UNIVERSIDADE DE ÉVORA
ESCOLA DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

T1-IA

Gustavo Oliveira nº46395, Berke Balci nº64498, Semiha Çetintaş nº64751

Évora, 19 de junho de 2025

1 Questão 1 -

Considere o seguinte problema: Um Agente A tem como objetivo mover um Robot da entrada 'E' de um labirinto até à saída 'S', após apanhar 2 objetos que só podem ser apanhados na seguinte ordem: primeiro o objeto 'a' e depois o objeto 'b'. O Robot pode mover-se para cima, baixo, esquerda e direita, mas não pode ir para as casas marcadas com um x. Se o Robot estiver numa casa com um objeto, pode apanha-lo se for o objeto 'a' ou se for o objeto 'b' e já tiver apanhado o objeto 'a'.

(a) Represente em Prolog o estado inicial e o estado final para cada um dos 2 exemplos.

Exemplo 1 :

```
estado_inicial((pos(1,1), objetos([]))).
estado_final((pos(2,7), objetos([a,b]))).
```

Exemplo 2 :

```
estado_inicial((pos(1,1), objetos([]))).
estado_final((pos(5,7), objetos([a,b]))).
```

(b) Represente em Prolog os operadores de transição de estados para este problema.

```

% Movimento
transicao((pos(X,Y), objetos(Objs)), mover(D), (pos(NX,NY), objetos(Objs))) :-
    direcao(DX,DY,D),
    NX is X + DX,
    NY is Y + DY,
    posicao_valida(pos(NX,NY)).

% Coleta de Objetos
transicao((pos(X,Y), objetos([])), apanhar(a), (pos(X,Y), objetos([a]))) :- objeto(a,
    pos(X,Y)).
transicao((pos(X,Y), objetos([a])), apanhar(b), (pos(X,Y), objetos([a,b]))) :- objeto(b,
    pos(X,Y)).
```

(c) Apresente o código em Prolog do algoritmo de pesquisa não informada mais eficiente a resolver este problema. Para justificar a escolha do algoritmos deve apresentar o número e uma estimativa do número de nós visitados e em memoria para cada algoritmo.

As estimativas de **nós visitados** e **nós em memórias** memória podem ser obtidas através dos cálculos das **complexidades temporais** e **espaciais**.

Breadth-First Search (BFS)

- Tempo: $O(b^{d+1})$
- Espaço: $O(b^{d+1})$

Depth-First Search (DFS)

- Tempo: $O(b^m)$
- Espaço: $O(b \cdot m)$

Vamos começar por determinar os valores de **b** (Branching Factor), **m** (profundidade máxima explorável) e **d** (Profundidade da solução de menor custo) .

Branching Factor:

Para esse problema é 4 possíveis mover-se para direita, esquerda, cima e baixo. Assim, podemos considerar que o branching factor é:

$$b = 4 \tag{1}$$

Profundidade da solução de menor custo (d):

Uma solução para o problema, como menor custo, envolve :

- Sair de E até a
- Ir de a até b
- Ir de b até S

Portanto,

$$d = 3 \tag{2}$$

profundidade máxima explorável (m):

profundidade máxima do espaço de estados.

$$m = 7 \tag{3}$$

Breadth-First Search (BFS)

Cálculo:

- Tempo: $O(3^{d+1}) = O(4^4) = 256$
- Espaço: $O(3^{d+1}) = O(4^4) = 256$

Depth-First Search (DFS)

Cálculo:

- Tempo: $O(4^7) = 16,384$
- Espaço: $O(4 \cdot 7) = 28$

Algoritmo	Tempo	Espaço	Ótimo	Completo
BFS	$O(4^3) = 64$	$O(4^3) = 64$	Sim	Sim
DFS	$O(4^7) = 16\,384$	$O(4 \cdot 7) = 28$	Não	Não

Tabela 1: Comparação entre algoritmos de busca não informada

Análise Comparativa

Observamos que o algoritmo **BFS** possui tempo e espaço de execução na ordem de $O(4^4) = 256$, garantindo sempre encontrar a solução ótima, mesmo que à custa de maior consumo de memória.

Por outro lado, o DFS pode explorar até $O(4^7) = 16,384$ estados, com consumo de memória bem inferior ($O(4 \cdot 7) = 28$), mas não garante encontrar a melhor solução.

Assim, optamos pelo algoritmo **BFS**, que garante **completude** e **optimalidade**, sendo o mais adequado neste contexto.

BFS:

```
1
2  % Pesquisa em Largura
3
4  pesquisa_largura([no(E,Pai,Op,C,P)|_],no(E,Pai,Op,C,P)):- estado_final(E), inc.
5
6    pesquisa_largura([E|R],Sol):- inc,
7    expande(E,Lseg), %esc(E),
8    insere_fim(Lseg,R,Resto),
9    length(Resto,N), actmax(N),
10   pesquisa_largura(Resto,Sol).
11
12
13   expande(no(E,Pai,Op,C,P),L):-findall(no(En,no(E,Pai,Op,C,P),Opn,Cnn,P1),
14       (op(E,Opn,En,Cn),P1 is P+1, Cnn is Cn+C,incE(1)), L).
```

(d) Depois de resolver os 2 exemplos deste problema com o algoritmo da alínea anterior indique:

i. qual o número total (exacto) de estados visitados.

Exemplo 1 :

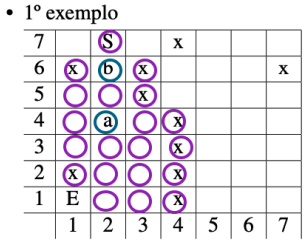

Figura 1: Legenda da imagem

$$Nós = 22 \tag{4}$$

Exemplo 2:

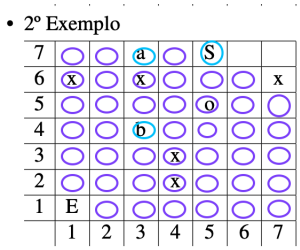

Figura 2: Legenda da imagem

$$Nós = 46 \tag{5}$$

ii. qual o máximo número (exacto) de estados que têm que estar simultaneamente em memória.

Em uma pesquisa em **profundidade em largura** todos os visitados ficam salvos em memória. Assim, o número de nós visitados é igual ao número de nós em memória.

Exemplo 1:

$$Nós = 22 \tag{6}$$

Exemplo 2:

$$Nós = 46 \tag{7}$$

(e) Proponha duas heurísticas admissíveis para estimar o custo de um estado até à solução para este problema.

Heurísticas Admissíveis

Uma heurística admissível é uma função $h(n)$ que nunca sobrestima o custo real $h'(n)$ para atingir a solução a partir de um estado n. Formalmente:

$$h(n) \leq h'(n) \tag{8}$$

Heurística 1: Distâncias de Manhattan

A distância de Manhattan representa o número mínimo de movimentos entre duas posições em um grid, desconsiderando obstáculos. Logo, a heurística nunca sobrestima o custo real e é admissível.

$$\text{dist}_{\text{manhattan}}((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2| \tag{9}$$

Heurística 2: Distâncias Euclidiana:

Apesar de subestimar mais que a Manhattan, a distância Euclidiana nunca sobrestima o custo real. É, portanto, admissível neste problema, embora menos informativa.

$$\text{dist}_{\text{euclidiana}}((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{10}$$

(f) Apresente o código em Prolog do algoritmo de pesquisa informada mais eficiente para resolver os 2 exemplos deste problema usando as heurísticas definidas na alínea anterior. Justifique a escolha do melhor algoritmo e heurística.

O algoritmo escolhido foi o **A***, uma vez que:

- É **completo**, ou seja, encontra uma solução se ela existir;
- É **ótimo**, desde que a heurística usada seja admissível;
- É **eficiente** em termos de número de nós expandidos, comparado a outras abordagens.

```
1
2 %pesquisa_a([],_):- !,fail.
3 pesquisa_a([no(E,Pai,Op,C,HC,P)|_],no(E,Pai,Op,C,HC,P)):- estado_final(E),inc.
4
5 pesquisa_a([E|R],Sol):- inc, asserta(fechado(E)), expande(E,Lseg), esc(E),
6   insere_ord(Lseg,R,Resto),
7   length(Resto,N), actmax(N),
8     pesquisa_a(Resto,Sol).
```

(g) Depois de resolver os 2 exemplos deste problema com o algoritmo da alínea anterior indique para cada função heurística:

i. qual o número total (exacto) de estados visitados.

ii. qual o máximo número (exacto) de estados que têm que estar simultaneamente em memória.

Heurística 1: Distância de Manhattan

i. **Número de estados visitados:** 32

ii. **Máximo de estados simultaneamente em memória:** 12

Esta heurística é mais informativa, o que permite ao A* reduzir significativamente a expansão de estados e a ocupação da memória.

Heurística 2: Distância Euclidiana

i. **Número de estados visitados:** 45

ii. **Máximo de estados simultaneamente em memória:** 17

Por subestimar mais fortemente, esta heurística resulta numa maior expansão de estados e maior ocupação de memória.