**Relatório do 1º Projeto de ASA**

**Introdução**

Este relatório descreve, no âmbito da cadeira de Análise e Síntese de Algoritmos, o desenvolvimento de um sistema de auditoria a redes de routers.

Os routers da rede são identificados por um número inteiro e as ligações entre eles são bidirecionais. Para além disso, uma rede pode ser composta por sub-redes, ligadas apenas entre si, que podem ser identificadas pelo identificador do router com maior identificador dentro da respetiva sub-rede.

A auditoria pretende recolher um conjunto de propriedades sobre a rede a analisar, das quais constam o número de sub-redes que a constituem, o identificador de cada sub-rede (dado num conjunto ordenado), o número de routers de ligação, que quando removidos da rede aumentam o número de sub-redes, e o número de routers da maior sub-rede resultante após a remoção de todos os routers de ligação.

O sistema recebe como dados iniciais o número de routers na rede, o número de ligações na rede e as ligações em si, representadas por pares de inteiros que identificam routers.

**Descrição da Solução**

O problema de auditoria a redes de routers foi traduzido para um problema de procura em grafos não dirigidos, onde a rede em si foi representada por um grafo, composto por nós (com ID inteiro) que representam routers e por arcos não dirigidos que representam ligações entre routers.

Nesta abstração, os dados da auditoria passam a ser equivalentes a encontrar:

1. O número de sub-grafos do grafo
2. O conjunto dos nós com maior inteiro em cada sub-grafo
3. O número de nós de articulação do grafo
4. O número de nós do maior sub-grafo, após a remoção dos nós de articulação

A procura destes dados foi feita através da aplicação de dois algoritmos DFS (Depth-First Search). O primeiro é uma versão modificada do algoritmo de Tarjan, aplicado a um grafo não dirigido, que é responsável por encontrar os três primeiros pontos da auditoria, e o segundo uma DFS simplificada, que determina o quarto ponto.

Para esta DFS simplificada (chamaremos DFSS), é apenas guardado o estado de descoberta de cada nó, representado por três possíveis cores: Branco para não visitado; Cinzento para visitado; Preto para visitado e finalizado.

No caso do algoritmo de Tarjan (chamaremos DFST), são guardados os tempos de descoberta de cada nó, o estado de descoberta de cada nó, os nós que são raízes da árvore DFS resultante do algoritmo, e os lows de cada nó.

Como o grafo é não dirigido, o primeiro ponto da auditoria é simplesmente dado pelo número de vezes que a DFST recomeça.

O segundo ponto é calculado durante a DFST, onde por cada recursão do algoritmo o ID do nó corrente é comparado com o último maior ID encontrado. Se for maior, substitui-o. Antes da DFST recomeçar uma nova procura, o ID guardado é marcado como ID do sub-grafo. É usado parte do algoritmo de ordenação counting-sort para marcar este ID, preservando assim a ordem crescente dos IDs.

Quanto ao terceiro ponto, um nó é nó de articulação quando:

1. É a raiz de uma árvore DFS e tem dois ou mais filhos
2. Não é a raiz e não existe nenhum caminho (backedge) de um nó descendente dele para um nó acima dele

Durante a DFST, sempre que um nó é marcado a preto, é também verificado o ponto i) com recurso ao vetor de raízes. Este vetor é uma versão simplificada do vetor de predecessores, que em vez de guardar o predecessor, marca apenas se ele existe ou não. O ponto ii) é verificado através do sistema de lows do Tarjan original. Após encontrados os lows, se low[filho\_de\_u] >= d[u], então u é um nó de articulação.

A remoção de todos os nós de articulação do grafo é uma tarefa complexa que poderia aumentar a complexidade assintótica do algoritmo. Devido a isso, a solução aplicada para o quarto ponto passa por ignorar os nós de articulação que fazem parte do grafo, sem que seja necessário removê-los. Durante a DFSS, se o caminho encontrado segue para um destes nós (encontrados no ponto anterior), o caminho é ignorado. Desta forma, o estado do grafo sem os nós de articulação pode ser simulado e o quarto ponto encontrado facilmente com um contador aquando do recomeço do algoritmo.

**Análise Teórica**

A solução para o projeto foi realizada em linguagem C, devido ao maior controlo que disponibiliza sobre recursos como ponteiros e gestão de memória e devido à maior velocidade de execução.

No total do projeto, estima-se que o tempo de execução tenha uma complexidade O(V+E) e a que a memória usada siga também uma complexidade O(V+E).

Começando pelo tempo de execução, a representação do grafo em código foi feita através de uma lista de adjacências. Isto implica um tempo de inicialização de O(V) e um tempo de destruição O(E), mas também um tempo de inserção de novos arcos constante. Desta maneira, ao serem lidas do stdin as ligações, o ciclo de leitura e inserção demorará tempo O(E).

Durante os algoritmos, ambas as DFSs respeitam a complexidade de tempo O(V+E) e na escrita dos dados da auditoria no stdout é corrido um ciclo com tempo O(V) relativo ao segundo ponto. No final, ficamos com O(4V+4E) = O(V+E).

Quanto ao espaço em memória, a lista de adjacências ocupa O(V+E), enquanto os algoritmos DFST e DFSS ocupam respetivamente O(5V) e O(2V) (contando com os vetores de estado das DFSs alocados e o stack de recursão em si, que no máximo é O(V) para cada caso). Adicionalmente são alocados dois vetores com espaço O(V), um para guardar os nós de articulação e outro para guardar os IDs de sub-grafos de forma ordenada. No final, ficamos com O(9V+E) = O(V+E).

**Análise Experimental**

De seguida são apresentados dois gráficos que representam a análise experimental da complexidade em termos de tempo de memória respetivamente:

**Fontes**

* **Introduction to Algorithms, Third Edition:** Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein
* Slides IAED1718: iaed18aula24.pdf e iaed18aula25.pdf
* **Definição de nó de articulação: https://en.wikipedia.org/wiki/Biconnected\_component**