Análise da complexidade temporal de DFS

O ciclo das linhas 1–3 [DFS] é executado $|V|$ vezes

DFS-VISIT é chamada para cada um dos $|V|$ vértices

Para cada vértice u (e considerando a implementação através de listas de adjacências), o ciclo das linhas 4–7 [DFS-VISIT] é executado

$$|\,\texttt{G.adj}[u]\,| \quad \text{vezes}$$

Tendo todas as operações custo constante, considerando todas as chamadas a DFS-VISIT, DFS corre em tempo

$$O(V + \sum_{u \in V} |\,\texttt{G.adj}[u]\,|) = O(V + E)$$

Conectividade (1)

Seja $G = (V, E)$ um grafo não orientado

G é conexo se existe algum caminho entre quaisquer dois nós

$V' \subseteq V$ é uma componente conexa de G se

- existe algum caminho entre quaisquer dois nós de V' e
- não existe qualquer caminho entre um nó de V' e um nó de $V \setminus V'$

Conectividade (2)

Seja $G = (V, E)$ um grafo orientado

G é fortemente conexo se existe algum caminho de qualquer nó para qualquer outro nó

$V' \subseteq V$ é uma componente fortemente conexa de G se

- existe algum caminho de qualquer nó de V' para qualquer outro nó de V' e
- se, qualquer que seja o nó $u \in V \setminus V'$
  - não existe qualquer caminho de um nó de V' para u ou
  - não existe qualquer caminho de u para um nó de V'

Grafo transposto

O grafo transposto do grafo orientado $G = (V, E)$ é o grafo

$$G^T = (V, E^T)$$

tal que

$$E^T = \{(v, u) \mid (u, v) \in E\}$$

Componentes fortemente conexas

Strongly connected components

G – grafo orientado

SCC(G)

1. Aplicar DFS(G) para calcular o instante `u.f` em que termina o processamento de cada vértice `u`
2. Calcular G^T
3. Aplicar DFS(G^T), processando os vértices por ordem decrescente de `u.f` (calculado em 1), no ciclo principal de DFS (linha 5)
4. Devolver os vértices de cada árvore da floresta da pesquisa em profundidade (construída em 3) como uma componente fortemente conexa distinta

Ordenação topológica

Seja $G = (V, E)$ um grafo orientado acíclico (DAG, de *directed acyclic graph*)

Ordem topológica

Se existe um arco de u para v, u está antes de v na ordem dos vértices

$$(u, v) \in E \;\Rightarrow\; u < v$$

TOPOLOGICAL-SORT(G)

1. Aplicar DFS(G)
2. Inserir cada vértice à cabeça de uma lista, quando termina o seu processamento
3. Devolver a lista, que contém os vértices por (alguma) ordem topológica

Ordenação topológica

Adaptação de DFS

G – grafo orientado acíclico (DAG)

TOPOLOGICAL-SORT(G)

```
for each vertex u in G.V do
    u.color <- WHITE
L <- EMPTY                              // lista, global
for each vertex u in G.V do
    if u.color = WHITE then
        DFS-VISIT'(G, u)
return L
```

DFS-VISIT'(G, u)

```
u.color <- GREY
for each vertex v in G.adj[u] do
    if v.color = WHITE then
        DFS-VISIT'(G, v)
u.color <- BLACK
LIST-INSERT-FIRST(L, u)
```

Ordenação topológica

Outro algoritmo

TOPOLOGICAL-SORT'(G)

```
for each vertex u in G.V do
    u.i <- 0
for each edge (u,v) in G.E do
    v.i <- v.i + 1               // arcos com destino v
L <- EMPTY                       // lista
S <- EMPTY                       // conjunto
for each vertex u in G.V do
    if u.i = 0 then
        SET-INSERT(S, u)
while S != EMPTY do
    u <- SET-DELETE(S)           // retira um nó de S
    for each vertex v in G.adj[u] do
        v.i <- v.i - 1
        if v.i = 0 then
            SET-INSERT(S, v)
    LIST-INSERT-LAST(L, u)
return L
```

Árvore de cobertura mínima

Minimum(-weight) spanning tree

Seja $G = (V, E)$ um grafo pesado não orientado conexo

Uma árvore de cobertura de G é um subgrafo $G' = (V, E')$ de G, com $E' \subseteq E$

- conexo e
- acíclico (é uma árvore)

(Retirando qualquer arco de G', obtém-se um grafo não conexo)

Uma árvore de cobertura mínima de G é uma árvore de cobertura G' de peso mínimo:

> Se $w(G')$ for a soma dos pesos dos arcos de G', para qualquer árvore de cobertura G'' de G tem-se

$$w(G') \leq w(G'')$$