Algoritmos e Estruturas de dados

TAD Graph

Abel José Enes Teixeira nº 113655
Filipe Pina de Sousa nº 114196
Janeiro 2024

Prof. Mário Antunes
Prof. Regente Joaquim Madeira

Ano letivo
2023-2024

deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

1 Introdução

Este trabalho tem como objetivo completar funções assinaladas, assim como elaborar três algoritmos de ordenação topológica dos vértices: **Cópia do grafo G, *Array* auxiliar, Manter o conjunto de candidatos em fila**.

2 Algoritmos

2.1 Alg.V1 - Cópia do grafo G

O Algoritmo V1 baseia-se na cópia do grafo original. Ele itera sobre os vértices, removendo aqueles com grau de entrada igual a zero, e continua esse processo até não haver mais vértices no grafo ou ocorrer um erro. Durante cada iteração, realiza comparações com a cópia do grafo, ajustando a estrutura para garantir a organização topológica. V = Número de vértices, E = Número de arestas.

2.1.1 Complexidade

Melhor caso: Quando nenhum vértice tem Indegree = 0.

$$\begin{aligned} T(V) &= \sum_{Vertex=0}^{V-1} 1 \\ &= V - 1 + 1 \\ &= V \\ &= O(V) \end{aligned} \tag{1}$$

Pior caso: Quando todos os vértices tem Indegree = 0.

$$\begin{aligned} T(V,E) &= \sum_{i=0}^{V-1} \left(\sum_{j=1}^{V-i-1} 1 \right) + \sum_{Vertex=0}^{V-1} \left(\sum_{i=1}^{GetInDegree(i)} 1 \right) \\ &= \sum_{i=0}^{V-1} (V - i + 1) + \sum_{Vertex=0}^{V-1} GetInDegree(i) \\ &= E + \sum_{i=0}^{V-1} V - \sum_{i=0}^{V-1} i + \sum_{i=0}^{V-1} 1 \\ &= E + V \cdot (V-1) - \frac{(V-1) \cdot (V-1+1)}{2} + V - 1 + 1 \\ &= E + V^2 - V + V - \frac{V^2 - V}{2} \\ &= E + \frac{V^2}{2} + \frac{V}{2} = O(\frac{V^2}{2} + \frac{V}{2} + E) \\ &= O(V^2 + E) \end{aligned} \tag{2}$$

2.2 Alg.V2 - *Array* auxiliar

O Algoritmo V2 utiliza dois *arrays* auxiliares. No primeiro, os vértices são adicionados no final à medida que são processados. No final do algoritmo, são removidos do início do *array*, reduzindo o grau de entrada dos vértices adjacentes. Essa abordagem pretende obter uma ordenação topológica eficiente dos vértices do grafo. V = Número de vértices, E = Número de arestas.

2.2.1 Complexidade

Melhor caso: Quando nenhum vértice tem Indegree = 0

$$\begin{aligned} T(V) &= \sum_{Vertex=0}^{V-1} 1 \\ &= V - 1 + 1 \\ &= V \\ &= O(V) \end{aligned} \tag{3}$$

Pior caso: Quando todos os vértices tem Indegree = 0

$$T(V,E) = \sum_{Vertex=0}^{V-1} \left(\sum_{i=1}^{GetInDegree(i)} 1 \right) + \sum_{Vertex=0}^{V-1} 1 + \sum_{Vertex=0}^{V-1} \left(\sum_{i=1}^{V-i+1} 1 \right) \tag{4}$$

$$\begin{aligned} &= V - 1 + 1 \sum_{Vertex=0}^{V-1} GetInDegree(i) + \sum_{Vertex=0}^{V-1} (V - i + 1) \\ &= E + V + \sum_{Vertex=0}^{V-1} V - \sum_{Vertex=0}^{V-1} i + \sum_{Vertex=0}^{V-1} 1 \\ &= E + V + (V-1) \cdot V - \frac{(V-1) \cdot (V-1+1)}{2} + V - 1 \\ &= E + V + V^2 - \frac{V^2 + V}{2} + V \\ &= E + V^2 - \frac{V^2}{2} - \frac{V}{2} + 2V \\ &= O(\frac{V^2}{2} + \frac{3V}{2} + E) \\ &= O(V^2 + E) \end{aligned}$$

2.3 Alg.V3 - Manter o conjunto de candidatos em fila

O Algoritmo V3 adota a estratégia de manter o conjunto de candidatos numa fila. A cada iteração, vértices com grau de entrada zero são (*enqueue*), e a fila é manipulada para garantir a ordenação topológica. Operações de *enqueue* e *dequeue* são realizadas para processar os vértices e seus adjacentes, proporcionando uma abordagem eficaz para a obtenção da ordenação desejada.V = Número de vértices, E = Número de arestas.

2.3.1 Complexidade

Melhor caso: Quando o grafo já está ordenado

$$\begin{aligned} T(V) &= \sum_{Vertex=0}^{V-1} 1 \\ &= O(V-1+1) \\ &= O(V) \end{aligned} \tag{5}$$

Pior caso: Quando o grafo não está ordenado.

$$\begin{aligned} T(V,E) &= \sum_{Vertex=0}^{V-1} \left(\sum_{i=1}^{GetInDegree(i)} 1 \right) + \sum_{Vertex=0}^{V-1} 1 + \sum_{Vertex=0}^{V-1} 1 \\ &= \sum_{Vertex=0}^{V-1} GetInDegree(i) + 2 \cdot \left(\sum_{Vertex=0}^{V-1} 1 \right) \\ &= E + 2 \cdot (V-1+1) \\ &= E + 2V \\ &= O(2V+E) \\ &= O(V+E) \end{aligned} \tag{6}$$

3 Métricas Adotadas

A utilização de contadores desempenham um papel essencial na visualização e compreensão dos valores obtidos em diversos tipos de grafos. Esses contadores permitem uma análise aprofundada dos diferentes tipos de acessos à memória realizados pelos algoritmos, contribuindo para o reconhecimento de padrões e comportamentos distintos em cada abordagem. A implementação dessas métricas proporciona uma avaliação mais precisa do desempenho e eficiência dos algoritmos implementados.

3.1 VertexAccess

O contador *VertexAccess* é fundamental, sendo incrementado a cada iteração do *while*, indicando o acesso a um novo vértice para ser ordenado. Essa métrica oferece informações importantes sobre a manipulação e acesso aos vértices, aspetos essenciais no processo de organização topológica.

3.2 AdjVertex

O contador *AdjVertex* é incrementado quando os vértices adjacentes são avaliados. Essa contagem fornece *insights* sobre a complexidade do processamento dos vizinhos de um vértice, desempenhando um papel na compreensão do comportamento dos algoritmos que dependem dessa análise para a ordenação topológica.

3.3 QueueAccess

O contador *QueueAccess* é incrementado sempre que a fila é acessada. A análise dessa métrica é precisa para compreender como os algoritmos interagem com a estrutura de fila, proporcionando informações sobre os padrões de acesso e manipulação que impactam diretamente a eficiência do processo.

3.4 NumIterations

O contador *NumIterations* é incrementado a cada iteração realizada. Essa métrica desempenha um papel fundamental na avaliação do desempenho global dos algoritmos, fornecendo uma medida clara do número de iterações necessárias para atingir o resultado final na ordenação topológica.

4 Tabelas

4.1 Grafo SWtinyDAG

Algoritmo	time	caltime	VertexAccess	AdjVertex	QueueAccess	NumIterations
Alg.V1	0.000009	0.000007	104	30	0	119
Alg.V2	0.000002	0.000002	104	15	0	132
Alg.V3	0.000002	0.000002	13	15	26	54

Table 1: *Output* dos 3 algoritmos do SWtinyDAG

4.2 Grafo DAG 1

Algoritmo	time	caltime	VertexAccess	AdjVertex	QueueAccess	NumIterations
Alg.V1	0.000005	0.000004	35	18	0	44
Alg.V2	0.000001	0.000001	35	9	0	51
Alg.V3	0.000001	0.000001	7	9	14	30

Table 2: *Output* dos 3 algoritmos do DAG 1

4.3 Grafo DAG 4

Algoritmo	time	caltime	VertexAccess	AdjVertex	QueueAccess	NumIterations
Alg.V1	0.000010	0.000009	135	50	0	160
Alg.V2	0.000003	0.000002	135	25	0	175
Alg.V3	0.000002	0.000002	15	25	30	70

Table 3: *Output* dos 3 algoritmos do DAG 4

4.4 Grafo DG 1

Algoritmo	time	caltime	VertexAccess	AdjVertex	QueueAccess	NumIterations
Alg.V1	0.000003	0.000002	7	7	0	8
Alg.V2	0.000001	0.000001	7	1	0	8
Alg.V3	0.000001	0.000001	1	1	2	14

Table 4: *Output* dos 3 algoritmos do DG 1

Nota: O Grafo DG 1 não pode ser organizado topologicamente porque é um digrafo cíclico.

4.5 Conclusões obtidas dos Algoritmos

O aumento no número de iterações do Algoritmo V2 em relação aos outros pode ser atribuído à natureza do seu método de processamento. O Algoritmo V2 utiliza dois *arrays* auxiliares e realiza operações de adição no final e remoção no início. Esse tipo de manipulação de *arrays* pode resultar em um número maior de operações.

Outra observação é, sempre que o grafo é possível, o conjunto de adjacências do Algoritmo V1 (*copy*) tem o dobro em comparação comparando com os outros, isso está relacionado com a lógica de remoção de vértices com grau de entrada igual a zero. O Algoritmo V1 remove esses vértices e, na cópia do grafo, essa remoção é refletida. Se considerarmos que a cópia do grafo é inicialmente idêntica ao grafo original, a remoção de um vértice no Algoritmo V1 implica na remoção do mesmo vértice e suas arestas correspondentes no grafo copiado, resultando numa duplicação do grau de entrada dos vértices adjacentes no grafo copiado.

Por fim, a abordagem do Algoritmo V3 é manter um conjunto de candidatos numa fila, e a operação de *enqueue* é utilizada para adicionar vértices com grau de entrada zero à fila. Isso proporciona uma manipulação eficiente da fila para garantir a ordenação topológica desejada, enquanto os outros algoritmos podem não ter uma estrutura de fila ou utilizar *enqueue* de maneira diferente.

5 Conclusão

Ao longo deste trabalho, a exploração do TAD (Tipo Abstrato de Dados) proporcionou uma melhoria em relação a Grafos, Algoritmos e Contadores. A implementação e análise dos algoritmos de ordenação topológica - Cópia do grafo G, *Array* auxiliar e Manter o conjunto de candidatos em fila - ofereceram insights valiosos sobre as complexidades envolvidas nesse processo.

Além disso, a experiência adquirida na manipulação de contadores permitiu uma compreensão melhor sobre o desempenho dos algoritmos, possibilitando a identificação e correção de erros. A abordagem de teste utilizada durante o desenvolvimento contribuiu para a robustez e confiabilidade nas soluções.

Concluindo este TAD não apenas enriqueceu nosso conhecimento teórico em Grafos e Algoritmos, mas também fortaleceu nossas habilidades práticas na implementação e otimização de algoritmos relacionados à ordenação topológica.