

Computer Science & Engineering 423/823  
Design and Analysis of Algorithms  
Lecture 02 — Elementary Graph Algorithms (Chapter 22)

Stephen Scott  
(Adapted from Vinodchandran N. Variyam)

sscott@cse.unl.edu

## Introduction

- ▶ Graphs are abstract data types that are applicable to numerous problems
  - ▶ Can capture *entities*, *relationships* between them, the *degree* of the relationship, etc.
- ▶ This chapter covers basics in graph theory, including representation, and algorithms for basic graph-theoretic problems (some content was covered in review lecture)
- ▶ We'll build on these later this semester

## Breadth-First Search (BFS)

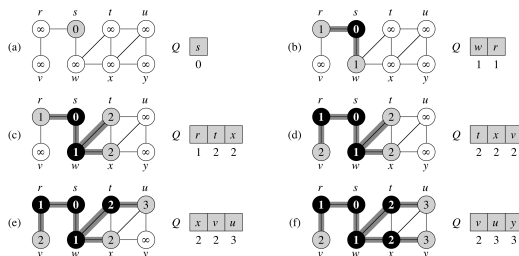
- ▶ Given a graph  $G = (V, E)$  (directed or undirected) and a *source* node  $s \in V$ , BFS systematically visits every vertex that is reachable from  $s$
- ▶ Uses a queue data structure to search in a breadth-first manner
- ▶ Creates a structure called a **BFS tree** such that for each vertex  $v \in V$ , the distance (number of edges) from  $s$  to  $v$  in tree is a shortest path in  $G$
- ▶ Initialize each node's **color** to WHITE
- ▶ As a node is visited, color it to GRAY ( $\Rightarrow$  in queue), then BLACK ( $\Rightarrow$  finished)

## BFS( $G, s$ )

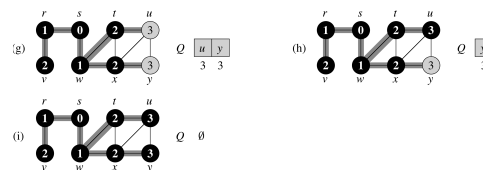
```

1 for each vertex  $u \in V \setminus \{s\}$  do
2    $color[u] = \text{WHITE}$ 
3    $d[u] = \infty$ 
4    $\pi[u] = \text{NIL}$ 
5 end
6  $color[s] = \text{GRAY}$ 
7  $d[s] = 0$ 
8  $\pi[s] = \text{NIL}$ 
9  $Q = \emptyset$ 
10 ENQUEUE( $Q, s$ )
11 while  $Q \neq \emptyset$  do
12    $u = \text{DEQUEUE}(Q)$ 
13   for each  $v \in \text{Adj}[u]$  do
14     if  $color[v] = \text{WHITE}$  then
15        $color[v] = \text{GRAY}$ 
16        $d[v] = d[u] + 1$ 
17        $\pi[v] = u$ 
18       ENQUEUE( $Q, v$ )
19   end
20    $color[u] = \text{BLACK}$ 
21 end
    
```

## BFS Example



## BFS Example (2)



## BFS Properties

- ▶ What is the running time?
  - ▶ Hint: How many times will a node be enqueued?
- ▶ After the end of the algorithm,  $d[v]$  = shortest distance from  $s$  to  $v$ 
  - ⇒ Solves unweighted shortest paths
  - ▶ Can print the path from  $s$  to  $v$  by recursively following  $\pi[v]$ ,  $\pi[\pi[v]]$ , etc.
- ▶ If  $d[v] = \infty$ , then  $v$  not reachable from  $s$ 
  - ⇒ Solves reachability

◀ ▶ 🔍 ↺ ↻ ⌂

## Depth-First Search (DFS)

- ▶ Another graph traversal algorithm
- ▶ Unlike BFS, this one follows a path as deep as possible before backtracking
- ▶ Where BFS is “queue-like,” DFS is “stack-like”
- ▶ Tracks both “discovery time” and “finishing time” of each node, which will come in handy later

◀ ▶ 🔍 ↺ ↻ ⌂

## DFS( $G$ )

```

1 for each vertex  $u \in V$  do
2    $color[u] = \text{WHITE}$ 
3    $\pi[u] = \text{NIL}$ 
4 end
5  $time = 0$ 
6 for each vertex  $u \in V$  do
7   if  $color[u] == \text{WHITE}$  then
8     DFS-VISIT( $u$ )
9 end
```

◀ ▶ 🔍 ↺ ↻ ⌂

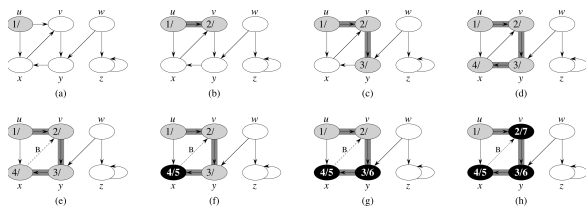
## DFS-VISIT( $u$ )

```

1  $color[u] = \text{GRAY}$ 
2  $time = time + 1$ 
3  $d[u] = time$ 
4 for each  $v \in Adj[u]$  do
5   if  $color[v] == \text{WHITE}$  then
6      $\pi[v] = u$ 
7     DFS-VISIT( $v$ )
8 end
9  $color[u] = \text{BLACK}$ 
10  $f[u] = time = time + 1$ 
```

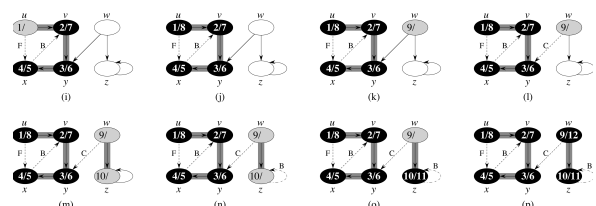
◀ ▶ 🔍 ↺ ↻ ⌂

## DFS Example



◀ ▶ 🔍 ↺ ↻ ⌂

## DFS Example (2)



◀ ▶ 🔍 ↺ ↻ ⌂

## DFS Properties

- ▶ Time complexity same as BFS:  $\Theta(|V| + |E|)$
- ▶ Vertex  $u$  is a proper descendant of vertex  $v$  in the DF tree iff  $d[v] < d[u] < f[u] < f[v]$ 
  - ⇒ **Parenthesis structure:** If one prints " $u$ " when discovering  $u$  and " $u$ " when finishing  $u$ , then printed text will be a well-formed parenthesized sentence

◀ ▶ ↻ 🔍

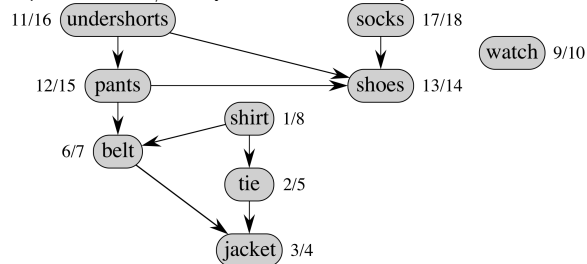
## DFS Properties (2)

- ▶ Classification of edges into groups
  - ▶ A **tree edge** is one in the depth-first forest
  - ▶ A **back edge**  $(u, v)$  connects a vertex  $u$  to its ancestor  $v$  in the DF tree (includes self-loops)
  - ▶ A **forward edge** is a nontree edge connecting a node to one of its DF tree descendants
  - ▶ A **cross edge** goes between non-ancestral nodes within a DF tree or between DF trees
  - ▶ See labels in DFS example
- ▶ Example use of this property: A graph has a cycle iff DFS discovers a back edge (application: deadlock detection)
- ▶ When DFS first explores an edge  $(u, v)$ , look at  $v$ 's color:
  - ▶  $color[v] == \text{WHITE}$  implies tree edge
  - ▶  $color[v] == \text{GRAY}$  implies back edge
  - ▶  $color[v] == \text{BLACK}$  implies forward or cross edge

◀ ▶ ↻ 🔍

## Application: Topological Sort

A directed acyclic graph (dag) can represent precedences: an edge  $(x, y)$  implies that event/activity  $x$  must occur before  $y$



◀ ▶ ↻ 🔍

## Application: Topological Sort (2)

A **topological sort** of a dag  $G$  is a linear ordering of its vertices such that if  $G$  contains an edge  $(u, v)$ , then  $u$  appears before  $v$  in the ordering



◀ ▶ ↻ 🔍

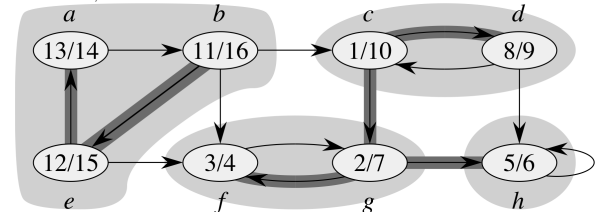
## Topological Sort Algorithm

1. Call DFS algorithm on dag  $G$
  2. As each vertex is finished, insert it to the front of a linked list
  3. Return the linked list of vertices
- ▶ Thus topological sort is a descending sort of vertices based on DFS finishing times
  - ▶ What is the time complexity?
  - ▶ Why does it work?
    - ▶ When a node is finished, it has no unexplored outgoing edges; i.e., all its descendant nodes are already finished and inserted at later spot in final sort

◀ ▶ ↻ 🔍

## Application: Strongly Connected Components

Given a directed graph  $G = (V, E)$ , a **strongly connected component** (SCC) of  $G$  is a maximal set of vertices  $C \subseteq V$  such that for every pair of vertices  $u, v \in C$   $u$  is reachable from  $v$  and  $v$  is reachable from  $u$

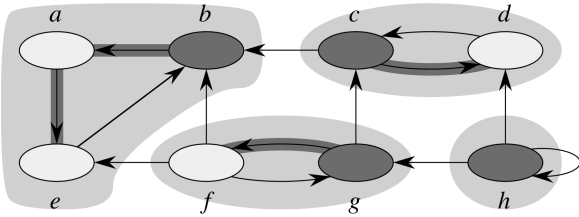


What are the SCCs of the above graph?

◀ ▶ ↻ 🔍

## Transpose Graph

- ▶ Our algorithm for finding SCCs of  $G$  depends on the **transpose** of  $G$ , denoted  $G^T$
- ▶  $G^T$  is simply  $G$  with edges reversed
- ▶ Fact:  $G^T$  and  $G$  have same SCCs. Why?



Navigation icons: back, forward, search, etc.

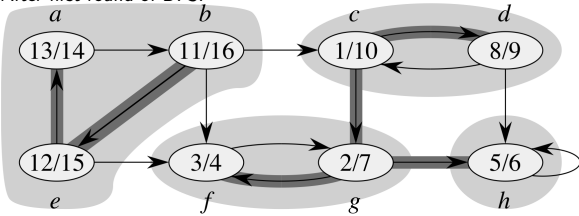
Navigation icons: back, forward, search, etc.

## SCC Algorithm

1. Call DFS algorithm on  $G$
2. Compute  $G^T$
3. Call DFS algorithm on  $G^T$ , looping through vertices in order of decreasing finishing times from first DFS call
4. Each DFS tree in second DFS run is an SCC in  $G$

## SCC Algorithm Example

After first round of DFS:



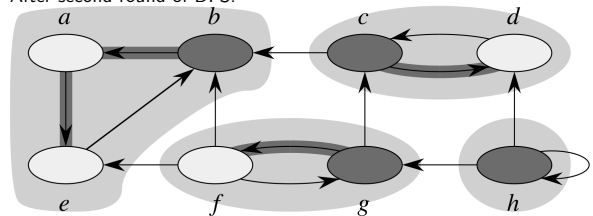
Which node is first one to be visited in second DFS?

Navigation icons: back, forward, search, etc.

Navigation icons: back, forward, search, etc.

## SCC Algorithm Example (2)

After second round of DFS:



Navigation icons: back, forward, search, etc.

## SCC Algorithm Analysis

- ▶ What is its time complexity?
- ▶ How does it work?
  1. Let  $x$  be node with highest finishing time in first DFS
  2. In  $G^T$ ,  $x$ 's component  $C$  has no edges to any other component (Lemma 22.14), so the second DFS's tree edges define exactly  $x$ 's component
  3. Now let  $x'$  be the next node explored in a new component  $C'$
  4. The only edges from  $C'$  to another component are to nodes in  $C$ , so the DFS tree edges define exactly the component for  $x'$
  5. And so on...

Navigation icons: back, forward, search, etc.