• Graphs are a formal description of a wide range of situations
  • most common example: road map
    ■ in graph terms, intersections are nodes
    ■ roads are edges
  • sometimes graphs are directed (one-way street)
  • sometimes edges are weighted with some cost
    ■ e.g., speed limit
  • look at:
    ■ basic concepts
    ■ methods for storing graphs
    ■ how to use and manipulate using algorithms
      ■ e.g., DFS, BFS
    ■ spanning trees (no cycles in graph)
    ■ shortest path

• Graph Background and Terminology
  • formally, a graph is an ordered pair
    ■ \( G = (V, E) \)
  • of two sets represented the nodes or vertices of the graph and the edges of the graph
    ■ \( V \) = vertices
    ■ \( E \) = edges
  • often talk of “traveling” an edge between vertices A and B
    ■ “traversing” from A to B
    ■ “moving” from A to B
  • normally just write the two node labels as shorthand for an edge
    ■ So AB represents edge A→B
    ■ B is adjacent to A

• Graphs can be either directed or undirected
  • undirected graph (or just “graph”)
    ■ has edges that can be traversed in any direction
    ■ an edge is a set containing labels of nodes on ends of edge
  • directed graph (aka digraph)
    ■ edges can be traversed in only one direction
    ■ set of edges will have ordered pairs
      ■ first item is where edge starts
      ■ second item is where edge ends

• Normally draw graphs instead of defining their sets
  ■ circles = nodes
  ■ lines = edges
    ■ node names inside circles
    ■ use arrows for directed graphs
  ■ E.g.,

![Figure 8.1A](image1.png) ![Figure 8.1B](image2.png)

• Terminology
  ■ complete graph - an edge between every pair of nodes
    ■ N nodes \( \Rightarrow (N^2 - N) / 2 \) edges
      ■ (excluding loops)
    ■ complete digraph has an edge allowing traversal b/t every pair of nodes
      ■ N nodes \( \Rightarrow N^2 - N \) edges
        ■ (twice as many edges)
  ■ subgraph \( (V_s, E_s) \) of graph \( (V,E) \) is one that has a subset of vertices
- \((V_s \subseteq V)\)
- \((E_s \subseteq E)\)
- only edges that are candidates for \(E_s\) must contain both nodes in \(V_s\)

- **Path** between two nodes of graph or digraph is a sequence of edges that can be traveled consecutively w/out passing through any node more than once
  - path b/t A and B
    - start at A
    - traverse set of edges until reaches B
    - without going through any node 2x
  - path has a length:
    - number of edges that make up the path
    - path AB, BC, CD, DE has length 4
  - **Cycle** - path that starts and ends at same node

- **Weighted Graph** - each edge has a value (weight) associated with it
  - when drawing, weights written near edge
  - formally, weight is 3rd component in set of edge or ordered pair (now triplet)
  - when working with weighted graphs, must consider weights during traversal
    - cost
    - path through weight graph has cost for traversing each edge
      - e.g. P1 has 5 edges and cost 24
      - P2 has 3 edges and cost 36
      - P1 is “shorted” because it costs less.. (even though path length is longer)

- **Connected** - graph or digraph is “connected” if there is at least one path b/t every pair of nodes
- **Acyclic Graph** - has no cycles
- **Unrooted Tree** - graph that is connected and acyclic-
  - no single node is the root..

**Data Structures for Graphs**

- we can store graphs or digraphs in two ways:
  - adjacency matrix
  - adjacency list

- **Adjacency Matrix**
  - gives us ability to quickly access edge information
  - but if graph is far from being complete, then there will be empty space in array

- **Adjacency List**
  - uses space that is proportional to the number of edges in graph
  - but time to access will be greater
  - no clear benefit to either method
  - adjacency list- useful when graph has many nodes but few edges
  - uses less space
  - adjacency matrix- useful when graph has few nodes or when nearly complete
  - not that big

**Adjacency Matrix**
- adjacency matrix for graph \(G = (V,E)\) with \(|V| = N\) will be stored in a 2D array of size \(N \times N\)
  - each location \([i,j]\) specifies an edge between node \(v_i\) and \(v_j\)
    - if 0, no edge
    - if 1, edge b/t nodes
  - allows us to do digraphs too...
  - for weighted graphs, value used instead of 0 and 1
    - with inf indicating no edge
  - diagonal elements all 0
    - (no edge b/t node and itself)
- **Adjacency list**
  - adjacency list for graph \( G = (V,E) \) with \( |V| = N \) will be stored as a 1D array of size \( N \) with each location being a pointer to a linked list
  - there will be one list for each node
  - list will have one entry for each adjacent node
  - for weighted graphs and digraphs
    - list entries have an additional field to hold weight of edge.
**FIGURE 8.1A**
The graph $G = ([1, 2, 3, 4, 5], \{(1, 2), (1, 3), (2, 3), (2, 4), (3, 5), (4, 5)\})$

**FIGURE 8.3A**
The adjacency list for the graph in Fig. 8.1(a)

**FIGURE 8.1B**
The directed graph $G = ([1, 2, 3, 4, 5], \{(1, 2), (1, 3), (2, 1), (3, 2), (4, 3), (4, 5), (5, 2), (5, 4)\})$

**FIGURE 8.3B**
The adjacency list for the graph in Fig. 8.1(b)