Relatório de Análise e Síntese de Algoritmos
1º Projeto

Introdução

Neste relatório, iremos abordar a solução encontrada para o problema proposto pelo corpo docente.

O problema deste projeto consistia na criação de um programa que permitisse dividir uma rede de distribuição de supermercados em sub-redes regionais, de forma a que numa região seja possível que qualquer ponto de distribuição tenha acesso a outro da mesma sub-rede.

O input dado permitia saber o número de pontos de distribuição da rede, o número de ligação na rede de distribuição e cada uma dessas ligações. E através deste o programa tem de identificar as várias sub-redes regionais.

Descrição da solução

O programa realizado foi implementado em linguagem C. Para solucionar o problema, as redes de distribuição foram interpretadas como um grafo dirigido e cada sub rede do grafo como uma componente fortemente ligada.

1. Criou-se um grafo dirigido através do input com o número de vértices igual ao número de pontos de distribuição e sendo as arestas obtidas através da lista, inserida pelo utilizador, de ligações entre os pontos de distribuição.

2. Implementou-se o algoritmo de Tarjan para descobrir o número de componentes fortemente ligadas (SCC), ou seja, o número de sub-redes regionais que poderiam existir na rede de distribuição.

A nossa versão do algoritmo de Tarjan segue os seguintes passos:

- ➢ Percorre todos os vértices do grafo e, caso esse vértice ainda não tenha sido descoberto (d=NIL (-1)), chama a função Tarjan_Visit.
- ➢ Dentro da função Tarjan_Visit, o vértice (U), que lhe foi dado como argumento é colocado na pilha, e seguidamente, percorremos todos os vértices adjacentes a esse.
- ➢ Para cada vértice adjacente (V), se o seu tempo de descoberta for NIL ou estiver na pilha, confirmamos novamente se o vértice V já foi descoberto, e caso não tenha sido descoberto chamamos recursivamente o Tarjan_Visit com o vértice V como argumento, após o retorno da função atualiza-se o valor de low do vértice U para o mínimo do low entre U e o seu adjacente V.

- ➢ A recursão acaba quando for encontrada uma raíz (tempo de descoberta igual ao valor de low), e nesse momento, incrementamos o contador de SCC.
- ➢ Seguidamente, enquanto topo da pilha for diferente do vértice U fazemos pop( ), e vamos guardando em cada vértice, a raíz e o id da SCC a que esse vértice pertence.
- ➢ No último pop( ) (valor retirado da pilha igual ao valor da raíz da SCC), guardamos nesse vértice o valor do menor identificador da SCC.

3. Percorreu-se todos os vértices do grafo e para cada vértice, os seus adjacentes. Caso ambos os vértices tivessem o id da componente fortemente ligada diferente, incrementava-se o contador de ligações entre SCC e encontrava-se uma ligação entre duas SCC que era adicionada a um vetor de ligações.

4. Usou-se a função *void qsort(void *base, size_t nitems, size_t size, int (*comp)(const void *, const void*))* da biblioteca do C, para organizar o nosso vetor de ligações por ordem crescente.

5. Percorreu-se o vetor de ligações já organizado crescentemente e assinalou-se as ligações que estivessem repetidas (decrementando também o contador de ligações entre SCC) de forma a que o seu destino ficasse a NIL.

6. Finalmente, ao percorrer o vetor de ligações, se o valor de destino estivesse a NIL, este não seria impresso, e desta forma conseguimos imprimir todas as ligações sem repetições.

Análise Teórica

Em tempo de execução estima-se que que a complexidade temporal do algoritmo seja O(V+E) (sendo V o número de vértices e E o número de arestas) nas linhas 215 a 217 da função Tarjan_Visit(Graph graph, Vertice u) onde as validações e inserções dessa função são feitas em tempo constante.

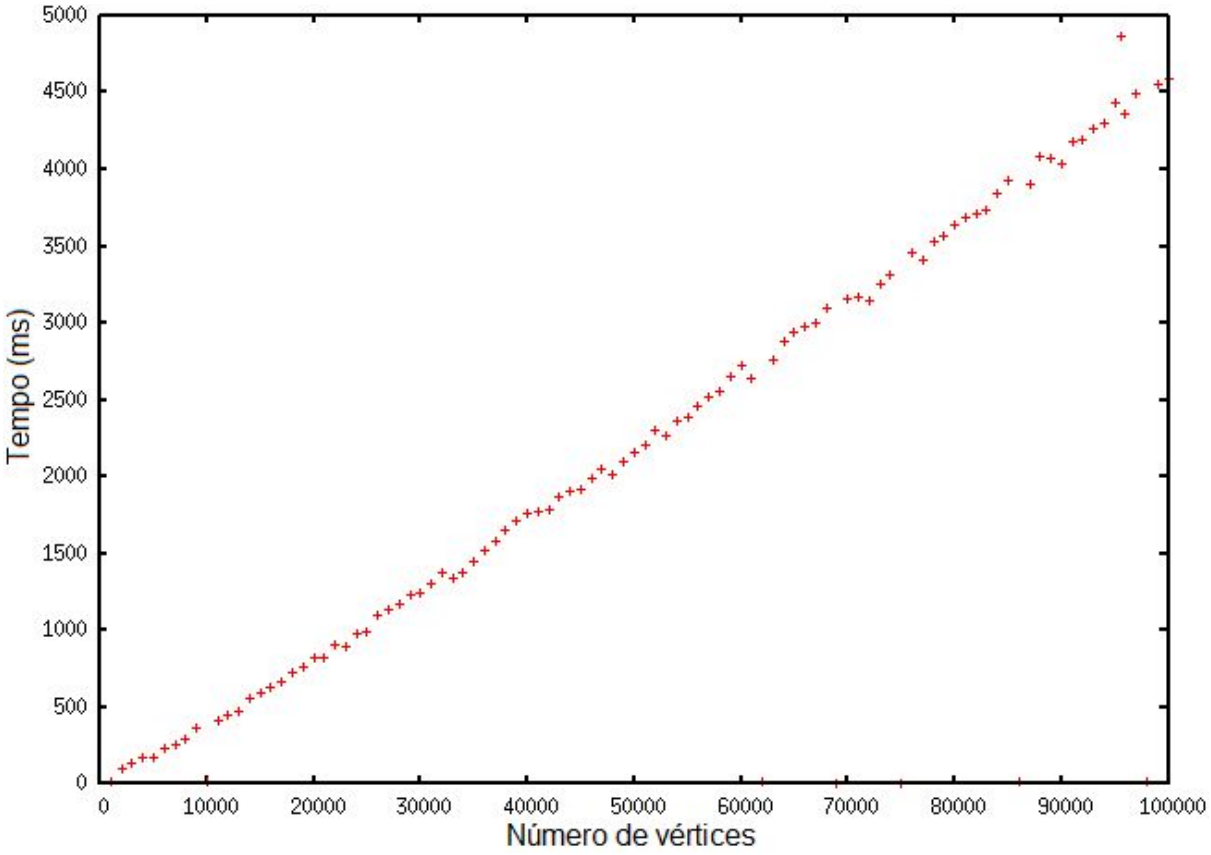
Quanto à complexidade de alocação de memória será O(V) para todos os vértices e ligações guardadas.

Relativamente à ordenação das ligações entre componentes fortemente ligadas é utilizada a função *void qsort(void *base, size_t nitems, size_t size, int (*comp)(const void *, const void*))* da biblioteca do C, que no pior caso (vetor de ligações entre SCC já estar ordenado) a sua complexidade será $O(L^2)$, no entanto na maioria das vezes será O(Llog(L)), sendo L o número de ligações existentes, incluindo repetições.

Por essa razão decidimos usar o algoritmo quick sort pois, existe uma probabilidade muito maior do vetor de ligações não estar previamente ordenado.

Análise Experimental

Gráfico da complexidade em função do tempo*



* - Os valores deste gráfico foram obtidos através de um gerador (fornecido pelo professor) de testes automático, com esses valores gerámos o respetivo gráfico com auxílio do programa "*gnuplot*".

Referências

- Diapositivos das aulas teóricas de ASA sobre algoritmos elementares em grafos (Algoritmo de Tarjan) e algoritmos de ordenação (QuickSort):
  https://fenix.tecnico.ulisboa.pt/downloadFile/1407993358870185/ch22_2.pdf
  https://fenix.tecnico.ulisboa.pt/downloadFile/1126518382187429/review7-10.pdf

- Diapositivos das aulas teóricas de IAED sobre pilhas, listas ligadas e grafos:
  https://fenix.tecnico.ulisboa.pt/downloadFile/845043405451077/iaed2016_17-2s-aula16-Estruturas_auto_referenciadas_e_listas.pdf
  https://fenix.tecnico.ulisboa.pt/downloadFile/282093452038711/iaed2016_17-2s-aula17-listasInteiros_listasStrings.pdf
  https://fenix.tecnico.ulisboa.pt/downloadFile/1970943312290820/iaed2016_17-2s-aula21-GrafosIeII.pdf

- https://www.tutorialspoint.com/c_standard_library/c_function_qsort.htm

- Gerador do gráfico: https://github.com/mikibakaiki/ist_asa_graphP1