1 Questões Teóricas

1. **Quais algoritmos de busca em espaço de estados você utilizaria para encontrar o caminho de menor custo entre um estado inicial e um estado final (ou meta)? Compare os algoritmos selecionados em termos de custo computacional (tempo de execução e espaço de memória) e apresente vantagens e desvantagens na utilização de cada um dos métodos.**
   Eu usaria o A*, o guloso ou de menor distância primeiro, o que esses algoritmos tem em comum é que eles possuem o caminho de menor custo como objetivo e eles lidam com heurística em sua resolução.

   Os algoritmos guloso e de menor distância podem entrar em loop ao processa ciclos, então esses algoritmos necessitariam implementar também uma poda de ciclos, gerando assim um custo computacional maior. Já o A* por considerar o custo dos caminho intermediários em sua função de estimativa eventualmente ele sairia o loop.

2. **O algoritmo Iterative Deepening (Aprofundamento Iterativo) aplica uma busca em profundidade impondo um limite na profundidade máxima a ser pesquisada. Este limite é incrementado de um em um até que um estado alvo seja encontrado ou até que a árvore toda seja pesquisada. Explique como esta estratégia elimina desvantagens e combina vantagens de ambos, Busca em largura e Busca em Profundidade.**
   Esse métodos une os conceitos de busca do DFS e do BFS. Consiste em aplica com DFS com uma profundidade definida que vai sendo incrementada ao decorrer das iterações, dessa forma ao final de uma iteração seria a arvore de busca seria percorrida até uma certa altura.
   y

   Para problemas com solução não tão "profundas", soluções que de forma relativa não possuem uma grande quantidade de arestas, o algoritmo pode ser bem eficiente uma vez que a profundidade da pesquisa é aumentada iterativamente e também possuem um menor gasto de memória de comparado com o BFS.

3. **Selecione a opção correta para cada célula da tabela.** $h(n)$ **é o valor da função heurística do nó** n**.** $c(S, n)$ **é o custo do caminho de um nó** S **até o nó** n**.**

Estratégia	Seleção da fronteira	Caminho Encontrado	Custo em Espaço
Busca em largura			
Busca em Profundidade			
Guloso			
Menor Caminho Primeiro			
A*			
Branch and Bound			

4. **Para o que serve função heurística em alguns algoritmos e busca?** Para alguns algoritmos é uma função que recebe um nó como entrada e estima o custo do menor caminho entre esse nó em questão e o nó objetivo. Função de heurística: $h(n)$.
   Essa função é muito utilizada como um componente que auxilia na tomada de decisões de alguns algoritmos heurísticos, considerado essa função como critério um dos critérios para estabelecer a viabilidade de caminhos.

5. **Como funciona o algoritmo de poda de ciclos?**
   Esse algoritmo é usado para evitar que algoritmos de busca processem ciclos de um grafo infinitamente. Para isso a poda de ciclos desconsidera caminhos em que há repetição de nós.

   A poda de ciclos garante que dado um caminho $p = \{n_0, n_1, ..., n_k\}$ temo que $\forall n_i, n_j \in p \wedge i \neq j \rightarrow n_i \neq n_j$.

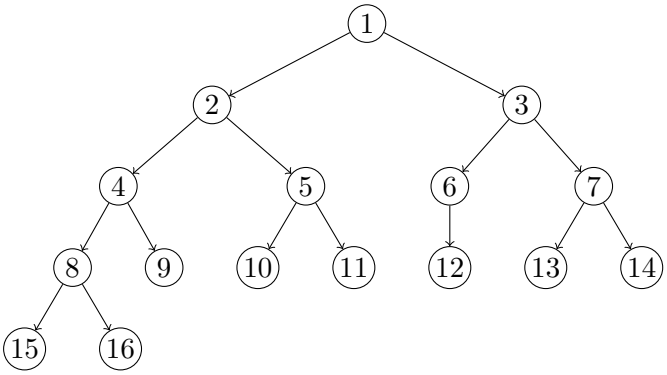
   Sendo assim o algoritmo de poda de ciclos verifica se essa condição é atendida para cada caminho da fronteira e desconsidera os caminhos que violam essa condição.

6. **Como funciona o algoritmo de poda de múltiplos caminhos?**
   É usado para poupar a memória e o processamento durante a execução de um algoritmo de busca evitando caminhos redundantes té um certo nó ao desconsidera caminhos que levam a um nó já percorrido pela fronteira.

   Durante sua busca ele desconsidera caminhos que até um nó que já foi percorrido anteriormente por algum caminho da fronteira, evitando assim armazenar múltiplos caminho para um mesmo nó.

   Para algoritmos que buscam o caminho de menor custo é necessária uma adaptação para esse algoritmo, ao encontrar um caminho até um nó que já foi alcançado por um caminho na fronteira verifica o custo de ambos e armazena na fronteira somente o de menor custo.

7. **Considere o grafo abaixo onde o 1 representa o estado inicial e o nó 11 é o objetivo. O custo de cada aresta é a diferença entre os valores do nó filho de do nó pai.**



(a)
F = ¡1¿
F = ¡1,2¿, ¡1,3¿
F = ¡1,3¿, ¡1,2,4¿, ¡1,2,5¿
F = ¡1,2,4¿, ¡1,2,5¿, ¡1,3,6¿, ¡1,3,7¿
F = ¡1,2,5¿, ¡1,3,6¿, ¡1,3,7¿, ¡1,2,4,8¿, ¡1,2,4,9¿
F = ¡1,3,6¿, ¡1,3,7¿, ¡1,2,4,8¿, ¡1,2,4,9¿, ¡1,2,5,10¿, ¡1,2,5,11¿ –¿ ¡1,2,5,11¿
(b)
F = ¡1¿
F = ¡1,2¿, ¡1,3¿
F = ¡1,2¿, ¡1,3,6¿, ¡1,3,7¿
F = ¡1,2¿, ¡1,3,6¿, ¡1,3,7,13¿, ¡1,3,7,14¿
F = ¡1,2¿, ¡1,3,6,12¿
F = ¡1,2,4¿, ¡1,2,5¿
F = ¡1,2,4¿, ¡1,2,5,10¿, ¡1,2,5,11¿ –¿ ¡1,2,5,11¿
(c)
F = ¡1¿(0)
F = ¡1,2¿(1), ¡1,3¿(2)
F = ¡1,3¿(2), ¡1,2,4¿(3), ¡1,2,5¿(4)
F = ¡1,2,4¿(3+4, 3+5), ¡1,2,5¿(4), ¡1,3,6¿(5), ¡1,3,7¿(6)
F = ¡1,2,5¿(4), ¡1,3,6¿(5), ¡1,3,7¿(6), ¡1,2,4,8¿(7), ¡1,2,4,9¿(8)
F = ¡1,2,5¿(4+5, 4+6), ¡1,3,6¿(5), ¡1,3,7¿(6), ¡1,2,4,8¿(7), ¡1,2,4,9¿(8)
F = ¡1,3,6¿(5), ¡1,3,7¿(6), ¡1,2,4,8¿(7), ¡1,2,4,9¿(8), ¡1,2,5,10¿(9), ¡1,2,5,11¿(10)
...
F = ¡1,2,5,11¿(10), ... –¿ ¡1,2,5,11¿

2 Atividades Práticas

GitHub Link: `https://github.com/ViniciusSamy/IA/tree/master/Listas/2/Codigo`