Análise e Síntese de Algoritmos

1º Projeto

Grupo tg046

Ricardo Gueifão, 87699

Manuel Goulão, 91049

Introdução

Neste relatório vamos analisar o problema proposto no 1º projeto de ASA e ainda a solução por nós encontrada para a resolução do mesmo.
Dada uma rede de routers é nos pedido a identificação de sub redes de routers que no caso de falharem aumentam o número de sub redes e posteriormente a determinação da maior sub rede no caso de todos esses routers serem removidos.

Descrição da solução

Para representar as redes de routers utilizámos grafo com ligações bidirecionais que representamos através de listas de adjacências.
A nossa solução passa por correr duas vezes o algoritmo Tarjan sobre o grafo. Depois de correr a primeira vez retiramos o número de sub redes identificadas, os identificadores e o número de routers que quebram as mesmas. Depois de retirarmos os routers que quebram uma sub rede corremos o Tarjan uma segunda vez e obtemos o número de routers na maior sub rede identificada.
O algoritmo Tarjan que usamos tem algumas alterações que iremos abordar de seguida e que nos permitem a deteção dos routers que quebram as sub redes.
No Tarjan temos guardados como é costume para este algoritmo um grafo, uma stack e um contador de nós visitados. Para além disso e para nos ajudar a resolver o problema de uma forma eficiente temos ainda uma lista com os vértices com o maior id de cada sub rede, um contador com o numero sub redes encontradas, o tamanho da maior sub rede e ainda o número de pontos de articulação na rede.
Um nó do grafo para além das informações normalmente usadas no Tarjan, tempo de descoberta e low, tem ainda o id do nó pai e um booleano que identifica se o nó é um ponto de articulação.
Um nó é um ponto de articulação se verificar um destes dois casos:

- Se o nó for raíz da árvore de procura e tiver apenas 2 ou mais filhos;
- Se o nó, por exemplo v, não for raíz da árvore de procura, tiver uma ligação para um nó u e o low de u for maior ou igual ao tempo de descoberta de v

Algoritmo:

- Começando num nó arbitrário, regista-se o valor do tempo de descoberta, o valor do low e começa-se a contar o número de nós na árvore atual. De seguida escolhe-se um dos filhos, e regista-se o nó pai como predecessor do filho.
- Quando um vértice tiver uma ligação para um nó já explorado este atualiza o seu valor low para o mínimo entre o seu valor low atual e o tempo de descoberta do filho.

- Caso um nó já não tenha mais nós para explorar vai-se recursivamente atualizar o valor do low do pai para o mínimo entre o seu low e o low do filho. E depois verifica-se se nó é um ponto de articulação usando os dois casos possíveis acima referidos.
- Chegando de novo à raiz da árvore, cujo o low é igual ao tempo de descoberta, começamos a remover todos os nós da stack até chegarmos a essa mesma raiz. Durante este processo vamos guardando e atualizando o id da sub rede atual, que corresponde ao maior id de um nó pertencente a essa sub rede. E ainda removemos do grafo os nós que sejam pontos de articulação.
- Repetimos até já não existirem mais nós por explorar

A solução foi implementada em C++

Análise teórica

Depois de uma cuidada análise ao problema, escolhemos a utilização do algoritmo Tarjan, devido aos fatores:

- Facilidade de Implementação: O Tarjan é um algoritmo que resolve o problema apresentado, evitando a necessidade de uma implementação de um algoritmo mais complexo.
- Complexidade: O Tarjan é um algoritmo de baixa complexidade e resolve o problema rapidamente.

Temporal:

O algoritmo Tarjan utiliza os seguintes passos:

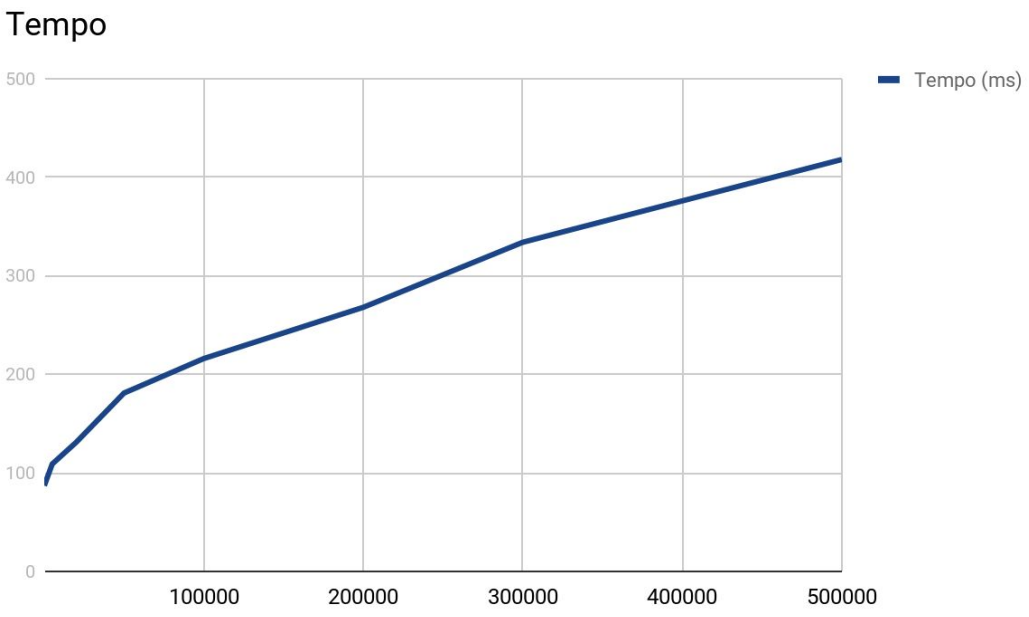
1. Inicialização dos vértices com uma variável a -1, complexidade O(V)
2. Chamadas à função Tarjan_Visit, complexidade O(V)
3. Listas de adjacências de cada vértice analisadas apenas uma vez : O(E)

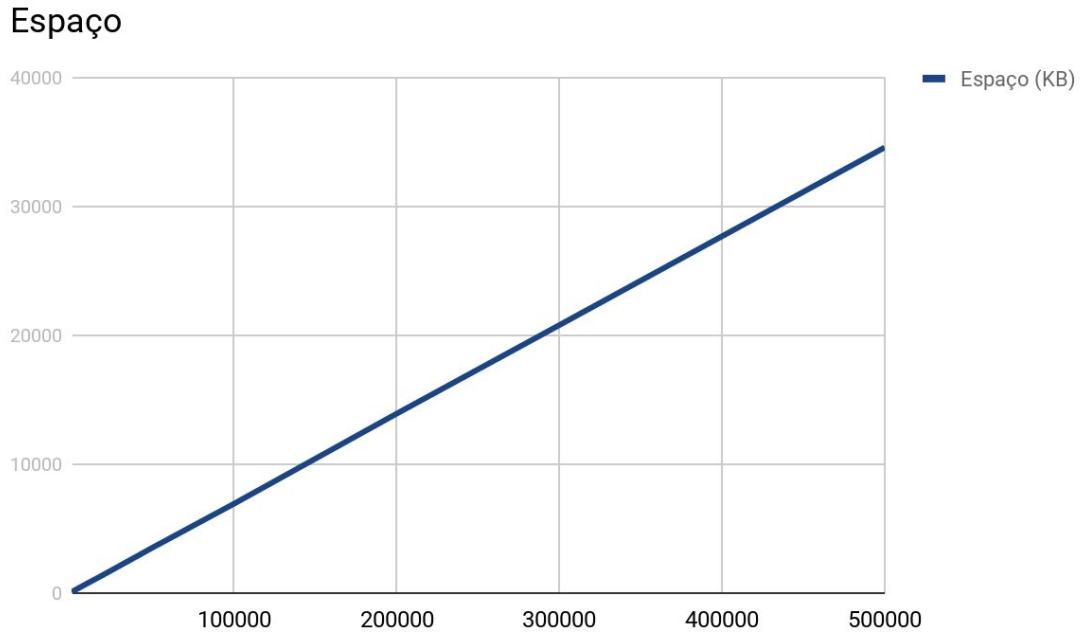
Desta forma, podemos considerar que, no pior caso, a complexidade do algoritmo Tarjan é caracterizada por uma complexidade assimptótica de O(V) + O(V) + O(E) = O(V + E).

No problema apresentado ainda era pedido para identificar o número de nós que, uma vez removidos da componente fortemente ligada, quebram a mesma componente. Para a descoberta destes pontos de articulação foi implementada a solução apresentada em “Descrição da solução”; depois da descoberta dos pontos de articulação, percorremos mais uma vez o algoritmo Tarjan para descobrir o tamanho da maior componente fortemente ligada resultante da remoção destes pontos de articulação. Por isso, no pior caso, teríamos outra vez uma complexidade de O(V + E). Apenas em casos astronómicos, para número elevado de vértices a complexidade total pode ser considerada 2 *O(V + E).

Espacial:
Para representar o grafo nós utilizamos listas de adjacências e portanto a complexidade será O(V + E).

Avaliação experimental dos resultados





V+E	Tempo (ms)	Espaço (KB)
20	87	87
5000	109	434
20000	131	1453
50000	181	3544
100000	216	6942
200000	268	13930
300000	334	20800
500000	418	34580

O primeiro gráfico respeitante ao tempo de execução não é linear. Este resultado não era o expectável depois fazermos a análise teórica, no entanto isto poderá estar relacionado com o facto de o tempo de execução do processo poder facilmente variar, já que existem outros processos a correr na máquina. Para obtermos os tempos de execução utilizámos o comando time(1).
Por sua vez o segundo gráfico já é linear, isto deve-se ao facto da execução de outros processos na mesma máquina não ter qualquer influência na memória usada pelo nosso processo. A obtenção dos valores foi feita utilizando a ferramenta massif do valgrind sendo o valor por nós utilizado o pico de memória usada pelo processo.

Referências

https://cp-algorithms.com/graph/cutpoints.html