**UNIVERSIDADE FEDERAL DE SANTA MARIA**

**DISCIPLINA DE INTELIGÊNCIA ARTIFICIAL – IA**

**Primeira Prova, Entrega: 19/06/2023, até 23.59, por email para o professor**

(Exerc. 01) 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 trasportar 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:

oper(vai, [e, L, O, R], [d, L, O, R]) ← L ≠ O; O ≠ R

oper(levaLobo, [e, e, O, R], [d, d, O, R]) ← O ≠ R

oper(levaOvelha, [e, L, e, R], [d, L, d, R])

oper(levaRepolho, [e, L, O, e], [d, L, O, d]) ← L ≠ O

oper(volta, [d, L, O, R], [e, L, O, R]) ← L ≠ O, O ≠ R

oper(trazLobo, [d, d, O, R], [e, e, O, R]) ← O ≠ R

oper(trazOvelha, [d, L, d, R], [e, L, e, R])

oper(trazRepolho, [d, L, O, d], [e, L, O, e]) ← L ≠ O

O estado inicial é s0 = [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** e pelo **algoritmo de busca em largura**

c) **desenhe a árvore de busca** criada pelo **algoritmo de busca em profundidade** e 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, identado, variáveis nomeadas de forma compreensível, comentado - padrão JavaDoc ou Doxigen, e orientado a objetos.

(Exerc. 2) 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, identado, variáveis nomeadas de forma compreensível, comentado - padrão JavaDoc ou Doxigen, e orientado a objetos.

(Exerc. 3) 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 s0 = 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 àquele encontrada pelo **algoritmo de busca gulosa pela melhor escolha**, quando s0 = a e G = [h].

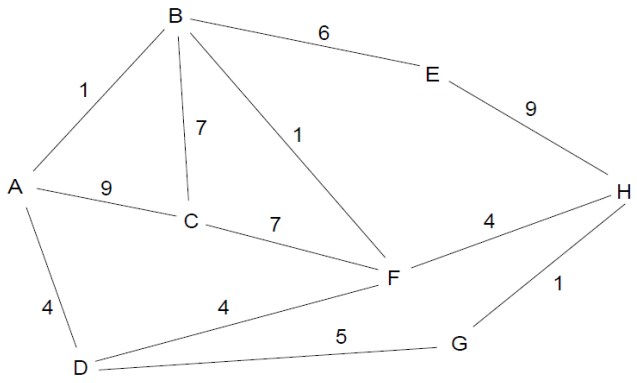
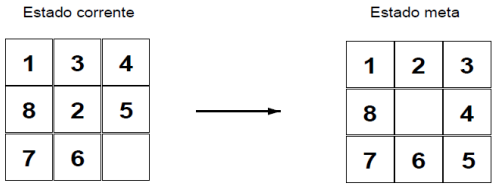


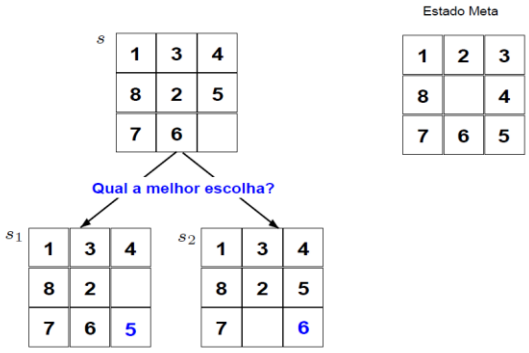

Figura 1 – Mapa de vôos entre cidades, onde as arestas do grafo apresentam os valores de custo g(n) de deslocamento (as vias são bidirecionais)

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, identado, variáveis nomeadas de forma compreensível, comentado - padrão JavaDoc ou Doxigen, e orientado a objetos.

(Exerc. 4) 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 conguração 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 conguração final e apontar como melhor estado aquele cuja soma das distâncias é mínima. Por exemplo, no estado s1, 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(s1) = 3. Analogamente, temos h(s2) = 5. Esses valores indicam que uma solução a partir do estado s1 pode ser obtida com no mínimo mais três expansões, enquanto que uma solução a partir de s2 requer no mínimo mais cinco expansões. Então, o algoritmo de busca deve expandir o estado s1.

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, identado, variáveis nomeadas de forma compreensível, comentado - padrão JavaDoc ou Doxigen, e orientado a objetos.

(Exerc. 5) Consider o seguinte problema.

Dados 10 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.

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

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, identado, variáveis nomeadas de forma compreensível, comentado - padrão JavaDoc ou Doxigen, e orientado a objetos.

**Obs.:**

a) **As implementações de cada questão são obrigatórias. Somente questões com implementações associadas serão consideradas nas avaliações.**
b) **Cópias de implementações, em qualquer questão, serão sumariamente avaliadas com ZERO.**
c) **Entregar código fonte das implementações, prints da execução dos algoritmos e vídeos de no máximo 5min para explicar cada uma das implementações construídas (um vídeo por implementação). Somente questões onde estes itens tenham sido entregues serão consideradas nas avaliações.**
d) **Qualquer entrega após o prazo determinado não será considerada nesta prova: o deadline é dia 19/06!**

Prof. Luis Alvaro