Análise e Síntese de Algoritmos

Relatório do 1º projecto

Grupo 96

1 Introdução

O 1º projecto da cadeira de Análise e Síntese de Algoritmos do ano curricular 2018/2019 consiste na identificação de sub-redes. Tem ainda como objectivo identificar os routers da rede que, ao serem atacados ou desligados, resultariam no aumento do número de sub-redes.

O **input** descreve a rede de routers. Cada router é caracterizado por um identificador inteiro entre 1 e N, sendo N o número de routers na rede. As ligações entre dois routers são sempre bi-direccionais. O input é então constituído por:

- uma linha com o número de routers na rede ($N \geq 2$)
- uma linha com o número de ligações entre routers na rede ($M \geq 1$)
- M linhas com dois inteiros separados por um espaço que identificam que existe uma ligação entre estes

No **output** uma sub-rede é identificada pelo router com maior identificador que pertence à sub-rede. É constituído por:

- uma linha com o número de sub-redes
- uma linha com os maiores identificadores de cada sub-rede de routers, separados por espaços em branco e ordenados de forma crescente
- uma linha com um inteiro que denota o número de routers que quebram uma sub-rede
- uma linha com um inteiro que denota o número de routers da maior sub-rede resultante da remoção de todos os routers que quebram uma sub-rede

2 Descrição da Solução

2.1 Linguagem de programação

A implementação do programa foi elaborada na linguagem de programação C++.

2.2 Estruturas de dados

Para a implementação do algoritmo recorremos a um **vetor de vetores de inteiros**, representativo da rede de routers. Recorremos ainda a um **conjunto de 5 vetores auxiliares** para a implementação do algoritmo: três para guardar inteiros(o predecessor, o tempo de descoberta e o low) e dois para guardar booleanos(a cor de cada vértice e os vértices de corte).

2.3 Solução

Para a resolução do problema organizámos os routers num **grafo não dirigido**, uma vez que a direção não é relevante, visto que as ligações entre pontos são feitas nos dois sentidos. Foi identificado que a solução podia ser resolvida utilizando o algoritmo da **DFS**(que permite identificar as sub-redes e o seu identificador máximo) e uma **variação do algoritmo de Tarjan** incorporado na DfsVisit(que permite identificar os routers que sendo desligados ou atacados resultariam no aumento do número de sub-redes e qual o identificador máximo da maior sub-rede criada) fazendo duas passagens pela DFS.

2.4 Algoritmo

O programa começa por ler o input, definindo o número de routers e de ligações existentes entre estes. De seguida, inicia os vetores auxiliares e o vetor principal que representa o grafo. Após isso, coloca as ligações de cada router, ou seja as adjacências dentro do seu vetor correspondente. Estando todos os vetores inicializados, o grafo é tratado pela DFS, que permite descobrir o número de subgrafos existentes e o router com maior identificador em cada um deles, e pela DfsVisit que permite determinar os pontos de articulação do grafo e o identificador(máximo) do maior subgrafo que surge através dos pontos de articulação. Por fim, estes resultados são tratados e é apresentado o output desejado.

O procedimento consiste em:

1. **Ler do input** o número total de vértices, número de ligações entre eles e as próprias ligações. Criar o grafo com a informação dada e em seguida criar e preencher os vários vetores auxiliares.
2. **Passagem pela primeira vez pela função Dfs()** que ao encontrar um vértice que ainda não foi visitado, ou seja, um ponto de desconexão entre grafos aumenta o número de subgrafos existentes e chama a função DfsVisit() (função adaptada para a resolução do problema).
3. Na função **DfsVisit()** são atualizados os valores do tempo, low e cor. É verificado se o identificador do vértice a ser visitado é superior a algum identificador visitado anteriormente no subgrafo em questão. Se for, este é atualizado como sendo o identificador máximo do subgrafo. De seguida, o algoritmo verifica se o vértice é um ponto de articulação ou não.
4. Os valores dos vetores auxiliares são reinicializados para executar a segunda DFS.
5. **Passagem pela segunda vez pela função Dfs()** em que os pontos de articulação são ignorados pelo algoritmo de forma a calcular o maior subgrafo existente sem a existência dos pontos de articulação.
6. Tratamento e posterior impressão dos dados produzidos pela função Dfs() e DfsVisit() correspondentes ao output pretendido.

A parte mais relevante de todo o procedimento é a execução do **algoritmo de Tarjan(adaptado de forma a descobrir os pontos de articulação)** presente na função DfsVisit(). Nesta função é verificado se para o vértice em questão(vértice de origem), os seus vértices adjacentes já foram visitados. Caso a cor do vértice adjacente a visitar seja 0(não visitado) então a função é chamada recursivamente para cada vértice adjacente. De seguida, dá-se a atualização do low(índice mais pequeno de qualquer vértice a que é possível chegar a partir do vértice atual usando a sua árvore DFS, incluindo o vértice atual) do vértice de origem e avaliam-se as seguintes **condições que tornam um vértice um ponto de articulação** caso alguma das duas se verifique:

- O número de filhos do vértice de origem ser superior a 1 e este não ter nenhum predecessor
- O low do vértice adjacente ser igual ao tempo de abertura do vértice de origem e o vértice de origem apresentar predecessor.

Caso o vértice já tenha sido visitado e não seja o predecessor do vértice de origem então atualiza-se o valor do low. Por fim, atualiza-se o valor do tempo.

3 Análise

3.1 Análise Teórica

Serão usados os seguintes parâmetros:

- N: número de routers(vértices do grafo)
- M: número de ligações entre os routers(arcos do grafo)

Na função main existem duas fases: leitura do input e resolução do problema. A leitura do input é feita em $\boldsymbol{\theta(M)}$. A resolução do problema implica duas chamadas à função Dfs() que chama a função DfsVisit(). A complexidade da função Dfs() trata-se de $\mathbf{O(N + M)}$ visto que no pior caso percorre cada um dos vértices e as suas adjacências.

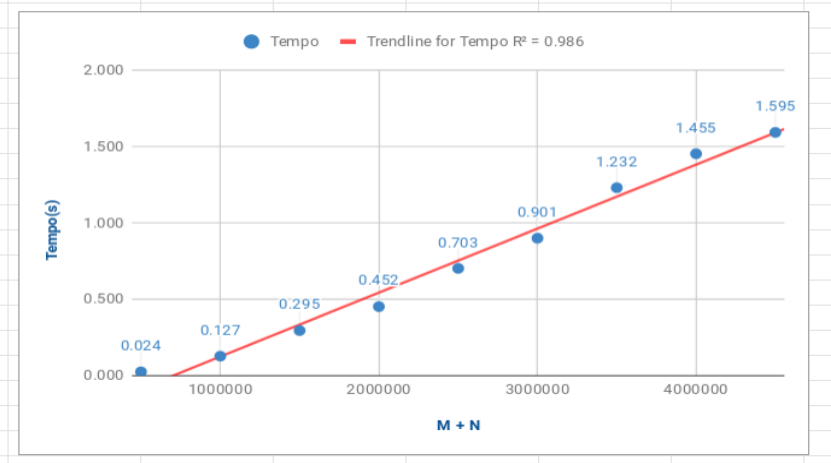
3.2 Análise Experimental

Para verificar a linearidade do tempo de execução do projecto e para uma melhor análise, recorremos ao gerador de grafos fornecido pelos docentes. No qual criamos 9 grafos com vértices entre os 100000 e os 900000 e ligações entre os 400000 e 3600000. O número de subgrafos que utilizamos foi 90000.

Figura 1: Tabela dos valores utilizados para a execução dos testes

N	M	M+N	Tempo
100000	400000	500000	0.024
200000	800000	1000000	0.127
300000	1200000	1500000	0.295
400000	1600000	2000000	0.452
500000	2000000	2500000	0.703
600000	2400000	3000000	0.901
700000	2800000	3500000	1.232
800000	3200000	4000000	1.455
900000	3600000	4500000	1.595

Figura 2: Representação gráfica dos valores



Tendo em conta os valores testados o algoritmo tem a complexidade de O(N+M), no entanto, no caso de um grafo esparso(grafo com um M muito pequeno) a complexidade passa a ser apenas O(N) e no caso de um grafo em que o valor de M é aproximadamente $\boldsymbol{N^2}$, ou seja, um grafo denso então a complexidade será $\mathbf{O}(\boldsymbol{N^2})$.

4 Referências

Os websites/obras consultados para realizar o projecto foram os seguintes:

- https://www.geeksforgeeks.org/articulation-points-or-cut-vertices-in-a-graph/
- https://www.youtube.com/watch?v=aZXi1unBdJA
- **Introduction to Algorithms, Third Edition**: Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein September 2009 ISBN-10: 0-262-53305-7; ISBN-13: 978-0-262-53305-8