



Data Structure and Algorithms

Session-27

Dr. Subhra Rani Patra
SCOPE, VIT Chennai

Disjoint Set

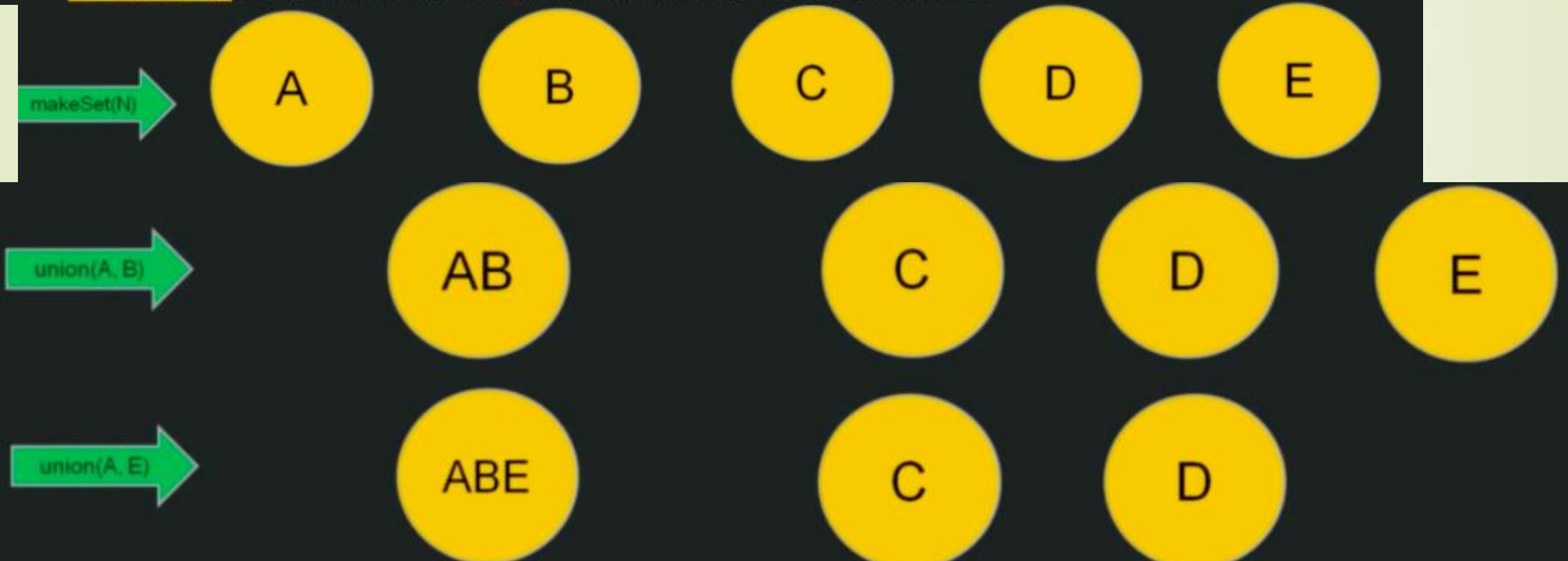
What & Why of Disjoint Set:

✓ What is it ?

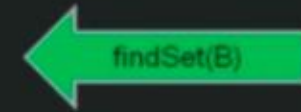
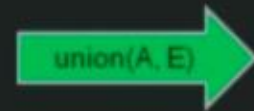
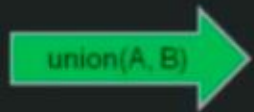
- ✓ It is a Data structure that keeps track of 'set of elements' which are partitioned into a number of disjoint and non-overlapping sets.
- ✓ Each set has a 'representative', which helps in identifying that set.

Standard operations of Disjoint Set:

- ✓ makeSet(N): used only once to create initial set
- ✓ union(x,y): used to merge both the sets.
- ✓ findSet(x): returns the set name in which this element is there.



✓ findSet(x): returns the set name in which this element is there.



makeSet (N):

create n sets for 'N' elements

findSet (x):

return representative of set in which element 'x' is there

union (s1,s2):

If s1 and s2 are in same set, then return

else

if s1 is bigger, then merge s2 into s1

else merge s2 into s1

return merged set

What & Why of Minimum Spanning Tree(MST) ?

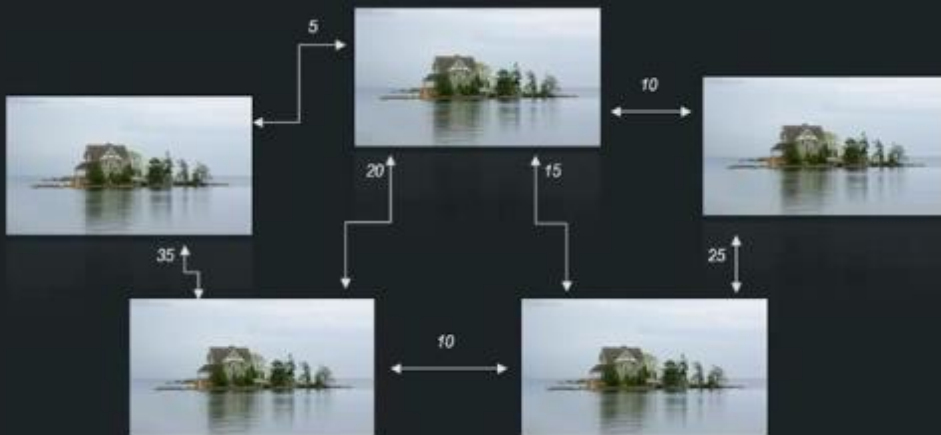
✓ Definition:

- ✓ A minimum spanning tree (MST) is a subset of the edges of a connected, weighted, undirected graph that::
 - ✓ Connects all the vertices together
 - ✓ Without any cycles

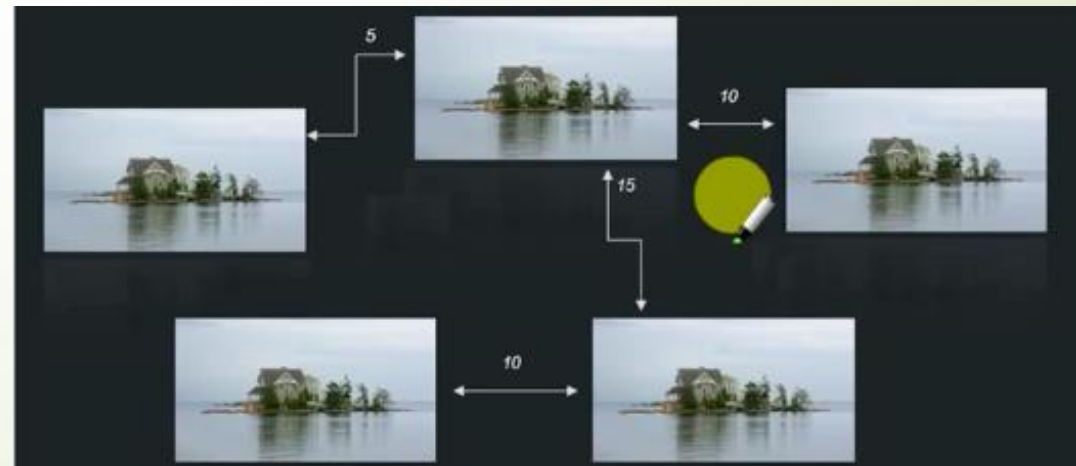
✓ Example:

- ✓ Let's say we want to connect 5 islands using bridges.
- ✓ Cost of each bridge varies depending on different factors.
- ✓ Which bridge to be constructed and which not ? How to find ?

Problem:



Minimum Spanning Tree:





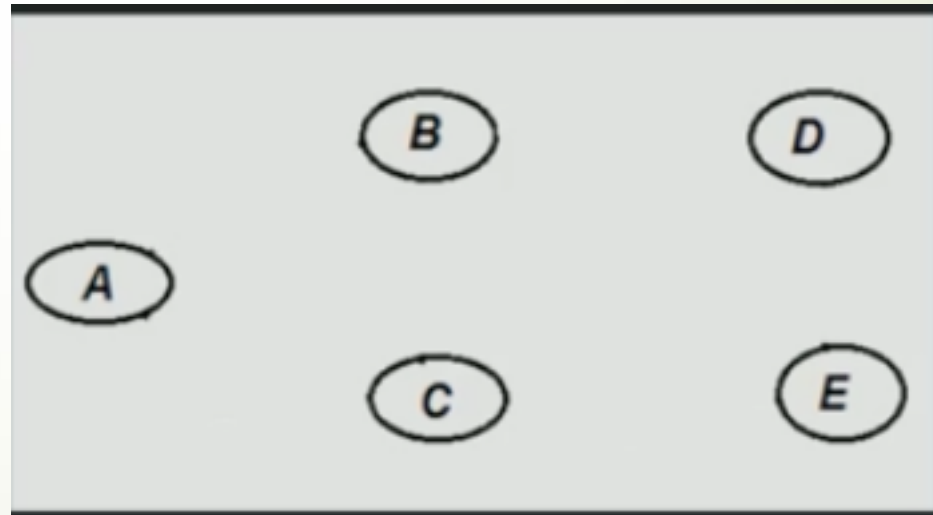
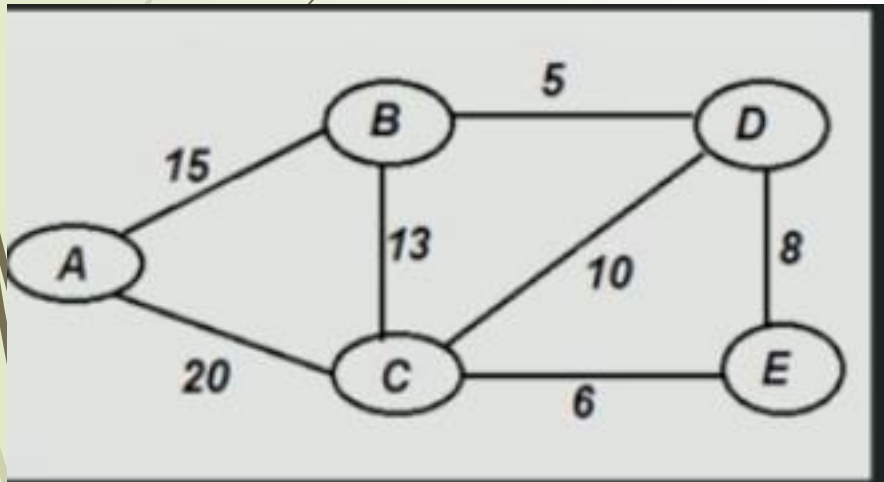
'Minimum Spanning Tree' Algorithms:

✓ *Kruskal*

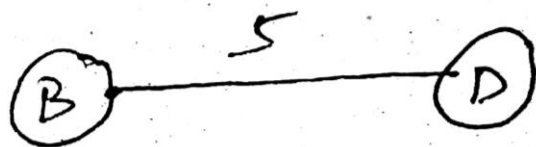
✓ *Prim's*

Kruskal Algorithm - Introduction:

- ✓ *Kruskal's algorithm is a greedy algorithm.*
- ✓ *It finds a minimum spanning tree for a connected weighted graph by:*
 - ✓ *Adding increasing cost arcs at each step*
 - ✓ *Also avoiding cycle in every step.*



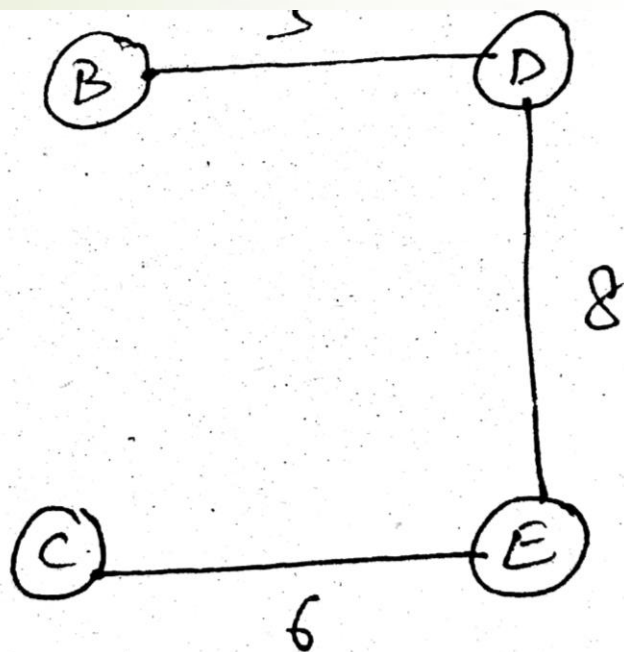
(A)



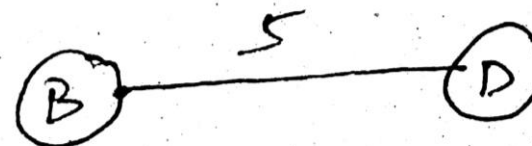
(C)

(E)

(A)

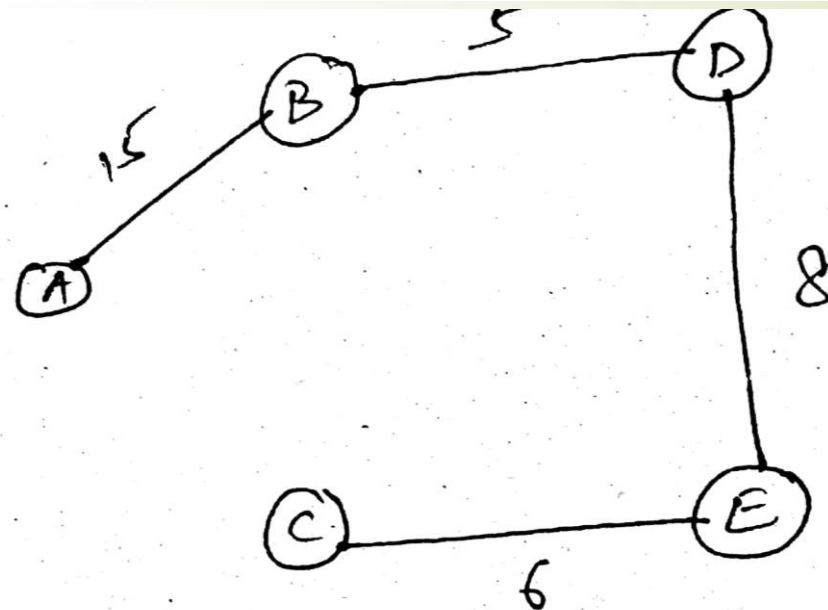


(A)

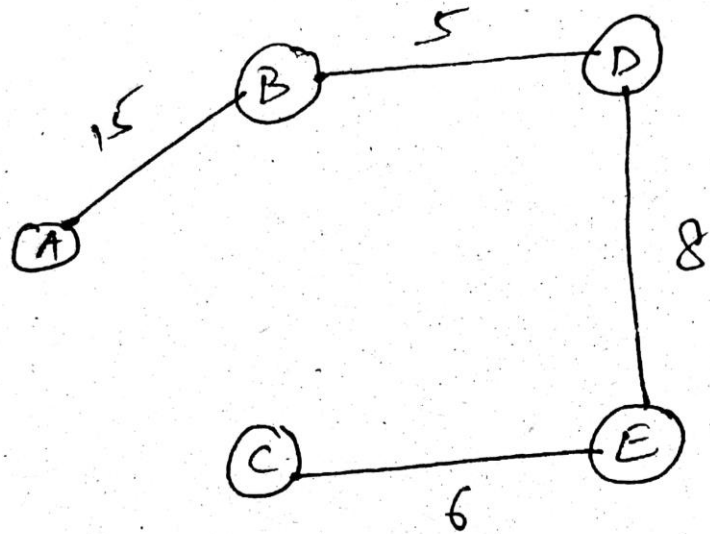


(C)


(E)



BD
EC
DE
CD
BC
AB
AC



$$15 + 5 + 8 + 6 = 34$$



MST-Kruskal(G)

for each vertex: Makeset(X)

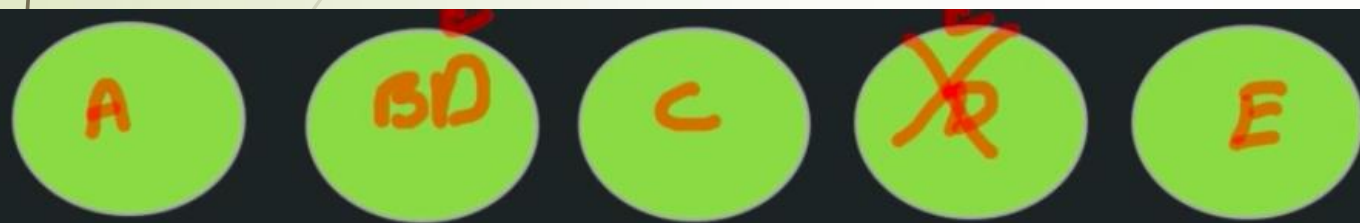
sort each edge in non-decreasing order by weight

for each edge(u, v) do:

if findSet(u) \neq findSet(v)

Union (u, v)

cost = cost + edge(u, v)



MST-Kruskal(G)

for each vertex: $\text{Makeset}(X)$

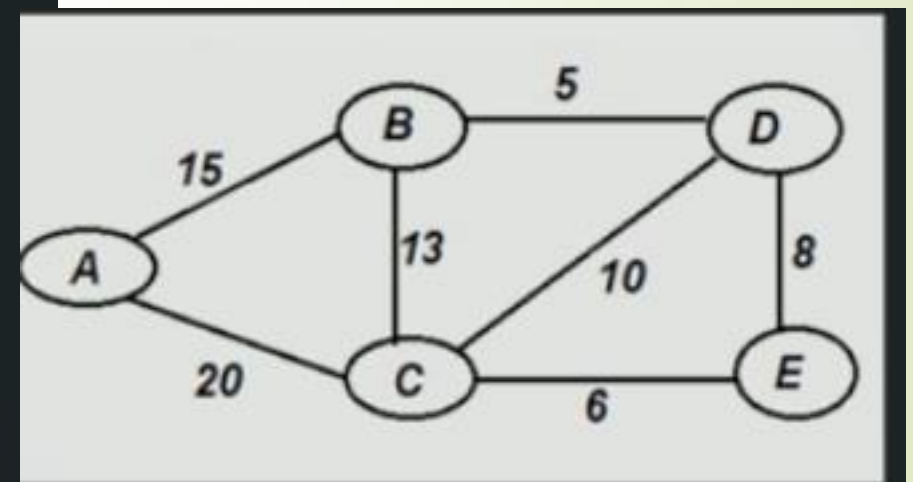
sort each edge in non-decreasing order by weight

for each edge(u, v) do:

if $\text{findSet}(u) \neq \text{findSet}(v)$

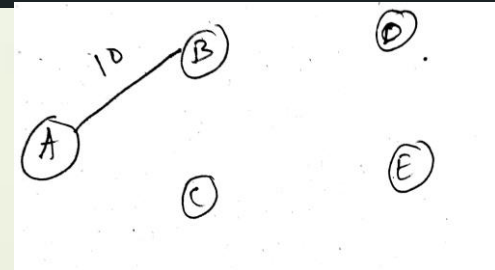
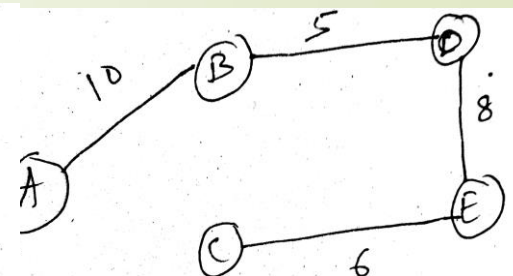
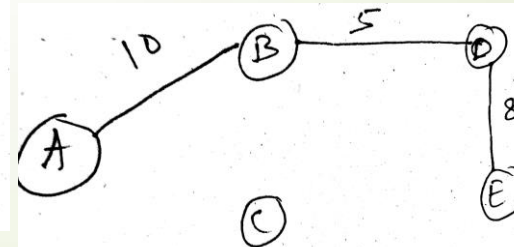
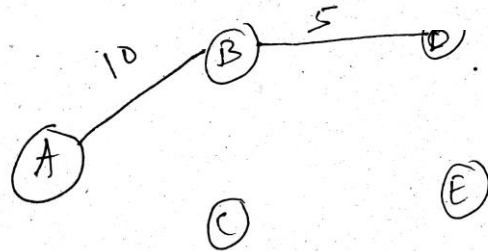
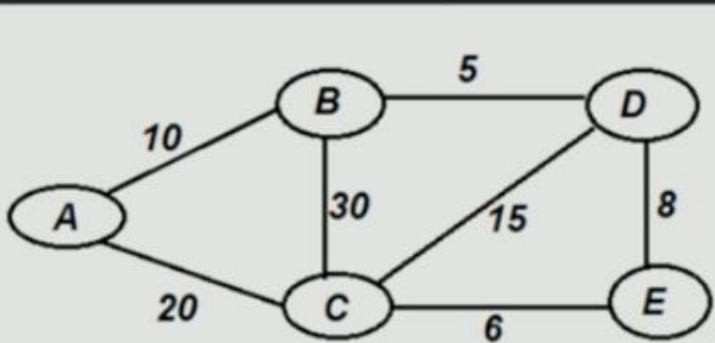
Union (u, v)

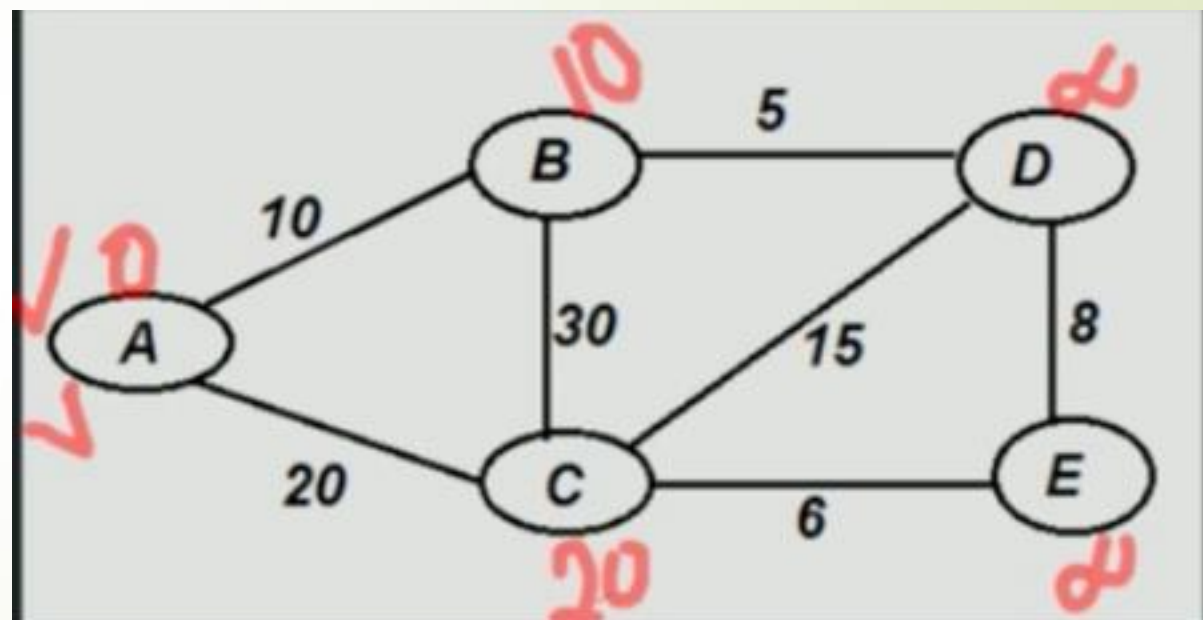
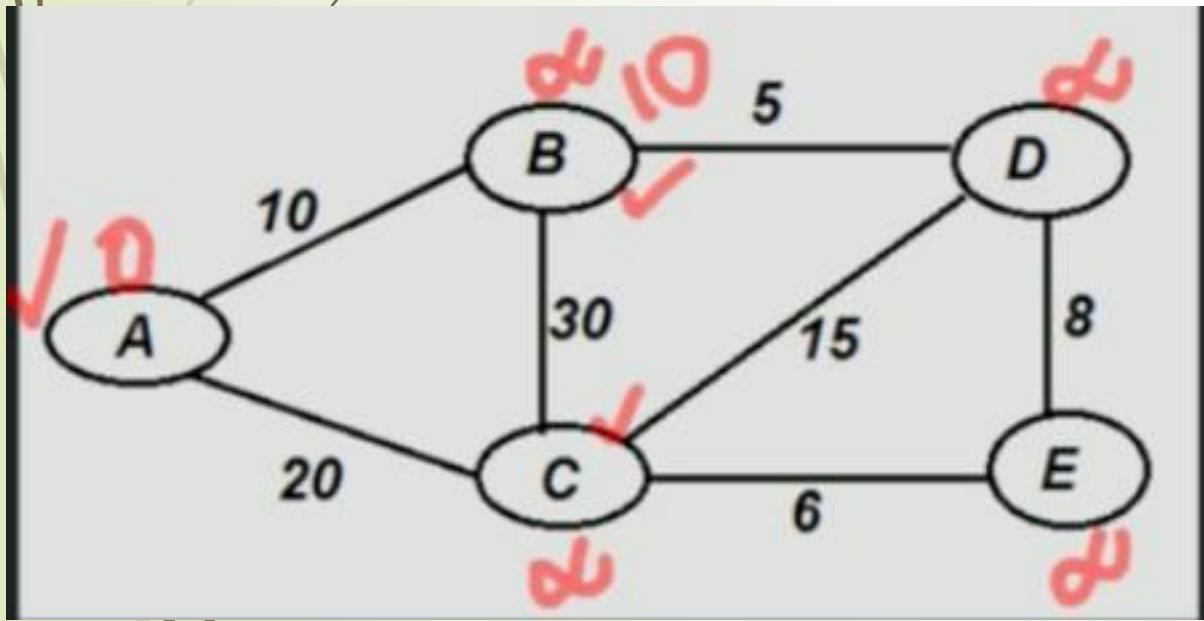
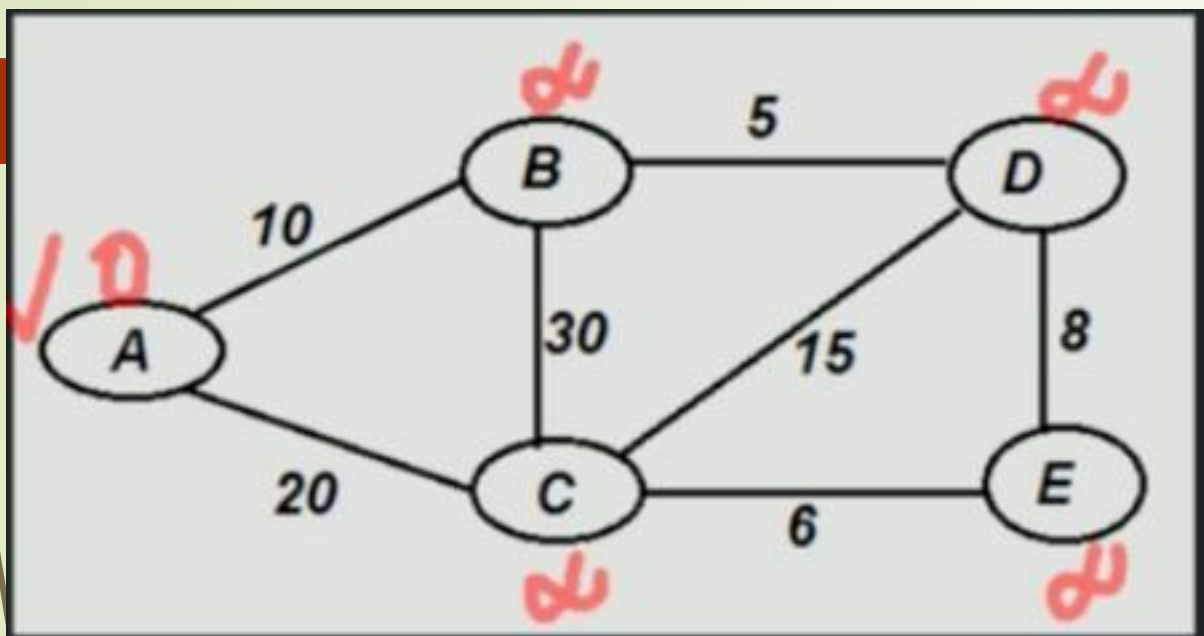
$\text{cost} = \text{cost} + \text{edge}(u, v)$

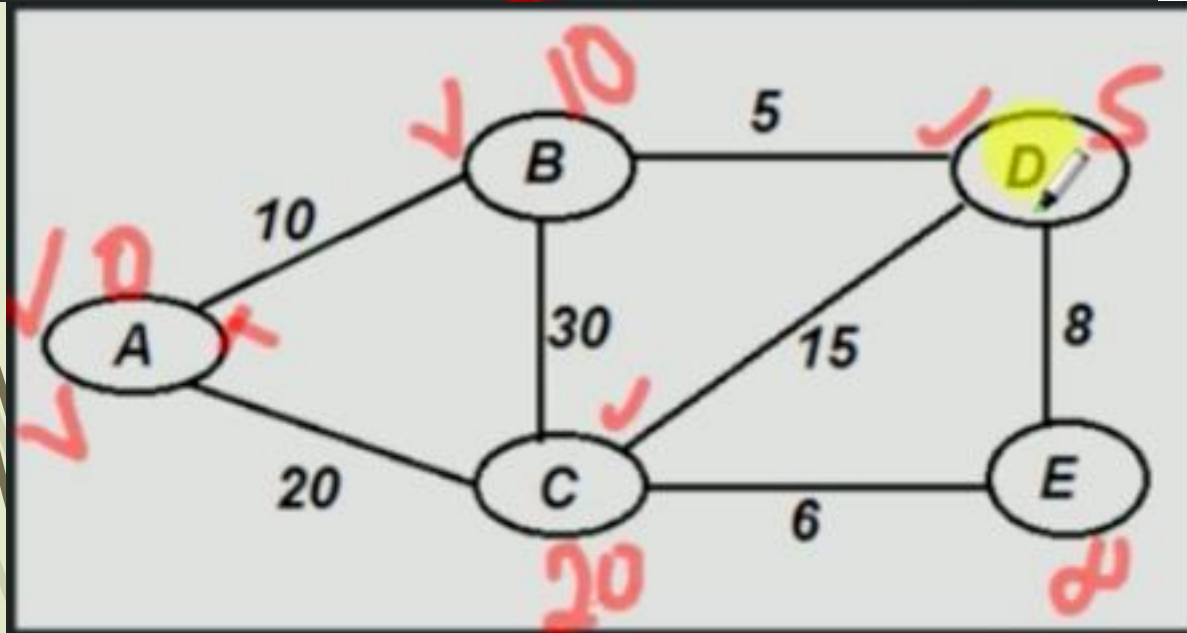
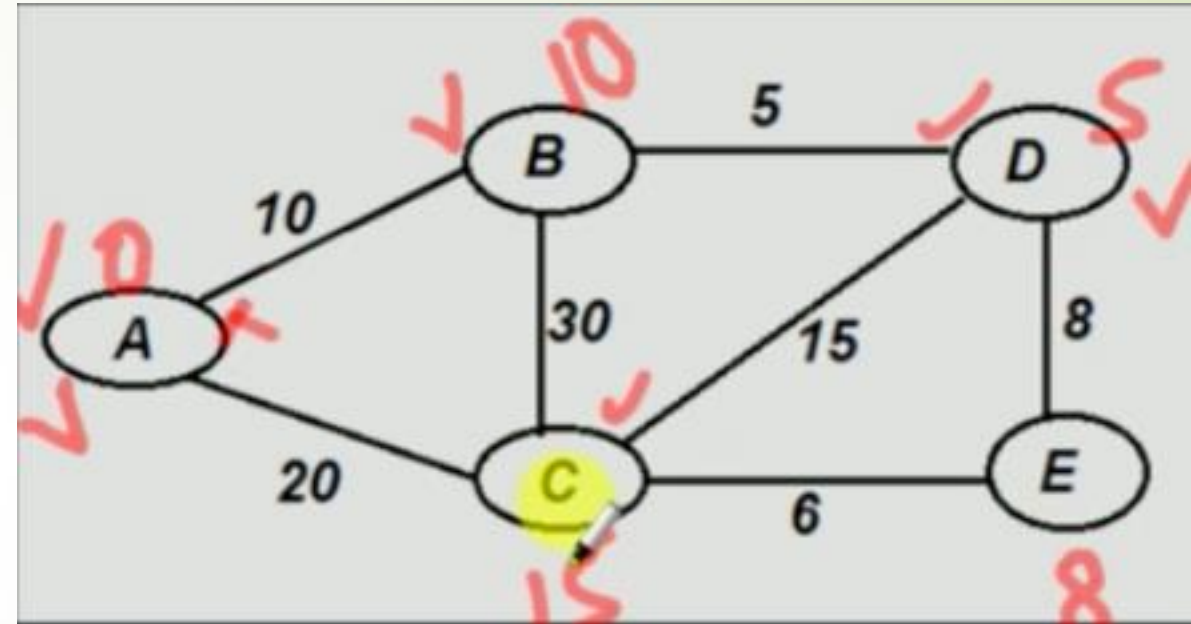
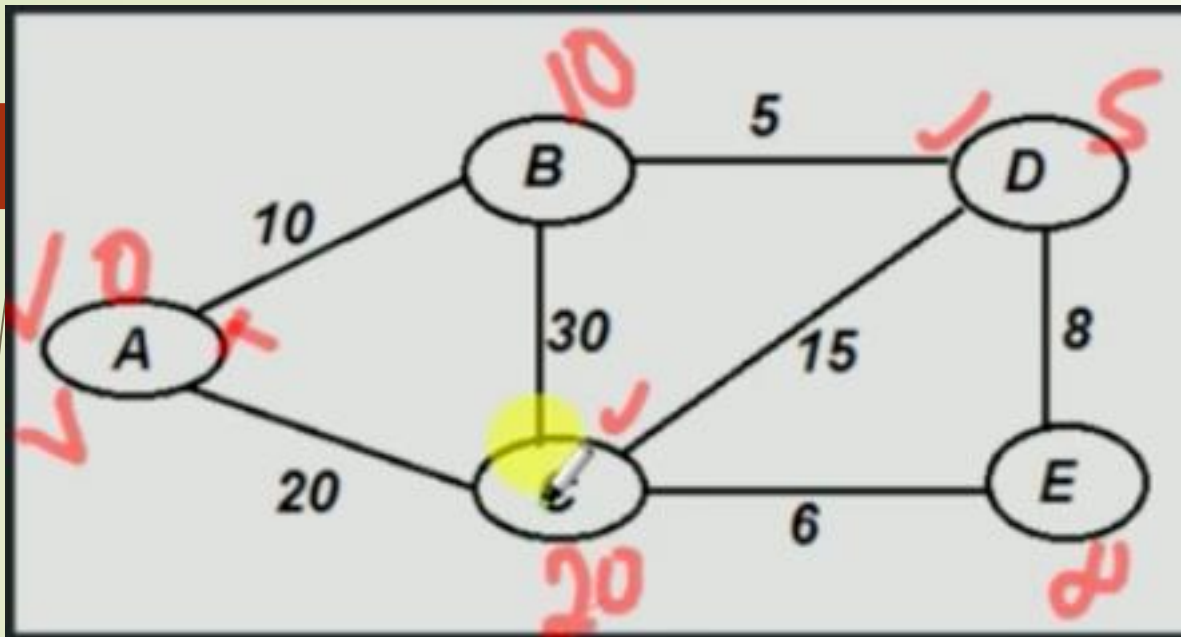


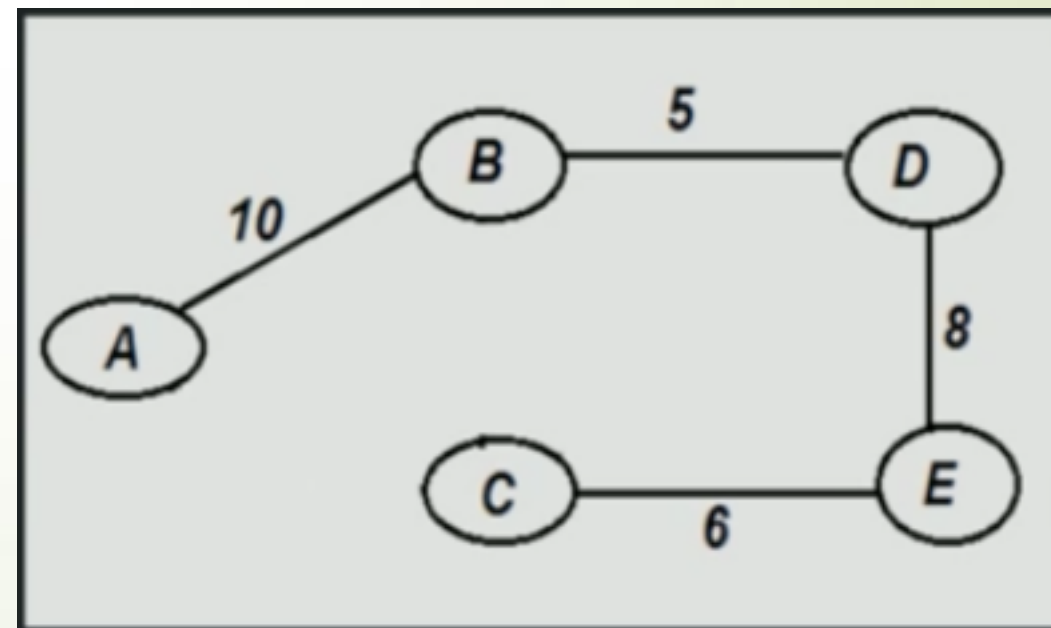
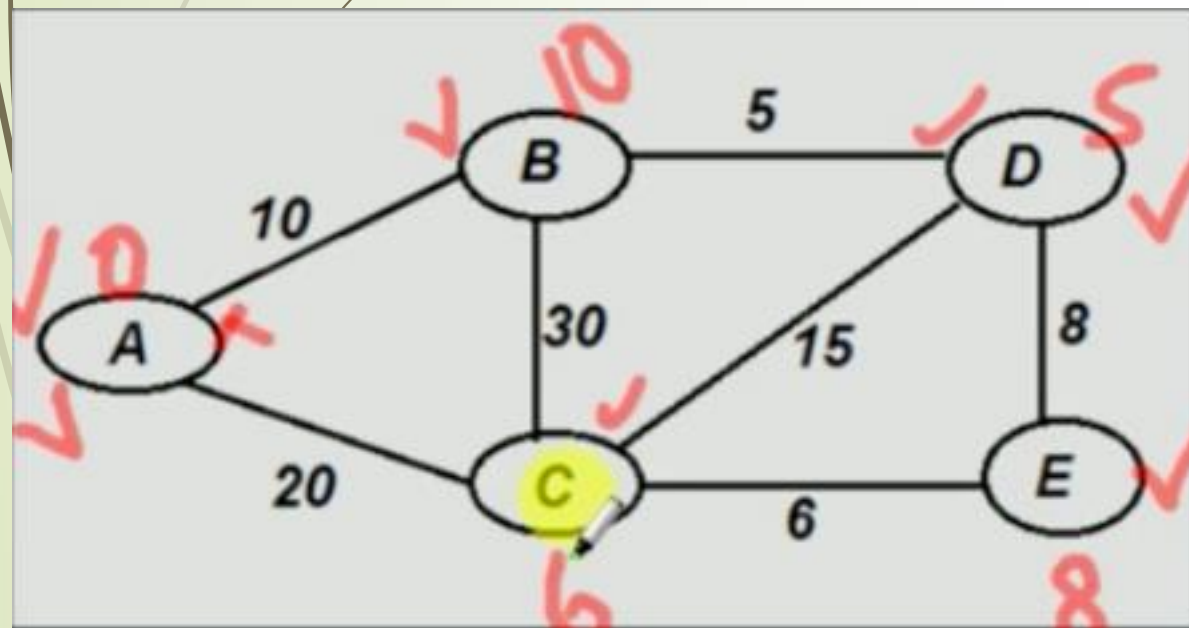
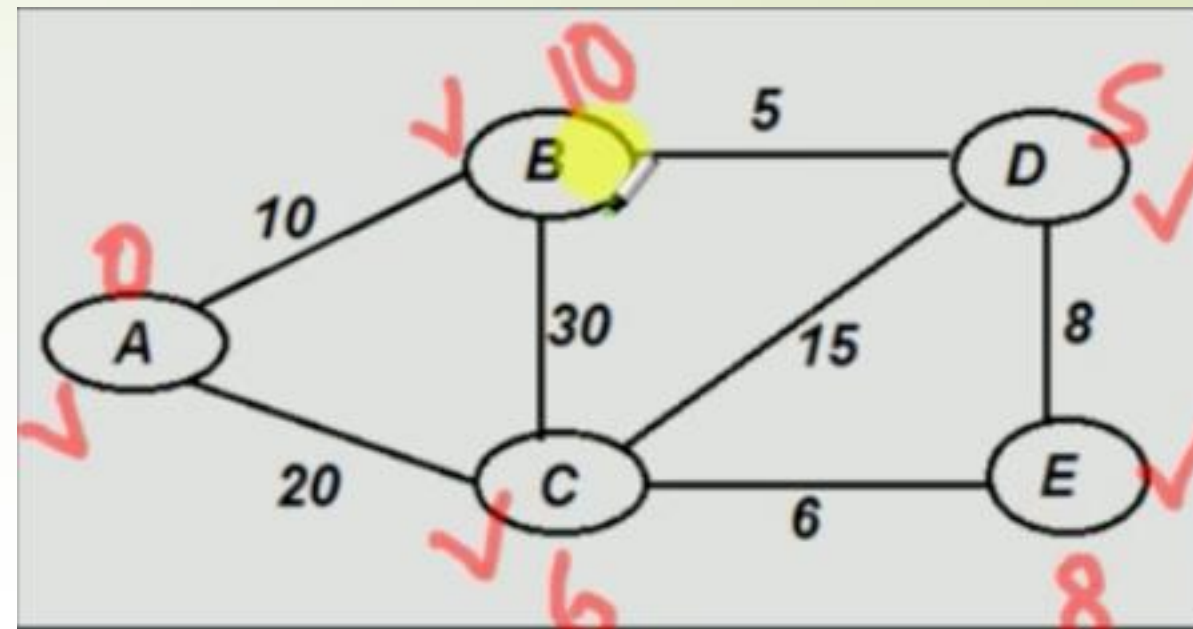
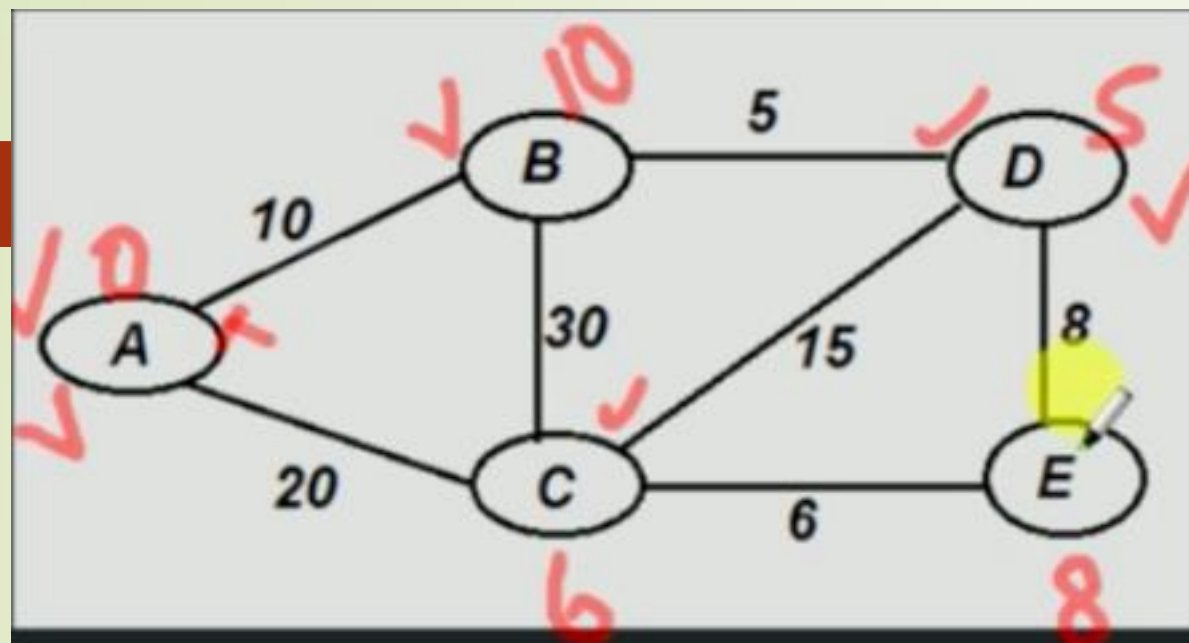
Prim's Algorithm:

- ✓ Prim's algorithm is a greedy algorithm.
- ✓ Take any Vertex as Source and mark weight of all the vertex as infinite and source as 0
- ✓ For every adjacent unvisited vertex of current vertex
- ✓ If current weight of this 'adjacent vertex' is more than current edge, then update 'adjacent vertex's' weight
- ✓ Mark currentVertex as Visited
- ✓ Do above steps for all the vertices in increasing order of weights









Prims Algorithm:

MST-Prims(G)

create a *PriorityQueue*(Q)



Insert all the vertices into Q such that key value of starting Vertex is 0 and others is infinite

while Q is not empty:

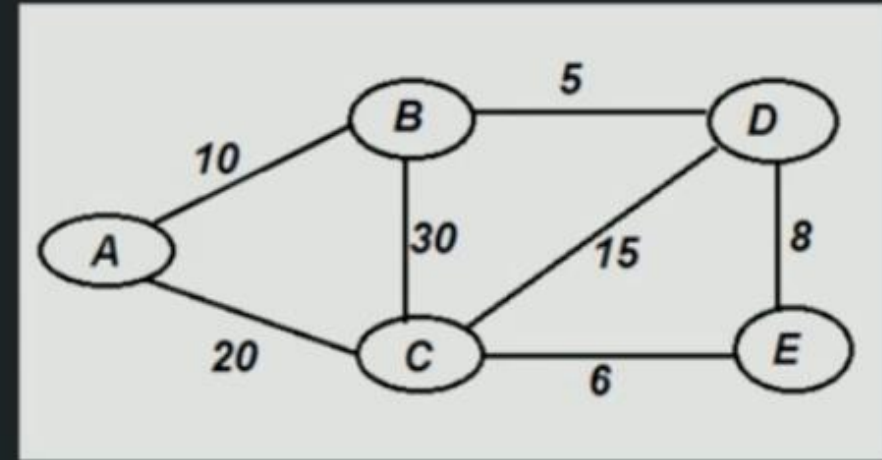
currentVertex = dequeue(Q)

for every adjacent unvisited Vertex of currentVertex

if current weight of this 'adjacent vertex' is more than current edge, then update 'adjacent vertex's' distance and parent

mark currentVertex as Visited

print all the vertices with weights





*Thank
you*