Relatório - 1º Projecto
Análise e Síntese de Algoritmos
2015/2016

Grupo 58
Pedro Orvalho 81151
António Lourenço 81796
19 de Março de 2016

1.Introdução

Este relatório visa a análise e solução da questão-problema: **como identificar da forma mais eficiente vértices fundamentais num grafo;** tal como proposto como primeiro projecto de Análise e Síntese de Algoritmos do segundo semestre do ano 2015/2016.

1.1.O problema

O conceito de vértice fundamental consiste num vértice de um grafo que se caracteriza como um elo de ligação entre duas árvores.
Podemos realizar um paralelo para as redes sociais, ou seja, as pessoas fundamentais numa rede social, por exemplo: a pessoa A conhece a pessoa B e a pessoa B conhece a pessoa C, logo, a pessoa B é fundamental para a difusão da informação na rede social.

1.2.*Input & Output*

Partindo de um *input* onde inicialmente é fornecido o número de vértices e arestas de um grafo em análise, seguem-se n linhas (sendo n o número de arestas do grafo) onde são indicados dois vértices que possuem uma ligação entre si. Daqui resulta um *output* onde é indicado o número de vértices fundamentais do grafo em análise seguidos dos vértices com o menor e maior índice, respectivamente.

2. Descrição da Solução

2.1. Escolha de Tecnologias e Estruturas de Dados

Após uma análise do problema concluímos que seria mais vantajoso utilizar a linguagem de programação C++ não só por questões de eficiência, mas também dado a disponibilidade de estruturas de dados já existentes. Destas estruturas é de salientar a utilização das bibliotecas do std, nomeadamente std::vector. Através desta estrutura designamos um grafo como sendo um vetor de vetores de inteiros (vector< vector<int> >), representando deste modo uma lista de adjacências.

2.2. Solução e Algoritmo

Representamos cada pessoa como um vértice e cada ligação entre duas pessoas como uma aresta, formando assim um grafo não dirigido que simboliza a rede social.

De forma a resolver o problema proposto baseamo-nos no algoritmo DFS e no Teorema de Tarjan para os pontos de articulação.

Começamos por atribuir a cada vértice do grafo as suas respectivas arestas, reservando também espaço na memória para os seguintes vetores: o vetor de inteiros do tempo de descoberta (pre[]), o vector de inteiros do tempo minimo de descoberta de um elemento da rede (low[]) e o vector de booleanos de nodes fundamentais (isFundamental[ ]).

A solução consiste na realização de uma DFS na qual, quando é visitado um novo vértice, é anotado o seu tempo de descoberta e atribuído um mesmo valor (temporariamente) para o tempo minimo de descoberta de um elemento da rede.

Para cada vértice, ainda não visitado, que forma uma ligação como o vértice a que estamos neste momento a explorar aplicamos, por suposto, o algoritmo DFS e adicionamos um “filho” ao vértice atual. Após o retorno da DFS do vértice “filho” vamos atualizar o tempo de descoberta minimo do vértice “pai”, como sendo o minimo entre o tempo de descoberta minimo do vértice “pai” e do vértice “filho”. Depois verificamos se uma de duas situações situações é verdadeira: a primeira é se o tempo de descoberta minimo do vértice “filho” é superior ou igual ao tempo de descoberta do vértice “pai” e o vértice “pai” tem um pai; a segunda é se o vértice “pai” não tem pai e o vértice “pai” tem um número de filhos superior a um. A primeira condição aplica-se aos casos gerais, para um vértice ser ponto de articulação. A segunda situação aplica-se para verificar se a raiz da árvore DFS é um ponto de articulação, juntamente a estas situações verificamos se o vértice já é um ponto de articulação. Se

verificar uma destas situações então estamos na presença de um novo ponto de articulação atualizamos o vector isFundamental[ ] na posição do vértice "pai", incrementamos o contador de pontos de articulação e verificamos se o índice do vértice é maior ou menor que o índice do ponto de articulação mais baixo e mais alto, respectivamente, e atualizamos em caso positivo.
No caso dos vértices já anteriormente visitados e que formam ligação com o vértice atual comparamos se é ou não pai do vértice atual e caso não o seja atualizamos o tempo minimo de descoberta do vértice "pai" como sendo o minimo entre o seu atual tempo minimo de descoberta e o tempo de descoberta do seu "filho".
Deste modo a nossa função int main() tratar de processar o input introduzido como já explicado em cima, e depois de atribuir as arestas aos respectivos vértices chama a função dfs(int u, int v), sendo u o vértice "pai" de v, que vai percorrer o grafo todo e descobrir todos os pontos de articulação deste. Assim após o retorno da função dfs(2), imprimimos como *output* o número de pessoas fundamentais na rede, e o número de identificador mais baixo e mais alto, respectivamente, das pessoas fundamentais à rede.

3.Análise Teórica

Analisando a complexidade temporal de cada função do nosso programa, considerando V o número de vértice e E o número de arestas do grafo, chegamos à conclusão que:

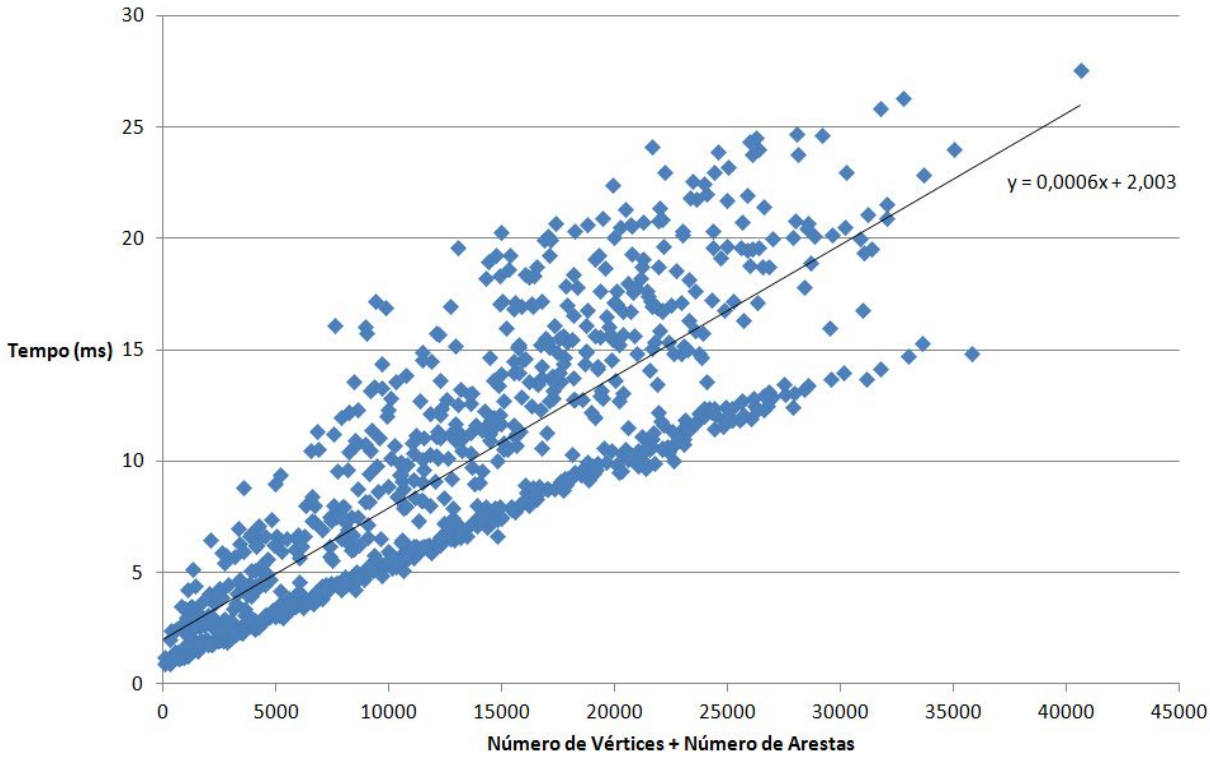
1. Função main() O(V+E):
a. A leitura do input é Θ(E);
b. A inicialização dos vetores tem uma complexidade de O(V), pois os 3 vetores usados tem comprimento V.
c. Função dfs(int v, int u) é O(V+E): pois a função percorre todos os vértices recursivamente O(V) e para cada vértice percorre as suas arestas, logo verifica cada aresta 2 vezes, 2E, logo O(E), o que resulta O(V+E).

A complexidade temporal do programa é a mesma que a da função main() que é O(V+E).
A complexidade espacial é também O(V+E). Visto que criamos um grafo, V+2E, que ocupa O(V+E), e 3 vetores, dois de inteiros e um booleano, os três de tamanho V logo ocupam, cada, O(V).

4.Avaliação Experimental dos Resultados

A fim de realizarmos uma avaliação experimental dos resultados decidimos que seria vantajoso criar um gerador de grafos. Foram então gerados cerca de 900

grafos completamente aleatórios tendo apenas, como limite, um máximo de 5000 vértices. Corremos o programa sobre cada um dos grafos e, utilizando a ferramenta do linux perf, obtemos os tempos de execução.
Por fim, tratamos os dados no excel do qual conseguimos obter o seguinte gráfico:



Podemos então concluir que os resultados experimentais estão de acordo com os pressupostos teóricos. O nosso algoritmo tem uma complexidade O(V + E), ou seja, o tempo de execução aumenta de forma linear numa relação com o número de vértices somado ao número de arestas.

5. Referências utilizadas

- http://stackoverflow.com/questions/15873153/explanation-of-algorithm-for-finding-articulation-points-or-cut-vertices-of-a-gr
- http://www.geeksforgeeks.org/articulation-points-or-cut-vertices-in-a-graph/
- https://en.wikipedia.org/wiki/Biconnected_component
- https://github.com/kartikkukreja/blog-codes/blob/master/src/Articulation%20Points%20or%20Cut%20Vertices%20with%20DFS.cpp
- https://www.youtube.com/watch?v=_nEaVxFvXO4
- CLRS, Introduction to Algorithms - 3rd Edition (2009)