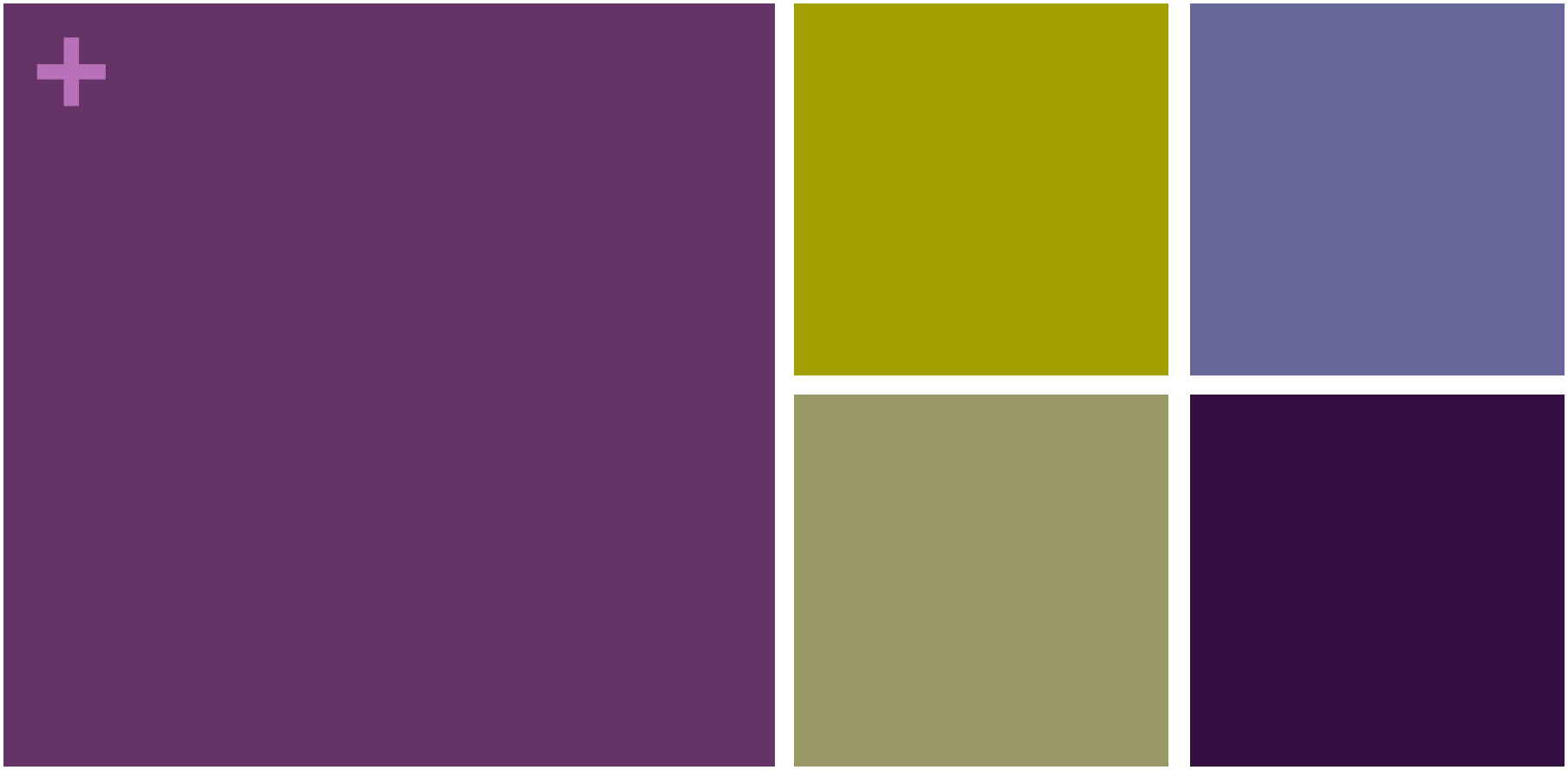


DISCRETE MATHEMATICS

GRAPHS

TOPICS

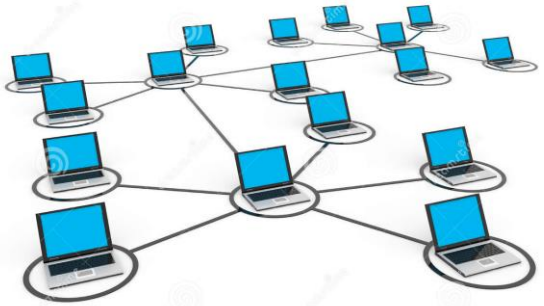
- Graphs and Graph Models
- Graph Terminology
- Special Types of Graphs
- Representing Graphs
- Connectivity
- Euler and Hamilton Paths
- Shortest-Path Problems

TOPICS

- **Graphs and Graph Models**
- Graph Terminology
- Special Types of Graphs
- Representing Graphs
- Connectivity
- Euler and Hamilton Paths
- Shortest-Path Problems

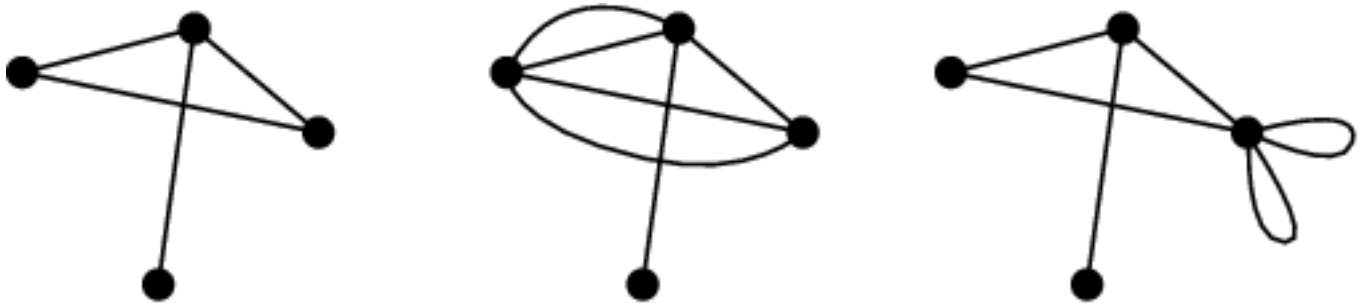
Definition

- A graph G = (V,E) consists of V, a **nonempty** set of **vertices** (or **nodes**) and E, a set of **edges**.
- Each edge has either one or two vertices associated with it, called its **endpoints**. An edge is said to **connect** its endpoints.
- A graph with an infinite vertex set or an infinite number of edges is called an **infinite graph**
- A graph with a finite vertex set and a finite edge set is called a **finite graph**



Graph classification

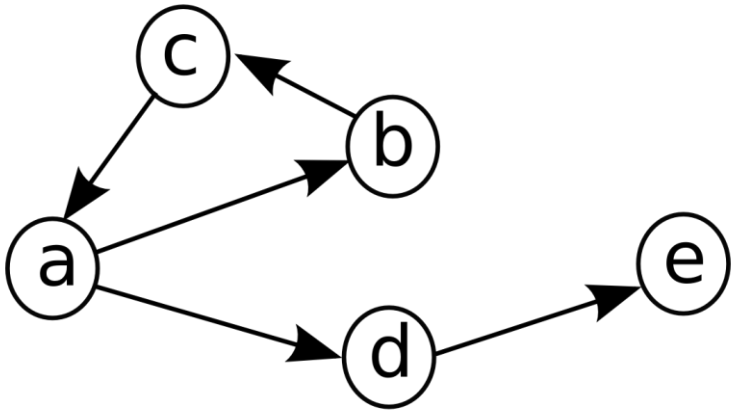
- Undirected graph: edges has no direction
  - Simple graph: no multiple edges, no loops
  - Multigraph: multiple edges, no loops
  - Pseudograph: multiple edges, loops



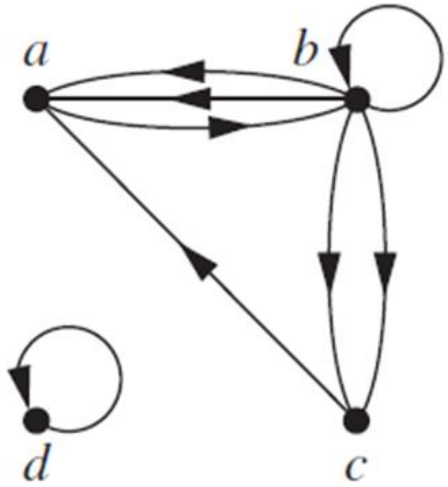
simple graph multigraph pseudograph

Graph classification

- Directed graph: edges have direction between endpoints
  - Simple directed graph (digraph): no multiple edges, no loops
  - Directed multigraph: multiple edges, loops



simple directed graph

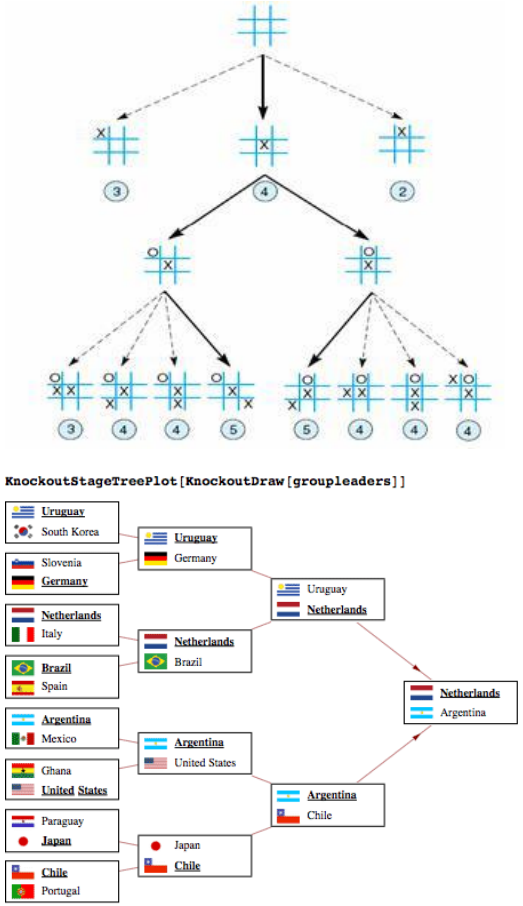


directed multigraph

- Mixed graph: both undirected & directed

Graph models

- Using graph as a model
  - Social network
  - Collaboration graphs
  - Web graph
  - State space (in AI)
  - Airline route
  - Football tournament
  - …



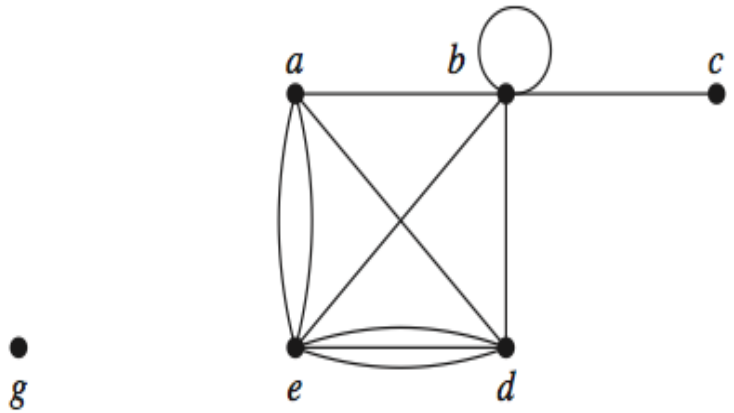
TOPICS

- Graphs and Graph Models
- **Graph Terminology**
- Special Types of Graphs
- Representing Graphs
- Connectivity
- Euler and Hamilton Paths
- Shortest-Path Problems

Basic terminologies for undirected graph

- If G is undirected, two vertices u and v are called **adjacent** if they are endpoints of an edge e, and e is called **incident** with u and v.
- The **degree** of a vertex in an undirected graph is the **number of edges** incident with it (a loop at a vertex contributes twice)
- A vertex of degree zero is called **isolated**. A vertex of degree one is called **pendant**.

- g is isolated
- c is pendant
- deg(a) = 4
- deg(b) = 6



The Handshaking theorem for undirected graph

- Let G = (V , E) be an undirected graph with m edges. Then

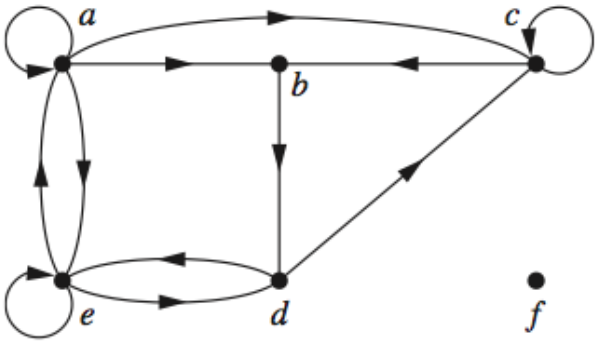
$$\sum_{v \in V} \deg(v) = 2m$$

- An undirected graph has an even number of vertices of odd degree.

- Example: Can we have a simple graph with 5 vertices of degrees: 1, 2, 3, 3, 4?

Basic terminologies for directed graph

- If e = (u,v) is an edge of a directed graph, u is said to be **adjacent to** v and v is **adjacent from** u. The vertex u is called the **initial** and v is called the **terminal** or end vertex of e.
- The **in-degree** of a vertex v in a directed graph, denoted by $\deg^-(v)$, is the number of edges with v as their terminal vertex. The **out-degree** of u, denoted by $\deg^+(v)$, is the number of edges with v as their initial vertex.

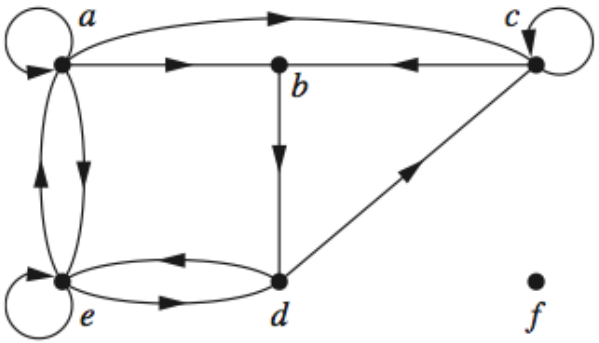


- a is adjacent to c
- d is adjancent from b
- $\deg^-(c) = 3$
- $\deg^+(c) = 2$
- $\deg^-(f) = \deg^+(f) = 0$

The Handshaking theorem for directed graph

- Let G = (V , E) be a graph with directed edges. Then

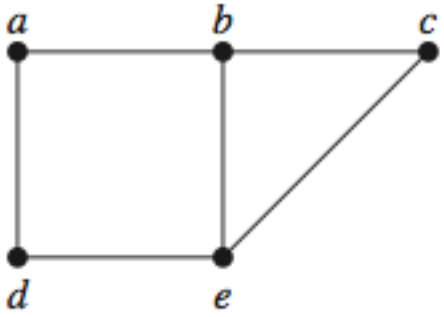
$$\sum_{v \in V} deg^{+}(v) = \sum_{v \in V} deg^{-}(v) = |E|$$



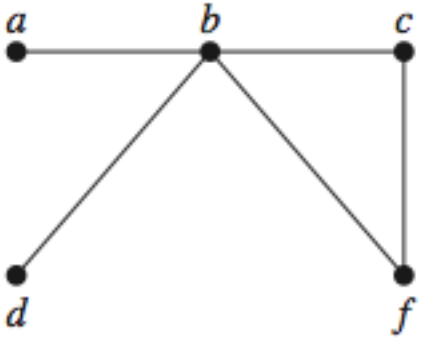
$$\sum_{v \in V} deg^{+}(v) = \sum_{v \in V} deg^{-}(v) = 12$$

Graph Union

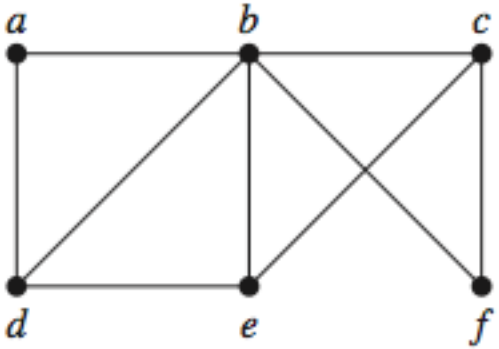
- The union of 2 graphs G and H is a new graph whose vertex set consists of vertices of G and H, and whose edge set consists of edges of G and H



G_1



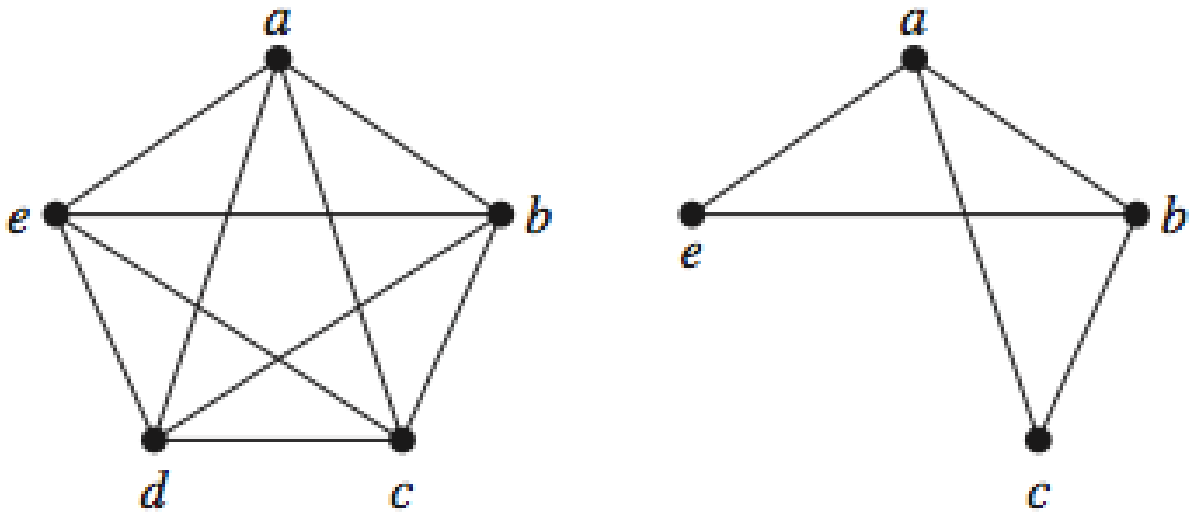
G_2



$G_1 \cup G_2$

Subgraph

- A subgraph of a graph G = (V,E) is a graph H = (W,F), where W ⊆ V and F ⊆ E. A subgraph H of G is a proper subgraph of G if H ≠ G.

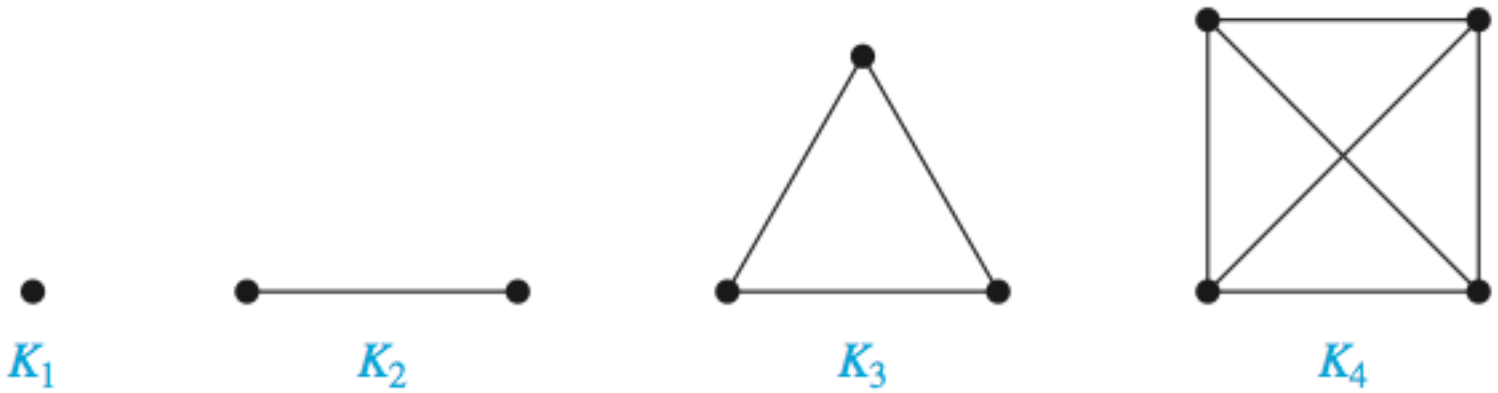


TOPICS

- Graphs and Graph Models
- Graph Terminology
- **Special Types of Graphs**
- Representing Graphs
- Connectivity
- Euler and Hamilton Paths
- Shortest-Path Problems

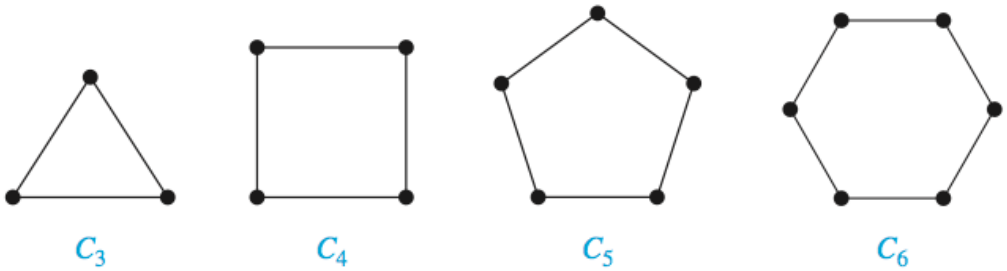
Complete graph

- A complete graph on n vertices, denoted by K_n, is a simple graph that contains **exactly one** edge between **each pair of distinct** vertices.

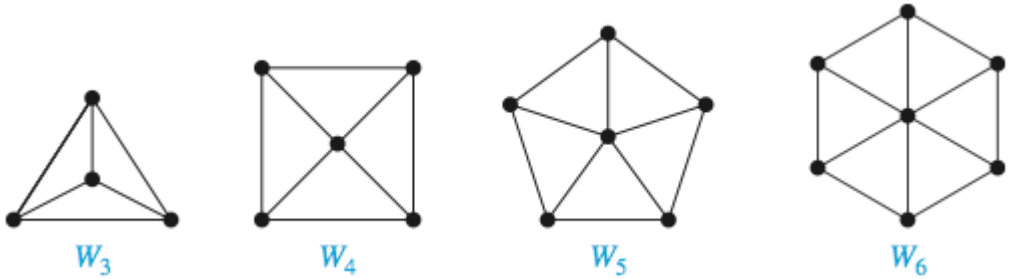


Cycle graph & Wheel graph

- A cycle C_n, $n \geq 3$, consists of n vertices $v_1, v_2, \ldots, v_n$ and edges $\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}$, and $\{v_n, v_1\}$.

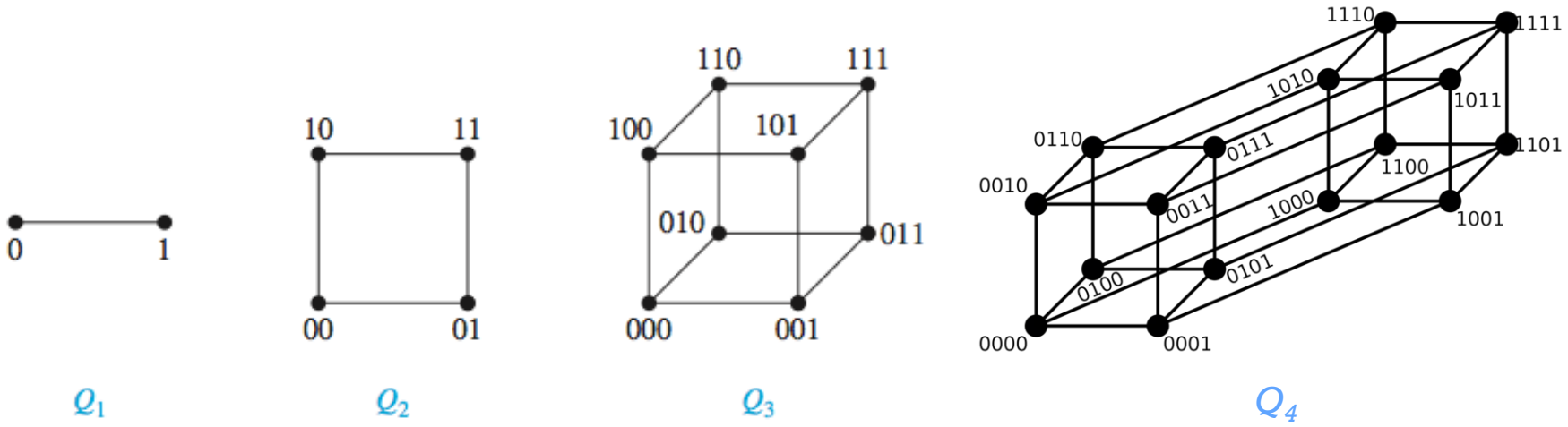


- We obtain a wheel W_n when we add an additional vertex to a cycle C_n, for $n \geq 3$, and connect this new vertex to each of the n vertices in C_n, by new edges.



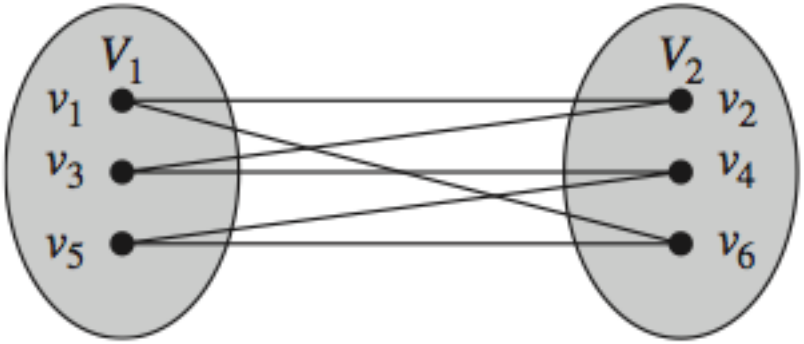
n-cube graph

- An n-dimensional hypercube, or n-cube, denoted by Q_n, is a graph that has vertices representing the bit strings of length n. Two vertices are **adjacent** if and only if the bit strings that they represent **differ in exactly one bit** position.



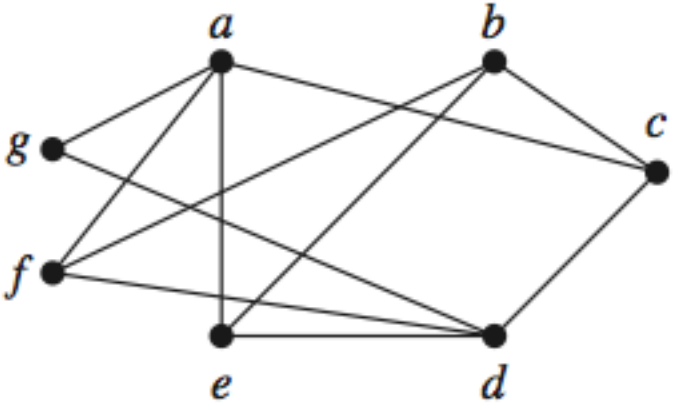
Bipartite graph

- A simple graph G is called bipartite if the vertex set can be divided in **two disjoint** subsets such that each edge connects **one** vertex from one of these two subsets **to another** vertex of the other subset.

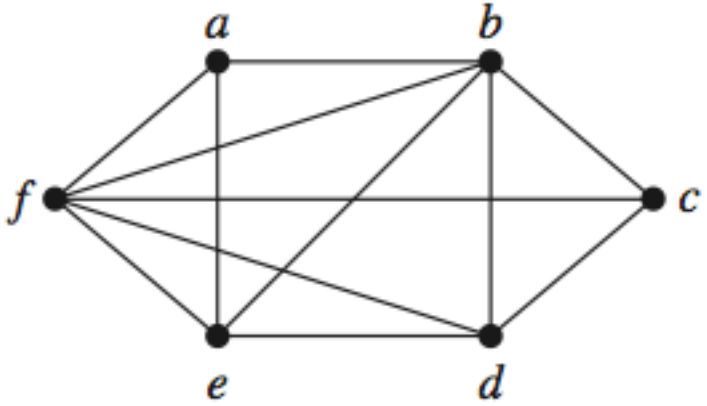


Bipartite graph

- A simple graph G is called bipartite if the vertex set can be divided in **two disjoint** subsets such that each edge connects **one** vertex from one of these two subsets **to another** vertex of the other subset.



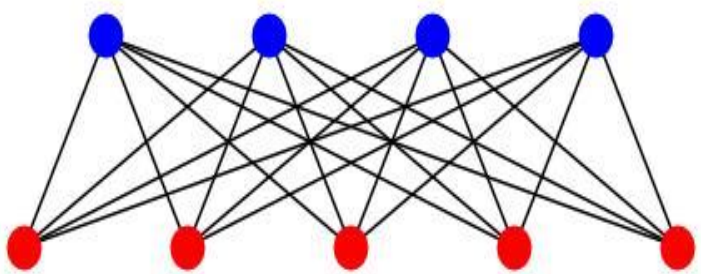
G



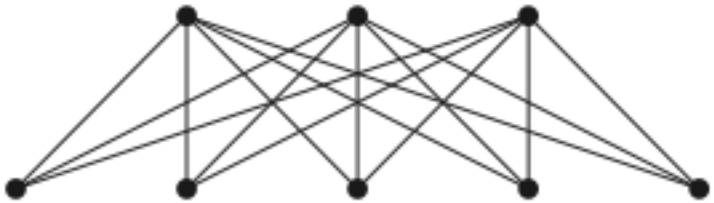
H

Bipartite graph

- A simple graph is bipartite if and only if it is possible to assign **1 of 2 different colors** to each vertex of the graph so that no two adjacent vertices are assigned the same color.



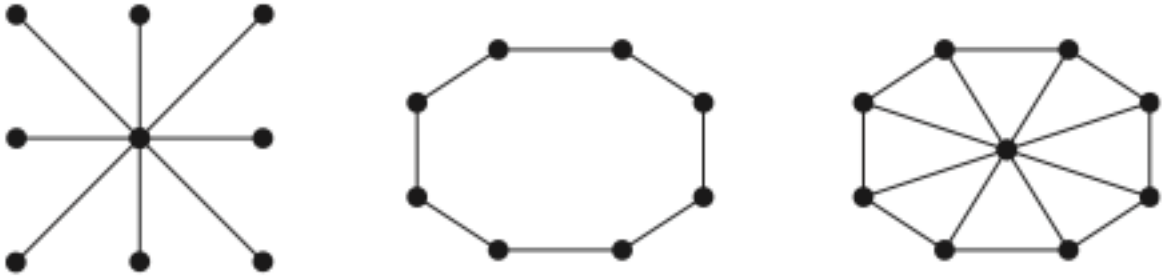
- A **complete bipartite graph** Km,n is a graph that has its vertex set partitioned into two subsets of m and n vertices, respectively with an edge between two vertices if and only if one vertex is in the first subset and the other vertex is in the second subset.



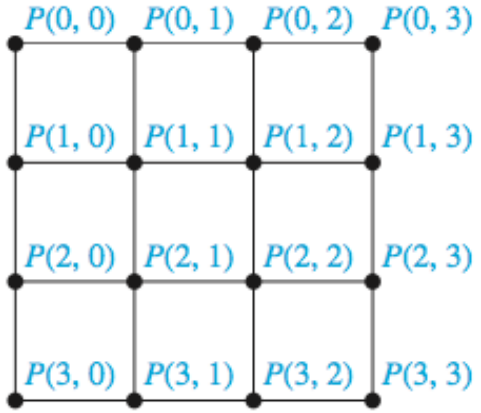
$K_{3,5}$

Applications of special types of graph

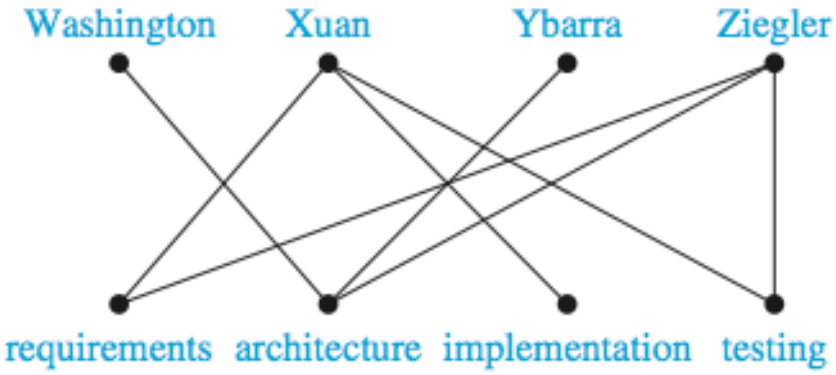
- Network topologies



- Network interconnection in parallel computing

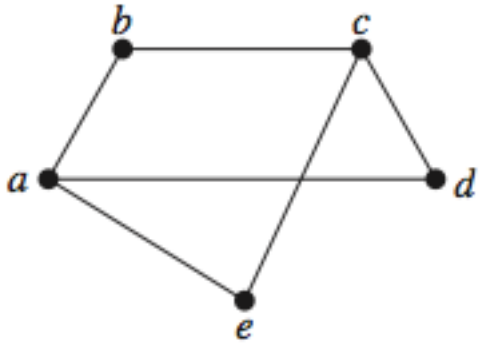


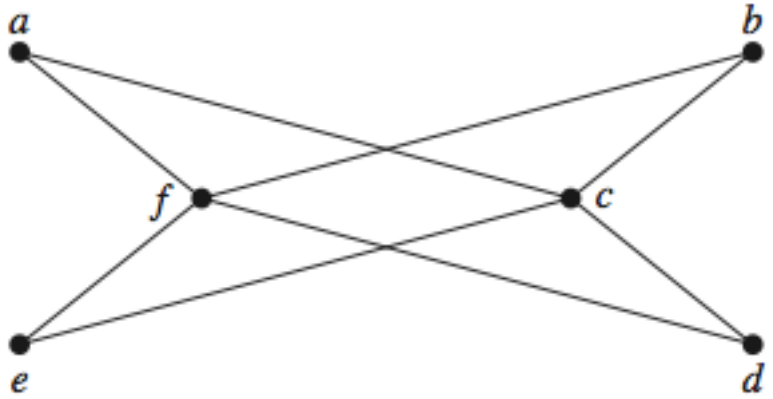
- Matching problems



Exercises

- Which of these graphs is bipartite?





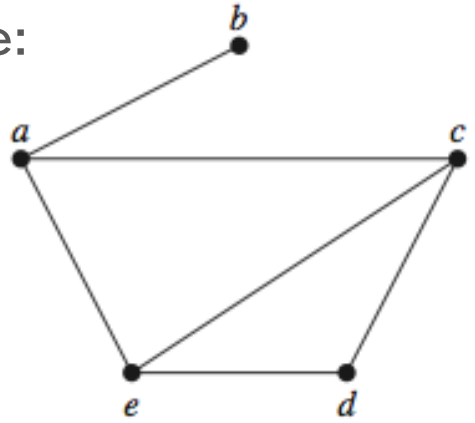
- Draw these graphs.
  - K_7
  - $K_{1,8}$
  - $K_{4,4}$
  - C_7
  - W_7

TOPICS

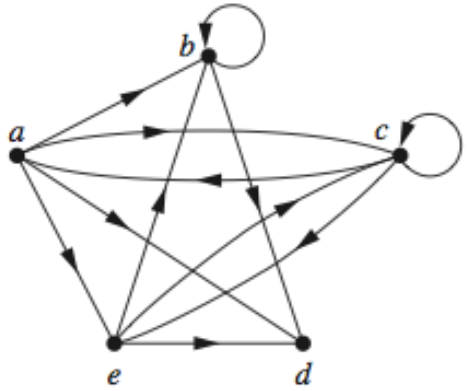
- Graphs and Graph Models
- Graph Terminology
- Special Types of Graphs
- **Representing Graphs**
- Connectivity
- Euler and Hamilton Paths
- Shortest-Path Problems

Using adjacency list

- Specify vertices that are adjacent to each vertex of the graph
- Example:



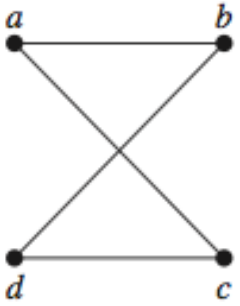
Vertex	Adjacent Vertices
a	b, c, e
b	a
c	a, d, e
d	c, e
e	a, c, d

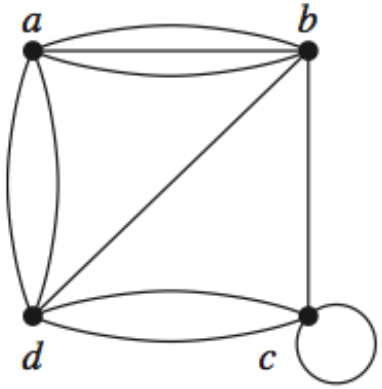


Initial Vertex	Terminal Vertices
a	b, c, d, e
b	b, d
c	a, c, e
d	
e	b, c, d

Using adjacency matrix

- Let G be a pseudograph with vertices $v_1, v_2, \ldots, v_n$. We can represent G by a square matrix $[a_{ij}]$ of order n, whose entries are determined as follows: a_{ij} = the number of edges in G connecting v_i and v_j



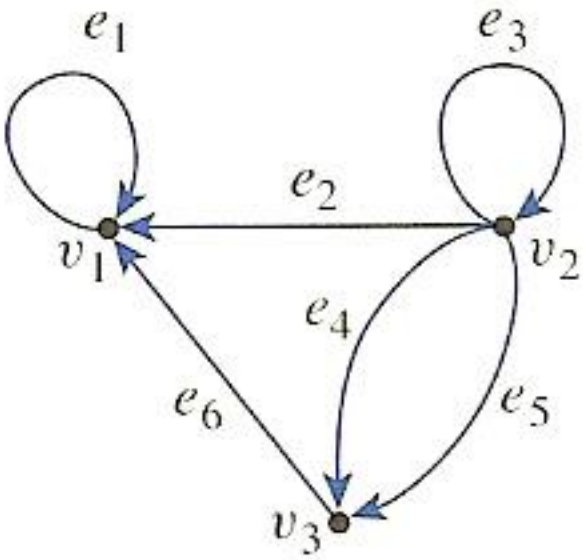


$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 3 & 0 & 2 \\ 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 2 & 1 & 2 & 0 \end{bmatrix}$$

Using adjacency matrix

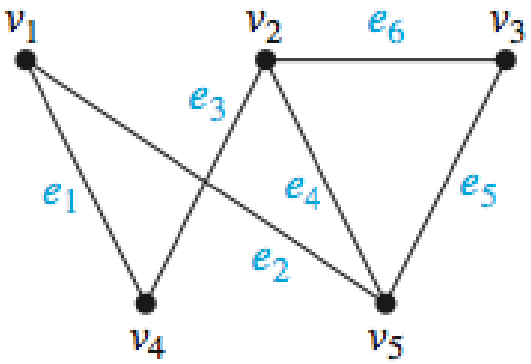
- Let G be a directed graph with vertices $v_1, v_2, \ldots, v_n$. We can represent G by a square matrix $[a_{ij}]$ of order n, whose entries are determined as follows: a_{ij} = the number of edges in G whose initial vertex is v_i and whose end vertex is v_j



$$\mathbf{A} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 2 \\ 1 & 0 & 0 \end{bmatrix} \end{array}$$

Using incidence matrix

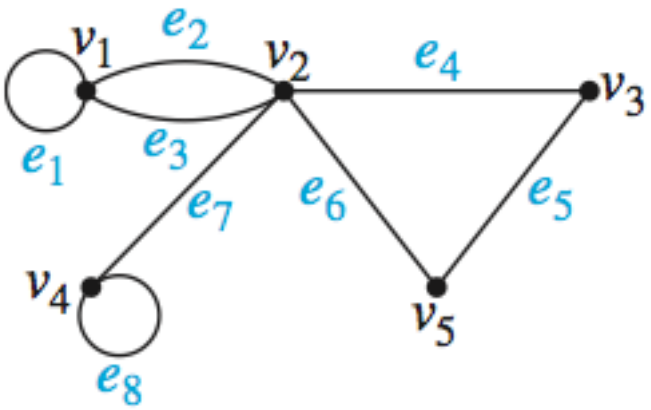
- Let G be a pseudograph with vertices $v_1, v_2, \ldots, v_n$ and edges $e_1, e_2, \ldots, e_m$. We can represent G by an incident matrix $[a_{ij}]$ of size $n \times m$, whose entries are determined as follows:
  - $a_{ij} = 1$ if e_j is incident to v_i,
  - and $a_{ij} = 0$ otherwise.



$$
\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array}
\begin{array}{c}
\begin{array}{cccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{array} \\
\left[\begin{array}{cccccc}
1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 0
\end{array}\right]
\end{array}.
$$

Using incidence matrix

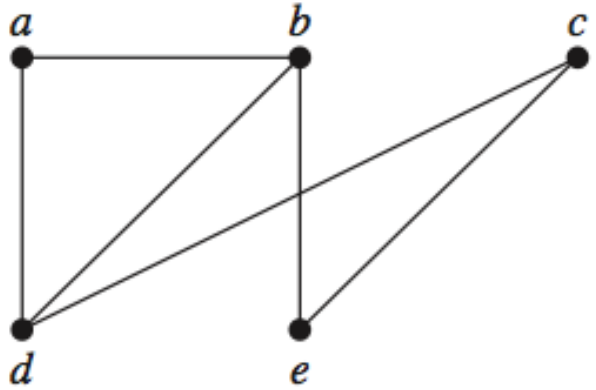
- Multiple edges: two identical columns
- Loop: exactly one entry is 1



	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
v_1	1	1	1	0	0	0	0	0
v_2	0	1	1	1	0	1	1	0
v_3	0	0	0	1	1	0	0	0
v_4	0	0	0	0	0	0	1	1
v_5	0	0	0	0	1	1	0	0

Exercises

- Use adjacency list, adjacency matrix and incidence matrix to represent the given graph



- Draw an undirected graph with the given adjacency matrix then represent it again by adjacency list and incidence matrix

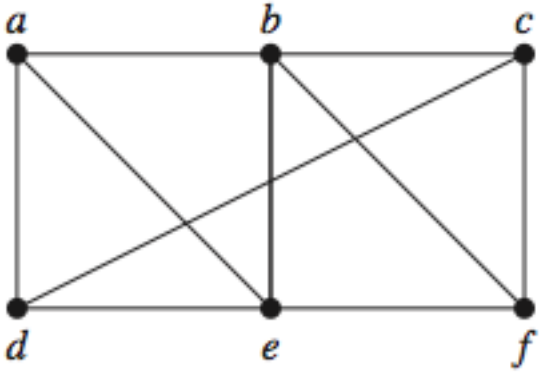
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

TOPICS

- Graphs and Graph Models
- Graph Terminology
- Special Types of Graphs
- Representing Graphs
- **Connectivity**
- Euler and Hamilton Paths
- Shortest-Path Problems

Path

- Let G be an undirected graph. A **path** of **length n** from u to v is a sequence of edges $e_0 = x_0x_1$, $e_1 = x_1x_2$, …, $e_n = x_{n-1}x_n$, where $x_0 = u$, $x_n = v$.
  - If u = v and n > 0, a path is called a **circuit**
  - A path / circuit is called **simple** if it doesn't contain the same edge more than once.



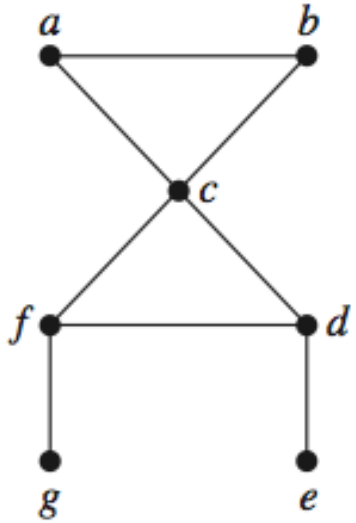
Path of length 4: a, b, e, f, c
Not a path: a, d, c, e
Circuit of length 4: b, c, d, e, b
Not a simple path: a, b, f, c, d, a, b, f

- Same definition for directed graph

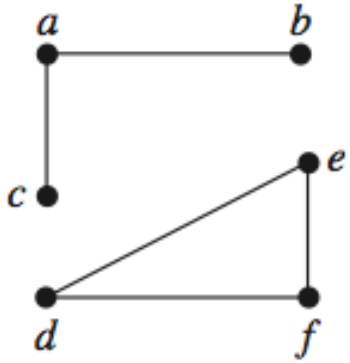
Connectedness in undirected graph

- An undirected graph is **connected** if there is a path between **any pair** of **distinct** vertices.

- An undirected graph that is **not connected** is called **disconnected**



G_1



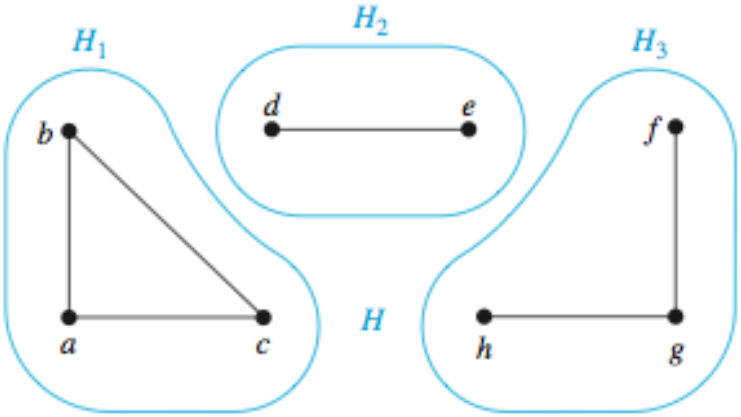
G_2

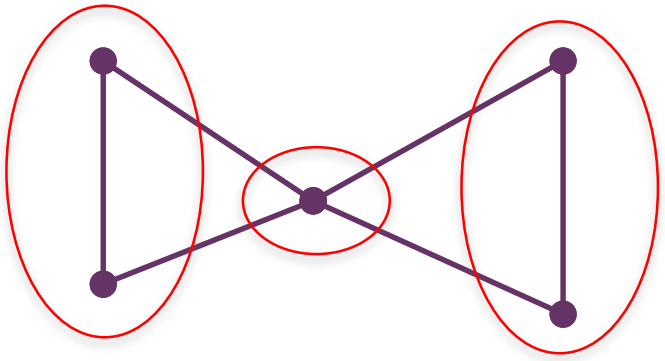
G_1 is connected

G_2 is disconnected

Connectedness in undirected graph

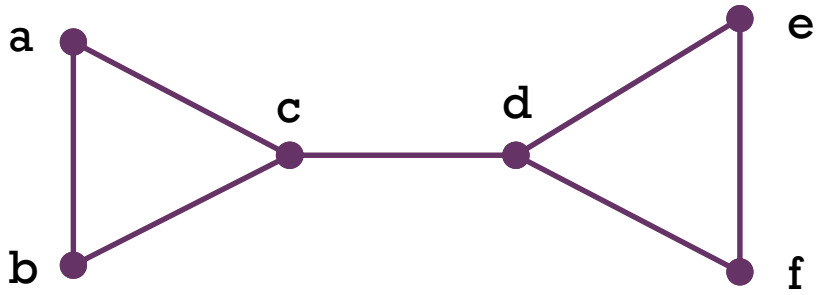
- A **connected component** of an undirected graph is a maximal subgraph that is connected.
- A **cut vertex**, or an **articulation point**, is a vertex that if we remove it and all the edges incident with it we will obtain a subgraph having more connected components than the original graph





Connectedness in undirected graph

- A subset V' of the vertex set V of G = (V, E) is a **vertex cut**, or **separating set**, if G − V' is disconnected.
- **Vertex connectivity** of a graph G, denoted by κ(G), is the minimum number of vertices in a vertex cut.
  - $0 \leq \kappa(G) \leq n - 1$
- A graph is ***k*-connected** (or ***k*-vertex-connected**), if $\kappa(G) \geq k$

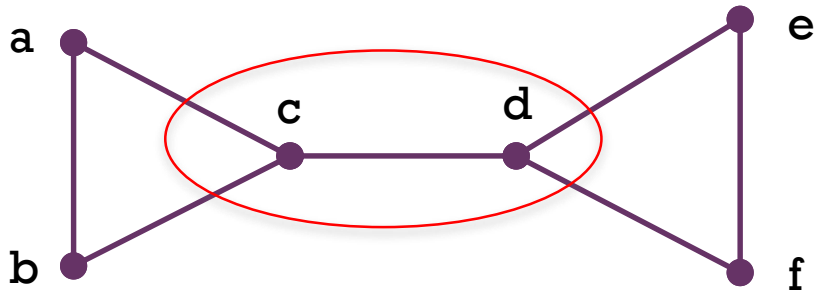


vertex cut examples: {c}, {d, e}
κ(G) = 1
G is 1-connected

Connectedness in undirected graph

- A **cut edge**, or a **bridge**, is an edge that if we remove it we will obtain a subgraph having more connected components than the original graph
- A set of edges E' is called an **edge cut** of G if the subgraph G − E' is disconnected
- The **edge connectivity** of a graph G, denoted by λ(G), is the minimum number of edges in an edge cut of G
  - $0 \leq \lambda(G) \leq n - 1$

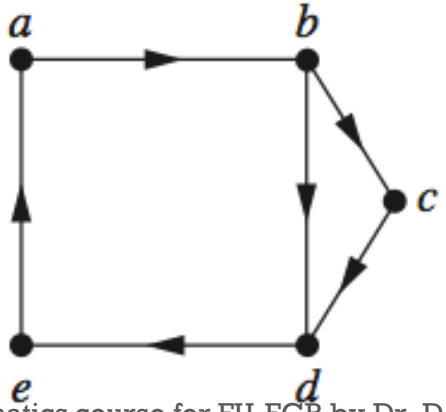
cut edge example: cd
edge cut examples: {cd}, {ac, bc}
λ(G) = 1

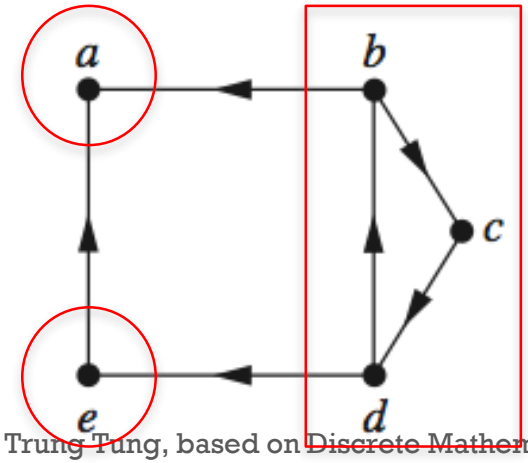


Connectedness in directed graph

- A directed graph is **strongly connected** if for all pairs of vertices u and v there is a path from u to v and vice versa.
- A directed graph is **weakly connected** if there is a path between every two vertices in the underlying undirected graph.
- A **strongly connected component** of a directed graph G is a maximal subgraph of G that is strongly connected.

strongly connected

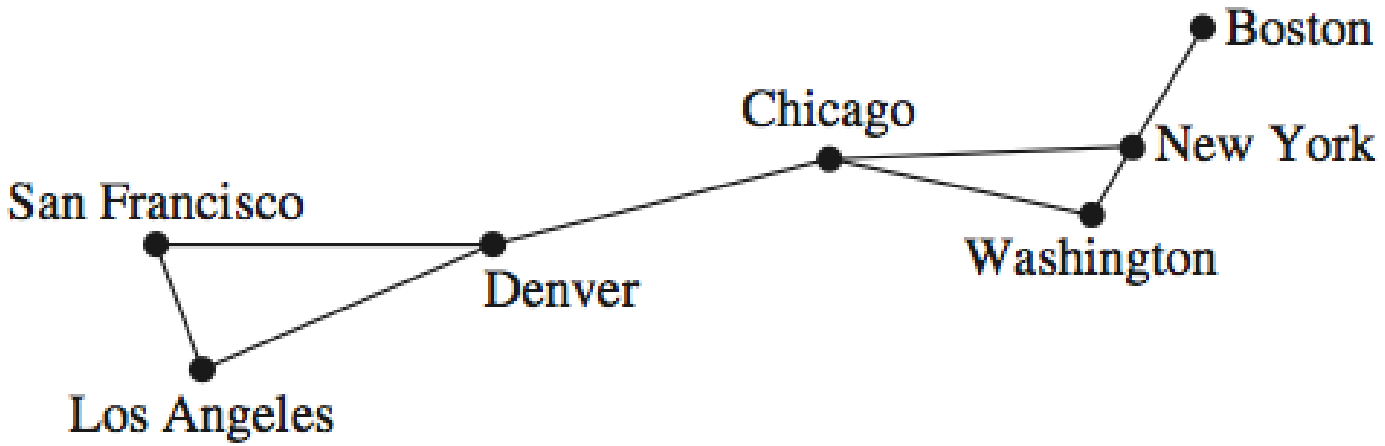




weakly connected

Exercises

- Find some cut vertices, vertex cuts, cut edges, edge cuts. Then what is $\kappa(G)$ and $\lambda(G)$?

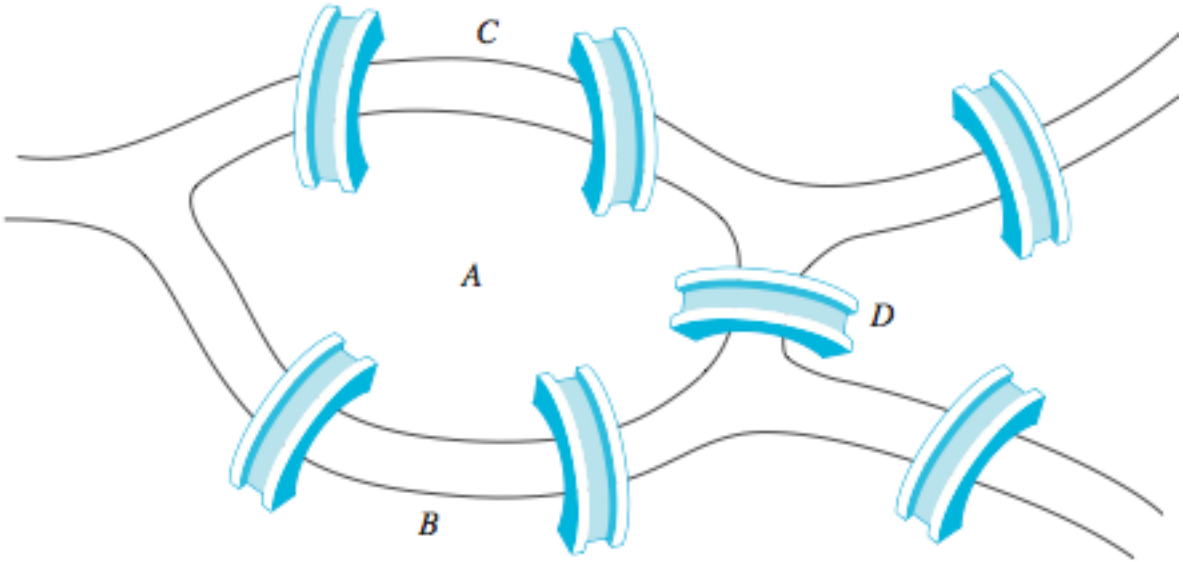


TOPICS

- Graphs and Graph Models
- Graph Terminology
- Special Types of Graphs
- Representing Graphs
- Connectivity
- **Euler and Hamilton Paths**
- Shortest-Path Problems

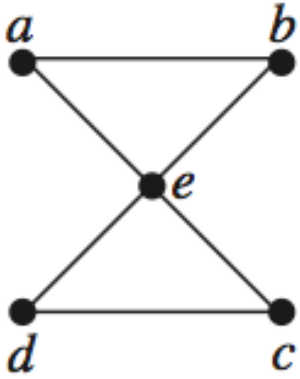
Euler problem

- Is this possible to start at some location, travel across all bridges without crossing any bridge twice, then return to the starting point?

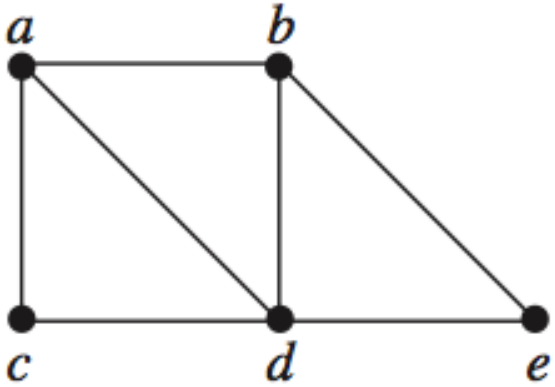


Definitions

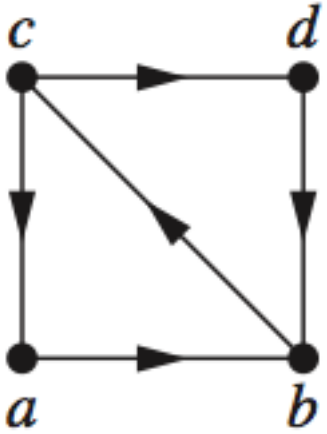
- A simple circuit containing all edges of a graph is called **Euler circuit**.
- A simple path containing all edges of a graph is called **Euler path**.



Euler circuit: a, e, c, d, e, b, a



Euler path: a, c, d, e, b, d, a, b

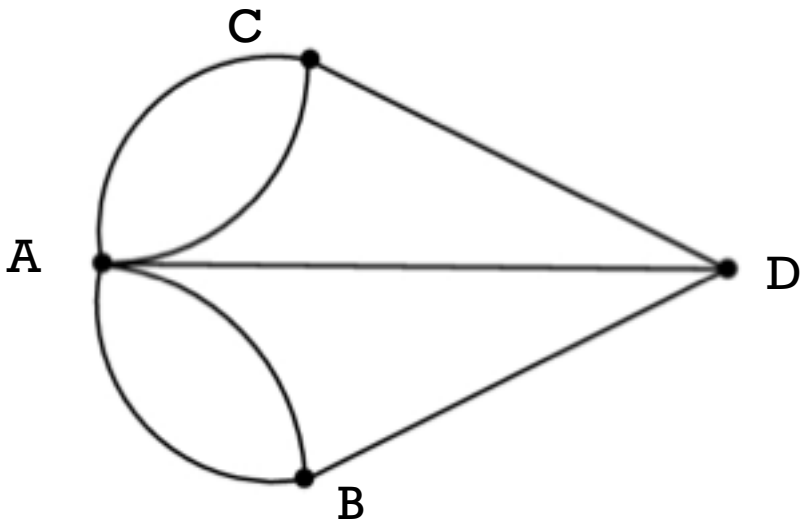


Euler path or circuit?

Conditions for existence of Euler path and circuit

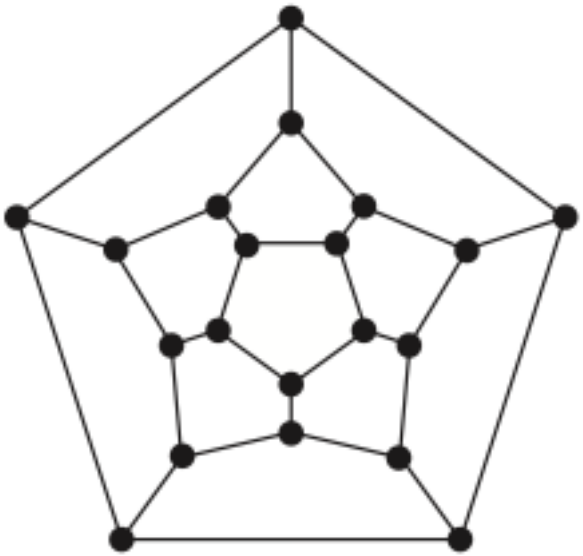
- A connected multigraph G has Euler circuits if and only if every vertex has even degree.
- If G does not have Euler circuits, then it has Euler paths if and only if it has exactly two vertices of odd degrees

7 bridges as a graph



Hamilton problem

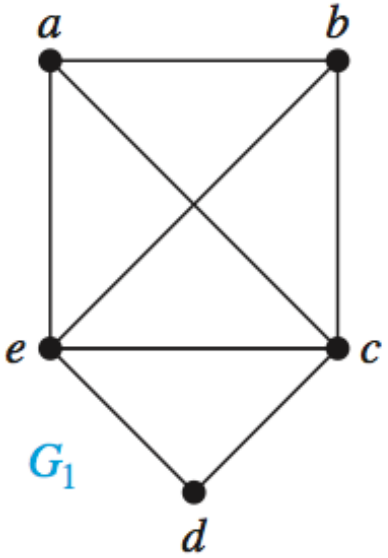
- Given a map of cities where two cities are connected if we can travel between them. Start at one city, can we travel all the cities, visiting each city exactly once the back to the first city?

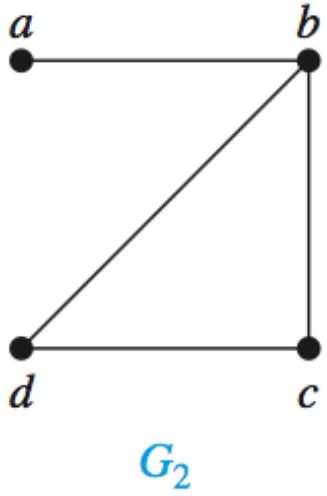


Definitions

- A simple path that passes through all vertices exactly once is called **Hamilton path**.
- A simple circuit that passes through all vertices exactly once is called **Hamilton circuit**.

Hamilton circuit:
a, b, c, d, e, a

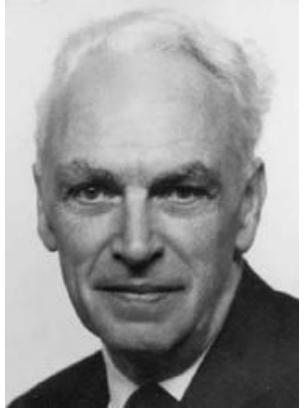




Hamilton path:
a, b, c, d

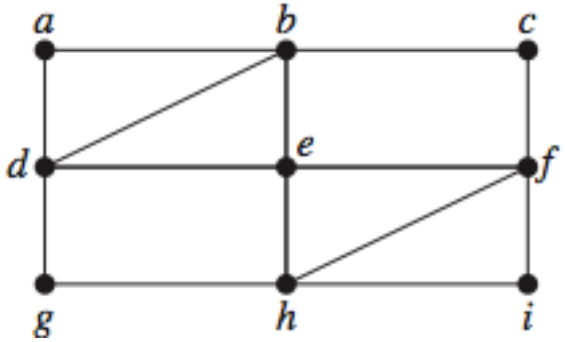
Conditions for existence of Hamilton path and circuit

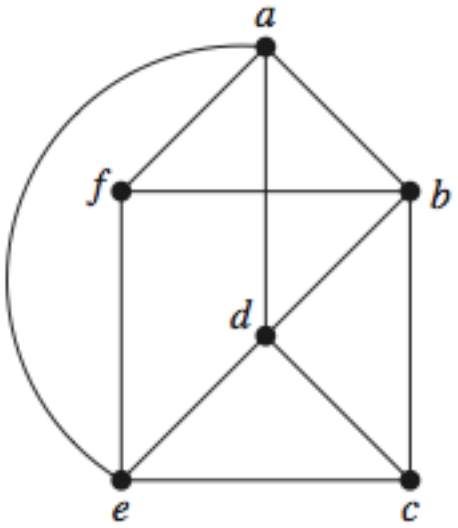
- A graph with a vertex of degree one cannot have a Hamilton circuit
- **Dirac's theorem**: If G is a simple graph with n vertices with n ≥ 3 such that the degree of every vertex in G is at least n/2, then G has a Hamilton circuit.
- **Ore's theorem**: If G is a simple graph with n vertices with n ≥ 3 such that deg(u) + deg(*v*) ≥ n for every pair of nonadjacent vertices u and *v* in G, then G has a Hamilton circuit.

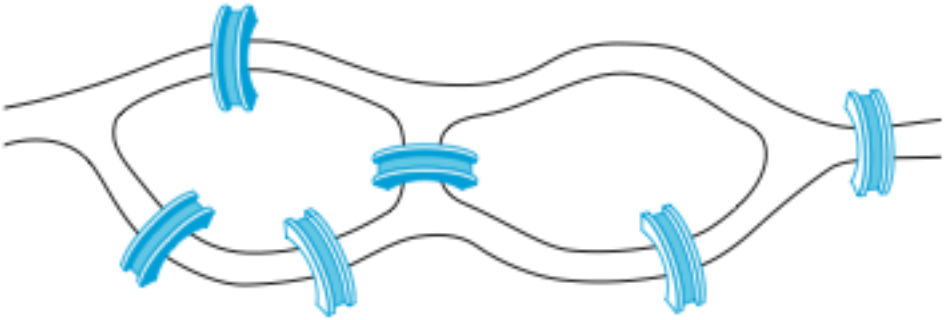


Exercises

- Determine whether the following graphs has an Euler circuit or path? If yes, construct these circuits / paths

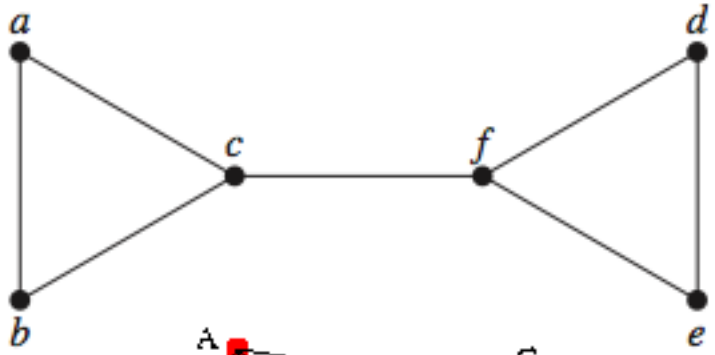


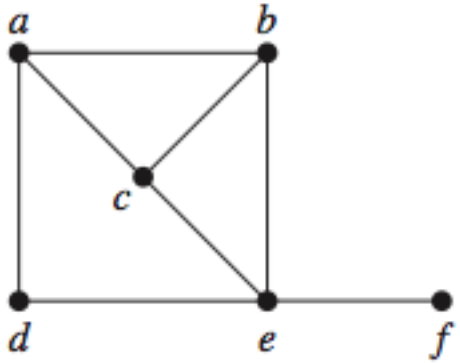


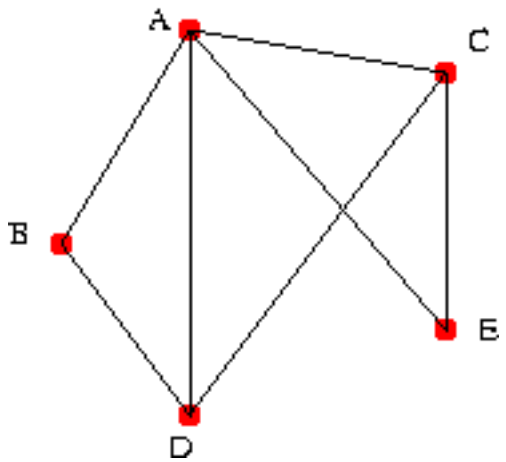


Exercises

- Determine whether the given graph has a Hamilton circuit / path?





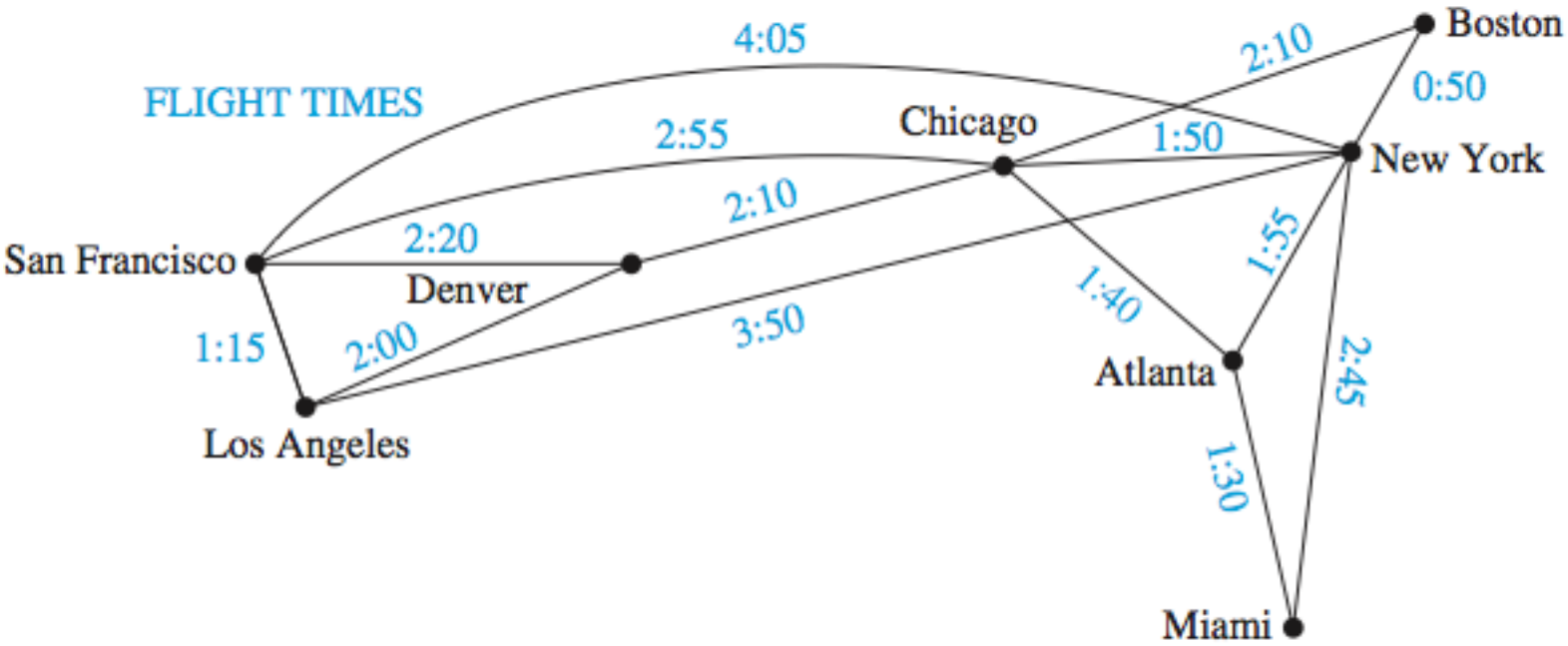


TOPICS

- Graphs and Graph Models
- Graph Terminology
- Special Types of Graphs
- Representing Graphs
- Connectivity
- Euler and Hamilton Paths
- **Shortest-Path Problems**

Weighted graph

- A graph that has a number assigned to each edge is called a **weighted graph**. The **length** of a path in a weighted graph is the **sum of all weights** of the edges of this path.



Dijkstra's Algorithm

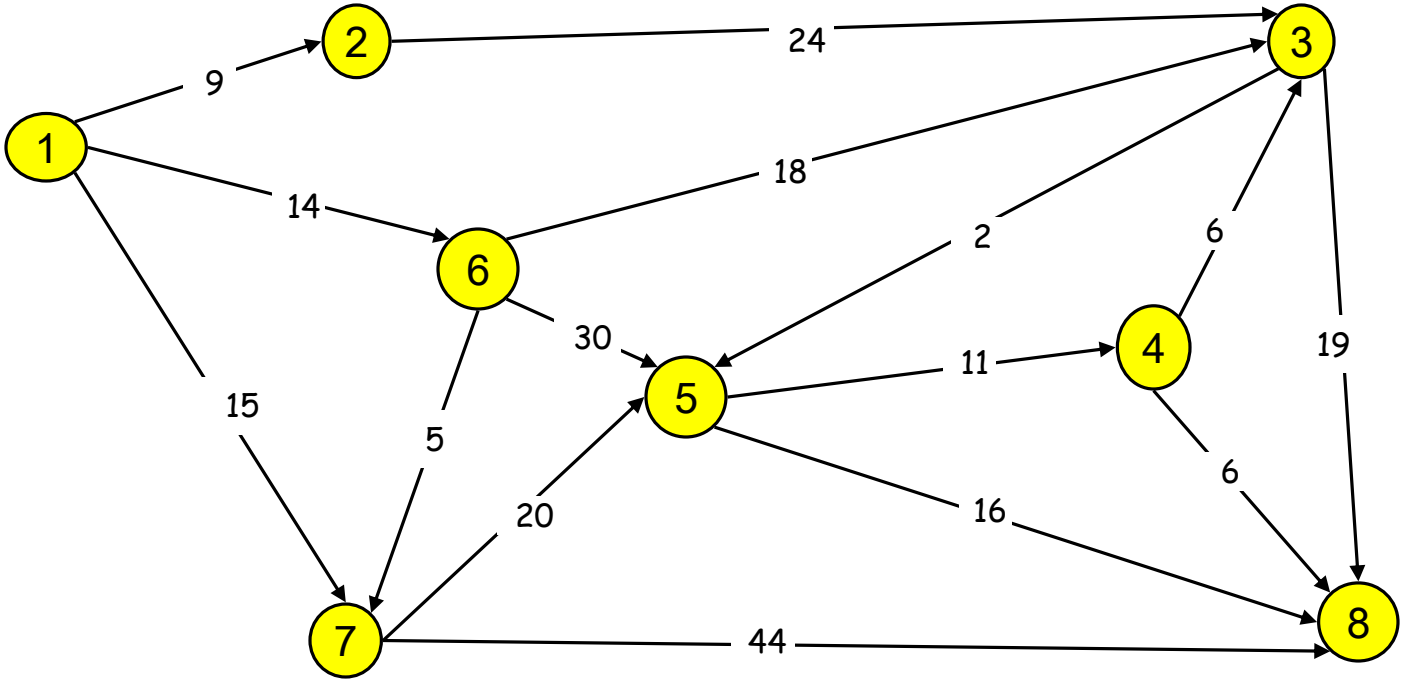
- Let G be a weighted graph. To find the shortest path between A and Z in G:
  - Finds the length of the shortest path from A to the first vertex.
  - Finds the length of the shortest path from A to the second vertex.
  - Finds the length of the shortest path from A to the third vertex.
  - ...
  - Continue the process until Z is reached.

Dijkstra's Algorithm

procedure *Dijkstra*(G: weighted connected simple graph, with all weights positive)
{G has vertices a = $v_0, v_1, \ldots, v_n$ = z and lengths $w(v_i, v_j)$ where $w(v_i, v_j) = \infty$ if $\{v_i, v_j\}$ is not an edge in G}
for i := 1 **to** n $L(v_i) := \infty$
L(a) := 0
C := ∅
{the labels are now initialized so that the label of a is 0 and all other labels are ∞, and C is the empty set}
while z ∉ C
    u := a vertex not in C with L(u) minimal
    C := C ∪ {u}
    for all vertices *v* not in C
        if L(u) + w(u, *v*) < L(*v*) **then** L(*v*) := L(u) + w(u, *v*)
{this adds a vertex to C with minimal label and updates the labels of vertices not in C}
return L(z) {L(z) = length of a shortest path from a to z}

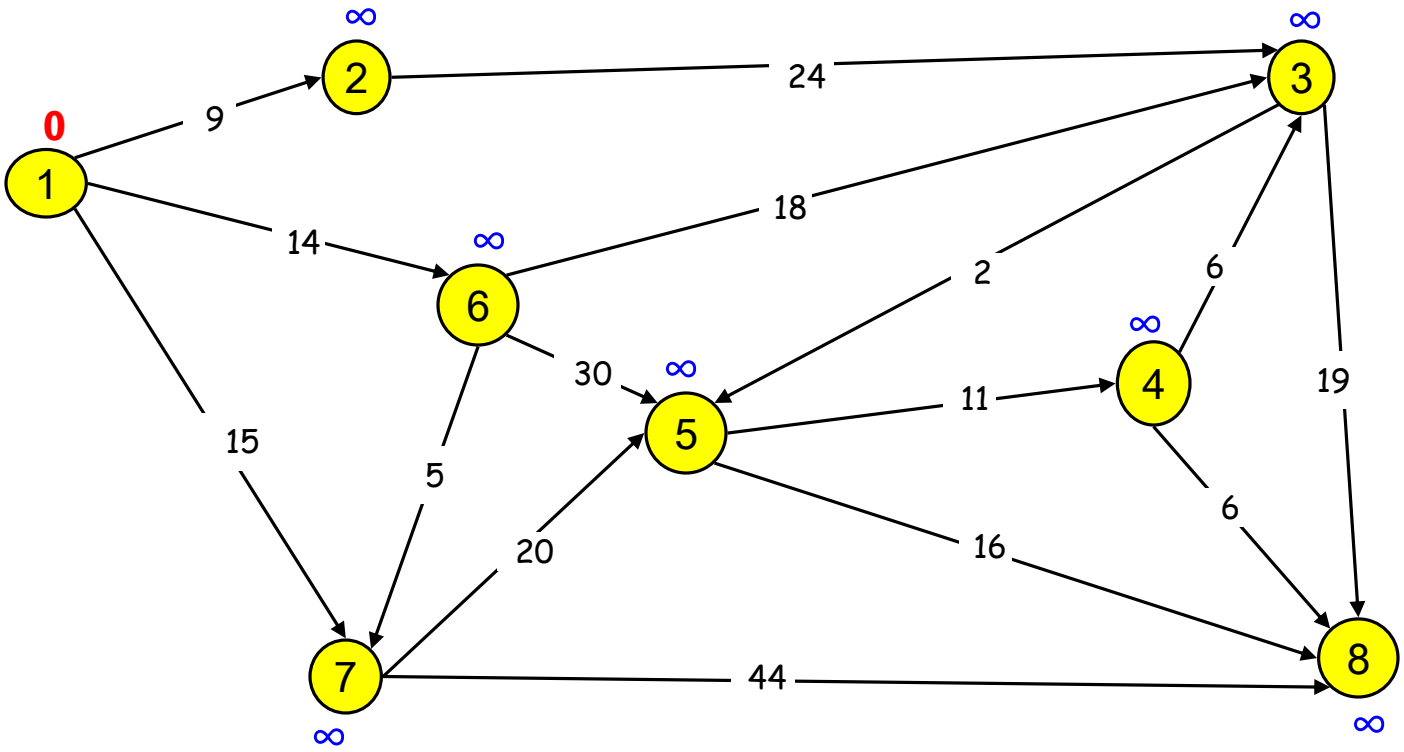
Dijkstra's Algorithm

- Example: find shortest-path from vertex 1 to other vertices



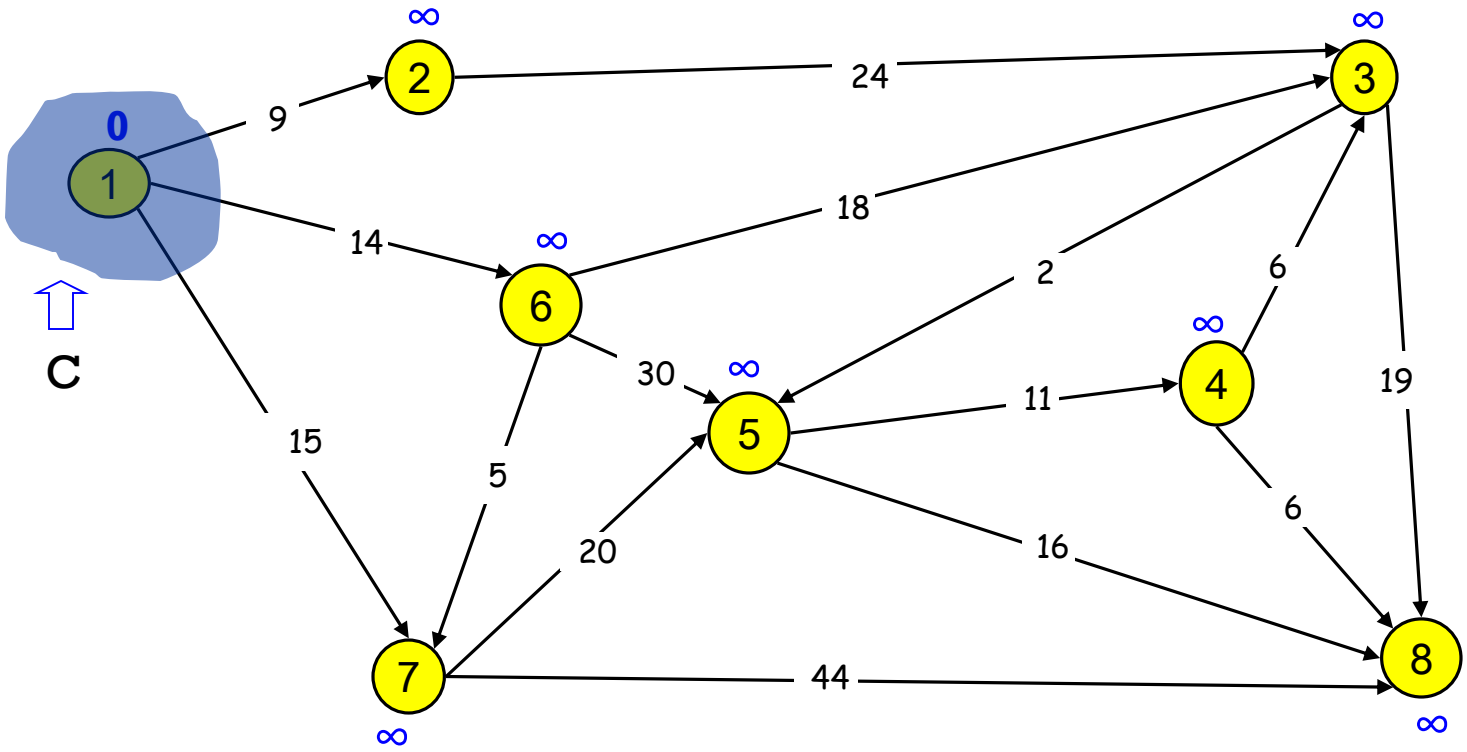
Dijkstra's Algorithm

- Initiate all distance from (1) to others = infinity



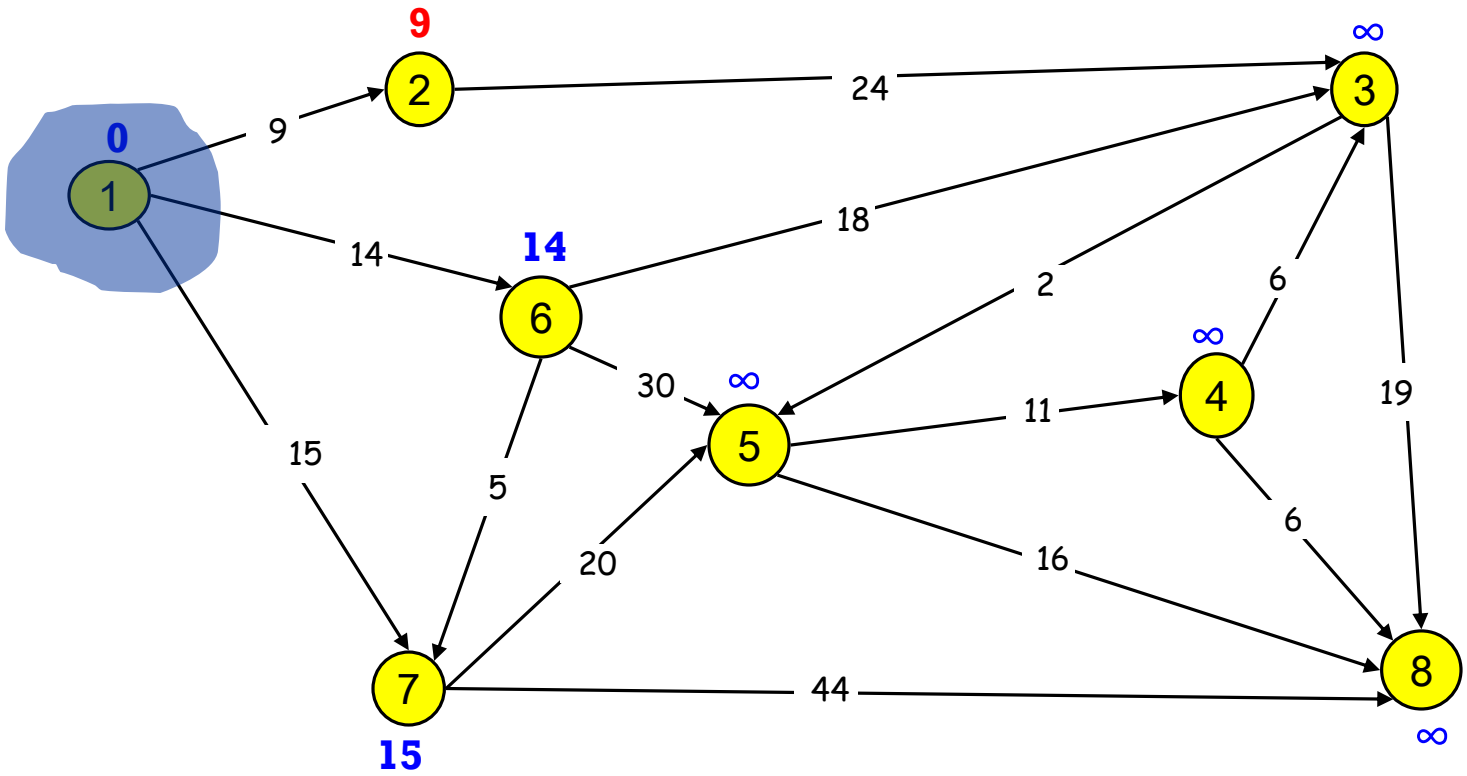
Dijkstra's Algorithm

- (0) is confirmed, put (0) into C



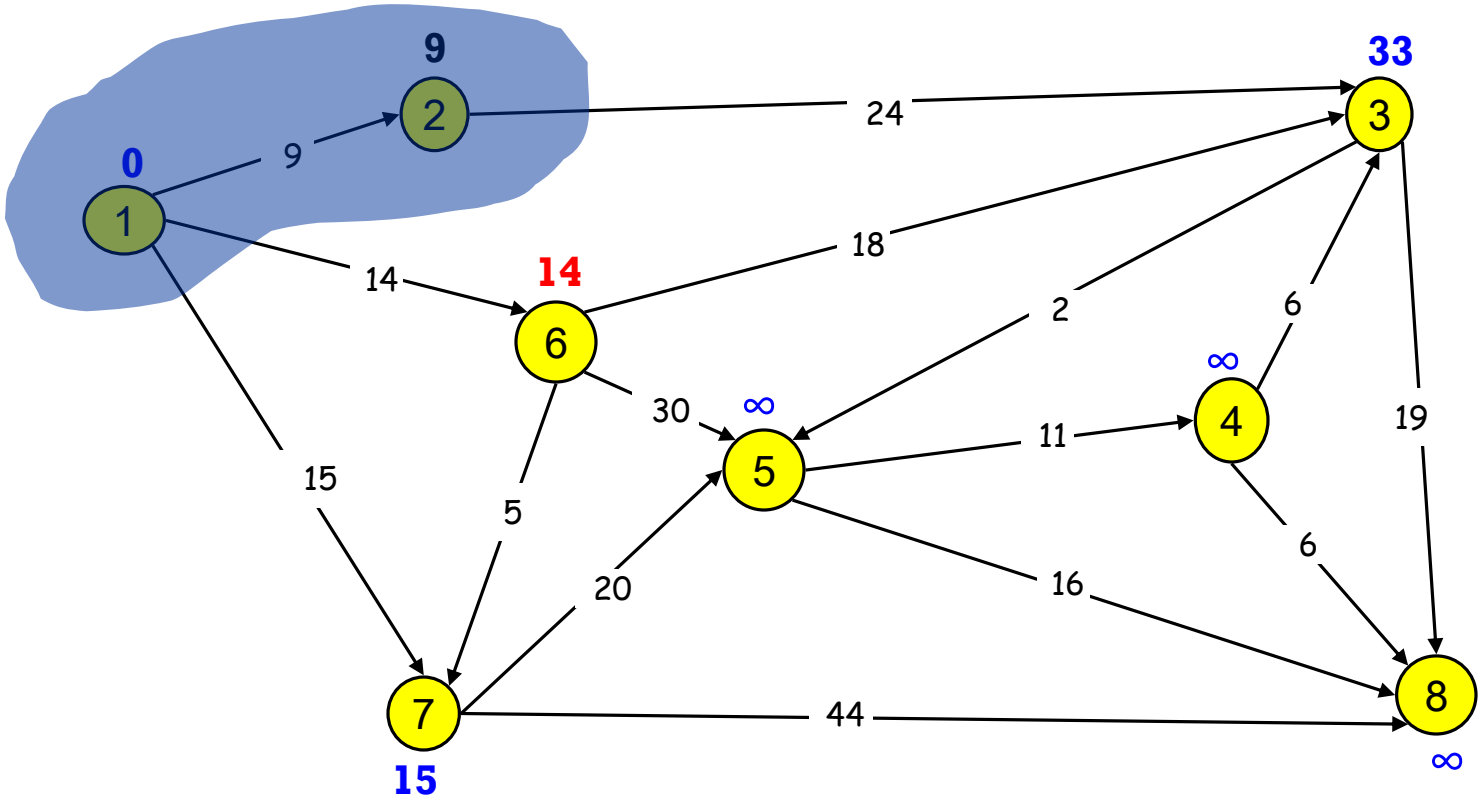
Dijkstra's Algorithm

- Update distance from (0) to (2), (6) and (7)



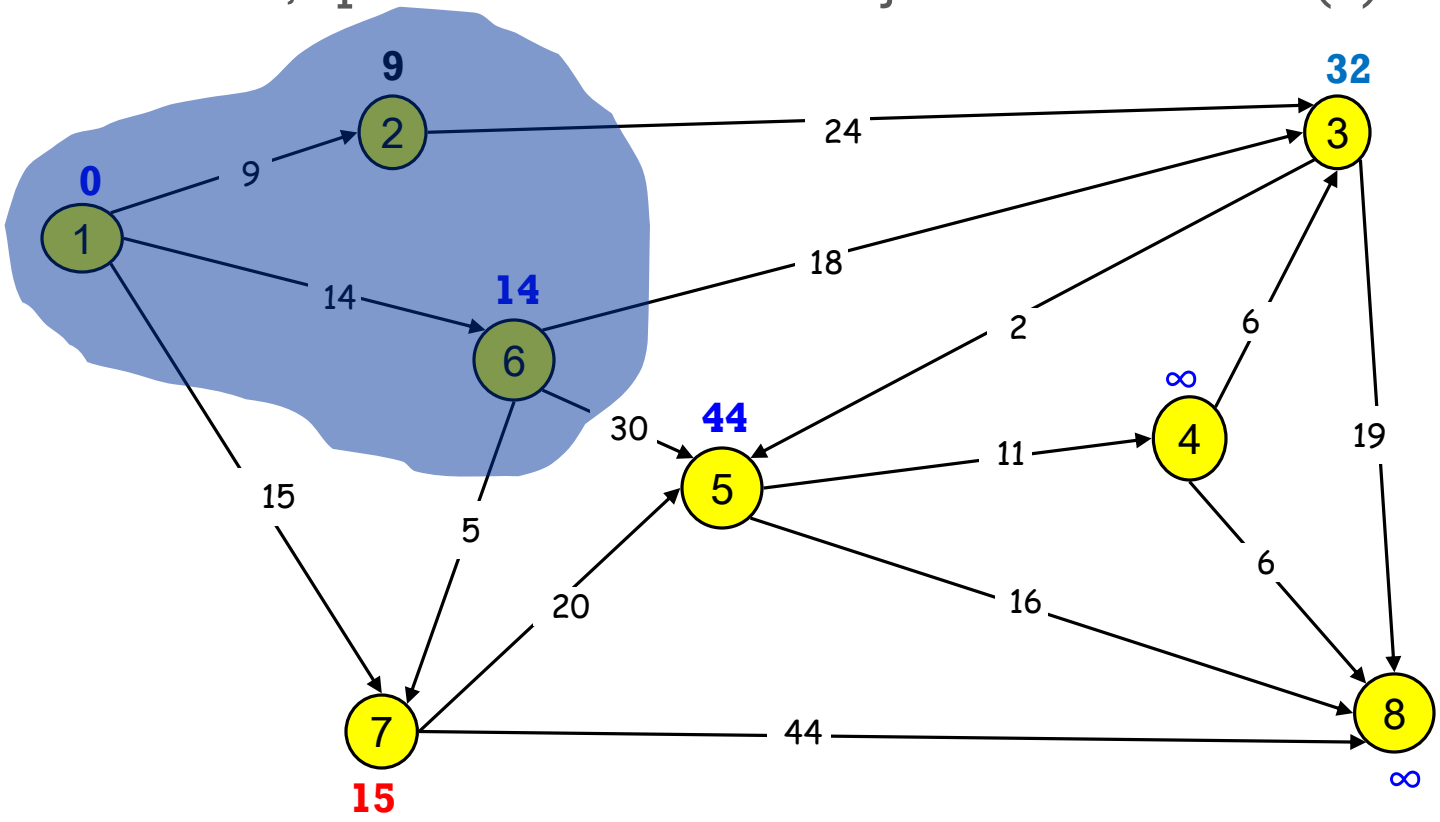
Dijkstra's Algorithm

- (2) is confirmed, update distances for adjacent vertices of (2)



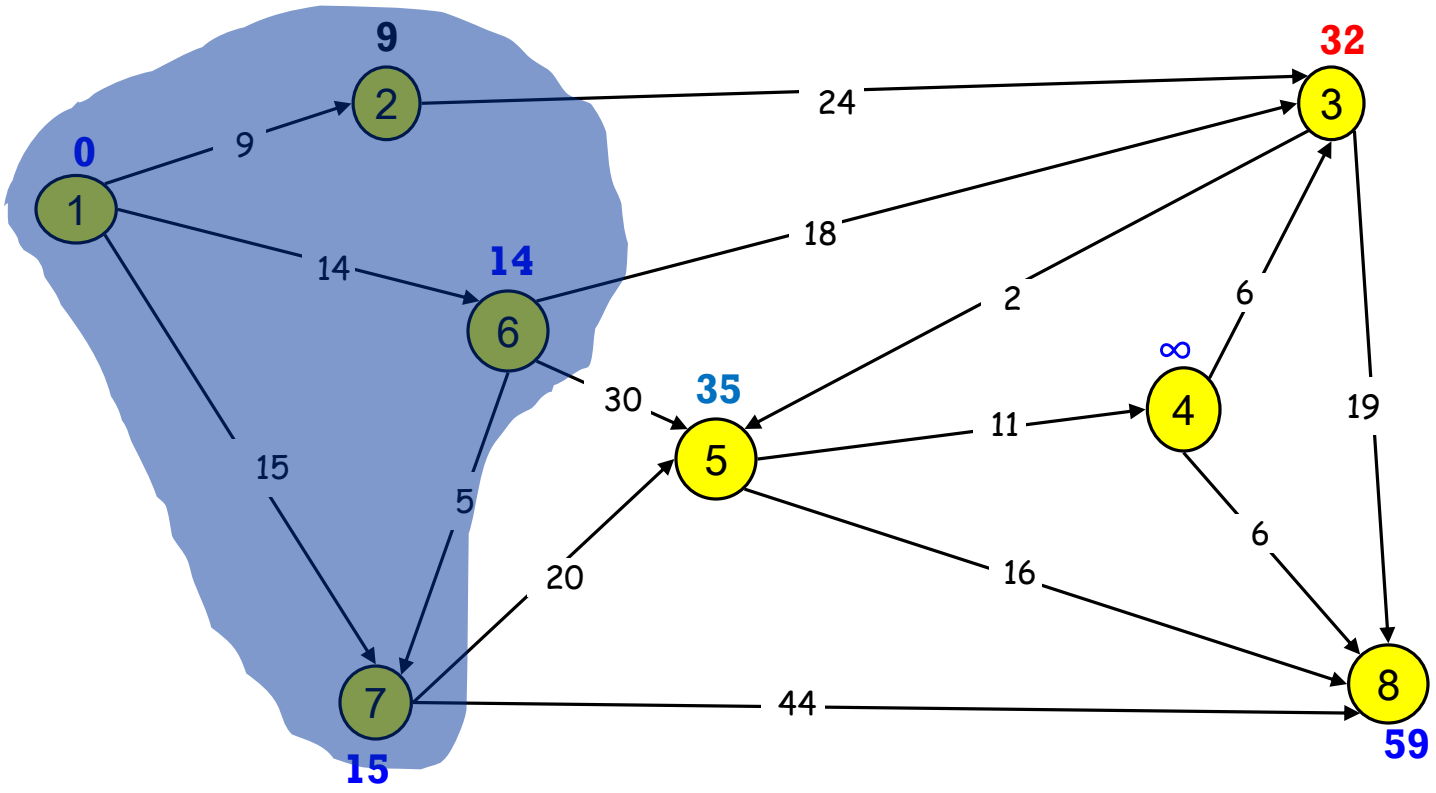
Dijkstra's Algorithm

- (6) is confirmed, update distances for adjacent vertices of (6)



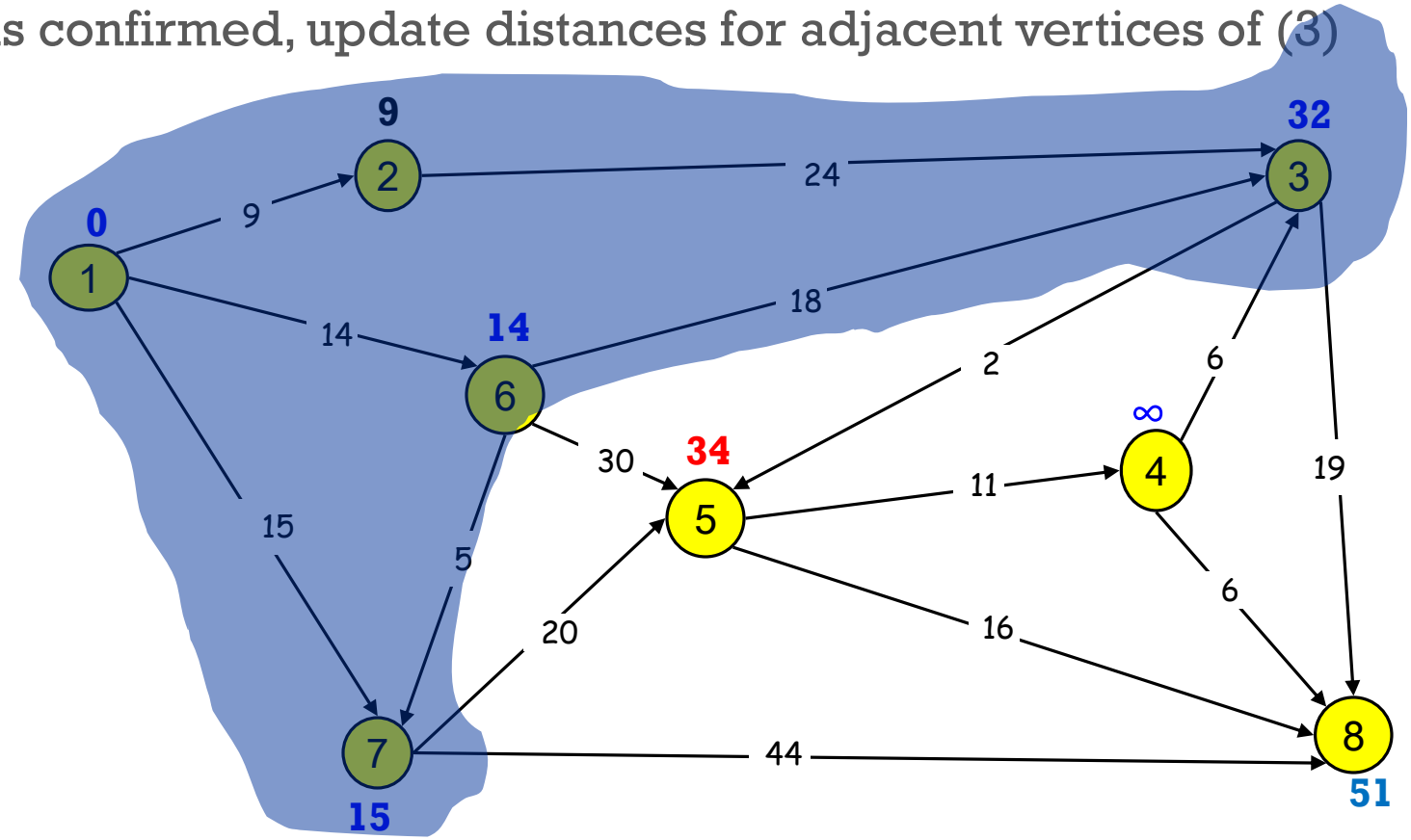
Dijkstra's Algorithm

- (7) is confirmed, update distances for adjacent vertices of (7)



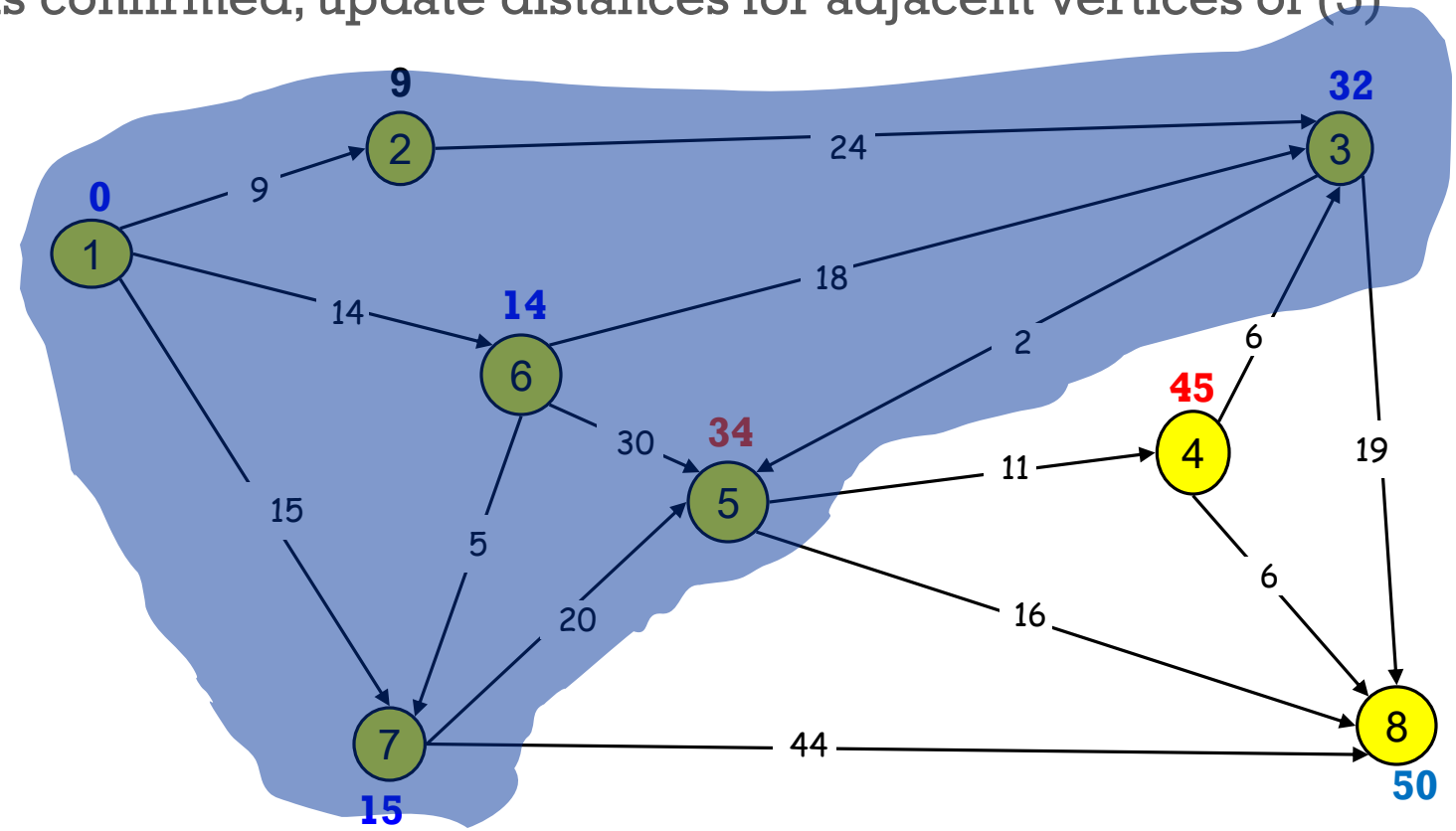
Dijkstra's Algorithm

- (3) is confirmed, update distances for adjacent vertices of (3)



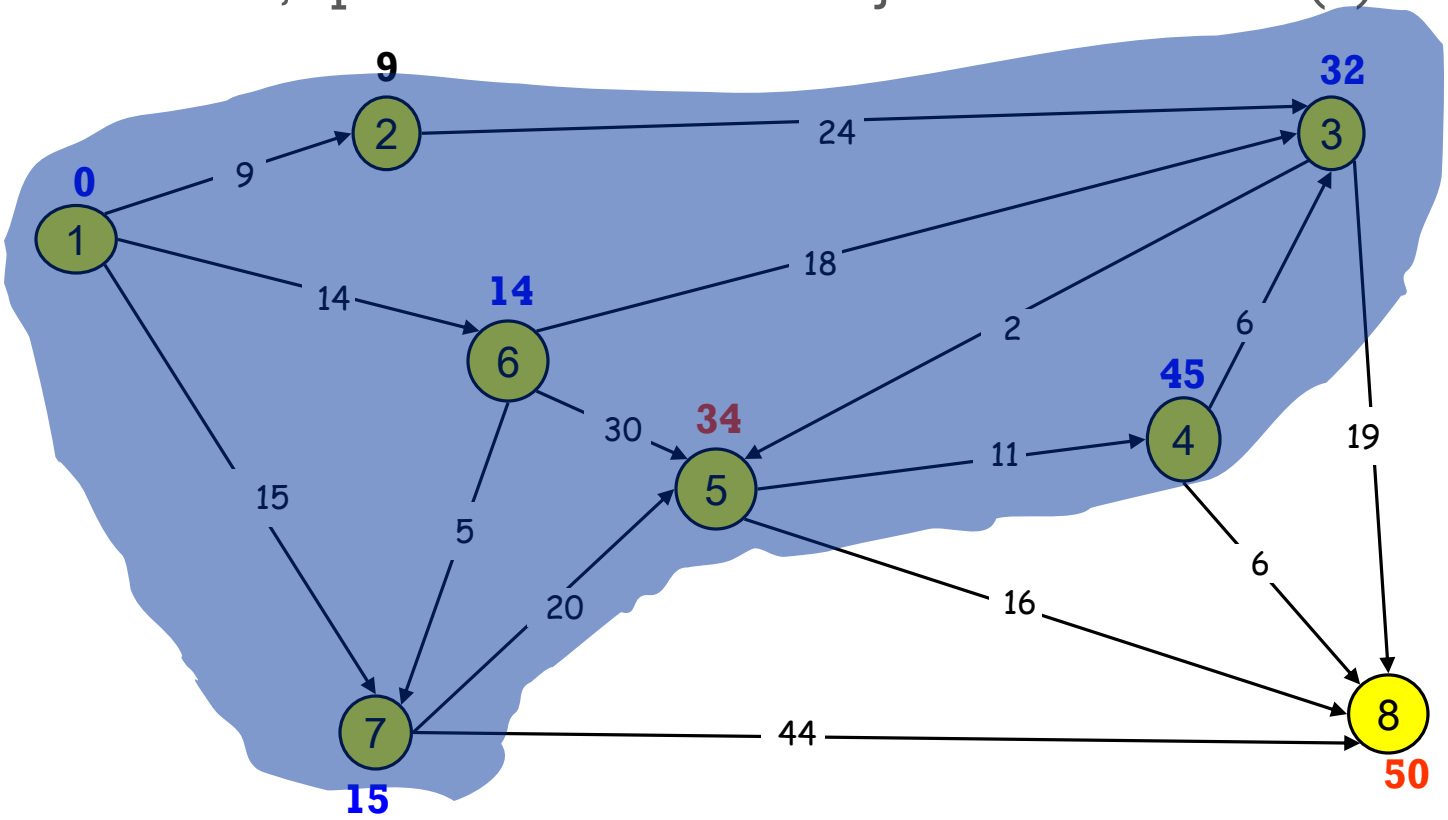
Dijkstra's Algorithm

- (5) is confirmed, update distances for adjacent vertices of (5)



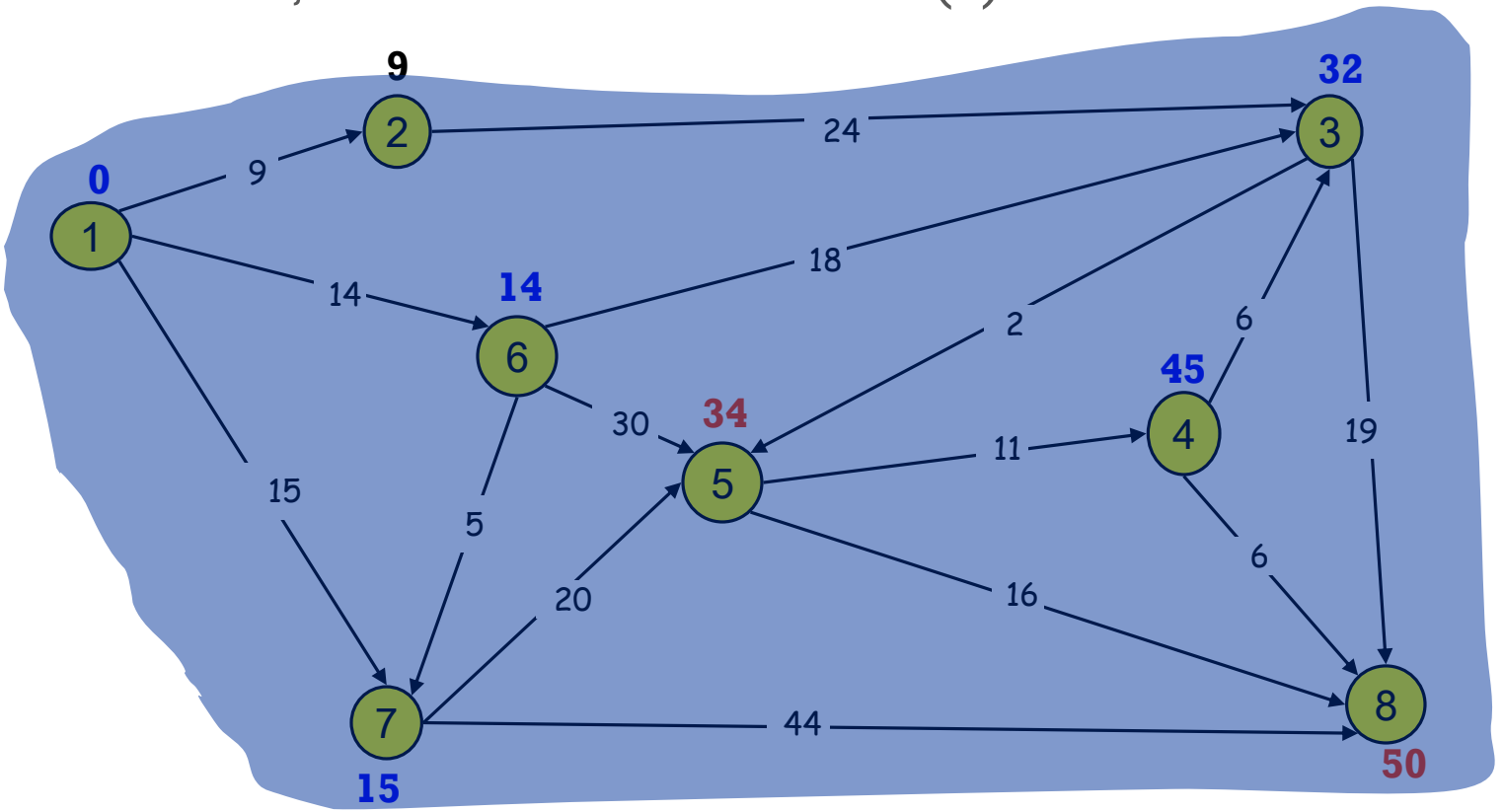
Dijkstra's Algorithm

- (4) is confirmed, update distances for adjacent vertices of (4)



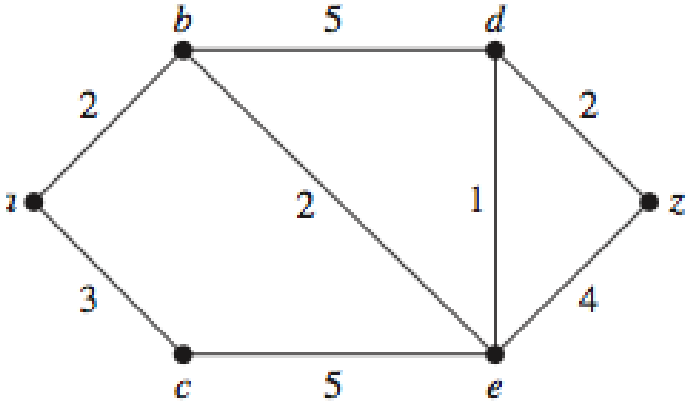
Dijkstra's Algorithm

- (8) is confirmed, all shortest distance from (1) are calculated

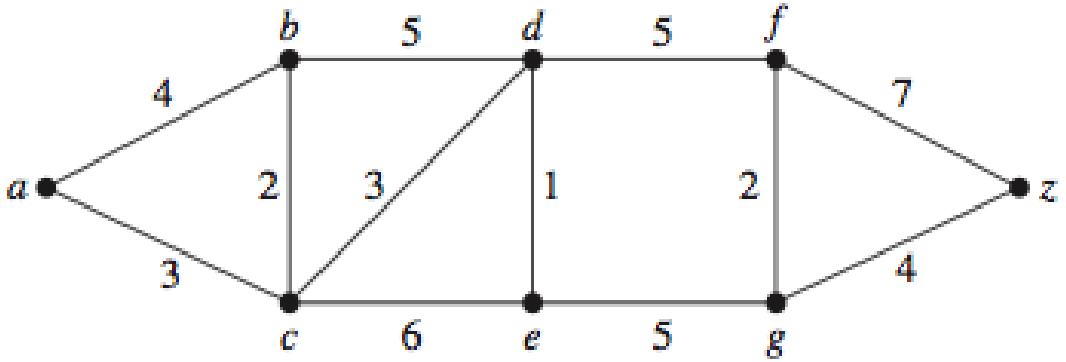


Exercises

- Find the length of a shortest path between a and z



- Find the length of a shortest path between a and g, b and z



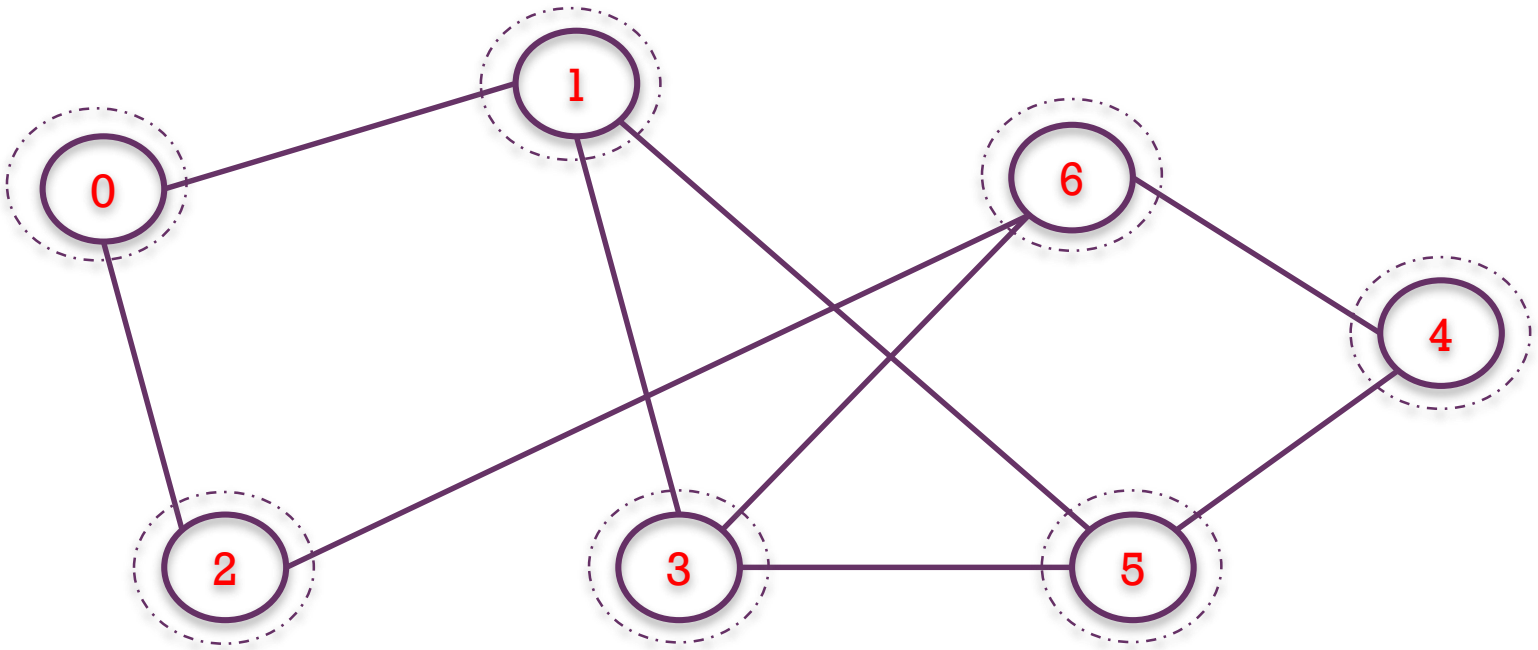
Graph traversal: BFS

- Problem: How to travel all vertices of a connected component of a graph?
- Breadth-First-Search: Start at one vertex, visit all of its 'neighbors', then for each neighbor, recursively do the same process.
- BFS can be implemented by:
  - Queue

BFS using queue

```
BFS(Graph G, start)
{
        all vertices of G are first painted white
        start vertex is painted grey and put in a queue
        while the queue is not empty
        {
                a vertex u is removed from the queue
                for all white successors v of u
                {
                        v is painted grey
                        v is added to the queue
                }
                u is painted black
        }
}
```

BFS using queue



Graph traversal: DFS

- Problem: How to travel all vertices of a connected component of a graph?
- Depth-First-Search: start at one vertex, visit one of its 'neighbors'. Then from this neighbor, recursively reach to as deep as possible other vertices through the graph. When all neighbors of a vertex are visited, go back to previous vertex and do it again.
- DFS can be implemented by:
  - Recursion
  - Stack

DFS using Stack

```
DFS(Graph G, start)
{
        all vertices of G are first painted white
        paint start grey then push start into stack
        while stack is not empty
        {
                a vertex u is popped out from the stack
                visit u and painted u black
                for all white successors v of u
                {
                        paint v grey
                        push v into stack
                }
        }
}
```

DFS using stack