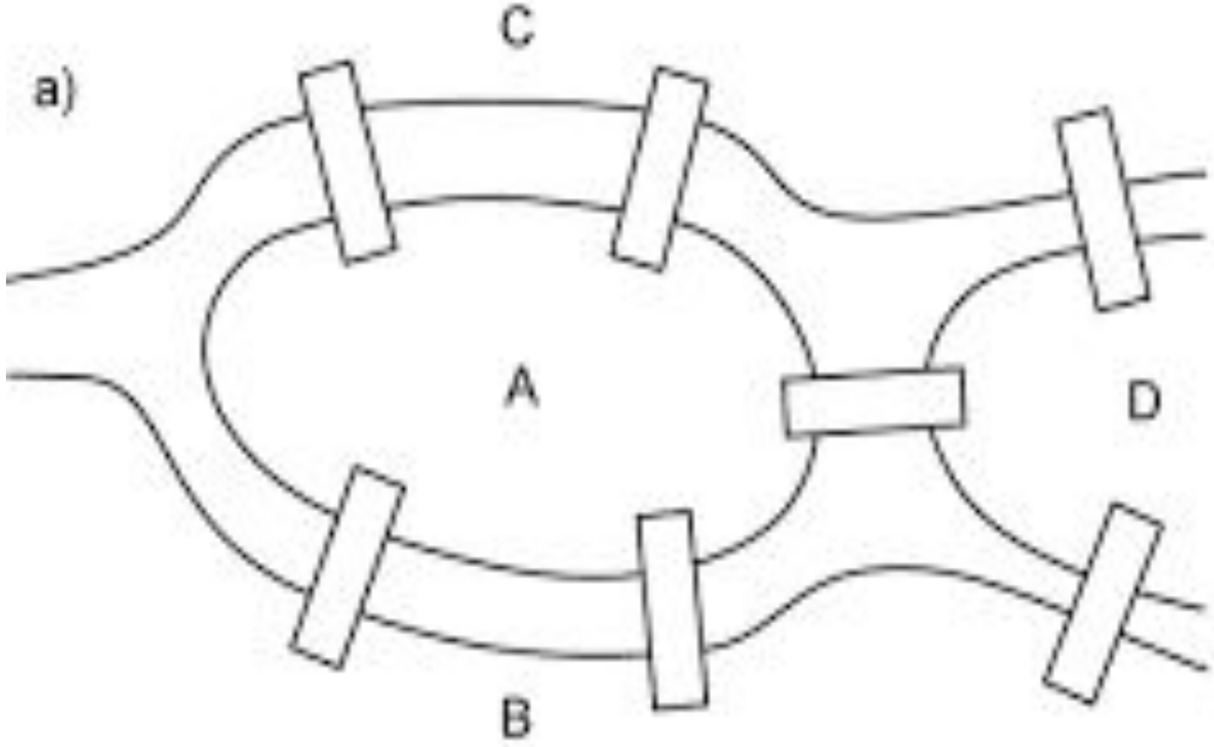
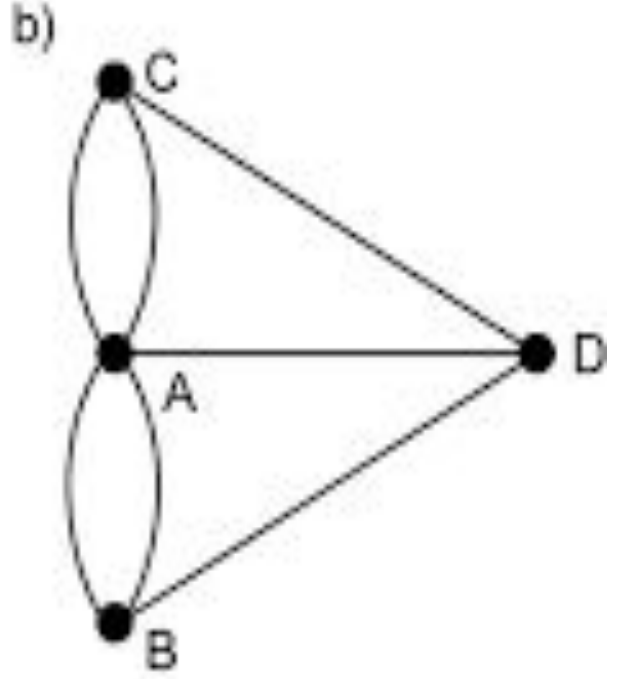
Graphs

The Königsberg bridge problem:

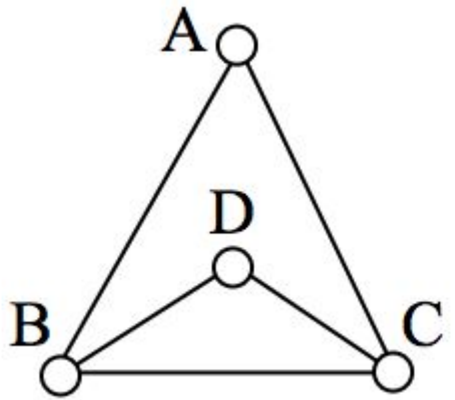


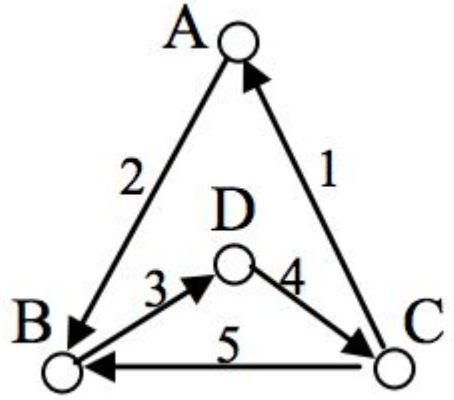


a) seven bridges of Königsberg; b) graph representation.

EULER PATH

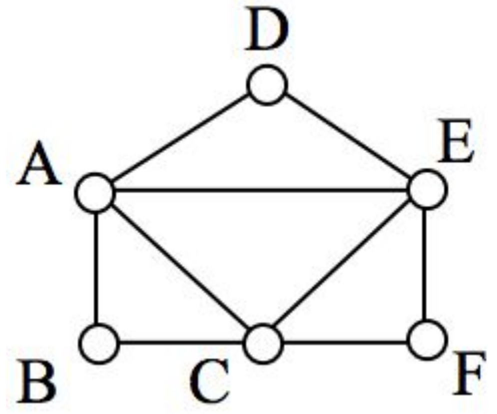
- An **Euler path** is a path that uses every edge in a graph with no repeats. Being a path, it does not have to return to the starting vertex.
- In the graph shown below, there are several Euler paths. One such path is CABDCB. The path is shown in arrows to the right, with the order of edges numbered.

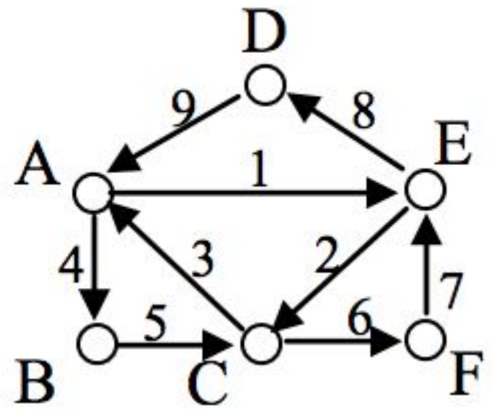




EULER CIRCUIT

- An **Euler circuit** is a circuit that uses every edge in a graph with no repeats. Being a circuit, it must start and end at the same vertex.
- The graph below has several possible Euler circuits. Here's a couple, starting and ending at vertex A: ADEACEFCBA and AECABCFEDA. The second is shown in arrows.





EULER’S PATH AND CIRCUIT THEOREMS

- A graph will contain an Euler path if it contains at **most two vertices of odd degree.**

- A graph will contain an Euler circuit if all vertices have even degree

Definitions

- ●***Graphs***
- A graph G=(V,E) consists of a set of vertices, V, and a set of edges, E.
- ***Edges***
- Each edge is a pair (v, w), where v, w ∈ V. Edges are sometimes referred to as arcs.
- ●***Digraphs***
- If the pair is ordered, then the graph is directed. Directed graphs are sometimes referred to as digraphs.
- ●

- *Adjacency*

  Vertex w is adjacent to v if and only if (v, w) $\in$ E.

- ● ***Path***
- A path in a graph is a sequence of vertices w_1 , w_2 , w_3 , . . . , w_N such that (w_i , w_{i+1}) $\in$ E for $1 \leq i < N$. Length of such path is the number of edges.
- ***Loop***
- If there exists an edge from a vertex to itself, it is referred to as a loop.

•

Simple Path
It is a path such that all the vertices are distinct, except the first and the last one could be same.

Cycle
A cycle is a simple path of length at least 1 such that $w_1 = w_N$;

- Cycle
  - A simple path of a positive length that starts and ends at the same vertex.
- Acyclic graph
  - A graph without cycles
  - DAG (Directed Acyclic Graph): Diagraph

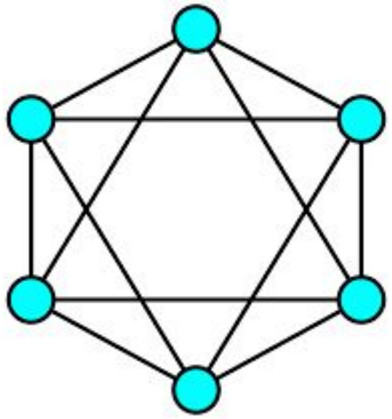
- ***Connected***
- An undirected graph is connected if there exists a path from every vertex to every other vertex. A directed graph with such property is called **strongly connected graph**
- ***Weakly connected Graph***
- If a directed graph is not strongly connected, but the underlying graph (without direction to the arcs) is connected, then the graph is said to be weakly connected

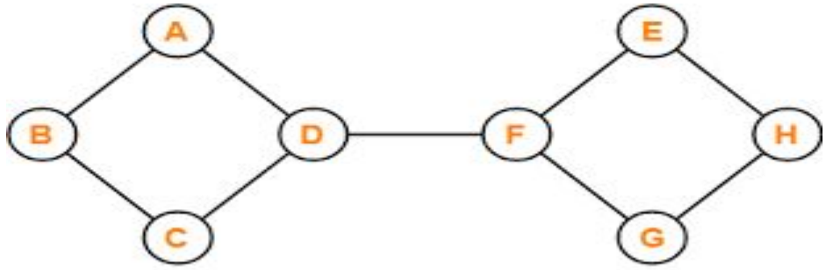
- *Complete graph*

A complete graph is a graph in which there is an edge between every pair of vertices.

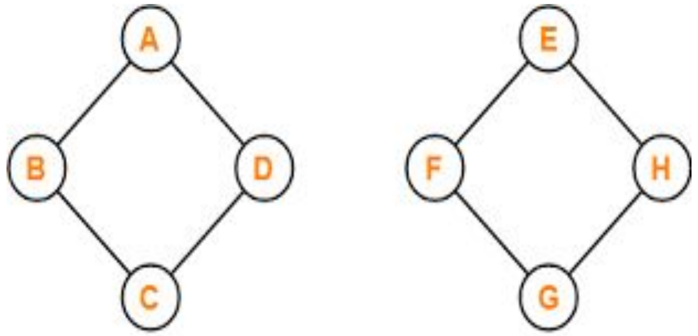
N(N-1)/2

3(2)/2 = 3   4(3)/2 = 6 5(4)/2 = 10



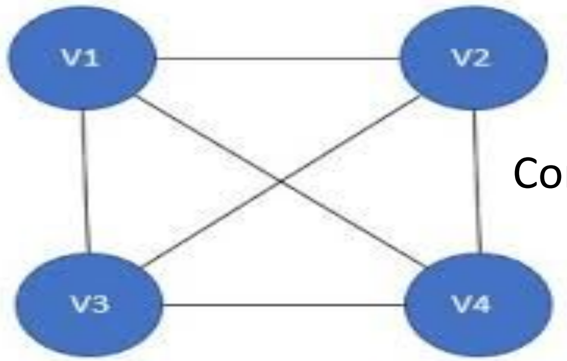


Example of Connected Graph



Example of Disconnected Graph

connected graph



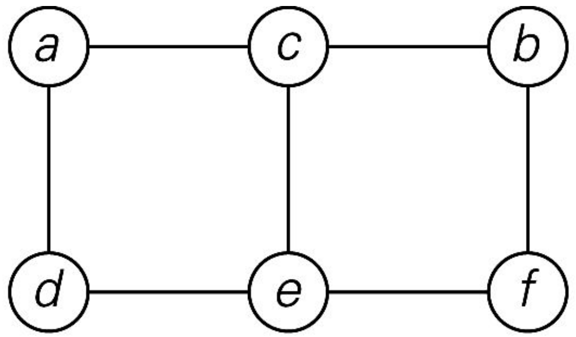
Complete Graph

Representation of Graphs

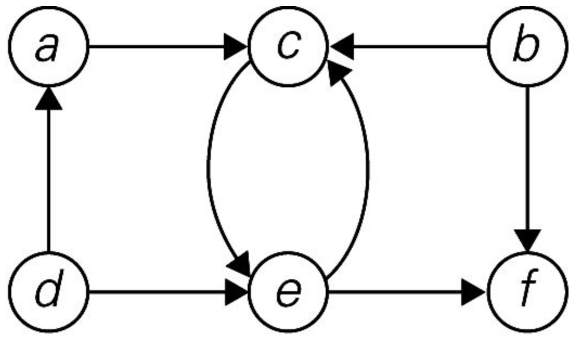
- Graphs can be represented as
  - Adjacency Matrix
- Adjacency linked list
-

Graphs

- Formal definition
  - A graph $G = <V, E>$ is defined by a pair of two sets: a finite set V of items called vertices and a set E of vertex pairs called edges.
- Undirected and directed graphs (digraph).
- What's the maximum number of edges in an undirected graph with |V| vertices?
- Complete, dense, and sparse graph
  - A graph with every pair of its vertices connected by an edge is called complete. $K_{|V|}$



(a)



(b)

Graph Representation

- Adjacency matrix
  - n x n boolean matrix if |V| is n.
  - The element on the ith row and jth column is 1 if there's an edge from ith vertex to the jth vertex;

$$\begin{array}{c|cccccc} & a & b & c & d & e & f \\ \hline a & 0 & 0 & 1 & 1 & 0 & 0 \\ b & 0 & 0 & 1 & 0 & 0 & 1 \\ c & 1 & 1 & 0 & 0 & 1 & 0 \\ d & 1 & 0 & 0 & 0 & 1 & 0 \\ e & 0 & 0 & 1 & 1 & 0 & 1 \\ f & 0 & 1 & 0 & 0 & 1 & 0 \end{array}$$

$\Updownarrow$

$a \to c \to d$

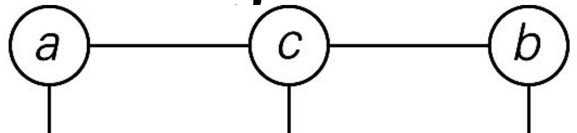
$b \to c \to f$

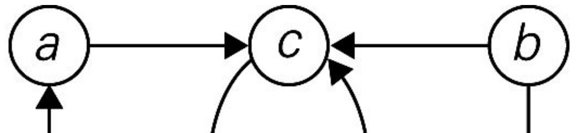
$c \to a \to b \to e$

$d \to a \to e$

$e \to c \to d \to f$

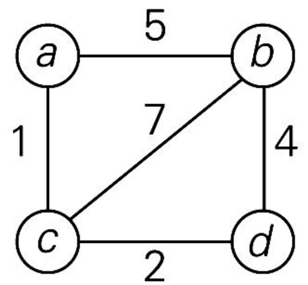
$f \to b \to e$





Weighted Graphs

- Weighted graphs
  - Graphs or digraphs with numbers assigned to the edges.



(a)

$$\begin{array}{c|cccc} & a & b & c & d \\ \hline a & \infty & 5 & 1 & \infty \\ b & 5 & \infty & 7 & 4 \\ c & 1 & 7 & \infty & 2 \\ d & \infty & 4 & 2 & \infty \end{array}$$

(b)

$a \rightarrow b, 5 \rightarrow c, 1$

$b \rightarrow a, 5 \rightarrow c, 7 \rightarrow d, 4$

$c \rightarrow a, 1 \rightarrow b, 7 \rightarrow d, 2$

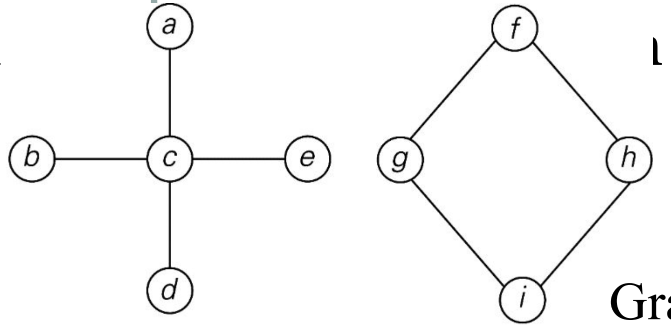
$d \rightarrow b, 4 \rightarrow c, 2$

(c)

(a) Weighted graph. (b) Its weight matrix. (c) Its adjacency lists.

Graph Properties -- Paths and Connectivity

- Paths
  - A path from vertex u to v of a graph G is defined as a sequence of adjacent (connected by an edge) vertices that starts with u and ends with v.
  - Simple paths: All edges of a path are distinct.
  - Path lengths: the number of edges, or the number of vertices – 1.
- Connected graphs
  - A graph is said to be connected if for every pair of its vertices u and v there is a path from u to v.
- Connected component
  - The maxim ı of a given graph.



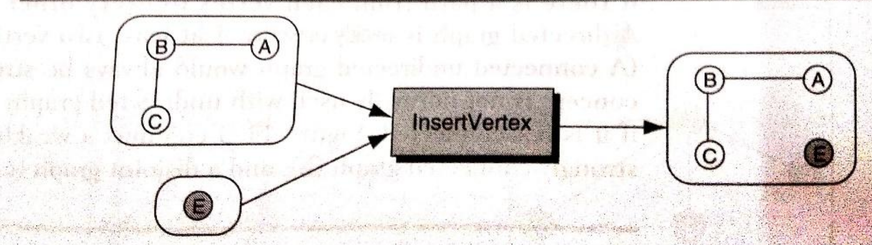
Graph that is not connected

Graph Properties -- Acyclicity

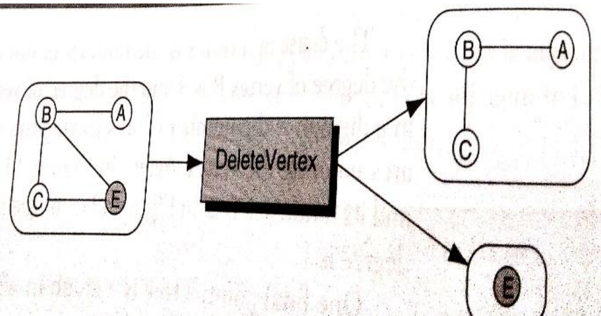
- Cycle
  - A simple path of a positive length that starts and ends a the same vertex.
- Acyclic graph
  - A graph without cycles
  - DAG (Directed Acyclic Graph)

Graph Operations

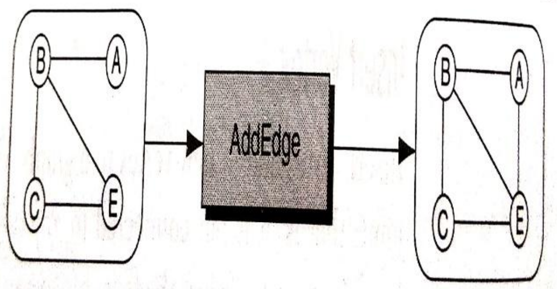
- Insert vertex
- Delete Vertex
- Add Edge
- Delete Edge
- Find Vertex



E 11-4 Insert Vertex

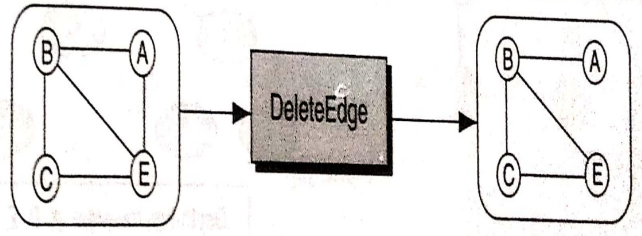


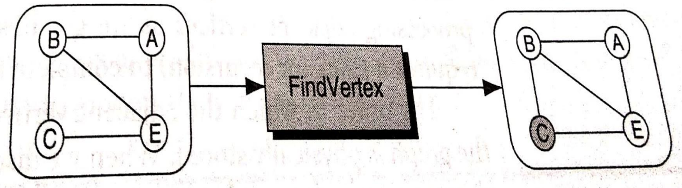
11-5 Delete Vertex



E 11-6 Add Edge

find vertex traverses the graph, looking for vertex C.





8 Find Vertex

Graph Traversals

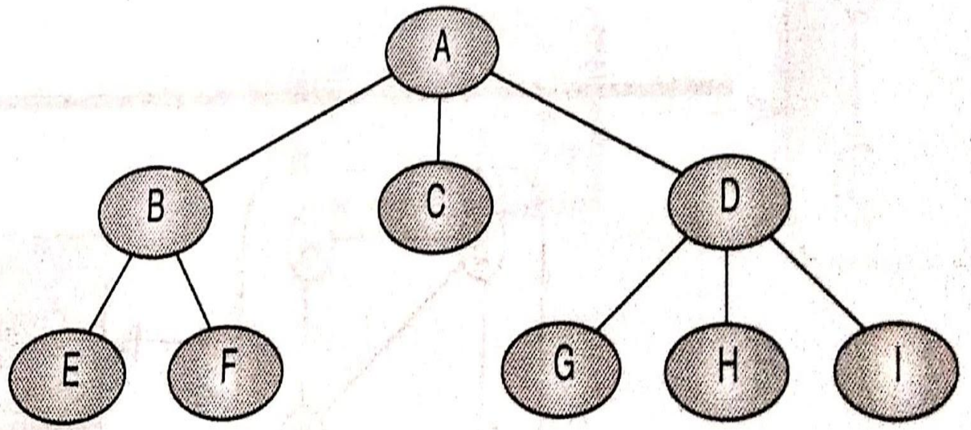
- Depth First Traversal

  Process all the descendants of a vertex before processing the next vertex.
- This can be best seen when a Graph is a Tree.

Algorithm

- Choose any vertex as the starting vertex.
- After processing the first vertex, select any vertex adjacent to it and process it.
- After processing each vertex select next adjacent unvisited vertex until a vertex with no adjacent unvisited vertex is reached(Dead end vertex).
- Once a dead end vertex is reached, Back track to the previous vertex, where the recent dead

Graph Traversals

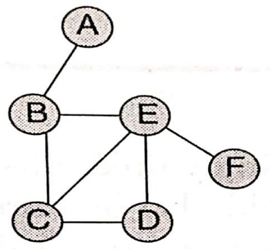


Depth-first traversal: A B E F C D G H I

Depth-first Traversal of a Tree

Grap

F: FEBACD
C: CBAEDF



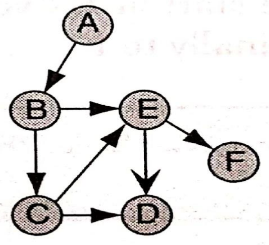
Vertex vector

A
B
C
D
E
F

Adjacency matrix

$$\begin{array}{c|cccccc} & A & B & C & D & E & F \\ \hline A & 0 & 1 & 0 & 0 & 0 & 0 \\ B & 1 & 0 & 1 & 0 & 1 & 0 \\ C & 0 & 1 & 0 & 1 & 1 & 0 \\ D & 0 & 0 & 1 & 0 & 1 & 0 \\ E & 0 & 1 & 1 & 1 & 0 & 1 \\ F & 0 & 0 & 0 & 0 & 1 & 0 \end{array}$$

(a) Adjacency matrix for nondirected graph



Vertex vector

A
B
C
D
E
F

Adjacency matrix

$$\begin{array}{c|cccccc} & A & B & C & D & E & F \\ \hline A & 0 & 1 & 0 & 0 & 0 & 0 \\ B & 0 & 0 & 1 & 0 & 1 & 0 \\ C & 0 & 0 & 0 & 1 & 1 & 0 \\ D & 0 & 0 & 0 & 0 & 0 & 0 \\ E & 0 & 0 & 0 & 1 & 0 & 1 \\ F & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

(b) Adjacency matrix for directed graph

Adjacency Matrix

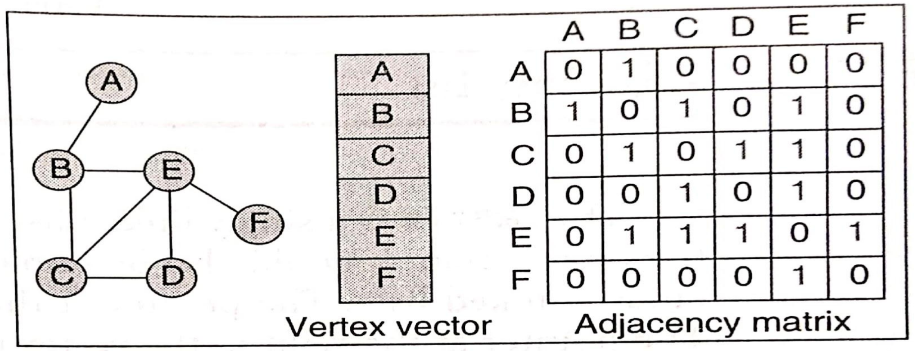
```
A   B    C    D    E       F
    A    B    C    D     E      E
         A    B   C     D     D     D
              A    B    C     C     C     C
                   A    B     B     B     B    B
                        A     A     A     A    A   A   --  DFS:   A  B  C  D E F   DFS:(starting F)    Starting at E:    starting at C:
```

Graph Traversals

- Breadth First Traversal

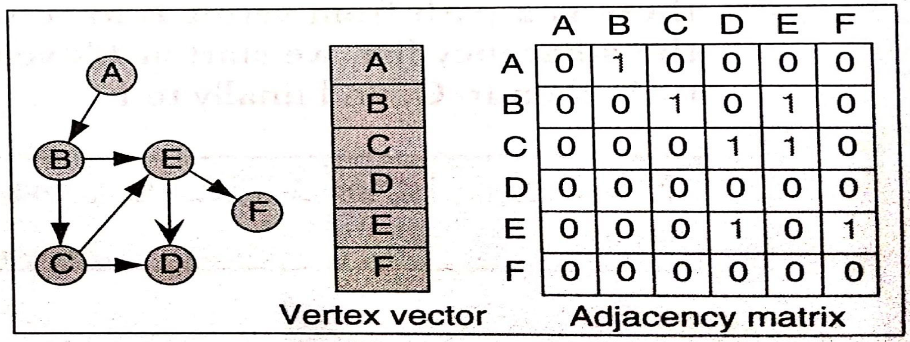
Algorithm

- Begin by picking a starting vertex
- After processing first vertex, process all of its **adjacent unvisited** vertices.
- After processing all the adjacent vertices of first vertex, pick its first adjacent vertex and process its adjacent unvisited vertices.
- Repeat the process until all the vertices are processed.



	A	B	C	D	E	F
A	0	1	0	0	0	0
B	1	0	1	0	1	0
C	0	1	0	1	1	0
D	0	0	1	0	1	0
E	0	1	1	1	0	1
F	0	0	0	0	1	0

(a) Adjacency matrix for nondirected graph



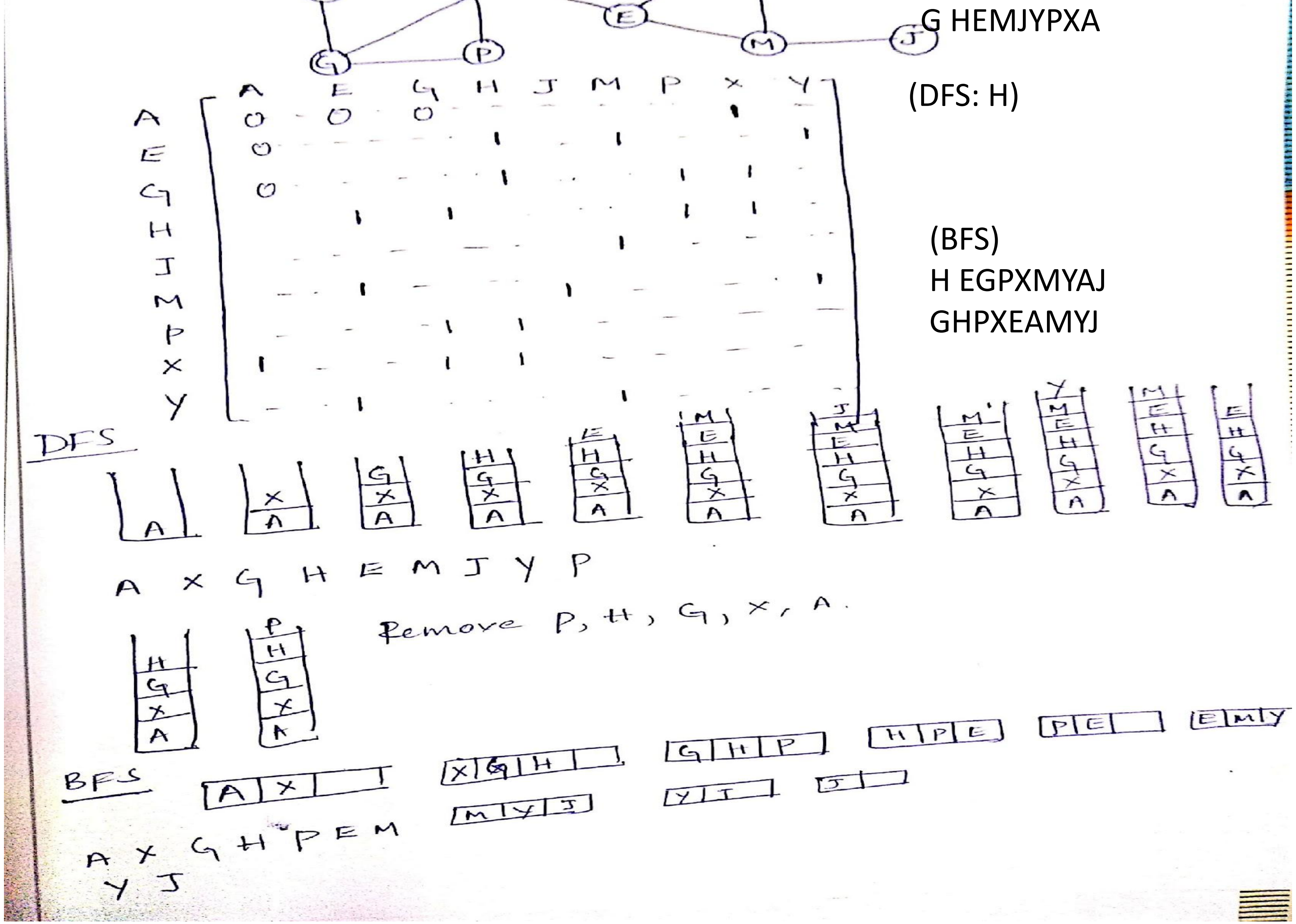
	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	0	1	0	1	0
C	0	0	0	1	1	0
D	0	0	0	0	0	0
E	0	0	0	1	0	1
F	0	0	0	0	0	0

(b) Adjacency matrix for directed graph

Adjacency Matrix

E: EBCDFA
C: CBDEAF
F: FEBCDA

A
A B
B
B C E
C E
C E D   E D   E D F   D F   F   F   --- BFS(A)- ABCEDF   BFS(C):

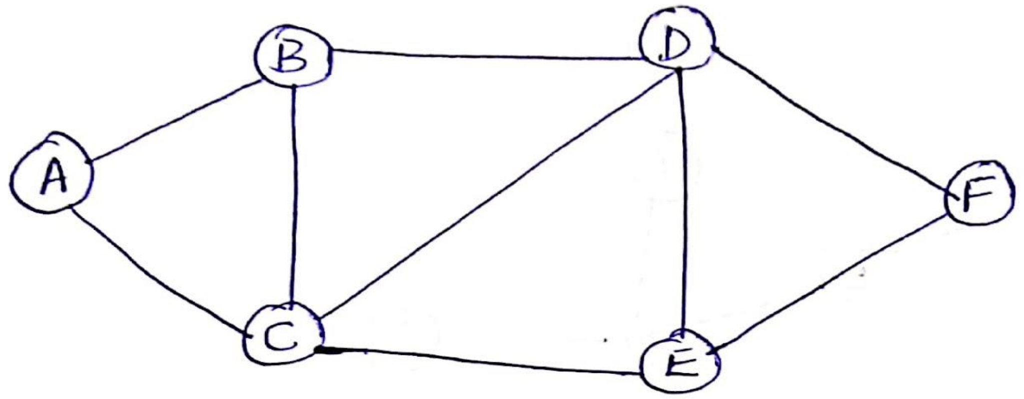


G HEMJYPXA

(DFS: H)

(BFS)
H EGPXMYAJ
GHPXEAMYJ

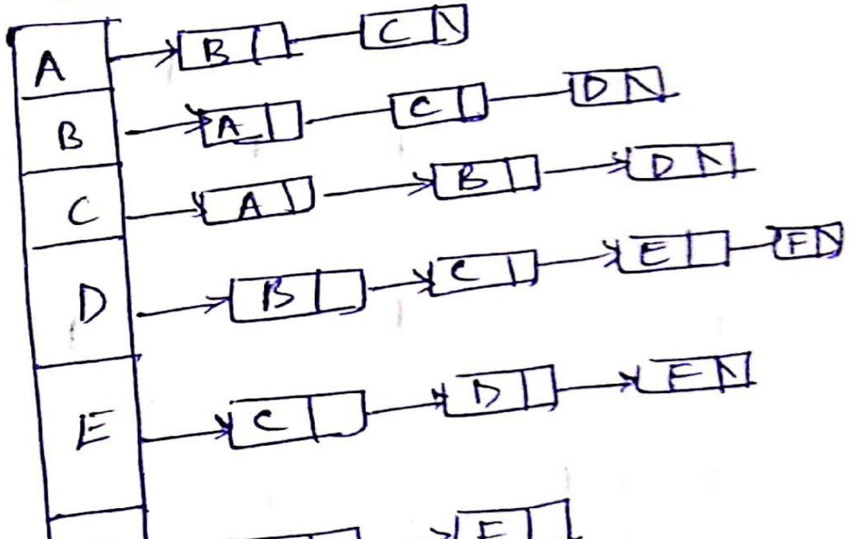
Graph shows the routes between different cities, nodes represents cities and the edge a route between two cities. Choose an appropriate data structure to represent this data with both array and linked storage structures and show its contents. Choose and write an appropriate algorithm to check whether all the cities are connected .



Adjacency Matrix

$$
\begin{array}{c|cccccc}
 & A & B & C & D & E & F \\
\hline
A & 0 & 1 & 1 & 0 & 0 & 0 \\
B & 1 & 0 & 1 & 1 & 0 & 0 \\
C & 1 & 1 & 0 & 1 & 0 & 0 \\
D & 0 & 1 & 1 & 0 & 1 & 1 \\
E & 0 & 0 & 1 & 1 & 0 & 1
\end{array}
$$

Adjacency list



A
B
C
D
E
B
C
A
C
D
A
B
D
B
C
E
F
C
D
E
F
E