Ata de exercício
MO417 - Análise de Algoritmos

Daniel Cason – RA100583

6 de junho de 2010

Questão 22.4-3 Faça um algoritmo que determina se um grafo não-direcionado $G = (V, E)$ contém algum ciclo ou não. O seu algoritmo deve ter complexidade de tempo $O(V)$, independentemente de $|E|$.

O algoritmo da questão deve verificar se um grafo G não-direcionado qualquer não possui ciclos. Em teoria de grafos, uma árvore é um grafo conexo sem ciclos e uma floresta é um grafo formado por várias árvores duas a duas desconexas. Dado um grafo qualquer, se ele for uma floresta ele necessariamente não tem ciclos.

Um algoritmo visto em sala é o que realiza uma *busca em profundidade* em um grafo. Dado um grafo $G = (V, E)$ qualquer, ele retorna uma floresta $T = (V, E_T)$, $E_T \subseteq E$. Esta floresta é construída iterativamente, a partir de um vértice $v \in V$ qualquer e um conjunto vazio de arestas. Como uma floresta não pode ter ciclos, quando $e \in E$ é inserido em E_T temos que ter certeza que o grafo induzido pelas arestas $\{E_T \cup e\}$ não possui um ciclo.

A característica de uma aresta gerar um ciclo é definida na busca em largura com a ajuda da cor dada aos vértices nela incidentes. Se uma aresta $e \in E$ analizada em uma iteração do algoritmo incide em dois vértices *cinzas* esta aresta é chamada de aresta "de retorno" e não é inserida em E_T, pois sua inserção geraria um ciclo em T. Estas são as únicas arestas que podem gerar ciclos. Assim, se G contém um ciclo, o algoritmo de busca em profundidade irá encontrar alguma aresta $e \in E \backslash E_T$ que seja de retorno.

O algoritmo proposto em sala consiste em algumas modificações no algoritmo de busca em profundidade apresentado pelo livro. O algoritmo do livro é formado por um procedimento `DFS(G)` que recebe um grafo como parâmetro, inicializa variáveis e chama, (possivelmente mais de uma vez) a rotina recursiva `DFS-VISIT(v)` que é responsável pela construção de uma árvore contendo o vértice v.

A rotina `DFS-Cycle(v)` adiciona à `DFS-VISIT(v)` uma cláusula que verifica se alguma aresta incidente ao vértice v é de retorno. Como v tem cor cinza (assinalada na linha 10), se algum vizinho de v em G e que não seja vizinho de v em T tiver a cor cinza, tem-se um ciclo. Isto é verificado na linha 18, fazendo

com que `DFS-Cycle(v)` retorne *true* (linha 19), o que significa que v faz parte de um ciclo de G.

Se o grafo G possui um ciclo então para algum v a rotina `DFS-Cycle(v)` irá retornar *true*. Este valor será retornado em dois pontos distintos do código:

i. Em uma chamada recursiva da rotina `DFS-Cycle(u)` com $u = \pi(v)$, ou seja, na linha 16. Neste caso, a linha 17 será executada e `DFS-Cycle(u)` também retornará *true*.

ii. Na rotina principal `hasCycle(G)` na linha 7, o que fará com que a linha 8 seja executada e o algoritmo termine com saída *true*. Quando isto ocorre $\pi(v) = NIL$, ou seja, v é a raiz de uma árvore.

É importante notar que em ambos os casos, o algoritmo não irá mais fazer comparações nem chamadas recursivas. O caso *i.* irá ocorrer $d(s, v)$ vezes, sendo $d(s, v)$ a distância do vértice v até a raiz s da árvore da qual ele faz parte. Na iteração seguinte a estas $d(s, v)$ ocorrências, ocorrerá o caso *ii.* e o algoritmo finalmente será finalizado. Assim, se G possui um ciclo o algoritmo `hasCycle(G)` irá retornar *true*.

Complexidade da solução Na busca em profundidade, para cada vértice $v \in V[G]$ a rotina `DFS-VISIT(v)` é chamada uma única vez. Nesta chamada todas as arestas adjacentes a v são checadas uma única vez. Assim, a complexidade do algoritmo `DFS(G)` da ordem de $|V| + \sum_{v \in V} |Adj[v]| = |V| + 2 * |E|$.

Se G não contiver ciclos, então G é uma floresta. Como para uma árvore T_i tem-se a relação $|E_{T_i}| = |V_{T_i}| - 1$ e G é formado por árvores, tem-se que $|E[G]| \leq |V[G]| - 1$ e o custo da `DSF(G)` será da ordem de $|V|$.

Como as modificações realizadas não aumentam a complexidade assintótica do algoritmo inicial, o algoritmo `hasCycle(G)` tem custo assintótico da ordem de $|V|$ no caso de G não possuir ciclos.

Se G possui ciclos então alguma chamada `DFS-Cycle(v)` irá retornar *true*. Neste caso, a recursão será abortada nas $d(s, v)$ chamadas antecedentes à chamada que analizou o nó v. Como um nó é visitado apenas uma vez pelo `DFS-Cycle`, $d(s, v) \leq |V|$. Assim, após encontrar um ciclo o algoritmo irá realizar no máximo $d(s, v)$ operações, com $d(s, v)$ sendo $O(|V|)$.

Como um grafo acíclico com n vértices possui ao máximo $n-1$ arestas, se G for acíclico uma aresta de retorno terá de ser necessariamente encontrada após a análise de $|V[G]| - 1$ arestas. Assim, o custo máximo das chamadas à rotina `hasCycle(G)` antes de encontrar um ciclo será também da ordem de $|V[G]|$.

Desta forma, o algoritmo proposto em sala possui complexidade da ordem $|V|$, independentemente do tamanho de E, como solicitado pelo enunciado da questão.

```
bool  hasCycle(G)
 1: for all v ∈ V[G] do
 2:    color[v] ← white
 3:    π[v] ← NIL
 4: time ← 0
 5: for all v ∈ V[G] do
 6:    if color[v] = white then
 7:       if DFS-Cycle(v) then
 8:          return true
 9: return false

bool  DFS-Cycle(v)
10: color[v] ← gray
11: time ← time + 1
12: d[v] ← time
13: for all u ∈ Adj[v] do
14:    if color[u] = white then
15:       π[u] = v
16:       if DFS-Cycle(u) then
17:          return true
18:    else if (color[u] = gray) and (π[v] ≠ u) then
19:       return true
20: color[v] ← black
21: f[v] ← time ← time + 1
22: return false
```