ANÁLISE E SÍNTESE DE ALGORITMOS

1º PROJETO – SUPERMARKET SUBNETWORKS

INTRODUÇÃO:
Para o primeiro projeto de Análise e Síntese de Algoritmos do ano 2017/2018 foi proposto aos alunos apresentar uma solução para uma divisão de rede de distribuição em sub-redes regionais de forma a que numa região seja possível qualquer ponto de distribuição enviar produtos para qualquer outro ponto da rede regional.
Para tal, é dado um input correspondente ao número de regiões e ligações entre estas, sendo as últimas, posteriormente, especificadas.

DESCRIÇÃO DA SOLUÇÃO:
A solução apresentada pelo grupo 44 encontra-se num único ficheiro, main.cpp, em C++. Esta linguagem foi escolhida pela facilidade em manusear conteúdo dentro de bibliotecas e estruturas de dados.
Foi escolhido o **algoritmo de Tarjan** por se revelar o **mais simples e eficaz** na descoberta de componentes fortemente ligados (SCCs), uma vez que apenas chama a DFS uma vez e não precisa de transpor o grafo para o cálculo destes.
Para melhor conhecimento do algoritmo referido: https://www.geeksforgeeks.org/tarjan-algorithm-find-strongly-connected-components/.
No mesmo, existe uma classe, Graph, que representa a descrição de um grafo. Para isto, foram-lhe atribuídas várias variáveis, um número de vértices correspondentes ao número de regiões a definir pelo input **(V)**, uma lista para as adjacências entre os vértices, as arestas **(E)**, uma stack e vetores auxiliares para apresentação de bridges e ainda outros para cálculo de tempos de descoberta de vértices.
Foram implementadas funções relativas a esta classe, sendo as mais relevantes **visit**, **findBridges** e ainda **orderBridges**. Existem, também, variáveis globais para contagem de componentes fortemente ligados, bridges e tempo de descoberta de um dado vértice.
É a partir da primeira e do algoritmo de Tarjan, **DFS (Depth First Search)**, aplicado nesta que são encontrados os componentes fortemente ligados e incrementado o contador criado especialmente para estes.
O procedimento de conseguir encontrar as bridges é feito através de uma análise cuidada da **lista de adjacências**, ou seja, cada lista de um determinado vértice é percorrida e caso exista um que não pertence ao mesmo componente verifica-se a existência de um caminho para outra região. Assim, o contador para bridges é incrementado. A função findBridges fica responsável por produzir estes resultados.
Finalmente, numa tentativa de apresentar as pontes encontradas de forma crescente, a função orderBridges ordena-as a partir do algoritmo pré-definido, **qsort**. Posteriormente filtra aquelas que são repetidas, isto é, caso encontre uma bridge repetida decrementa o contador. Caso ainda não tenha encontrado uma bridge igual, limita-se a imprimi-la.

O output pedido consiste apenas no número de SCCs, o total de bridges e finalmente uma sequência ordenada crescentemente pelo identificador da sub-rede ou região origem.

ANÁLISE TEÓRICA:

Sabendo que a inicialização é dada por $O(V)$, que a análise da lista de adjacências é dada por $\theta(E)$ e ainda que a chamada recursiva da função visit custa $O(V)$, temos que a complexidade do programa descrito acima é dada pela expressão: $O(|V|+|E|)$.

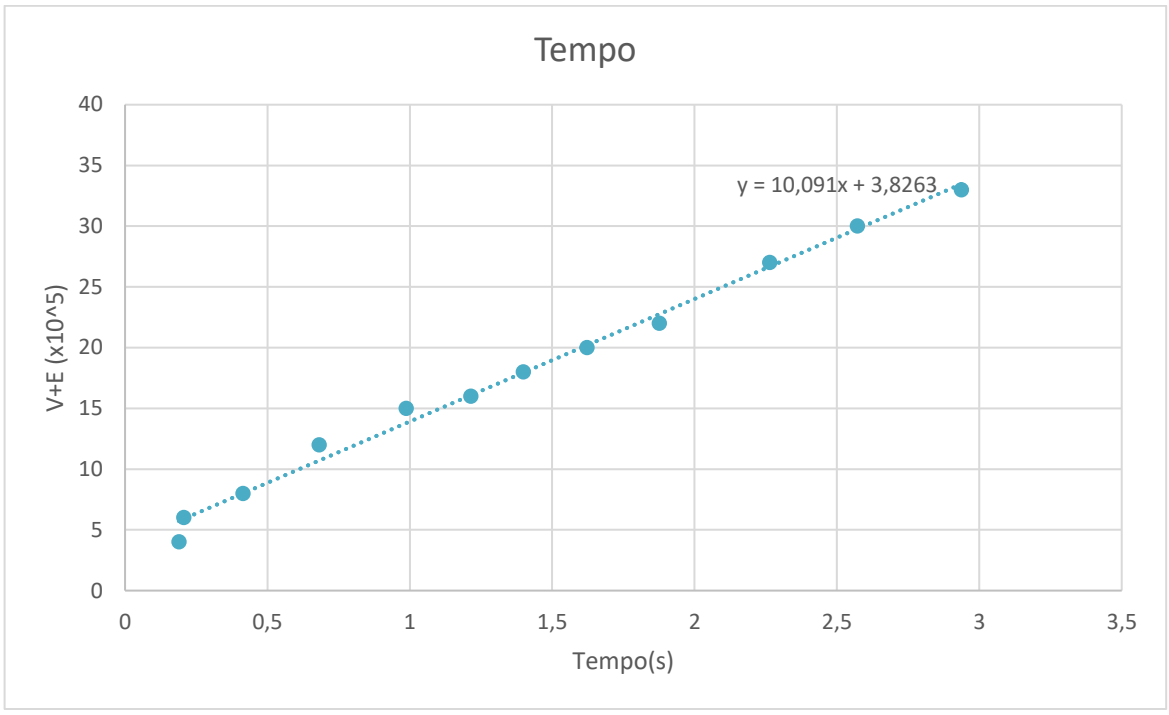
Descrição das várias funções implementadas para a realização da solução:

- **int getDiscTime(int index)**: Getter do tempo de descoberta do vértice de índice index.
- **void addVertex(const int v, const int u)**: Adiciona a um vértice(v), na lista de adjacências, uma aresta para outro vértice(u).
- **void visit(const int ind)**: função recursiva que aplica o algoritmo de Tarjan visitando vértices ainda não explorados, colocando-os na stack. No caso do vértice de índice ind já ter sido visitado procedem-se a verificações para concluir se existe um caso em que foi descoberta uma SCC, como já foi explicado no ponto anterior.
- **void findBridges(const int ind)**: Encontra as bridges a partir da lista de adjacências, uma vez que cada vértice pertence a uma componente fortemente ligada que possui um pai.
- **void orderBridges()**: Ordena as bridges encontradas pela função descrita anteriormente e seleciona, antes de imprimir, as que não se repetem e uma apenas das que repetem, decrementando o contador de pontes.
- **int main()**: função principal, como o nome indica, onde aceita o input e todas as outras funções são chamadas para gerar o output.

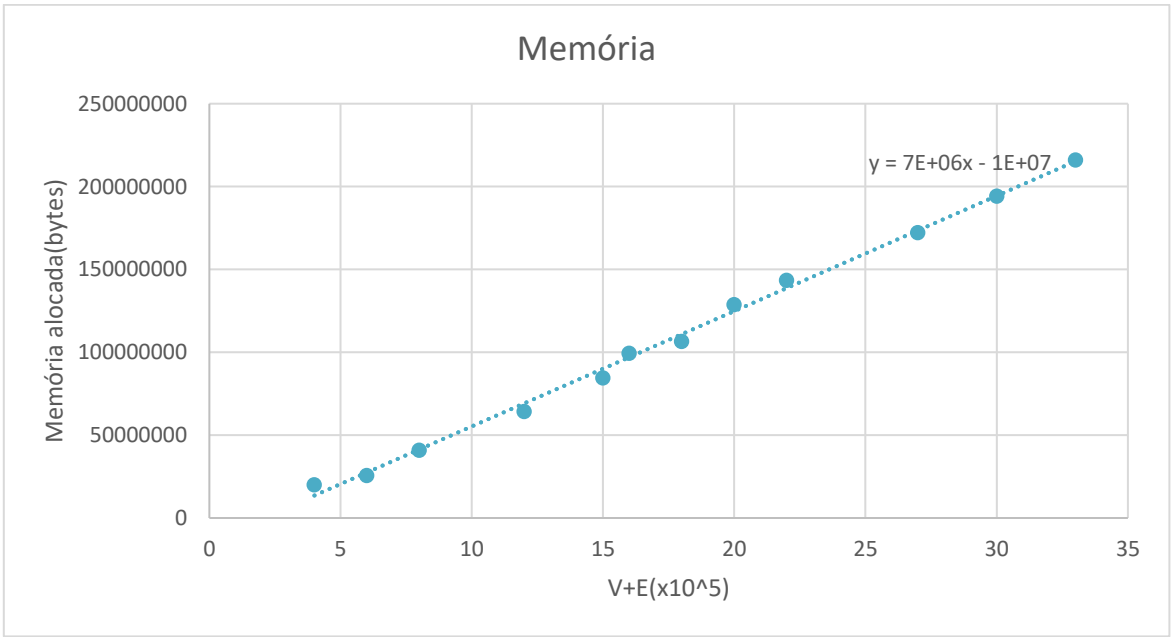
ANÁLISE EXPERIMENTAL:

De maneira a fazer uma análise experimental **o ficheiro foi submetido a vários testes aleatórios criados pelo gerador**, fornecido pelos docentes da cadeira. A partir do uso das ferramentas **Valgrind** e **Time**, foi possível obter a informação apresentada nos gráficos seguintes (Gráfico1 e Gráfico2).

Conclui-se que, em ambos, **existe um crescimento linear**, isto é quanto maior for o número de vértices e arestas, maior espaço será necessário, logo maior memória alocada assim como mais tempo é preciso para a solução ser apresentada corretamente.



(Gráfico 1 – Avaliação experimental)



(Gráfico 2 – Avaliação experimental)

REFERÊNCIAS:

Os sites usados para consulta auxiliar no processo de implementação do projeto foram os seguintes:

- https://www.geeksforgeeks.org/bridge-in-a-graph/
- https://stackoverflow.com/questions/28917290/how-can-i-find-bridges-in-an-undirected-graph
- https://www.geeksforgeeks.org/bridge-in-a-graph/
- https://www.geeksforgeeks.org/tarjan-algorithm-find-strongly-connected-components/