**Instituto Superior Técnico**

Análise e Síntese de Algoritmos

**1º Projecto – Relatório**

**Introdução**

Este relatório aborda a nossa solução ao primeiro projeto de Análise e Síntese de Algoritmos do 2º Semestre do ano curricular 2018/2019.

O problema proposto é fazer uma auditoria a uma rede que foi crescendo de forma desordenada e determinar várias propriedades da mesma. Esta rede, pode estar dividida em sub-redes (sub-conjuntos de routers ligados entre si).

O objectivo deste projecto é a partir de um input que nos indica o número de routers (N), o número de ligações (M) e os dois routers envolvidos em cada uma dessas ligações (u e v), conseguirmos obter o número de sub-redes (R), o maior identificador do router pertencente à sua respetiva sub-rede (bigId), o número de routers que quebram uma dada sub-rede (C) e por fim o número de routers da maior sub-rede resultante da remoção de todos os C (pontos de articulação) .

**Descrição da Solução**

Este programa foi implementado em linguagem C.

A partir do input dado, elaboramos o mapeamento dos elementos da rede de um grafo não dirigido, através de três estruturas:

1. Estrutura geral (general), onde guardamos o N, o M, o R, o C, o tempo e dois ponteiros, para dois vetores em que cada uma das posições pertence a um dado router (identificador do router = posição no vector + 1);
2. Um dos vetores contem uma estrutura (link) que é uma lista ligada das adjacências do seu router, possui o identificador do router e um ponteiro para a próxima adjacência;
3. O outro vector contem uma estrutura (extra) que possui variáveis que vão ser importantes para o programa.

Para encontrar o R e o bigId de cada uma das sub-redes e os pontos de articulação baseamo-nos no algoritmo de Tarjan (1º algoritmo). Para encontrar a maior sub-rede após apagar os pontos de articulação baseamo-nos num algoritmo de DFS (2º algoritmo).

O algoritmo Tarjan segue o mesmo princípio que a procura feita numa DFS e guardamos as seguintes informações uteis na estrutura extra para cada um dos vértices:

* A sua profundidade (visitTime);
* O menor valor de profundidade pertencente ao seu ancestral (lowTime);
* Predecessor (pi);
* Número de filhos (nSons);
* Se está terminado, visitado ou não visitado (state);
* Se é o maior identificador da sua sub-rede (bigId);
* Se é um ponto de articulação e vai estar apagado (erased).

Para ser um ponto de articulação tem de satisfazer uma das seguintes condições:

* Vértice root (primeiro vértice visto de cada uma das sub-redes) tem de ter 2 ou mais filhos;
* VisitTime do vértice <= lowTime de um vértice adjacente (edge).

Etapas do 1º algoritmo:

1. Temos um ciclo que vai percorrer todos os routers;
2. Escolhe um router e se este ainda não foi visitado incrementamos o número de sub-redes e sabemos que é uma root, caso contrário volta a este ponto com um novo router;
3. Atualizamos o visitTime, lowTime, state, nSons, tempo e fazemos um ciclo que vai percorrer todas as adjacências;
4. Caso tenha uma adjacência seleciona-a, se já foi visitada e não é uma predecessora do vértice vai para o ponto 5, se não foi visitada vai para o ponto 6, caso contrário volta a fazer este ponto para outra adjacência, por fim se já não têm mais adjacências vai para o ponto 7;
5. O lowTime do vértice é igual ao mínimo entre o seu lowTime e o visitTime da adjacência e volta para o ponto 4, com uma nova adjacência;
6. Verificamos se o id da adjacência é maior que o bigId guardado da sub-rede, se for substituímos, incrementamos o número de filhos, atualizamos o predecessor com o id do vértice e voltamos ao ponto 4 de forma recursiva desta vez pesquisar as adjacências desta adjacência;
7. Atualiza o estado do vértice para terminado e retorna o bigId. Caso o vértice seja uma root vai para o ponto 8, caso contrário vai para o ponto 9;
8. Retornou da recursão, a adjacência atualiza o seu lowTime para o mínimo entre o seu lowTime e o do seu predecessor. Caso ainda não seja um ponto de articulação e satisfaz a segunda condição para ser um ponto de articulação é aumentado o seu número,(C), e atualiza-se o erased. Volta para o ponto 4;
9. O bigId retornado é atualizado, e se satisfizer a primeira condição para ser um ponto de articulação então aumenta-se o seu número,(C), e atualiza-se o erased. Se ainda não pesquisou todos os routers volta para o ponto dois, caso contrário termina.

Para encontrar a maior sub-rede após apagar os pontos de articulação fazemos uma DFS muito simples onde apenas usamos o estado e possuímos uma variável que fica com o maior número de routers numa sub-rede (max).

Etapas do 2º algoritmo:

1. Vamos percorrer o vetor de routers, se o router selecionado estiver apagado ou já tiver sido visitado ignoramo-lo;
2. Caso não tenha sido visitado vamos incrementar o contador de routers e vamos aplicar o ponto 2 às suas adjacências recursivamente;
3. Quando chegamos ao fim da sub-rede comparamos o número de routers com o maior número de routers até ao momento e se este for maior substituímos;
4. Assim no final teremos o número de router da maior sub-rede.

**Análise Teórica**

Em termos de tempo:

* Inicialização das estruturas e das variáveis - O(N), devido a termos de inicializar o vector de routers;
* Leitura de inputs e adicionar as adjacências à lista ligada – O(M), devido a termos de ler todas as conexões;
* 1º algoritmo – É chamado para cada router, O(N), e é chamado para cada uma das suas adjacências que, por ser um grafo não dirigido, é duas vezes o número de conexões, O(M).Assim a combinação das duas complexidades é O(N + M);
* 2ª algoritmo – O(N + M), pela mesma razão que o 1º algoritmo;
* Imprimir resultados – O(R), pelo facto de termos de imprimir o maior id de cada sub-rede;
* Fazer free – O(N + M), pela mesma razão que o 1º e 2º algoritmo;

Assim, o nosso projecto tem teoricamente complexidade temporal O(N+M).

Em termos de memória:

Temos um vector que tem N routers, O(N), e por sua vez cada um destes tem a sua própria lista de adjacências, que no total são 2M conexões, O(M). Não havendo mais nenhuma ocupação de memoria com uma escala de crescimento maior que esta podemos concluir que a complexidade espacial teórica é O(N + M).

**Avaliação Experimental**

Informações sobre a plataformas de testes - Virtual Box - Ubuntu 16.04 – Intel® Core™ i7-7700HQ – Clock Rate: 2.80GHz – 4 cores - 10Gb RAM.

Estes gráficos foram construídos com dados obtidos a partir de um script feito em bash shell que consistia em 2 ciclos que permitiu variar o N, o M e o R(100 ou 1000), de forma crescente.

Em termos de Tempo (130 testes):

Cada teste foi feito 1000 vezes e o tempo foi medido a partir do “user time” da função time.

Em termos de memória (250 testes):

Para cada teste, os picos de memória foram obtidos a partir da ferramenta massif do valgrind.

Resultados experimentais:

Os dois gráficos permitem constatar que o programa desenvolvido é, de facto, linear, dado que a memória e o tempo utilizados crescem proporcionalmente com o número de routers + número de conexões (N+M), comprovando assim os resultados obtidos teoricamente.

**Referências**

Como pontos de referência utilizamos:

* <https://www.youtube.com/watch?v=2kREIkF9UAs>;
* Introduction to Algorithms, Third Edition: Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein