



PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS  
Instituto de Ciências Exatas e de Informática

## Trabalho Prático - Métodos de Busca\*

Tabalho Disciplina Inteligência Artificial

Lucas Cicutti  
Guilherme Galvão  
Octávio Queiroz  
Yan Humphreis

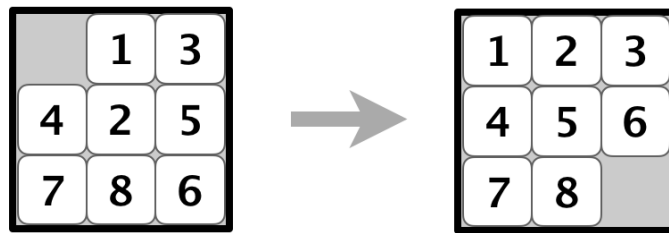
---

\* Artigo apresentado ao Instituto de Ciências Exatas e de Informática da Pontifícia Universidade Católica de Minas Gerais na disciplina de Inteligência Artificial.

## 1 - PUZZLE 8

O puzzle8 (também chamado de Gem Puzzle, Boss Puzzle, Mystic Square e muitos outros) é um quebra-cabeça deslizante que consiste em um quadro de quadrados numerados em ordem aleatória com um azulejo faltando. O quebra-cabeça também existe em outros tamanhos, particularmente como o maior, com 15 peças, porém o grupo preferiu implementar os métodos com o quadro de 8 peças por questões de performance. O objetivo do quebra-cabeça é colocar as peças em ordem, fazendo movimentos deslizantes que usam o espaço vazio.

**Figura 1 – Exemplo de estado inicial e final do puzzle, respectivamente**



## 2 - MÉTODOS UTILIZADOS

### 2.1 - Busca em Largura

Em inglês, Breadth First Search (BFS), consiste em, a partir de um vértice de origem, explorar primeiramente todos os seus vizinhos e, em seguida repetir o procedimento para cada vizinho, o que nos permite calcular a distância (caminho mínimo) do vértice de origem até qualquer vértice que possa ser alcançado.

### 2.2 - Busca em Profundidade

Em inglês, Depth First Search (DFS), consiste em, a partir de um vértice de origem, busca um vértice adjacente, até que não existam mais vértices a visitar. Ou seja, ao contrário do BFS, o DFS não expande todos os nós do vértice em que se encontra, ele expande apenas um, e vai descendo na árvore até encontrar uma solução.

## **2.3 - A\***

A\* é um algoritmo de busca informado, ou uma pesquisa best-first, significando que é formulado em grafos ponderados: a partir de um nó inicial específico de um grafo, ele procura encontrar um caminho para o nó de meta fornecido tendo o menor custo (menor distância percorrida, menor tempo, etc.). Isso é feito mantendo uma árvore de caminhos originados no nó inicial e estendendo esses caminhos, uma aresta de cada vez até que ele chegue no nó final.

## **3 - MÉTRICAS DE DESEMPENHO**

Para medir o desempenho de cada algoritmo, foram escolhidas 3 métricas: O tempo de execução, o número de nós que o algoritmo visita, e uma formula de custo que consiste no somatório dos valores absolutos entre o valor cada peça em uma posicao, menos o valor da peça que deveria estar naquela posicao. O custo de cada nó será o seu próprio custo, mais a soma do custo de todos os seus ancestrais.

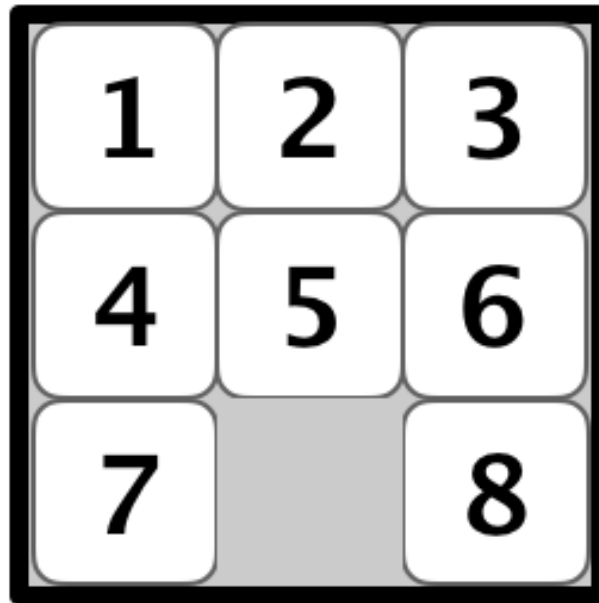
## **4 - RESULTADOS OBTIDOS**

### **4.1 Exemplo "fácil"**

No que se diz respeito ao algoritmo que o grupo concluiu ser o pior para esta instância, ou o menos confiável para fazer esta busca, este é facilmente a busca em profundidade.

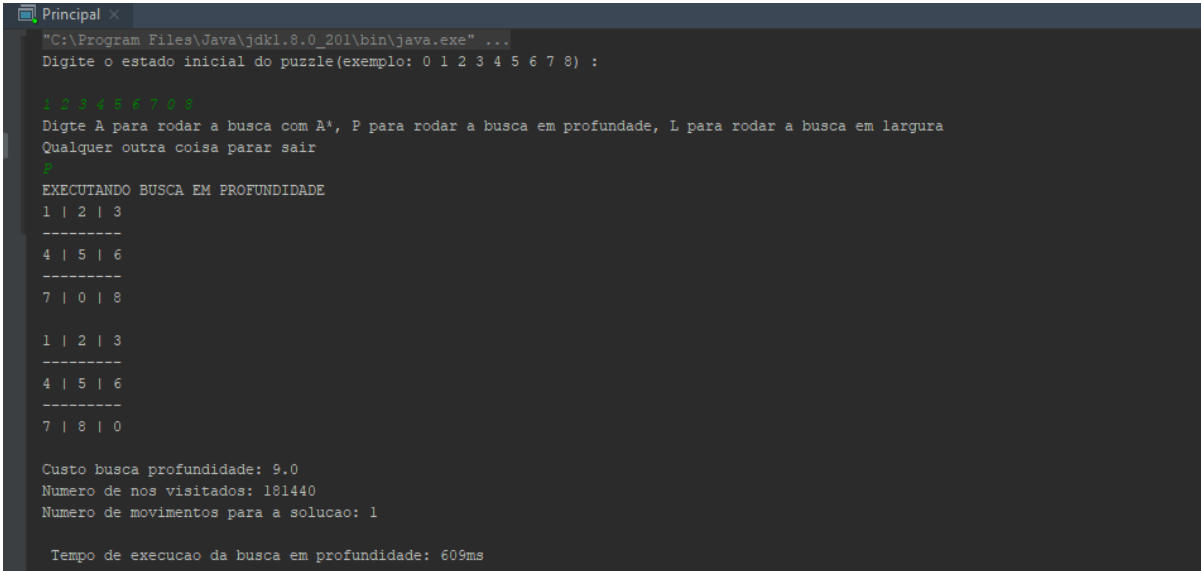
Iremos rodar o puzzle8 mais simples, que necessita de apenas 1 movimento para ser solucionado, para demonstrar a diferença entre eles:

**Figura 2 – Puzzle8 resolvível com 1 movimento**



Rodando a busca em profundidade para o puzzle acima, apesar de ter obtido a solução com apenas 1 movimento e com um custo baixo, o algoritmo visitou 181440 nós, e levou 609ms para ser executado.

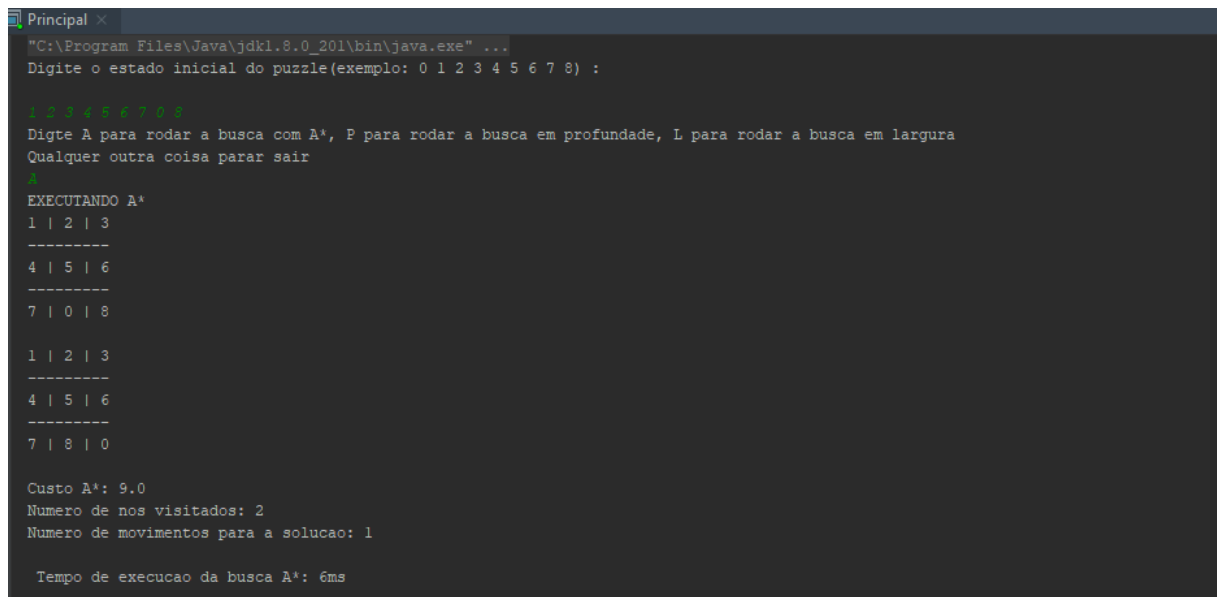
**Figura 3 – Resultado obtido com busca em profundidade**



```
Principal x
"C:\Program Files\Java\jdk1.8.0_201\bin\java.exe" ...
Digite o estado inicial do puzzle(exemplo: 0 1 2 3 4 5 6 7 8) :
1 2 3 4 5 6 7 8
Digite A para rodar a busca com A*, P para rodar a busca em profundidade, L para rodar a busca em largura
Qualquer outra coisa para sair
P
EXECUTANDO BUSCA EM PROFUNDIDADE
1 | 2 | 3
-----
4 | 5 | 6
-----
7 | 0 | 8
-----
1 | 2 | 3
-----
4 | 5 | 6
-----
7 | 8 | 0
-----
Custo busca profundidade: 9.0
Numero de nos visitados: 181440
Numero de movimentos para a solucao: 1
Tempo de execucao da busca em profundidade: 609ms
```

Já o A\* teve desempenho bem melhor, chegando na solução com apenas 1 movimento, e com tempo de execução de 6ms, com um custo de 9, porém visitando apenas 2 nós.

**Figura 4 – Resultado obtido com busca A\***



```
Principal x
"C:\Program Files\Java\jdk1.8.0_201\bin\java.exe" ...
Digite o estado inicial do puzzle(exemplo: 0 1 2 3 4 5 6 7 8) :

1 2 3 4 5 6 7 8
Digite A para rodar a busca com A*, P para rodar a busca em profundidade, L para rodar a busca em largura
Qualquer outra coisa parar sair
A
EXECUTANDO A*
1 | 2 | 3
-----
4 | 5 | 6
-----
7 | 0 | 8

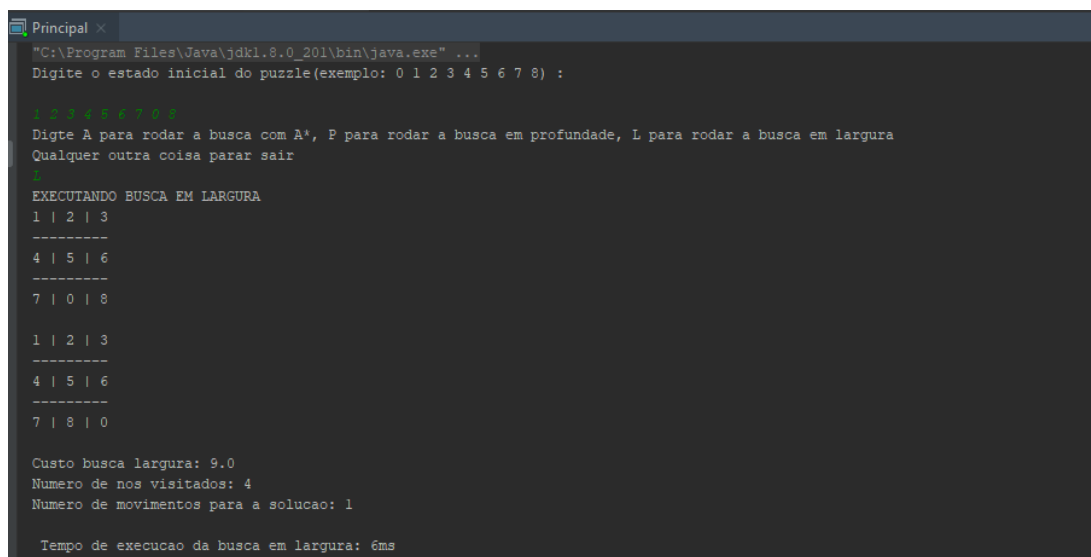
1 | 2 | 3
-----
4 | 5 | 6
-----
7 | 8 | 0

Custo A*: 9.0
Numero de nos visitados: 2
Numero de movimentos para a solucao: 1

Tempo de execucao da busca A*: 6ms
```

A busca em largura para este exemplo teve comportamento similar ao A\*, apesar de ter visitado mais nós para chegar na solução, ela ainda conseguiu solucionar com apenas um movimento, com o mesmo custo de 9, levando 6 ms.

**Figura 5 – Resultado obtido com busca em largura**



```
Principal x
"C:\Program Files\Java\jdk1.8.0_201\bin\java.exe" ...
Digite o estado inicial do puzzle(exemplo: 0 1 2 3 4 5 6 7 8) :

1 2 3 4 5 6 7 8
Digite A para rodar a busca com A*, P para rodar a busca em profundidade, L para rodar a busca em largura
Qualquer outra coisa parar sair
L
EXECUTANDO BUSCA EM LARGURA
1 | 2 | 3
-----
4 | 5 | 6
-----
7 | 0 | 8

1 | 2 | 3
-----
4 | 5 | 6
-----
7 | 8 | 0

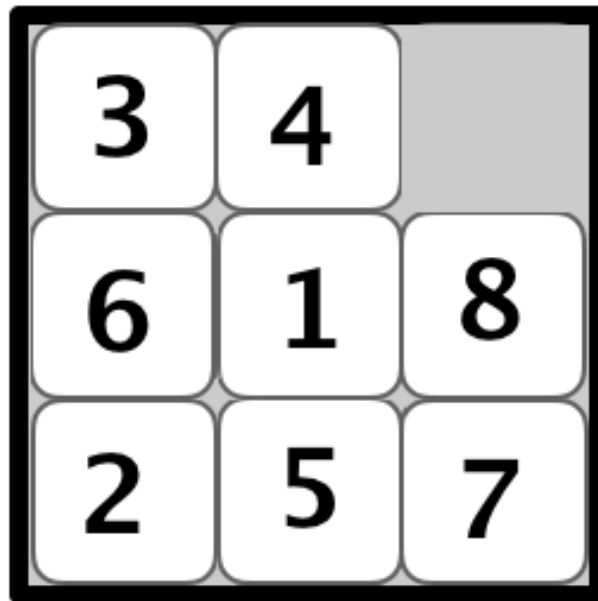
Custo busca largura: 9.0
Numero de nos visitados: 4
Numero de movimentos para a solucao: 1

Tempo de execucao da busca em largura: 6ms
```

## 4.2 Exemplo "difícil"

Em seguida realizamos o teste com uma instância mais complexa, que necessita de 26 movimentos para a solução, para ver como o algoritmo se comporta com problemas maiores.

**Figura 6 – Puzzle8 resolvível com 26 movimentos**



A busca em profundidade para esta instância teve um custo de 131286, visitando 177034 nós, e obtendo uma solução com 4804 movimentos, com um tempo de execução de 702ms, um resultado longe do esperado, devido ao elevado número de movimentos realizados.

**Figura 7 – Resultado obtido com busca em profundidade**

```
Principal x
1 | 2 | 3
-----
0 | 5 | 6
-----
4 | 7 | 8

1 | 2 | 3
-----
4 | 5 | 6
-----
0 | 7 | 8

1 | 2 | 3
-----
4 | 5 | 6
-----
7 | 0 | 8

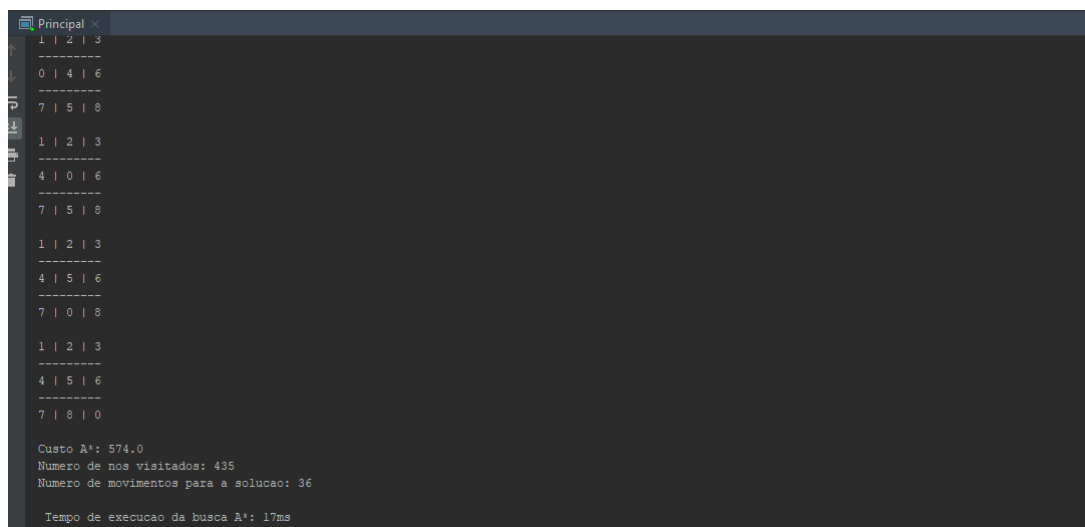
1 | 2 | 3
-----
4 | 5 | 6
-----
7 | 8 | 0

Custo busca profundidade: 131286.0
Numero de nos visitados: 177034
Numero de movimentos para a solucao: 4804

Tempo de execucao da busca em profundidade: 702ms
```

Ao rodar o A\* obtivemos um resultado substancialmente melhor, com um custo de 574, 435 nós visitados, e a solução obtida com 36 movimentos, executado em 17ms.

**Figura 8 – Resultado obtido com busca A\***



```
Principal x
1 | 2 | 3
-----
0 | 4 | 6
-----
7 | 5 | 8

1 | 2 | 3
-----
4 | 0 | 6
-----
7 | 5 | 8

1 | 2 | 3
-----
4 | 5 | 6
-----
7 | 0 | 8

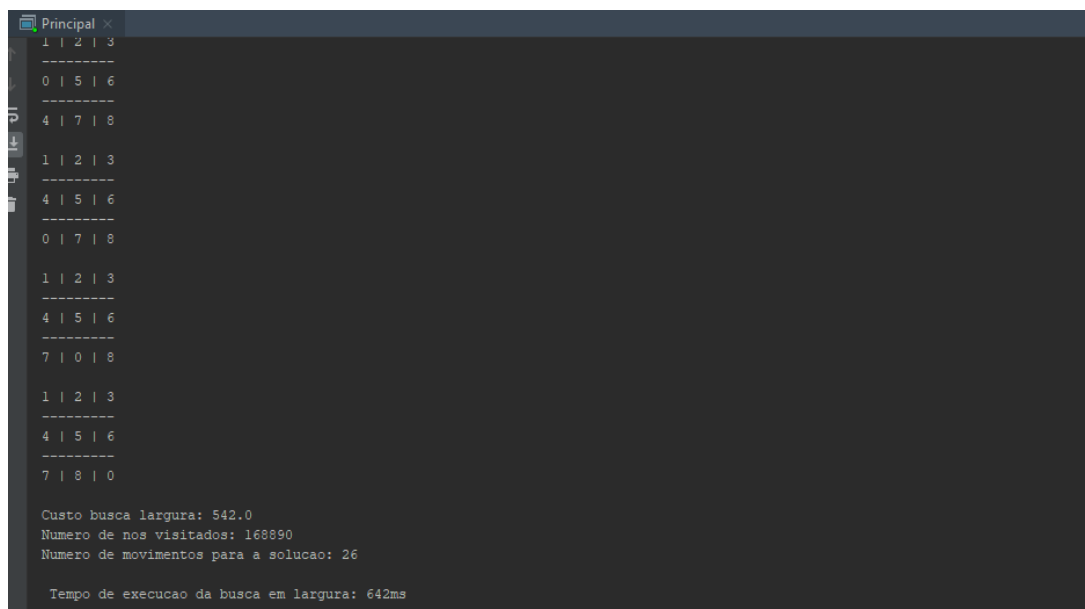
1 | 2 | 3
-----
4 | 5 | 6
-----
7 | 8 | 0

Custo A*: 574.0
Numero de nos visitados: 435
Numero de movimentos para a solucao: 36

Tempo de execucao da busca A*: 17ms
```

A busca em largura obteve um resultado com um custo de 542, 168890 nós visitados, e a solução obtida com os 26 movimentos, executado em 642ms.

**Figura 9 – Resultado obtido com busca em largura**



```
Principal x
1 | 2 | 3
-----
0 | 5 | 6
-----
4 | 7 | 8

1 | 2 | 3
-----
4 | 5 | 6
-----
0 | 7 | 8

1 | 2 | 3
-----
4 | 5 | 6
-----
7 | 0 | 8

1 | 2 | 3
-----
4 | 5 | 6
-----
7 | 8 | 0

Custo busca largura: 542.0
Numero de nos visitados: 168890
Numero de movimentos para a solucao: 26

Tempo de execucao da busca em largura: 642ms
```



## 5 - CONCLUSÃO

Ao analisar os resultados obtidos, concluímos que não existe um melhor algoritmo "absoluto" em relação aos outros. Tudo depende da métrica que se deseja otimizar, por exemplo, se é desejado rodar um algoritmo mais rápido e com melhor performance, o A\* seria recomendado com base em nossos testes, mas ele nem sempre obtém a solução com menor número de passos, como a busca em largura consegue, porém em média ela visita mais nós, e possui tempo de execução maior que o A\*. A única conclusão que os testes nos permitem chegar é que o algoritmo de busca em profundidade aplicado para este problema não pode ser considerado uma boa solução, uma vez que além de tempos de execução elevados, nem sempre ele obtém uma solução desejável.