INSTITUTO SUPERIOR TÉCNICO

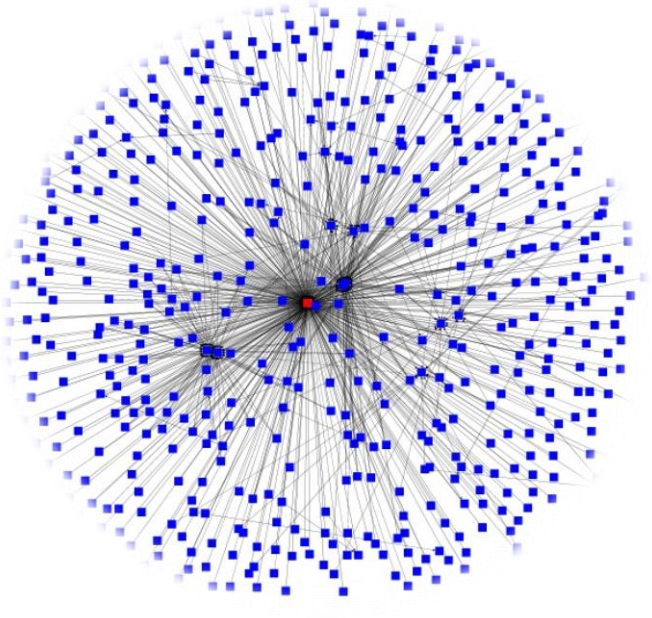
Análise e Síntese de Algoritmos
2015/2016

Relatório 1º Projeto

Grupo 97

Carolina Inês Xavier - 81172

Inês Leite - 81328



Introdução:

O seguinte relatório aborda a solução e respetivos testes e análise ao problema proposto como primeiro projeto de Análise e Síntese de Algoritmos do segundo semestre do ano escolar 2015/2016.

O problema trata a difusão de informação em redes sociais que acontece através da comunicação entre utilizadores. Assim, o objetivo é identificar quais as pessoas fundamentais na rede, sendo que uma pessoa p é considerada fundamental se o único caminho para a partilha de informação entre outras duas pessoas r e s passa necessariamente por p (onde p!= r e p!= s).

Através de um *input* com o número de pessoas e o número de ligações de partilha, seguidos da identificação das ligações individuais entre duas pessoas, o programa resultante tem de apresentar como output o número de pessoas fundamentais na rede, bem como os identificadores mais alto e mais baixo deste conjunto de pessoas.

Abordagem inicial:

Primeiramente, tivemos de optar pela escolha da linguagem de programação na qual iríamos escrever o código, optámos por C++.

Para conceber o programa criámos um mapeamento entre os conceitos:

- Rede de utilizadores para grafo não dirigido pois na difusão de informação em redes sociais, quando há uma ligação de partilha entre duas pessoas, a informação pode ser transmitida em ambos os sentidos;
- Pessoa para vértice do grafo e consequentemente uma pessoa fundamental para ponto de articulação do grafo (vértice de um grafo tal que a remoção deste vértice resulta num grafo desconectado).

Numa primeira análise a este problema pensámos numa solução mais simples para o resolver, que consiste em retirar cada um dos vértices do grafo individualmente e por cada vértice retirado aplicar o algoritmo DFS, de forma a ver se o grafo continua conectado. Caso não continue o vértice é um ponto de articulação.

No entanto, esta solução apresenta uma complexidade O(V(V+E)) e por isso não é uma solução eficiente para o problema dado. Optámos então por utilizar uma aplicação diferente do algoritmo DFS, que permite encontrar pontos de articulação, que graças à sua complexidade O(V+E) é uma melhor e mais eficiente opção para resolver este problema.

Descrição da Solução:

Do *input* retiramos o número de vértice e o número de arestas que o grafo vai ter. Com o número de vértices podemos inicializar todas as nossas estruturas de dados sendo uma delas um *array* de listas que representa o grafo, em que cada uma das posições do *array* corresponde a um vértice do grafo e a lista que existe nessa posição contém os vértices adjacentes a este vértice.

Com o número de arestas sabemos quantas linhas temos que ler do *input* que contém as ligações, que serão adicionadas no *array* de listas.

Depois de termos o grafo representado, de forma a identificar os pontos de articulação aplicamos o algoritmo DFS que nos permite obter os pontos de articulação, começando por visitar o vértice 1 (posição 0 no *array* de listas):

- O vértice é marcado como visitado e é-lhe atribuído um tempo de descoberta e um valor *low*, inicialmente igual ao valor de descoberta;
- A lista com os vértices adjacentes é percorrida;
    1. Caso o vértice adjacente tenha sido visitado:
        - O número de vértices filhos é incrementado;
        - O vértice é adicionado como pai do vértice adjacente, num array;

- O algoritmo é aplicado ao vértice adjacente;
- O valor *low* do vértice é atualizado, passa a ser o valor mínimo entre o *low* do vértice atual e o *low* do vértice filho;
- Determina-se se o vértice é ou não um ponto de articulação, o que acontece quando se regista um dos seguintes casos:
  - O vértice não tem pai (é a raiz da árvore) e tem mais que um vértice filho;
  - O vértice tem pai e o seu tempo de descoberta é menor que o *low* do vértice filho.
- Se o vértice for um ponto de articulação é marcado como tal num *array*.

2. Caso o vértice adjacente não tenha sido visitado e não seja pai do vértice:
   - O valor low é atualizado, passa a ser o valor mínimo entre o low do vértice atual e o low do vértice filho;

Quando o algoritmo termina já temos todos os pontos de articulação do grafo. O que nos permite gerar o output esperado: o número de pontos de articulação do grafo e desses pontos de articulação os pontos que contêm o identificador máximo e mínimo.

Análise Teórica:

O ciclo de inicialização das estruturas de dados tem uma complexidade O(V).

O ciclo que cria as ligações entre vértices do grafo tem complexidade O(E).

O algoritmo DFS que nos permite obter os pontos de articulação é executado em O(V+E), uma vez que são visitados todos os vértices e os respetivos arcos.

Na geração do output, o ciclo em que calculamos o número de pontos de articulação tem complexidade O(V) e cada uma das funções que devolve o identificador mais baixo e mais alto tem complexidade O(V).

Concluímos assim que a nossa solução tem uma complexidade O(V+E), o que faz dela uma solução eficiente.

Avaliação Experimental dos Resultados:

Submetemos o nosso programa aos testes dados na página da cadeira, aos quais passou com sucesso. Na tabela é possível observar o número de vértices do grafo (correspondente ao número de pessoas), o número de arestas do grafo (correspondente ao número de ligações) e o tempo de execução (obtido com o comando Unix time).

Testes	Número de vértices	Número de arestas	Tempo de execução
1	8	7	< 0,01
2	10	9	< 0,01
3	500	5000	< 0,01
4	20000	22803	~ 0,025
5	30000	58783	~ 0,048

Para além destes testes, recorremos a um programa de geração de grafos, que usámos para gerar vários gráficos aleatórios para testar o nosso programa. E obtivemos os seguintes resultados:

Número de vértices	Número de arestas	Número de pontos de articulação	Tempo de execução
500	1000	37	< 0.01
1500	2000	375	< 0.01
10000	15000	1846	~ 0.015
20000	35000	2202	~ 0.030
25000	40000	3717	~ 0.035
30000	40000	7692	~ 0.040
40000	50000	12153	~ 0.050

Como se pode ver pelos resultados obtidos, o tempo de execução aumentando à medida que o número de vértices e arestas também aumenta, o que era esperado tendo em conta a complexidade do algoritmo (O(V+E)).

Referências:

http://www.eecs.wsu.edu/~holder/courses/CptS223/spr08/slides/graphapps.pdf

http://www.mif.vu.lt/~valdas/ALGORITMAI/Laboratorinis_darbas2015/Uzduotys2015/16%20(BICONNECTED%20COMPONENTS)/biconnected_components.pdf

https://en.wikipedia.org/wiki/Tarjan%27s_strongly_connected_components_algorithm

http://wwwmayr.informatik.tu-muenchen.de/lehre/2012WS/algoprak/uebung/tutorial4.english.pdf