



CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS V - UNIDADE DIVINÓPOLIS - Engenharia da Computação

Inteligência Artificial
Alisson Marques

GABRIEL OLIVEIRA ALVES
PEDRO PINHEIRO DE SIQUEIRA

Algoritmos de Busca

DIVINÓPOLIS - MG
2024

Sumário

| | | |
|----------|--|-----------|
| 1 | Introdução | 4 |
| 2 | Descrição dos Algoritmos Implementados | 5 |
| 2.1 | Breadth-First Search - BFS | 5 |
| 2.2 | Depth-First Search - DFS | 5 |
| 2.3 | Greedy Search | 6 |
| 2.4 | A* | 7 |
| 2.5 | Observação | 7 |
| 2.5.1 | Heurística nos Algoritmos Informados | 7 |
| 3 | Implementação | 9 |
| 4 | Resultados das Medições de Desempenho | 10 |
| 4.1 | Análise Comparativa dos Algoritmos | 11 |
| 4.1.1 | Dados Normalizados | 11 |
| 4.1.2 | Desempenho em Tempo | 11 |
| 4.1.3 | Uso de Memória | 11 |
| 4.1.4 | Compleitude | 12 |
| 4.1.5 | Optimalidade | 13 |
| 5 | Conclusões | 14 |

| | | |
|----------|-------------------------------------|-----------|
| 5.1 | Possíveis Melhorias | 14 |
| 6 | Anexo A | 16 |
| 6.1 | Pseudocódigo Busca Gulosa | 16 |
| 7 | Anexo B | 17 |
| 7.1 | Imagens de Teste | 17 |

1 Introdução

Neste trabalho, foram implementados e comparados quatro algoritmos de busca aplicados ao clássico problema do labirinto em uma grade 10x10, neste caso, com 30% das posições sendo obstáculos. Os algoritmos escolhidos foram dois de busca não informada, Breadth-First Search (BFS) e Depth-First Search (DFS), e dois de busca informada, Greedy e A*. A implementação foi realizada em Python 3.11, utilizando a biblioteca **NetworkX** [3] para modelagem e manipulação de grafos, representando o labirinto como um grafo onde os nós correspondem às posições, se são obstáculos ou não e as arestas correspondem aos possíveis movimentos.

Foram realizadas medições de tempo utilizando a biblioteca **time** e de uso de memória com a biblioteca **tracemalloc** para avaliar o desempenho. Além disso, foram consideradas as métricas de completude e optimalidade para uma análise comparativa mais abrangente. A análise dos resultados visa discutir as diferenças de desempenho de acordo com as medições realizadas, oferecendo insights sobre as condições nas quais cada algoritmo se destaca.

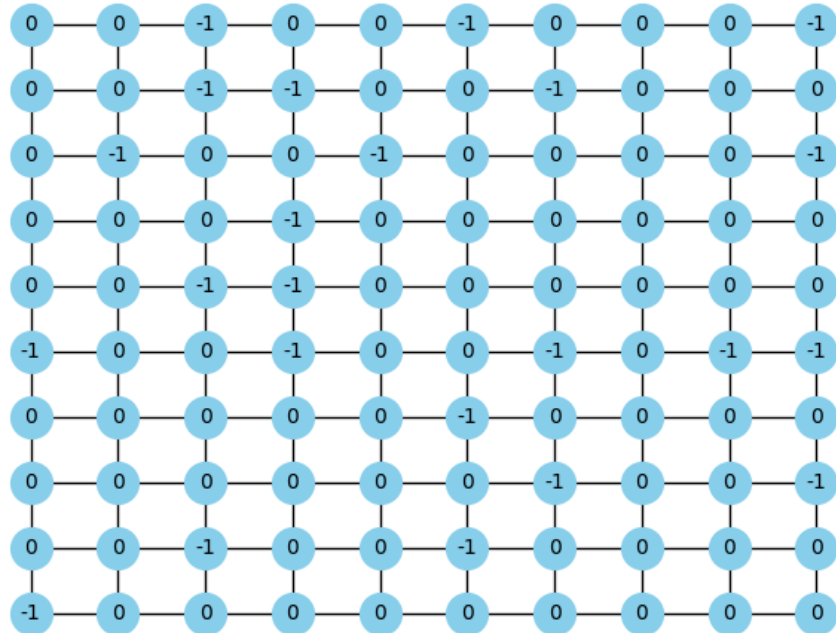


Figura 1: Labirinto 10x10 representado por um grafo (-1 = obstáculo)

2 Descrição dos Algoritmos Implementados

Para descrever os algoritmos escolhidos, fez-se necessário além de pesquisas na internet, o uso do livro Algoritmos - Teoria e Prática [2] para melhor embasamento geral.

2.1 Breadth-First Search - BFS

A busca em largura (BFS) explora o espaço de busca nível por nível, iniciando pelo nó inicial e expandindo todos os nós de um nível antes de passar para o próximo. Ela garante encontrar a solução mais curta em termos de número de passos para problemas onde todos os passos têm o **mesmo custo**.

BFS é completo, desde que o espaço de busca seja finito e as ações conduzam a novos estados, pois explora todos os nós de um nível antes de seguir para o próximo, isso garante que, se uma solução existir, será encontrada.

Em termos de optimalidade, BFS é ótimo quando o custo de cada passo é igual, pois sempre encontra a solução com o menor número de passos. Entretanto, se os custos das arestas variam, BFS não garante a solução de menor custo.

O custo de tempo do BFS é $O(V + E)$, onde V é o número de vértices e E é o número de arestas do grafo. O custo de espaço também é $O(V)$, pois todos os nós em um nível precisam ser armazenados na memória.

Vantagens do BFS incluem sua completude, optimalidade - para arestas de custo uniforme - e sua simplicidade. Contudo, suas desvantagens incluem o **alto consumo de memória**, especialmente em grafos grandes, e a falta de optimalidade quando os custos das arestas variam.

2.2 Depth-First Search - DFS

A busca em profundidade DFS não é completa em espaços de busca infinitos ou em grafos com ciclos, pois pode entrar em um loop infinito. No entanto, se o espaço de busca for finito e os ciclos forem evitados, DFS é completo, garantindo que uma solução será encontrada se existir.

Em termos de optimalidade, DFS não é ótimo. Ela não garante encontrar a solução de menor custo ou o caminho mais curto, pois pode seguir profundamente por um caminho que leva a uma solução mais

longa antes de explorar outras opções que poderiam ser mais curtas ou menos custosas.

O custo de tempo do DFS é $O(V + E)$, onde V é o número de vértices e E é o número de arestas do grafo, semelhante a outros algoritmos de busca em grafos. O custo de espaço é $O(V)$ em um cenário de busca completa, devido ao armazenamento da pilha de recursão, que geralmente consome menos memória que o BFS.

Vantagens do DFS incluem **menor consumo de memória** comparado ao BFS e sua eficiência em encontrar soluções profundas em grafos grandes. Contudo, suas desvantagens incluem a falta de completude em espaços de busca infinitos, a ausência de optimalidade, e a possibilidade de ficar preso em ciclos, a menos que uma verificação de nós visitados seja implementada.

2.3 Greedy Search

A busca gulosa (Greedy Search) explora o espaço de busca priorizando os nós que parecem estar mais próximos da solução, com base em uma função heurística. Ela seleciona o nó que parece oferecer o caminho mais promissor em direção à meta, sem considerar o custo total do caminho.

Greedy Search não é completa, pois pode ficar presa em mínimos locais ou em caminhos que parecem promissores inicialmente, mas não levam à solução. Isso significa que, mesmo que uma solução exista, a busca gulosa pode não encontrá-la, especialmente em espaços de busca complexos.

Em termos de optimalidade, Greedy Search não é ótima. Ela não garante encontrar a solução mais curta ou de menor custo, pois toma decisões baseadas na heurística local sem considerar o custo acumulado. Isso pode levar a soluções subótimas.

O custo de tempo do Greedy Search é $O(V + E)$, onde V é o número de vértices e E é o número de arestas do grafo, similar a outras buscas em grafos. O custo de espaço pode variar dependendo da implementação, mas geralmente é menor que o BFS, pois não precisa armazenar todos os nós de um nível.

Vantagens do Greedy Search incluem sua rapidez e eficiência em muitos problemas práticos, especialmente quando uma boa heurística está disponível. No entanto, suas desvantagens incluem a falta de completude e optimalidade, além da possibilidade de ficar presa em soluções subótimas.

2.4 A*

A busca A* (A-Star) explora o espaço de busca utilizando uma função de custo que combina o custo acumulado do caminho até o nó atual com uma estimativa heurística do custo restante até a solução. Ela seleciona o nó que minimiza essa função de custo, equilibrando o custo total do caminho e a estimativa heurística.

A* é completo, desde que o espaço de busca seja finito e a heurística seja admissível (isto é, nunca superestime o custo real para alcançar a solução). A busca A* garante que, se uma solução existir, ela será encontrada, pois explora todos os caminhos possíveis de maneira sistemática até identificar a solução ótima.

Em termos de optimalidade, A* é ótimo quando a heurística é admissível e consistente. Isso significa que ele sempre encontra a solução de menor custo, equilibrando eficientemente a exploração do espaço de busca e a estimativa do custo restante, evitando caminhos subótimos.

O custo de tempo do A* é $O(V + E)$, onde V é o número de vértices e E é o número de arestas do grafo, semelhante a outros algoritmos de busca em grafos. No entanto, o custo de espaço pode ser significativo, pois A* precisa manter todos os nós na memória até encontrar a solução ótima, o que pode exigir grande capacidade de armazenamento em grafos extensos.

Vantagens do A* incluem sua completude e optimalidade, tornando-o um dos algoritmos de busca mais eficazes para uma ampla variedade de problemas. Contudo, suas desvantagens incluem o alto consumo de memória, especialmente em problemas com grandes espaços de busca, e a dependência da qualidade da heurística para manter a eficiência.

2.5 Observação

2.5.1 Heurística nos Algoritmos Informados

Para realizar a medição de desempenho demonstrada na Seção (4) no problema do labirinto 10x10 para os algoritmos da **Busca Gulosa** e do A*, foi utilizado a *Distância de Manhattan* como forma de heurística.

$$D_{\text{Manhattan}} = |x_2 - x_1| + |y_2 - y_1| \quad (1)$$

Sendo o **ponto 2**, a posição a qual se quer alcançar - saída do labirinto. E **ponto 1**, a posição de início do labirinto.

3 Implementação

Foi necessário, inicialmente, criar um script para gerar 50 labirintos com obstáculos aleatórios, cobrindo 30% das posições de cada labirinto. Tal porcentagem foi escolhida porque, em valores maiores, haveria uma maior incidência de labirintos impossíveis de solucionar.

Em seguida, utilizou-se as classes `Node`, que representa os nós e suas propriedades; `Maze`, que lê os labirintos gerados e os transforma em uma matriz de nós, atualizando suas propriedades; e `Generate`, que cria o grafo e fornece funções para ler, construir e imprimir cada labirinto, além de testar e medir. O arquivo `main` é usado para implementar as funções da classe `Generate`.

Para os algoritmos **BFS**, **DFS** e **A***, foram utilizados os métodos já implementados e disponíveis na biblioteca **NetworkX**. Essa escolha foi feita porque a biblioteca oferece implementações otimizadas e testadas pela comunidade Python, garantindo maior eficiência e confiabilidade.

A **Busca Gulosa (Greedy Search)** foi implementada manualmente, pois não há uma implementação disponível na biblioteca **NetworkX**. Além disso, fez-se necessário utilizar a biblioteca **heapq** para trabalhar com uma fila de prioridade. O pseudocódigo pode ser conferido na secção 6 anexo B.

O repositório com o código fonte está armazenado no Github - **Comparative Search Algorithms [1]**

4 Resultados das Medições de Desempenho

Os testes foram feitos em duas máquinas diferentes, utilizando 50 labirintos. Abaixo estão as tabelas dos resultados médios de tempo de execução e uso de memória para cada algoritmo. Os 50 labirintos foram testados 30 vezes em cada algoritmo e seus resultados médios podem ser vistos na Tabela 1. Na Tabela 2 é possível ver o desvio padrão das medidas. Todos os resultados podem ser conferidos na seção 7 - Anexo A.

- Máquina 1 - i7-9750H, 20 GB de RAM, Windows 11 Home Single Language, Versão 23H2
- Máquina 2 - Apple Silicon M1 (ARM), 8GB de RAM, MacOS Sonoma - Docker Container com Debian aarch64

| | Máquina 1 | | Máquina 2 | |
|--------|-----------|--------------|-----------|--------------|
| | Tempo (s) | Memória (MB) | Tempo (s) | Memória (MB) |
| BFS | 0.081454 | 0.139929 | 0.050281 | 0.139219 |
| DFS | 0.067894 | 0.144237 | 0.051053 | 0.144183 |
| GREEDY | 0.070699 | 0.104384 | 0.055811 | 0.104495 |
| A* | 0.058098 | 0.097603 | 0.045654 | 0.097585 |

Tabela 1: Tabela com os resultados Médios

| | Máquina 1 | | Máquina 2 | |
|--------|-----------|--------------|-----------|--------------|
| | Tempo (s) | Memória (MB) | Tempo (s) | Memória (MB) |
| BFS | 0.018017 | 0.005225 | 0.000685 | 0.001684 |
| DFS | 0.010628 | 0.001698 | 0.001499 | 0.001073 |
| GREEDY | 0.004866 | 0.000035 | 0.000938 | 0.000010 |
| A* | 0.003455 | 0.001523 | 0.000553 | 0.001529 |

Tabela 2: Tabela com os Desvios Padrões

4.1 Análise Comparativa dos Algoritmos

4.1.1 Dados Normalizados

| | Máquina 1 | | Máquina 2 | |
|--------|-----------|---------|-----------|---------|
| | Tempo | Memória | Tempo | Memória |
| BFS | 1,40201 | 1,43365 | 1,1013 | 1,42664 |
| DFS | 1,16861 | 1,47779 | 1,11825 | 1,47751 |
| GREEDY | 1.21689 | 1,06947 | 1,22247 | 1,07081 |
| A* | 1 | 1 | 1 | 1 |

Tabela 3: Dados Normalizados Pelo Melhor

Pelos dados normalizados é possível reparar que na máquina mais otimizada, a piora temporal foi menor, não passando de 22%, enquanto na outra máquina o pior algoritmo chegou a ser 40% mais devagar. No entanto, é notável que em ambas as máquinas o consumo de memória segue o mesmo padrão para ambas. Tendo um consumo acima de 40% para o BFS e DFS em comparação ao A*, enquanto o guloso utiliza próximo dos 10% a mais de memória em comparação com A*.

4.1.2 Desempenho em Tempo

O algoritmo A* é consistentemente o mais rápido em ambas as máquinas, enquanto o BFS tende a ser o mais lento na primeira. Curiosamente, na Máquina 2, o DFS apresentou desempenho inferior ao do BFS, mesmo ambos utilizando a mesma biblioteca Python. O algoritmo mais lento na Máquina 2 foi o guloso, o que pode ser atribuído ao fato de ser o único implementado manualmente, sem o suporte de uma implementação otimizada como as da biblioteca NetworkX, já que tal algoritmo não está presente na biblioteca.

É importante observar que o tempo de execução na Máquina 2 foi menor. Isso pode ser indicativo de que a diferença na arquitetura do processador em relação à Máquina 1 contribuiu para esse "ganho" de desempenho.

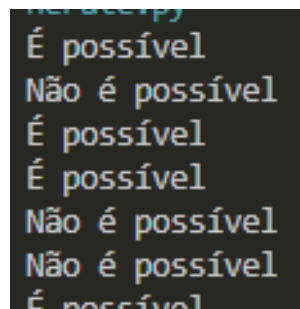
4.1.3 Uso de Memória

Todos os algoritmos apresentam uso de memória semelhante em ambas as máquinas, com o DFS utilizando ligeiramente mais memória e o A* utilizando menos. Esses valores próximos se devem ao

fato de os labirintos utilizados serem muito pequenos. Para uma avaliação mais precisa do uso de memória, seria necessário testar algoritmos em labirintos de escala maior, como centenas por centenas ou milhares por milhares de posições.

4.1.4 Completude

Todos os algoritmos rodaram completamente e perfeitamente, encontrando ou não o caminho caso fosse possível. Sendo assim todos demonstraram sua completude. Na Figura 2 é possível ver o começo da saída do algoritmo BFS informando se é possível ou não sair do ponto (0,0) até o ponto (9,9). Todos os quatro algoritmos chegaram ao mesmo resultado em todos os labirintos.



```

É possível
Não é possível
É possível
É possível
Não é possível
Não é possível
É possível

```

Figura 2: Exemplo da saída BFS para os labirintos

Pela Figura 2 é possível ver que o Labirinto 0 é possível e o Labirinto 1 não é possível, e isso corresponde a realidade ao serem analisadas as Figuras 3 e 4.

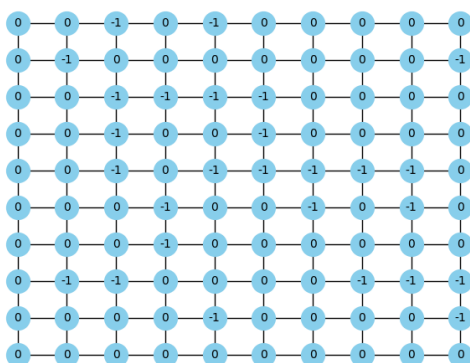


Figura 3: Labirinto 0

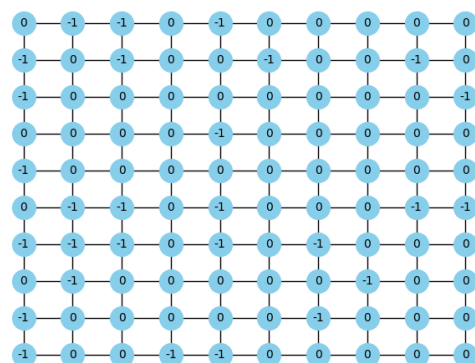


Figura 4: Labirinto 1

4.1.5 Optimalidade

Devido ao tamanho do labirinto, à quantidade de obstáculos e ao fato de sempre começarmos no ponto (0,0) e sairmos no ponto (9,9), todos os labirintos testados resultaram em caminhos considerados ótimos. Isso torna difícil avaliar a eficácia das soluções encontradas.

5 Conclusões

A análise comparativa dos algoritmos BFS, DFS, Guloso e A* em diferentes máquinas revelaria pontos importantes sobre desempenho, uso de memória, completude e optimalidade dos algoritmos. No entanto, dada a escala dos labirintos, não é possível afirmar com muita certeza que um algoritmo é muito melhor do que o outro em cada quesito.

Pelos dados, foi demonstrado que o A* é o algoritmo preferível para labirintos de pequena escala, devido ao seu rápido tempo de execução e uso eficiente de memória em ambas as máquinas. O uso de bibliotecas otimizadas é crucial para garantir um desempenho consistente, enquanto a completude dos algoritmos foram adequadamente demonstradas nos cenários testados.

5.1 Possíveis Melhorias

Para aprimorar a análise e o desempenho dos algoritmos de busca, as seguintes melhorias são sugeridas:

- **Greedy otimizado:** Encontrar uma biblioteca que implemente e otimize a busca gulosa, caso possível, que implemente as quatro buscas, tornando o cenário mais "justo" para este algoritmo ser comparado com outros.
- **Teste em Labirintos Maiores e Mais Complexos:** Expandir os testes para incluir labirintos de diferentes tamanhos e níveis de complexidade ajudará a avaliar melhor a escalabilidade e eficiência dos algoritmos em condições variadas.

Implementar essas melhorias pode levar a uma análise melhor e crível dos algoritmos utilizados.

Referências

- [1] Gabriel Alves and Pedro Siqueira. Comparative search algorithms. <https://github.com/gabriel0alvesz/Comparative-Search-Algorithms>, 2024. Accessed: 2024-08-11.
- [2] T.H. Cormen. *Algoritmos - Teoria e Prática*. GEN LTC, 2012. ISBN 9788535236996. URL <https://books.google.com.br/books?id=6iA4LgEACAAJ>.
- [3] Aric A. Hagberg, Daniel A. Schult, Pieter E. J. van der Walt, et al. Networkx. <https://networkx.github.io/>, 2024. Accessed: 2024-08-11.

6 Anexo A

6.1 Pseudocódigo Busca Gulosa

Algorithm 1 Busca Gulosa

```
1: function greedy_search(initial_position, finish_position)
2:   start  $\leftarrow$  nó em self.maze_graph onde (node.position_x, node.position_y) =
   initial_position
3:   goal  $\leftarrow$  nó em self.maze_graph onde (node.position_x, node.position_y) =
   finish_position
4:   open_set  $\leftarrow$  fila de prioridade vazia
5:   Adicionar (0, start) em open_set
6:   came_from  $\leftarrow$  dicionário vazio
7:   came_from[start]  $\leftarrow$  None
8:   while open_set não está vazio do
9:     (priority, current)  $\leftarrow$  remover o elemento com menor prioridade de open_set
10:    if current = goal then
11:      path  $\leftarrow$  lista vazia
12:      while current  $\neq$  None do
13:        Adicionar current em path
14:        current  $\leftarrow$  came_from[current]
15:      end while
16:      return OK
17:    end if
18:    for all neighbor em self.maze_graph.neighbors(current) do
19:      if neighbor.obstacle  $\neq$  -1 e neighbor  $\notin$  came_from then
20:        priority  $\leftarrow$  heurística de Manhattan entre neighbor e goal
21:        Adicionar (priority, neighbor) em open_set
22:        came_from[neighbor]  $\leftarrow$  current
23:      end if
24:    end for
25:  end while
26:  return None
27: end function
```

7 Anexo B

7.1 Imagens de Teste

```
BFS:
Tempo de execução de test_bfs: 0.069370 segundos
Tempo de execução de test_bfs: 0.073333 segundos
Tempo de execução de test_bfs: 0.066694 segundos
Tempo de execução de test_bfs: 0.118802 segundos
Tempo de execução de test_bfs: 0.110278 segundos
Tempo de execução de test_bfs: 0.127943 segundos
Tempo de execução de test_bfs: 0.086041 segundos
Tempo de execução de test_bfs: 0.094038 segundos
Tempo de execução de test_bfs: 0.082858 segundos
Tempo de execução de test_bfs: 0.100672 segundos
Tempo de execução de test_bfs: 0.125393 segundos
Tempo de execução de test_bfs: 0.079072 segundos
Tempo de execução de test_bfs: 0.073739 segundos
Tempo de execução de test_bfs: 0.065095 segundos
Tempo de execução de test_bfs: 0.068197 segundos
Tempo de execução de test_bfs: 0.071840 segundos
Tempo de execução de test_bfs: 0.066686 segundos
Tempo de execução de test_bfs: 0.065347 segundos
Tempo de execução de test_bfs: 0.075698 segundos
Tempo de execução de test_bfs: 0.071199 segundos
Tempo de execução de test_bfs: 0.066008 segundos
Tempo de execução de test_bfs: 0.069134 segundos
Tempo de execução de test_bfs: 0.078320 segundos
Tempo de execução de test_bfs: 0.075022 segundos
Tempo de execução de test_bfs: 0.071676 segundos
Tempo de execução de test_bfs: 0.091622 segundos
Tempo de execução de test_bfs: 0.074110 segundos
Tempo de execução de test_bfs: 0.080039 segundos
Tempo de execução de test_bfs: 0.077659 segundos
Tempo de execução de test_bfs: 0.067745 segundos
Tempo médio de execução: 0.081454 segundos
Desvio padrão do tempo de execução: 0.018017 segundos
```

Figura 5: Tempo BFS - M1

```
DFS:
Tempo de execução de test_dfs: 0.064634 segundos
Tempo de execução de test_dfs: 0.064587 segundos
Tempo de execução de test_dfs: 0.072139 segundos
Tempo de execução de test_dfs: 0.071448 segundos
Tempo de execução de test_dfs: 0.071148 segundos
Tempo de execução de test_dfs: 0.079729 segundos
Tempo de execução de test_dfs: 0.114333 segundos
Tempo de execução de test_dfs: 0.064543 segundos
Tempo de execução de test_dfs: 0.058966 segundos
Tempo de execução de test_dfs: 0.067776 segundos
Tempo de execução de test_dfs: 0.060906 segundos
Tempo de execução de test_dfs: 0.064543 segundos
Tempo de execução de test_dfs: 0.065716 segundos
Tempo de execução de test_dfs: 0.056422 segundos
Tempo de execução de test_dfs: 0.061628 segundos
Tempo de execução de test_dfs: 0.062492 segundos
Tempo de execução de test_dfs: 0.064708 segundos
Tempo de execução de test_dfs: 0.063860 segundos
Tempo de execução de test_dfs: 0.066054 segundos
Tempo de execução de test_dfs: 0.059113 segundos
Tempo de execução de test_dfs: 0.059007 segundos
Tempo de execução de test_dfs: 0.063558 segundos
Tempo de execução de test_dfs: 0.065593 segundos
Tempo de execução de test_dfs: 0.070615 segundos
Tempo de execução de test_dfs: 0.071082 segundos
Tempo de execução de test_dfs: 0.064293 segundos
Tempo de execução de test_dfs: 0.078677 segundos
Tempo de execução de test_dfs: 0.071237 segundos
Tempo de execução de test_dfs: 0.079155 segundos
Tempo de execução de test_dfs: 0.061667 segundos
Tempo médio de execução: 0.067894 segundos
Desvio padrão do tempo de execução: 0.010628 segundos
```

Figura 6: Tempo DFS - M1

```
Greedy:
Tempo de execução de test_greedy: 0.071073 segundos
Tempo de execução de test_greedy: 0.072073 segundos
Tempo de execução de test_greedy: 0.071235 segundos
Tempo de execução de test_greedy: 0.072150 segundos
Tempo de execução de test_greedy: 0.072290 segundos
Tempo de execução de test_greedy: 0.071790 segundos
Tempo de execução de test_greedy: 0.060975 segundos
Tempo de execução de test_greedy: 0.070106 segundos
Tempo de execução de test_greedy: 0.066744 segundos
Tempo de execução de test_greedy: 0.070540 segundos
Tempo de execução de test_greedy: 0.072669 segundos
Tempo de execução de test_greedy: 0.071815 segundos
Tempo de execução de test_greedy: 0.073606 segundos
Tempo de execução de test_greedy: 0.072439 segundos
Tempo de execução de test_greedy: 0.069555 segundos
Tempo de execução de test_greedy: 0.068203 segundos
Tempo de execução de test_greedy: 0.085989 segundos
Tempo de execução de test_greedy: 0.065901 segundos
Tempo de execução de test_greedy: 0.072085 segundos
Tempo de execução de test_greedy: 0.074933 segundos
Tempo de execução de test_greedy: 0.071092 segundos
Tempo de execução de test_greedy: 0.065205 segundos
Tempo de execução de test_greedy: 0.067854 segundos
Tempo de execução de test_greedy: 0.074031 segundos
Tempo de execução de test_greedy: 0.071589 segundos
Tempo de execução de test_greedy: 0.069208 segundos
Tempo de execução de test_greedy: 0.079785 segundos
Tempo de execução de test_greedy: 0.064481 segundos
Tempo de execução de test_greedy: 0.061037 segundos
Tempo de execução de test_greedy: 0.070527 segundos
Tempo médio de execução: 0.070699 segundos
Desvio padrão do tempo de execução: 0.004866 segundos
```

Figura 7: Tempo Greedy - M1

```
A*:
Tempo de execução de test_a_star: 0.055643 segundos
Tempo de execução de test_a_star: 0.059684 segundos
Tempo de execução de test_a_star: 0.060413 segundos
Tempo de execução de test_a_star: 0.067010 segundos
Tempo de execução de test_a_star: 0.060250 segundos
Tempo de execução de test_a_star: 0.056099 segundos
Tempo de execução de test_a_star: 0.059007 segundos
Tempo de execução de test_a_star: 0.059353 segundos
Tempo de execução de test_a_star: 0.058530 segundos
Tempo de execução de test_a_star: 0.059180 segundos
Tempo de execução de test_a_star: 0.062312 segundos
Tempo de execução de test_a_star: 0.065475 segundos
Tempo de execução de test_a_star: 0.053125 segundos
Tempo de execução de test_a_star: 0.057029 segundos
Tempo de execução de test_a_star: 0.054793 segundos
Tempo de execução de test_a_star: 0.055125 segundos
Tempo de execução de test_a_star: 0.055851 segundos
Tempo de execução de test_a_star: 0.057214 segundos
Tempo de execução de test_a_star: 0.060640 segundos
Tempo de execução de test_a_star: 0.059642 segundos
Tempo de execução de test_a_star: 0.058597 segundos
Tempo de execução de test_a_star: 0.053102 segundos
Tempo de execução de test_a_star: 0.056178 segundos
Tempo de execução de test_a_star: 0.057904 segundos
Tempo de execução de test_a_star: 0.057740 segundos
Tempo de execução de test_a_star: 0.056344 segundos
Tempo de execução de test_a_star: 0.060607 segundos
Tempo de execução de test_a_star: 0.050241 segundos
Tempo de execução de test_a_star: 0.055945 segundos
Tempo de execução de test_a_star: 0.059902 segundos
Tempo médio de execução: 0.058098 segundos
Desvio padrão do tempo de execução: 0.003455 segundos
```

Figura 8: Tempo A* - M1

Figura 11: Memória Greedy - M1

Figura 12: Memória A* - M1


```
BFS:
```

| | | | |
|----------------------------|-----------|-------------|--------|
| Uso de memória de | test_bfs: | 0.148135 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138847 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138981 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138981 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138847 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138981 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138847 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Uso de memória de | test_bfs: | 0.138914 MB | (pico) |
| Memória Média de execução: | | 0.139219 MB | |

Desvio padrão da memória de execução: 0.001684 MB

Figura 17: Memória BFS - M2

```
Dfs:
```

| | | |
|---------------------------------------|-------------|--------|
| Uso de memória de test_dfs: | 0.149863 MB | (pico) |
| Uso de memória de test_dfs: | 0.144010 MB | (pico) |
| Uso de memória de test_dfs: | 0.143943 MB | (pico) |
| Uso de memória de test_dfs: | 0.144010 MB | (pico) |
| Uso de memória de test_dfs: | 0.144010 MB | (pico) |
| Uso de memória de test_dfs: | 0.143943 MB | (pico) |
| Uso de memória de test_dfs: | 0.144010 MB | (pico) |
| Uso de memória de test_dfs: | 0.144010 MB | (pico) |
| Uso de memória de test_dfs: | 0.143943 MB | (pico) |
| Uso de memória de test_dfs: | 0.144010 MB | (pico) |
| Uso de memória de test_dfs: | 0.143943 MB | (pico) |
| Uso de memória de test_dfs: | 0.144010 MB | (pico) |
| Uso de memória de test_dfs: | 0.143943 MB | (pico) |
| Uso de memória de test_dfs: | 0.144010 MB | (pico) |
| Uso de memória de test_dfs: | 0.143943 MB | (pico) |
| Uso de memória de test_dfs: | 0.144010 MB | (pico) |
| Uso de memória de test_dfs: | 0.143943 MB | (pico) |
| Uso de memória de test_dfs: | 0.144010 MB | (pico) |
| Uso de memória de test_dfs: | 0.143943 MB | (pico) |
| Uso de memória de test_dfs: | 0.144010 MB | (pico) |
| Uso de memória de test_dfs: | 0.143943 MB | (pico) |
| Uso de memória de test_dfs: | 0.144010 MB | (pico) |
| Uso de memória de test_dfs: | 0.143943 MB | (pico) |
| Memória Média de execução: | 0.144183 MB | |
| Desvio padrão da memória de execução: | 0.001073 MB | |

Figura 18: Memória DFS - M2

