Lista 4 - Inteligência Artificial

Nome: **Victor Ferraz de Moraes**
Matrícula: **802371**

Questão 01)

1) Busca em Largura:
    - Nós visitados: **A, B, C, D, E, F, G, H, I**
    - Solução Obtida: A solução será a soma dos pesos das arestas visitadas, que no caso são {5 + 2 + 9 + 4 + 9 + 10 + 7 + 7} = 53
    - A heurística para este caso não faz diferença, pois o algoritmo não a utiliza.

2) Busca em Profundidade:
    - Nós visitados: **A, B, D, E, H, I**
    - Solução Obtida: A solução será a soma dos pesos das arestas visitadas, que caso são {5 + 9 + 4 + 7 + 7} = 32
    - A heurística para este caso não faz diferença, pois o algoritmo não a utiliza.

3) Custo Uniforme:
    - Nós visitados: (De acordo com a fila de prioridade)
      [{A,0}]
      [{C,2}, {B,5}]
      [{B,5}, {F,11}, {G,12}]
      [{E,9}, {F,11}, {G,12}, {D,14}]
      [{F,11}, {G,12}, {D,14}, {H,16}, {I,16}]
      [{G,12}, {D,14}, {H,16}, {I,16}]
      [{D,14}, {K,15}, {H,16}, {I,16}, {J,16}]
      [{K,15}, {H,16}, {I,16}, {J,16}]
      [{H,16}, {I,16}, {J,16}] // FIM DO ALGORITMO

      A, C, B, E, F, G, D, K
    - Solução Obtida: A solução é o valor armazenado no vértice de destino quando ele sai da fila de prioridade, ou seja, 15.
    - A heurística para este caso não faz diferença, pois o algoritmo não a utiliza.

4) Algoritmo de Busca Gulosa:
    - Nós visitados: **A, B, E, I**
    - Solução Obtida: {5 + 4 + 7} = 16
    - A heurística é admissível?
      h(A) <= h*(A) {10 <= 16 ou 15} **Sim**
      h(B) <= h*(B) {4 <= 11} **Sim**
      h(C) <= h*(C) {8<=13} **Sim**
      h(D) <= h*(D) {10 <=INF} **Sim**

h(E) <= h*(E) {1 <= 7} **Sim**
h(F) <= h*(F) {13 <= INF} **Sim**
h(H) <= h*(H) {3 <= INF} **Sim**
h(I) <= h*(I) {0 <= 0} **Sim**
h(J) <= h*(J) {8 <= INF} **Sim**
h(K) <= h*(K) {0 <= 0} **Sim**

Portanto, a heurística é admissível

5) Algoritmo A*
   - Nós visitados: (De acordo com a fórmula f(n) = g(n) + h(n), onde g(n) é o custo acumulado a partir do vértice de origem e h(n) é uma estimativa do custo restante do vértice até o destino final)

     f(A) = 0 + 10 -> [{A, 10}]
     //Retira A da Fila
     //Adiciona seus filhos
     f(B) = 5 + 4 -> [{B,9}]
     f(C) = 2 + 8 -> [{B,9}, {C,10}]
     //Retira B da fila, adiciona seus filhos
     f(D) = 14 + 10 -> [{C,10}, {D,24}]
     f(E) = 9 + 1 -> [{C,10}, {E,10}, {D,24}]
     //Retira C da fila, adiciona seus filhos
     f(F) = 11 + 13 -> [{E,10}, {D,24}, {F, 24}]
     f(G) = 12 + 2 -> [{E,10},{G,14}, {D,24}, {F, 24}]
     //Retira E da fila, adiciona seus filhos
     f(H) = 16 + 3 -> [{G,14}, {H,19}, {D,24}, {F, 24}]
     f(I) = 16 + 0 -> [{G,14}, {I,16}, {H,19}, {D,24}, {F, 24}]
     //Retira G da fila, adiciona seus filhos
     f(J) = 16 + 8 -> [{I,16}, {H,19}, {D,24}, {F, 24}, {J,24}]
     f(K) = 15 + 0 -> [{K,15} ,{I,16}, {H,19}, {D,24}, {F, 24}, {J,24}]
     //Retira K da fila
     //FIM DO ALGORITMO

     Visitados: **A, B, C, E, G, K**

   - Solução obtida: A solução é o valor armazenado no vértice de destino quando ele sai da fila de prioridade, ou seja, 15.
   - A heurística é admissível?
     h(A) <= h*(A) {10 <= 16 ou 15} **Sim**
     h(B) <= h*(B) {4 <= 11} **Sim**
     h(C) <= h*(C) {8<=13} **Sim**
     h(D) <= h*(D) {10 <=INF} **Sim**
     h(E) <= h*(E) {1 <= 7} **Sim**

h(F) <= h*(F) {13 <= INF} **Sim**
h(H) <= h*(H) {3 <= INF} **Sim**
h(I) <= h*(I) {0 <= 0} **Sim**
h(J) <= h*(J) {8 <= INF} **Sim**
h(K) <= h*(K) {0 <= 0} **Sim**

Portanto, a heurística é admissível

Questão 02)

1. A heurística de Manhattan é admissível porque, ao não superestimar o custo real para atingir o objetivo, mantém essa propriedade. Ela soma as distâncias horizontais e verticais de cada peça à sua posição final no Puzzle de 8, sem considerar obstáculos, e garante que o custo estimado nunca seja menor que o caminho mais curto necessário, assegurando sua admissibilidade.
2. A alternativa é a heurística do número de peças mal posicionadas, chamada Misplaced Tiles. Ela conta quantas peças estão fora de suas posições corretas. Se, por exemplo, 3 peças estiverem no lugar errado, a heurística terá valor 3. Como essa heurística nunca superestima o custo real e o número de movimentos será sempre igual ou maior que o número de peças mal posicionadas, ela é admissível.

Questão 03)

Letra **B**

Questão 04)

Letra **A**

Questão 05)

Letra **E**

Questão 06)

Letra **A**

Questão 07)

Letra **B**

Questão 08)

Letra **B**

Questão 09)

Quando w é 0 ele realiza uma busca focada no custo acumulado, similar a uma busca de custo uniforme, porém com o custo multiplicado por 2.
Quando w é 1 ele realiza a busca do tipo A*, equilibrando custo e heurística.
Quando w é 2 ele realiza uma busca puramente gulosa, mas com a heurística duplicada.

f(n) = (2 - w).g(n) + w.h(n)
f(n) = (2 - 0).g(n) + 0.h(n) = 2.g(n)
f(n) = (2 - 1).g(n) + 1.h(n) = g(n) + h(n)
f(n) = (2 - 2).g(n) + 2.h(n) = 2.h(n)

Questão 10)

1) Busca A*
   (**h1):**
   a) (Fila de Prioridade):
      f(S) = 0 + 0 = 0 -> [{S,0}]
      //Retira S da fila e adiciona seus filhos
      f(A) = 5 + 0 = 5 -> [{A,5}]
      f(B) = 2 + 0 = 2 -> [{B,2}, {A,5}]
      //Retira B da fila e adiciona seus filhos
      f(C) = 4 + 0 = 4 -> [{C,4}, {A,5}]
      f(D) = 3 + 0 = 3 -> [{D,3}, {C,4}, {A,5}]
      //Retira D da fila e adiciona seus filhos
      f(G) = 8 + 0 = 8 -> [{C,4}, {A,5}, {G,8}]
      //Retira C da fila e adiciona seus filhos
      f(G) = 6 + 0 = 6 -> [{A,5}, {G,6}, {G,8}]
      //Retira A da fila e adiciona seus filhos
      f(G) = 10 + 0 = 10 -> [{G,6}, {G,8}, {G,10}]
      //Retira G da fila
      //FIM DO ALGORITMO

      Nós visitados: S, B, D, C, A , G

   b) Caminho encontrado: **S -> B -> C -> G**

   c) A heurística é admissível?
      h(S) <= h*(S) {0 <= 6} **Sim**
      h(A) <= h*(A) {0 <= 5} **Sim**
      h(B) <= h*(B) {0<=4} **Sim**
      h(C) <= h*(C) {0 <=2} **Sim**
      h(D) <= h*(D) {0 <= 5} **Sim**
      h(G) <= h*(G) {0 <= 0} **Sim**

Portanto, a heurística é admissível

(h2):

a) (Fila de Prioridade)
   f(S) = 0 + 5 = 5 -> [{S,5}]
   //Retira S da fila e adiciona seus filhos
   f(A) = 5 + 3 = 5 -> [{A,8}]
   f(B) = 2 + 4 = 2 -> [{B,6}, {A,8}]
   //Retira B da fila e adiciona seus filhos
   f(C) = 4 + 2 = 4 -> [{A,5},{C,6}]
   f(D) = 3 + 5 = 8 -> [{A,5},{C,6}, {D,8}]
   //Retira A da fila e adiciona seus filhos
   f(G) = 10 + 0 = 10 -> [{C,6}, {D,8}, {G,10}]
   //Retira C da fila e adiciona seus filhos
   f(G) = 6 + 0 = 6 -> [{G,6},{D,8},{G,10}]
   //Retira G da fila
   //FIM DO ALGORITMO

   Nós visitados: S, B, A, C, G

b) Caminho encontrado: **S -> B -> C -> G**

c) A heurística é admissível?
   h(S) <= h*(S) {5 <= 6} **Sim**
   h(A) <= h*(A) {3 <= 5} **Sim**
   h(B) <= h*(B) {4<=4} **Sim**
   h(C) <= h*(C) {2 <=2} **Sim**
   h(D) <= h*(D) {5 <= 5} **Sim**
   h(G) <= h*(G) {0 <= 0} **Sim**
   Portanto, a heurística é admissível

(h3):

a) (Fila de Prioridade)
   f(S) = 0 + 6 = 6 -> [{S,6}]
   //Retira S da fila e adiciona seus filhos
   f(A) = 5 + 5 = 10 -> [{A,10}]
   f(B) = 2 + 2 = 4 -> [{B,4}, {A,10}]
   //Retira B da fila e adiciona seus filhos
   f(C) = 4 + 5 = 9 -> [{C,9}, {A,10}]
   f(D) = 3 + 3 = 6 -> [{D,6},{C,9}, {A,10}]
   //Retira D da fila e adiciona seus filhos
   f(G) = 8 + 0 = 8 -> [{G,8},{C,9}, {A,10}]
   //Retira G da fila
   //FIM DO ALGORITMO

Nós visitados: S, B, D, G

b) Caminho encontrado: **S -> B -> D -> G**

c) A heurística é admissível?
h(S) <= h*(S) {6 <= 6} **Sim**
h(A) <= h*(A) {5 <= 5} **Sim**
h(B) <= h*(B) {2<=4} **Sim**
h(C) <= h*(C) {5 <=2} **Não**
h(D) <= h*(D) {3 <= 5} **Sim**
h(G) <= h*(G) {0 <= 0} **Sim**

A heurística de todos os vértices não deve superestimar o custo real para chegar até o objetivo. Como esta condição não ocorre para todos os vértices do grafo, a heurística não é admissível.

2) Busca Gulosa

a) Os nós expandidos pela busca gulosa vai variar dependendo da Heurística utilizada. Por exemplo, se usarmos a h1, todos os nós terão heurística 0 e, portanto, não haverá preferência de escolha entre vértices e o algoritmo se comportará como uma busca em largura ou profundidade.
H2: Nós expandidos -> S, A, G
H3: Nós expandidos -> S, B, D, G

b) **H1:** Depende da escolha do algoritmo. Por exemplo, caso ele expanda A primeiro, ele encontrará o caminho **S -> A -> G.**
H2: S -> A -> G
H3: S -> B -> D -> G

3) Busca em Profundidade

Supondo que a ordem de preferência seja em ordem alfabética:

a) Nós expandidos: S, A, G, B, C, D
Caso o algoritmo termine quando alcança o objetivo -> S, A, G

b) Caminho encontrado: **S -> A -> G**

4) Busca em Largura

Supondo que a ordem de preferência seja em ordem alfabética:

a) Nós expandidos: S, A, B, G, C, D
Caso o algoritmo termine quando alcança o objetivo -> S, A, G

b) Caminho encontrado: **S -> A -> G**

Questão 11)

Calculando o somatório das distâncias dos três estados: E1, E2, E3.
E1: d(1) = 2 + 1 = 3
d(2) = 2 + 1 = 3
d(3) = 2 + 0 = 2
d(4) = 1 + 1 = 2
d(5) = 1 + 2 = 3
d(6) = 2 + 1 = 3
d(7) = 0 + 2 = 2
d(8) = 0 + 2 = 2

h(E1) = 3 + 3 + 2 + 2 +3 + 3 + 2 + 2 = **20**

E2: d(1) = 2 + 1 = 3
d(2) = 2 + 1 = 3
d(3) = 0 + 2 = 2
d(4) = 1 + 1 = 2
d(5) = 1 + 2 = 3
d(6) = 2 + 1 = 3
d(7) = 1 + 2 = 3
d(8) = 1 + 0 = 1

h(E2) = 3 + 3 + 2 + 2 + 3 + 3 + 3 + 1 = **20**

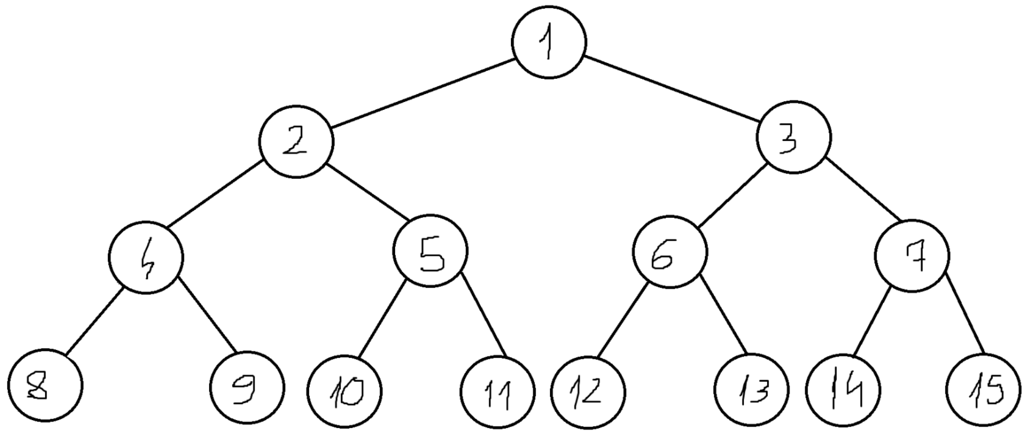
E3: d(1) = 2 + 1 = 3
d(2) = 2 + 1 = 3
d(3) = 2 + 0 = 2
d(4) = 1 + 1 = 2
d(5) = 1 + 2 = 3
d(6) = 1 + 1 = 2
d(7) = 0 + 2 = 2
d(8) = 0 + 1 = 1

h(E3) = 3 + 3 + 2 + 2 + 3 + 2 + 2 + 1 = **18**

Letra A

Questão 12)

a.



b. Busca em Extensão (Busca em Largura): 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
   Busca em Profundidade limitada com limite 3: 1, 2, 4, 8, 9, 5, 10, 11
   Busca por Aprofundamento Iterativo:
   Primeira Iteração: 1
   Segunda Iteração: 1, 2, 3
   Terceira Iteração: 1, 2, 4, 5, 3, 6, 7
   Quarta Iteração: 1, 2, 4, 8, 9, 5, 10, 11

Questão 13)

Vantagens:

Caminho Ótimo Garantido: se a heurística utilizada for admissível, ou seja, nunca superestimar o custo restante, o A* sempre encontra o caminho mais curto (ótimo) do ponto inicial até o destino.
Eficiência: combinando a heurística com o custo real, o A* pode ser mais eficiente que o algoritmo de Dijkstra em termos de exploração de nós, especialmente em grandes grafos, pois concentra a busca nas áreas mais promissoras.
Flexibilidade: o A* pode ser aplicado a problemas como mapeamento, inteligência artificial em jogos, robótica, planejamento de rotas, desde que haja uma função de custo e uma heurística adequada.
Controlabilidade: a escolha da função heurística ajusta o comportamento do algoritmo, tornando-o mais rápido ou mais preciso dependendo da aplicação.
Completo: se houver um caminho, o A* o encontrará, desde que tenha memória e tempo suficientes.

Desvantagens:

Consumo de Memória: o A* armazena todos os nós visitados, resultando em alto consumo de memória, especialmente em grafos grandes.
Desempenho em Ambiente Grande: embora eficaz em problemas menores, em ambientes

grandes ou complexos, o desempenho do A* pode se degradar.
Dependência da Heurística: o sucesso do A* depende da qualidade da heurística, e uma heurística ruim pode aumentar o tempo de execução.
Complexidade Computacional: embora eficiente em encontrar o caminho ótimo, o A* pode ser mais lento que algoritmos mais simples quando o caminho mais curto não é necessário.
Difícil de ajustar: encontrar uma heurística que garanta eficiência e tempo de execução adequado pode ser desafiador, tornando o desenvolvimento de uma heurística admissível e eficiente uma tarefa complicada.

Questão 14)

Dentre as melhorias no algoritmo A*, algumas se destacam por suas próprias características e são mais adequadas para certos usos. Por exemplo, o IDA* (Iterative Deepening A*) é uma das mais importantes, pois ajuda a reduzir o maior problema do A*, seu alto consumo de memória. O algoritmo determina um limite de profundidade iterativo e explora os nós de maneira muito mais eficiente em memória do que o A*, sendo uma excelente escolha para espaços de busca muito grandes, embora possa ser um pouco mais lento, pois reexplora os nós.
Outra melhoria relevante é o SMA* (Simplified Memory-Bounded A*), que controla diretamente o uso de memória. Ele descarta os nós menos promissores quando a memória se esgota para assegurar que o algoritmo não exceda a capacidade de armazenamento, embora possa ser mais lento quando houver memória extremamente limitada.
Para ambientes dinâmicos, uma das melhores escolhas é o D* (Dynamic A*), que permite que o caminho seja ajustado se o grafo mudar durante a execução, o que é extremamente útil para coisas como navegação robótica e videogames, onde os obstáculos podem se mover.
Uma menção notável também é o A* Bidirecional. Esse algoritmo procura a partir do início e do ponto de destino e pode diminuir drasticamente o tempo de execução em grafos grandes.

Questão 15)

MAX não pode ganhar o jogo devido a capacidade de MIN manipular a quantidade de palitos que sobrará na próxima vez que MAX jogar. Dessa forma, MAX sempre irá retirar o último palito. Isso, claro, é considerando que MIN sempre irá minimizar suas escolhas e jogar de forma ótima.

Exemplo:

Palitos Restantes: 5 (vez de MAX)
MAX pode retirar 1, 2 ou 3 palitos. Se MAX retirar:

1 palito (restam 4 palitos),
2 palitos (restam 3 palitos),
3 palitos (restam 2 palitos).

Palitos Restantes: 4 (vez de MIN)

MIN pode retirar 1, 2 ou 3 palitos. Se MIN retirar:

1 palito (restam 3 palitos),
2 palitos (restam 2 palitos),
3 palitos (resta 1 palito).

MIN pode forçar MAX a pegar o último palito com a sequência certa de movimentos, ou seja, caso escolha retirar 3 palitos, ele ganha o jogo.

Questão 16)

Letra **B**