Análise e Síntese de Algoritmos (1º Projeto)

Grupo 57

Isabel Soares (89466)

Tiago Afonso (89546)

1. Breve introdução

Foi-nos proposto desenvolver um projeto no âmbito da cadeira de Análise e Síntese de Algoritmos que se baseia na identificação de sub-redes e em reconhecer routers que levariam ao aumento do número de sub-redes caso estes tenham sido atacados ou desligados.
Deste modo, apresentamos o problema como um grafo não dirigido onde utilizamos um algoritmo de procura de componentes fortemente ligadas nomeadamente o algoritmo de Tarjan lecionado nas aulas.

2. Descrição da solução

A implementação do programa foi realizada em linguagem C.

Neste grafo, os vértices(N) correspondem aos routers e as ligações entre eles(M) correspondem às arestas.

Começámos por criar uma lista de adjacência que contém toda a informação sobre a rede no grafo. De seguida, adicionámos cada uma das ligações entre routers da rede no mesmo grafo.

Para resolver o primeiro ponto (determinar o número de sub-redes), utilizámos um algoritmo DFS recursivo (**NETWORKcompsligadas**) que por cada vez que atravessa a rede, descobre uma sub-rede e coloca no array **cc** (componentes conectadas), de tamanho N, a sub-rede a que cada vértice pertence. No final da função, retorna o número de sub-redes.

Para o segundo ponto (identificadores das sub-redes) é criado um array **ids** que é preenchido pelos routers com maior identificador para cada componente conectada, que é posteriormente ordenada utilizando o QuickSort (da biblioteca stdlib.h), escolhido pela complexidade e pela pequena complexidade espacial (O(N) no pior caso).

Para o terceiro ponto (número de routers que desligados formam sub-redes), foi utilizado o Algoritmo de Tarjan, que descobre os routers que quebram a rede e assinala-os no array **brknRouters**, imprimindo de seguida a quantidade de routers partidos (**numbrokenrouters**). A função conjuga o terceiro e o quarto pontos (número de routers da maior sub-rede resultante da remoção dos routers que quebram), ao utilizar de novo o array **cc**, e colocando o valor 1 em todas as posições do array que simbolizam routers que quebram a rede. Esse array é usado na nova DFS (**NETWORKsubcompsligadas**), que ao invés de começar a

contar sub-redes do 0, começa a contar a partir de 2 (os routers que quebram têm o valor 1). É chamada a função **HighestFrequency**, que retorna a quantidade de vezes que o valor mais comum aparece num array, e ignora estes pontos, começando só a contar a partir da sub-rede com valor 2, retornando assim a maior sub-rede.

No final do programa, o array **cc** é libertado de memória, em conjunto com a rede.

3. Análise Teórica

Em relação à análise teórica do nosso algoritmo, mais concretamente relacionado com a execução de cada ciclo, tendo em conta que V é o número de vértices de um grafo (neste caso o número de routers) e E é o número de arestas de um grafo (neste caso o número de ligações temporais entre elas), temos:

- Inicialização do grafo: O(V);
- Inserção de um arco: O(1);
- Remoção de um arco: O(V);
- DFS: O(V + E);
- Algoritmo QuickSort: O(V log V);
- Algoritmo Tarjan: O(V + E);

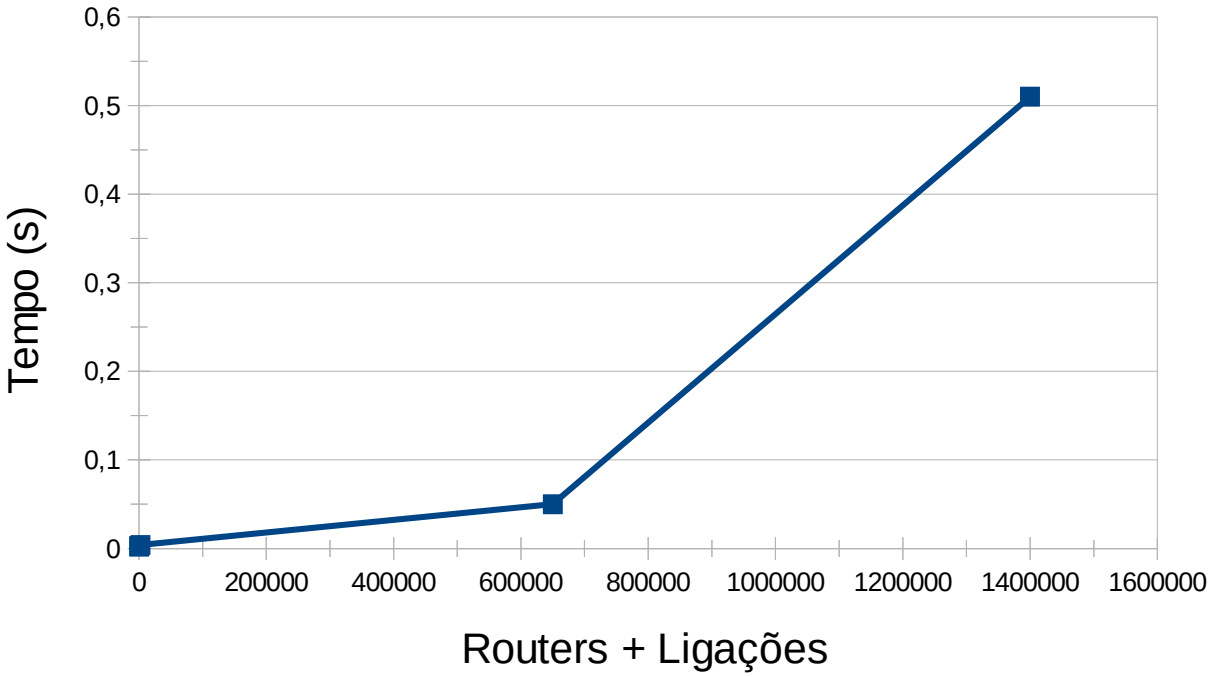
Logo, a complexidade final do nosso algoritmo é de O(V log V).

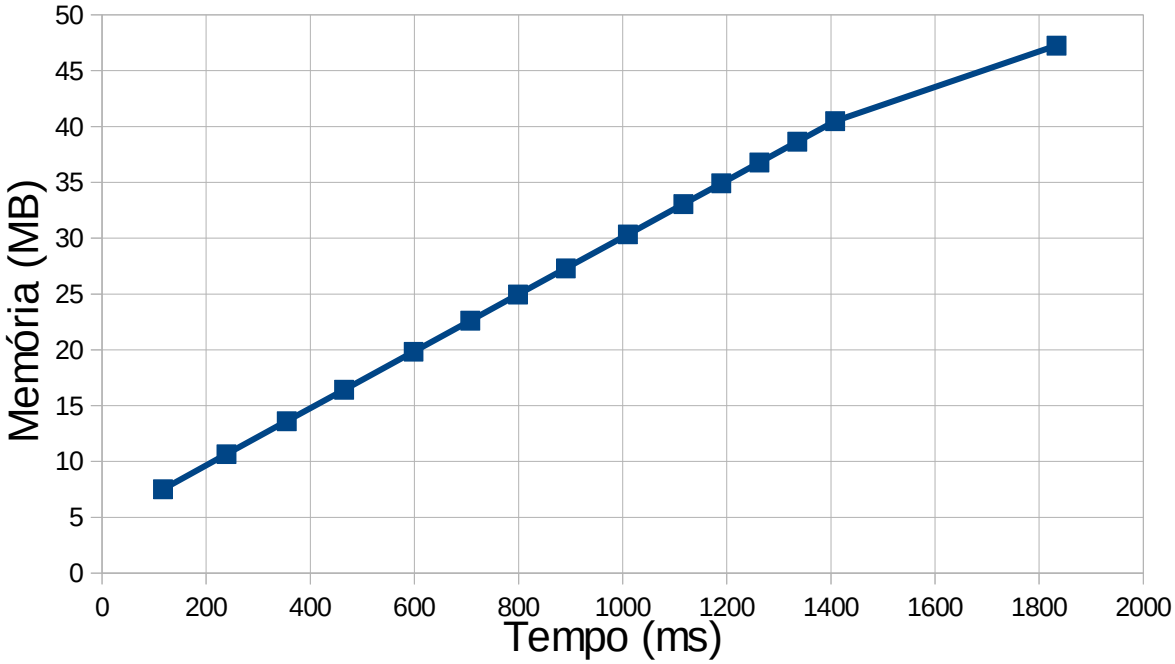
4. Análise Experimental dos Resultados

Para verificar o uso de memória e tempo de execução do algoritmo, foram testados os inputs dados como exemplo, utilizando as funções *time* e *Valgrind* do Linux.

Como se pode ver, o gasto de memória é linear, aumentando com o número de routers e ligações, não sendo tão fácil de ver no gráfico Tempo-Routers+Ligações, devido à diferença de valores usados nos exemplos e à diferença de precisão entre o comando *time* e o *Valgrind.*

V + E	time
11	0.002
20	0.002
450	0.003
2400	0.004
650000	0.05
1400000	0.51





5. Referências

- **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
- https://en.wikipedia.org/wiki/Biconnected_component
- https://www.geeksforgeeks.org/tarjan-algorithm-find-strongly-connected-components/