Análise e Síntese de Algoritmos

1º Projeto - 23 de março de 2018

77906 António Sarmento   81947 Marta Simões

Introdução

O exemplo do Sr. João Caracol com a sua cadeia de supermercados e o seu desejo de dividir a rede de distribuição em sub-redes regionais serve para evidenciar um problema relacionado com componentes fortemente ligadas (SCC, *Strongly Connected Component*) num dado grafo dirigo, tendo em conta que uma sub-rede é uma SCC. O objetivo do é que seja possível numa região seja possível enviar produtos (ou comunicar) para qualquer outro ponto da rede regional.

Portanto, abstraindo estes dados, procuramos:

1. O número de SCCs na região;
2. As ligações entre as SCCs;
3. Representar as ligações entre as SCCs pelo ponto mais importante.

Análise Teórica

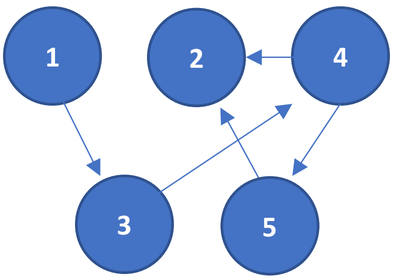
Como estrutura de dados para representar o grafo dirigido, temos os seguintes dados:

- Nº Vértices — V
- Nº Arcos — E

Dois grafos (G e SCC), cada um representado por um vetor de arcos que recorre a três vetores:

1. Tamanho: V; Índice: Número do Vértice; Conteúdo: Índice do Arco;
2. Tamanho: E; Índice: Número do Arco; Conteúdo: Vértice Final;
3. Tamanho: E; Índice: Número do Arco; Conteúdo: Arco Adjacente;

Exemplo da representação de um Grafo Dirigido:



#	Vértice
1	1
2	0
3	2
4	3
5	5

#	Arco	Irmão
1	3	-
2	4	-
3	2	4
4	5	-
5	2	-

A nossa estrutura de dados tem a seguinte eficiência para cada operação:

- Espaço: O(V+E)
- Inicialização: O(1)
- Inserir Arco: O(E)
- Encontrar Arco: O(E)

Para a procura de SCCs, aplicamos o algoritmo de Tarjan da seguinte forma:

1. Visitamos todos os vértices de um grafo *G* aplicando uma DFS, começando no vértice 1.
2. A cada visita de um vértice-fonte *s*:
    a. Guardamos *s* numa pilha.
    b. Percorrer os adjacentes. Se cada vértice adjacente *d* não tiver sido visitado, visitar recursivamente e atualizar o *low* da fonte com o menor entre *s* e *d*. Caso *d* esteja já na pilha, atualizar o *low* de s com o menor entre
    c. Quando chegarmos a um vértice previamente visitado cujo o tempo de descoberta e o low sejam iguais, começamos a fazer *pop* dos elementos da pilha, guardando o vértice-mestre correspondente a cada vértice.
3. Criar um 2º grafo *SCC* que contenha apenas as ligações entre vértices-mestre encontradas durante a DFS.
4. Expomos (por *print*) o conteúdo de *SCC*, ou seja o número de vértices e de arcos, e os respetivos arcos.

Esta aplicação tem uma complexidade O(V+E).

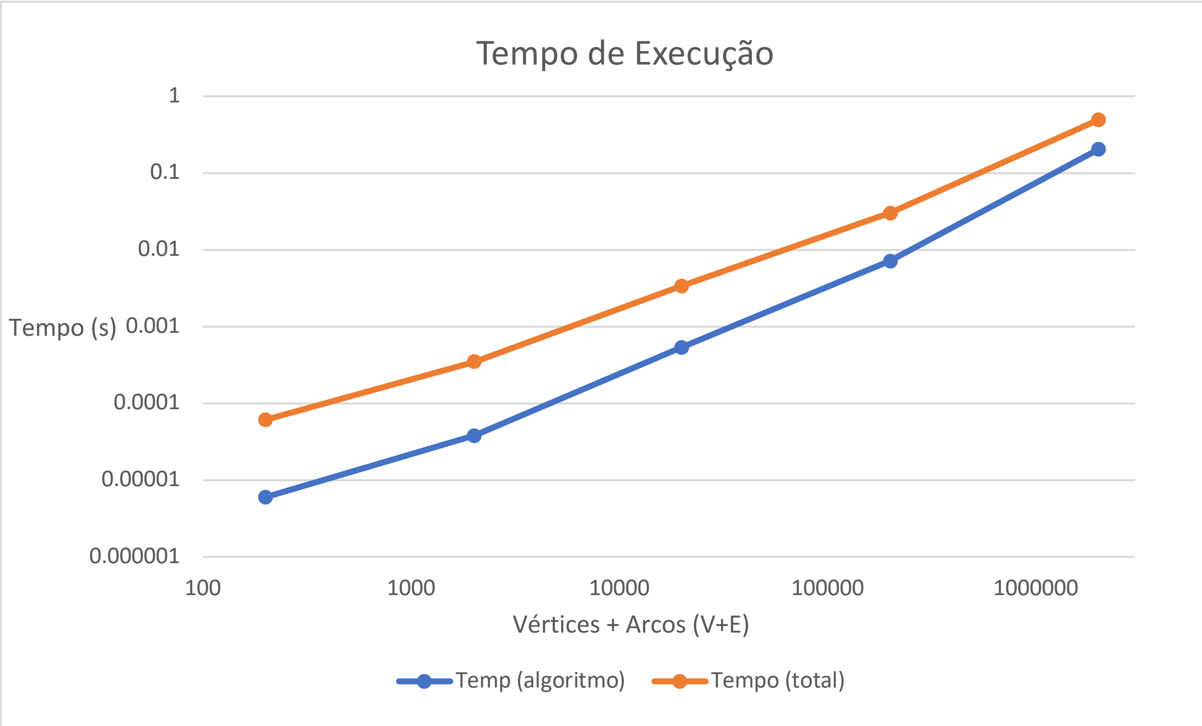
Implementação e Resultados

Implementámos o nosso programa em C pela familiaridade com a linguagem, por ser suficiente para a estrutura de dados que queríamos implementar e por ser *extremamente* eficiente. No sistema Mooshak passámos a todos os testes, obtendo os 16 valores totais.

Apresentamos de seguida cinco testes automaticamente gerados pelo programa fornecido na página da cadeira e aplicado ao nosso programa (testados num portátil com i5 a 2.3GHz):

V	E	V + E	T(Criação)	T(Algoritmo)	T(Ordenação)	T(Total)
100	100	200	0,000049	0,000006	0,000006	0,000061
1 000	1 013	2 013	0,000308	0,000038	0,000001	0,000347
10 000	10 000	20 000	0,002588	0,000536	0,000275	0,003399
100 000	100 000	200 000	0,020743	0,007138	0,002267	0,030148
1 000 000	1 000 000	2 000 000	0,262878	0,20354	0,025723	0,492141

O gráfico correspondente apresenta-se da seguinte forma:



Como podemos observar, o tempo demorado para criar o grafo é geralmente superior à aplicação do algoritmo de Tarjan. Podemos igualmente observar que os tempos obtidos experimentalmente formam uma reta que corresponde à complexidade teórica esperada de O(V+E).

Referências

- Introduction to Algorithms (3rd ed.), MIT Press and McGraw-Hill, ISBN 0-262-03293-7.

- Grafos: Slides Introdução Algoritmos e Estruturas de Dados, Profº Francisco Santos IST