TÉCNICO LISBOA

Análise e Síntese de Algoritmos

Relatório 1º Projeto (2015/2016)

Grupo 50

Rafael Belchior (80970), João Calisto (81329)

19 de Março de 2016

Introdução

O problema proposto é referente à partilha de informação (ligação) entre vários utilizadores de uma rede social, nomeadamente descobrir as pessoas fundamentais numa rede. Há pessoas consideradas fundamentais, sendo que uma pessoa é considerada fundamental se o único caminho para a partilha de informação entre outras duas pessoas r e s passa necessariamente por p. Este problema pode resumir-se a encontrar os pontos de articulação de um grafo **[1]**.

Descrição da Solução

Para resolver o problema proposto foi utilizada uma versão modificada da DFS (*Depth-first search*). Foram utilizadas as convenções dadas nas aulas teóricas para algumas propriedades dos vértices, no âmbito da pesquisa DFS: no início nenhum vértice foi visitado e o seu predecessor é NULL (NIL).

Visto uma ligação ser uma relação bidirecional, a escolha foi representar um grafo não dirigido, recorrendo a listas de adjacência. A estrutura Grafo contém o maior e menor identificador de uma pessoa fundamental e um vetor de *nodelinks*, sendo que cada índice representa um vértice. Cada *nodelink* tem uma estrutura *Edge*, cuja função é representar os arcos entre o próprio vértice e os restantes.

Lê-se do *standart input* o número de vértices (pessoas) e arcos (ligações). O terceiro número lido corresponde ao vértice em que se aplicará o algoritmo desenvolvido. A função getFundamentals, que corresponde à solução desenvolvida, e recebe como argumentos o grafo em questão, a lista de nodelinks e o vértice inicial.

O algoritmo que leva à solução funciona da seguinte forma: para cada vértice adjacente ao inicial, se ainda não foi visitado, vai-se aplicar o algoritmo recursivamente. O número de filhos é incrementado e, caso o low do vértice adjacente seja maior ou igual à profundidade da raiz, estamos perante um ponto de articulação. O valor de low é atualizado. Caso o um vértice adjacente já tenha sido visitado, o valor low é atualizado. Por fim, verifica-se o caso da raiz.A função *getFundamentals* foi baseada na

implementação do pseudo-código sugerido na Wikipédia **[2]**, e também baseada no pseudo-código sugerido nas aulas de ASA 2015/2016, na parte da implementação de uma *DFS*.

Análise Teórica

Para facilitar a análise teórica da complexidade, podemos dividir o programa em 2 partes. A primeira parte será a construção do grafo, e a segunda parte será a execução do algoritmo.

A primeira parte é toda efetuada em tempo constante uma vez que inserir na lista é O(1). A segunda parte é executada em tempo linear, O(V+E) em que V é o número de vértices e E o número de arcos, correspondente à complexidade de uma pesquisa *DFS*.

Pseudo-código da função getFundamentals

```
GetArticulationPoints(i, d)
   visited[i] = true
   depth[i] = d
   low[i] = d
   childCount = 0
   isArticulation = false
                for each ni in adj[i] Chamado tantas vezes quanto o número de vértices
      if not visited[ni]
         parent[ni] = i
         GetArticulationPoints(ni, d + 1) Chamada recursiva
         childCount = childCount + 1
         if low[ni] >= depth[i]
            isArticulation = true
         low[i] = Min(low[i], low[ni])
      else if ni <> parent[i]
         low[i] = Min(low[i], depth[ni])
   if (parent[i] <> null and isArticulation) or (parent[i] == null and childCount > 1)
      Output i as articulation point
```

Com base na complexidade das divisões feitas, podemos concluir que o programa é executado em tempo linear, com uma complexidade O(V+E).

TÉCNICO LISBOA

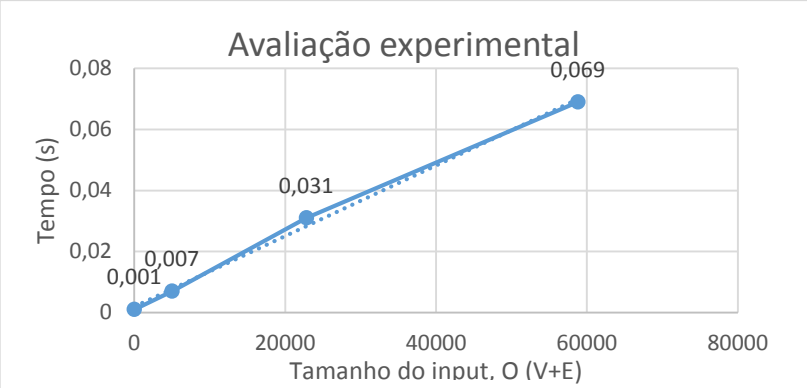
Avaliação Experimental

Para a avaliação experimental, foram feitos vários testes, sob diferentes condições. Variou-se o tamanho do input. O tempo de execução foi obtido com o comando *time*, e nos valores apresentados foi usada a média temporal de três ensaios para cada um dos inputs, para cada teste.

Teste 1 – Testes públicos fornecidos

Analisando a execução do programa com os inputs disponibilizados na página da cadeira, concluiu-se que o crescimento do tempo de execução é praticamente linear, como esperado pelo estudo realizado Análise Teórica.

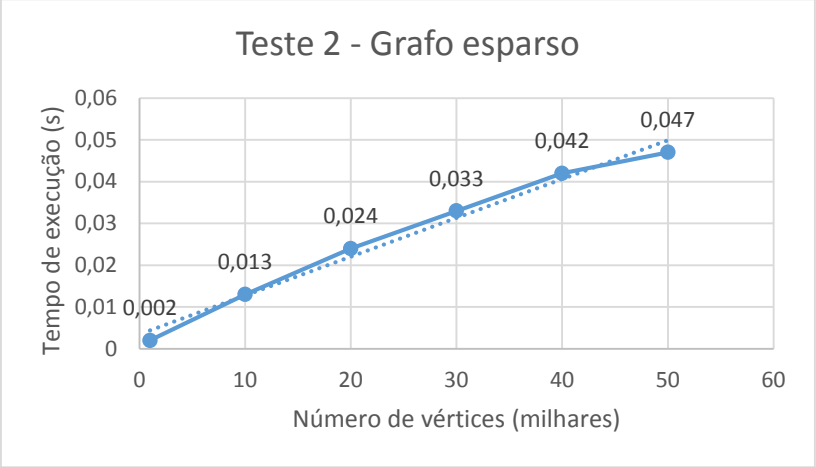
O seguinte gráfico mostra a curva que melhor se ajusta aos resultados experimentais.



Teste 2 – Grafo esparso

No 2º teste o programa foi testado com um grafo esparso, com cada vértice ligado ao próximo. Analisando a execução do programa, concluiu-se que o crescimento do tempo de execução é praticamente linear, como esperado pelo estudo realizado Análise Teórica.

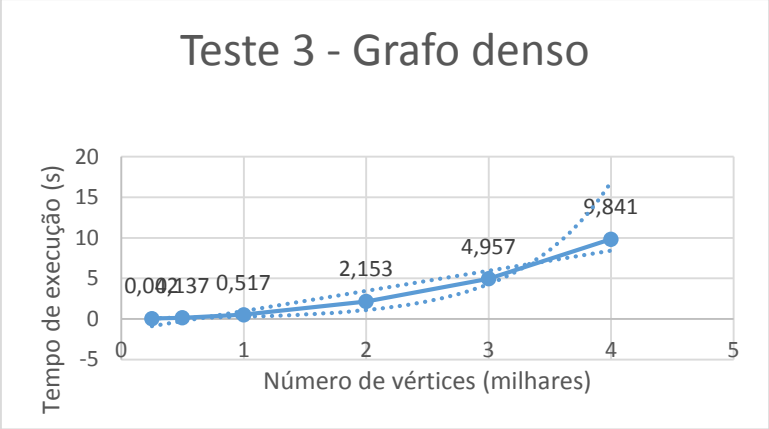
O seguinte gráfico mostra a curva que melhor se ajusta aos resultados experimentais.



Teste 3 – Grafo denso

Para o 3º teste o programa foi testado com um grafo denso em que cada vértice tem arcos para todos os outros. Como o algoritmo é O(V + E) e E = V² o pior caso passa então a ser V² concluiu-se que o crescimento do tempo de execução é praticamente quadrático.

O seguinte gráfico mostra a curva que melhor se ajusta aos resultados experimentais.



Referências

[1] https://pt.wikipedia.org/wiki/V%C3%A9rtice_de_corte_(teoria_dos_grafos)

[2] https://en.wikipedia.org/wiki/Biconnected_component