Nonlinear Data Structure

Unit#3

Marwadi University

Department of Information Technology

Data Structure
01CE0301 / 3130702

Ravikumar Natarajan

Highlights

- Trees
- Graphs

Till Now

- Trees
- **Graphs**

- Graphs and their understanding
- Matrix representations of a given graph
- Depth First Search (DFS)
- Breadth First Search (BFS)
- Minimum Spanning Trees Algorithms (Prims, Kruskal, Dijkstra)
- Path Matrix
- Warshall's Algorithm

Up Next

- Trees
- **Graphs**

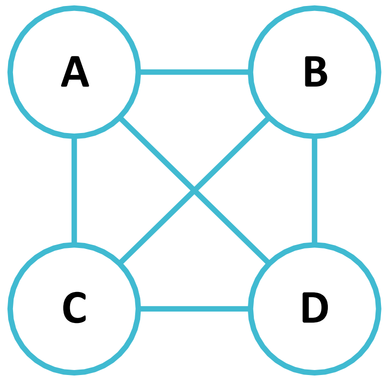
- Graphs and their understanding
- Matrix representations of a given graph
- Depth First Search (DFS)
- Breadth First Search (BFS)
- Minimum Spanning Trees Algorithms (Prims, Kruskal, Dijkstra)
- Path Matrix
- Warshall's Algorithm

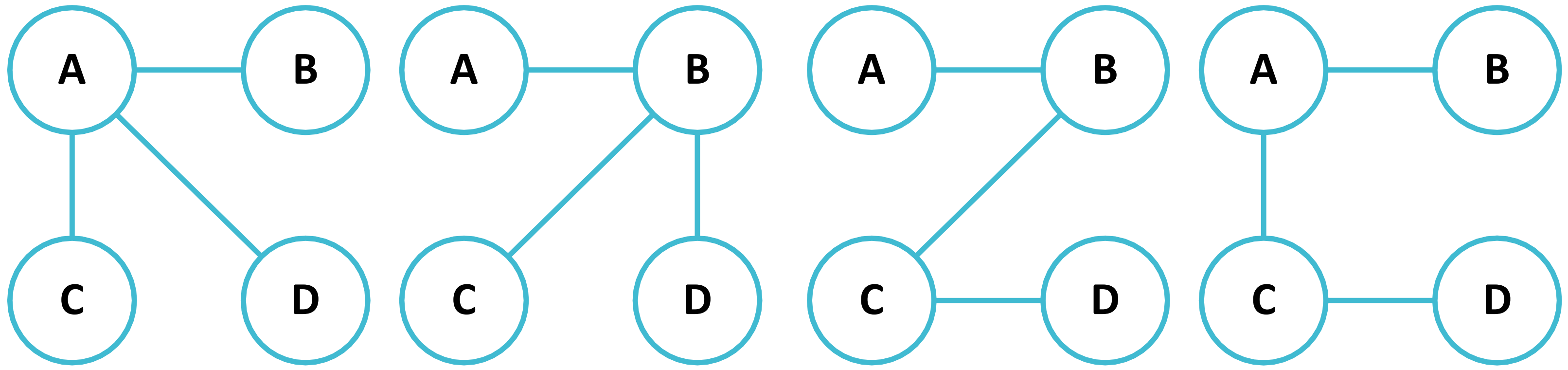
Shortest Path Algorithms

- Minimum spanning tree
- Dijkstra's algorithm
- Warshall's algorithm

Spanning Tree

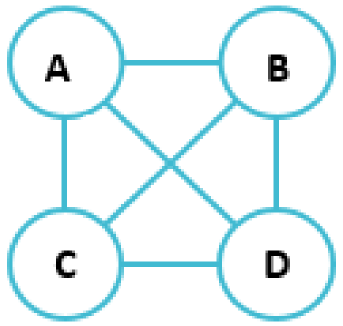
- A spanning tree of a connected, undirected graph G is a sub-graph of G which is a tree that connects all the vertices together.



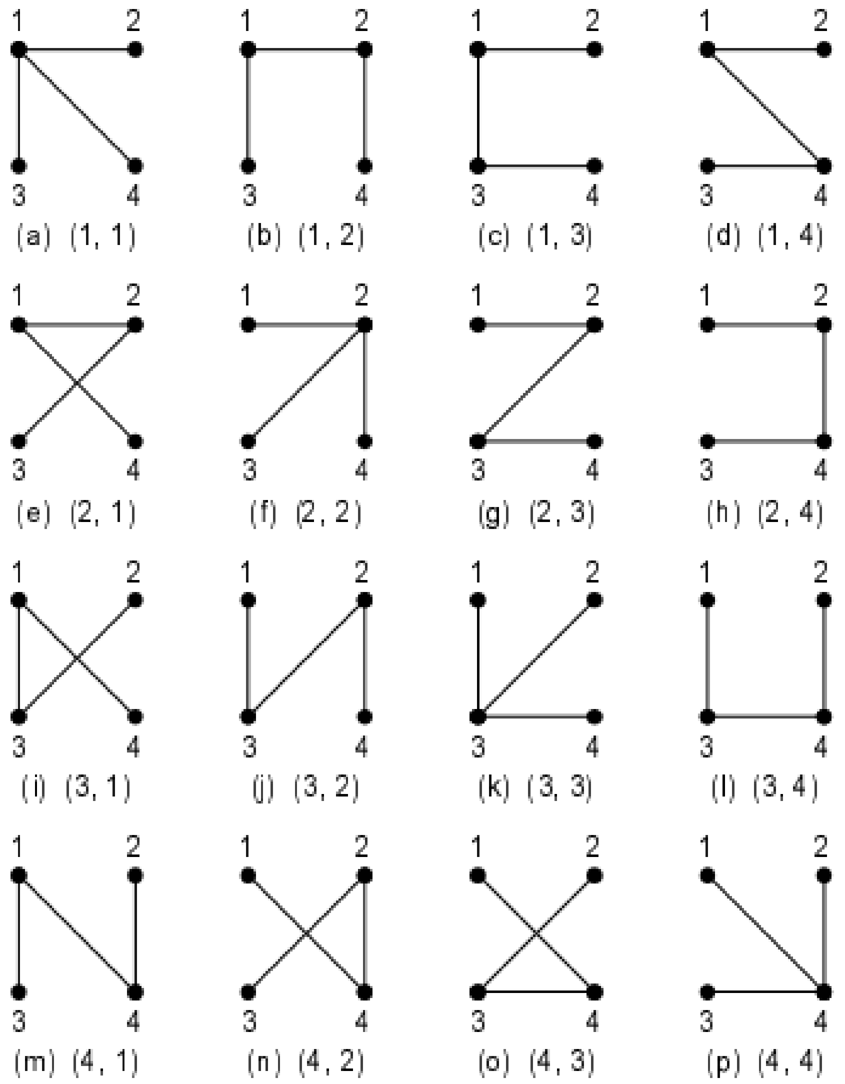


And many more…

Spanning Tree



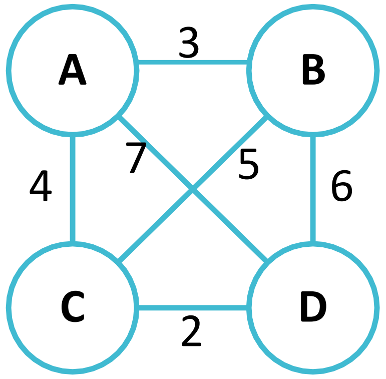
- A spanning should satisfy V – 1 edges
- **HOW MANY NUMBER OF SPANNING TREE POSSIBILITIES?**
- $n^{(n-2)}$ where n is number of vertices

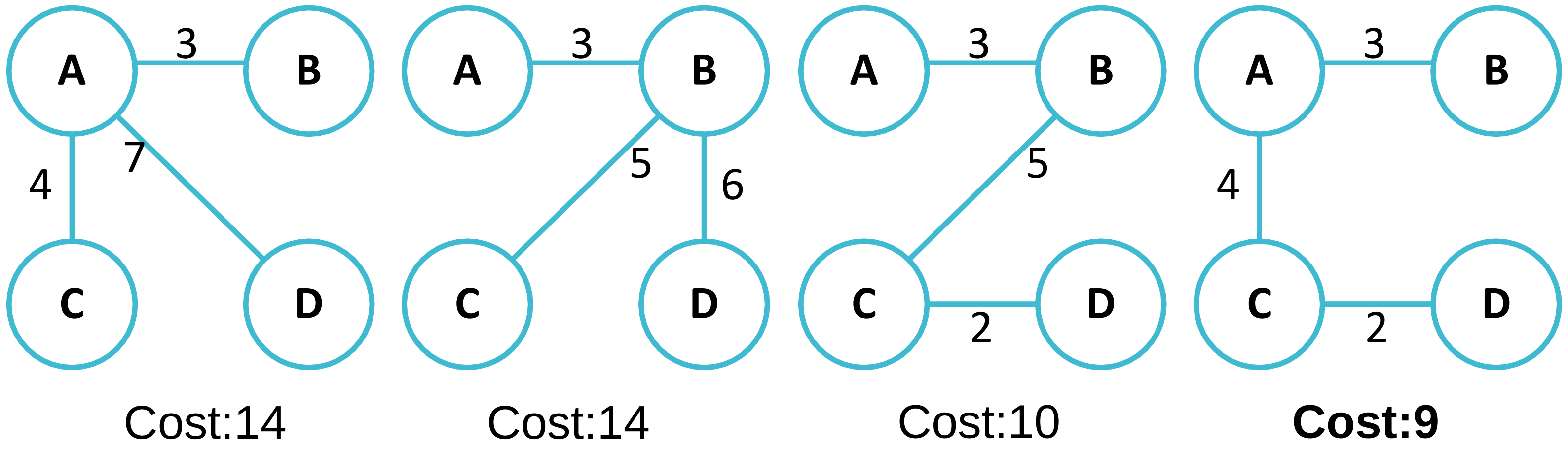


All 16 spanning trees of K4

Minimum Spanning Tree

- A minimum spanning tree (MST) is defined as a spanning tree with weight less than or equal to the weight of other spanning tree.





Applications of MST

- Network Designs. Like telephone network, to choose least costly path.
- To find airline routes. Vertices denote cities, edges represent routes between cities.
- MSTs are applied in routing algorithm for finding the most efficient path.

Algorithms

- Kruskal's Algorithm
  - Minimum Spanning Tree
- Prim's Algorithm
  - Minimum Spanning Tree
  - Shortest Path from Src$\longrightarrow$Dest

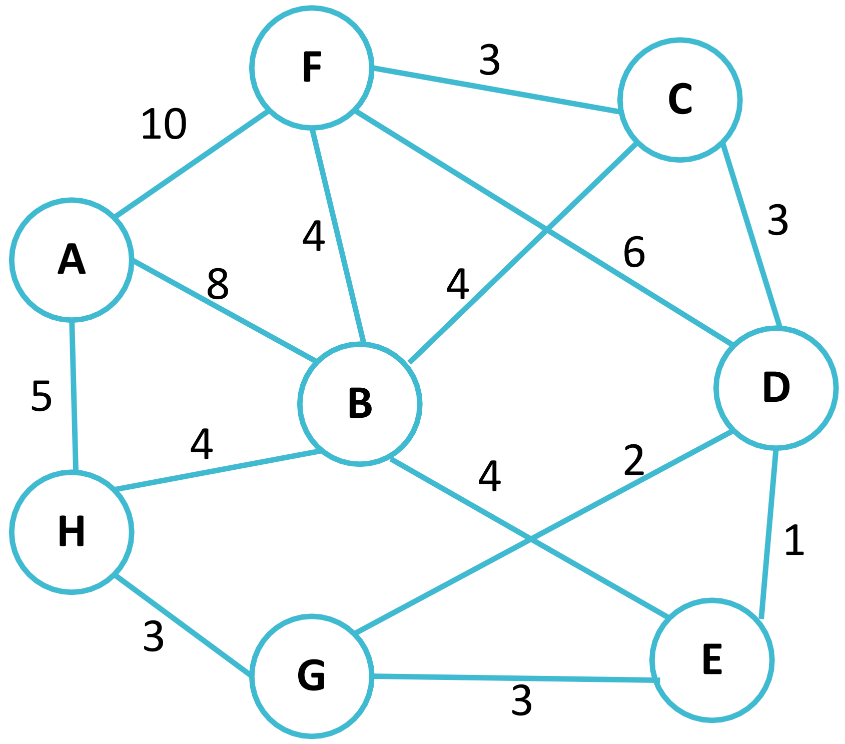
Kruskal's Algorithm

- Kruskal's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which
  - form a tree that includes every vertex
  - has the minimum sum of weights among all the trees that can be formed from the graph

- Kruskal performs better in typical situations (sparse graphs) and is easier to implement because it uses disjoint sets and simpler data structures.
- It falls under a class of algorithms called greedy algorithms that find the local optimum in the hopes of finding a global optimum.

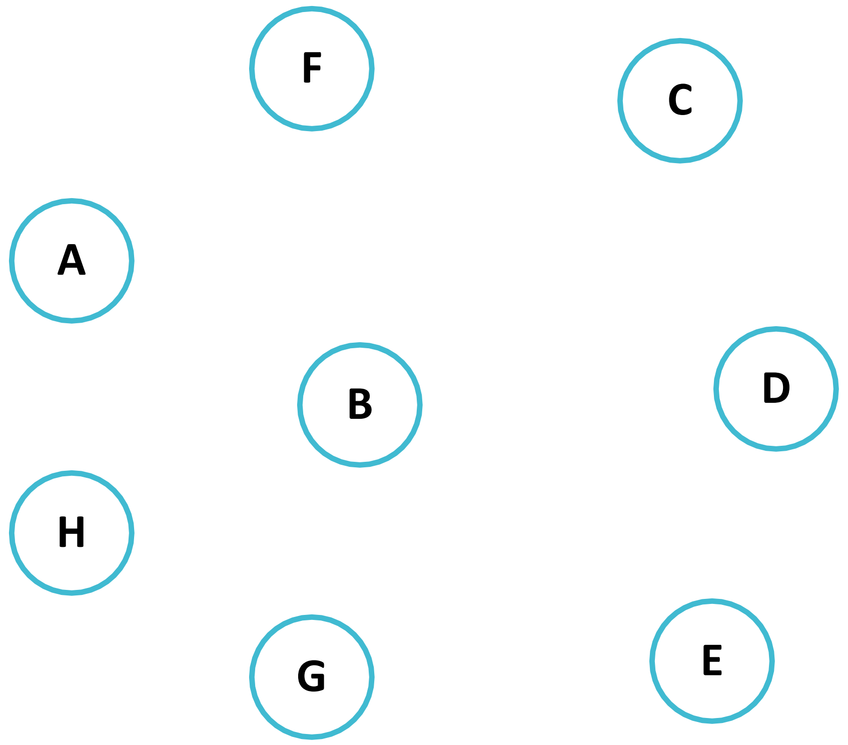
Kruskal's Algorithm

- Problem: Find Minimum Spanning Tree of a Graph



Kruskal's Algorithm: Solution Steps

Initialize:
Sort the edges by increasing edge weight

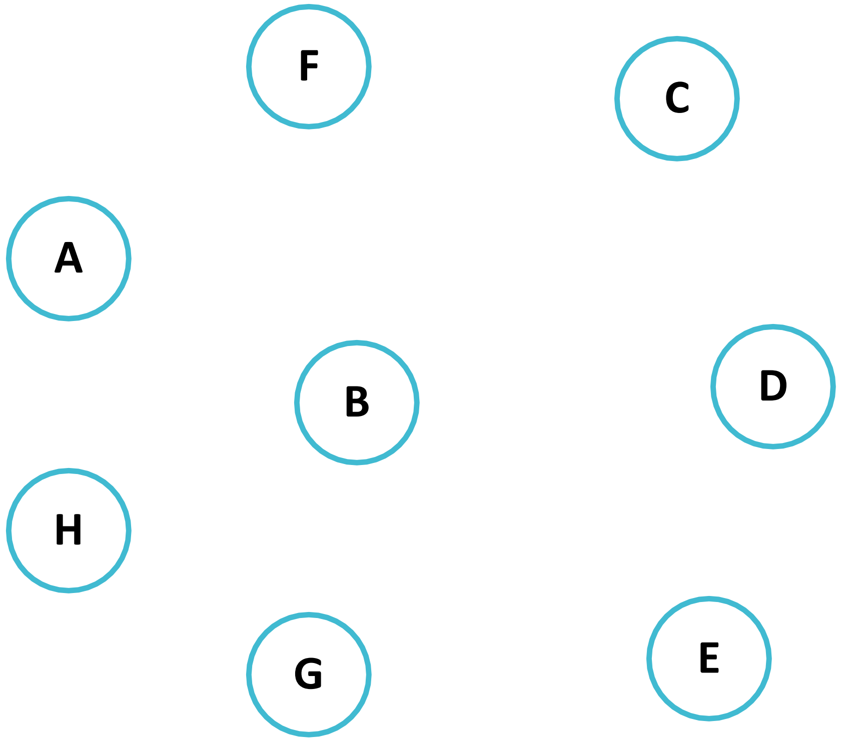


S : { }

Edge	Ew
(D,E)	1
(D,G)	2
(E,G)	3
(C,D)	3
(G,H)	3
(C,F)	3
(B,C)	4
(B,E)	4
(B,F)	4
(B,H)	4
(A,H)	5
(D,F)	6
(A,B)	8
(A,F)	10

Kruskal's Algorithm: Solution Steps

Select first |V|–1 edges which do not generate a cycle

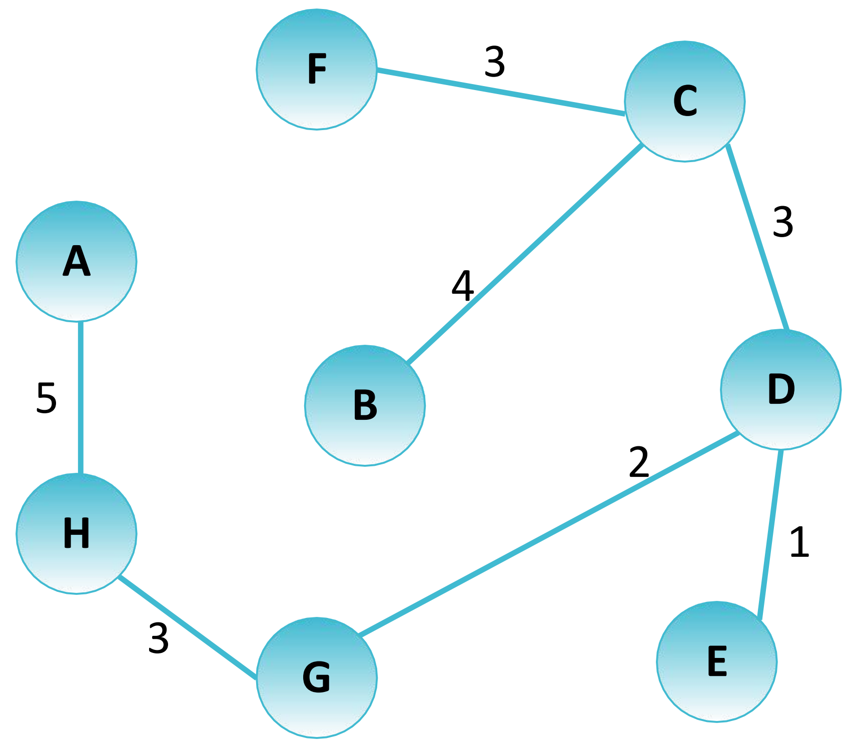


S :

Edge	Ew
(D,E)	1
(D,G)	2
(E,G)	3
(C,D)	3
(G,H)	3
(C,F)	3
(B,C)	4
(B,E)	4
(B,F)	4
(B,H)	4
(A,H)	5
(D,F)	6
(A,B)	8
(A,F)	10

Kruskal's Algorithm: Solution

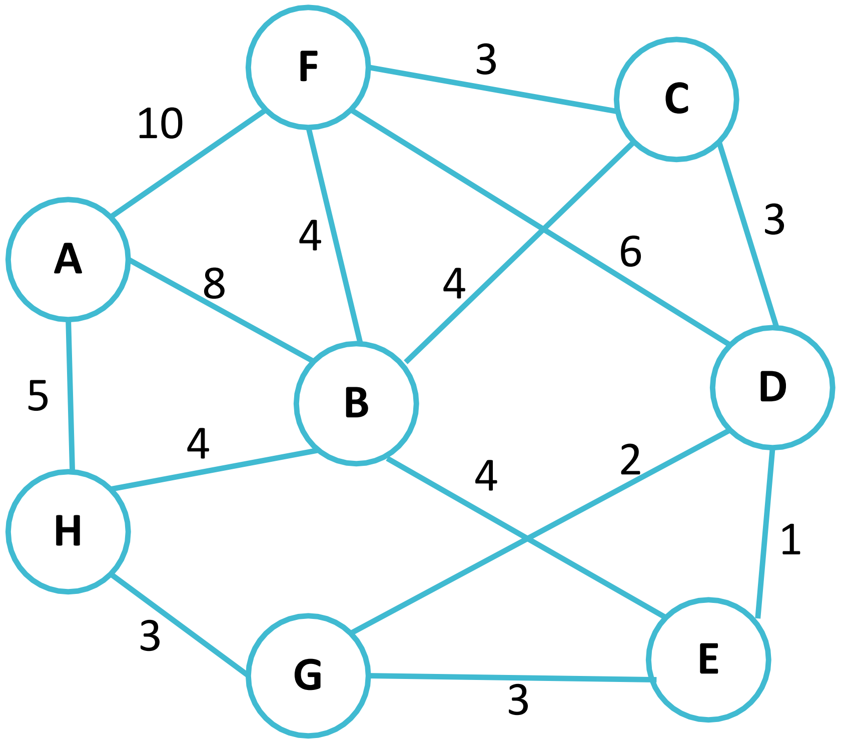
First |V|–1 edges selected
Total Cost = 21

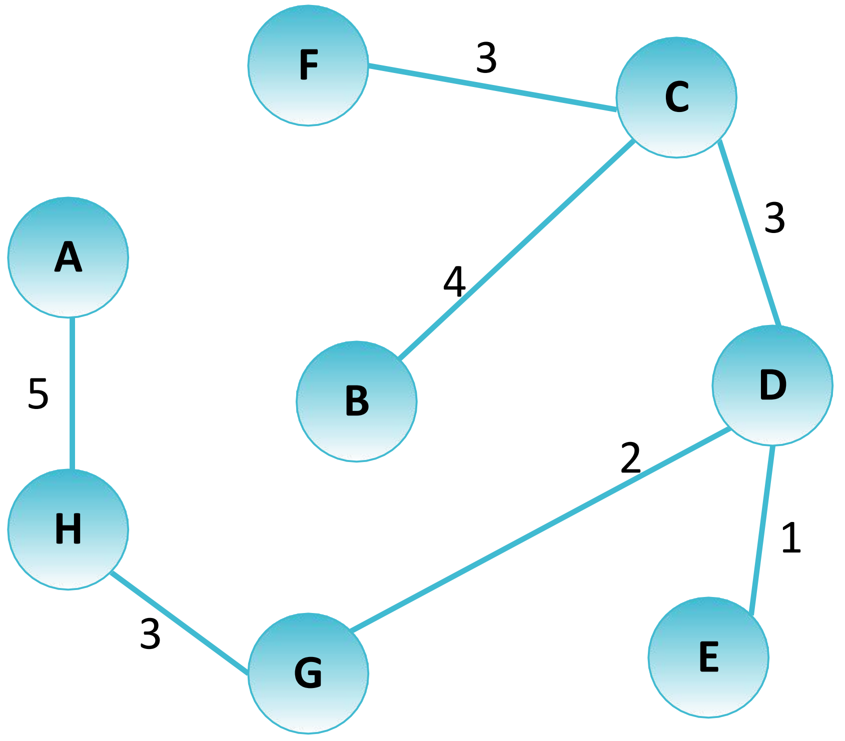


S : { (D,E),(D,G),(C,D),(G,H),(C,F),(B,C),(A,H) }

Edge	Ew	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	X
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√
(B,E)	4	X
(B,F)	4	X
(B,H)	4	X
(A,H)	5	√
(D,F)	6	-
(A,B)	8	-
(A,F)	10	-

Kruskal's Algorithm: Solution





Kruskal's Algorithm

1. Create a forest in such a way that every vertex is a separate tree.
2. Create a priority queue Q that contains all the edges of the graph.
3. Repeat steps 4 and 5 while Q is not empty
4. Remove and edge from Q
5. IF the edge obtained in step 4 connects two different trees, then add it to the forest (combining two trees to one tree) ELSE discard the edge.
6. End.

Time complexity & Applications of Kruskal's Algorithm

- **Time Complexity**
- The time complexity Of Kruskal's Algorithm is: O(E log V).

- **Applications**
- In order to layout electrical wiring
- In computer network (LAN connection)

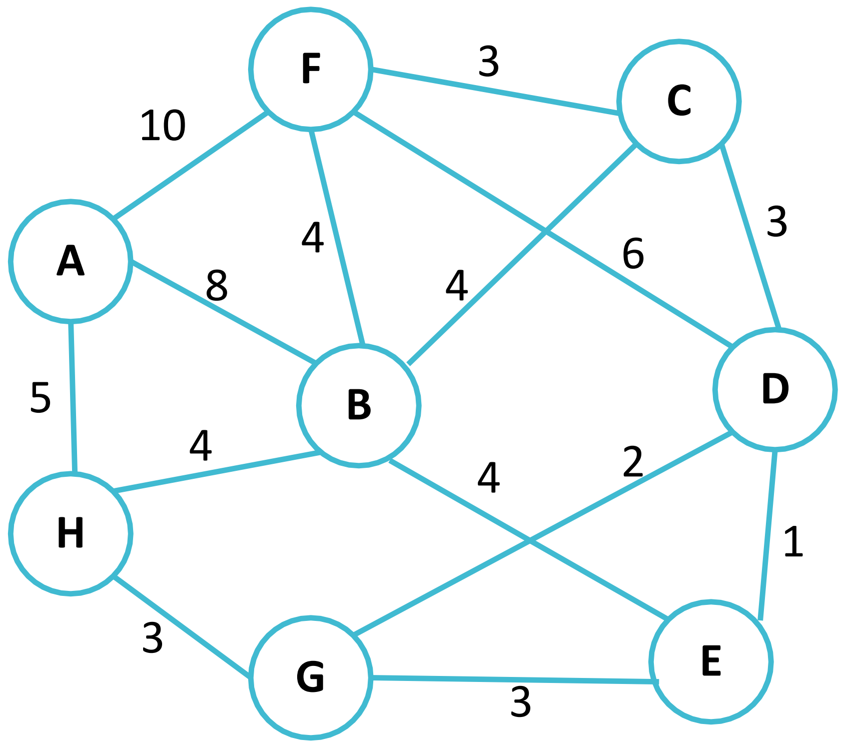
Prim's Algorithm

- Prim's Algorithm is used to find the minimum spanning tree from a graph. Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.

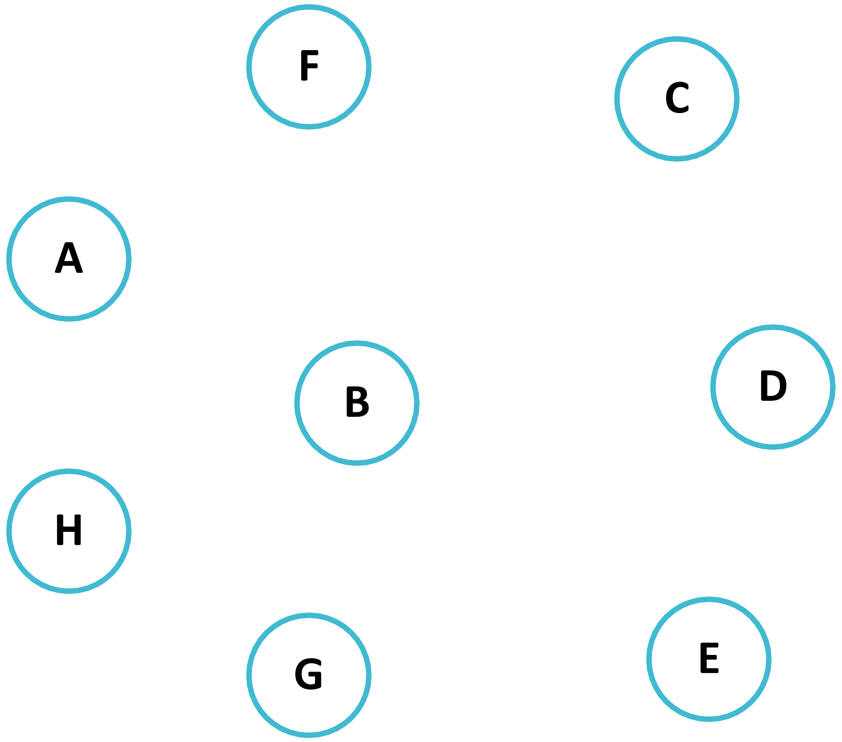
- Prim's algorithm starts with the single node and explore all the adjacent nodes with all the connecting edges at every step. The edges with the minimal weights causing no cycles in the graph got selected.
- Prim’s algorithm is significantly faster in the limit when you’ve got a really dense graph (many more edges than vertices)
- Prim’s algorithm uses adjacency matrix, binary heap or Fibonacci heap.

Prim's Algorithm

- Problem: Find Minimum Spanning Tree of a Graph, Find Optimal path from D to A



Prim's Algorithm: Solution Steps

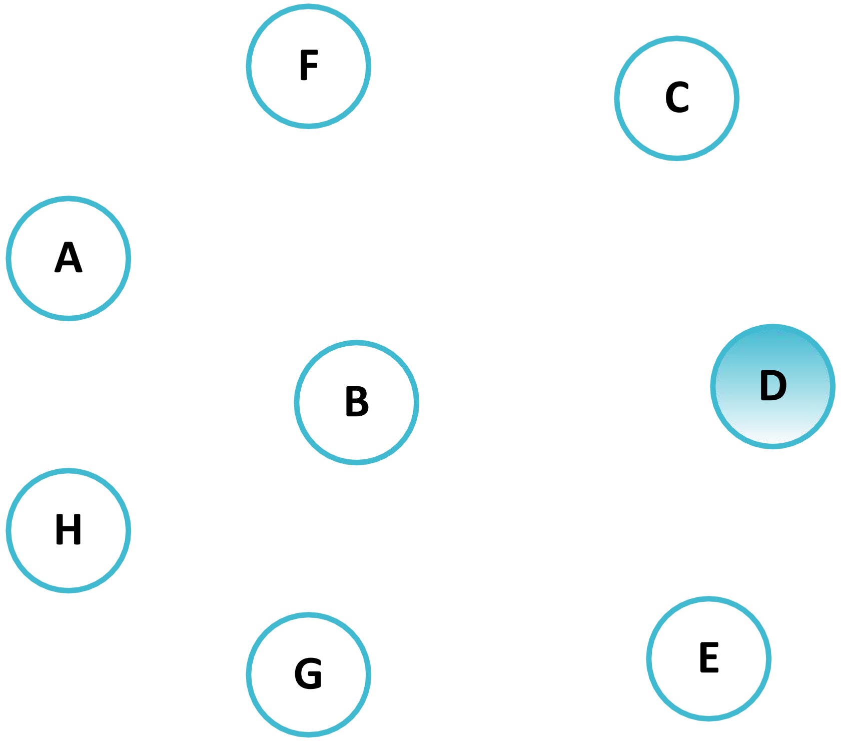


Initialize

	Vs	Dv	Pv
A	-	∞	-
B	-	∞	-
C	-	∞	-
D	-	∞	-
E	-	∞	-
F	-	∞	-
G	-	∞	-
H	-	∞	-

Prim's Algorithm: Solution Steps

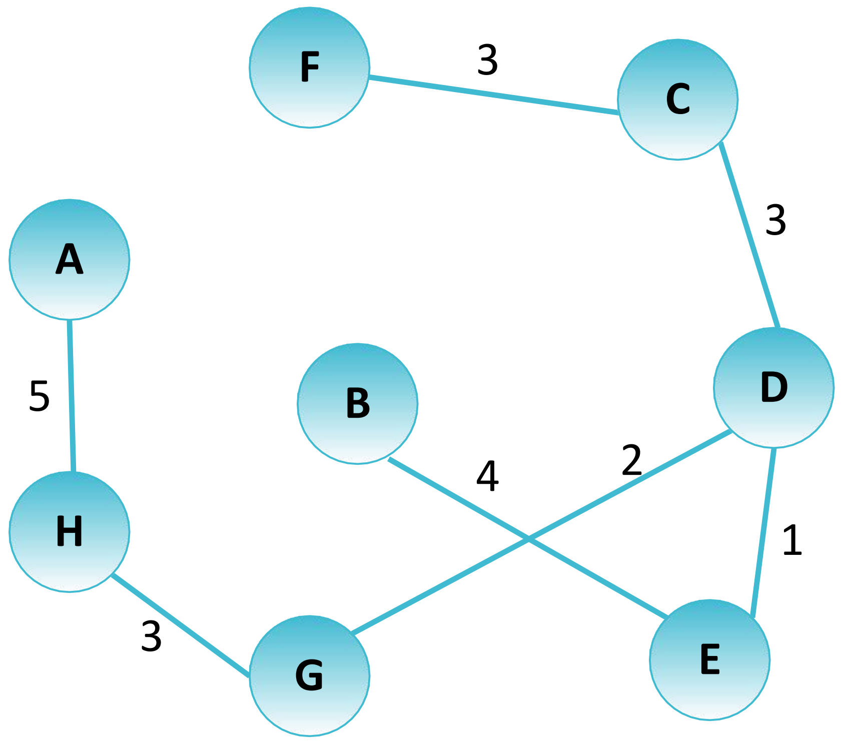
Start with node
Update distances of adjacent, unselected nodes
Select node with minimum distance



	Vs	Dv	Pv
A	u	∞	-
B	u	∞	-
C	u	∞	-
D	T	0	-
E	u	∞	-
F	u	∞	-
G	u	∞	-
H	u	∞	-

u : Unseen vertex
f : Fringe vertex
T : Tree vertex

Prim's Algorithm: Solution

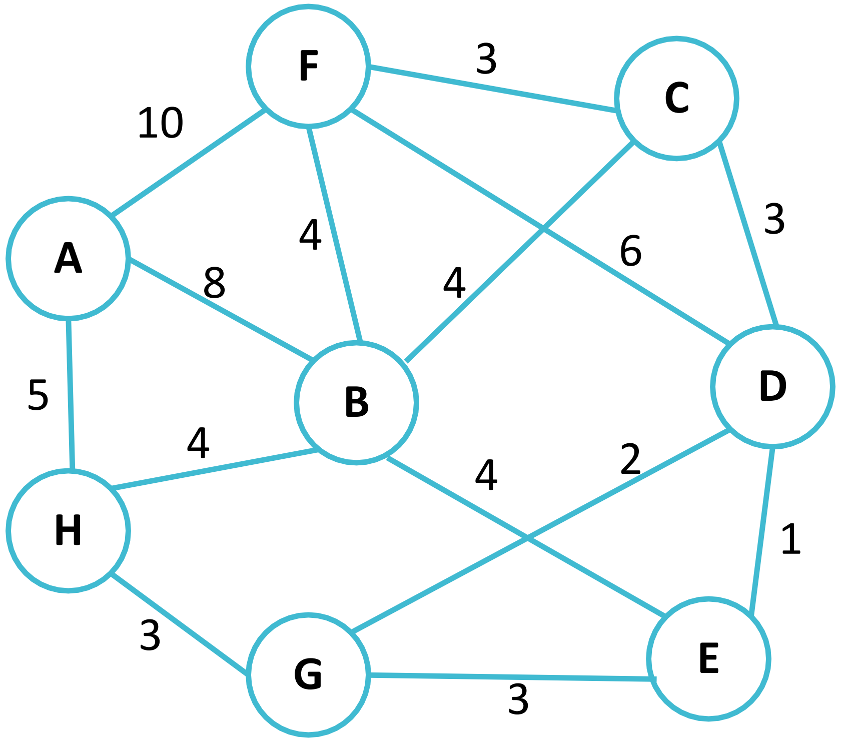


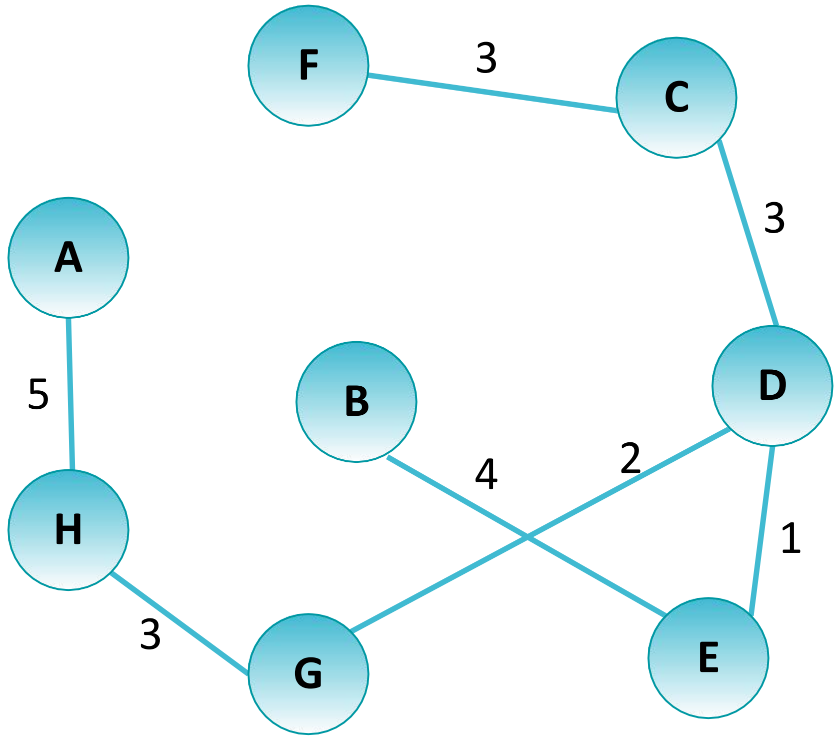
Cost of Minimum Spanning Tree
$\Sigma Dv = 21$

	Vs	Dv	Pv
A	T	5	H
B	T	4	E
C	T	3	D
D	T	0	-
E	T	1	D
F	T	3	C
G	T	2	D
H	T	3	G

Path From D → A:
D→G→H→A

Prim's Algorithm: Solution





Prim's Algorithm

1. Select a starting vertex.
2. Repeat steps 3 and 4 until there are fringe vertices.
3. Select an edge e connecting the tree vertex and fringe vertex that has minimum weight.
4. Add the selected edge and the vertex to the minimum spanning tree T.
5. End.

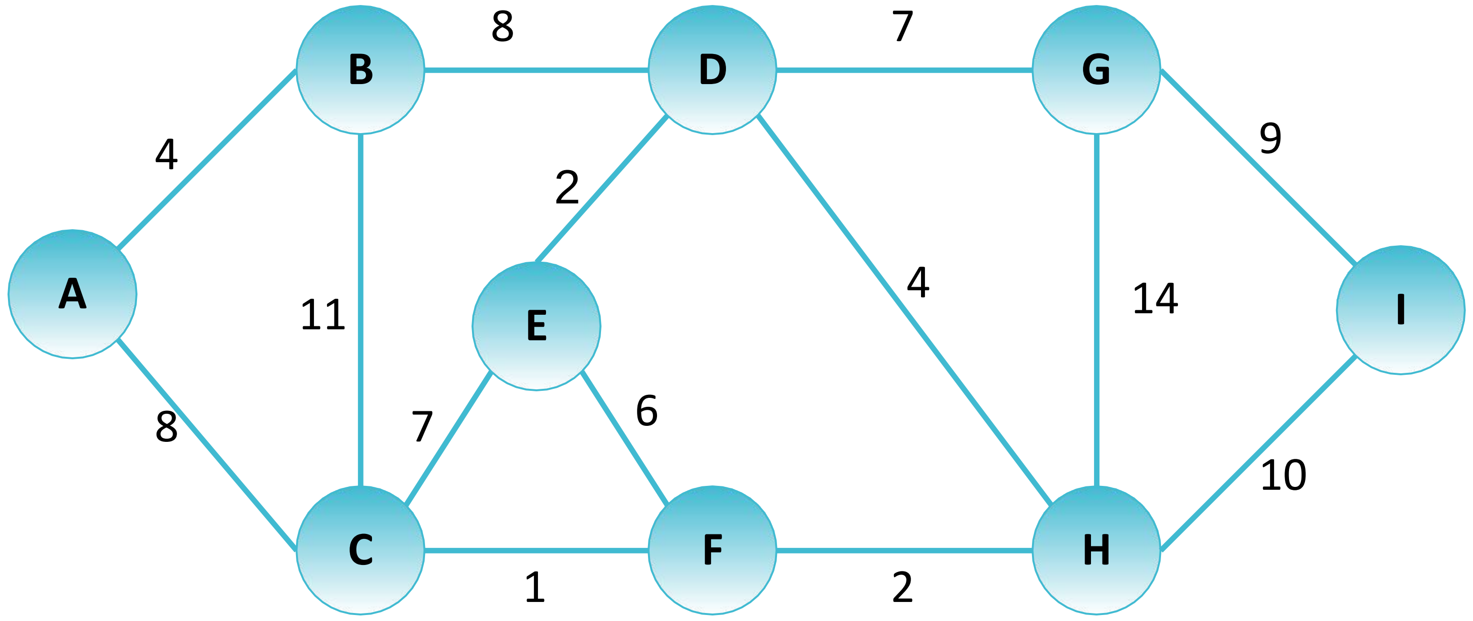
Time complexity & Applications of Prims Algorithm

- **Time Complexity**
- The time complexity Of Kruskal's Algorithm is: O(E+V.log(V)).

- **Applications**
- Path finding
- Travelling salesman problem
- Cluster analysis
- Network for roads and rail tracks

Example

- Given Graph



Algorithms

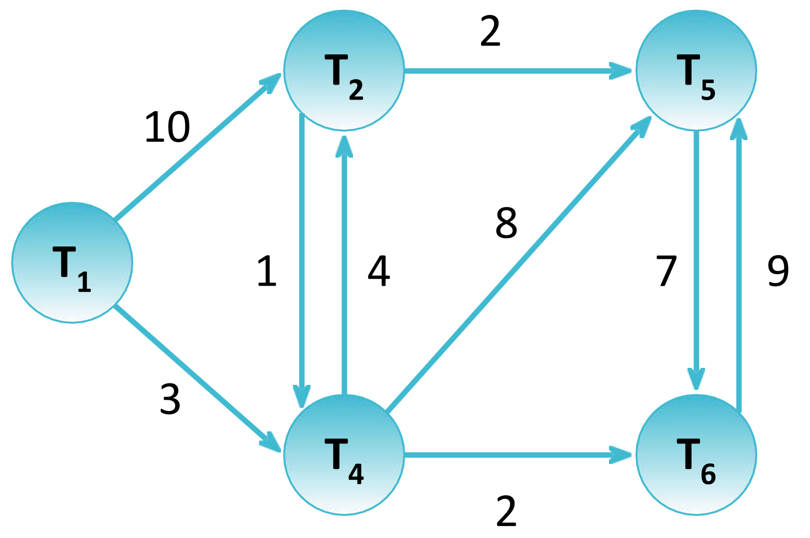
- Dijkstra's Algorithm
  - Single-source Shortest Path
- Warshall's Algorithm
  - All pair Shortest Path

Dijkstra's Algorithm

- Dijkstra's algorithm - is a solution to the single-source shortest path problem in graph theory.
- Works on both directed and undirected graphs.
- However, all edges must have nonnegative weights.
- It works only for connected graphs
- The algorithm predominantly follows Greedy approach for finding locally optimal solution. But, it also uses Dynamic Programming approach for building globally optimal solution, since the previous solutions are stored and further added to get final distances from the source vertex.
- Relaxation: If (distance(u) + cost(u,v) < distance(v)) then
- distance (v) = distance (u) + cost (u,v)

Dijkstra's Algorithm

- Problem:
- Single-Source Shortest Path Problem
- The problem of finding shortest paths from a source vertex T_1 to all other vertices in the graph.



Dijkstra's Algorithm

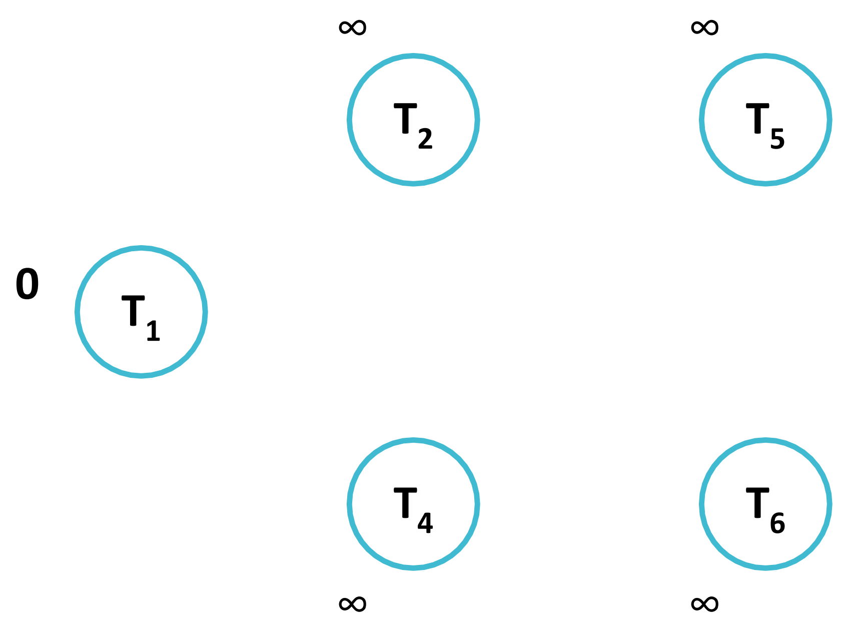
```
Step1: Distance to source vertex is zero
Step2: Set all other distances to infinity
Step3: S, the set of visited vertices is initially empty
Step4: Q, the queue initially contains all vertices and initial
       distances
Step5: Repeat step 6 to 8 while the queue Q is not empty
Step6: Select the element of Q with the min. distance: Ti
Step7: Add Ti to list of visited vertices S.
Step8: For all V ∈ neighbors[Ti]
       Check dist[V] > dist[Ti] + w(Ti, V)
       if new shortest path found
       set new value of shortest path
```

Dijkstra's Algorithm: Solution Steps

Initialize

Q:

T_1	T_2	T_4	T_5	T_6
0	∞	∞	∞	∞

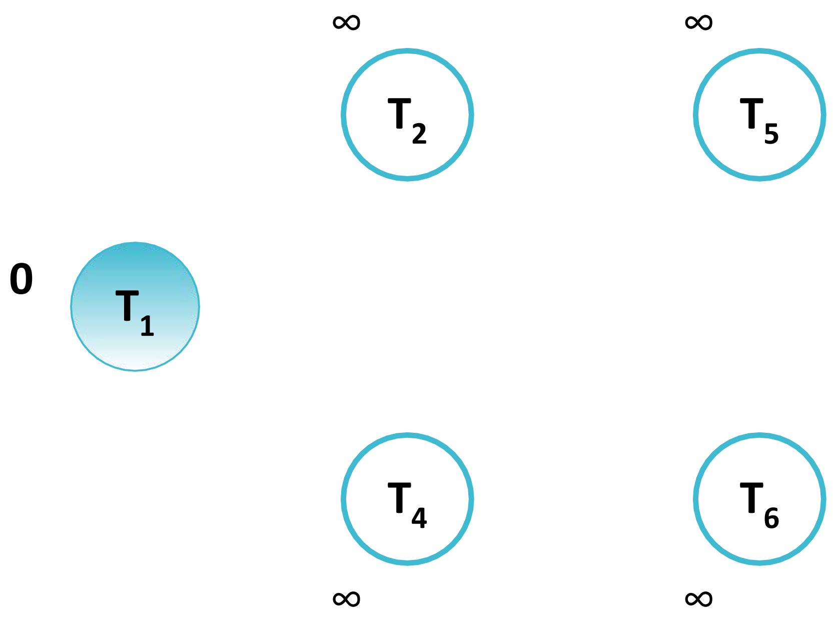


S: { }

Dijkstra's Algorithm: Solution Steps

Q:

T_1	T_2	T_4	T_5	T_6
0	∞	∞	∞	∞



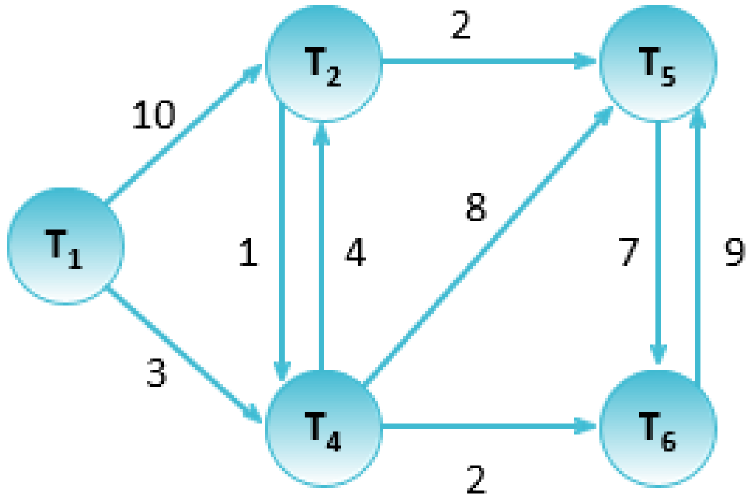
S: {T_1 }

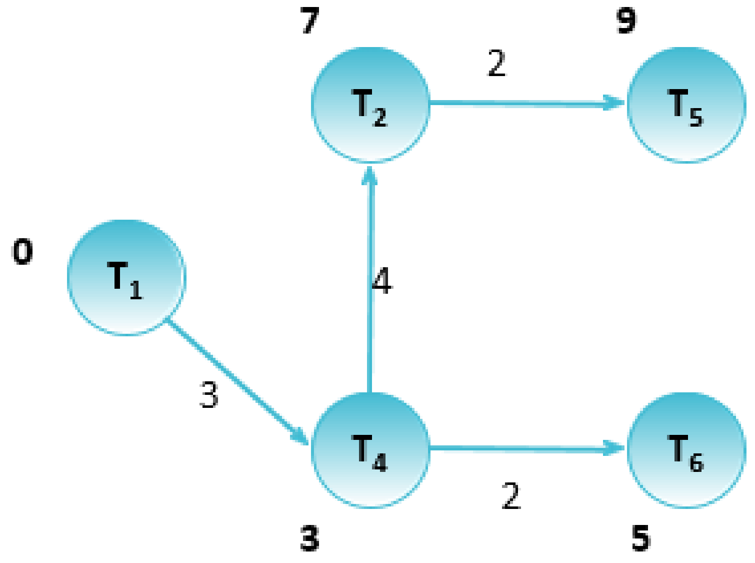
Dijkstra's Algorithm: Solution

Q:

T_1	T_2	T_4	T_5	T_6
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	
			9	

S: {T_1 , T_4 , T_6 , T_2 , T_5}





Time and Space Complexity of Dijkstra's Algorithm

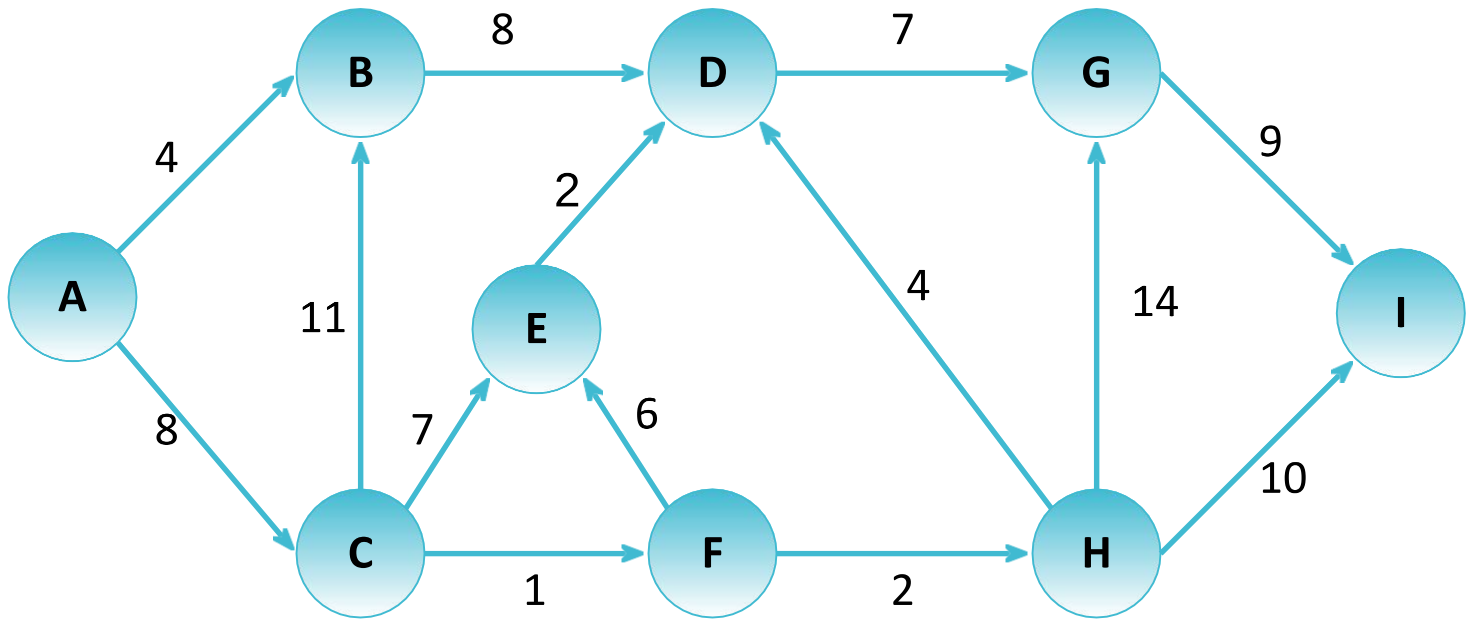
- Time Complexity: O(E Log V)
- where, E is the number of edges and V is the number of vertices.
- Space Complexity: O(V)

Applications of Dijkstra's Algorithm

- To find the shortest path
- In social networking applications
- In a telephone network
- To find the locations in the map

Example

- Given Graph



Path Matrix

- A path matrix is a matrix representing a graph where each value in m’th row and n’th column project whether there is a path from m to n.
- The path may be direct or indirect. It may have a single edge or multiple edge.

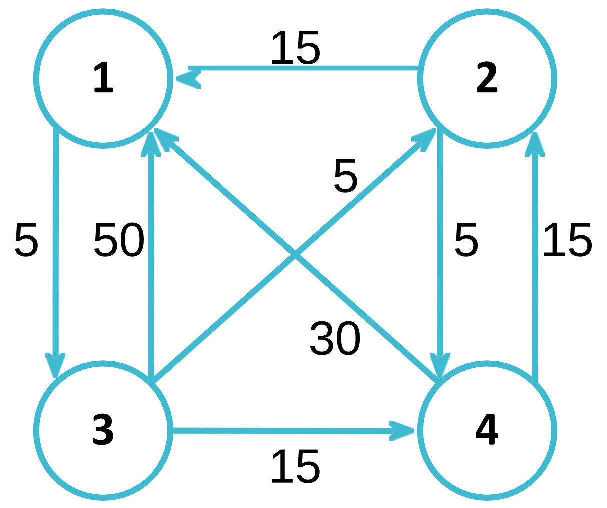
k

Path Matrix

- $G = (V,E)$
- A : adjacency matrix
- $P = A + A2 + A3 + \dots$

- P0,P1,P2, … Pn:
  - $P_k[i][j] = 1$ { if there is a path from v_i to v_j, path should not use any other vertex except $v_1, v_2, \dots, v_k$}
  - $P_k[i][j] = 0$ {Otherwise}

Path Matrix



Warshall's Algorithm

- Floyd–Warshall algorithm is an algorithm for finding the shortest paths in a weighted graph with positive or negative edge weights (but with no negative cycles).

- It does so by comparing all possible paths through the graph between each pair of vertices and that too with $O(V^3)$ comparisons in a graph.

- $P_k[i][j] = 1$
  - iff $P_{k-1}[i][j] = 1$
  - or $P_{k-1}[i][k] = 1$ and $P_{k-1}[k][j] = 1$

- $P_k[i][j] = P_{k-1}[i][j] \vee (P_{k-1}[i][k] \wedge P_{k-1}[k][j])$

Warshall's Algorithm

For a graph with N vertices:

Step 1: Initialize the shortest paths between any 2 vertices with Infinity.

Step 2: Find all pair shortest paths that use 0 intermediate vertices, then find the shortest paths that use 1 intermediate vertex and so on.. until using all N vertices as intermediate nodes.

Step 3: Minimize the shortest paths between any 2 pairs in the previous operation.

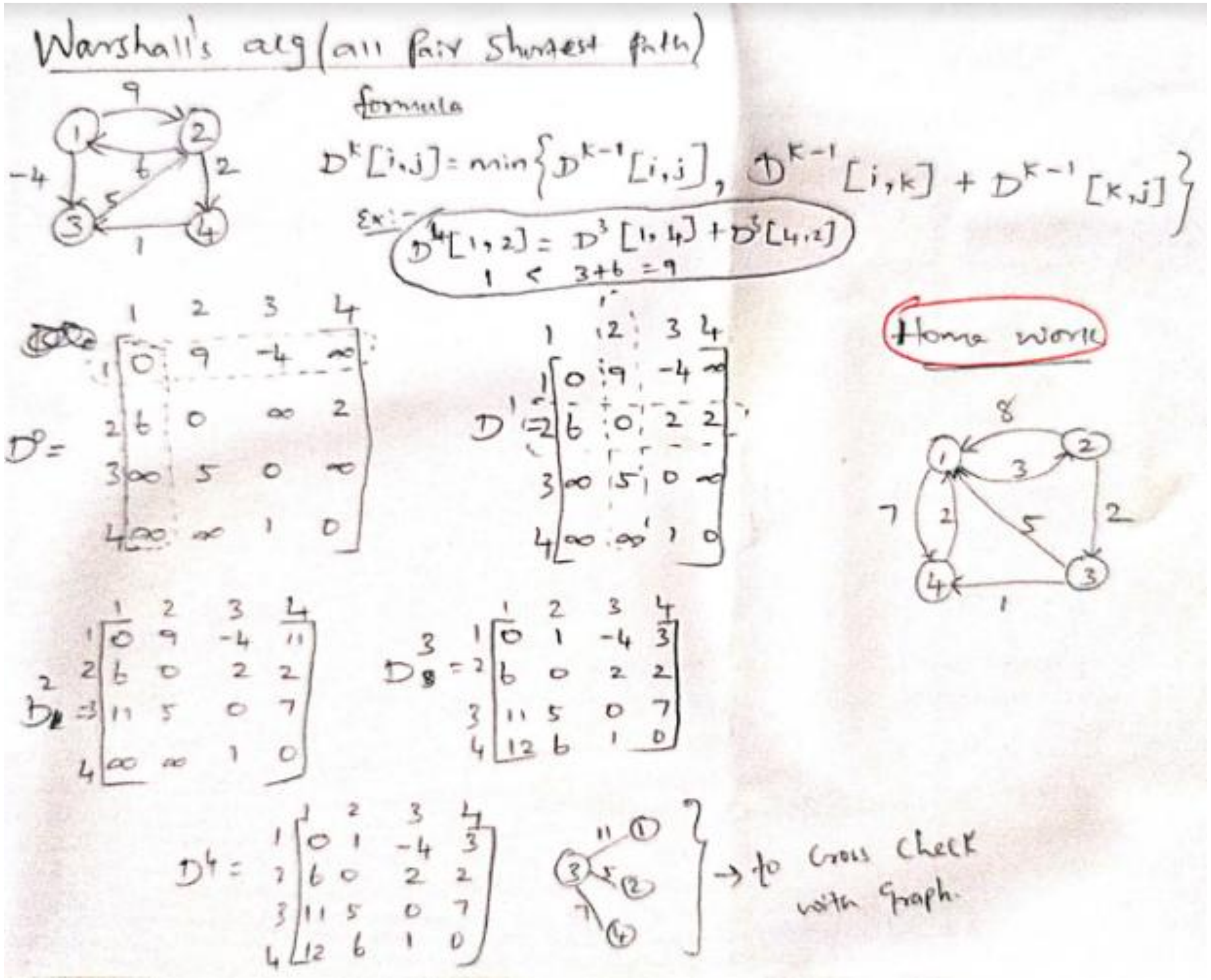
Step 4: For any 2 vertices (i,j) , one should actually minimize the distances between this pair using the first K nodes, so the shortest path will be: min(dist[i][k]+dist[k][j],dist[i][j]).

dist[i][k] represents the shortest path that only uses the first K vertices, dist[k][j] represents the shortest path between the pair k,j. As the shortest path will be a concatenation of the shortest path from i to k, then from k to j.

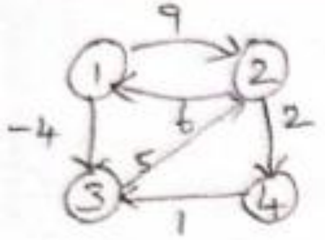
Warshall's Algorithm Pseudocode

```
Floyd-Warshall(w, n) // w: weights, n: number of vertices
{
   for i = 1 to n do      // initialize, D (0) = [wij]
     for j = 1 to n do
     {
       d[i, j] = w[i, j];
      }
   for k = 1 to n do      // Compute D (k) from D (k-1)
     for i = 1 to n do
      for j = 1 to n do
        if (d[i, k] + d[k, j] < d[i, j])
        {
          d[i, j] = d[i, k] + d[k, j];
        }
  return d[1..n, 1..n];
 }
```

Warshall's Algorithm - Example



Warshall's alg (all pair shortest path)



formula

$$D^k[i,j] = \min\{D^{k-1}[i,j],\ D^{k-1}[i,k] + D^{k-1}[k,j]\}$$

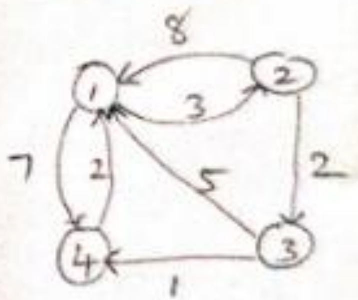
Ex:- $D^4[1,2] = D^3[1,4] + D^3[4,2]$

$1 < 3+6 = 9$

$$D^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 9 & -4 & \infty \\ 2 & 6 & 0 & \infty & 2 \\ 3 & \infty & 5 & 0 & \infty \\ 4 & \infty & \infty & 1 & 0 \end{array}$$

$$D^1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 9 & -4 & \infty \\ 2 & 6 & 0 & 2 & 2 \\ 3 & \infty & 5 & 0 & \infty \\ 4 & \infty & \infty & 1 & 0 \end{array}$$

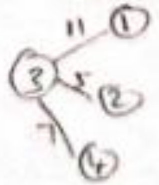
Home work



$$D^2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 9 & -4 & 11 \\ 2 & 6 & 0 & 2 & 2 \\ 3 & 11 & 5 & 0 & 7 \\ 4 & \infty & \infty & 1 & 0 \end{array}$$

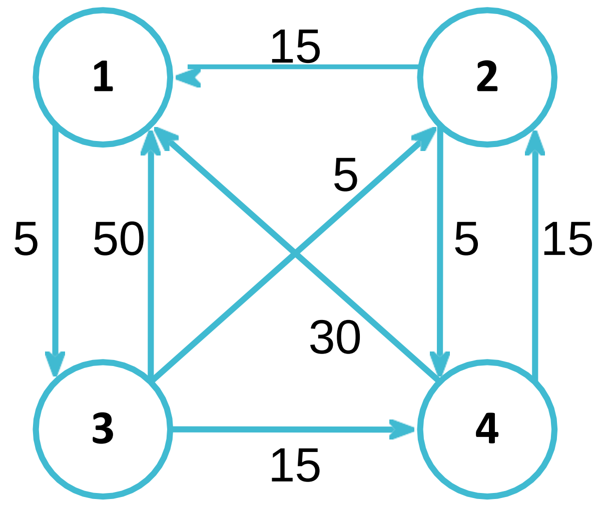
$$D^3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & -4 & 3 \\ 2 & 6 & 0 & 2 & 2 \\ 3 & 11 & 5 & 0 & 7 \\ 4 & 12 & 6 & 1 & 0 \end{array}$$

$$D^4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & -4 & 3 \\ 2 & 6 & 0 & 2 & 2 \\ 3 & 11 & 5 & 0 & 7 \\ 4 & 12 & 6 & 1 & 0 \end{array}$$



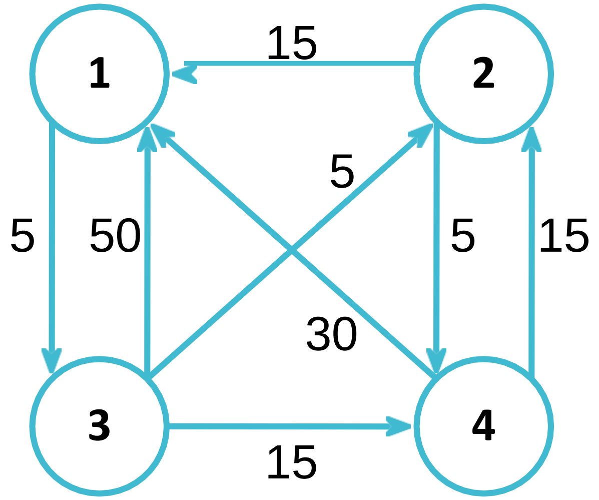
→ to Cross check with graph.

Warshall's Algorithm



$$\begin{array}{c|cccc} P & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 1 & 1 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 1 & 1 & 0 & 1 \\ 4 & 1 & 1 & 1 & 0 \end{array}$$

Modified Warshall's Algorithm



A	1	2	3	4
1	0	∞	5	∞
2	15	0	∞	5
3	50	5	0	15
4	30	15	∞	0

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

D	1	2	3	4
1	0	10	5	15
2	15	0	20	5
3	20	5	0	10
4	30	15	35	0

Modified Warshall's Algorithm

- If D_k represents the matrix D after the kth iteration , it can be implemented by:

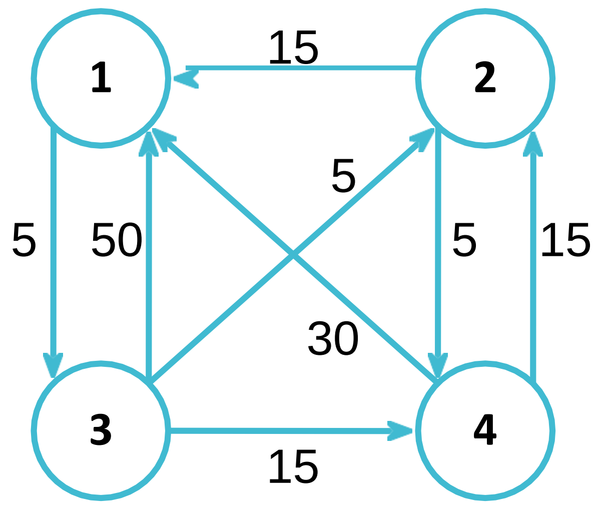
$$D_k[i, j] = \min (D_{k-1}[i, j], D_{k-1}[i, k]+D_{k-1}[k, j])$$

- We also use a second matrix P, for path, P[i, j] contains the number of the last iteration that caused a change in D[i, j].

Modified Warshall's Algorithm

```
Function Warshall(L[1…n, 1…n])
array D[1…n, 1…n], P[1…n, 1…n]
P □   0
D □   L
for k □   1 to n do
    for i □   1 to n
    do
        for j □   1 to n do
            if (D[i, k] + D[k, j] < D[i, j])
            then D[i, j] □   D[i, k] + D[k,
                 j] P[i, j] □   k
```

Modified Warshall's Algorithm



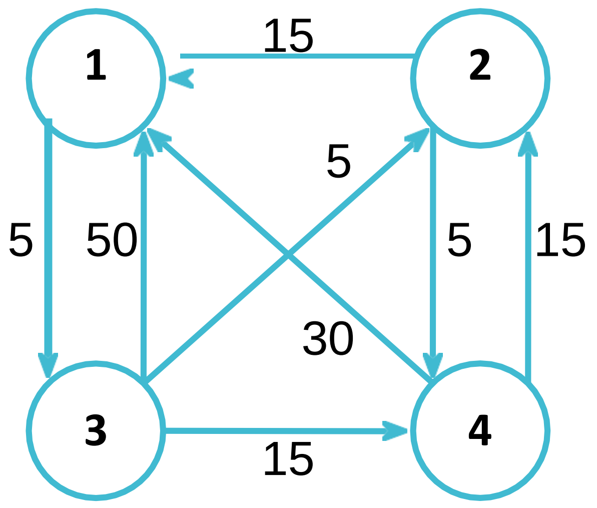
D_0	1	2	3	4
1	0	∞	5	∞
2	15	0	∞	5
3	50	5	0	15
4	30	15	∞	0

D_1	1	2	3	4
1				
2				
3				
4				

$$D_k[i, j] = \min(D_{k-1}[i, j], D_{k-1}[i, k]+D_{k-1}[k, j])$$

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

Modified Warshall's Algorithm



$$D_4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 10 & 5 & 15 \\ 2 & 15 & 0 & 20 & 5 \\ 3 & 20 & 5 & 0 & 10 \\ 4 & 30 & 15 & 35 & 0 \end{array}$$

$$P = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 0 & 3 \\ 2 & 0 & 0 & 1 & 0 \\ 3 & 2 & 0 & 0 & 2 \\ 4 & 0 & 0 & 1 & 0 \end{array}$$

$$D_k[i, j] = \min(D_{k-1}[i, j], D_{k-1}[i, k]+D_{k-1}[k, j])$$

Till Now

- Trees
- **Graphs**

- Graphs and their understanding
- Matrix representations of a given graph
- Depth First Search (DFS)
- Breadth First Search (BFS)
- Minimum Spanning Trees Algorithms (Prims, Kruskal, Dijkstra)
- Path Matrix
- Warshall's Algorithm

Thank You.