Metodos de Busca

Gabriel Campos, Luigi Sorrentino, Matheus Pôssas, Vítor França

Junho 2019

1 Jogo escolhido

Para o desenvolvimento deste trabalho, foi selecionado o jogo *8-puzzle*, uma implementação menor de outro jogo chamado de *15-Puzzle* [1].

O jogo *8-puzzle* consiste em um pequeno tabuleiro 3x3 com 8 peças, sendo que o espaço destinado para a nona peça é vazio. Primeiramente todas as peças são embaralhadas utilizando o espaço vazio para a movimentação das peças em sua adjacência, logo após o embaralhamento das peças o jogador deve re-organizar as peças até encontrar uma configuração estabelecida, como por exemplo os números ordenados de 1 até 8, tendo o espaço da primeira linha e primeira coluna destinado para o espaço em branco.

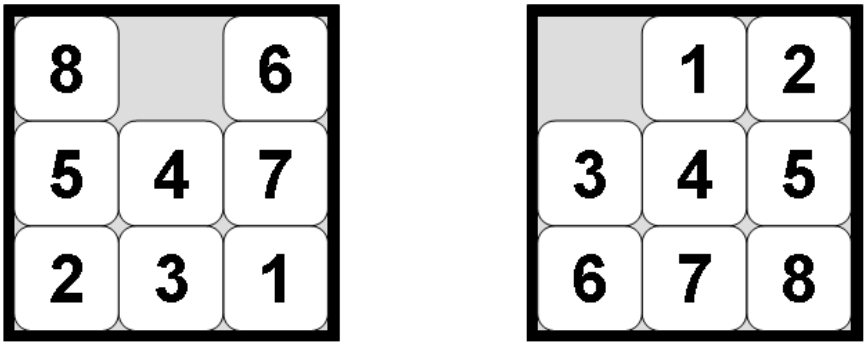

Figure 1: Exemplo de configuração durante o jogo a esquerda e objetivo final à direita

2 Descrição dos Métodos

2.1 Busca em largura

A busca em largura é um algoritmo de busca em grafos para realizar uma busca ou travessia num grafo ou estrutura de dados do tipo árvore, onde escolhe visitar

primeiro os vértices mais próximos da raiz, sendo necessário manter um conjunto de vértices candidatos alternativos composto por todo o nível inferior da árvore de busca. Encontra a solução ótima quando todos os operadores de mudança de estado têm o mesmo custo.

2.2 Busca gulosa

A busca gulosa tenta expandir o vértice mais próximo ao vértice final, assumindo que isso provavelmente levará a uma solução rápida. Dessa forma, a busca avalia os vértices utilizando apenas a função heurística. Não é completa nem ótima.

2.3 Busca A*

A busca pela melhor escolha é um refinamento da busca em largura, que em sua forma completa é conhecido como busca A*. Ambas estratégias começam pelo vértice inicial e mantêm um conjunto de caminhos candidatos. A busca A* calcula uma estimativa heurística para cada candidato e escolhe expandir o melhor candidato segundo esta estimativa. Se a heurística utilizada é admissível, isto é, nunca superestima a distância até o vértice final, então o algoritmo A* é completo e ótimo.

3 Resultados obtidos por cada método

Na figura abaixo, demonstra os códigos que foram utilizados para resolução do problema, sendo ele o main.py para chamar os 3 algoritmos utilizados, o A_Estrela.py utiliza do puzzle.py para utilização da sua heurística, o Gulosa.py utiliza do puzzle2.py para utilização de sua heurística e o BFS.py por ser uma busca não informada não utiliza heurística. A figura também demonstra como era a forma inicial do puzzle-8 e a forma desejada final.

main.py | A_Estrela.py | puzzle.py | BFS.py | Gulosa.py | puzzle2.py

main.py > ...

```
1  from time import time
2  from BFS import bfs
3  from Gulosa import Gulosa
4  from A_Estrela import A_Estrela
5  from puzzle import Puzzle
6  from puzzle2 import puzzle2
7
8  #Posicao inicial do puzzle8
9  inicio=[[2, 6, 1,
10         7, 5, 3,
11         0, 8, 4]]
12 #posicao final   [1,2,3,
13 #                8,0,4,
14 #                7,6,5]
15
```

Figure 2: Descrição dos algoritmos

Na figura 3,é demonstrado como é chamado cada método citado acima e seus parâmetros, que é o formato inicial do puzzle-8, o formato final é criado dentro de cada método. O algoritmo demonstra as alterações que deve ser feita no puzzle-8 da peça em branco, se ela deve subir,descer, para esquerda ou para direita, além do tempo gasto para cada método.

```
15
16  for i in range(0,1):
17      t0=time()
18      bfs_met=bfs(inicio[i])
19      t1=time()-t0
20      print('Caminho do BFS até a resposta:', bfs_met)
21      print('Tempo Gasto pelo BFS:',t1,"\n")
22
23      t0=time()
24      AlgGuloso = Gulosa(inicio[i])
25      t1=time()-t0
26      print('Caminho do Algoritmo Guloso até a resposta:', AlgGuloso)
27      print('Tempo Gasto:',t1,"\n")
28
29      t0 = time()
30      aEstrela = A_Estrela(inicio[i])
31      t1 = time() - t0
32      print('caminho do Algoritmo A* até a resposta:',aEstrela)
33      print('Tempo Gasto:', t1,"\n")
34
```

Figure 3: Chamada dos métodos

A figura 4 apenas demonstra a execução do algoritmo e o seus resultados, a ordem de movimentação da peça em branco (0) e o tempo gasto por cada método.

```
D:\PUC\IA\8puzzle>C:/Users/gabri/AppData/Local/Programs/Python/Python37-32/python.exe d:/P
UC/IA/8puzzle/main.py
Caminho do BFS até a resposta: ['Direita', 'Cima', 'Esquerda', 'Cima', 'Direita', 'Direita
', 'Baixo', 'Baixo', 'Esquerda', 'Cima', 'Esquerda', 'Cima', 'Direita', 'Baixo', 'Esquerda
', 'Baixo', 'Direita', 'Cima']
Tempo Gasto pelo BFS: 13.405142068862915

Caminho do Algoritmo Guloso até a resposta: ['Direita', 'Cima', 'Cima', 'Direita', 'Baixo'
, 'Baixo', 'Esquerda', 'Cima', 'Cima', 'Esquerda', 'Baixo', 'Direita', 'Cima', 'Esquerda',
 'Baixo', 'Baixo', 'Direita', 'Cima', 'Cima', 'Esquerda', 'Baixo', 'Direita', 'Baixo', 'Es
querda', 'Cima', 'Cima', 'Direita', 'Baixo']
Tempo Gasto: 0.021955499649047854

caminho do Algoritmo A* até a resposta: ['Direita', 'Cima', 'Esquerda', 'Cima', 'Direita',
 'Direita', 'Baixo', 'Baixo', 'Esquerda', 'Cima', 'Esquerda', 'Cima', 'Direita', 'Baixo',
'Esquerda', 'Baixo', 'Direita', 'Cima']
Tempo Gasto: 0.019916057586669922
```

Figure 4: Respostas dos problemas

4 Considerações finais sobre os métodos

Analisando a figura 2, podemos observar os tempos e as movimentações que cada algoritmo executou até encontrar a resposta final. Como esperado o BFS obteve um tempo muito alto já que ele é um algoritmo de busca sem informação. Ele percorre a árvore analisando todas as possibilidades a partir do nó inicial. Enumerando os vértices por onde passou é montada uma estrutura de fila para saber o próximo vértice a ser expandido. O BFS para esse problema não gera solução ótima.

Ao analisar o próximo resultado, vemos o uso do algoritmo guloso, esse obteve o segundo melhor tempo na resolução do problema. Diferentemente do BFS ele tem é uma heurística analisando o menor "h(n)" onde ele pega sempre a menor heurística no momento. A posição inicial do puzzle-8 irá influenciar bastante nos testes, já que ele usa o menor "h(n)" na hora da análise. Esse nem sempre obtém a solução ótima.

Por fim temos o algoritmo A*, que mostrou ter os melhores resultados entre os outros algoritmos já citados. O A* é uma alteração do Guloso. Essa alteração consiste no A* manter uma fila de possíveis caminhos a serem analisados. Ele vai expandindo o menor da fila, considerando o peso da aresta e a heurística do guloso. O A* gera a solução ótima no Puzzle-8.

References

[1] 8-puzzle. http://www.aiai.ed.ac.uk/ gwickler/eightpuzzle-inf.html. Último acesso em: 12/06/2019.