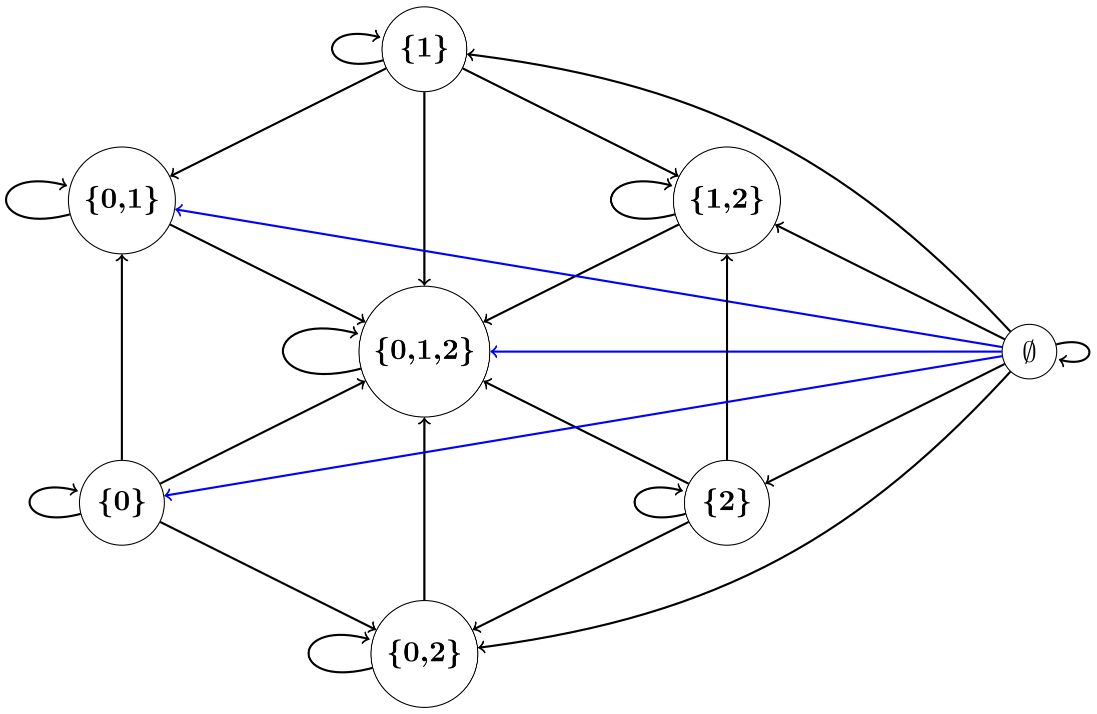
Student Information

Full Name : Batuhan Karaca
Id Number : 2310191

Answer 1

a)



b)

In order that (S, R) is poset, R needs to be a partial ordering on S. Then, R needs to be

1. reflexive
2. antisymmetric
3. transitive

1)reflexivity

For all s in S, s is a set, since the set S is the power set of another set. Every set is a subset of itself. Then s is a subset of itself. Then $(s, s) \in R$. We proved for all $s \in S$, $(s, s) \in R$. Then R is reflexive.

2)antisymmetry

For any sets s_1 and s_2, if $s_1 \subseteq s_2$ and $s_2 \subseteq s_1$, then $s_1 = s_2$. Then for any sets s_1 and s_2 in S, if $(s_1, s_2) \in R$, $s_1 \subseteq s_2$ and if $(s_2, s_1) \in R$, $s_2 \subseteq s_1$. Then $s_1 = s_2$, hence R is antisymmetric.

3)transitivity

For any sets s_1,s_2 and s_3, if $s_1 \subseteq s_2$ and $s_2 \subseteq s_3$, then $s_1 \subseteq s_3$. Then for any sets s_1,s_2 and s_3 in S, if $(s_1, s_2) \in R$, $s_1 \subseteq s_2$ and if $(s_2, s_3) \in R$, $s_2 \subseteq s_3$. Then $s_1 \subseteq s_3$, hence $(s_1, s_3) \in R$. Hence R is transitive.

R is a partial ordering on S, hence (S, R) is a poset.

c)

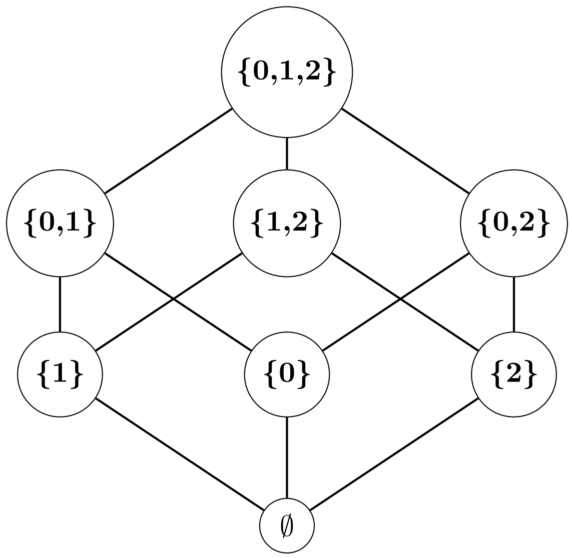
From the textbook(Eighth Edition) page 651

The elements a and b of a poset (S, R) are called comparable if either aRb or bRa.

If (S, R) is a poset and every two elements of S are comparable, (S, R) is a total order.

Assume $a = \{1, 2\} \in S$ and $b = \{0, 1\} \in S$. Then neither $a \subseteq b$ nor $b \subseteq a$. Then a and b are incomparable. Since there is at least two elements that are not comparable within, in S; by contradiction, (S, R) is not a total order.

d)



e)

From the textbook(Eighth Edition) page 657

If u is an element of S such that aRu for all elements $a \in A$, then u is called an upper bound of A. If l is an element of S such that lRa for all elements $a \in A$, then l is called a lower bound of A. (1)

The element x is called the least upper bound of the subset A if x is an upper bound that is less than every other upper bound of A. Similarly, the element y is called the greatest lower bound of A if y is a lower bound of A and zRy whenever z is a lower bound of A. (2)

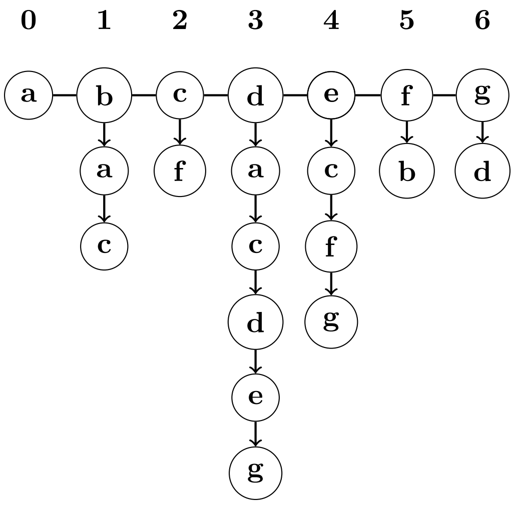
From the textbook(Eighth Edition) page 658

A partially ordered set in which every pair of elements has both a least upper bound and a greatest lower bound is called a lattice. (3)

(S, R) is a poset from part **(b)**. Assume subsets s_1 and s_2, s_{max}, s_{min} of S given, such that $s_1 \subseteq s_{max}$ and $s_2 \subseteq s_{max}$ and $s_{min} \subseteq s_1$ and $s_{min} \subseteq s_2$. Then by the definition(s) (1), s_{min} is a lower bound and s_{max} is an upper bound for the set $\{s_1, s_2\}$. If s_{max} is the least of all upper bounds, then $s_{max} = s_1 \cup s_2$, similiarly if s_{min} is the greatest of all lower bounds, then $s_{min} = s_1 \cap s_2$.For all s_1, s_2 in S, we can find such s_{max}, s_{min}. Then every pair $(s_1, s_2) \in R$ has both a least upper bound and a greatest lower bound. Hence (S, R) constitutes a lattice.

Answer 2

a)



b)

Ordering the vertices a, b, c, d, e, f, g

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad (1)$$

c)

vertex	**indegree**	**outdegree**
a	2	0
b	1	2
c	3	1
d	2	5
e	1	3
f	2	1
g	2	1

d)

$d - c - f - b - a$
$d - e - f - b - a$
$d - e - g - d - a$
$d - e - g - d - c$
$e - f - b - c - f$
$e - g - d - c - f$

e)

$b - c - f - b$
$c - f - b - c$
$d - d - g - d$
$d - e - c - d$
$d - e - g - d$
$d - g - d - d$
$e - g - d - e$
$f - b - c - f$
$g - d - e - g$
$g - d - d - g$

f)

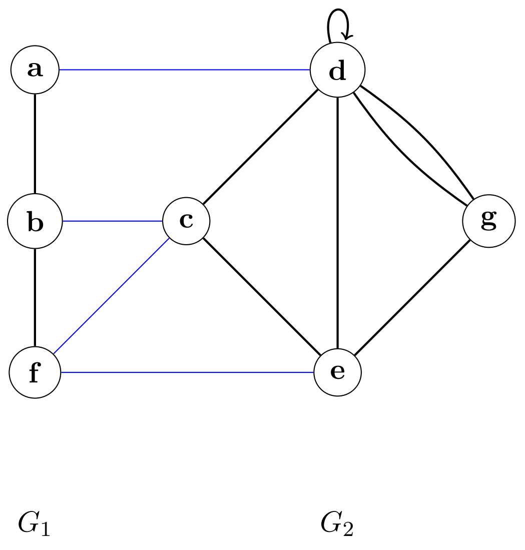
From the textbook(Eighth Edition) page 717

> *An undirected graph is called connected if there is a path between every pair of distinct vertices of the graph.*

Assume an undirected graph $G = (V, E)$ has two subgraphs $G_1 = (V_1, E_1)$, $G_2 = (V_2, G_2)$ that are weakly connected components of G, that is there exists an edge $E_1 \in E$ and $E_1 = \{v_1, v_2\}$ such that $v_1 \in V_1$ and $v_2 \in V_2$. Further assume, G_1 and G_2 are both connected graphs. This means every vertex in G_1 is reachable from every vertex of G_1. This holds also for G_2. Then starting from any vertex of G_1, we can reach v_1 and for G_2, v_2. We know E connects v_1, v_2; then, any vertex of G_1 is reachable from any vertex of G_2 and vice-versa. Hence, G is a connected graph. Therefore, if we prove for any undirected graph G, its two subgraphs are connected components of G, and they are connected subgraphs, we also prove that G is a connected graph. From the textbook(Eighth Edition) page 721

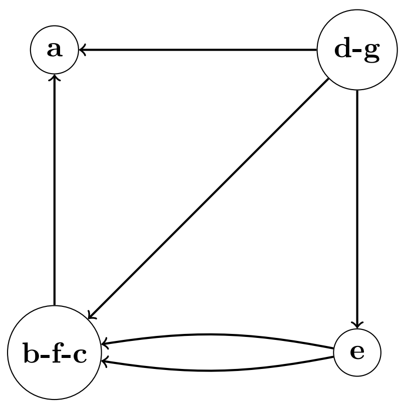
> *A directed graph is weakly connected if there is a path between every two vertices in the underlying undirected graph.*

If we prove that there is a path between every two vertices in the underlying undirected graph of the directed graph G in the question, in other words that the underlying undirected graph is connected, we also prove that the directed graph G is weakly connected. We transform the graph into its corresponding undirected graph H and divide it to two subgraphs $G_1 = (\{a, b, f\}, \{\{a, b\}, \{b, f\}\})$ and $G_2 = (\{c, d, e, g\}, \{\{c, d\}, \{c, e\}, \{d, e\}, \{d, g\}, \{d, d\}, \{e, g\}\})$.



Similarly, we divide G_2 into subgraphs $G_3 = (\{c, d\}, \{\{c, d\}, \{d, d\}\})$ and $G_4 = (\{e, g\}, \{\{e, g\}\})$. We can traverse between c and d through the edge $\{c, d\}$, then G_3 is connected. Similarly, we can traverse between e and g through the edge $\{e, g\}$, then G_4 is connected. G_3 and G_4 are connected via the edges $\{c, e\}, \{d, e\}$ and $\{d, g\}$ with multiplicity of 2; hence, G_2 is connected. Vertices a and b are adjacent, we can traverse from a to b, Vertices b and f are adjacent, we can traverse from b to f, then we can traverse from a to f also; hence G_1 is connected. G_1 and G_2 are connected via the edges $\{a, d\}, \{b, c\}, \{c, f\}, \{e, f\}$; hence, H is connected. We conclude that the underlying undirected graph H of the directed graph G is connected, hence the directed graph G is weakly connected.

g)



h)

Answer 3

a)

From the textbook(Eighth Edition) page 729

An Euler path in G is a simple path containing every edge of G.

The simple path $d - c - b - d - e - f - g - h - e - d - a - b$ is an Euler path, since it contains every edge of G. Then G has an Euler Path.

b)

From the textbook(Eighth Edition) page 729

A connected multigraph with at least two vertices has an Euler circuit if and only if each of its vertices has even degree.

G has 8 vertices, more than two. Since the vertex b has 3 edges incident with it, then its degree is 3, which is odd. Then G does not have any Euler circuits

c)

From the textbook(Eighth Edition) page 734

A simple path in a graph G that passes through every vertex exactly once is called a Hamilton path.

The simple path $a - b - c - d - e - f - g - h$ is a Hamiltonian path, since it passes through every vertex of G once. Then G has an Hamiltonian Path.

d)

From the textbook(Eighth Edition) page 734

A simple circuit in a graph G that passes through every vertex exactly once is called a Hamilton circuit.

Let us observe two subgraphs of G, namely $G_1 = \{V_1, E_1\} = \{\{a, b, c, d\}, \{\{a, b\}, \{b, c\}, \{c, d\}, \{b, d\}, \{a, d\}\}\}$, $G_2 = \{V_2, E_2\} = \{\{e, f, g, h\}, \{\{e, f\}, \{f, g\}, \{g, h\}, \{e, h\}\}\}$. Assume G has a Hamiltonian circuit with an initial vertex v_0.

The circuit should pass through d and e exactly once (*).
The circuit should pass through every vertex of G_1 and G_2 exactly once (**).

If $v_0 \in G_1$, since circuit passes through d and e exactly once, it must terminate at a vertex of G_2, in order to pass through every vertex of G_2; however, $v_0 \notin V_2$, then this path is not a circuit.

Similarly if $v_0 \in G_2$, since circuit passes through d and e exactly once, it must terminate at a vertex of G_1; however, $v_0 \notin V_1$. Then, by using (*) and (**), we disprove that there is such a circuit. Then, there exists no such circuit. Hence by contradiction, G has no Hamilton circuits.

Answer 4

From a work of David Glickenstein, namely "Graph Theory Notes 5: Graphs as matrices, spectral graph theory, and PageRank", at page 1

Definition 1 *A permutation matrix is a matrix gotten from the identity by permuting the columns (i.e., switching some of the columns).*

Proposition 2 *The graphs G and G′ are isomorphic if and only if their adjacency matrices are related by*

$$A = P^T A' P \quad (*)$$

for some permutation matrix P.

In our case, adjacency matrix of G, Ordering the vertices a, b, c, d, e

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{2}$$

adjacency matrix of G', Ordering the vertices a', b', c', d', e'

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{3}$$

We observe that $G = G'$. Then the permutation matrix P can be 5x5 identity matrix, holding the equality (*). Since there exists such permutation matrix $P = I$, G an G' are isomorphic.

Work of David Glickenstein can be accessed via the below link

https://www.math.arizona.edu/(tilde)glickenstein/math443f14/notes5matrices.pdf

NOTE: Please change (tilde) with tilde sign -ascii of the character is 126- in order to access. I could not have the full url link with tilde sign, since I was not allowed to use extra packages such as the url package.

Answer 5

a)

A pseudocode for Djikstra's Algorithm is below and at each step, variables are shown with a table below that contains the values of the vertices and previous vertices that are connected to them in the shortest path from the initial vertex. The tables below show the values after the operations done at each step. (*) sign before the vertices' labels, mark adjacent unvisited vertices, since at each step only possible elements that are subject to change are adjacent unvisited vertices.

Pseudocode for Djikstra's Algorithm

At initial step, assign 0 for the initial vertex, ∞ for other vertices.

At each step do the following(s)

1)Take the vertex which has the minimum value in unvisited vertices, as the next vertex -Assuming that the vertices are sorted, then in case of equality, first one is chosen.
2)if the next vertex is None (the set of unvisited vertices is empty), halt
3)else if the current vertex is None, assign next vertex to current vertex, pop current vertex from unvisited vertices pushing it to visited vertices and continue (proceed to the next step)
4)else proceed for each vertex in unvisited adjacent vertices, assign next vertex to current vertex , pop current vertex from unvisited vertices pushing it to visited vertices and continue

Initial step (0) Current vertex: None

Next vertex: a
Visited vertices: {}
Unvisited vertices: {a,b,c,d,e,f,g,h,i,j,k}
Adjacent vertices that are unvisited: {}

vertex	distance	previous
a	0	
b	∞	
c	∞	
d	∞	
e	∞	
f	∞	
g	∞	
h	∞	
i	∞	
j	∞	
k	∞	

Step (1) Current vertex: a
Next vertex: b
Visited vertices: {a}
Unvisited vertices: {b,c,d,e,f,g,h,i,j,k}
Adjacent vertices that are unvisited: {b,e,h}

vertex	distance	previous
a	0	
*b	3	a
c	∞	
d	∞	
*e	5	a
f	∞	
g	∞	
*h	8	a
i	∞	
j	∞	
k	∞	

Step (2) Current vertex: b
Next vertex: c
Visited vertices: {a,b}
Unvisited vertices: {c,d,e,f,g,h,i,j,k}
Adjacent vertices that are unvisited: {c,e,f}

vertex	distance	previous
a	0	
b	3	a
*c	5	b
d	∞	
*e	5	a
*f	10	b
g	∞	
h	8	a
i	∞	
j	∞	
k	∞	

Step (3) Current vertex: c
Next vertex: e
Visited vertices: {a,b,c}
Unvisited vertices: {d,e,f,g,h,i,j,k}
Adjacent vertices that are unvisited: {d,f,g}

vertex	distance	previous
a	0	
b	3	a
c	5	b
*d	8	c
e	5	a
*f	7	c
*g	11	c
h	8	a
i	∞	
j	∞	
k	∞	

Step (4) Current vertex: e
Next vertex: f
Visited vertices: {a,b,c,e}
Unvisited vertices: {d,f,g,h,i,j,k}
Adjacent vertices that are unvisited: {f,h}

vertex	distance	previous
a	0	
b	3	a
c	5	b
d	8	c
e	5	a
*f	7	c
g	11	c
*h	8	a
i	∞	
j	∞	
k	∞	

Step (5) Current vertex: f
Next vertex: d
Visited vertices: {a,b,c,e,f}
Unvisited vertices: {d,g,h,i,j,k}
Adjacent vertices that are unvisited: {g,h,i,j}

vertex	distance	previous
a	0	
b	3	a
c	5	b
d	8	c
e	5	a
f	7	c
*g	11	c
*h	8	a
*i	11	f
*j	10	f
k	∞	

Step (6) Current vertex: d
Next vertex: h
Visited vertices: {a,b,c,e,f,d}
Unvisited vertices: {g,h,i,j,k}
Adjacent vertices that are unvisited: {g,k}

vertex	distance	previous
a	0	
b	3	a
c	5	b
d	8	c
e	5	a
f	7	c
*g	11	c
h	8	a
i	11	f
j	10	f
*k	10	d

Step (7) Current vertex: h
Next vertex: i
Visited vertices: {a,b,c,e,f,d,h}
Unvisited vertices: {g,i,j,k}
Adjacent vertices that are unvisited: {i}

vertex	distance	previous
a	0	
b	3	a
c	5	b
d	8	c
e	5	a
f	7	c
g	11	c
h	8	a
*i	10	h
j	10	f
k	10	d

Step (8) Current vertex: i
Next vertex: j
Visited vertices: {a,b,c,e,f,d,h,i}
Unvisited vertices: {g,j,k}
Adjacent vertices that are unvisited: {j}

vertex	distance	previous
a	0	
b	3	a
c	5	b
d	8	c
e	5	a
f	7	c
g	11	c
h	8	a
i	10	h
*j	10	f
k	10	d

Step (9) Current vertex: j
Next vertex: k
Visited vertices: {a,b,c,e,f,d,h,i,j}
Unvisited vertices: {g,k}
Adjacent vertices that are unvisited: {g,k}

vertex	distance	previous
a	0	
b	3	a
c	5	b
d	8	c
e	5	a
f	7	c
*g	11	c
h	8	a
i	10	h
j	10	f
*k	10	d

Step (10) Current vertex: k
Next vertex: g
Visited vertices: {a,b,c,e,f,d,h,i,j,k}
Unvisited vertices: {g}
Adjacent vertices that are unvisited: {g}

vertex	distance	previous
a	0	
b	3	a
c	5	b
d	8	c
e	5	a
f	7	c
*g	11	c
h	8	a
i	10	h
j	10	f
k	10	d

Step (11) Current vertex: g
Next vertex: None
Visited vertices: {a,b,c,e,f,d,h,i,j,k,g}
Unvisited vertices: {}
Adjacent vertices that are unvisited: {}

vertex	distance	previous
a	0	
b	3	a
c	5	b
d	8	c
e	5	a
f	7	c
*g	11	c
h	8	a
i	10	h
j	10	f
k	10	d

After halting, we observe the last state of the variables in the above table. Starting from the terminal vertex, if the values are to be backtracked by checking their previous vertices on the table (i.e., j to f, f to c and so forth), then we see the path

$a - b - c - f - j$

is to shortest path, from a to j.

b)

The procedure for Prim's algorithm is as below

At initial step, remove a random initial vertex from unvisited vertices -preferably the first element in the list, and push it to visited vertices.

At each step do the following(s)

1)Halt if unvisited vertices is empty
2)Inspect every edge that connects a visited vertex to an unvisited vertex

3)Pick the edge with the least weight and store it in the visited edges -Assuming that the vertices are sorted, then in case of equality, first one is chosen
4)Pop the unvisited vertex that the chosen edge is incident to and push it to visited vertices.
5)Proceed to the next step

Initial step (0) Visited vertices: {}
Visited edges: {}
Unvisited vertices: {a,b,c,d,e,f,g,h,i,j,k}
Adjacent vertices that are unvisited: {}

Step (1) Visited vertices: {a}
Visited edges: {}
Unvisited vertices: {b,c,d,e,f,g,h,i,j,k}
Adjacent vertices that are unvisited: {{a,b},{a,e},{a,h}}
{a,b} has the minimum weight.

Step (2) Visited vertices: {a,b}
Visited edges: {{a,b}}
Unvisited vertices: {c,d,e,f,g,h,i,j,k}
Adjacent vertices that are unvisited: {{a,e},{a,h},{b,c},{b,e},{b,f}}
{b,c} has the minimum weight.

Step (3) Visited vertices: {a,b,c}
Visited edges: {{a,b},{b,c}}
Unvisited vertices: {d,e,f,g,h,i,j,k}
Adjacent vertices that are unvisited: {{a,e},{a,h},{b,e},{b,f},{c,d},{c,f},{c,g}}
{c,f} has the minimum weight.

Step (4) Visited vertices: {a,b,c,f}
Visited edges: {{a,b},{b,c},{c,f}}
Unvisited vertices: {d,e,g,h,i,j,k}
Adjacent vertices that are unvisited: {{a,e},{a,h},{b,e},{c,d},{c,g}}
{c,d} has the minimum weight.

Step (5) Visited vertices: {a,b,c,d,f}
Visited edges: {{a,b},{b,c},{c,f},{c,d}}
Unvisited vertices: {e,g,h,i,j,k}
Adjacent vertices that are unvisited: {{a,e},{a,h},{b,e},{c,g} {d,g},{d,k},{f,e},{f,g},{f,h},{f,i},{f,j}}
{d,k} has the minimum weight.

Step (6) Visited vertices: {a,b,c,d,f,k}
Visited edges: {{a,b},{b,c},{c,f},{c,d},{d,k}}
Unvisited vertices: {e,g,h,i,j}
Adjacent vertices that are unvisited: {{a,e},{a,h},{b,e},{c,g} {d,g},{f,e},{f,g},{f,h},{f,i},{f,j},{k,g},{k,j}}
{f,j} has the minimum weight.

Step (7) Visited vertices: {a,b,c,d,f,j,k}
Visited edges: {{a,b},{b,c},{c,f},{c,d},{d,k},{f,j}}
Unvisited vertices: {e,g,h,i}

Adjacent vertices that are unvisited: {{a,e},{a,h},{b,e},{c,g} {d,g},{f,e},{f,g},{f,h},{f,i},{k,g},{j,g},{j,i}}
{f,e} has the minimum weight.

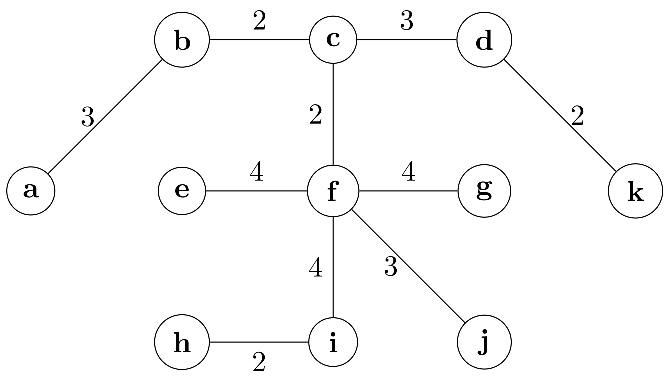
Step (8) Visited vertices: {a,b,c,d,e,f,j,k}
Visited edges: {{a,b},{b,c},{c,f},{c,d},{d,k},{f,j},{f,e}}
Unvisited vertices: {g,h,i}
Adjacent vertices that are unvisited: {{a,h},{d,g},{e,h},{f,g},{f,h},{f,i},{j,g},{j,i}}
{f,g} has the minimum weight.

Step (9) Visited vertices: {a,b,c,d,e,f,g,j,k}
Visited edges: {{a,b},{b,c},{c,f},{c,d},{d,k},{f,g},{f,j},{f,e}}
Unvisited vertices: {h,i}
Adjacent vertices that are unvisited: {{a,h},{e,h},{f,h},{f,i},{j,i}}
{f,i} has the minimum weight.

Step (10) Visited vertices: {a,b,c,d,e,f,g,i,j,k}
Visited edges: {{a,b},{b,c},{c,f},{c,d},{d,k},{f,g},{f,j},{f,e},{f,i}}
Unvisited vertices: {h}
Adjacent vertices that are unvisited: {{a,h},{e,h},{f,h}},{i,h}}
{i,h} has the minimum weight.

Step (11) Visited vertices: {a,b,c,d,e,f,g,h,i,j,k}
Visited edges: {{a,b},{b,c},{c,f},{c,d},{d,k},{f,g},{f,j},{f,e},{f,i},{f,h}}
Unvisited vertices: {}
Adjacent vertices that are unvisited: {}

After halting, connecting the visited edges by their common vertices, we have a minimum spanning tree for the initial vertex A.



Answer 6

a)

There are 7 vertices, and 6 edges in T. f and g have largest levels, 3; then height of T is 3.

b)

a-b-c-d-e-f-g

c)

b-d-f-g-e-c-a

d)

b-a-d-c-f-e-g

e)

From the textbook(Eighth Edition) page 783-784

> *In a rooted tree, Vertices that have children are called internal vertices.* (1)
> *The tree is called a full m-ary tree if every internal vertex has exactly m children. An m-ary* (2)
> *tree with* $m = 2$ *is called a binary tree.* (3)

In question, a is given as a root, then the tree is a rooted tree. Using (1), we see that a, c, e are internal vertices. Vertices a, c and e have two children ($m = 2$), then the given tree is a full binary tree.

f)

From the textbook(Eighth Edition) page 792, question 27

> *A complete m-ary tree is a full m-ary tree in which every leaf is at the same level.* (1)

From the textbook(Eighth Edition) page 783

> *A vertex of a rooted tree is called a leaf if it has no children.* (2)

Leaves of the given tree are b, d, f, g. b, d and f are at different levels; hence, T is not a complete tree, then not a complete binary tree.

g)

From the textbook(Eighth Edition) page 794, binary search trees have the following property

> *Vertices are assigned keys so that the key of a vertex is both larger than the keys of all vertices in its left subtree and smaller than the keys of all vertices in its right subtree.*

NOTE Below, the term *ordered* or *sorted* is used for a sequence, when for every two values a and b in the sequence, if a is on the left of b, then $a \leq b$.

Lemma 1 An in-order traversal of a binary search tree with vertices that have unique keys always prints the vertices of the tree in sorted order

Proof by induction
BASE STEP Printing a single vertex gives a sorted list with a length 1
INDUCTIVE STEP Assume An in-order traversal of a binary search tree with $1, 2, 3...k$ vertices that have unique keys always prints the vertices of the tree in sorted order. Assume an additional vertex a with a unique key is inserted. If we print the subtree T_a that a is the root of, in inorder, we print a in the middle, subtree with smaller values than a on the left, subtree with larger values on the right. We

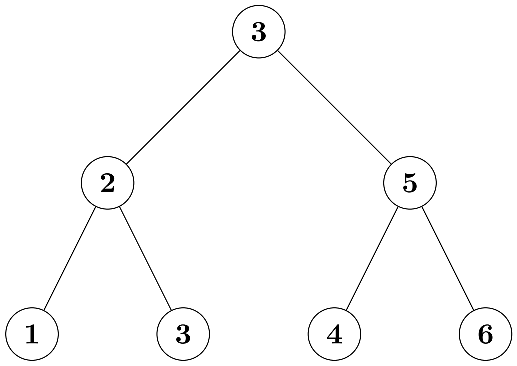
know that inorder print of both subtrees of a are ordered, since their number of vertices n is less than or equal to k - assuming an empty list is also ordered (trivial case $n = 0$). Then inorder print of T_a is ordered. Similarly we print the parent of a in the middle, T_a to the left if each value of T_a is smaller than of parent of a, right if larger, in inorder traversal. Then inorder print of T_p, the tree that parent of a is root of, is also ordered. We know that inorder print of remaining vertices of the tree was already ordered, before the insertion. Then inorder print of the new tree is also ordered, completing the proof.

Let us observe the sequence b,a,d,c,f,e,g which is obtained by inorder traversal in part **(d)**. Substituting their values,the sequence becomes $13, 17, 19, 24, 23, 43, 58$ -every key is unique. The sequence is not ordered, since 24 is on the left of 23, but $24 \leq 23$ does not hold. Then, by *Lemma 1*, T is not a binary search tree.

h)

the minimum number of nodes for a full binary tree with height 5 is $2 * height + 1 = 2 * 5 + 1 = 11$

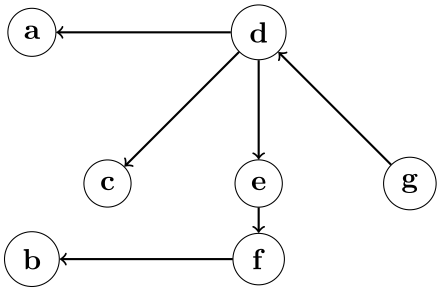
i)



j)

The last value is the value found, and values before are the values compared with the key. To find key 1, $3, 2, 1$ To find key 6, $3, 5, 6$

k)



g is the root

l)

From the textbook(Eighth Edition) page 789

The height of a rooted tree is the maximum of the levels of vertices. In other words, the height of a rooted tree is the length of the longest path from the root to any vertex.

The longest possible length from the root of a rooted tree with k vertices is $k-1$.

Proof by induction
BASE STEP A single vertex is a tree, has $1-1=0$ height, which is the maximum possible height.
INDUCTIVE STEP Assume the longest possible length from the root of a rooted tree with t vertices is $t-1$. If, an additional vertex is inserted as a child of the most distant vertex (from the root), the height $h=t$, else $h=t-1$. Then, the longest possible length from the root of a rooted tree with $t+1$ vertices is t, completing the proof.