CISC 235: Topic 9

Introduction to Graphs

Outline

- Graph Definition
- Terminology
- Representations
- Traversals

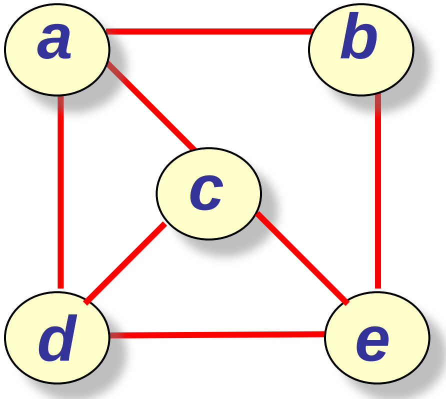
Graphs

A *graph* $G = (V, E)$ is composed of:

V: set of Vertices

E: set of edges connecting the vertices in V

An *edge* $e = (u,v)$ is a pair of vertices

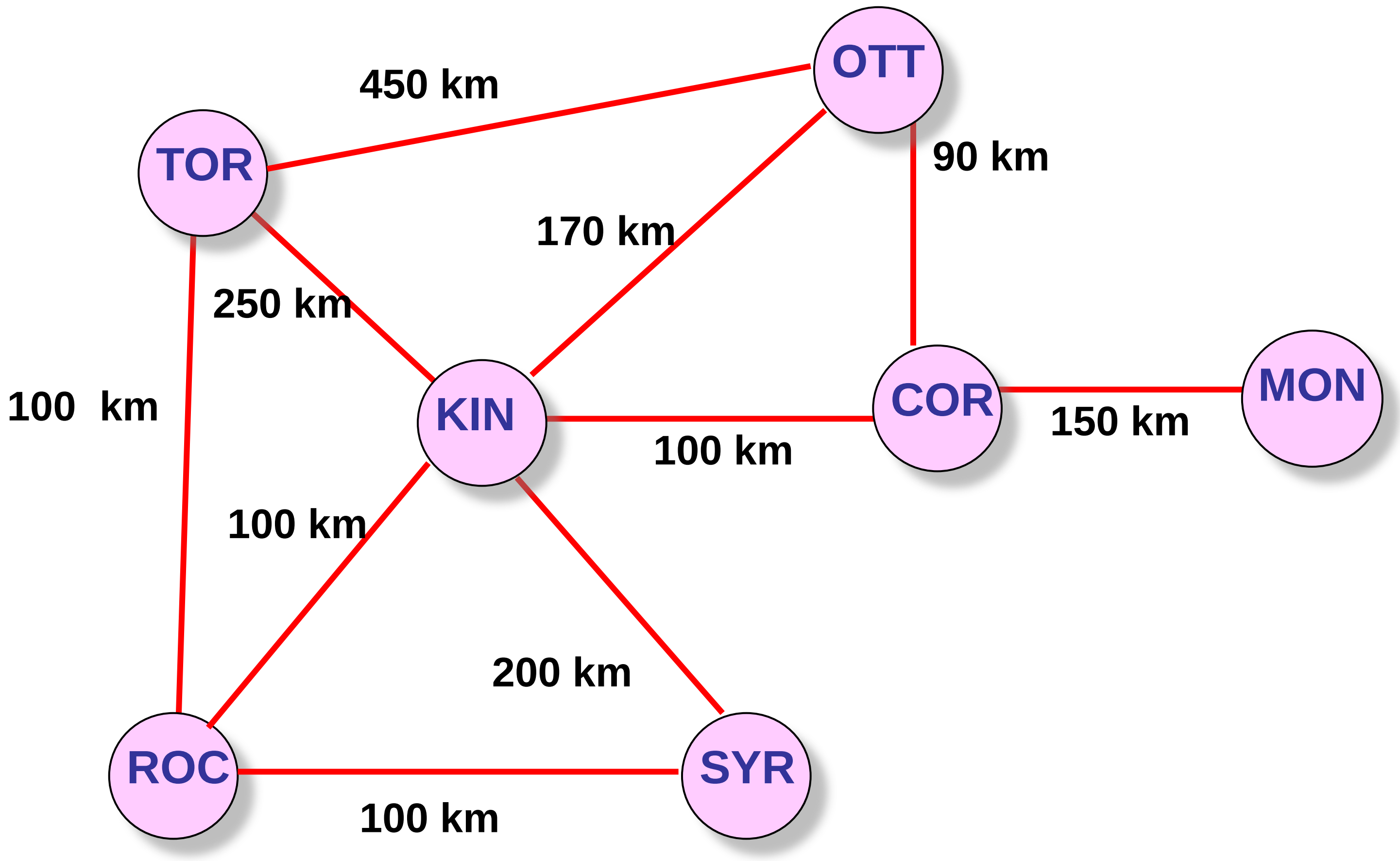


Example:

$V = \{a,b,c,d,e\}$

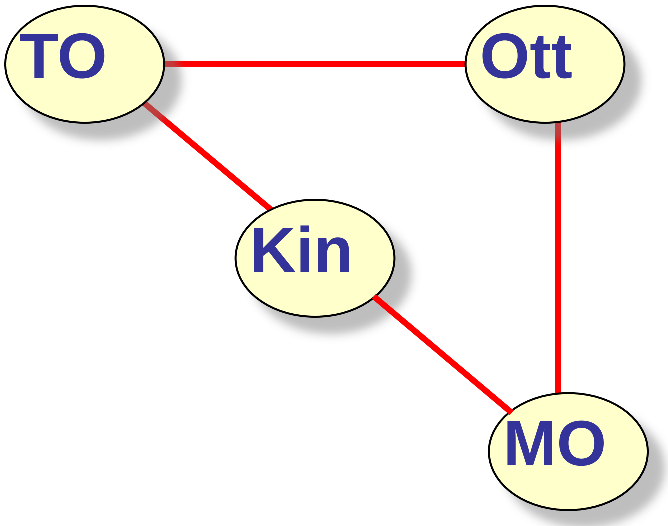
$E = \{(a,b),(a,c),(a,d),(b,e),(c,d),(c,e),(d,e)\}$

Example


450 km
OTT
TOR
90 km
170 km
250 km
100 km
KIN
COR
MON
150 km
100 km
100 km
200 km
ROC
SYR
100 km

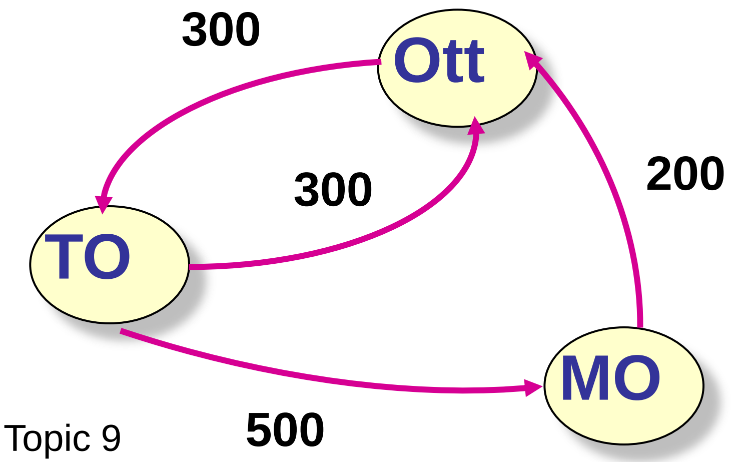
Terminology

An *undirected graph* has *undirected edges*. Each edge is associated with an unordered pair.

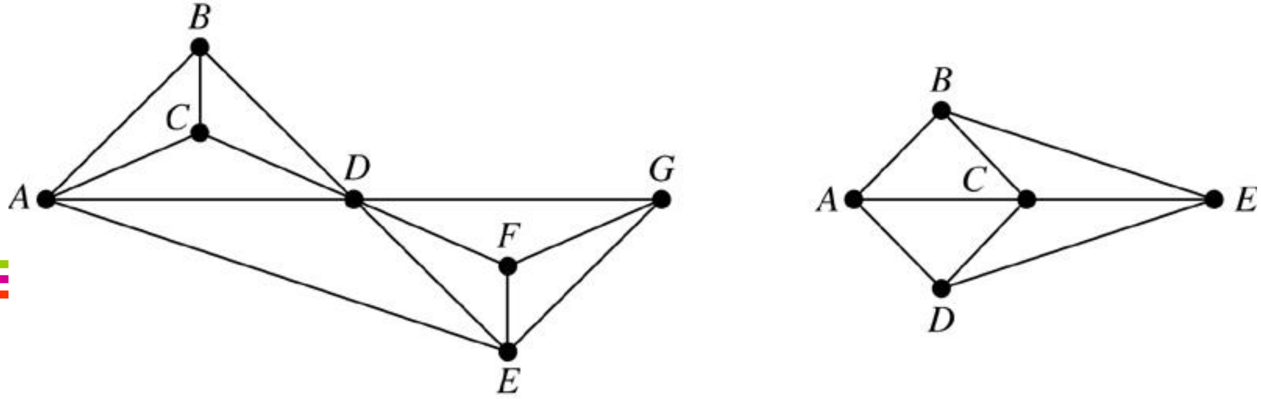


A *directed graph*, or *digraph*, has *directed edges*. Each edge is associated with an ordered pair.

A *weighted graph* is one in which the edges are labeled with numeric values.



Undirected Graphs

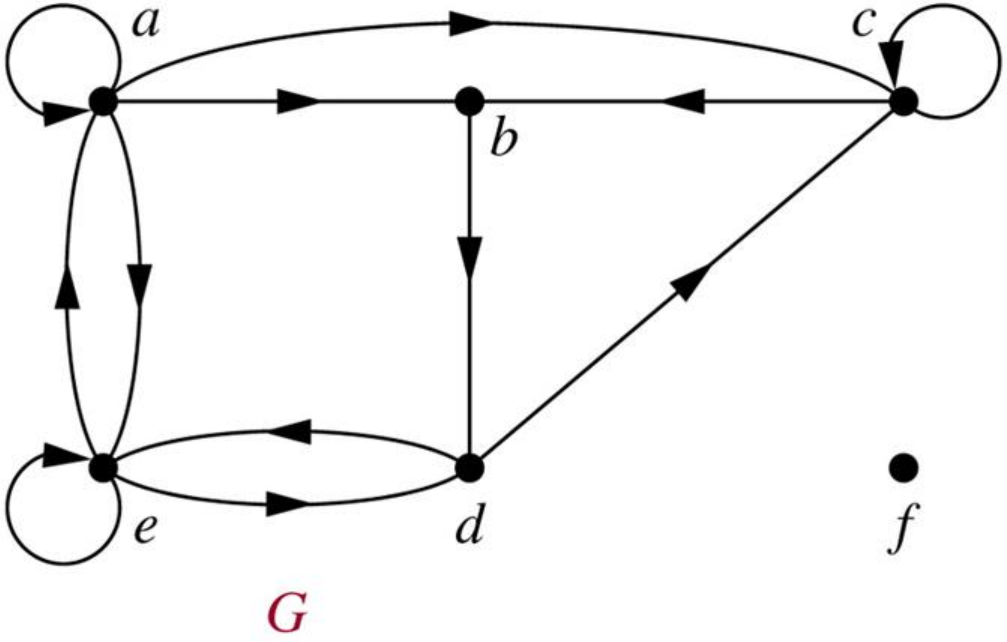


Adjacent (Neighbors): Two vertices connected by an edge are adjacent.

Incident: The edge that connects two vertices is incident on both of them.

Degree of a Vertex *v*, deg(*v*): The number of edges incident on it (loop at vertex is counted twice)

Directed Graphs



Edge (u,v): u is adjacent to v

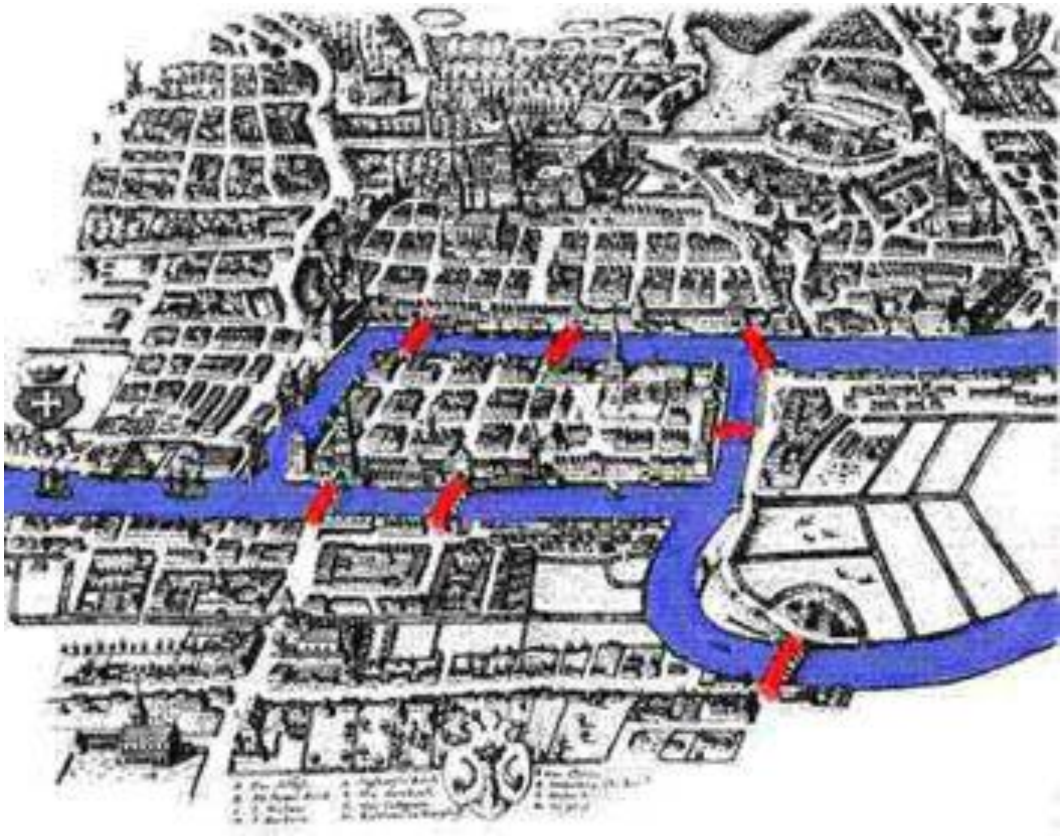
v is adjacent from u

$\deg^-(v)$: The in-degree of v, the number of edges entering it

$\deg^+(v)$: The out-degree of v, the number of edges leaving it

Euler & the Bridges of Koenigsberg

Can one walk across each bridge exactly once and return to the starting point?



Eulerian Tour

What characteristics are required of an undirected graph for a Eulerian Tour to be possible?

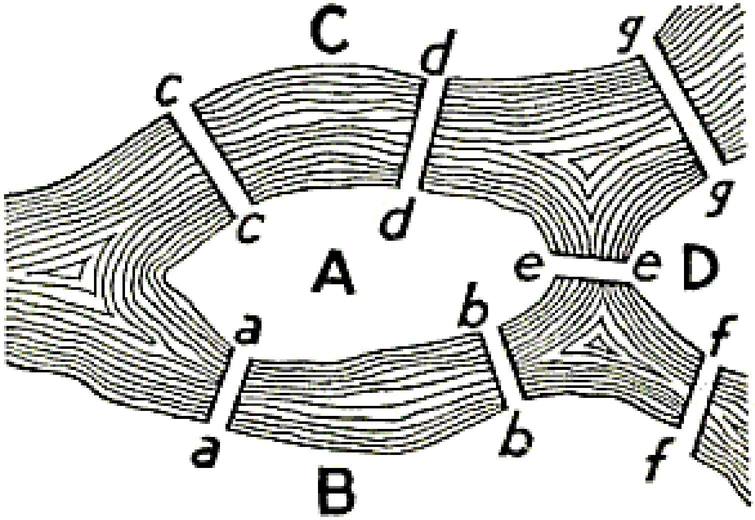
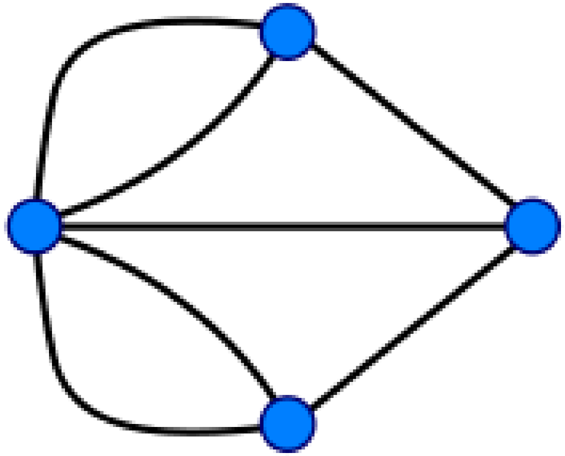
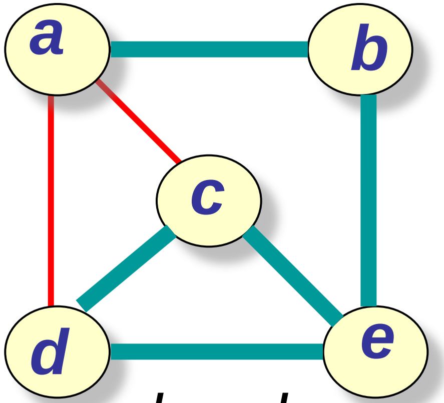


FIGURE 98. *Geographic Map: The Königsberg Bridges.*

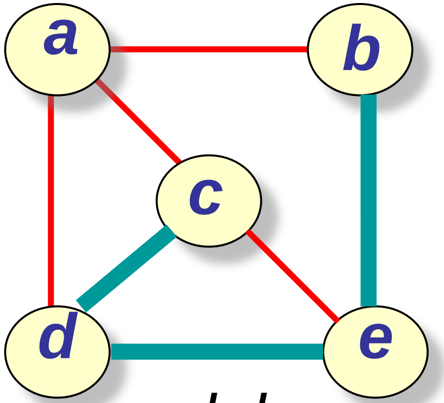


Terminology

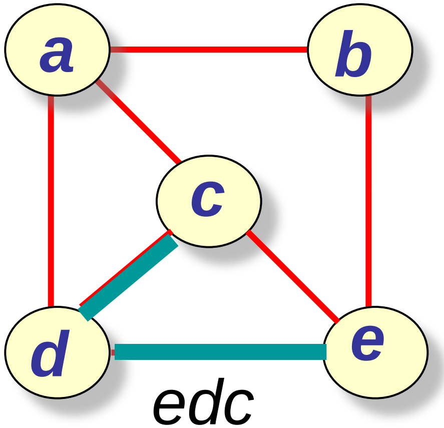
- A *path* is a sequence of vertices $v_1, v_2, \ldots v_k$ such that v_i and v_{i+1} are adjacent.
- A *simple path* is a path that contains no repeated vertices, except for perhaps the first and last vertices in the path.
- A *cycle* is a simple path, in which the last vertex is the same as the first vertex.



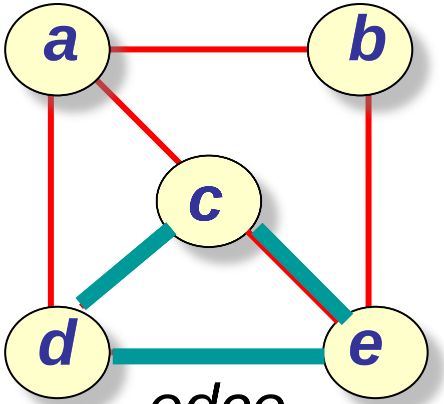
abecde



cdeb



edc



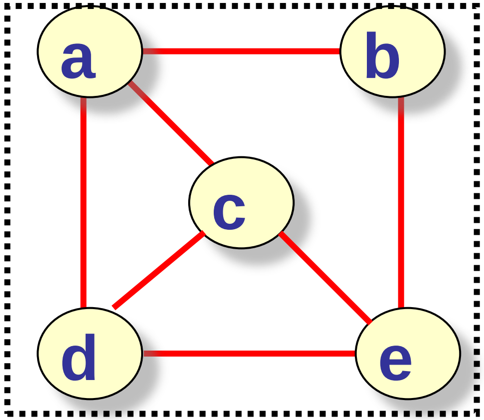
edce

Terminology

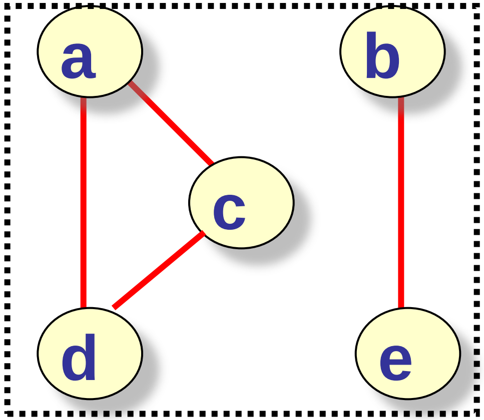
A graph is *connected* if, for any two vertices, there is a path between them.

A *tree* is a connected graph without cycles.

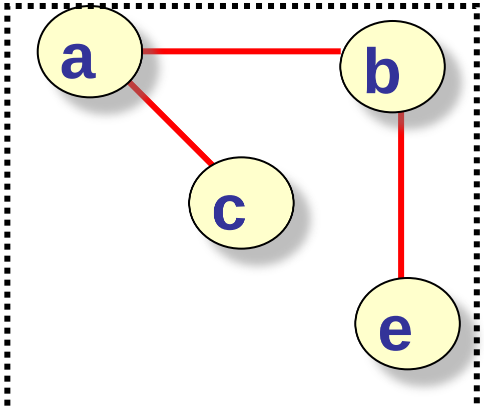
A *subgraph* of a graph G is a graph H whose vertices and edges are subsets of the vertices and edges of G.



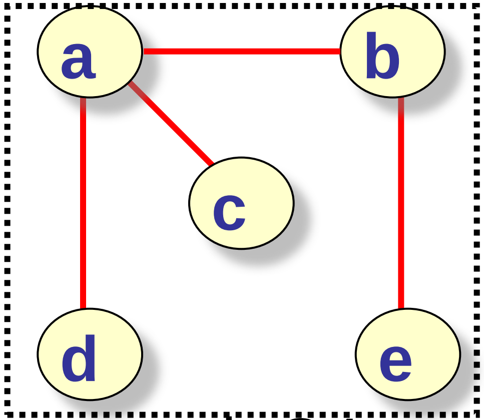
graph G1



graph G2



graph G3

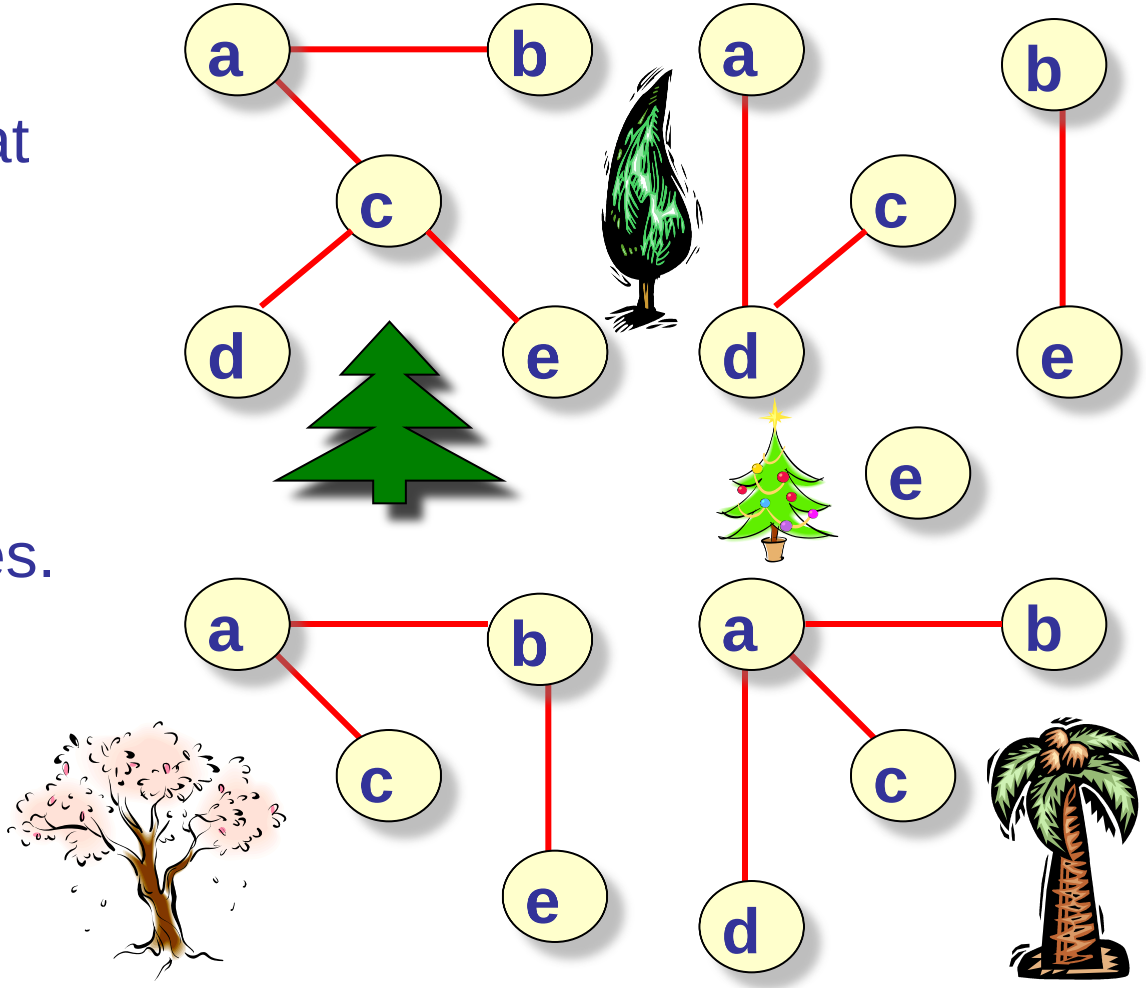


graph G4

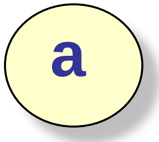
Terminology

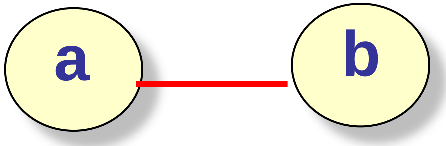
A *forest* is a graph that is a collection of trees.

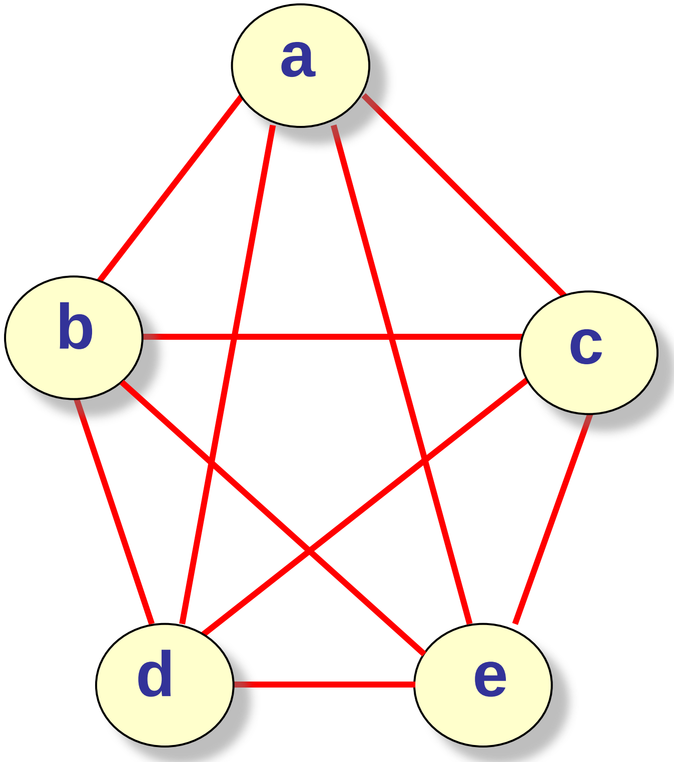
More simply, it is a graph without cycles.



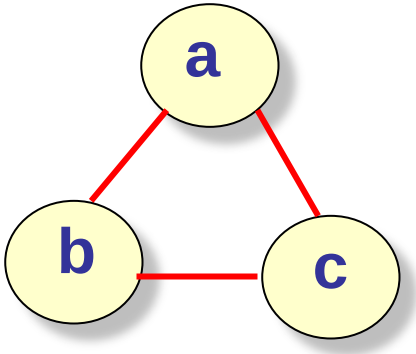
Terminology

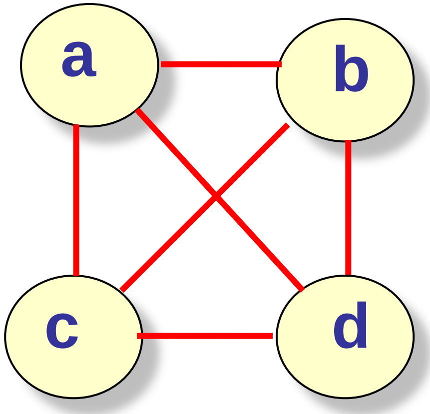






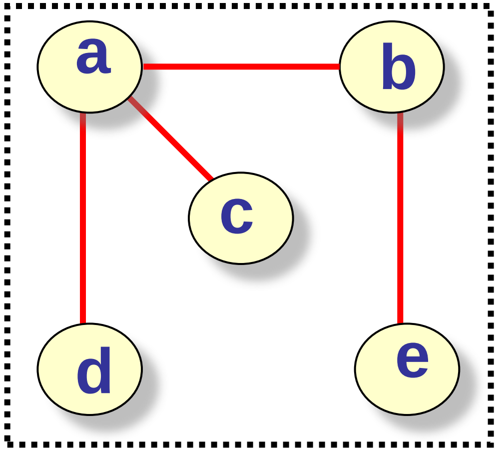
A *complete* graph is an undirected graph with every pair of vertices adjacent.



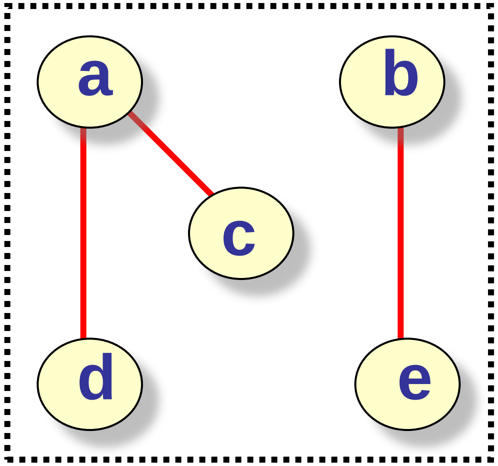


Undirected Graphs: Properties

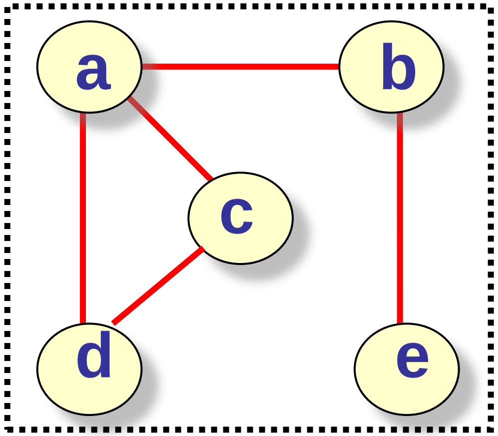
If $|E| = |V| - 1$ and the graph is connected, the graph is a tree



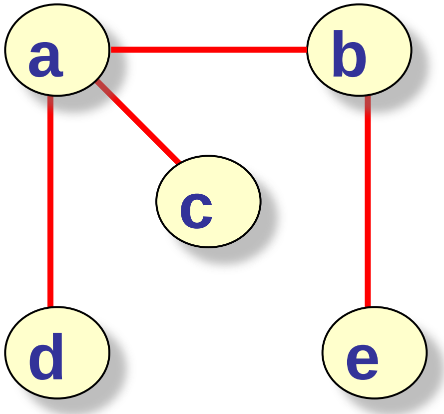
If $|E| < |V| - 1$, the graph is not connected



If $|E| > |V| - 1$, the graph has at least one cycle



Undirected Graphs: Properties

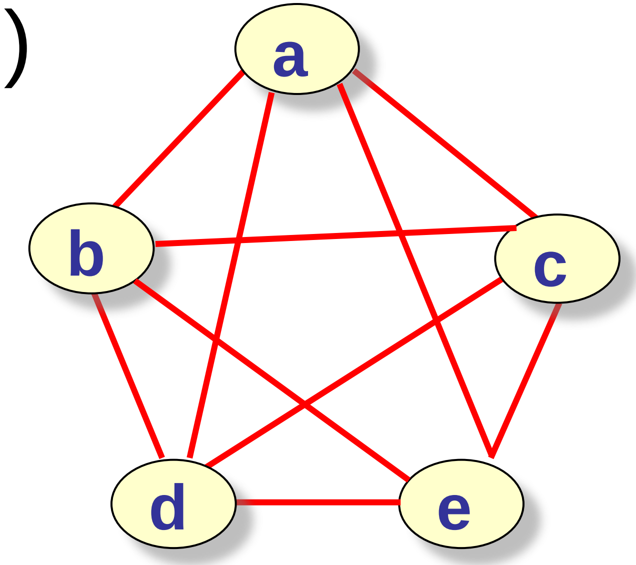


Let $n = |V|$

Let $m = |E|$

Sparse Graphs : m is O(n)

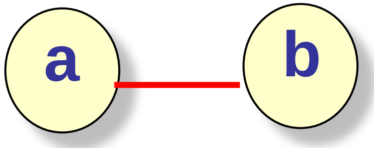
Dense Graphs : m is O(n^2)



Are complete graphs dense graphs?

Complete Graphs

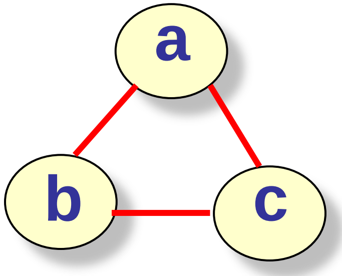
K_2



$n = 2$

$m = 1$

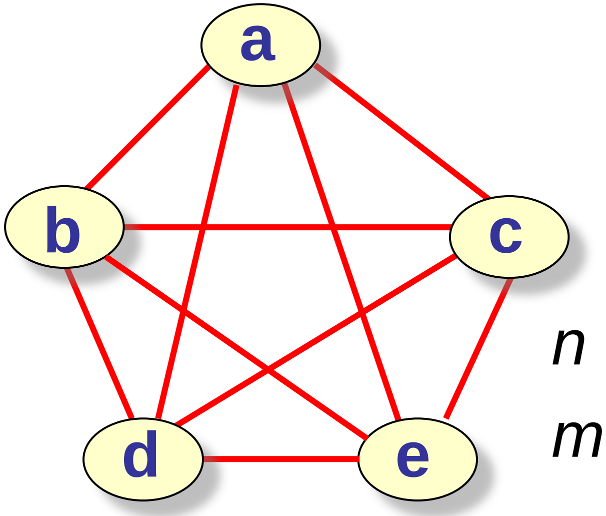
K_3



$n = 3$

$m = 3$

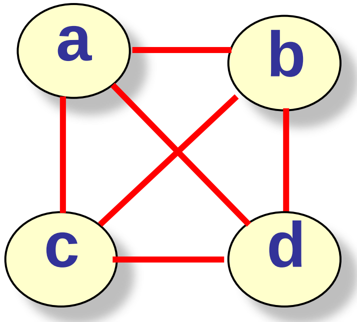
K_5



$n = 5$

$m = 10$

K_4



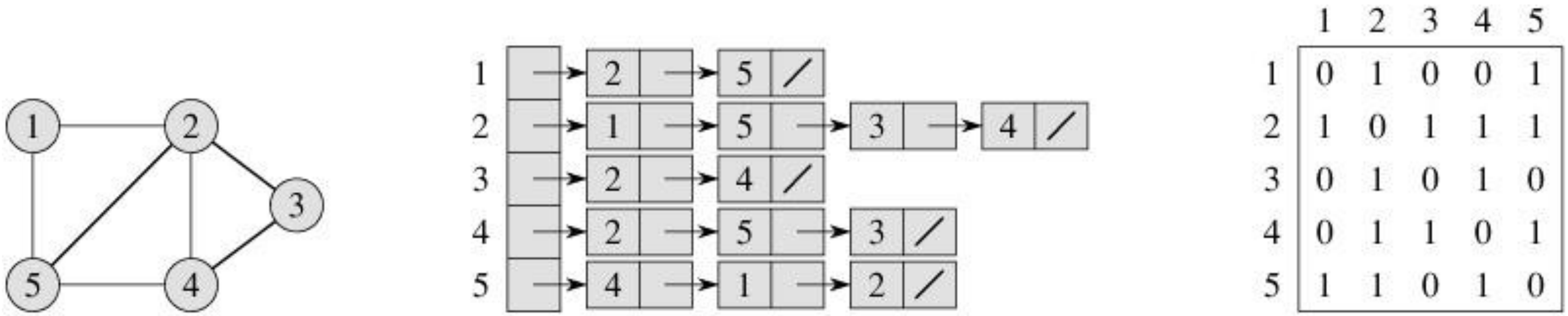
$n = 4$

$m = 6$

For K_n, $m = n(n-1)/2$

Representations of Graphs

Adjacency List and Adjacency Matrix Representations of an Undirected Graph

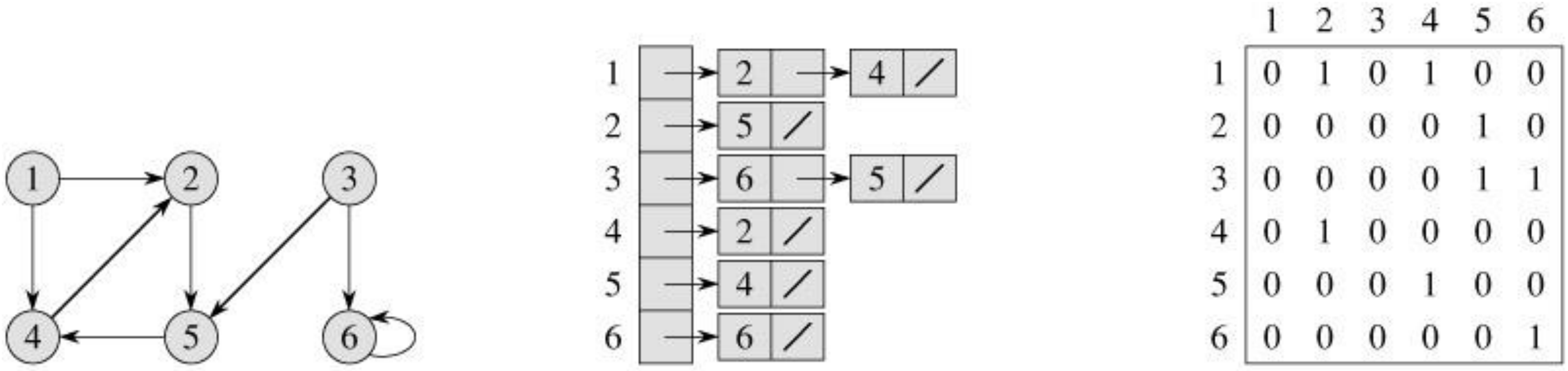


$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 & 1 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 \end{array}$$

(a) (b) (c)

Representations of Graphs

Adjacency List and Adjacency Matrix Representations of a Directed Graph



$$
\begin{array}{c|cccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 0 & 0 & 1 & 1 \\
4 & 0 & 1 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 1 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}
$$

(a) (b) (c)

Graph Implementation

Data

- Store two sets of info: vertices & edges
- Data can be associated with both vertices & edges

A Few Typical Operations

- adjacentVertices(v) – Return list of adjacent vertices
- areAdjacent(v, w) – True if vertex v is adjacent to w
- insertVertex(o) – Insert new isolated vertex storing o
- insertEdge(v, w, o) – Insert edge from v to w, storing o at this edge
- removeVertex(v) – Remove v and all incident edges
- removeEdge(v, w) – Remove edge (v,w)

Graph Representations
Space Analysis

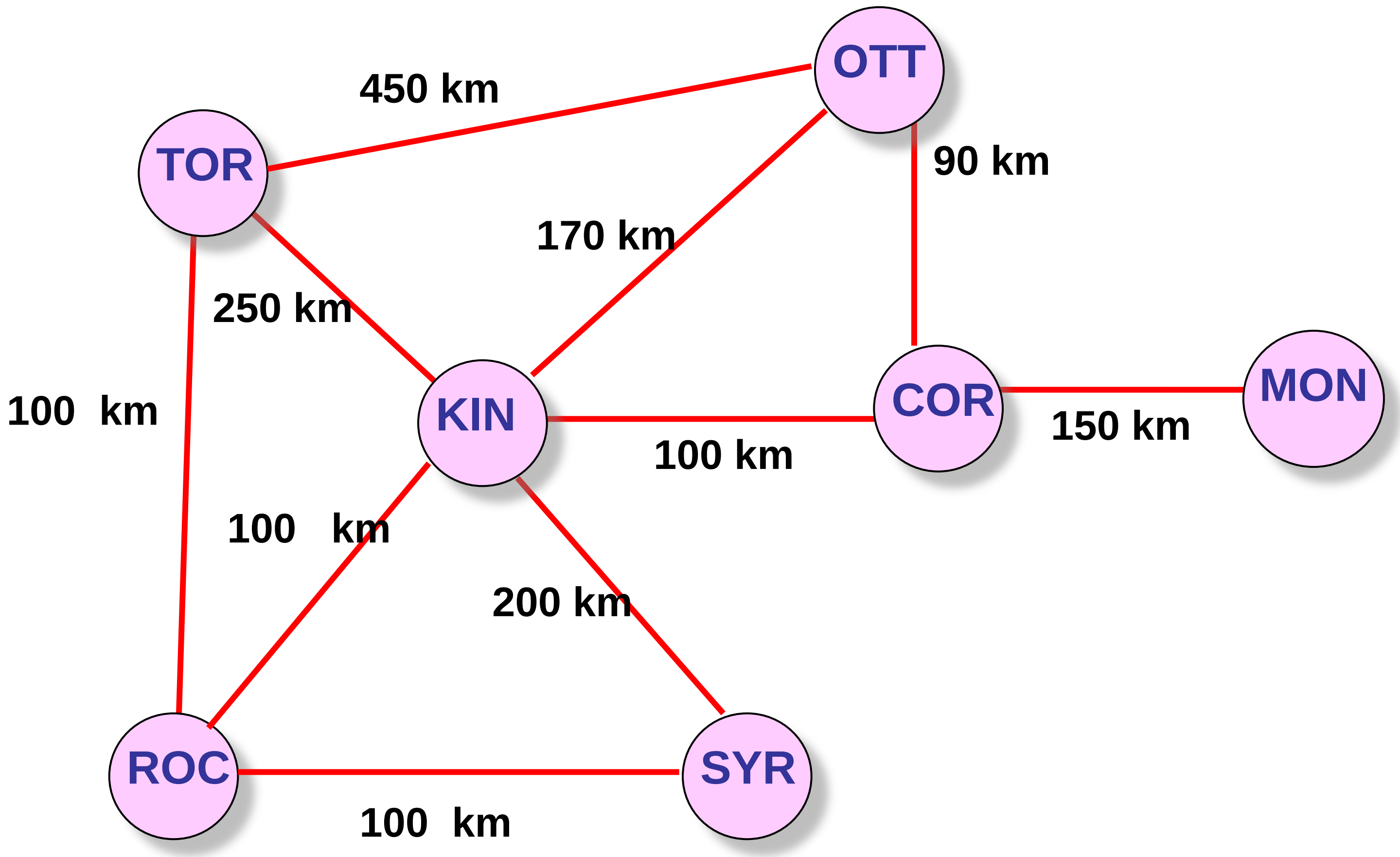
Adjacency List:

Adjacency Matrix:

Graph Representations
Time Analysis

A Few Common Operations	Adjacency List	Adjacency Matrix
areAdjacent(v, w)		
adjacentVertices(v)		
removeEdge(v, w)		

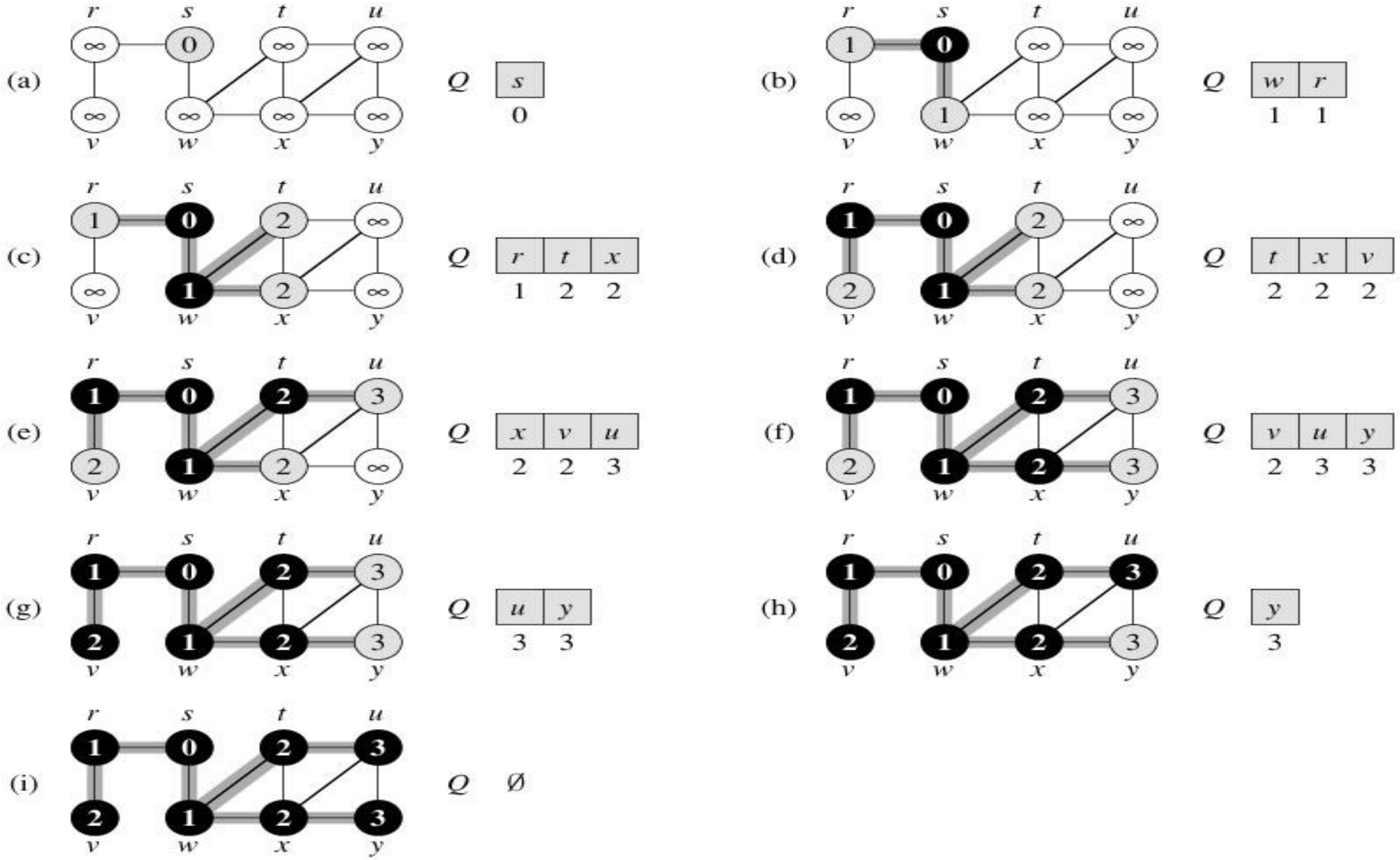
Traversals: Breadth-First Search & Depth-First Search



Breadth-First Search

```
bfs( vertex v )
    Create a queue, Q, of vertices, initially empty
    Visit v and mark it as visited
    Enqueue ( v, Q )
    while not empty( Q )
        w = dequeue( Q )
        for each unvisited vertex u adjacent to w
            Visit u and mark it as visited
            Enqueue( u, Q )
```

Breadth-First Search on an Undirected, Connected Graph



Depth-First Search

```
dfs( vertex v )
    Visit v and mark it as visited
    for each unvisited vertex u adjacent to v
            dfs( v )
```

Analysis of BFS & DFS

Let $n = | V |$

Let $m = | E |$

Application: Java Garbage Collection

C & C++: Programmer must explicitly allocate and deallocate memory space for objects - source of errors

Java: Garbage collection deallocates memory space for objects no longer used. How?

Mark-Sweep Garbage Collection Algorithm

- Suspend all other running threads.
- Trace through the Java stacks of currently running threads and mark as “live” all of the “root” objects.
- Traverse each object in the heap that is active, by starting at each root object, and mark it as “live”.
- Scan through the entire memory heap and reclaim any space that has not been marked.

Algorithms Related to BFS & DFS

- How could we test whether an undirected graph *G* is connected?
- How could we compute the connected components of *G*?
- How could we compute a cycle in *G* or report that it has no cycle?
- How could we compute a path between any two vertices, or report that no such path exists?
- How could we compute for every vertex *v* of *G*, the minimum number of edges of any path between *s* and *v*?