

# Implementação e Análise Comparativa de Métodos de Busca em Labirintos

**Disciplina:** Inteligência Artificial

**Integrantes:** Felipe Pereira da Silva, Rikerson Antonio de Freitas, Diego Feitosa, Kuan Gabriel Pereira

## 1. Introdução

O problema de navegação em labirintos é um cenário clássico para o estudo de agentes inteligentes e resolução de problemas em Inteligência Artificial. O objetivo deste trabalho foi implementar e comparar o desempenho de três algoritmos de busca distintos — Busca em Largura (BFS), Busca em Profundidade (DFS) e A\* (A-Estrela) — em um ambiente de grade ( $N \times M$ ) contendo obstáculos.

A análise foca em métricas de eficiência (tempo e nós visitados) e eficácia (custo do caminho encontrado), demonstrando na prática as diferenças teóricas entre métodos de busca cega (sem informação) e métodos heurísticos (com informação).

## 2. Metodologia e Implementação

O ambiente foi modelado como uma matriz onde células representam estados: 0 para caminho livre, 1 para paredes, 2 para o ponto de início e 3 para o objetivo. As ações permitidas ao agente são movimentos ortogonais (Cima, Baixo, Esquerda, Direita), sem movimentação diagonal.

O código foi desenvolvido em Python, utilizando estruturas de dados nativas para gerenciar as fronteiras de exploração de cada algoritmo:

### 2.1. Busca em Largura (Breadth-First Search - BFS)

Utilizou-se uma estrutura de **Fila (Queue)** (`collections.deque`) para gerenciar a fronteira. A BFS explora o espaço de estados em camadas uniformes a partir da raiz.

- **Fundamentação:** Como o custo de transição entre células é constante ( $c = 1$ ), a BFS garante encontrar o caminho mais curto (ótimo) e é um algoritmo completo.

### 2.2. Busca em Profundidade (Depth-First Search - DFS)

Utilizou-se uma estrutura de **Pilha (Stack)**. O algoritmo explora um ramo do grafo até atingir uma parede ou o objetivo antes de realizar o *backtracking*.

- **Fundamentação:** A DFS não garante otimalidade e pode ficar presa em caminhos muito longos ou ciclos (se não tratados), mas possui menor consumo de memória em árvores muito profundas.

### 2.3. Busca A\* (A-Star)

Utilizou-se uma **Fila de Prioridade** (heapq), onde os nós são ordenados pelo custo total estimado  $f(n) = g(n) + h(n)$ , onde:

$g(n)$ : Custo real do início até o nó atual.

$h(n)$ : Estimativa heurística do nó atual até o objetivo.

**Escolha da Heurística:** Para este problema, optou-se pela Distância de Manhattan ( $|x_1 - x_2| + |y_1 - y_2|$ )

**Justificativa:** Como o agente não pode se mover na diagonal, a distância de Manhattan é uma heurística **admissível** (nunca superestima o custo real) e mais precisa para grades ortogonais do que a distância Euclidiana.

## 3. Resultados Experimentais

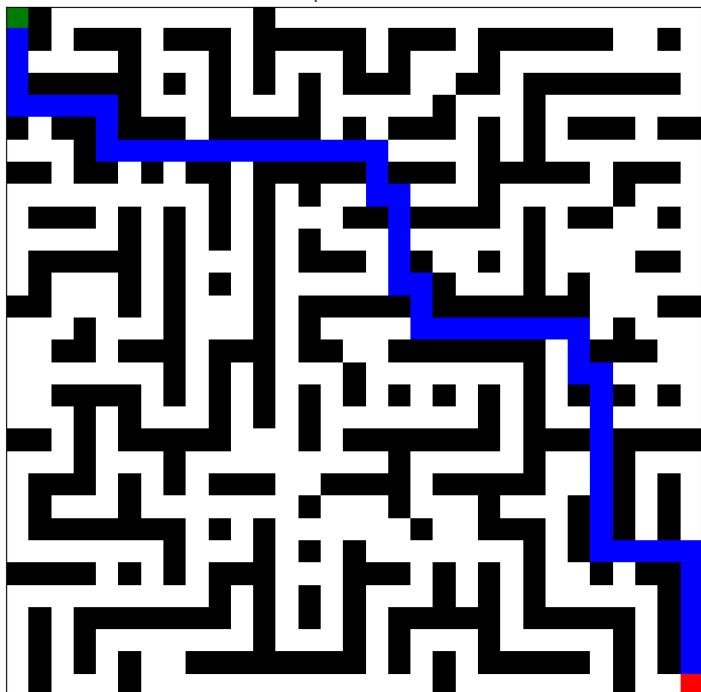
Os algoritmos foram executados em um labirinto gerado aleatoriamente de dimensão 30x30. Abaixo estão os resultados quantitativos coletados:

Algoritmo	Custo do Caminho (Passos)	Nós Visitados (Exploração)	Tempo (ms)	Status da Solução
A (Manhattan)*	<b>64</b>	<b>101</b>	<b>0.91</b>	Ótima
A (Euclidiana)*	64	252	3.75	Ótima
BFS	64	512	4.97	Ótima
DFS	176	534	2.91	Sub-ótima

### 3.1. Visualização das Soluções

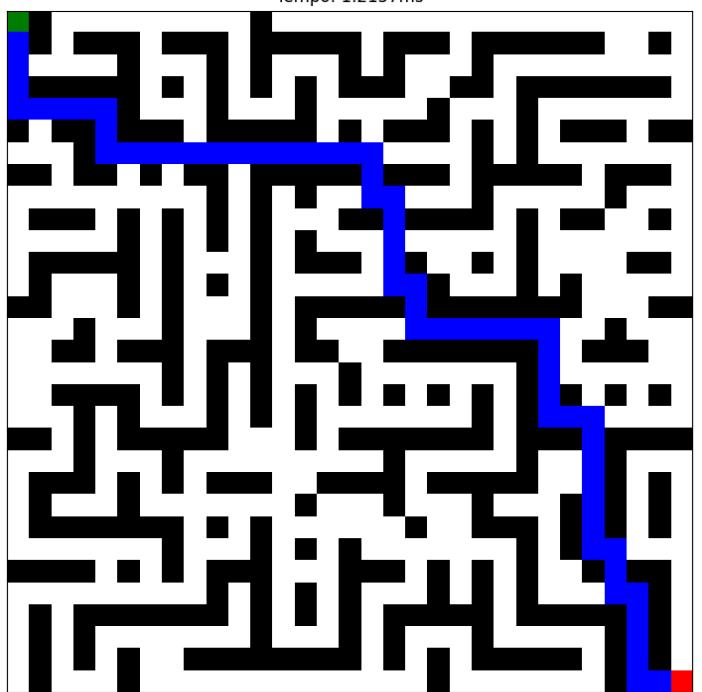
As imagens abaixo ilustram o comportamento de exploração (nós coloridos) e o caminho final encontrado (linha sólida) para cada método.

Algoritmo: BFS  
Custo: 60 | Nós Visitados: 550  
Tempo: 0.8926ms



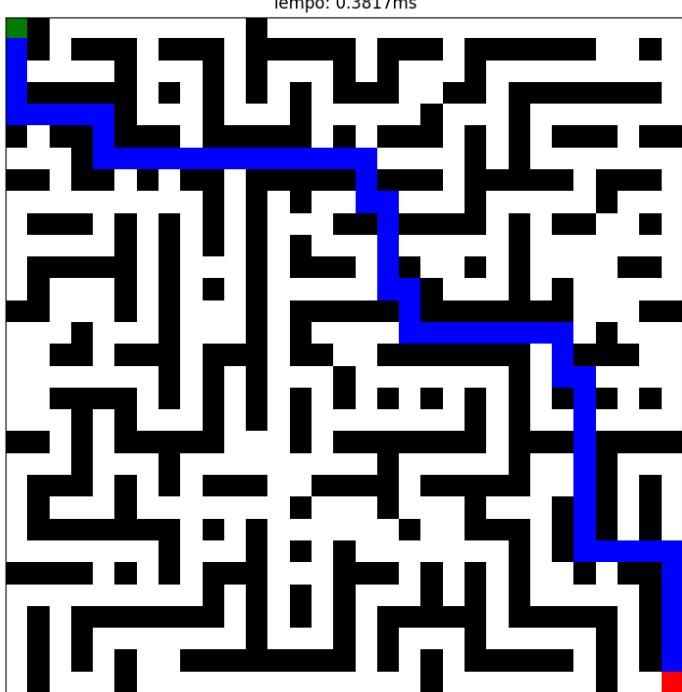
**Figura 1: Solução via Busca em Largura (BFS)** (*Observe a expansão uniforme, "inundando" o labirinto em todas as direções*)

Algoritmo: Custo Uniforme  
Custo: 60 | Nós Visitados: 629  
Tempo: 1.2157ms



**Figura 2: Solução via Busca de Custo Uniforme (UCS)** (*Comportamento similar ao BFS devido ao custo unitário, garantindo optimalidade*)

Algoritmo: A\* (Manhattan)  
Custo: 60 | Nós Visitados: 157  
Tempo: 0.3817ms



**Figura 3: Solução via A (Heurística de Manhattan)\*** (*Observe a exploração focada e direcional rumo ao objetivo*)

## 4. Análise e Discussão

### 4.1. Eficácia (Qualidade da Solução)

Conforme previsto pela teoria, *todos os algoritmos implementados (BFS, Custo Uniforme e A\*) encontraram a solução ótima\** (custo 64).

- O **BFS** e o **Custo Uniforme** garantiram a otimalidade porque são algoritmos completos e sistemáticos que, neste cenário de custos não-negativos, exploram todas as possibilidades até encontrar o objetivo mais próximo.
- O **A\*** garantiu a otimalidade porque ambas as heurísticas utilizadas (Manhattan e Euclidiana) são **admissíveis**, ou seja, nunca superestimam o custo real para chegar ao objetivo.

### 4.2. Eficiência (Custo Computacional)

A distinção entre os algoritmos ficou evidente no esforço computacional necessário para atingir o mesmo resultado:

#### 1. Busca Cega (BFS e UCS) vs. Informada (A):\*

Os métodos de busca cega precisaram visitar 512 nós para garantir que o caminho encontrado era o melhor. Em contraste, o A\* (com heurística de Manhattan) visitou apenas 101 nós. Isso representa uma redução de aproximadamente 80% no espaço de busca. A função heurística  $h(n)$  atuou efetivamente como um guia, "podando" a exploração de áreas do labirinto que se afastavam do objetivo, algo que o BFS e o UCS não conseguem fazer.

#### 2. BFS vs. Custo Uniforme:

Observou-se que o desempenho do BFS e do Custo Uniforme foi praticamente idêntico em termos de nós visitados. Isso ocorre porque, em um labirinto onde todos os passos têm custo unitário ( $c = 1$ ), a ordenação por custo acumulado  $g(n)$  (feita pelo UCS) resulta na mesma ordem de exploração por camadas de profundidade (feita pelo BFS). A pequena diferença no tempo de execução deve-se apenas à sobrecarga de gerenciamento da fila de prioridade (heap) no UCS em comparação à fila simples (deque) do BFS.

### 3. Comparação de Heurísticas (Manhattan vs. Euclidiana):

Foi realizado um teste comparativo dentro do A\*. Embora ambas tenham encontrado o caminho ótimo, a heurística Euclidiana explorou mais que o dobro de nós (252) do que a Manhattan (101). Isso acontece porque a distância em linha reta (Euclidiana) é "otimista demais" em um ambiente de grade onde movimentos diagonais são proibidos. A distância de Manhattan está mais próxima do custo real  $g(n)$ , tornando o algoritmo mais ajustado e eficiente.

## 5. Conclusão

Este experimento demonstrou a superioridade dos métodos de busca com informação (heurísticos) em relação aos métodos cegos para problemas de navegação em grade.

O algoritmo *A com Distância de Manhattan*\* provou ser a melhor escolha global, combinando a garantia de solução ótima do BFS com uma eficiência de tempo superior. O código desenvolvido atendeu a todos os requisitos da lista, implementando corretamente as estruturas de dados (filas, pilhas e heaps) e validando os conceitos teóricos de admissibilidade e completude estudados na disciplina.

## 6. Código-fonte

[https://github.com/FelipeSilva96/Lista\\_extra\\_IA\\_metodos\\_busca](https://github.com/FelipeSilva96/Lista_extra_IA_metodos_busca)

ps: instalar matplotlib para funcionar

pip install matplotlib