4 UNIT

Graphs

CONTENTS

Part-1 : Graphs : Terminology used with Graphs ....**4–2A to 4–4A**

Part-2 : Data Structure for Graph Representations : Adjacency Matrices, Adjacency List, Adjacency ....**4–4A to 4–7A**

Part-3 : Graph Traversal : Depth First Search and Breadth First Search, Connected Component ....**4–7A to 4–12A**

Part-4 : Spanning Tree, Minimum Cost Spanning Trees : Prim's and Kruskal's Algorithm .... **4–12A to 4–28A**

Part-5 : Transitive Closure and Shortest Path Algorithm : Warshall Algorithm and Dijkstra Algorithm .... **4–29A to 4–38A**

PART-1

Graphs : Terminology used with Graph.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.1. What is a graph ? Describe various types of graph. Briefly explain few applications of graph.

Answer

1. A graph is a non-linear data structure consisting of nodes and edges.
2. A graph is a finite sets of vertices (or nodes) and set of edges which connect a pair of nodes.

Types of graph :

1. Undirected graph :

a. If the pair of vertices is unordered then graph G is called an undirected graph.

b. That means if there is an edge between v_1 and v_2 then it can be represented as (v_1, v_2) or (v_2, v_1) also. It is shown in Fig. 4.1.1.

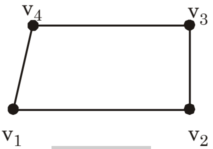


Fig. 4.1.1.

2. Directed graph :

a. If the pair of vertices is ordered then graph G is called directed graph.

b. That is, a directed graph or digraph is a graph which has ordered pair of vertices (v_1, v_2) where v_1 is the tail and v_2 is the head of the edge.

c. If the graph is directed then the line segments of arcs have arrow heads indicating the direction. It is shown in Fig. 4.1.2.

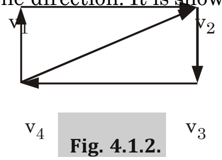


Fig. 4.1.2.

3. Weighted graph : A graph is said to be a weighted graph if all the edgesin it are labelled with some numbers. It is shown in the Fig. 4.1.3.

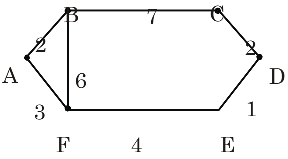


Fig. 4.1.3.

4. **Simple graph :** A graph or directed graph which does not have any self-loop or parallel edges is called a simple graph.
5. **Multi-graph :** A graph which has either a self-loop or parallel edges or both is called a multi-graph.
6. **Complete graph :**
   a. A graph is complete graph if each vertex is adjacent to every other vertex in graph or there is an edge between any pair of nodes in the graph.
   b. An undirected complete graph will contain $n(n-1)/2$ edges.
7. **Regular graph :**
   a. A graph is regular if every node is adjacent to the same number of nodes.
   b. Here every node is adjacent to 3 nodes.

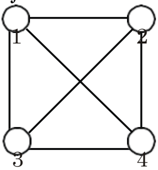
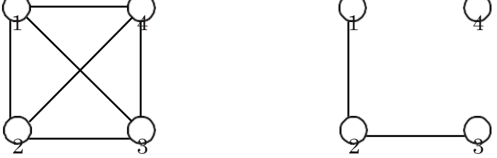


Fig. 4.1.4.

8. **Planar graph :** A graph is planar if it can be drawn in a plane without any two intersecting edges.
9. **Connected graph :**
   a. In a graph G, two vertices v_1 and v_2 are said to be connected if there is path in G from v_1 to v_2 or v_2 to v_1.
   b. Connected graph can be of two types :
      i. Strongly connected graph
      ii. Weakly connected graph



(a) Connected graph (b) Not connected graph

Fig. 4.1.5.

10. **Acyclic graph :** If a graph (digraph) does not have any cycle then it is called as acyclic graph.
11. **Cyclic graph :** A graph that has cycles is called a cyclic graph.
12. **Biconnected graph :** A graph with no articulation points is called a biconnected graph.

Applications of graph :

1. Graph is a non-linear data structure and is used to present various operations and algorithms.
2. Graphs are used for topological sorting.
3. Graphs are used to find shortest paths.
4. They are required to minimize some aspect of the graph, such as distance among all the vertices in the graph.

Que 4.2. **What is graph ? Discuss various terminologies used ingraph.**

Answer

Graph : Refer Q. 4.1, Page 4–2A, Unit-4.

Various terminologies used in graphs are :

1. **Self loop :** If there is an edge whose starting and end vertices are same that is (v_2, v_2) is an edge then it is called a self loop or simply a loop.
2. **Parallel edges :** A pair of edges *e* and *e*′ of *G* are said to be parallel iff they are incident on precisely the same vertices.
3. **Adjacent vertices :** A vertex *u* is adjacent to (or the neighbour of) other vertex *v* if there is an edge from *u* to *v*.
4. **Incidence :** In an undirected graph the edge (*u*, *v*) is incident on vertices *u* and *v*. In a digraph the edge (*u*, *v*) is incident from node *u* and is incident to node *v*.
5. **Degree of vertex :** The degree of a vertex is the number of edges incident to that vertex. In an undirected graph, the number of edges connected to a node is called the degree of that node.

PART-2

Data Structure for Graph Representations : Adjacency Matrices,Adjacency List, Adjacency.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.3. **Discuss the various types of representation of graph.**

Answer

Two types of graph representation are as follows :

1. **Matrix representation :** Matrices are commonly used to represent graphs for computer processing. Advantages of representing the graph in matrix lies on the fact that many results of matrix algebra can be readily applied to study the structural properties of graph from an algebraic point of view.

   a. **Adjacency matrix :**

      i. **Representation of undirected graph :** The adjacency matrixof a graph G with n vertices and no parallel edges is a $n \times n$matrix A = $[a_{ij}]$ whose elements are given by
      a_{ij} = 1, if there is an edge between i^{th} and j^{th} vertices
      = 0, if there is no edge between them

      ii. **Representation of directed graph :** The adjacency matrix of a digraph D, with n vertices is the matrix

      $A = [a_{ij}]_{n \times n}$ in which
      $a_{ij} = 1$ if arc (v_i, v_j) is in D
      $= 0$ otherwise

      For example : Representation of following undirected and directed graph is :

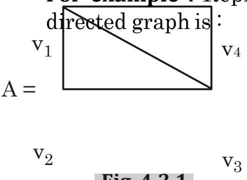


Fig. 4.3.1.

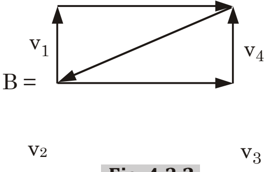


Fig. 4.3.2.

$$A = \begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & 0 & 1 & 1 & 1 \\ v_2 & 1 & 0 & 1 & 0 \\ v_3 & 1 & 1 & 0 & 1 \\ v_4 & 1 & 0 & 1 & 0 \end{array} \qquad B = \begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & 0 & 0 & 0 & 1 \\ v_2 & 1 & 0 & 1 & 0 \\ v_3 & 0 & 0 & 0 & 1 \\ v_4 & 0 & 1 & 0 & 0 \end{array}$$

   b. **Incidence matrix**

      i. **Representation of undirected graph :** Consider a undirected graph $G = (V, E)$ which has n vertices and m edges all labelled. The incidence matrix $I(G) = [b_{ij}]$, is then $n \times m$ matrix, where

      $b_{ij} = 1$ when edge e_j is incident with v_i
      $= 0$ otherwise

      ii. **Representation of directed graph :** The incidence matrix $I(D) = [b_{ij}]$ of digraph D with n vertices and m edges is the $n \times m$ matrix in which.

$$\begin{aligned} b_{ij} &= 1 && \text{if arc } j \text{ is directed away from vertex } v_i \\ &= -1 && \text{if arc } j \text{ is directed towards vertex } v_i \\ &= 0 && \text{otherwise.} \end{aligned}$$

Find the incidence matrix to represent the graph shown in Fig. 4.3.3.

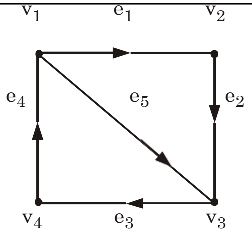


Fig. 4.3.3.

The incidence matrix of the digraph of Fig. 4.3.3 is

$$I(D) = \begin{bmatrix} 1 & 0 & 0 & -1 & 1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & -1 \\ 0 & 0 & -1 & 1 & 0 \end{bmatrix}$$

2. **Linked representation :**

i. In linked representation, the two nodes structures are used :

a. For non-weighted graph :

INFO	Adj-list

b. For weighted graph :

Weight	INFO	Adj-list

where Adj-list is the adjacency list *i.e.*, the list of vertices which are adjacent for the corresponding node.

ii. The header nodes in each list maintain a list of all adjacent vertices of that node for which the header node is meant.

iii. Suppose a directed graph

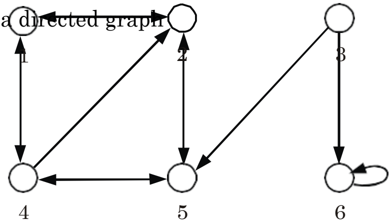


Fig. 4.3.4.

iv. The adjacency list representation of this graph.

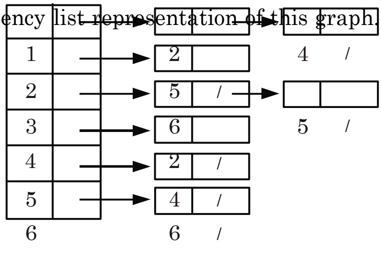


Fig. 4.3.5.

Que 4.4. **Explain adjacency multilists.**

Answer

1. Adjacency multilist representation maintains the lists as multilists, that is, lists in which nodes are shared among several lists.
2. For each edge there will be exactly one node, but this node will be in two lists *i.e.*, the adjacency lists for each of the two nodes, it is incident to. The node structure now becomes :

Mark	vertex 1	vertex 2	path 1	path 2

where mark is a one bit mark field that may be used whether or not the edge has been examined. The declarations in C are :

```
#define n 20
typedef struct edge {BOOLEAN mark;
      int vertex1, vertex2;
      NEXTEDGE path1, path2;
      }*NEXTEDGE;
      NEXTEDGE headnode [n];
```

PART-3

Graph Traversal : Depth First Search and Breadth First Search,Connected Component.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.5. **Write a short note on graph traversal.**

Answer

Traversing a graph :

i. Graph is represented by its nodes and edges, so traversal of each node is the traversing in graph.

ii. There are two standard ways of traversing a graph.

iii. One way is called a breadth first search, and the other is called a depth first search.

iv. During the execution of our algorithms, each node N of G will be in one of three states, called the status of N, as follows :

Status = 1 ⇒ (Ready state). The initial state of the node N.

Status = 2 ⇒ (Waiting state). The node N is on the queue or stack, waiting to be processed.

Status = 3 ⇒ (Processed state). The node N has been processed.

1. **Breadth First Search (BFS) :** The general idea behind a breadth firstsearch beginning at a starting node *A* is as follows :
   a. First we examine the starting node *A*.
   b. Then, we examine all the neighbours of *A*, and so on.
   c. We need to keep track of the neighbours of a node, and that no node is processed more than once.
   d. This is accomplished by using a queue to hold nodes that are waiting to be processed, and by using a field STATUS which tells us the current status of any node.

Algorithm : This algorithm executes a breadth first search on a graph *G* beginning at a starting node *A*.

```
i.   Initialize all nodes to ready state (STATUS=1).
ii.  Put the starting node A in queue and change its status to the waiting
     state (STATUS = 2).
iii. Repeat steps (iv) and (v) until queue is empty.
iv.  Remove the front node N of queue. Process N and change the status of
     N to the processed state (STATUS = 3).
v.   Add to the rear of queue all the neighbours of N that are in the ready
     state (STATUS=1) and change their status to the waiting state
     (STATUS = 2).
         [End of loop]
vi.  End.
```

2. **Depth First Search (DFS) :** The general idea behind a depth first search beginning at a starting node *A* is as follows :
   a. First, we examine the starting node *A*.
   b. Then, we examine each node *N* along a path *P* which begins at *A*; that is, we process neighbour of *A*, then a neighbour of neighbour of *A*, and so on.
   c. This algorithm uses a stack instead of queue.

Algorithm :

```
i.   Initialize all nodes to ready state (STATUS = 1).
ii.  Push the starting node A onto stack and change its status to the waiting
     state (STATUS = 2).
iii. Repeat steps (iv) and (v) until queue is empty.
iv.  Pop the top node N of stack, process N and change its status to the
     processed state (STATUS = 3).
v.   Push onto stack all the neighbours of N that are still in the ready state
     (STATUS = 1) and change their status to the waiting state (STATUS
     = 2).
     [End of loop]
vi.  End.
```

Que 4.6. **Write and explain DFS graph traversal algorithm.**

OR

Write DFS algorithm to traverse a graph. Apply same algorithm for the graph given in Fig. 4.6.1 by considering node 1 as startingnode.

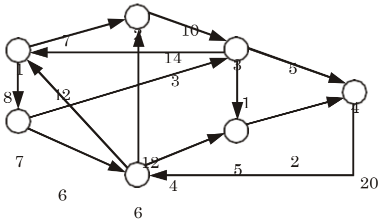


Fig. 4.6.1.

AKTU 2014-15, Marks 10

Answer

DFS : Refer Q. 4.5, Page 4–7A, Unit-4.

Numerical : Adjacency list of the given graph :

1 → 2, 7
2 → 3
3 → 5, 4, 1
4 → 6
5 → 4
6 → 2, 5, 1
7 → 3, 6

1. Initially set STATUS = 1 for all vertex
2. Push 1 onto stack and set their STATUS = 2

1

3. Pop 1 from stack, change its STATUS = 1 and
   Push 2, 7 onto stack and change their STATUS = 2; DFS = 1

7
2

4. Pop 7 from stack, Push 3, 6; DFS = 1, 7

6
3
2

5. Pop 6 from stack, Push 5; DFS = 1, 7, 6

5
3
2

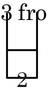
6. Pop 5 from stack, Push 4; DFS = 1, 7, 6, 5

4
3
2

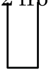
7. Pop 4 from stack; DFS = 1, 7, 6, 5, 4

3
2

8. Pop 3 from stack; DFS = 1, 7, 6, 5, 4, 3



9. Pop 2 from stack; DFS = 1, 7, 6, 5, 4, 3



Now, the stack is empty, so the depth first traversal of a given graph is 1, 7, 6, 5, 4, 3.

Que 4.7. **Implement BFS algorithm to find the shortest path fromnode *A* to *J*.**

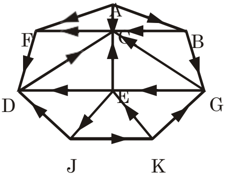


Fig. 4.7.1.

OR

Explain in detail about the graph traversal techniques withsuitable example.

AKTU 2018-19, Marks 07

Answer

Following are the two traversal techniques :

1. **Depth First Search (DFS) :** Refer Q. 4.5, Page 4–7A, Unit-4.
   Example : Refer Q. 4.6, Page 4–8A, Unit-4.
2. **Breadth First Search (BFS) :** Refer Q. 4.5, Page 4–7A, Unit-4. To find the shortest path from node *A* to node *J*

Adjacency list of the graph is :

A	:	F, C, B
B	:	G, C
C	:	F
D	:	C
E	:	D, C, J
F	:	D
G	:	C, E
J	:	D, K
K	:	E, G

a. Initially set STATUS=1 for all vertex.

b. Now add 'A' to Queue and set STATUS = 2
   Queue: A

c. Remove A from Queue and set STATUS = 3
   and add F, C, B in Queue and change their STATUS = 2
   BFS = A Queue: F, C, B

d. Remove F, add D in Queue
BFS = A, F Queue = C, B, D,

e. Remove C, add F, but F is already visited. So no vertex will be added in this step
BFS = A, F, Queue = B, D

f. Remove B, add G, BFS = A, F, C, B, Queue = D, G

g. Remove D, BFS = A, F, C, B, D, Queue = G

h. Remove G, add E, BFS = A, F, C, B, D, G, Queue = E

i. Remove E, add J, BFS = A, F, C, B, D, G, E, Queue = J

j. Remove J, BFS = A, F, C, B, D, G, E, J
J is our final destination. We now back track from J to find the path from J to A : J ← E ← G ← B ← A

Que 4.8. Illustrate the importance of various traversing techniques in graph along with its applications.

AKTU 2016-17, Marks 10

Answer

Various types of traversing techniques are :

1. Breadth First Search (BFS) 2. Depth First Search (DFS)

Importance of BFS :

1. It is one of the single source shortest path algorithms, so it is used to compute the shortest path.
2. It is also used to solve puzzles such as the Rubik's Cube.
3. BFS is not only the quickest way of solving the Rubik's Cube, but also the most optimal way of solving it.

Application of BFS : Breadth first search can be used to solve many problemsin graph theory, for example :

1. Copying garbage collection.
2. Finding the shortest path between two nodes *u* and *v*, with path length measured by number of edges (an advantage over depth first search).
3. Ford-Fulkerson method for computing the maximum flow in a flow network.
4. Serialization/Deserialization of a binary tree vs serialization in sorted order, allows the tree to be re-constructed in an efficient manner.
5. Construction of the failure function of the Aho-Corasick pattern matcher.
6. Testing bipartiteness of a graph.

Importance of DFS : DFS is very important algorithm as based upon DFS, there are $O(V + E)$-time algorithms for the following problems :

1. Testing whether graph is connected.
2. Computing a spanning forest of *G*.
3. Computing the connected components of *G*.
4. Computing a path between two vertices of *G* or reporting that no such path exists.
5. Computing a cycle in *G* or reporting that no such cycle exists.

Application of DFS : Algorithms that use depth first search as a building block include :

1. Finding connected components.
2. Topological sorting.
3. Finding 2-(edge or vertex)-connected components.
4. Finding 3-(edge or vertex)-connected components.
5. Finding the bridges of a graph.
6. Generating words in order to plot the limit set of a group.
7. Finding strongly connected components.

Que 4.9. **Define connected component and strongly connected component. Write an algorithm to find strongly connected components.**

Answer

Connected component : Connected component of an undirected graph is a sub-graph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the super graph.

Strongly connected component : A directed graph is strongly connected if there is a path between all pairs of vertices. A strong component is a maximal subset of strongly connected vertices of subgraph.

Kosaraju's algorithm is used to find strongly connected components in a graph.

Kosaraju's algorithm :

1. For each vertex *u* of the graph, mark *u* as unvisited. Let *L* be empty.
2. For each vertex *u* of the graph do Visit(*u*), where Visit(*u*) is the recursive subroutine. If *u* is unvisited then :
    a. Mark *u* as visited.
    b. For each out-neighbour *v* of *u*, do Visit(*v*).
    c. Prepend *u* to *L*. Otherwise do nothing.
3. For each element *u* of L in order, do Assign(*u*, *u*) where Assign (*u*, root) is the recursive subroutine. If *u* has not been assigned to a component then :
    a. Assign *u* as belonging to the component whose root is root.
    b. For each in-neighbour *v* of *u*, do Assign (*v*, root).

    Otherwise do nothing.

PART-4

Spanning Tree, Minimum Cost Spanning Trees : Prim's andKruskal's Algorithm.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.10. **What do you mean by spanning tree and minimum spanning tree ?**

Answer

Spanning tree :

1. A spanning tree of an undirected graph is a sub-graph that is a tree which contains all the vertices of graph.
2. A spanning tree of a connected graph *G* contains all the vertices and has the edges which connect all the vertices. So, the number of edges will be 1 less than the number of nodes.
3. If graph is not connected, *i.e.*, a graph with *n* vertices has edges less than $n - 1$ then no spanning tree is possible.
4. A connected graph may have more than one spanning trees.

Minimum spanning tree :

1. In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph.
2. There are number of techniques for creating a minimum spanning tree for a weighted graph but the most famous methods are Prim's and Kruskal's algorithm.

Que 4.11. **Write down Prim's algorithm to find out minimal spanning tree.**

Answer

First it chooses a vertex and then chooses an edge with smallest weight incident on that vertex. The algorithm involves following steps :

Step 1 : Choose any vertex V_1 of *G*.

Step 2 : Choose an edge $e_1 = V_1V_2$ of *G* such that $V_2 \neq V_1$ and e_1 has smallest weight among the edge *e* of *G* incident with V_1.

Step 3 : If edges $e_1, e_2, \ldots\ldots\ldots, e_i$ have been chosen involving end points $V_1, V_2, \ldots\ldots\ldots\ldots, V_{i+1}$, choose an edge $e_{i+1} = V_jV_k$ with $V_j = \{V_1 \ldots\ldots\ldots V_{i+1}\}$ and $V_k \notin \{V_1 \ldots\ldots\ldots V_{i+1}\}$ such that e_{i+1} has smallest weight among the edges of *G* with precisely one end in $\{V_1 \ldots\ldots\ldots V_{i+1}\}$.

Step 4 : Stop after $n - 1$ edges have been chosen. Otherwise goto step 3.

Que 4.12. **Define spanning tree. Find the minimal spanning treefor the following graph using Prim's algorithm.**

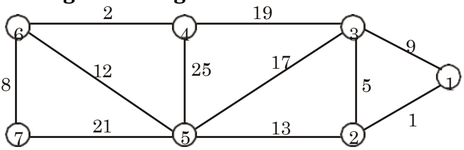


Fig. 4.12.1.

AKTU 2014-15, Marks 10

Answer

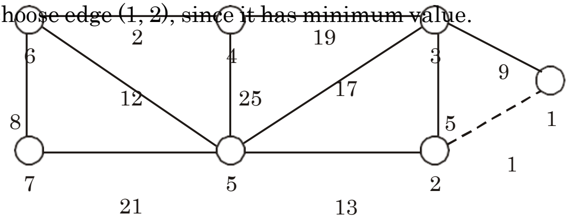
Spanning tree : Refer Q. 4.10, Page 4–13A, Unit-4.

Numerical :

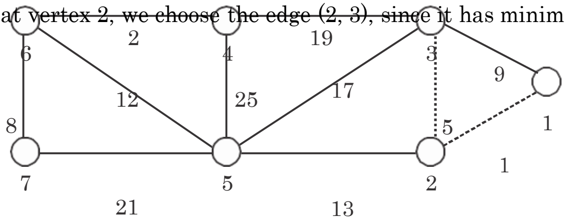
$$\begin{array}{c|ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
1 & - & 1 & 9 & - & - & - & - \\
2 & 1 & - & 5 & - & 13 & - & - \\
3 & 9 & 5 & - & 19 & 17 & - & - \\
4 & - & - & 19 & - & 25 & 2 & - \\
5 & - & 13 & 17 & 25 & - & 12 & 21 \\
6 & - & - & - & 2 & 12 & - & 8 \\
7 & - & - & - & - & 21 & 8 & -
\end{array}$$

According to Prim's algorithm, we choose vertex 1.

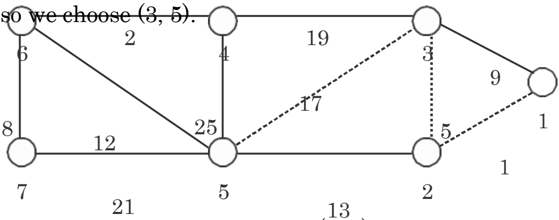
We choose edge (1, 2), since it has minimum value.



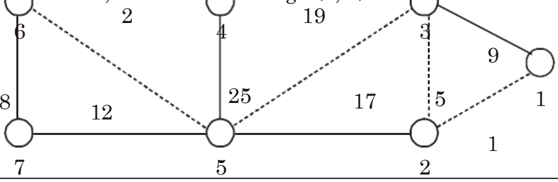
Now at vertex 2, we choose the edge (2, 3), since it has minimum value.



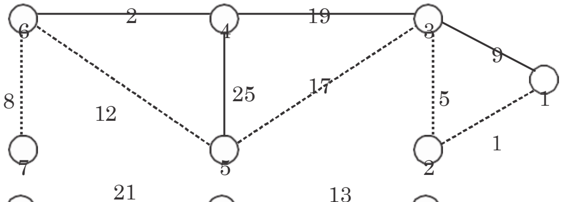
Now at vertex 3, we cannot choose edge (3, 1) because it will create a cycle so we choose (3, 5).

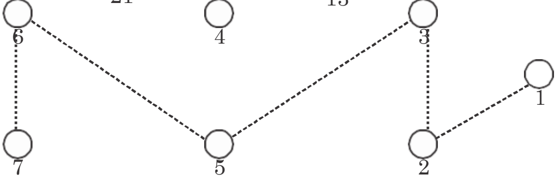


Now at vertex 5, we choose the edge (5, 6) since it has minimum value.



Now at vertex 6, we choose the edge (6, 7) since it has minimum value.





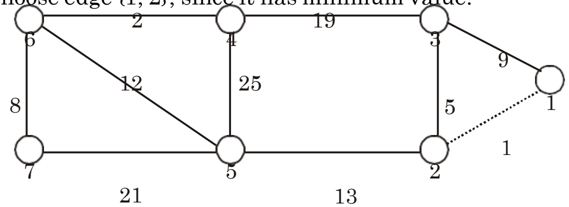
Since in spanning tree, the tree should cover all the vertices and should not make cycle.

But in the above tree, 4 is remaining so the above asked question is wrong. If we assume to remove the edge from {3, 5} then the spanning tree is :

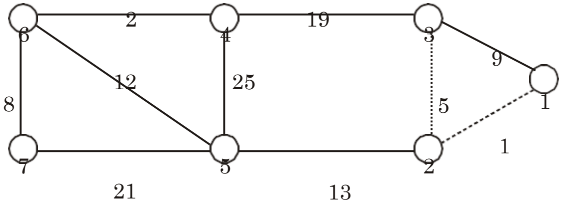
$$\begin{array}{c|ccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
1 & – & 1 & 9 & – & – & – & – \\
2 & 1 & – & 5 & – & 13 & – & – \\
3 & 9 & 5 & – & 19 & – & – & – \\
4 & – & – & 19 & – & 25 & 2 & – \\
5 & – & 13 & 17 & 25 & – & 12 & 21 \\
6 & – & – & – & 2 & 12 & – & 8 \\
7 & – & – & – & – & 21 & 8 & –
\end{array}$$

According to Prim's algorithm, let's choose vertex 1.

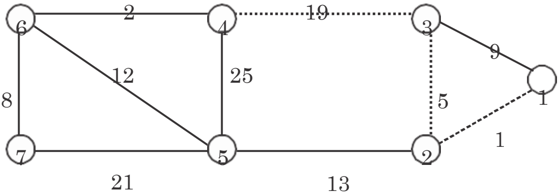
We choose edge {1, 2}, since it has minimum value.



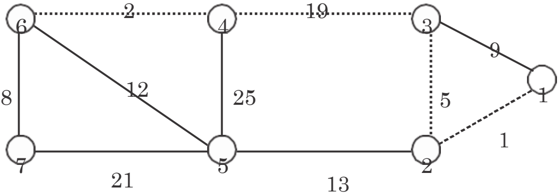
Now at vertex 2, we choose the edge (2, 3), since it has minimum value.



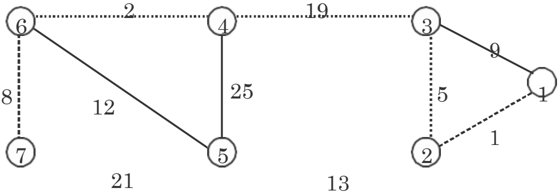
Now at vertex 3, we choose the edge (3, 4), since it has minimum value.



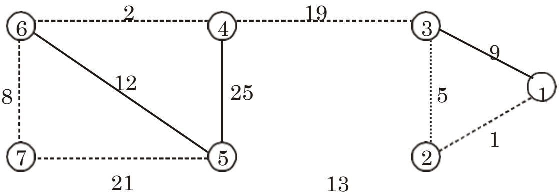
Now at vertex 4, we choose the edge (4, 6), since it has minimum value.



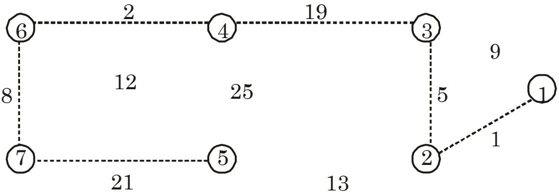
Now at vertex 6, we choose the edge (6, 7), since it has minimum value.



Now at vertex 7, we cannot choose the edge (7, 6), because we have already traversed this edge these we choose (7, 5).



∴ The spanning tree is



Que 4.13. Define spanning tree. Also construct minimum spanning tree using Prim's algorithm for the given graph.

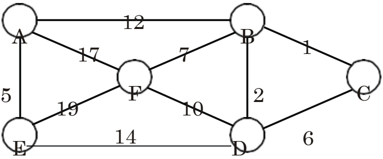


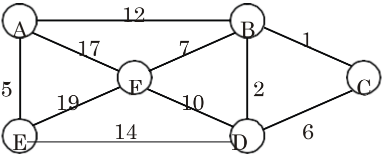
Fig. 4.13.1.

AKTU 2017-18, Marks 07

Answer

Spanning tree : Refer Q. 4.10, Page 4–13A, Unit-4.

Numerical :

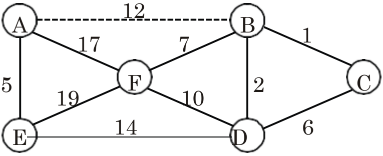


Let us take A as source node.

Now we look on weight

$w(A, B) = 12$, $w(A, F) = 17$, $w(A, E) = 15$

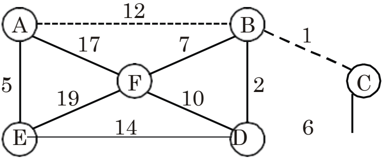
$\because$ $w(A, B)$ is smallest. Choose $e = (AB)$



Now we look on weight

$w(B, F) = 7$, $w(B, D) = 2$, $w(B, C) = 1$

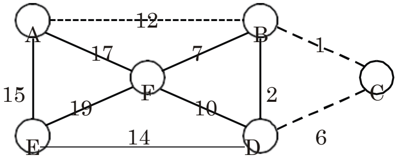
$\because$ $w(B, C)$ is smallest $\therefore$ choose $e = (BC)$



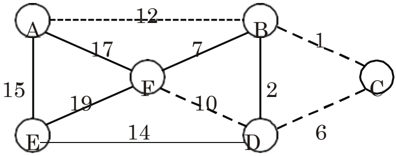
Now we look on weight

$w(C, D) = 6$

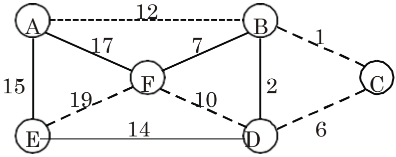
$\because$ $w(C, D)$ is smallest $\therefore$ choose $e = (CD)$



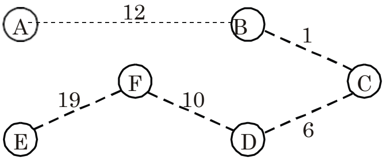
Now we look on weight
$w(D, B) = 2$, $w(D, F) = 10$, $w(D, E) = 14$
∵ $w(D, B)$ is smallest but forms a cycle
∴ Discard it.
Now $w(D, F) = 10$ is smallest ∴ Choose $e = (DF)$



Now we look on weight
$w(F, B) = 7$, $w(F, A) = 17$, $w(F, E) = 19$
∵ $w(F, B)$ is smallest but forms cycle
∴ Discard it
∵ $w(F, A)$ is smallest but forms cycle
∴ Discard it
∴ choose $e = (FE)$



The final minimum spanning tree is :



Que 4.14. Write Kruskal's algorithm to find minimum spanning tree.

Answer

i. In this algorithm, we choose an edge of *G* which has smallest weight among the edges of *G* which are not loops.

ii. This algorithm gives an acyclic subgraph *T* of *G* and the theorem given below proves that *T* is minimal spanning tree of *G*. Following steps are required :

Step 1 : Choose e_1, an edge of G, such that weight of e_1, $w(e_1)$ is as smallas possible and e_1 is not a loop.

Step 2 : If edges e_1, e_2,............, e_i have been selected then choose an edge e_{i+1} not already chosen such that

i. the induced subgraph, $G[\{e_1 .... e_{i+1}\}]$ is acyclic and

ii. $w(e_{i+1})$ is as small as possible

Step 3 : If G has n vertices, stop after $n - 1$ edges have been chosen. Otherwise repeat step 2.

If G be a weighted connected graph in which the weight of the edges are all non-negative numbers, let T be a sub-graph of G obtained by Kruskal's algorithm then, T is minimal spanning tree.

Que 4.15. Consider the following undirected graph.

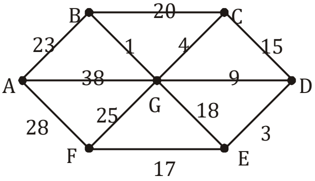


Fig. 4.15.1.

a. Find the adjacency list representation of the graph.

b. Find a minimum cost spanning tree by Kruskal's algorithm.

AKTU 2015-16, Marks 10

Answer

a.

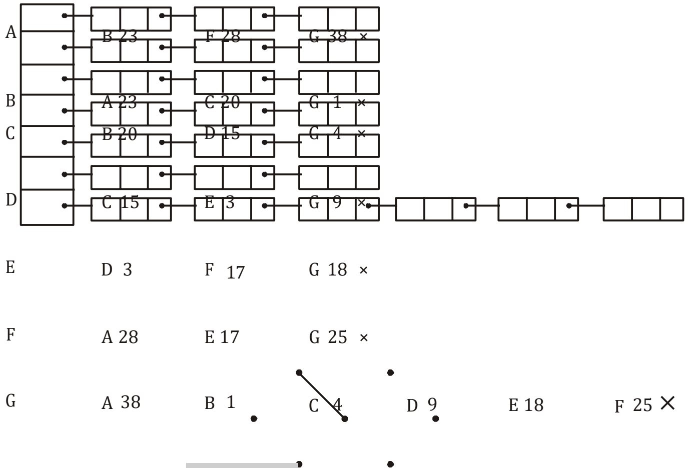
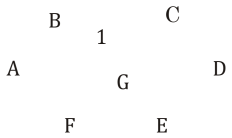


Fig. 4.15.2.

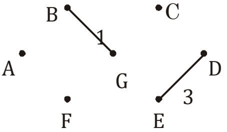
b.

Kruskal's algorithm :

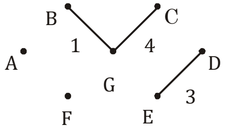
i. We will choose $e = BG$ as it has minimum weight.



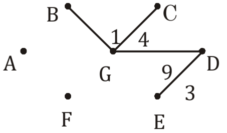
ii. Now choose $e = ED$.



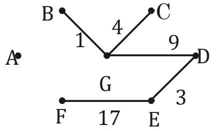
iii. Choose $e = CG$, since it has minimum weights.



iv. Choose $e = GD$.



v. Choose $e = EF$ and discard BC, CD and GE because they form cycle.



vi. Now choose $e = AB$ and discard AG, FG and AF because they form cycle. Final minimum spanning tree is given as :

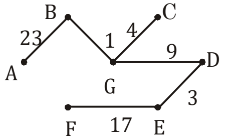


Fig. 4.15.3.

Que 4.16. What is spanning tree ? Describe Kruskal's and Prim's algorithm to find the minimum cost spanning tree and explain the complexity. Determine the minimum cost spanning tree for the graph given below :

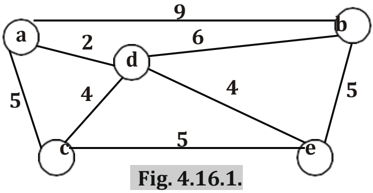


Fig. 4.16.1.

Answer

Spanning tree : Refer Q. 4.10, Page 4–13A, Unit-4. **Kruskal's algorithm :** Refer Q. 4.14, Page 4–18A, Unit-4.**Prim's algorithm :** Refer Q. 4.11, Page 4–13A, Unit-4.

Complexity :

A. Time complexity of Prim's algorithm :

1. The time complexity of Prim's algorithm depends on the data structures used for the graph and for ordering the edges by weight.
2. A simple implementation of Prim's, using an adjacency matrix or an adjacency list graph representation and linearly searching an array of weights to find the minimum weight edge to add, requires $O(|V|2)$ running time.

B. Time complexity of Kruskal's algorithm :

1. Kruskal's algorithm can be shown to run in $O(E \log E)$ time, or equivalently, $O(E \log V)$ time, where E is the number of edges in the graph and V is the number of vertices, all with simple data structures.
2. These running times are equivalent because :
   a. E is at most V^2 and $\log V^2 = 2 \log V$ is $O(\log V)$.
   b. Each isolated vertex is a separate component of the minimum spanning forest. If we ignore isolated vertices we obtain $V \leq 2E$, so $\log V$ is $O(\log E)$.

Numerical :

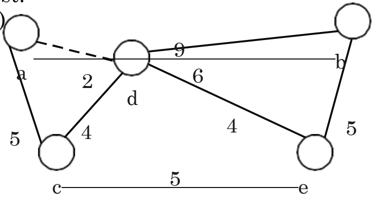
Let us take '*a*' as a source node.

Now look on weight

$$w(a, d) = 2, w(a, b) = 9$$
$$w(a, c) = 5$$

∵ $w(a, d)$ is smallest.

∴ Choose $e = (a, d)$

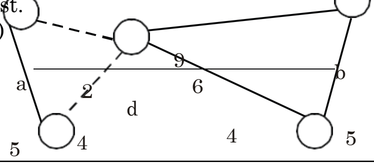


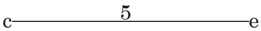
Now look on weight

$$w(d, b) = 6, w(d, c) = 4, w(d, e) = 4$$

∵ $w(d, c)$ is smallest.

∴ Choose $e = (d, c)$



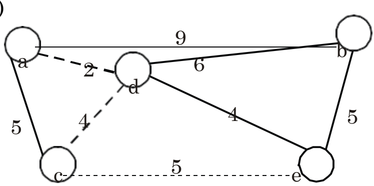

c
5
e

Now look on weight : $w(c, e) = 5$, $w(c, a) = 5$

$\because$ $w(c, a)$ is smallest but forms a cycle. So discard it.

Now $w(c, e)$ is smallest.

$\therefore$ Choose $e = (c, e)$

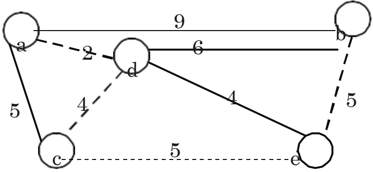


Now look on weight : $w(e, b) = 5$, $w(e, d) = 4$

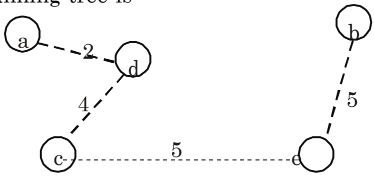
$\because$ $w(e, d)$ is smallest but forms a cycle. So discard it.

Now $w(e, b)$ is smallest.

$\therefore$ Choose $e = (e, b)$



Final minimum spanning tree is :



The minimal spanning tree is *adceb*.

Cost of minimal spanning tree is = 2 + 4 + 5 + 5 = 16.

Que 4.17. Find the minimum spanning tree for following graphusing Prim's and Kruskal's algorithms.

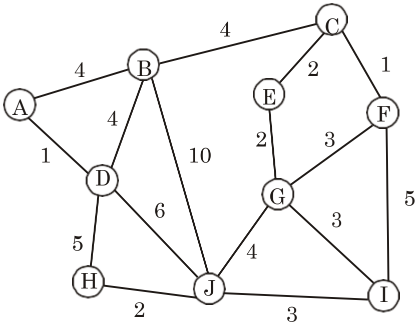


Fig. 4.17.1.

Answer

Prim's algorithm :

1. According to algorithm we choose vertex A from the set $\{A, B, C, D, E, F, G, H, I, J\}$.

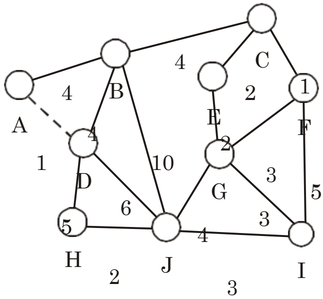


Fig. 4.17.2.

2. Now edge with smallest weight incident on A is $e = (AD)$

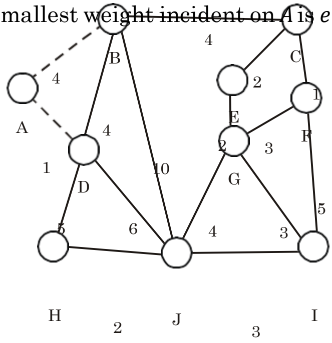


Fig. 4.17.3.

Now we look on weight

$$w(A, B) = 4, w(D, B) = 4$$
$$w(D, H) = 5, w(D, J) = 6$$

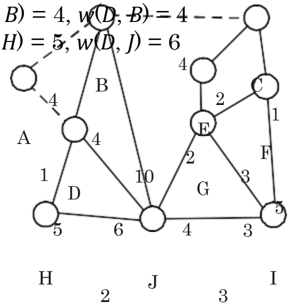


Fig. 4.17.4.

We choose $e = AB$ since it is minimum.

$w(D, B)$ can also be chosen because it has same value.

Again, $w(B, C) = 4, w(B, J) = 10, w(D, H) = 5, w(D, J) = 6$

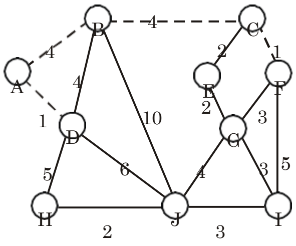


Fig. 4.17.5.

We choose $e = BC$ since it has minimum value.

Now, $w(B, J) = 10, w(C, E) = 2, w(C, F) = 1$

We choose $e = CF$ because $w(C, F)$ has minimum value.

Now, $w(C, E) = 2, w(F, G) = 3, w(F, I) = 5$

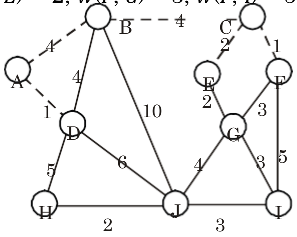


Fig. 4.17.6.

We choose $e = CE$, since $w(C, E)$ has minimum value.

$w(E, G) = 2, w(F, G) = 3, w(F, I) = 5$

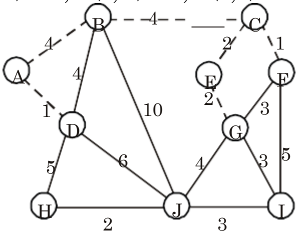


Fig. 4.17.7.

We choose $e = EG$, since $w(E, G)$ has minimum value.

$w(G, J) = 4, w(G, I) = 3, w(F, I) = 5$

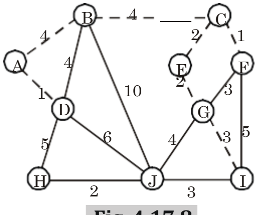


Fig. 4.17.8.

We choose $e = GI$, since $w(G, I)$has minimum value.

$$w(I, J) = 3, w(G, J) = 4$$

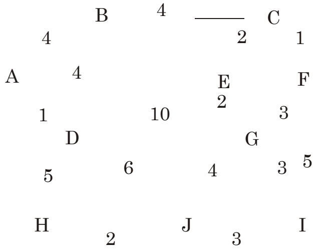


Fig. 4.17.9.

We choose $e = IJ$, since $w(I, J)$ has minimum value, $w(J, H) = 2$ Hence, $e = JH$ will be chosen. The final minimal spanning tree is :

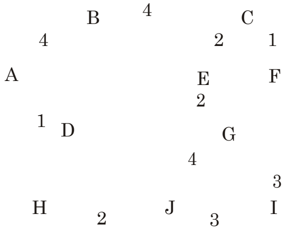


Fig. 4.17.10.

Kruskal's algorithm :

i. We will choose $e = AD$ and CF as it has minimum weight.

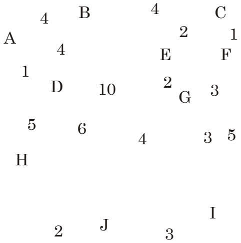


Fig. 4.17.11.

ii. Now choose $e = CF$.

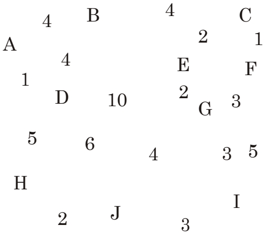
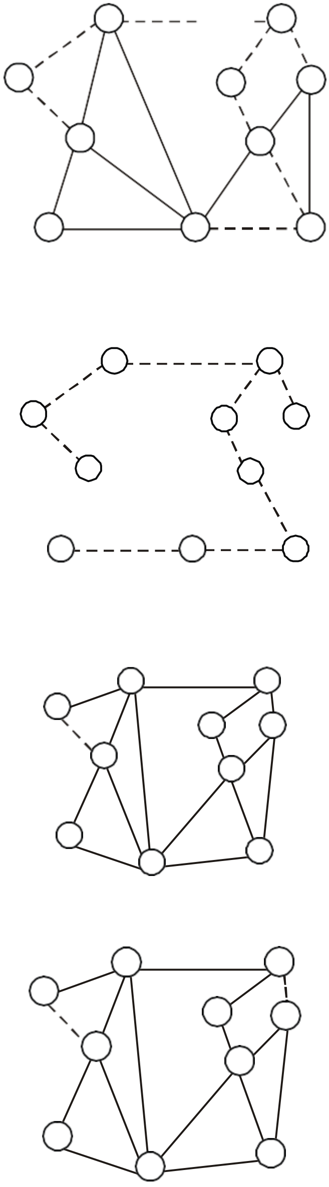


Fig. 4.17.12.



iii. Choose *CE*, *EG* and *HJ* since they have same and minimum weights.

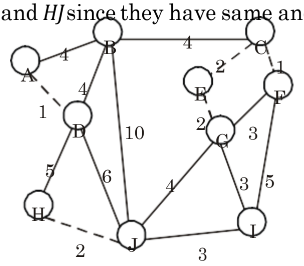


Fig. 4.17.13.

iv. Choose *IJ* and *GI* as it has minimum weight and discard *GF* becauseit forms cycle.

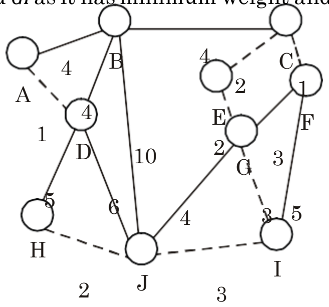


Fig. 4.17.14.

v. Choose *AB* and *BC* and discard *BD*, *GJ*, *DH*, *DJ*, *BJ*, *FI* becausethey form cycle.

We get the final minimal spanning tree as

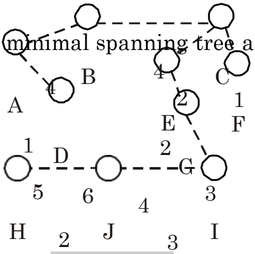


Fig. 4.17.15.

Que 4.18. Discuss Prim's and Kruskal's algorithm. Construct minimum spanning tree for the below given graph using Prim's algorithm (Source node = *a*). **AKTU 2016-17, Marks 15**

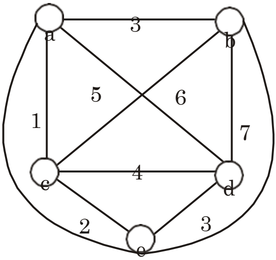


Fig. 4.18.1.

Answer

Prim's algorithm : Refer Q. 4.11, Page 4–13A, Unit-4. **Kruskal's algorithm :** Refer Q. 4.14, Page 4–18A, Unit-4. **Numerical :**

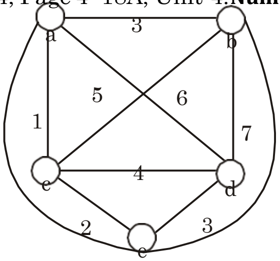


Fig. 4.18.2.

Start with source node = *a*

Now, edge with smallest weight incident on *a* is $e = (a, c)$.

So, we choose $e = (a, c)$.

Now we look on weights :

$$w(c, d) = 4, w(c, e) = 2, w(c, b) = 5$$

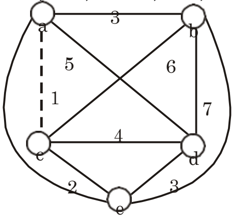


Fig. 4.18.3.

Since minimum is $w(c, e) = 2$. We choose $e = (c, e)$

Again, $w(e, d) = 3$

$w(e, a) = 8$

$w(e, b) = 7$

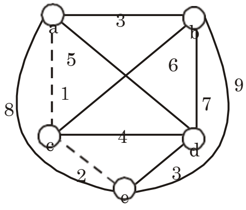


Fig. 4.18.4.

Since minimum is $w(e, d) = 3$, we choose $e = (e, d)$

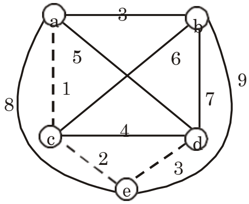


Fig. 4.18.5.

Now, $w(d, b) = 7$, and $w(a, b) = 3$
Since minimum is $w(a, b) = 3$, we choose $e = (a, b)$

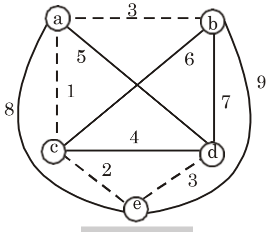


Fig. 4.18.6.

Therefore, the minimum spanning tree is :

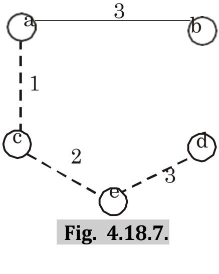


Fig. 4.18.7.

PART-5

Transitive Closure and Shortest Path Algorithm : WarshallAlgorithm and Dijkstra Algorithm.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.19. Explain transitive closure.

Answer

1. The transitive closure of a graph G is defined to be the graph G' such that G' has the same nodes as G and there is an edge (v_i, v_j) in G' whenever there is a path from v_i to v_j in G.
2. Accordingly the path matrix P of the graph G is precisely the adjacency matrix of its transitive closure G'.
3. The transitive closure of a graph G is defined as G^* or $G' = (V, E^*)$.
   where, $E^* = \{(i, j)$ there is a path from vertex i to vertex j in $G\}$
4. We construct the transitive closure $G^* = (V, E^*)$ by putting edge (i, j) into E^* if and only if $t_{ij}(n) = 1$.
5. The recursive definition of $t_{ij}^{(k)}$ is

$$t_{ij}^{(0)} = \begin{cases} 0 \text{ if } i \neq j \text{ and } (i, j) \notin E \\ 1 \text{ if } i = j \text{ or } (i, j) \in E \end{cases}$$

and for $k \geq 1$

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$$

Que 4.20. Write down Warshall's algorithm for finding all pair shortest path.

Answer

1. Floyd Warshall algorithm is a graph analysis algorithm for finding shortest paths in a weighted, directed graph.
2. A single execution of the algorithm will find the shortest path between all pairs of vertices.
3. It does so in $\Theta(V^3)$ time, where V is the number of vertices in the graph.
4. Negative-weight edges may be present, but we shall assume that there

are no negative-weight cycles.

5. The algorithm considers the "intermediate" vertices of a shortest path, where an intermediate vertex of a simple path $p = (v_1, v_2, ..., v_m)$ is any vertex of p other than v_1 or v_m, that is, any vertex in the set $\{v_2, v_3, ..., v_{m-1}\}$.
6. Let the vertices of G be $V = \{1, 2, ..., n\}$, and consider a subset $\{1, 2, ..., k\}$ of vertices for some k.
7. For any pair of vertices $i, j \in V$, consider all paths from i to j whose intermediate vertices are all drawn from $\{1, 2, ..., k\}$, and let p be a minimum-weight path from among them.
8. Let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to vertex j with all intermediate vertices in the set $\{1, 2, ..., k\}$.

   A recursive definition is given by

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1 \end{cases}$$

Floyd Warshall (w) :

```
$n \leftarrow$ rows $[w]$
$D^{(0)} \leftarrow w$
for $k \leftarrow 1$ to $n$
    do for $i \leftarrow 1$ to $n$
        do for $j \leftarrow 1$ to $n$

do $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
return $D^{(n)}$
```

Que 4.21. Write the Floyd Warshall algorithm to compute the all pair shortest path. Apply the algorithm on following graph :

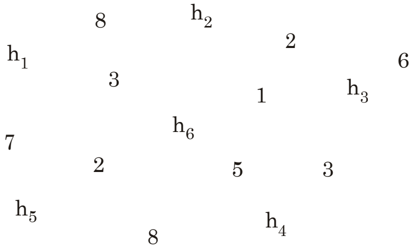
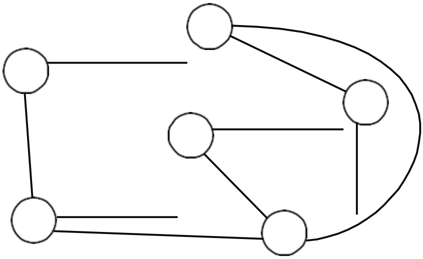


Fig. 4.21.2.

AKTU 2018-19, Marks 07

Answer

Floyd's Warshall algorithm : Refer Q. 4.20, Page 4–29A, Unit-4. **Numerical :** We cannot solve this using Floyd Warshall algorithm because the given graph is undirected.



Que 4.22. **Write and explain Dijkstra's algorithm for finding shortest path.**

OR

Write and explain an algorithm for finding shortest path betweenany two nodes of a given graph.

Answer

a. Dijkstra's algorithm, is a greedy algorithm that solves the single-source shortest path problem for a directed graph $G = (V, E)$ with non-negative edge weights, *i.e.*, we assume that $w(u, v) \geq 0$ each edge $(u, v) \in E$.

b. Dijkstra's algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined.

c. That is, for all vertices $v \in S$, we have $d[v] = \delta(s, v)$.

d. The algorithm repeatedly selects the vertex $u \in V - S$ with the minimum shortest-path estimate, inserts u into S, and relaxes all edges leaving u.

e. We maintain a priority queue Q that contains all the vertices in $v - s$, keyed by their d values.

f. Graph G is represented by adjacency list.

g. Dijkstra's always chooses the "lightest or "closest" vertex in $V - S$ to insert into set S, that it uses as a greedy strategy.

```
DIJKSTRA ($G, w, s$)
    INITIALIZE-SINGLE-SOURCE ($G, s$)
    $S \leftarrow \phi$
    $Q \leftarrow V[G]$
    while $Q \neq \phi$
    do $u \leftarrow$ EXTRACT-MIN ($Q$)
    $S \leftarrow S \cup \{u\}$
    for each vertex $v \in$ Adj [$u$]
    do RELAX ($u, v, w$)
```

Que 4.23. **Find out the shortest path from node 1 to node 4 in agiven graph (Fig. 4.23.1) using Dijkstra shortest path algorithm.**

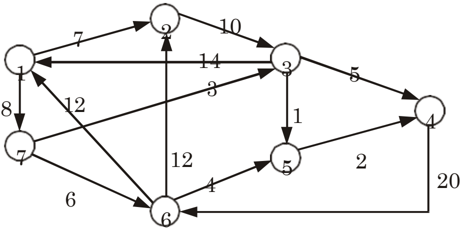


Fig. 4.23.1.

AKTU 2014-15, Marks 10

Answer

$\boxed{1}$	2	3	4	5	6	7
$\boxed{0}$	$\boxed{\infty}$	$\boxed{\infty}$	∞	∞	∞	∞
$\boxed{0}$	$\boxed{7}$	$\boxed{17}$	∞	∞	∞	$\boxed{8}$
$\boxed{0}$	$\boxed{7}$	$\boxed{17}$	∞	∞	∞	$\boxed{8}$
$\boxed{0}$	$\boxed{7}$	$\boxed{11}$	∞	$\boxed{\infty}$	14	$\boxed{8}$
$\boxed{0}$	$\boxed{7}$	$\boxed{11}$	16	$\boxed{12}$	14	$\boxed{8}$
$\boxed{0}$	$\boxed{7}$	$\boxed{11}$	$\boxed{14}$	$\boxed{12}$	$\boxed{14}$	$\boxed{8}$
0	7	11	14	12	14	8

Shortest path from node 1 to node 4 = 0 + 7 + 11 + 14 = 32

Que 4.24. **Describe Dijkstra's algorithm for finding shortest path.Describe its working for the graph given below.**

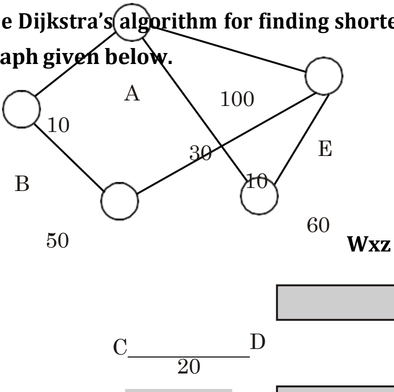


Fig. 4.24.1.

AKTU 2017-18, Marks 07

OR

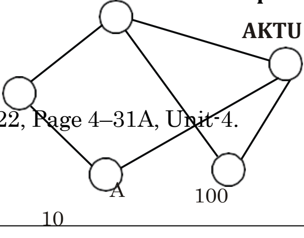
Explain Dijkstra's algorithm with suitable example.

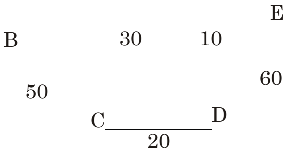
AKTU 2015-16, Marks 10

Answer

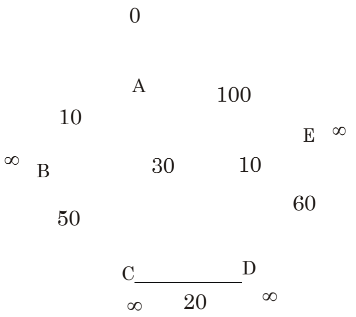
Algorithm : Refer Q. 4.22, Page 4–31A, Unit-4.

Numerical :



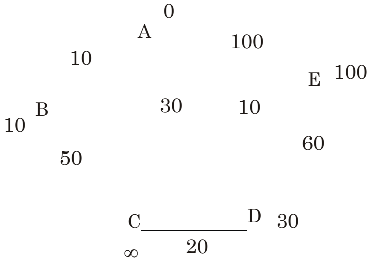

E
B
30
10
60
50
C
D
20

Extract min (*A*) :



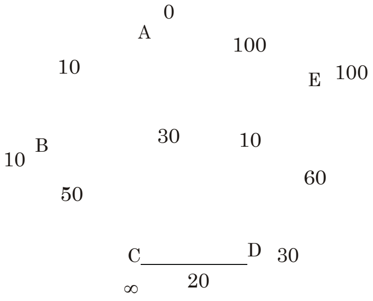
A	B	C	D	E
0	∞	∞	∞	∞

All edges leaving *A* :



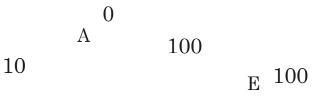
A	B	C	D	E
0	∞	∞	∞	∞
	10	∞	30	100

Extract min (*B*) :



A	B	C	D	E
0	∞	∞	∞	∞
	10	∞	30	100

All edges leaving *B* :



A	B	C	D	E

B 30 10 0 ∞ ∞ ∞ ∞

10

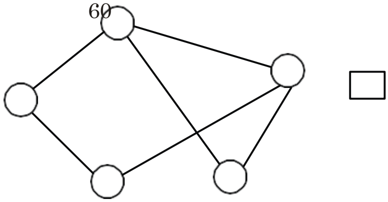
60 10 ∞ 30 100

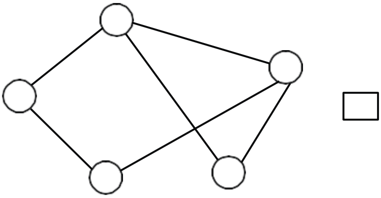
50

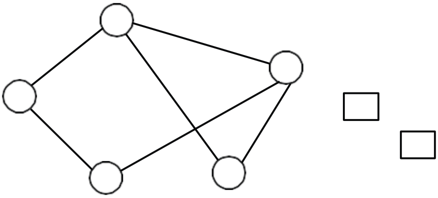
C D 30 60 30 100

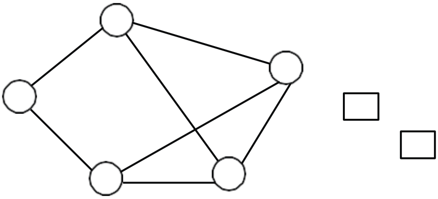
20

60

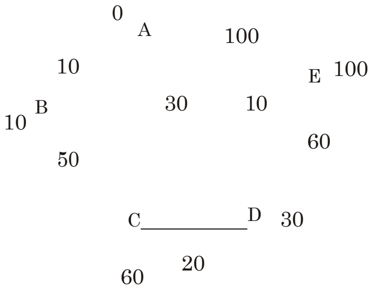






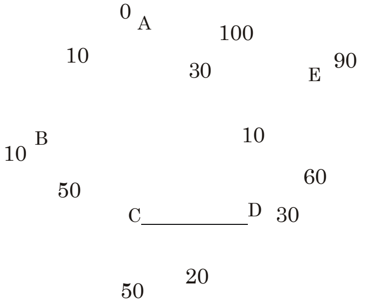


Extract min(*D*) :



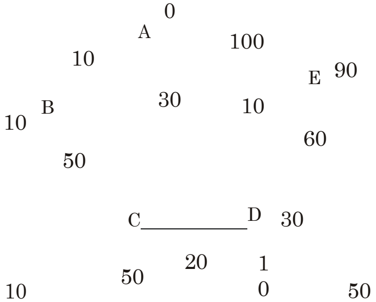
A	B	C	D	E
0	∞	∞	∞	∞
	10	∞	30	100
		60	30	100

All edges leaving (*D*) :



A	B	C	D	E
0	∞	∞	∞	∞
	10	∞	30	100
		60	30	100
		50		90

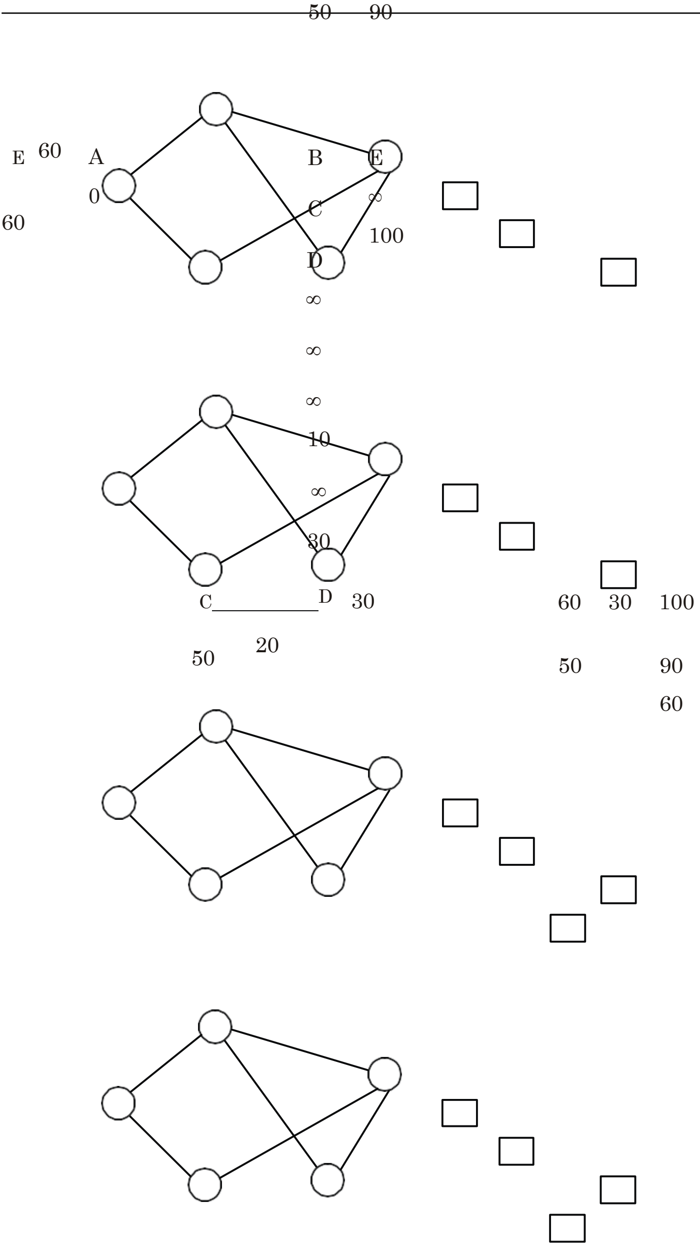
Extract min(*C*) :



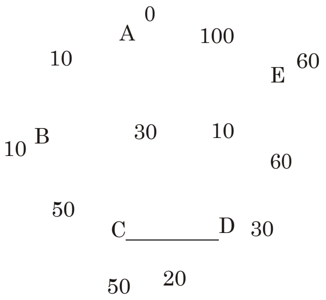
A	B	C	D	E
0	∞	∞	∞	∞
	10	∞	30	100
		60	30	100

10

All edges leaving *C* :

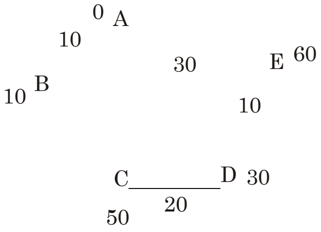

50
90
E
60
A
B
E
0
∞
60
C
100
D
∞
∞
∞
10
∞
30
C
D
30
60
30
100
20
50
50
90
60

Extract min(*E*) :



A	B	C	D	E
0	∞	∞	∞	∞
	10	∞	30	100
		60	30	100
		50		90
				60

Shortest path :



Que 4.25. **By considering vertex ‘1’ as source vertex, find the shortest paths to all other vertices in the following graph usingDijkstra’s algorithms. Show all the steps.**

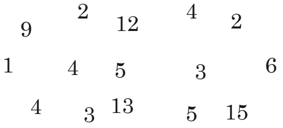


Fig. 4.25.1.

AKTU 2018-19, Marks 07

Answer

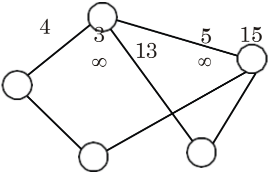
Initialize :

2 12 4 2

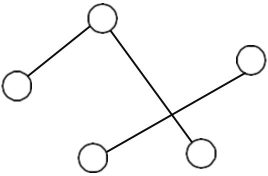
9

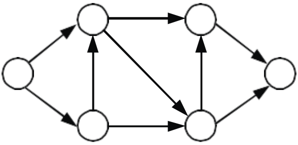
S = { }

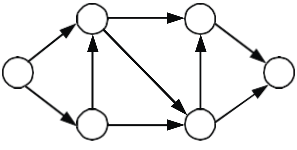
0 1 4 5 3 6 ∞



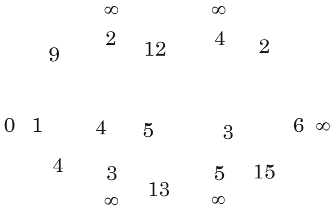
Q：	1	2	3	4	5	6
	0	∞	∞	∞	∞	∞







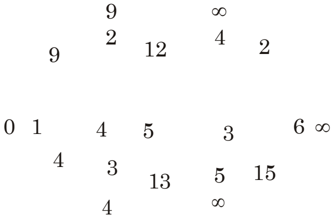
EXTRACT – MIN (1) :



S = { 1 }

Q :	1	2	3	4	5	6
	0	∞	∞	∞	∞	∞

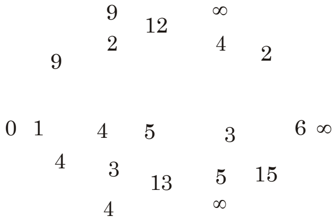
Relax all edges leaving 1 :



S = { 1 }

Q :	1	2	3	4	5	6
	0	∞	∞	∞	∞	∞
		9	4	–	–	–

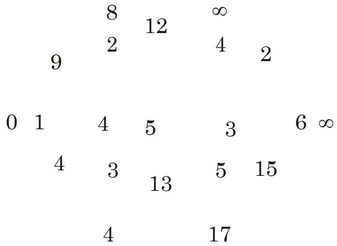
EXTRACT – MIN (3) :



S = { 1, 3 }

Q :	1	2	3	4	5	6
	0	∞	∞	∞	∞	∞
		9	4	–	–	–

Relax all edges leaving 3 :



S = { 1, 3 }

Q :	1	2	3	4	5	6
	0	∞	∞	∞	∞	∞
		9	4	–	–	–
		8		–	17	–

EXTRACT – MIN (2) :

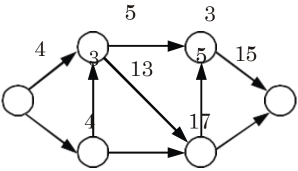
∞

4 2

{ 1, 3, 2 }

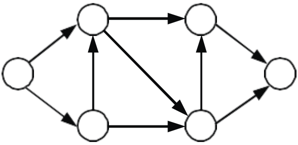
S =

6 ∞

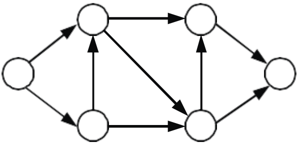


Q : 1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	9	4	–	–	–
	8		–	17	–

□

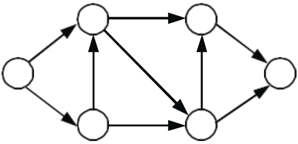


□



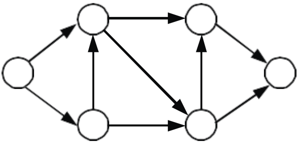
□

□



□

□

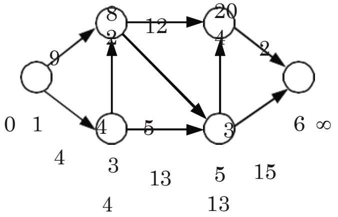


□

□

□

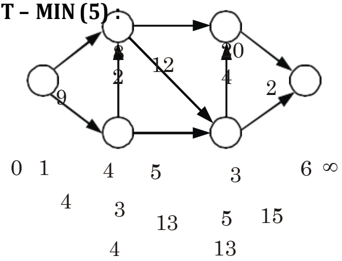
Relax all edges leaving 2 :



S = { 1, 3, 2 }

Q :	1	2	3	4	5	6
	0	∞	∞	∞	∞	∞
		9	4	–	–	–
		8		–	17	–
				20	13	–

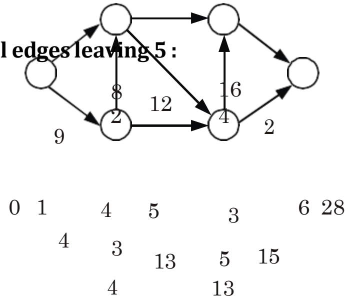
EXTRACT – MIN (5) :



S = { 1, 3, 2, 5 }

Q :	1	2	3	4	5	6
	0	∞	∞	∞	∞	∞
		9	4	–	–	–
		8		–	17	–
				20	13	–

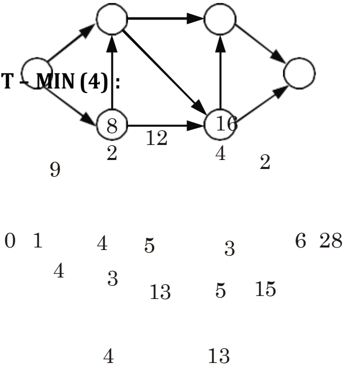
Relax all edges leaving 5 :



S = { 1, 3, 2, 5 }

Q :	1	2	3	4	5	6
	0	∞	∞	∞	∞	∞
		9	4	–	–	–
		8		–	17	–
				20	13	–
				16		28

EXTRACT – MIN (4) :

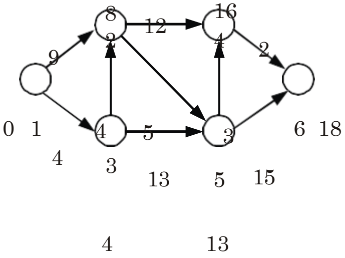


S = { 1, 3, 2, 5, 4 }

Q :	1	2	3	4	5	6
	0	∞	∞	∞	∞	∞
		9	4	–	–	–
		8		–	17	–
				20	13	–

16 28

Relax all edges leaving 4 :

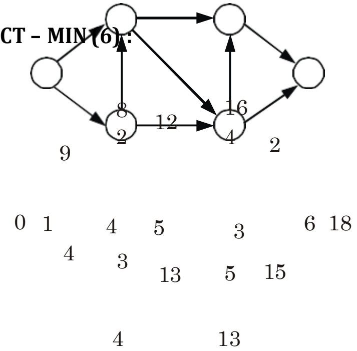


S = { 1, 3, 2, 5, 4 }

Q :

1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	9	4	–	–	–
	8		–	17	–
			20	13	–
			16		28
					18

EXTRACT – MIN (6) :



S = { 1, 3, 2, 5, 4, 6 }

Q :

1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	9	4	–	–	–
	8		–	17	–
			20	13	–
			16		28
					18

VERY IMPORTANT QUESTIONS

Following questions are very important. These questionsmay be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. Write DFS algorithm to traverse a graph. Apply same algorithm for the graph given in Fig. 1 by considering node 1 as starting node.

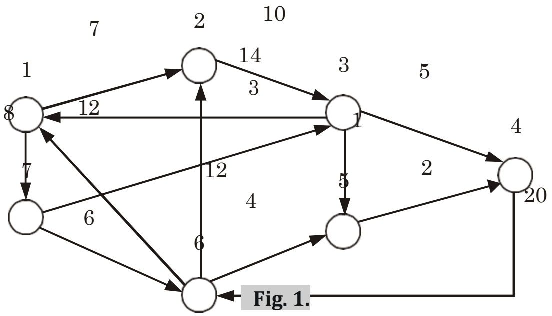


Fig. 1.

Ans. Refer Q. 4.6.

Q. 2. Illustrate the importance of various traversing techniquesin graph along with its applications.

Ans. Refer Q. 4.8.

Q. 3. Define spanning tree. Also construct minimum spanningtree using Prim's algorithm for the given graph.

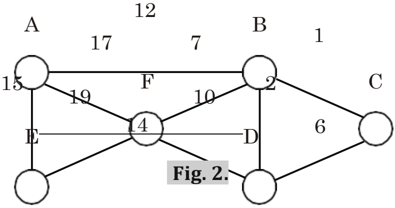


Fig. 2.

Ans. Refer Q. 4.13.

Q. 4. Consider the following undirected graph.

a. Find the adjacency list representation of the graph.

b. Find a minimum cost spanning tree by Kruskal's algorithm.

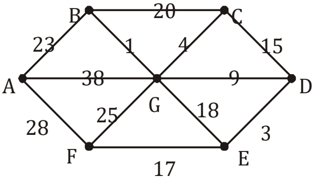


Fig. 3.

Ans. Refer Q. 4.15.

Q. 5. Discuss Prim's and Kruskal's algorithm. Construct minimum spanning tree for the below given graph using Prim's algorithm (Source node = *a*).

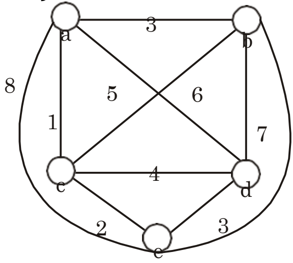


Fig. 4.

Ans. Refer Q. 4.18.

Q. 6. Write the Floyd Warshall algorithm to compute the all pairshortest path. Apply the algorithm on following graph :

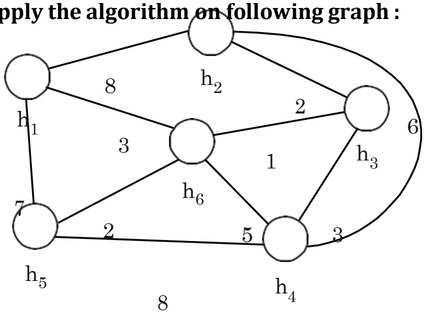


Fig. 5.

Ans. Refer Q. 4.21.

Q. 7. Describe Dijkstra's algorithm for finding shortest path.Describe its working for the graph given below.

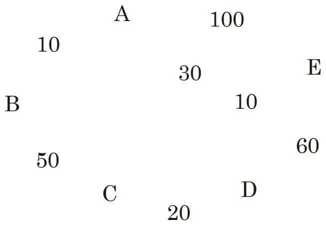


Fig. 6.

Ans. Refer Q. 4.24.

Q. 8. Find out the shortest path from node 1 to node 4 in a givengraph (Fig. 7) using Dijkstra shortest path algorithm.

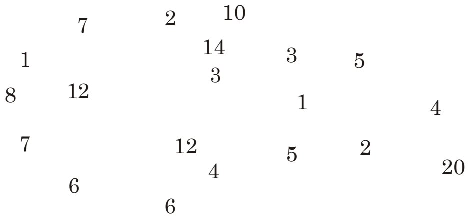
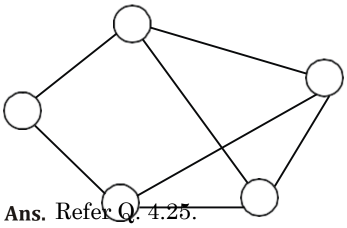


Fig. 7.

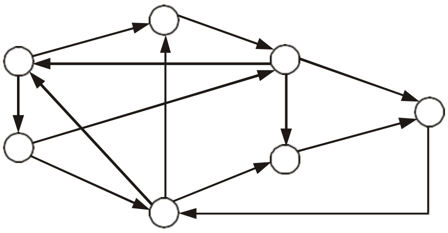
Ans. Refer Q. 4.23.

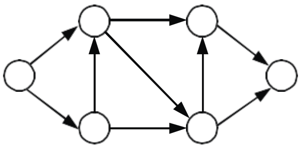
Q. 9. By considering vertex '1' as source vertex, find the shortest paths to all other vertices in the following graph using Dijkstra's algorithms. Show all the steps.



Ans. Refer Q. 4.25.

☺☺☺





1

4