Projeto AED - O TAD GRAPH

Pedro Pinto nº115304 ; João Pinto nº104384

---

1. Introdução

No contexto da disciplina de Algoritmos e Estruturas de Dados, este projeto conduziu ao desenvolvimento do Tipo Abstrato de Dados (TAD) *Graph*. Para além disso, desenvolvemos 3 algoritmos para determinar a ordenação topológica dos vértices de um grafo orientado, caso exista. Para analisar a complexidade computacional dos algoritmos desenvolvidos, optou-se por desenvolver mecanismos de teste simples e eficientes. Para testar os algoritmos, basta executar o ficheiro *main.m* no diretório *testTopo*. Este script *Matlab* possibilita diversos testes ajustáveis, alterando apenas algumas constantes. O script inicia executando um ficheiro *shell*, *execute_ topoTest.sh*, que compila e executa testes no ficheiro em C, *GraphTestTopo.c*. Este ficheiro realiza testes sobre *GraphTopologicalSorting.c*, gerando e processando diferentes grafos. Esses grafos são gerados a partir do script *GraphGenerator.c*. Após a execução, o script *shell* escreve os resultados no ficheiro *data_ topoTests.txt*. Esses resultados são lidos e processados pelo script *Matlab*, fornecendo gráficos conforme solicitado. Este módulo de teste (Figura 1) permite uma análise robusta e abrangente do desempenho do TAD *Graph* quando aplicados os algoritmos de ordenação topológica.

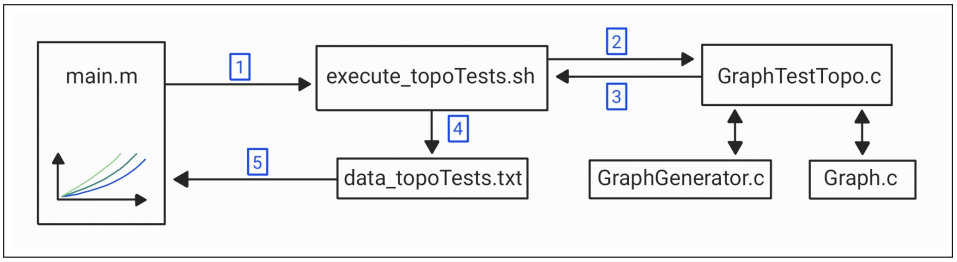
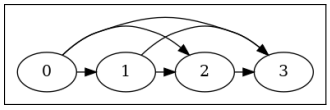


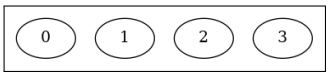
Figura 1. Módulo de testes desenvolvido para as ordenações topológicas.

2. Análise da complexidade dos 3 algoritmos desenvolvidos

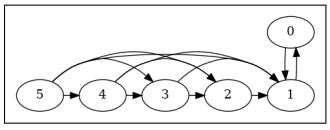
No ficheiro *GraphTopologicalSorting.c* estão presentes os 3 algoritmos desenvolvidos para ordenação topológica. Para analisar estes algoritmos optámos por analisar os melhores e piores casos, divididos em duas secções: **Sucesso** e **Insucesso**. Para isso tivemos que criar um ficheiro responsável por gerar grafos, *GraphGenerator.c*, sendo constituído por 4 funções que irão gerar cada caso consoante o número de vértices. Em baixo apresentamos exemplos de grafos que representam cada uma das situações.



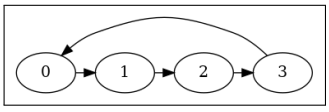
(a) Grafo 1 - Sucesso (Pior caso).



(b) Grafo 2 - Sucesso (Melhor caso).



(c) Grafo 3 - Insucesso (Pior caso).



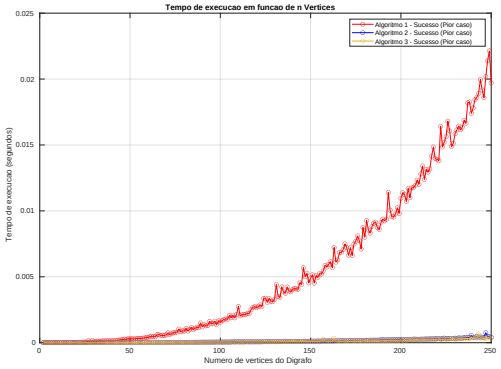
(d) Grafo 4 - Insucesso (Melhor caso).

Figura 2. Exemplos de 4 grafos que representam as diferentes situações analisadas ao longo do relatório.

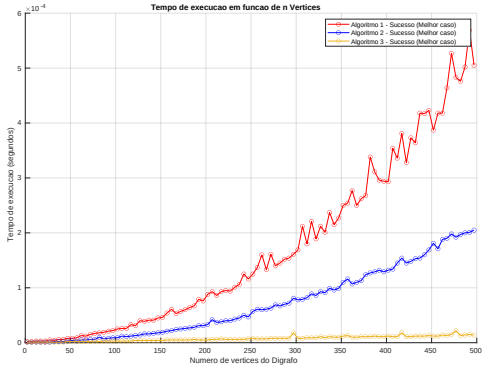
O grafo da Figura 2a é um grafo direcionado acíclico (DAG) que apresenta o maior número de arestas possível com V vértices satisfazendo a condição de ter ordenação topológica, sendo por isso o caso de sucesso mais demorado - Sucesso: (Pior caso). O grafo da Figura 2b não tem nenhuma aresta, sendo por isso o melhor caso possível, satisfazendo a condição de ter ordenação topológica, uma vez que não têm de ser realizadas eliminações - Sucesso: (Melhor caso). O grafo da Figura 2c representa o caso Insucesso: (Pior caso) e trata-se de uma "cópia

invertida" do grafo da Figura 2a mas não satisfazendo a condição pois criámos um subgrafo cíclico que contém os dois vértices de menor índice. Para que o número de sucessivas procuras seja o maior possível é preferível não contabilizar os dois vértices de menor índice do que dois de maior índice por isso optámos por estabelecer um ciclo entre estes dois vértices e fazer uma "cópia invertida" para ter o maior número de iterações. O grafo da Figura 2d representa o caso Insucesso: (Melhor caso) uma vez que o grafo tem um ciclo contendo todos os vértices com o menor número de arestas possível (V arestas). Como o algoritmo 1 necessita de fazer uma cópia do grafo considerámos aqui o menor número de arestas possível para termos o melhor caso possível em todos os algoritmos.

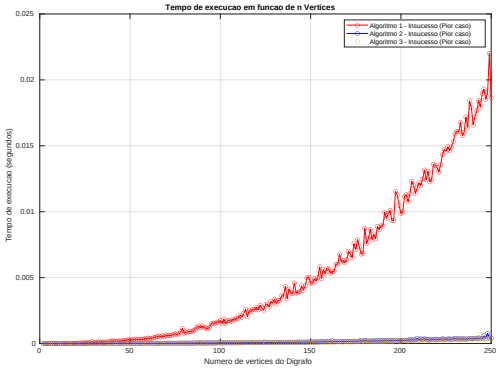
Tal como referido na introdução do relatório utilizámos o script *Matlab* para gerar os gráficos que nos permitem comparar os 3 algoritmos desenvolvidos para cada situação. Na Figura 3 apresentamos os tempos de execução em função do número de vértices para os 3 algoritmos para os 4 diferentes casos. Na realização destes testes comentámos os *assert(GraphCheckInvariants(g))* para não condicionarem o tempo de execução.



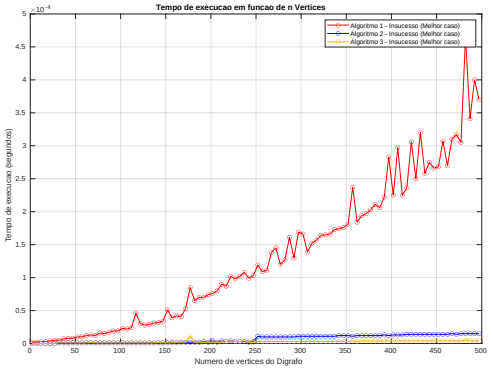
(a) Sucesso (Pior caso).



(b) Sucesso (Melhor caso).



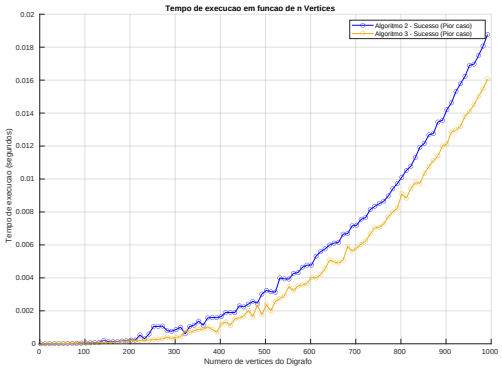
(c) Insucesso (Pior caso).



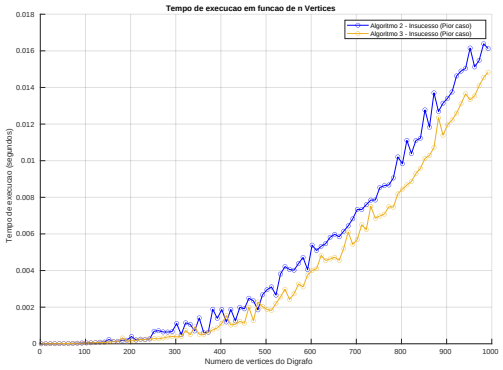
(d) Insucesso (Melhor caso).

Figura 3. Tempos de execução para os três algoritmos - 4 casos de estudos.

Observando os gráficos da Figura 3, conseguimos claramente identificar um grande contraste entre o tempo de execução do algoritmo 1 e do algoritmo 3, sendo o algoritmo 3 uma versão muito mais rápida em função do número de vértices. Além disso, observando diretamente a Figura 3b conseguimos concluir que, embora nas outras três figuras os algoritmos 2 e 3 sejam bastante parecidos, o algoritmo 2 tem um pior desempenho nesta situação. No entanto, para ficar claro que o algoritmo 3 tem um menor tempo de execução que o algoritmo 2, apresentamos na Figura 4 uma comparação entre esses dois algoritmos para os casos Sucesso/Insucesso: (Pior caso).



(a) Sucesso (Pior caso).



(b) Insucesso (Pior caso).

Figura 4. Comparação entre o algoritmo 2 e o algoritmo 3.

De forma a justificarmos os tempos obtidos decidimos analisar o número de iterações em cada algoritmo. Para analisar estes três algoritmos tivemos que associar um peso (médio ou estimado) de cada iteração, este peso serve para estabelecer uma relação de aproximação entre tempo de execução e número de iterações. Estes pesos foram definidos nas operações mais demoradas dos diferentes algoritmos. No algoritmo 1 definimos para a função *GraphCopy* a aproximação $\approx \frac{V^2}{2}+E$, para as sucessivas procuras através do conjunto de vértices atribuímos um peso unitário e para a remoção utilizando *GraphRemoveEdge* considerámos o caso médio $\frac{V-1}{2} \approx \frac{V}{2}$ para V elevado. Para os algoritmos 2 e 3 considerámos para a inicialização das respetivas estruturas (array/fila) V iterações. Para as sucessivas procuras através do conjunto de vértices e para as remoções (decrementar valores de inDegree/adicionar à fila) atribuímos um peso unitário.

Assim chegámos às expressões para o número de iterações de cada algoritmo presentes na Tabela 1. Estas fórmulas foram deduzidas analisando o código e estão todas em função de V (número de vértices), pois mesmo o número de arestas (nestes casos) consegue ser deduzido em função de V. Achámos demasiado complexo explicar todas as fórmulas ao pormenor, apresentamos apenas as suas simplificações. Todas as expressões apresentadas na Tabela 1 foram comparadas com os valores práticos obtidos com o contador ITERATIONS e cujos gráficos estão representados na Figura 5.

Considerando os pesos por iteração e a sua respetiva análise para cada algoritmo, conseguimos chegar a um majorante e a um minorante do número de iterações, ver Tabela 2. Nesta Tabela apresentamos a ordem de complexidade dos 3 algoritmos. A dedução de todas as expressões encontram-se a seguir às Tabelas 1 e 2.

Análise das Iterações: Sucesso - consegue fazer a ordenação topológica. V - número de vértices do grafo.

Iterações	**Sucesso**		**Insucesso**	
	Pior Caso	Melhor Caso	Pior Caso	Melhor Caso
Alg. 1	$\frac{V^3}{4}+\frac{5V^2}{4}+V$	$V^2+\frac{3V}{2}$	$\frac{V^3}{4}+\frac{3V^2}{4}+\frac{V}{2}$	$\frac{V^2}{2}+2V$
Alg. 2	V^2+2V	$\frac{1}{2}V^2+\frac{5}{2}V$	V^2+V-2	$2V$
Alg. 3	$\frac{1}{2}V^2+\frac{3}{2}V$	$2V$	$\frac{V^2}{2}+\frac{V}{2}-1$	V

Tabela 1. Comparação dos três algoritmos relativamente ao número de iterações.

Iterações	**Sucesso**		**Insucesso**	
	Pior Caso	Melhor Caso	Pior Caso	Melhor Caso
Alg. 1	$\mathcal{O}(V^3)$	$\Omega(V^2)$	$\mathcal{O}(V^3)$	$\Omega(V^2)$
Alg. 2	$\mathcal{O}(V^2)$	$\Omega(V^2)$	$\mathcal{O}(V^2)$	$\Omega(V)$
Alg. 3	$\mathcal{O}(V^2)$	$\Omega(V)$	$\mathcal{O}(V^2)$	$\Omega(V)$

Tabela 2. Comparação da complexidade dos três algoritmos relativamente ao número de iterações.

Apresentamos a dedução das expressões das Tabelas 1 e 2 para os quatro casos analisados:

Sucesso (Pior Caso):

- Algoritmo 1 : $\frac{V^2}{2} + \sum_{x=1}^{V-1} x + \sum_{x=1}^{V} x + V + \sum_{x=1}^{V-1} x \times \frac{V}{2} = \frac{V^3}{4} + \frac{5V^2}{4} + V \implies \mathcal{O}(V^3)$
- Algoritmo 2 : $3V + 2\sum_{x=1}^{V-1} x = V^2 + 2V \implies \mathcal{O}(V^2)$
- Algoritmo 3 : $2V + \sum_{x=1}^{V-1} x = \frac{V^2}{2} + \frac{3V}{2} \implies \mathcal{O}(V^2)$

Sucesso (Melhor Caso):

- Algoritmo 1 : $\frac{V^2}{2} + \sum_{x=1}^{V} x + V = V^2 + \frac{3V}{2} \implies \Omega(V^2)$
- Algoritmo 2 : $2V + \sum_{x=1}^{V} x = \frac{V^2}{2} + \frac{5V}{2} \implies \Omega(V^2)$
- Algoritmo 3 : $V + V = 2V \implies \Omega(V)$

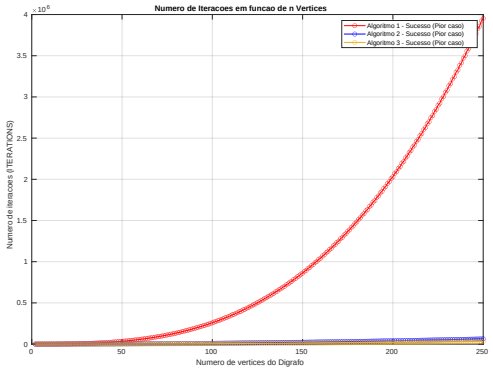
Insucesso (Pior Caso):

- Algoritmo 1 : $\frac{V^2}{2} + \sum_{x=1}^{V-2} x + 2 + \sum_{x=3}^{V} x + V + \left(\sum_{x=1}^{V-2} x\right) \times \frac{V}{2} = \frac{V^3}{4} + \frac{3V^2}{4} + \frac{V}{2} \implies \mathcal{O}(V^3)$
- Algoritmo 2 : $V + \sum_{x=3}^{V} x + V + \sum_{x=1}^{V-2} x = V^2 + V - 2 \implies \mathcal{O}(V^2)$
- Algoritmo 3 : $V + (V-2) + \sum_{x=1}^{V-2} x = \frac{V^2}{2} + \frac{V}{2} - 1 \implies \mathcal{O}(V^2)$

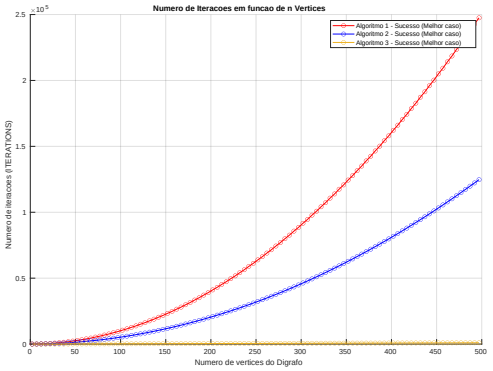
Insucesso (Melhor Caso):

- Algoritmo 1 : $\frac{V^2}{2} + 2V \implies \Omega(V^2)$
- Algoritmo 2 : $2V \implies \Omega(V)$
- Algoritmo 3 : $V \implies \Omega(V)$

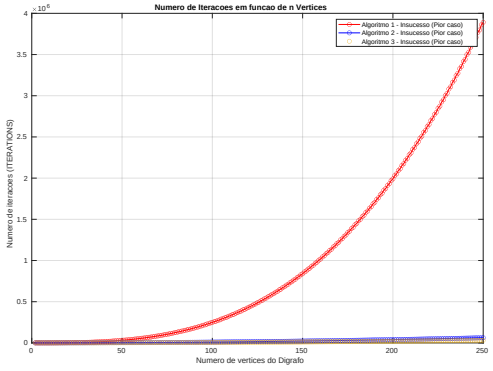
De forma a verificarmos as expressões presentes nas tabelas anteriores traçámos quatro gráficos, onde comparámos o número de iterações para os três algoritmos para os diferentes casos analisados.



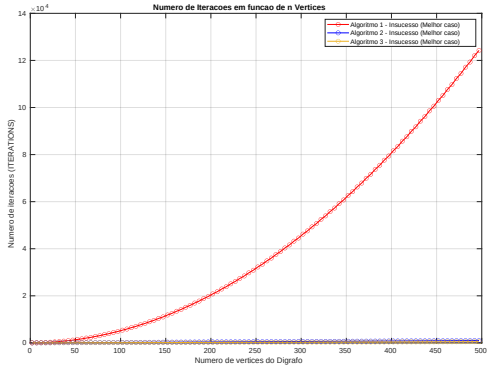
(a) Sucesso (Pior caso).



(b) Sucesso (Melhor caso).



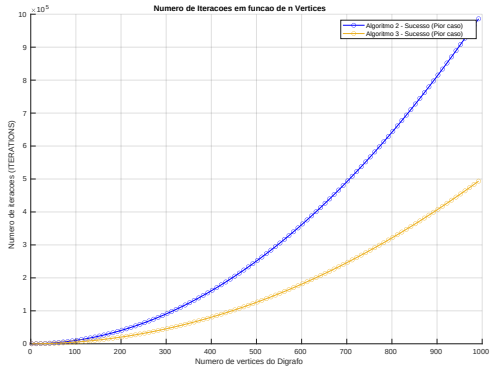
(c) Insucesso (Pior caso).



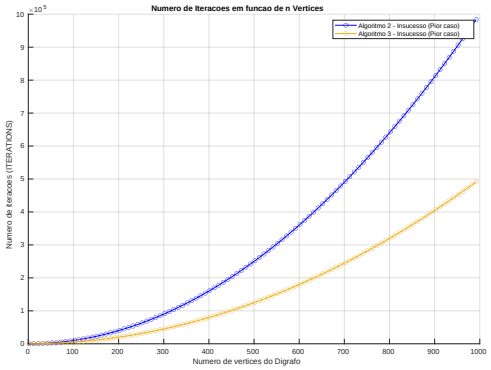
(d) Insucesso (Melhor caso).

Figura 5. Número de iterações para os três algoritmos - 4 casos de estudos.

Como podemos constatar pelos resultados, o algoritmo 3 é o que tem menor número de iterações em todos os casos. O algoritmo 2 tem um comportamento semelhante com o do algoritmo 3 diferindo bastante apenas no caso Sucesso (Melhor caso). No entanto, para ficar claro que o algoritmo 3 tem um menor número de iterações que o algoritmo 2, apresentamos na Figura 6 uma comparação entre esses dois algoritmos para os casos Sucesso/Insucesso: (Pior caso).



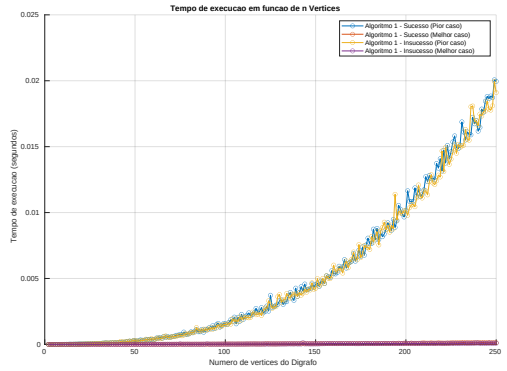
(a) Sucesso (Pior caso).



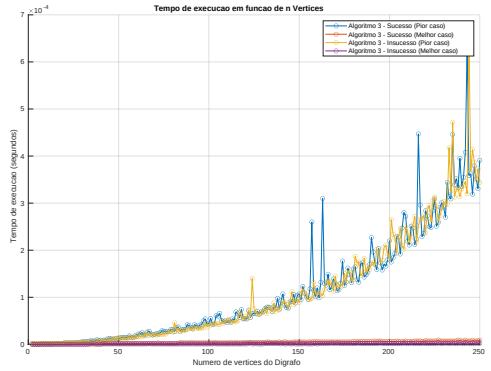
(b) Insucesso (Pior caso).

Figura 6. Comparação entre o algoritmo 2 e o algoritmo 3.

Em seguida apresentamos a comparação das diferentes situações para o algoritmo mais lento, algoritmo 1, e para o algoritmo mais rápido, algoritmo 3. Devido ao espaço limitado do relatório não colocámos o do algoritmo 2. Contudo, constatou-se nos gráficos anteriores que apresenta um comportamento semelhante ao algoritmo 3, apresentando no entanto um maior número de iterações e um tempo de execução superior. Os resultados da Tabela 2 também confirmam que o algoritmo 2 é mais lento que o algoritmo 3, por exemplo, para o caso Sucesso: (Melhor caso) o algoritmo 2 apresenta uma ordem de complexidade superior.



(a) Algoritmo 1.



(b) Algoritmo 3.

Figura 7. Comportamento dos algoritmos 1 e 3 para os diferentes 4 casos de estudo.

Para o algoritmo 1, podemos observar na Figura 7a que a ordem de complexidade de sucesso e insucesso é igual. Isto confirma o resultado presente na Tabela 2 onde para o algoritmo 1 tínhamos obtido para ambos os casos Sucesso/Insucesso: (Pior caso) $\implies \mathcal{O}(V^3)$ e para ambos os casos Sucesso/Insucesso: (Melhor caso) $\implies \Omega(V^2)$. Relativamente ao algoritmo 3, também podemos observar na Figura 7b que a ordem de complexidade de sucesso e insucesso é igual, o que confirma o resultado presente na Tabela 2 para o algoritmo 3. Comparando os dois gráficos apresentados conseguimos também verificar uma diferença significativa entre os tempos de execução em função do número de vértices, o que comprova o apresentado até ao momento.

3. Conclusão

No decorrer deste projeto, desenvolvemos três algoritmos deterministas para determinar a ordenação topológica dos vértices de um grafo orientado, caso exista. A complexidade destes algoritmos foi analisada ao longo deste relatório. Um dos principais desafios enfrentados foi a definição do peso (médio ou aproximado) de cada iteração, visando alcançar uma boa aproximação com o tempo de execução. Acreditamos ter superado este desafio, uma vez que, ao observarmos a evolução do tempo de execução em função do número de vértices, Figura 3, e do número de iterações, Figura 5, constatamos uma significativa correlação entre os dados. Isso permitiu-nos extrair conclusões sobre a complexidade, utilizando as fórmulas exatas derivadas do número de vértices, associando-as também ao tempo de execução.

Este trabalho foi desafiante tendo nos permitido conjugar diferentes linguagens de programação, de forma a podermos realizar os testes da forma mais eficiente e robusta possível, como explicado no esquema da Figura 1.