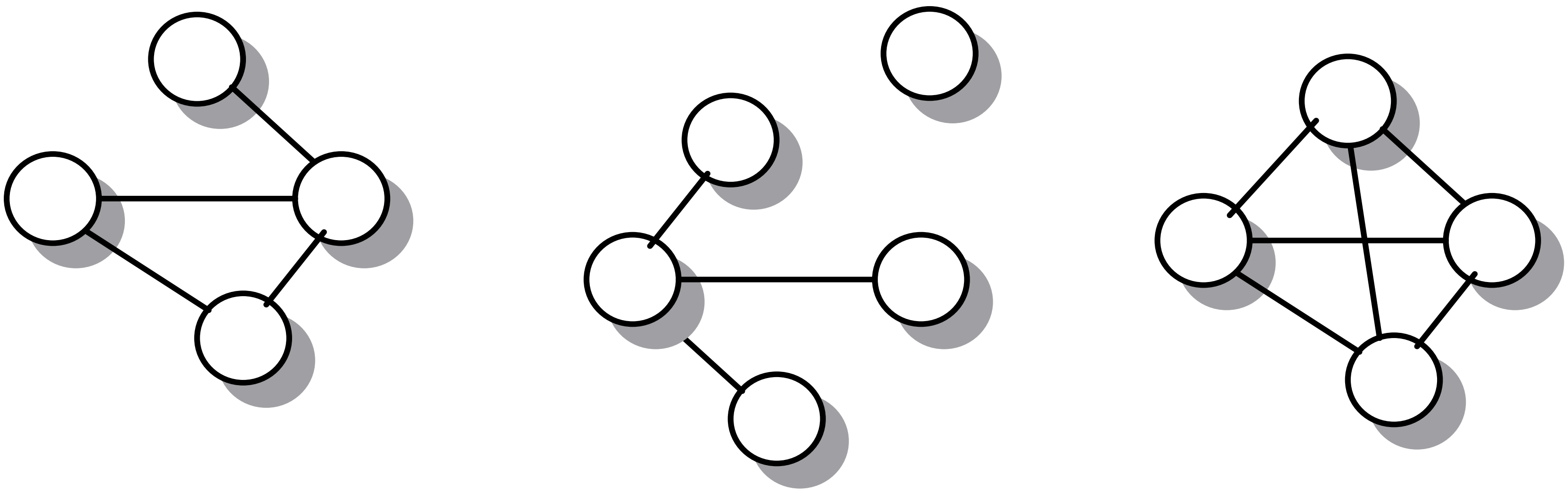
Graph Algorithms

Graph Categories

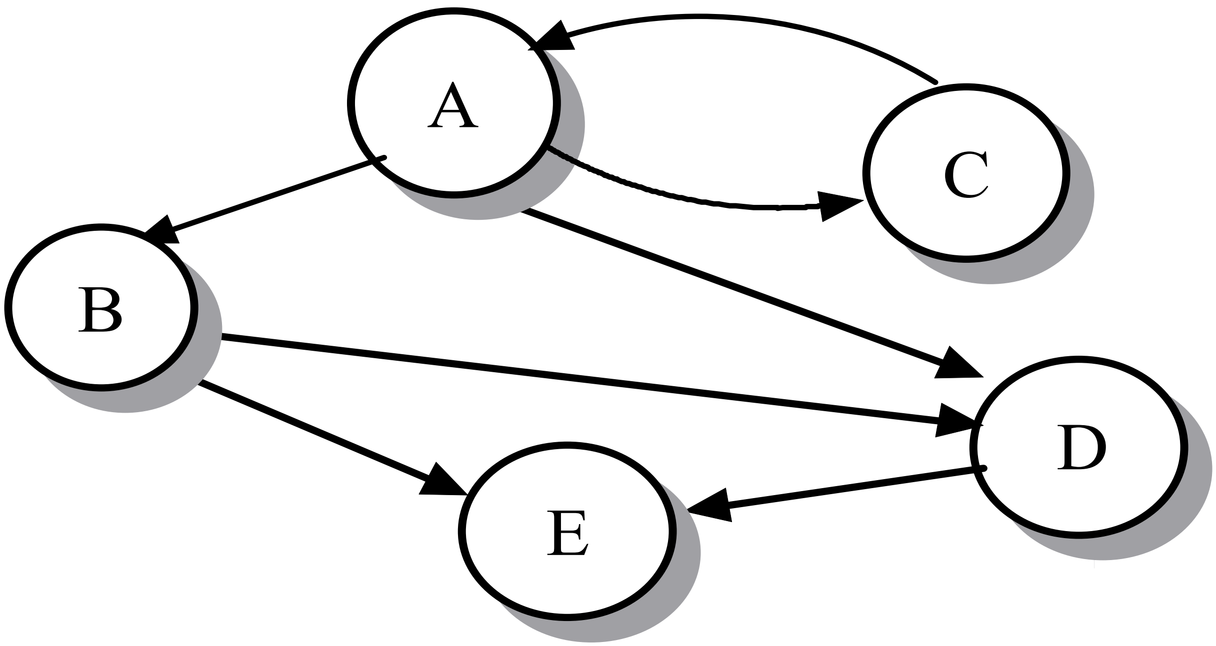
- A graph is *connected* if each pair of vertices have a path between them
- A *complete graph* is a connected graph in which each pair of vertices are linked by an edge



(a: Connected) (b: Disconnected) (c: Complete)

Example of Digraph

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

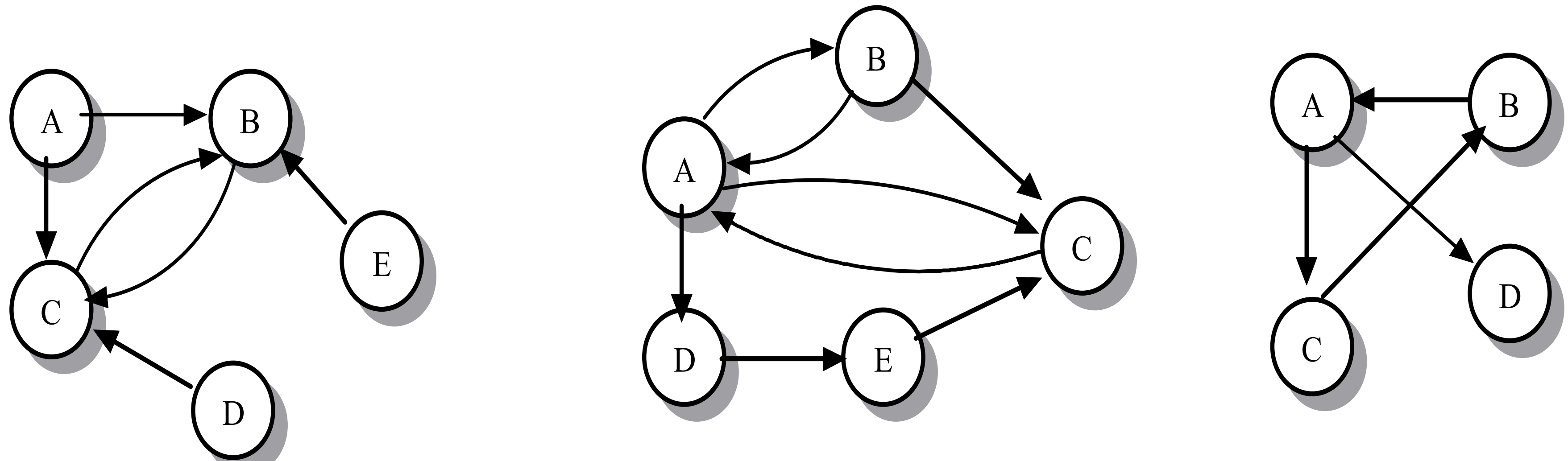


Vertices V = {A, B, C, D, E}

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(vj,vi).



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

(a)

Strongly Connected

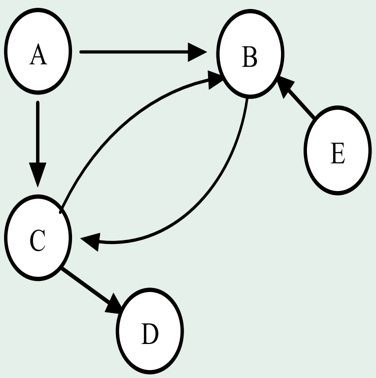
(b)

Weakly Connected
(No path from D to a vertex)

(c)

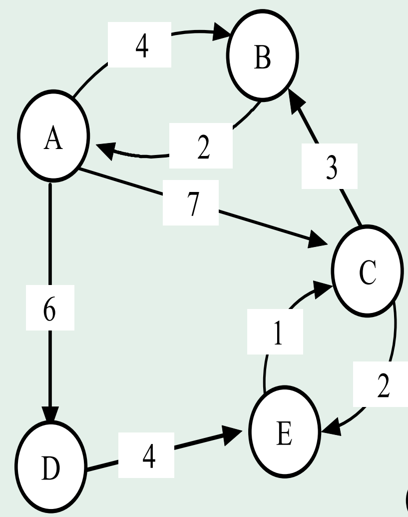
Adjacency Matrix

- 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)

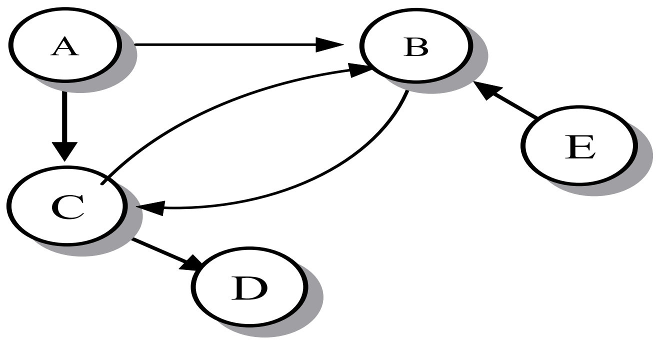


	A	B	C	D	E
A	-1	4	7	6	-1
B	2	-1	-1	-1	-1
C	-1	3	-1	-1	2
D	-1	-1	-1	-1	4
E	-1	-1	1	-1	-1

(b)

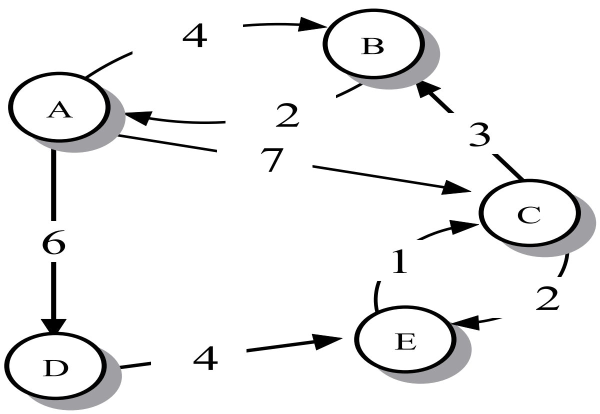
Adjacency Set

(a)



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

(b)

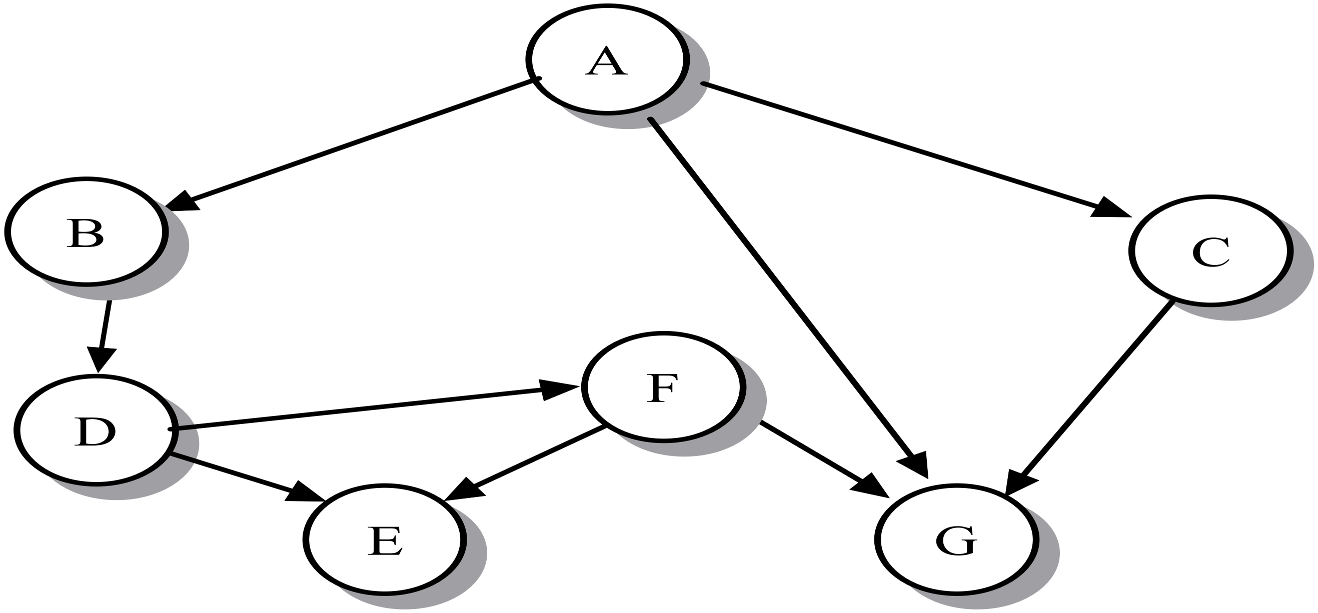


Vertices	Set of Neighbors		
A	B 4	C 7	D 6
B	A 2		
C	B 3	E 2	
D	E 4		
E	C 1		

Breadth-First Search Algorithm

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

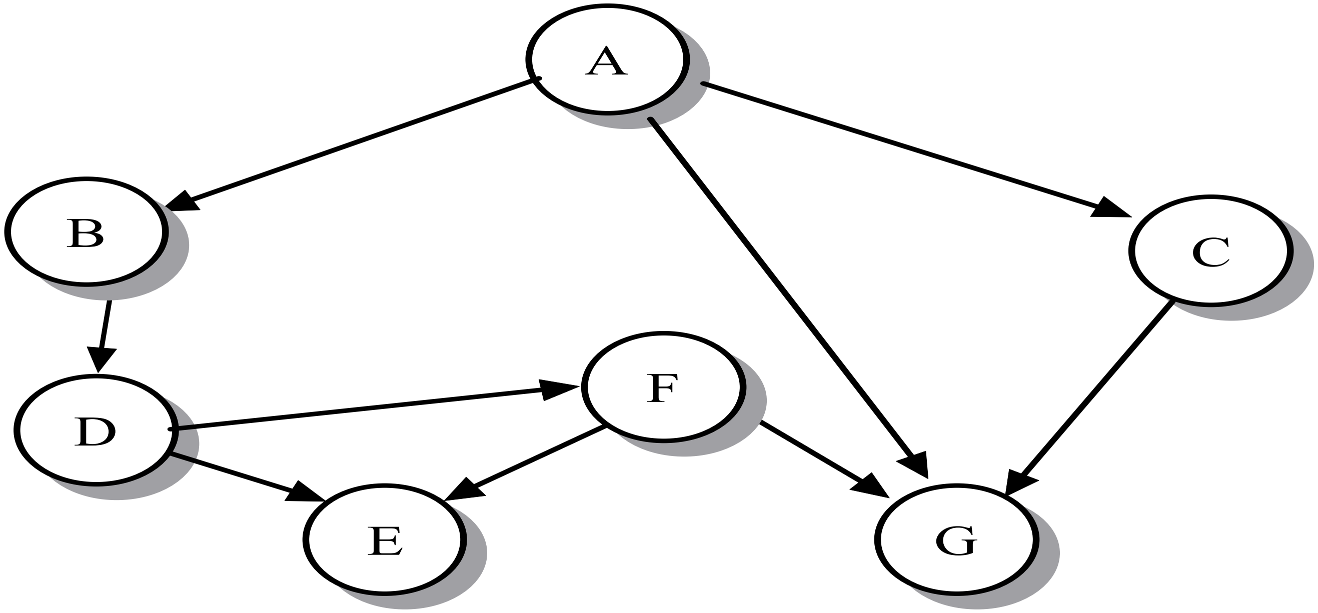
Breadth-First Search Algorithm



Depth-First Search Algorithm

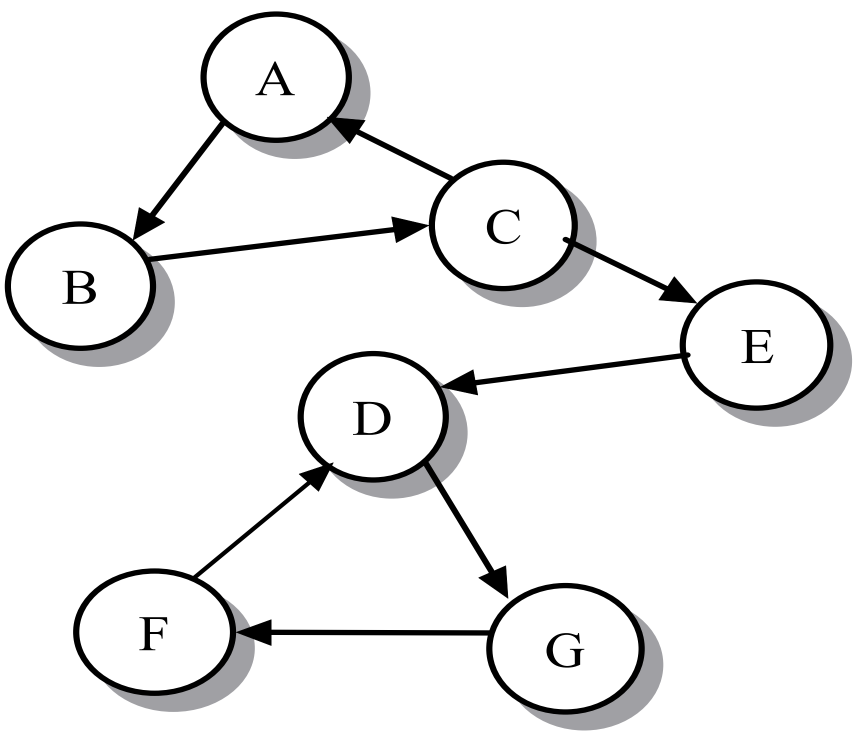
```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)

DFS-VISIT(G, u)
time = time + 1              // white vertex u has just been discovered
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]        // explore edge (u, v)
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK              // blacken u; it is finished
time = time + 1
u.f = time
```

Depth-First Search Algorithm



Strong Components

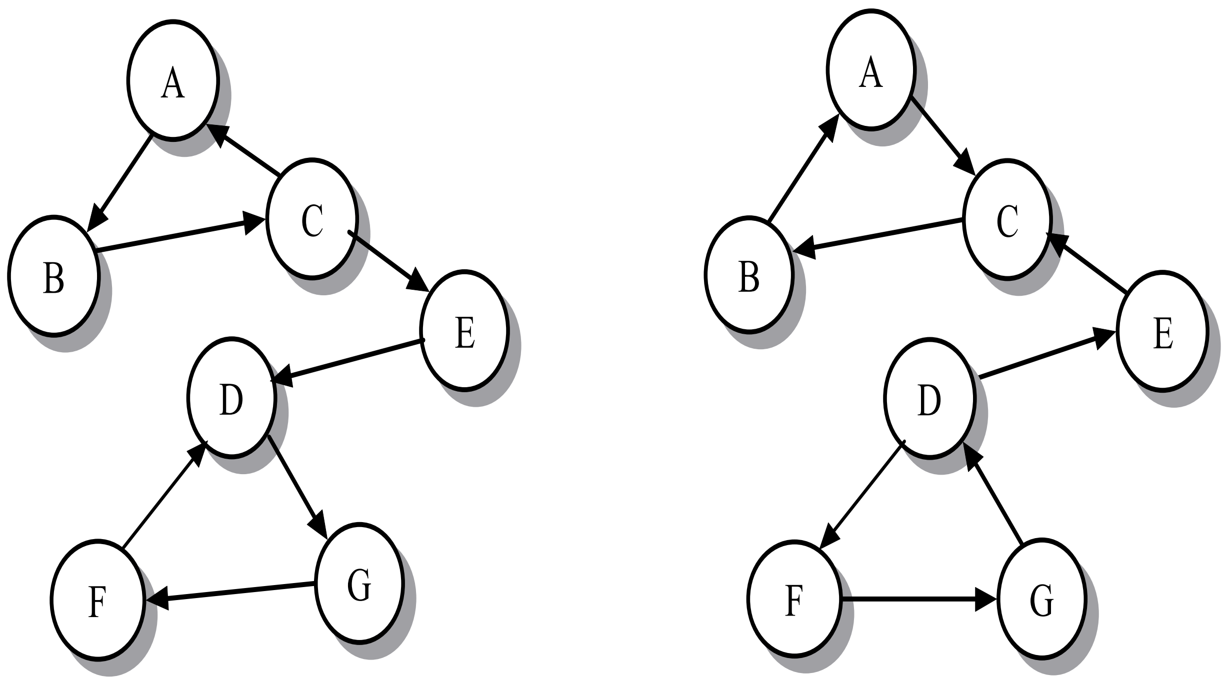
- A *strongly connected component* of a graph G is a maximal set of vertices SC in G that are mutually accessible.



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

Graph G and Its Transpose G^T

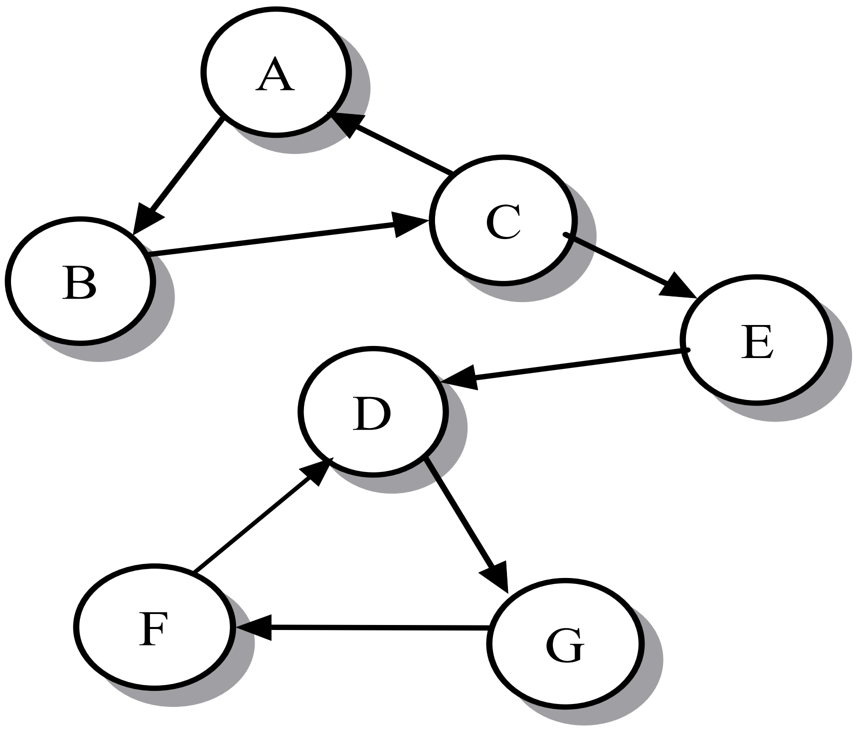
- The transpose has the same set of vertices V as graph G but a new edge set E^T consisting of the edges of G but with the opposite direction.



Graph G

Graph G^T

Strong Components

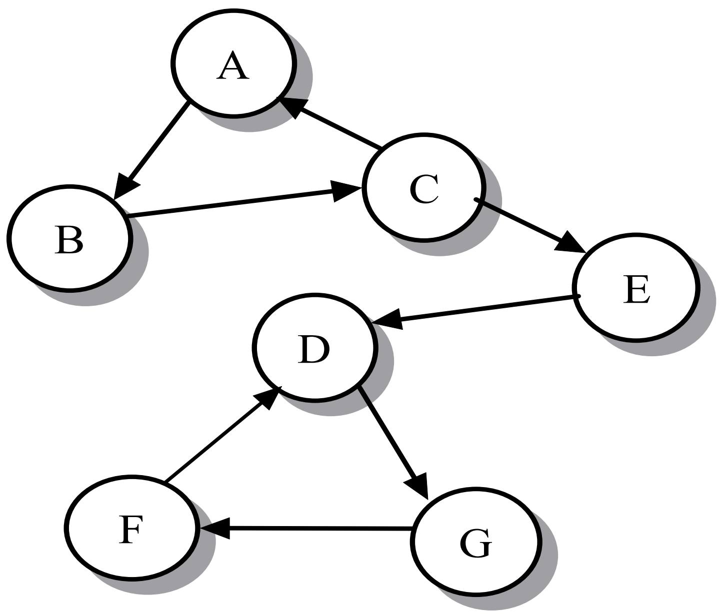


Components
A, B, C
D, F, G
E

Strong Components

STRONGLY-CONNECTED-COMPONENTS(G)

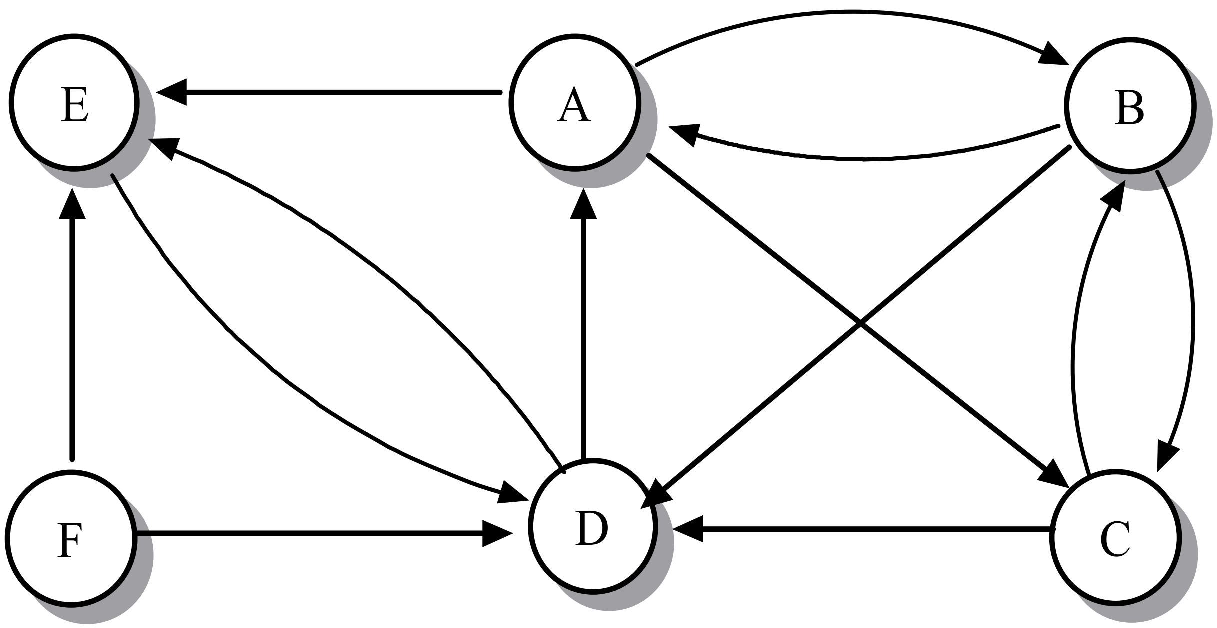
```
1  call DFS(G) to compute finishing times u.f for each vertex u
2  compute G^T
3  call DFS(G^T), but in the main loop of DFS, consider the vertices
         in order of decreasing u.f (as computed in line 1)
4  output the vertices of each tree in the depth-first forest formed in line 3 as a
         separate strongly connected component
```



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

Shortest-Path Example

- Use breadth-first search

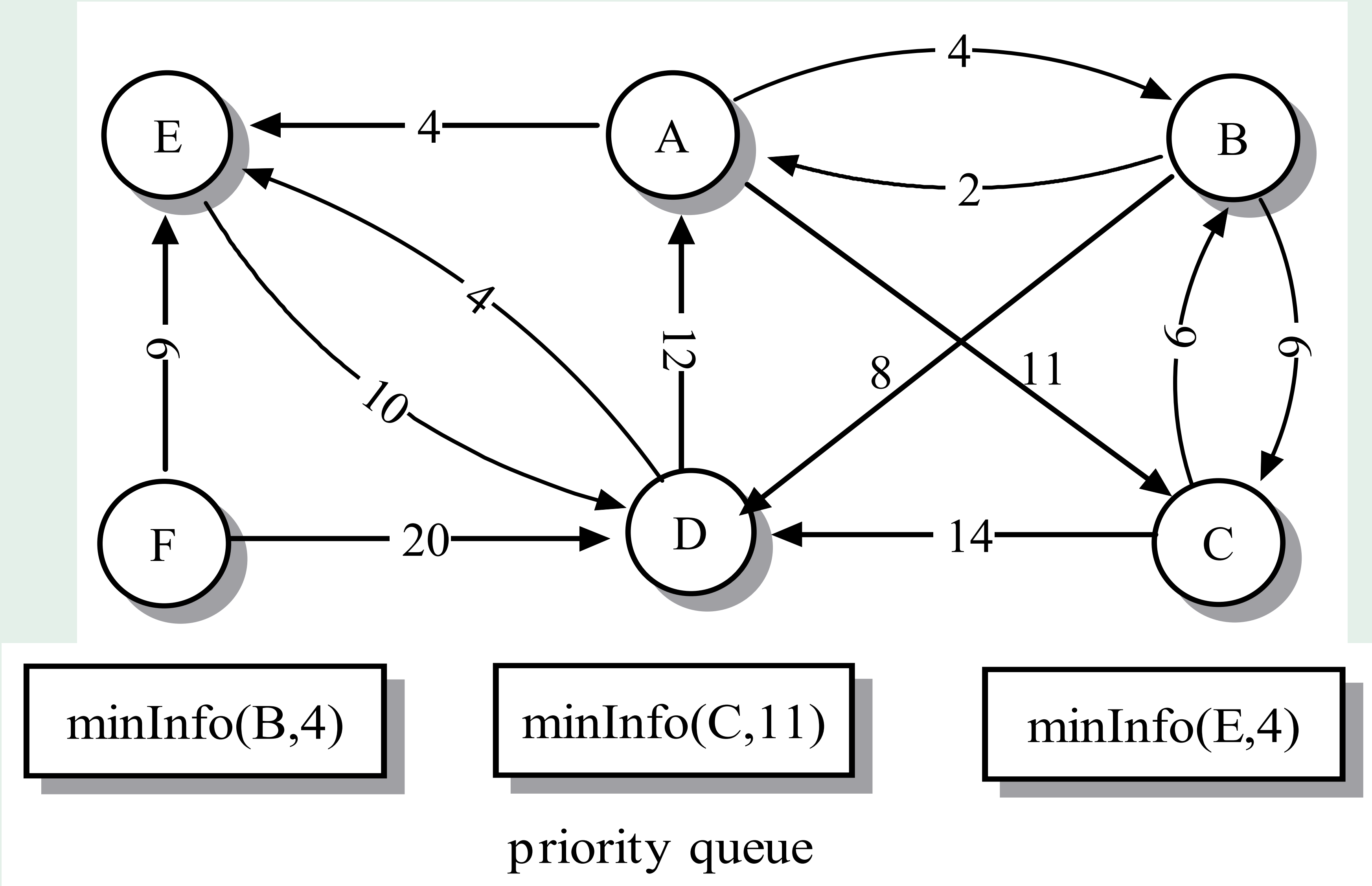


Dijkstra Minimum-Path Algorithm

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

Dijkstra Minimum-Path Algorithm From A to D Example



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)

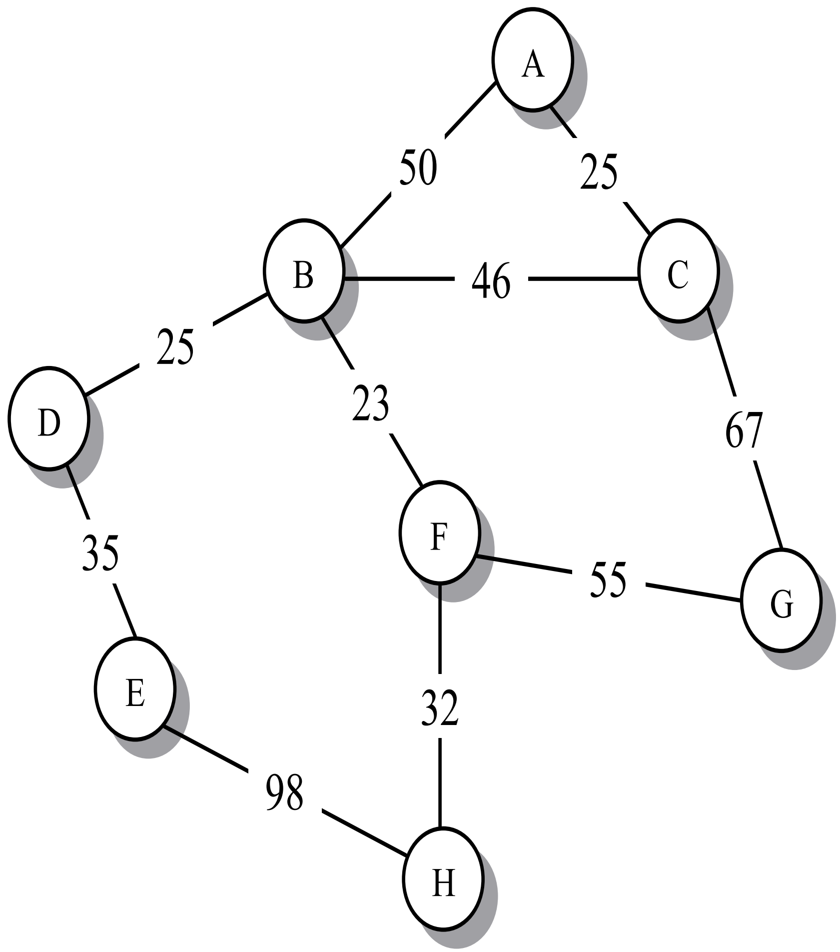
priority queue

minInfo(D,12)

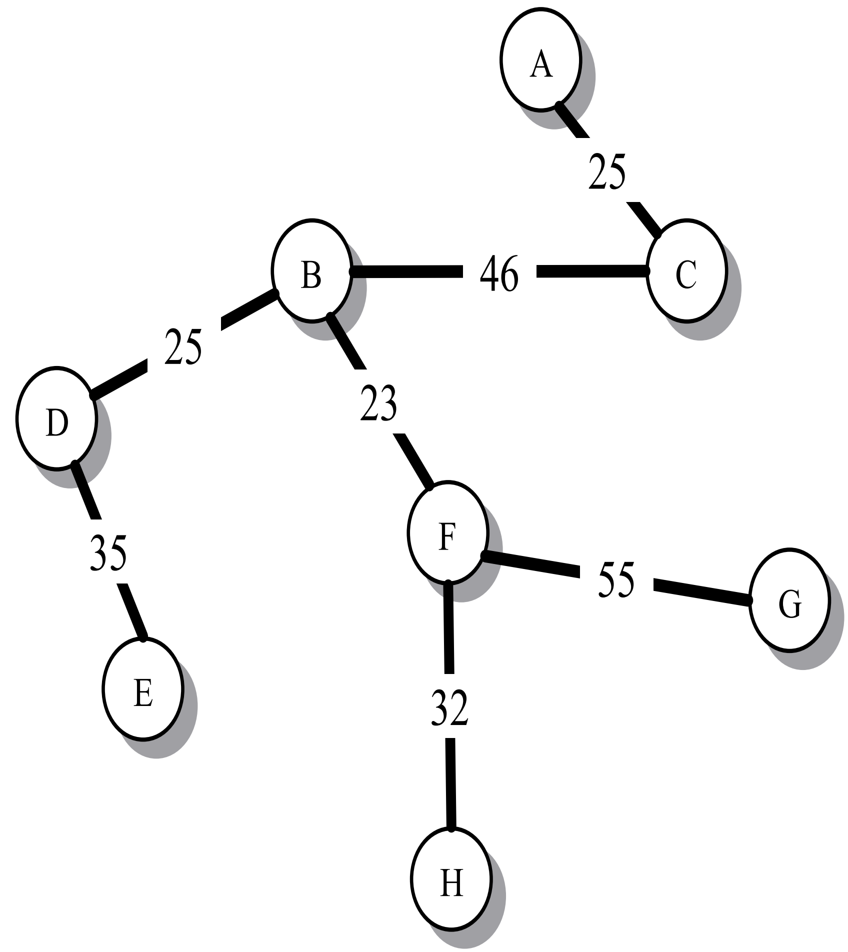
priority queue

Minimum Spanning Tree Example

Network of Hubs

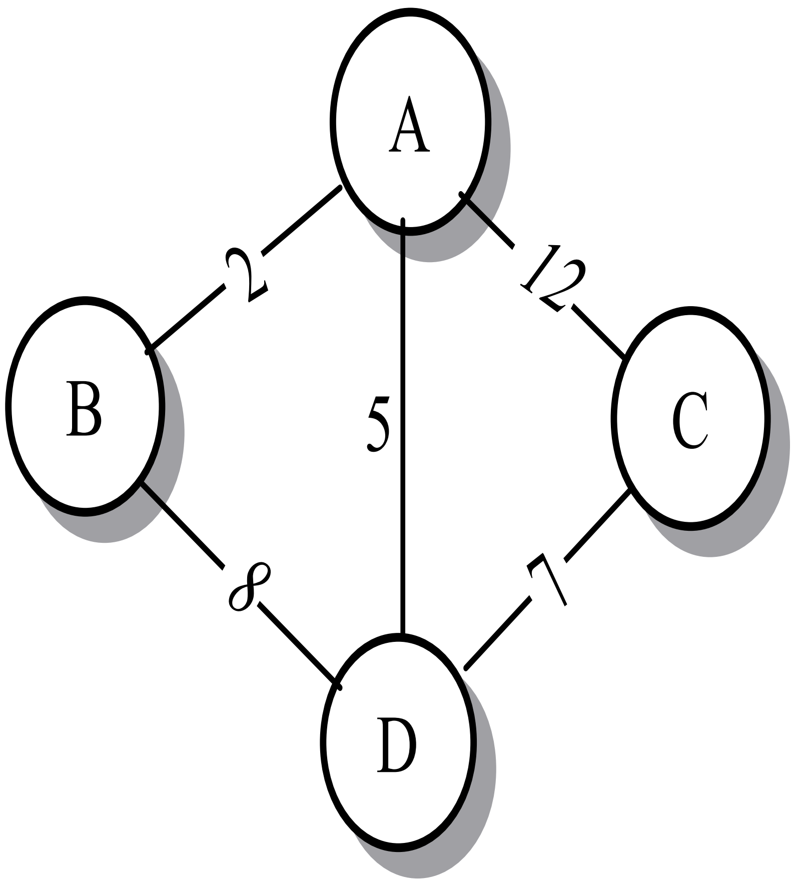


Minimum spanning tree

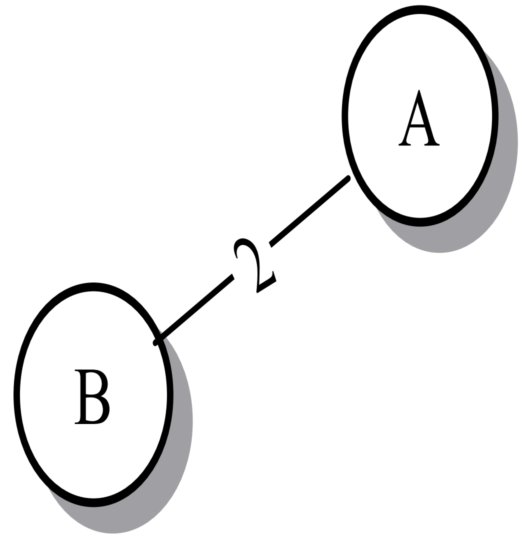


Minimum amount of cable = 241

Minimum Spanning Tree: Vertices A and B



(a)

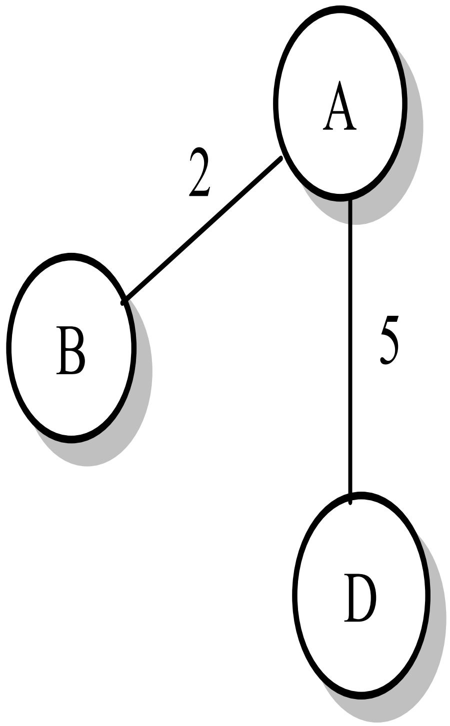


Spanning tree with vertices A, B

minSpanTreeSize = 2, minTreeWeight = 2

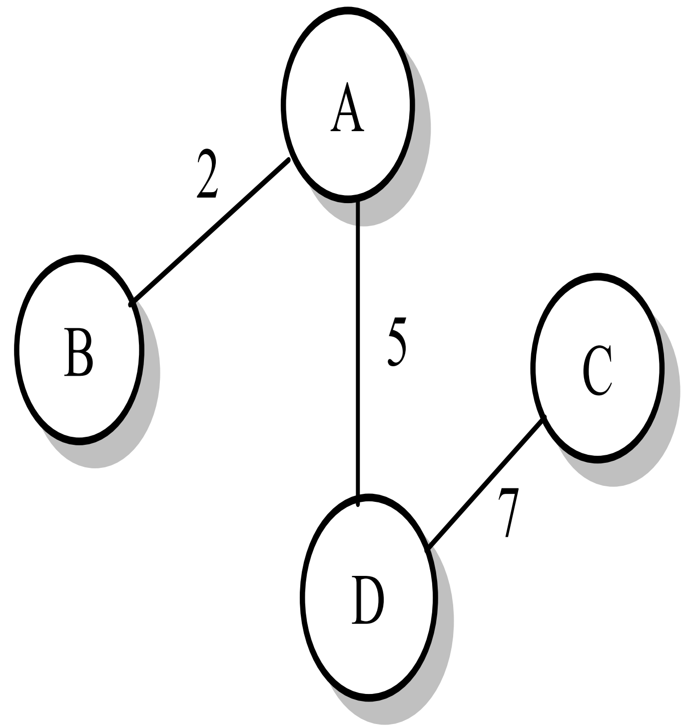
(b)

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



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

(a)



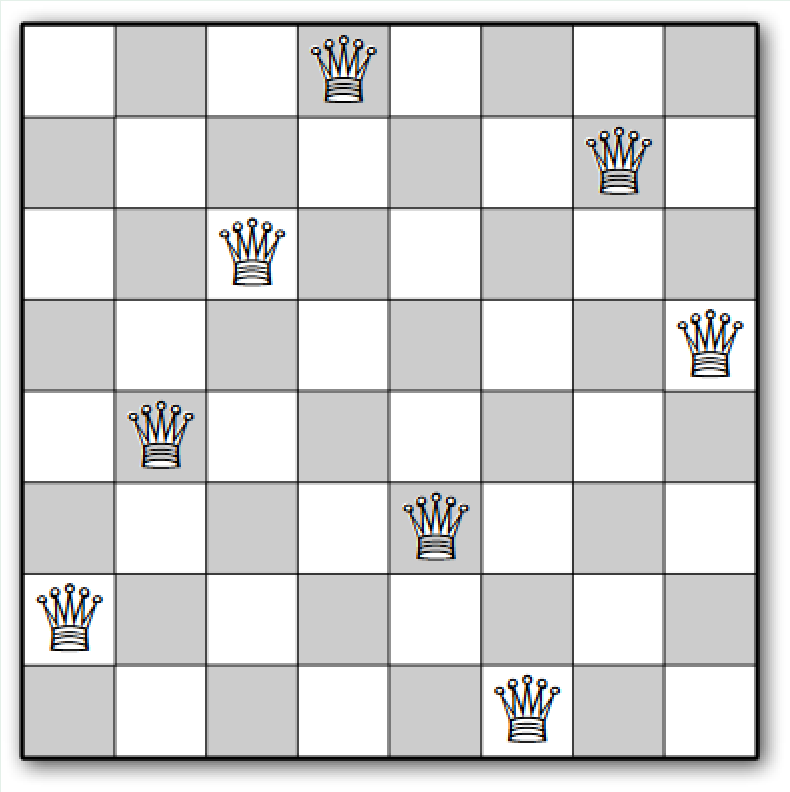
Spanning tree with vertices A, B, D, C
minSpanTreeSize = 4, minTreeWeight = 14

(b)

Backtracking

```
solve(S):
    if (final(S)):
        return true
    A = actions(S)
    foreach (a in A):
        S = move(S,a)
        if (solve(S)):
            return true
        S = undo(S,a)
    return false
```

N-queens Problem



Magic-Square Problem

8	1	6
3	5	7
4	9	2

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1