Análise e Síntese de Algoritmos

Relatório do 1º Projecto

Introdução

Com este projecto foi introduzido o problema da identificação de pessoas fundamentais na transmissão de informação numa rede, sendo uma pessoa fundamental w tal que se o único caminho entre duas pessoas u e v, com $w \neq u$ e $w \neq v$, passa necessariamente por w.

Sendo a rede representada por um grafo conexo não dirigido (a partilha de informação é bidireccional) no qual cada vértice corresponde a uma pessoa e cada aresta representa a partilha de informação entre duas pessoas, o problema traduz-se a identificar os pontos de articulação de um grafo.

É-nos fornecido um ficheiro de input com informação sobre o grafo em questão, sabendo premeditadamente o número de vértices V e arestas E (expostos na primeira linha do ficheiro). As restantes E linhas contêm 2 vértices cada uma, um de origem e outro de destino, representando as arestas do grafo.

Descrição da Solução

A solução do problema foi concebida utilizando a linguagem de programação C, com a qual implementámos um grafo, utilizando listas de adjacências. Esta implementação consiste em manter, para cada vértice do grafo, uma lista de todos os outros vértices com os quais este partilha uma aresta (definição de adjacência).

A direção é representada através da presença de um nó nessa lista: Num grafo não dirigido é adicionada uma aresta nas listas de adjâcências dos vértices de origem e destino enquanto que num grafo dirigido a adjacência apenas é adicionada à lista de adjacências do vértice de origem (indicando que a aresta tem sentido do vértice de origem para o vértice de destino).

A ideia básica do algoritmo é: uma pesquisa em profundidade que começa a partir de um vértice arbitrário, esta pesquisa é feita exatamente uma vez a cada vértice do grafo, recusando-se a rever qualquer vértice já explorado.

Tomámos a decisão de implementar uma adaptação do Algoritmo de Tarjan de modo a determinar se um vértice é ou não um ponto de articulação, e como tal, tivemos em consideração 1 de 2 critérios:

1. Um vértice u é um ponto de articulação se se tratar da raíz da procura (i.e. não ter predecessor) e tiver 2 ou mais adjacências/sucessores independentes.
2. Um vértice u é um ponto de aritculação se, não sendo a raíz da procura, ter um sucessor/adjacência v tal que nenhum vértice na sub-árvore com raíz em v tem uma *back edge* (na árvore DFS) para um dos predecessores de u.

O primeiro critério é algo evidente, uma vez que um vértice sem predecessor que tem 2 sucessores/adjacências ou mais independentes, significa que esses sucessores fazem parte de componentes que apenas comunicam através do vértice em questão, não existindo qualquer outro caminho pelo meio.

O segundo critério é facilmente verificável através do *low time* dos vértices, i.e., numa DFS em u, ao encontrar um vértice adjacente v que já fora visitado anteriormente, podemos concluir que a aresta (u,v) é uma *back edge*, havendo portanto outro caminho que lide a u. Deste modo, propaga-se o *low time* do vértice v até atingir um vértice w no qual se verifique que o tempo de descoberta é inferior ou igual ao de uma adjacência, confirmando que não existirá uma *back edge* de volta para um dos antecessores de w, declarando o vértice como fundamental.

Análise Teórica

Pseudo-código do algoritmo implementado.

```
AP_Tarjan(G)

 1.  articulationPoints = 0
 2.  apMin = ∞
 3.  apMax = ∞
 4.  time = 0
 5.  for each vertex u ∈ V[G]
 6.      do d[u] = π[u] = ∞
 7.  for each vertex u ∈ V[G]
 8.      do if d[u] = ∞
 9.            then DFS_Tarjan(u)


DFS_Tarjan(u)

 1.  childCount = 0
 2.  isArticulationPoint = false
 3.
 4.  d[u] = low[u] = time
 5.  time = time + 1
 6.
 7.  for each vertex v ∈ Adj[u]
 8.      do if d[v] = ∞
 9.         then π[v] = u;
10.              childCount = childCount + 1
11.              DFS_Tarjan(v);
12.
13.              if d[u] ≤ low[v]
14.                 then isArticulationPoint = true
15.              low[u] = min(low[u], low[v])
16.      else if v ≠ π[u]
17.              then low[u] = min(low[u], d[v])
18.
19.  if (pi[u] = ∞ && childCount ≥ 2)
20.      || (pi[u] ≠ ∞ && isArticulationPoint)
21.     then articulationPoints = articulationPoints + 1
22.          if (apMin = ∞ || apMax = ∞)
23.             then apMin = apMax = u
24.          else
25.             do apMin = min(apMin, u)
26.                apMax = max(apMax, u)
```

Em termos de complexidade, o algoritmo equipara-se ao Algoritmo de Tarjan, sendo a sua complexidade O(V + E). Em termos de inicialização, é O(V), percorrendo os vetores dos tempos de descoberta e dos predecessores de cada vértice uma vez e inicializando os mesmos.

A função recursiva DFS_Tarjan é invocada sobre cada adjacência do vértice a ser explorado (linha 11). Adicionalmente, a função apenas é invocada no caso do vértice não ter sido explorado, obtendo uma complexidade de O(V) para esta função.

Finalmente, as listas de adjacências de cada vértice são analisadas apenas uma vez, percorrendo todas as arestas do grafo uma única vez, o que resulta numa complexidade de Θ(E).

Avaliação Experimental de Resultados

Inicialmente, sujeitámos a solução aos cinco testes disponibilizados pelo corpo docente, tendo aparentemente passado aos testes sem problemas, uma vez que, o output do programa não diferia do esperado.

De seguida, fizemos a submissão no sistema de avaliação mooshak, o qual sujeitaria a solução aos testes que nos foram fornecidos acrescido de testes mais rigorosos, obtendo resultados satisfatórios.

Por fim, testámos o tempo de execução do programa e comparámos os resultados com as expectativas que tinhamos, tendo em conta a complexidade do algoritmo, e verificámos que está de acordo com o esperado. Tomámos um exemplo, dados os testes publicados pelo corpo docente.

Utilizando a ferramenta *time* presente em sistemas linux, cronometrámos várias vezes o programa com os inputs t04.in e t05.in. Concluímos que, dada a ordem da complexidade de ambos apresentar um factor de aproximadamente dois, i.e., V + E de t04.in é cerca de metade de t05.in, o tempo de execução do programa sujeito ao teste t04.in foi também cerca de metade do de t05.in, verificando a linearidade do algoritmo.

Referências

- Articulation Points Graph Algorithm, *on* YouTube. https://www.youtube.com/watch?v=2kREIkF9UAs, by Tushar Roy.
- Explanation of Algorithm for finding articulation points or cut vertices of a graph, *in* Stack Overflow. http://stackoverflow.com/questions/15873153/explanation-of-algorithm-for-finding-articulation-points-or-cut-vertices-of-a-gr
- Articulation Points (or Cut Vertices) in a Graph, *in* Geeks for Geeks. http://www.geeksforgeeks.org/articulation-points-or-cut-vertices-in-a-graph/
- Introduction to Algorithms, Third Edition: Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein September 2009 ISBN-10: 0-262-53305-7; ISBN-13: 978-0-262-53305-8