Exercício prático - Agente de resolução de problemas - Parte 02

Alunos: Antonio Carlos Ramos filho, Fabiano Amaral Alves, Pablo Junio Souza Santos

1. Linguagem e IDE utilizadas

Nesta segunda parte do exercício continuamos a utilizar a linguagem python juntamente com o pycharm pelos mesmo motivos ditos na parte 1 deste exercício

2. Espaço de estados

Para fazer essa parte do exercício, baseamos nas definições e algoritmos disponibilizados nas *videoaulas* e no livro *Inteligência Artificial, 3°ed, Russel e Norvig*, então explicaremos o Espaço de estados apresentando conceitos e exemplo de implementação.

Estados: O estado é determinado pela "posição" do agente (ponto inicial) e o objetivo (ponto final). O agente pode está entre *n* posições e cada uma pode ter *n* caminhos possíveis, desse modo, temos `n*n^n` estados do mundo possíveis.

> Para um agente que quer encontrar o menor caminho em entre cidade, implementamos o espaço de estados através de um grafo, onde cada vértice é um estado e seus adjacentes são os próximos estados possíveis.

```
self.grafo = Arquivo.extrair_dados(nome_arquivo)
```

- Os pequenos trechos de código abaixo dos conceitos são apenas para exemplificar algoritmicamente cada conceito, e serão apresentados em sua classe completa mais adiante.

Estado inicial: Qualquer estado pode ser escolhido para se o estado inicial, sendo definido a partir de um vértice do grafo.

> Continuando com o agente que quer encontrar o melhor caminho, seu estado inicial será definido a partir de uma cidade (vértice) qualquer existente no grafo.

```
self.cidade_inicial = self.grafo[cidade_inicial]
```

Ações: Acontecem conforme a decisão definida no espaço de estados, podendo variar entre (Tomando um agente aspirador de pó como exemplo) Direita, Esquerda, Aspira, Não faz nada.

> No nosso agente, a ação a ser tomada é ir para uma cidade vizinha caso ela não seja a cidade pretendida na *formulação do problema* ou parar caso o objetivo foi alcançado.

```
# Retorna uma lista par (Ação, Resultado da ação) a partir do estado atual
def estados_sucessores(self, estado):
    return [(sucessor, sucessor) for sucessor in
self.grafo.get(estado).keys()]
```

Teste de objetivo: Verifica se o estado atual do agente é o estado ao qual ele quer chegar.

> No nosso caso, partindo de salinas em direção a montes claros, se a cidade atual for montes claros ele cumpriu o seu papel.

```
def verifica_objetivo(self, estado_atual):
    if estado_atual == self.objetivo:
        return True
    return False
```

Custo do caminho: A soma dos pesos de todos os saltos (ações) até chegar ao objetivo. Varia de acordo ao ponto inicial e o objetivo.

> O custo do caminho do nosso agente será a soma das distancias das cidades visitadas partindo da inicial até a de destino

```
def custo_caminho(self, custo_parcial, A, B):
    return custo_parcial + (self.grafo.get(A, B) or infinity)
```

Implementação simples de um espaço de estados

```
# Arquivo: estado.py

from dados import Arquivo

class Estado(object):

    def __init__(self, cidade_inicial, objetivo, nome_arquivo):
        self.grafo = Arquivo.extrair_dados(nome_arquivo)
        self.cidade_inicial = self.grafo[cidade_inicial]
        self.objetivo = self.grafo[objetivo]

    def estados_sucessores(self, estado):
        return [(sucessor, sucessor) for sucessor in
self.grafo.get(estado).keys()]

    def verifica_objetivo(self, estado_atual):
        if estado_atual == self.objetivo:
            return True
        return False
```

```
    '''
        recebe o custo até o estado anterior, a cidade atual e o resultante
da ação
        retorna a soma entre o custo do estado anterior e o custo do
resultado da ação corrente
    '''
    def custo_caminho(self, custo_parcial, A, B):
        return custo_parcial + (self.grafo.get(A, B) or infinity)
```

3. Extração das informações do arquivo e preenchimento do Grafo (Espaço de estados)

Para extrair os dados do arquivo de texto, criamos a `class Arquivo:` com um único método `def extrair_dados(nome_arquivo):` que recebe o arquivo extrai os dados e devolve a representação de um grafo em uma lista de adjacências.

O grafo retornando poderá ser atribuído a uma classe que representará o espaço de estados, ex: `self.grafo = Arquivo.extrair_dados(nome_arquivo)`.

Escolhemos preencher o 'grafo de estados' usando uma estrutura como *dicionário* por que é uma maneira mais simples ao nosso ver, e assim pode ser usado em algoritmos de busca diretamente.

Na estrutura utilizada o 'grafo de estados' ficaria da seguinte forma:

```
{
    'Vertice': [{'Vertice1': poso1}, {'VerticeN': pesoN}]
}
```

Onde `'Vertice'` é tomado como o estado atual, e os `{'VerticeN': pesoN}` são os estados acessíveis a partir dele sendo `pesoN` o custo para chegar do estado `'Vertice'` a `'VerticeN'`

A seguir veremos a implementação da classe Arquivo.

```
#arquivo: dados.py

import os
from collections import defaultdict

class Arquivo:

    def extrair_dados(nome_arquivo):
        #coloca os dados do arquivo em um dicionário
        #{chave: [valores]}
        grafo = defaultdict(list)

        #abre o arquivo com fechamento automático após o
        #fim do bloco with
```

```
        with open(os.path.abspath(nome_arquivo), 'r') as file:
            file.readline() # lê a primeira linha e descarta

            # Também é descartado pois não é necessário em dicionário
            quant_v = file.readline().strip().split('VERTICES: ')
            quant_a = file.readline().strip().split('ARESTAS: ')

            # transforma as linhas restantes em uma lista
            linhas_rest = file.readlines()
            for linha in linhas_rest:
                #divide a linha em um 'array' com três posições
                line = linha.strip().split(', ')

                cidade1 = line[0]
                cidade2 = line[1]
                custo= line[2]

                #cria uma espécie de lista de adjacências no grafo
                grafo[cidade1].append({cidade2: int(custo)})
                grafo[cidade2].append({cidade1: int(custo)})
        # retorna a estrutura do grafo a ser usada na classe estado
        return grafo
```