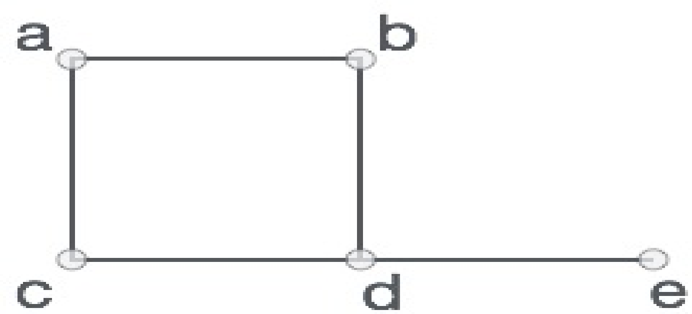
Graph

Graph

- A graph is a collection of nodes called **vertices** and collection of segments called **line or edges**
- A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links
- Formally, a graph is a pair of sets **(V, E),** where **V** is the set of vertices and **E** is the set of edges, connecting the pairs of vertices. Take a look at the following graph



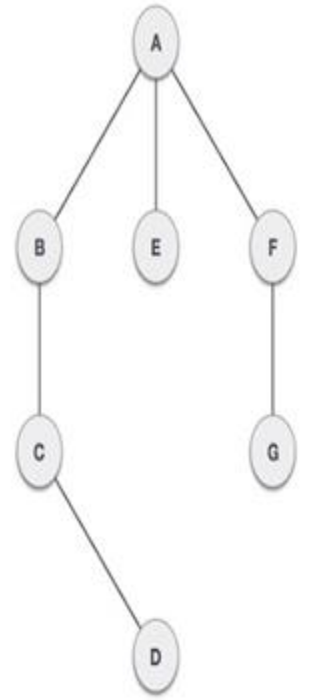
In the above graph,

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

$E = \{ab, ac, bd, cd, de\}$

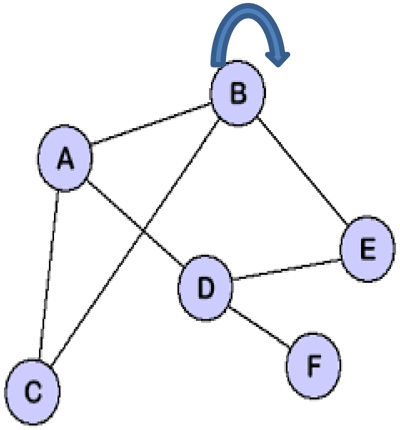
Definitions

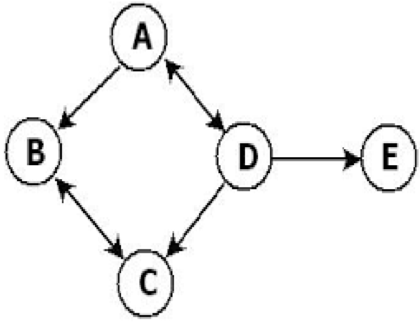
- **Vertex** − Each node of the graph is represented as a vertex. labeled circle represents vertices. So A to G are vertices
- **Edge** − Edge represents a path between two vertices or a line between two vertices. lines from A to B, B to C and so on represents edges
- **Adjacency** − Two node or vertices are adjacent if they are connected to each other through an edge. B is adjacent to A, C is adjacent to B and so on
- **Path** − Path represents a sequence of edges between two vertices. In example given below, ABCD represents a path from A to D



Definitions

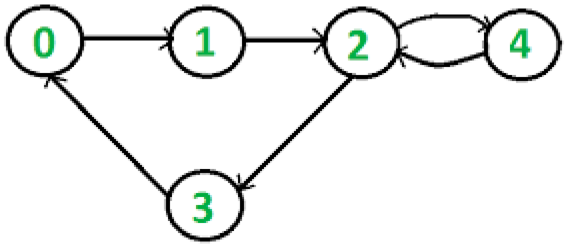
- **Cycle:** A cycle is path that consists of at least three vertices that starts and end at same vertex. Here ABC,ABDE are cycle
- **Loop:** A special case of cycle in which a single arc begins and end with the same vertex
- **Directed graph or digraph:** each edge has direction to its successors
- **Undirected Graph:** No direction on any line
- **Connected Graph:** Two vertices are connected if there is a path between them. A graph is said to be connected if ignoring direction there is a path from any vertex to any other vertex



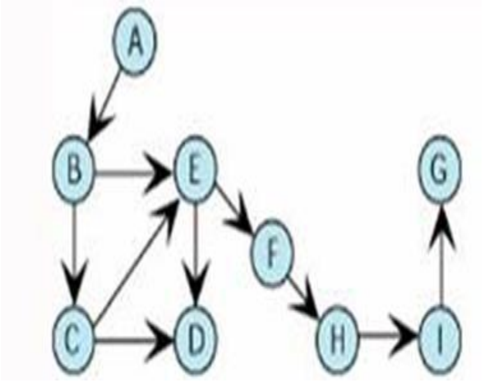


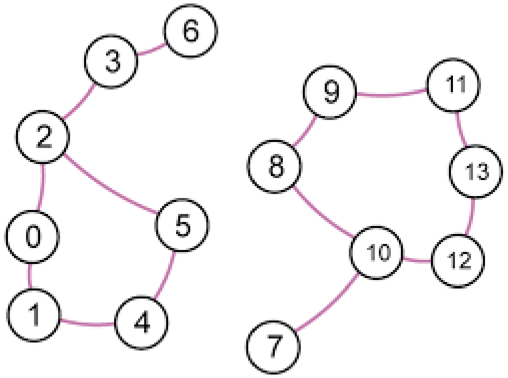
Definitions

- **Strongly Connected:** A directed graph is strongly connected if there is a path from each vertex to other vertex
- **Weakly connected :** A directed graph is weakly connected if at least two vertices are not connected
- **Disjoint Graph:** if graph is not connected



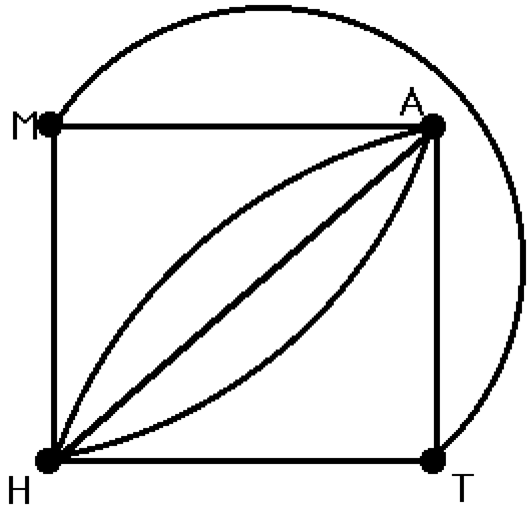
Strongly Connected





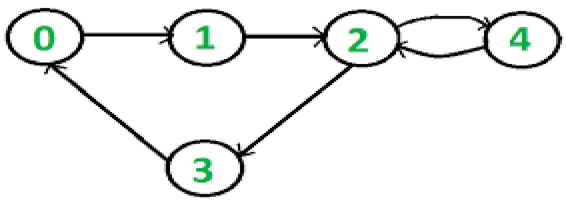
Definitions

- **Degree:** the degree is to count the number of edges which has that vertex as an endpoint. The degree of the graph will be its largest vertex degree. The degree of the graph is 5.



Vertex	Degree
M	3
A	5
T	3
H	5

- **Indegree :** the number of edges entering the vertex of bigraph
- **Outdegree :** the number of edges leaving the vertex of bigraph



Strongly Connected

Operations

- Add vertex
  - Inserting a new vertex
- Delete vertex
  - Remove all connecting edge
- Add edge
- Delete Edge
- Find Vertex
- Traverse Graph
  - Depth-first Traversal
  - Breadth-first Traversal

Fig.11-4,5 Add/Delete Vertex

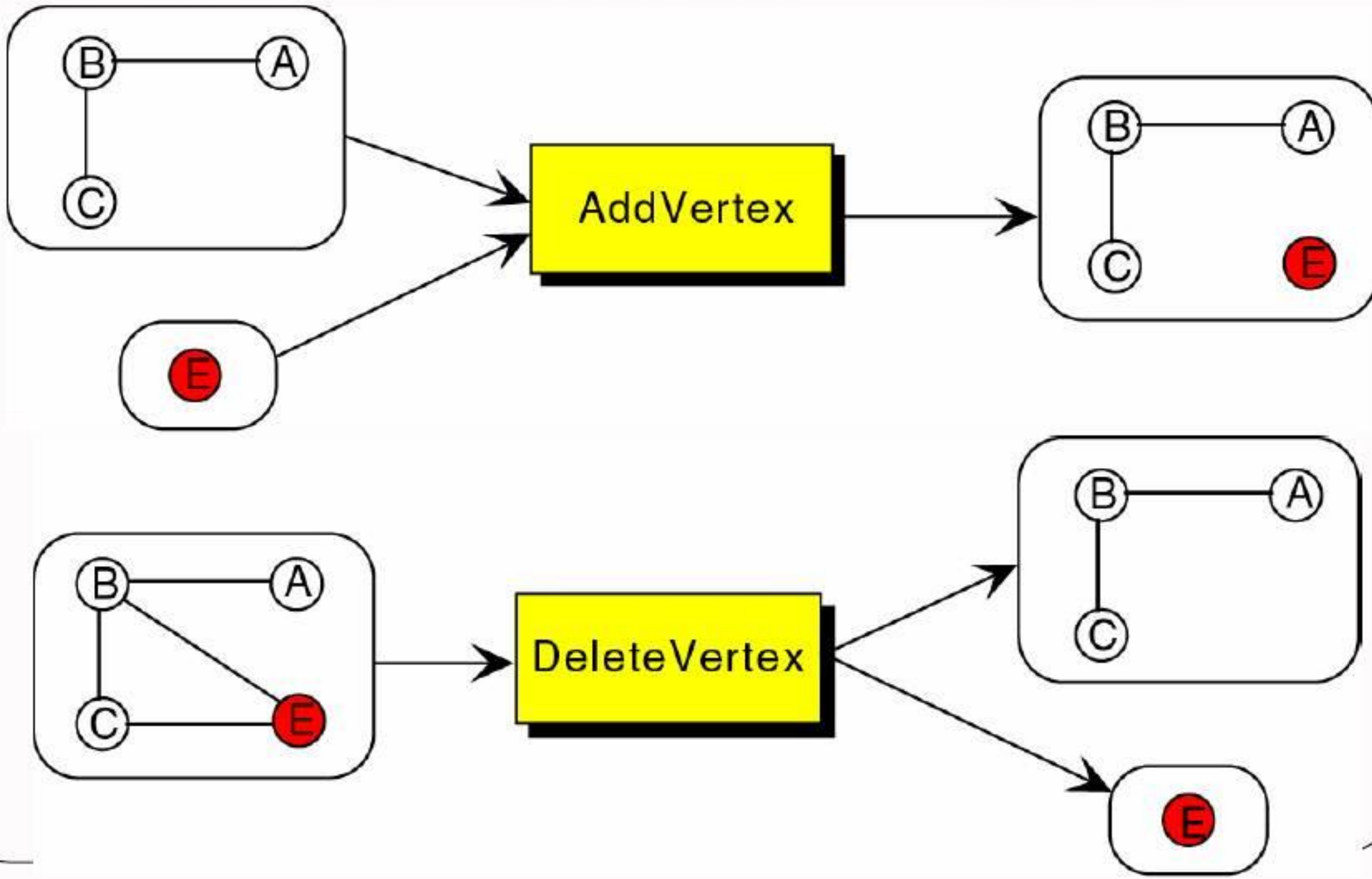
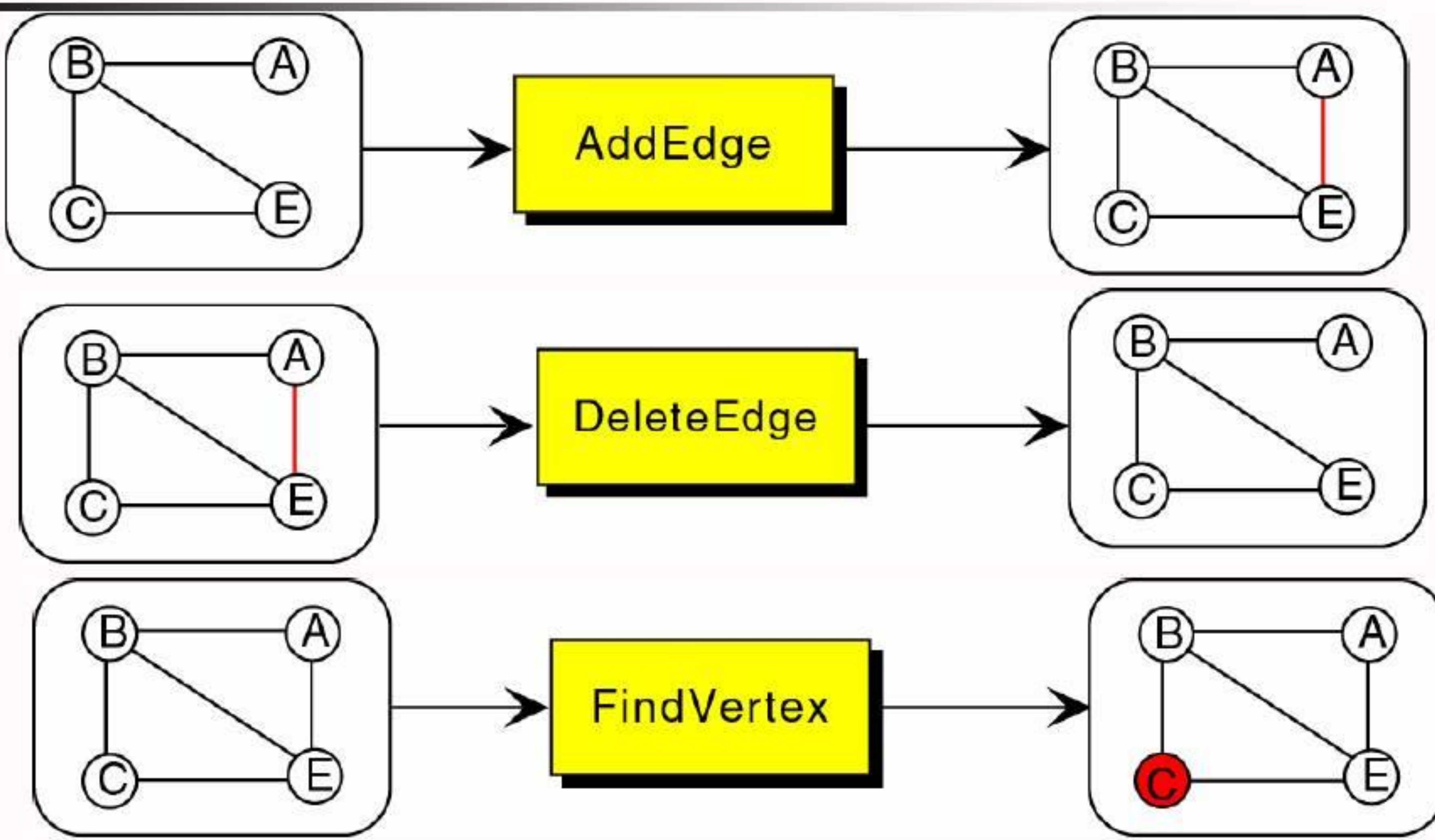


Fig.11-6,7,8



B
A
C
E
AddEdge
B
A
C
E
B
A
C
E
DeleteEdge
B
A
C
E
B
A
C
E
FindVertex
B
A
C
E

Traversals

- New challenge – a vertex in a graph having multiple parents
  - Possible to have different paths to a vertex
- Goals – to assure processing the data
  - To use a “visited flag”
    - Set all flags off initially
    - Set the flag of a vertex on when paying a visit

Depth-First Traversal

- All of a vertex's *descendants* are processed before we move to an adjacent vertex.
- Steps
  - Process the first vertex of the graph
  - Select any vertex adjacent to the first vertex and process
  - Select and process any adjacent vertex until reaching a vertex with no adjacent vertex

Fig.11-9 Depth First Traversal

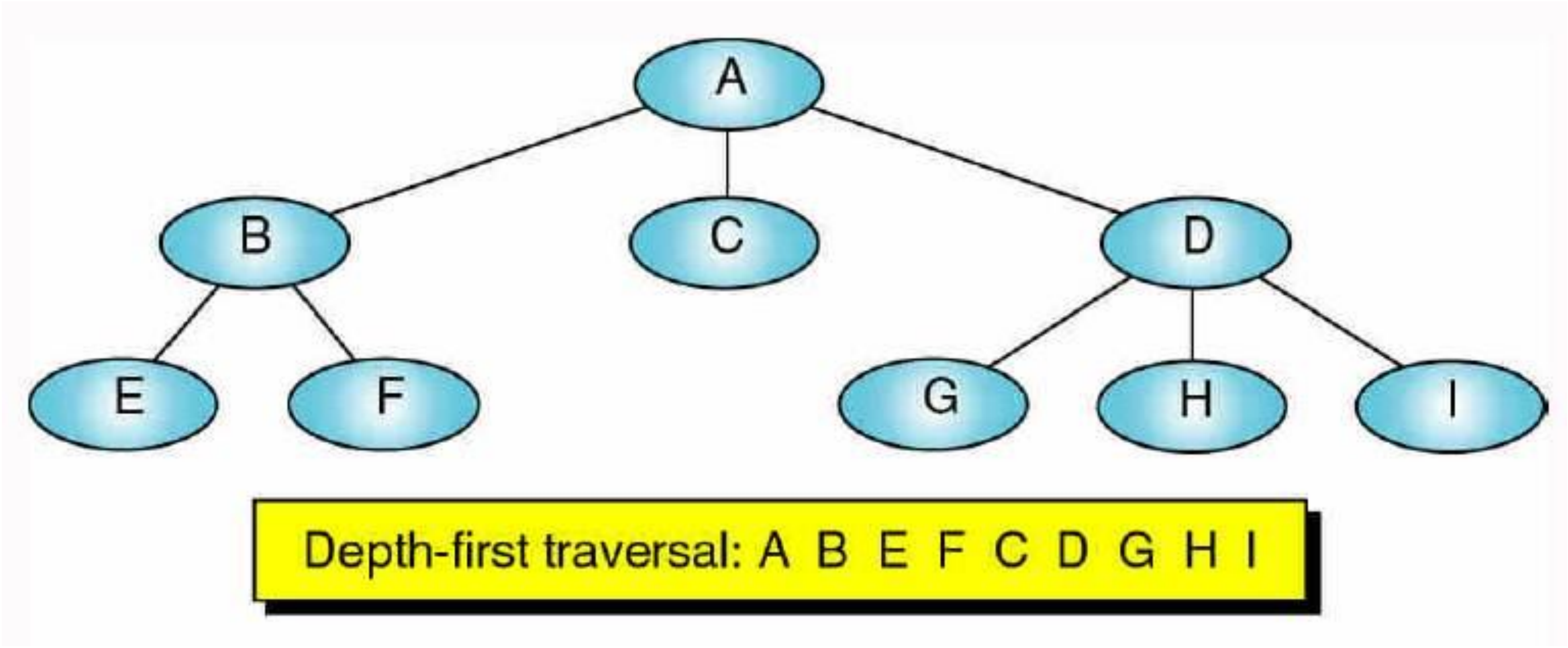
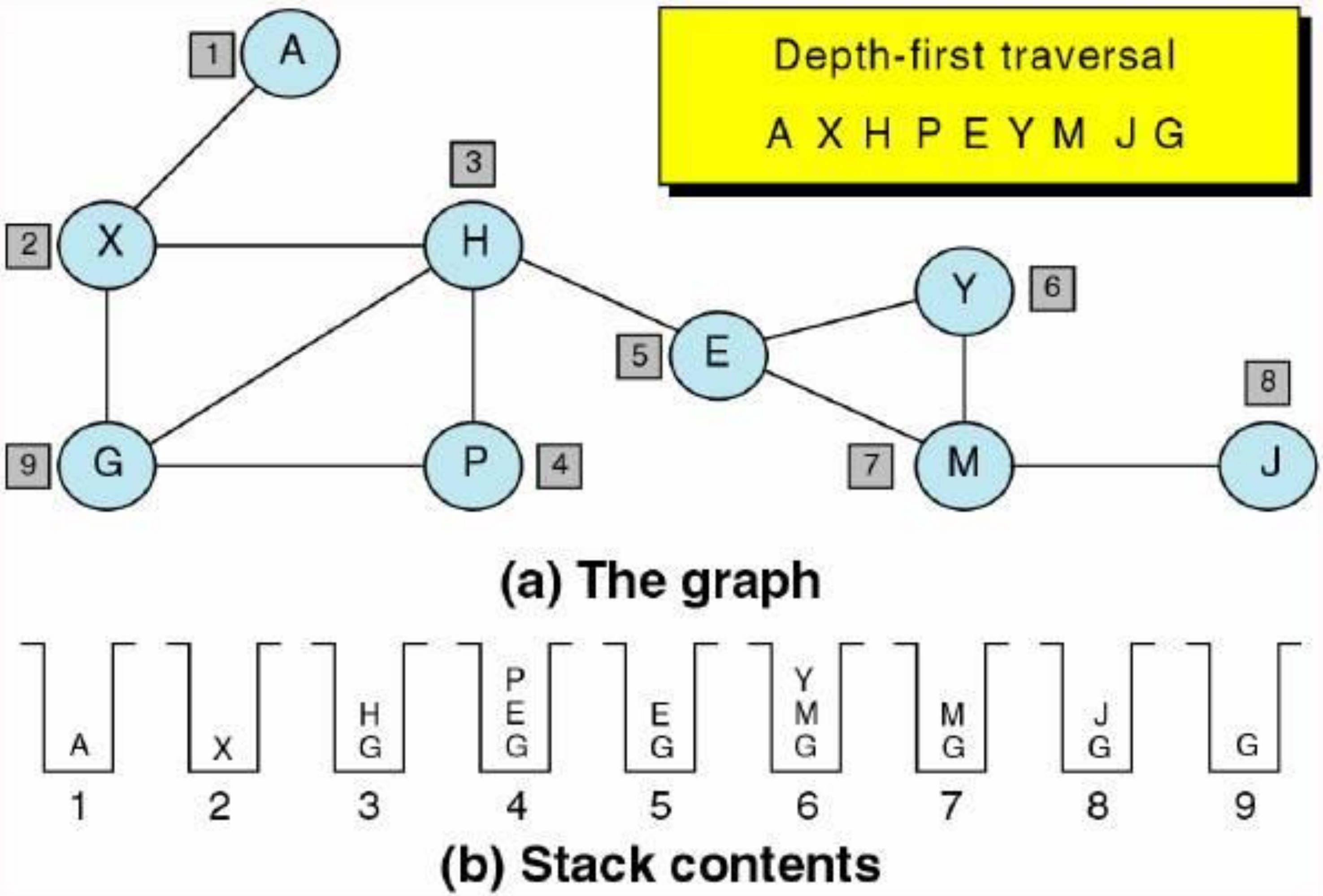


Fig.11-10 DFT with a Stack



Breadth-First Traversal

- All adjacent vertices of a vertex are processed before going to the next level.
- Steps
  - Process the first vertex of the graph
  - Process all the of its adjacent vertices
  - Select the each adjacent vertex and process vertices adjacent it.
- Repeat …

Fig.11-11 Breadth-First Traversal

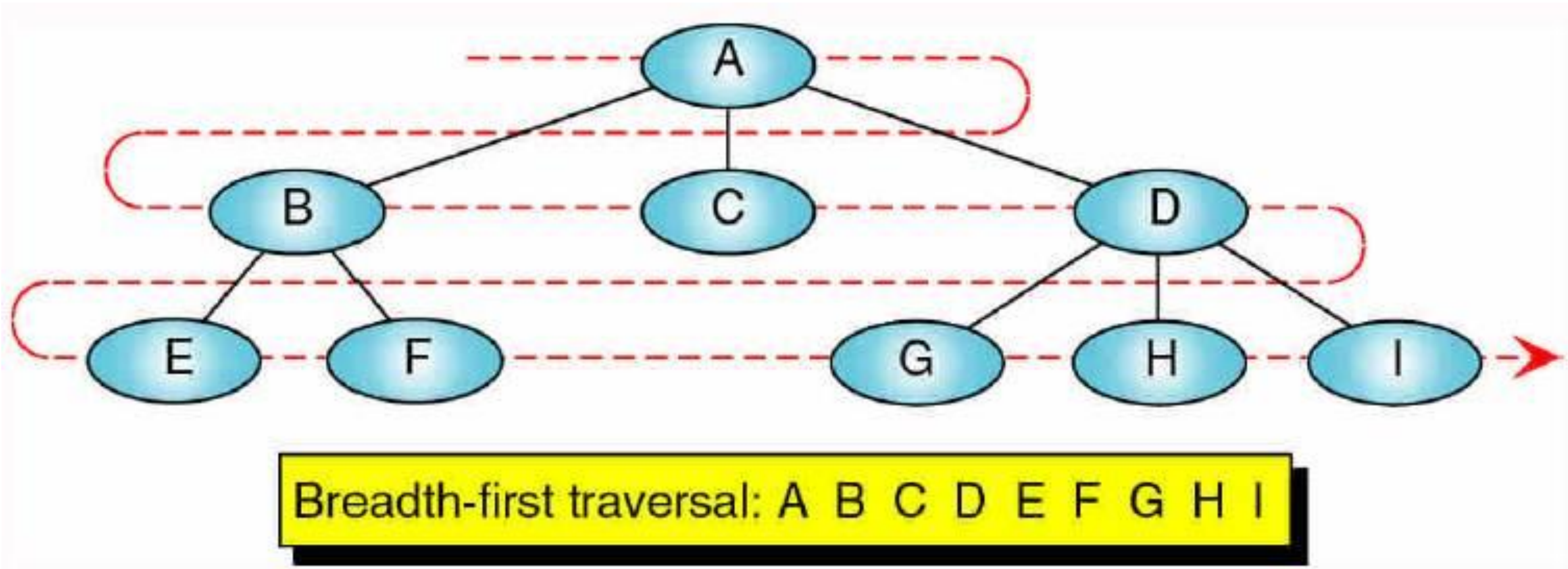
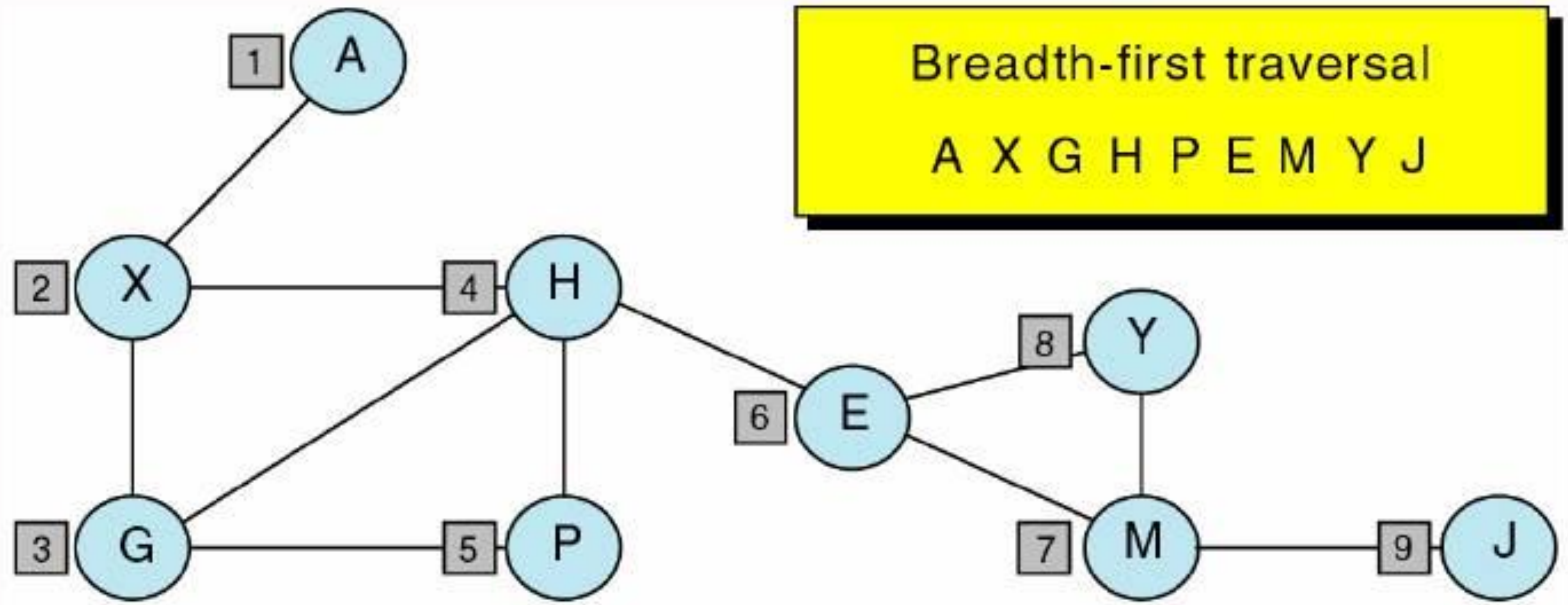
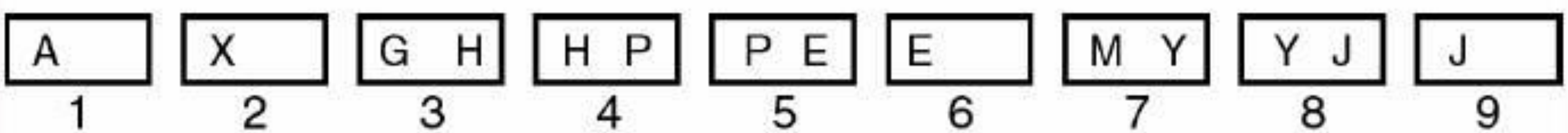


Fig.11-12 BFT with a Queue



(a) The graph

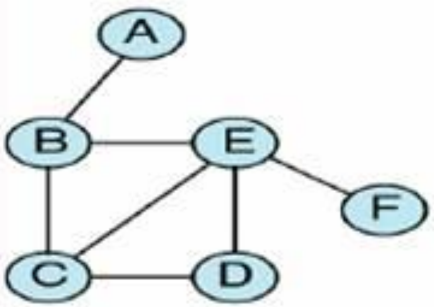


(b) Queue contents

Graph Storage Structures

- Need to store two sets
  - Vertices
  - Edges
- Data structures
  - Adjacent matrix
  - Adjacent list

Fig.11-13 Adjacent Matrix



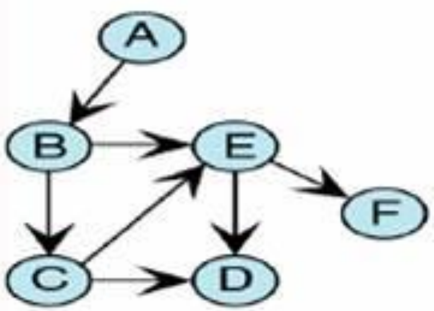
Vertex vector
A
B
C
D
E
F

Vertex vector

	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

Adjacency matrix

(a) Adjacency matrix for non-directed graph



Vertex vector
A
B
C
D
E
F

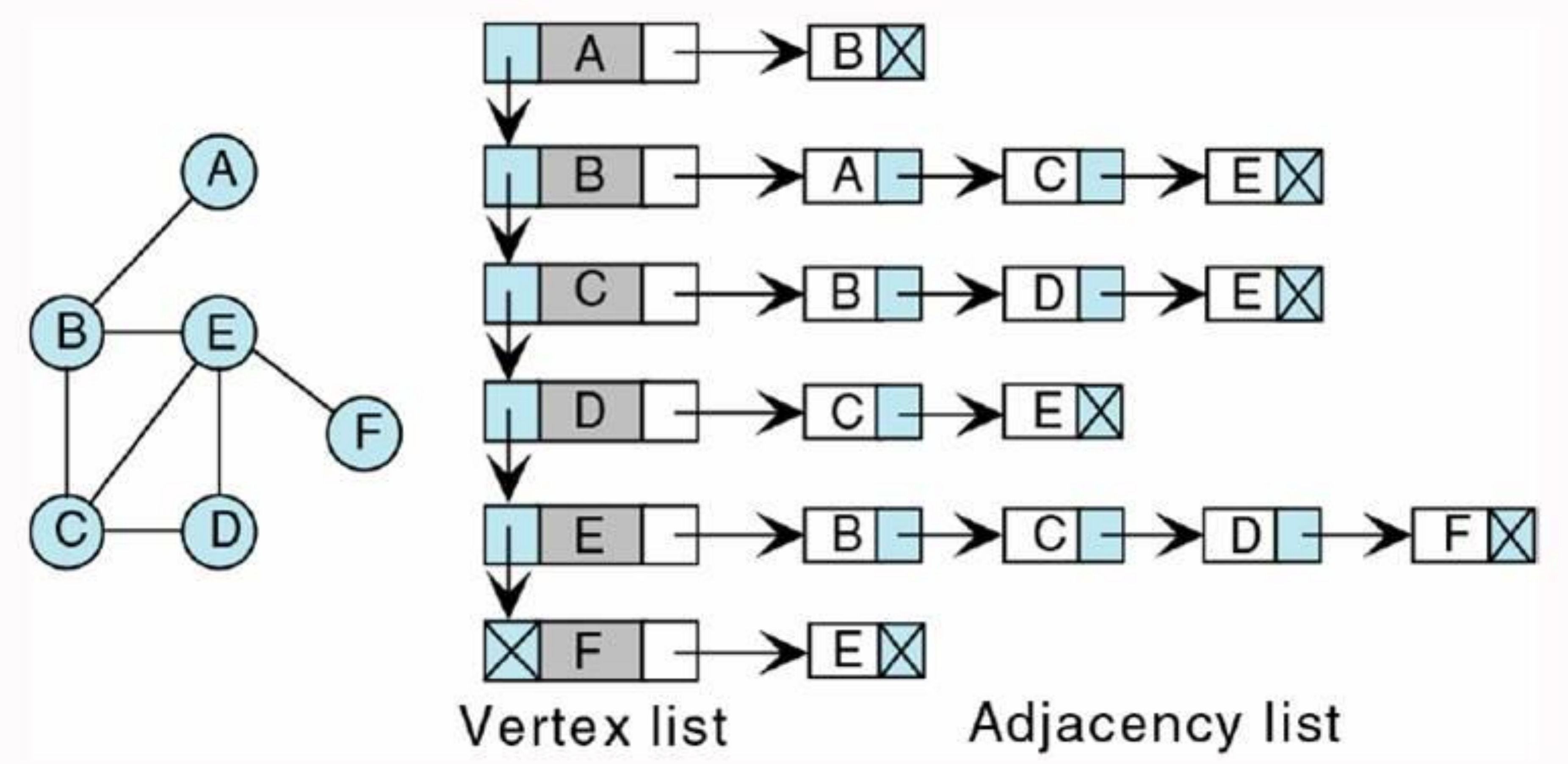
Vertex vector

	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

Adjacency matrix

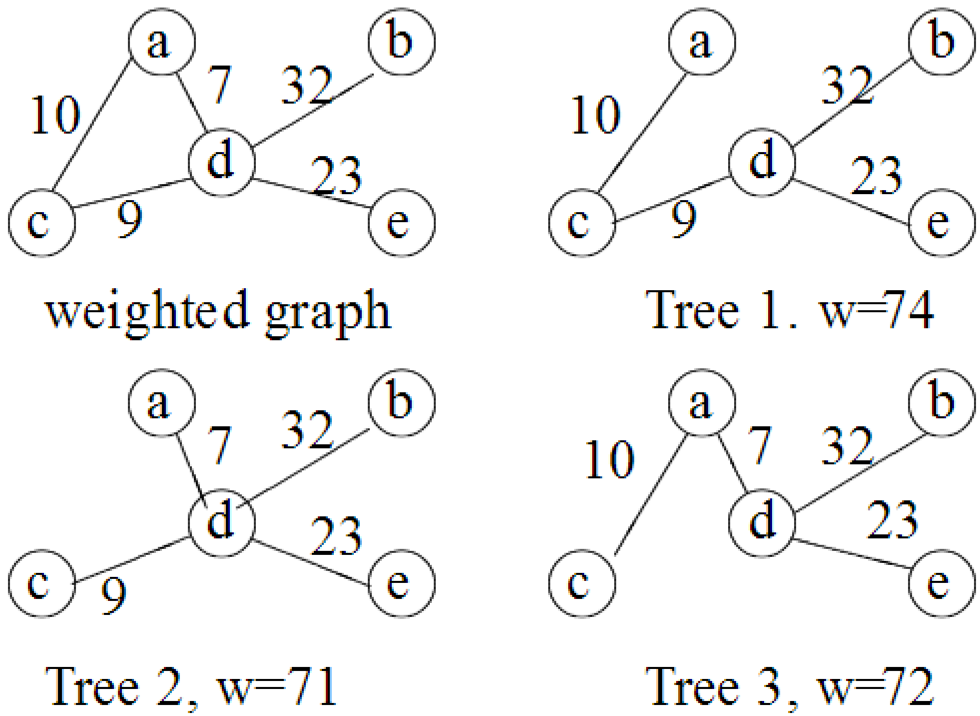
(a) Adjacency matrix for directed graph

Fig.11-14 Adjacent List



Weighted Graphs

- A weighted graph is a graph, in which each edge has a weight (some real number).
- **Weight of a Graph:** The sum of the weights of all edges



Networks

- Graph with weighted arcs –Weighted Graph
- Fig.11-18

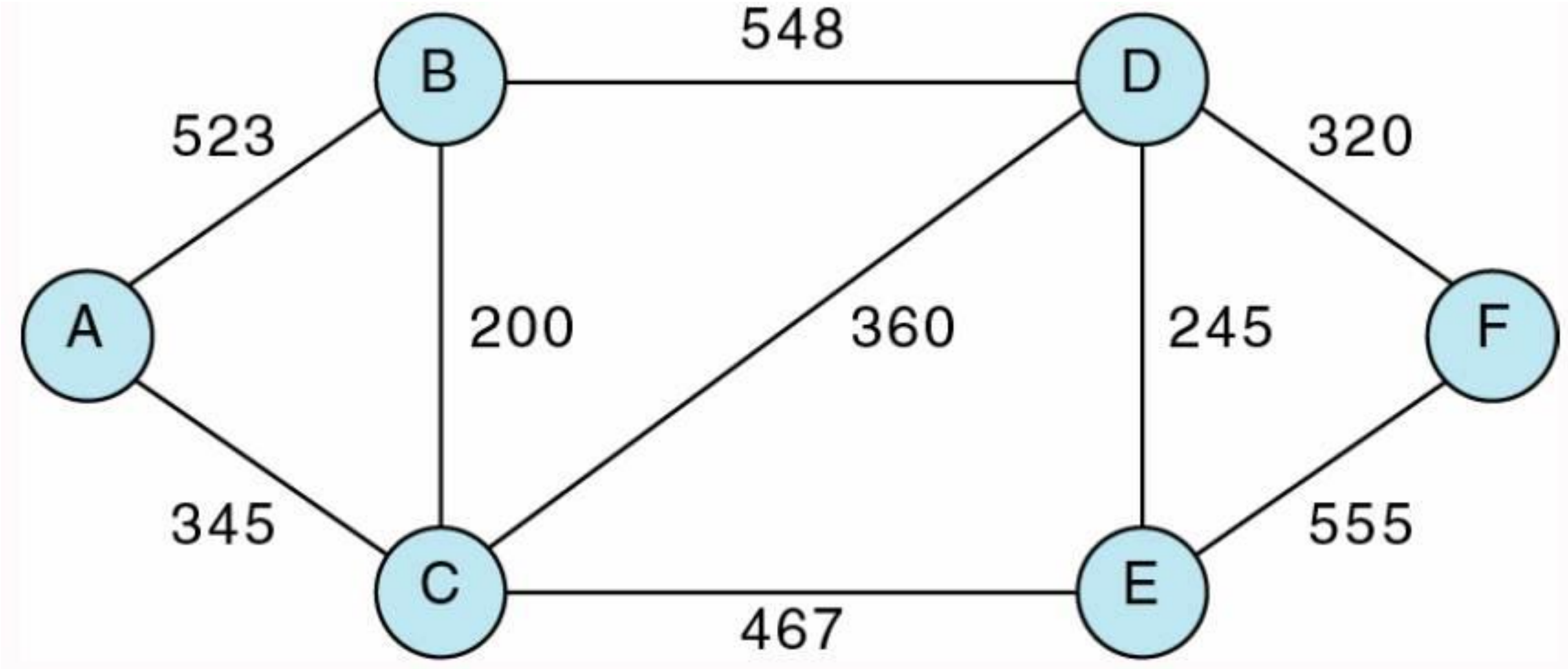
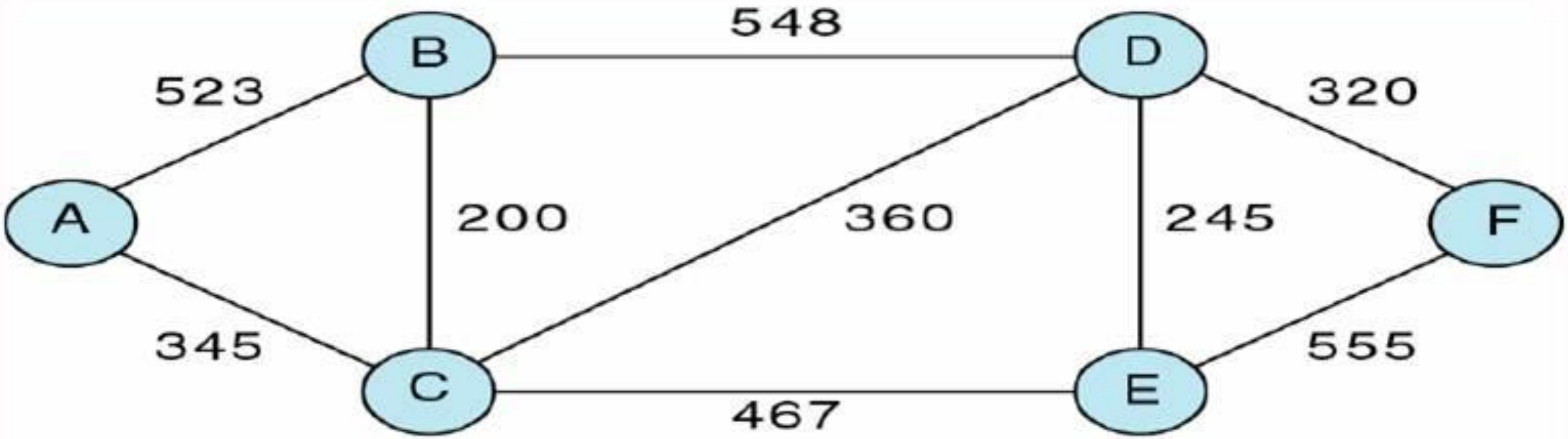


Fig.11-19 Adjacency Matrix



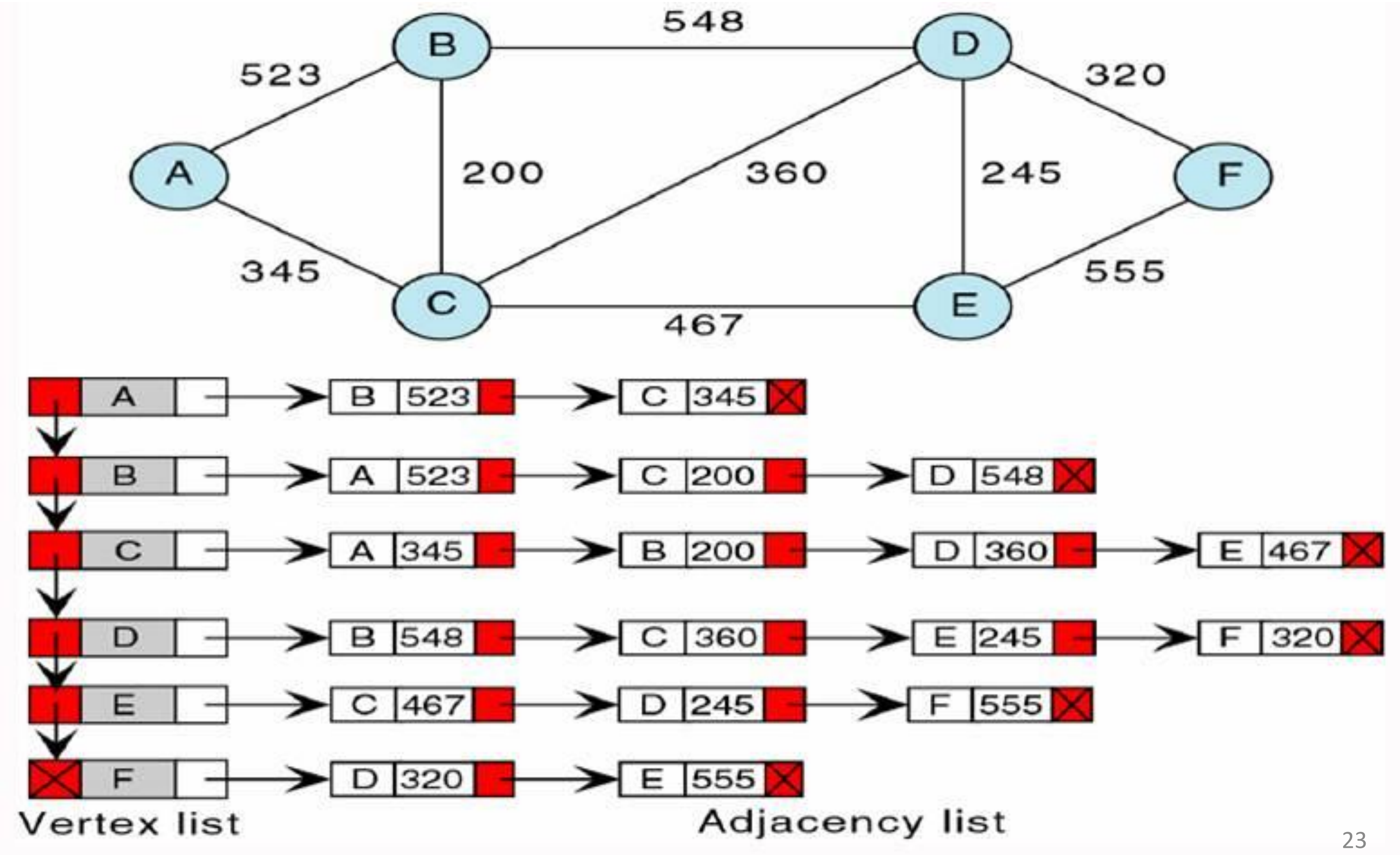
A
B
C
D
E
F

Vertex vector

	A	B	C	D	E	F
A	0	523	345	0	0	0
B	523	0	200	548	0	0
C	345	200	0	360	467	0
D	0	548	360	0	245	320
E	0	0	467	245	0	555
F	0	0	0	320	555	0

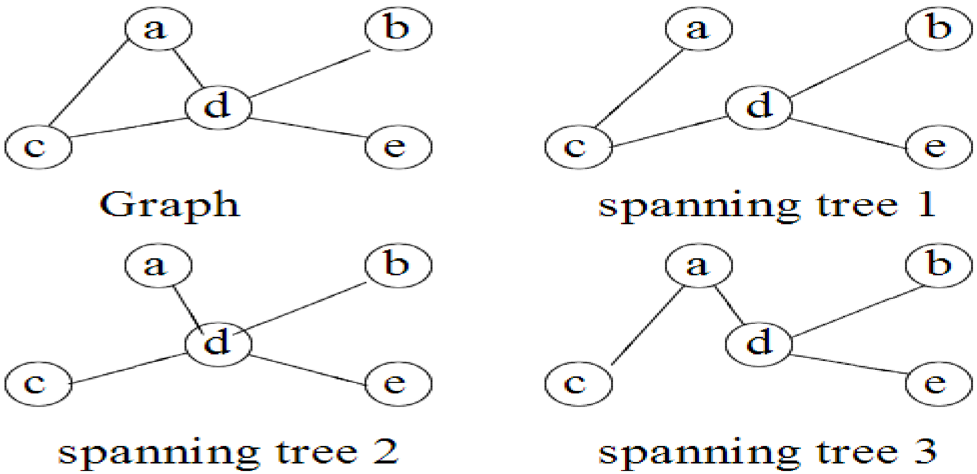
Adjacency matrix

Fig.11-19 Adjacency List



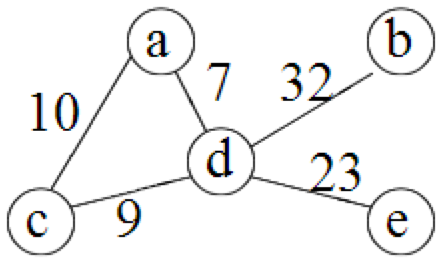
Minimum Spanning Tree

- Spanning Tree
  - A subgraph of a connected undirected graph G is called spanning tree if the tree containing all of the vertices in the graph

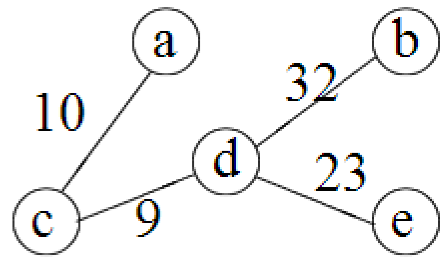


Minimum Spanning Tree

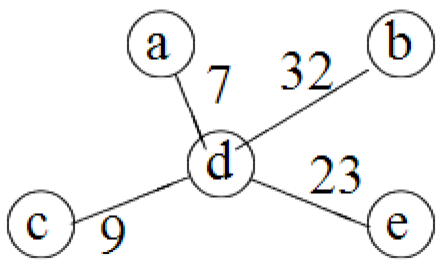
- A Minimum Spanning Tree in an undirected connected weighted graph is a spanning tree of minimum weight (among all spanning trees).



weighted graph

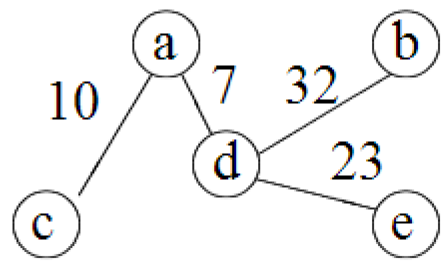


Tree 1. w=74



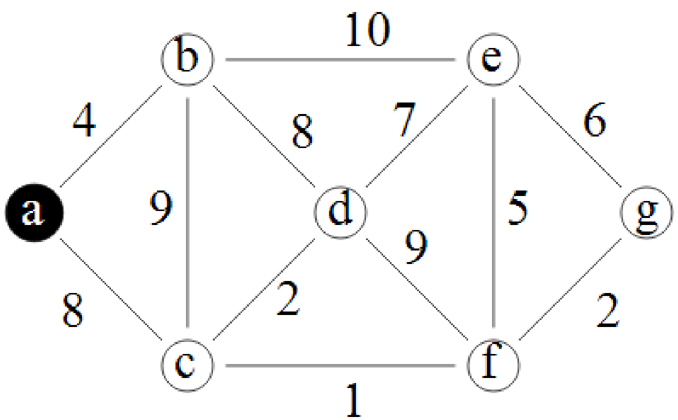
Tree 2, w=71

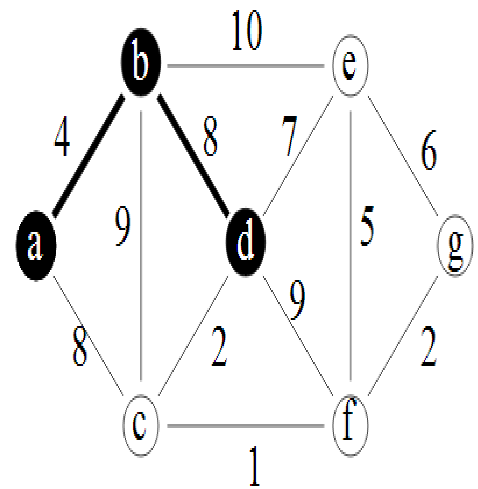
Minimum spanning tree

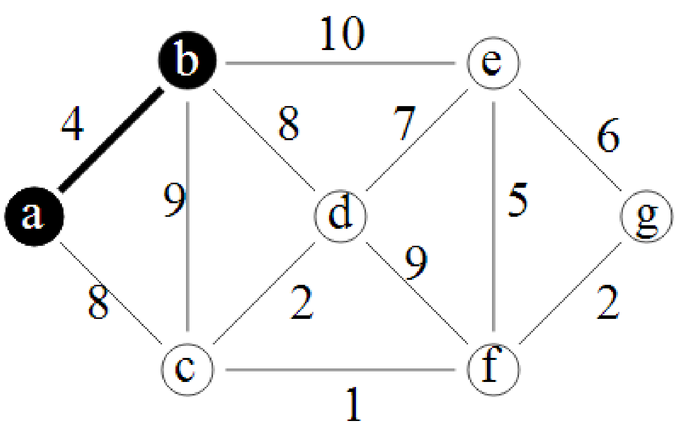


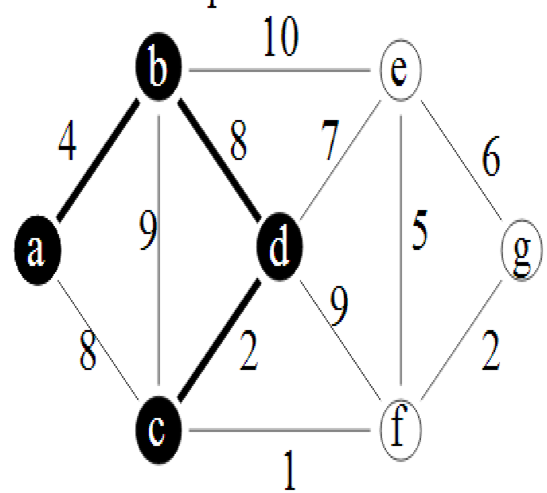
Tree 3, w=72

Minimum Spanning Tree

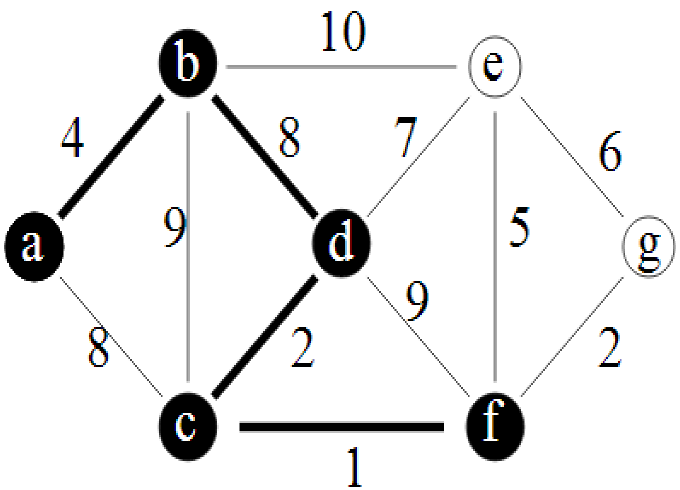


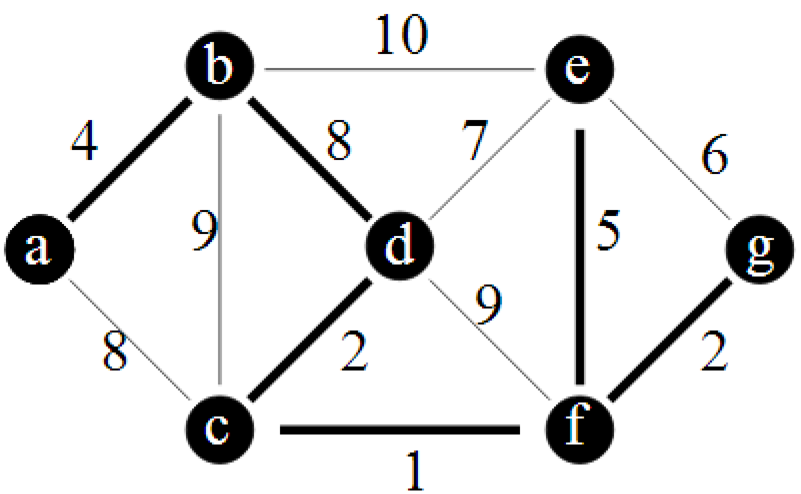


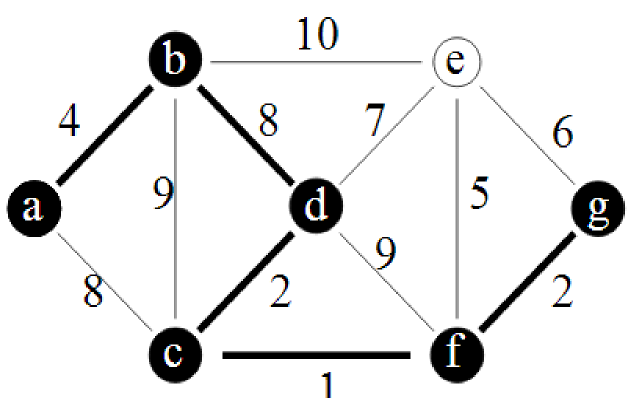




Minimum Spanning Tree







Shortest Path Algorithm

- Finding the shortest path between two vertices in a network
- Application: to find the least expensive route between home and our destination

Dijkstra's algorithm

notation

- $c_{x,y}$: direct link cost from node x to y; $= \infty$ if not direct neighbors
- $D(v)$: *current* estimate of cost of least-cost-path from source to destination v
- $p(v)$: predecessor node along path from source to v
- N': set of nodes whose least-cost-path *definitively* known

Dijkstra's algorithm

Initialization:

$N' = \{u\}$ /* compute least cost path from u to all other nodes */

for all nodes v

  if v adjacent to u /* u initially knows direct-path-cost only to direct neighbors */

    then $D(v) = c_{u,v}$ /* but may not be *minimum* cost! */

  else $D(v) = \infty$

Loop

  find w not in N' such that $D(w)$ is a minimum

  add w to N'

  update $D(v)$ for all v adjacent to w and not in N' :

    $\boldsymbol{D(v) = \min(D(v),\ D(w) + c_{w,v})}$

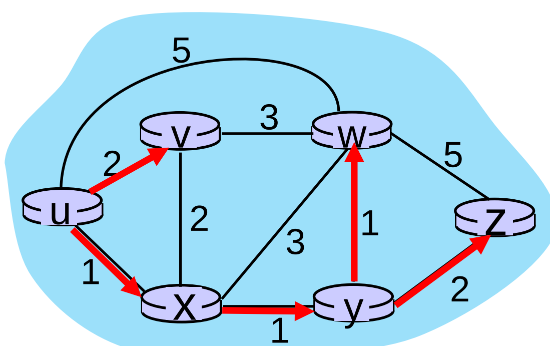
  /* new least-path-cost to v is either old least-cost-path to v or known

  least-cost-path to w plus direct-cost from w to v */

until all nodes in N'

Dijkstra's algorithm: an example

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



Initialization (step 0): For all a: if *a* adjacent to then $D(a) = c_{u,a}$

find *a* not in *N'* such that *D(a)* is a minimum
add *a* to *N'*
update *D(b)* for all *b* adjacent to *a* and not in *N'* :
$D(b) = \min(D(b), D(a) + c_{a,b})$

Applications of Graph Theory

- Graph theory has its applications in diverse fields of engineering −
- **Electrical Engineering** − The concepts of graph theory is used extensively in designing circuit connections. The types or organization of connections are named as topologies. Some examples for topologies are star, bridge, series, and parallel topologies.
- **Computer Science** − Graph theory is used for the study of algorithms. For example,
  - Kruskal's Algorithm
  - Prim's Algorithm
  - Dijkstra's Algorithm
- **Computer Network** − The relationships among interconnected computers in the network follows the principles of graph theory.
- **Science** − The molecular structure and chemical structure of a substance, the DNA structure of an organism, etc., are represented by graphs.
- **Linguistics** − The parsing tree of a language and grammar of a language uses graphs.
- **General** − Routes between the cities can be represented using graphs. Depicting hierarchical ordered information such as family tree can be used as a special type of graph called tree.