Chapter 16
– Graphs

Graph Categories

Example of Digraph

Connectedness of Digraph

Adjacency Matrix

Adjacency Set

vertexInfo Object

Vertex Map and Vector vInfo

VtxMap and Vinfo Example

Breadth-First Search Algorithm (2 slides)

Dfs()

Strong Components

Graph G and Its Transpose G^T

Shortest-Path Example (2 slides)

Dijkstra Minimum-Path Algorithm (2 slides)

Minimum Spanning Tree Example

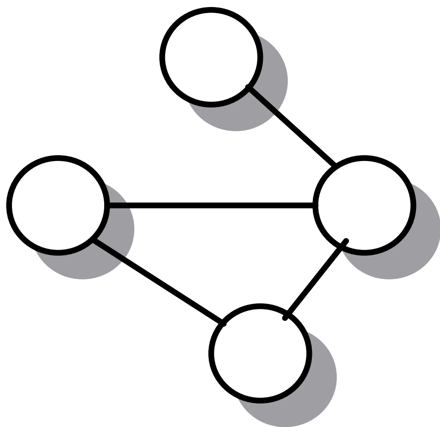
Minimum Spanning Tree: vertices A and B

Completing the Minimum Spanning-Tree with Vertices C and D

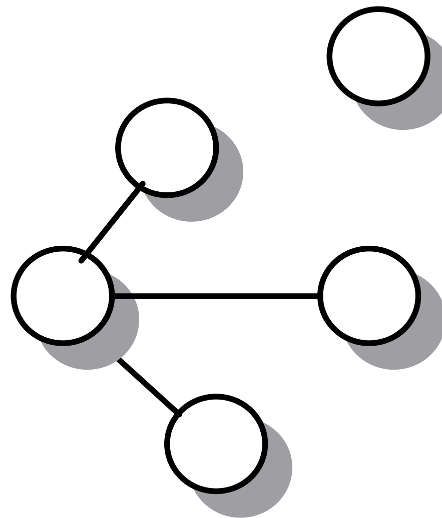
Summary Slides (4 slides)

Graph Categories

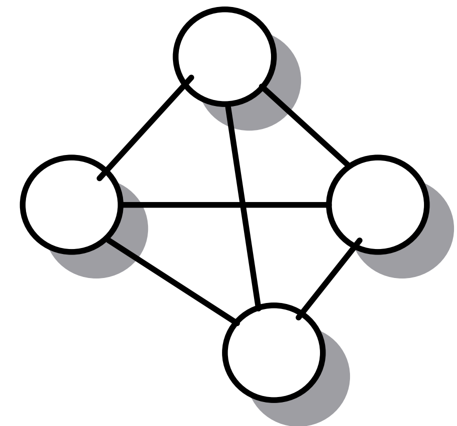
- A graph is *connected* if each pair of vertices have a path between them
- A *complete graph* is a connected graph in which each pair of vertices are linked by an edge



(a: Connected)



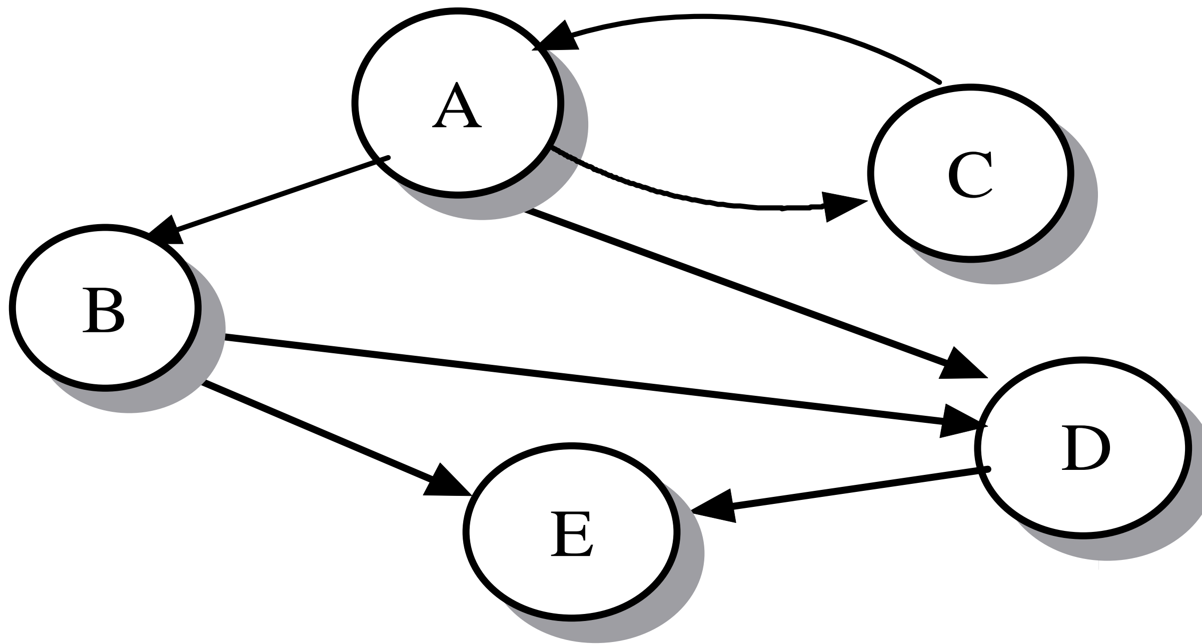
(b: Disconnected)



(c: Complete)

Example of Digraph

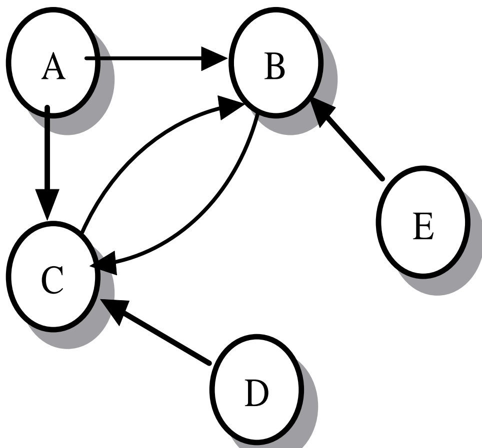
- Graph with ordered edges are called *directed graphs* or *digraphs*



Vertices V = {A, B, C, D, E}
Edges E = {(A,B), (A,C), (A,D), (B,D), (B,E), (C,A), (D,E)}

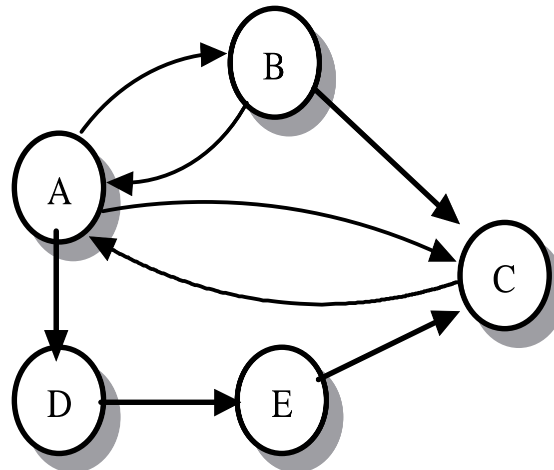
Connectedness of Digraph

- *Strongly connected* if there is a path from any vertex to any other vertex.
- *Weakly connected* if, for each pair of vertices vi and vj, there is either a path P(vi, vj) or a path P(vi,vj).



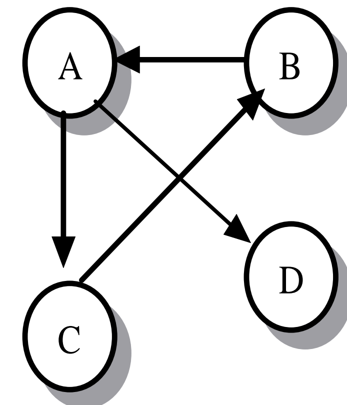
Not Strongly or Weakly Connected
(No path E to D or D to E)

(a)



Strongly Connected

(b)

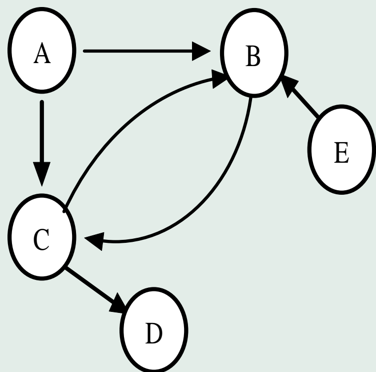


Weakly Connected
(No path from D to a vertex)

(c)

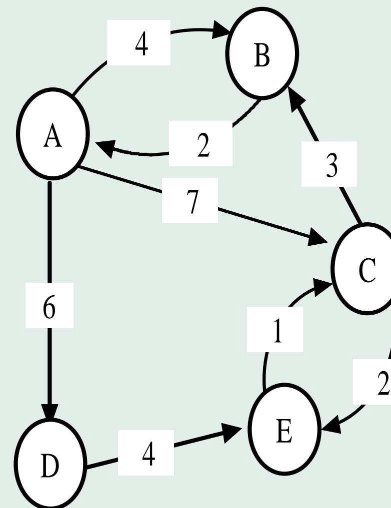
Adjacency Matrix

- An m by m matrix, called an *adjacency matrix*, identifies the edges. An entry in row i and column j corresponds to the edge e = (v_i, v_j). Its value is the weight of the edge, or -1 if the edge does not exist.



	A	B	C	D	E
A	-1	1	1	-1	-1
B	-1	-1	1	-1	-1
C	-1	1	-1	1	-1
D	-1	-1	-1	-1	-1
E	-1	1	-1	-1	-1

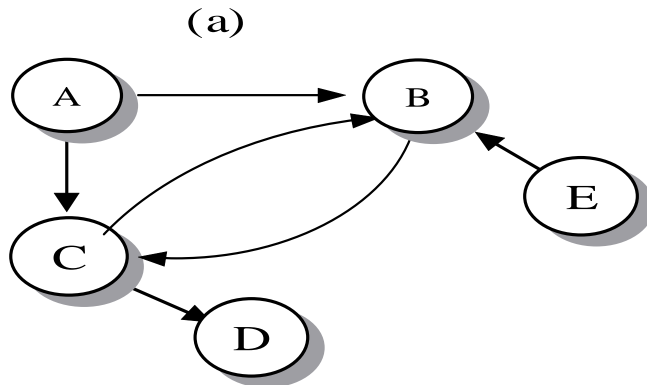
(a)



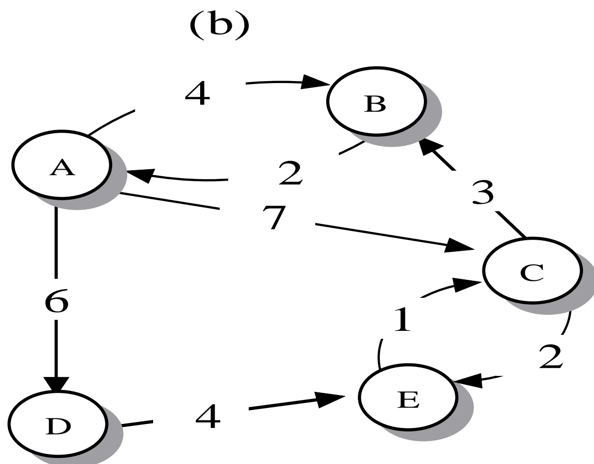
	A	B	C	D	E
A	-1	4	7	6	-1
B	2	-1	-1	-1	-1
C	-1	3	-1	-1	2
D	-1	-1	-1	-1	4
E	-1	-1	1	-1	-1

(b)

Adjacency Set



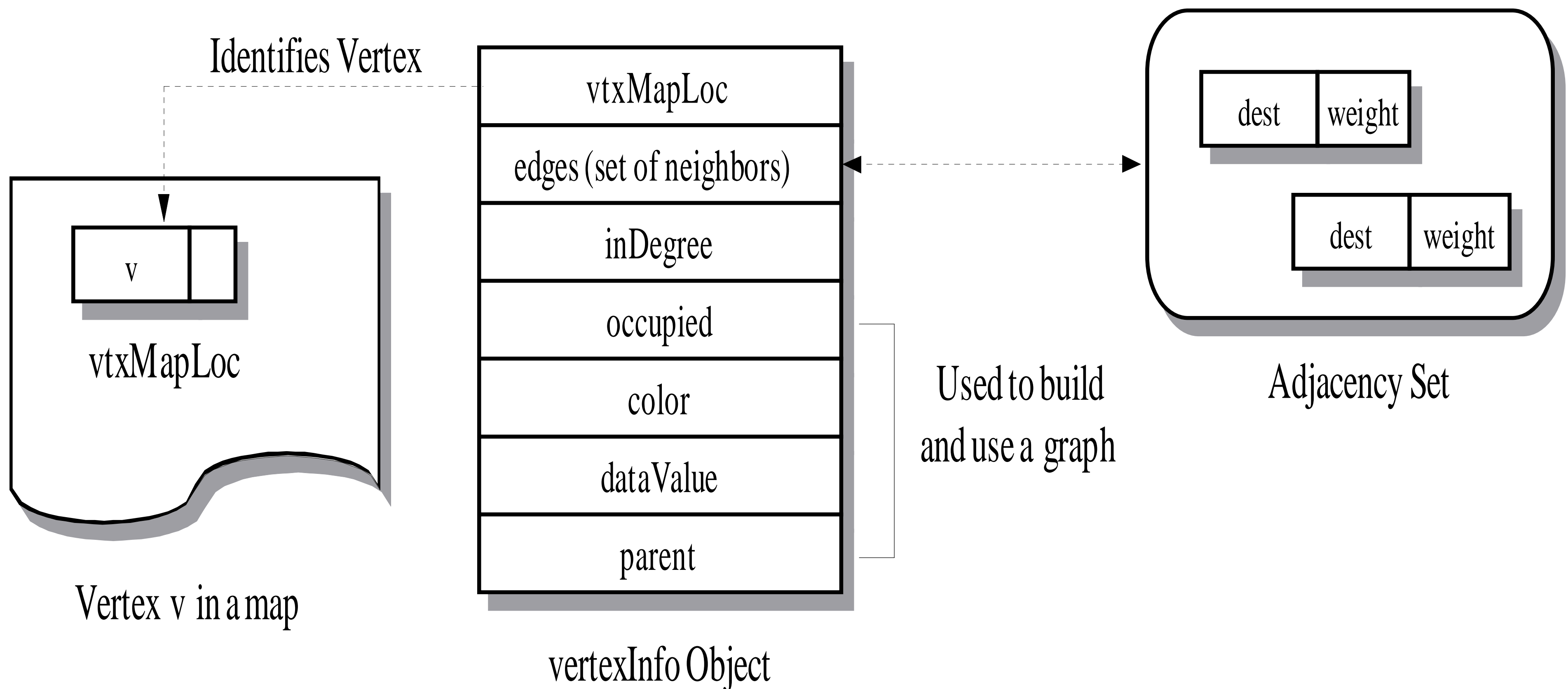
Vertices	Set of Neighbors	
A	B 1	C 1
B	C 1	
C	B 1	D 1
D		
E	B 1	



Vertices	Set of Neighbors		
A	B 4	C 7	D 6
B	A 2		
C	B 3	E 2	
D	E 4		
E	C 1		

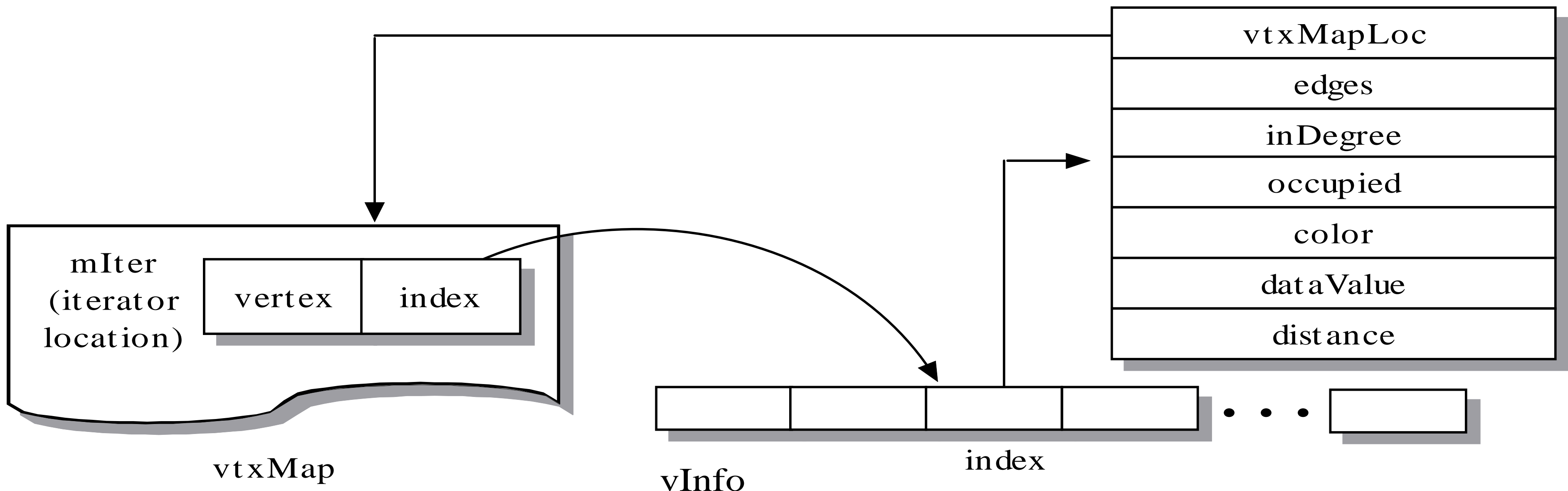
vertexInfo Object

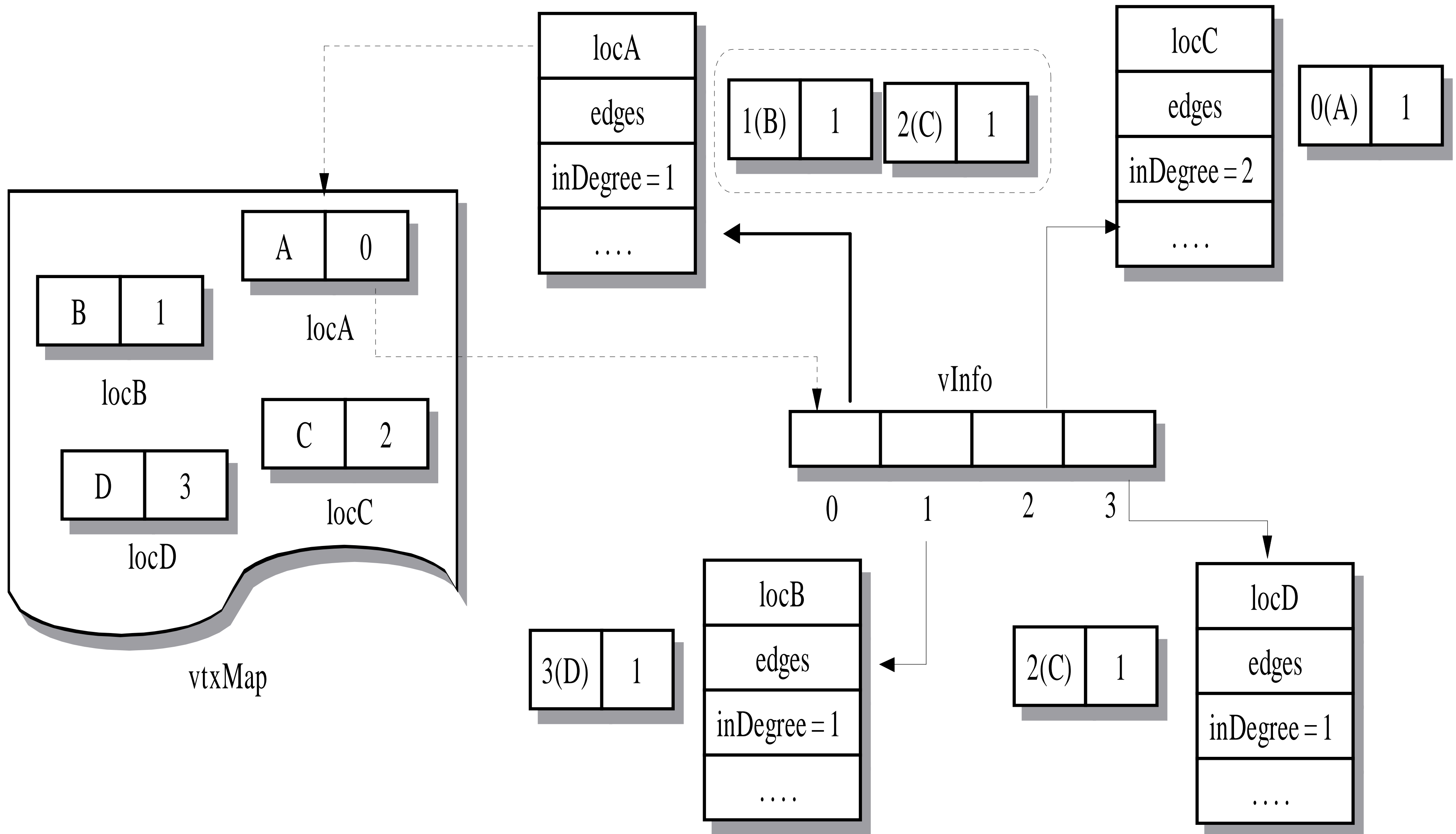
- A vertexInfo object consists of seven data members. The first two members, called vtxMapLoc and edges, identify the vertex in the map and its adjacency set.



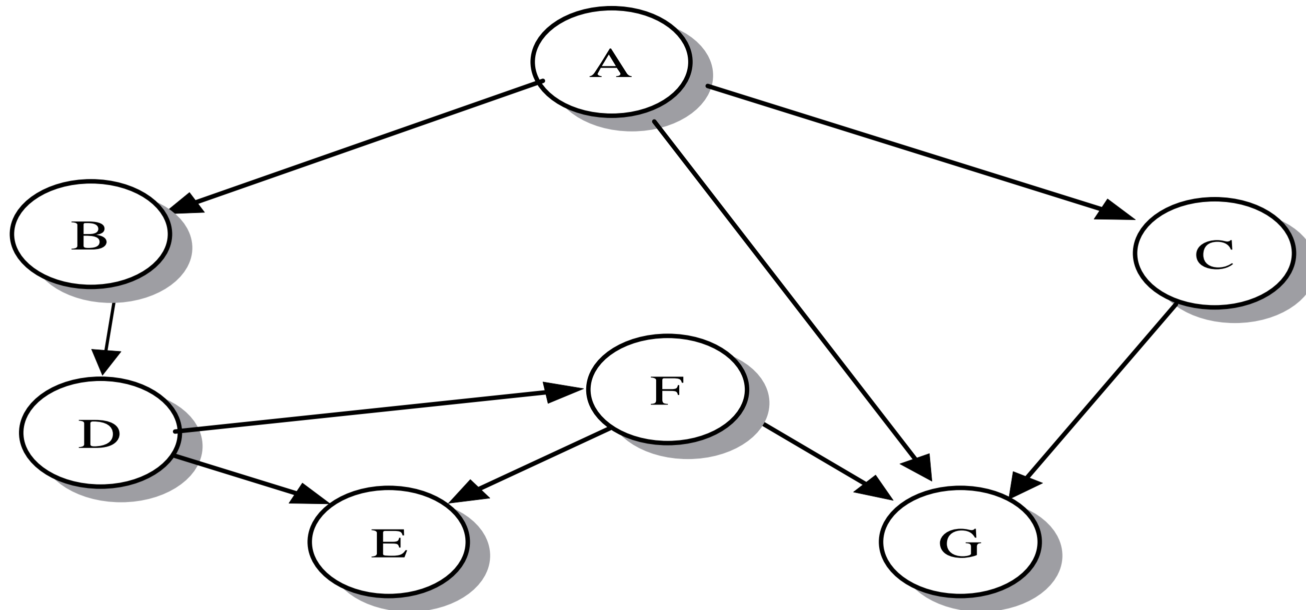
Vertex Map and Vector vInfo

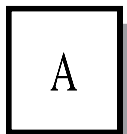
- To store the vertices in a graph, we provide a map<T,int> container, called vtxMap, where a vertex name is the key of type T. The int field of a map object is an index into a vector of vertexInfo objects, called vInfo. The size of the vector is initially the number of vertices in the graph, and there is a 1-1 correspondence between an entry in the map and a vertexInfo entry in the vector

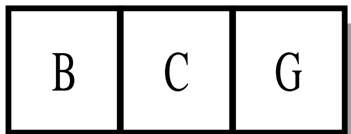



locA
edges
inDegree = 1
. . . .
1(B)
1
2(C)
1
locC
edges
inDegree = 2
. . . .
0(A)
1
A
0
locA
B
1
locB
C
2
locC
D
3
locD
vtxMap
vInfo
0
1
2
3
3(D)
1
locB
edges
inDegree = 1
. . . .
2(C)
1
locD
edges
inDegree = 1
. . . .

Breadth-First Search Algorithm

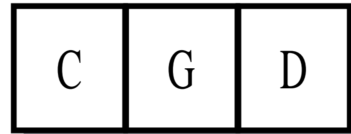


visitSet 

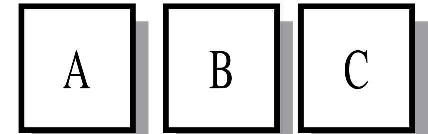
visitQueue 

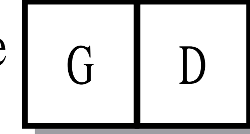
(a)

visitSet

visitQueue 

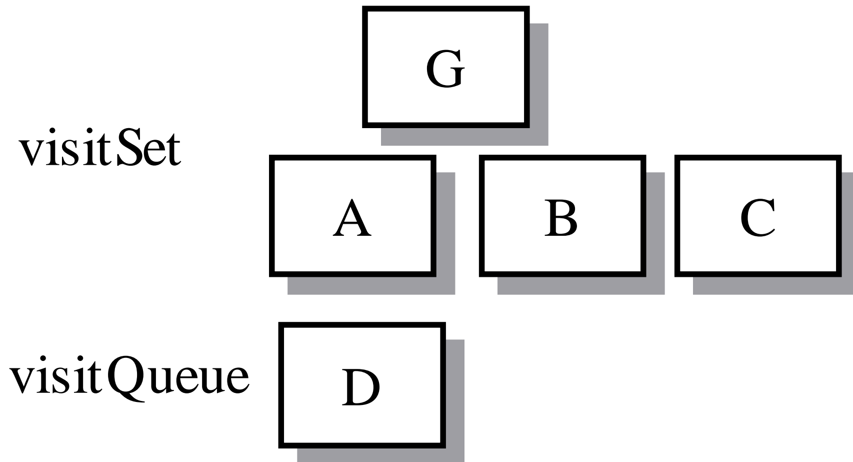
(b)

visitSet 

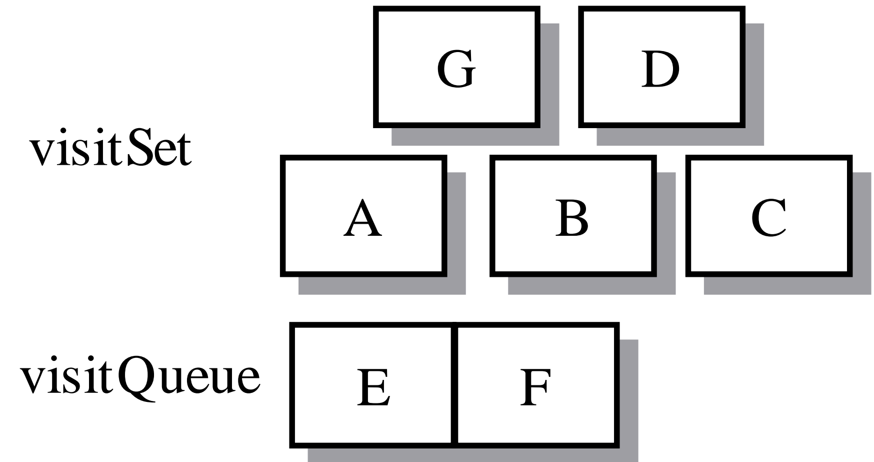
visitQueue 

(c)

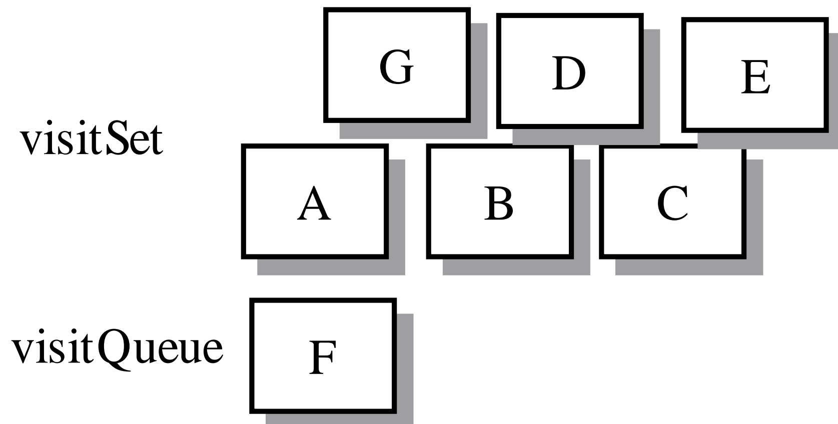
Breadth-First Search… (Cont.)



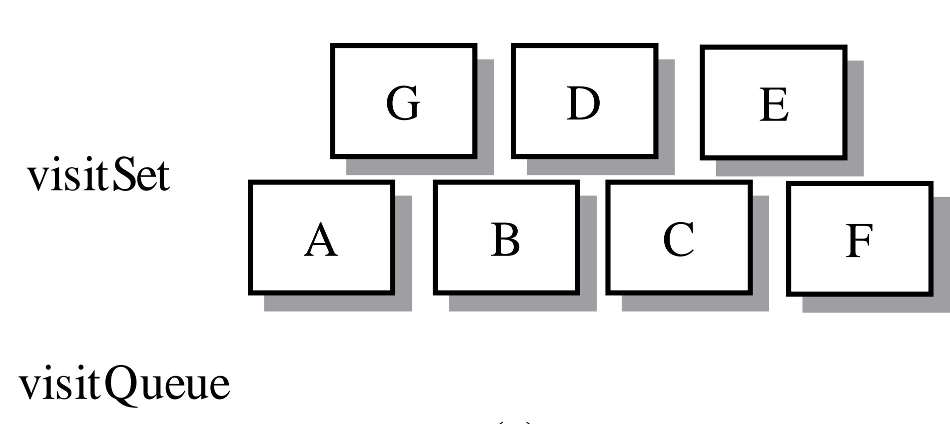
(d)



(e)

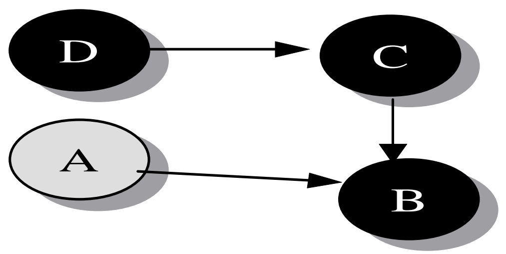


(f)



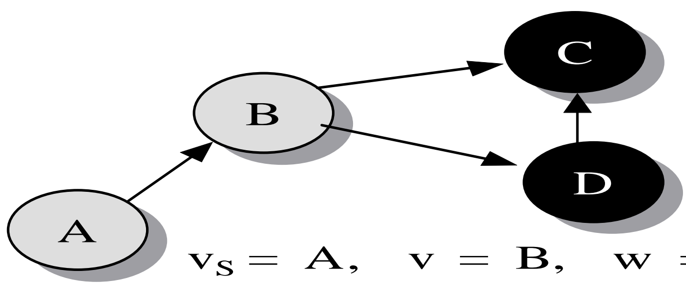
(g)

dfs()



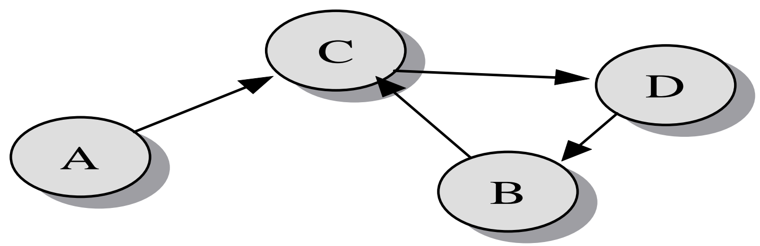
v = A, w = B
First call to dfsVisit() starts at D.
Second call to dfsVisit() starts at A.
dfsList: A D C B

(a)



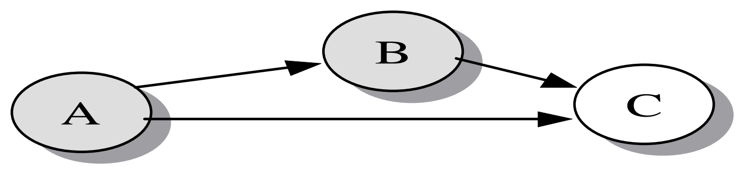
v_S = A, v = B, w = C
dfsVisit() starts at B, discovers neighbor D, and then neighbor C.
dfsList: A B D C

(b)



v_S = A, v = B, w = C
dfsVisit() starts at A, discovers C, discovers D, and finally discovers B. The neighbor of B (C) is GRAY (a back edge).
dfsList: <empty>
path C D B C is a cycle

(c)

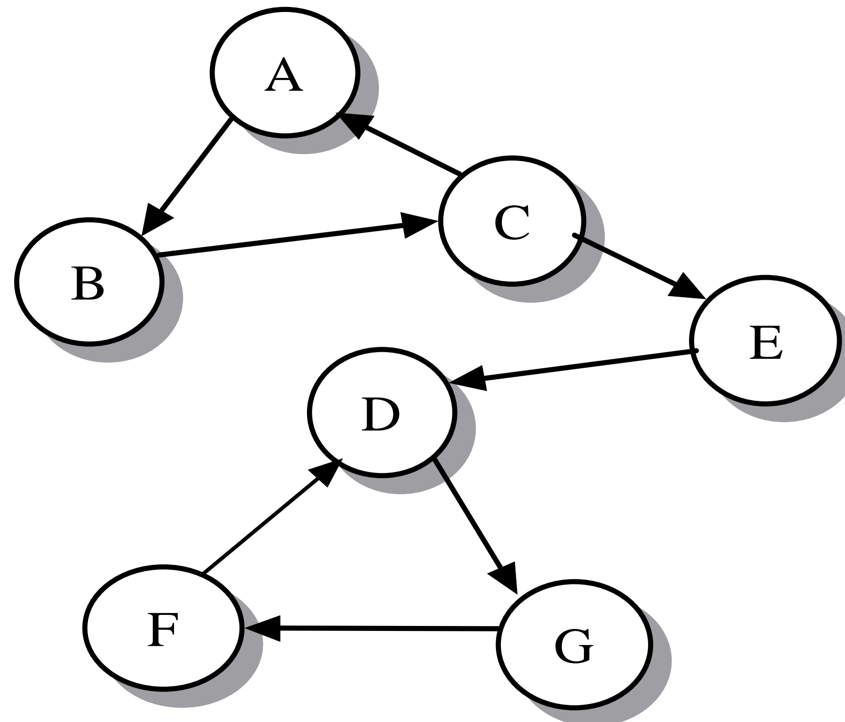


v_S = A, v = B, w = C
dfsVisit() starts at A and proceeds along the path from B to C.
dfsList: A B C

(d)

Strong Components

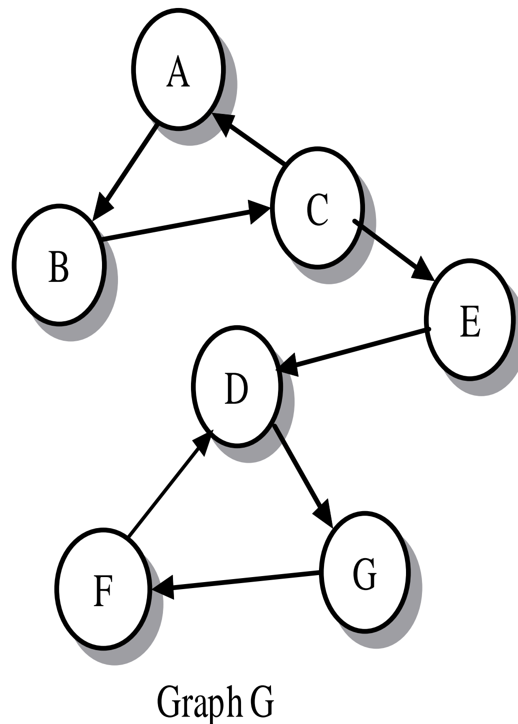
- A *strongly connected component* of a graph G is a maximal set of vertices SC in G that are mutually accessible.



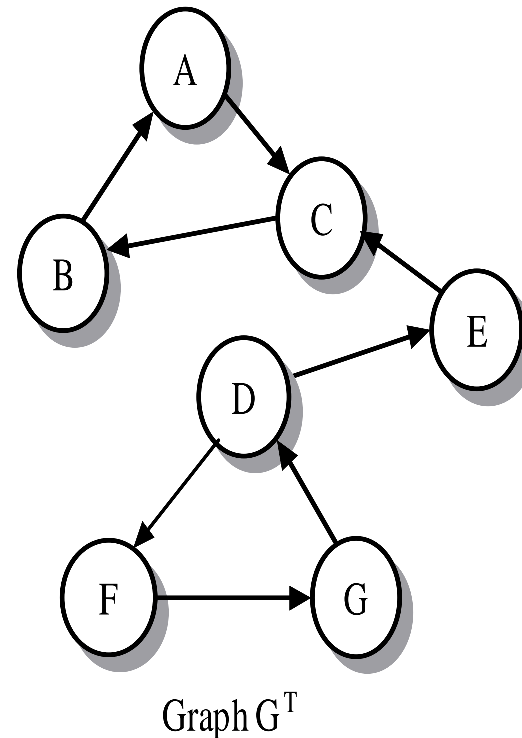
Components
A, B, C
D, F, G
E

Graph G and Its Transpose G^T

- The transpose has the same set of vertices V as graph G but a new edge set E^T consisting of the edges of G but with the opposite direction.



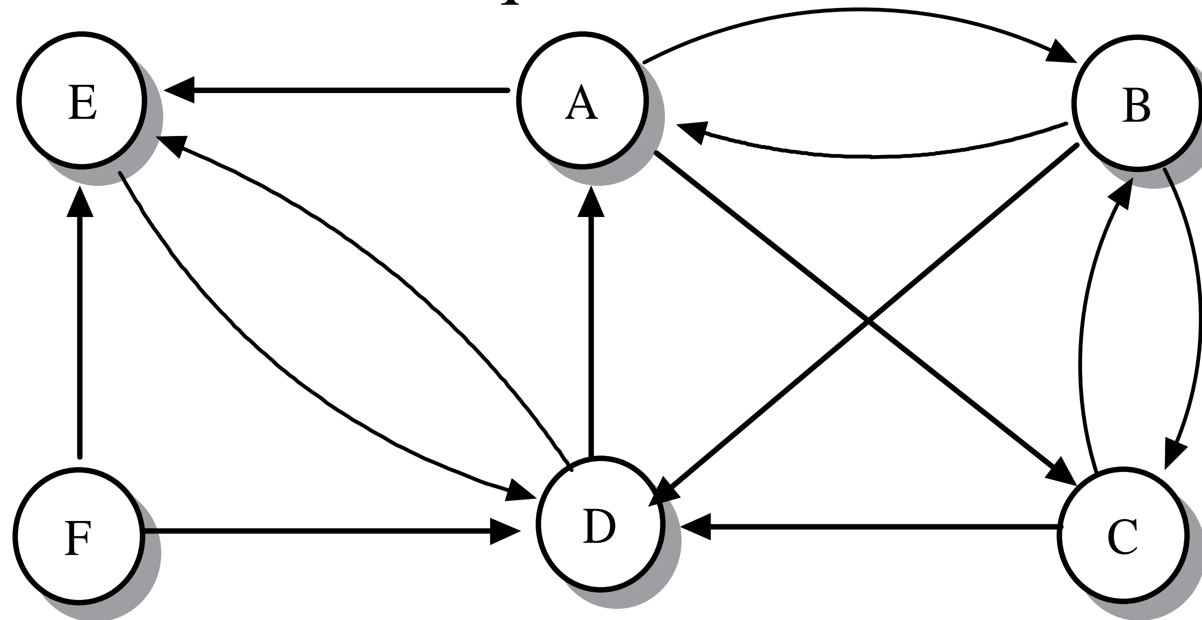
Graph G



Graph G^T

Shortest-Path Example

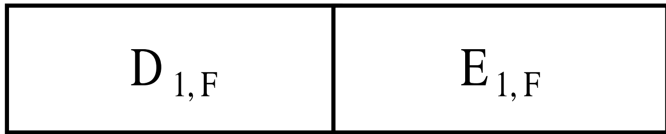
- The shortest-path algorithm includes a queue that indirectly stores the vertices, using the corresponding vInfo index. Each iterative step removes a vertex from the queue and searches its adjacency set to locate all of the unvisited neighbors and add them to the queue.



Shortest-Path Example

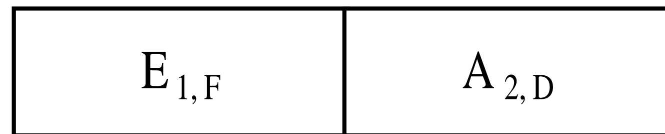
- Example: Find the shortest path for the previous graph from F to C.

visitQueue



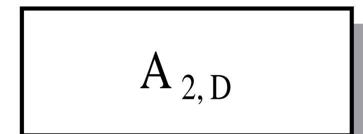
(a)

visitQueue



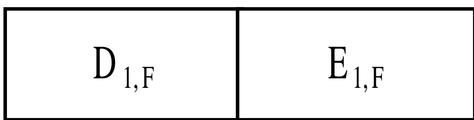
(b)

visitQueue



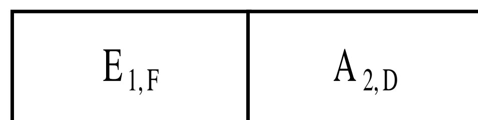
(c)

visitQueue



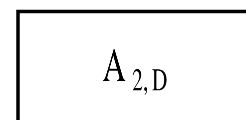
(a)

visitQueue



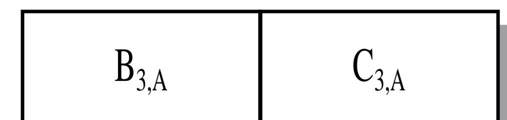
(b)

visitQueue



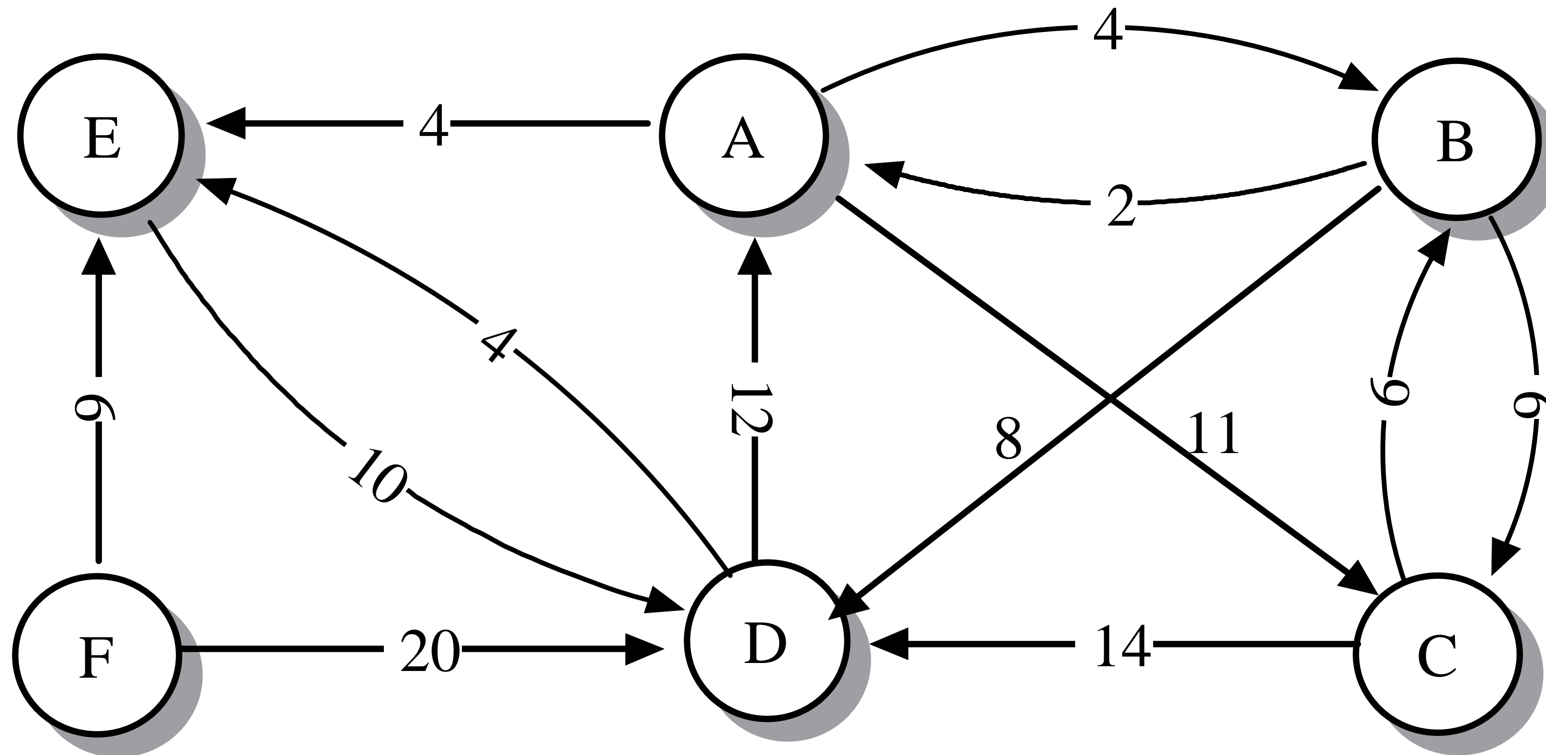
(c)

visitQueue



(d)

Dijkstra Minimum-Path Algorithm From A to D Example



minInfo(B,4)

minInfo(C,11)

minInfo(E,4)

priority queue

Dijkstra Minimum-Path Algorithm From… (Cont…)

minInfo(C,10) | minInfo(C,11) | minInfo(E,4) | minInfo(D,12)

priority queue

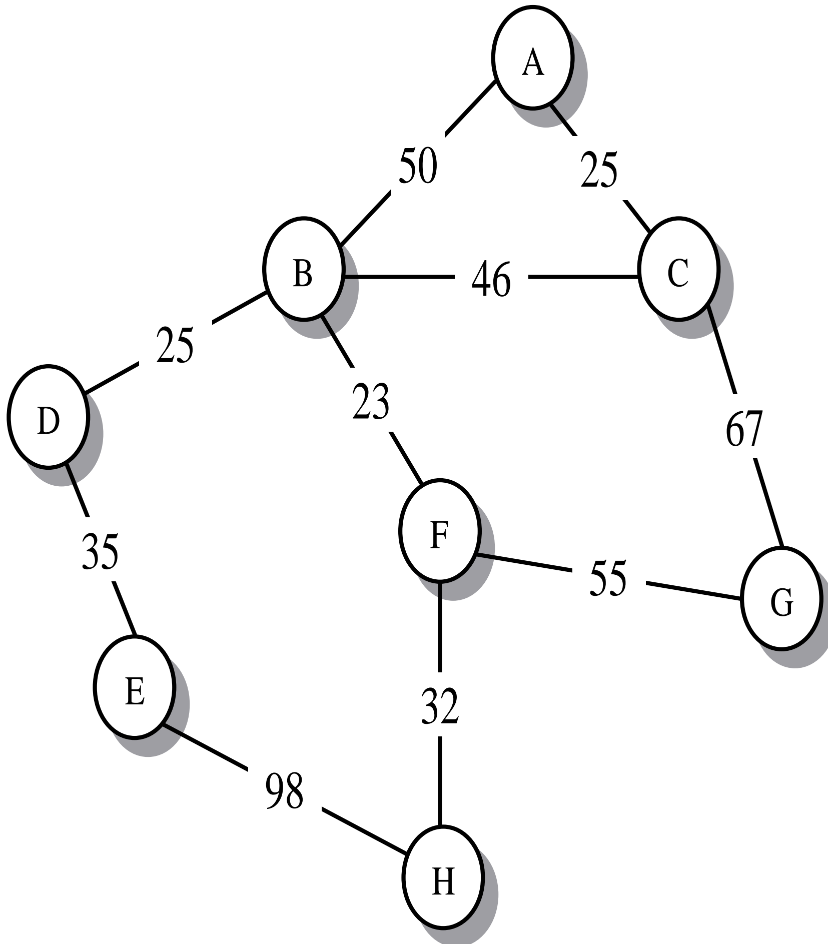
minInfo(C,10) | minInfo(C,11) | minInfo(D,12)

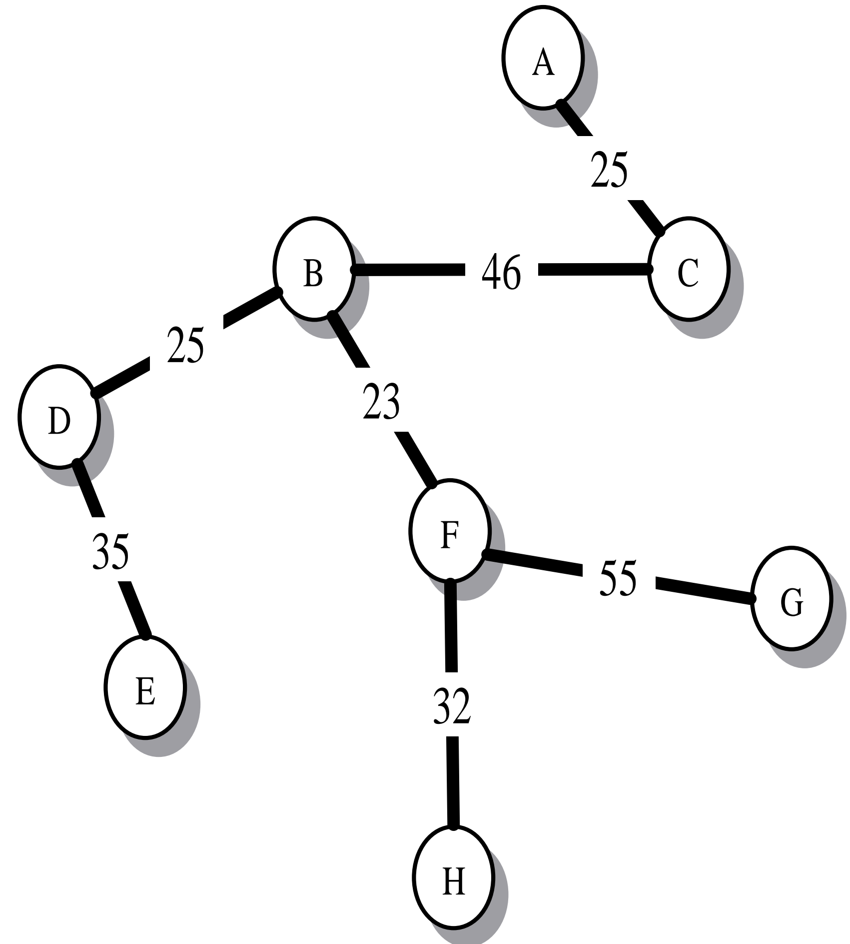
priority queue

minInfo(D,12)

priority queue

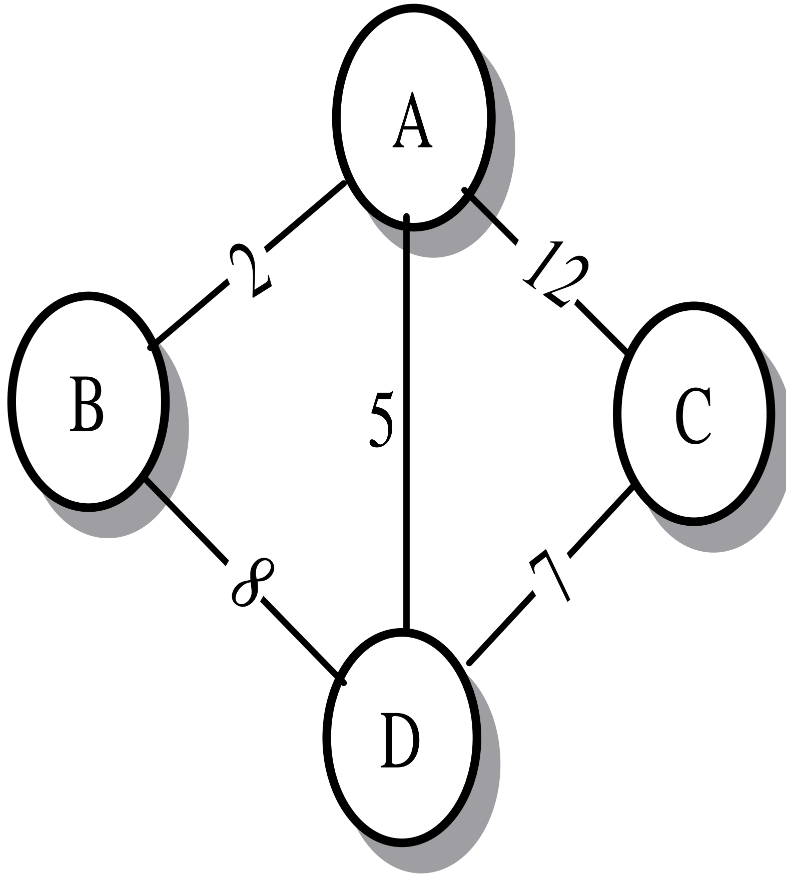
Minimum Spanning Tree Example



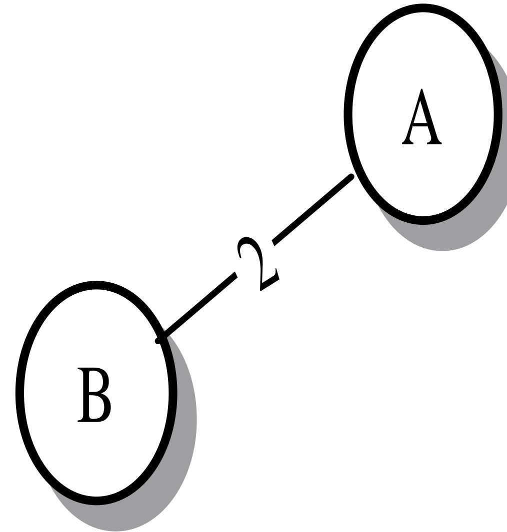


Minimum amount of cable = 241

Minimum Spanning Tree: Vertices A and B



(a)

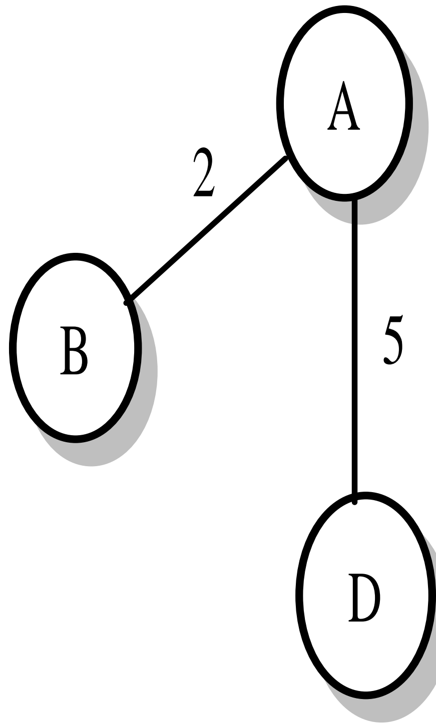


Spanning tree with vertices A, B

minSpanTreeSize = 2, minTreeWeight = 2

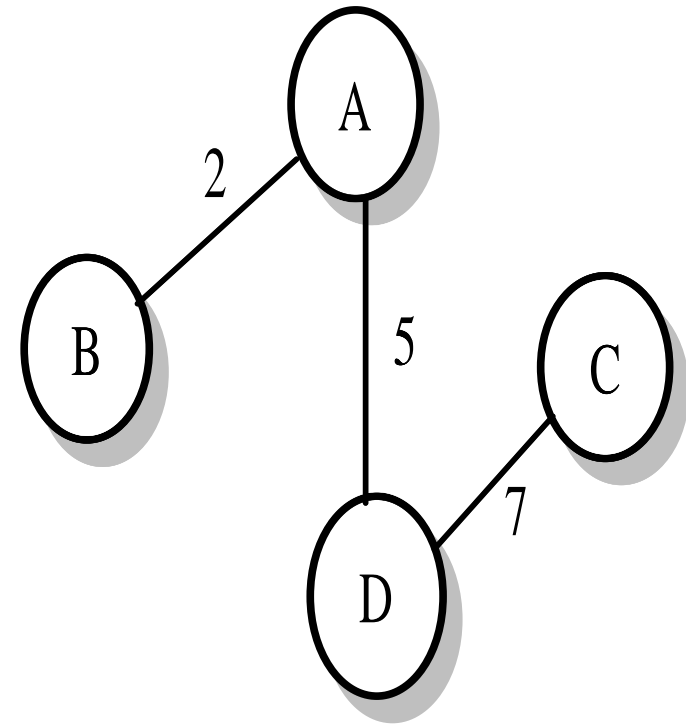
(b)

Completing the Minimum Spanning-Tree Algorithm with Vertices C and D



Spanning tree with vertices A, B, D
minSpanTreeSize = 3, minTreeWeight = 7

(a)



Spanning tree with vertices A, B, D, C
minSpanTreeSize = 4, minTreeWeight = 14

(b)

Summary Slide 1

§- Undirected and Directed Graph (digraph)

- Both types of graphs can be either weighted or nonweighted.

Summary Slide 2

§- Breadth-First, bfs()

- locates all vertices reachable from a starting vertex
- can be used to find the minimum distance from a starting vertex to an ending vertex in a graph.

Summary Slide 3

§- Depth-First search, dfs()

- produces a list of all graph vertices in the reverse order of their finishing times.
- supported by a recursive depth-first visit function, dfsVisit()
- an algorithm can check to see whether a graph is acyclic (has no cycles) and can perform a topological sort of a directed acyclic graph (DAG)
- forms the basis for an efficient algorithm that finds the strong components of a graph

Summary Slide 4

§- Dijkstra's algorithm

- if weights, uses a priority queue to determine a path from a starting to an ending vertex, of minimum weight
- This idea can be extended to Prim's algorithm, which computes the minimum spanning tree in an undirected, connected graph.