Student Information

Full Name : mergen

Answer 1

a)

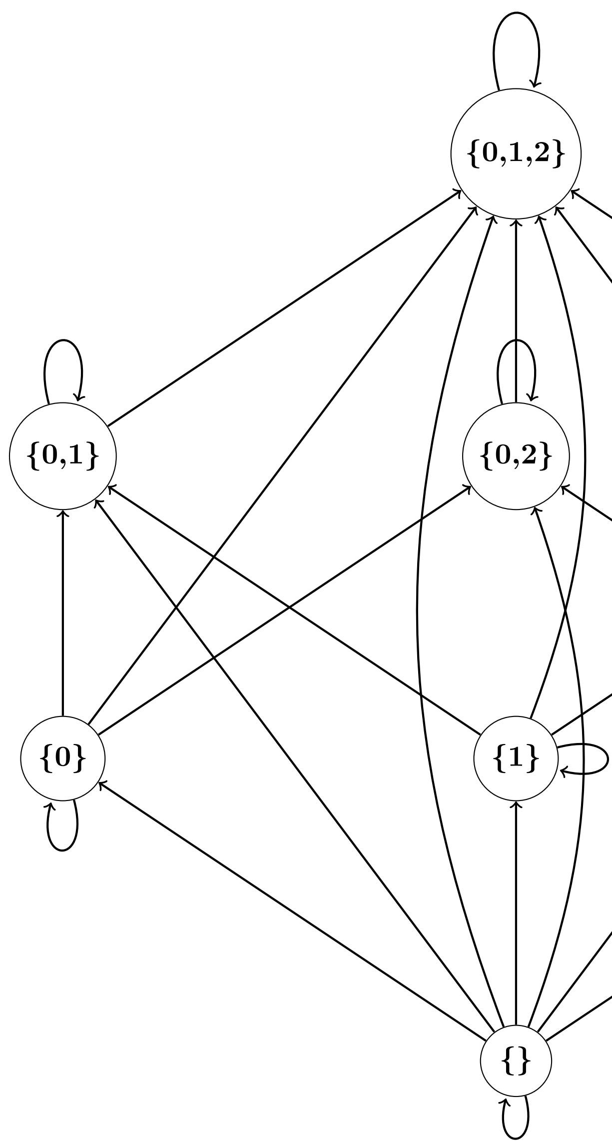

Figure 1: R in Q1, as a directed graph.

b)

Proving that **(S,R)** is a poset requires 3 properties of partial ordering sets to be acquired: reflexivity, anti-symmetry and transitivity.
From now on we use the definition of **R** and **S**:
(1)

$$S = \{w|w \in P(\{0, 1, 2\})\}$$

(2)

$$R = \{(w_1, w_2) | w_1 \in S, w_2 \in S, w_1 \subseteq w_2\}$$

- reflexivity: As we know each set is a subset of itself(every element on the directed graph has a loop bending to itself), we can simply say the graph is reflexive.
- Since there are no two edges between two vertices with different directions, the graph is anti-symmetric.
- From the graph we can see that does not matter which path taken, it is possible to go {0,1,2} from {} that is; $\forall \mathrm{w}_1\ \forall \mathrm{w}_2\ \forall \mathrm{w}_3 \in \mathrm{S}((\mathrm{w}_1\mathrm{Rw}_2 \land \mathrm{w}_2\mathrm{Rw}_3) \rightarrow \mathrm{w}_1\mathrm{Rw}_3)$. Therefore, from the definition of transitivity the graph is transitive.
- Since we have shown that the Graph 1 is reflexive, anti-symmetric, and transitive, the partial ordering (S,R) is a poset.

c)

- **Definition 3, pg.651**

  If $(S, \propto)$ is a poset and every two elements of S are comparable, S is called a totally ordered set, and $\propto$ is called a total order.

- (S, R) is a poset, but since not each member of S is comparable, S is not a totally ordered set. That is, for example {1,2} $\subseteq$ {1,2,3} but {1,2,3} $\nsubseteq$ {1,2}.

d)

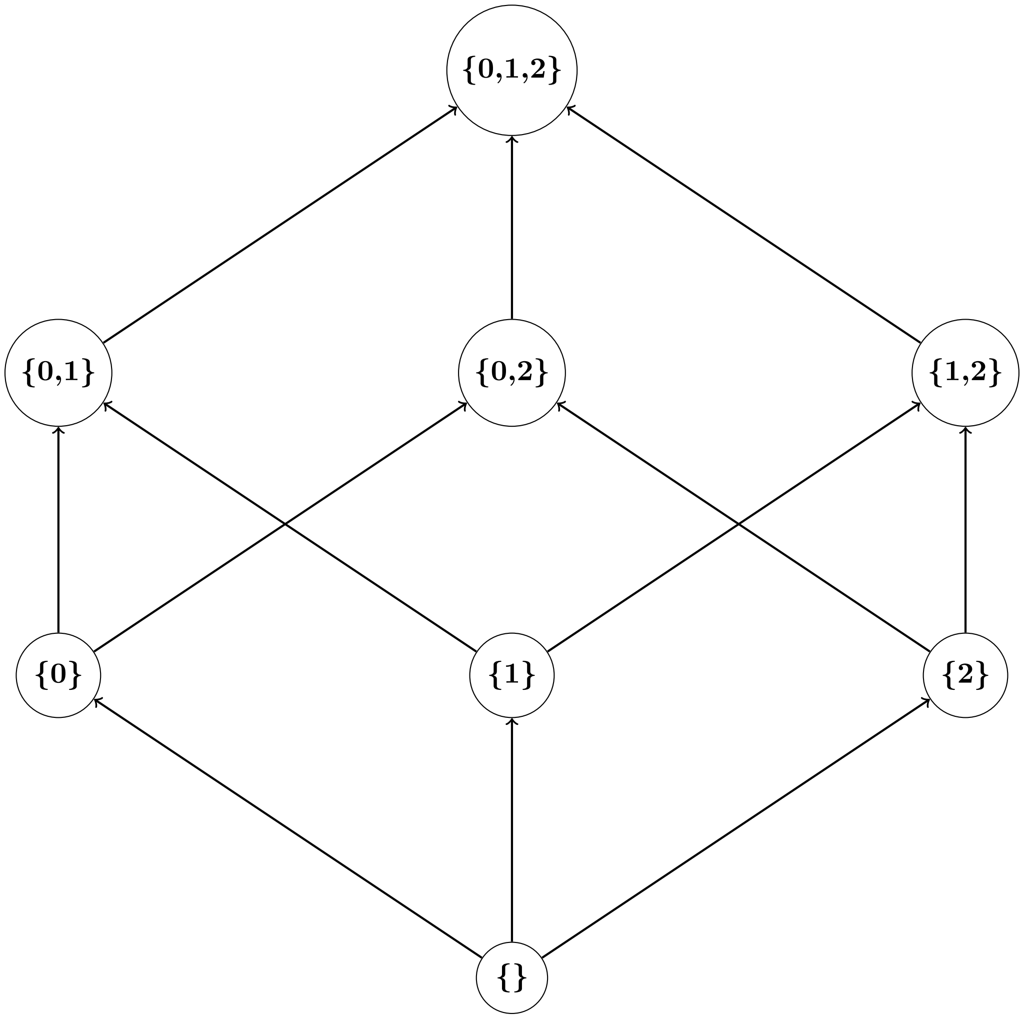

Figure 2: The Hasse-Diagram of (P({0,1,2}),$\subseteq$)

Maximal element is {0,1,2}
Minimal element is {}

e)

- The Lattice definition from the Section 9.6 of the textbook says that

  > 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.

- As each two elements w_1 and w_2 in S, the least upper bound is as follows $w_1 \cup w_2$ and the greatest lower bound is as follows $w_1 \cap w_2$.

- Therefore, (P({0,1,2}),$\subseteq$) **constitutes a lattice**.

Answer 2

a)

Table 1: Adjacency list for directed graph G

Initial Vertex	Terminal Vertex
a	
b	a,c
c	f
d	a,c,d,e,g
e	c,f,g
f	b
f	d

b)

$$A_G = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

c)

$\deg^-(\mathrm{a}) = 2$, $\deg^+(\mathrm{a}) = 0$
$\deg^-(\mathrm{b}) = 1$, $\deg^+(\mathrm{b}) = 2$
$\deg^-(\mathrm{c}) = 3$, $\deg^+(\mathrm{c}) = 1$
$\deg^-(\mathrm{d}) = 2$, $\deg^+(\mathrm{d}) = 5$
$\deg^-(\mathrm{e}) = 1$, $\deg^+(\mathrm{e}) = 3$
$\deg^-(\mathrm{f}) = 2$, $\deg^+(\mathrm{f}) = 1$
$\deg^-(\mathrm{g}) = 2$, $\deg^+(\mathrm{g}) = 1$

	In-degrees Count	**Out-degrees Count**
a	2	0
b	1	2
c	3	1
d	2	5
e	1	3
f	2	1
g	2	1

Table 2: In-degrees and Out-degrees of every vertex of Graph G in Q2

d)

- $d \to e \to f \to b \to a$
- $d \to c \to f \to b \to a$

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

e)

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

f)

- Since there is not a path from a to d, we can say that the graph is **not** strongly connected.
- If we draw the **undirected graph** of the Graph G in Q2, we get the following;

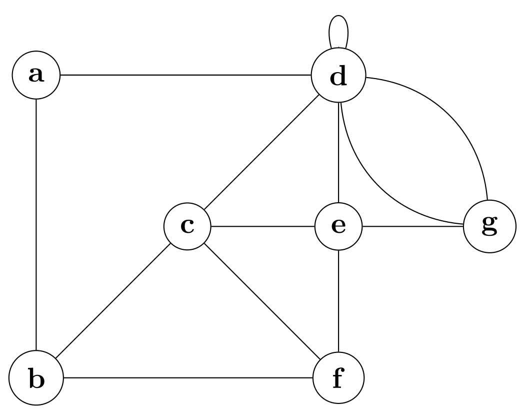

Figure 3: The Undirected Graph of Graph G in Q2.

- Since all vertices are connected to at least one other vertex, we can create paths between any given two vertices.
- Since there is a path between every two vertices in the above undirected graph, we say Graph G is **weakly-connected.** (Definition 5, pg.721)

g)

The strongly-connected components of G are as follows;

- the vertex a
- the subgraph consisting of the vertices b, c, f and edges $(b, c), (c, f), and\ (f, b)$
- the subgraph consisting of the vertices d, e, g and edges $(d, e), (e, g), and\ (g, d)$

h)

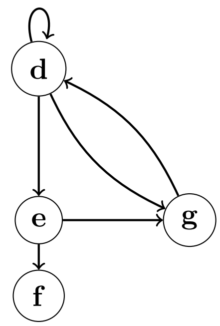

Figure 4: Subgraph H of G induced by the vertices {d, e, f, g} ⊂ V

$$A_H = \begin{array}{c} \\ d \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{cccc} d & e & f & g \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

From the Theorem(pg.723), number of different paths of length 3 from d to g in the subgraph H will be given by the value of A_H^3's $a_{1,4}$ element.

$$A_H^3 = \begin{array}{c} \\ d \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{cccc} d & e & f & g \end{array} \\ \begin{pmatrix} 4 & 2 & 1 & 3 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 2 \end{pmatrix} \end{array}$$

Since $a_{1,4} = 3$, there exist 3 different paths from d to g of length 3 in the subgraph H.

Answer 3

Vertex	a	b	c	d	e	f	g	h
Degree	2	3	2	5	4	2	2	2

Table 3: Degrees of each Vertex in Graph G in Q3

a

- **Theorem 2** from the Section 10.5 of the textbook says that

  > A connected multigraph has an Euler path but not an Euler circuit if and only if it has exactly two vertices of odd degree.

- Since the *Graph G* is a connected multigraph, to check if the given graph has an Euler Path, we can simply check the degree of each vertex. On a graph with n vertices, if $n-2$ vertices have even degree, and if the remaining 2 vertices have odd degrees, the given graph has an **Euler Path**.
- When we look at Table 3, we can clearly see that not all vertices are of even degree, 2 vetices have odd degrees.

- Therefore, *Graph G* have an **Euler Path**.
- An example **Euler Path**: $b \to a \to d \to b \to c \to d \to e \to f \to g \to h \to e \to d$

b

- **Theorem 1** from the Section 10.5 of the textbook says that

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

- Since the *Graph G* is a connected multigraph with 8 vertices, to check if the given graph has an Euler Path, we can simply check the degree of each vertex. If all vertices have even degree, the given graph has an **Euler Circuit**.
- When we look at Table 3, we can clearly see that not all vertices are of even degree.
- Therefore, *Graph G*does not have an **Euler Circuit**.

c

- **Definition 2** from the Section 10.5 of the textbook says that

  > A simple path in a *Graph G* that passes through every vertex exactly once is called a *Hamilton Path*.

- By picking the following simple route, we can create a *Hamilton Path*;

  $c \to b \to a \to d \to e \to h \to g \to f$

- Since we were able to create a *Hamilton Path*, we can say that *Graph G* has a **Hamilton Path**.

d

- **Theorem 4 (Ore's Theorem)** from the Section 10.5 of the textbook says that

  > If G is a simple graph with n vertices with $n \geq 3$ such that $deg(u) + deg(v) \geq n$ for every pair of nonadjacent vertices u and v in G, then G has a *Hamilton Circuit.*

- Since the *Graph G* is a simple graph with 8 vertices, if we find a pair of nonadjacent vertices with summed degrees less than 8, we can say that the given graph does not have a **Hamilton Circuit**.
- Let us pick the nonadjacent vertices b and g. When we look at Table 3, we can see that their degrees are 3 and 2, respectively.
- We can easily see that $deg(b) + deg(g) \geq 8$ does not hold, and therefore *Graph G* does not have a **Hamilton Circuit**.

Answer 4

Both **G** and **G'** have five vertices and five edges. Both have five vertices of degree two. Because **G** and **G'** agree with respect to these invariants, it is reasonable to try to find an isomorphism *f.* The function *f* with *f(a) = a'*, *f(b) = e'*, *f(c) = d'*, *f(d) = c'*, *f(e) = b'* is a one-to-one correspondence between **G** and **G'**. To see that correspondence preserves adjacency, note that adjacent vertices in **G** are a and b, a and e, b and c, c and d, and d and e, and each of the pairs *f(a) = a'* and *f(b) = e'*, *f(a) = a'* and *f(e) = b'*, *f(b) = e'* and *f(c) = d'*, *f(c) = d'* and *f(d) = c'*, and *f(d) = c'* and *f(e) = b'* consists of two adjacent vertices in *G'*; with such adjacency matrices A_G and $A_{G'}$ respectively. To see whether *f* preserves edges, we examine the adjacency matrix of **G**,

$$A_G = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

and the adjacency matrix of **G'** with the rows and columns labeled by the images of the corresponding vertices in **G**,

$$A_{G'} = \begin{array}{c} \\ a' \\ e' \\ d' \\ c' \\ b' \end{array} \begin{array}{c} \begin{array}{ccccc} a' & e' & d' & c' & b' \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Because $A_G = A_{G'}$, it follows that f preserves edges. We conclude that f is an 'isomorphism', so **G** and **G'** are isomorphic.

Answer 5

a)

Step 0

For this question, I used superscripts in the nodes as shortest distances from the starting node, namely a, for this graph

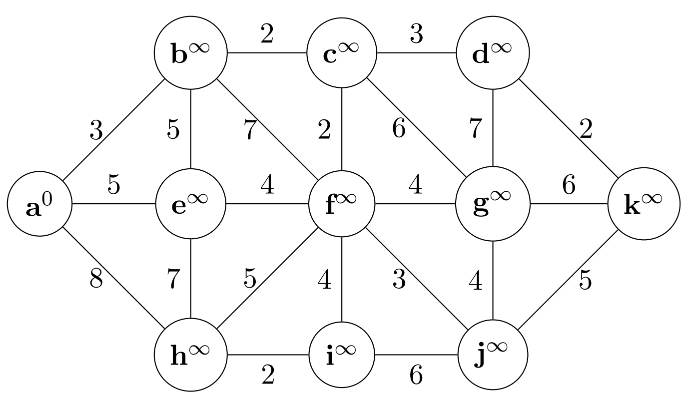

Figure 5: Initial conditions

Step 1

By visiting the *unvisited* neighbouring nodes of a, we get the following graph.

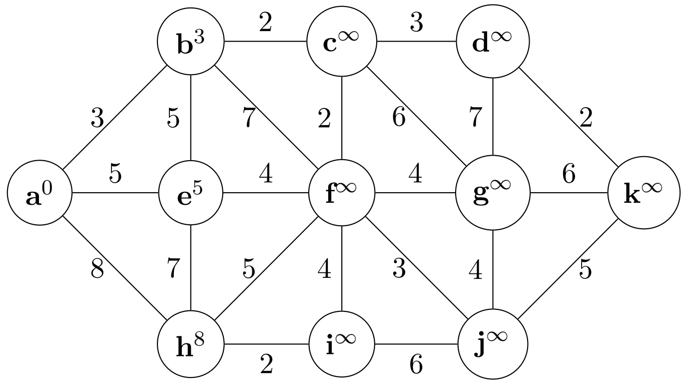

Figure 6: The graph after node a has visited

The distance costs and routes from node a are as follows;

- b: 3 (a)
- e: 5 (a)
- h: 8 (a)

Visited Nodes : a

Step 2

By visiting the *unvisited* neighbouring nodes of b, we get the following graph.

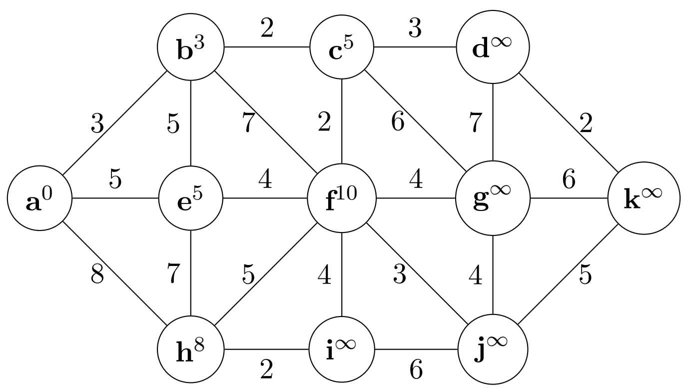

Figure 7: The graph after node b has visited

The distance costs and routes from node a are as follows;

- b: 3 (a)
- e: 5 (a)
- h: 8 (a)
- c: 5 (a,b)
- f: 10 (a,b)

Visited Nodes : a,b

Step 3

By visiting the *unvisited* neighbouring nodes of e, we get the following graph.

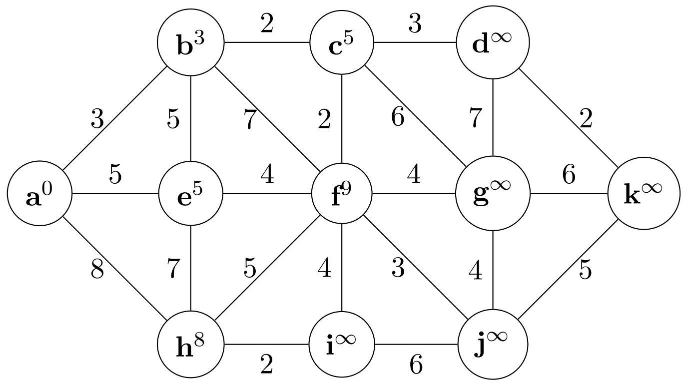

Figure 8: The graph after node e has visited

The distance costs and routes from node a are as follows;

- b: 3 (a)
- e: 5 (a)
- h: 8 (a)
- c: 5 (a,b)
- f: 9 (a,e)

Visited Nodes : a,b,e

Step 4

By visiting the *unvisited* neighbouring nodes of h, we get the following graph.

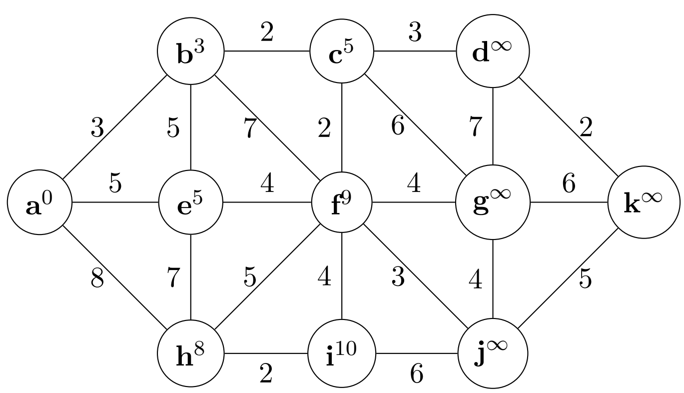

Figure 9: The graph after node h has visited

The distance costs and routes from node a are as follows;

- b: 3 (a)
- e: 5 (a)
- h: 8 (a)

- c: 5 (a,b)
- f: 9 (a,e)
- i: 10 (a,h)

Visited Nodes : a,b,e,h

Step 5

By visiting the *unvisited* neighbouring nodes of c, we get the following graph.

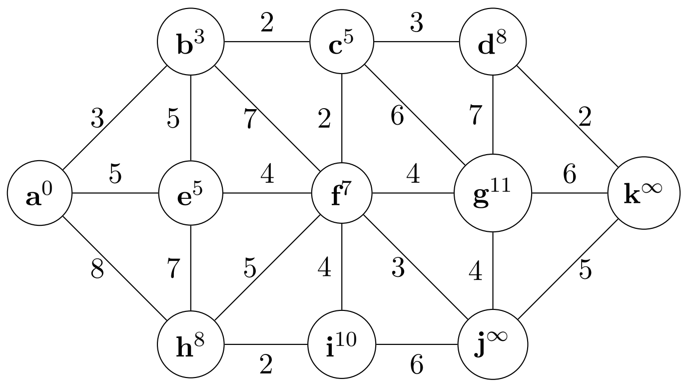

Figure 10: The graph after node c has visited

The distance costs and routes from node a are as follows;

- b: 3 (a)
- e: 5 (a)
- h: 8 (a)
- c: 5 (a,b)
- f: 7 (a,b,c)
- i: 10 (a,h)
- d: 8 (a,b,c)
- g: 11 (a,b,c)

Visited Nodes : a,b,e,h,c

Step 6

By visiting the *unvisited* neighbouring nodes of f, we get the following graph.

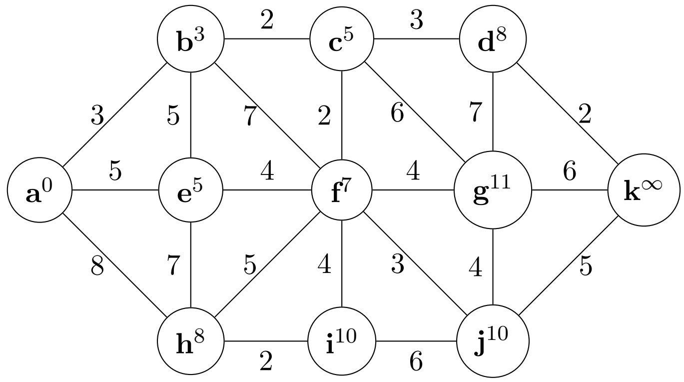

Figure 11: The graph after node f has visited

The distance costs and routes from node a are as follows;

- b: 3 (a)
- e: 5 (a)
- h: 8 (a)
- c: 5 (a,b)
- f: 7 (a,b,c)
- i: 10 (a,h)
- d: 8 (a,b,c)
- g: 11 (a,b,c)
- j: 10 (a,b,c,f)

Visited Nodes : a,b,e,h,c,f

Step 7

By visiting the *unvisited* neighbouring nodes of i, we get the following graph.

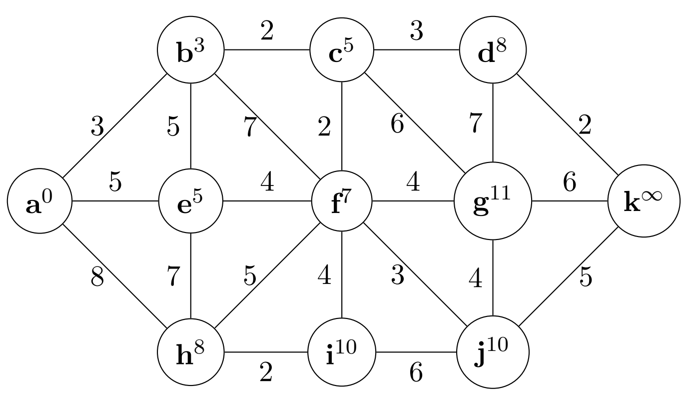

Figure 12: The graph after node i has visited

The distance costs and routes from node a are as follows;

- b: 3 (a)

- e: 5 (a)
- h: 8 (a)
- c: 5 (a,b)
- f: 7 (a,b,c)
- i: 10 (a,h)
- d: 8 (a,b,c)
- g: 11 (a,b,c)
- j: 10 (a,b,c,f)

Visited Nodes : a,b,e,h,c,f,i

Step 8

By visiting the *unvisited* neighbouring nodes of d, we get the following graph.

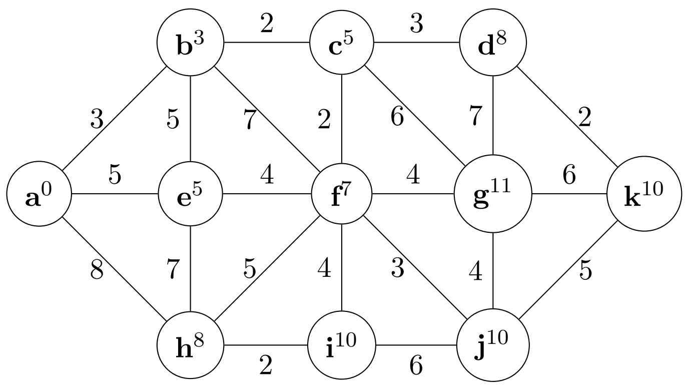

Figure 13: The graph after node d has visited

The distance costs and routes from node a are as follows;

- b: 3 (a)
- e: 5 (a)
- h: 8 (a)
- c: 5 (a,b)
- f: 7 (a,b,c)
- i: 10 (a,h)
- d: 8 (a,b,c)
- g: 11 (a,b,c)
- j: 10 (a,b,c,f)
- k: 10 (a,b,c,d)

Visited Nodes : a,b,e,h,c,f,i,d

Step 9

By visiting the *unvisited* neighbouring nodes of j, we get the following graph.

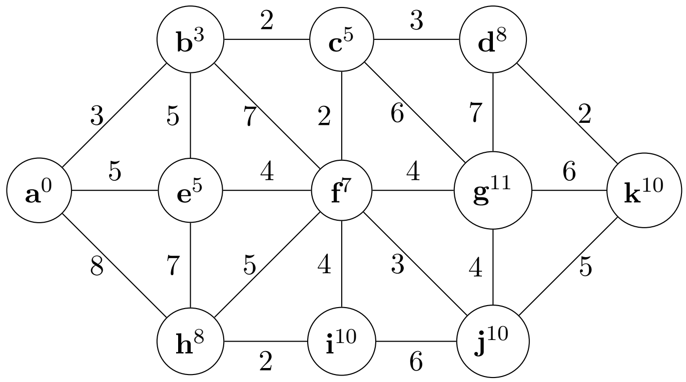

Figure 14: The graph after node j has visited

The distance costs and routes from node a are as follows;

- b: 3 (a)
- e: 5 (a)
- h: 8 (a)
- c: 5 (a,b)
- f: 7 (a,b,c)
- i: 10 (a,h)
- d: 8 (a,b,c)
- g: 11 (a,b,c)
- j: 10 (a,b,c,f)
- k: 10 (a,b,c,d)

Visited Nodes : a,b,e,h,c,f,i,d,j

Step 10

By visiting the *unvisited* neighbouring nodes of d, we get the following graph.

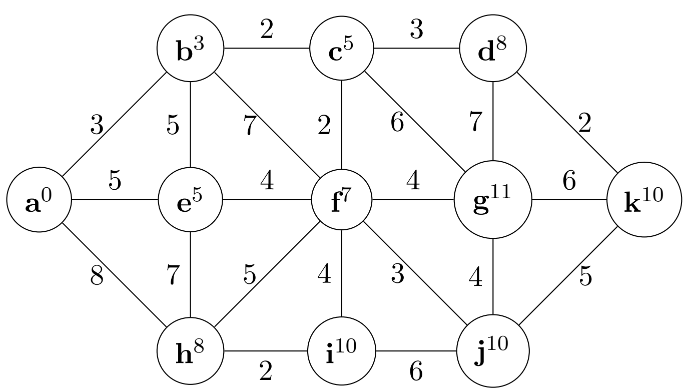

Figure 15: The graph after node d has visited

The distance costs and routes from node a are as follows;

- b: 3 (a)
- e: 5 (a)
- h: 8 (a)
- c: 5 (a,b)
- f: 7 (a,b,c)
- i: 10 (a,h)
- d: 8 (a,b,c)
- g: 11 (a,b,c)
- j: 10 (a,b,c,f)
- k: 10 (a,b,c,d)

Visited Nodes : a,b,e,h,c,f,i,d,j,g

Step 11

Since there are no *unvisited* neighbouring nodes of j, we concluded our traversal.
The shortest path from a to j by using **Dijkstra's Algorithm** is: $a \to b \to c \to f \to j$, and it has a distance cost of 10.

b)

By following the choices below,

Choice	Edge	Weight	Reason
1	[a,b]	3	From the vertex a, we start with picking the smallest weighted edge.
2	[b,c]	2	From the vertices a and b, the smallest weighted edge is $b-c$, with a weight of 2.
3	[c,f]	2	From the vertices a, b and c, the smallest weighted edge is $c-f$, with a weight of 2.
4	[c,d]	3	From the vertices a, b,c, and f, the smallest weighted edge is $c-d$, with a weight of 3.
5	[d,k]	2	From the vertices a, b, c, d and f, the smallest weighted edge is $d-k$, with a weight of 2.
6	[f,j]	3	From the vertices a, b, c, f, d and k, the smallest weighted edge is $f-j$, with weights of 3.
7	[f,i]	4	From the vertices a, b, c, f, d, k and j, the smallest weighted edges are $f-i$ and $f-e$, and $f-g$, with a weight of 4. We can choose any of them, and we proceed with $f-i$.
8	[i,h]	2	From the vertices a, b, c, f, d, k, i and j, the smallest weighted edge is $i-h$, with weight of 2.
9	[f,e]	4	From the vertices a, b, c, f, d, k,h, i and j, the smallest weighted edges are $f-e$ and $j-g$, and $f-g$, with a weight of 4. We can choose any of them, and we proceed with $f-e$.
10	[j,g]	4	From the vertices a, b, c, f, d, k,h, i and j, the smallest weighted edges are $j-g$ and $f-g$ with a weight of 4. We can choose any of them, and we proceed with $j-g$.

Table 4: Procedure of creating a minimum spanning tree of Graph G in Q5 produced using Prim's Algorithm

We conclude the minimum spanning tree, since picking any of the remaining edges would form *simple circuits*.

The minimum spanning tree of the graph can be seen below(the used edges to find the minimum spanning tree, are denoted with a *thicker* edge).

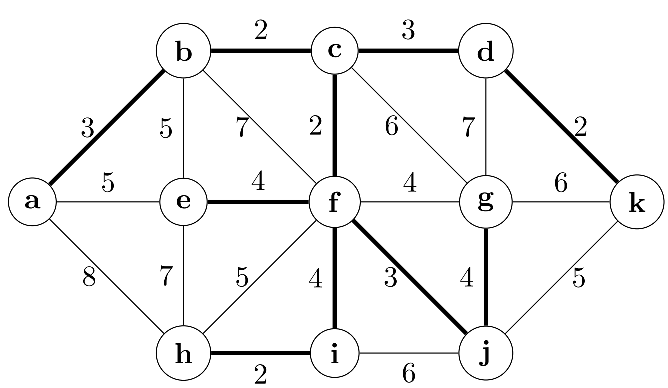

Figure 16: A minimum spanning tree of Graph G in Q5 produced using Prim's Algorithm

Answer 6

a)

There are **7** vertices, **6** edges on T, and it has a height of **3**.

b)

<a:17>,<b:13>,<c:24>,<d:19>,<e:43>,<f:23>,<g:58>

c)

<b:13>,<d:19>,<f:23>,<g:58>,<e:43>,<c:24>,<a:17>

d)

<b:13>,<a:17>,<d:19>,<c:24>,<f:23>,<e:43>,<g:58>

e)

- The tree is called a *full m-ary tree* if every internal vertex has exactly m children. An *m-ary* tree with $m = 2$ is called a binary tree.(Definition 3, pg.784)
- When we look at T, we can see that <a:17>,<c:24>,and <e:43> have 2 children each. Therefore T is a full binary tree.

f)

T is a full binary tree however, not all the leaves are at the same level. From that, T is not a complete binary tree.

g)

- On a Binary Search Tree, if we do a Inorder Traversal, since the inorder traversal would follow left-root-right order, and since the left child will always be smaller than the root, similarly the right child will always be greater than the root; we should get ordered keys as the output.
- When we take a look at the inorder traversal of T, we can see that the pair <c:24>,<f:23> is not sorted properly.
- Therefore we can easily say that the given tree T is not a binary search tree.

h)

One can see that by developing a recursive definition, $\mathrm{n}_h = \mathrm{n}_{h-1} + 2$; with initial condition $\mathrm{n}_0 = 1$.
Where h is the height and n_h is the minimum number of nodes for a full binary tree with height h
$\mathrm{n}_5 =$**11**

i)

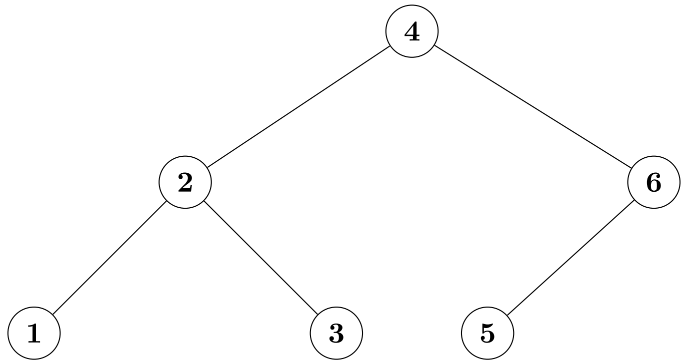

Figure 17: Binary Search Tree by using the integer keys {1,2,3,4,5,6}, employing the $\leq$ relation defined on $\mathbf{Z}$*$\mathbf{Z}$

j)

- Sequence to find 1: $\texttt{4} \xrightarrow{\text{left}} \texttt{2} \xrightarrow{\text{left}} \texttt{1}$
- Sequence to find 6: $\texttt{4} \xrightarrow{\text{right}} \texttt{6}$

k)

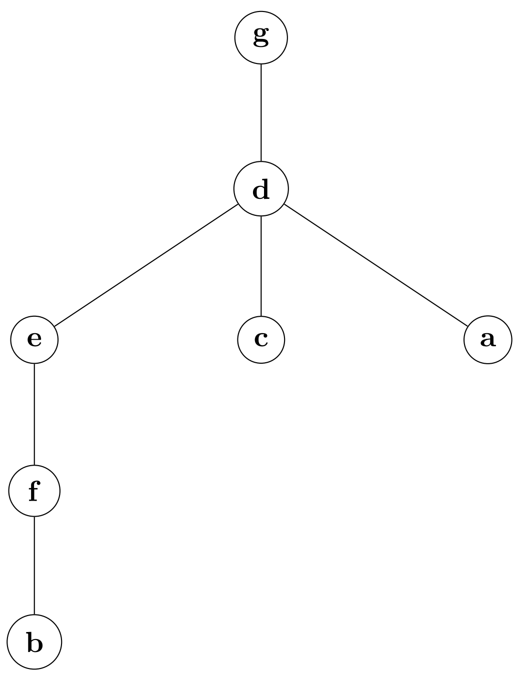

Figure 18: Spanning tree for the directed graph G in Q1 via 'Breadth-first Search'

l)

One can see that by developing a recursive definition, $h_k = h_{k-1} + 1$; with initial condition $h_1 = 0$. Where k is the number of vertices in binary search tree and h_k is the maximum height to create a binary search tree containing k vertices. Therefore, $h_k = h_{k-1} + 1$