Projeto AED - O TAD GRAPH

Henrique Lopes 119954
Eduardo Rosário 119234

2024/2025 - 1º Semestre

---

1 Introdução

O presente projeto tem como objetivo implementar e analisar algoritmos relacionados à manipulação de grafos utilizando o Tipo Abstrato de Dados (TAD) Graph. Este projeto faz parte da cadeira de **Algoritmos e Estruturas de Dados** e foca em desenvolver soluções eficientes para problemas clássicos em grafos, como encontrar caminhos mais curtos e determinar o fecho transitivo de um grafo orientado.

1.1 O que é um TAD Graph?

Um TAD Graph é uma abstração utilizada para representar e manipular grafos de forma eficiente e estruturada. No contexto deste projeto, o TAD Graph é implementado utilizando uma lista de vértices, onde cada vértice contém uma lista de adjacências que representa as arestas incidentes. Esse modelo é especialmente útil para representar tanto grafos orientados quanto não orientados, com ou sem pesos nas arestas.

O TAD Graph disponibiliza operações básicas para manipulação de grafos, como adicionar vértices e arestas, e operações de consulta sobre os elementos do grafo. Outros algoritmos mais complexos são desenvolvidos de forma modular, utilizando o TAD como base.

1.2 Algoritmos Implementados

Neste trabalho, foram implementados dois algoritmos principais:

- **Bellman-Ford:** Algoritmo clássico para encontrar os caminhos mais curtos a partir de um vértice inicial em grafos orientados sem pesos. Ele é utilizado para verificar a alcançabilidade de vértices e calcular distâncias mínimas.
- **Fecho Transitivo:** Algoritmo que, utilizando o Bellman-Ford como base, constrói o fecho transitivo de um grafo orientado. O fecho transitivo adiciona arestas entre todos os pares de vértices onde exista um caminho no grafo original.

1.3 Abordagem no Relatório

No relatório, serão apresentados os seguintes tópicos:

- Descrição detalhada e análise dos algoritmos implementados.
- Discussão sobre a complexidade computacional de cada algoritmo, com foco em tempo e espaço.
- Apresentação de resultados experimentais obtidos por meio de testes realizados com grafos de diferentes tamanhos e densidades.
- Conclusões sobre a eficiência dos algoritmos e a adequação das soluções implementadas.

Por meio deste trabalho, espera-se validar a implementação do TAD Graph e dos algoritmos propostos, além de compreender as implicações práticas de sua complexidade em diferentes cenários.

---

2 Algoritmos

2.1 Bellman-Ford

2.1.1 Descrição

O algoritmo de Bellman-Ford tem como objetivo encontrar os caminhos mais curtos a partir de um vértice inicial em um grafo. Ele é especialmente útil para grafos com pesos negativos, desde que não existam ciclos negativos acessíveis a partir do vértice inicial. No entanto, neste projeto a implementação do Algoritmo é feita para grafos sem pesos associados às arestas.

O funcionamento do algoritmo pode ser descrito em duas etapas principais:

1. **Inicialização:** Todos os vértices do grafo têm as suas distâncias iniciais definidas como infinito (representado como -1 no programa), exceto o vértice inicial, cuja distância é 0. O predecessor de cada vértice também é inicializado como -1.

2. **Relaxação de arestas:** Para cada aresta (u, v), verifica-se se a distância para v pode ser reduzida ao passar por u. Caso positivo, atualiza-se a distância e o predecessor de v.

2.2 Complexidade

O algoritmo `GraphBellmanFordAlgExecute` possui uma complexidade que varia dependendo do caso analisado: melhor caso, caso médio e pior caso. Abaixo, apresentamos a análise detalhada de sua complexidade.

2.2.1 Complexidade por Caso

- **Melhor Caso:** No melhor caso, o grafo não tem vértices adjacentes, ou então, o vértice inicial não tem vértices adjacentes. Logo apenas serão feitas operações na inicialização.

  Iterações:

  - Inicialização: V
  - Relaxação: 0
  - **Total**: V

  Complexidade: $O(V)$.

  Demonstração:

  $$\sum_{i=1}^{V} 1 = V$$

  Grafo correspondente ao melhor caso:

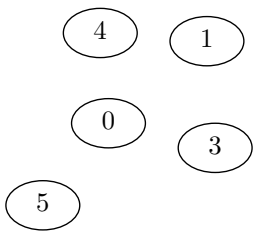

Figure 1: Caption

- **Pior Caso:** No pior caso, o grafo vai ser um grafo completo (cada vértice tem V - 1 arestas). O programa vai fazer a inicialização e vai fazer duas iterações do loop principal. Na primeira iteração, o algoritmo vai apenas relaxar as arestas do vértice inicial, como não há pesos associados às arestas este vai ser o caminho mais curto para todos os vértices. Na segunda iteração, o algoritmo vai relaxar as arestas de todos os vértices e como as distâncias não vão ser atualizadas o algoritmo vai parar o loop e terminar a sua execução.

  Iterações:

  - Inicialização: V.

- Primeira Iteração: V - 1
- Segunda Iteração: V · (V - 1).
- **Total**: $\mathbf{V^2}$ + V - 1

Complexidade: $O(V^2)$.

Demonstração:

$$\sum_{i=1}^{V} 1 + \sum_{i=1}^{1} \sum_{i=1}^{V-1} 1 + \sum_{i=1}^{V} \sum_{i=1}^{V-1} 1 = V + V - 1 + V \cdot (V - 1) = V^2 - V + 2V - 1 = V^2 + V - 1$$

Grafo correspondente ao pior caso:

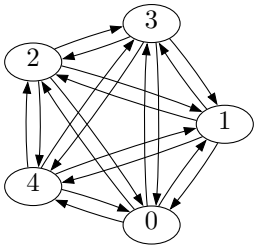

Figure 2: Caption

2.2.2 Resumo da Complexidade

Caso	Complexidade
Melhor Caso	$O(V)$
Pior Caso	$O(V^2)$

Table 1: Resumo da Complexidade do Algoritmo Bellman-Ford

2.2.3 Testes

Melhor Caso:

Num Vértices	Num Arestas	Iterações
3	0	3
4	0	4
5	0	5
6	0	6
9	0	7
8	0	8
9	0	9

Table 2: Testes do melhor caso Algoritmo Bellman-Ford

Pior Caso (digrafo):

Num Vértices	Num Arestas	Iterações
3	6	11
4	12	19
5	20	29
6	30	41
9	42	55
8	56	71
9	72	89

Table 3: Testes do melhor caso Algoritmo Bellman-Ford

2.3 Fecho Transitivo

2.3.1 Descrição

O algoritmo de construção do fecho transitivo tem como objetivo determinar a conectividade em um grafo orientado. Para isso, ele adiciona arestas entre vértices onde existe um caminho no grafo original. Em outras palavras, se um vértice u pode alcançar um vértice v, o fecho transitivo incluirá uma aresta $u \to v$.

O funcionamento do algoritmo pode ser descrito em duas etapas principais:

1. **Para cada vértice do grafo:** O algoritmo utiliza o Bellman-Ford para determinar todos os vértices alcançáveis a partir do vértice de origem u.
2. **Adicionação de arestas:** Para cada vértice alcançável v, uma aresta $u \to v$ é adicionada ao grafo do fecho transitivo.

2.4 Complexidade

O algoritmo `GraphComputeTransitiveClosure` possui uma complexidade que varia dependendo do caso analisado: melhor caso e pior caso. Abaixo, apresentamos a análise detalhada de sua complexidade.

Como o caso analisado corresponde ao caso do algoritmo de Bellman-Ford para facilitar os cálculos iremos usar o diretamente o número de iterações calculado no ponto anterior.

2.4.1 Complexidade por Caso

- Formula Geral

$$\sum_{i=1}^{V}(\text{Num\_Iterations\_BellmanFord} + \sum_{i=1}^{V} 1)$$

- **Melhor Caso:**

  Iterações:

  - Execução do Bellman-Ford: V
  - **Total:** $2V^2$

  Complexidade. $O(V^2)$

  Demonstração:

$$\sum_{i=1}^{V}(V + \sum_{i=1}^{V} 1) = \sum_{i=1}^{V} V + \sum_{i=1}^{V}\sum_{i=1}^{V} 1 = V^2 + V^2 = 2V^2$$

- **Pior Caso:**

  - Execução do Bellman-Ford: $V^2 + V - 1$
  - **Total:** $V^3 + 2V^2 - V$.

  Complexidade. $O(V^3)$

  Demonstração:

$$\sum_{i=1}^{V}(V^2 + V - 1 + \sum_{i=1}^{V} 1) = \sum_{i=1}^{V} V^2 + \sum_{i=1}^{V} V + \sum_{i=1}^{V}(-1) + \sum_{i=1}^{V}\sum_{i=1}^{V} 1 = V^3 + V^2 - V + V^2 = V^3 + 2V^2 - V$$

2.4.2 Resumo da Complexidade

Caso	Complexidade
Melhor Caso	$O(V^2)$
Pior Caso	$O(V^3)$

Table 4: Resumo da Complexidade do Algoritmo de Fecho Transitivo

2.4.3 Testes

Melhor Caso:

Num Vértices	Num Arestas	Iterações
3	0	18
4	0	32
5	0	50
6	0	72
9	0	98
8	0	128
9	0	162

Table 5: Testes do melhor caso Algoritmo Transitive Closure

Pior Caso:

Num Vértices	Num Arestas	Iterações
3	6	42
4	12	92
5	20	170
6	30	282
9	42	434
8	56	632
9	72	882

Table 6: Testes do melhor caso Algoritmo Transitive Closure

3 Representação gráfica:

As seguintes representações gráficas comparam os casos (melhor, pior e médio) dos algoritmos de Bellman-Ford e Transitive Closure, respetivamente. Para o caso médio foram usados grafos onde foram adicionadas arestas aleatoriamente e o número total de arestas varia entre 1 e V - 1, sendo V o número de vértices.

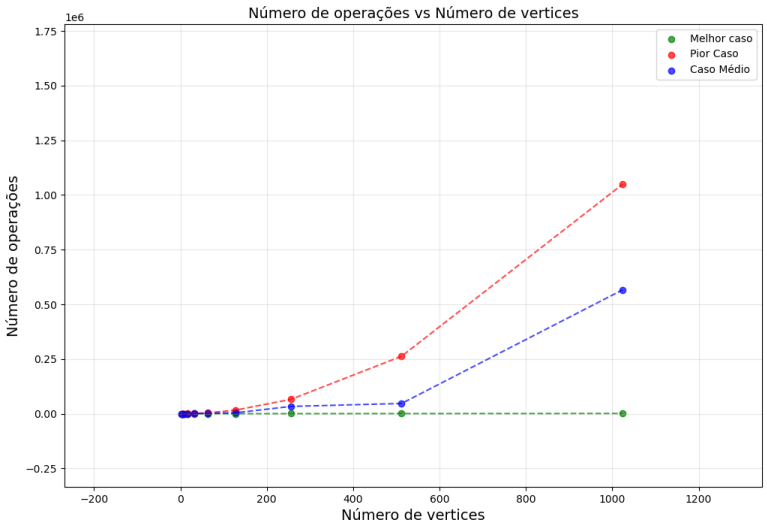

Figure 3: Representação gráfica do algoritmo de Bellman-Ford

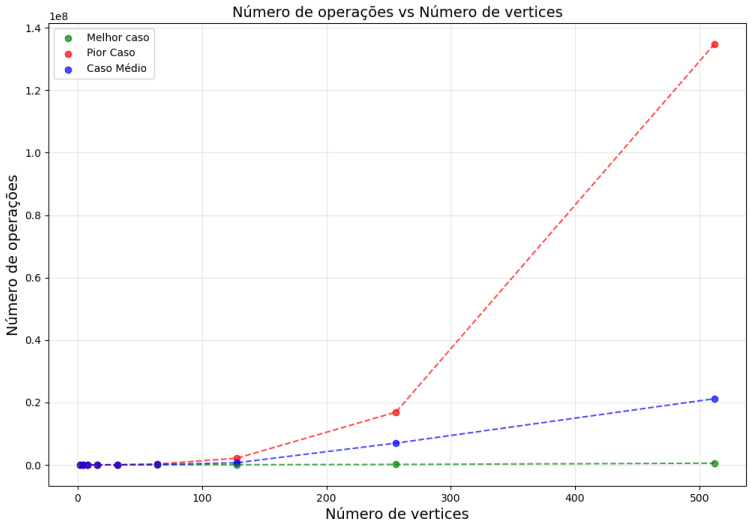

Figure 4: Representação gráfica do algoritmo de Transitive Closure