Resolução do quebra-cabeças de 8 peças com BFS, DFS, Subida da Encosta e A*

Mariana Bastos dos Santos

Agosto 2019

1 Introdução

Muitos dos algoritmos de busca mais conhecidos foram desenvolvidos no século passado, numa época onde os dados e as informações caminhavam de forma embrionária para se tornarem o que hoje chamamos de big data. Esses algoritmos se tornaram de suma importância para aplicação à problemas reais como na robótica, por exemplo. Um dos problemas mais utilizados para testar e desenvolver algoritmos de busca, é o problema do quebra-cabeça com peças deslizantes, no qual o objetivo é sempre encontrar a sequência dos números. Este trabalho propõe o desenvolvimento dos algoritmos de busca A*, Subida de Encosta, Busca em Largura e Busca em Profundidade para a resolução do problema do quebra-cabeça 8 peças. A linguagem utilizada será python em que serão desenvolvidas as classes *Puzzle* a aplicação da as ações do jogo, a classe *Grafo* para a criação e expansão dos vértices e seus filhos e, por fim, a criação de cada uma das classes para os algoritmos de busca: *BFS*, *DFS*, *SubidaEncosta* e *Aestrela*.

2 Conceitos Fundamentais

2.1 Busca em largura

[1] A busca em largura (BFS - Breadth-first search) inicia expandindo o nó raiz, na sequência expande um a um de seus nós sucessores e em seguida, expande os sucessores dos últimos nós que foram expandidos. Para isso, é

utilizada uma fila na qual são enfileirados os nós a serem visitados, sendo assim, os nós mais novos, sempre mais profundos do que seus antecessores, são inseridos no final da fila e os nós mais antigos, sempre mais rasos em relação aos novos, são desenfileirados e então visitados.

2.2 Busca em profundidade

[1] A busca em profundidade (DFS – Depth-first search) expande o nó mais profundo do nível efetivo do grafo. Sendo assim, a busca prossegue até o nível mais profundo da árvore de busca, onde os nós não têm sucessores. Quando o nó não tem sucessor, retorna ao nó seguinte mais profundo que ainda tem sucessores inexplorados. Para isso, pode ser utilizada uma pilha, na qual são inseridos os vértices do nível visitado e são desempilhados aqueles vértices que serão visitados.

2.3 Subida de Encosta

[1] A busca de subida de encosta expande o nó que possui o maior valor, ou seja, conforme vai expandido os nós, o algoritmo segue de forma contínua no sentido do valor crescente, numa espécie de subida de encosta. O algoritmo é finalizado quando alcança o valor máximo no qual nenhum de seus sucessores terá um valor mais alto.

2.4 Busca A*

[1] A busca A* expande seus nós a partir de uma combinação de *g(n)*, o custo para alcançar o nó, e *h(n)*, o custo para ir do nó ao objetivo, selecionando o caminho no qual essa combinação $f(n) = g(n) + h(n)$ seja a de menor custo, para uma solução de menor custo, ou selecionando o caminho no qual a combinação seja de maior custo, para uma solução de maior custo.

3 Implementação

Neste trabalho foram implementados os algoritmos: busca em largura, busca em profundidade, subida de encosta e A* para a solução do problema do quebra-cabeças de 8 peças. Para a implementação foi utilizada a linguagem python.

As classes implementadas foram: *Puzzle*, *Grafo*, *BFS*, *DFS*, *SubidaEncosta* e *Aestrela*.

A classe *Puzzle* é responsável por gerar os movimentos do quebra-cabeça através da função *movimento*, ou seja, ela é responsável por gerar as ações do quebra-cabeça. As ações são: abaixo, acima, direita e esquerda. Essas ações ocorrem dependendo da posição que se encontra o espaço vazio do quebra-cabeça. Por exemplo, quando o espaço vazio se encontra no canto inferior direito, ela só pode fazer dois movimentos: acima e esquerda. O espaço que se encontra no canto superior esquerdo só pode realizar dois movimentos: direita e abaixo. E assim por diante. Cada posição têm ações especificas que podem ser aplicadas.

A classe *Grafo* é responsável por expandir os nós sucessores ao nó atual e ainda, é responsável por determinar o custo de cada uma das ações do quebra-cabeça. Para expandir o grafo, é chamado, na função *expandir*, o método *movimento* da classe *Puzzle*, pois este determina quais serão os sucessores do nó atual. Obtendo os nós sucessores, a função então insere na lista de adjacência os sucessores do vértice e guarda seus custos e movimentos que geraram os sucessores. Para o cálculo do custo de cada vértice é utilizada a distância de manhattan a qual calcula a distância do vértice em relação ao objetivo.

A classe *BFS* é responsável por gerar a busca em largura. Essa classe utiliza o método *expandir* da classe *Grafo* e então verifica cada um dos sucessores do vértice expandido e escolhe o de menor custo, sendo assim, esse nó escolhido será expandido e assim sucessivamente até encontrar o vértice objetivo.

A classe *DFS* é responsável por gerar a busca em profundidade. Essa classe utiliza o método *expandir* da classe *Grafo* e então verifica o sucessor com o menor custo, para na sequência o expandir e assim por diante até encontrar o vértice objetivo.

A classe *SubidaEncosta* é responsável por gerar a busca subida de encosta. Também utiliza o método *expandir* da classe *Grafo* e então verifica qual é o primeiro sucessor que possui custo maior ao vértice pai, aquele sucessor que for maior é então visitado e se repete a procura do sucessor desse vértice que tenha custo maior até encontrar o vértice objetivo.

A classe *Aestrela* é responsável por gerar a busca A*. Utiliza o método *expandir* da classe *Grafo* e então calcula a heurística dos sucessores do vértice para escolher o que obtiver menor custo. A heurística é calcula utilizando distância de manhattan.

4 Experimentos e Resultados

Foi aplicado em todos os algoritmos, o mesmo experimento, para que pudesse ser avaliado o comportamento dos diferentes algoritmos de busca. O experimento consistiu, primeiramente, em avaliar a saída das ações (acima, abaixo, direita ou esquerda) de cada um dos algoritmos e verificar se as 10 primeiras ações estão seguindo com o esperado para a resolução do problema do quebra-cabeça de 8 peças. A escolha das 10 primeiras ações, se deu pelo fato das saídas serem muito extensas, o que inviabiliza a avaliação de todas as ações realizadas por cada um dos algoritmos. Em paralelo, foi avaliado o tempo que cada um dos algoritmos leva para realizar a busca.

Os algoritmos BFS e DFS, resolveram o problema do quebra-cabeças de 8 peças com soluções diferentes e com tempos de execução diferentes. O BFS para resolver o problema precisou executar 747 ações, enquanto o DFS para o mesmo caso executou 1411 ações. O tempo de execução do BFS também é superior ao do DFS, sendo executada em 0.053 segundos (BFS) e o DFS executado em 0.13 segundos. A diferença ocorre pelo fato do algoritmo BFS passar por todos os sucessores de um vértice, em comparação com o DFS, que sempre busca o primeiro sucessor com menor custo possível, possibilitando maior procura pela reposta do problema pois pode deixar de olhar algum dos sucessores que possuem menor custo, mas não é o primeiro a ser testado e este ser o vértice solução.

O algoritmo subida da encosta não encontrou a solução do quebra-cabeça, o algoritmo em pouco tempo encontra um vértice maior que seu vizinho e então não prossegue com a busca. Já o algoritmo estrela, segue uma sequência de ações, e portanto, não chega a solução do problema pois a heurística sempre leva o algoritmo a escolher os vértices das mesmas ações.

5 Conclusão

Esse trabalho propôs a resolução do quebra-cabeça de 8 peças, por meio dos algoritmos de busca: BFS, DFS, Subida de Encosta e A*. Nos experimentos foram alcançados resultados esperados para os algoritmos BFS e DFs. O algoritmo subida de encosta não resolveu o problema em nenhum dos experimentos realizados, pois o vértice alcança valor maior que o de seus sucessores, e então não prossegue com a busca.

Referências

[1] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.