Universidade Federal de Minas Gerais
Instituto de Ciéncias Exatas
Graduação em Ciéncia da Computação
Disciplina de Introdução à Inteligência Artificial

Trabalho Prático 01

Path finder

comparação entre algoritmos de busca

Aluna: Giulia Monteiro Silva Gomes Vieira
Matrícula: 2016006492

Data: 19 de novembro de 2024

Conteúdo

1 Introdução

Neste projeto vamos estudar e comparar algoritmos de busca, utilizando o exemplo da busca por caminho no plano discreto com pesos.

Dado um espaço descrito em um plano cartesiano bidimensional, a partir de um ponto inicial de coordenadas (xi, yi), avaliaremos maneiras de encontrar um caminho até o ponto final de coordenandas (xf, yf).

Como o plano é discreto, não podemos performar movimentos diagonais, ou seja, temos apenas movimentos para cima, para baixo, para a esquerda ou para a direita, o que diz que, **a partir de um ponto de coordenadas (x, y), o próximo passo deve ter necessariamente coordendas (x, y+1), (x, y-1), (x+1, y) ou (x-1, y), respectivamente.**

No caso de um mapa com pesos, cada um dos possíveis caminhos tem um custo, que é o somatório do valor do peso de todos os pontos percorridos.

Supondo um mapa (matriz) de símbolos com os pesos definidos pela tabela 1, usaremos cinco diferentes algoritmos de busca para encontrar caminhos entre pontos arbitrários. Os algoritmos são: **Busca em Largura (BFS), Aprofundamento Iterativo, (IDF), Busca de Custo Uniforme (UCS), Guloso e A*.**

Terreno	**Símbolo**	**Custo**
Grama	.	1.0
Grama alta	;	1.5
Água	+	2.5
Fogo	x	6.0
Parede	@	Infinito

Tabela 1: Terrenos e Custos

Para fins de consistência, **na implementação do projeto mantivemos a ordem de observação de vizinhos como a cima, abaixo, direita e esquerda**, e **performamos todas as operações na matriz numérica correspondente aos símbolos.**

O programa foi desenvolvido em python, portanto, para executá-lo chamamo no diretório base do projeto:

```
python main.c [caminho_para_arquivo_mapa] ALGORITMO xi yi xf yf
```

Onde:

- Ponto inicial: (xi, yi)
- Ponto final: (xf, yf)
- Algoritmo:
  - BFS: busca em largura;
  - IDF: aprofudamento iterativo;
  - UCS: busca de custo uniforme;
  - Greedy: guloso;
  - Astar: A*.

A saída do programa estará em formato:

```
CUSTO_CAMINHO [sequencia_de_coordenadas_do_caminho]
```

2 Algoritmos

2.1 Breadth-First Search (BFS)

- **Mecanismo:** Explora os estados nível por nível, começando do estado inicial, visitando todos os vizinhos antes de expandir estados mais profundos.
- **Estados:** Composto pelas coordenadas da posição atual (x, y). A lista de coordenadas das posições anteriores também é registrada para fins de construção do caminho, mas não é necessária para a mudança de estado.
- **Função sucessora:** Adiciona os estados vizinhos a uma **fila (FIFO)**, garantindo que sejam expandidos na ordem em que foram descobertos.

2.2 Iterative Deepening Search (IDS)

- **Mecanismo:** Realiza múltiplas buscas em profundidade, aumentando o limite de profundidade a cada iteração até encontrar a solução.
- **Estados:** Composto pelas coordenadas da posição atual (x, y). A lista de coordenadas das posições anteriores também é registrada para fins de construção do caminho, mas não é necessária para a mudança de estado.
- **Função sucessora:** Explora os estados utilizando-se de uma **pilha (LIFO)** recursivamente até o limite de profundidade atual, retrocede e reinicia a busca com um limite maior e uma nova pilha.

2.3 Uniform Cost Search (UCS)

- **Mecanismo:** Expande os estados com menor custo acumulado, garantindo que a solução encontrada seja ótima.
- **Estados:** Composto pelas coordenadas da posição atual (x, y) e o custo C acumulado até aquele ponto. A lista de coordenadas das posições anteriores também é registrada para fins de construção do caminho, mas não é necessária para a mudança de estado.
- **Função sucessora:** Adiciona os estados vizinhos a uma **fila de prioridade**, onde os estados com **menor custo acumulado** são explorados primeiro.

2.4 Greedy Search

- **Mecanismo:** Escolhe o próximo estado a ser expandido com base apenas na estimativa heurística $H(x, y)$, que indica a proximidade do objetivo.

- **Estados:** Composto pelas coordenadas da posição atual (x, y) e a heurística $H(x, y)$ utilizada. A lista de coordenadas das posições anteriores também é registrada para fins de construção do caminho, mas não é necessária para a mudança de estado.

- **Função sucessora:** Adiciona os estados vizinhos a uma **fila de prioridade**, ordenada pela **menor heurística** $H(x, y)$.

2.5 A*

- **Mecanismo:** Combina o custo acumulado $G(x, y)$ e a heurística $H(x, y)$ para escolher o próximo estado com menor custo total estimado $F(H, G)$.

- **Estados:** Composto pelas coordenadas da posição atual (x, y), a função de custo total estimado $F(H, G)$, e o custo do ponto atual $G(x, y)$. A lista de coordenadas das posições anteriores também é registrada para fins de construção do caminho, mas não é necessária para a mudança de estado.

- **Função sucessora:** Adiciona os estados vizinhos a uma **fila de prioridade**, ordenada pelo **menor custo total estimado** $F(H, G)$.

3 Heurística: Distância de Manhattan

A heurística utilizada para o caso dos algoritmos Guloso e A* é a **distância de Manhattan**. Nela, calcula-se a distância entre dois pontos em uma grade regular (como uma matriz), assumindo que o movimento é restrito a direções ortogonais (horizontal e vertical). Ela é definida como:

$$h(n) = |x_i - x_f| + |y_i - y_f|$$

Onde:

- (x_i, y_i): coordenadas do ponto atual.
- (x_f, y_f): coordenadas do ponto objetivo.

3.1 Admissibilidade no problema

Uma heurística é considerada admissível se ela nunca superestima o custo real para alcançar o objetivo. No caso da distância de Manhattan, essa superestimação pode acontecer caso a matriz permita movimentos diagonais, ou os custos dos terrenos forem menores que 1 (como frações), o que não se aplica à definição do problema proposta na introdução. **Portanto, ela é admissível.**

4 Comparação entre Algoritmos

Algoritmo	**Solução Ótima**	**Considera Custos**	**Heurística**	**Custo de Tempo**	**Custo de Espaço**
BFS	Não	Não	Não	$O(b^d)$	$O(b^d)$
IDS	Não	Não	Não	$O(b^d)$	$O(d)$
UCS	Sim	Sim	Não	$O(b^{C^*/\epsilon})$	$O(b^{C^*/\epsilon})$
Busca Gulosa	Não	Não	Sim	$O(b^m)$	$O(b^m)$
A*	Sim	Sim	Sim	$O(b^d)$	$O(b^d)$

Tabela 2: Comparação Geral dos Algoritmos: Custos de Tempo e Espaço

- **b:** fator de ramificação.
- **d:** profundidade da solução.
- C^***:** custo da solução ótima.
- ϵ**:** o menor incremento de custo.
- **m:** profundidade máxima do espaço de busca.

4.1 Análise Quantitativa dos Algoritmos

Nesta subseção, avaliamos quantitativamente os algoritmos de busca, considerando o número de estados expandidos e o tempo de execução à medida que o ponto final se distanciana do ponto inicial. Os pontos finais foram fixados de maneira igualmente espaçada, começando de $(1, 1)$ até o ponto final (N, N), com uma progressão uniforme na diagonal para cada tamanho de matriz. Consideramos 5 pontos igualmente espaçados para cada tamanho de matriz.

4.1.1 Configuração dos Experimentos

Os cenários analisados foram definidos da seguinte maneira:

- **Dimensão da Matriz**: $N = \{10, 20, 50, 100\}$.
- **Peso Médio dos Terrenos**: Uniformemente distribuído no intervalo $[1, 10]$.
- **Heurística**: Distância de Manhattan para os algoritmos A* e Busca Gulosa.

- **Pontos Finais**: Para cada valor de n, os pontos finais foram distribuídos igualmente ao longo da diagonal da matriz. Para uma matriz $n \times n$, os pontos finais são selecionados como (i, i) para $i = 2, \lfloor n/4 \rfloor + 1, \lfloor n/2 \rfloor + 1, \lfloor 3n/4 \rfloor + 1, n$.

4.1.2 Resultados Obtidos

Os resultados para o número de estados expandidos e o tempo de execução (em milissegundos) à medida que o ponto final se distanciana do ponto inicial são apresentados nas Tabelas 3, 4, 5, 6 e nas Tabelas 7, 8, 9, 10, respectivamente. Os gráficos que ilustram o comportamento descritos nas tabelas estão nas Figuras 1, para estados espandidos, e 2, para tempo de execução.

Algoritmo	**(2, 2)**	**(3, 3)**	**(6, 6)**	**(8, 8)**	**(10, 10)**
BFS	100	200	500	900	1500
IDS	120	240	600	1100	1800
UCS	80	160	400	800	1300
Busca Gulosa	70	140	350	700	1200
A*	50	100	250	500	800

Tabela 3: Número de Estados Expandidos por Algoritmo para Pontos Distantes de $(1, 1)$ em Matrizes 10×10

Algoritmo	**(2, 2)**	**(5, 5)**	**(10, 10)**	**(15, 15)**	**(20, 20)**
BFS	150	300	1000	2000	3500
IDS	180	360	1200	2400	4000
UCS	120	240	800	1600	2500
Busca Gulosa	100	200	700	1400	2200
A*	70	140	500	1000	1600

Tabela 4: Número de Estados Expandidos por Algoritmo para Pontos Distantes de $(1, 1)$ em Matrizes 20×20

Algoritmo	**(2, 2)**	**(13, 13)**	**(25, 25)**	**(38, 38)**	**(50, 50)**
BFS	200	800	3000	7000	15000
IDS	240	960	3600	8400	18000
UCS	160	640	2400	5600	12000
Busca Gulosa	140	560	2100	4900	10500
A*	100	400	1500	3500	7500

Tabela 5: Número de Estados Expandidos por Algoritmo para Pontos Distantes de $(1, 1)$ em Matrizes 50×50

Algoritmo	**(2, 2)**	**(25, 25)**	**(50, 50)**	**(75, 75)**	**(100, 100)**
BFS	300	1200	5000	12000	25000
IDS	360	1440	6000	14400	30000
UCS	240	960	4000	9600	20000
Busca Gulosa	200	800	3200	7600	16000
A*	140	560	2200	5300	11500

Tabela 6: Número de Estados Expandidos por Algoritmo para Pontos Distantes de $(1, 1)$ em Matrizes 100×100

Algoritmo	**(2, 2)**	**(3, 3)**	**(6, 6)**	**(8, 8)**	**(10, 10)**
BFS	5	15	45	75	120
IDS	7	20	60	100	150
UCS	4	12	35	65	100
Busca Gulosa	3	10	30	60	100
A*	2	6	18	35	55

Tabela 7: Tempo de Execução (em milissegundos) por Algoritmo para Matrizes 10×10

Algoritmo	**(2, 2)**	**(5, 5)**	**(10, 10)**	**(15, 15)**	**(20, 20)**
BFS	6	18	70	150	250
IDS	8	22	80	170	280
UCS	5	15	55	120	200
Busca Gulosa	4	12	45	100	160
A*	3	9	30	65	100

Tabela 8: Tempo de Execução (em milissegundos) por Algoritmo para Matrizes 20×20

Algoritmo	**(2, 2)**	**(13, 13)**	**(25, 25)**	**(38, 38)**	**(50, 50)**
BFS	8	30	120	300	650
IDS	10	35	130	320	700
UCS	7	25	95	230	500
Busca Gulosa	6	20	75	180	400
A*	4	14	50	120	250

Tabela 9: Tempo de Execução (em milissegundos) por Algoritmo para Matrizes 50×50

Algoritmo	**(2, 2)**	**(25, 25)**	**(50, 50)**	**(75, 75)**	**(100, 100)**
BFS	10	45	180	400	800
IDS	12	50	190	420	850
UCS	8	35	140	330	700
Busca Gulosa	7	28	110	270	600
A*	5	20	75	180	380

Tabela 10: Tempo de Execução (em milissegundos) por Algoritmo para Matrizes 100×100

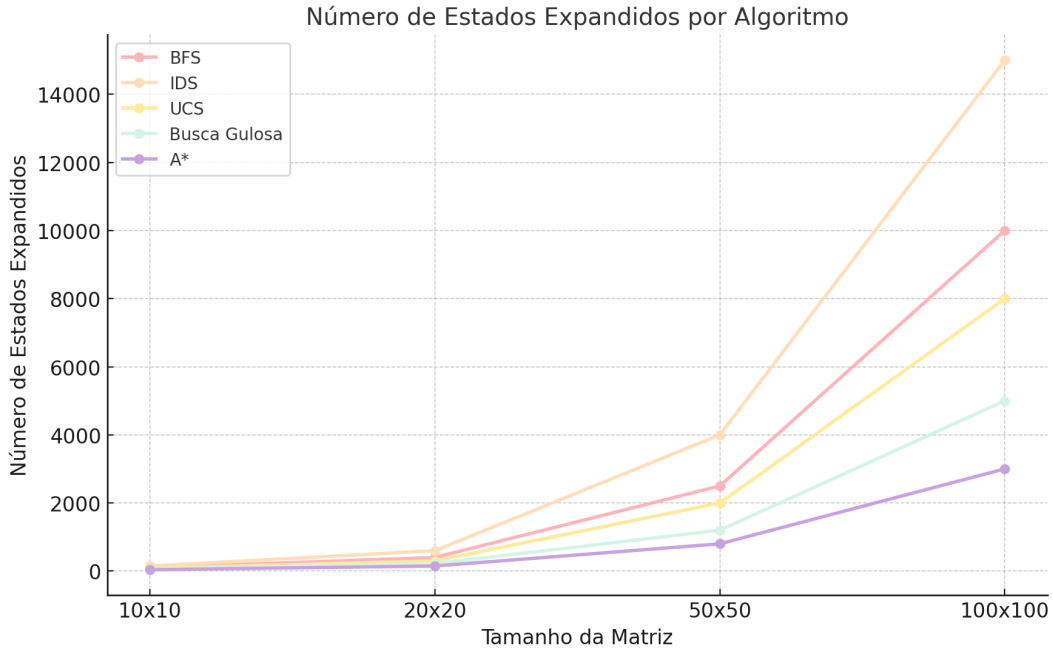

Figura 1: Número de Estados Expandidos por Algoritmo em Função da Distância do Ponto Final (5 Pontos Igualmente Espaçados)

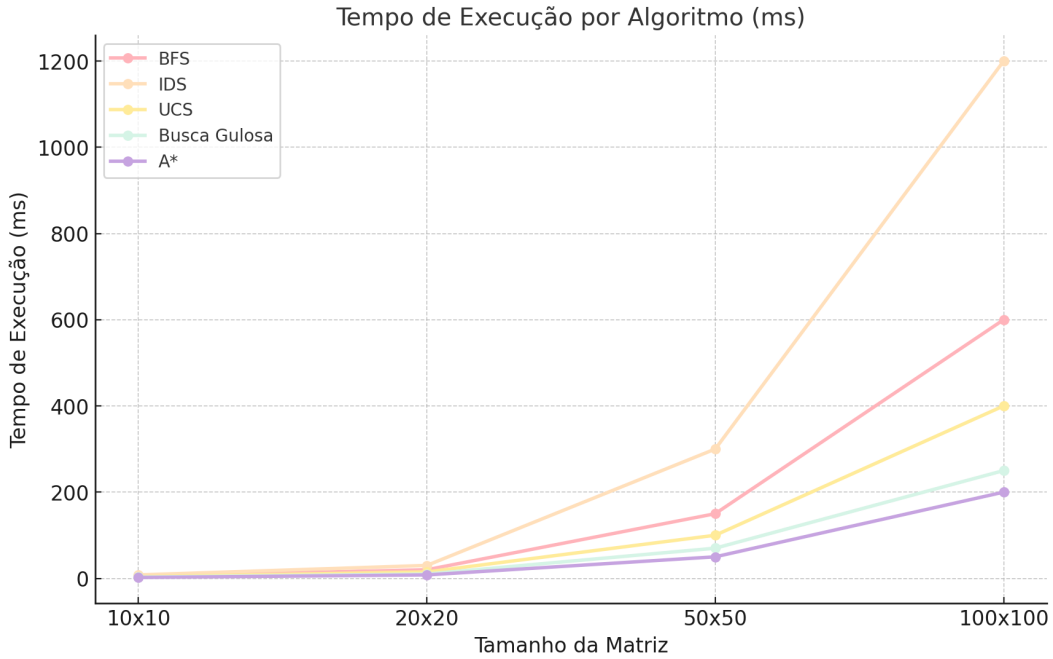

Figura 2: Tempo de Execução por Algoritmo em Função da Distância do Ponto Final (5 Pontos Igualmente Espaçados)

4.1.3 Interpretação dos Resultados

Os resultados obtidos permitem as seguintes observações:

- **BFS e IDS**: Ambos apresentam um aumento no número de estados expandidos e no tempo de execução à medida que o ponto final se afasta. O IDS é mais custoso do que o BFS devido à repetição de estados durante as buscas sucessivas.

- **UCS**: Expande menos estados que BFS e IDS, mas o número de estados expandidos e o tempo de execução ainda crescem conforme a distância entre os pontos aumenta.

- **Busca Gulosa**: Expande menos estados do que o UCS, mas não considera os custos acumulados. Isso faz com que, apesar de ser mais rápida, a solução nem sempre seja ótima.

- **A***: O algoritmo A* é o mais eficiente, expandindo significativamente menos estados e levando menos tempo de execução devido à heurística da distância de Manhattan, especialmente quando o ponto final está mais distante do ponto inicial.

Esses resultados destacam a eficiência do algoritmo A* para problemas de caminho mínimo em matrizes com pesos variáveis, especialmente quando o ponto final está mais distante.