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*[1, 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

```
function initialize-graph(G)
    foreach v ∈ V[G] do
        d[v] ← ∞;
        Π [v] ← NIL;
        low[v] ← ∞;
    fundamental-count ← 0;
    min-element ← +∞;
    max-element ← −∞;
    time ← 0;
```

Algorithm 2: Encontrar os vértices de corte

```
function Find-Fundamental-Vertices(u)
    d[u] ← time;
    low[u] ← time;
    time ← time + 1;
    is-fundamental ← false;
    child-count ← 0;
    foreach v ∈ Adj[u] do
        if d[v] = ∞ then
            Π[v] ← u;
            child-count ← child-count + 1;
            Find-Fundamental-Vertices(v);
            if Low[v] ≥ d[u] then
                is-fundamental ← true;
            Low[u] ← MIN(Low[u], Low[v]);
        else if v ≠ Parent[u] then
            Low[u] ← MIN(Low[u], d[v]);          ▷ Possível arco para trás
    if (Parent[u] ≠ NIL and is-fundamental) or
     (Parent[u] = NIL and child-count > 1) then
        fundamental-count ← fundamental-count + 1;
        min-element ← MIN(min-element, v);
        max-element ← MAX(max-element, v);
```

Algorithm 3: Função principal

```
Let G = grafo formado a partir do input;
initialize-graph(G);
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.

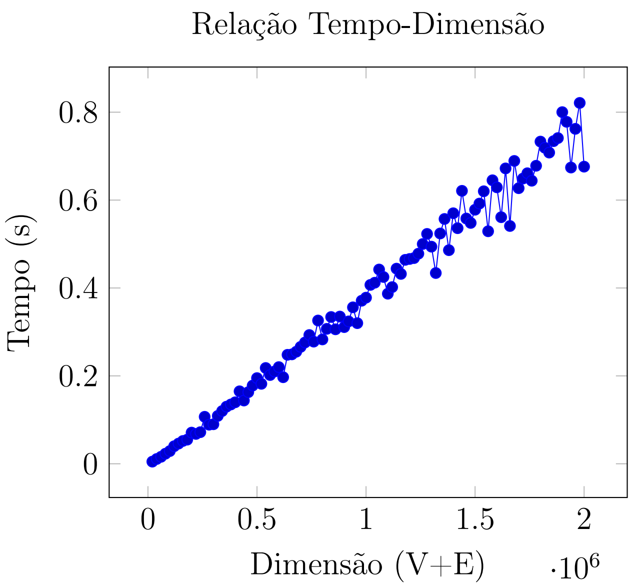
No caso de ser outro tipo de vértice, v será um vértice de corte se e só se tiver um descendente w tal que nem w nem os seus descendentes têm um arco para trás para um antecessor do vértice v. A prova desta afirmação é também simples. Se w não tiver um arco para trás que o ligue a um antecessor de v, 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 assimptótica $\Theta(|V| + |E|)$ 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