Relatório do 1ºProjeto de ASA

Introdução

O primeiro projeto de ASA consistia em ajudar o Sr. João Caracol em montar e gerir redes de router seguras. Para realizar esta tarefa, representamos as redes como grafos, os routers como vértices e as ligações como arestas.

O Sr. João Caracol queria as seguintes funções:

- Descobrir o número de sub-redes na sua rede. No contexto do nosso projeto consiste em descobrir subgrafos.
- Descobrir o maior identificador das sub-redes. No contexto do nosso projeto consiste em descobrir o vértice com maior identificador de cada subgrafo.
- Descobri o número de routers que quebram uma sub-rede. No contexto do nosso projeto consiste em descobrir os pontos de articulações de cada subgrafo.
- Descobrir a dimensão da maior sub-rede após remove os routers que quebram sub-redes. No contexto do nosso projeto consiste em descobrir a dimensão da maior subgrafo.

Descrição da Solução

A nossa solução consiste em utilizar um DFS modificado, que se parece bastante com o algoritmo de Tarjan, no entanto não é utilizado para encontrar ***SCC (Strongly Connected Components)***, mas sim com o objetivo de encontrar pontos de articulação.

Para representar, os grafos, os vértices e as arestas utilizamos as seguintes *structs*:

1. ***Struct graph***: Representa o grafo e consiste em **int numVertexes**, que contem o número de vértices, um **int numEdges** que contem o número de arestas, uma **lista de ponteiros para struct vertexs vertexList**, um **node idList**, que contem os identificadores de sub-redes, o **int C** que contem o número de router de que quebram uma sub-rede e **int Max** que representam a dimensão do maior sub-rede depois de tirar os routers que quebram.

2. ***Struct vertex:*** Representam os vértices do grafo e consistem em um **int parent** que é o seu predecessor, um **int startime** que é o seu tempo de descoberta, o **int low** que é o identificador do vértice mais baixo que pode ser chegado pelo o vértice, um **bitfield de unsigned char ap** que representa um se o vértice é um ponto de articulação e por ultimo um **ponteiro para um struct node adj** que consiste numa lista de adjacências, que contem os identificadores dos vértices aos quais este vértices se encontra ligado.

3. ***Struct node***: Não representam exatamente as arestas, mas devido ao facto de representarmos as ligações entre vértices como uma lista de adjacência, pode dizer-se que representam as arestas. Cada struct node contem um **int value**, que corresponde ao vértice de destino da aresta e o **ponteiro para outro struct node next**, que corresponde ao próximo struct node da lista. Se é o último da lista, então o valor de next é NULL.

O programa começa por ler do standard input, o número de vértices, número de arestas e por fim as ligações(arestas) entre esses vértices. Constrói um ***struct graph*** com essas informações na função ***graphInit***.

A função ***DFS*** consiste em percorrer a lista de vértices. Se o vértice tiver não tiver sido visitado (startime != NIL), então realiza a função ***DFSAlgorythm*** e a adiciona o valor do vértice a idList, senão segue para o próximo vértice da lista. Depois de percorrer a lista de vértices, então imprime o número R, o número de subgrafos, depois imprime a idList, que corresponde aos maiores identificadores dos R subgrafos e por último imprime C que é o número de pontos de articulação.

A função ***DFSAlgorythm*** é executada num vértice, começa por meter o startime e o low ao valor do D, que é um ponteiro para o int que representa o *discovery time*. Logo a seguir, incrementa-se ao *discovery time*. Inicializa-se o childNum a 0, que é o número de filhos que este vértice tem na arvore DFS. Cria-se um Node v, para percorrer a lista de adjacências do vértice. Verifica-se o vértice da lista de adjacência foi visitado (startime != NIL). Se tiver sido visitado, então verifica se o vértice adjacente não é pai do vértice e se o low do vértice é maior que o startime do vértice adjacente, o low do vértice passa a ser o startime do vértice adjacente. Se não tiver sido visitado mete o parent desse vértice, como o vértice, incrementa-se o childNum e percorre-se o DFSAlgorythm no novo vértice (vértice adjacente). Após retornar do DFSAlgorythm, atualiza o low do vértice e entra entre dois ifs para verificar se é ponto de articulação ou não. Um vértice é ponto de articulação se DFS se for uma raiz (parent == NIL) e tem dois ou mais filhos(childNum) ou se não é raiz(parent != NIL) e o low do vértice adjacente e maior ou igual ao startime do vértice, ou seja, nenhum dos seus descentes tem um ***backwards edge*** para um predecessor dele . Se for ponto de articulação a variável ap do vértice passa a ser 1.

Após se realizar o DFS, realiza-se o ***CutLength***, que faz um DFS mais simples, mas em vez de apenas verificar se foi visitado, verifica também se o vértice é não é ponto de articulação (ap = 0). Se a verificação for verdadeira, então entra na função ***Deep*** para calcular o tamanho com o variável max. Quando acaba o DFS imprime no standard input o valor do variável max.

A função ***Deep*** é um DFSAlgorythm, mas que apenas visita, ou seja, utiliza apenas o startime e soma o variável ***length***, que é passado entre chamadas porque é um ponteiro.

Por último, faz-se frees para limpar a memoria utilizada a correr os algoritmos, com função ***freeGraph***.

Análise Teórica

Na alocação de memória para os vértices percorremos a lista de adjacências, o que leva a uma complexidade $\Theta(V)$.

Quando adicionamos uma aresta ao grafo, temos uma complexidade O(1), pois adicionamos a mesma sempre no início da lista ligada.

No total, estima-se uma complexidade O(V) na inicialização e construção integral do grafo.

Parte da nossa solução ao problema baseou-se no algoritmo de Tarjan simples, a qual apresenta uma complexidade O(V+E).

O algoritmo utilizado para achar o número de routers da maior sub-rede resultante da remoção de todos os C routers que quebram uma sub-rede baseou-se num *depth-first search*, tendo uma complexidade O(V+E).

Concluindo, a complexidade total do nosso programa é O(E+V).

Avaliação experimental dos resultados

Para a realização da avaliação experimental dos resultados, usamos um script desenvolvido em *python* que gera vários inputs e avalia o seu tempo de execução e que gera gráficos para os mesmos, sendo que foram realizados mais de 200 testes para cada situação.

O gráfico 1 traduz a relação entre o tempo (milissegundos) e um número crescente tanto de arestas como de vértices (milhares) de 0 a 10000.

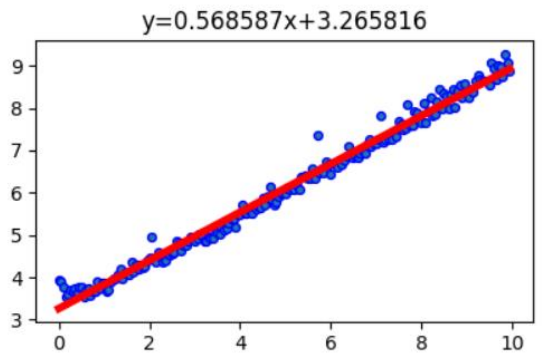

Gráfico 1: Relação entre tempo (eixo yy) e número vértices+arestas (eixo xx).

No gráfico 2 podemos observar a relação entre o tempo(milissegundos) e um número crescente de arestas(milhares) de 0 a 10000 e 4000 vértices fixos.

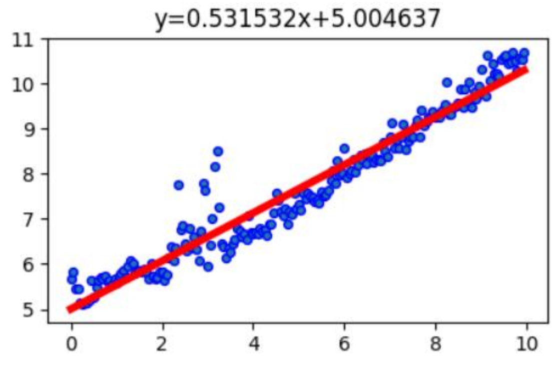

Gráfico 2: Relação entre tempo (eixo yy) e número de arestas (eixo xx).

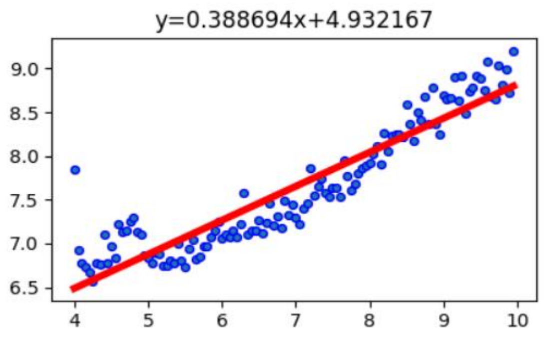

Gráfico 3: Relação entre tempo (eixo yy) e número vértices (eixo xx).

No gráfico 3 está explicito a relação entre o tempo(milissegundos) e um número crescente de vértices(milhares) de 0 a 10000 e 4000 arestas fixas.

No gráfico 4 encontra-se explícito a relação entre o tempo (milissegundos) e o espaço ocupado, stack e heap (mb).

Concluindo, podemos observar que os resultados obtidos em todos os gráficos estão de acordo com os resultados teóricos, havendo assim uma relação linear em todas as situações.

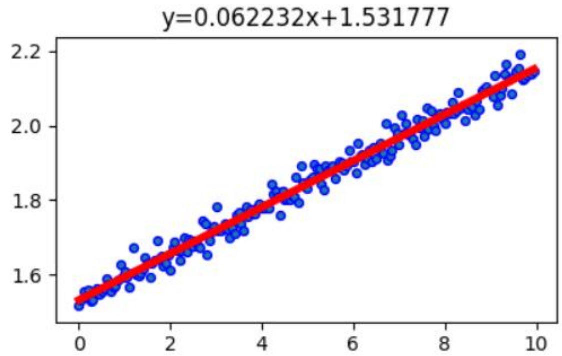

Gráfico 4: Relação entre o tempo (eixo yy) e o espaço+stack+heap (eixo xx)

Referências

https://en.wikipedia.org/wiki/Tarjan%27s_strongly_connected_components_algorithm (visitado a 18/03/2019)

https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/articulations.html (visitado a 18/03/2019)

https://stackoverflow.com/questions/44983431/time-complexity-of-adjacency-list-representation (visitado a 18/03/2019)

https://www.geeksforgeeks.org/articulation-points-or-cut-vertices-in-a-graph (visitado a 14/03/2019)

Cormen, Thomas H., Charles Eric. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms. Cambridge, MA: MIT, 2009. Print.

Relatório realizado por G031:

Pedro Moreira ist190768

Miguel Mota ist190964