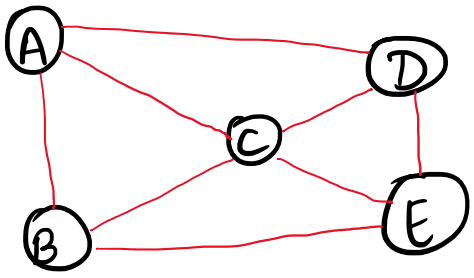
Week 4 & 5

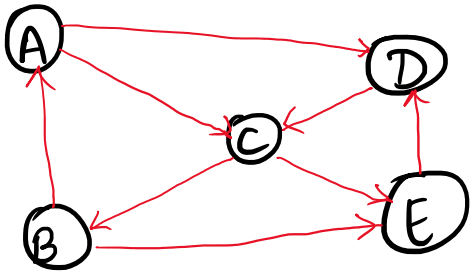
Monday, July 7, 2025 11:35 AM

WEEK - 04

Undirected Graph :



Directed Graph :



ADJACENCY MATRIX : For undirected Graph

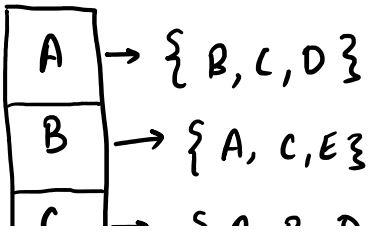
	A	B	C	D	E
A	0	1	1	1	0
B	1	0	1	0	1
C	1	1	0	1	1
D	1	0	1	0	1
E	0	1	1	1	0

For Directed Graph

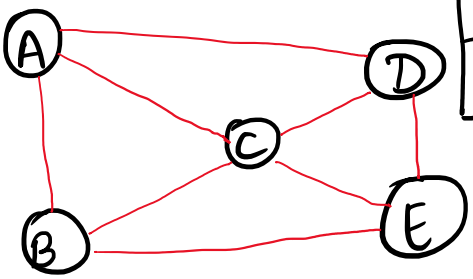
	A	B	C	D	E
A	0	0	1	1	0
B	1	0	0	0	1
C	0	1	0	0	1
D	0	0	1	0	0
E	0	0	0	1	0

ADJACENCY LIST :

for undirected

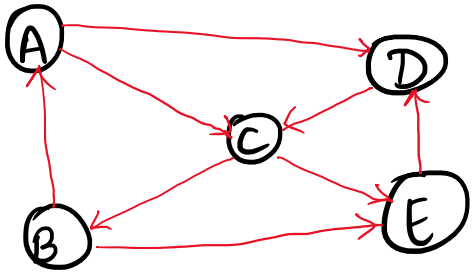


for undirected Graph



B	→ {A, C, E}
C	→ {A, B, D, E}
D	→ {A, C, E}
E	→ {B, C, D}

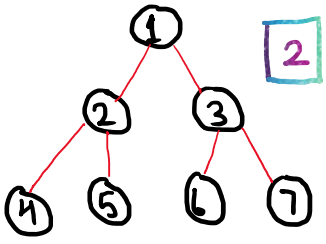
For directed Graph



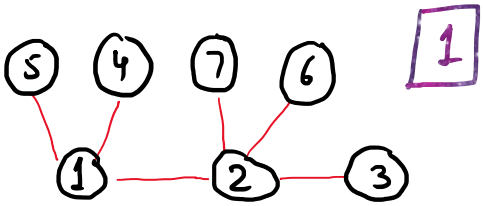
A	→ {C, D}
B	→ {A, E}
C	→ {B, E}
D	→ {C}
E	→ {D}

TRAVERSING ALGORITHMS:

Breadth first search (BFS)



1. Visit a node.
2. Explore that node.
3. then move to new node.



1 => 1, 2, 4, 5, 3, 6, 7

2 => 1, 2, 3, 4, 5, 6, 7

Depth First Search (DFS):

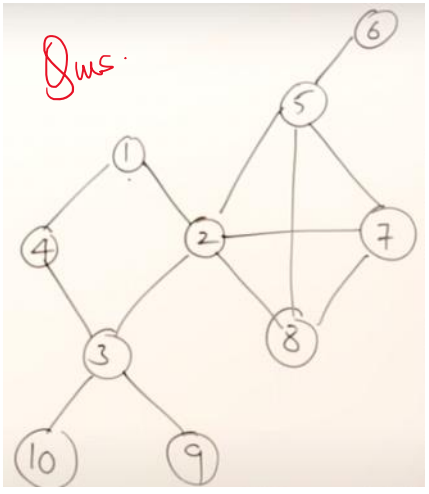
1. Pick and visit any node.
2. Move to any other node connected to it.
3. For at previous node Explore new node.

search (DFS)

3. Forgot previous node. Explore new node.
4. Repeat step 2.
5. Return to the forgoted node after new nodes end.

1 ⇒ 1, 2, 3, 6, 7, 4, 5

2 ⇒ 1, 2, 4, 5, 3, 6, 7



Queue

BFS : Q = | 1 | 2 | 4 | 3 | 5 | 7 | 8 | 9 | 10 | 6 |

BFS ⇒ {1, 2, 4, 3, 5, 7, 8, 9, 10, 6}

DFS : S (stack)

1, 2, 3, 4, 9, 10, 5, 6, 7, 8

For BFS : using Adjacency matrix ; $O(V^2)$

and using Adjacency list ; $O(V+E)$ (V: no. of vertices, E: no. of edges)

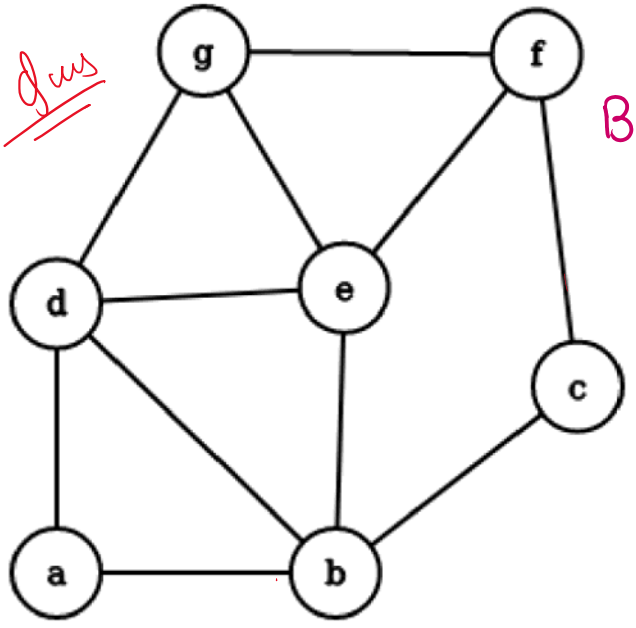
* For an undirected graph, if I have no of vertices as V then max no of edges are ${}^{V}C_2$ ⇒ $\frac{V(V-1)}{2}$.

* For a graph, if V vertices & E edges, then sum of degree of

all vertices ⇒ for undirected = 2E

⇒ for directed ⇒ in-degree = E
out-degree = E
} Total = 2E

Ques

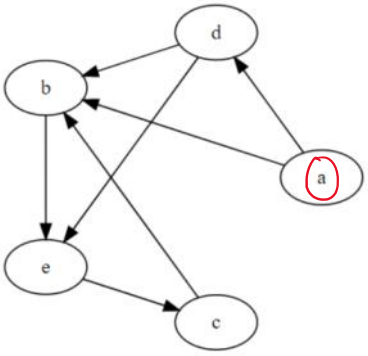


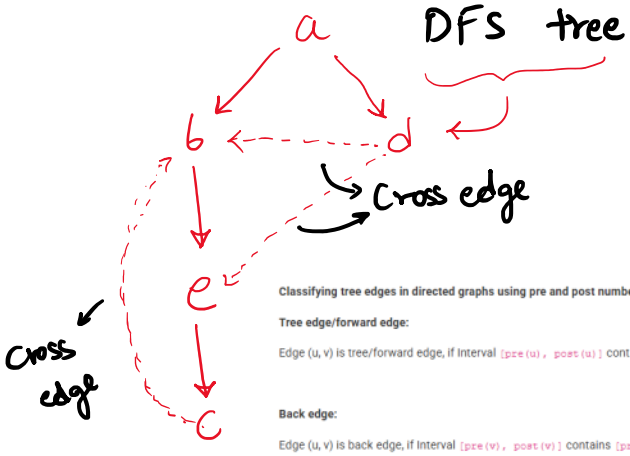
BFS ⇒ a, b, d, c, e, g, f

→ From a BFS tree to the original graph; we cannot have an edge b/w 2 levels that differ by more than 1.

DFS ⇒ a, b, c, f, e, d, g

→ From a DFS tree to Graph, we cannot have edges between two vertices which are not on "same branch".





Classifying tree edges in directed graphs using pre and post numbering ⇒Remember

Tree edge/forward edge:

Edge (u, v) is tree/forward edge, if Interval `[pre(u), post(u)]` contains `[pre(v), post(v)]`

Back edge:

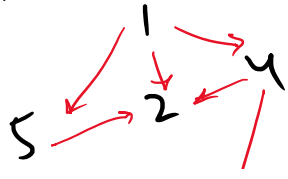
Edge (u, v) is back edge, if Interval `[pre(v), post(v)]` contains `[pre(u), post(u)]`

Back edges generate cycles

Cross edge:

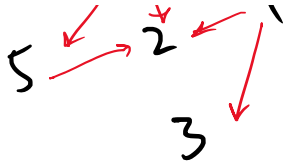
Edge (u, v) is cross edge, if Intervals `[pre(u), post(u)]` and `[pre(v), post(v)]` are disjoint

DIRECTED ACYCLIC GRAPH:



→ A directed graph.

→ No complete cycle in the graph.

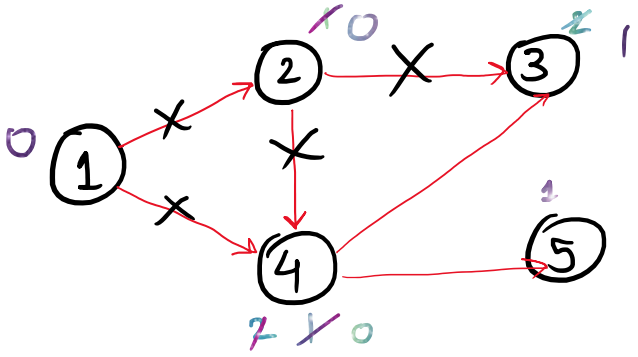


→ No complete cycle in the graph.

Topological sorting → 1. Calculate in-degree of each vertice.

2. Select and queue the vertice with in-degree = 0

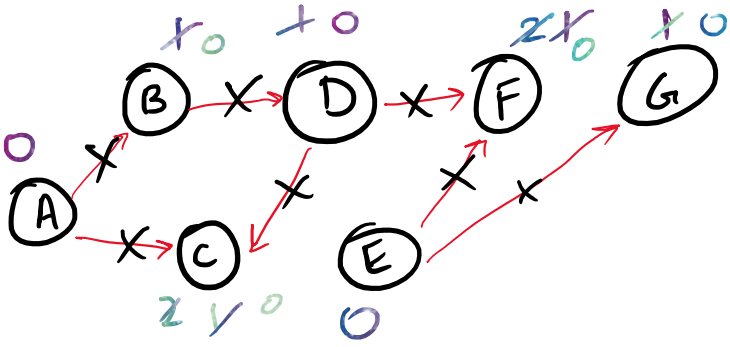
3. Remove it from the graph, repeat 1 → 3.



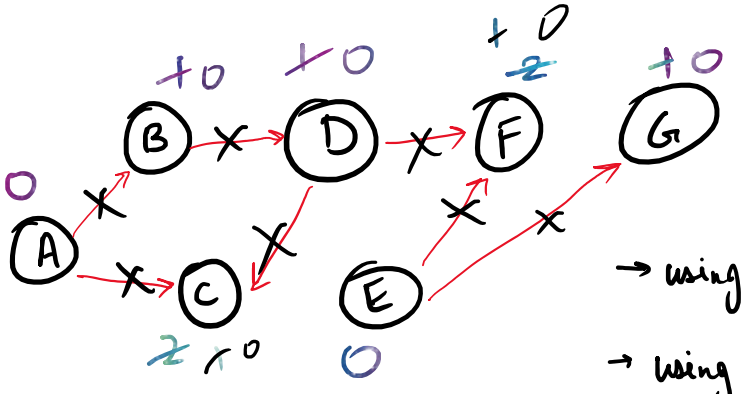
→ 1, 2, 4, 3, 5

→ 1, 2, 4, 5, 3

→ A case, where the in-degree of any vertice can be 0, is impossible in the case of DAG; not at start, not in b/w.



Case1 : A, B, D, C, E, F, G

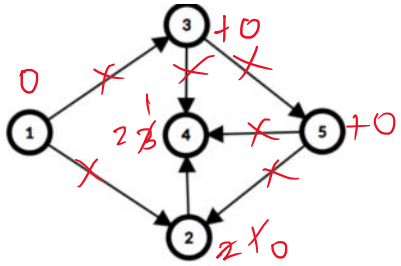


Case2 : E, G, A, B, D, C, F

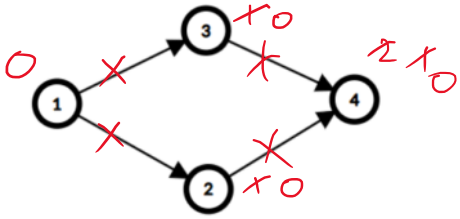
Time complexity of Topological sort →

→ using adjacency list : $O(V+E)$

→ using adjacency matrix : $O(V^2)$

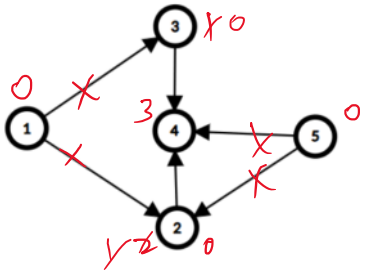


Topological sort: 1, 3, 5, 2, 4



⇒ Topological sort:

⇒ 1,2,3,4 ⇒ 1,3,2,4

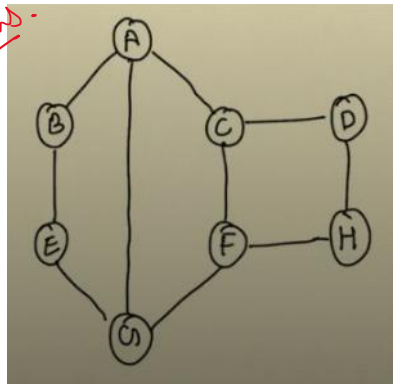


⇒ 1,3,5,2,4 ⇒ 1,5,3,2,4

⇒ 5,0 2,3,4 ⇒ 1,5, 2, 3, 4

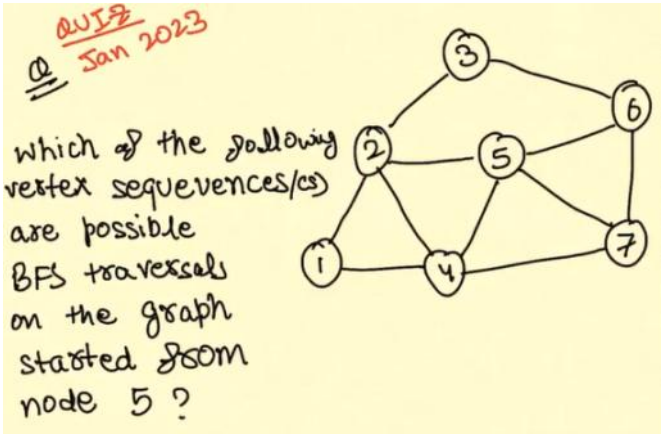
⇒ 5,0,3,2,4

5 Combinations.

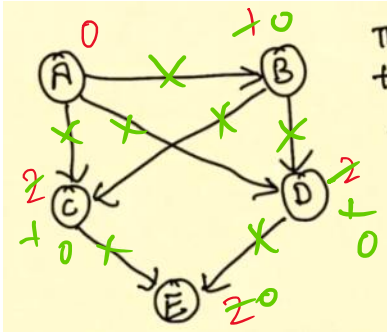


APPLY BFS from (A).

A, B, C, G, E, D, F, H



BFS ⇒ 5, 2, 4, 6, 7, 1, 3



The number of possible topological order(s) for the given graph is ____ ?

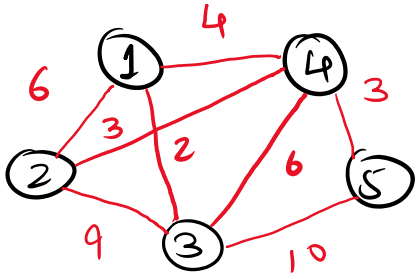
Topological order:

A, B, C, D, E
A, B, D, C, E
} 2 combinations.

WEEK-05

Weighted Graph:-

→ The edges in a weighted graph hold some weight.



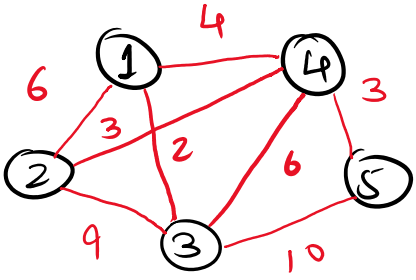
Adjacency matrix:

For undirected graph.

	1	2	3	4	5
1	0	6	2	4	0
2	6	0	9	3	0
3	2	9	0	6	10
4	4	3	6	0	3
5	0	0	10	3	0

Adjacency list:

For undirected graph.



1 → [(2,6), (3,2), (4,4)]

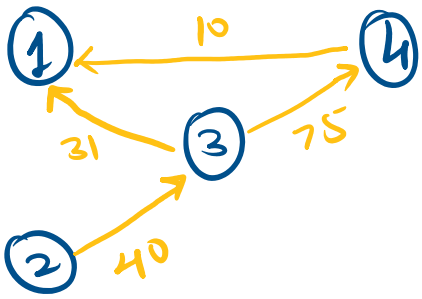
2 → [(1,6), (3,9), (4,2)]

3 → [(1,2), (2,9), (4,6), (5,10)]

4 → [(1,4), (2,2), (3,6), (5,3)]

5 → [(3,10), (4,3)]

For Directed graphs →



Adjacency Matrix →

	1	2	3	4
1	0	0	0	0
2	0	0	40	0
3	31	0	0	75
4	10	0	0	0

Adjacency List →

1 → [ ]

2 → [(3,40)]

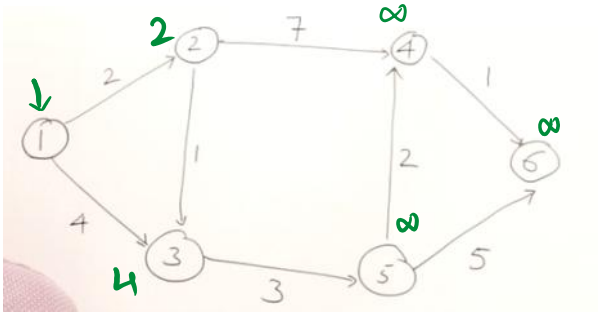
3 → [(1,31), (4,75)]

4 → [(1,10)]

SHORTEST PATH ALGORITHMS → Finding shortest path from source to any vertex.

1) Dijkstra's Algorithm : Aim is to find the shortest path to all the vertices.

→ It is a greedy approach.



→ Take a starting node. Then those connected to it and eventually the path is the associated wt.

→ Don't go back to any vertex, which is covered earlier.

→ Do Relaxation, i.e. Convert or reduce the distance wherever it is possible.

In the above figure, lets start with vertex ①.

starting vertex = 1 ⟵ selected | 2 3 4 5 6

starting vertex = 1

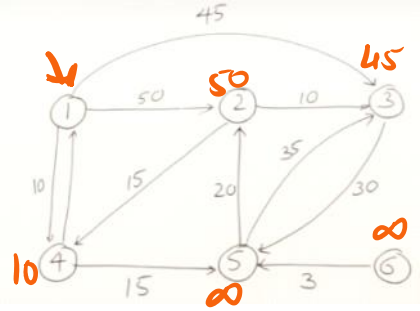
Relaxation Table

selected vertex	2	3	4	5	6
	2	4	∞	∞	∞
2	(2)	3	9	∞	∞
3	(2)	(3)	9	6	∞
5	(2)	(3)	8	(6)	11
4	(2)	(3)	(8)	(6)	9
6	(2)	(3)	(8)	(6)	(9)

⇒ Shortest Path:

1 → 2 : 2
1 → 3 : 3
1 → 4 : 8
1 → 5 : 6
1 → 6 : 9

Q.



Find shortest path to all other vertices from (1) using Dijkstra's Algorithm.

starting vertex = 1

⇒ Shortest Path = 1 → 2 45
1 → 3 45
1 → 4 10
1 → 5 25
1 → 6 ∞

⇒

selected vertex	2	3	4	5	6
	50	45	10	∞	∞
4	50	45	(10)	25	∞
5	45	45	(10)	(25)	∞
2	(45)	(45)	(10)	(25)	∞
3	(45)	(45)	(10)	(25)	(∞)

→ Complexity of Dijkstra's Algorithm is $O(V^2) \sim O(n^2)$ (Both using Adjacency list or matrix)

→ Dijkstra's Algorithm can fail for -ve weight edges. (Drawback) (it's GREEDY).

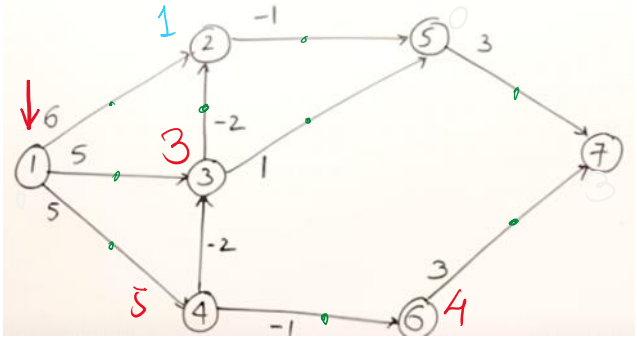
2) Bellman FORD Algorithm: For a graph with V vertices, the

2) Bellman FORD Algorithm: For a graph with V vertices, the Bellman FORD algorithm will run for V-1 iterations.

→ In this algo, we need to keep a list for all the edges in our graph, and eventually keep updating the distance associated with each vertex, by performing relaxation, so as to get the smallest distance by end of all iterations.

Consider this graph→

Apply bellman-ford on it to get the smallest distance from starting node.



Starting node is (1).

Edge list: (1,2), (1,3), (1,4), (2,5), (3,2), (3,5), (4,3), (4,6), (5,7), (6,7)

→ At start, the distance for each node is ∞

→ Go through all edges, that will be 1 iteration. Change the distances accordingly

After all iterations, these are the results that we got:

1→2	1	1→5	0
1→3	3	1→6	4
1→4	5	1→7	3

↓

(Algorithm based on Dynamic Programming)

→ For best and avg case scenario; the complexity is $O(V \times E)$ (using adjacency list) ⇒ $O(n^2)$

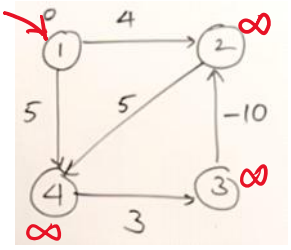
→ For worst case, that is for complete graph, complexity is

For worst case, that is for complete graph, complexity is

$\Rightarrow O(V \times E)$ $\quad$ if $E = V(V-1)$

$\Rightarrow O(V \times V \times (V-1)) \Rightarrow O(n^3)$ (using adjacency matrix)

Ques.



Edges list (1,2), (1,4), (2,4), (3,2), (4,3)

Starting vertex = 1 $\Rightarrow$ No of iterations = 4 - 1 = 3.

But we observe, that if we go, 1 more iteration, the value of weights change. (which should not happen).

Drawback of Bellman Ford Algorithm →

If there is a cycle present within the graph, such that the total weight of that cycle is -ve, the Bellman ford Algorithm fails!

3) Floyd - Warshall Algorithm : For finding all pairs of shortest path.

→ Equivalent of running Dijkstra's OR (for every pair of vertex) Bellman Ford's on each vertex.

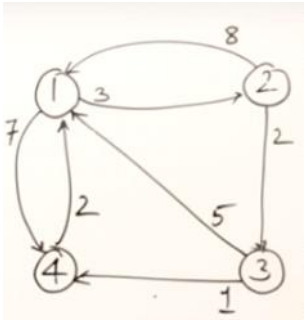
→ Follows the Dynamic Programming approach.

STEP 1. To create an adjecency matrix with weight of all connecting edges.
If there is no edge, the weight is ∞.
Call this matrix A^0.

STEP 2. Run for |V| (no. of vertices) times iteration. Update the path weights using the formula;

$$\Rightarrow \quad A^{K}[i,j] = \min\{ A^{K-1}[i,j],\ A^{K-1}[i,k] + A^{K-1}[k,j] \}$$

Ques.



Find min distance for all vertex pairs using floyd warshall algorithm.

$A^0 =$

	1	2	3	4
1	0	3	∞	7
2	8	0	2	∞
3	5	∞	0	1
4	2	∞	∞	0

Now lets compute,

$A^1 \rightarrow A^2 \rightarrow A^3 \rightarrow A^4$.

This row and column will retain from last iteration.

$A^1 =$

	1	2	3	4
1	0	3	∞	7
2	8	0	2	15
3	5	8	0	1
4	2	5	∞	0

$\Rightarrow A^2 =$

	1	2	3	4
1	0	3	5	7
2	8	0	2	15
3	5	8	0	1
4	2	5	7	0

$\Rightarrow A^3 =$

	1	2	3	4
1	0	3	5	6
2	7	0	2	3
3	5	8	0	1
4	2	5	7	0

Let's say, to find $A^1[2,3]$

I will look for the value of $A^0[2,3]$ and $A^0[2,1]+A^0[1,3]$ and take the min of these.

$\Rightarrow$ $A^0[2,3]$ will be taken.

$A^4 \Rightarrow$

	1	2	3	4
1	0	3	5	6
2	5	0	2	3
3	3	6	0	1
4	2	5	7	0

The final output matrix.
Can be used to find the distance between every pair of vertices.

→ Drawback is that it fails for the graphs having -ve weight cycles.

→ Works on both, Directed and Un-directed graphs, but must ensure

→ Works on both, Directed and Un-directed graphs, but must ensure that they have non-negative weights cycles.

⇒ Time Complexity = $O(V^3) \approx O(n^3)$

Minimum Cost Spanning Tree → for this, add the cost of all edges of the tree

- Retain a minimal set of edges so that graph remains connected
- Recall that a minimally connected graph is a tree
- Adding an edge to a tree creates a loop
- Removing an edge disconnects the graph
- Want a tree that connects all the vertices – **spanning tree**
- More than one spanning tree, in general

For a tree with n-vertices it has exactly $(n-1)$ edges.

→ In tree, every pair of vertices is connected by unique path.

⇒ Among all the spanning trees, select the one with Least COST!

for a graph with; $|V|$ vertices, $|E|$ edges and some cycles;

the total number of spanning trees can be →

⇒ $|E|C_{|V|-1}$ – no. of cycles.

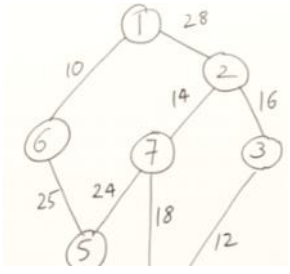
→ Spanning trees cannot be found for disconnected graphs.

PRIM'S ALGORITHM: Follows greedy approach.

(1). Start from the least weighted edge.

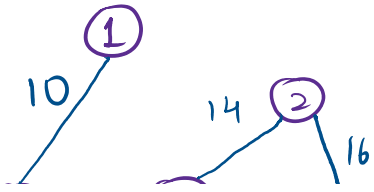
(2). Follow up with connecting with the next vertex using the least weighted edge arising from the vertex of previous edge.

Ques.



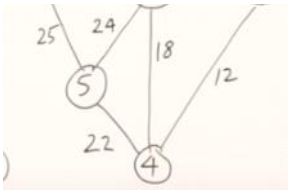
Find the min cost spanning tree for this weighted graph.

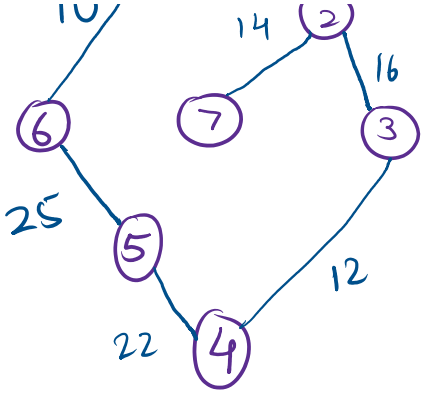
→



⇒ 7 vertices ∴ 6 edges.

Solution MCST.



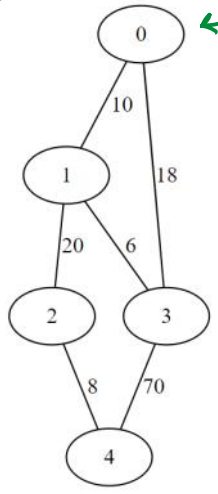


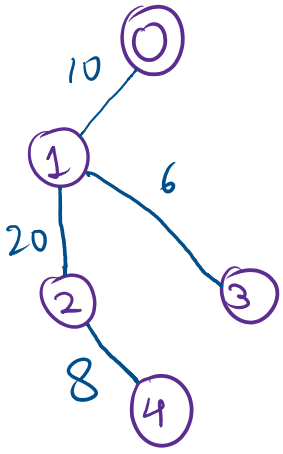
Solution MCST.

Cost = 99 !

→ It is a greedy algorithm. → For weighted undirected graphs

Find MCST for this graph!





⇒ 5 vertices

→ 4 edges!

⇒ cost = 10+20+6+8

= 44.

→ Complexity of Prim's Algorithm $O(V^2) \sim O(n^2)$

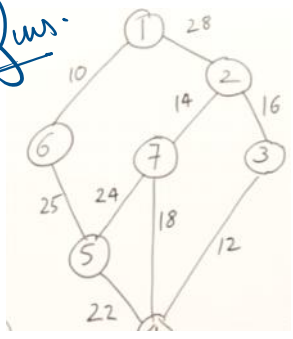
KRUSKAL's ALGORITHM:

→ Aim is to find the MCST

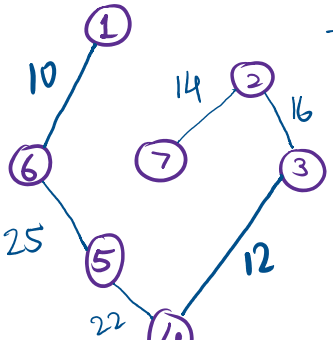
→ It is also a greedy algorithm. → For weighted, undirected graphs.

→ Approach is to iteratively add all the edges with the least weight into the graph such that no cycle is formed.

Qus.



→ Find MCST using Kruskal's Algorithm.

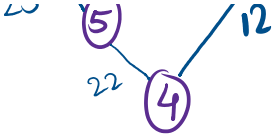


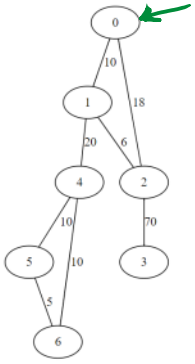
→ 7 vertices

⇒ 6 edges.

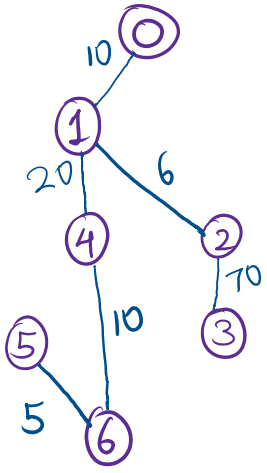
Cost = 44

Algorithm.



Ques. 

Implement Kruskal's Algorithm to find the MCST!



→ 7 vertices

⇒ 6 edges.

Cost = 121.

→ Complexity of Kruskal's Algorithm is; $O(V^2) \sim O(n^2)$.