Inteligência Artificial - Lista de Exercícios Teóricos - 01

Roberto Sérgio Ribeiro de Meneses - 520403

30/11/2024

1. Diferença entre busca informada e busca não informada

- **Busca informada**
  - Executam buscas sem aproveitar o conhecimento prévio sobre o problema em questão.
- **Busca não informada**
  - Faz uso de heurísticas (dados prévios utilizados no momento da implementação que aprimoram a solução).

2. Problema dos missionários e canibais utilizando busca em largura

```
// Realiza o (BFS) para encontrar o caminho do estado inicial até o estado destino.
Função BFS(estadoInicial):
    estadoDestino <- (0, 0, 0)
    filaDeEstados <- [(estadoInicial, caminhoPercorrido=[])]
    estadosVisitados <- Conjunto vazio
    Enquanto filaDeEstados NÃO estiver vazia:
        (estadoAtual, caminhoPercorrido) <- filaDeEstados.desenfileirar()
        Se estadoAtual em estadosVisitados:
            Continuar para o próximo estado
        Adicionar estadoAtual em estadosVisitados
        novoCaminho <- caminhoPercorrido + [estadoAtual]
        Se estadoAtual == estadoDestino:
            Retornar novoCaminho
        Para cada proximoEstado em gerarEstadosVizinhos(estadoAtual):
            Enfileirar (proximoEstado, novoCaminho)
    Retornar NULO

// Gera os possíveis estados vizinhos para um estado atual.
Função gerarEstadosVizinhos(estado):
    (missRestante, caniRestantes, posicaoBarco) <- estado
    movimentosPossiveis <- [(1, 0), (0, 1), (1, 1), (2, 0), (0, 2)]
    estadosGerados <- Lista vazia
    Para cada (deltaM, deltaC) em movimentosPossiveis:
        Se posicaoBarco == 1:
            novoEstado <- (missRestante - deltaM, caniRestantes - deltaC, 0)
        Caso contrário:
            novoEstado <- (missRestante + deltaM, caniRestantes + deltaC, 1)
        Se estadoSeguro(novoEstado):
            Adicionar novoEstado em estadosGerados
```

```
    Retornar estadosGerados

// Verifica se um estado é seguro.
Função estadoSeguro(estado):
    (missRestante, caniRestantes, posicaoBarco) <- estado
    Retornar missRestante >= 0 e caniRestantes >= 0 e
           missRestante <= 3 e caniRestantes <= 3 e
           (missRestante == 0 OU missRestante >= caniRestantes) e
           ((3 - missRestante) == 0 OU (3 - missRestante) >= (3 - caniRestantes))

// main
estadoInicial <- (3, 3, 1)
solucao <- BFS(estadoInicial)
Se solucao NÃO for NULO:
    Imprimir "Solução encontrada:"
    Para cada passo em solucao:
        Imprimir passo
Caso contrário:
    Imprimir "Não existe solução"
```

3. Determinação de nós visitados em busca em largura e profundidade

- **Busca em largura:**
  - Expande a fronteira para os nós B e C, depois para D, E, F e G; então expande novamente para H, e por fim para I. Portanto, todos os nós são visitados.
- **Busca em profundidade:**
  - Desce pela esquerda em B ao expandir a fronteira, olhando B e D; então pela direita de B, olhando E e H; por fim, expande o lado direito de E e encontra I.
- **Conclusão:** O DFS foi mais eficiente neste caso, realizando 6 checagens, pois encontrou o nó objetivo rapidamente explorando da esquerda para a direita. O BFS fez 9 checagens devido à sua abordagem exaustiva, mas garente o caminho mais curto.

4. Heurística composta no quebra-cabeça das 8 peças

Para resolver o quebra-cabeça das 8 peças, foram utilizadas duas heurísticas como parte do algoritmo A*:

- **Heurística 1 (h1):** Distância de Manhattan entre o local da peça e o local correto.
- **Heurística 2 (h2):** Número de peças na posição incorreta.

A heurística composta é definida como:

$$h_{\text{composta}}(n) = h_1(n) + h_2(n)$$

Cada heurística é admissível separadamente:

$$h_1(n) \leq h^*(n) \quad \text{e} \quad h_2(n) \leq h^*(n)$$

Entretanto, a soma pode resultar em:

$$h_{\text{composta}}(n) > h^*(n)$$

pois h_1 e h_2 podem contar parcialmente o mesmo custo, levando a uma superestimação.