COMP 182 Algorithmic Thinking

Graphs: Terminology, Representation, and Connectivity

Luay Nakhleh
Computer Science
Rice University

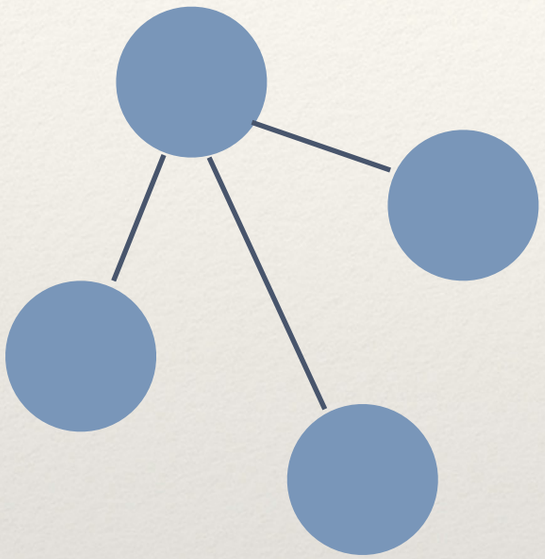
Reading Material

- Chapter 10, Sections 1-4

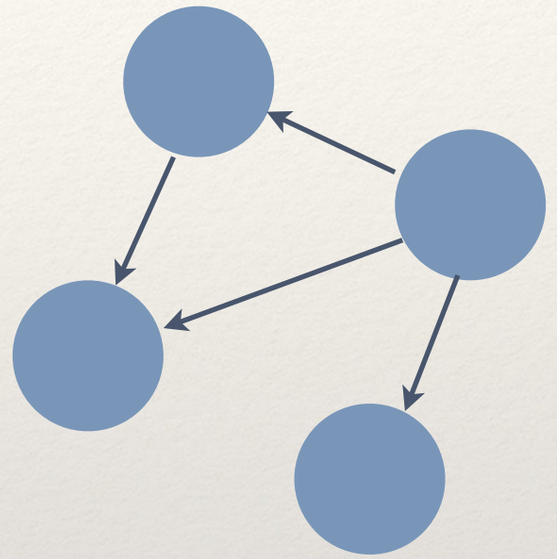
Why Graphs?

- Biological networks
- Power networks
- Road maps
- Social networks
- Scheduling problems
- …

Directed and Undirected Graphs

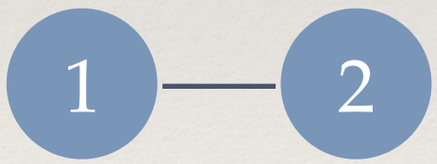


Undirected graph



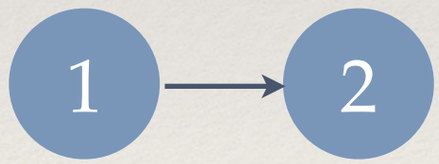
Directed graph

: node
(or, vertex)



: edge

{1,2}



: directed edge

(1,2)

Undirected Graphs

- An undirected graph, or graph for short, G, is a pair (V,E), where
  - V is a non-empty set of nodes, and
  - $E \subseteq \{\{a,b\} \mid a,b \in V\}$ is a set of (undirected) edges.

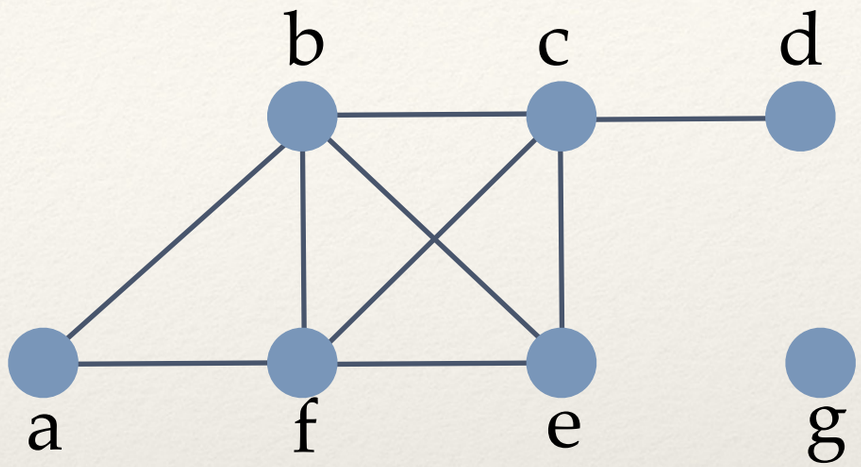
Directed Graphs

- A directed graph, or digraph for short, G, is a pair (V,E), where
  - V is a non-empty set of nodes, and
  - $E \subseteq V \times V$ is a set of (directed) edges.

IMPORTANT

- In this course, we will not consider graphs that have any of the following:
  - An edge whose two endpoints consist of the same node (self-loop)
  - More than one undirected edge between the same two nodes (parallel undirected edges)
  - More than one directed edge between the same two nodes in the same direction (parallel directed edges)

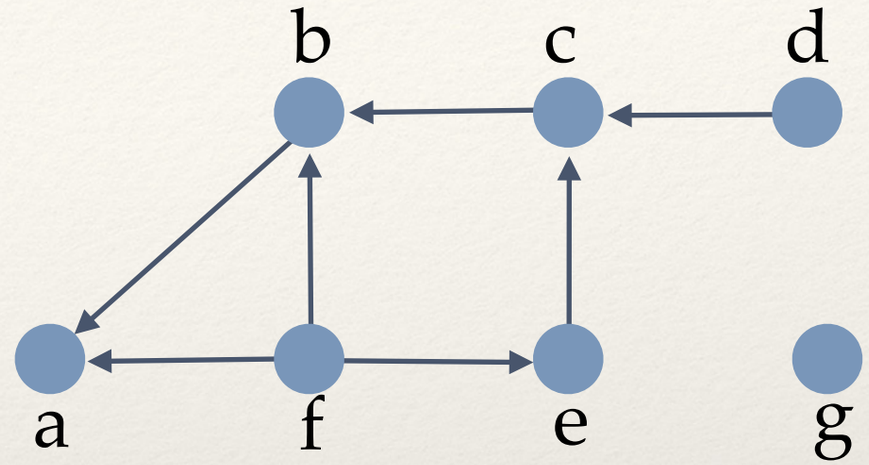
Basic Terminology



- Let $G=(V,E)$ be a graph.
- Two nodes $u,v \in V$ are <u>adjacent</u> or <u>neighbors</u> if $\{u,v\} \in E$.
- The <u>degree</u> of node $u \in V$, denoted by $deg(u)$, is the number of u's neighbors. That is,

$$deg(u) = |\{v \in V : \{u, v\} \in E\}|$$

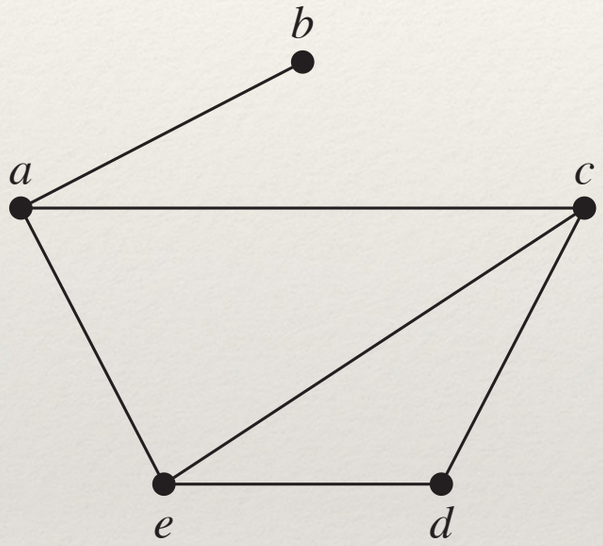
Basic Terminology



- Let $G=(V,E)$ be a directed graph.
- For edge $(u,v) \in E$, we call u the tail of the edge and v the head of the edge.
- The in-degree of node $u \in V$, denoted by $indeg(u)$, is the number of edges whose head is the node u.
- The out-degree of node $u \in V$, denoted by $outdeg(u)$, is the number of edges whose tail is the node u.

Graph Representation: Adjacency Lists

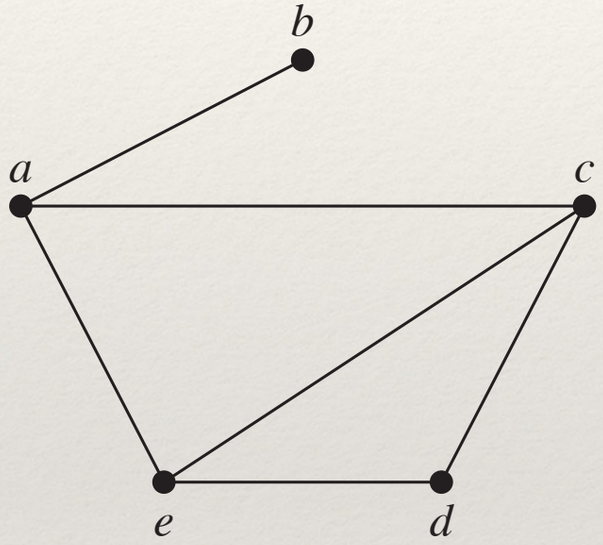
- The adjacency list of a graph specifies the neighbors of each node.



Vertex	*Adjacent Vertices*
a	b, c, e
b	a
c	a, d, e
d	c, e
e	a, c, d

Graph Representation: Adjacency Lists

- The adjacency list of a graph specifies the neighbors of each node.



Vertex	*Adjacent Vertices*
a	b, c, e
b	a
c	a, d, e
d	c, e
e	a, c, d

Python dictionary representation

{a: set([b,c,e]),
 b: set([a]),
 c: set([a,d,e]),
 d: set([c,e]),
 e: set([a,c,d])}

Graph Representation: Adjacency Matrices

- If the nodes of a graph $g=(V,E)$ are $v_1,v_2,\ldots,v_n$, then the adjacency matrix of the graph, denoted by A_G, is an $n \times n$ 0-1 matrix such that

$$A_G[i,j] = \begin{cases} 0 & \text{if } \{v_i, v_j\} \notin E \\ 1 & \text{if } \{v_i, v_j\} \in E \end{cases}$$

Graph Representation: Adjacency Matrices

- If the nodes of a graph $g=(V,E)$ are $v_1,v_2,\ldots,v_n$, then the adjacency matrix of the graph, denoted by A_G, is an $n \times n$ 0-1 matrix such that

$$A_G[i,j] = \begin{cases} 0 & \text{if } \{v_i, v_j\} \notin E \\ 1 & \text{if } \{v_i, v_j\} \in E \end{cases}$$

(v_i,v_j) in the case of directed graphs

Tradeoffs Between the Two Representations

- When the graph is sparse, i.e., contains relatively few edges, it is usually preferable to use adjacency lists. Why?
- When the graph is dense, i.e., contains relatively many edges, it is usually preferable to use adjacency matrices. Why?

Tradeoffs Between the Two Representations

relative to what?

- When the graph is sparse, i.e., contains relatively few edges, it is usually preferable to use adjacency lists. Why?
- When the graph is dense, i.e., contains relatively many edges, it is usually preferable to use adjacency matrices. Why?

Graph Connectivity: Paths

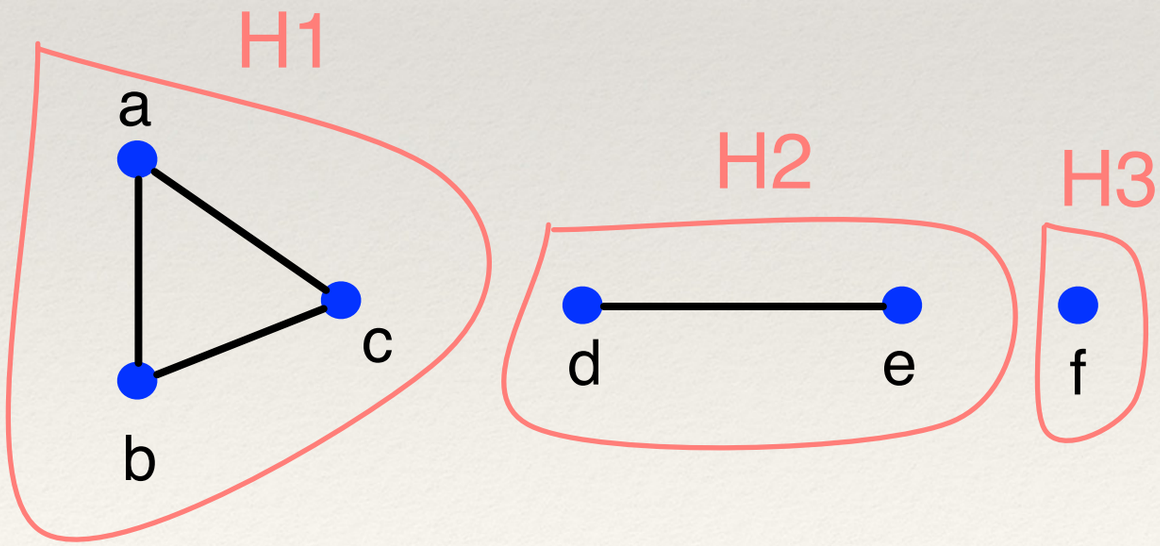
- Let k be a nonnegative integer.
- A path of length k between nodes v_0 and v_k in graph $G=(V,E)$ is a sequence of k edges $e_1,e_2,\ldots,e_k \in E$ such that $e_1=\{v_0,v_1\}$, $e_2=\{v_1,v_2\},\ldots,e_k=\{v_{k-1},v_k\}$ and $v_1,v_2,\ldots,v_{k-1} \in V$.
- A path of length k from node v_0 to v_k in digraph $G=(V,E)$ is a sequence of k edges $e_1,e_2,\ldots,e_k \in E$ such that $e_1=(v_0,v_1)$, $e_2=(v_1,v_2),\ldots,e_k=(v_{k-1},v_k)$ and $v_1,v_2,\ldots,v_{k-1} \in V$.
- We often denote such a path by its node sequence $(v_0,v_1,\ldots,v_k)$.

Graph Connectivity: Paths

- A path is simple if it does not contain the same node more than once.
- A cycle is a simple path that begins and ends at the same node.
- A path (not necessarily simple) that begins and ends at the same node is called a circuit.

Graph Connectivity: Connected Components

- A graph is called connected if there is a path between every pair of nodes in the graph.
- A connected component (CC) of a graph G is a maximal subset of nodes C such that there is a path between every pair of nodes in C (the path uses only nodes in C).

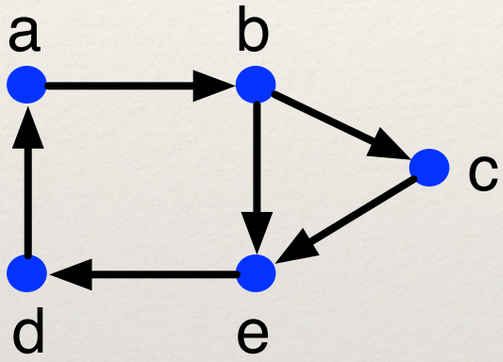


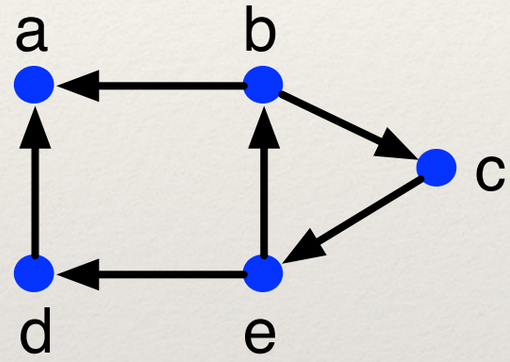
A graph with 3 CCs

Graph Connectivity: Connected Components

- A digraph is strongly connected if there is a path from every node to every other node in the digraph.
- A digraph is weakly connected if its underlying undirected graph is connected.
- A strongly connected component (SCC) of a digraph *G* is a maximal set of nodes *C* such that there is a path from every node to every other node in *C* (the path uses only nodes in *C*).

Graph Connectivity: Connected Components





Strongly connected? Weakly connected? What are the SCCs?

Questions?