Relatório - Projecto Análise e Síntese de Algoritmos Parte 1

Grupo AL233
Alexandre Anacleto, 65862
Leo Valente, 67030

Introdução do problema

O problema proposto consiste em desenvolver um algoritmo capaz de reconhecer grupos, sendo a seguinte descrição apresentada no enunciado

"Suponha que cada pessoa tem um conjunto de outras pessoas com quem partilha o que recebe. (...) **quando uma das pessoas do grupo recebe algo, todas as outras também irão receber.**"

O factor importante a reter aqui é a parte a negrito isto porque a descrição apresentada corresponde a um SCC (Strongly Connected Component). Existem várias soluções para fazer o pedido e as nossas escolhas encontram-se justificadas mais abaixo.

Descrição da solução

Recorremos a uma implementação do tipo lista simplesmente ligada para fazer a implementação do grafo em si. Tendo em conta que os nódulos vão ser numerados entre 1 e N decidimos implementar o grafo como um vector. Na nossa implementação um grafo corresponde a um vector de listas de tamanho N+1 sendo que o espaço 0 não é usado (por uma questão de simplificação de código). Cada espaço desse vector vai corresponder ao ID relativo ao nódulo (id 3 corresponderá ao nódulo n=3) e vai conter uma lista de ligações que serão lidas do input.

Para descobrir as linhas pedidas no enunciado do projecto optámos por implementar o algoritmo de **Kosaraju-Sharir**.

O algoritmo de Tarjan tem vantagem em relação ao algoritmo de Kosaraju simplesmente porque **não exige a construção da transposta** do grafo G. Numa tentativa de combater esta discrepância decidimos implementar dois grafos em simultâneo aquando da recepção dos inputs do ficheiro, o grafo G e o grafo G_t.

O ganho de tempo é discutivelmente favorável em relação ao espaço de memória ocupado neste caso, simplesmente porque esta implementação evita-nos a necessidade de percorrer o vector uma vez mais para fazer a inversão das direcções dos arcos.

É de notar também que como auxiliar ao algoritmo criámos um vector global de tamanho N+1, que servirá para a visita em profundidade saber os nódulos visitados ou fechados.

Portanto, para descobrir a primeira e segunda linha eis a sequência de passos (implementação do algoritmo Kosaraju-Sharir):

1.1. **DFS do grafo G**. Obtemos na pilha uma sequência de vértices.

1.2. **DFS do grafo G_t** pela ordem obtida no ponto 1. O resultado deste passo vai ser uma lista de vertices. Colocámos um '-1' entre cada visita em profundidade para marcar o fim de um SCC.

1.3. **Percurso da lista e contagem dos '-1'**. Por cada '-1' encontrado o contador de grupos é incrementado. Durante o percurso da lista cada avanço incrementa a variavel tamanho, que é reiniciada quando o '-1' é encontrado e é nesse caso comparada com o tamanho máximo. Caso a variável seja maior que o tamanho máximo até agora encontrado, este é alterado, garantindo assim que temos sempre o tamanho do SCC maior nesta variável.

Destes 3 passos obtemos a resposta às 2 primeiras linhas.

Para descobrir a terceira linha explorámos o facto de que, caso um SCC esteja isolado, a visita em profundidade relativa a qualquer um dos vértices do grupo terá exatamente o mesmo tamanho do grupo. Esta parte implica mais uma procura na lista do ponto 1.2. do algoritmo anterior. Este algoritmo é executado uma vez por grupo (contados no ponto 1.3.).

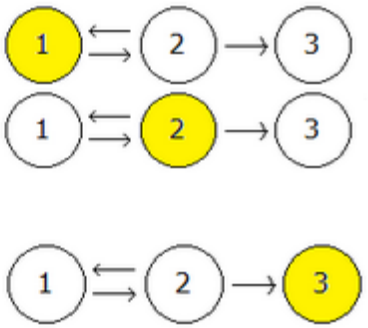
2.1. Procura na lista obtida em 1.2. por um '-1' que sinaliza o fim de um SCC (o numero de avanços é contado, obtendo assim o tamanho do SCC).

2.2. Faz uma procura em profundidade do ultimo vértice do SCC corrente.

2.3. Descobre o tamanho da lista obtida pela visita em profundidade e compara com o tamanho do SCC. Se forem iguais podemos concluir que o grupo é isolado e nesse caso o contador é incrementado.

Explica-se com esta ilustração o funcionamento do algoritmo.

Count	Loop	Size	Size2	Stack
	1	1		1 2 -1 3 -1
	1	2		1 2 -1 3 -1
0	1	2	3	
	2	1		1 2 -1 3 -1
1	2	1	1	



Legenda:
1. O algoritmo é executado uma vez por grupo contado anteriormente (Loop).
2. Usa a stack devolvida pelo algoritmo de kosaraju. (Stack)
3. Variavel size incrementada com o percorrer da stack

$\forall$ k $\in$ stack, if k != -1 then size++.

4. Guarda-se na variavel size2 o número de elementos devolvidos na dfs e compara-se com size. Se forem iguais, count é incrementado.

No final disto tudo é então imprimida a resposta tal como pedido.

Análise teórica e avaliação experimental dos resultados

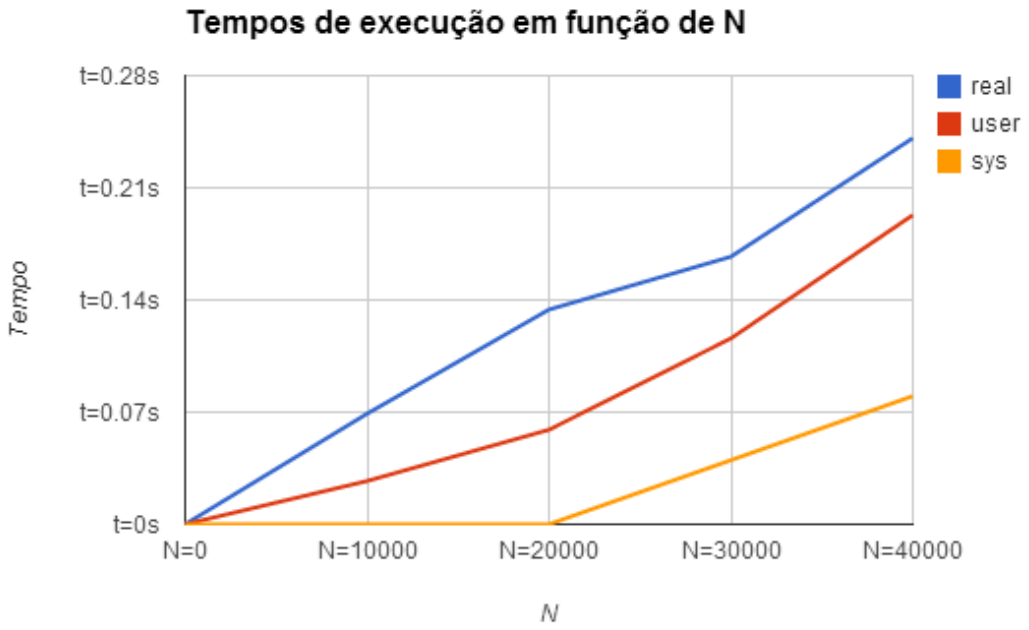
O algoritmo funciona principalmente com pesquisas dos vectores que crescem com o número de elementos N fornecidos. Cada vez que reiniciamos os estados percorremos N posições, cada vez que fazemos uma pesquisa em profundidade fazemos (no pior caso) N operações e cada vez que fazemos uma contagem temos obrigatoriamente que fazer N operações. Posto isto, da observação do código submetido temos as seguintes operações:

Inicialização	$\forall$ N, G[n] = NULL	, N = #Nódulos
Partilhas	$\forall$ P, Novo arcoem G	, P = #Arcos
Kosaraju	N + P	
Contagem grupos	N	
Contagem isolados	$\forall$ K { N_a + N_b + N_c }	, K = #SCC

N_a=Contagem N_b=DFS N_c=Reset estados

No seu conjunto e no seu pior caso possivel, o código tem a seguinte complexidade:

$$O(N + P + (N + P) + N + K(3N))$$
$$O(3N + 3KN + 2P)$$
$$O(3N(K + 1) + 2P)$$



Da análise do pior caso possivel e tempos de computação com valores variáveis de N podemos observar que o algoritmo tem uma ordem de **crescimento linear.** Os tempos de execução crescem de forma linearmente proporcional aos inputs.

Fontes utilizadas

Páginas wikipedia:

http://en.wikipedia.org/wiki/Strongly_connected_component
http://en.wikipedia.org/wiki/Kosaraju's_algorithm
http://en.wikipedia.org/wiki/Tarjan's_strongly_connected_components_algorithm
http://en.wikipedia.org/wiki/Depth-first_search
http://en.wikipedia.org/wiki/Transpose_graph

Outros links

http://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/strong-comps.html
https://cw.felk.cvut.cz/wiki/_media/courses/ae4m33pal/lectures/2012pal03.pdf
http://www.dsc.ufcg.edu.br/~abrantes/CursosAnteriores/TG051/scc.pdf
http://homepages.ius.edu/rwisman/C455/html/notes/Chapter22/DFS.htm