

# Algoritmos e Estruturas de Dados II

## Grafos V: Percursos em Digrafos e Ordenação Topológica

Ricardo J. G. B. Campello

Parte deste material é baseado em adaptações e extensões de slides disponíveis em <http://ww3.datastructures.net> (Goodrich & Tamassia).

1

## Organização

- ◆ Revisão (DFS)
- ◆ Percursos em Digrafos
  - Exemplo de Execução (DFS)
  - Propriedades e Aplicações
- ◆ Digrafos Acíclicos (DAGs)
- ◆ Ordenação Topológica

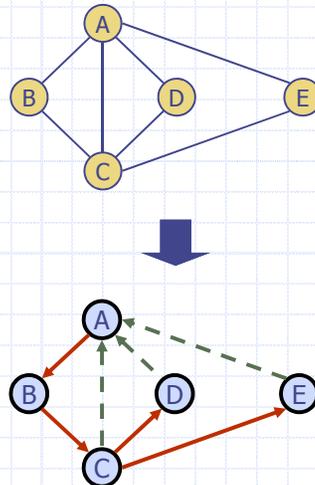
Algoritmos & Estruturas de Dados II

2

## Revisão (Algoritmo DFS)

```
Algoritmo DFS(G, v)
  v.label ← DESCOBERTO
  process_vertex(v)
  para todo e ∈ incidentEdges(G, v)
    y ← opposite(G, v, e)
    se y.label = NÃO-DESCOBERTO
      y.parent ← v
      DFS(G, y)
    senão
      se ¬ (y.label = EXPLORADO)
        process_edge(e)
  v.label ← EXPLORADO
```

Assume-se que inicialmente os vértices de *G* são rotulados como “não descobertos”.



3

## Percursos em Digrafos

- ◆ DFS e BFS podem ser adaptados para digrafos.
- ◆ Essas adaptações são mais genéricas:
  - qualquer grafo não-direcionado pode ser transformado em um grafo direcionado.
- ◆ **Mudança:**
  - percurso só é feito no sentido correto das arestas, ou seja, apenas através das arestas de saída.
    - ◆ logo, não é preciso se preocupar em não processar duas vezes a mesma aresta
      - arestas podem ser processadas no vértice de saída

Algoritmos & Estruturas de Dados II

4

Algoritmos & Estruturas de Dados II

## DFS em Digrafos

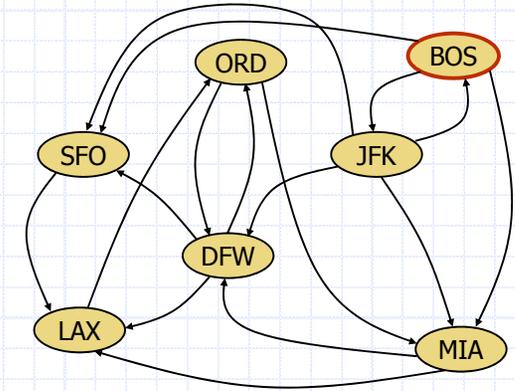
### ◆ DFS direcionado:

- as arestas que não são de descoberta não são mais necessariamente de retorno, pois não mais necessariamente apresentam a propriedade de conectar um vértice a um ancestral na árvore DFS.
- De fato, as arestas podem também ser de:
  - ◆ **avanço**: conectam um vértice a um descendente na árvore DFS
  - ◆ **cruzamento**: conectam um vértice a outro que não é nem descendente nem ancestral na árvore DFS

5

## DFS em Digrafos

### ◆ Exemplo:

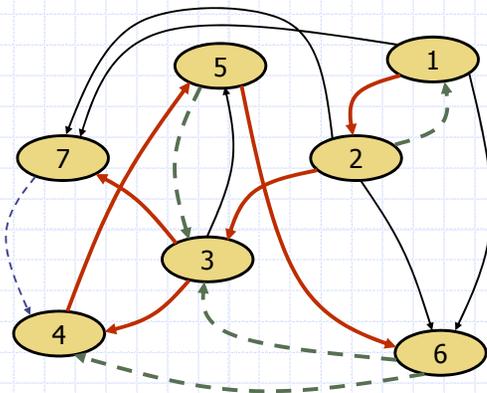


6

## DFS em Digrafos

### ◆ Arestas:

- De descoberta
- De retorno
- De avanço
- De cruzamento



7

## Percursos em Digrafos

### ◆ As adaptações de DFS e BFS para digrafos permitem:

- Obter, para cada vértice de  $G$ , o **subgrafo alcançável** a partir daquele vértice.
- Calcular os **componentes fortemente conexos** de  $G$  e testar se  $G$ , como um todo, é fortemente conexo.
- Encontrar um **ciclo direcionado** em  $G$ .
- Obter um **caminho com o menor número de arestas** entre dois vértices (BFS).
- ...

8

## Percursos em Digrafos

- ◆ **Propriedade 1:** DFS ou BFS em um digrafo  $G$  partindo de um vértice  $s$  explora todos os vértices e arestas alcançáveis a partir de  $s$ .
  - Exercício: Justificar a Propriedade 1
- ◆ **Propriedade 2:** As arestas de descoberta DFS ou BFS formam uma árvore com caminhos direcionados de  $s$  para cada um dos vértices alcançáveis a partir de  $s$ .
  - Exercício: Justificar a Propriedade 2

9

## Percursos em Digrafos

- ◆ **Análise:**
  - **Tempo:** Se  $G$  for implementado com uma lista de adjacências ou estrutura alternativa, então DFS (BFS) roda em tempo  $O(n_s + m_s)$ , onde  $n_s$  e  $m_s$  são respectivamente os números de vértices e arestas alcançáveis a partir de  $s$ .
    - ♦ Cada vértice e aresta alcançáveis são explorados uma única vez
      - no caso de aresta, a partir da sua origem
  - **Espaço:**
    - ♦ DFS utiliza  $O(n_s)$  espaço auxiliar com a pilha de recursão devido à  $n_s$  chamadas recursivas com espaço constante em cada uma delas.
    - ♦ BFS não possui recursão, mas utiliza espaço auxiliar  $O(n_s)$  para armazenar a fila de vértices  $Q$ .

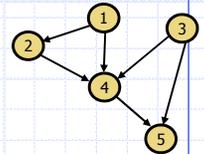
10

## Percursos em Digrafos

- ◆ **Teste de Conexão Forte:** Podemos executar DFS ou BFS múltiplas vezes e verificar se o grafo é fortemente conexo verificando se a partir de cada vértice tomado como origem todos os demais vértices são alcançáveis ou não:
  - Tempo  $O(n(n+m))$  no pior caso.
    - ♦ Qual o pior caso?
  - Nota: É possível executar um teste de conexão forte em tempo  $O(n+m)$  com apenas duas execuções de DFS ou BFS, uma sobre o digrafo original  $G$  e a outra sobre o seu transposto  $G^T$ :
    - ♦ Desafio: Descubra o porquê sem checar a literatura!!!

11

## Digrafos Acíclicos



- ◆ **Grafos Direcionados Acíclicos (DAGs):** Como sugere o nome, são digrafos que não possuem ciclos. Exemplos:
  - Hierarquia de heranças entre classes em orientação a objetos
  - Pré-requisitos entre disciplinas de um curso
  - Restrições de cronograma entre tarefas de um projeto
- ◆ **Ordenação Topológica:** Trata-se de uma ordenação dos vértices  $v_1, \dots, v_n$  de um DAG  $G$  tal que para qualquer aresta direcionada  $(v_i, v_j)$  tem-se  $i < j$ .
  - Caminhos direcionados percorrem os vértices em ordem crescente.
    - ♦ Qualquer caminho entre  $v_i$  e  $v_j$  não passa por  $v_k$  tal que  $k < i$  ou  $k > j$ .
  - Vide exemplo simples acima (no canto superior direito)

12

# Ordenação Topológica

## Algoritmo mais popular utiliza as seguintes Propriedades de DAGs:

- Necessariamente possuem ao menos um vértice sem arestas incidentes de entrada (apenas arestas de saída ou nenhuma).
    - Se todo vértice possui ao menos uma aresta de entrada, necessariamente existe ao menos um ciclo.
  - Se tais vértices e as suas arestas de saída forem removidas, o grafo restante também é um DAG.
- ◆ **Idéia do Algoritmo:** Remover sucessivamente aqueles vértices sem arestas incidentes de entrada, rotulando os mesmos em ordem crescente de remoção e removendo também as respectivas arestas de saída.

13

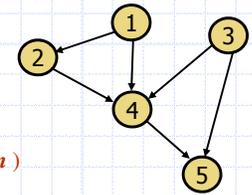
# Ordenação Topológica

## Algoritmo *TopologicalSort(G)*

```

S ← Pilha Vazia
para todo u ∈ vertices(G)
  se u.inDegree = 0
    push(S, u)
t ← 1
enquanto ¬ empty(S)
  u ← pop(S)
  u.topsort ← t
  t ← t + 1
  para todo e ∈ outgoingEdges(G, u)
    v ← opposite(G, u, e)
    v.inDegree ← v.inDegree - 1
    se v.inDegree = 0
      push(S, v)
  
```

- Note que os vértices não precisam ser de fato removidos do grafo.
- É suficiente modificar artificialmente uma contagem de arestas de entrada de cada vértice.
- Exemplo:



- Espaço e tempo de execução de pior caso:  $O(n + m)$
- Detecta Existência de Ciclos:
  - $G$  não é DAG se um ou mais vértices não for removido / rotulado.

# Ordenação Topológica

```

compute_indegreess(graph *g, int in[]) {
  int i, j;
  for (i=1; i<=g->nvertices; i++) in[i] = 0;
  for (i=1; i<=g->nvertices; i++)
    for (j=0; j<g->degree[i]; j++)
      in[g->edges[i][j]]++;
}
  
```

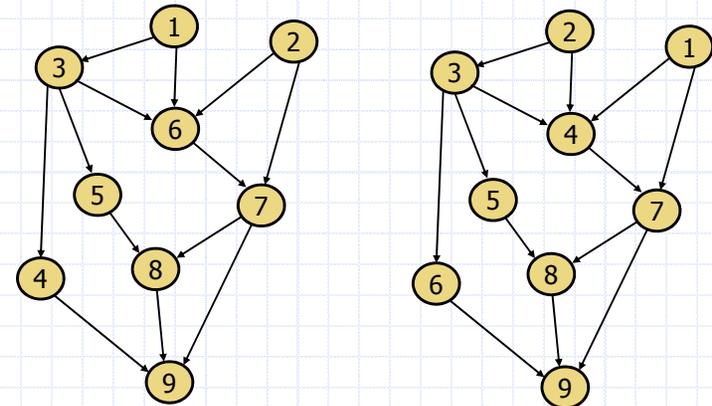
```

topsort(graph *g, int sorted[]) {
  int indegree[MAXV]; /* indegree of each vertex
  queue zeroIn;      /* vertices of indegree 0
  int x, v;          /* current and next vertex
  int i, j;          /* counters */
  compute_indegreess(g, indegree);
  init_queue(&zeroIn);
  for (i=1; i<=g->nvertices; i++)
    if (indegree[i] == 0) enqueue(&zeroIn, i);
  j=0;
  while (empty(&zeroIn) == FALSE) {
    j = j+1;
    x = dequeue(&zeroIn);
    sorted[j] = x;
    for (i=0; i<g->degree[x]; i++) {
      v = g->edges[x][i];
      indegree[v]--;
      if (indegree[v] == 0) enqueue(&zeroIn, v);
    }
  }
  if (j != g->nvertices)
    printf("Not a DAG -- only %d vertices found\n", j);
}
  
```

- Note o uso de uma fila ao invés de pilha.
- De fato, qualquer ED pode ser utilizada.
- Diferentes EDs podem produzir ordens topológicos distintos.
- Ordenação topológica não é única!

# Ordenação Topológica

## Exemplos:



## Exercícios

1. Modifique o pseudo-código DFS de grafos não direcionados para que este seja válido para grafos direcionados. Para tanto, faça as modificações necessárias ao TAD grafo apresentado em aula.
  - Dica: Note que não mais é necessário se preocupar em não processar cada aresta mais de uma vez, apenas no vértice de saída.
2. Modifique a implementação C do algoritmo `dfs` vista em aula para que esta seja válida para grafos direcionados.
3. É necessário mudar algo na especialização do algoritmo DFS para busca de ciclos vista em aula se o grafo for direcionado? Explique.
  - Nota: Observe que o *princípio* do uso de DFS para busca de ciclos não muda, ou seja, arestas de *retorno* continuam caracterizando ciclos (agora direcionados) e continuam sendo caracterizadas por levarem até um vértice já *descoberto* mas ainda não totalmente *explorado*.
4. Repita os Exercícios 1 e 2 para busca em largura (BFS).

17

## Exercícios

5. Descreva com suas palavras uma forma de calcular o fechamento transitivo de um digrafo  $G$  usando BFS ou DFS. Assumindo que o digrafo possui  $m$  arestas,  $n$  vértices e é fortemente conexo, qual a complexidade do seu método em termos de tempo de execução?
  - Dica: Tome como base a Propriedade 1 de percursos em digrafos
6. Bob pretende fazer um conjunto de disciplinas de especialização. Ele está interessado nos seguintes cursos: LA15, LA16, LA22, LA31, LA32, LA126, LA127, LA141 e LA169. Dados os pré-requisitos desses cursos abaixo, mostre como usar ordenação topológica para encontrar uma seqüência de cursos que permita satisfazer todos os pré-requisitos:
  - **LA15 e LA22:** nenhum
  - **LA16 e LA31:** LA15
  - **LA32:** LA16 e LA31
  - **LA126:** LA22 e LA32
  - **LA127:** LA16
  - **LA141:** LA22 e LA16
  - **LA169:** LA32

{ Mostre uma tal seqüência e responda justificadamente se ela é única ou não!

18

## Referências

- ◆ M. T. Goodrich and R. Tamassia, *Data Structures and Algorithms in C++/Java*, John Wiley & Sons, 2002/2005.
- ◆ N. Ziviani, *Projeto de Algoritmos*, Thomson, 2a. Edição, 2004.
- ◆ T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, 2<sup>nd</sup> Edition, 2001.
- ◆ S. Skiena e M. Revilla, *Programming Challenges: The Programming Contest Training Manual*, Springer-Verlag, 2003.

22