MO417 - ATA de Exercício

Priscila Tiemi Maeda Saito - RA100576

19 de maio de 2010

Exercício 22.4-2

Forneça um algoritmo de tempo linear que tome como entrada um grafo acíclico orientado $G = (V, E)$ e dois vértices s e t, e retorne o número de caminhos de s para t em G. Por exemplo, no grafo acíclico orientado da Figura 1, existem exatamente quatro caminhos do vértice p para o vértice v: pov, $poryv$, $posryv$ e $psryv$. (Seu algoritmo só precisa contar os caminhos, não listá-los.)

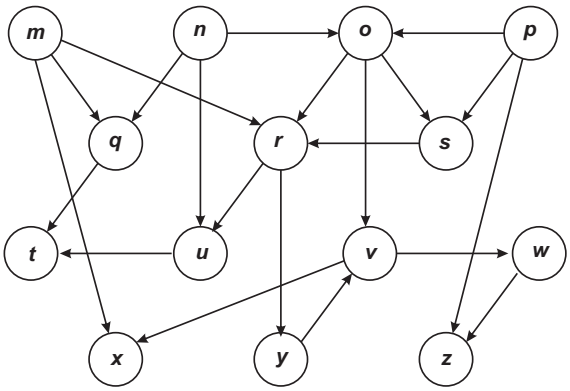

Figura 1: Representação de um gao (grafo acíclico orientado).

Resolução

Seja ST o conjunto de vértices contidos em caminhos de s para t, a solução para o exercício é calcular para cada vértice de ST, de quantas maneiras pode-se chegar dele a t.

A estratégia do algoritmo é realizar uma busca em profundidade tendo s como raiz até chegar em t, repetindo-se o processo de busca para cada vértice pertencente a este caminho, de forma a obter-se caminhos alternativos de s a t.

A solução dada ao exercício é representada pelos Algoritmos 1 e 2, referentes à contagem dos caminhos e ao programa principal, respectivamente.

A análise do algoritmo, tanto em termos de seu funcionamento como de sua complexidade, é apresentada a seguir:

Análise do Algoritmo

- **Descrição do Funcionamento**

  O vetor $C[v]$ é utilizado para a contagem do número de caminhos existentes entre v e t.

  Sendo assim, para todos os vértices v pertencentes ao grafo, $C[v]$ é inicializado com o valor -1, ou seja, nenhum vértice, até então, foi encontrado (linhas 3 a 5 do Algoritmo 2).

  Na linha 6 do Algoritmo 2, $C[t]$ é inicializado com o valor 1, pois há um único caminho entre t e t no grafo. Além disso, nenhuma pesquisa deve ser realizada aos seus filhos, a recursão, portanto, deve parar quando encontrá-lo.

  A função recursiva $Conta\_Caminhos(v)$, representada pelo Algoritmo 1, é chamada inicialmente para o vértice de partida s (linha 7 do Algoritmo 2).

Algoritmo 1: Conta_Caminhos(v)

```
1  begin
2  |  if (C[v] ≠ −1) then
3  |  |  return C[v];
4  |  end
5  |  C[v] ← 0;
6  |  foreach u ∈ Adj[v] do
7  |  |  C[v] ← C[v] + Conta_Caminhos(u);
8  |  end
9  |  return C[v];
10 end
```

Para o vértice v, $C[v]$ é calculado como a soma do número de caminhos existentes entre cada vértice u e t, sendo u adjacente a v. Se $C[u]$ não foi encontrado ($C[u] = -1$), o mesmo princípio é aplicado a u, chamando a função $Conta\_Caminhos(u)$ recursivamente (linhas 6 a 8).

Algoritmo 2: main()

```
1 begin
2 |  ...
3 |  foreach v ∈ V[G] do
4 |  |  C[v] ← −1;
5 |  end
6 |  C[t] ← 1;
7 |  return Conta_Caminhos(s);
8 end
```

- **Descrição da Complexidade**

  A complexidade do algoritmo apresentado como solução é exatamente igual à de uma busca em profundidade comum, pois a manipulação extra de $C[v]$ não aumenta a complexidade assintótica da chamada à busca em profundidade para o vértice v. Assim como na busca

em profundidade (Seção 22.3 de [Cormen et al., 2002]), cada vértice do grafo $G(V, E)$ é visitado uma única vez e cada aresta é analisada uma única vez também.

Dessa forma, a função $Conta\_Caminhos(v)$ é chamada uma vez para cada vértice. Sendo assim, o custo apresentado pelo Algoritmo 1 é dado por: linhas 2 e 3 apresentam um custo $O(V)$, dadas as $|V|$ chamadas à função. As linhas 6 a 8 apresentam um custo $O(E)$, pois a linha 7 é executada, na chamada para o vértice v, o grau de v vezes. Como o vértice v é visitado uma única vez, o custo da linha 7 é a soma dos graus de saída dos vértices, a qual corresponde à $|E|$. As linhas 5 e 9 apresentam um custo $O(V)$.

Com relação ao $main()$ (Algoritmo 2), as linhas 3 e 4 apresentam um custo $O(V)$ e a linha 6 apresenta um custo $O(1)$. Portanto, a complexidade total do algoritmo (Algoritmos 1 e 2) apresentado como solução ao exercício é **linear**, mais especificamente $O(V + E)$.

Referências

[Cormen et al., 2002] Cormen, T. H., Leiserson, C. E., Rivest, R. L., e Stein, C. (2002). *Algoritmos: Teoria e Prática*. Tradução da Segunda Edição Americana, 2 edition.