TÉCNICO LISBOA

Análise e Síntese de Algoritmos

2013/2014

Projeto

Parte 1

Prefácio

Introdução: o objetivo fundamental deste Relatório consiste em explicar e expor a solução teórica da aplicação do Algoritmo de Tarjan implementado para resolver o problema da partilha de informação entre conjuntos de pessoas num dado Grafo.

Problema: resumidamente, o nosso problema consiste na partilha de informação entre pessoas, que por sua vez, em termos genéricos, indica que certas pessoas partilham informações relevantes entre outras, formando consecutivamente uma "network" de partilha de informação. Essa "network" por outro lado pode ser modelada e transformada num Grafo Dirigido.

Tarefas: traduzindo o Output dado pelo enunciado para Teoria dos Grafos;

- Percorrer o Grafo e calcular o número de Componentes Fortemente Ligadas;

  `scc_calc`

- Calcular o número de vértices da maior Componente Fortemente Ligada;

  `scc_size`

- Calcular o número total de Componentes Fortemente Ligadas que estejam isoladas umas das outras;

  `scc_max`

Após o problema identificado, planificado e estruturado, torna-se trivial implementar a sua solução aplicando Algoritmos de procura em Grafos tais como o Algoritmo de Tarjan.

Solução: a solução para o nosso problema passou por implementar o Algoritmo de Tarjan, descrito nas aulas teóricas e descrito no Livro. No entanto foi pesquisada também na Internet alguma informação sobre, nomeadamente da Wikipedia o mesmo, de modo a compreender melhor o Algoritmo.

FRANCISCO MARIA CALISTO | 70916 GRUPO: 172

CARLOS COSTA | 69854

Algoritmo de Tarjan – Descrição

Resumidamente o que o Algoritmo faz é procurar as Componentes Fortemente Ligadas (SSC's) de um dado Grafo. Embora o proceda cronologicamente, pode ser visto como uma melhor versão do Algoritmo Kosaraju daí não o termos implementado e sim termos implementado mas sim o Algoritmo de Tarjan, sendo este último comparado também ao Algoritmo Path-Based Strong Component, juntando assim duas características desejáveis ao sucesso deste projeto.

Este algoritmo toma como input um Grafo Direto, e procede à partição dos vértices do grafo em Componentes Fortemente Ligadas do Grafo. Qualquer vértice que não esteja num ciclo direto forma uma SCC, ou seja, se por exemplo o Grafo for acíclico.

Uma ideia simples do algoritmo consiste numa busca em profundidade que começa a partir de um nó de inicio arbitrário, cometendo pesquisas em profundidade subsequentes que são realizadas em todos os nós que ainda não foram encontrados. Estas, em conjunto e resultante da pesquisa, denominam-se como árvore de dispersão de um Grafo, a SCC será devolvida através de uma certa subdiretoria de árvore da floresta de árvores. Cada nó da Componente deve servir de raiz, sendo que, se tal acontecer e se este for o primeiro dos nós percorridos, este será o primeiro da pesquisa.

Os nós serão postos numa pilha na ordem que são visitados, após a primeira busca recursiva explorar o nó n e os seus descendentes, aqueles que não estavam necessariamente fora da pilha antes de serem retornados pela sua chamada. A parte crucial desta secção baseia-se no facto de haver uma propriedade de invariância de um dado nó que permanece na pilha após a exploração se e só se haja um caminho para algum nó da pilha anteriormente percorrido.

No fim da chamada dessa mesma exploração n e os seus descendentes, sabemos então que qualquer que seja n e o seu próprio caminho para algum dos seus nós que estejam anteriormente na pilha, retorna a chamada, preservando assim n na pilha, caso contrário, então n terá que ser a raiz da sua própria SCC, em que consiste no facto de n estar agrupado com qualquer que seja o nó anterior na pilha, sendo que cada nó terá que ter caminhos para trás de n mas não terá caminhos mais recentes do que o próprio nó, sendo esta Componente toda ela libertada devolvendo assim os valores da pilha, novamente preservando assim invariância.

Por fim cada nó n é assinalado unicamente por `n.index`, estando cada numero do nó consecutivamente em ordem de descoberta. Também é mantido um valor `n.lowlink` que basicamente representa o índice mais pequeno descoberto pelo nó e atingível por n, incluindo o próprio n.

Sempre que o valor de `n.lowlink` for menor do que o valor de `n.index`, n tem que ser deitado para fora da pilha. Caso o nosso `n.lowlink` for igual a `n.index` é removida da raiz a SCC de n. O valor de `n.lowlink` é então processado durante a primeira passagem em profundidade por n, até que encontre o nó atingível por n.

Algoritmo de Tarjan – Complexidade

Em termos de Complexidade o Algoritmo de Tarjan tem um performance no pior caso de possível:

O(|V| + |E|)

Desta forma o nosso Algoritmo corre em tempo linear e executa uma excelente performance no que toca à resolução do problema para a primeira parte do projeto.

Implementação

Para a implementação deste projeto utilizamos a Linguagem de Programação C++ pois tem um excelente rácio entre simplicidade de Programação e Eficiência de execução não retirando grandes recursos.

No que toca à nossa Implementação a ideia a transmitir é que sabendo o Número Total de Componentes Fortemente Ligadas:

`scc_calc`

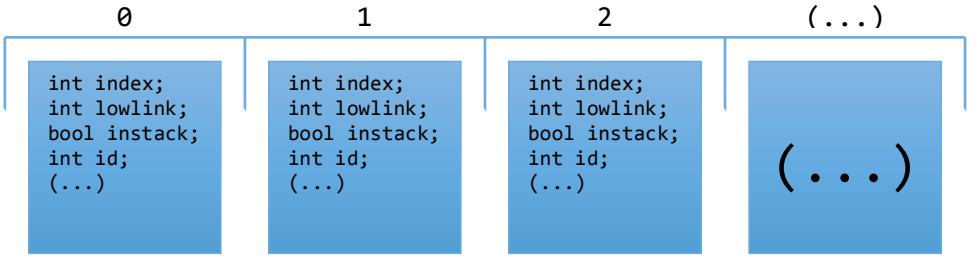
Consecutivamente é simples de implementar uma forma de sabermos onde as Componentes estão armazenadas, ou seja, os vértices que compõem cada uma das Componentes Fortemente Ligadas.

Ao longo da computação do Algoritmo de Tarjan são utilizadas três Estrutura de Dados auxiliares para contabilizar e nos ajudar no resultado final do Output:

1. `scc` – Os vértices dos Componentes Fortemente Ligadas (SCC's) são então guardas num vector;
2. `graph` – Os nós do Grafo serão então guardados num vector;
3. `stack_aux` – é uma Pilha com os ponteiros para o nó do Grafo;

Tendo nós na Estrutura de um nó todas as componentes necessárias à implementação do Algoritmo de Tarjan, foi fácil e intuitivo arranjar uma implementação criativa de guardar a informação numa Estrutura de Dados, no nosso caso um vector de vectores de estruturas, tal que fosse possível concretizar o nosso problema com sucesso.

A ideia essencial da nossa Estrutura de Dados está em seguida exemplificada:



Como se pode observar esta é uma implementação simples e bastante intuitiva de programar, assumindo que não havia ambiguidades após a solução criativa.

Começando por programar o Calculo das Componentes Fortemente Ligadas `scc_calc` definindo o índice de profundidade do nó de n, após isto é implementado um ciclo considerando os sucessores do nó de n, onde, no caso de o sucessor ainda não foi visitado, este é percorrido recursivamente. Numa situação em que o sucessor já se encontra na Pilha, então, também estará na atual Componente Fortemente Ligada, não sendo necessário recursão nesta situação.

No caso em que n é uma raiz do Grafo, é retirada da Pilha o valor do mesmo, gerando assim uma nova Componente Fortemente Ligada.

A função `scc_create` cria a nova SCC através de um iterador que itera do inicio ao fim do Grafo.

Para sabermos a SCC com mais elementos implementamos a função `scc_max` que funciona através de uma função `unsigned`, basicamente tem como objectivo comprar os dois valores em causa.

Utilizando a variável `scc_act` como atual Componente Fortemente Ligada. Enquanto a nossa SCC não for uma e uma só Componente Fortemente Ligada, a SCC passa a ser o sucessor em zero, e se o `lowlink` do sucessor for diferente da SCC actual, então finaliza o ciclo. Se o sucessor já estiver quebrado, o ciclo acaba imediatamente. Por outro lado se o sucessor não está quebrado, então passa à seguinte referenciação seguinte.

Conclusão

Em jeito de conclusão, na nossa opinião esta implementação foi a mais correta, tendo em conta a intensa pesquisa feita a nível Teórico por nós numa tentativa de compreender primeiramente qual o melhor Algoritmo e quais as melhores Estruturas e Componentes a serem implementados para o sucesso do mesmo. Tivemos alguma dificuldade em compreender a melhor forma de guardar a informação e mais tarde retorna-la no caso de Vectores de Vectores das Estruturas.