**Relatório - Projeto 1 – Análise e Síntese de Algoritmos**

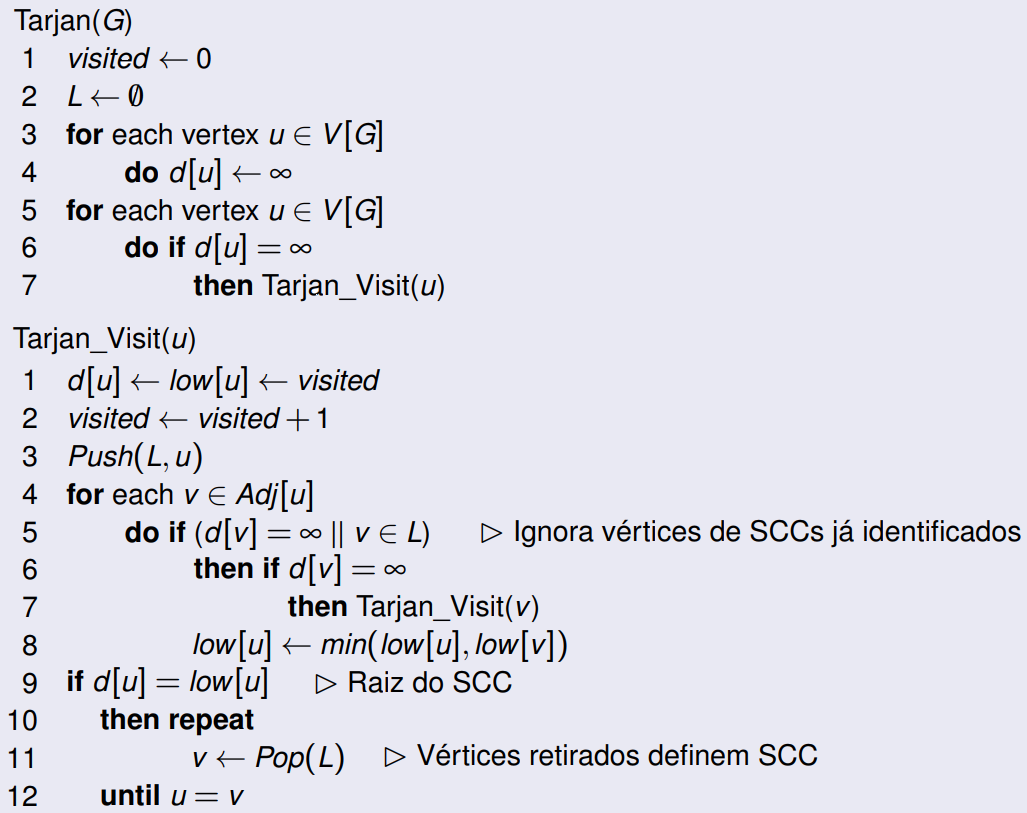
**I. Introdução**

Pretende-se desenvolver um programa que permita simplificar uma rede de distribuição, agrupando todos os seus pontos em sub-redes regionais. Uma sub-rede regional é um conjunto de pontos dos quais, a partir de qualquer um deles, é possível atingir qualquer outro ponto que também pertença a essa mesma sub-rede. É possível que uma sub-rede regional contenha apenas um ponto.

O programa lê o número de pontos da rede de distribuição, tal como o número total de ligações entre estes. Em seguida, lê todas as ligações específicas entre pontos, lendo primeiro o ponto de partida seguido do ponto de chegada. É então devolvido o número de sub-redes regionais encontradas, tal como o número de ligações entre estas, seguido da descrição de todas estas ligações. Essa descrição consiste na sub-rede de partida seguida da sub-rede de chegada da ligação em questão.

**II. Descrição da solução**

Para a implementação da solução, considera-se um grafo dirigido *G*=(*V*,*E*), em que cada vértice *v*∈*V* representa um ponto da rede de distribuição e cada aresta (*u*,*v*), *u*,*v*∈*V* significa que existe uma ligação que tem como ponto de partida o ponto *u* e como ponto de chegada o ponto *v*. As sub-redes regionais a serem encontradas são interpretadas como SCC (Strong Connected Component) do grafo dirigido *G*.



Para a pesquisa de todas as SCC presentes no grafo dirigido *G*, foi utilizado o algoritmo de Tarjan, cujo pseudocódigo está representado na Fig. 1.

**Fig. 1: Pseudocódigo do Algoritmo de Tarjan**

A solução é assim composta por x passos distintos:

1. São lidos dois inteiros do *stdin*: O número de pontos da rede de distribuição e o número de ligações entre estes. É então inicializado o grafo *G*, e são lidas em seguida todas as ligações individualmente, sendo estas adicionadas a *G*.
2. Aplica-se o Algoritmo de Tarjan a *G*, guardando todas as SCC encontradas.
3. Itera-se por entre todas as SCC encontradas anteriormente e cria-se um novo grafo *Gout*=(Vout,*Eout*) que contém os vértices de menor referência de cada SCC.
4. Itera-se então por todos os vértices *v*∈*Vout* , convergindo todas as adjacências de todos os vértices da sua SCC que não são relativas a vértices da sua SCC para *v*. Neste processo também se garante que todas as adjacências são relativas ao ponto de menor referência de cada SCC e que não existem adjacências duplicadas.
5. Imprime-se para o *stdout* o número de vértices de *Vout*(número de SCC/sub-redes regionais), seguido da soma do número de adjacências/ligações de todos eles. Por fim, imprime-se então cada ligação individualmente, consistindo do ponto de partida seguido do ponto de chegada correspondente a cada uma das ligações.

**III. Análise Teórica**

Considera-se *V* o número de vértices do grafo e *E* o seu número de arestas

O grafo é representado através de uma lista de vértices, em que cada vértice contém informação relativa a todas as suas adjacências, sendo estas representadas através de uma lista de ponteiros para os vértices adjacentes. Assim, o espaço reservado para o grafo depende do número de vértices dado como input, e o espaço para as adjacências de cada vértice é realocado para cada nova adjacência, sendo este dinâmico. A leitura e a reserva de espaço relativas á representação do grafo têm assim uma complexidade *O*(*V*+*E*).

Após a inicialização, procede-se á aplicação do Algoritmo de Tarjan, sendo assim preciso inicializar uma stack que segue a lógica FILO (First In Last Out), de tamanho dinâmico. A realocação de espaço é feita em incrementos de potências de 2, sendo a sua complexidade *O*(*1*) amortizado. Isto porque a complexidade de cada realocação é *O*(*n*) (sendo n o número de elementos na stack), mas esta é dividida por todos os seus elementos, cuja inserção é, na maioria das vezes, *O*(*1*). Para se guardar o output foi também inicializada uma lista de SCC de tamanho dinâmico que é realocada cada vez que é encontrada uma nova SCC, sendo a sua complexidade de inserção *O*(*V*). Cada SCC consiste também numa lista de tamanho dinâmico, que é realocada cada vez que se insere um novo vértice, tendo assim uma complexidade de inserção *O*(*V*). A complexidade relativa ao armazenamento do output do Algoritmo de Tarjan é assim *O*(*V*). Assim, tendo o Algoritmo de Tarjan uma complexidade associada de *O*(*V*+*E*), concluímos que a sua aplicação e armazenamento de resultados tem uma complexidade *O*(*V*+*E*)+*O*(*V*)+*O*(*1*)=*O*(*V*+*E*).

Em seguida, cria-se um novo grafo (grafo final), apenas com os vértices de menor referência de cada SCC. Para isso percorre-se todos os SCC e encontra-se o vértice de menor referência de cada uma deles, adicionando-o ao novo grafo. Para encontrar o vértice de menor referência de cada SCC é preciso percorrer todos os vértices contidos nesta. Sendo que o ciclo percorre todos os vértices do grafo original, a criação do novo grafo tem uma complexidade *O*(*V*).

Para todos os vértices do grafo final, é criada uma nova lista de adjacências, na qual constam todas as adjacências de todos os vértices pertencentes á sua SCC, á exceção das adjacências que se referem a vértices da sua SCC. As adjacências são também actualizadas para se passarem a referir ao vértice de menor referência das suas respectivas SCC, e é garantido que não há adjacências repetidas. Para a criação da nova lista de adjacências é preciso percorrer todas as adjacências de todos os vértices pertencentes á SCC do vértice em questão, verificar se estas obedecem às condições referidas anteriormente, e adicionar o vértice de menor referência da SCC a que se referem á lista. Sendo que a lista de adjacências é de tamanho dinâmico, a sua memória é realocada a cada nova inserção e o seu tamanho máximo é *V*, a sua complexidade de inserção é *O*(*V*). Ainda se procede á invocação do Algoritmo QuickSort para a ordenação de cada lista de adjacências, tendo este uma complexidade *O*(*V2*). Este ciclo percorre todas as adjacências do grafo original, tendo por isso uma complexidade *O*(*E*). Assim, a criação de todas as novas listas de adjacências ordenadas tem complexidade *O*(*V*)\**O*(*E*)+*O*(*V*)\**O*(*V2*)=*O*(*V3*).

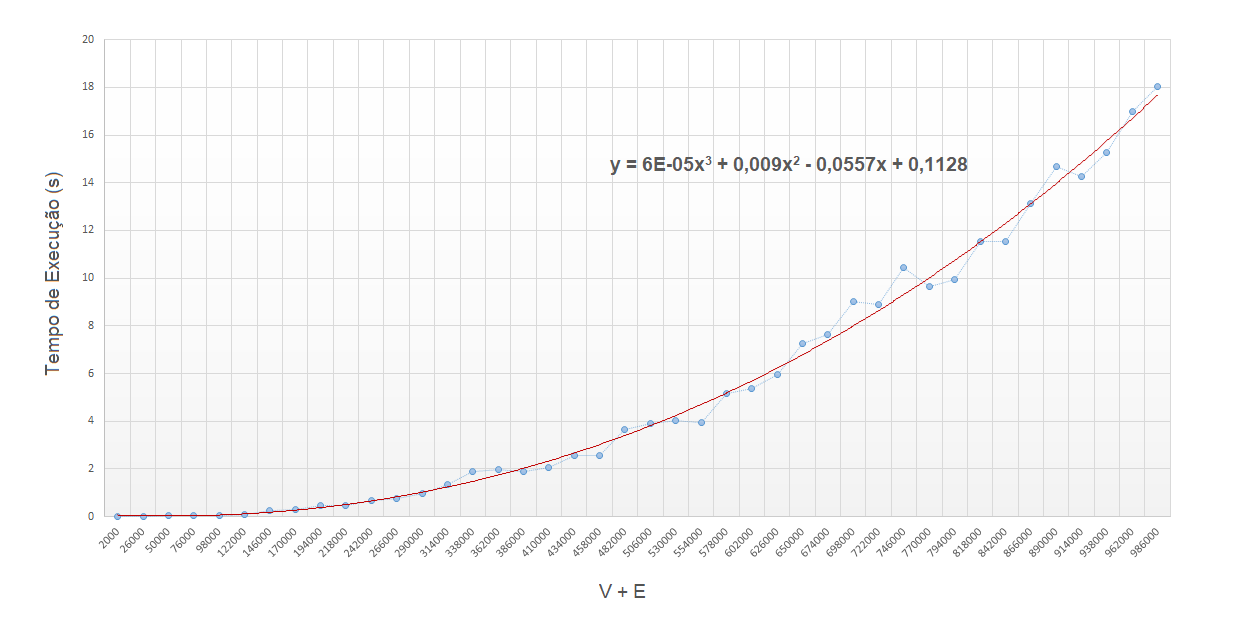
O grafo final é também depois ordenado, recorrendo ao Algoritmo QuickSort, tendo esta operação uma complexidade *O*(*V2*).

Por fim, apresenta-se o output. Após imprimir o número de SCC encontrados e o número de adjacências do grafo final, que é feito em tempo constante, é necessário percorrer todos os vértices do grafo final e imprimir todas as suas adjacências. Esta operação tem, por isso, uma complexidade *O*(*V*+*E*).

Assim, a complexidade total do programa, tendo em conta que todas as outras pequenas operações não mencionadas são realizadas em tempo constante e, por isso, não escalam com o tamanho do input, é *O*(*V*+*E*)+*O*(*V*+*E*)+*O*(*V*)+*O*(*V3*)+*O*(*V2*)+*O*(*V*+*E*)=*O*(*V3*).

**IV. Análise dos Resultados Experimentais**

O gráfico anexado sob a forma da Fig. 2, representa os resultados experimentais obtidos após a execução do programa com tamanhos de input variáveis, numa máquina com outros processos em execução, e cuja estabilidade não é garantida. Cada valor representa a média de 3 medições sequenciais, com input constante.

Através da análise deste, podemos verificar que suporta a hipótese teórica apresentada anteriormente, sendo que a linha aproximada representativa da relação entre o tempo de execução e o tamanho do input (linha vermelha) é gerada por uma função polinomial de 3º grau.

**Fig. 2: Relação entre o tempo de execução (s) e o tamanho do input (V+E)**

João Galinho (LEIC-T 87667)

Filipe Henriques (LEIC-T 87653)