ANÁLISE E SÍNTESE DE ALGORITMOS
RELATÓRIO 1º PROJECTO

TRABALHO REALIZADO POR:

Diogo Freitas, 81586
Francisco Mira, 80888
Grupo 105

Introdução

O 1º projecto da cadeira Análise e Síntese de Algoritmos consiste na identificação das pessoas fundamentais na difusão de informação através da comunicação entre utilizadores numa rede social, i.e., identificar as pessoas pelas quais a informação necessita de passar para chegar a outra determinada pessoa.

O input contém a informação necessária sobre o número de pessoas pertencentes à rede social e o número de ligações entre estas, bem como as próprias ligações. O input é recebido sob a forma de L + 1 linhas, sendo que L corresponde ao número de ligações entre utilizadores. Na linha inicial é recebido o número de utilizadores e o número de ligações totais da rede, separados por um espaço. Nas linhas seguintes é recebida, por linha, uma ligação entre dois utilizadores com o número identificador de cada um separados por um espaço.

O output é constituido por duas linhas, sendo que na primeira linha se encontra o número total de pessoas fundamentais na difusão de informação na rede e na segunda linha se encontram dois utilizadores fundamentais descritos pelo menor e maior número identificador.

Descrição da Solução

Linguagem de Programação

Como linguagem de programação para a implementação da solução, foi escolhida a linguagem C++ devido a um maior à vontade com a linguagem por parte dos elementos do grupo e à existência de bibliotecas predefinidas que iriam auxiliar na implementação da estrutura do código.

Estruturas de Dados

Para a implementação do algoritmo foram utilizadas duas estruturas de dados: um vector representativo da rede social, no qual é inserida uma lista por pessoa representativa das ligações que esta possui na rede e vectores auxiliares à execução do algoritmo, nos quais são inseridos números inteiros e *bools*.

std::vector um vector dinâmico de memória contígua.

std::list uma lista.

std::vector< std::list<int> > um vector de listas de inteiros.

Solução

Como solução para o problema apresentado no enunciado do projecto, organizámos os utilizadores num grafo não dirigido, pois o problema diz respeito a relações entre utilizadores, para as quais a direcção não é relevante. Este grafo representa a rede social, em que cada partilha de informação entre utilizadores é representada por uma aresta entre vértices do grafo. Foi identificado que as pessoas fundamentais para a difusão de informação na rede social correspondiam a pontos de articulação no grafo, i.e., um vértice que, se removido, torna o grafo desconexo.

Algoritmo

O programa começa por ler o input, definindo o número de vértices e arcos existentes no grafo. De seguida, cria o grafo e todos os vectores auxiliares necessários para a execução do Algoritmo de Tarjan. No passo seguinte, o grafo é tratado pelo Algoritmo de Tarjan que determina quais os pontos de articulação, quantos são e quais têm o menor e maior identificador. Por fim, o programa fornece o output desejado.

A nossa abordagem tem como base o seguinte procedimento:

I. Inferir a informação sobre o número total de utilizadores, número de ligações entre eles e as próprias ligações e criar o grafo correspondente à informação dada.
II. Passagem pela função *tarjanAlgorithm()* onde está implementado o Algoritmo de Tarjan que procura os pontos de articulação.
III. Impressão dos dados produzidos pela função *tarjanAlgorithm()* que correspondem ao output pretendido.

Na nossa solução, a parte que merece mais relevância é o funcionamento do Algoritmo de Tarjan na função *tarjanAlgorithm()*. A nossa implementação do algoritmo tem por base a utilização de cinco vectores e três variáveis auxiliares, respectivamente, de nome *visited*, que é utilizado para guardar a informação sobre se um vértice já foi visitado; *discovery*, que é utilizado para guardar a informação sobre o tempo de descoberta de um vértice; *low*, que é utilizado para guardar a informação sobre o menor valor de descoberta de outro vértice atíngivel por um arco para trás ou de cruzamento; *parent*, que é utilizado para guardar a informação sobre o identificador do vértice-pai de um vértice e, por fim, *ap*, que é utilizado para guardar a informação sobre se um vértice é ou não um ponto de articulação, relativamente às variáveis, são elas: *countAP*, número de pontos de articulação, *Min* e *Max*, menor e maior identificador entre os pontos de articulação.

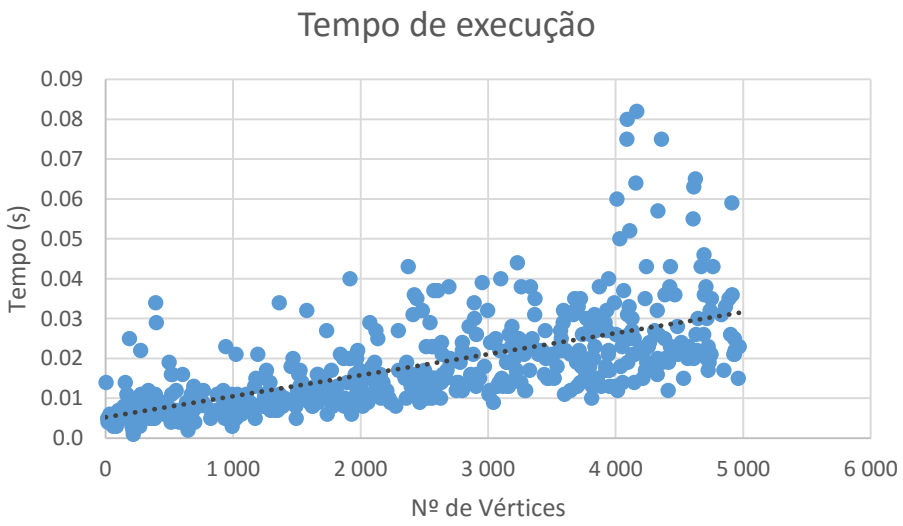
O nosso algoritmo corre um total de N vezes recorrendo a recursividade, sendo N o número de vértices. No interior da função é verificado através de um ciclo for se, para cada vértice, os seus vértices adjacentes já foram visitados. Em caso negativo, é incrementada a variável que conta os filhos do vértice-pai, o vector *parent* é actualizado e é corrida a função *tarjanAlgorithm()* no vértice. De seguida, dá-se a actualização do vector *low* no que diz respeito ao vértice-pai e verifica-se se este é um ponto de articulação, se sim, o vector *ap* é actualizado com essa informação, a variável auxiliar *countAP* é incrementada e é verificado se o identificador do vértice corresponde ao menor ou maior identificador até ao momento, que caso se verifique irá actualizar as variáveis *Min* e/ou *Max*.

Análise Teórica

O nosso programa é executado em tempo linear ao tamanho do problema. Sendo assim, a complexidade do nosso programa é:

- Função *main()* - O(V + E), sendo V o número de vértices e E o número de ligações
- Funções de inicialização dos vectores auxiliares - O(V)
- Função *tarjanAlgorithm()* - O(V + E)

Análise Experimental



Vértices: 100

Ligações: 208

N° de Pontos de Articulação: 5

Tempo de execução: 0.001s

Vértices: 1000

Ligações: 3107

N° de Pontos de Articulação: 18

Tempo de execução: 0.006s

Vértices: 5000

Ligações: 27093

N° de Pontos de Articulação: 1

Tempo de execução: 0.016s

Vértices: 15000

Ligações: 82949

N° de Pontos de Articulação: 3

Tempo de execução: 0.116s

Vértices: 50000

Ligações: 150440

N° de Pontos de Articulação: 5289

Tempo de execução: 0.197s

Vértices: 150000

Ligações: 421614

N° de Pontos de Articulação: 17061

Tempo de execução: 0.438s

O nosso projecto passa a todos os 16 testes fornecidos.
Para verificar a linearidade do tempo de execução do nosso programa, criámos 500 grafos com um número de vértices aleatório entre 2 e 5000 e testámos o tempo de execução deste para cada um deles, obtendo os resultados representados no gráfico acima. Para complementar este gráfico, criámos também 6 grafos com o número de vértices acima assinalado e testámos de novo o tempo de execução do programa para cada um deles, obtendo os resultados apresentados onde podemos verificar uma linearidade do tempo de execução de acordo com o número de vértices do grafo.

Pesquisa e suporte para a elaboração do nosso algoritmo:
https://en.wikipedia.org/wiki/Biconnected_component