# **Relatório do Trabalho Prático 1**

# **Descrição do Puzzle**

O **8-Puzzle** é um jogo clássico de resolução de problemas que consiste em um tabuleiro 3x3 contendo 8 peças numeradas de 1 a 8 e um espaço vazio (representado pelo número 0). O objetivo do jogo é mover as peças para transformar um estado inicial dado em um estado final, utilizando uma sequência de movimentos válidos (cima, baixo, esquerda e direita).

## **Algoritmos Implementados**

* 1. **Breadth-First Search (BFS)**

O algoritmo **BFS** é uma estratégia de busca não informada que explora todos os nós de um determinado nível antes de passar para o próximo. Para este problema, BFS garante a solução ótima caso todos os movimentos tenham o mesmo custo.

Passos:

* Verificar se o problema tem solução.
* Criar nó inicial e colocá-lo na fila.
* Manter um conjunto de estados já visitados para evitar ciclos.
* Executar loop de busca:
  + Remover primeiro nó da fila (FIFO).
  + Se o estado for a solução, retorna o caminho encontrado.
  + Caso contrário, gerar os estados vizinhos aplicando os movimentos possíveis (cima, baixo, esquerda, direita).
  + Para cada estado gerado:
    - Se ainda não foi visitado, adicioná-lo à fila.
* Se a fila acabar e não encontrar solução o problema é insolúvel
  1. **A\***

**A\*** é um algoritmo de busca informada que utiliza **heurísticas** para priorizar a exploração dos caminhos mais promissores. Utiliza uma fila de prioridade (heapq), onde os estados são ordenados pelo custo total *f(n) = g(n) + h(n).*

Passos:

* Verificar se o problema tem solução.
* Criar nó inicial e inseri-lo na fila de prioridade com custo f(n) = h(n).
* Manter um conjunto de estados visitados para evitar ciclos.
* Executar loop de busca:
  + Remover o nó com custo menor total.
  + Se o estado for a solução, retorna o caminho encontrado.
  + Caso contrário, gerar os estados vizinhos aplicando os movimentos possíveis.
  + Para cada estado gerado:
    - Calcular g(n) = custo real e h(n) = custo heurístico (Manhattan ou Hamming).
    - Se o estado não for encontrado ou se encontrou um caminho melhor para ele, adicioná-lo à fila de prioridade.
* Se a fila acabar e não encontrar solução o problema é insolúvel.

Heurísticas utilizadas:

1. **Distância de Manhattan:** Soma as distâncias verticais e horizontais entre as peças e as suas posições corretas.
2. **Distância de Hamming:** Conta quantas peças estão fora do lugar.
3. **Testes e Resultados**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Caso** | **Estado Inicial** | **Estado Final** | **Movimentos BFS** | **Movimentos A\* (Manhattan)** | **Movimentos A\* (Hamming)** |
| **1** | [[1,2,3],  [4,5,6],  [7,0,8]] | [[1,2,3],  [4,5,6],  [7,8,0]] | **1** | **1** | **1** |
| **2** | [[1,2,3],  [0,4,6],  [7,5,8]] | [[1,2,3],  [4,5,6],  [7,8,0]] | **3** | **3** | **3** |
| **3** | [[8, 6, 7], [2, 5, 4],  [3, 0, 1]] | [[1,2,3],  [4,5,6],  [7,8,0]] | **31** | **31** | **31** |
| **4** | [[6, 4, 7],  [8, 5, 0],  [3, 2, 1]] | [[1,2,3],  [4,5,6],  [7,8,0]] | **31** | **31** | **31** |
| **5** | [[4,1,3],  [0,2,5],  [7,8,6]] | [[1,2,3],  [4,5,6],  [7,8,0]] | **5** | **5** | **5** |

1. **Estudo de Custo de Tempo e de Memória**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso** | **Algoritmo** | **Heurística** | **Tempo(s)** | **Memória (bytes)** |
| **3** | **BFS** | **-** | **45,598** | **69145836** |
| **3** | **A\*** | **Manhattan** | **11,452** | **17250726** |
| **3** | **A\*** | **Hamming** | **77.11784** | **64083894** |
| **4** | **BFS** | **-** | **45,328** | **68836284** |
| **4** | **A\*** | **Manhattan** | **11,459** | **17256822** |
| **4** | **A\*** | **Hamming** | **77,419** | **64018922** |
| **5** | **BFS** | **-** | **0,009** | **18516** |
| **5** | **A\*** | **Manhattan** | **0,006** | **4544** |
| **5** | **A\*** | **Hamming** | **0,003** | **4624** |

Os valores acima mostram que:

* BFS consome muita mais memória em instâncias mais complexas.
* A\* com heurística de Manhattan tende a ser mais rápido e eficiente que com Hamming.
* A heurística Hamming apresentou um tempo muito maior que Manhattan em instâncias mais complexas.
* Para a instâncias simples, como o caso 5, os tempos e memórias utilizados são muito baixos.

1. **Discussão dos Resultados**

* BFS resolve bem problemas pequenos, mas é ineficiente para problemas mais profundos devido ao grande consumo de memória.
* A\* é significativamente mais eficiente, pois utiliza heurísticas para priorizar caminhos promissores.
* A heurística Manhattan superou Hamming em termos de tempo de execução e memória utilizada.
* A heurística Hamming demonstrou ser mais custosa para instâncias maiores, apresentando um tempo de execução consideravelmente alto.

1. **Conclusão**

Neste trabalho, foram implementados dois algoritmos para resolver o 8-Puzzle: BFS e A\*. A análise comparativa mostrou que, embora BFS garanta soluções ótimas, seu consumo de memória é exponencial, tornando-o impraticável para instâncias mais complexas. O algoritmo A\* provou ser a alternativa mais eficiente, principalmente quando utiliza a heurística Manhattan, que apresentou bons resultados tanto em problemas mais complexos quanto em problemas simples, além de menor uso de memória. A heurística Hamming, apesar de apresentar melhores resultados em instâncias simples, mostrou maior ineficiência em instâncias mais complexas. No geral, os resultados destacam a importância do uso de heurísticas informadas para a resolução eficiente de problemas de busca.