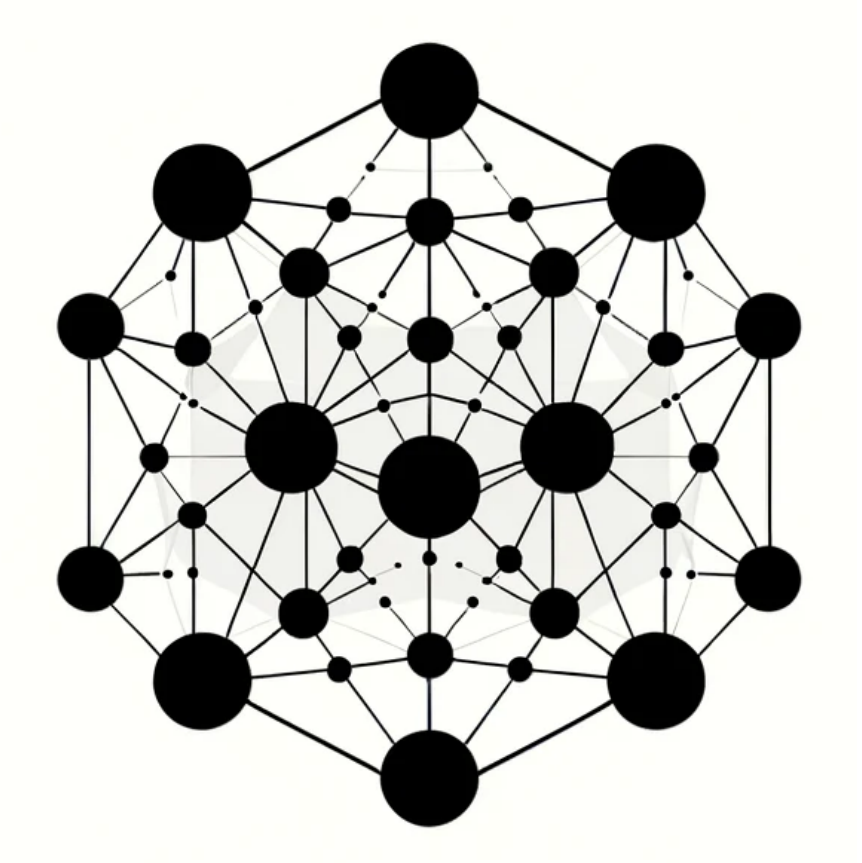
deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

Trabalho 02

O TAD GRAPH

2023/24



Algorítmos e Estrutura de Dados

Prof. Joaquim João Estrela Ribeiro Silvestre Madeira

Maria Luís Delgado Linhares - 113534
Rui de Faria Machado - 113765

Índice

Introdução

Este relatório aborda o desenvolvimento de um projeto na disciplina de Algoritmos e Estruturas de Dados, centrado no Tipo Abstrato de Dados (TAD) GRAPH para manipulação de grafos. O projeto iniciou-se com a implementação do script graph.c, onde foram desenvolvidas as funções GraphCopy, GraphFromFile, GraphRemoveEdge e GraphCheckInvariants, essenciais para a manipulação básica de grafos.

Posteriormente, o foco deslocou-se para a implementação de três algoritmos de ordenação topológica, cada um adotando uma abordagem única para ordenar os vértices de um grafo direcionado.

Este relatório detalha o processo de desenvolvimento desses componentes, enfatizando os testes realizados e as análises de complexidade algorítmica, refletindo a contribuição do projeto para a compreensão e aplicação de estruturas de dados em grafos.

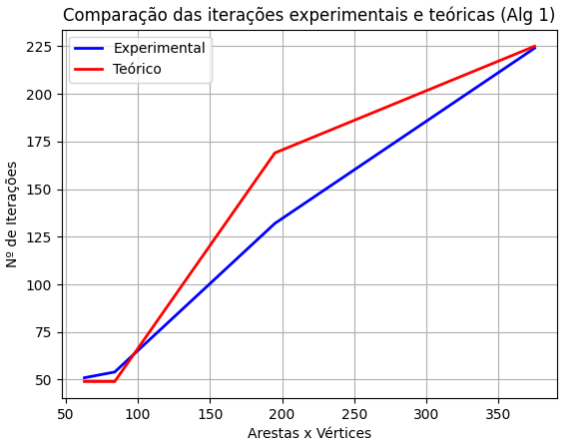
Algoritmos de Ordenação Topológica

Os algoritmos de ordenação topológica abordados neste projeto foram inicialmente introduzidos e explorados durante as aulas teóricas, fornecendo uma base sólida para a sua implementação e análise subsequente.
Para a análise da complexidade, foram usadas as terminologias n e m, que representam o número de vértices e arestas respetivamente.

Algoritmo 1: Cópia do Grafo G

Este algoritmo começa pela criação de uma cópia *G'* do grafo original *G*. O processo consiste em selecionar repetidamente um vértice que não possui arestas incidentes (InDegree igual a zero), imprimir seu identificador e, em seguida, eliminar esse vértice e todas as arestas que dele emergem de G'.
Uma limitação deste algoritmo é a ineficiência causada pela necessidade de criar uma cópia do grafo e pela execução de buscas sucessivas através do conjunto de vértices para encontrar vértices sem arestas incidentes.

Análise da Complexidade:

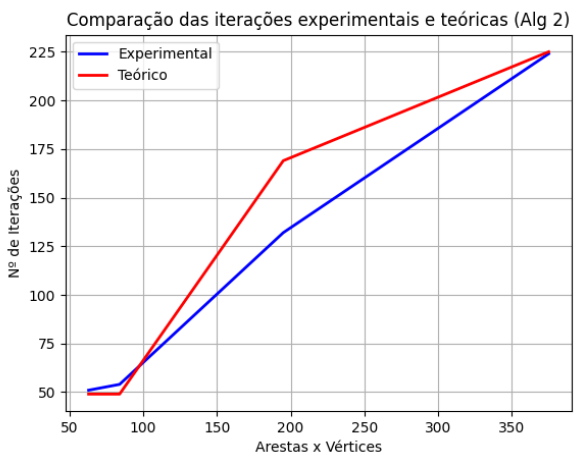


Complexidade Teórica: O(n²)

Algoritmo 2: Array Auxiliar

Este algoritmo utiliza um array auxiliar numEdgesPerVertex para registar o grau de entrada (InDegree) de cada vértice do grafo. O processo continua enquanto for possível identificar um vértice v tal que numEdgesPerVertex[v] == 0 e que ainda não tenha sido marcado como parte da ordenação topológica. Após imprimir o ID deste vértice, o vértice é marcado como parte da ordenação. Em seguida, para cada vértice w adjacente a v, o valor de numEdgesPerVertex[w] é decrementado.
A principal ineficiência deste método advém das repetidas buscas através do conjunto de vértices.

Análise da Complexidade:



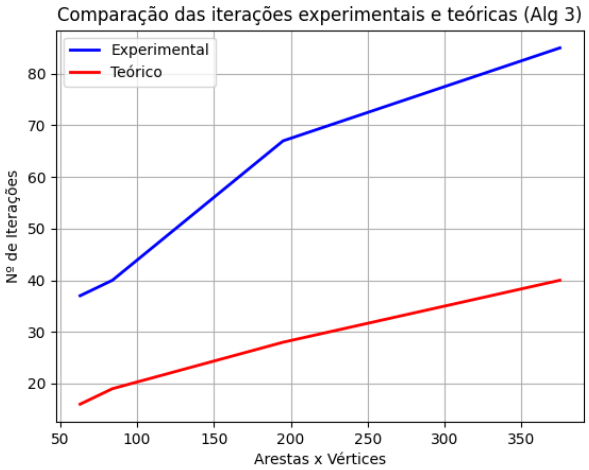
Complexidade Teórica: O(n²)

j

Algoritmo 3: Manutenção do Conjunto de Candidatos

Este algoritmo começa guardando, num array auxiliar numEdgesPerVertex, o grau de entrada (InDegree) de cada vértice. Em seguida, cria-se uma fila vazia onde são inseridos os vértices v cujo numEdgesPerVertex[v] == 0. O processo continua enquanto a fila não estiver vazia, retirando-se o próximo vértice v da fila e imprimindo seu ID. Para cada vértice w adjacente a v, numEdgesPerVertex[w] é decrementado e, se numEdgesPerVertex[w] passar a ser zero, w é inserido na fila.
Uma questão importante deste algoritmo é determinar o que acontece caso exista um ciclo no grafo, o que causaria a impossibilidade de estabelecer uma ordenação topológica para o grafo em questão.

Análise da Complexidade:



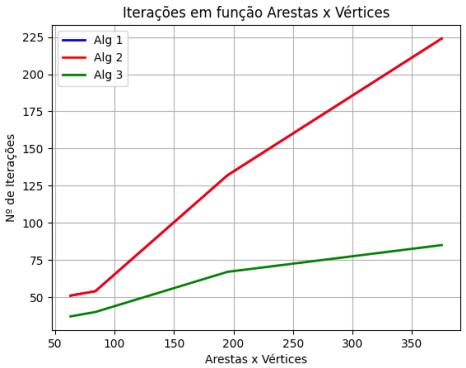
Complexidade Teórica: O(n + m)

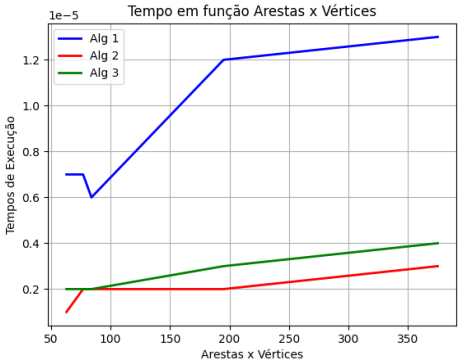
Verificação de fugas de memória

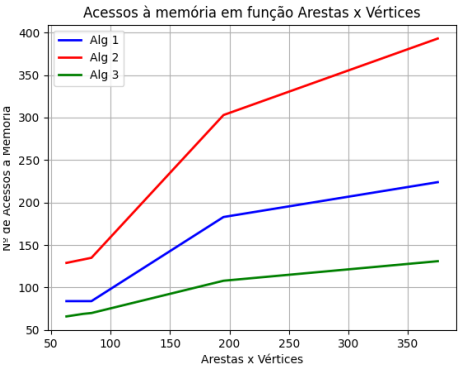
Para assegurar a integridade e eficiência do projeto, foi fundamental a realização de testes específicos para detetar possíveis fugas de memória. Utilizando a ferramenta Valgrind, uma análise meticulosa foi conduzida, revelando que o software desenvolvido está livre de quaisquer fugas de memória. Esta verificação reforça a robustez e confiabilidade das soluções implementadas.

Dados Experimentais

	AxV	t1	t2	t3	its1	its2	its3	acess1	acess2	acess3
DAG_1	63	0.000007	0.000001	0.000002	51	51	37	84	129	66
DAG_3	77	0.000007	0.000002	0.000002	53	53	39	84	133	69
DAG_2	84	0.000006	0.000002	0.000002	54	54	40	84	135	70
SW_DAG	195	0.000012	0.000002	0.000003	132	132	67	183	303	108
DAG_4	375	0.000013	0.000003	0.000004	224	224	85	224	393	131







De modo a completar a tabela acima e de seguida fazer os gráficos, foram criados dois macros ITS que conta o número de iterações e ACESSMEN que conta o número de acessos à memória e este foram incrementados no código cada vez que há mais uma iteração e um acesso à memória (tanto para leitura como escrita da memória). Na tabela os valores t1, t2, t3 correspondem aos tempos de execução dos três algoritmos, os valores its1, its2 e its3 ao número de iterações de cada algoritmo e os valores acess1, acess2 e acess3 ao número de acessos à memória que cada algoritmo faz. Foram criados vários scripts em python onde foram adicionados os valores respectivos para cada gráfico e plotados dos gráficos.

Conclusão

Em suma, este projeto não só cumpriu os seus objetivos como também proporcionou uma compreensão aprofundada sobre ordenação topológica em grafos. Acreditamos que o projeto foi concluído com êxito.