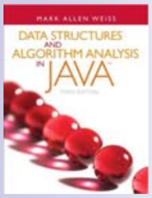
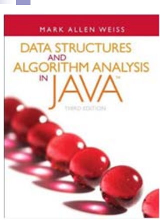


FLORIDA INTERNATIONAL UNIVERSITY

Some Problems on Graphs (I)

Dr. Antonio L. Bajuelos

FIU School of Computing & Information Sciences



COP-3530 - Data Structures

Module #7: Some Problems on Graphs (part I)

Outline:

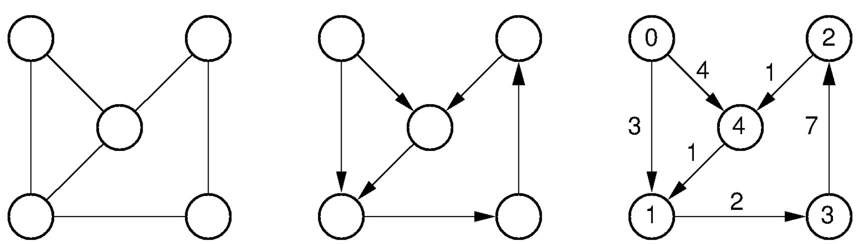
- **Preliminary concepts and results.**
- **Graphs and its representations.**

Graphs. Some Definitions

- A **graph** G = <V,E> consists of a set of **vertices**, V, and a set of **edges**, E.
- Each edge is a pair (v,w), where v and w ∈ V. Edges are sometimes referred to as **arcs**.
- If the pair is ordered, then the graph is **directed**. Directed graphs are sometimes referred to as **digraphs**.
- Vertex w is **adjacent** to v if and only if (v,w) ∈ E.
- In an **undirected graph** with edge (v,w), and hence (w,v), w is **adjacent** to v and v is adjacent to w.
- Sometimes an edge has a third component, known as either a **weight** or a **cost** of the edge.

Graphs. Some Definitions

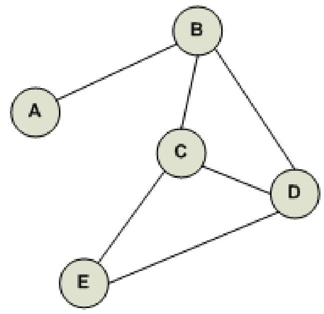
- Some examples of Graphs:



Undirected Graph   Direct Graph (Digraph)   Weighed Graph

Undirected Graphs.

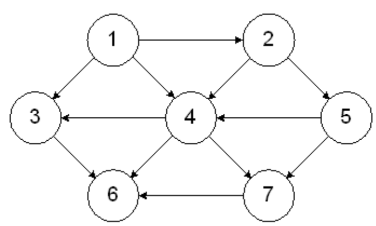
- Edges in the **undirected graphs**, have no specific direction i.e. **edges** are always “two-way”
- Thus, $(u,v) \in E \Rightarrow (v,u) \in E$
  - Only one of these edges needs to be in the set
- **Degree** of a **vertex**:
  - number of edges containing that vertex **or**
  - number of adjacent vertices
  - Example:
    - Degree of A is 1
    - Degree of B, and C is 3



5

Directed Graphs.

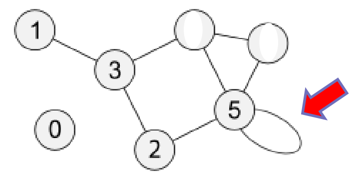
- Edges in the **directed graphs** (**digraphs**) have a direction
  - $(u,v) \in E \not\Rightarrow (v,u) \in E$.
  - The u is the **source** and v the **destination**
  - **In-degree** of a vertex: edges where the vertex is the destination
  - **Out-degree** of a vertex: edges where the vertex is the source.
  - Example:
    - In-degree of vertex 4 is 3
    - Out-degree of vertex 1 is 3
    - In-degree of vertex 1 is 0

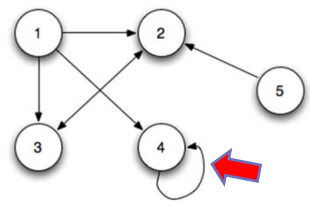


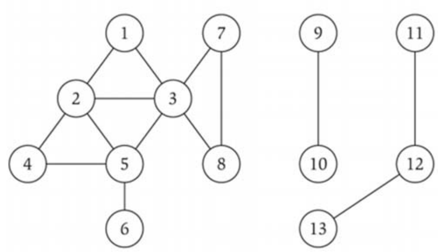
6

Graphs. Connectedness

- **Loop** is an edge that connects a vertex to itself i.e. (u,u).
- A **simple graph** contains no **loops**.
- A vertex of the graph can have a degree (in-degree or/and out-degree) equal to zero.
- A graph does not have to be connected







7

Graphs. Some bounds

- Let a **graph** G = <V,E>
- Assume that:
  - |V| is the number of vertices
  - |E| is the number of edges
- **Minimum of |E|?**
  - 0 (1 if self-edges (loops) allowed)
- **Maximum of |E| for undirected?**
  - **((|V||V-1|)/2 + |V|)** $\in$**O(|V|^2)**

  (assuming self-edges allowed, else subtract |V|)
- **Maximum of |E| for directed?**
  - **|V|^2**

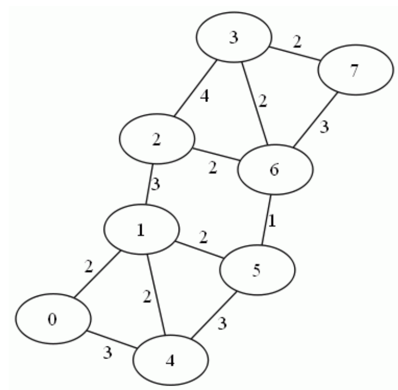
  (assuming self-edges allowed, else subtract |V|)
- In real-life situations: **|V| < |E| << |V|^2**
- Given a (**directed** or **undirected**) graph G= <V,E>,

$$2|\mathrm{E}| = \sum_{v \in V} \deg(v) .$$

8

Graphs. Path and Cycles

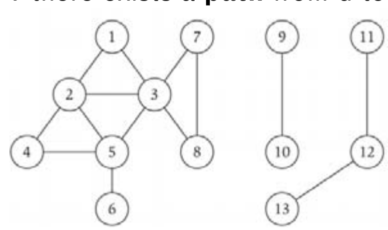
- Let a **graph** G = <V,E>
- A **path** in the graph G is a list of vertices $[v_j,\ldots,v_k]$ such that $(v_i,v_{i+1}) \in E$ for all $j \leq i < k$. Say "a path from v_j to v_k"
- A **cycle** is a path that begins and ends at the same node ($v_j=v_k$).
- **Path length:** Number of edges in a path.
- **Path cost:** Sum of weights of edges in a path



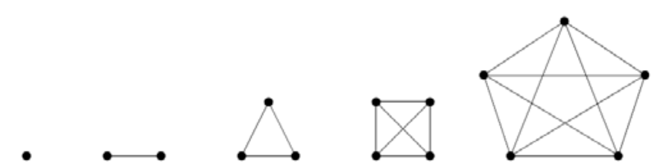
9

Graphs. Undirected-Graph Connectivity

- An **undirected graph** is **connected** if for all pairs of vertices u and v there exists a **path** from u to v



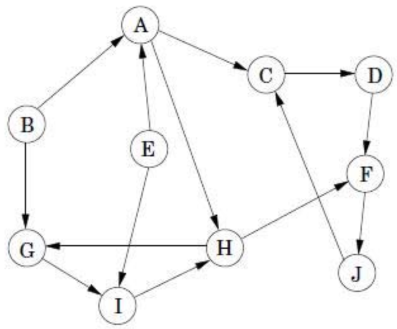
- An **undirected graph** is **complete** (or fully connected) if for all pairs of vertices u and v there exists an **edge** from u to v



10

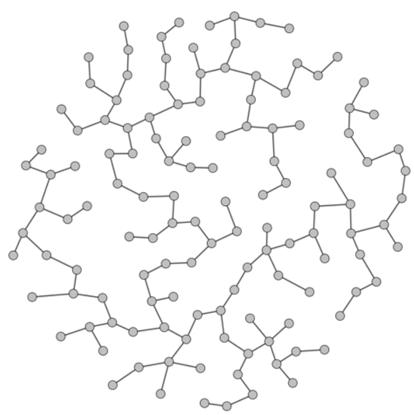
Strongly vs Weakly Connected

- Let a **graph** G = <V,E>
- A directed graph is **strongly connected** if there is a path from every vertex to every other vertex
- A directed graph is **weakly connected** if there is a path from every vertex to every other vertex ignoring direction of edges
- **Example:**



Trees and Graphs

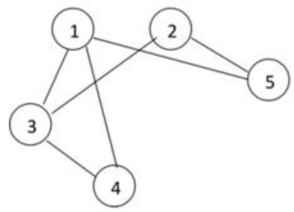
- Equivalent forms of the definition of a **tree**:
  - A **tree** is a **connected graph** that contains **no cycles**.
  - A **tree** is a **graph** with exactly one **path** between any two vertices.
  - A **connected graph** of n vertices is a **tree** iff it has n-1 edges.
- All trees are **graphs**, but not all **graphs** are **trees**



Graphs. Adjacency Matrix Representation

- Let a **graph** G = <V,E> and assume that |V|=N
- In thi adjacency matrix representation, each graph of N nodes is represented by an N x N matrix A, that is, a two-dimensional array A
- The nodes are (re)-labeled 1,2,…,n (or 0 to n-1)
  - A[i][j] = 1 if (i,j) is an edge in the graph
  - A[i][j] = 0 if (i,j) is not an edge in the graph

0	0	1	1	1
0	0	1	0	1
1	1	0	1	0
1	0	1	0	0
1	1	0	0	0

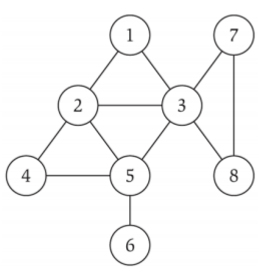


  - Adjacency Matrix representation for **digraphs**??

13

Graphs. Adjacency Matrix Representation

- **Example:**



	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

14

Graphs. Adjacency Matrix Properties

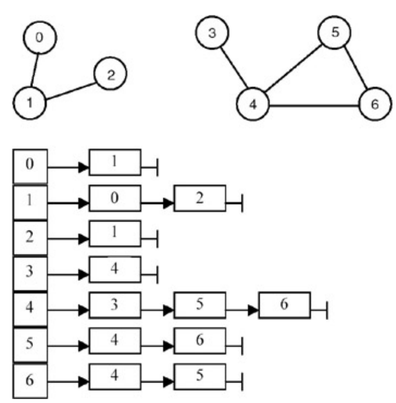
- Running time to:
  - Get a vertex's out-edges: **O(|V|)**
  - Get a vertex's in-edges: **O(|V|)**
  - Decide if some edge exists: **O(1)**
  - Insert an edge: **O(1)**
  - Delete an edge: **O(1)**
- Space requirements: **$O(|V|^2)$** bits
- Good representation for **dense graphs***
- Undirected will be symmetric around the diagonal
- How can we adapt the representation for weighted graphs?
  - Store a number in each cell (weight)

***dense graph** is a graph in which the number of edges is close to the maximal number of edges (**$O(|V|^2)$**).

15

Graphs. Adjacency List Representation

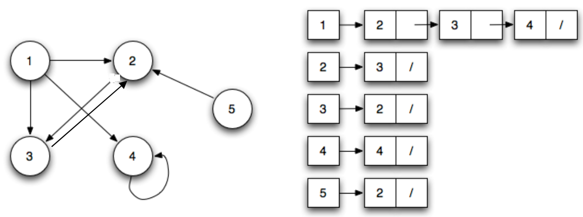
- Let a **graph** G = <V,E> and assume that |V|=N
- A graph of N vertices is represented by a one dimensional array L of linked lists, where:
  - L[i] is the linked list containing all the nodes adjacent from node i.
  - The nodes in the list L[i] are in no particular order



16

Graphs. Adjacency List Properties

- Let a **graph** G = <V,E> and assume that |V|=N
- Running time to:
  - Get all of a vertex's out-edges:
    - **O(d)** where d is out-degree of vertex
  - Get all of a vertex's in-edges:
    - **O(|E|)** (but could keep a second adjacency list for this)
  - Decide if some edge exists:
    - **O(d)** where d is out-degree of source
  - Insert an edge:
    - **O(1)** (unless you need to check if it's there)
  - Delete an edge:
    - **O(d)** where d is out-degree of source
- Space requirements: **O(|V|+|E|)**



17