Documentação do 8-Puzzle Solver

Marcos Laine

29 de abril de 2025

Resumo

Este documento apresenta a descrição detalhada do projeto *8-Puzzle Solver* implementado em Python com interface gráfica `Tkinter`. São abordados os algoritmos de busca implementados (BFS, Busca Gulosa, Dijkstra e A*), as heurísticas utilizadas, a estrutura do código e instruções de uso.

1 Introdução

O *8-puzzle* é um quebra-cabeça clássico que consiste em um tabuleiro 3×3 contendo oito peças numeradas e um espaço vazio. O objetivo é alcançar uma configuração meta a partir de uma disposição inicial, movimentando as peças para o espaço vazio.

2 Visão Geral do Projeto

O projeto consiste em um único arquivo Python:

- `8-puzzle.py`: implementação completa da lógica de resolução e interface gráfica.

A interface gráfica permite:

1. Inserir manualmente um estado inicial.
2. Embaralhar automaticamente para um estado aleatório solucionável.
3. Selecionar o método de busca (BFS, Busca Gulosa, Dijkstra ou A*).
4. Escolher a heurística (Manhattan ou Misplaced) quando aplicável (A*).
5. Exibir tempo de execução, número de nós expandidos e número de passos.
6. Navegar passo a passo pela solução encontrada usando os botões `Anterior` e `Próximo`.
7. O espaço vazio é representado por uma célula em branco, como no quebra-cabeça tradicional.

3 Algoritmos de Busca

3.1 Busca em Largura (BFS)

O BFS explora o espaço de estados nível a nível, garantindo encontrar a solução com o menor número de movimentos, mas com alto consumo de memória.

3.2 Busca Gulosa (Greedy Best-First Search)

A busca gulosa utiliza apenas a heurística para escolher o próximo estado, sem considerar o custo do caminho já percorrido. É rápida, mas não garante o menor caminho.

3.3 Dijkstra

O algoritmo de Dijkstra encontra o caminho de menor custo sem utilizar heurística, considerando apenas o número de movimentos realizados.

3.4 A*

O A* combina o custo percorrido e a estimativa heurística.

- **Heurística Manhattan**: soma das distâncias de Manhattan de cada peça até sua posição alvo.
- **Heurística Misplaced**: número de peças fora do lugar.

O A* oferece compromisso ideal entre qualidade de solução e desempenho, dependendo da heurística.

4 Uso

Para executar o programa:

1. `execute o arquivo 8_puzze.exe`
    - `é possível que precise marcar como seguro em seu antivírus, pois como é um arquivo .exe gerado a partir de um código em python sem assinatura, ele detecte como malicioso`
2. **OU**
3. No terminal, execute:

```
cd caminho/para/o/arquivo
python 8-puzzle.py
```

4. Na janela, escolha ou embaralhe o estado e clique em *Resolver*.
5. Use os botões `Anterior` e `Próximo` para visualizar cada passo da solução.

5 Análise de Desempenho

Nesta seção, apresentamos os resultados reais obtidos para o mesmo estado inicial do 8-puzzle, utilizando diferentes algoritmos e heurísticas. Foram considerados o tempo de execução, o número de nós expandidos e o número de passos da solução.

5.1 Resultados Obtidos

- **A* (Manhattan)**:
  Passos: 18
  Tempo: 0,0062s
  Nós expandidos: 349

- **A* (Misplaced)**:
  Passos: 18
  Tempo: 0,0065s
  Nós expandidos: 1464

- **BFS**:
  Passos: 18
  Tempo: 0,0780s
  Nós expandidos: 22545

- **Busca Gulosa (Manhattan)**:
  Passos: 54
  Tempo: 0,0054s
  Nós expandidos: 737

- **Dijkstra**:
  Passos: 18
  Tempo: 0,0836s
  Nós expandidos: 19279

5.2 Discussão

Os resultados mostram que o algoritmo A* com heurística de Manhattan foi o mais eficiente em termos de nós expandidos e tempo, encontrando a solução ótima rapidamente. O A* com Misplaced também encontra a solução ótima, mas expande mais nós. O BFS e o Dijkstra garantem a solução ótima, porém com custo computacional significativamente maior. A Busca Gulosa é rápida e expande poucos nós, mas não garante a solução ótima, resultando em um caminho mais longo (mais passos).

6 Resolução de Problemas

- Se o antivírus detectar o executável como ameaça, adicione-o à lista de exceções ou assine digitalmente o arquivo.

- Certifique-se de que o Python e o Tkinter estão corretamente instalados.