# Análise de Síntese e Algorítmos

## Relatório 1ºProjeto

### Grupo 71

### 

Com a evolução das novas tecnologias da comunicação, tornou-se imperativo criar mecanismos de partilha de informação automáticos e controlados. As pessoas querem partilhar cada vez mais informação, mas apenas com um determinado grupo de pessoas, com o intuito de manter a informação pessoal privada. E à medida que a informação partilhada não pára de aumentar, a preocupação com a segurança é cada vez mais evidente.

Quando uma informação é partilhada, é importante saber qual a forma mais eficaz de efetuar essa partilha e quem consegue aceder a essa informação.

O projeto tem como objetivo analisar os dados referentes à partilha de informação numa determinada rede, detetar os grupos de partilha e devolver algumas estatísticas sobre esses grupos: o número de grupos existentes, o tamanho do maior grupo, e o número de grupos que apenas recebem informação, não partilhando informação para fora do grupo. Denomina-se por um grupo de partilha, o conjunto de várias pessoas que partilham toda a informação entre si. Isto é, quando qualquer pessoa partilha uma informação, todas as pessoas que pertencem a esse grupo recebem a mesma informação.

Os dados de entrada são: o número de pessoas e o número de partilhas que existem entre as pessoas e a descrição dessas mesmas partilhas (quem partilha e com quem partilha). Optou-se por guardar esta informação num grafo, por ser a forma mais eficaz de o fazer.

Num contexto geral, um grafo é uma estrutura constituida por um conjunto de vértices V (Vertex, sendo neste caso, pessoas), e um conjunto de arcos E (Edges, relação de partilha de informação entre duas pessoas) que ligam dois quaisquer vértices.

Um grafo pode ser implementado de diversas formas, sendo duas dessas opções uma lista de adjacências e uma matriz de adjacências. Existem diversas opiniões sobre qual destas duas opções de implementação é preferível a nível de memória ocupada e tempo gasto. Quanto à memória ocupada, a lista de adjacências é apenas eficaz quando se trata de grafos com poucos arcos relativamente ao número de vértices, visto esta guardar em memória apenas os arcos existentes. Por outro lado, a matriz guarda todas as possibilidades de ligações (com um valor booleano ,Verdadeiro ou Falso, que indicam se um dado arco existe ou não). Assim, a lista de adjacências é mais eficiente para o objetivo deste projecto, pois esta encontra as adjacências de um vértice em O(1), enquanto a matriz encontra em O(V) (a matriz apenas é mais eficiente se o objectivo for testar se dois vértices aleatórios são adjacentes) Assim, optou-se por implementar uma lista de adjacências.

Relativamente aos grafos, chama-se grafo fortemente ligado quando é possível aceder a todos os seus vértices a partir de um qualquer vértice inicial. Quando é possível dividir um grafo em grafos fortemente ligados de menor tamanho, cada um desses sub-grafos designa-se por componente fortemente ligada ou SCC (do inglês Strongly Connected Component). Num dado vértice de uma SCC, é possível aceder todos os vértices pelos quais a SCC é constituida. No contexto do problema, um grupo de partilha de informação é, precisamente, uma SCC.

Existem diversos algoritmos para detetar SCCs num grafo. Os três mais conhecidos são: Kosaraju’s algorithm, Tarjan’s algorithm e Path-based strong component algorithm.

Optou-se por usar o Algoritmo Tarjan visto que necessita de menos tempo e de menos espaço em memória para ser executado. Por comparação, o Algoritmo de Kosaraju é mais fácil de implementar, mas percorre o grafo duas vezes, ou seja, demora duas vezes mais tempo que os outros dois algoritmos (contudo continua a correr em tempo linear). o Path-based strong component algorithm necessita de duas “pilhas” para guardar os vértices que ainda não foram inseridos em nenhuma SCC e para guardar o caminho atual da pesquisa em profundidade, enquanto que o Algoritmo Tarjan apenas guarda em memória os vértices que ainda não foram inseridos em nenhuma SCC. Assim, embora ambos os algoritmos, no pior caso possível, corre em tempo O(|V| + |E|), o Algoritmo Tarjan precisa de menos espaço em memória.

De modo a detetar um grupo de partilha isolado, adicionou-se duas variáveis boolenas a cada vértice. Para um dado vertice v1, a variável OUT, fica a TRUE se a partir v1 for possivel aceder a um vértice branco (um vértice que ainda nao tinha sido acedido), e volta a ficar a FALSE caso esse vértice branco tenha o mesmo valor na variável lowlink que v1. A variável NOT serve para proteger a passagem da variável OUT de TRUE para FALSE. Assim, basta uma vertice adjacente a v1 ter um valor de lowlink diferente de v1 para a variavel NOT ficar a TRUE e impedir que a variável OUT a FALSE possa ser avaliada de forma errada. No fim, ao criar uma SCC nova, caso algum dos vértices tenha a variável OUT ou NOT a TRUE, então a SCC não é isolada. Caso contrário é contada como SCC isolada.

Desta forma a complexidade do algoritmo implementado não é alterada, não é necessário correr o algoritmo Tarjan múltipas vezes nem guardar as SCC para verificar todas as adjacências posteriormente.

No decorrer do desenvolvimento do projecto, deparámo-nos com algumas situações em o algoritmo necessitava de um grande período de tempo. Após uma análise mais cuidada do código, descobrimos um modo mais eficiente para verificar se um dado elemento está na “pilha”. Após esta alteração, o algoritmo ficou com complexidade O(|V|+|E|), a complexidade pedida. Também nos deparámos com dados de saída incorretos, que descobrimos depois serem pequenos erros no código.

Após serem realizados vários testes, chegou-se a conclusão que o algoritmo implementado era de facto linear com o número de vértices e arcos fornecidos, pois para uma quantidade de dados n (n = v + e, vertices + arcos) o tempo de execução foi t. Ao serem fornecidos 2n elementos, o tempo de execução passou a ser aproximadamente 2t, tal como acontece com todos os pares de input/output testados: (5n => 5t;10n => 10t, 100n => 100t; 1000n => 1000t, etc).