UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Mestrado em Computação

Comparação entre os algoritmos de busca DFS, BFS, IDS e A* no contexto do jogo 8-puzzle

Karine Pestana Ramos

Pelotas, 2019

SUMÁRIO

1 INTRODUÇÃO

Esse relatório tem como objetivo descrever sobre o desenvolvimento de alguns algoritmos de busca sem informação e de busca informada com o objetivo de resolver o problema do quebra cabeça deslizante. Assim, este relatório contém informações sobre o desenvolvimento de cada um dos algoritmos implementados e também, discussões sobre a execução de cada um neles, tal qual resultados obtidos e comparações entre eles.

1.1 Proposta do trabalho

A proposta do trabalho é de implementar em qualquer linguagem escolhida quatro algoritmos de busca diferentes para o problema do quebra cabeça deslizante, para fins deste trabalho foi utilizado Python, importando-se a biblioteca Numpy, por ser uma biblioteca matemática que facilitaria as operações e tratamento de matrizes. Previamente, antes da execução da busca em si é necessário embaralhar a matriz inicial para garantir que o problema tenha uma solução possível. Os algoritmos de busca implementados são busca em largura, busca em profundidade, busca em profundidade iterativa e o A* (com o uso de duas heurísticas diferentes). As buscas foram executadas para matrizes 3x3 e 4x4.

2 OS ALGORITMOS E SUAS IMPLEMENTAÇÕES

2.1 Busca em Largura (BFS)

O código feito para embaralhar a matriz inicial é o mesmo utilizado em todos os algoritmos de busca. Sendo a busca em largura a primeira a ter sido desenvolvida foi a mais demorada pois foi necessário a construção dessa etapa de embaralhamento das peças do quebra cabeça e do algoritmo de busca em si. A implementação desse algoritmo pode ser encontrada no arquivo "BFS.py".

2.2 Busca em Profundidade (DFS)

O desenvolvimento desse algoritmo de busca em comparação ao anterior dado que eles funcionam de modos similares somente percorrendo a árvore de busca de maneira diferente foi relativamente simples. Ambos os códigos são bem similares, com uma pequena diferença no modo de inserção na fila utilizada ao abrir os nós filhos do nó que está sendo analisado no momento. A implementação desse algoritmo pode ser encontrada no arquivo "DFS.py".

2.3 Busca em Profundidade Iterativa (IDS)

A ideia desse algoritmo é buscar em profundidade de forma iterativa, assim é definido um limite para a busca na árvore e para cada vez que é percorrida a árvore inteira e não se encontra uma solução é acrescentado mais um nível de busca. Isso ocorre iterativamente até que uma solução seja encontrada. A implementação desse algoritmo pode ser encontrada no arquivo "BuscaIterativa.py".

2.4 A*

O algoritmo A* percorre a árvore de busca gerada de maneira diferente dos anteriores e faz uso de um critério para decidir o próximo nó a ser visitado. Esse critério é decidido através de uma função heurística. Para fins deste trabalho foi desenvolvido o

A* com o uso de duas heurísticas diferentes, sendo essas uma a contagem do número de peças erradas no tabuleiro e a outra a distância de Manhatan, que consiste na distância para se deslocar a peça até a posição correta. Assim como os anteriores também é necessário embaralhar as peças antes de sua execução. As implementações desse algoritmo podem ser encontradas nos arquivos "AEstrela.py"e "AEstrela2.py", onde no primeiro foi implementada a heurística da contagem do número de peças em posição errada no tabuleiro e no segundo foi implementada a heurística da distância de Manhatan.

2.5 Considerações gerais sobre as implementações

Para a execução de qualquer um dos algoritmos pelo menos dois comandos devem ser executados:

1. `python Embaralha.py -d m -n t`
2. `python nome_do_arquivo.py -d m`

   Onde:

   - `-d`: espera um inteiro m como entrada e define o tamanho da matriz que será gerada;
   - `-n`: espera um inteiro t como entrada que define o número de vezes que a matriz será embaralhada.

O código de cada algoritmo funciona de maneira individual e foram feitos em arquivos diferentes. Contudo é garantido que para cada execução uma vez que a matriz quadrada de tamanho x é embaralhada n vezes todos tentam solucionar a mesma matriz. O embaralhamento ocorre de forma aleatória, assim dado um estado atual do tabuleiro é possível ele gerar qualquer um dos movimentos permitidos pelo jogo, podendo até mesmo em um momento fazer um jogada e posteriormente desfazê-la, já que ocorre de maneira probabilística.

Tratando-se da implementação só o ato de gerar uma matriz original e embaralhar a mesma já tem por si uma certa dificuldade. Quanto aos algoritmos de forma geral eles todos usam funções iguais para fazer as movimentações possíveis do tabuleiro, ou seja é a mesma função para todos quando há um estado atual e é necessário saber quais os próximos possíveis estados. Essa função foi implementada usando principalmente condicionais, porém os algoritmos estavam lentos então houve necessidade de alteração (para otimizá-la). A requisição dessa função é alta pois constamente a busca na árvore está abrindo nós, criando a necessidade de gerar outros estados. Acredito que isso tenha sido um desafio ou até mesmo dificuldade pois dado que já estava

implementado com uma lógica muitas vezes é difícil enxergar outras possibilidades de solução. Outra dificuldade que é válido destacar seria a respeito da primeira implementação, que foi a busca em largura. Por ser o primeiro até o momento da lógica e do racíocinio serem construídos demandou um certo tempo e esforço. Ademais os próximos tiveram suas dificuldades particulares, já que cada um percorre a árvore de maneira diferente, contudo fluíram de maneira um pouco melhor.

3 RESULTADOS

Com o intuito de facilitar a execução dos algoritmos os códigos foram implementados de maneira que o tamanho da matriz pode ser facilmente alterado como um parâmetro de entrada. Ao embaralhar é possível incluir o tamanho da matriz desejada e o número de vezes que se deseja embaralhar.

Para a geração dos resultados, tempo de busca e quantidade de nós abertos foi estabelecido um limite de tempo de busca de no máximo uma hora para o algoritmo encontrar uma solução. Para isso é feita uma verificação constante se o limite de tempo já foi atingido para permitir a continuidade da busca. Além disso a execução dos testes ocorreu fazendo uso de dois scripts shell para facilitar a obtenção dos resultados. Ambos scripts foram feitos para executar todos os algoritmos em matrizes embaralhadas 30, 40 e 50 vezes. Um script trata-se das execuções de matrizes 3x3 e sua saída pode ser vista no arquivo "saida_3x3.txt". Já o segundo tem o objetivo de executar as matrizes 4x4 e sua saída pode ser vista no arquivo "saida_4x4.txt". O caminho percorrido por cada algoritmo para encontrar a solução pode ser encontrado nos arquivos de saída. Ambos os scripts podem ser vistos no anexo através das Figuras 3 e 4. Os resultados das saídas dos scripts (tempos de execução, número de nós abertos e se concluiu a busca) podem ser vistos de forma resumida das tabelas 1 e 2.

3x3	Busca em Largura			Busca em Profundidade			Busca Iterativa			Busca A* - Heurística 1			Busca A* - Heurística 2		
Número de vezes que a matriz foi embaralhada	Tempo	Nós abertos	Encontrou solução?	Tempo	Nós abertos	Encontrou solução?	Tempo	Nós abertos	Encontrou solução?	Tempo	Nós abertos	Encontrou solução?	Tempo	Nós abertos	Encontrou solução?
30	0.01 s	512	Sim	1 hora	700173	Não	2 horas	4194304	Não	0.10 s	703	Sim	0.02 s	280	Sim
40	0.02 s	860	Sim	1 hora	706779	Não	2 horas	4194304	Não	0.08 s	661	Sim	0.02 s	265	Sim
50	51.61 s	256335	Sim	1 hora	1039273	Não	2 horas	4194304	Não	14 min	65229	Sim	11 min	61212	Sim

Figura 1 – Resultado das execuções das matrizes 3x3

4x4	Busca em Largura			Busca em Profundidade			Busca Iterativa			Busca A* - Heuristica 1			Busca A* - Heuristica 2		
Número de vezes que a matriz foi embaralhada	Tempo	Nós abertos	Encontrou solução?	Tempo	Nós abertos	Encontrou solução?	Tempo	Nós abertos	Encontrou solução?	Tempo	Nós abertos	Encontrou solução?	Tempo	Nós abertos	Encontrou solução?
30	0.049 s	14500	Sim	1 hora	697586	Não	2 horas	4194304	Não	0.42 s	1503	Sim	0.85 s	2114	Sim
40	1 hora	1493337	Não	1 hora	705031	Não	2 horas	4194304	Não	1 hora	121882	Não	1 hora	129509	Não
50	Não houve execução			Não houve execução			Não houve execução			Não houve execução			Não houve execução		

Figura 2 – Resultado das execuções das matrizes 4x4

3.1 Discussão dos resultados

Nas execuções das matrizes 3x3 os algoritmos que se destacaram foram a Busca em Largura e o A* fazendo uso das duas heurísticas. Em todas as execuções as buscas foram capazes de encontrar a solução do quebra cabeça. O A* com a heurística da distância de Manhatan ainda foi o algoritmo que entre os três teve a busca que abriu menos nós em todas as execuções. O que tem um fundamento considerando que nessa execução é uma busca informada com uma boa heurística. Comparando o número de nós abertos do A* com a primeira heurística com a Busca em Largura também é possível observar um número menor de nós abertos. A Busca em Profundidade em nenhuma execução encontrou a solução, sempre estourando o limite de tempo. A Busca em Profundidade Iterativa também não foi capaz de encontrar a solução em nenhum das execuções, contudo ao observar a tabela é possível reparar que as execuções sempre abrem o exato mesmo número de nós, assim acredito que talvez o código tenha excedido o limite de memória disponível (o mesmo ocorre na execução 4x4 também).

Nas execuções das matrizes 4x4 o destaque também ocorre para os mesmos algoritmos na primeira execução. E o algoritmo com o menor tempo e menor número de nós abertos é o A* com a primeira heurística. Na segunda execução nenhum algoritmo foi capaz de encontrar a solução, levando então a não execução da terceira vez com as matrizes 4x4.

De maneira geral apesar da Busca em Largura se destacar em algumas execuções é visto que a busca informada com o uso do A* gera resultados melhores, pois além de se destacar em mais de um caso o algoritmo sempre encontra a solução ótima para o problema.

ANEXO A SCRIPTS

```
#!/bin/bash
python Embaralha.py -d 3 -n 30
python AEstrela.py -d 3
python AEstrela2.py -d 3
python BFS.py -d 3
python DFS.py -d 3
python BuscaIterativa.py -d 3
python Embaralha.py -d 3 -n 40
python AEstrela.py -d 3
python AEstrela2.py -d 3
python BFS.py -d 3
python DFS.py -d 3
python BuscaIterativa.py -d 3
python Embaralha.py -d 3 -n 50
python AEstrela.py -d 3
python AEstrela2.py -d 3
python BFS.py -d 3
python DFS.py -d 3
python BuscaIterativa.py -d 3
```

Figura 3 – Script de execução dos algoritmos com matrizes 3x3

```
#!/bin/bash
python Embaralha.py -d 4 -n 30
python AEstrela.py -d 4
python AEstrela2.py -d 4
python BFS.py -d 4
python DFS.py -d 4
python BuscaIterativa.py -d 4
python Embaralha.py -d 4 -n 40
python AEstrela.py -d 4
python AEstrela2.py -d 4
python BFS.py -d 4
python DFS.py -d 4
python BuscaIterativa.py -d 4
python Embaralha.py -d 4 -n 50
python AEstrela.py -d 4
python AEstrela2.py -d 4
python BFS.py -d 4
python DFS.py -d 4
python BuscaIterativa.py -d 4
```

Figura 4 – Script de execução dos algoritmos com matrizes 4x4