Unit-VI: GRAPH

Graph:

- A graph is a non-linear data structure.
- A graph is denoted by G.
- A graph is consist of two things:
  1. Set of nodes.
  2. Set of edges. An edge is a line drawn in between two nodes.
     The set of Edges is denoted by E and a single edge is denoted by e.
- A graph can be represented mathematically as G = (V,E)
- Map is an example of Graph

Undirected Graph:

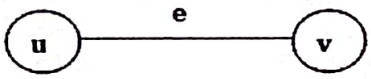
- An undirected graph is a graph in which the edges do not point in any direction.
- In undirectional graph the edges are bidirectional.

Nodes | Points | Vertices:

- The nodes in the graph are also called as Points or Vertices.
- The group of nodes is called as vertices and denoted by V.
- A single node is called as vertex or point.

End Points | Adjacent Nodes | Neighbors:

- Let u and v are two nodes connected to each other by using an edge e then u and v are called as adjacent nodes or neighbors.



- It can be represented as e = [u, v]
- Here u and v are called as endpoints of edge e.

Degree of node:

- It is the number of edges containing the node (for undirected graph).

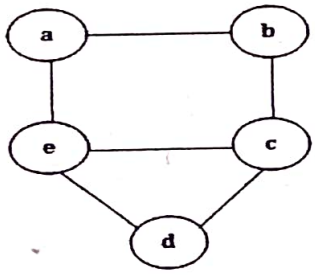
Isolated node:

- If the node does not belong to any edge then the node is called as isolated node. In other words if deg(node)= 0 then the node is called as isolated node.

Path:

- The sequence of consecutive edges is called as path
- A path from a node u and node v is denoted by P.
- It can be represented mathematically as:
  $P = (v1, v2, v3, \ldots, v_n)$

Example:



A path between a and d can be:
P1 = (a, b, c, d)
P2 = (a, e, d)
P3 = (a, e, c, d)

Closed Path:

- If the start node and end node is same then such path is called as closed path.

Simple Path:

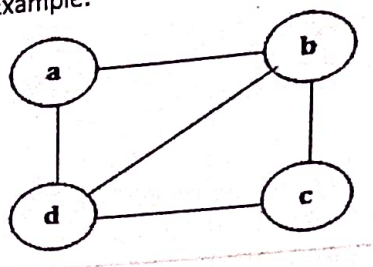
- If all the nodes in the path are distinct or different then such path is called as simple path.

Reference: Data Structures With C By Seymour Lipschutz
For Educational Purpose Only. Not For Sale.

Cycle:

- A cycle is closed path with minimum length 3.
- A cycle of length k is called as k-cycle.

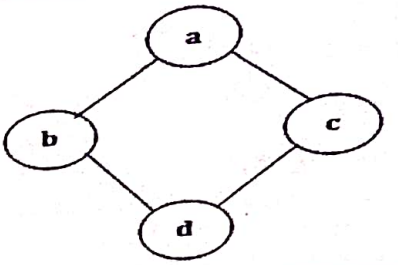
Example:



a->b->d->a is a cycle of length 3.
b->d->c->b is a cycle of length 3.
a->b->c->d->a is a cycle of length 4.

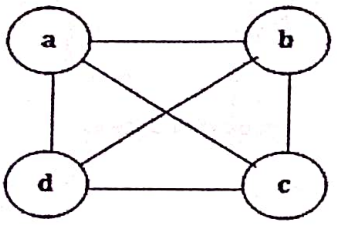
Connected graph:

- A graph is said to be connected if there is always a simple path exist between any two nodes.



Complete Graph:

- A graph is said to be complete if every node is adjacent to any other node.

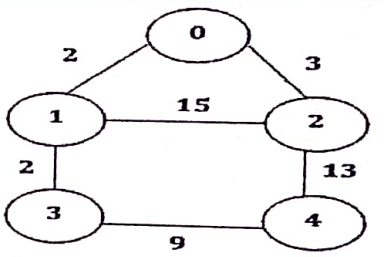


Adjacency matrix:
A =

	a	b	c	d
a	0	1	1	1
b	1	0	1	1
c	1	1	0	1
d	1	1	1	0

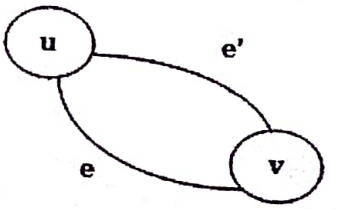
Weighted Graph:

- A graph is said to be weighted if each edge in the graph is assigned a non-negative numeric value.
- This value is called as weight or length of that edge.



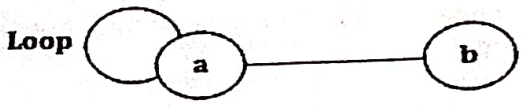
Multiple edges:

- Distinct edges e and e' are called as multiple edges if they connect the same end points. i.e. if e = [u, v] and e' = [u, v]



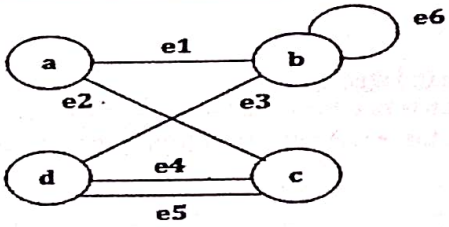
Loop or Self edge:

- An edge is called as a loop if it has identical start and end points.



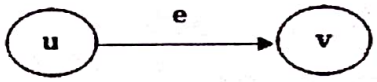
Multigraph:

- If a graph contains multiple edges and/or loops then such graph is called as multigraph.



Directed Graph:

- It is same as multigraph except that each edge e in graph is assigned a direction.
- In directed graph the edge is also called as arc.
- For the directed graph the following terminology is used:
  1) e begins at u and ends at v
  2) u is the origin or initial point of e and v is the destination or terminal point of e.
  3) u is predecessor of v and v is successor or neighbor of u.
  4) Adjacent of u is v. But adjacent of v is not u.



Out-degree:

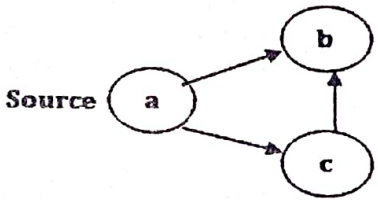
- It is the number of edges beginning from a node.

In-degree:

- It is the number of edges ending at node.

Source:

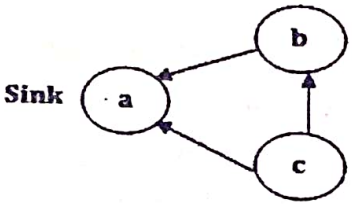
- If a node has positive out-degree and zero in-degree then it is called as source.



Out-degree(a) = 2
In-degree(a) = 0

Sink:

If a node has zero outdegree but positive indegree then it is called as sink.



Out-degree(a) = 0
In-degree(a) = 2

Reachable node:

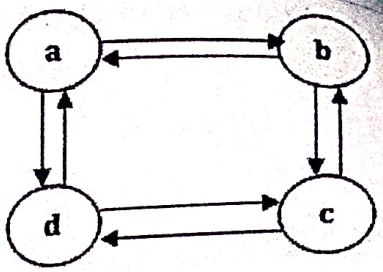
- A node v is said to be reachable from node u if there is a simple path or directed path from u to v.

e program is executing\ ...

Reference: Data Structures With C By Seymour Lipschutz
For Educational Purpose Only. Not For Sale.

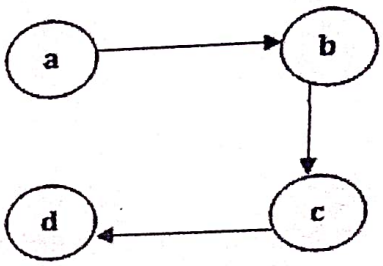
Strongly connected graph:

- A directed graph is said to be strongly connected if for each pair [u, v] of nodes in graph there is a path from u to v and there is also a path from v to u.



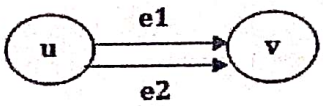
Unilaterally connected graph:

- A directed graph is said to be unilaterally connected if for any pair [u, v] of nodes in graph there is a path either from u to v or a path from v to u.



Parallel edges:

- If two or more edges have same origin node and same terminal node then such edges are called as parallel edges.



In this graph e1 and e2 are parallel edges.

Adjacency Matrix:

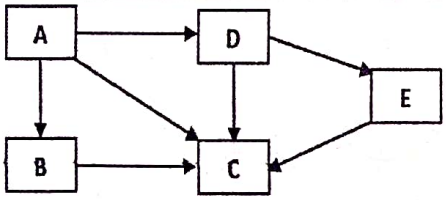
- Adjacency matrix is also called as bit matrix or a boolean matrix.
- The adjacency matrix is denoted by A.
- Let G is a simple directed graph with m nodes then the adjacency matrix will be an order of m x m.
- Adjacency matrix gives details about whether two nodes are adjacent to each other or not.
- In other words it indicates whether there is (directed) edge in between two nodes or not.
- If there is not an (directed) edge in between two nodes then it will be represented by 0 (zero) and if there is a (directed) edge in between two nodes then it will be represented by 1 (one).
- The total number of 1's in adjacency matrix of directed graph is equal to the number of edges in graph.

Note:

- If the graph is weighted then its adjacency matrix is also called as weight matrix.
- In weight matrix instead of 1 write weight of the edge.

Adjacency List:

List of adjacent nodes is called as adjacency list.



Adjacency List:

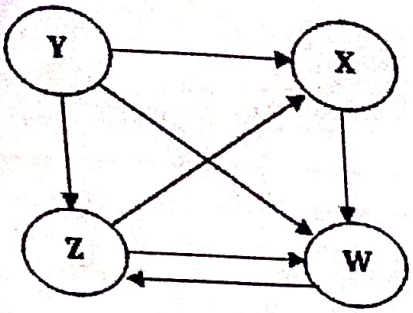
Node	Adjacency List
A	B, C, D
B	C
C	
D	C, E
E	C

Adjacency Matrix:

	A	B	C	D	E
A	0	1	1	1	0
B	0	0	1	0	0
C	0	0	0	0	0
D	0	0	1	0	1
E	0	0	1	0	0

Reference: Data Structures With C By Seymour Lipschutz
For Educational Purpose Only. Not For Sale.

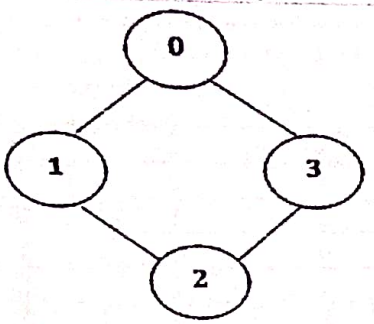
Example-1: Find adjacency matrix of following graph



A =

	X	Y	Z	W
X	0	0	0	1
Y	1	0	1	1
Z	1	0	0	1
W	0	0	1	0

Example-2: Find Adjacency Matrix for the following graph

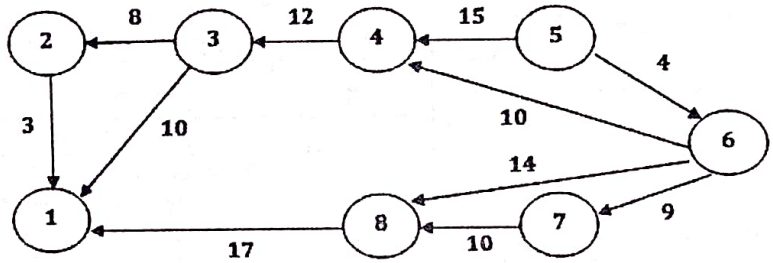


A =

	0	1	2	3
0	0	1	0	1
1	1	0	1	0
2	0	1	0	1
3	1	0	1	0

Example-3: For the following graph obtain

1) The indegree & outdegree of each vertex
2) It's adjacency matrix
3) It's adjacency list



Answer:

Indegree and Outdegree

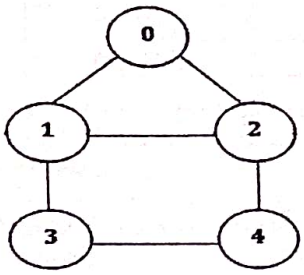
Vertex Number	Indegree	Outdegree
1	3	0
2	1	1
3	1	2
4	2	1
5	0	2
6	1	3
7	1	1
8	2	1

Adjacency List

- Adjacency Matrix

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	3	0	0	0	0	0	0	0
3	10	8	0	0	0	0	0	0
4	0	0	12	0	0	0	0	0
5	0	0	0	15	0	4	0	0
6	0	0	0	10	0	0	9	14
7	0	0	0	0	0	0	0	10
8	17	0	0	0	0	0	0	0

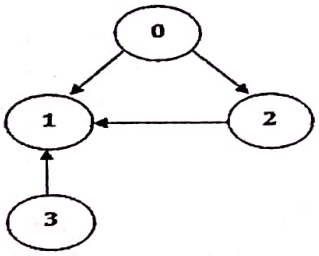
Example-4: Find Adjacency Matrix for the following graph



A =

	0	1	2	3	4
0	0	1	1	0	0
1	1	0	1	1	0
2	1	1	0	0	1
3	0	1	0	0	1
4	0	0	1	1	0

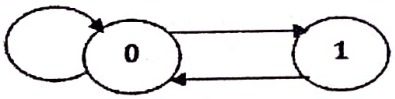
Example-5: Find Adjacency Matrix for the following graph



A =

	0	1	2	3
0	0	1	1	0
1	0	0	0	0
2	0	1	0	0
3	0	1	0	0

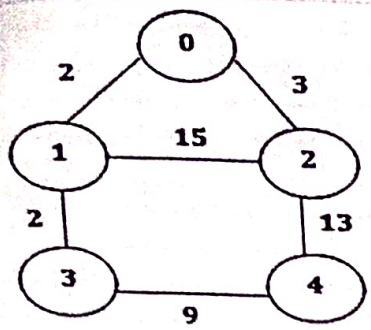
Example-6: Find Adjacency Matrix for the following graph



A =

	0	1
0	1	1
1	1	0

Example-7: Find Adjacency Matrix for the following graph



A =

	0	1	2	3	4
0	0	2	3	0	0
1	2	0	15	2	0
2	3	15	0	0	13
3	0	2	0	0	9
4	0	0	13	9	0

Example: Consider the following specification of a graph G
V(G) = {1, 2, 3, 4}
E(G) = ((1, 2), (1, 3), (3, 3), (3, 4), (4, 1))
Draw an directed graph
Draw its adjacency matrix

Traversing Graph:

Breadth First Search:

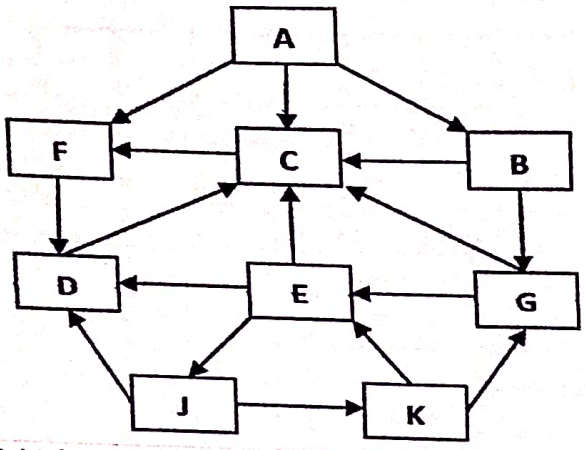
- The Breadth First search algorithm is used to find a path in between two nodes.
- Once the algorithm visits and marks the starting node, then it moves towards the unvisited nodes and analyses them.
- BFS is useful for analyzing the nodes in a graph and constructing the shortest path of traversing through these.
- This algorithm uses a Queue while processing.

Algorithm:
This algorithm executes a Breadth First Search on a graph G beginning at a starting node A.
Variables:

QUEUE	It is an array to implement a queue.
ORIGIN	It is an array to implement a queue

1. Initialize all nodes to the ready state (i.e. STATUS = 1)
2. Put the starting node A in QUEUE and change its status to the waiting state (i.e. STATUS = 2)
3. Repeat steps 4 and 5 until QUEUE is empty
4. Remove the front node N of QUEUE. Process N and change the status of N to the processed state (i.e. STATUS = 3).
5. Add to the rear of QUEUE all the neighbors of N that are in the ready state (i.e. STATUS = 1), and change their status to the waiting state (STATUS = 2).
[End of step-3 loop]
6. Exit.

Example: Apply Breadth First Search to the following graph to find path from A to J.



Adjacency List:

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

Status Chart:

Node	Status		
A	1		
B	1		
C	1		
D	1		
E	1		
F	1		
G	1		
J	1		
K	1		

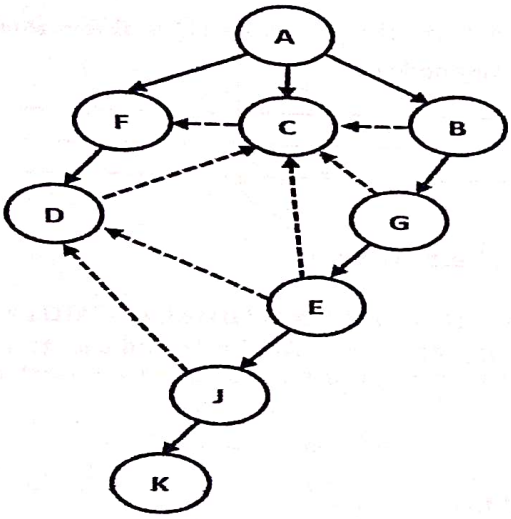
Initial Step:

- Insert start node A in the QUEUE array and change its status to 2.
- Insert origin of A in the ORIGIN array. As A is considered as start so its origin is Φ.
- Draw A in BFS spanning tree

A								
QUEUE[1]	QUEUE[2]	QUEUE[3]	QUEUE[4]	QUEUE[5]	QUEUE[6]	QUEUE[7]	QUEUE[8]	QUEUE[9]

Φ								
ORIGIN[1]	ORIGIN[2]	ORIGIN[3]	ORIGIN[4]	ORIGIN[5]	ORIGIN[6]	ORIGIN[7]	ORIGIN[8]	ORIGIN[9]

BFS Spanning Tree:



Repeating Steps:

1. Remove first node from QUEUE array and change its status to 3.
2. Insert all the neighbors of the removed node in the QUEUE array whose status is 1.
3. After inserting the adjacent nodes in the QUEUE array, change their status to 2.
4. Also, insert the origin (i.e. removed node) of adjacent nodes in the ORIGIN array.
5. Draw the adjacent nodes in the BFS spanning tree.
6. Repeat above steps till the destination node (i.e. J) is inserted in the array QUEUE.
7. Now, backtrack from J, using the array ORIGIN to find path P.

- Backtrack: J <- E <- G <- B <- A
- That Means:
  1. Origin of J = E
  2. Origin of E = G
  3. Origin of G = B
  4. Origin of B = A
- Thus, the path is: A -> B -> G -> E -> J
- Check the spanning tree to confirm.

Depth First Search:

- The following algorithm will process only those nodes which are reachable from the starting node A.

Algorithm:

Variables:

STACK	It is an array to implement stack

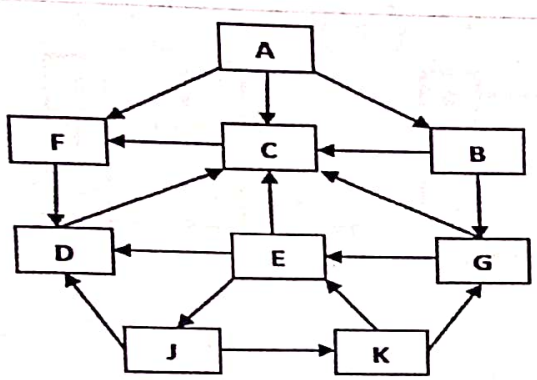
1. **Initialize all nodes to the ready state (STATUS = 1).**
2. **Push the starting node A onto STACK and change its status to the waiting state (STATUS = 2).**
3. **Repeat Steps 4 and 5 until STACK is empty.**
4. **Pop the top node N of STACK. Process N and change its status to the processed state (STATUS = 3).**
5. **Push onto STACK all the neighbors of N that are still in the ready state (STATUS = 1), and change their status to the waiting state (STATUS = 2).**

[End of Step-3 loop]

6. **Exit.**

Example: Find all the nodes which are reachable from J

Answer:



Adjacency List:

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

Status Chart:

Node	Status		
A	1		
B	1		
C	1		
D	1		
E	1		
F	1		
G	1		
J	1		
K	1		

Initial Step:

- Push the starting node J in the STACK and change its status to 2.
- Draw J in the DFS spanning tree

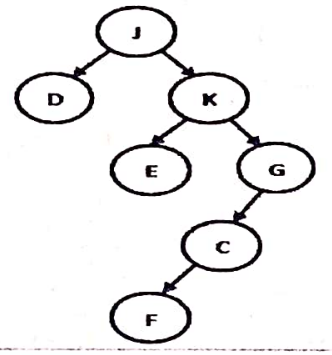
STACK[5]	
STACK[4]	
STACK[3]	
STACK[2]	
STACK[1]	J

Repeating Steps:

1. Pop top node from STACK and change its status to 3
2. Write down the popped node
3. Push all the adjacent nodes of popped node in the STACK whose status is 1.
4. After pushing them change their status to 2.
5. Draw the adjacent nodes in the DFS spanning tree.
6. Repeat the above steps until the STACK is empty.

Reachable Nodes from J: J, K, G, C, F, E, D

DFS Spanning Tree:



10

Spanning Tree:

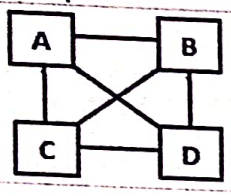
Connected Graph:

- A connected graph is a graph in which there is always a path from a vertex to any other vertex.

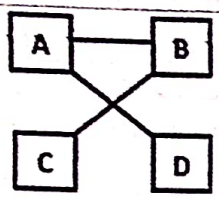
Spanning Tree:

- A spanning tree is a sub-graph or subset of a connected graph, which connects all the vertices of the graph with a minimum number of edges.
- A spanning tree never contains a cycle.
- The edges in the graph may or may not have weights assigned to them.
- The total number of spanning trees of a complete graph with n vertices is n ^ (n-2). This statement is valid for a complete graph only.
- The total number of edges in a spanning tree = (number of vertices - 1)

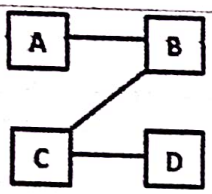
Example:



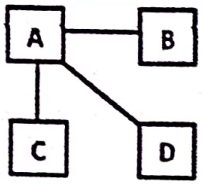
- For the given graph the possible spanning trees are as follows:
- Number of spanning trees = 4 ^ (4 - 2) = 4 ^ 2 = 16
- Number of edges in the spanning tree = 4 – 1 = 3
- Some spanning trees of given graph are as follows:



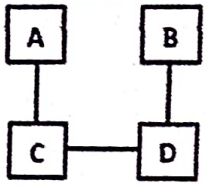
Spanning Tree: 1



Spanning Tree: 2



Spanning Tree: 3

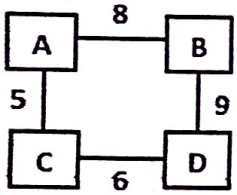


Spanning Tree: 4

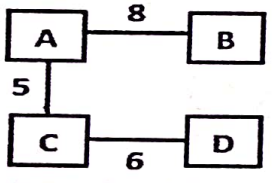
Minimum Spanning Tree:

- A minimum spanning tree can be created for a weighted graph only.
- The spanning tree in which the sum of the weight of the edges is minimum, is called as minimum spanning tree.
- The sum of weight of edges is also called as cost of the spanning tree.
- Example:
- Suppose we want to create a computer network then we can create a minimum spanning tree and by using minimum spanning tree we can connect the computers by using the cable of shortest length.

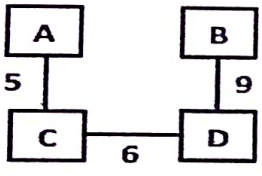
Example:



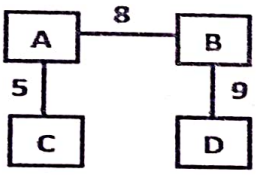
The possible spanning tree for the above graph are as follows:



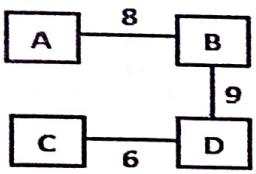
Sum of weight of Edges = 19



Sum of weight of Edges = 20



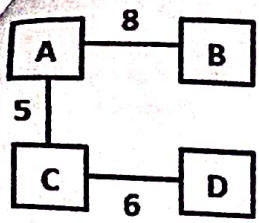
Sum of weight of Edges = 22



Sum of weight of Edges = 23

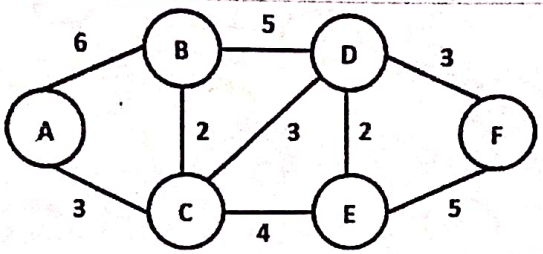
∴ So, the minimum spanning tree for the above graph is as follows:

lled a

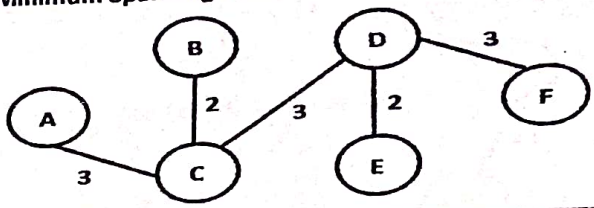


Sum of weight of Edges = 19

Example:



Minimum Spanning Tree:



Prim's Algorithm:

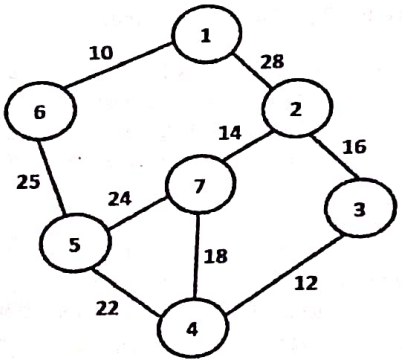
- Prim's algorithm is used to find minimum spanning tree
- It is a greedy algorithm.
- The time complexity of Prim's algorithm is O(E log V).

Algorithm:

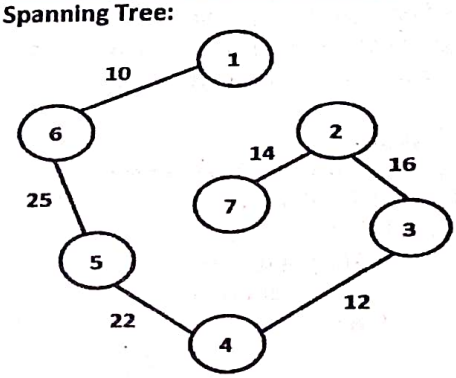
1. Select an edge with minimum weight with a vertex chosen at random
2. For rest of the procedure always select a connected edge (i.e. an edge forming a tree) with minimum weight.
3. If the edge is forming a cycle then reject/discard it and find another edge.
4. Keep adding edges until we connect all vertices.

Example:

Graph:



Minimum Spanning Tree:



Sum of weight of edges = 99

12

Kruskal's Algorithm:

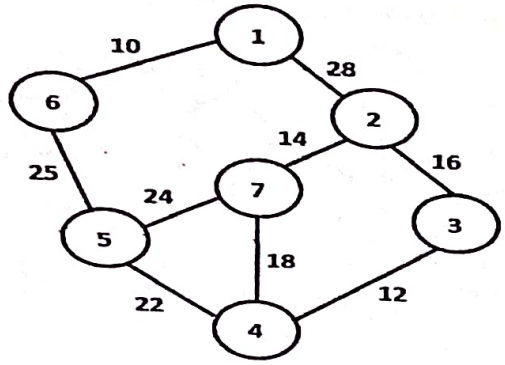
- Kruskal's algorithm is used to find minimum spanning tree
- It is a greedy algorithm.
- Complexity of Kruskal's algorithm is Θ(n^2)

Algorithm:

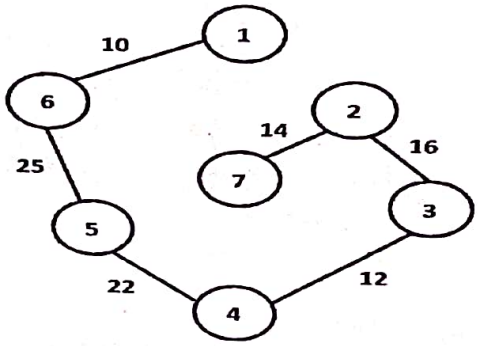
1. Always select an edge with minimum weight (no matter whether it is connected or not).
2. If the edge is forming a cycle then reject/discard it and find another edge.
3. Keep adding edges until we connect all vertices.

Example:

Graph:



Minimum Spanning Tree:



Sum of weight of edges = 99

Warshall's Algorithm:

- The Warshall's algorithm is used to find Boolean Path Matrix of a directed graph.

Warshall's Algorithm:

Steps:

- Let G is a directed graph with M nodes.
- First, find the adjacency matrix A of given graph G.
- The adjacency matrix is the initial path matrix P_0.
- P_0 is the first input to the Warshall's algorithm
- As the graph contains M nodes so, the Warshall's algorithms will produce $P_1, P_2, .. P_M$ path matrices.
- P_M will be the required path matrix P.

Warshall's Algorithm:

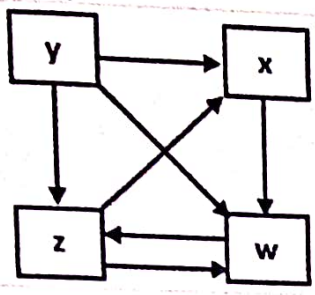
[Initialize P_0, P_0 is same as Adjacency Matrix]

```
1. Repeat for I = 1 To M, By 1:
2.     Repeat for J = 1 To M, By 1:
3.         If A[I, J] = 0, then:
4.             Set P[I, J] := 0
5.         Else:
6.             Set P[I, J] := 1
           [End of If]
       [End of loop]
   [End of loop]

   [v, s, d]
7. Repeat Steps 8 and 9 for K = 1 To M, By 1:
8.     Repeat Steps 9  for I = 1 To M, By 1:
9.         Repeat for J = 1 To M, By 1:
```

10. Set P[I, J] := P[I, J] OR (P[I, K] AND P[K, J])
[End of loop]
[End of loop]
[End of loop]
11. Exit

Example:



Adjacency Matrix (A) =

	x	y	z	w
x	0	0	0	1
y	1	0	1	1
z	1	0	0	1
w	0	0	1	0

(Initial Matrix) P_0 =

		x	y	z	w
		1	2	3	4
x	1	0	0	0	1
y	2	1	0	1	1
z	3	1	0	0	1
w	4	0	0	1	0

Hint for calculations:
Set P[I, J] := P[I, J] OR (P[I, K] AND P[K, J])

Note:

- K = 1 denotes first node x, it means finding paths from node i to node j via node x,
- For example:

When I = 1
K = 1, I = 1, J = 1 means path from node x to node x via node x.
K = 1, I = 1, J = 2 means path from node x to node y via node x.
K = 1, I = 1, J = 3 means path from node x to node z via node x.
K = 1, I = 1, J = 4 means path from node x to node w via node x.

When I = 2
K = 1, I = 2, J = 1 means path from node y to node x via node x.
K = 1, I = 2, J = 2 means path from node y to node y via node x.
K = 1, I = 2, J = 3 means path from node y to node z via node x.
K = 1, I = 2, J = 4 means path from node y to node w via node x.

When I = 3
K = 1, I = 3, J = 1 means path from node z to node x via node x.
K = 1, I = 3, J = 2 means path from node z to node y via node x.
K = 1, I = 3, J = 3 means path from node z to node z via node x.
K = 1, I = 3, J = 4 means path from node z to node w via node x.

When I = 4
K = 1, I = 4, J = 1 means path from node w to node x via node x.
K = 1, I = 4, J = 2 means path from node w to node y via node x.
K = 1, I = 4, J = 3 means path from node w to node z via node x.
K = 1, I = 4, J = 4 means path from node w to node w via node x.

K = 2 denotes second node y, it means finding paths from node i to node j via node x, y.

(14)

K = 2, I = 1, J = 1 means path from node x to node x via node x, y.
K = 2, I = 1, J = 2 means path from node x to node y via node x, y.
K = 2, I = 1, J = 3 means path from node x to node z via node x, y.
K = 2, I = 1, J = 4 means path from node x to node w via node x, y.

K = 3 denotes second node z, it means finding paths from node i to node j via node x, y, z.
K = 3, I = 1, J = 1 means path from node x to node x via node x, y, z.
K = 3, I = 1, J = 2 means path from node x to node y via node x, y, z.
K = 3, I = 1, J = 3 means path from node x to node z via node x, y, z.
K = 3, I = 1, J = 4 means path from node x to node w via node x, y, z.

K = 4 denotes second node z, it means finding paths from node i to node j via node x, y, z, w.
K = 4, I = 1, J = 1 means path from node x to node x via node x, y, z, w.
K = 4, I = 1, J = 2 means path from node x to node y via node x, y, z, w.
K = 4, I = 1, J = 3 means path from node x to node z via node x, y, z, w.
K = 4, I = 1, J = 4 means path from node x to node w via node x, y, z, w.

- Similar for other nodes.

1. **For K = 1, I = 1, J = 1 to 4 (i.e. number of nodes)**
   - P[1, 1] = P[1, 1] OR (P[1, 1] AND P[1, 1]) = 0 OR (0 AND 0) = 0
   - P[1, 2] = P[1, 2] OR (P[1, 1] AND P[1, 2]) = 0 OR (0 AND 0) = 0
   - P[1, 3] = P[1, 3] OR (P[1, 1] AND P[1, 3]) = 0 OR (0 AND 0) = 0
   - P[1, 4] = P[1, 4] OR (P[1, 1] AND P[1, 4]) = 1 OR (0 AND 1) = 1
2. **For K = 1, I = 2, J = 1 to 4 (i.e. number of nodes)**
   - P[2, 1] = P[2, 1] OR (P[2, 1] AND P[1, 1]) = 1 OR (1 AND 0) = 1
   - P[2, 2] = P[2, 2] OR (P[2, 1] AND P[1, 2]) = 0 OR (1 AND 0) = 0
   - P[2, 3] = P[2, 3] OR (P[2, 1] AND P[1, 3]) = 1 OR (1 AND 0) = 1
   - P[2, 4] = P[2, 4] OR (P[2, 1] AND P[1, 4]) = 1 OR (1 AND 1) = 1
3. **For K = 1, I = 3, J = 1 to 4 (i.e. number of nodes)**
   - P[3, 1] = P[3, 1] OR (P[3, 1] AND P[1, 1]) = 1 OR (1 AND 0) = 1
   - P[3, 2] = P[3, 2] OR (P[3, 1] AND P[1, 2]) = 0 OR (1 AND 0) = 0
   - P[3, 3] = P[3, 3] OR (P[3, 1] AND P[1, 3]) = 0 OR (1 AND 0) = 0
   - P[3, 4] = P[3, 4] OR (P[3, 1] AND P[1, 4]) = 1 OR (1 AND 1) = 1
4. **For K = 1, I = 4, J = 1 to 4 (i.e. number of nodes)**
   - P[4, 1] = P[4, 1] OR (P[4, 1] AND P[1, 1]) = 0 OR (0 AND 0) = 0
   - P[4, 2] = P[4, 2] OR (P[4, 1] AND P[1, 2]) = 0 OR (0 AND 0) = 0
   - P[4, 3] = P[4, 3] OR (P[4, 1] AND P[1, 3]) = 1 OR (0 AND 0) = 1
   - P[4, 4] = P[4, 4] OR (P[4, 1] AND P[1, 4]) = 0 OR (0 AND 1) = 0

(For K = 1)
P_1 =

		x	y	z	w
		1	2	3	4
x	1	0	0	0	1
y	2	1	0	1	1
z	3	1	0	0	1
w	4	0	0	1	0

1. **For K = 2, I = 1, J = 1 to 4 (i.e. number of nodes)**
   - P[1, 1] = P[1, 1] OR (P[1, 2] AND P[2, 1]) = 0 OR (0 AND 1) = 0
   - P[1, 2] = P[1, 2] OR (P[1, 2] AND P[2, 2]) = 0 OR (0 AND 0) = 0
   - P[1, 3] = P[1, 3] OR (P[1, 2] AND P[2, 3]) = 0 OR (0 AND 1) = 0
   - P[1, 4] = P[1, 4] OR (P[1, 2] AND P[2, 4]) = 1 OR (0 AND 1) = 1

15

2. **For K = 2, I = 2, J = 1 to 4 (i.e. number of nodes)**
   - P[2, 1] = P[2, 1] OR (P[2, 2] AND P[2, 1]) = 1 OR (0 AND 1) = 1
   - P[2, 2] = P[2, 2] OR (P[2, 2] AND P[2, 2]) = 0 OR (0 AND 0) = 0
   - P[2, 3] = P[2, 3] OR (P[2, 2] AND P[2, 3]) = 1 OR (0 AND 1) = 1
   - P[2, 4] = P[2, 4] OR (P[2, 2] AND P[2, 4]) = 1 OR (0 AND 1) = 1

3. **For K = 2, I = 3, J = 1 to 4 (i.e. number of nodes)**
   - P[3, 1] = P[3, 1] OR (P[3, 2] AND P[2, 1]) = 1 OR (0 AND 1) = 1
   - P[3, 2] = P[3, 2] OR (P[3, 2] AND P[2, 2]) = 0 OR (0 AND 0) = 0
   - P[3, 3] = P[3, 3] OR (P[3, 2] AND P[2, 3]) = 0 OR (0 AND 1) = 0
   - P[3, 4] = P[3, 4] OR (P[3, 2] AND P[2, 4]) = 1 OR (0 AND 1) = 1

4. **For K = 2, I = 4, J = 1 to 4 (i.e. number of nodes)**
   - P[4, 1] = P[4, 1] OR (P[4, 2] AND P[2, 1]) = 0 OR (0 AND 1) = 0
   - P[4, 2] = P[4, 2] OR (P[4, 2] AND P[2, 2]) = 0 OR (0 AND 0) = 0
   - P[4, 3] = P[4, 3] OR (P[4, 2] AND P[2, 3]) = 1 OR (0 AND 1) = 1
   - P[4, 4] = P[4, 4] OR (P[4, 2] AND P[2, 4]) = 0 OR (0 AND 1) = 0

(For K = 2)
P_2 =

	x	y	z	w
	1	2	3	4
x 1	0	0	0	1
y 2	1	0	1	1
z 3	1	0	0	1
w 4	0	0	1	0

1. **For K = 3, I = 1, J = 1 to 4 (i.e. number of nodes)**
   - P[1, 1] = P[1, 1] OR (P[1, 3] AND P[3, 1]) = 0 OR (0 AND 1) = 0
   - P[1, 2] = P[1, 2] OR (P[1, 3] AND P[3, 2]) = 0 OR (0 AND 0) = 0
   - P[1, 3] = P[1, 3] OR (P[1, 3] AND P[3, 3]) = 0 OR (0 AND 0) = 0
   - P[1, 4] = P[1, 4] OR (P[1, 3] AND P[3, 4]) = 1 OR (0 AND 1) = 1

2. **For K = 3, I = 2, J = 1 to 4 (i.e. number of nodes)**
   - P[2, 1] = P[2, 1] OR (P[2, 3] AND P[3, 1]) = 1 OR (1 AND 1) = 1
   - P[2, 2] = P[2, 2] OR (P[2, 3] AND P[3, 2]) = 0 OR (1 AND 0) = 0
   - P[2, 3] = P[2, 3] OR (P[2, 3] AND P[3, 3]) = 1 OR (0 AND 0) = 1
   - P[2, 4] = P[2, 4] OR (P[2, 3] AND P[3, 4]) = 1 OR (1 AND 1) = 1

3. **For K = 3, I = 3, J = 1 to 4 (i.e. number of nodes)**
   - P[3, 1] = P[3, 1] OR (P[3, 3] AND P[3, 1]) = 1 OR (0 AND 1) = 1
   - P[3, 2] = P[3, 2] OR (P[3, 3] AND P[3, 2]) = 0 OR (0 AND 0) = 0
   - P[3, 3] = P[3, 3] OR (P[3, 3] AND P[3, 3]) = 0 OR (0 AND 0) = 0
   - P[3, 4] = P[3, 4] OR (P[3, 3] AND P[3, 4]) = 1 OR (0 AND 1) = 1

4. **For K = 3, I = 4, J = 1 to 4 (i.e. number of nodes)**
   - P[4, 1] = P[4, 1] OR (P[4, 3] AND P[3, 1]) = 0 OR (1 AND 1) = 1
   - P[4, 2] = P[4, 2] OR (P[4, 3] AND P[3, 2]) = 0 OR (1 AND 0) = 0
   - P[4, 3] = P[4, 3] OR (P[4, 3] AND P[3, 3]) = 1 OR (1 AND 0) = 1
   - P[4, 4] = P[4, 4] OR (P[4, 3] AND P[3, 4]) = 0 OR (1 AND 1) = 1

(For K = 3)
P_3 =

	x	y	z	w
	1	2	3	4
x 1	0	0	0	1
y 2	1	0	1	1
z 3	1	0	0	1
w 4	1	0	1	1

1. **For K = 4, I = 1, J = 1 to 4 (i.e. number of nodes)**
   - P[1, 1] = P[1, 1] OR (P[1, 4] AND P[4, 1]) = 0 OR (1 AND 1) = 1
   - P[1, 2] = P[1, 2] OR (P[1, 4] AND P[4, 2]) = 0 OR (1 AND 0) = 0
   - P[1, 3] = P[1, 3] OR (P[1, 4] AND P[4, 3]) = 0 OR (1 AND 1) = 1
   - P[1, 4] = P[1, 4] OR (P[1, 4] AND P[4, 4]) = 1 OR (1 AND 1) = 1

2. **For K = 4, I = 2, J = 1 to 4 (i.e. number of nodes)**
   - P[2, 1] = P[2, 1] OR (P[2, 4] AND P[4, 1]) = 1 OR (1 AND 1) = 1
   - P[2, 2] = P[2, 2] OR (P[2, 4] AND P[4, 2]) = 0 OR (1 AND 0) = 0
   - P[2, 3] = P[2, 3] OR (P[2, 4] AND P[4, 3]) = 1 OR (1 AND 1) = 1
   - P[2, 4] = P[2, 4] OR (P[2, 4] AND P[4, 4]) = 1 OR (1 AND 1) = 1

3. **For K = 4, I = 3, J = 1 to 4 (i.e. number of nodes)**
   - P[3, 1] = P[3, 1] OR (P[3, 4] AND P[4, 1]) = 1 OR (1 AND 1) = 1
   - P[3, 2] = P[3, 2] OR (P[3, 4] AND P[4, 2]) = 0 OR (1 AND 0) = 0
   - P[3, 3] = P[3, 3] OR (P[3, 4] AND P[4, 3]) = 0 OR (1 AND 1) = 1
   - P[3, 4] = P[3, 4] OR (P[3, 4] AND P[4, 4]) = 1 OR (1 AND 1) = 1

4. **For K = 4, I = 4, J = 1 to 4 (i.e. number of nodes)**
   - P[4, 1] = P[4, 1] OR (P[4, 4] AND P[4, 1]) = 1 OR (1 AND 1) = 1
   - P[4, 2] = P[4, 2] OR (P[4, 4] AND P[4, 2]) = 0 OR (1 AND 0) = 0
   - P[4, 3] = P[4, 3] OR (P[4, 4] AND P[4, 3]) = 1 OR (1 AND 1) = 1
   - P[4, 4] = P[4, 4] OR (P[4, 4] AND P[4, 4]) = 1 OR (1 AND 1) = 1

(For K = 4)
P_4 = P =

	x	y	z	w
	1	2	3	4
x 1	1	0	1	1
y 2	1	0	1	1
z 3	1	0	1	1
w 4	1	0	1	1

Dijkstra's Algorithm:

- The modified version of Warshall's algorithm is known as Dijkstra's algorithm.
- The Dijkstra's Algorithm is used to find the shortest path matrix from the given point to all the points in a directed weighted graph.
- The path matrix tells us whether or not there are paths between the nodes.
- The Dijkstra's algorithm is also known as Single-Source Shortest Paths Algorithm.

Steps:

- Let G is a directed and weighted graph with M nodes.
- G is maintained in the memory by its weight matrix (i.e. Adjacency matrix) W = (W_{ij})

Reference: Data Structures With C By Seymour Lipschutz
For Educational Purpose Only. Not For Sale.

$w_{ij} = w(e)$, where w(e) indicated weight of an edge	If there is an edge from v_i to v_j
$w_{ij} = 0$	If there is no edge from v_i to v_j

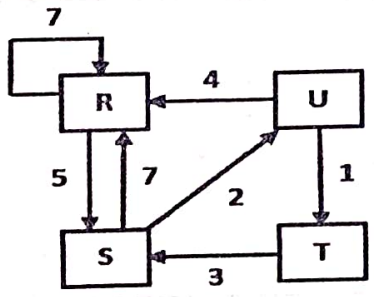
- Here we want to find the matrix Q which will tell us the length of the shortest paths between the nodes.
- Find the Adjacency matrix (i.e. weight matrix (W)) of given graph.
- The initial matrix Q_0 is the same as the weight matrix W except that each 0 in W is replaced by ∞ (i.e. infinity, a very large number).
- Q_0 will be the first input to the Dijkstra's algorithm
- As the graph G contains M nodes so, the Dijkstra's algorithm will produce Q_1, Q_2 ..., Q_M matrices.
- The final matrix Q_M will be the desired matrix Q.
- MIN() is a function which return smallest value.

Dijkstra's Algorithm (Shortest-Path Algorithm):

[Initialize Q_0]
1. Repeat for I = 1, 2, ..., M:
2. Repeat for J = 1, 2, ..., M:
If W[I, J] = 0, Then:
Set Q[I, J] := INFINITY
Else:
Set Q[I, J] := W[I, J]
[End of If]
[End of loop]
[End of loop]

2. Repeat Steps 3 and 4 for K = 1, 2, ..., M:
3. Repeat Steps 4 for 1, 2, ..., M:
4. Repeat for J = 1, 2, ..., M:
Set Q[I, J] := MIN(Q[I, J], Q[I, K] + Q[K, J])
[End of loop]
[End of loop]
[End of Step-2 loop]
5. Exit

Example:



Adjacency Matrix A =

	R	S	T	U
R	7	5	0	0
S	7	0	0	2
T	0	3	0	0
U	4	0	1	0

(Initial Matrix) Q_0 =

		R	S	T	U
		1	2	3	4
R	1	7	5	∞	∞
S	2	7	∞	∞	2
T	3	0	3	∞	∞
U	4	4	∞	1	∞

Hint for calculations:
Set Q[I, J] := MIN(Q[I, J], Q[I, K] + Q[K, J])

[redacted]

Note:

- K = 1 denotes first node R, it means finding paths from node i to node j via node R,
- For example:

  When I = 1

  K = 1, I = 1, J = 1 means path from node R to node R via node R.
  K = 1, I = 1, J = 2 means path from node R to node S via node R.
  K = 1, I = 1, J = 3 means path from node R to node T via node R.
  K = 1, I = 1, J = 4 means path from node R to node U via node R.

  When I = 2

  K = 1, I = 2, J = 1 means path from node S to node R via node R.
  K = 1, I = 2, J = 2 means path from node S to node S via node R.
  K = 1, I = 2, J = 3 means path from node S to node T via node R.
  K = 1, I = 2, J = 4 means path from node S to node U via node R.

  When I = 3

  K = 1, I = 3, J = 1 means path from node T to node R via node R.
  K = 1, I = 3, J = 2 means path from node T to node S via node R.
  K = 1, I = 3, J = 3 means path from node T to node T via node R.
  K = 1, I = 3, J = 4 means path from node T to node U via node R.

  When I = 4

  K = 1, I = 4, J = 1 means path from node U to node R via node R.
  K = 1, I = 4, J = 2 means path from node U to node S via node R.
  K = 1, I = 4, J = 3 means path from node U to node T via node R.
  K = 1, I = 4, J = 4 means path from node U to node U via node R.

  K = 2 denotes second node S, it means finding paths from node i to node j via node R, S,
  K = 2, I = 1, J = 1 means path from node R to node R via node R, S.
  K = 2, I = 1, J = 2 means path from node R to node S via node R, S.
  K = 2, I = 1, J = 3 means path from node R to node T via node R, S.
  K = 2, I = 1, J = 4 means path from node R to node U via node R, S.

  K = 3 denotes second node T, it means finding paths from node i to node j via node R, S, T.
  K = 3, I = 1, J = 1 means path from node R to node R via node R, S, T.
  K = 3, I = 1, J = 2 means path from node R to node S via node R, S, T.
  K = 3, I = 1, J = 3 means path from node R to node T via node R, S, T.
  K = 3, I = 1, J = 4 means path from node R to node U via node R, S, T.

  K = 4 denotes second node z, it means finding paths from node i to node j via node x, y, z, w.
  K = 4, I = 1, J = 1 means path from node R to node R via node R, S, T, U.
  K = 4, I = 1, J = 2 means path from node R to node S via node R, S, T, U.
  K = 4, I = 1, J = 3 means path from node R to node T via node R, S, T, U.
  K = 4, I = 1, J = 4 means path from node R to node U via node R, S, T, U.
- Similar for other nodes.

1. **K = 1, I = 1, J = 1 to 4**
   - Q[1, 1] = MIN(Q[1, 1], Q[1, 1] + Q[1, 1]) = MIN(7, 7 + 7) = 7
   - Q[1, 2] = MIN(Q[1, 2], Q[1, 1] + Q[1, 2]) = MIN(5, 7 + 5) = 5
   - Q[1, 3] = MIN(Q[1, 3], Q[1, 1] + Q[1, 3]) = MIN(∞, 7 + ∞) = ∞
   - Q[1, 4] = MIN(Q[1, 4], Q[1, 1] + Q[1, 4]) = MIN(∞, 7 + ∞) = ∞

2. **K = 1, I = 2, J = 1 to 4**

- Q[2, 1] = MIN(Q[2, 1], Q[2, 1] + Q[1, 1]) = MIN(7, 7 + 7) = 7
- Q[2, 2] = MIN(Q[2, 2], Q[2, 1] + Q[1, 2]) = MIN(∞, 7 + 5) = 12
- Q[2, 3] = MIN(Q[2, 3], Q[2, 1] + Q[1, 3]) = MIN(∞, 7 + ∞) = ∞
- Q[2, 4] = MIN(Q[2, 4], Q[2, 1] + Q[1, 4]) = MIN(2, 7 + ∞) = 2

3. K = 1, I = 3, J = 1 to 4

- Q[3, 1] = MIN(Q[3, 1], Q[3, 1] + Q[1, 1]) = MIN(∞, ∞ + 7) = ∞
- Q[3, 2] = MIN(Q[3, 2], Q[3, 1] + Q[1, 2]) = MIN(3, ∞ + 5) = 3
- Q[3, 3] = MIN(Q[3, 3], Q[3, 1] + Q[1, 3]) = MIN(∞, ∞ + ∞) = ∞
- Q[3, 4] = MIN(Q[3, 4], Q[3, 1] + Q[1, 4]) = MIN(∞, ∞ + ∞) = ∞

4. K = 1, I = 4, J = 1 to 4

- Q[4, 1] = MIN(Q[4, 1], Q[4, 1] + Q[1, 1]) = MIN(4, 4 + 7) = 4
- Q[4, 2] = MIN(Q[4, 2], Q[4, 1] + Q[1, 2]) = MIN(∞, 4 + 9) = 9
- Q[4, 3] = MIN(Q[4, 3], Q[4, 1] + Q[1, 3]) = MIN(1, 4 + ∞) = 1
- Q[4, 4] = MIN(Q[4, 4], Q[4, 1] + Q[1, 4]) = MIN(∞, 4 + ∞) = ∞

Q_1 =

		R	S	T	U
		1	2	3	4
R	1	7	5	∞	∞
S	2	7	12	∞	2
T	3	∞	3	∞	∞
U	4	4	9	1	∞

1. K = 2, I = 1, J = 1 to 4

- Q[1, 1] = MIN(Q[1, 1], Q[1, 2] + Q[2, 1]) = MIN(7, 5 + 7) = 7
- Q[1, 2] = MIN(Q[1, 2], Q[1, 2] + Q[2, 2]) = MIN(5, 5 + 12) = 5
- Q[1, 3] = MIN(Q[1, 3], Q[1, 2] + Q[2, 3]) = MIN(∞, 5 + ∞) = ∞
- Q[1, 4] = MIN(Q[1, 4], Q[1, 2] + Q[2, 4]) = MIN(∞, 5 + 2) = 7

2. K = 2, I = 2, J = 1 to 4

- Q[2, 1] = MIN(Q[2, 1], Q[2, 2] + Q[2, 1]) = MIN(7, 12 + 7) = 7
- Q[2, 2] = MIN(Q[2, 2], Q[2, 2] + Q[2, 2]) = MIN(12, 12 + 12) = 12
- Q[2, 3] = MIN(Q[2, 3], Q[2, 2] + Q[2, 3]) = MIN(∞, 12 + ∞) = ∞
- Q[2, 4] = MIN(Q[2, 4], Q[2, 2] + Q[2, 4]) = MIN(2, 12 + 2) = 2

3. K = 2, I = 3, J = 1 to 4

- Q[3, 1] = MIN(Q[3, 1], Q[3, 2] + Q[2, 1]) = MIN(∞, 3 + 7) = 10
- Q[3, 2] = MIN(Q[3, 2], Q[3, 2] + Q[2, 2]) = MIN(3, 3 + 12) = 3
- Q[3, 3] = MIN(Q[3, 3], Q[3, 2] + Q[2, 3]) = MIN(∞, 3 + ∞) = ∞
- Q[3, 4] = MIN(Q[3, 4], Q[3, 2] + Q[2, 4]) = MIN(∞, 3 + 2) = 5

4. K = 2, I = 4, J = 1 to 4

- Q[4, 1] = MIN(Q[4, 1], Q[4, 2] + Q[2, 1]) = MIN(4, 9 + 7) = 4
- Q[4, 2] = MIN(Q[4, 2], Q[4, 2] + Q[2, 2]) = MIN(9, 9 + 12) = 9
- Q[4, 3] = MIN(Q[4, 3], Q[4, 2] + Q[2, 3]) = MIN(1, 9 + ∞) = 1
- Q[4, 4] = MIN(Q[4, 4], Q[4, 2] + Q[2, 4]) = MIN(∞, 9 + 2) = 11

Q_2 =

		R	S	T	U
		1	2	3	4
R	1	7	5	∞	7
S	2	7	12	∞	2
T	3	10	3	∞	5
U	4	4	9	1	11

1. **K = 3, I = 1, J = 1 to 4**
   - Q[1, 1] = MIN(Q[1, 1], Q[1, 3] + Q[3, 1]) = MIN(7, ∞ + 10) = 7
   - Q[1, 2] = MIN(Q[1, 2], Q[1, 3] + Q[3, 2]) = MIN(5, ∞ + 5) = 5
   - Q[1, 3] = MIN(Q[1, 3], Q[1, 3] + Q[3, 3]) = MIN(∞, ∞ + ∞) = ∞
   - Q[1, 4] = MIN(Q[1, 4], Q[1, 3] + Q[3, 4]) = MIN(7, ∞ + 5) = 7

2. **K = 3, I = 2, J = 1 to 4**
   - Q[2, 1] = MIN(Q[2, 1], Q[2, 3] + Q[3, 1]) = MIN(7, ∞ + 10) = 7
   - Q[2, 2] = MIN(Q[2, 2], Q[2, 3] + Q[3, 2]) = MIN(12, ∞ + 3) = 12
   - Q[2, 3] = MIN(Q[2, 3], Q[2, 3] + Q[3, 3]) = MIN(∞, ∞ + ∞) = ∞
   - Q[2, 4] = MIN(Q[2, 4], Q[2, 3] + Q[3, 4]) = MIN(2, ∞ + 5) = 2

3. **K = 3, I = 3, J = 1 to 4**
   - Q[3, 1] = MIN(Q[3, 1], Q[3, 3] + Q[3, 1]) = MIN(10, ∞ + 10) = 10
   - Q[3, 2] = MIN(Q[3, 2], Q[3, 3] + Q[3, 2]) = MIN(3, ∞ + 3) = 3
   - Q[3, 3] = MIN(Q[3, 3], Q[3, 3] + Q[3, 3]) = MIN(∞, ∞ + ∞) = ∞
   - Q[3, 4] = MIN(Q[3, 4], Q[3, 3] + Q[3, 4]) = MIN(5, ∞ + 5) = 5

4. **K = 3, I = 4, J = 1 to 4**
   - Q[4, 1] = MIN(Q[4, 1], Q[4, 3] + Q[3, 1]) = MIN(4, 1 + 10) = 4
   - Q[4, 2] = MIN(Q[4, 2], Q[4, 3] + Q[3, 2]) = MIN(9, 1 + 3) = 4
   - Q[4, 3] = MIN(Q[4, 3], Q[4, 3] + Q[3, 3]) = MIN(1, 1 + ∞) = 1
   - Q[4, 4] = MIN(Q[4, 4], Q[4, 3] + Q[3, 4]) = MIN(11, 1 + 5) = 6

Q_3 =

	R	S	T	U
	1	2	3	4
R 1	7	5	∞	7
S 2	7	12	∞	2
T 3	10	3	∞	5
U 4	4	4	1	6

1. **K = 4, I = 1, J = 1 to 4**
   - Q[1, 1] = MIN(Q[1, 1], Q[1, 4] + Q[4, 1]) = MIN(7, 7 + 4) = 7
   - Q[1, 2] = MIN(Q[1, 2], Q[1, 4] + Q[4, 2]) = MIN(5, 7 + 4) = 5
   - Q[1, 3] = MIN(Q[1, 3], Q[1, 4] + Q[4, 3]) = MIN(∞, 7 + 1) = 8
   - Q[1, 4] = MIN(Q[1, 4], Q[1, 4] + Q[4, 4]) = MIN(7, 7 + 6) = 7

2. **K = 4, I = 2, J = 1 to 4**
   - Q[2, 1] = MIN(Q[2, 1], Q[2, 4] + Q[4, 1]) = MIN(7, 2 + 4) = 6
   - Q[2, 2] = MIN(Q[2, 2], Q[2, 4] + Q[4, 2]) = MIN(12, 2 + 4) = 6
   - Q[2, 3] = MIN(Q[2, 3], Q[2, 4] + Q[4, 3]) = MIN(∞, 2 + 1) = 3
   - Q[2, 4] = MIN(Q[2, 4], Q[2, 4] + Q[4, 4]) = MIN(2, 2 + 6) = 2

3. **K = 4, I = 3, J = 1 to 4**
   - Q[3, 1] = MIN(Q[3, 1], Q[3, 4] + Q[4, 1]) = MIN(10, 5 + 4) = 9
   - Q[3, 2] = MIN(Q[3, 2], Q[3, 4] + Q[4, 2]) = MIN(3, 5 + 4) = 3
   - Q[3, 3] = MIN(Q[3, 3], Q[3, 4] + Q[4, 3]) = MIN(∞, 5 + 1) = 6
   - Q[3, 4] = MIN(Q[3, 4], Q[3, 4] + Q[4, 4]) = MIN(5, 5 + 6) = 5

4. **K = 4, I = 4, J = 1 to 4**
   - Q[4, 1] = MIN(Q[4, 1], Q[4, 4] + Q[4, 1]) = MIN(4, 6 + 4) = 4
   - Q[4, 2] = MIN(Q[4, 2], Q[4, 4] + Q[4, 2]) = MIN(4, 6 + 4) = 4
   - Q[4, 3] = MIN(Q[4, 3], Q[4, 4] + Q[4, 3]) = MIN(1, 6 + 1) = 1

- Q[4, 4] = MIN(Q[4, 4], Q[4, 4] + Q[4, 4]) = MIN(6, 6 + 6) = 6

$Q_4 = Q =$

		R	S	T	U
		1	2	3	4
R	1	7	5	8	7
S	2	6	6	3	2
T	3	9	3	6	5
U	4	4	4	1	6