Relatório - Projecto Análise e Síntese de Algoritmos Parte 2

Grupo AL233
Alexandre Anacleto, 65862
Leo Valente, 67030

Introdução do problema

O problema proposto consiste em desenvolver um algoritmo capaz de contar o número minimo de caminhos a bloquear entre 2 pontos num grafo, de forma a isolá-los um do outro.

É importante notar que, num dado grafo, dois pontos podem estar ligados de várias formas diferentes (pode haver mais do que uma combinação de caminhos), mas isso, no entanto, não significa que para isolar um ponto de outro baste barrar todas as ligações de um desses pontos.

Posto isto, podemos concluir que a resposta não é trivial e é necessário analisar o grafo para chegar a um número.

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 0 e N decidimos implementar o grafo como um vector. Na nossa implementação um grafo corresponde a um vector de listas de tamanho N. Cada espaço desse vector vai corresponder ao ID relativo ao nódulo (id 3 corresponderá ao nódulo n=3 por exemplo) e vai conter uma lista de ligações que serão lidas do input. Tendo em conta que o grafo é bidirecionado, as ligações são criadas em pares. Isto é, quando se cria a ligação A-B também se cria a ligação B-A.

Para descobrir as linhas pedidas no enunciado do projecto optámos por implementar o algoritmo de **Ford-Felkurson** inicialmente mas viemos a postriori alterar a nossa implementação para o algoritmo de **Edmonds-Karp.**.

O algoritmo de Edmonds-karp é uma implementação discutivelmente melhor, isto porque, existem casos que o algoritmo **Ford-Fulkerson** não consegue resolver. O algoritmo falha em certos casos porque, sendo uma travessia em profundidade, existe a possibilidade que a travessia venha a consumir um caminho longo que poderá incluir um caminho alternativo, fazendo assim com que a contagem saia errada, menor para ser mais preciso.

O algoritmo **Edmonds-karp** faz uso de uma travessia em largura em vez de profundidade, pelo que o caminho escolhido será <u>sempre</u> o mais curto, fazendo assim com que o problema que ocorre na travessia em profundidade não ocorra.

Para tomar em conta os fluxos e capacidade dos caminhos associámos à prévia implementação de lista um inteiro, referente à capacidade do caminho.

É de notar também que como auxiliar ao algoritmo criámos:

- Um vector global de tamanho N, que servirá para a visita em largura saber os nódulos visitados ou fechados.
- Um vector global de tamanho N, que servirá para a visita em largura saber qual o nódulo a partir do qual fez a visita.

Portanto, para descobrir a resposta ao problema a unica coisa que temos de fazer é uma travessia em largura até encontrar o ID destino.

O algoritmo funciona porque tira partido do facto de que, se um grafo pesado tiver o mesmo peso em todos os seus arcos, o fluxo do grafo entre 2 pontos U e V será equivalente ao número de caminhos diferentes entre U e V.

Tendo isto em conta podemos concluir que, ao aplicar o algoritmo para todas as combinações de pontos criticos, o número mais baixo de entre os resultados obtidos será o número mínimo de caminhos a bloquear para isolar um único ponto.

Por isso, se considerarmos K o numero de pontos criticos num dado problema o que vai acontecer é o seguinte:

C^{k}_{2} Aplicações de BFS com armazenamento do menor resultado.

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

Análise teórica e avaliação experimental dos resultados

O algoritmo funciona principalmente com pesquisas dos vectores, pesquisas essas que crescem com os números de K, P e N.

Cada vez que reiniciamos os estados dos nódulos percorremos N posições, cada vez que fazemos uma pesquisa em largura fazemos (no pior caso) N operações, mas, principalmente, para cada aumento de K aumenta o numero de vezes que estas operações devem ser executadas.

Posto isto, da observação do código submetido temos as seguintes operações:

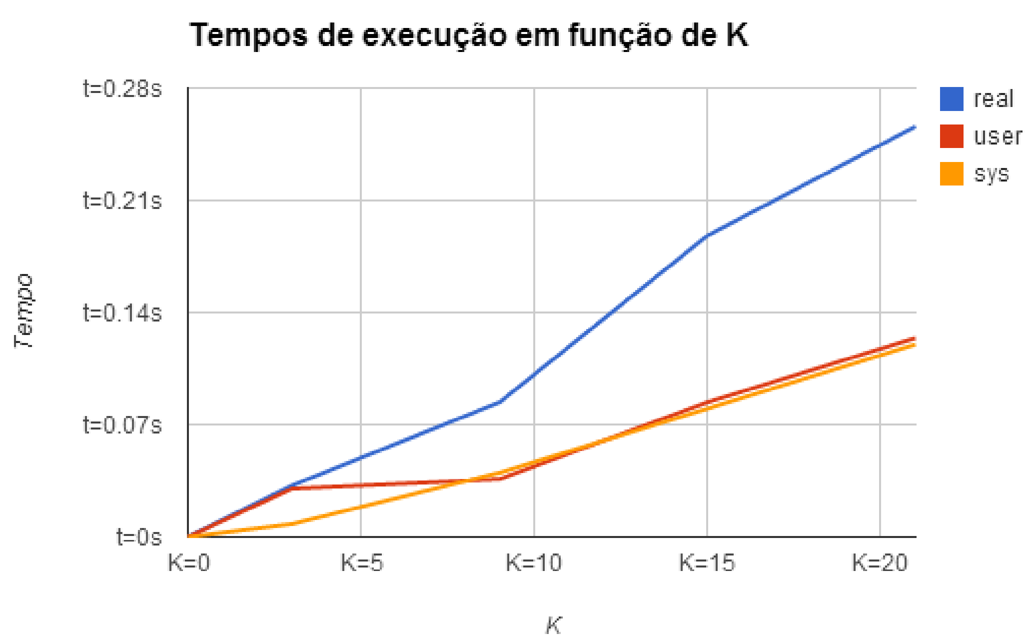
Inicialização	$\forall$ N, G[n] = NULL	, N = #Nódulos
Partilhas	$\forall$ P, 2x Novo arco em G	, P = #Arcos
Edmonds-Karp	NP^2	, f = max flow do grafo
Número de aplicações	C^k_2	, k = #Pontos criticos

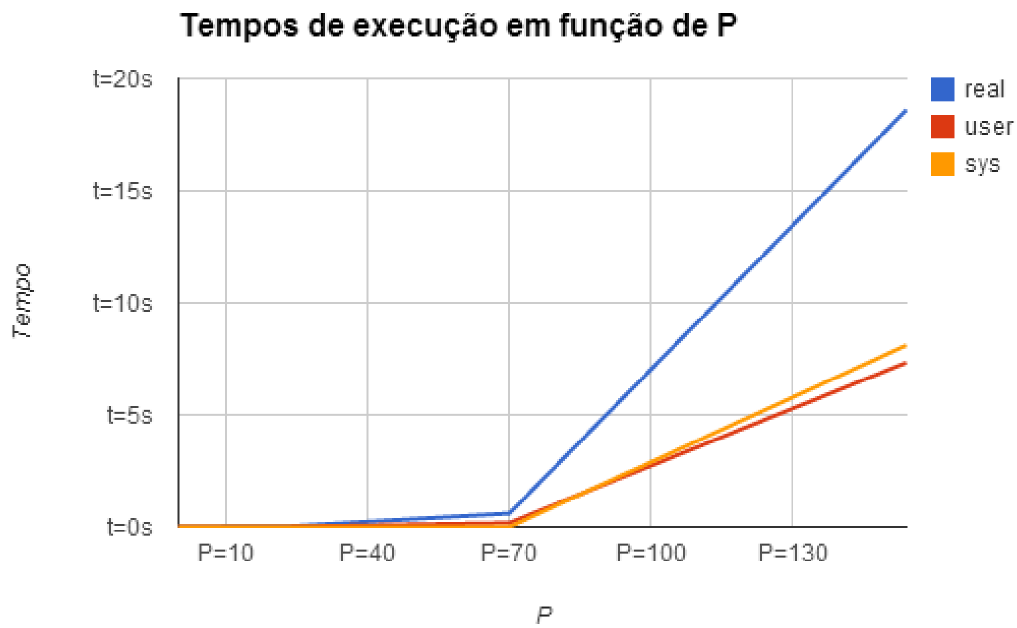
No seu conjunto e no seu pior caso possivel, o código tem a seguinte complexidade:

$$O(N + 2P + C^k_2NP^2) = O(N + P(2 + C^k_2NP))$$

Podemos concluir que o P tem um peso provavelmente mais importante nos tempos de execução do programa. Com isto em conta tirámos os tempos de execução para valores de P variáveis ao invés de valores de N varáveis.

Apresentam-se de seguida gráficos para valores de P variáveis e K variáveis.





Da análise do pior caso possivel e tempos de computação com valores variáveis de K e P podemos observar que o algoritmo tem uma ordem de **crescimento quadrática.** Os tempos de execução crescem de forma quadrática com os seu inputs.

Fontes utilizadas
Páginas wikipedia:
http://en.wikipedia.org/wiki/Ford–Fulkerson_algorithm
http://en.wikipedia.org/wiki/Edmonds–Karp_algorithm
http://en.wikipedia.org/wiki/Breadth-first_search
http://en.wikipedia.org/wiki/Depth-first_search
http://en.wikipedia.org/wiki/Maximum_flow_problem

Outros links
video: https://www.youtube.com/watch?v=_yOSku77w0o
http://w3.ualg.pt/~hshah/algoritmos/aula6/aula6/ford-fulkerson.html
http://www.cs.fsu.edu/~asriniva/courses/alg06/Lec28.pdf
http://www.cse.unt.edu/~tarau/teaching/AnAlgo/Edmonds%E2%80%93Karp%20algorithm.pdf