Uma imagem com Tipo de letra, Gráficos, design gráfico, logótipo

Descrição gerada automaticamente

**Sprint 2**

**Turma 3NA-Grupo 76**

1200583 – João Carrinho

1200586 – Mário Borja

1200618 – Jorge Cunha

1210815 – João Barros

**Professor:**

Francisco Samuel Paiva da Silva, FPS

**Unidade Curricular:**

Algoritmia Avançada (ALGAV)

**Data: 26/11/2023**

**Explicação dos Métodos em Prolog**

Este documento apresenta uma explicação detalhada dos métodos utilizados em um programa Prolog para a gestão de edifícios e navegação espacial.

**Declarações Dinâmicas**

**:-dynamic m/5**

Esta declaração permite que fatos do tipo m(Building, Floor, Column, Line, Value) sejam alterados em tempo de execução. É usado para representar uma matriz de valores em um edifício e andar específicos.

**:-dynamic pisos/2**

Esta declaração não é utilizada no código fornecido, mas geralmente seria usada para permitir a modificação dinâmica de fatos com dois argumentos relacionados a pisos.

**:-dynamic ligacel/3**

Permite a modificação dinâmica de fatos do tipo ligacel(Cell1, Cell2, Cost), representando a ligação e o custo entre duas células (ou espaços) em um edifício.

**:-dynamic elevador/2**

Permite a modificação dinâmica de fatos do tipo elevador(Building, Floors), representando a existência de um elevador em um edifício e os andares que ele atende.

**Criação de Matriz**

O método cria\_matriz é utilizado para criar uma matriz com todos os valores inicializados a 0. Ele começa por pedir ao usuário o número de colunas e linhas, e depois utiliza o método cria\_matriz\_0 para preencher a matriz.

**Criação de Grafos**

O método graphs e suas funções auxiliares (graphs\_aux, create\_passageway\_connections, create\_elevator\_connections) são usados para criar ligações entre as células da matriz, representando passagens e conexões de elevador entre diferentes andares e edifícios.

**Busca em Profundidade (DFS)**

O método dfs e suas variantes (all\_dfs, semiBetter\_dfs, better\_dfs1) implementam o algoritmo de busca em profundidade para encontrar caminhos entre dois pontos na matriz.

**Busca em Largura (BFS)**

O método bfs implementa o algoritmo de busca em largura para encontrar o caminho mais curto entre dois pontos.

**A\* Search**

O método aStar implementa o algoritmo A\* para encontrar o caminho mais eficiente entre dois pontos, considerando tanto o custo quanto a estimativa de distância restante.

**Big O**

**Criação de Matriz (cria\_matriz e cria\_matriz\_0):**

A complexidade deste método depende do número de colunas (NCol) e linhas (NLin) da matriz. Como ele percorre cada célula da matriz uma vez, a complexidade é O(NCol \* NLin).

**Criação de Grafos (graphs, graphs\_aux, create\_passageway\_connections, create\_elevator\_connections):**

A complexidade destes métodos depende do número de células na matriz e das conexões entre elas. Em geral, a complexidade pode ser considerada O(N^2), onde N é o número de células, já que cada célula pode potencialmente se conectar com todas as outras.

**Busca em Profundidade (DFS - dfs, all\_dfs, semiBetter\_dfs, better\_dfs1):**

A complexidade do DFS é O (V + E), onde V é o número de vértices (células) e E é o número de arestas (conexões). No pior caso, isso pode ser aproximadamente O(N^2) para uma matriz densamente conectada.

**Busca em Largura (BFS - bfs):**

Similar ao DFS, a complexidade do BFS é O(V + E). Novamente, no pior caso, isso pode ser aproximadamente O(N^2).

**\*A Search (aStar)\*\*:**

A complexidade do A\* depende da heurística usada e da estrutura do grafo. Em geral, é difícil determinar uma complexidade exata, mas no pior caso, pode ser O(E), onde E é o número de arestas. Com uma boa heurística, pode ser significativamente mais rápido.

Abaixo está representada a tabela de análise de complexidade dos vários métodos de busca e os gráficos para melhor interpretação de dados.

**Tabela de análise da complexidade**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| NColxNLin | Nº nós | Nº lig. | Solução 1ºProf dfs | Custo e Tempo | Solução 1ºLarg bfs | Custo e Tempo | Melhor Solução - todas 1º Prof aStar | Custo e Tempo |
| 4X4 | 16 | 48 | [cel(d, d1, 1, 1), cel(d, d1, 2, 1), cel(d, d1, 3, 1), cel(d, d1, 4, 1), cel(d, d1, 4, 2), cel(d, d1, 3, 2), cel(d, d1, 2, 2), cel(d, d1, 1, 2), cel(..., ..., ..., ...)|...] | Time:  0.0065  Cost = 12 | [cel(d, d1, 1, 1), cel(d, d1, 2, 2), cel(d, d1, 3, 3), cel(d, d1, 4, 4)] | Time:0.0217  Cost = 4.2 | [cel(d, d1, 1, 1), cel(d, d1, 2, 2), cel(d, d1, 3, 3), cel(d, d1, 4, 4)] | Time:0.00116  Cost = 4.2 |
| 5X5 | 25 | 80 | [cel(d, d1, 1, 1), cel(d, d1, 2, 1), cel(d, d1, 3, 1), cel(d, d1, 4, 1), cel(d, d1, 5, 1), cel(d, d1, 5, 2), cel(d, d1, 4, 2), cel(d, d1, 3, 2), cel(..., ..., ..., ...)|...], | Time: 0.000123 Cost 24 | [cel(d, d1, 1, 1), cel(d, d1, 2, 2), cel(d, d1, 3, 3), cel(d, d1, 4, 4), cel(d, d1, 5, 5)], | Time: 1.850  Cost: 5.6 | [cel(d, d1, 1, 1), cel(d, d1, 2, 2), cel(d, d1, 3, 3), cel(d, d1, 4, 4), cel(d, d1, 5, 5)], | Time: 0.000713  Cost: 5.6 |
| 6X6 | 36 | 120 | [cel(d, d1, 1, 1), cel(d, d1, 2, 1), cel(d, d1, 3, 1), cel(d, d1, 4, 1), cel(d, d1, 5, 1), cel(d, d1, 5, 2), cel(d, d1, 4, 2), cel(d, d1, 3, 2), cel(..., ..., ..., ...)|...] | Time: 0.000248  Cost: 26 | Error | Error | [cel(d, d1, 1, 1), cel(d, d1, 2, 2), cel(d, d1, 3, 3), cel(d, d1, 4, 4), cel(d, d1, 5, 5), cel(d, d1, 6, 6) | Time: 0.00174  Cost: 7.0 |
| 7X7 | 49 | 168 | [cel(d, d1, 1, 1), cel(d, d1, 2, 1), cel(d, d1, 3, 1), cel(d, d1, 4, 1), cel(d, d1, 5, 1), cel(d, d1, 6, 1), cel(d, d1, 7, 1), cel(d, d1, 7, 2), cel(..., ..., ..., ...)|...] | Time: 0.000164  Cost: 48 | Error | Error | [cel(d, d1, 1, 1), cel(d, d1, 2, 2), cel(d, d1, 3, 3), cel(d, d1, 4, 4), cel(d, d1, 5, 5), cel(d, d1, 6, 6), cel(d, d1, 7, 7)] | Time: 0.00832  Cost: 8.4 |
| 8X8 | 64 | 224 | [cel(d, d1, 1, 1), cel(d, d1, 2, 1), cel(d, d1, 3, 1), cel(d, d1, 4, 1), cel(d, d1, 5, 1), cel(d, d1, 6, 1), cel(d, d1, 7, 1), cel(d, d1, 7, 2), cel(..., ..., ..., ...)|...] | Time: 0.000187  Cost: 50 | Error | Error | [cel(d, d1, 1, 1), cel(d, d1, 2, 2), cel(d, d1, 3, 3), cel(d, d1, 4, 4), cel(d, d1, 5, 5), cel(d, d1, 6, 6), cel(d, d1, 7, 7), cel(d, d1, 8, 8)] | Time: 0.00276  Cost: 9.8 |
| … |  |  |  |  |  |  |  |  |

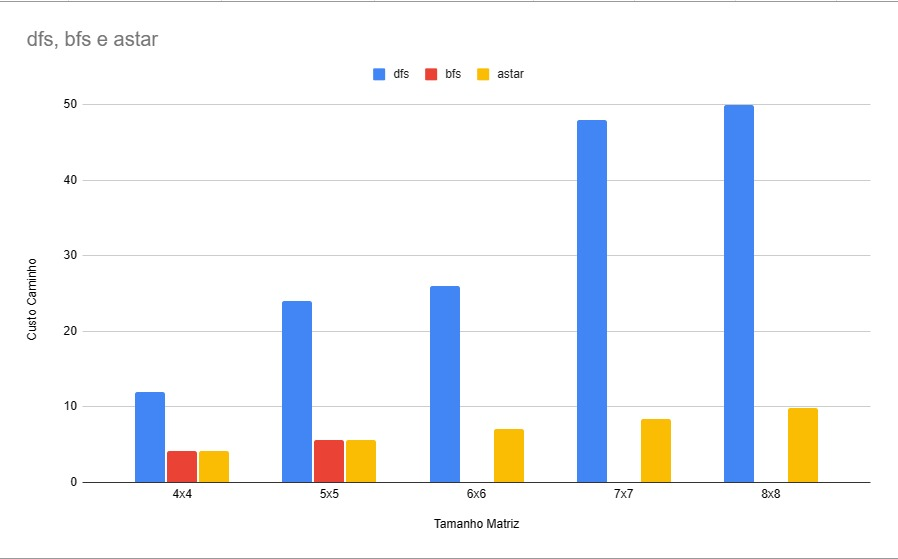
****

Figura - Tamanho Matriz dfs, bfs e astar

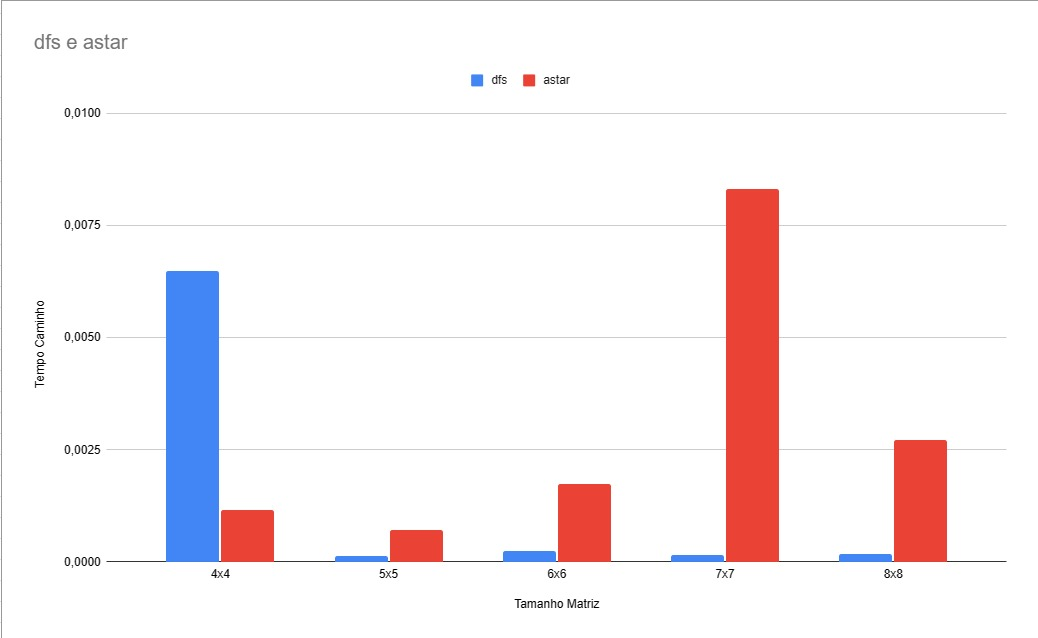
****

Figura - Tamanho Matriz dfs e astar

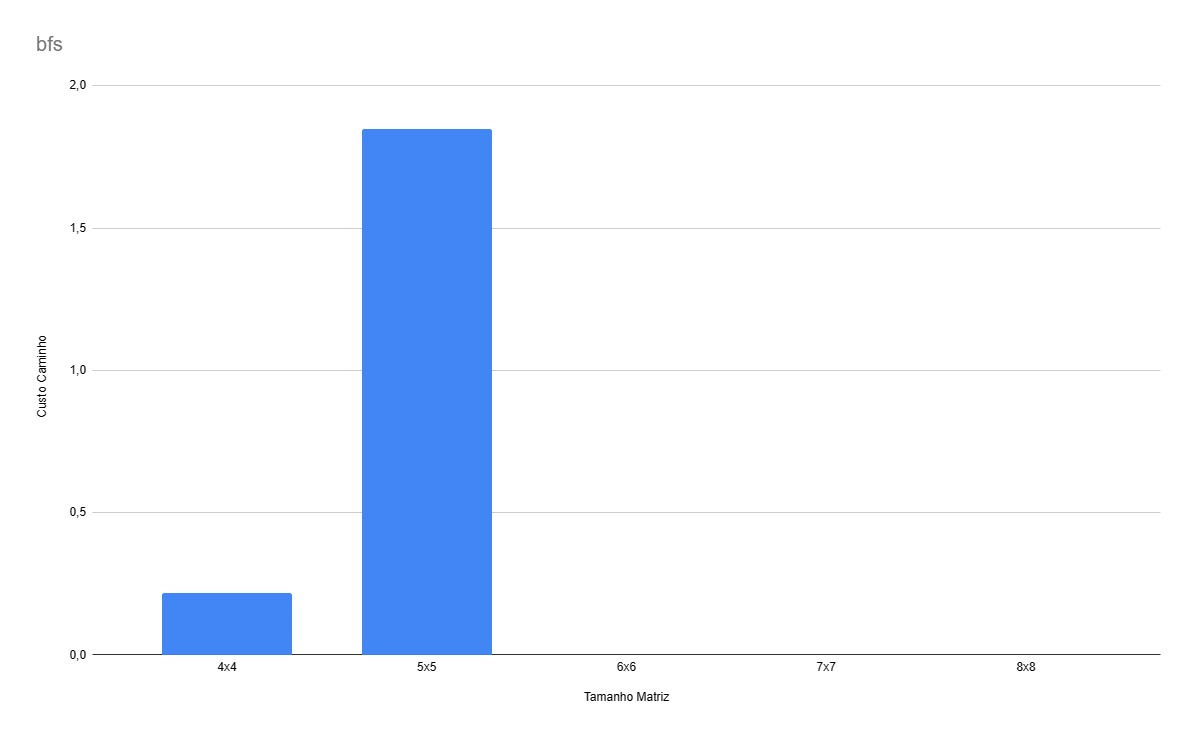
****

Figura - Tamanho Matriz **bfs**

**Conclusão**

Estes métodos fornecem uma base robusta para a modelagem de espaços de edifícios e a navegação entre eles, utilizando diferentes técnicas de busca para otimizar o percurso.

Com a analise dos gráficos, sentimos também a necessidade de separar os gráficos de tempo/tamanho em bfs (figura3) e dfs/astar (figura 2), pois reparamos que a dado tamanho de matriz o bfs deixava de dar resposta e os tempos em relação aos restantes tamanhos os tempos mesmo assim eram elevados em comparação para com o dfs/astar.

É possível reiterar que as soluções através do método BFS, pararam de evidenciar resultados (ou demoraram demasiado tempo para recebermos uma resposta) a partir da matriz 6x6. Isto acontece devido à possível complexidade do grafo, ou seja, quanto maior for a quantidade de nodes que o BFS precisará de explorar, devido à natureza exponencial do BFS em termos de espaço de busca, o tempo de resposta também aumenta exponencialmente.

Podemos também perceber que o método A Search, apesar de, por vezes, fornecer valores temporais superiores aos outros métodos, no geral, é o método que melhor equilibra o custo e o tempo associado à procura.

Para efeitos de teste, fizemos a análise dos métodos de busca até uma matriz de 30x30 e, apesar de a discrepância de tempo ser negligenciável, a discrepância de custo aumenta cada vez mais, com o DFS a atingir valores de 600 no custo.