

Computer Science & Engineering 423/823

Design and Analysis of Algorithms

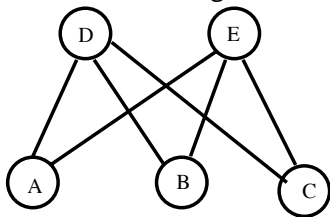
Lecture 03 — Elementary Graph Algorithms (Chapter 22)

Stephen Scott
(Adapted from Vinodchandran N. Variyam)

Spring 2010

- 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
- We'll build on these later this semester

- A **(simple, or undirected)** graph $G = (V, E)$ consists of V , a nonempty set of vertices and E a set of *unordered* pairs of distinct vertices called *edges*



$$V = \{A, B, C, D, E\}$$

$$E = \{ (A, D), (A, E), (B, D), (B, E), (C, D), (C, E) \}$$

Types of Graphs (2)

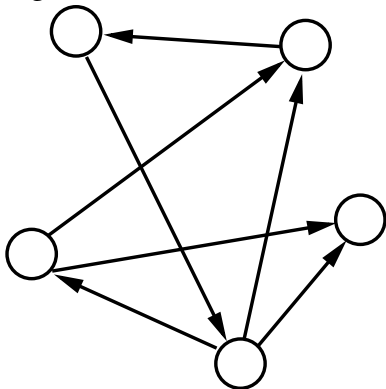
CSCE423/823

Introduction

Types of
GraphsRepresentations
of GraphsElementary
Graph
Algorithms

Applications

- A **directed** graph (digraph) $G = (V, E)$ consists of V , a nonempty set of vertices and E a set of *ordered* pairs of distinct vertices called *edges*



Types of Graphs (3)

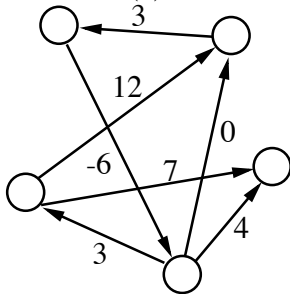
CSCE423/823

Introduction

Types of
GraphsRepresentations
of GraphsElementary
Graph
Algorithms

Applications

- A **weighted** graph is an undirected or directed graph with the additional property that each edge e has associated with it a real number $w(e)$ called its *weight*



- Other variations: multigraphs, pseudographs, etc.

- Two common ways of representing a graph: **Adjacency list** and **adjacency matrix**
- Let $G = (V, E)$ be a graph with n vertices and m edges

Adjacency List

CSCE423/823

Introduction

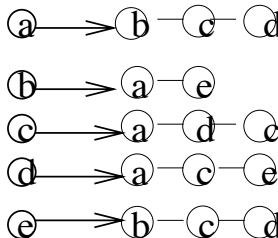
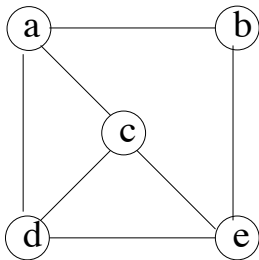
Types of
GraphsRepresentations
of Graphs

Adjacency List

Adjacency
MatrixElementary
Graph
Algorithms

Applications

- For each vertex $v \in V$, store a list of vertices adjacent to v
- For weighted graphs, add information to each node
- How much is space required for storage?



Adjacency Matrix

CSCE423/823

Introduction

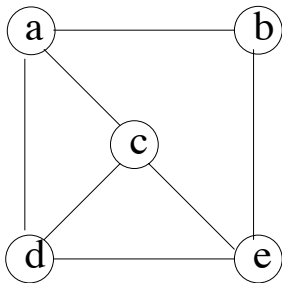
Types of
GraphsRepresentations
of Graphs

Adjacency List

Adjacency
MatrixElementary
Graph
Algorithms

Applications

- Use an $n \times n$ matrix M , where $M(i, j) = 1$ if (i, j) is an edge, 0 otherwise
- If G weighted, store weights in the matrix, using ∞ for non-edges
- How much is space required for storage?



	a	b	c	d	e
a	0	1	1	1	0
b	1	0	0	0	1
c	1	0	0	1	1
d	1	0	1	0	1
e	0	1	1	1	0

Breadth-First Search (BFS)

CSCE423/823

Introduction

Types of
GraphsRepresentations
of GraphsElementary
Graph
AlgorithmsBreadth-First
SearchDepth-First
Search

Applications

- 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 the 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 Algorithm

CSCE423/823

Introduction

Types of
Graphs

Representations
of Graphs

Elementary
Graph
Algorithms

Breadth-First
Search

Depth-First
Search

Applications

```

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  $\text{ENQUEUE}(Q, s)$ 
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              $\text{ENQUEUE}(Q, v)$ 
19         end
20     end
21      $color[u] = \text{BLACK}$ 
22 end
    
```

BFS Example

CSCE423/823

Introduction

Types of
Graphs

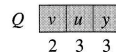
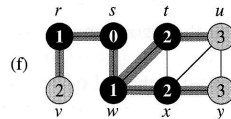
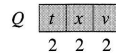
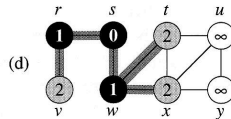
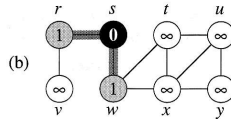
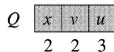
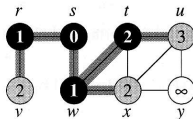
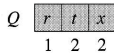
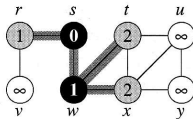
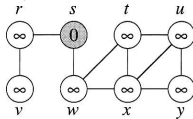
Representations
of Graphs

Elementary
Graph
Algorithms

Breadth-First
Search

Depth-First
Search

Applications



BFS Example (2)

CSCE423/823

Introduction

Types of
Graphs

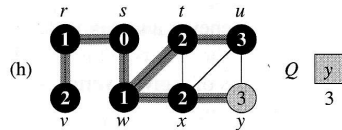
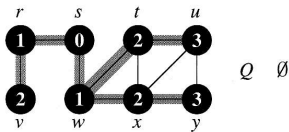
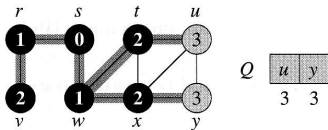
Representations
of Graphs

Elementary
Graph
Algorithms

Breadth-First
Search

Depth-First
Search

Applications



- 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)

CSCE423/823

Introduction

Types of
Graphs

Representations
of Graphs

Elementary
Graph
Algorithms

Breadth-First
Search

Depth-First
Search

Applications

- 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

```
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
10 end
```

Algorithm 2: DFS(G)

DFS Algorithm (2)

CSCE423/823

Introduction

Types of
GraphsRepresentations
of GraphsElementary
Graph
AlgorithmsBreadth-First
SearchDepth-First
Search

Applications

```
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  end
10  $color[u] = \text{BLACK}$ 
11  $f[u] = time = time + 1$ 
```

Algorithm 3: DFS-Visit(u)

DFS Example

CSCE423/823

Introduction

Types of
Graphs

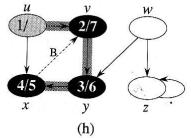
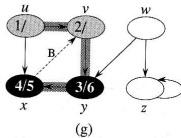
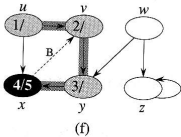
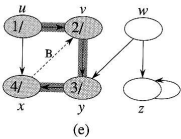
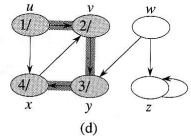
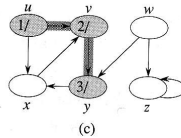
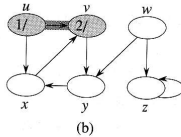
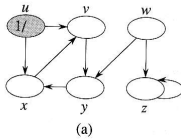
Representations
of Graphs

Elementary
Graph
Algorithms

Breadth-First
Search

Depth-First
Search

Applications



DFS Example (2)

CSCE423/823

Introduction

Types of
Graphs

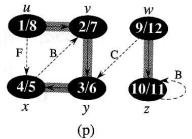
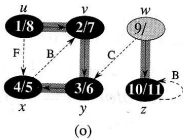
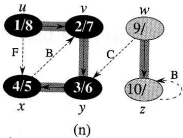
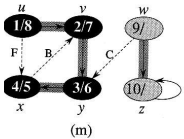
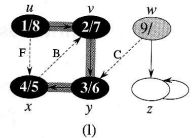
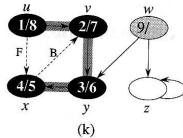
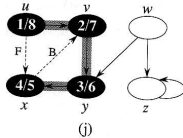
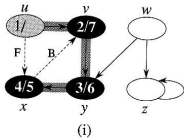
Representations
of Graphs

Elementary
Graph
Algorithms

Breadth-First
Search

Depth-First
Search

Applications



- 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)

CSCE423/823

Introduction

Types of
GraphsRepresentations
of GraphsElementary
Graph
AlgorithmsBreadth-First
SearchDepth-First
Search

Applications

- 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 edges 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

CSCE423/823

Introduction

Types of
Graphs

Representations
of Graphs

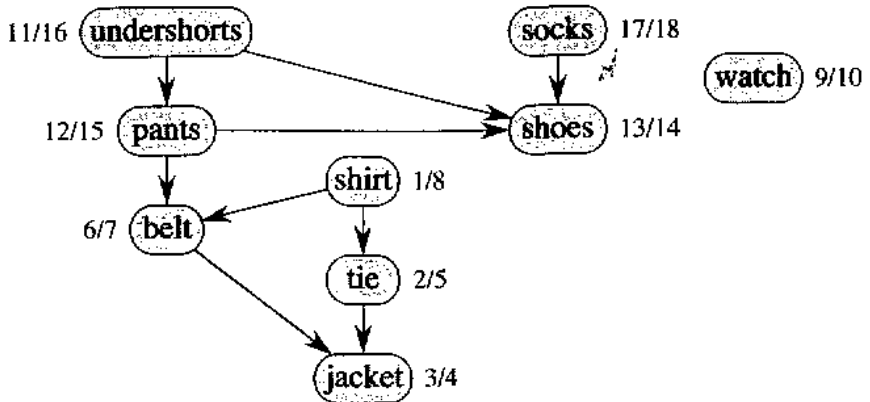
Elementary
Graph
Algorithms

Applications

Topological Sort

Strongly
Connected
Components

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)

CSCE423/823

Introduction

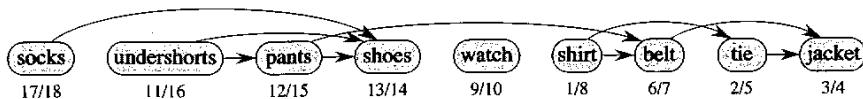
Types of
GraphsRepresentations
of GraphsElementary
Graph
Algorithms

Applications

Topological Sort

Strongly
Connected
Components

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

CSCE423/823

Introduction

Types of Graphs

Representations of Graphs

Elementary Graph Algorithms

Applications

Topological Sort

Strongly Connected Components

- 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
 - 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

CSCE423/823

Introduction

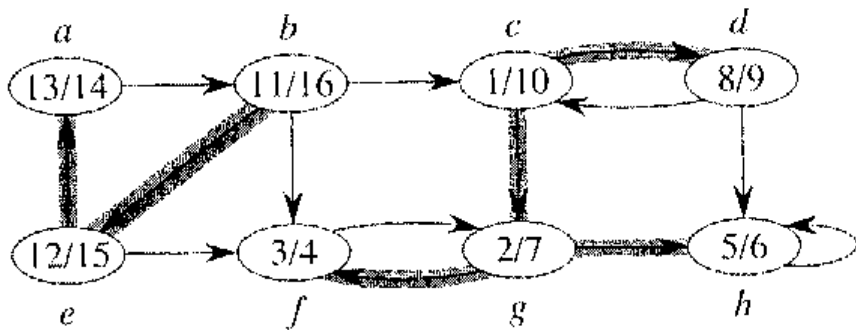
Types of
Graphs

Representations
of Graphs

Elementary
Graph
Algorithms

Applications
Topological Sort
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

CSCE423/823

Introduction

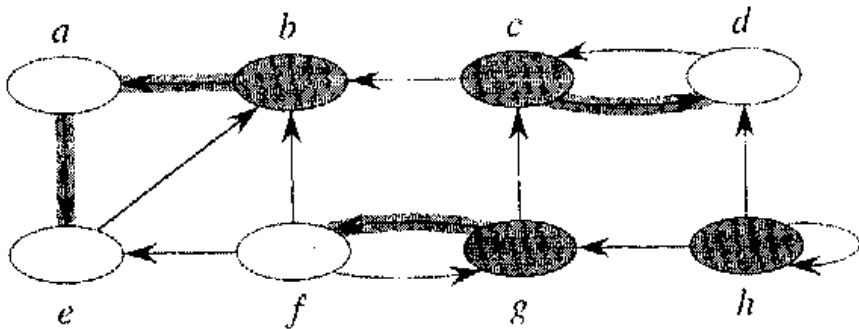
Types of
GraphsRepresentations
of GraphsElementary
Graph
Algorithms

Applications

Topological Sort

Strongly
Connected
Components

- 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?



- 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

CSCE423/823

Introduction

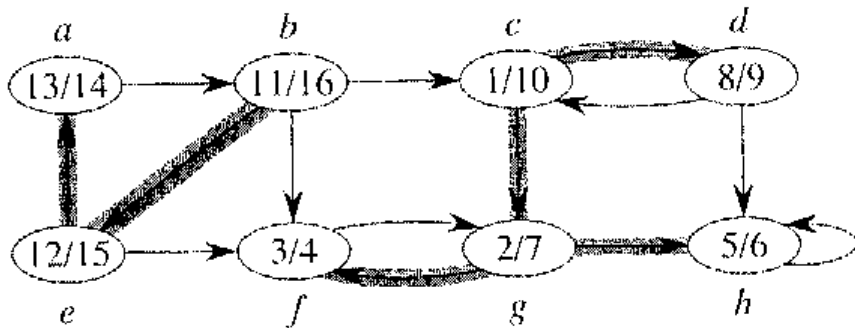
Types of
Graphs

Representations
of Graphs

Elementary
Graph
Algorithms

Applications
Topological Sort
Strongly
Connected
Components

After first round of DFS:



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

SCC Algorithm Example (2)

CSCE423/823

Introduction

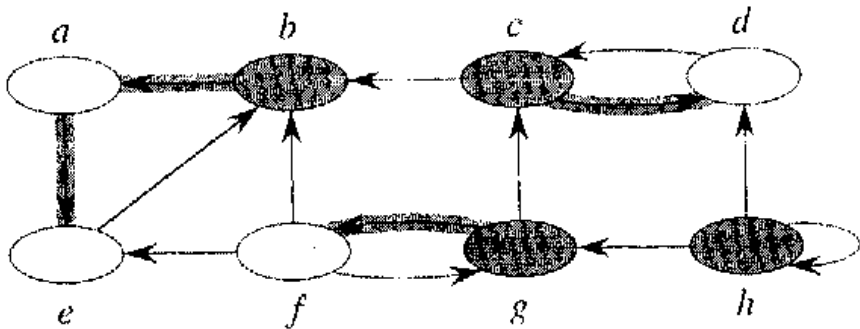
Types of
Graphs

Representations
of Graphs

Elementary
Graph
Algorithms

Applications
Topological Sort
Strongly
Connected
Components

After second round of DFS:



- 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, 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...