Primeiro projeto de Análise e Síntese de Algoritmos 2015-2016
Relatório

Ana Sofia Costa 81105 João Pedro Silvestre 80996

19 de Março de 2016

1 Introdução

O presente relatório apresenta a análise ao problema, solução e respetivos testes referentes ao segundo projeto desenvolvido no ambito da UC de Análise e Síntese de Algoritmos do primeiro semestre do ano escolar 2015/2016. A primeira secção descreve o problema proposto no âmbito das redes sociais. A segunda secção reporta o modo de solucionar o problema proposta través de optimização algoritmica. A terceira secção apresenta a análise teórica que sustenta a solução descrita.

1.1 Problema

O problema proposto consiste na analise de difusão de informação entre redes sociais. Assim, o objetivo consiste na analise da comunicação entre utilizadores e aferir quais as pessoas fundamentais numa rede, ou seja, pessoas pelas quais a informação tem de passar obrigatoriamente para ser difundida.

1.2 Input

O input fornecido é constituido por um linha que contém o número de pessoas (V) e número de arestas entre estas (E), seguido de E linhas em que cada linha representa um ligação, sendo esta constituida por dois inteiros separado por um espaço em branco, indicando que o primeiro partilha informação com o segundo. Estes inteiros são numerados de 1 a V, representando o número identificativo de cada pessoa.

1.3 Output

O output indica, na primeira linha, o número de pessoas fundamentais na rede (F), na segunda o menor(m) e maior (M) identificador fundamental. Caso não haja pessoas fundamentais na rede, ou seja, F = 0, o mínimo e o máximo serão -1.

2 Descrição da Solução

2.1 Linguagem de Programação

Para implementar uma solução foi escolhida a linguagem C++. Das três possibilidades oferecidas, esta revelou-se mais adequada pois permite a gestão de memória e tempo. Permite também utilizar estruturas já implementadas em bibliotecas.

2.2 Solução

A partir do enunciado foi possível aferir que a solução se debatia num grafo não dirigido, pois a interação de duas pessoas numa rede é bidirecional, assim, as pessoas são respresentadas como vértices (V) e as relações como arestas (E). Na procura dos vértices fundamentais identificou-se a necessidade

de eliminar parte dos vértices que correspondiam a Componentes Fortemente Ligados do grafo. Para os identificar utilizou-se um algoritmo de Tarjan modificado para grafos não dirigidos.

DFS: "Grafo pesquisado dando prioridade aos arcos dos vértices mais recentemente visitados. Dado G=(V,E), não dirigidos, cada arco é arco de árvore ou arco para trás."

2.3 Estrutura de Dados Desta forma, considerou-se mais conveniente implementar uma estrutura semelhante à lista de adjacências, pois esta é vantajosa na análise de arestas e na sua adição. A estrutura implementada foi a seguinte:

- **class Vertex** que tem quatro inteiros (número do vértice _num, tempo de descoberta _dis, o tempo mais curto _low, número referente ao vértice pai _parent) e dois booleanos (indicação se o vértice já foi visitado _visited, indicação se é fundamental).
- **typedef list<Vertex*> adlist**; O primeiro ponteiro da lista corresponde a um vetor(pai) e todos os outros correspondem a ponteiros de vértices seus adjacentes.
- **class Graph** que possui cinco inteiros (número total de vértices _vertexNum, o tempo de pesquisa _time, o número de vértices fundamentais _nFund, o número fundamental máximo e minimo, respetivamente, _minf e _maxf) e um vetor **vector< adlist>** _graph.

A razão de escolha das estruturas anteriormente mencionadas prende-se com o facto de ser possível navegar de forma praticamente imediata entre vértices.

2.4 Algoritmo

Com vista a resolver o problema proposto pela UC foram selecionadas dois algoritmos analisados nas aulas teóricas: DFS e Tarjan. Após uma revisão bibliográfica, concluímos que o algoritmo de Tarja utiliza o algoritmo DFS como base na pesquisa de um grafo e tem como principal objetivo identificar componentes fortemente ligados. Averiguamos também que existia uma derivação do algoritmo de Tarjan para grafos não dirigidos cuja complexidade é O(|V| + |E|) . Assim optou-se por escolher uma adaptação deste algoritmo como solução.

Para navegar, explorar o grafo e descobrir os vértices fundamentais, implementámos uma variação do Algoritmo de Tarjan. Esta variação consiste na utilização do algoritmo Tarjan, embora com uma finalidade diferente. A forma de determinar se um vertise, é fundamental é apartir da condição seguinte:

```
if(((begin->getParent()) == NIL && (nChildren > 1))
  || ((begin->getParent() != NIL) && (a->getLow() >= begin->getDis() ))){
    if( begin->getFund() == false){
```

Figura 1: Excerto do Algoritmo

Ou seja, é feita inicialmente uma verificação se o vértice é o vértice de origem, caso seja, é necessario verificar se o seu número de filhos é maior do que um, caso contrário não pode ser considerado fundamental. Caso não seja origem é necessario verificar se pertence a algum componente fortemente ligado, comparando o low do vértice e o tempo de descoberta do seu pai. Caso seja aceite, é necessario verificar se o vértice já foi incrementado aos vértices fundamentais.

3 Análise Teórica

(Com o intuito de simplificar a compreensão, o vértice será designado por V e a ligação entre dois vértices E.)

Inicialmente, na criação do grafo o algoritmo recebe o número de vertices que são necessários para efetuar a criação do grafo que executa um ciclo *for* para criar V listas dentro do vetor O(V). Posteriormente entra num ciclo *for* para adicionar à lista de adjacências de cada um dos vértices uma ligação, assim, a sua complexidade é um O(E) e a criação do arco é realizada em O(1)

$$\Theta\left(E\right) \tag{1}$$

Após a criação da estrututa e armazenamento dos vértices adjacentes é chamada a **função DFS** que chama a função **FundFinder** (que corresponde à alteração do algoritmo de Tarjan). FundFinder é chamado uma vez por cada vértice,

$$v \in V \tag{2}$$

desde que o vértice ainda não tenha sido visitado. Quando entra no loop for da linha 103 à 112 é executado consoante o número dos vértices adjacentes

$$|Adj\left[V\right]| \tag{3}$$

Como o algoritmo descrito acima é executado uma vez por cada vértice, a sua complexidade é:

$$\sum_{v \in V} \left(V + |Adj\left[V\right]|\right) = \theta\left(V + E\right) \tag{4}$$

A complexidade da receção do input, da execução do algoritmo e retorno do output corresponde à seguinte soma:

$$\theta\left(V\right) + \theta\left(E\right) + \theta\left(V + E\right) + \theta\left(1\right) \tag{5}$$

4 Avaliação Experimental

Neste secção é feita um analise dos resultados obtido de forma a comprovar a complexidade do algoritmo implementado. Nesta análise é executada uma série de testes, gerados por um programa que cria inputs e é medido a duração de execução do código realizado.

Os inputs gerados para os testes seguintes são aleatórios, havendo variação na sua densidade e no numero de pessoas e as suas correspondentes ligações.

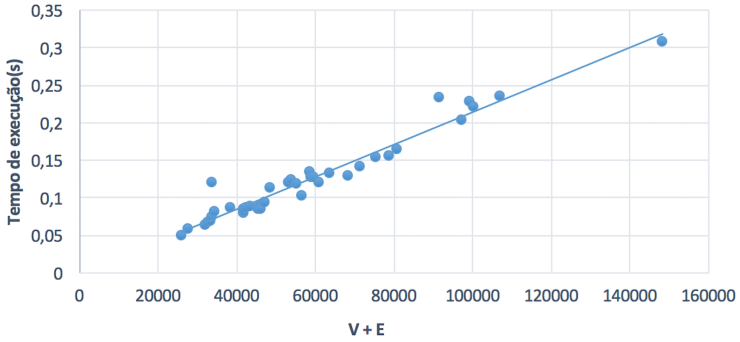

Figura 2: Gráfico da avaliação experimental

Nos testes seguintes, foram testados 2 casos contrarios, o primeiro foi gerado inputs, com o minimo de ligações entre pessoas, ou seja, o grafo terá V vertices e V-1 arestas, em que o resultado seria o máximo de fundamentais possíveis. No segundo caso é gerado inputs com o máximo de ligações entre pessoas, isto é,

$$\frac{V(V-1)}{2}(arestas) \tag{6}$$

onde o resultado será não encontrar pessoas fundamentais(devido à grande quantidade de ciclos) e consequentemente não haver nem máximo nem mínimo fundamental.

Para estas ultimas experiências utilizamos grafos esparsos como sendo grafos em linha, ou seja, os vertices tinham todos 2 vertices adjacente excepto o inicial e final. Em relação aos grafos densos são grafos com maior quantidade de arestas possível e onde há uma grande quantidade de ciclos.

Tipo Grafo\Numero de vértices(V)	100	1000	10000	100000	1000000	10000000
Esparso (Mínimo ligações)	0,002 (s)	0,01 (s)	0,039 (s)	0,18 (s)	1,044 (s)	8,642 (s)
Denso (Máximo ligações)	0,013 (s)	0,302 (s)	32,109 (s)	-	-	-

Figura 3: Tabela

5 Conclusão

Dos resultados apresentados,verificou-se que o algoritmo é linear proporcionalmente ao tamanho do input. É de notar que o algoritmo aumenta o tempo de execução quando o número de arestas se aproxima o máximo possivel ao número de vértices existentes, isto deve-se ao facto de haver uma maior quantidade de ciclos no grafo e o gasto em processamento aumentar.

Referências

[1] Tarjan, R. (1972). Depth-first search and linear graph algorithms. SIAM journal on computing, 1(2), 146-160.