deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

Algoritmos e Estruturas de Dados

TAD Graph

Análise da complexidade dos algoritmos

1 - Cópia do grafo orientado

2 - Array auxiliar

3 – Fila de vértices

TRABALHO REALIZADO POR:

DIOGO FERNANDES 114137

RAQUEL VINAGRE 113736

PROF. JOAQUIM MADEIRA

Ano letivo 2023/2024 – 1º Semestre

Introdução

Neste relatório será analisada a complexidade de três algoritmos de ordenação topológica dos vértices de um grafo, implementados no TAD Graph.

Estes algoritmos usam um ciclo *while* até todos os vértices serem processados. Em cada iteração, procuram por um vértice com zero arestas de entrada, removem/decrementam as arestas de saída e atualizam as arestas de entrada dos seus vértices adjacentes, marcando o vértice atual como visitado.

1° Algoritmo - Cópia do grafo orientado

O primeiro algoritmo usa uma cópia do grafo orientado e efetua sucessivos apagamentos dos arcos emergentes de vértices que não tenham arcos incidentes. Guardando cada vértice na sequência após ser removido.

- Análise da Complexidade

Sendo V o número de vértices e E o número de arestas:

Inicializar a variável *numIncomingEdges* para cada vértice tem uma complexidade de O(V).

O ciclo *while* corre até todos os vértices serem processados (até n_vertices for igual a 0), portanto, iterando, no máximo, V vezes. Dentro deste loop, há outro loop remove as arestas de saída (cada aresta é processada uma vez. No máximo, há E iterações, tendo este loop uma complexidade de O(E).

$$\left(\sum_{i=0}^{V} 1\right) + \left(\sum_{j=0}^{E} 1\right) = (V + E)$$

Assim, a complexidade global do 1° algoritmo é $O(V) + O(E) = O(V + E)$

2º Algoritmo - Array auxiliar

O segundo algoritmo não necessita de uma cópia do grafo dado e usa um array auxiliar (um dos campos do registo) para sucessivamente procurar o próximo vértice a juntar à ordenação topológica. Em vez de remover as arestas e os vértices, decrementa apenas o elemento do array auxiliar.

- Análise da Complexidade

Sendo V o número de vértices e E o número de arestas:

Inicializar a variável *numIncomingEdges* para cada vértice tem uma complexidade de O(V).

O ciclo *while* corre até todos os vértices serem processados (até n_vertices for igual a 0), portanto, no pior caso possível, itera V vezes. Dentro deste loop, há outro loop que decrementa as arestas de saída (cada aresta é processada uma vez). Este loop tem uma complexidade de O(E).

$$\left(\sum_{i=0}^{V} 1\right) + \left(\sum_{j=0}^{E} 1\right) = (V + E)$$

Assim, a complexidade global do 2º algoritmo é $O(V) + O(E) = O(V + E)$

3º Algoritmo – Fila de vértices (queue)

O terceiro algoritmo usa uma fila para manter o conjunto dos vértices que irão ser sucessivamente adicionados à ordenação topológica.

- Análise da Complexidade

Sendo V o número de vértices e E o número de arestas:

Inicializar o array incomingEdgesCount tem uma complexidade de O(V). O for loop exterior percorre os vértices (V) e o loop interior percorre as arestas (E). Como cada aresta é apenas processada uma vez, a complexidade será linear e não quadrática.

$$\left(\sum_{i=0}^{V} 1\right) + \left(\sum_{j=0}^{E} 1\right) = (V + E)$$

Deste modo, a complexidade global do 3º algoritmo é $O(V) + O(E) = O(V + E)$

Testes e tabela de resultados

Ficheiro	Algoritmo	Tempo (10^-5) s	Sequência de vértices
SWtinyDAG.txt	1º	2.4	2 0 1 3 5 8 7 6 4 9 10 11 12
	2º	5	
	3º	7	
DAG_1.txt	1º	1.8	0 6 1 2 5 3 4
	2º	0.4	
	3º	0.5	
DAG_2.txt	1º	1.6	0 1 4 3 2 6 5
	2º	0.3	
	3º	0.6	
DAG_3.txt	1º	1.9	0 1 6 2 5 4 3
	2º	0.5	
	3º	0.5	
DAG_4.txt	1º	3.1	0 2 1 6 5 4 3 14 13 12 11 10 9 8 7
	2º	0.6	
	3º	0.6	

Podemos concluir que o 2º algoritmo é o mais rápido, sendo que o tempo de execução deste é muito similar ao do 1º e 3º algoritmos, o que faz sentido, pois todos apresentam a mesma complexidade. No entanto, como o 1º algoritmo necessita da cópia do grafo, o tempo de execução é ligeiramente superior.

Nos grafos que apresentam ciclos, a ordenação topológica não pode ser efetuada. Nestes casos, aparecia como resultado a mensagem "The topological sorting could not be computed!!", como era de esperar.

Conclusão

Após a realização deste trabalho, concluímos que a ordenação topológica pode ser processada através de diferentes algoritmos, e que a sua eficiência depende da complexidade da sua implementação.