Instituto Superior Técnico

Relatório 1º Projecto de ASA

18 de Março de 2016

Grupo AL009
André Mendonça (82304)
Gonçalo Ribeiro (82303)

1 - Introdução:

1.1 - O problema:

O problema consiste no cálculo do número de pessoas que têm mais influência na partilha de conteúdo numa rede social, isto é, saber quais são as pessoas que entre todos os utilizadores da rede social conseguiram obter mais partilhas de conteudo por parte dos outros utilizadores (relações entre utilizadores).

1.2 - O Input:

O input consiste numa primeira linha com o número de pessoas **N** (Sendo que N tem de ser obrigatoriamente maior ou igual a 2) , e o número de ligações de partilha **L** e numa lista de **L** linhas em que cada linha contém dois inteiros de ***u*** e ***v*** que representam que a pessoa identificada por ***u*** partilha informação pelo menos uma vez com a pessoa identificada por ***v***.
Todas as pessoas são representadas por números inteiros entre **1** e **N**.

1.3 - O Output:

O output é representado com uma primeira linha F, sendo o número de pessoas fundamentais na rede dado pelo input. Existindo em seguida outra linha que representa o identificador mínimo ***m*** e máximo **M** do conjunto **F**, tendo em conta que obrigatoriamente ***m*** é sempre maior ou igual a **M** (***m***≥**M**) e que ***m*** e **M** podem variar entre **1** e **L** (**1** ≤ ***m***/**M** ≤ **L**).

2 - Descrição da Solução:

2.1 - Linguagem de programação:

Para a implementação do projecto foi escolhida a linguagem C++ porque das três linguagem disponíveis para a realização do problema esta possui várias estruturas de dados já implementadas de raiz.

2.2 - Estruturas de dados:

As estruturas de dados utilizadas no projecto foram:

- **Graph** - estrutura criada para simular um grafo que contém os vértices e as suas ligações
- **std::list** - Lista dinâmica
- **adj** - Lista dinâmica dos vértices adjacentes
- **visited/ap** - Lista de valores booleanos
- **Discovery_time** - Lista de inteiros

2.3 - A solução:

Após a análise do problema concluímos que seria necessário utilizar um ***Grafo*** (não dirigido) para a resolução do problema, pois existem os utilizadores (vértices) que partilham conteudo entre si (arcos), como a solução do problema passava por determinar o número de pessoas fundamentais na rede foi necessário implementar o algoritmo ***Depth-First Search (DFS)***.

> ***Grafo***: G = (V,E), definido por um conjunto V de vértices e um conjunto E de arcos.
>
> ***Depth-First Search***: Dados G = (V,E) e vértice fonte *s*, o algoritmo BFS explora sistematicamente o conjunto E a partir do vértice fonte *s* até chegar a um ponto em que não existam mais elementos E, retrocedendo de seguida e realizando o mesmo processo novamente.

2.4 - Algoritmo:

O programa começa por obter o input necessário sendo este, o número de pessoas, o número de ligações entre as pessoas e as ligações por si mesmas. Este input vai permitir criar um grafo *graph*, executando de seguida o algoritmo de procura **DFS** em *graph* utilizando também uma função recursiva auxiliar **findAP** que permite guardar os pontos de articulação ou vértices de corte retornando no fim as informações requeridas (número de vértices de corte, o menor e o maior destes):

1. O programa começa por ler a primeira linha do input e cria um grafo **graph** com N vértices (var **n_people**) e L arcos (var **n_connections**).
2. De seguida é então criado o grafo graph com **N** vértices.
3. São lidas as restantes linhas de input do programa ligando arcos aos vértices do grafo **graph**.

4. É chamada a função **DFS** para o grafo **graph**.
    a. Os arrays **parent**, **visited** e **ap** são inicializados com os valores -1 para **parent** e valor booleano false para **visited** e **ap**.
    b. Para cada vértice verifica-se se este já foi visitado, no caso de não ter sido criado é executada a função **findAp** para encontrar pontos de articulação a partir do vértice atual como raiz do grafo.
        i. Começa por definir o vértice como visitado.
        ii. Regista os tempos de descoberta de cada vértice adjacente e para cada um destes verifica se já foi visitado. No caso de não ter sido regista em **parent** o "pai" deste vértice adjacente.
        iii. De seguida a função é chamada por si própria para fazer as mesmas verificações a todos os vértices adjacentes ao vértice adjacente atual.
    c. Voltando à função principal **DFS**, percorre-se o array **ap**, para verificar quais dos vértices são pontos de articulação (valor booleano true), incrementando a variável **nr_ap** e para encontrar o menor e o maior destes.
    d. Após o fim do ciclo mencionado na alínea anterior, inicia-se a verificação da existência de pelo menos 1 ponto de articulação e no caso de não existir definem-se o menor (var **min_ap**) e o maior (var **max_ap**) com o valor -1 como requerido.
    e. No fim de todas estas verificações são imprimidos os valores pretendidos (número de pontos de articulação, o menor e o maior dos mesmos).

3 - Análise Teórica:

Este programa executa em tempo linear ao tamanho do problema. Analisando a complexidade de cada função do programa:

- Função *main* é ***O(n + m):***
    - Leitura do Input é ***O(n)***
    - Construtor do grafo é ***O(n + m)***
    - Função DFS é ***O(n + m)***:
        - Ciclo principal é ***O(n + m)***
        - Output é ***O(n)***

4 - Avaliação Experimental:

A solução apresentada para este projecto, passou a todos os testes disponíveis no sistema Mooshak, com distinção. Sendo que os testes do Mooshak não são acessíveis para avaliar

os tempos de execução optamos por usar os testes públicos e criar mais alguns testes para avaliar o desempenho do nosso programa.
Os resultados desses testes estão descritos na seguinte tabela e representados no gráfico.

Nº de Vértices	Nº de Arcos	Tempo (ms)
8	7	007
10	9	013
250	2000	014
500	5000	042
2000	20 000	106
5000	22 500	127
7500	24 000	163
10 000	25 000	137
12 500	25 000	147
15 000	50 000	283
17 500	52 500	315
20 000	22 803	133
25 000	35 000	207
30 000	58 783	332
35 000	68 580	515