Algoritmos e Estruturas de Dados III

Algoritmos de Caminhos Mínimos Aplicados a Mapas Bidimensionais

Luccas Vinicius P. A. Santos Carneiro
20.1.8015

Thiago Ker Gama Nunes Carvalho
23.2.8019

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
26 de janeiro de 2026

Resumo

Este trabalho prático apresenta a implementação e a análise comparativa de algoritmos clássicos de caminhos mínimos em grafos, aplicados à determinação do menor custo entre um ponto inicial e um ponto final em mapas bidimensionais. O problema considera diferentes tipos de terreno, cada um associado a um custo específico, bem como obstáculos intransponíveis, permitindo apenas movimentos ortogonais. Foram implementados os algoritmos de Dijkstra, Bellman-Ford e Floyd-Warshall, seguindo fielmente os pseudocódigos apresentados em sala de aula e na especificação deste trabalho. Além disso, foi desenvolvido um módulo de experimentos computacionais que executa múltiplas rodadas por mapa, coletando métricas de tempo de execução e custo do caminho, possibilitando uma análise empírica do comportamento dos algoritmos em diferentes cenários.

1 Introdução

O problema de determinação de caminhos mínimos em grafos é amplamente estudado na área de algoritmos e estruturas de dados, possuindo aplicações práticas em roteamento de redes, sistemas de navegação, planejamento de trajetos e análise de mapas. Neste contexto, o presente trabalho prático tem como objetivo implementar, analisar e comparar algoritmos clássicos de caminhos mínimos aplicados a mapas bidimensionais com terrenos ponderados e obstáculos.

O problema consiste em encontrar o menor custo total entre um ponto inicial (I) e um ponto final (F), considerando custos distintos associados aos diferentes tipos de terreno e restrições de movimentação impostas por obstáculos. Para tal, o mapa bidimensional é modelado como um grafo direcionado e ponderado, permitindo a aplicação direta dos algoritmos estudados.

Foram implementados os algoritmos de **Dijkstra**, **Bellman-Ford** e **Floyd-Warshall**, seguindo rigorosamente os pseudocódigos apresentados em sala de aula. Adicionalmente, foi desenvolvido um módulo específico para experimentos computacionais, possibilitando a execução repetida dos algoritmos e a coleta sistemática de métricas de desempenho, conforme especificado no enunciado do trabalho.

2 Modelagem do Problema

O mapa de entrada é representado como uma matriz bidimensional de caracteres, na qual cada célula corresponde a um tipo de terreno ou a um obstáculo intransponível. Os símbolos utilizados possuem os seguintes significados: `I` representa o ponto inicial, `F` o ponto final, `#` um obstáculo, e `G`, `S` e `W` representam terrenos com custos distintos.

Os custos associados aos terrenos foram definidos da seguinte forma:

- `G = 1`
- `S = 3`
- `W = 5`
- `I` e `F = 0`

O objetivo do problema é determinar o caminho de menor custo acumulado entre `I` e `F`, permitindo apenas movimentos ortogonais (cima, baixo, esquerda e direita), respeitando as restrições impostas pelos obstáculos.

3 Conversão do Mapa em Grafo

Para possibilitar a aplicação dos algoritmos de caminhos mínimos, o mapa bidimensional é convertido em um grafo direcionado e ponderado. Cada célula válida do mapa é mapeada para um vértice único por meio da fórmula:

$$v = i \times colunas + j \tag{1}$$

Esse mapeamento transforma uma estrutura bidimensional em uma representação linear unidimensional, permitindo que cada posição do mapa seja identificada por um único índice inteiro. A conversão é determinística e reversível, garantindo que cada vértice corresponda exatamente a uma célula do mapa e vice-versa. Além disso, essa abordagem permite numerar os vértices de forma contínua, variando de 0 até ($linhas \times colunas - 1$), o que simplifica a implementação dos algoritmos e das estruturas de dados do grafo.

Arestas são criadas entre vértices correspondentes a células adjacentes, desde que a célula vizinha não seja um obstáculo. São considerados apenas movimentos ortogonais (cima, baixo, esquerda e direita). O peso de cada aresta é definido como o custo do terreno da célula de destino, modelando o custo de *entrar* naquela posição. Dessa forma, o custo total de um caminho corresponde exatamente à soma dos custos dos terrenos efetivamente percorridos, refletindo fielmente o problema original.

4 Estrutura de Dados Utilizada

Foi adotada a estrutura de **lista de adjacências** para a representação do grafo. Essa escolha se justifica pelo fato de o grafo gerado a partir do mapa ser naturalmente esparso, uma vez que cada vértice possui, no máximo, quatro vizinhos correspondentes aos movimentos ortogonais permitidos.

O uso de listas de adjacências reduz significativamente o consumo de memória em comparação com uma matriz de adjacências, além de tornar mais eficiente a iteração sobre os vértices vizinhos, operação frequentemente realizada pelos algoritmos de caminhos mínimos. Essa escolha mostrou-se fundamental para viabilizar a execução dos experimentos em mapas de maior dimensão.

5 Implementação

O programa foi estruturado em módulos, visando clareza e organização:

- **Mapa.py**: leitura do arquivo de mapa e conversão para grafo;
- **Grafo.py**: definição das estruturas de dados do grafo;

- **Algoritmos.py**: implementação dos algoritmos de caminhos mínimos;
- **main.py**: execução simples (1 rodada) e geração dos mapas de saída;
- **main_benchmark.py**: execução dos experimentos com limite de tempo;
- **main_benchmark_v2.py**: execução dos experimentos sem limite de tempo (sem Floyd-Warshall).

Cada algoritmo foi executado de forma independente, com medição de tempo utilizando `time.perf_counter()`, permitindo a coleta precisa das métricas analisadas.

6 Metodologia Experimental

Para cada mapa analisado, foi adotada a seguinte metodologia:

- Execução dos algoritmos **Dijkstra**, **Bellman-Ford** e **Floyd-Warshall**;
- Cada algoritmo foi executado **10 vezes por mapa**;
- Coleta de:
  - tempo de execução (segundos);
  - custo do caminho mínimo entre `I` e `F`.
- Cálculo das médias de tempo e custo a partir das 10 execuções;
- Quando aplicável, registro de **TEMPO LIMITE** em caso de execução superior ao limite configurado;
- No Benchmark V2, **sem timeout** e **sem Floyd-Warshall**, visando observar Dijkstra e Bellman-Ford sem interrupções artificiais.

7 Ambientes de Execução

Os experimentos foram realizados em três máquinas distintas, permitindo análise comparativa entre ambientes. As especificações completas de todos os Ambientes e os logs correspondentes encontram-se anexados ao relatório.

7.1 Ambiente A (macOS)

Os experimentos do Ambiente A foram executados no seguinte hardware/software:

- Sistema Operacional: macOS Sonoma 14.8.1
- Processador: Intel Core i7 2,6 GHz (6 núcleos)
- Memória: 16 GB DDR4 2400 MHz

7.2 Ambiente B (WindowsA)

Os experimentos do Ambiente B foram executados no seguinte hardware/software:

- Sistema Operacional: Windows 11
- Processador: Intel(R) Core(TM) i5 2.10 GHz (8 núcleos)
- Memória: 16 GB DDR5

7.3 Ambiente C (WindowsB)

Os experimentos do Ambiente C foram executados no seguinte hardware/software:

- Sistema Operacional: Windows 11
- Processador: Intel(R) Core(TM) i7 2.80GHz (8 núcleos)
- Memória: 32 GB DDR4

7.4 Execução Simples (main.py)

Na execução simples, o programa executa Dijkstra, Bellman-Ford e Floyd-Warshall uma vez para o mapa informado, imprimindo custo e tempo no terminal e gerando arquivos de saída com o caminho marcado.

7.5 Benchmark com Timeout (main_benchmark.py)

No benchmark com timeout, cada algoritmo é executado 10 vezes por mapa. Em caso de execução superior ao limite estabelecido, o resultado é registrado como **TEMPO LIMITE**. Esse cenário é alinhado ao enunciado do trabalho e permite comparar comportamento prático sob restrições.

7.6 Benchmark V2 Sem Timeout (main_benchmark_v2.py)

No benchmark V2, o algoritmo de Floyd-Warshall foi removido e não há limite de tempo por execução. O objetivo foi observar o comportamento real de Dijkstra e Bellman-Ford em mapas grandes sem interrupções artificiais. Em alguns ambientes, especialmente no Windows, instâncias maiores podem falhar por memória (*MemoryError*) dependendo da alocação interna e do sistema.

8 Resultados Experimentais

Os resultados experimentais evidenciaram diferenças claras de desempenho entre os algoritmos, principalmente à medida que o tamanho dos mapas aumentou.

Nos mapas de pequeno porte, todos os algoritmos foram capazes de encontrar corretamente o caminho mínimo, apresentando tempos de execução reduzidos e custos consistentes. Nesses casos, as diferenças de desempenho entre os algoritmos foram pouco significativas.

À medida que mapas de tamanho médio passaram a ser analisados, o tempo de execução do algoritmo de Dijkstra cresceu de forma mais acentuada, enquanto o Bellman-Ford manteve um comportamento mais regular. O algoritmo de Floyd-Warshall, por sua vez, passou a apresentar dificuldades para concluir sua execução dentro do tempo limite estabelecido.

Em mapas de grande porte, o algoritmo de Dijkstra tornou-se inviável devido à sua implementação didática, que não utiliza estruturas auxiliares para otimização. O Bellman-Ford, apesar de sua complexidade teórica mais elevada, continuou apresentando tempos aceitáveis. Já o Floyd-Warshall mostrou-se impraticável tanto em termos de tempo de execução quanto de consumo de memória, conforme previsto pela análise teórica de sua complexidade.

9 Análise Comparativa

De modo geral, os resultados obtidos confirmam a análise teórica de complexidade dos algoritmos estudados. No entanto, a análise empírica revelou comportamentos distintos em função das características específicas do problema e das implementações adotadas.

Embora, do ponto de vista teórico, o algoritmo de Bellman-Ford possua complexidade assintoticamente superior à de Dijkstra, os experimentos indicaram tempos menores para o Bellman-Ford em mapas de grande dimensão. Esse resultado está relacionado ao fato de o grafo gerado a partir do mapa ser altamente esparso, com grau máximo constante, além da implementação didática do Dijkstra, que realiza a seleção do próximo vértice de menor custo por meio de varredura linear.

Nesse contexto, o Bellman-Ford apresentou um crescimento mais previsível, enquanto o Dijkstra tornou-se inviável para instâncias maiores. O Floyd-Warshall, devido à sua complexidade cúbica, mostrou-se impraticável já em mapas de tamanho médio.

De forma resumida, observam-se os seguintes comportamentos:

- **Dijkstra**: apresenta crescimento expressivo do tempo de execução em mapas grandes, caracterizando comportamento próximo de $O(V^2)$;
- **Bellman-Ford**: apresentou crescimento mais previsível em grafos esparsos, mostrando-se mais viável nos experimentos realizados;
- **Floyd-Warshall**: revelou-se inviável em mapas médios e grandes devido à complexidade $O(V^3)$ e ao elevado consumo de memória.

Como o número de vértices cresce proporcionalmente à área do mapa ($V = linhas \times colunas$), o custo computacional dos algoritmos aumenta rapidamente à medida que mapas maiores são analisados.

10 Tabelas de Resultados Consolidados

As tabelas consolidadas solicitadas no enunciado com médias de tempo e custo por mapa e algoritmo.

10.1 Ambiente A [macOS]

10.1.1 Cenário 1: Com Timeout

Tabela 1: Comparação de Desempenho dos Algoritmos

Grafo	**Dijkstra**		**Bellman-Ford**		**Floyd-Warshall**	
	T.médio (s)	**Custo médio**	**T.médio (s)**	**Custo médio**	**T.médio (s)**	**Custo médio**
map_10x10	0.000441	4	0.000285	4	0.070568	4
map_10x20	0.001310	5	0.000262	5	0.521229	5
map_20x10	0.001208	4	0.001111	4	0.524994	4
map_20x20	0.006016	26	0.000457	26	4.099596	26
map_50x30	0.511423	44	0.050591	44	TIME OUT	TIME OUT
map_50x50	1.103297	37	0.035400	37	TIME OUT	TIME OUT
map_100x50	2.477946	139	0.313892	139	TIME OUT	TIME OUT
map_100x100	4.741751	211	0.123594	211	TIME OUT	TIME OUT
map_300x100	59.191273	49	1.889402	49	TIME OUT	TIME OUT
map_300x300	TIME OUT	TIME OUT	15.564049	460	TIME OUT	TIME OUT

10.1.2 Cenário 2: Sem Timeout

Tabela 2: Comparação de Desempenho dos Algoritmos

Grafo	Dijkstra		Bellman-Ford	
	T.médio (s)	Custo médio	T.médio (s)	Custo médio
map_10x10	0.000421	4	0.000269	4
map_10x20	0.001361	5	0.000249	5
map_20x10	0.001272	4	0.001109	4
map_20x20	0.006431	26	0.000456	26
map_50x30	0.526066	44	0.085048	44
map_50x50	1.363612	37	0.034352	37
map_100x50	1.333298	139	0.141513	139
map_100x100	5.238727	211	0.130145	211
map_300x100	60.169914	49	1.067940	49
map_300x300	673.420817	460	23.175675	460

10.2 Ambiente B [WindowsA]

10.2.1 Cenário 1: Com Timeout

Tabela 3: Comparação de Desempenho dos Algoritmos

Grafo	Dijkstra		Bellman-Ford		Floyd-Warshall	
	T.médio (s)	Custo médio	T.médio (s)	Custo médio	T.médio (s)	Custo médio
map_10x10	0.000474	4	0.000265	4	0.060079	4
map_10x20	0.001484	5	0.000215	5	0.472516	5
map_20x10	0.001771	4	0.001045	4	0.472380	4
map_20x20	0.009476	26	0.000602	26	4.099596	26
map_50x30	0.511423	44	0.050591	44	TIME OUT	TIME OUT
map_50x50	1.103297	37	0.035400	37	TIME OUT	TIME OUT
map_100x50	1.253544	139	0.117066	139	TIME OUT	TIME OUT
map_100x100	4.867780	211	0.106941	211	TIME OUT	TIME OUT
map_300x100	59.191273	49	1.889402	49	TIME OUT	TIME OUT
map_300x300	TIME OUT	TIME OUT	8.216189	460	TIME OUT	TIME OUT

10.3 Ambiente C [WindowsB]

10.3.1 Cenário 1: Com Timeout

Tabela 4: Comparação de Desempenho dos Algoritmos

Grafo	**Dijkstra**		**Bellman-Ford**		**Floyd-Warshall**	
	T.médio (s)	**Custo médio**	**T.médio (s)**	**Custo médio**	**T.médio (s)**	**Custo médio**
map_10x10	0.000441	4	0.000285	4	0.070568	4
map_10x20	0.001310	5	0.000262	5	0.521229	5
map_20x10	0.001208	4	0.001111	4	0.524994	4
map_20x20	0.006016	26	0.000457	26	4.099596	26
map_50x30	0.511423	44	0.050591	44	TIME OUT	TIME OUT
map_50x50	1.103297	37	0.035400	37	TIME OUT	TIME OUT
map_100x50	2.477946	139	0.313892	139	TIME OUT	TIME OUT
map_100x100	4.741751	211	0.123594	211	TIME OUT	TIME OUT
map_300x100	59.191273	49	1.889402	49	TIME OUT	TIME OUT
map_300x300	TIME OUT	TIME OUT	15.564049	460	TIME OUT	TIME OUT

10.3.2 Cenário 2: Sem Timeout

Tabela 5: Comparação de Desempenho dos Algoritmos

Grafo	**Dijkstra**		**Bellman-Ford**	
	T.médio (s)	**Custo médio**	**T.médio (s)**	**Custo médio**
map_10x10	0.000651	4	0.000525	4
map_10x20	0.001942	5	0.000789	5
map_20x10	0.002167	4	0.002062	4
map_20x20	0.011235	26	0.000676	26
map_50x30	0.175849	44	0.032938	44
map_50x50	0.470520	37	0.066566	37
map_100x50	1.934977	139	0.214881	139
map_100x100	6.891802	211	0.211772	211
map_300x100	89.071553	49	1.337910	49
map_300x300	970.559414	460	11.268442	460

11 Análise Comparativa

Os resultados observados confirmam, de modo geral, a análise teórica de complexidade dos algoritmos estudados. Entretanto, a análise empírica evidenciou comportamentos distintos em função das características específicas do problema e das implementações adotadas.

Apesar de, do ponto de vista teórico, o algoritmo de Bellman-Ford apresentar complexidade assintoticamente superior à de Dijkstra, os resultados experimentais indicaram tempos menores em mapas de grande dimensão. Esse comportamento está associado às características do grafo gerado a partir do mapa, que é altamente esparso e possui grau máximo constante, bem como à implementação didática do algoritmo de Dijkstra, que realiza a seleção do próximo vértice de menor custo por meio de varredura linear. Nessas condições, o Bellman-Ford apresentou crescimento mais previsível, enquanto o Dijkstra tornou-se inviável para instâncias maiores.

De forma mais específica, observam-se os seguintes comportamentos:

- **Dijkstra**: apresenta crescimento expressivo em mapas grandes devido à seleção do menor custo por varredura linear, caracterizando comportamento próximo de $O(V^2)$;
- **Bellman-Ford**: apresenta crescimento mais previsível em grafos esparsos, com complexidade $O(V \cdot E)$, mostrando-se mais viável nas instâncias analisadas;
- **Floyd-Warshall**: torna-se inviável em mapas médios e grandes devido à complexidade cúbica $O(V^3)$ e ao elevado consumo de memória decorrente do uso de estruturas matriciais.

O mapeamento mapa $\rightarrow$ grafo implica que o número de vértices cresce proporcionalmente à área do mapa, $V = linhas \times colunas$, amplificando de forma significativa o custo computacional dos algoritmos à medida que o tamanho do mapa aumenta.

12 Limitações Observadas

Foram identificadas limitações práticas em mapas de grande porte:

- O Floyd-Warshall é impraticável para instâncias grandes, tanto por tempo quanto por memória.
- Dijkstra, sem estruturas auxiliares como fila de prioridade, não escala adequadamente.
- Em alguns ambientes, pode ocorrer *MemoryError* em mapas grandes, dependendo do padrão de alocação e limites do sistema.

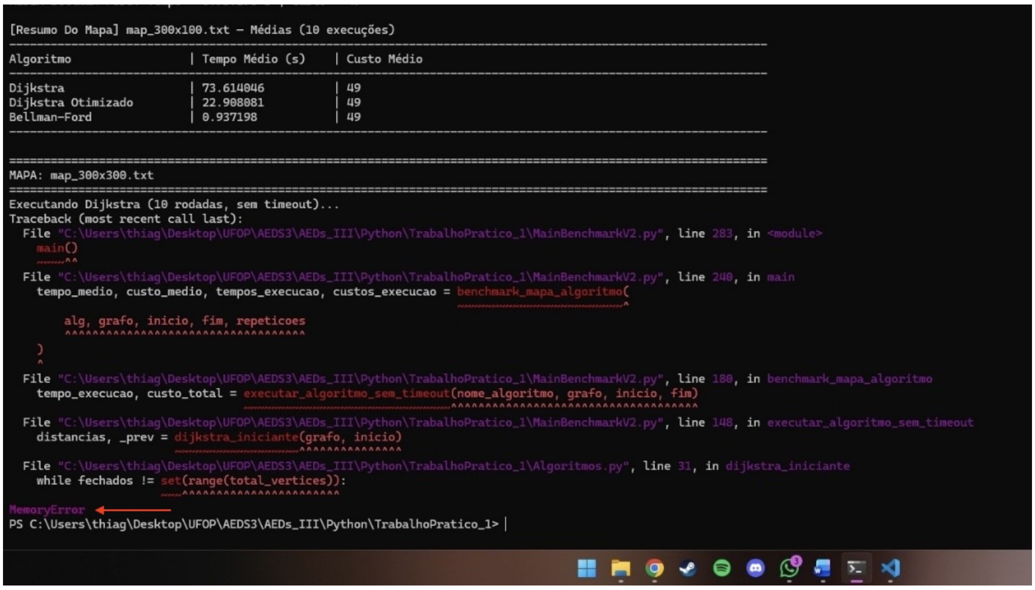

Figura 1: Exemplo de erro de memória (*MemoryError*) durante a execução de algoritmos em mapas de grande porte.

13 Conclusão

Os experimentos realizados permitiram observar, de forma clara, o impacto do tamanho do grafo no desempenho dos algoritmos de caminhos mínimos. O Bellman-Ford apresentou o melhor equilíbrio entre clareza didática e desempenho prático, mantendo-se viável mesmo em mapas de maior dimensão.

O algoritmo de Dijkstra, na implementação fiel ao pseudocódigo e sem o uso de estruturas auxiliares, apresentou crescimento acentuado no tempo de execução, tornando-se inadequado para mapas extensos. Já o Floyd-Warshall, embora correto do ponto de vista conceitual, mostrou-se impraticável para mapas médios e grandes, tanto em termos de tempo de execução quanto de consumo de memória, conforme previsto por sua complexidade teórica.

De modo geral, os resultados obtidos reforçam os conceitos estudados na disciplina, destacando a importância da modelagem adequada do problema, da escolha correta das estruturas de dados e da análise da complexidade algorítmica na construção de soluções eficientes.

Referências

Todo o conteúdo teórico, conceitual e metodológico utilizado neste trabalho foi fundamentado no material disponibilizado pelo docente da disciplina, incluindo:

- Slides e apresentações utilizados em sala de aula;

- Codigo fonte dos Grafos e orientações fornecidas pelo professor em aula.

Os algoritmos abordados (Dijkstra, Bellman-Ford e Floyd-Warshall) foram estudados e aplicados conforme a abordagem apresentada na disciplina.