Student Information

Full Name : Mustafa Ozan ALPAY
Id Number : 2309615

Answer 1

$$S = \{\text{“”}, \text{“1”}, \text{“0”}, \text{“10”}, \text{“01”}, \text{“010”}, \text{“011”}, \text{“101”}, \text{“110”}\}$$

a

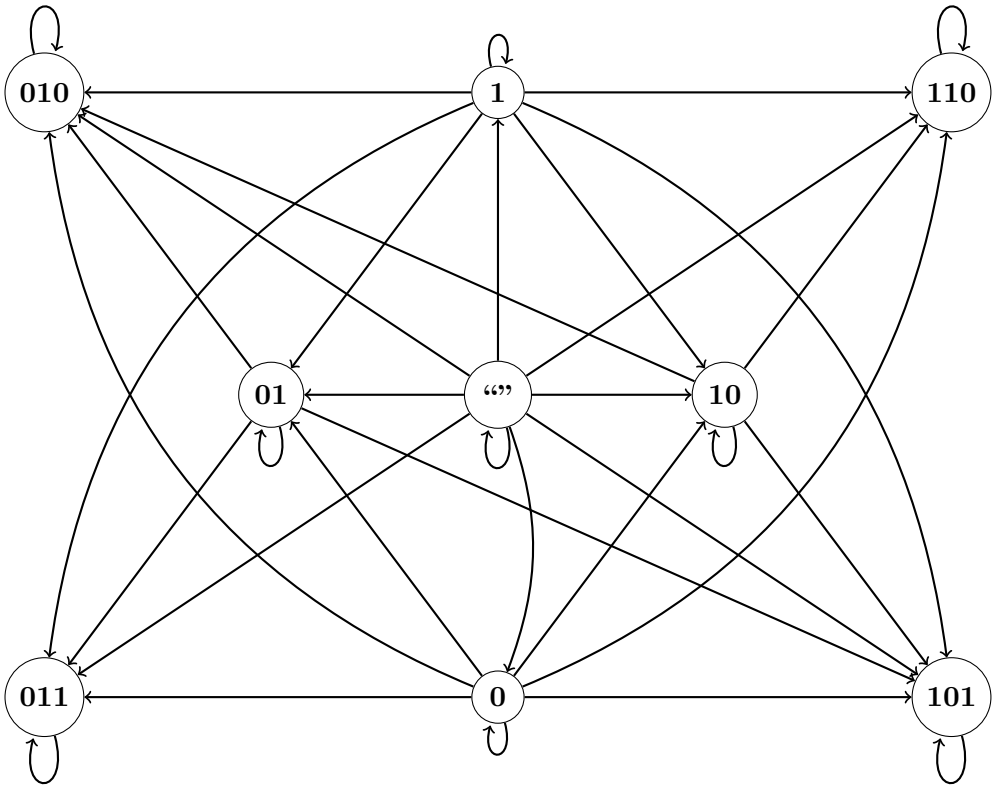

Figure 1: R in Q1, as a directed graph.

b

- **Definition 1** from the Section 9.6 of the textbook says that

  > A relation R on a set S is called a partial ordering if it is reflexive, antisymmetric, and transitive. A set S together with a partial ordering R is called a poset.

  Therefore, we need to prove that Graph 1 is reflexive, antisymmetric, and transitive.

- **Reflexivity**: Since each element has a loop to itself, the graph is reflexive.
- **Antisymmetry**: Since there are no two edges between two vertices with different directions, the graph is antisymmetric.
- **Transitivity**: By looking at the graph, we can see that vertices $1, 0, 01, 10$ have edges to and from them. Investigating these vertices should be enough to determine if the graph is transitive.

- Vertex 01 has 3 incoming edges: “”, 0, and 1. When we look at the outgoing edges of vertex 01, we can see that 010, 011, and 101 also have incoming edges from Vertex 01's incoming edges.
- Similarly, Vertex 10 has 3 incoming edges: “”, 0, and 1. When we look at the outgoing edges of Vertex 01, we can see that 010, 101, and 110 also have incoming edges from Vertex 10's incoming edges.
- Vertex 1 has 1 incoming edge: “”. When we look at the outgoing edges of Vertex 1, we can see that 010, 01, 011, 10, 101, and 110 also have incoming edges from Vertex 1's incoming edge.
- Similarly, Vertex 0 has 1 incoming edge: “”. When we look at the outgoing edges of Vertex 0, we can see that 010, 01, 011, 10, 101, and 110 also have incoming edges from Vertex 0's incoming edge.

- Since we have shown that that Graph 1 is reflexive, antisymmetric, and transitive, the partial ordering (S, R) is a poset.

c

- **Definition 3** from the Section 9.6 of the textbook says that

  > 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.

- Since (S, R) is a poset, and since each member of S is a string, thus comparable, S is a totally ordered set, and R is a total order.

d

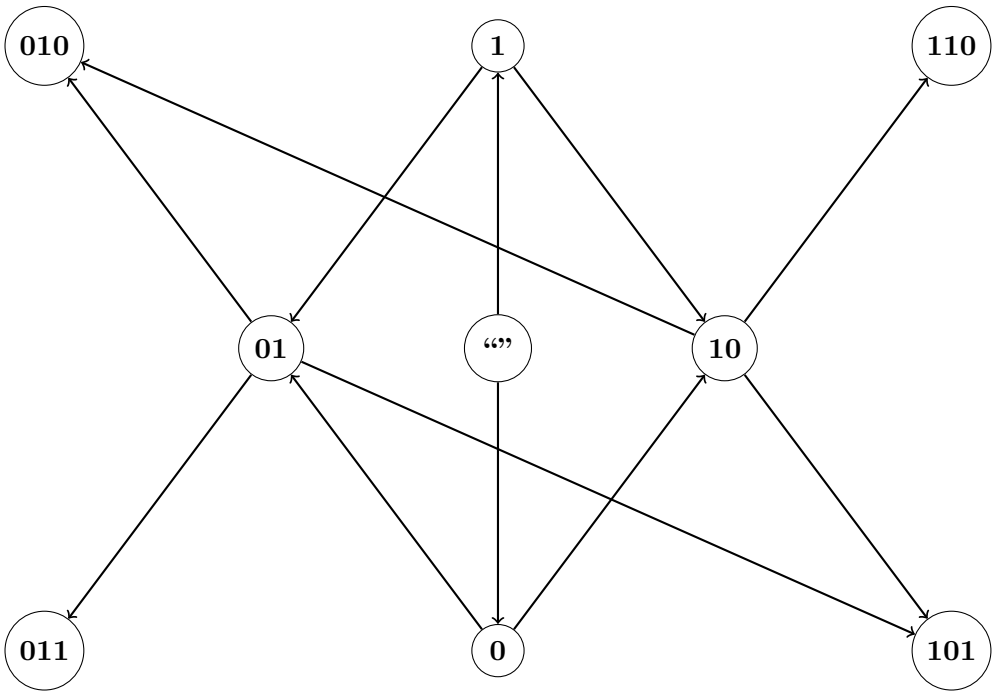

Figure 2: (S,R) as a Hasse Diagram.

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.

- Let us pick 011 and 10. When we look at the Hasse Diagram in Figure 2, we can clearly see that they have two great lower bounds, namely 0 and 1. However when we try to find a least upper bound, we can see that it does not exist.
- Therefore, the Hasse Diagram in Figure 2 is not a lattice.

Answer 2

a

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

Table 1: Adjacency List for Graph G in Q2

b

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

Figure 3: Adjacency Matrix for Graph G in Q2

c

	In-degrees Count	**Out-degrees Count**
a	1	3
b	4	2
c	3	1
d	2	1
e	1	2
f	3	3
g	0	2

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

d

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

e

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

f

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

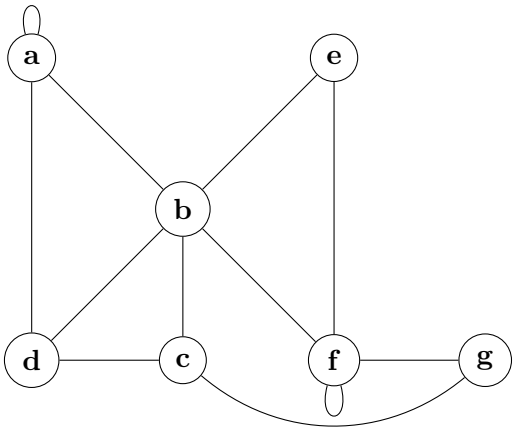

Figure 4: The Underlying Undirected Graph of Graph G in Q2.

- Since all vertices are connected to at least one other vertex, we can create paths between any given two vertices.
- By using the Definition 5 from the Section 10.4 of the book,

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

  we can say that *Graph G* is **weakly-connected**.

g

The strongly-connected components of G are as follows;

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

h

By using the adjacency matrix from **part b**, we can generate the following adjacency matrix for the subgraph H of G.

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 5: Adjacency Matrix for Subgraph H

Since we need to find the number of different paths of length 3, we need the A^3 matrix. (By using the Theorem 2 from the Section 10.4 of the book)

$$A^3 = \begin{bmatrix} 1 & 3 & 3 & 2 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Therefore there are;

- **1** path on $a \to a$
- **3** paths on $a \to b$
- **3** paths on $a \to c$
- **2** paths on $a \to d$
- **0** paths on $b \to a$
- **1** path on $b \to b$
- **1** path on $b \to c$
- **1** path on $b \to d$
- **0** paths on $c \to a$
- **1** path on $c \to b$
- **1** paths on $c \to c$

- **0** paths on $c \to d$
- **0** paths on $d \to a$
- **0** paths on $d \to b$
- **1** path on $d \to c$
- **1** path on $d \to d$

exist.

Answer 3

Vertex	a	b	c	d	e	f	g	h	i	j	k
Degree	2	6	4	4	6	6	4	4	4	6	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 all vertices are of even degree.
- Therefore, *Graph G* does not have an **Euler Path**.

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 11 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 all vertices are of even degree.
- Therefore, *Graph G* has an **Euler Circuit**.
- An example **Euler Circuit**: $j \to j \to c \to f \to e \to a \to b \to b \to e \to f \to b \to f \to i \to e \to h \to h \to i \to c \to d \to g \to j \to k \to g \to d$

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*;

  $k \to j \to g \to d \to c \to i \to h \to e \to a \to b \to f$

- 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 11 vertices, if we find a pair of nonadjacent vertices with summed degrees less than 11, we can say that the given graph does not have a **Hamilton Circuit**.

- Let us pick the nonadjacent vertices a and k. When we look at Table 3, we can see that their degrees are 2 and 2, respectively.

- We can easily see that $deg(a) + deg(k) \geq 11$ does not hold, and therefore *Graph G* does not have a **Hamilton Circuit**.

Answer 4

a

- By the definition of the *Complete Bipartite Graph*, $K_{m,n}$ has two different subsets of vertices, namely m and n vertices each.

- Therefore, the total number of vertices are $m + n$.

- Since each vertex from the subset m will have edges with each vertex from subset n, we can say that the total number of edges are $m \cdot n$.

b

- **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 *Complete Bipartite Graphs* are simple, we can continue with the further requirements.

- Let us say $m + n = \beta$, and since m is odd and n is even, $\beta > 3$ also holds.

- Since each vertex in subset m has edges with each vertex in subset n, the only nonadjacent vertices are the vertices within the same subset.

- Each vertex in subset m will have exactly n edges; similarly each vertex in subset n will have exactly m edges.

- By applying *Ore's Theorem* to two vertices in subset m, we get;

$$deg(m_i) + deg(m_j) \geq m + n, \text{ where } m_i, m_j \in m$$

- Since the degree of each vertex in m is n, we can write the following;

$$n + n \geq m + n$$

$$n \geq m \tag{1}$$

- Similarly, by applying *Ore's Theorem* to two vertices in subset n, we get;

$$deg(n_p) + deg(n_r) \geq m + n, \text{ where } n_p, n_r \in n$$

- Since the degree of each vertex in n is m, we can write the following;

$$m + m \geq m + n$$

$$m \geq n \tag{2}$$

- By combining inequalities (1) and (2), we can see that the solution only exists when $m = n$. Since by the initial conditions we are given that m is odd and n is even, $m = n$ does not hold. Therefore, we can conclude that $K_{m,n}$ with odd m and even n does not have a Hamiltonian circuit.

Answer 5

a

Step 0

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

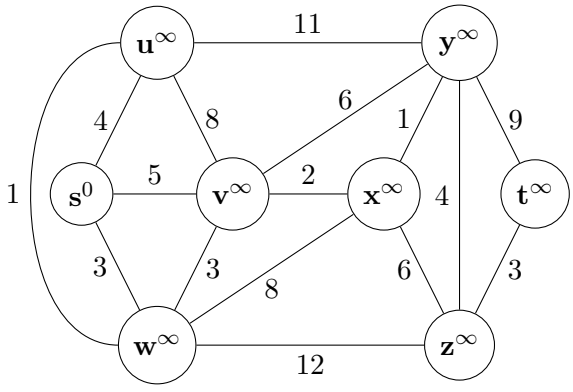

Figure 6: Initial condition

Step 1

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

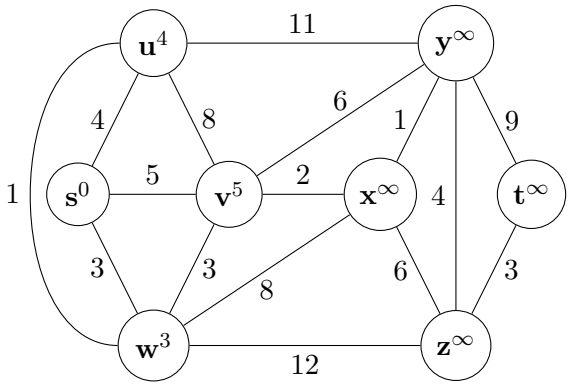

Figure 7: The graph after node s has visited

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

- w: 3 (s)
- u: 4 (s)
- v: 5 (s)

Visited Nodes : s

Step 2

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

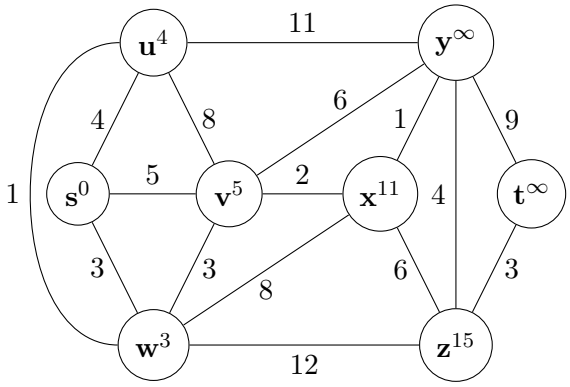

Figure 8: The graph after node w has visited

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

- w: 3 (s)
- u: 4 (s)
- v: 5 (s)
- x: 11 (s,w)
- z: 15 (s,w)

Visited Nodes : s, w

Step 3

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

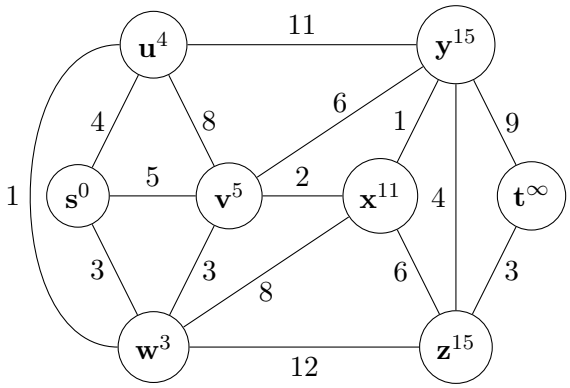

Figure 9: The graph after node u has visited

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

- w: 3 (s)
- u: 4 (s)
- v: 5 (s)
- x: 11 (s,w)
- z: 15 (s,w)
- y: 15 (s,u)

Visited Nodes : s, w, u

Step 4

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

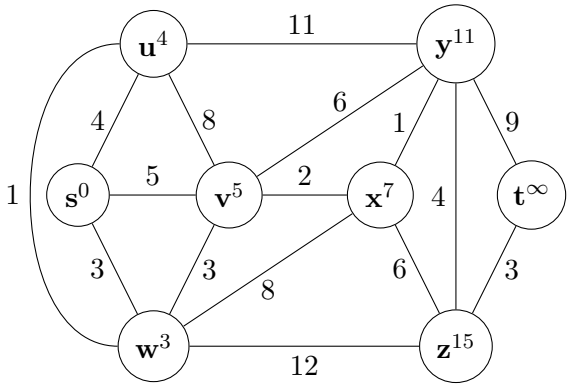

Figure 10: The graph after node v has visited

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

- w: 3 (s)
- u: 4 (s)

- v: 5 (s)
- x: 7 (s,v)
- y: 11 (s,v)
- z: 15 (s,w)

Visited Nodes : s, w, u, v

Step 5

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

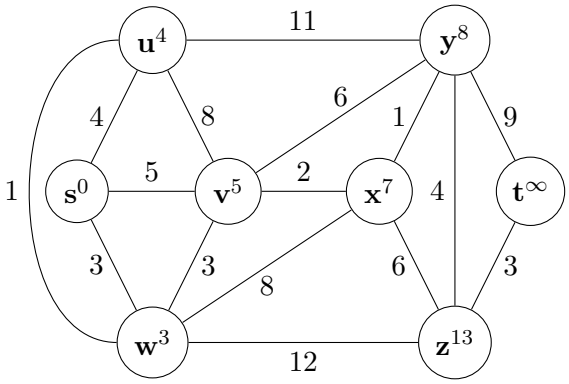

Figure 11: The graph after node x has visited

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

- w: 3 (s)
- u: 4 (s)
- v: 5 (s)
- x: 7 (s,v)
- y: 8 (s,v,x)
- z: 13 (s,v,x)

Visited Nodes : s, w, u, v, x

Step 6

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

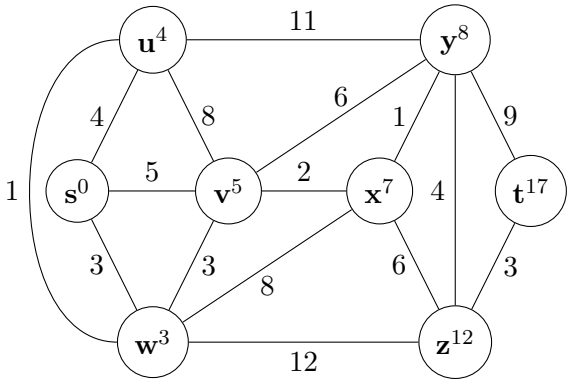

Figure 12: The graph after node y has visited

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

- w: 3 (s)
- u: 4 (s)
- v: 5 (s)
- x: 7 (s,v)
- y: 8 (s,v,x)
- z: 12 (s,v,x,y)
- t: 17 (s,v,x,y)

Visited Nodes : s, w, u, v, x, y

Step 7

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

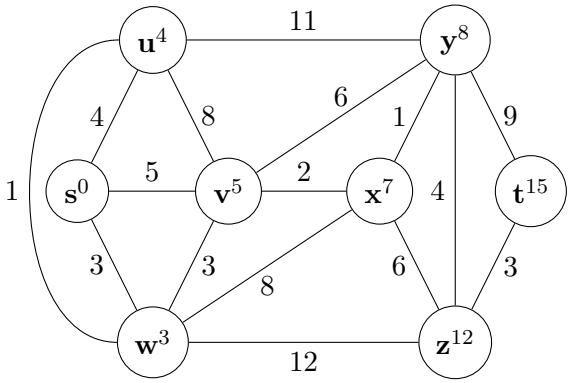

Figure 13: The graph after node y has visited

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

- w: 3 (s)
- u: 4 (s)
- v: 5 (s)

- x: 7 (s,v)
- y: 8 (s,v,x)
- z: 12 (s,v,x,y)
- t: 15 (s,v,x,y,z)

Visited Nodes : s, w, u, v, x, y, z

Step 8

Since there are no *unvisited* neighbouring nodes of t, we concluded our traversal.
The shortest path from s to t by using **Dijkstra's Algorithm** is: $s \to v \to x \to y \to z \to t$, and it has a distance cost of 15.

b

By following the choices below,

Choice	Edge	Weight	Reason
1	[x,y]	1	From the vertex x, we start with picking the smallest weighted edge.
2	[x,v]	2	From the vertices x and y, the smallest weighted edge is $x - v$, with a weight of 2.
3	[v,w]	3	From the vertices x, y and v, the smallest weighted edge is $v - w$, with a weight of 3.
4	[w,u]	1	From the vertices x, y, v and w, the smallest weighted edge is $w - u$, with a weight of 1.
5	[w,s]	3	From the vertices x, y, v, w and u, the smallest weighted edge is $w - s$, with a weight of 3.
6	[y,z]	4	From the vertices x, y, v, w, u and s, the smallest weighted edges are $s - u$ and $y - z$, with weights of 4. But since picking $u - s$ would form a *simple circuit*, we cannot pick it. Therefore we pick $y - z$.
7	[z,t]	3	From the vertices x, y, v, w, u, s and z, the smallest weighted edge is $z - t$, with a weight of 3.

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 above (the used edges are denoted with a *ultra thick* edge).

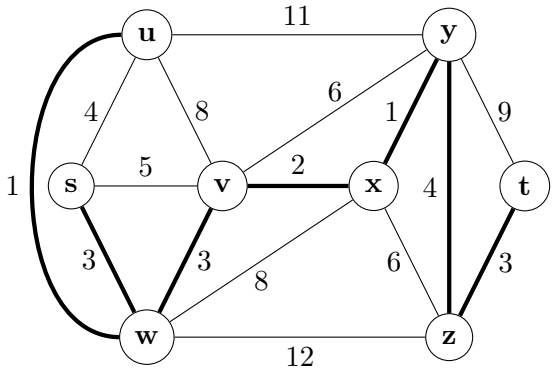

Figure 14: A minimum spanning tree of Graph G in Q5 produced using Prim's Algorithm

c

The initial minimum spanning tree of the graph as follows (the used edges are denoted with a *ultra thick* edge).

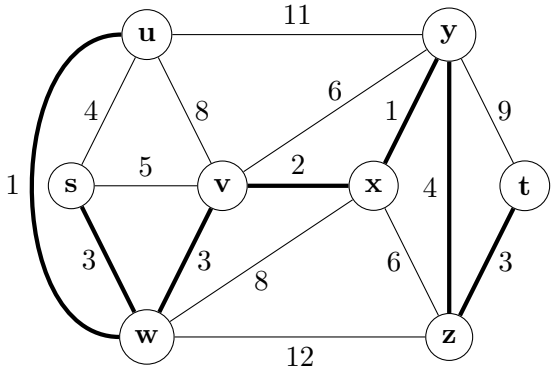

Figure 15: A minimum spanning tree of Graph G in Q5 produced using Prim's Algorithm

Adding (s,x,1)

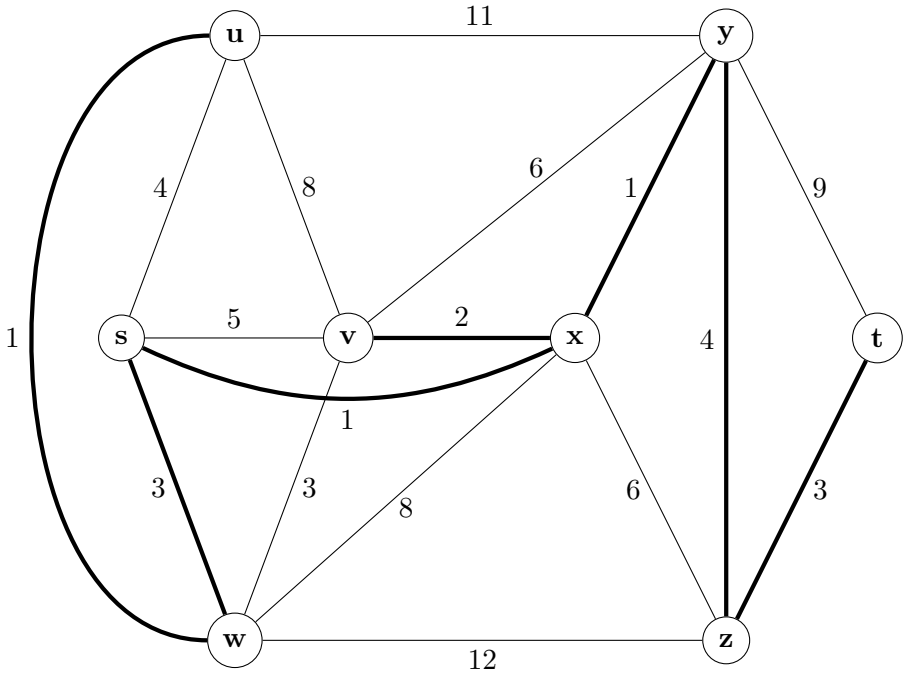

Figure 16: The minimum spanning tree of Graph G in Q5 after adding (s,x,1)

Adding (t,u,6)

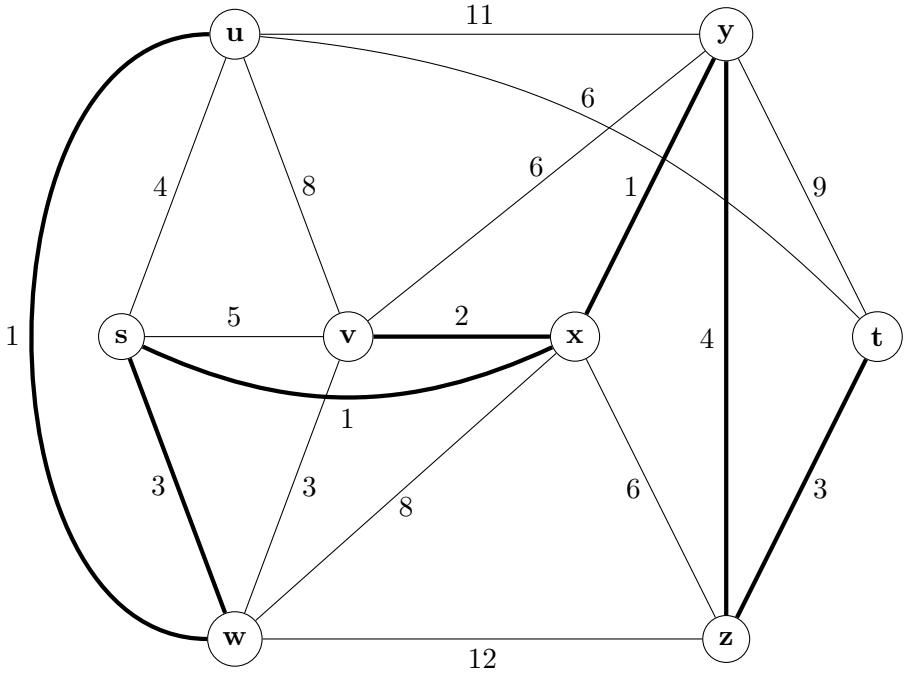

Figure 17: The minimum spanning tree of Graph G in Q5 after adding (t,u,6)

Adding (s,z,-3)

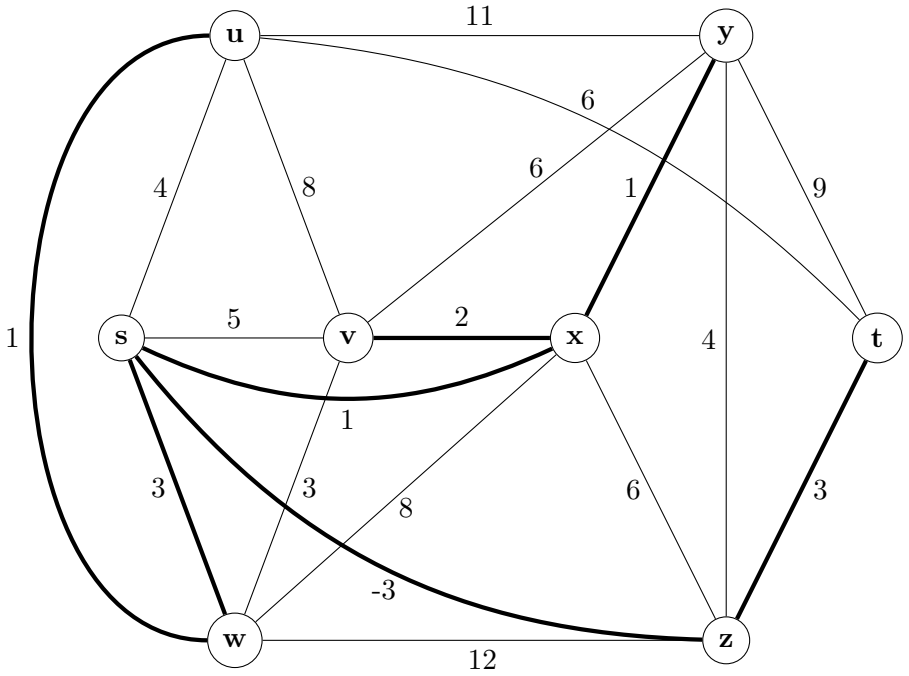

Figure 18: The minimum spanning tree of Graph G in Q5 after adding (s,z,-3)

Adding (u,y,3)

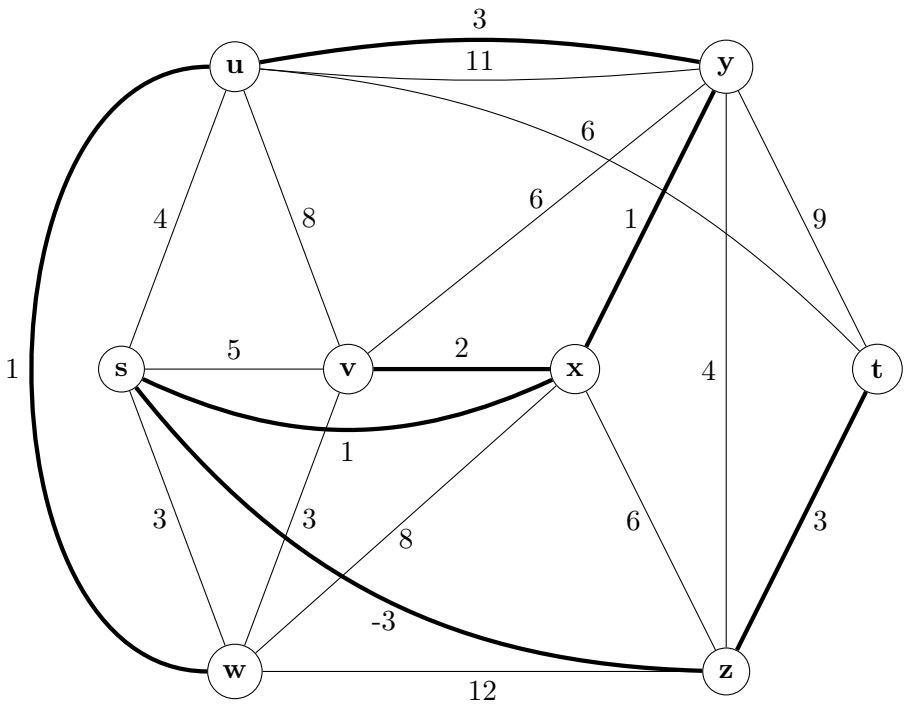

Figure 19: The minimum spanning tree of Graph G in Q5 after adding (u,y,3)

Adding (w,z,-1)

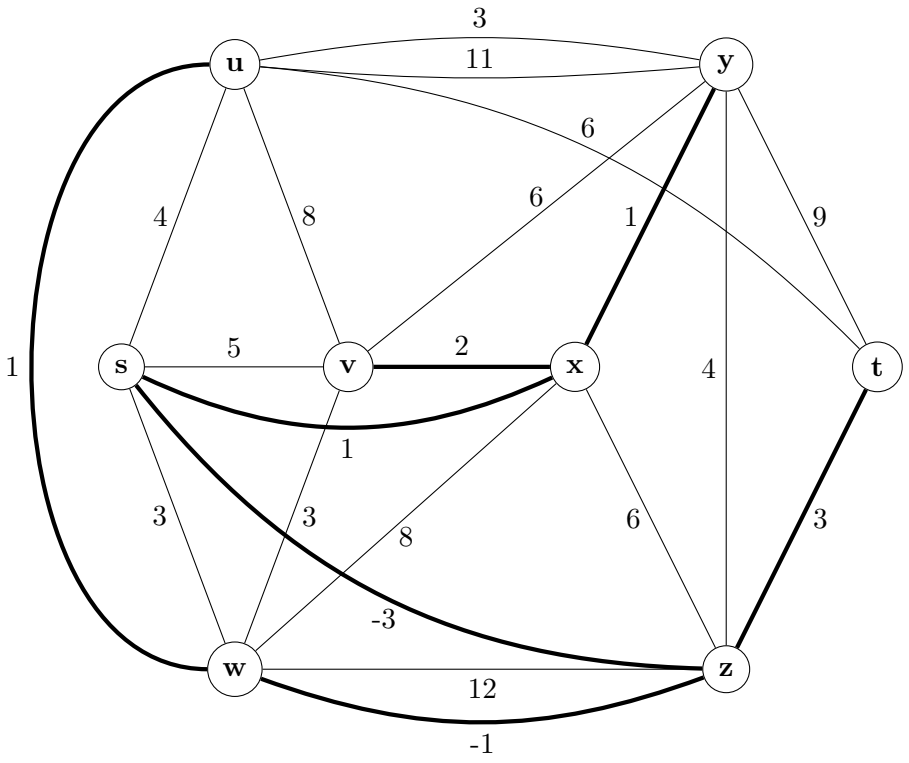

Figure 20: The minimum spanning tree of Graph G in Q5 after adding (w,z,-1)

d

Yes we can update the shortest path without calling **Dijkstra's Algorithm**.
In our initial findings, the shortest path had the following route: $s \to v \to x \to y \to z \to t$, and the cost between each vertex is as follows: $\{5, 2, 1, 4, 3\}$. If any of the newly added routes includes any of the two points in our initial route, it *may* shorten our path.

1. After adding $(s, x, 1)$, the distance between s and x shortens, and becomes 1. We can simply update the shortest path cost as $\{1, 1, 4, 3\}$ with the route $s \to x \to y \to z \to t$, since $s \to v \to x$ has a cost of 7, and the newly added edge has a cost of 1.

2. After adding $(t, u, 6)$, it does not affect any of the routes.

3. After adding $(s, z, -3)$, the distance between s and z shortens, and becomes -3. We can simply update the shortest path cost as $\{-3, 3\}$ with the route $s \to z \to t$, since $s \to x \to y \to z$ has a cost of 6, and the newly added edge has a cost of -3.

4. After adding $(u, y, 3)$, it does not affect any of the routes.

5. After adding $(w, z, -1)$, it does not affect any of the routes.

Answer 6

a

There are **13** vertices, **12** edges on T, and it has a height of **4**.

b

<w:33>,<s:26>,<m:35>,<t:37>,<q:34>,<x:41>,<n:61>,<y:71>,<u:63>,<z:99>,<v:98>,
<r:75>,<p:42>

c

<s:26>,<w:33>,<q:34>,<m:35>,<t:37>,<p:42>,<x:41>,<u:63>,<n:61>,<y:71>,<r:75>,
<v:98>,<z:99>

d

<p:42>,<q:34>,<s:26>,<w:33>,<t:37>,<m:35>,<r:75>,<u:63>,<x:41>,<y:71>,<n:61>,
<v:98>,<z:99>

e

- **Definition 3** from the Section 11.1 of the textbook says that

  > 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.

- When we look at T, we can see that <s:26>,<t:37>,<y:71>,<v:98> have only 1 children each. Therefore T is not a full binary tree.

f

- 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 pairs <p:42>,<x:41> and <u:63>,<n:61> are not sorted properly.
- Therefore we can easily say that the given tree T is not a binary search tree.

g

- **Theorem 5** from the Section 11.1 of the textbook says that

  > There are at most m^h leaves in an m-ary tree of height h.

- Since our tree is 3-ary, and it's a full tree, there must be exactly 3^n leaves at height n.
- If we sum all leaves from $n = 0$ to h, we would get the following sum;

$$\sum_{n=0}^{h} 3^n$$

  which equals to

$$\sum_{n=0}^{h} 3^n = \frac{1}{2} \cdot \left(3^{h+1} - 1\right)$$

h

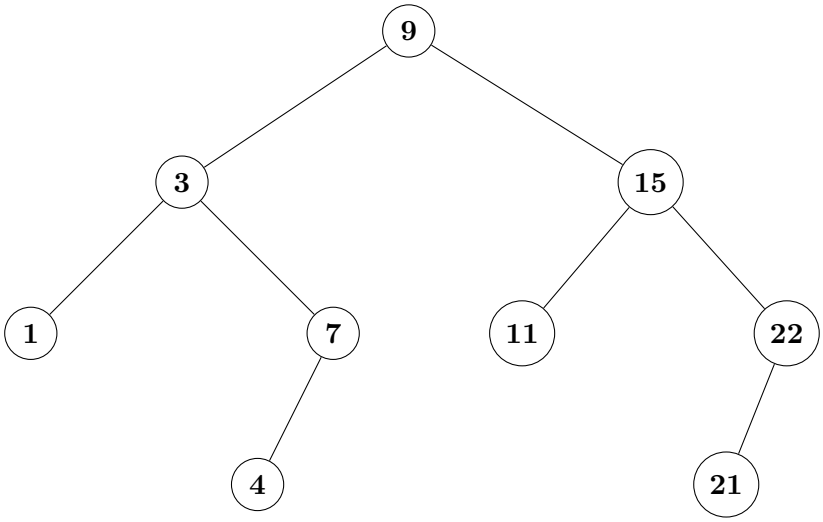

Figure 21: Binary Search Tree T with minimum height for Q6.h

i

- Sequence to find 2: `9` $\xrightarrow{\text{left}}$ `3` $\xrightarrow{\text{left}}$ `1` $\xrightarrow{\text{right}}$ `EMPTY : NOT FOUND`
- Sequence to find 22: `9` $\xrightarrow{\text{right}}$ `15` $\xrightarrow{\text{right}}$ `22 :  FOUND`

j

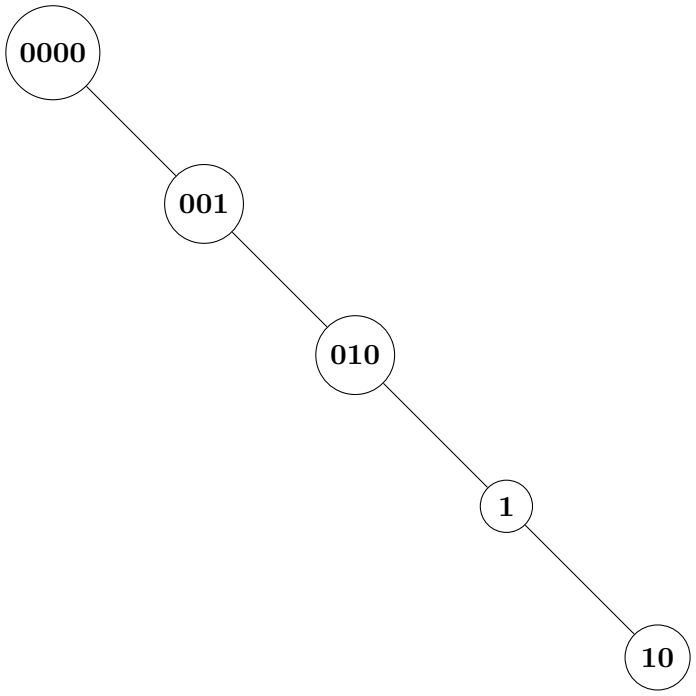

Figure 22: Binary Search Tree T with maximum height for Q6.j

k

- Sequence to find 001: `0000` $\xrightarrow{\text{right}}$ `001 :  FOUND`

- Sequence to find 011: `0000` $\xrightarrow{\text{right}}$ `001` $\xrightarrow{\text{right}}$ `010` $\xrightarrow{\text{right}}$ `1` $\xrightarrow{\text{left}}$ `EMPTY : NOT FOUND`

l

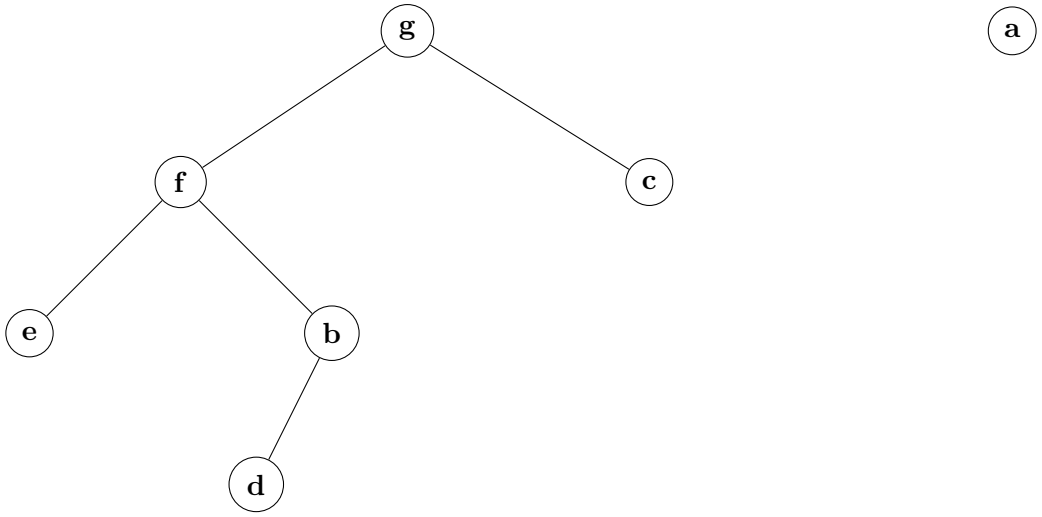

Figure 23: Spanning Forest for Graph G via Breadth-first Search