Algoritmos em Grafos: Pesquisa e Ordenação

R. Rossetti, A. P. Rocha, L. Ferreira, J. P. Fernandes, F. Ramos, G. Leão
CAL, MIEIC, FEUP

Índice

- Pesquisa em profundidade
- Pesquisa em largura
- Ordenação topológica

3

Pesquisa em profundidade *(depth-first search)*

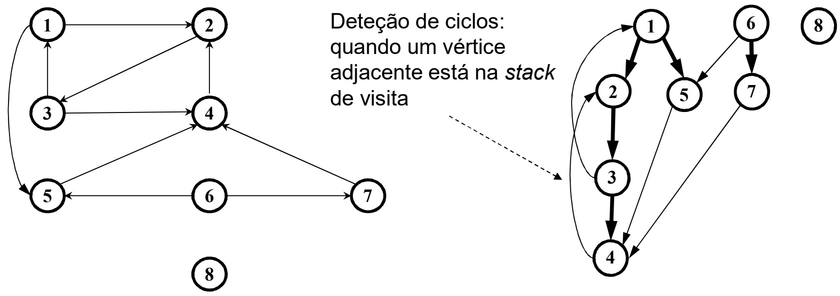
4

Pesquisa em profundidade

- Arestas são exploradas a partir do vértice *v* mais recentemente descoberto que ainda tenha arestas a sair dele.
- Quando todas as arestas de *v* forem exploradas, retorna para explorar arestas que saíram do vértice a partir do qual *v* foi descoberto.
- Se se mantiverem vértices por descobrir, um deles é seleccionado como a nova fonte e o processo de pesquisa continua a partir daí.
- Todo o processo é repetido até todos os vértices serem descobertos.

Exemplo

Vértices numerados por (pré)ordem de visita e dispostos por profundidade de recursão:



Arestas a traço forte (que acederam a vértices ainda não visitados): floresta DFS constituída por uma ou mais árvores DFS (de expansão em profundidade).

Pseudo-código

```
G = (V, E)
Adj(v) = {w | (v, w) ∈ E} (∀ v ∈ V)

DFS(G):
for each v ∈ V
   visited(v) ← false
for each v ∈ V
   if not visited(v)
      DFS-VISIT(G, v)

DFS-VISIT(G, v):
visited(v) ← true
pre-process(v)
for each w ∈ Adj(v)
   if not visited(w)
      DFS-VISIT(G, w)
post-process(v)
```

Numa aplicação concreta de DFS, há processamento a fazer num destes pontos

Pesquisa em largura *(breadth-first search)*

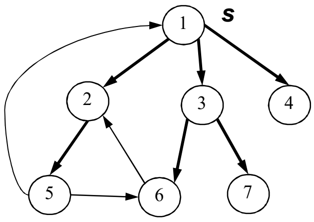
Ideia base

- Dado um vértice fonte *s*, explora-se sistematicamente o grafo descobrindo todos os vértices a que se pode chegar a partir de *s* (vértices adjacentes).
- Só depois é que se passa para outro vértice.

9

Exemplo

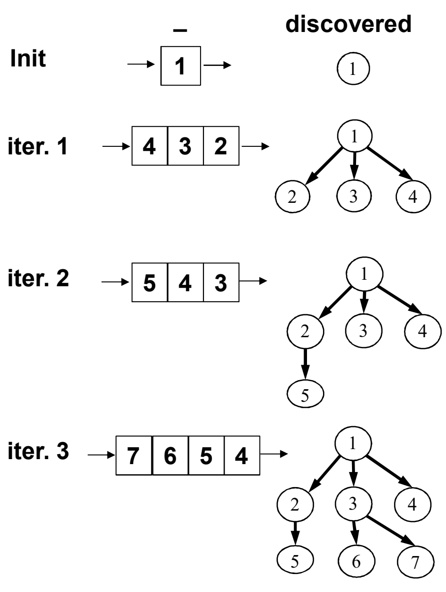
Exemplo em grafo dirigido, com vértices numerados por (pré)ordem de visita e dispostos por níveis de distância ao vértice inicial (*s*).

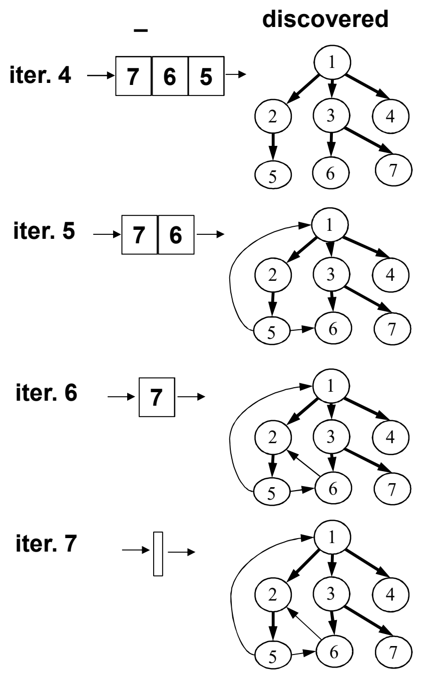


Arestas a traço forte formam uma árvore de expansão em largura (árvore BFS), com raiz *s*.

10

Passo a passo





Pseudo-código

```
BFS(G, s):
    for each v ∈ V do discovered(v) ← false
    Q ← ∅

    ENQUEUE(Q, s)
    discovered(s) ← true

    while Q ≠ ∅ do
        v ← DEQUEUE(Q)
        pre-process(v)
        for each w ∈ Adj(v) do
            if not discovered(w) then
                ENQUEUE(Q, w)
                discovered(w) ← true
        post-process(v)
```

Numa aplicação concreta de BFS, há processamento a fazer num destes pontos

Notas

- Para qualquer vértice *v* atingível a partir de s, o caminho na árvore BFS é o caminho mais curto no grafo (com menor número de arestas).

- BFS é um dos métodos mais simples e é o arquétipo para muitos algoritmos importantes de grafos.
  - Prim's Minimum-Spanning Tree
  - Dijkstra Single-Source Shortest-paths

- Se em vez de uma fila for usada uma pilha, obtém-se um algoritmo iterativo de visita em profundidade!

13

Ordenação topológica

Algoritmos em Grafos: Introdução • CAL - MIEIC/FEUP

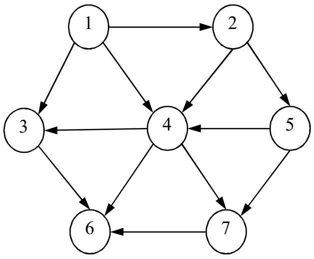
14

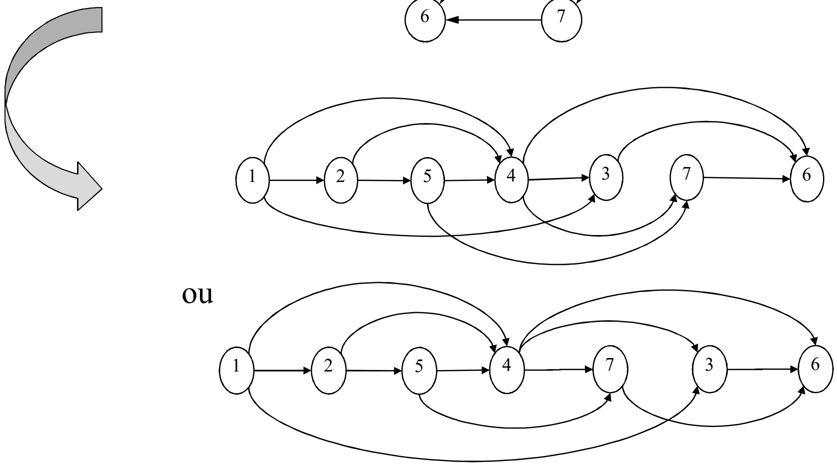
Problema

- Ordenar os vértices de um DAG (grafo acíclico dirigido) tal que, se existe uma aresta *(v, w)* no grafo, então *v* aparece antes de *w*
  - Intuitivamente, dispor as setas todas no mesmo sentido
  - Impossível se o grafo for cíclico
  - Pode existir mais do que uma ordenação (solução)

Algoritmos em Grafos: Introdução • CAL - MIEIC/FEUP

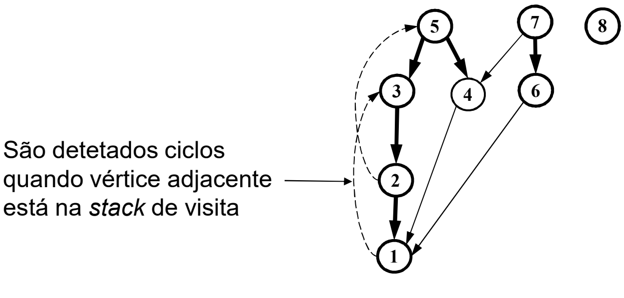
Exemplo





Método baseado em DFS

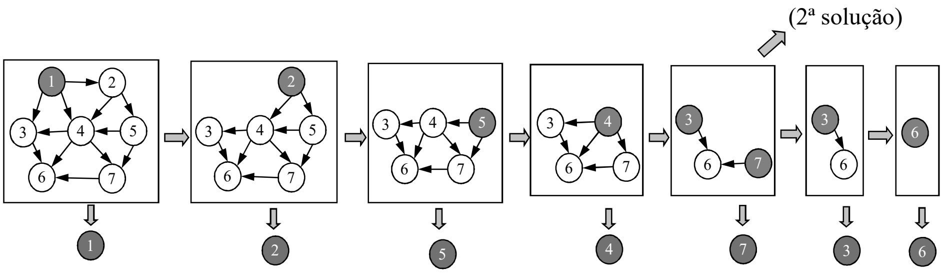
- Na visita em profundidade (DFS) de um DAG, a pós-ordem de visita dá uma ordenação topológica inversa



- Mas o método não é genérico, pois algumas ordenações topológicas válidas não podem ser obtidas desta forma
  - e.g., vértice 4 nunca fica entre vértices 3 e 2

Método geral

1. Descobrir um vértice sem arestas de chegada (*indegree=0*)
2. Imprimir o vértice
3. Eliminá-lo, assim como as arestas que dele saem
4. Repetir o processo no grafo restante



Algoritmo de ordenação topológica

- Refinamento do método geral:
  - simular eliminação atualizando indegree dos vértices adjacentes
  - memorizar numa estrutura auxiliar vértices por imprimir c/ indegree=0
- Dados de entrada:
  - V - conjunto de vértices
  - adj(v) - conjunto (ou lista) de vértices adjacentes a cada vértice v
    - ou conj. de arestas que saem de v, que por sua vez indicam vértices adj.
- Dados de saída:
  - T – sequência (ou lista) dos vértices por ordem topológica
    - ou numTop(v) – número atribuído a cada vértice v por ordem topológica
- Dados temporários:
  - indegree(v) - nº de arestas que chegam a v, partindo de vértices por visitar
  - C - conjunto de vértices por visitar cujo indegree é 0 (candidatos)

Algoritmo de ordenação topológica

```
TOP-SORT( in G=(V,E),  out  T):
1.  for each v∈V do indegree(v) ← 0
2.  for each v∈V do for each w∈adj(v) do indegree(w) ← indegree(w) + 1
3.  C ← { } // Pode ser uma fila (Queue), pilha (Stack), etc.
4.  for each v∈V do if indegree(v) = 0 then C ← C ∪ {v}
5.  T ← [ ] // Pode ser uma lista  (LinkedList)
6.  while C ≠ {} do
7.      v ← remove-one(C)
8.      T ← T concatenado-com [v]
9.      for each w ∈ adj(v) do
10.         indegree(w) ← indegree(w) - 1
11.         if indegree(w) = 0 then C ← C ∪ {w}
12. if |T| ≠ |V| then Fail("O grafo tem ciclos")
```


Análise do algoritmo

- As diferentes escolhas do próximo vértice no ponto 7 dão as diferentes soluções possíveis

- Se as inserções e eliminações em C forem efectuadas em tempo constante (usando por exemplo uma fila FIFO), o algoritmo pode ser executado em tempo O(|V|+|E|)
  - o corpo do ciclo de atualização do indegree (passos 9, 10, 11) é executado no máximo uma vez por aresta
  - as operações de inserção e remoção na fila (nos passos 4, 7 e 11) são executadas no máximo uma vez por vértice
  - a inicialização leva um tempo proporcional ao tamanho do grafo

Aplicações

- Grafos Acíclicos Dirigidos (DAG) são usados em aplicações onde é necessário indicar a precedência entre eventos.
- Exemplo: escalonamento de sequências de tarefas.
- A ordenação topológica de um DAG dá-nos uma ordem (sequência) dos eventos (tarefas, trabalhos, etc.) representados nesse DAG.
- No caso de existirem múltiplas soluções, podem-se usar critérios adicionais

Referências e mais informação

- T.H. Cormen, C. E. Leiserson, R. L. Rivest , C. Stein. Introduction to Algorithms, 3rd Edition. MIT Press, 2009
  - Capítulo 22
- "Data Structures and Algorithm Analysis in Java", Second Edition, Mark Allen Weiss, Addison Wesley, 2006