MO417 – Complexidade de Algoritmos

Nathana Facion - RA 191079 - Lista 7

12/05/2017

Questão 1.

Solução

Seja G um grafo. Seja T sua árvore geradora e seja v o vértice raiz. Para determinar se T é uma possível DFS (Depth first search) de G, usaremos o algoritmo de DFS modificado. Nessa modifcação, adicionamos uma variável de controle chamada naoTemBranco, que será false caso T não for uma árvore DFS de G.

O algoritmo se inicia na raíz v da árvore T, e irá, recursivamente, visitar cada um dos vértices adjacentes do vértice atualmente visitado. Cada vértice visitado é pintado de cinza. Quando não há mais adjacências para esse vértice, então o algoritmo verifica as adjacências dele no grafo em G. Se algum deles for branco, significa que este é uma cross edge, e que T não pode ser uma árvore gerada por DFS em G. Quando isso acontece, a variável de controle naoTemBranco é setado como false, indicando a inviabilidade de T.

Essa variável serve para determinar se em algum ponto da busca em profundidade existe uma adjacência com um vértice de cor branca, ou seja, é uma cross edge, e portanto, T não pode ser uma árvore gerada pela DFS em G.

A complexidade do algoritmo é a mesma vista em sala de aula para o algoritmo DFS, $O(V + E)$, já que o algoritmo é uma DFS alterada.

Algoritmo

Algorithm 1 Depth first search

```
function DFS-VISIT(G, T, v)
    cor[v] ← cinza
    tempo ← tempo + 1
    d[v] ← tempo
    for a até T.Adj[v] do                          ▷ Pega as adjacencias de v em T
        if cor[a] == branco then
            π[a] ← v
            DFS − VISIT(G, T, a)
    for a até G.Adj[v] do                          ▷ Pega as adjacencias de v em G
        if cor[a] == branco then
            naoTemBranco ← False                   ▷ Não é uma árvore enraizada
            return
    cor[v] ← preto
    f[v] ← tempo ← tempo + 1

function DFS(G, T, v)
    for v até G.V do                               ▷ Pega os vertices de G
        cor[v] ← branco
        π[v] ← NULL
    tempo ← 0
    naoTemBranco ← True
    DFS − VISIT(G, T, v)
    return naoTemBranco ▷ True: tem arvore enraizada. False: Não tem.
```

Argumentação do funcionamento

Seja T uma árvore, v um vértice de T. Seja naoTemBranco uma variável global responsável por identificar se T é uma árvore gerada por uma DFS em G, e caso sim, teremos naoTemBranco como true, caso não, false.

O algoritmo visita todos os filhos de v da árvore T . Recursivamente, visita os descendentes destes. Cada vértice visitado é pintado de cinza.

Ao retornar das visitas, verificamos as mesmas adjacências no grafo G. Segundo o teorema "Em uma busca em profundidade sobre um grafo não direcionado G, cada aresta de G ou é aresta da árvore ou é aresta de retorno", encontrar um vértice que é filho de v em G e que esteja branco ainda, significa que esse vértice não é um filho de v em T.

Portanto, ainda segundo o teorema, a aresta que liga v a este vértice em branco é uma cross edge. Então T não é uma árvore gerada pela DFS em G. Quando isso ocorre, a variável naoTemBranco é marcada como false e o algoritmo se encerra.

Podemos garantir que, quando o algoritmo se encerra, se a variável naoTem-

Branco for verdadeiro, então T é uma arvore de DFS em G pois essa variável tem seu valor modificado para false quando uma cross-edge é identificada, e o algoritmo é interrompido.

Suponha que após rodarmos o algoritmo, todos os vértices sejam coloridos de preto. Considere que G tem cross-edge. De acordo com o teorema: "Em uma busca em profundidade sobre um grafo não direcionado G, cada aresta de G ou é aresta da árvore ou é aresta de retorno." Entrtanto, como o algoritmo encerra sua execução ao detectar uma cross-edge, existirá pelo menos um vértice que está branco ao final do algoritmo, o que nos leva a uma contradição.

Questão 2.

Solução

Seja G a matriz de adjacência do grafo. Para que G tenha um ciclo de tamanho 4, então existirão 2 vértices, considere u e v esses vértices, que tenham mais do que 1 vizinho em comum.

Para qualquer dois vértices u e v podemos verificar os seus vizinho em tempo $O(V)$. Por meio da matriz de adjacência de G, podemos comparar a linha de u e a linha de v, e dessa forma conseguir identificar os vizinhos em comum entre eles.

Percorrer toda a matriz de adjacência tem complexidade $O(V^2)$, então com a verificação de vizinhos que vamos fazer em $O(V)$, teremos uma complexidade de execução $O(V)^3$. Segue abaixo o algoritmo proposto:

Observação : Se G não for simples a condição da linha 6 deve ser alterada para diferente de zero, pois nesse caso as adjacências não serão apenas 0 ou 1. Por exemplo: As arestas paralelas podem receber adjacências 2.

Algoritmo

Algorithm 2 Encontrar um ciclo de tamanho 4

```
1: function EncontraCiclo(G, V)                ▷ G é uma matriz de adjacencia
2:     contador ← 0
3:     for i até V do
4:         for j até V do
5:             for k até V do
6:                 if (G[i][k] == 1) then
7:                     if G[i][k] == G[j][k] and (i!=k and j!=k) then
8:                         contador ← contador + 1
9:             if contador ≥ 2 then        ▷ Tem ao menos 2 vizinhos em comum
10:                return True                               ▷ Achou o ciclo
11:            else
12:                contador ← 0
13:    return False
```

Argumentação do funcionamento

O algoritmo usa o método que testa todos os casos para determinar a presença do ciclo. Ele irá verificar todas as adjacências de todos os vértices do grafo em busca de 2 vértices que tenham 2 ou mais vizinhos em comum. Isto é o mínimo necessário para se ter um ciclo de tamanho 4 pois teríamos um ciclo formado pelos 2 vértices i e j (índices no algorito) e pelos seus 2 vizinhos em comum.

Um ciclo de tamanho 4 exige que existam 4 vértices no ciclo, e que 2 desses vértices tenham 2 adjacências em comum. Sejam v_1 e v_3 esses vértices e v_2 e v_4 os adjacências em comum de v_1 e v_3. Então existe um ciclo $v_1 - v_2 - v_3 - v_4 - v_1$

de tamanho 4. É fácil perceber que se existem mais do que 2 vértices em comum, então teremos os 2 vértices em comum necessários e esses serão suficiente para termos o ciclo de tamanho 4.

Mostraremos que isso não é possível se tivermos 1 ou 0 adjacências em comum

Caso 1: Nenhuma adjacência em comum. Suponha, por contradição, que v_1 e v_3 estejam em um mesmo ciclo de tamanho 4, mas que não tenham 2 adjacências em comum.

Então v_1 tem 2 adjacências, v_2 e v_4, e v_3 terá 2 adjacências diferentes, v_5 e v_6. Então teremos que o ciclo entre v_1 e v_3 passará também por v_2, v_4, v_5 e v_6, uma contradição, pois o ciclo será maior do que 4.

Isso só seria possível se v_2 e v_4 fossem os mesmos vértices v_5 e v_6, o que implica que v_1 e v_3 têm 2 adjacências em comum .

Caso 2: Apenas uma adjacência em comum. Suponha, por contradição, que v_1 e v_3 estejam em um mesmo ciclo de tamanho 4, mas que tenham 1 adjacências em comum.

Então v_1 e v_3 tem 1 adjacência em comum, seja v_2 essa adjacência. Então v_1 tem a adjacência v_4 e v_3 tem a adjacência v_5. Com isso o ciclo passará por todos esses vértice e voltará a v_1, totalizando ao menos 5 vértices. Isso é impossível em um ciclo de tamanho 4.

Conclusão: Concluimos então que caso o número de adjacências fosse menor do que 2, então teríamos 1 ou 0 adjacências. Nesse caso, podemos concluir apenas que não existe um ciclo de tamanho 4, portanto retornando Falso.

Questão 3.

Seja G um grafo direcionado. Seja E seu conjunto de arestas. Seja $(G)^T$ o grafo transposto, onde $E^T = \{\ (u, v) : (v, u) \in E\}$ .Vamos demonstrar que $((G^T)^{SCC})^T = G^{SCC}$.

Considere que G tenha componentes fortemente conectado $C_1, C_2, ..C_k$. O conjunto de vértices V^{SCC} é { $v_1, v_2..v_k\}$ e contém um vértice v_i para cada componente fortemente conectado C_i de G. Existe uma aresta$(v_i, v_j) \in E^{SCC}$ se G contém uma aresta orientada (x,y) para algum x $\in C_i$ e algum y $\in C_j$. Podemos visualizar de outro modo, pela contração de todas as arestas cujos vértices incidentes estão dentro do mesmo componente fortemente conexo de G, o grafo resultante é G^{SCC}.

Como o relacionamento fortemente conectado é uma relação de equivalência, G e G^T sempre terão as mesmas componentes conectados. Seja u_1 e u_2 vértices do grafo G. Se dois vértices u_1 e u_2 não são fortemente conectados, então existe um caminho único entre u_1 e u_2 em G se e somente se existe um caminho único de u_2 a u_1 em G^T.

Assim, os conjuntos de vértices de G^{SCC} e de $(G^T)^{SCC}$ são os mesmos, o que implica que o conjunto de vértices de $((G^T)^{SCC})^T$ e G^{SCC} são os mesmos.

É suficiente mostrarmos que seus conjuntos de arestas são iguais. Suponha (v_i, v_j) é uma aresta em $((G^T)^{SCC})^T$. Portanto, para algum $x \in C_j$ e $y \in C_i$ existe uma aresta (x, y) que é uma aresta em $(G^T)^{SCC}$, implicando que (y, x) é uma aresta de G^{SCC}. Como os componentes são preservados, isso significa que (v_j, v_i) é aresta em $(G^T)^{SCC})$, e (v_i, v_j) é uma aresta em G^{SCC}.

Para concluir, podemos notar que se (v,u) é uma aresta de E, então $E^T = \{\ (u, v) : (v, u) \in E\}$, $(E^T)^T = \{\ (v, u) : (u, v) \in E^T\}$, então $(E^T)^T = E$. O que implica que $((G^T)^{SCC})^T = G^{SCC}$.

Questão 4.

Seja G o grafo. Seja v o primeiro vértice que começou espalhar a informação.
A coloração deve ser a seguinte:
- Se não recebeu mensagem ou chave: branco
- Se recebeu chave: cinza
- Se recebeu mensagem: azul
- Se recebeu as duas: preto

1. Faça uma busca em largura, com inicio v.

2. O descobridor vai mandar para n matemáticos a mensagem e a chave para m físicos. As pessoas que receberem mensagem mudam de cor para azul. As pessoas que receberem a chave para cinza.

3. Quando a pessoa receber as mensagem e chave, muda a cor para preto e adiciona um no contador. Esse vértice não recebe e nem envia mais mensagem ou chave.

4. Continua o algoritmo da BFS normalmente.

5. O contador terá ao final o número com todas pessoas que sabem sobre a descoberta, uma mesma pessoa não é adicionada mais de uma vez.