**1. Introdução**

O objetivo do projeto foi implementar e comparar as seguintes abordagens:

1. **Busca em Largura (BFS)** - Cega
2. **Busca em Profundidade (DFS)** - Cega
3. **Busca Gulosa (Greedy Best-First Search)** - Heurística
4. *Busca A (A-Estrela)*\* - Heurística

A análise compara os algoritmos com base em métricas de eficiência (tempo e memória), qualidade da solução (custo/profundidade) e propriedades teóricas como completude e otimalidade.

**2. Abordagem de Desenvolvimento e Implementação**

A solução foi desenvolvida em **Python 3**, utilizando bibliotecas padrão para otimizar a performance das estruturas de dados.

* **Estrutura de Dados Principal:**
  + **Nó (Classe Node)**: Uma classe foi criada para representar cada nó na árvore de busca. Ela armazena o estado do tabuleiro (state), o nó pai (parent), a ação que levou ao estado atual (action) e o custo do caminho desde a raiz (cost, ou g(n)).
  + **Estado do Tabuleiro (state)**: Utilizou-se uma **tupla** de inteiros para representar o estado. Esta é uma escolha de implementação crucial, pois tuplas são imutáveis e "hasheáveis", permitindo a utilização de sets e dicionários para um controle de nós visitados (visited) com performance de O(1) em média.
* **Fronteira de Estados:**
  + **BFS**: Implementada com collections.deque, uma fila de alta performance, garantindo a operação FIFO (First-In, First-Out).
  + **DFS**: Implementada com uma lista (list) atuando como pilha, com operações append e pop para o comportamento LIFO (Last-In, First-Out).
  + **Gulosa e A**\*: Implementadas com a biblioteca heapq, que fornece uma fila de prioridade eficiente (min-heap), essencial para ordenar os nós com base em seu custo heurístico.
* **Função Heurística - Distância de Manhattan:**
  + Para as buscas informadas (Gulosa e A\*), foi implementada a heurística da **Distância de Manhattan**. Ela calcula, para cada peça, a soma das distâncias (linhas + colunas) entre sua posição atual e sua posição no estado-objetivo.
  + **Admissibilidade**: Esta heurística é **admissível**, pois nunca superestima o número real de movimentos necessários para resolver o quebra-cabeça. A distância de Manhattan para uma peça é o número mínimo de movimentos que ela precisa fazer, e a soma para todas as peças é uma estimativa otimista do custo total. A admissibilidade é um requisito para que a busca A\* garanta uma solução ótima.

**3. Resultados e Análise Comparativa**

A tabela abaixo consolida os resultados observados nos testes e as propriedades teóricas de cada algoritmo.

| Métrica | Busca em Largura (BFS) | Busca em Profundidade (DFS) | Busca Gulosa (Greedy) | A\* (A-Estrela) |
| --- | --- | --- | --- | --- |
| **Tempo de Execução** | Rápido em casos fáceis, mas se degrada exponencialmente em casos complexos. | Imprevisível, geralmente lento devido à exploração de caminhos irrelevantes. | Consistentemente o mais rápido para encontrar *uma* solução. | Excelente. Muito rápido, mesmo em casos complexos. |
| **Memória (Nós na Fronteira)** | Exponencial com a profundidade da solução. Muito alto em casos difíceis (ex: Teste 3). | Moderado, pois armazena apenas um caminho por vez. | Geralmente baixo, pois foca em um caminho promissor. | Menor que BFS, mas pode crescer consideravelmente em casos muito difíceis. |
| **Nós Gerados** | Alto. Explora todos os nós de uma profundidade antes de prosseguir. | Muito alto e ineficiente. Gera muitos nós em caminhos longos e desnecessários. | Muito baixo. A heurística poda a maior parte da árvore de busca. | **O mais baixo** entre os algoritmos ótimos. |
| **Profundidade da Solução** | **Ótima**. Sempre encontra o caminho mais curto. | **Não ótima**. Encontra a primeira solução que aparece, geralmente em caminhos muito profundos. | **Não ótima**. A solução encontrada frequentemente possui custo elevado. | **Ótima**. Garante encontrar o caminho de menor custo. |
| **Completa?** | Sim. | Sim (com controle de estados visitados, como implementado). | Sim (com controle de estados visitados). | Sim. |
| **Ótima?** | **Sim** | **Não** | **Não** | **Sim** |

**4. Discussão dos Resultados**

* **Busca em Largura (BFS)**: Provou ser um algoritmo confiável para encontrar a solução ótima. No entanto, sua ineficiência em problemas mais complexos (Teste 3) ficou evidente, onde o número de nós gerados e o tempo de execução cresceram drasticamente. É uma boa base de comparação, mas inviável para problemas difíceis.
* **Busca em Profundidade (DFS)**: Mostrou-se completamente inadequada para este problema. As soluções encontradas tiveram custos (profundidade) muito altos (41, 43, 48), enquanto as soluções ótimas eram muito mais rasas. A busca se perde em ramos profundos da árvore, resultando em um alto número de nós gerados para uma solução de péssima qualidade.
* **Busca Gulosa (Greedy)**: Destacou-se pela velocidade e baixo número de nós gerados. Contudo, essa "pressa" em minimizar a heurística h(n) sem considerar o custo já percorrido (g(n)) a levou a soluções não ótimas e, no Teste 3, a uma solução de qualidade inferior à da DFS. É rápida, mas não confiável para encontrar boas soluções.
* *A (A-Estrela)*\*: Foi o **algoritmo de melhor desempenho geral**. Ao combinar o custo do caminho percorrido (g(n)) com a estimativa da heurística (h(n)), a A\* conseguiu encontrar a solução ótima em todos os testes de forma muito mais eficiente que a BFS. No Teste 3, a A\* explorou aproximadamente **16% dos nós** que a BFS precisou para encontrar a mesma solução, demonstrando sua superioridade.

**5. Conclusão**

Os testes e a análise do código demonstram de forma conclusiva que a *Busca A com a heurística da Distância de Manhattan é a abordagem mais eficaz*\* para resolver o problema do 8-Puzzle. Ela une a garantia de otimalidade da BFS com a eficiência direcionada da Busca Gulosa, resultando em um algoritmo robusto, rápido e que sempre encontra a melhor solução.

O projeto serve como uma excelente demonstração prática dos trade-offs entre os diferentes paradigmas de algoritmos de busca:

* **Buscas Cegas**: São simples, mas sofrem com a complexidade do espaço de estados.
* **Buscas Heurísticas**: São poderosas, mas sua performance e qualidade de solução dependem inteiramente da "inteligência" da função heurística e da forma como ela é utilizada.