Análise e Síntese de Algoritmos

Relatório do 1º projeto - 2017-18, 2º semestre

Grupo 08 (Taguspark) - João Freitas (87671), Pedro Soares (87693)

Introdução

O problema deste projeto consiste em conseguir identificar sub-redes regionais, numa vasta cadeia de supermercados com diversas rotas de distribuição de produtos. Estas sub-redes estão feitas de forma a que numa região seja possível qualquer ponto de distribuição enviar produtos para qualquer outro ponto da rede regional. Se um ponto *u* da rede de distribuição tem uma rota para um ponto *v* e do ponto *v* também existe uma rota para o ponto *u*, então ambos os pontos fazem parte da mesma sub-rede regional.

Representando esta situação com um grafo conexo dirigido, em que um ponto de distribuição é representado por um vértice e as rotas de distribuição por arcos, o problema resume-se a encontrar as componentes fortemente ligadas do grafo (sub-redes regionais) e as ligações entre as mesmas.

Descrição da solução

A solução assume três fases gerais:

- Encontrar as **componentes fortemente ligadas** do grafo (SCC's - Strongly connected component) e atribuir a cada **vértice** um ***id* da sua SCC** correspondente e **outros dados relevantes** (valor mínimo de vértice numa SCC) para a fase seguinte;
- Encontrar as **ligações entre as SCC's**;
- **Ordenar** os valores de origem e os valores de destino entre as ligações a apresentar no output.

No início do programa, após a inicialização do grafo e da stack (***initGraph*** *e* ***initStack***), recebe-se, na função ***main***, um dado input, que fornece o número de vértices para o grafo (pontos de distribuição da rede), o número de arcos (número de ligações na rede de distribuição; valor guardado na variável *numheads* na *estrutura Graph*) e as ligações entre os vértices, preenchendo-se o grafo à medida que se recebe os valores dos vértices (com ***insertEdge***).

De seguida, aplica-se uma versão modificada do algoritmo de Tarjan (***SCC_Tarjan*** e ***Tarjan_Visit***), que atribui a cada vértice o id da sua SCC (*scc_id*) e, na raíz, guarda o valor do vértice com o menor *storeNumber* dentro da SCC em questão (*min*, na *estrutura store*). Em cada vértice também é armazenado o *storeNumber* da raíz (*root*, na *estrutura store*), para mais tarde se poder fazer a verificação se é ou não raíz e para se poder aceder ao valor mínimo da sua SCC (através do acesso a esse valor pela raíz). Cada vez que se deteta uma raíz, ao longo da remoção de vértices na stack, é incrementada a variável global *nScc*, que guarda o número total de SCC's, e é atribuído um *id_Atual* de SCC a cada vértice.

Após obtermos estas informações essenciais relativas às SCC's, avançamos para duas funções, ***IdentifyConnections*** e ***MakeConnectionsArray***. A função ***IdentifyConnections*** percorre os vértices do grafo e os seus arcos adjacentes, de modo a identificar e a somar os arcos essenciais de ligação entre vértices em diferentes SCC's (*flag* marcada a 1 em cada arco especial; número total de arcos especiais: variável global *connections*). A cada arco adjacente, verifica se o vértice de destino é de outra SCC e se ainda não foi anteriormente registada uma ligação da sua SCC para a SCC do vértice de destino (visto que, entre duas SCC's, só se deseja uma ligação, não podendo ocorrer repetição). Caso se cumpram ambas as condições, regista-se esta nova ligação em *connectionsSCC* (vetor de inteiros que indicam as SCC's; as SCC's de destino são marcadas a 1 neste vetor) na estrutura do vértice de raíz da SCC de origem e incrementamos o número de ligações para output (*connections*).

Na função ***MakeConnectionsArray***, já temos todos os dados para a preparação dum vetor que armazena as ligações relevantes para mostrar no output (vetor de *estruturas connection*, compostas por dois inteiros para os valores de origem e destino, *source* e *destiny*) . Percorremos novamente o grafo e os arcos adjacentes de cada vértice e adicionamos ao vetor as ligações relevantes (correspondentes aos arcos com *flag* = 1) com os valores mínimos nas SCC's de origem e destino.

Em seguida, passamos à ordenação das ligações para output, que envolve a ordenação dos valores dos vértices de SCC de origem e a ordenação dos valores dos vértices de SCC de destino. De modo a obter a "sequência de ligações ordenada de forma não decrescente primeiro pelo identificador de origem da sub-rede e depois pelo identificador da sub-rede destino", foi aplicado o algoritmo de ordenação MergeSort, primeiro para ordenar as ligações de forma decrescente pelo destino e em seguida de forma crescente pela origem.

Por fim, é impresso no terminal: o número de SCC's (variável global *nScc*), o número de ligações (variável global *connections*) e as ligações já ordenadas, utilizando um ciclo for que percorre o vetor de ligações (*connectionsArray*) e imprime cada elemento desse vetor (ligação).

Análise teórica

Foi escolhida como estrutura de dados para a implementação deste problema um grafo representado como lista de adjacências. Sendo V o número de vértices e E o número de arcos num grafo, a complexidade associada à inicialização de um grafo sem ligações é de **O (V)** e à posterior inserção de todos os arcos é de **O (E)**. Deste modo, a complexidade total da construção do grafo é de **O (V+E)**.

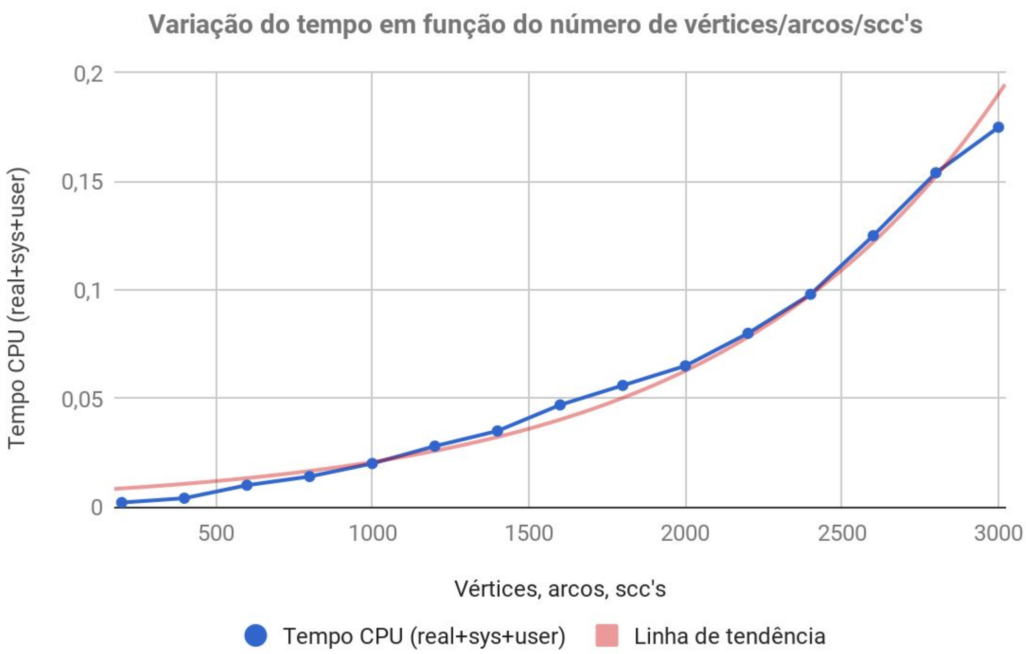
O programa apresentado aplica uma versão modificada do algoritmo de Tarjan, que procura as componentes fortemente ligadas do grafo e envolve mais algumas operações de tempo constante (identificar a SCC correspondente a cada vértice, por um valor de identificação), tendo, assim, uma complexidade de **O(V+E)** para listas de adjacências.

De seguida, são aplicadas duas funções (*IdentifyConnections* e *MakeConnectionsArray*) que percorrem os vértices e os seus correspondentes arcos adjacentes, tendo ambos a complexidade de **O (VE)**.

Por fim, para se ordenar as ligações apresentadas em output, são usadas duas modificações de MergeSort, *mergesortSource* (para os valores de origem, por ordem crescente) e *mergesortDestiny* (para os valores de destino, por ordem decrescente), que são aplicadas ao vetor *connectionsArray* A complexidade de ambas é por isso **O(Vlog(V))** (complexidade do algoritmo Mergesort).

Deste modo, a complexidade final do programa é, aproximadamente, **O(VE).**

Análise experimental



O gráfico acima revela a variação do tempo gasto na execução do programa em função do número (igual) de vértices, arcos e scc's. Um exemplo de dados de input no gerador fornecido pelo corpo docente é: 600 vértices (número de pontos na rede de distribuição), 600 arcos (número de rotas entre pontos na rede de distribuição) e 600 scc's (número de sub-redes regionais). Como esperado, de acordo com a análise teórica, a complexidade total do programa é, aproximadamente, O(VE). O gráfico apresentado acima representa, para este caso particular, uma função do tipo quadrático, pelo que, quando consideramos V = E, temos complexidade O(V^2).

Referências bibliográficas

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