**Lista 08**
**Giuseppe Sena Cordeiro - 801779**

**1)**

1. Algoritmo de Busca em Largura
   Ordem de Visitação = {A, B, C, D, E, F, G, H, I}
   Caminho Solução = {A, B, E, I}
   Não possui heurística
2. Algoritmo de Busca em Profundidade
   Ordem de Visitação = {A, B, D, E, H, I}
   Caminho Solução = {A, B, E, I}
   Não possui heurística
3. Custo Uniforme
   Ordem de Visitação = {A, C, B, E, F, G, D, K}
   Caminho Solução = {A, C, G, K}
   Não possui heurística
4. Algoritmo de Busca Gulosa
   Ordem de Visitação = {A, B, E, I}
   Caminho Solução = {A, B, E, I}
   A heurística é admissível
5. Algoritmo A*
   Ordem de Visitação = {A, B, E, C, G, K}
   Caminho Solução = {A, C, G, K}
   A heurística é admissível

**2)**

**1)** A heurística de Manhattan é admissível porque nunca superestima o custo real para alcançar o estado final. Ela calcula a soma das distâncias verticais e horizontais de cada peça até sua posição correta, considerando apenas movimentos válidos no tabuleiro. Como ela sempre fornece um valor igual ou inferior ao custo mínimo necessário para resolver o puzzle, garante a optimalidade da solução.

**2)** Uma outra heurística possível é contar o número de peças fora do lugar. Ela também é admissível, pois cada peça mal posicionada exige ao menos um movimento para chegar à posição correta, o que garante que o valor da heurística nunca ultrapassa o custo real. No entanto, essa heurística é menos precisa do que a de Manhattan, já que não considera a distância que cada peça precisa percorrer.

**3)** b) I e III.

**4)** A) ABCDEF

**5)** E) apebas as afurnatuvas I, IV, V sao corretas

**6)** A) a busca gulosa minimiza h(n).

**7)** A) $h(n) \leq h°(n)$

**8)** C) a b e I

**9)** Se a função objetivo do algoritmo de busca é:
$f(n) = (2 – w)\cdot g(n) + w\cdot h(n)$

então o tipo de busca que ele realiza vai depender do valor de w, já que esse parâmetro define o peso dado ao custo real do caminho (g(n)) e à heurística (h(n)). Analisando os casos:
Quando w = 0:
A função fica f(n) = 2·g(n), ou seja, depende apenas do custo do caminho real percorrido até o nó.
→ Isso equivale a uma Busca de Custo Uniforme (Uniform Cost Search).
Quando w = 1:
A função vira f(n) = g(n) + h(n).
→ Esse é exatamente o comportamento da Busca A*, que equilibra custo real e estimativa heurística de forma ótima.
Quando w = 2:
A função se torna f(n) = 2·h(n), ou seja, considera apenas a heurística (multiplicada por 2, mas ainda sem influência de g(n)).
→ Isso corresponde à Busca Gulosa (Greedy Best-First Search), que tenta chegar mais rápido ao objetivo baseado na estimativa heurística.
Então resumindo:
w = 0 → Busca de Custo Uniforme
w = 1 → Busca A*
w = 2 → Busca Gulosa

**10)**
1)
a) Nós expandidos por A* usando cada heurística:
- Usando h1: os nós visitados são S, B, C e G.
- Usando h2: os nós visitados são S, B, D e G.
- Usando h3 (que é o h0): os nós visitados são S, A, C e G.

b) Caminho encontrado por cada uma:
- Com h1: S → B → C → G (custo total: 6)
- Com h2: S → B → D → G (custo total: 8)
- Com h3: S → A → C → G (custo total: 9)

c) Quais heurísticas são admissíveis?
- h1 é admissível, porque nunca estimou um custo maior que o real até o objetivo.
- h2 **não** é admissível, pois no nó D ela diz que faltam 3 até o G, mas o custo real é 5.
- h3 (h0) é admissível, porque sempre retorna zero — e zero nunca é uma superestimativa.

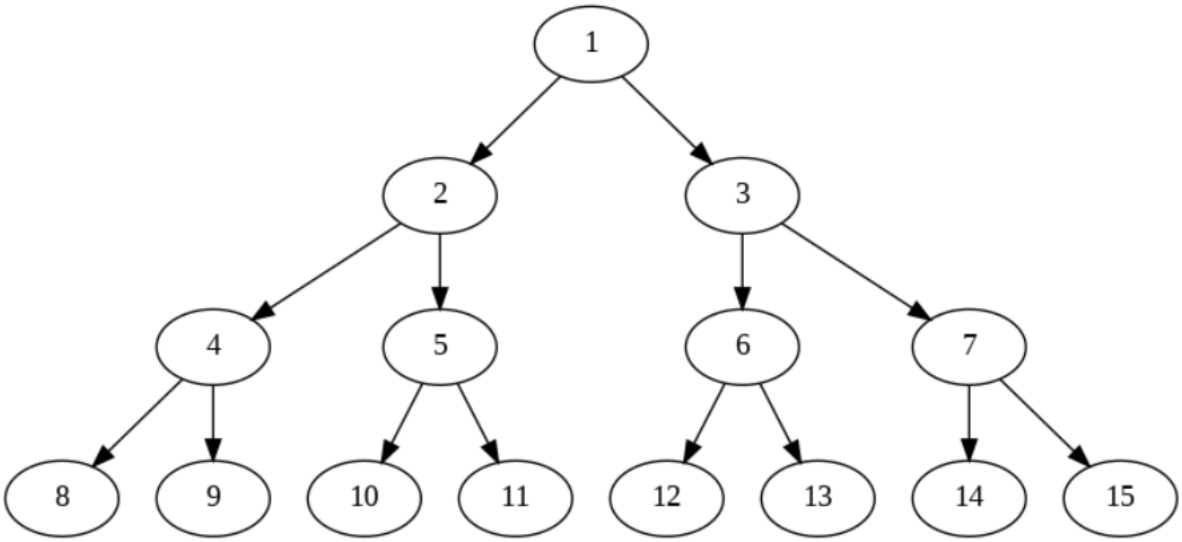
2)
a) Nós expandidos:
- Usando heurística gulosa com h1: S, B, C, G (escolhe sempre o nó com menor h).
b) Caminho encontrado:
- S → B → C → G (custo total: 6)

3)
c) Nós expandidos:
- S, A, C, G (assumindo que a ordem dos filhos é A, B — ou seja, profundidade pela esquerda)
d) Caminho encontrado:
- S → A → C → G (custo total: 9)

4)
e) Nós expandidos:
- S, A, B, C, G (explora todos os vizinhos antes de ir para o próximo nível)
f) Caminho encontrado:
- S → B → C → G (caminho mais curto em número de passos e custo: 6)

**11)** E)  as duas asserçoes sao proposiçoes

**12)** a-




B-
Busca por largura:
V = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
Busca em profundidade limitada:
V= { 1, 2, 4, 8, 9, 5, 10, 11}
Busca em profundidadeinterativa:
1: 1,
2: 1, 2,
3: 1, 2, 4, 5, 3, 6, 7,
4: 1, 2, 4, 8, 9, 5, 10, 11,
V = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

**13)**

1. Vantagens:
   - Ótimo (se a heurística for admissível).
   - Completo (se custo mínimo > 0).
   - Pode ser mais eficiente que BFS ou DFS.
   - Com uma heurística bem desenhada, encontra soluções rápidas e boas.

2. Desvantagens:
   - Uso intensivo de memória: A* guarda todos os nós gerados.
   - Pode ser lento se a heurística for ruim (piora para BFS).
   - Difícil criar uma boa heurística em muitos problemas.

**14)**

- IDA* (Iterative Deepening A*)
    - Combina A* com busca em profundidade iterativa.
    - Usa menos memória.
    - Ideal para grandes espaços de estados.
- RBFS (Recursive Best-First Search)
    - Usa recursão e backtracking, mantendo apenas os caminhos necessários na memória.
    - Menor uso de memória que A*.
- Memory-Bounded  A* (como MA*, SMA*)
    - Limitam o uso de memória explicitamente.
    - Trocam nós menos promissores quando falta espaço.
- WA* (Weighted A*)
    - Usa $f(n) = g(n) + w * h(n)$ com $w > 1$
    - Mais rápido, mas não garante solução ótima se $w \neq 1$.