ASA - Relatório 1

Marisa Roque - 76653

21 de Março de 2014

Introdução

Foi apresentado um problema que consistia em analisar a forma como se dá a partilha de informação entre pessoas e em como algumas pessoas acabam por formar grupos, supondo que cada pessoa tem um conjunto de pessoas com quem partilha o que recebe. O objectivo deste trabalho é classificar as pessoas em grupos de partilha, de tal forma que quando uma das pessoas do grupo recebe algo, todas as outras desse mesmo grupo também recebem.

Nesse sentido, o programa desenvolvido está preparado para receber informação sobre o modo como se dá a partilha entre pessoas. Esta informação deve conter uma linha inicial com o número de pessoas N e com o respectivo número de partilhas P realizadas por essas mesmas pessoas. Nas restantes linhas de entrada deve vir a informação descriminada entre que pessoas se deu a partilha de informação. As pessoas devem ser identificadas como inteiros entre 1 e N, como tal, por exemplo, o facto de vir 1 2 na segunda linha de entrada número 1 quer dizer que a pessoa 1 partilha o que recebe com a pessoa 2.

O programa tem como principal função devolver informação sobre os grupos de partilha, nomeadamente e por ordem de saída, o número de grupos máximos de pessoas que partilham informação, o tamanho do maior grupo máximo de pessoas que partilham informação e, por fim, uma linha com o número de grupos máximos de pessoas que partilham informação apenas dentro do grupo.

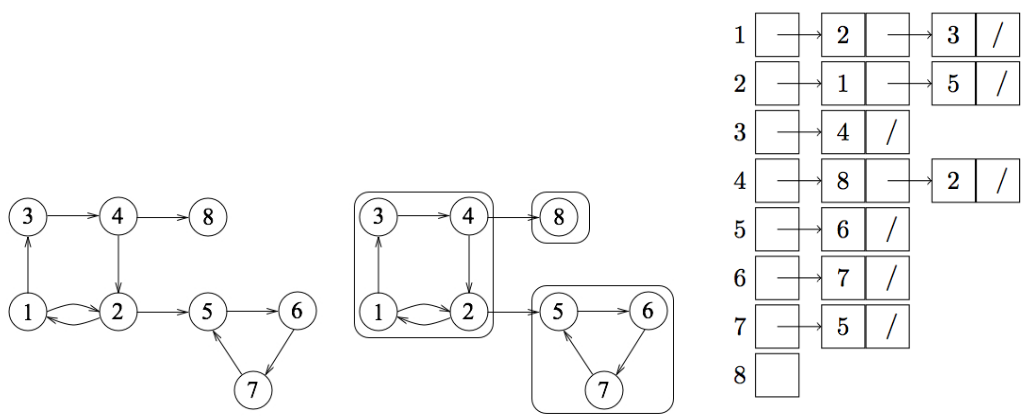
Descrição da solução

Para chegar a uma solução eficiente passou-se por várias etapas. A primeira foi analisar e perceber detalhadamente os dois exemplos apresentados no enunciado. Para isso foi necessário recorrer à teoria dos grafos, onde são estudadas as estruturas chamadas de grafos, $G(V, E)$, onde V é um conjunto não vazio de objectos denominados vértices e E é um conjunto de pares não ordenados de V, chamado arestas. Daqui rapidamente se partiu para a ideia geral de que se tratavam de grafos dirigidos, dada a natureza do problema, na sua maioria grafos esparsos - aqueles para os quais $|E|$ é muito menor que

$|V|^2$ -, e que o que se pretendia era descobrir as componentes fortemente ligadas (SCC) dos mesmos.

Segundo Thomas H. Cormen et al. [1], um componente fortemente ligado de uma grafo orientado $G = (V, E)$ é um conjunto máximo de vértices $C \subseteq V$, tal que, para todo o par de vértices v e w em C, temos ao mesmo tempo $v \rightsquigarrow w$ e $w \rightsquigarrow v$, isto é, os vértices v e w são acessíveis um a partir do outro. Exemplos das componentes fortemente ligadas do exemplo 1 podem ser observados na figura 1 (c).

Como visto em cima e pelo facto de os grafos serem esparsos, optou-se por fazer a representação dos grafos através de listas de adjacências ao invés da representação por matriz de adjacências (figura 1 (b)). Por conseguinte, para um grafo $G = (V, E)$, fez-se um arranjo de $|V|$ listas, uma para cada vértice em V. Para cada $v \in V$, a lista de $Adj[u]$ contém ponteiros para todos os vértices w tais que existe uma aresta $(v, w) \in E$, ou seja, $Adj[v]$ consiste em todos os vértices adjacentes a v em G.



(a) (b) (c)

Figura 1: Exemplo 1 do enunciado. (a) Representação do grafo orientado G com oito vértices e 10 arcos. (b) Representação dos componentes fortemente ligados de G. (c) Representação de G como uma lista de adjacências.

Assim, ao nível do código, foram criadas três estruturas essenciais. Uma estrutura chamada *graph* que contém o vector de listas de adjacência, o número de vértices N (ou pessoas, se virmos da perspectiva do problema), o número de arcos do grafo P ou de partilhas entre as pessoas e, por fim, um vector com o número da componente fortemente ligada. O vector de listas de adjacências de um grafo tem uma lista ligada para cada vértice do grafo. A lista do vértice v contém todos os adjacentes de v. A lista de adjacência de um vértice v é composta por nós do tipo *node*. Cada nó da lista corresponde a um arco e contém um adjacente w de v e o endereço do nó seguinte da lista. Um *link* é um ponteiro para um node. Uma segunda estrutura essencial no programa, *scc_result*, com o objectivo de reunir a informação resultante do algoritmo de Tarjan, contém o número de SCC, um vector de vectores com os SCC e outro vector com o tamanho de cada SCC.

Análise teórica

Na modelação deste problema usou-se um algoritmo recursivo de tempo linear, ou seja, de tempo $O(V + E)$, para encontrar componentes fortemente ligados de um grafo dirigido $G = (V, E)$, chamado algoritmo de Tarjan. Como se pode ver pelo pseudo-código em baixo, este é um algoritmo que faz uso do algoritmo de busca em profundidade de grafos (DFS). Resumindo, a inicialização dá-se em tempo O(V), as chamadas à função DFSscc igualmente em O(V), e, por fim as listas de adjacências são analisadas apenas uma vez, logo, O(E).

Algorithm 1 Algoritmo de Tarjan

```
procedure SCC_Tarjan(G)
    visited = 0
    L = 0
    for all vertex u ∈ V[G)] do
        d[v] = ∞
    end for
    for all vertex v ∈ V[G] do
        if d[v] = ∞ then
            DFSscc[v]
        end if
    end for
end procedure

procedure DFSscc(G,v)
    d[v] =low[v]=visited
    visited = visited + 1
    Push(L,v)
    for all v ∈ Adj[v] do
        if d[w] = ∞ then
            DFSscc(v)
        end if
        if low[w] < low[v] then
            low[v] = low[w]
        end if
    end for
    if d[v] = low[v] then
        repeat
            w = Pop(L)
        until v = w
    end if
end procedure
```

Podia-se também ter optado pelo algoritmo de Kosaraju, no entanto, este tinha à partida a desvantagem aparente de ter que realizar duas buscas em profundidade (DFS), o que o tornou logo menos atractivo em termos de eficiência, em comparação com o de Tarjan.

Resultados da avaliação experimental

Após completa a codificação, a solução foi submetida a diferentes testes. Numa primeira fase, foram feitos os testes disponibilizados pelos docentes, bem como os dois exemplos do enunciado. Nesta fase, a solução parecia ter um comportamento aceitável, pois as saídas estavam todas correctos. No entanto, estes testes não era suficientemente exaustivos e não testavam todas as hipóteses de entradas. Ao submeter o ficheiro no *Mooshak*, o algoritmo foi submetido a testes mais rigorosos. Das primeiras vezes que foi submetido ocorria um *Time Limit Exceeded* em três testes (10, 12, 15). Posteriormente, melhorou-se o nosso algoritmo mudando os acessos à lista de adjacências e optou-se por marcar os vértices para indicar a qual componente fortemente ligado eles pertencem, verificação essa que demora tempo constante.

No entanto, depois dos devidos ajustes, submeteu-se uma solução optimizada que passou com sucesso aos 16 testes do sistema.

Referências

[1] Thomas H. Cormen, *Introduction to Algorithms.* Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, Third Edition, September 2009.