Inteligência Artificial

Atividade Prática 2

Autor: Júlio César Gonzaga Ferreira Silva

Introdução

O objetivo deste trabalho foi desenvolver um jogo de labirinto simples, onde métodos clássicos de busca são aplicados para análise de métricas e estatísticas de desempenho.

Além do jogo em si, foi realizada a implementação e comparação de três algoritmos de busca: Busca em Profundidade, Busca em Largura e A* com duas heurísticas distintas. A análise inclui dados como nós visitados, tempo de execução e uso de memória.

Descrição do Jogo

O labirinto é representado como uma matriz, onde:

- **0** representa caminhos livres.
- **1** representa paredes.
- 🏳 é o ponto de partida.
- 🏁 é o ponto de chegada.

O jogador, representado por ☆, pode se mover utilizando as teclas **W** (cima), **S** (baixo), **A** (esquerda) e **D** (direita). A cada movimento, o terminal é atualizado para simular a movimentação no labirinto.

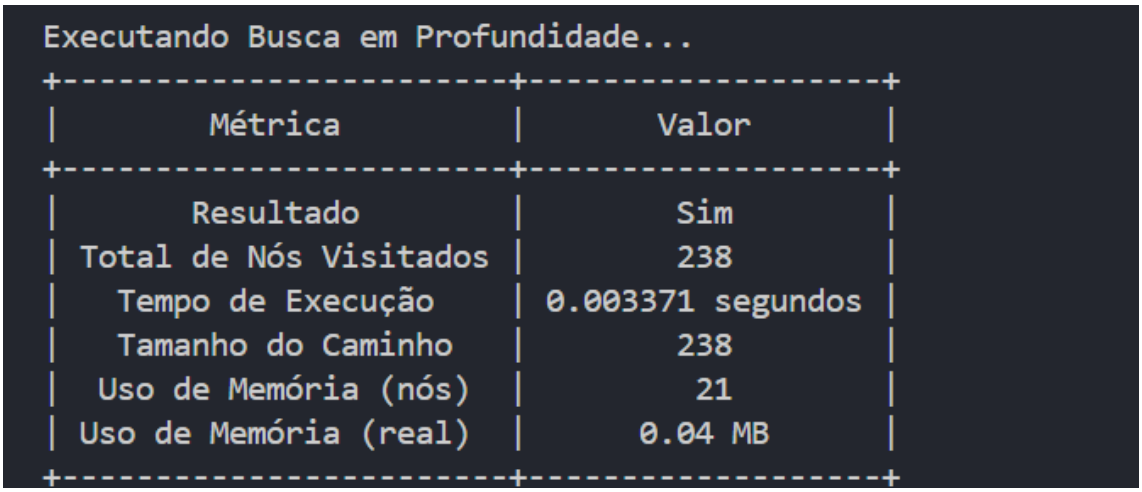
2. Algoritmos de Busca

Foram implementados três algoritmos de busca para encontrar o caminho do ponto de partida ao objetivo:

2.1. Busca em Profundidade (DFS)

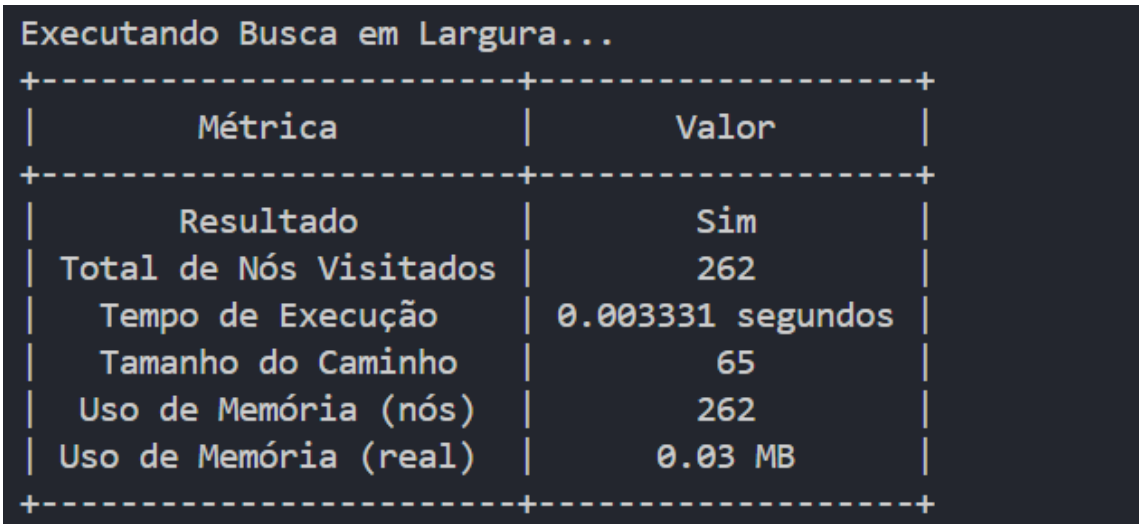
- **Descrição**: Explora o labirinto priorizando a profundidade antes de recuar. Usa uma pilha para gerenciar os estados.
- **Vantagens**: Eficiência em termos de memória em certos cenários.
- **Limitações**: Não garante o menor caminho e pode explorar caminhos mais longos antes de encontrar o objetivo.
- **Métricas**: Usei algumas ideias de métricas para analise como se encontrou um caminho pro objetivo ou não, tempo de execução, Nós visitados, Tamanho do

Caminho, Tempo de Execução, Uso de Memória (nós) e Uso de Memória (real).



2.2. Busca em Largura (BFS)

- **Descrição:** Explora o labirinto em níveis, garantindo o menor caminho caso exista.
- **Vantagens:** Sempre encontra o caminho mais curto.
- **Limitações:** Alto uso de memória, especialmente em labirintos grandes.
- **Métricas:** Usei algumas ideias de métricas para analise como se encontrou um caminho para o objetivo ou não, tempo de execução, Nós visitados, Tamanho do Caminho, Tempo de Execução, Uso de Memória (nós) e Uso de Memória (real).



2.3. Algoritmo A*

- **Descrição**: Combina características do DFS e BFS, utilizando heurísticas para guiar a busca.
- **Heurísticas Utilizadas**:

1. **Distância Manhattan**: Calcula a soma das diferenças absolutas das coordenadas.
2. **Distância Euclidiana**: Calcula a distância reta entre os pontos.

- **Vantagens**: Alta eficiência, especialmente com heurísticas bem definidas.
- **Métricas**: Usei algumas ideias de métricas para analise como se encontrou um caminho para o objetivo ou não, tempo de execução, Nós visitados, Tamanho do Caminho, Tempo de Execução, Uso de Memória (nós) e Uso de Memória (real).

Métrica	Heurística Manhattan	Heurística Euclidiana
Resultado	Sim	Sim
Total de Nós Visitados	213	216
Tempo de Execução	0.001408 segundos	0.002958 segundos
Tamanho do Caminho	65	65
Uso de Memória (nós)	13	12
Uso de Memória (real)	0.01 MB	0.01 MB

3. Análise de Métricas

Após implementar os algoritmos, realizamos uma análise comparativa considerando a mesma configuração de labirinto. As métricas de cada algoritmo foram apresentadas em tabelas e comparadas lado a lado.

Largura

Valor
Sim
262
0.003331 segundos
65
262
0.03 MB

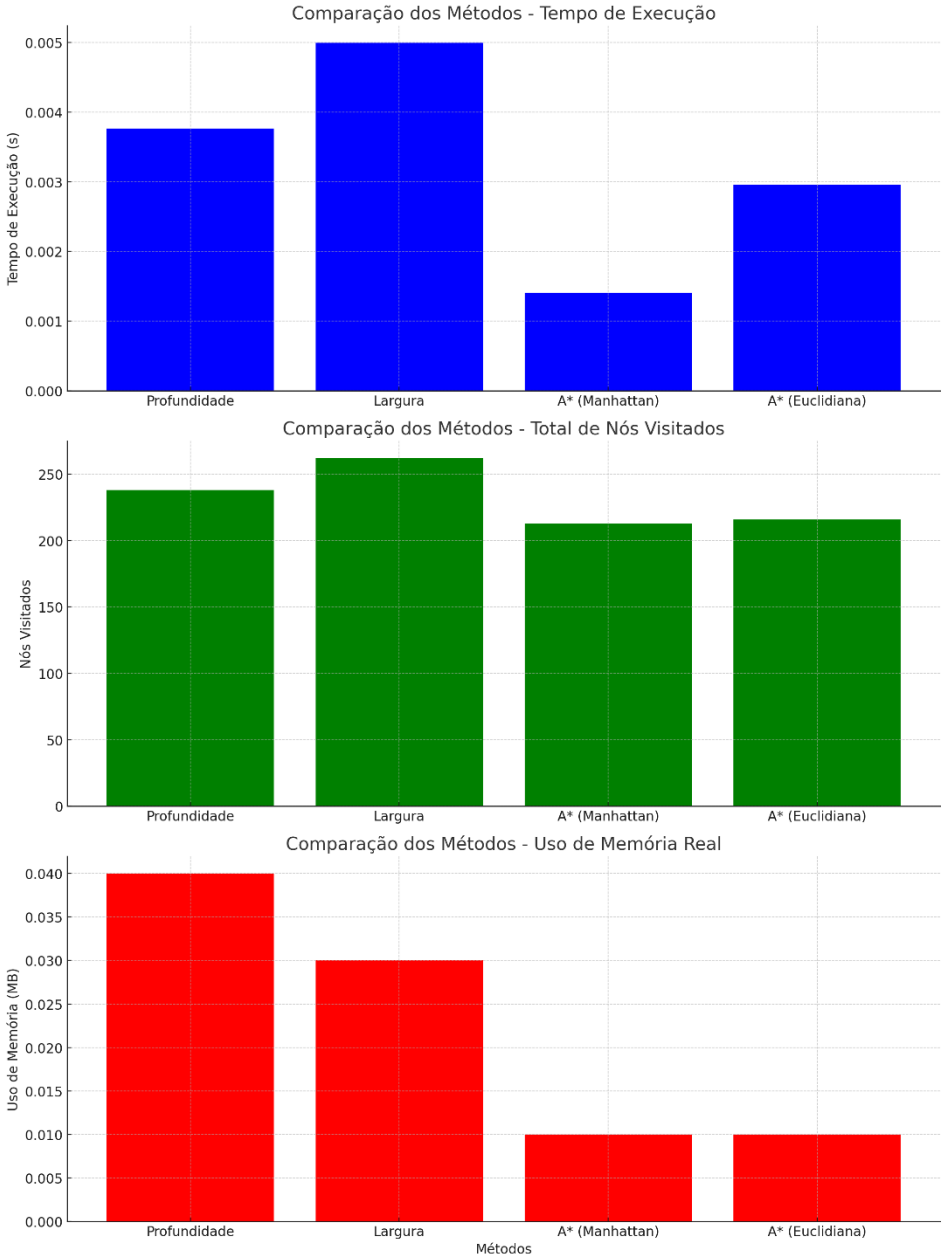
Profundidade

Valor
Sim
238
0.003371 segundos
238
21
0.04 MB

Manhattan Euclidiana

Heurística Manhattan	Heurística Euclidiana
Sim	Sim
213	216
0.001408 segundos	0.002958 segundos
65	65
13	12
0.01 MB	0.01 MB

Comparacao dos Métodos



Dentre os métodos de busca implementados, todos foram capazes de encontrar um caminho válido para o objetivo final no labirinto. Entretanto, as diferenças nas métricas de desempenho revelam características distintas de cada método. A busca A* foi a mais eficiente em termos de tempo de execução, especialmente quando empregada com a heurística Manhattan. Essa heurística, por medir a soma das diferenças absolutas das coordenadas entre o nó atual e o objetivo, proporciona uma estimativa direta e menos custosa computacionalmente. A heurística Euclidiana apresentou desempenho semelhante em termos de tempo e custo, mas é ligeiramente mais cara devido ao cálculo de raízes quadradas, embora isso não tenha causado impacto significativo no tempo de execução neste caso. A busca A* demonstrou ser uma das menos custosas em termos de número de nós explorados, graças à sua capacidade de priorizar caminhos promissores, minimizando a exploração desnecessária.

A busca em largura explorou o maior número de nós entre todos os métodos, refletindo sua característica de expansão uniforme, onde todos os caminhos em um nível são explorados antes de passar para o próximo. Curiosamente, nesta configuração do labirinto, a busca em largura consumiu menos memória real (bytes) do que a busca em profundidade. Isso pode ser atribuído à eficiência da fila usada para armazenar os nós a serem explorados e ao descarte contínuo de nós já processados. Apesar disso, a busca em largura demonstrou um custo mais elevado em termos de nós armazenados, dado que mantém todos os nós em um nível na memória enquanto explora o próximo.

A busca em profundidade apresentou um comportamento interessante: apesar de explorar menos nós do que a busca em largura, ela consumiu mais memória real. Isso pode ser explicado pela estrutura de pilha utilizada, que armazena todos os estados ativos até que um caminho válido seja encontrado ou a pilha esvazie. Em termos de custo de "nós ativos" (nós mantidos na pilha durante a execução), a busca em profundidade foi mais econômica, armazenando apenas 21 elementos no caso analisado. Isso reflete a natureza de seu algoritmo, que segue um único caminho até sua conclusão antes de retroceder. Entretanto, no caso específico deste labirinto, a busca em profundidade encontrou um caminho válido, mas não o mais curto. Isso ocorre devido à sua abordagem de priorizar profundidade sobre a otimização do caminho.

3.1 Comparação Geral

A busca A*, com a heurística Manhattan, foi a mais eficiente em termos de tempo de execução e custo total de nós explorados, mostrando-se ideal para situações em que a otimização do caminho é essencial.

A busca em largura, apesar de explorar o maior número de nós, demonstrou um consumo eficiente de memória em termos reais (bytes), sendo adequada para labirintos mais simples onde o custo de expansão é menos crítico.

A busca em profundidade, embora menos eficiente em encontrar o caminho mais curto, destacou-se pelo baixo custo de nós ativos na pilha, sendo útil para casos

onde a memória disponível é limitada, mas o tempo de execução não é um fator crítico.

4. Conclusão

Com base nos resultados das implementações e análises realizadas, foi possível compreender as vantagens, limitações e características de cada método de busca aplicado ao problema do labirinto.

O estudo demonstrou que, embora todos os métodos possam ser utilizados para encontrar caminhos em um labirinto, suas características e resultados variam significativamente. A busca A* provou ser a solução mais robusta, especialmente com o uso de heurísticas bem definidas, sendo indicada para aplicações que requerem otimização de caminhos. A busca em largura garantiu o caminho mais curto, enquanto a busca em profundidade foi mais econômica no uso de memória ativa. A escolha do algoritmo mais adequado deve ser guiada pelos requisitos e limitações do problema em questão.

5. Referência para o Executável do Jogo e Analise

https://github.com/JcKser/Inteligencia-Artificial/blob/2675841b5867f37b7cd289a2cc66bc99971f31ed/Atividade%20Pr%C3%A1tica%202/Atividade_pratica2.py