INSTITUTO FEDERAL MINAS GERAIS (IFMG) - CAMPUS BAMBUÍ
Inteligência Artificial
Prof. Marcos Roberto Ribeiro

Lista de Exercícios 03

Exercício 1:

Considere as estruturas de dados e algoritmos de buscas estudados.

(a) Implemente uma classe **Problema** para representar problemas de buscas. A classe deve ter as seguintes características:

i) Construtor com estado **inicial** e **meta** (opcional). Além do estado inicial e meta, a classe deve ter um atributo que informa se o custo usa heurística;

ii) Métodos abstratos **acoes(estado)** e **resultado(estado, proximo_estado)**;

iii) Método **ativa_heuristica()** para ativar o uso da heurística no custo;

iv) Método **heuristica(estado, proximo_estado)** para retornar uma heurística de **estado** para **proximo_estado**. Nessa classe esse método pode retornar sempre 0 (zero);)

v) Método **custo(custo_atual, estado, acao, proximo_estado)** para retornar o custo de chegar ao **proximo_estado**, com aplicação de **acao** em **estado**, partindo do **custo_atual**

- Inicialmente, o custo de cada passo pode ser considerando como 1 (um);
- Se o uso de heurística estiver ativado, o custo deve somar também o valor da heurística.

vi) Método **teste_meta(estado)** para testar se o **estado** é a meta.

(b) Implemente a classe **ProblemaTeste** herdando de **Problema** para realização de comparações dos algoritmos de busca. Considere as seguintes características:

i) O construtor deve receber o **problema** real armazená-lo como atributo da classe. O construtor deve inicializar atributos para guardar o número de estados **visitados** e **gerados** e também a solução do problema;

ii) Reimplemente os métodos **acoes()**, **custo()** e **ativa_heuristica()** para chamar os métodos correspondentes do problema real;

iii) Sobrescreva o método **resultado(estado, acao)** para chamar o mesmo método do problema real e contabilizar os estados gerados;

iv) Reescreva o método **teste_meta(estado)** para testar a meta com o problema real, contabilizar os estados visitados e pegar a solução (se for encontrada);

v) Crie o seguinte método para acesso aos métodos do problema real:

```
1 def __getattr__(self, attr):
2     return getattr(self._problema, attr)
```

(c) Implemente uma classe **Nodo** para representar os nós da árvore de busca considerando as seguintes características:

i) Construtor com **estado** e, como opcional, **pai**, **acao** e **custo**. Além desses atributos, a classe deve guardar a profundidade do nó, inicializada com a profundidade do **pai** mais 1 (um);

ii) Implementar o método especial **__lt__(nodo)** para fazer comparações de *menor* usando o custo dos nós (necessário para ordernar os nós pelo custo);

iii) Implementar o método especial **__eq__(nodo)** para testar se o nó atual é igual a **nodo** (utilize o **estado** dos nós);

iv) Método **cria_filho(problema, acao)** para criar um nó filho com a aplicação de **acao** no nó atual (observe os parâmetros necessários para a criação de um nó);

v) Método **expande(problema)** para retornar a lista de filhos do nó atual com base no **problema**;

vi) Métodos **caminho()** e **solucao()** para retornar o caminho do nó atual até a raiz da árvore de busca e a sequência de ações para chegar à solução (utilize o atributo com a informação do **pai**)

(d) Implemente os seguintes algoritmos utilizando estruturas para armazenar a **borda** e os nós já **explorados**:

i) Busca em profundidade;
ii) Busca em largura;
iii) Busca de aprofundamento iterativo;
iv) Busca de custo uniforme;
v) Busca A* (a busca de custo uniforme ser reutilizada, depois de ativar a heurística do problema.

(e) Faça uma função que receba uma lista de problemas e utilize a classe **ProblemaTeste** para testar os algoritmos de busca desenvolvidos em cada problema recebido.

Exercício 2:

Represente o problema do aspirador de pó através de uma classe herdando da classe **Problema**. Considere os seguintes pontos:

- Existe mais de um estado meta? Como tratar essa questão?
- Implemente uma heurística para usar com o algoritmo A*. Seria correto deixar o cômodo atual sujo e ir para outro cômodo?
- Execute os algoritmos de buscas desenvolvidos partindo de um estado em que ambos cômodos estejam sujos.

Exercício 3:

Implemente a classe **QuebraCabeca** herdando da classe **Problema** para implementar o problema do quebra cabeça deslizante de 8 peças. Considere os seguintes pontos:

- Implemente uma heurística para usar com o algoritmo A*;
- Execute os algoritmos de buscas desenvolvidos partindo de um estado aleatório.

Exercício 4:

Considere a resolução de problemas de busca em grafos.

(a) Implemente uma classe **Grafo** para representar grafos como mapas. Considere os seguintes pontos:
- O grafo deve permitir arestas com pesos e coordenadas para os vértices;
- É interessante usar dicionários para representar as arestas e as coordenadas;
- Pode ser interessante que a classe tenha métodos para retornar os vértices, coordenadas de vértices, vizinhos de vértices e peso de arestas (distância entre vértices).

(b) Implemente a classe **ProblemaGrafo** herdando de **Problema** para lidar com problemas de buscas em grafos. Considere os seguintes pontos:
- O construtor deve receber, além do estado inicial e meta, o grafo que representa o problema;
- As ações aplicáveis a um estado são os vértices vizinhos desse estado;
- O resultado de uma ação é a própria ação (que é um vizinho);
- escreva o método **grafo_heuristica(nodo)** para retornar a distância em linha reta de **nodo** até a meta;
- Implemente o método **custo_grafo(custo_atual, nodo_a, nodo_b)** para calcular o custo (a partir do **custo_atual**) para chegar a **nodo_b** a partir de **nodo_a** (considere a possibilidade da heurística estar ativada);
- Sobrescreva o método **custo()** simplesmente chamando o método **custo_grafo()**;

(c) Escreva uma função para calcular a distância em linha reta entre dois vértices.

(d) Implemente o problema do mapa da Romênia e execute os algoritmos de buscas desenvolvidos.