Implementação da Comparação de Algoritmos de Busca no Problema do Labirinto com Obstáculos

21 de abril de 2025

1 Introdução

O problema de encontrar o caminho mais curto entre dois pontos em um labirinto com obstáculos é um exemplo clássico em Inteligência Artificial (IA). Ele simula desafios enfrentados em áreas como robótica, jogos, logística e sistemas de navegação autônoma, onde é necessário tomar decisões com base em um ambiente parcialmente ou totalmente conhecido.

Diante disso, algoritmos de busca tornam-se fundamentais para a resolução desse tipo de problema, permitindo que um agente explore o espaço de estados em busca da melhor rota. Tais algoritmos podem ser classificados em dois grandes grupos: buscas cegas, que não utilizam informação adicional sobre o objetivo (como a Busca em Largura e a Busca em Profundidade), e buscas heurísticas, que utilizam estimativas para guiar a exploração (como o A* e a Busca Gulosa).

Este trabalho tem como objetivo implementar e comparar quatro estratégias de busca aplicadas à resolução do problema do labirinto, analisando seu desempenho em termos de tempo de execução, uso de memória e qualidade da solução obtida. Os testes foram conduzidos em uma instância específica do problema, representada por um labirinto com obstáculos fixos, permitindo uma análise detalhada do comportamento de cada algoritmo.

2 Algoritimos de busca

Nesta seção, são descritas as quatro estratégias de busca utilizadas para resolver o problema do labirinto, divididas entre algoritmos de busca cega e heurística.

1. Busca em Largura (Breadth-First Search – BFS) - A Busca em Largura explora todos os nós em uma determinada profundidade antes de avançar para níveis mais profundos. Em um labirinto representado por uma matriz, isso significa que todos os caminhos com o mesmo número de passos são verificados antes de aumentar a contagem de movimentos. Como resultado, o BFS é garantidamente completo e ótimo em grafos não ponderados, ou seja, sempre encontra o caminho mais curto se houver um.

2. Busca em Profundidade (Depth-First Search – DFS) - A Busca em Profundidade segue um caminho até que não seja mais possível prosseguir,

retornando então ao último ponto de decisão para explorar alternativas. Apesar de simples de implementar, a DFS pode ficar presa em caminhos muito longos ou até mesmo entrar em ciclos, caso não haja controle de estados visitados. Além disso, ela não garante a obtenção do menor caminho até o objetivo.

3. Busca Gulosa - A Busca Gulosa utiliza uma heurística para decidir qual caminho seguir, sempre priorizando o nó que parece mais próximo do objetivo. Neste trabalho, foi utilizada a distância de Manhattan como função heurística, que calcula a soma das distâncias horizontais e verticais entre dois pontos. A principal vantagem da busca gulosa está em sua velocidade, embora ela não assegure a solução ótima, podendo tomar decisões precipitadas ao ignorar o custo acumulado.

4. Algoritmo A* (A Estrela) - O algoritmo A* combina a eficiência da busca heurística com a garantia de encontrar soluções ótimas ao considerar tanto o custo já percorrido (g(n)) quanto a estimativa do custo restante até o objetivo (h(n)). Assim, a função de avaliação do A* é dada por f(n) = g(n) + h(n). Com a heurística de Manhattan, o A* é geralmente eficiente e encontra soluções ótimas, sendo amplamente utilizado em sistemas reais de navegação e jogos.

3 Implementação

A implementação dos algoritmos foi realizada na linguagem Python, escolhida pela sua clareza sintática, ampla comunidade e suporte a bibliotecas úteis para manipulação de estruturas de dados e medição de desempenho. O labirinto foi representado por uma matriz 10x10, onde cada célula contém um valor inteiro:

- 0 indica uma posição livre;
- 1 representa um obstáculo.

O ponto de partida foi fixado na coordenada (0, 0) e o objetivo na coordenada (8, 9), seguindo a configuração fornecida por uma imagem de referência. Todos os algoritmos operam sobre essa matriz, utilizando uma abordagem baseada em listas para filas, pilhas e estruturas auxiliares como conjuntos de visitados e dicionários de predecessores para reconstrução do caminho final.

1. Ambiente de Execução: Todos os testes foram realizados em um mesmo

ambiente computacional para garantir uniformidade nos resultados. As especificações da máquina utilizada foram:

Sistema Operacional: win10

Processador: AMD Ryzen 7 5700g

Memória RAM: 16 GB

Interpretador Python: versão 3.10

4 Comparação de resultados dos algoritimos de busca

1. Busca em Largura (BFS)
   - Nós visitados: 21
   - Tempo: 0.000081 segundos

   Ponto inicial: (0, 0)
   Ponto final: (8, 9)
   Estado do labirinto no início:
   0, 1, 1, 1, 1, 1, 1, 1, 1, 1
   0, 0, 1, 0, 0, 0, 1, 0, 0, 1
   1, 0, 1, 1, 1, 0, 1, 0, 1, 1
   1, 0, 0, 0, 1, 0, 0, 0, 1, 1
   1, 1, 1, 0, 1, 1, 1, 0, 0, 1
   1, 0, 0, 0, 0, 1, 0, 0, 1, 1
   1, 0, 1, 1, 0, 1, 1, 0, 1, 1
   1, 0, 1, 0, 0, 0, 1, 0, 0, 1
   1, 0, 1, 0, 1, 0, 1, 1, 0, 0
   1, 1, 1, 1, 1, 1, 1, 1, 1, 1

2. Busca em Profundidade (DFS)
   - Nós visitados: 21
   - Tempo: 0.001878 segundos
   - Caminho encontrado:

Estado do labirinto no final:
0, 1, 1, 1, 1, 1, 1, 1, 1, 1
0, 0, 1, 0, 0, 0, 1, 0, 0, 1
1, 0, 1, 1, 1, 0, 1, 0, 1, 1
1, 0, 0, 0, 1, 0, 0, 0, 1, 1
1, 1, 1, 0, 1, 1, 1, 0, 0, 1
1, 0, 0, 0, 0, 1, 0, 0, 1, 1
1, 0, 1, 1, 0, 1, 1, 0, 1, 1
1, 0, 1, 0, 0, 0, 1, 0, 0, 1
1, 0, 1, 0, 1, 0, 1, 1, 0, 0
1, 1, 1, 1, 1, 1, 1, 1, 1, 1

3. Busca Gulosa:
   - Nós visitados: 21
   - Tempo: 0.001816 segundos
   - Caminho encontrado:

   Estado do labirinto no final:
   0, 1, 1, 1, 1, 1, 1, 1, 1, 1
   0, 0, 1, 0, 0, 0, 1, 0, 0, 1
   1, 0, 1, 1, 1, 0, 1, 0, 1, 1
   1, 0, 0, 0, 1, 0, 0, 0, 1, 1
   1, 1, 1, 0, 1, 1, 1, 0, 0, 1
   1, 0, 0, 0, 0, 1, 0, 0, 1, 1
   1, 0, 1, 1, 0, 1, 1, 0, 1, 1
   1, 0, 1, 0, 0, 0, 1, 0, 0, 1
   1, 0, 1, 0, 1, 0, 1, 1, 0, 0
   1, 1, 1, 1, 1, 1, 1, 1, 1, 1

4. Algoritmo A*:
   - Nós visitados: 21
   - Tempo: 0.001749 segundos
   - Caminho encontrado:

   Estado do labirinto no final:
   0, 1, 1, 1, 1, 1, 1, 1, 1, 1

0, 0, 1, 0, 0, 0, 1, 0, 0, 1
1, 0, 1, 1, 1, 0, 1, 0, 1, 1
1, 0, 0, 0, 1, 0, 0, 0, 1, 1
1, 1, 1, 0, 1, 1, 1, 0, 0, 1
1, 0, 0, 0, 0, 1, 0, 0, 1, 1
1, 0, 1, 1, 0, 1, 1, 0, 1, 1
1, 0, 1, 0, 0, 0, 1, 0, 0, 1
1, 0, 1, 0, 1, 0, 1, 1, 0, 0
1, 1, 1, 1, 1, 1, 1, 1, 1, 1

5 Conclusão

Os experimentos realizados demonstram que todos os algoritmos avaliados — BFS, DFS, Busca Gulosa e A* — foram capazes de encontrar uma solução viável para o problema do labirinto proposto. No entanto, ao comparar seus desempenhos em termos de tempo, esforço computacional e confiabilidade na solução, é possível destacar algumas observações relevantes.

A Busca em Largura (BFS) foi a mais eficiente em termos de tempo de execução, apresentando o menor tempo entre todas as estratégias testadas. Além disso, por explorar os caminhos em ordem crescente de profundidade e garantir a menor quantidade de passos até o destino, também apresentou qualidade ótima de solução, sendo adequada para labirintos com peso uniforme.

A Busca em Profundidade (DFS), apesar de encontrar uma solução válida, apresentou um tempo de execução consideravelmente maior. Isso ocorre devido à sua natureza de exploração profunda, que pode levar a caminhos menos eficientes e a uma maior quantidade de chamadas recursivas. Ainda assim, o número de nós visitados foi igual ao dos demais algoritmos, o que indica que, neste caso específico, a DFS não caiu em ciclos ou caminhos muito longos.

A Busca Gulosa demonstrou ser rápida, porém seu desempenho em outros cenários mais complexos pode ser inconsistente. Isso ocorre porque ela considera apenas a estimativa heurística e ignora o custo acumulado, o que pode levá-la a escolher caminhos que parecem promissores, mas se revelam mais longos.

O algoritmo A* combinou o melhor dos dois mundos: apresentou desempenho muito próximo ao da busca gulosa em termos de tempo, mas com a

vantagem de garantir a solução ótima ao levar em conta tanto a heurística quanto o custo do caminho percorrido. Isso o torna o algoritmo mais robusto e confiável entre os testados, especialmente para labirintos mais complexos ou ambientes dinâmicos.