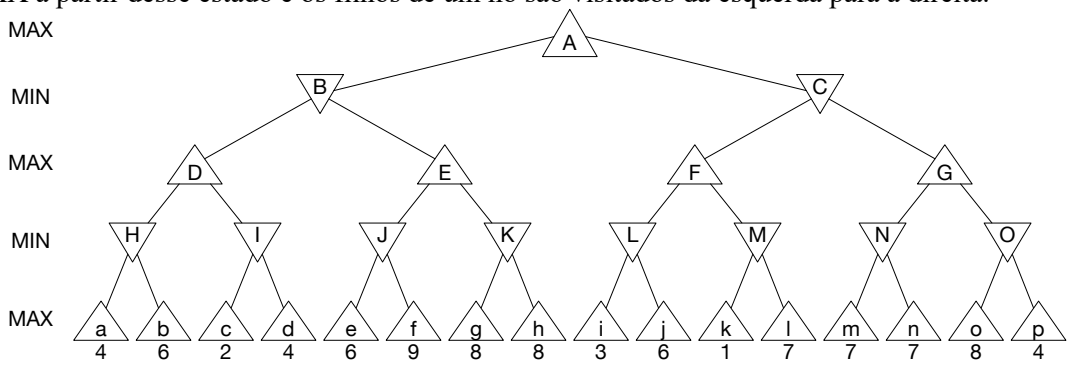
i. Efetue a primeira passagem, removendo os valores inconsistentes. Elimine (na folha de resposta) os valores que seriam removidos (elimine igualmente os valores eliminados pela aplicação das restrições unárias).
ii. Efetue a segunda passagem, atribuindo valores para a solução. Se houver mais do que uma alternativa, atribua o valor mais elevado. Indique os valores atribuídos.

IV) [2,5val] Verifique, usando o algoritmo DPLL, se a seguinte fórmula de lógica proposicional é satisfazível:

$$(A \Rightarrow B) \wedge \big((A \wedge C) \Rightarrow D\big) \wedge (C \Leftrightarrow B) \wedge (\neg C \vee D) \wedge \big(\neg(C \wedge D)\big)$$

Apresente a sequência de paços realizados pelo DPLL indicando, para cada um, o tipo de passo, a variável envolvida e o valor de verdade atribuído. Sempre que for necessário desempatar, escolha as variáveis por ordem alfabética e atribua o valor "Verdadeiro" antes de "Falso".

V) [4val] Considere a árvore de jogo de dois jogadores (MAX e MIN), onde os valores nas folhas são estimativas do ganho para MAX a partir desse estado e os filhos de um nó são visitados da esquerda para a direita.



a) Calcule o valor MINIMAX de cada nó não terminal e indique qual o movimento que MAX deverá escolher.
b) Assumindo que a árvore é percorrida da esquerda para a direita, indique quais os nós que nunca chegariam a ser visitados pelo algoritmo de procura α−β.
c) Considere agora que os nós D, E, F e G da árvore da figura são nós estocásticos (CHANCE) onde cada sucessor é equiprovável. Calcule o valor EXPECTEDMINIMAX dos nós A, B e C.
d) Considere a existência de uma função `EVAL(node)` que devolve o valor minimax correto quando invocada sobre um nó terminal, e que devolve um valor minimax com um erro não superior a 2 quando invocada sobre um nó não terminal. Ou seja, se v for o valor minimax de um nó e a função devolver o valor e, sabemos que $e - 2 \leq v \leq e + 2$ e que $e = v$ no caso de nós terminais. Usando esta função, podemos modificar o algoritmo de procura α−β para fazer mais cortes. Considere a seguinte implementação do algoritmo de procura α−β (apenas a recursão do MAX-VALUE). Acrescente código na secção sombreada indicada com (1) e substitua o código da linha indicada com (2) de forma a que o algoritmo resultante elimine o maior número possível de nós tendo em conta as propriedades da função `EVAL(node)`.

```
function MAX-VALUE(node, α, β) returns a utility value
    e ← EVAL(node)
    if TERMINAL-TEST(node) then return e
    end if
    
                                                    (1)
    v ← -∞
    for s in SUCCESSORS(node) do

        v ← MAX(v, MIN-VALUE(s, α , β))     (2)

        if v ≥ β then return v
        end if
        α ← MAX(α ,v)
    end for
    return v
end function
```

VI) [1,5val] Considere a seguinte implementação do algoritmo de pesquisa em grafos, que pode estar errada!!

```
function GRAPH-SEARCH( problem, frontier ) returns a solution, or failure
    explored ← an empty set
    node ← node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    frontier ← INSERT(node, frontier)
    loop do
        if EMPTY?(frontier) then return failure
        end if
        node ← POP( frontier )
        if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
        end if
        ADD node.STATE to explored
        frontier ← INSERT-ALL(EXPAND(node,problem),frontier)
    end loop
end function
```

Para cada uma das seguintes alíneas, indique se ela é verdadeira ou falsa, justificando de forma concisa a resposta.

a) A chamada a `EXPAND(node,problem)` pode ocorrer várias vezes sobre nós com o mesmo estado.
b) O algoritmo não é completo.
c) O algoritmo pode devolver uma solução sub-ótima.
d) A implementação está incorreta, mas nenhum dos problemas acima ocorrerá.
e) A implementação está correta.

VII) [Bónus: até 4val] Para cada alínea, indique se ela é verdadeira (V) ou falsa (F). Cada resposta incorrecta desconta o equivalente a uma resposta correcta. A pergunta tem uma cotação mínima de 0 valores.

Num problema de pesquisa, a solução devolvida pelo algoritmo de procura de custo uniforme em grafos pode mudar se:

a) Multiplicarmos o custo de cada ação por um valor α: $0 < \alpha < 1$.
b) Multiplicarmos o custo de cada ação por um valor α: $1 < \alpha < 2$.
c) Adicionarmos ao custo de cada ação um valor constante C.

Assuma que estamos a usar o algoritmo A* em grafos com uma heurística h consistente. Seja p o nó que está prestes a ser expandido. Ao expandir p, descobrimos que exatamente um dos seus descendentes corresponde ao estado objetivo G e que o custo para atingir G através de p é K. Considere que $g(p)$ denota o custo do caminho até p que levou à inserção de p na fronteira. Então:

d) O caminho encontrado para atingir G através de p é um caminho mais curto.
e) O caminho encontrado para atingir G através de p é um caminho que garantidamente não é mais comprido do que o caminho mais curto acrescido de $K - g(p)$.
f) O caminho encontrado para atingir G através de p é um caminho que garantidamente não é mais comprido do que o caminho mais curto acrescido de $K - g(p) + h(p)$.
g) O caminho encontrado para atingir G através de p é garantidamente o caminho mais curto para o objetivo através de p.

Sejam $h_1(s)$ e $h_2(s)$ duas heurísticas consistentes usadas pelo algoritmo A* em grafos para minimizar o custo do caminho num grafo. Assuma que não ocorrem empates na fronteira. Se $h_1(s) \leq h_2(s)$ para todo o s, então:

h) O A* usando $h_1(s)$ poderá encontrar um caminho mais curto do que o A* usando $h_2(s)$.
i) O A* usando $h_2(s)$ poderá encontrar um caminho mais curto do que o A* usando $h_1(s)$.
j) O A* usando $h_1(s)$ não irá expandir mais nós do que o A* usando $h_2(s)$.
k) O A* usando $h_2(s)$ não irá expandir mais nós do que o A* usando $h_1(s)$.

O algoritmo de procura α−β:

l) pode não encontrar a estratégia ótima do minimax.
m) corta o mesmo número de sub-árvores independentemente da ordem pela qual os estados sucessores são expandidos.
n) geralmente requer mais tempo de execução do que minimax na mesma árvore de jogo.
o) tem uma complexidade espacial de $O(bm)$ onde b é o fator de ramificação e m a profundidade máxima da árvore de jogo.

Considere um jogo adversarial de soma zero. MAX é jogado por um agente computacional suficientemente rápido para realizar a procura minimax até ao fim do jogo e joga de acordo com os movimentos encontrados. É a vez do MAX jogar, e o agente devolve uma vitória com algum valor positivo para MAX. Assim, temos que:

p) o MAX só tem a vitória garantida se MIN também jogar com a estratégia minimax.
q) o MAX só tem a vitória garantida se MIN jogar uma estratégia determinística.
r) se soubermos que MIN joga de forma uniforme e aleatória em cada jogada, então o MAX não está necessariamente a maximizar o retorno ao jogar o que o agente indica.

FIM!

Inteligência Artificial – Folha de Respostas

19/Out/2024 – 14:00 **DI/FCT/UNL, 1º Teste, 2024/25, Duração: 2h** **Modelo: _____**

Nome:	**Número:**

Ia)	
i	**Não termina (S→B→B→B ....)**
ii	**S→C→D→G2**
iii	**S→E→G1**
iv	**S→K→M→N→G1**
v	**Não termina (S→F→J→J ....)**
vi	**S→F→G→H→I→G2**
vii	**S→K→L→N→G1**

Ib)	**C**	**O**
i	**N**	**N**
ii	**N**	**N**
iii	**S**	**N**
iv	**S**	**S**
v	**N**	**N**
vi	**S**	**N**

IIa)	9!

II	**b)**	**c)**
i	**D**	
ii	**C**	
iii	**C**	**X**
iv	**D**	
v	**A**	
vi	**D**	

IIId)	**ii**

III a)

B	**D**	**F**	**J**	**K**	**M**	**N**
■	1	1	1	1	1	1
■	2	2	2	2	■	2
■	3	3	3	3	3	3
■	4	■	4	■	■	4
5	■	■	5	■	5	5
■	6	■	6	■	■	■

IIIb)	**B**
IIIc)	**5, 3, 1**

III d) ii.

B	**5**	**J**	**6**	**M**	**5**
D	**2**	**K**	**1**	**N**	**4**
F	**3**				

III d) i.

B	**D**	**F**	**J**	**K**	**M**	**N**
■	1	1	■	1	1	1
■	2	2	2	■	■	2
■	■	3	3	■	3	3
■	■	■	4	■	■	4
5	■	■	5	■	5	■
■	■	■	6	■	■	■

IV

	tipo	**variável**	**valor**		**tipo**	**variável**	**valor**
1	Pure	**A**	Falso	**6**	Pure	**C**	Falso
2	Split	**B**	Verdadeiro	**7**	-	-	-
3	Unit	**C**	Verdadeiro	**8**	-	-	-
4	Unit	**D**	Verdadeiro	**9**	-	-	-
5	Backtrack	**B**	Falso	**10**	-	-	-

V

a)	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**
	4	**4**	**3**	**4**	**8**	**3**	**7**	**4**	**2**	**6**	**8**	**3**	**1**	**7**	**4**

a) Mov. de MAX	**B**	**b)**	d Kgh j l GNmnOop	**c)**	**A:**	**3**	**B:**	**3**	**C:**	**2**

d (1)

```
if e - 2 ≥ β then
    return e - 2
end if
```

d(2)

```
v ← MAX(v,MIN-VALUE(s,MAX(α,e-2),MIN(β,e+2)))
```

VI	[V/F]	Justificação:
a)	**V**	Apesar do nome, o código apresentado está a implementar o algoritmo de procura em árvore, não verificando se os estados dos nós a explorar já estão na variável `explored`. Assim, temos os problemas habituais do algoritmo de procura em árvore, nomeadamente poder ter várias chamadas de `EXPAND(node,problem)` sobre nós com o mesmo estado, não garantir uma solução ótima (por exemplo, se a fronteira funcionar como uma LIFO, este algoritmo seria idêntico ao algoritmo de procura em profundidade em árvore, cuja solução devolvida depende da ordem pela qual os ramos são explorados, podendo não ser a ótima) e, em geral, ser incompleto (por exemplo, se a fronteira funcionar como uma LIFO, pode entrar em ciclos infinitos).
b)	**V**	
c)	**V**	
d)	**F**	
e)	**F**	

VII	**a)**	**b)**	**c)**	**d)**	**e)**	**f)**	**g)**	**h)**	**i)**	**j)**	**k)**	**l)**	**m)**	**n)**	**o)**	**p)**	**q)**	**r)**
[V/F]	**F**	**F**	**V**	**F**	**V**	**V**	**F**	**F**	**F**	**V**	**F**	**F**	**F**	**F**	**V**	**F**	**F**	**V**