CI209 - Inteligência Artificial

Prof. Aurora Pozo
DInf - UFPR
2020/2

Especificação do primeiro trabalho prático da disciplina [1]

Trabalho prático 1

Este trabalho é composto por cinco exercícios que foram retirados e adaptados do curso de inteligência artificial da Universidade da Califórnia, Berkeley. Você deverá implementar os seguintes algoritmos de busca vistos em aula: Busca em Largura, Busca em Profundidade, Busca Uniforme e A*. Cada algoritmo será testado por diferentes casos de teste e poderão ser testados utilizando o seguinte comando:

```
$ python2 autograder.py
```

Este comando permitirá observar se as implementações feitas estão corretas. A seguinte mensagem deverá aparecer se todos os testes forem bem sucedidos:

```
Provisional grades
==================
Question q1: 3/3
Question q2: 3/3
Question q3: 3/3
Question q4: 3/3
------------------
Total: 12/12
```

Busca

O projeto consiste na implementação de um agente, na linguagem de programação *python2*, que encontra um caminho entre o jogador e um objetivo final dentro de um labirinto. Você deverá baixar o código base e alterar o arquivo *busca.py*. Neste arquivo, quatro funções referentes aos diferentes algoritmos de busca mencionados anteriormente devem ser implementados.

As funções recebem o argumento *problem* como parâmetro. Este parâmetro é uma instância de uma classe que possui os seguintes métodos:

[1] Elaborado por Bruno Henrique Meyer e Augusto Lopez Dantas

- *problem.getStartState()*: Retorna um elemento único (tupla/string/inteiro) que identifica o estado inicial do problema
- *problem.isGoalState(**id**)*: Retorna *True* se o id de um estado seja o estado objetivo (final) ou *False* caso contrário
- *problem.getSuccessors(**id**)*: Retorna uma lista de tuplas. Cada tupla representa a ramificação a partir das ações possíveis de um estado. Cada tupla possui três elementos no seguinte formato: (**id do estado**, **nome da ação**, **custo da ação**).

Ao final de cada função implementada deve-se retornar uma lista contendo as ações (na ordem correta) necessárias para encontrar o estado objetivo a partir do estado inicial. Os valores dos elementos que representam as ações são os mesmos retornados pelo método *problem.getSuccessors*

A função *aStarSearch* aceita, além do parâmetro *problem*, um parâmetro chamado *heuristic*. Esse parâmetro representa uma função que recebe o identificador de um estado como argumento e retorna uma heurística que estima a distância entre o estado atual e o estado objetivo.

Divisão do trabalho

Parte 1 *Busca em Profundidade*

Implemente a função *depthFirstSearch* no arquivo *busca.py*. A ordem em que os nós são escolhidos, quando não especificado pelo algoritmo, é a ordem em que o método *problem.getSuccessors* retorna os estados sucessores.

Você pode visualizar o desempenho da sua implementação utilizando o agente para controlar o jogador no simulador. Para isso, utilize um dos seguintes comandos:

```
$ python2 pacman.py -l tinyMaze -p SearchAgent -a fn=dfs
$ python2 pacman.py -l mediumMaze -p SearchAgent -a fn=dfs
$ python2 pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=dfs
```

Na simulação, você poderá visualizar a ordem e quantidade de nós expandidos (possíveis movimentações do jogador), além de verificar a solução final retornada pelo seu agente (caminho executado pelo jogador).

Parte 2 *Busca em Largura*

As instruções necessárias para realizar esta parte do trabalho são semelhantes às mencionadas na Parte 1. Você deverá alterar a função *breadthFirstSearch* no arquivo *busca.py* e implementar o algoritmo de Busca em Largura. Para visualizar o resultado do agente na simulação execute os seguintes comandos:

```
$ python2 pacman.py -l tinyMaze -p SearchAgent -a fn=bfs
$ python2 pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
$ python2 pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=bfs
```

Parte 3 *Busca Uniforme*

As instruções necessárias para realizar esta parte do trabalho são semelhantes às mencionadas na Parte 1. Você deverá alterar a função *uniformCostSearch* no arquivo *busca.py* e implementar o algoritmo de Busca Uniforme. Note que nesse caso será necessário utilizar os valores relacionados ao custo das ações obtidos por meio do método *problem.getSuccessors*. Para visualizar o resultado do agente na simulação execute os seguintes comandos:

```
$ python2 pacman.py -l tinyMaze -p SearchAgent -a fn=ucs
$ python2 pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
$ python2 pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=ucs
```

Parte 4 *A**

As instruções necessárias para realizar esta parte do trabalho são semelhantes às mencionadas na Parte 1. Você deverá alterar a função *aStarSearch* no arquivo *busca.py* eggjimplementar o algoritmo A*. Note que a função recebe um parâmetro adicional que representa uma função que estima a distância entre o estado atual e o estado objetivo (heurística). Para visualizar o resultado do agente na simulação execute os seguintes comandos:

```
$ python2 pacman.py -l tinyMaze -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
$ python2 pacman.py -l mediumMaze -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
$ python2 pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
```

Parte 5 *Conclusões e Relatório*

Execute os comandos mencionados nas partes anteriores do trabalho para simular o jogo com diferentes agentes de busca.

Crie um relatório de até 3 páginas explicando as suas conclusões e observações em relação às diferenças observadas entre os algoritmos de busca. Seu relatório deverá conter:

- Seu nome completo
- Explicações sobre vantagens e desvantagens de cada algoritmo
- Quais aspectos do problema influenciam (e como influenciam) os comportamentos dos algoritmos de busca

Se preferir, utilize figuras, tabelas entre outros recursos. Recomenda-se o uso do LaTeX para construir o seu relatório.

Dicas

- Há diversas semelhanças entre os algoritmos que serão implementados. Se concentre em fazer uma boa implementação do primeiro algoritmo, o que facilitará a implementação dos restantes
- Você poderá implementar um algoritmo de busca genérica como base que será usado para implementar os outros algoritmos que variam as estruturas de fila/pilha para gerenciar a ordem em que os estados são explorados
- O uso de estruturas de grafos para inserir e gerenciar os estados podem auxiliar na tarefa de expandir os estados somente quando necessário
- Tome cuidado com a ordem em que as implementações de Busca em Largura e Busca em Profundidade expandem os estados. Os casos de testes só serão executados com sucesso se esses algoritmos acessarem os nós na ordem retornada pelo método *problem.getSuccessors*
- Tome cuidado com a quantidade de vezes que um estado é expandido. Cada estado deveria ser expandido no máximo uma única vez
- Utilize as estruturas de fila, pilha e fila de prioridade já implementadas. Há uma breve descrição de como usa-las no início do arquivo *busca.py*. Se preferir, você pode implementar as suas estruturas

Entrega

Você deverá entregar um arquivo compactado com o nome *login.tar.gz*, onde *login* é o nome do seu usuário no sistema do Departamento de Informática, que contenha os seguintes arquivos:

- *busca.py*
- *login.pdf*

Trabalhos atrasados serão aceitos com um atraso máximo de 2 semanas com desconto de nota. Após o prazo máximo o trabalho não será mais aceito.

Observações

Você deverá desenvolver e compreender todas implementações feitas no seu trabalho. Não serão admitidos quaisquer tipos de plágio.

Dúvidas

Dúvidas poderão ser retiradas por e-mail:

bhmeyer@inf.ufpr.br
aldantas@inf.ufpr.br