Relatório - 1º Projeto

ANÁLISE E SÍNTESE DE ALGORITMOS

2018/2019

Grupo al035 - Sérgio Nóbrega 86806

1. Introdução

O presente relatório de projeto é elaborado no âmbito da unidade curricular de Análise e Síntese de Algoritmos do ano letivo 2018/2019. O projeto consistiu em **identificar SCC (*Strongly Connected Components*) assim como vértices de corte de uma rede da forma mais eficiente**.

1.1. Descrição do Problema

Dado um grafo não-dirigido, queremos, primeiramente, encontrar todos os seus SCC (ou componentes conectados). Um SCC é largamente definido como um grafo em que cada um dos seus vértices é atingível a partir de qualquer outro. Identificados os SCCs, queremos cada um dos seus identificadores, definido pelo vértice com maior número dentro do SCC.

Para além disto, queremos encontrar todos os vértices de corte do grafo. Um vértice de corte é um vértice de um grafo tal que a sua remoção implica um aumento do número de SCC. É também pedido o número de vértices do maior sub-grafo resultante da remoção de todos os vértices de corte do grafo.

***Input* e Output**

O ficheiro de entrada caracteriza o grafo a ser trabalhado. É dado o número de vértices, o número de ligações e todas as ligações do grafo.

O *output* deve conter o número de SCCs do grafo introduzido, os identificadores de cada rede de forma crescente, o número de vértices de corte da rede e o número de vértices do maior sub-grafo resultante da remoção de todos os vértices de corte.

2. Descrição da Solução

2.1. Tecnologias e Estruturas de Dados

Para a resolução do problema, foi utilizada a linguagem de programação C++ por ter estruturas de dados eficientemente implementadas e com uma boa

coleção de métodos. Em particular, para este projeto, foram utilizadas estruturas e métodos do *namespace* std nomeadamente std::list e std::vector. Foram também utilizados métodos da biblioteca de C stdio, nomeadamente a função printf() e scanf().

Como complemento, foi também utilizado o valgrind para garantir que não existem *leaks* de memória.

Em termos de estruturas de dados, para a representação do grafo foi utilizado um vetor de listas (std::vector<std::list<int>>()).

2.2. Solução e Algoritmo

Para resolver o problema descrito, foram utilizados os algoritmos de Tarjan e DFS (*Depth First Search*). Foi criada uma classe Graph que contém a lista de adjacências assim como todas as estruturas de dados auxiliares aos algoritmos sendo elas: vetor dos tempos de descoberta (vector<int> d), vetor do tempo mínimo de descoberta (vector<int> low), vetor de booleanos dos vértices já visitados (vector<bool> visited), vetor de vértices de corte (vector<int> cutRouters) e vetor de vetores com os SCCs do grafo (vector<vector<int>> v_sccs).

O programa começa por criar o grafo através do ficheiro de entrada e procede à execução do algoritmo de Tarjan para o cálculo dos SCCs, à medida que estes são calculados são adicionados a v_sccs. Como queremos identificar os SCCs através do maior vértice, estamos a fazer o algoritmo do maior vértice até o primeiro (identificador 1). O algoritmo foi ligeiramente alterado para suportar simultaneamente o cálculo dos vértices de corte. Foram identificados dois casos em que os vértices são de corte:

(1) O vértice é a raiz da árvore DFS e tem dois ou mais filhos;

(2) O vértice filho tem um low superior ao seu pai.

Assim sendo, foi adicionado um argumento p que indica se o nó a ser avaliado é raiz de uma árvore DFS ou não e, dentro do tratamento de vértices não visitados em tarjanVisit() foi adicionado um *if-statement* que verifica as condições (raíz e mais de 1 filho ou filho com low superior ao pai). O resultado do algoritmo de Tarjan é um vetor de vetores que contém os SCCs do grafo e o vetor cutRouters com os vértices de corte a *true*.

O main() passa a tratar do v_sccs (vetor de SCCs), como fizemos o algoritmo do maior para o menor, a raíz de cada SCC é o identificador e os SCCs já estão ordenados (pela ordem inversa) pelo que temos apenas de percorrer o número de SCCs e imprimir o primeiro elemento de cada árvore.

É também impresso o número de vértices de corte (mantido por um contador dentro do *if-statement* falado do tarjanVisit).

Finalmente, tendo todos os vértices de corte, é executada uma DFS em que no vetor de vértices visitados são colocados todos os vértices de corte como visitados para que não sejam explorados. É passado um contador por referência que calcula o maior SCC (em número de vértices) resultante.

3. Análise Teórica

Passando à análise de complexidade da solução pensada, temos:

1. **Leitura do input e criação do grafo** - para a leitura de input desprezamos o número de vértices e ligações e na criação do grafo inicializamos 6 vetores com V entradas $\Theta(E) + O(V)$.
2. **Algoritmo de Tarjan tarjan()** - percorre todos os vértices V e percorre as suas arestas, como estamos a falar de um grafo não-dirigido, cada aresta é percorrida duas vezes. Temos ainda um conjunto de operações constantes de push, pop e Temos então $O(V+E)$.
3. **Percorre o vetor de SCCs** - percorremos o vetor de forma linear, temos no máximo $O(V)$.
4. **Algoritmo DFS fundamentalDFS()** - cria um vetor traverse com V entradas e percorre-o $O(V) + O(V)$, percorre cada vértice V e as suas arestas 2 vezes, o que nos dá $O(V+E)$. A complexidade temporal é então $O(V+E)$.

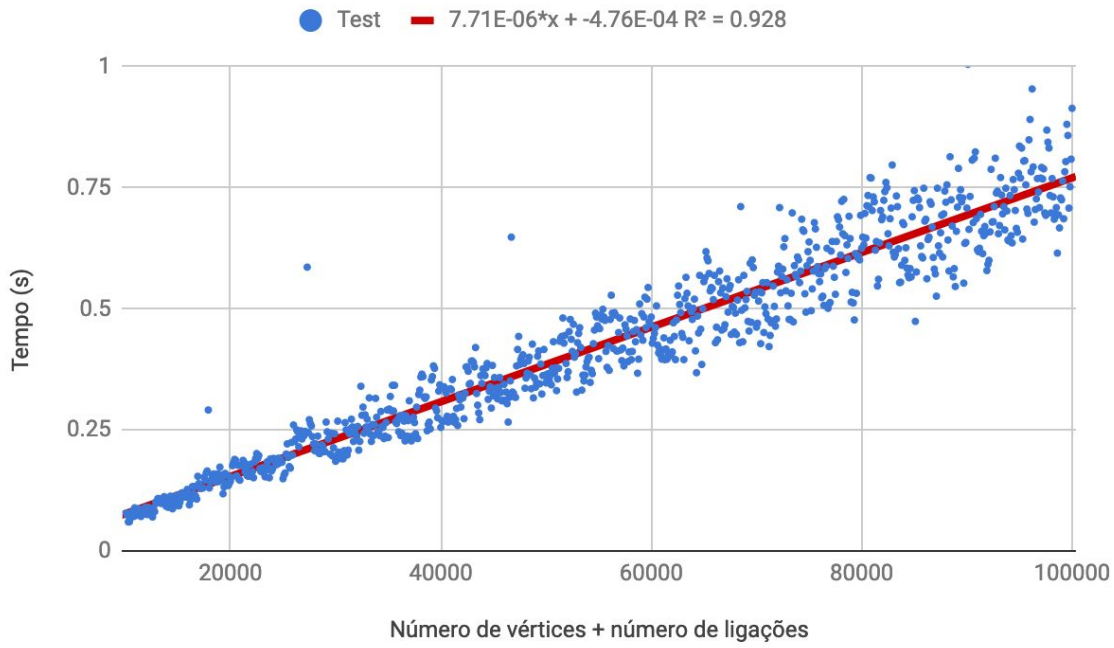
Assim sendo, a complexidade temporal da solução é $O(V+E)$.

Em relação à análise de complexidade espacial, temos também $O(V+E)$ pois a estrutura de dados mais significativa é a lista de adjacências que tem V + 2E entradas (no total).

4. Avaliação Experimental dos Resultados

Para a avaliação experimental dos resultados, foram criados 900 testes no gerador com número de vértices entre 10000 e 100000. Criou-se um script para fazer a geração dos testes e respetivo parsing dos resultados do comando time em Linux.

Fazendo *plot* dos dados obtidos numa *spreadsheet*, obtemos o seguinte resultado:



Assim sendo, podemos concluir que os resultados experimentais estão de acordo com os pressupostos da análise teórica. Como esperado, o nosso algoritmo tem complexidade temporal **O(V+E)**.

5. Referências

Cormen, Thomas H. (2009). Introduction to Algorithms. Cambridge: MIT Press.