

# Análise e Síntese de Algoritmos

1º Projeto

Tomás Cunha, nº 81201, Grupo 15

## 1. Introdução

Este trabalho tem como objetivo desenvolver um algoritmo que permita descobrir as pessoas fundamentais de uma rede. Uma pessoa é considerada fundamental se o único caminho para partilha de informação entre outras duas pessoas passa obrigatoriamente por essa pessoa. Este problema pode ser reduzido a encontrar os vértices de corte de um grafo conexo não dirigido, em que os vértices correspondem às pessoas e as arestas às ligações entre as mesmas, sendo semelhante ao problema 22-2 do livro *Introduction to Algorithms*[?, p. 622], em cujos exercícios me baseei para desenvolver a solução.

## 2. Descrição da Solução

O algoritmo utilizado na solução deste problema é uma variação do algoritmo DFS estudado na aula, usando também a noção de *low* que é usada no algoritmo de Tarjan para encontrar componentes fortemente ligadas. Este *low* é usado para encontrar o vértice menos profundo no grafo ao qual é possível chegar a partir de um dado vértice, permitindo encontrar arcos para trás. O grafo é representado por listas de adjacências.

A solução, cuja justificação teórica será dada na secção seguinte, pode ser representada em pseudocódigo da seguinte forma:

---

**Algorithm 1:** Inicializar as estruturas necessárias

---

```
1 function initialize-graph( $G$ )
2   foreach  $v \in V[G]$  do
3      $d[v] \leftarrow \infty$ ;
4      $\Pi[v] \leftarrow \text{NIL}$ ;
5      $low[v] \leftarrow \infty$ ;
6   fundamental-count  $\leftarrow 0$ ;
7   min-element  $\leftarrow +\infty$ ;
8   max-element  $\leftarrow -\infty$ ;
9   time  $\leftarrow 0$ ;
```

---

---

**Algorithm 2:** Encontrar os vértices de corte

---

```
1 function Find-Fundamental-Vertices( $u$ )
2    $d[u] \leftarrow \text{time};$ 
3    $\text{low}[u] \leftarrow \text{time};$ 
4    $\text{time} \leftarrow \text{time} + 1;$ 
5    $\text{is-fundamental} \leftarrow \text{false};$ 
6    $\text{child-count} \leftarrow 0;$ 
7   foreach  $v \in \text{Adj}[u]$  do
8     if  $d[v] = \infty$  then
9        $\Pi[v] \leftarrow u;$ 
10       $\text{child-count} \leftarrow \text{child-count} + 1;$ 
11      Find-Fundamental-Vertices( $v$ );
12      if  $\text{Low}[v] \geq d[u]$  then
13         $\text{is-fundamental} \leftarrow \text{true};$ 
14         $\text{Low}[u] \leftarrow \text{MIN}(\text{Low}[u], \text{Low}[v]);$ 
15      else if  $v \neq \text{Parent}[u]$  then
16         $\text{Low}[u] \leftarrow \text{MIN}(\text{Low}[u], d[v]);$   $\triangleright$  Possível arco para trás
17  if ( $\text{Parent}[u] \neq \text{NIL}$  and  $\text{is-fundamental}$ ) or
    ( $\text{Parent}[u] = \text{NIL}$  and  $\text{child-count} > 1$ ) then
18     $\text{fundamental-count} \leftarrow \text{fundamental-count} + 1;$ 
19     $\text{min-element} \leftarrow \text{MIN}(\text{min-element}, v);$ 
20     $\text{max-element} \leftarrow \text{MAX}(\text{max-element}, v);$ 
```

---

---

**Algorithm 3:** Função principal

---

```
1 Let  $G$  = grafo formado a partir do input;
2 initialize-graph( $G$ );
3 Find-Fundamental-Vertices(1);
Output: fundamental-count, min-fundamental, max-fundamental
```

---

### 3. Análise Teórica

Os dados do enunciado garantem que haverá sempre um caminho entre qualquer par de pessoas, o que indica que a floresta obtida após uma DFS será composta por uma única árvore. Há então dois tipos possíveis de vértices que é necessário avaliar: a raiz da árvore, e os restantes vértices.

No caso de o vértice ser a raiz da árvore, será um vértice de corte se e só se tiver pelo menos dois descendentes. É fácil provar que isto se sucede: se tiver apenas um descendente, uma vez que é a raiz da árvore, não poderá ser um vértice de corte, pois não tem antecessores aos quais o seu antecessor já não estaria ligado. Se tiver dois ou mais descendentes, uma vez que a procura é em profundidade primeiro, isso significa que, após a recursão terminar para o primeiro filho (ou, no caso de ter  $n$  filhos, para os primeiros  $n-1$  filhos), o descendente restante não era adjacente a nenhum outro vértice da árvore. Isto significa que, ao remover a raiz da árvore, haverá uma divisão da árvore em duas (ou mais) subárvores.

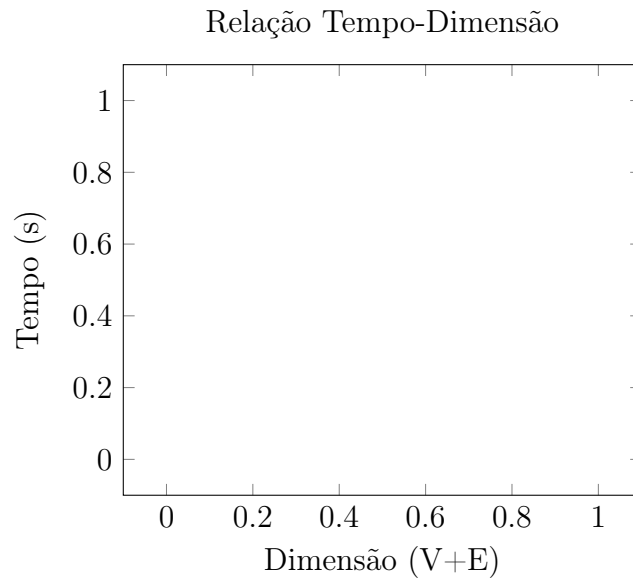
No caso de ser outro tipo de vértice,  $v$  será um vértice de corte se e só se tiver um descendente  $w$  tal que nem  $w$  nem os seus descendentes têm um arco para trás para um antecessor do vértice  $v$ . A prova desta afirmação é também simples. Se  $w$  não tiver um arco para trás que o ligue a um antecessor de  $v$ , e nenhum dos seus descendentes tiver esta ligação, a remoção de  $v$  irá cortar a única ligação de  $w$  e dos seus descendentes à restante árvore, que será composta por pelo menos um vértice, uma vez que  $v$  não é a raiz da árvore. Se o contrário se suceder, ou seja, ou  $w$  ou um dos seus descendentes tem um arco para trás, então mesmo que  $v$  seja removido haverá uma ligação entre os antecessores de  $v$  e os seus descendentes, ou seja, a remoção de  $v$  não irá ter um efeito no número de componentes ligadas, logo  $v$  não será um vértice de corte.

Estas duas observações são suficientes para chegar à solução do problema: basta guardar o *lowpoint* de cada vértice ao longo da visita, ou seja, a profundidade mais baixa à qual os descendentes de um dado vértice  $v$  estão ligados, e conseguimos saber se estes têm ou não um arco para trás. Para encontrar o vértice raiz da árvore, basta ver qual o vértice que não tem predecessor.

A complexidade da função **Initialize-Graph** é  $\Theta(|V|)$  uma vez que percorre todos os vértices do grafo uma única vez. Analisando as linhas do **Algorithm 2** da secção anterior, vemos que as linhas 2–6 e 17–20 são realizadas em  $\Theta(1)$ , havendo um ciclo nas linhas 7–16 que é executado  $E$  vezes, em que  $E$  representa o número de arestas do vértice. A linha 10 chamará a função recursivamente para todos os vértices do grafo, uma única vez. Como cada chamada à função percorre todas as arestas de cada vértice, e é chamada uma única vez para cada vértice do grafo, a complexidade total da função, e consequentemente do algoritmo, será  $\Theta(|V| + |E|)$ , como seria expectável visto que se trata de uma variação da DFS habitual.

## 4. Avaliação Experimental dos Resultados

Foram realizados 100 testes, aplicando o algoritmo a grafos com entre 20 000 e 2 000 000 Vértices+Arestas. Após desenhar um gráfico com os resultados da medição do tempo de execução desses testes em função da dimensão do grafo, é possível observar que é próximo do gráfico de uma função do tipo  $y = mx + b$ , sendo coerente com a complexidade assintótica  $\Theta(|V|+|E|)$  obtida na análise teórica do algoritmo.



## Referências

- [1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms*, 3rd Edition, September 2009