

UNIVERSIDADE FEDERAL DE SANTA MARIA

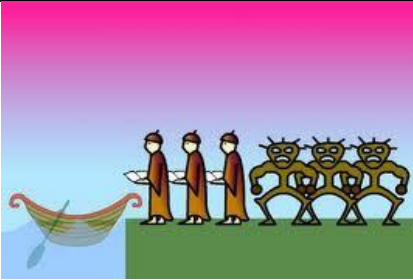
DISCIPLINA DE INTELIGÊNCIA ARTIFICIAL – IA

Lista de Exercícios

(Exerc. 01) Implementar um algoritmo de busca em largura e usar a implementação desenvolvida para resolver o problema dos Missionários e Canibais (descrito na aula). Apresentar:

a) print (em pdf) do passo a passo de execução do algoritmo e da solução do problema (plano para levar todos para o outro lado do rio) para 3 Missionários e 3 Canibais (veja exemplo deste print apresentado em aula) e

b) código fonte da implementação: legível, identado, variáveis nomeadas de forma compreensível, comentado - padrão JavaDoc ou Doxygen, parametrizado para: N missionários, N canibais, N operadores válidos, e orientado a objetos.

	<p>Estado <M, C, PosiçãoDoBarco> M: número de missionários na margem esquerda do rio C: número de canibais na margem esquerda do rio PosiçãoDoBarco: posição do barco (MargemEsq, MargemDir)</p> <p>1 barco pode carregar 2 Missionários nunca devem estar em menor número que canibais</p> <p>Obs.: Somente embarcam e desembarcam do barco os missionários e os canibais que desejarem fazer isso</p>
--	---

(Exerc. 02) Problema dos Jarros consiste no seguinte:

Há dois jarros com capacidades de 3 e 4 litros, respectivamente. Nenhum dos jarros contém qualquer medida ou escala, de modo que só se pode saber o conteúdo exato quando eles estão cheios. Sabendo-se que podemos encher ou esvaziar um jarro, bem como transferir água de um jarro para outro, encontre:

a) uma sequência de passos que deixe o jarro de 4 litros com exatamente 2 litros de água.

Para representar os estados desse problema, podemos usar um par $[X, Y]$, onde $X \in \{0, 1, 2, 3\}$ representa o conteúdo do primeiro jarro e $Y \in \{0, 1, 2, 3, 4\}$ representa o conteúdo do segundo jarro.

As ações podem ser representadas pelos seguintes operadores:

$\text{oper}(\text{enche1}, [X, Y], [3, Y]) \leftarrow X < 3$

$\text{oper}(\text{enche2}, [X, Y], [X, 4]) \leftarrow Y < 4$

$\text{oper}(\text{esvazia1}, [X, Y], [0, Y]) \leftarrow X > 0$

$\text{oper}(\text{esvazia2}, [X, Y], [X, 0]) \leftarrow Y > 0$

$\text{oper}(\text{despeja1em2}, [X, Y], [0, X + Y]) \leftarrow X > 0, Y < 4, X + Y \leq 4$

$\text{oper}(\text{despeja1em2}, [X, Y], [X + Y - 4, 4]) \leftarrow X > 0, Y < 4, X + Y > 4$

$\text{oper}(\text{despeja2em1}, [X, Y], [X + Y, 0]) \leftarrow X < 3, Y > 0, X + Y \leq 3$

$\text{oper}(\text{despeja2em1}, [X, Y], [3, X + Y - 3]) \leftarrow X < 3, Y > 0, X + Y > 3$

O estado inicial é $s_0 = [0, 0]$ e o conjunto de estados meta é $G = \{[X, 2]\}$. Com base nessa especificação, apresentar:

b) o **passo a passo o estado das listas** de novos abertos e nodos fechados usada pelo **algoritmo de busca em largura**

c) **desenho da árvore de busca** criada pelo **algoritmo de busca em largura** ao procurar a solução do problema.

c) Implementar algoritmos para solucionar as questões propostas. Entregar (i) print (em pdf) do passo a passo de execução dos algoritmos e das soluções do problema e (ii) código fonte das implementações: legível, indentado, variáveis nomeadas de forma compreensível, comentado - padrão JavaDoc ou Doxygen, e orientado a objetos.

(Exerc. 03) O Problema do Fazendeiro consiste no seguinte:

Um fazendeiro encontra-se na margem esquerda de um rio, levando consigo um lobo, uma ovelha e um repolho. O fazendeiro precisa atingir a outra margem do rio com toda a sua carga intacta, mas para isso dispõe somente de um pequeno bote com capacidade para levar apenas ele mesmo e mais uma de suas cargas. O fazendeiro poderia cruzar o rio quantas vezes fossem necessárias para transportar seus pertences, mas o problema é que, na ausência do fazendeiro, o lobo pode comer a ovelha e essa, por sua vez, pode comer o repolho. Encontrar:

a) uma sequência de passos que resolva esse problema.

Para representar os estados desse problema, podemos usar uma estrutura da forma $[F, L, O, R]$, cujas variáveis denotam, respectivamente, as posições do fazendeiro, do lobo, da ovelha e do repolho. Cada variável pode assumir os valores e ou d, dependendo da margem do rio onde o objeto se encontra. As ações podem ser representadas pelos seguintes operadores:

$\text{oper}(\text{vai}, [e, L, O, R], [d, L, O, R]) \leftarrow L \neq O; O \neq R$

$\text{oper}(\text{levaLobo}, [e, e, O, R], [d, d, O, R]) \leftarrow O \neq R$

$\text{oper}(\text{levaOvelha}, [e, L, e, R], [d, L, d, R])$

$\text{oper}(\text{levaRepolho}, [e, L, O, e], [d, L, O, d]) \leftarrow L \neq O$

$\text{oper}(\text{volta}, [d, L, O, R], [e, L, O, R]) \leftarrow L \neq O, O \neq R$

$\text{oper}(\text{trazLobo}, [d, d, O, R], [e, e, O, R]) \leftarrow O \neq R$

$\text{oper}(\text{trazOvelha}, [d, L, d, R], [e, L, e, R])$

$\text{oper}(\text{trazRepolho}, [d, L, O, d], [e, L, O, e]) \leftarrow L \neq O$

O estado inicial é $s_0 = [e; e; e; e]$ e o conjunto de estados meta é $G = \{[d, d, d, d]\}$. Com base nessa especificação, apresentar:

b) o **passo a passo o estado das listas** de novos abertos e nodos fechados usada pelo **algoritmo de busca em profundidade**

c) **desenhe a árvore de busca** criada pelo **algoritmo de busca em profundidade** ao procurar a solução do problema.

c) Implementar algoritmos para solucionar as questões propostas. Entregar (i) print (em pdf) do passo a passo de execução dos algoritmos e das soluções do problema e (ii) código fonte das implementações: legível, indentado, variáveis nomeadas de forma compreensível, comentado - padrão JavaDoc ou Doxygen, e orientado a objetos.

(Exerc. 4) Considere os seguintes operadores que descrevem os vôos existentes entre cidades de um país:

oper(1, a, b), oper(2, a, b), oper(3, a, d), oper(4, b, e), oper(5, b, f), oper(6, c, g), oper(7, c, h),

oper(8, c, i), oper(9, d, j), oper(10, e, k), oper(11, e, l), oper(12, g, m), oper(13, j, n), oper(14, j, o),

oper(15, k, f), oper(16, l, h), oper(17, m, d), oper(18, o, a), oper(19, n, b)

Por exemplo, o operador oper(1, a, b) indica que o vôo 1 parte da cidade A e chega na cidade B. Com base nesses operadores, e supondo que eles sejam usados na ordem em que eles foram declarados, apresentar:

a) o **passo a passo o estado das listas** de novos abertos e nodos fechados usados pelo **algoritmo de busca em largura** e **algoritmo de busca em profundidade** que levem da cidade A até a cidade J

b) **desenhe a árvore de busca** criada pelo **algoritmo de busca em largura** e **algoritmo de busca em profundidade** ao procurar uma sequência de vôos que levem da cidade A até a cidade J.

c) Implementar algoritmos para solucionar as questões propostas. Entregar (i) print (em pdf) do passo a passo de execução dos algoritmos e das soluções do problema e (ii) código fonte das implementações: legível, indentado, variáveis nomeadas de forma compreensível, comentado - padrão JavaDoc ou Doxygen, e orientado a objetos.

(Exerc. 5) Considere o mapa de vôos da Figura 1, representado pelos operadores a seguir:

voo(a, b, 1), voo(a, c, 9), voo(a, d, 4), voo(b, c, 7), voo(b, e, 6), voo(b, f, 1), voo(c, f, 7), voo(d, f, 4),

voo(d, g, 5), voo(e, h, 9), voo(f, h, 4), voo(g, h, 1)

Sejam A o conjunto de ações acima:

(a) Apresentar o b) **passo a passo o estado das listas** de novos abertos e nodos fechados e c) **desenhe a árvore de busca** produzida pelo **algoritmo de busca gulosa pela melhor escolha** para $s_0 = a$ e $G = [h]$. Neste exercício, o algoritmo deve usar os valores de custos $g(n)$ (ver Figura 1) apresentados na descrição do problema.

(b) Mostrar que, usando os operadores na ordem declarada acima, **os algoritmos de busca em largura e em profundidade** podem encontrar soluções de custo superior àquela encontrada pelo **algoritmo de busca gulosa pela melhor escolha**, quando $s_0 = a$ e $G = [h]$.

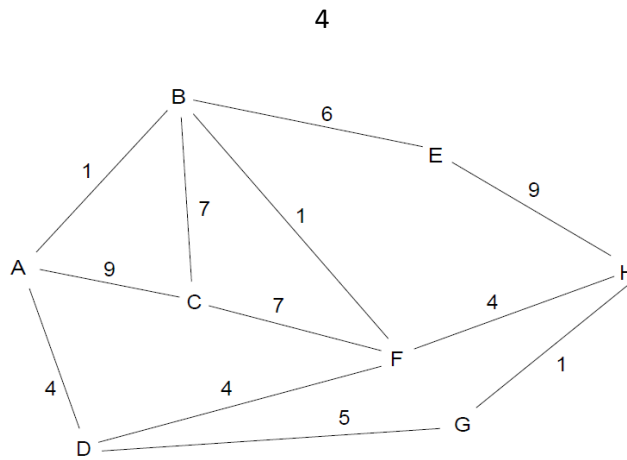


Figura 1 – Mapa de vôos entre cidades, onde as arestas do grafo apresentam os valores de custo $g(n)$ de deslocamento (as vias são bidirecionais)

c) Implementar algoritmos para solucionar as questões propostas. Entregar (i) print (em pdf) do passo a passo de execução dos algoritmos e das soluções do problema e (ii) código fonte das implementações: legível, indentado, variáveis nomeadas de forma compreensível, comentado - padrão JavaDoc ou Doxygen, e orientado a objetos.

(Exerc. 6) Vamos considerar novamente o Problema das Rotas, onde rotas entre cidades são especificadas na Figura 2.

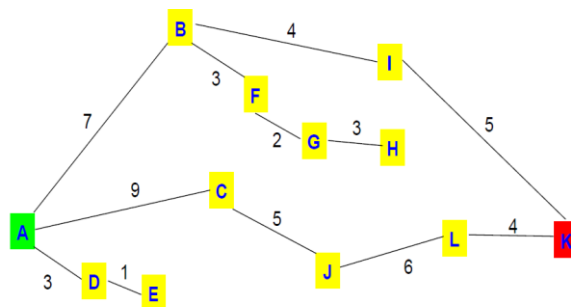


Figura 2 – Mapa de rotas entre cidades, onde as arestas do grafo apresentam os valores de custo $g(n)$ de deslocamento (as vias são bidirecionais)

Como heurística $h(n)$, usaremos a distância em linha reta entre a cidade corrente e a cidade que se deseja atingir. Vamos encontrar uma rota que leve da **cidade A** à **cidade K** e, para facilitar a exposição, vamos definir a função heurística $h(n)$ da seguinte forma:

$$h(a) = 15, h(b) = 7, h(c) = 6, h(d) = 14, h(e) = 15, h(f) = 7, h(g) = 8, h(h) = 5, h(i) = 5, h(j) = 3, h(k) = 0, h(l) = 4$$

Para o Problema das Rotas, apresentar:

a) o **passo a passo o estado das listas** de novos abertos e nodos fechados para o algoritmo **de busca gulosa pela melhor escolha** para $s_0 = a$ e $G = [k]$

b) **desenhe a árvore de busca** produzida quando o **algoritmo de busca gulosa pela melhor escolha** é chamado com $s_0 = a$ e $G = [k]$.

Neste exercício, o algoritmo deve usar os valores heurísticos $h(n)$ apresentados acima.

c) Implementar algoritmos para solucionar as questões propostas. Entregar (i) print (em pdf) do passo a passo de execução dos algoritmos e das soluções do problema e (ii) código fonte das implementações: legível, identado, variáveis nomeadas de forma compreensível, comentado - padrão JavaDoc ou Doxygen, e orientado a objetos.

(Exerc. 7) Para o Problema das Rotas especificado no Exercício 6, apresentar:

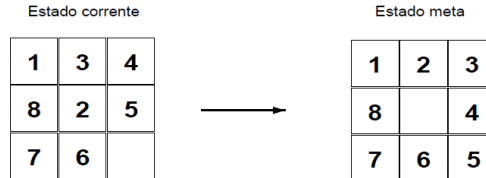
a) o **passo a passo o estado das listas** de novos abertos e nodos fechados usados pelo **algoritmo A*** para $s_0 = a$ e $G = [k]$

b) **desenhe a árvore de busca** produzida quando o **algoritmo A*** é chamado com $s_0 = a$ e $G = [k]$.

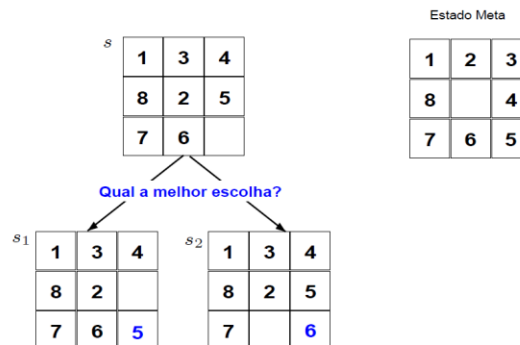
Neste exercício, o algoritmo deve usar os valores de custo $g(n)$ e heurísticos $h(n)$ apresentados anteriormente.

c) Implementar algoritmos para solucionar as questões propostas. Entregar (i) print (em pdf) do passo a passo de execução dos algoritmos e das soluções do problema e (ii) código fonte das implementações: legível, identado, variáveis nomeadas de forma compreensível, comentado - padrão JavaDoc ou Doxygen, e orientado a objetos.

(Exerc. 8) O problema do Quebra-Cabeça de 8 consiste em movimentar as peças do quebra-cabeça horizontal ou verticalmente (para ocupar a posição vazia adjacente à peça) de modo que a congruação final seja alcançada:



Por exemplo, expandindo o estado corrente acima, temos:



Agora, usando uma função heurística, o algoritmo de busca deveria expandir o melhor entre esses dois estados sucessores. Mas como decidir qual deles é o melhor? Uma possibilidade é verificar o quão longe cada peça encontra-se de sua posição na congruação final e apontar como melhor estado aquele cuja soma das distâncias é mínima. Por exemplo, no estado s_1 , as peças 1, 5, 6, 7 e 8 já estão em suas posições finais. Para as peças 2, 3 e 4, a distância é 1. Portanto, $h(s_1) = 3$. Analogamente, temos $h(s_2) = 5$. Esses valores indicam que uma solução a partir do estado s_1 pode

ser obtida com no mínimo mais três expansões, enquanto que uma solução a partir de s_2 requer no mínimo mais cinco expansões. Então, o algoritmo de busca deve expandir o estado s_1 .

a) Para esse problema, qual algoritmo seria mais apropriado: (i) o **algoritmo de busca gulosa pela melhor escolha** considerando que cada ação tem custo 1 ou (ii) o **algoritmo de busca gulosa pela melhor escolha** considerando as estimativas heurísticas calculadas?

Apresentar o b) **passo a passo o estado das listas** de novos abertos e nodos fechados e c) **desenhe a árvore de busca** produzida pelos algoritmos citados em (i) e ii) para justificar a resposta apresentada.

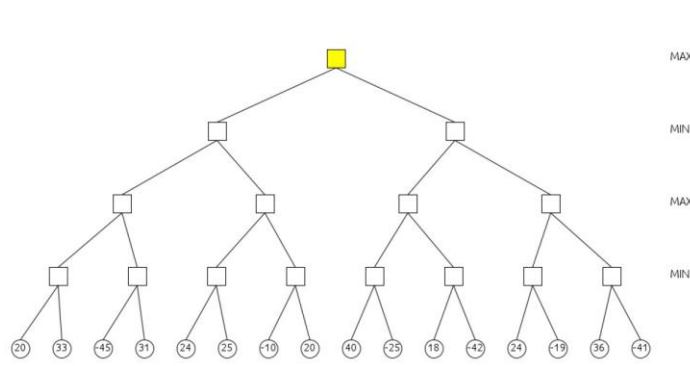
Considere que no Quebra-Cabeça de 8 cada ação tem custo 1. Usando a heurística da soma das distâncias, apresentar:

d) o **passo a passo o estado das listas** de novos abertos e nodos fechados usados pelo **algoritmo A***

e) **desenhe a árvore de busca** produzida pelo **algoritmo A*** quando o estado inicial do quebra-cabeça é $[[1, 2, 3], [b, 6, 4], [8, 7, 5]]$.

f) **implementação dos algoritmos** considerados neste exercício, onde deve ser entregue (i) print (em pdf) do passo a passo de execução dos algoritmos e das soluções do problema e (ii) código fonte das implementações: legível, indentado, variáveis nomeadas de forma compreensível, comentado - padrão JavaDoc ou Doxygen, e orientado a objetos.

(Exerc. 9) Considere a seguinte árvore de busca.



a) Apresentar os valores de min e max propagados na árvore de busca

b) Adotando a poda alfa-beta, nos sentidos i) da esquerda para a direita e ii) da direita para a esquerda, indicar quais arestas/subárvores serão podadas.

c) Implementar algoritmos para solucionar as questões propostas. Entregar (i) print (em pdf) do passo a passo de execução dos algoritmos e das soluções do problema e (ii) código fonte das implementações: legível, indentado, variáveis nomeadas de forma compreensível, comentado - padrão JavaDoc ou Doxygen, e orientado a objetos.

(Exerc. 10) Consider o seguinte problema.

Dados 5 palitos cada jogador pode retirar 1, 2 ou 3 por turno. Perde o jogador que retira o último palito. A pergunta é: será que max pode ganhar o jogo?



Usando a seguinte função de utilidade: $F(S) = +1$ se MAX ganhar, -1 se MIN ganhar, desenhar a árvore de busca.

- Apresentar os valores de min e max propagados na árvore de busca construída
- Adotando a poda alfa-beta, nos sentidos i) da esquerda para a direita e ii) da direita para a esquerda, indicar quais arestas/subárvores serão podadas.
- Implementar algoritmos para solucionar as questões propostas. Entregar (i) print (em pdf) do passo a passo de execução dos algoritmos e das soluções do problema e (ii) código fonte das implementações: legível, indentado, variáveis nomeadas de forma compreensível, comentado - padrão JavaDoc ou Doxygen, e orientado a objetos.

Prof. Luis Alvaro