Análise e Síntese de Algoritmos

1º Projeto 2015/2016 | Grupo 022 | 78974 | 82448

# Descrição do Projeto

O problema abordado neste projeto corresponde a encontrar as chamadas “pessoas fundamentais” numa rede. Suponhamos a existência de 3 pessoas, A, B e C, tal que A conhece B e B conhece C. A transmissão de informação entre A e C faz-se através de B. Se esta for removida, deixa de haver essa passagem de informação. Generalizando, quando uma pessoa fundamental é removida, a transmissão de informação deixa de existir entre duas áreas de uma rede. Representando esta situação através de um grafo conexo não dirigido, em que pessoas são representadas por vértices e ligações entre elas por arestas, o problema resume-se a encontrar os pontos de articulação do grafo. A remoção de um ponto de articulação do grafo causa a separação do grafo inicial em dois grafos conexos separados.

# Implementação do Programa

O programa foi implementado na linguagem C, apenas com a utilização da biblioteca <stdio.h>.

## Implementação da estrutura de dados

O problema acima descrito foi implementado com a utilização de grafos não dirigidos, representados por listas de adjacências.

A implementação da lista de adjacências foi feita com base num vetor primário para representar cada um dos vértices, e uma lista simplesmente ligada para representar os arcos.

São definidas as operações de inicialização do grafo, criação de arcos, e inserção dos arcos.

## Algoritmo de procura de pontos de articulação

A ideia base para o algoritmo é a utilização de uma DFS. Numa árvore DFS, um vértice **v** é pai de **u** caso **u** seja descoberto por **v**. Um vértice **v** é um ponto de articulação se:

É a raiz da árvore DFS e tem pelo menos 2 filhos;

Não é a raiz, mas tem um filho **u** tal que nenhum vértice na subárvore em que **u** é raiz tem um arco de retorno para um dos ancestrais do vértice **v**.

Para se verificar os pontos de articulação do grafo faz-se então uma passagem do grafo com 3 vetores auxiliares:

Um vetor **parent[]** onde **parent[u]** guarda o predecessor de u;

Um vetor **discoveryTime[]** para guardar o tempo de descoberta de cada vértice;

Um vetor **low[]** em que **low[u]** representa o menor tempo de descoberta de qualquer vértice que está ou na subárvore de raiz em **u** ou ligado a um vértice nessa subárvore por um arco de retorno.

Cada vértice tem também um contador dos seus filhos. Caso **parent[u]==NIL** e o contador tenha 2 ou mais filhos, estamos perante o primeiro caso.

Para o segundo caso, por cada nó **u** da árvore, precisamos de averiguar se existe algum vértice com menor tempo de descoberta que pode ser atingido na subárvore com raiz em u. Caso **low[u] >= discoveryTime[u]**, quer dizer que não existe nenhum caminho de um dos filhos de **u** até um dos seus antecessores e, portanto, **u** é um ponto de articulação.

## Função main

A função main lê do stdin o número de vértices com que é necessário inicializar o grafo e o número de arestas, que corresponde ao número restante de linhas de input que serão lidas. Cada uma destas linhas é então transformada numa aresta e adicionada ao grafo. Terminado este passo, corre o algoritmo de busca de pontos de articulação, terminando por enviar para o stdout o número de pontos de articulação, o ponto mínimo e o máximo.

# Análise Teórica

# Avaliação Experimental dos Resultados

Para procedermos à avaliação da eficiência do nosso programa, criámos 6 casos de teste de input com tamanhos definidos (Gráfico 1). Como primeiro caso, utilizámos o input do 5º teste disponibilizado aos alunos, com 30000 vértices e 58783 arestas, somando um total de 88783 vértices+arestas (V+E). O segundo, terceiro e quarto caso continham, respetivamente, 2x(V+E), 3x(V+E) e 4x(V+E) relativamente ao primeiro caso. Para aumentar ainda mais a dimensão do input, criámos ainda casos com 6x(V+E) e 8x(V+E), somando este último um total de 710271 vértices e arestas. O Gráfico 1 relaciona os casos de teste descritos com o tempo de execução do nosso programa, medido no terminal com a função *time*. Como se pode constatar, a linha de tendência é linear, o que nos confirma a complexidade linear O(|V|+|E|) do algoritmo criado neste projeto.