Análise e Síntese de Algoritmos
2018/2019

1º Projeto – Relatório

André Rodrigues - nº 70119 – Grupo 56

Introdução:

Neste projeto é dado um conjunto de routers que constituem uma rede sobre a qual, em princípio se consegue definir um caminho que permite a comunicação entre qualquer par de routers. Caso isso não aconteça, então a rede está dividida em subconjuntos de routers ligados entre si.

Através de um input com o número de routers da rede, o número de ligações entre routers na rede e a ligação entre u e v (sendo u e v dois inteiros identificando os routers em questão), o programa tem de indicar o número de sub-redes, bem como os identificadores das sub-redes existentes (router com maior identificador que pertence à sub-rede) por ordem crescente.
Para além disso, é também necessário verificar se a rede é robusta. Em particular, identificar os routers da rede que, se forem atacados (não permitindo a receção ou envio de informação) ou desligados, resultariam no aumento do número de sub-redes, indicando também o número de routers da maior sub-rede resultante.

Descrição da Solução:

Para a implementação da solução foi usada a linguagem de programação C++ pela sua simplicidade em implementar a lista de adjacências.

Neste algoritmo um vértice (router) é um ponto de articulação de um grafo se, após a remoção desse vértice o número de componentes ligados entre si aumenta. A ideia é que dado um vértice **u** que não seja a raiz, ou seja, um vértice que tem vértice pai, é um ponto de articulação se, e apenas se, tem um filho **v** onde o menor tempo de descoberta **low[v]** é maior ou igual que o seu tempo de descoberta **discovery[u]**.
Para o caso em que o vértice **u** é uma raiz (não tem vértice pai), o vértice é um ponto de articulação se, e apenas se, tem pelo menos dois vértices filhos.

A solução consiste, portanto, em usar uma depth-first search sobre o conjunto de routers que constituem uma rede**.**
A depth-first search vai percorrer o grafo começando no vértice de menor valor, analisando os vértices que estão ligados entre si. Se ainda houver vértices por visitar, a dfs vai começar no vértice de menor valor até todos os vértices estarem visitados.

É durante a execução da depth-first search que é feita a análise do problema, guardando as seguintes informações:

- Tempo de descoberta de um vértice, designado por **discovery**;
- Para cada vértice, o menor valor de descoberta pertencente à sua descendência, designado por **low**;
- Número de filhos do vértice, designado por **childCount**.
- Vértice predecessor, ou pai, de um dado vértice, designado por **parent**.
- Indicação de que o vértice foi visitado, indicado pelo valor *true* no booleano **visited** (ou *false*, caso não tenha sido visitado).
- Indicação de que o vértice é um ponto de articulação, indicado pelo valor *true* no booleano **isArticulation.**
- Maior identificador (número do vértice) de um dado subconjunto de routers ligados entre si, guardado numa variável auxiliar designada **max_value.**
- Indicação do número de pontos de articulação existentes numa dada rede, designado por **ap_points**.
- Número de vértices num dado subconjunto de routers ligados entre si, designado por **vertices_in_subgraph**.

Para além disso, de forma a guardar o valor do maior vértice que identifica a sub-rede após a depth-first search visitar um conjunto de vértices conectados entre si, o vértice de maior valor é adicionado ao vetor com o nome **subgraph_max_values**, contendo este vetor no final todos os identificadores das sub-redes existentes.

Após todos os vértices estarem visitados, o número de vértices que forem identificados como a raiz (que não tenham nenhum vértice pai) indica quantas sub-redes existem.

No enunciado é também pedido que sejam impressos de forma crescente os vértices que identificam as sub-redes (vértices de maior valor).
Como nesta solução os valores vão sendo adicionados ao vetor **subgraph_max_value** à medida que vão sendo descobertos, os vértices não vêm ordenados.

Para tal é usado o algoritmo de **Counting Sort** para fazer a ordenação dos mesmos, dado que tem complexidade O(V+E) tal como a dfs e assim a complexidade de toda a solução continua O(V+E).

Como os pontos de articulação vão sendo descobertos dentro da depth-first search, para obter o número de routers da rede que, se forem atacados ou desligados, resultariam no aumento do número de sub-redes, basta imprimir o valor da variável **ap_points**.

Para descobrir o número de routers da maior sub-rede resultante da remoção de todos os routers que quebram uma sub-rede (pontos de articulação), é necessário correr a dfs novamente, reinicializando os vetores **visited**, **discovery** e **low** , com a diferença que, desta vez os vértices que têm o booleano **isArticulation** com o valor *true* não serem colocados com o **visited** a *false* de forma à dfs não passar por eles.

É então aqui que é dado uso à variável auxiliar, **vertices_in_subgraph**, dentro da depth-first search de forma a obter o número de routers da maior sub-rede.

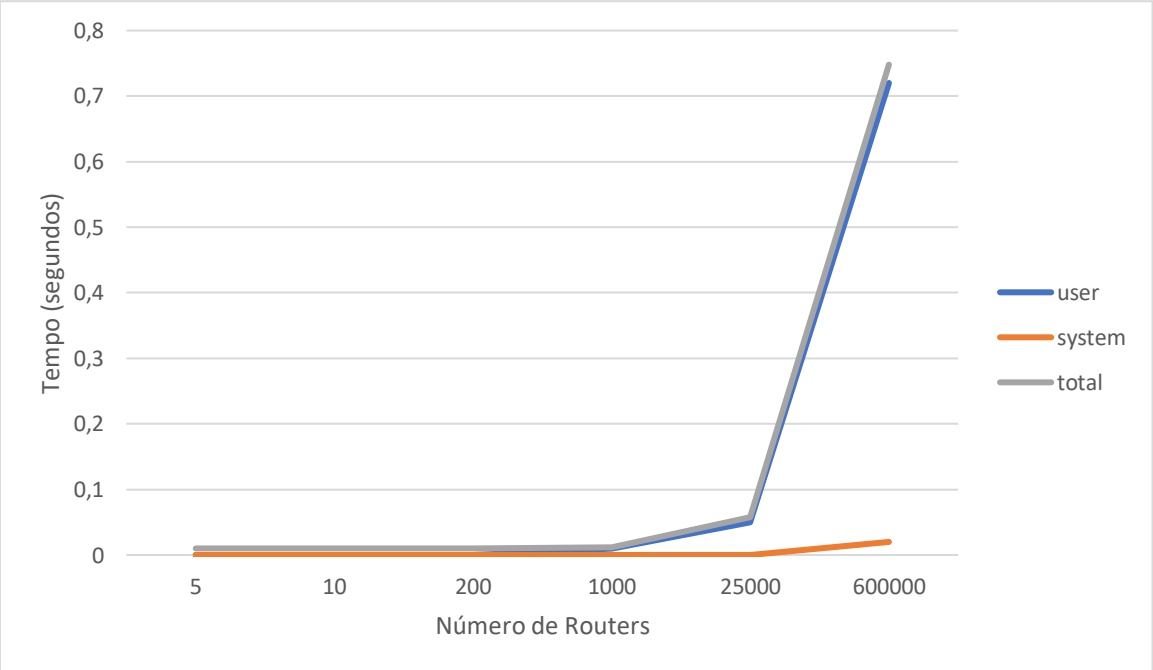
Análise Teórica:

- Criar Grafo: O(E)
- Depth-first Search: O(V+E)
- Devolver o número de sub-redes: O(V)
- Counting Sort: O(V+E)
- Devolver os identificadores das sub-redes existentes: O(P), *onde P é o tamanho do vetor que vai guardando os vértices.*
- Número de routers que quebram uma sub-rede: O(1).
- Número de routers da maior sub-rede resultante da remoção de todos os routers que quebram uma sub-rede: O(1).

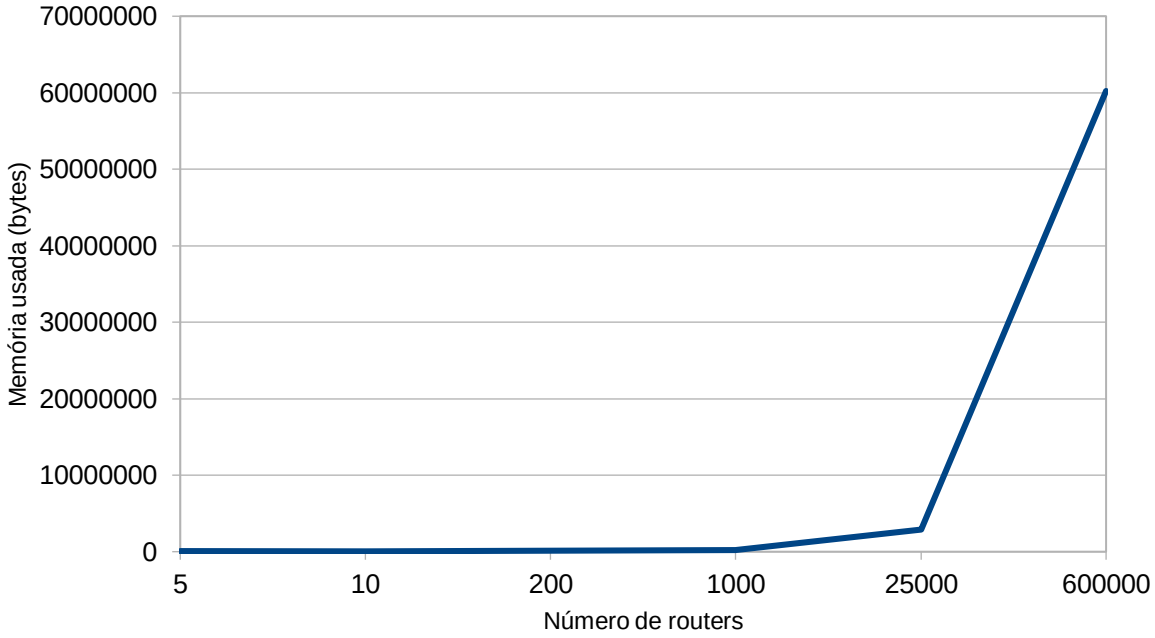
- Complexidade Total: O(V+E)

Avaliação Experimental dos Resultados

Para a realização da avaliação experimental dos resultados foram utilizados os ficheiros de exemplo dados pelos docentes da disciplina, com as funcionalidades *time* e *valgrind* para obter os tempos e a memória utilizada por cada teste.



user: tempo total de segundos do CPU usados pelo processo em modo utilizador
system: número total de segundos de CPU usado em modo núcleo.
total: tempo total usado pelo processo, em segundos.



Da análise de ambos os gráficos, não é possível ter uma análise muito conclusiva dado que apenas foram usados os exemplos fornecidos pelos docentes.
Ainda assim acredita-se que se fossem usados mais dados como entrada, seria mais concreto que a solução implementada é linear, dado que à medida que o input vai crescendo, o tempo e a memória vão aumentando de forma linear.

Referências:

- "Problem 22-2" em:
  Introduction to Algorithms, by Thomas H.Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.

- http://pisces.ck.tp.edu.tw/~peng/index.php?action=showfile&file=fb1f19a9be617037cb419c5d454b184bead47e243

- - Pseudocódigo utilizado:
  https://en.wikipedia.org/wiki/Biconnected_component
  https://en.wikipedia.org/wiki/Counting_sort