Graph Data Structure

Graph

A graph ***G*** is defined by two sets

- ***V*** : set of vertices
- ***E*** : set of edges

Notation:

- A graph ***G*** consisting of vertices ***V*** and edges ***E*** is denoted by

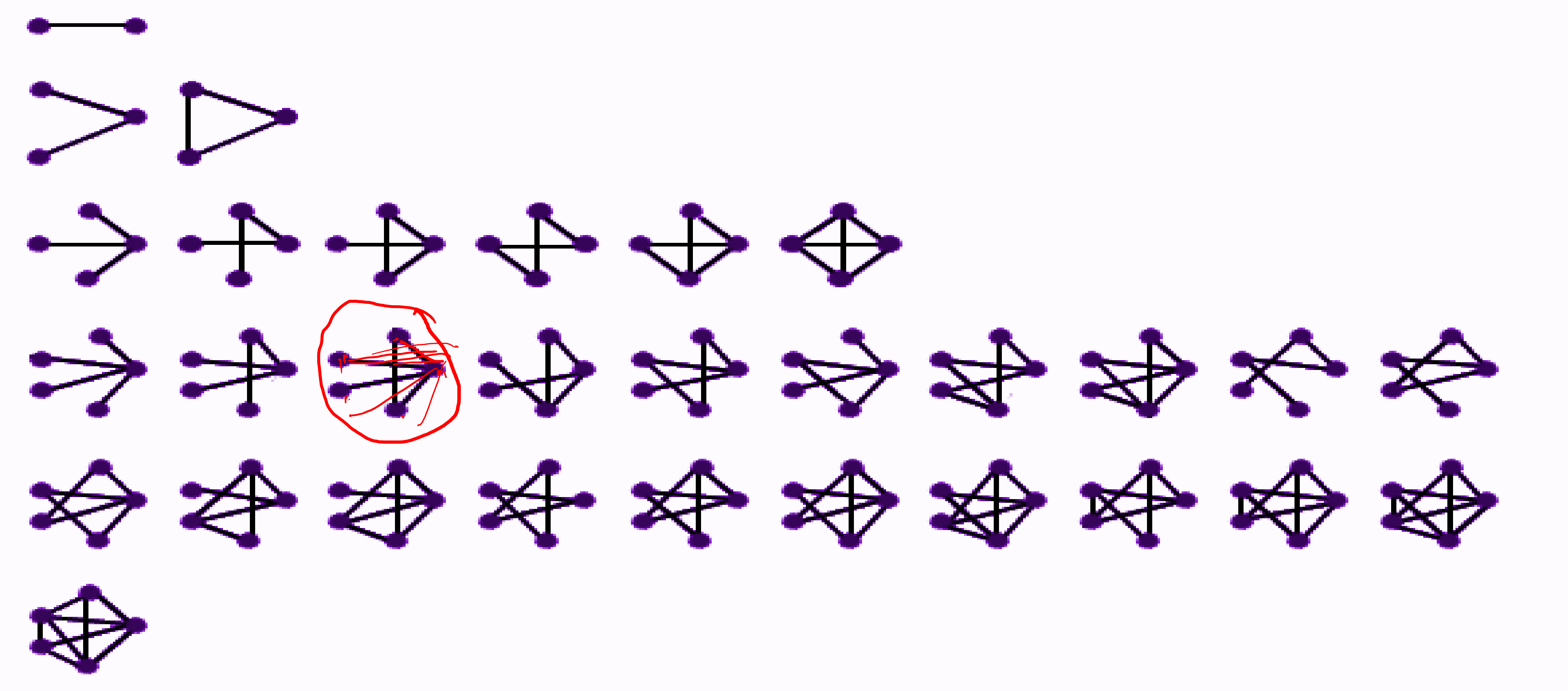
$$G(V,E)$$

Order of a graph

- The order of a graph is number of vertices in the graph.

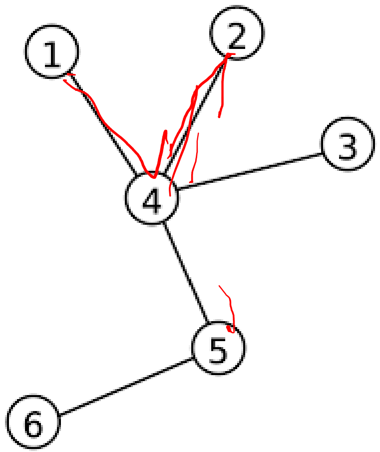
Connected graph

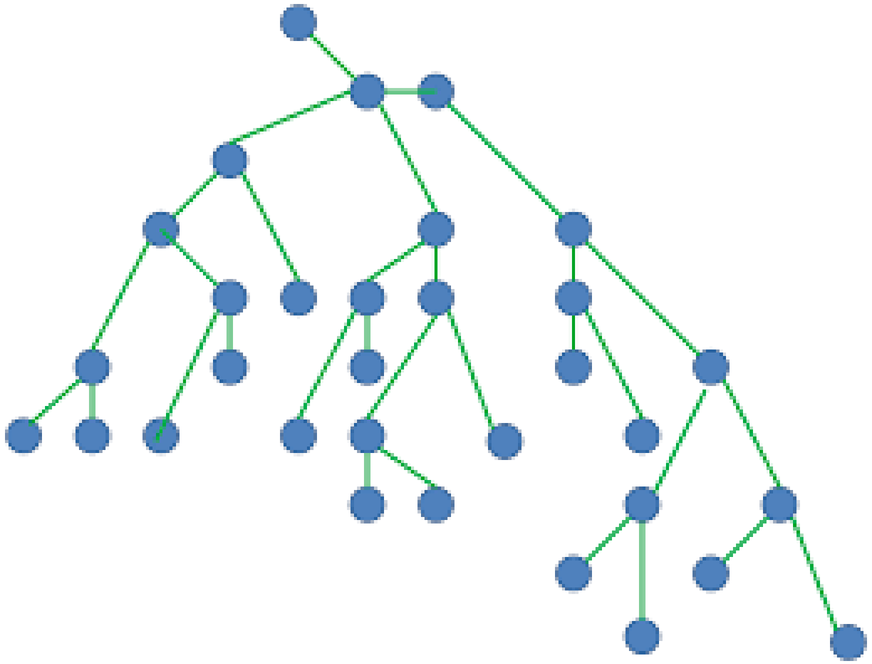
- A graph is a connected graph **if, for each pair of vertices, there exists at least one single path which joins them**.



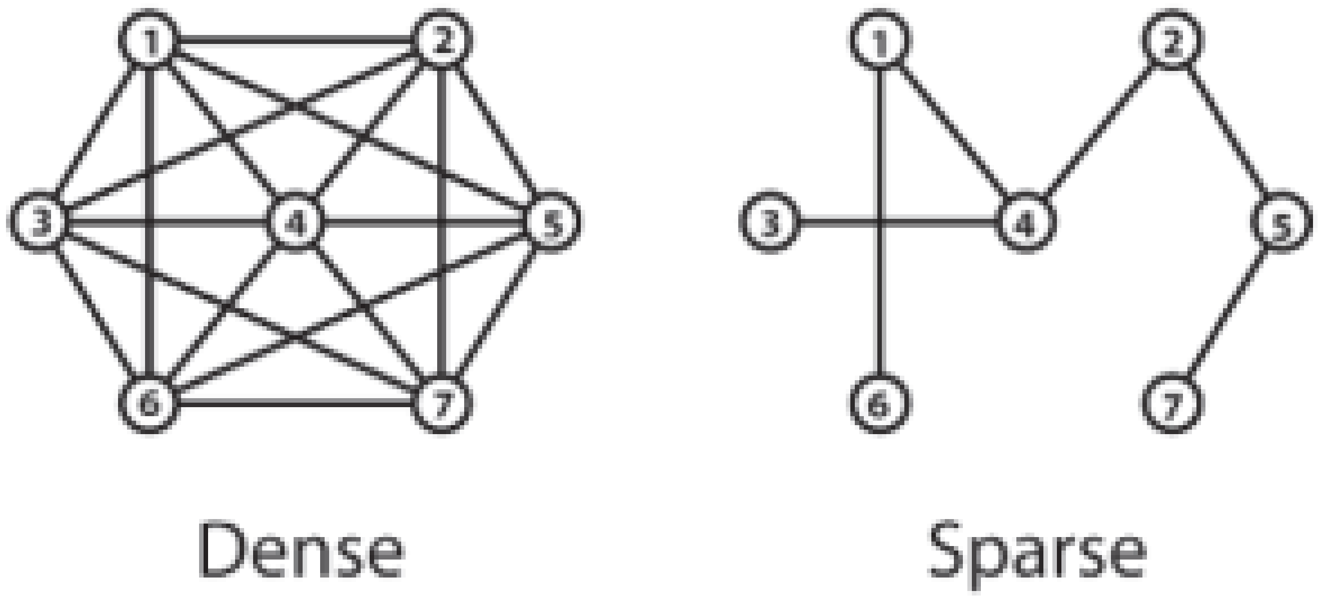
Tree graph

- A connected acyclic **graph** is called a **tree**. In other words, a connected **graph** with no cycles is called a **tree**.
- A **tree** is an undirected **graph** in which any two vertices are connected by exactly one path.





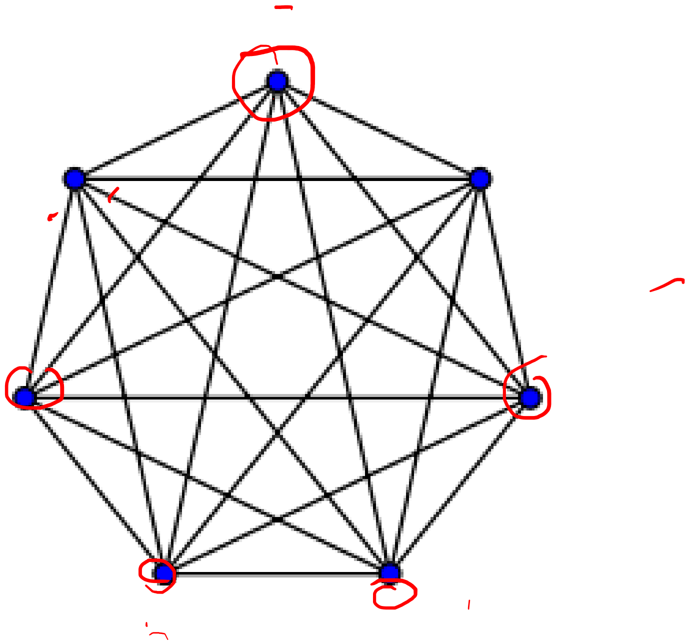
Sparsely connected graph

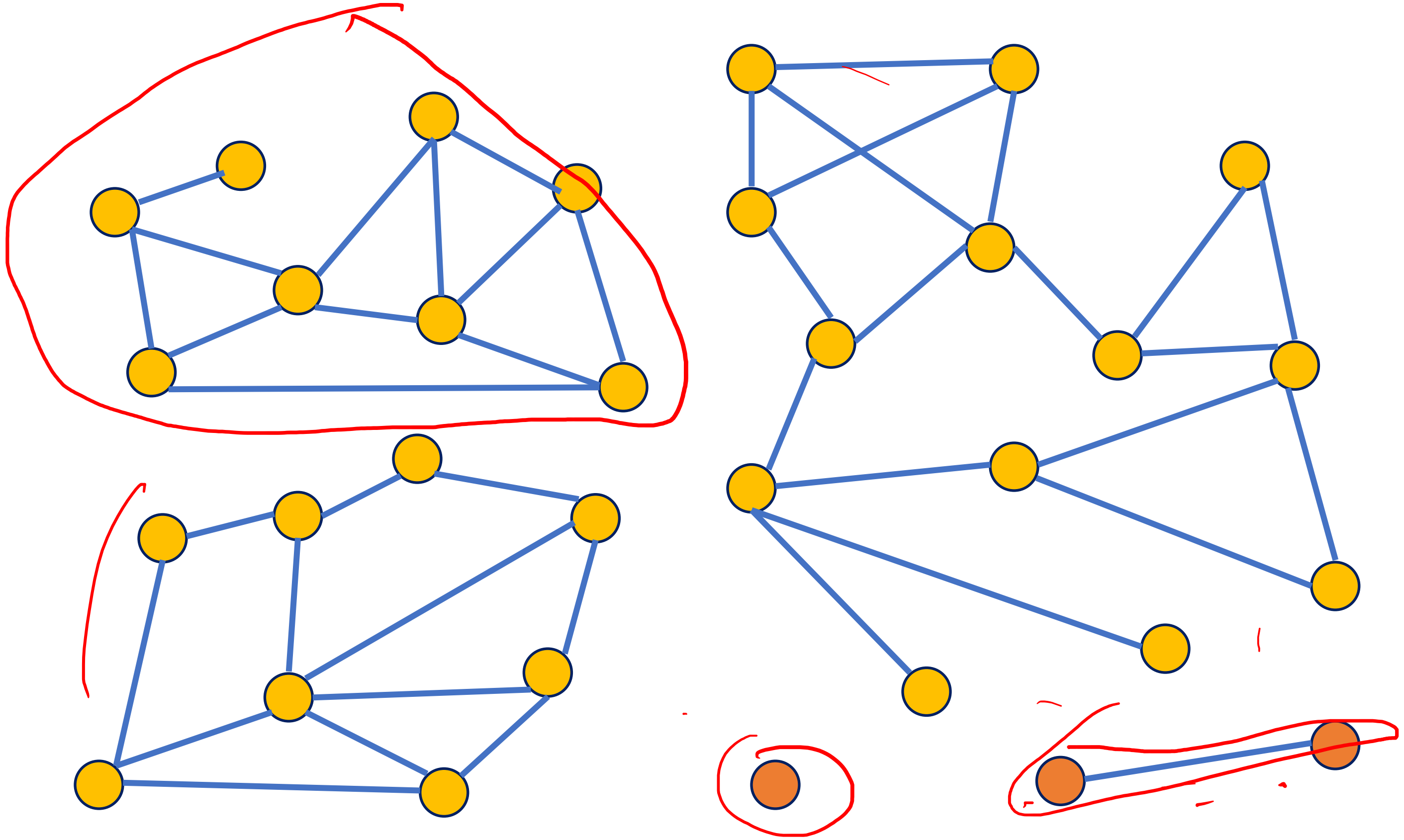


Fully connected graph

Complete Graph:

Every vertex is having an edge to all other vertices





Graph with 5 Connected components

Bi-connected Graph

- A graph is said to be Biconnected if:

1. It is connected, i.e. it is possible to reach every vertex from every other vertex,

2. Even after removing any vertex the graph remains connected.

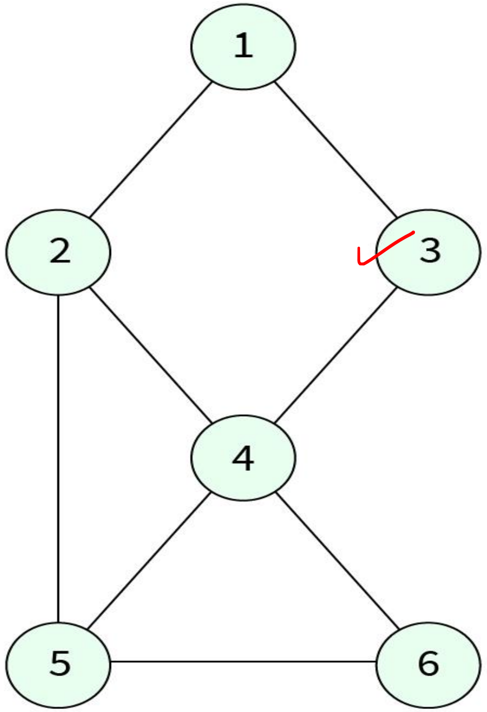


fig-1

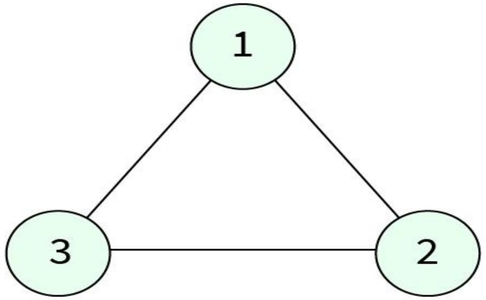


fig-2

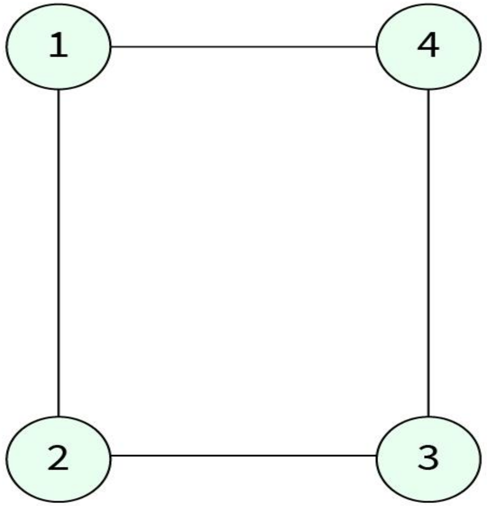


fig-3

Biconnected component

- A bioconnected component of a graph is

a connected subgraph

that **cannot be** broken into disconnected pieces

by deleting any single node (and its incident links)

Biconnected Graph

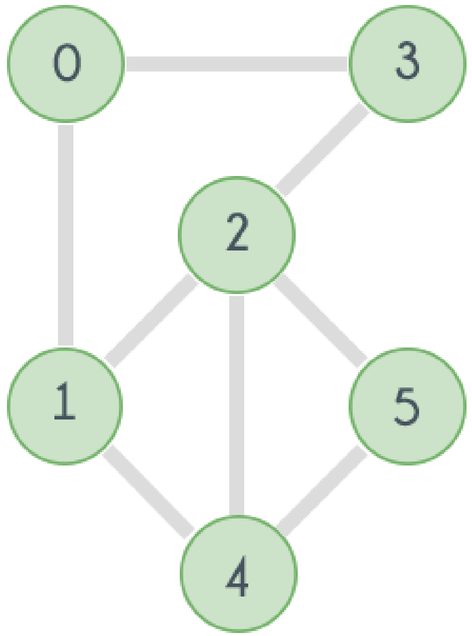


Fig. 1

- Removing any vertex from this graph does not increase the number of connected components.

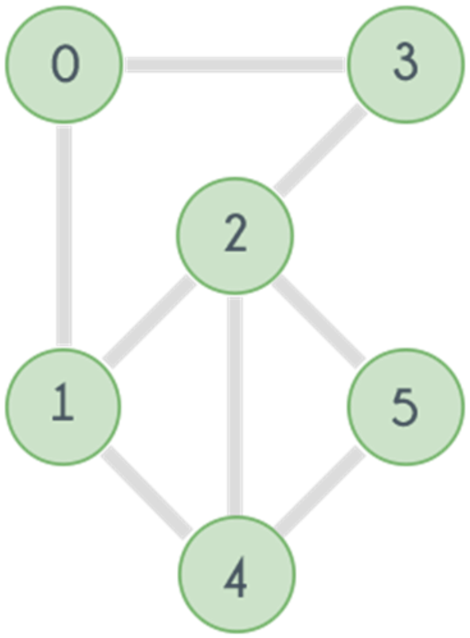


Fig. 1

Check if this graph is a Biconnected graph

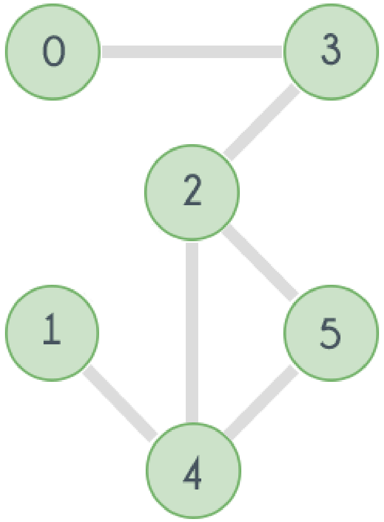


Fig. 2

- Removing vertex 2 increases number of connected components

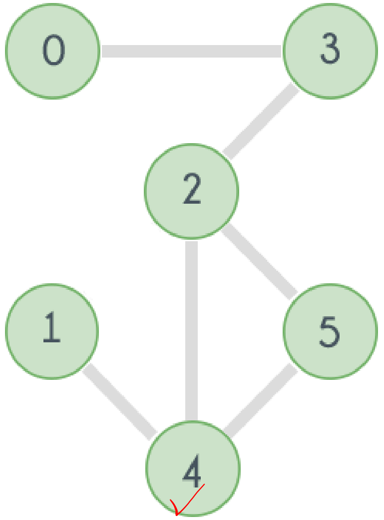


Fig. 2

1

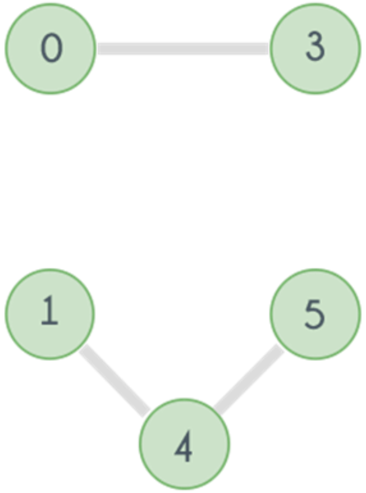


Fig. 3

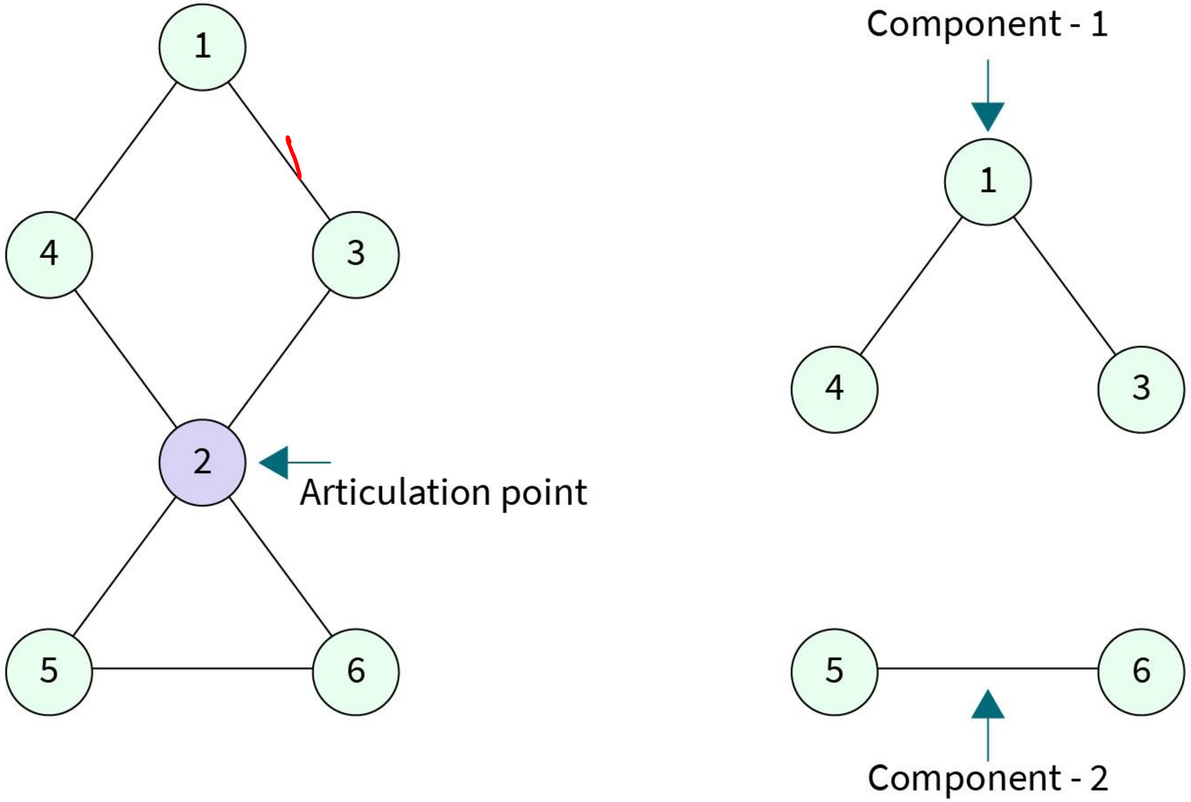
2

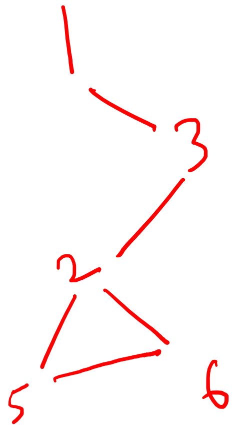
- so it is not a Biconnected graph

Articulation Point

- If removal of a vertex increases the number of connected components in the graph, then it is not Biconnected.

- A vertex is called an Articulation Point whose removal increases the number of connected components in the graph


1
4
3
2
Articulation point
5
6
Component - 1
1
4
3
5
6
Component - 2


1
3
2
5
6

Biconnected Component?

- It is one of the subgraphs which is Biconnected.
- Four biconnected components of the given graph

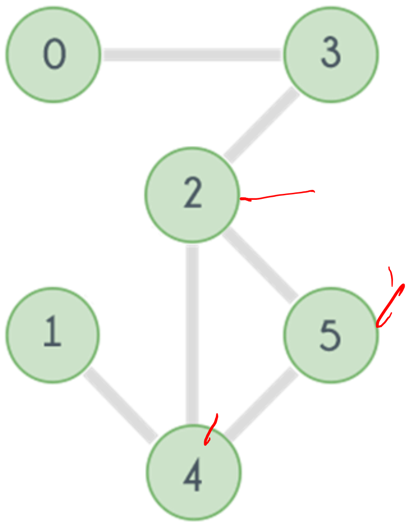


Fig. 2

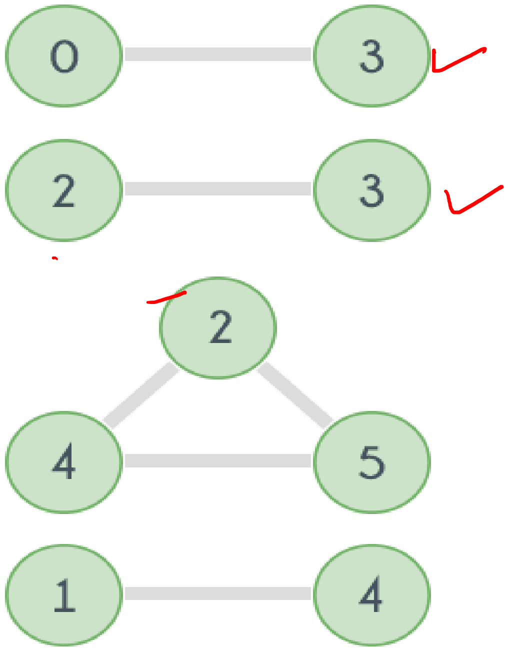


Fig. 6

- For a given graph, a Biconnected Component is one of its subgraphs that is Biconnected.

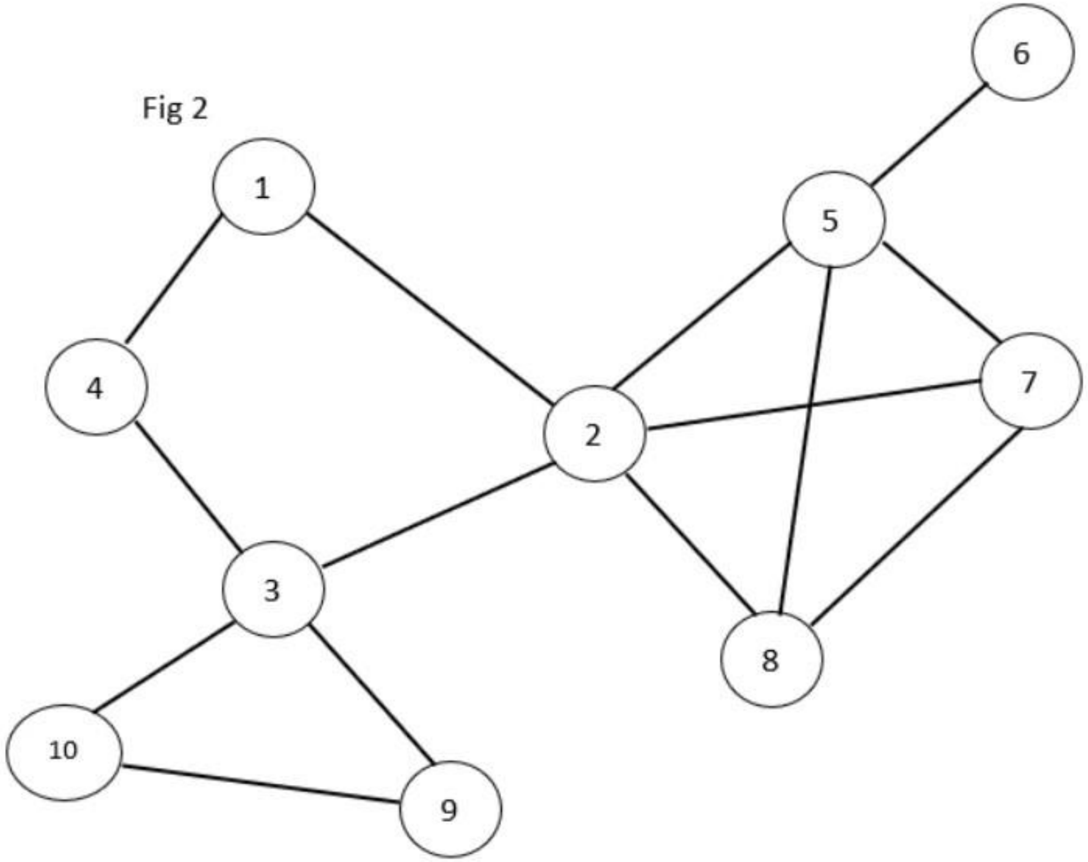
- This means there is always a path between any two nodes in the component, even after removing any node from the component.

Biconnected Component

- A *biconnected component* is a group of vertices and edges that are all connected to each other in a way

- that you can always get from one vertex to another using *two different paths.*

Fig 2



- How to discover biconnected components?

- *https://www.hackerearth.com/practice/algorithms/graphs/biconnected-components/tutorial/*

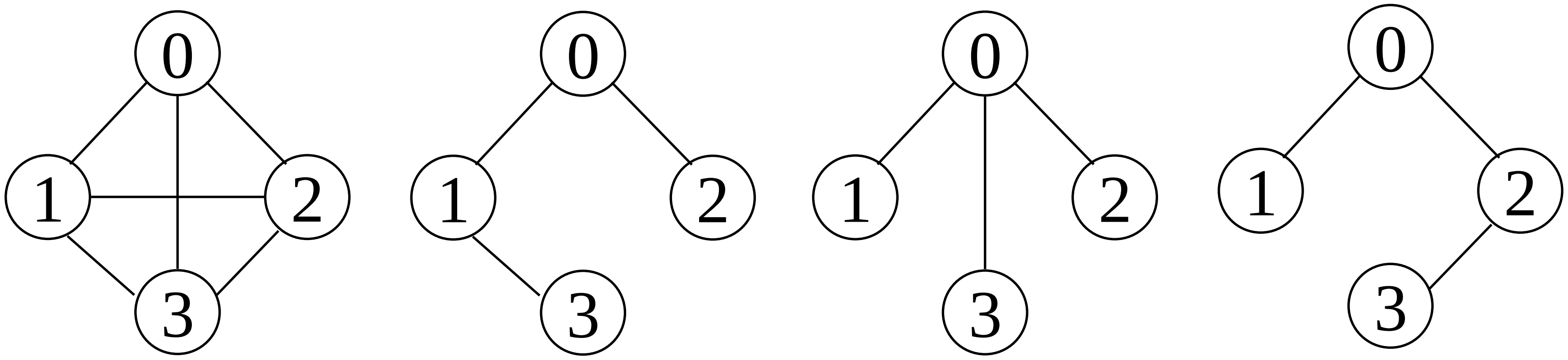
Spanning Trees

Spanning Tree : Tree Graph

Tree formed of graph edges which connect all the vertices of the graph.

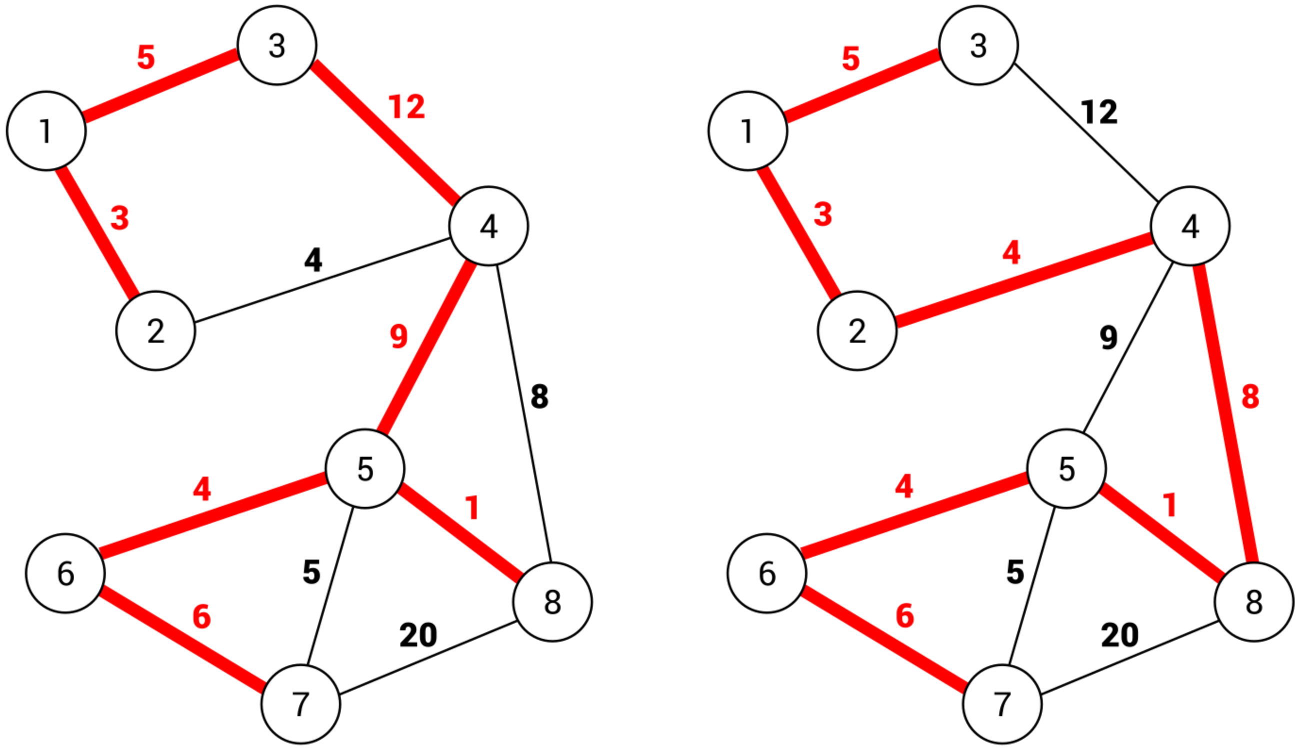
Spanning tree does not have a cycle

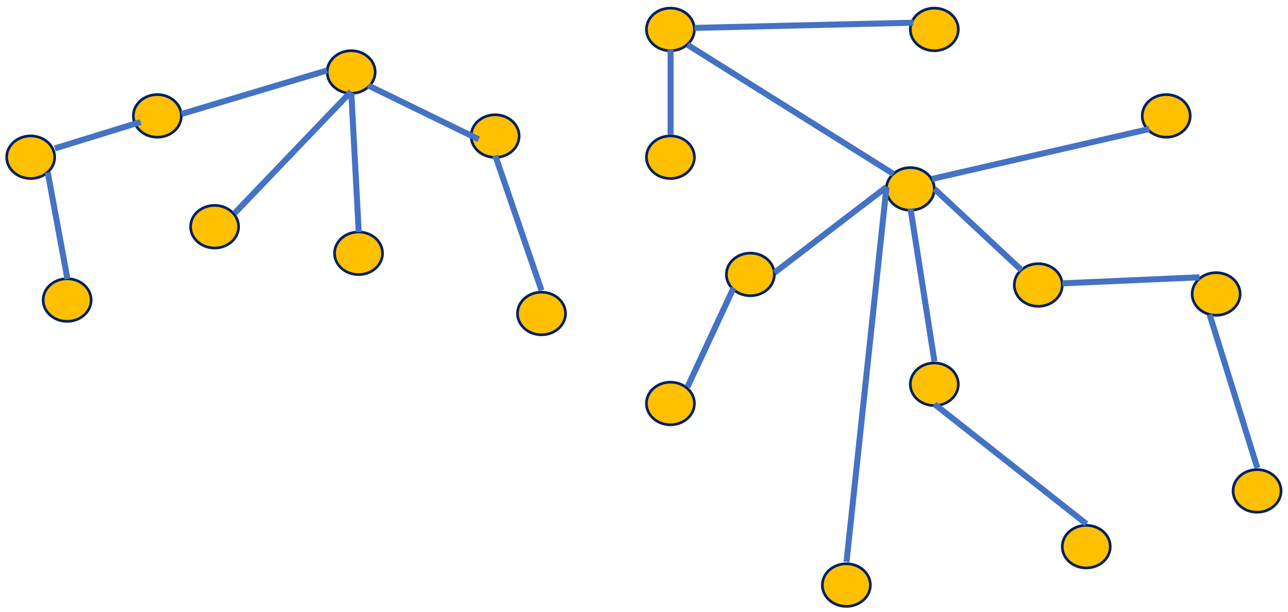
Spanning Tree examples



G_1

Possible spanning trees


1
2
3
4
5
6
7
8
5
3
12
4
9
8
4
1
5
6
20
1
2
3
4
5
6
7
8
5
3
12
4
9
8
4
1
5
6
20

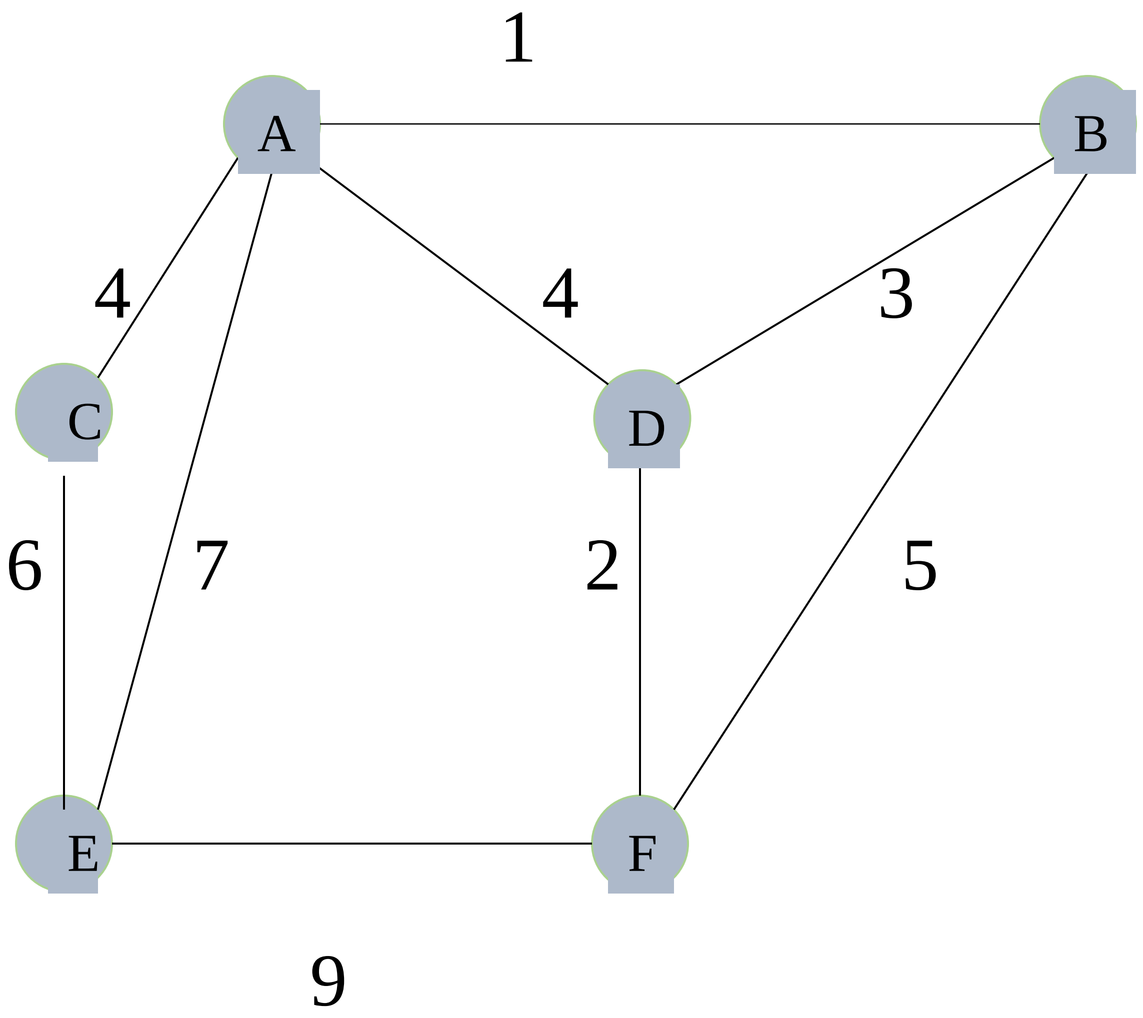


Spanning Trees

Spanning Tree

- Spanning Tree (ST) of an undirected graph
  - includes all its nodes,
  - is connected,
  - *(you can go from any node to any other node)*
  - is acyclic

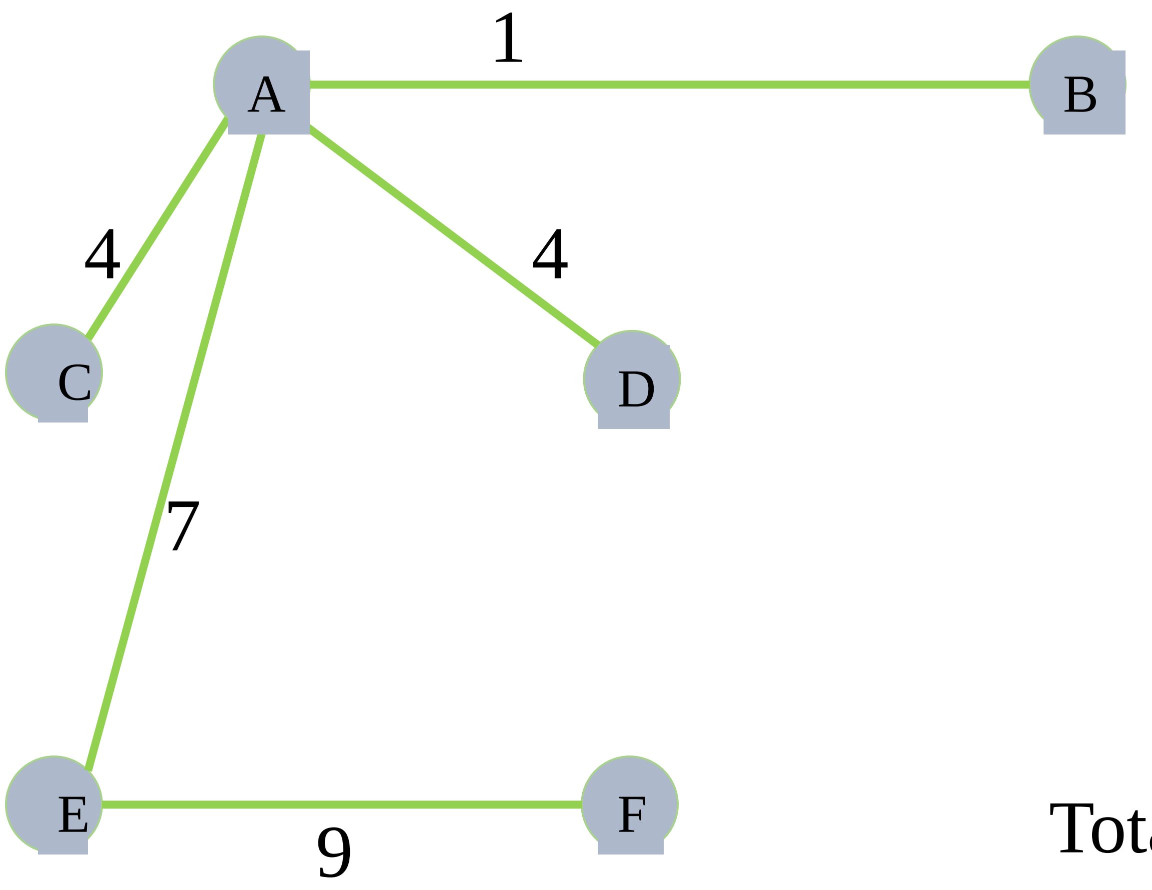
Consider the spanning trees for the following graph with 5 vertices.



There can be number of possible Spanning Trees. Here is one.

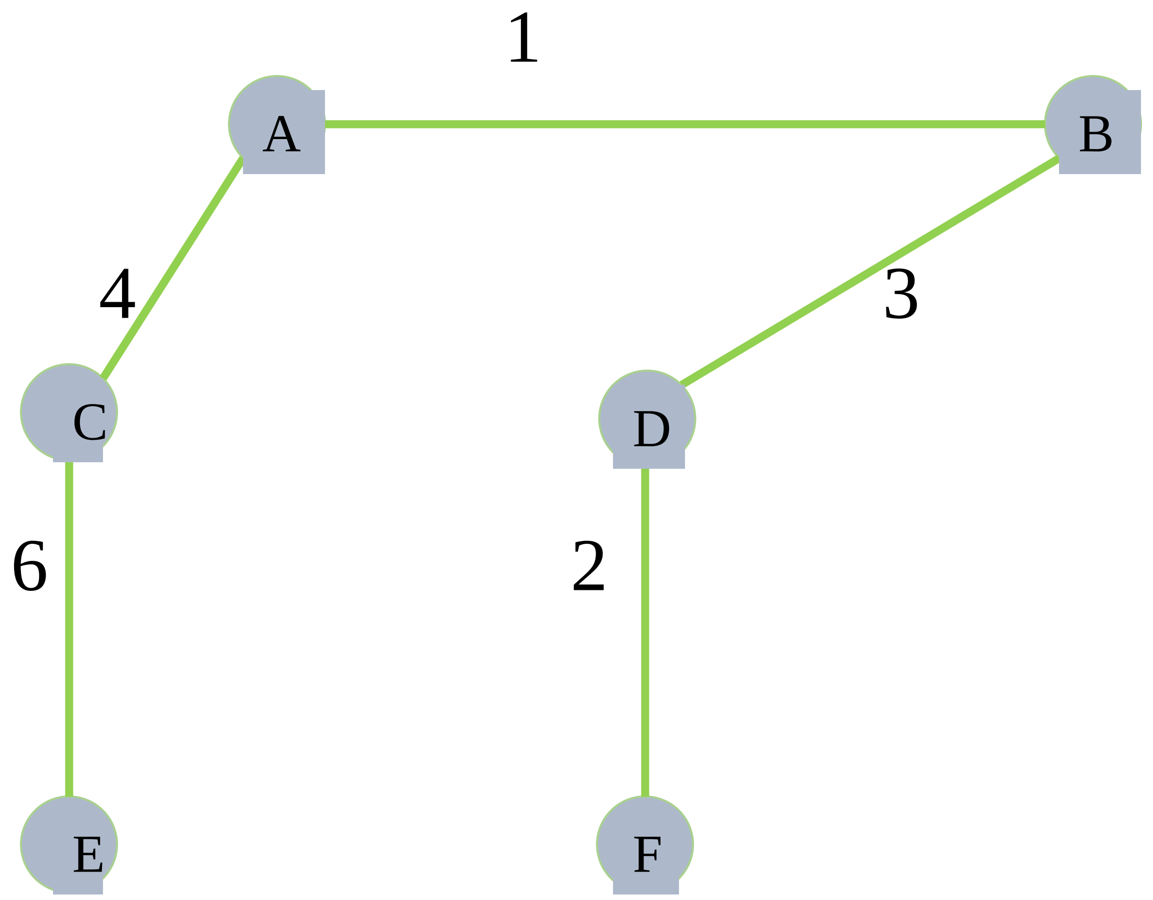
All nodes are connected. No cycles.

We are interested in Cost of Spanning Tree



Total Cost : 25

Another spanning tree of same graph. All 5 nodes are connected. But its weight is only 16.



Smallest cost network

- Each edge of a real world graph will have some weight, such as path length, cost to travel, time to travel etc.
- The problem is to select subset of all edges, such that the network has smallest cost
- This is done by creating a spanning tree of the graph which has smallest cost

MST

- Which spanning tree is most interesting or useful?
- All spanning trees provide connectivity between various nodes
- However, the edges have different costs (path lengths).

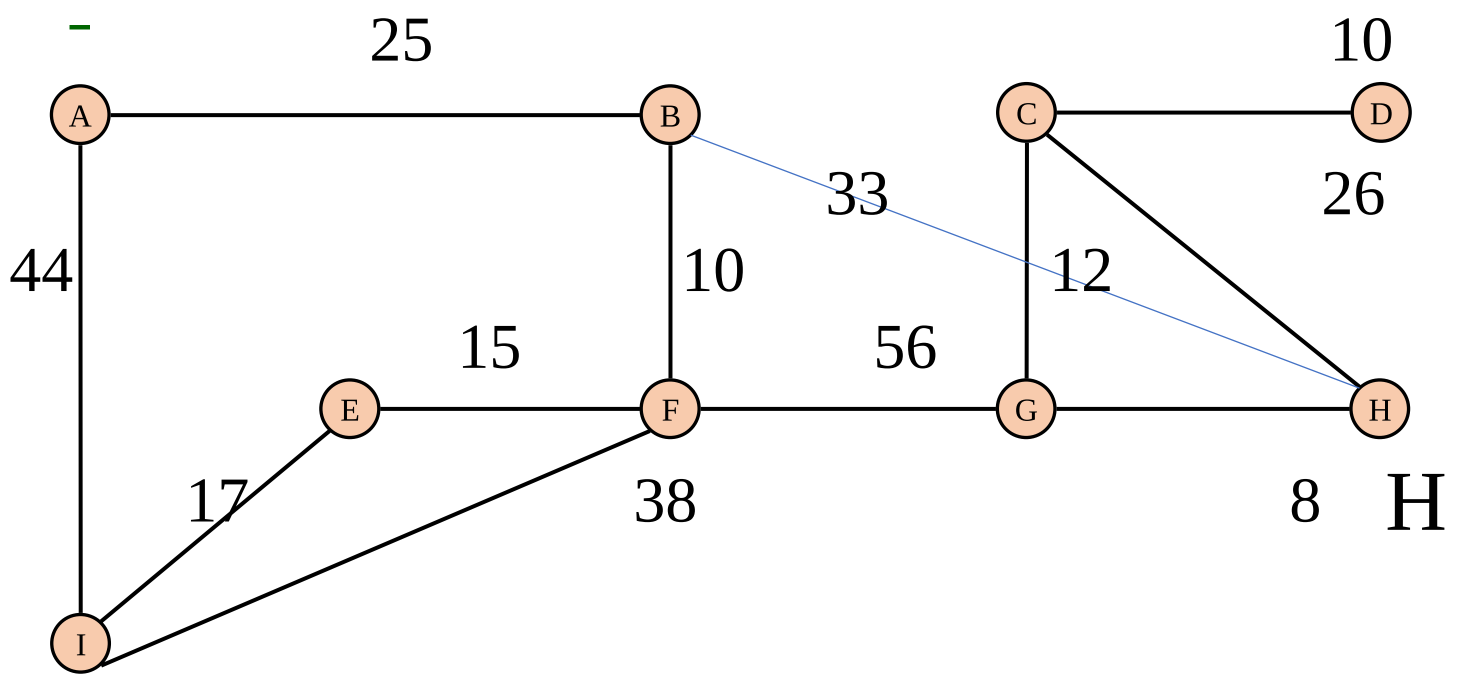
- The useful spanning tree would be the one for which total cost of the edges is minimum.
- Such a tree is called the Minimum Spanning Tree (MST)

MST of a Graph

Applications of MST in real world Graphs

- Design of minimum cost network while providing connections to all points
  - Road network
  - cable network
  - electrical network
  - telephone network
  - wire routing in printed circuit board
  - Supply chain network

Select minimum cost road network which connects all cities



MST

- A Minimum Spanning Tree (MST) of an undirected graph
  - includes all its nodes,
  - is connected,
  - is acyclic, and
  - has minimum total edge weight

Greedy Strategy for MST

MST Algorithms

- Two algos for MST
    - <u>Kruskal’s algorithm</u>
    - <u>Prim’s algorithm</u>

Kruskal’s Algorithm

- It builds the spanning tree by adding edges one by one into a growing spanning tree.

- *It follows the greedy strategy to build the MST*
- Since objective is to build smallest cost tree, the greedy approach would be to locate an edge which has least weight
- and add it to the growing spanning tree.
- Repeat this for all iterations.

Kruskal’s Algorithm

- **Algorithm Steps:**

- Each edge has a weight.
- Sort the edges with respect to their weights.
- Add edges to the MST, starting with the smallest weight, until the edge with the largest weight.
- Take care not to add an edge if it forms a cycle

- This could be implemented using DFS which starts from the first vertex, then check if the second vertex is visited or not.
- But DFS will make time complexity large as it has an order of O(V+E)
- where V is the number of vertices, E is the number of edges.

- A better way is to make use of **"Disjoint Sets“.**

Kruskal’s method for MST

- Kruskal’s algorithm
- starts with an empty spanning tree, and
- creates a tree by progressively adding edges with lowest cost in the graph

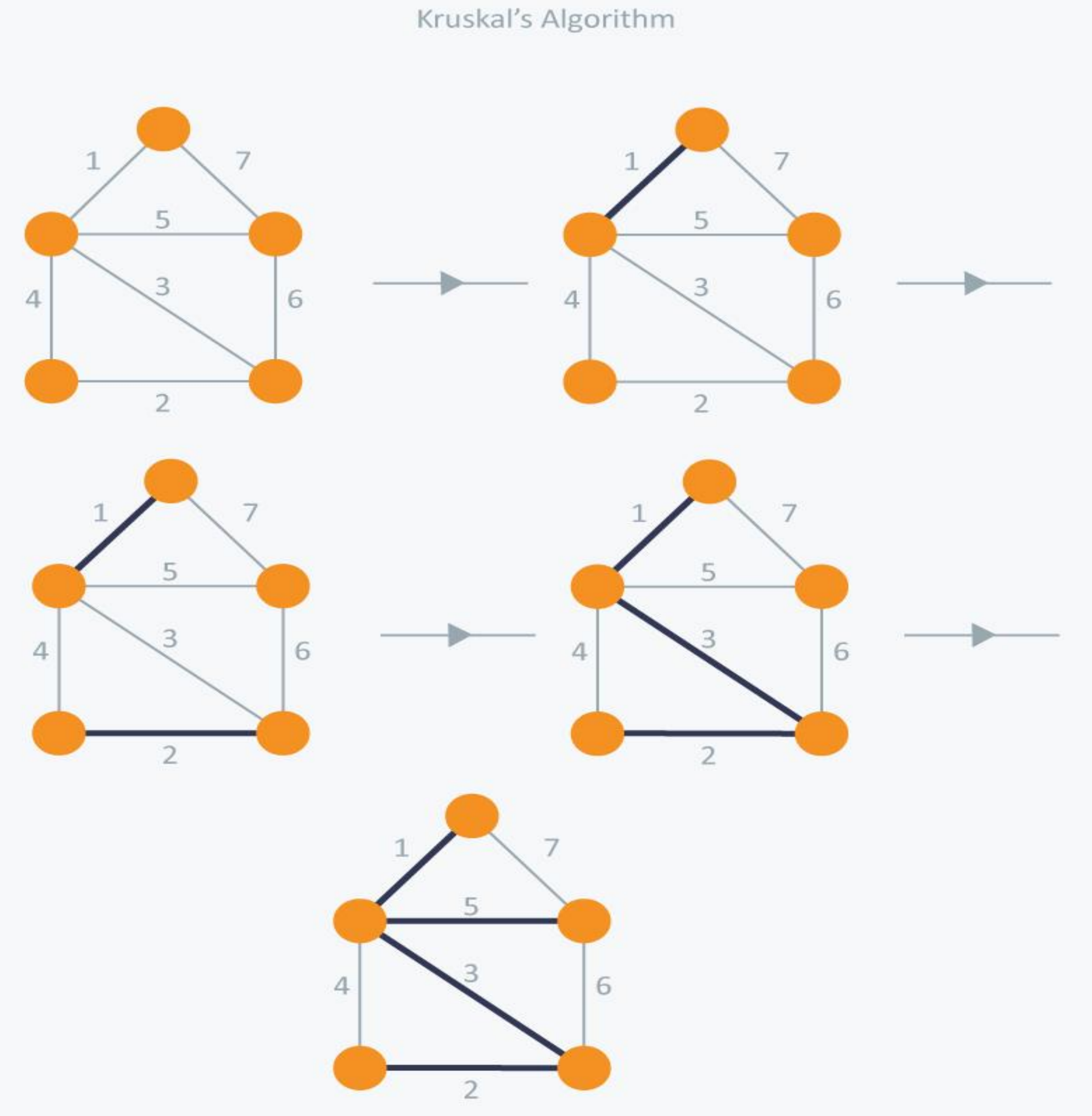
- Let there be n vertices and m edges in a graph

- Sort the edge list in increasing order.
- Make a set of n disjoint sets.
- Pick up the edges one by one from the list.
- Add the edge to MST if it does not form a cycle

Consider a graph with edge weights 1,2,…,7

Edge 1: OK
Edge 2: OK
Edge 3: OK
Edge 4: forms cycle
Edge 5: OK
Edge 6: forms cycle
Edge 7: forms cycle

Total cost: 11

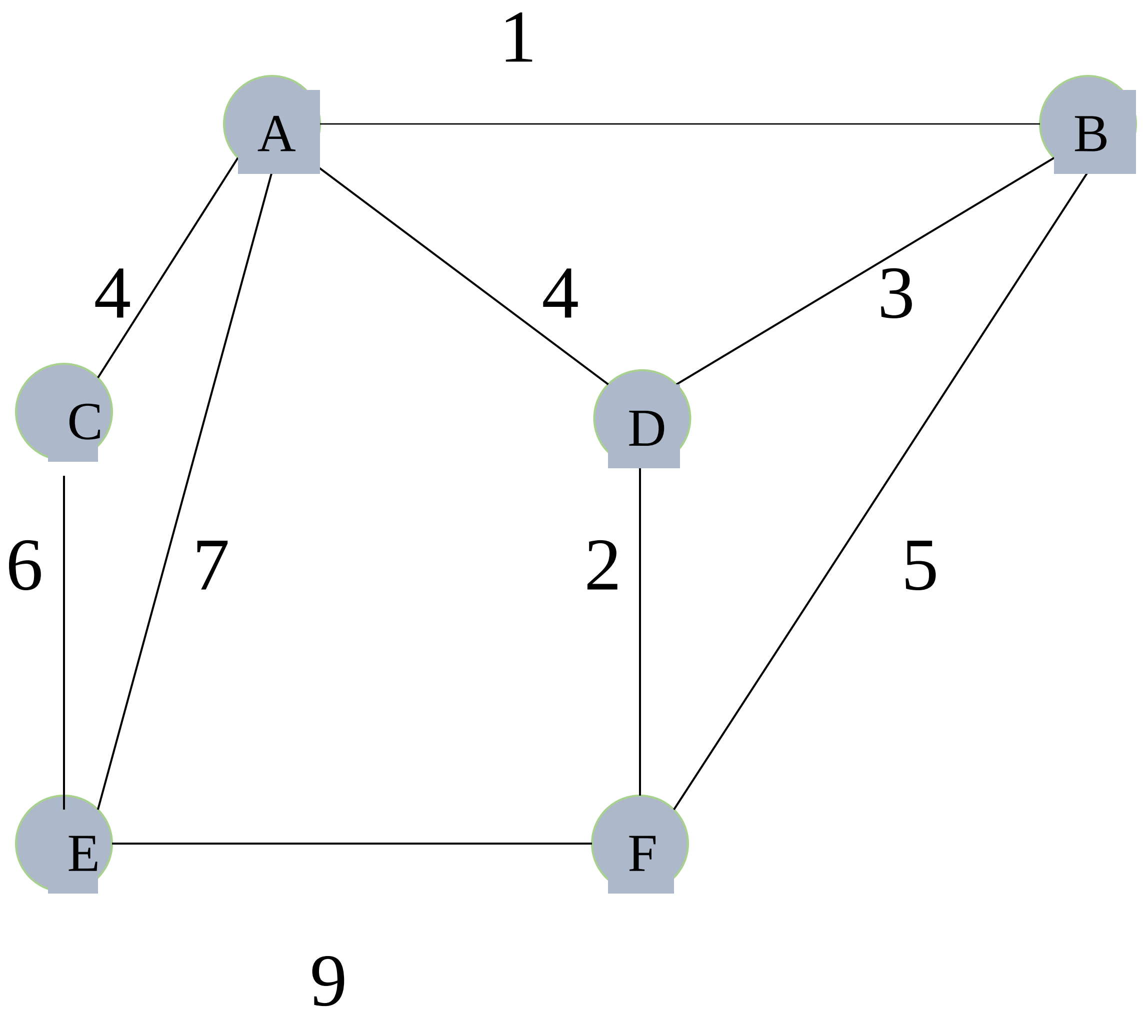


```
//Kruskal algorithm using Disjoint Sets
Kruskal( edgelist, v)  {
        sort(edgelist)
        for i = 1 to n
                makeset(i)
        count = 0;        i=1
        while ( count < m – 1)   {
            if (find(edgelist[i].v   !=   find(edgelist[i].w))  { // check if edges have
               println ( edge  )                                  //common root
               count= count + 1
               union(edgelist[i].v , edgelist[i].w)   }
            i = i + 1  }
}
```

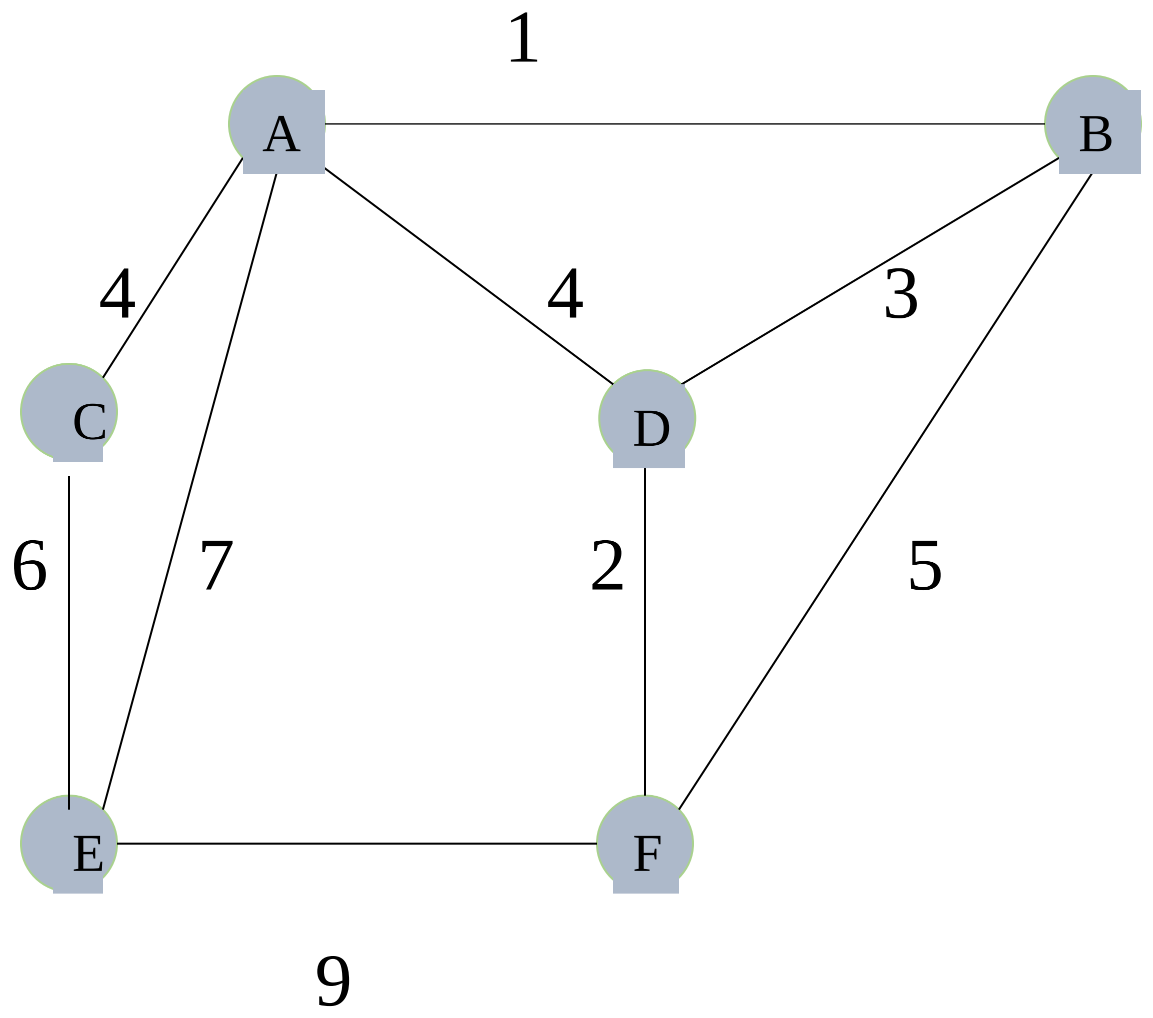
Complexity of Kruskal Implementation

- There are m makeset operations, atmost 2m find operations and n-1 union operations.
- because graph is connected, m >= n-1
- Disjoint set graph is of height at most log m
- number of union and find operations is O(m log m)
- sorting edges would also take atmost Θ(m log m)
- Thus worst time for Kruskal's algorithm is Θ(m log m)

Create the MST for the following graph with 6 vertices.



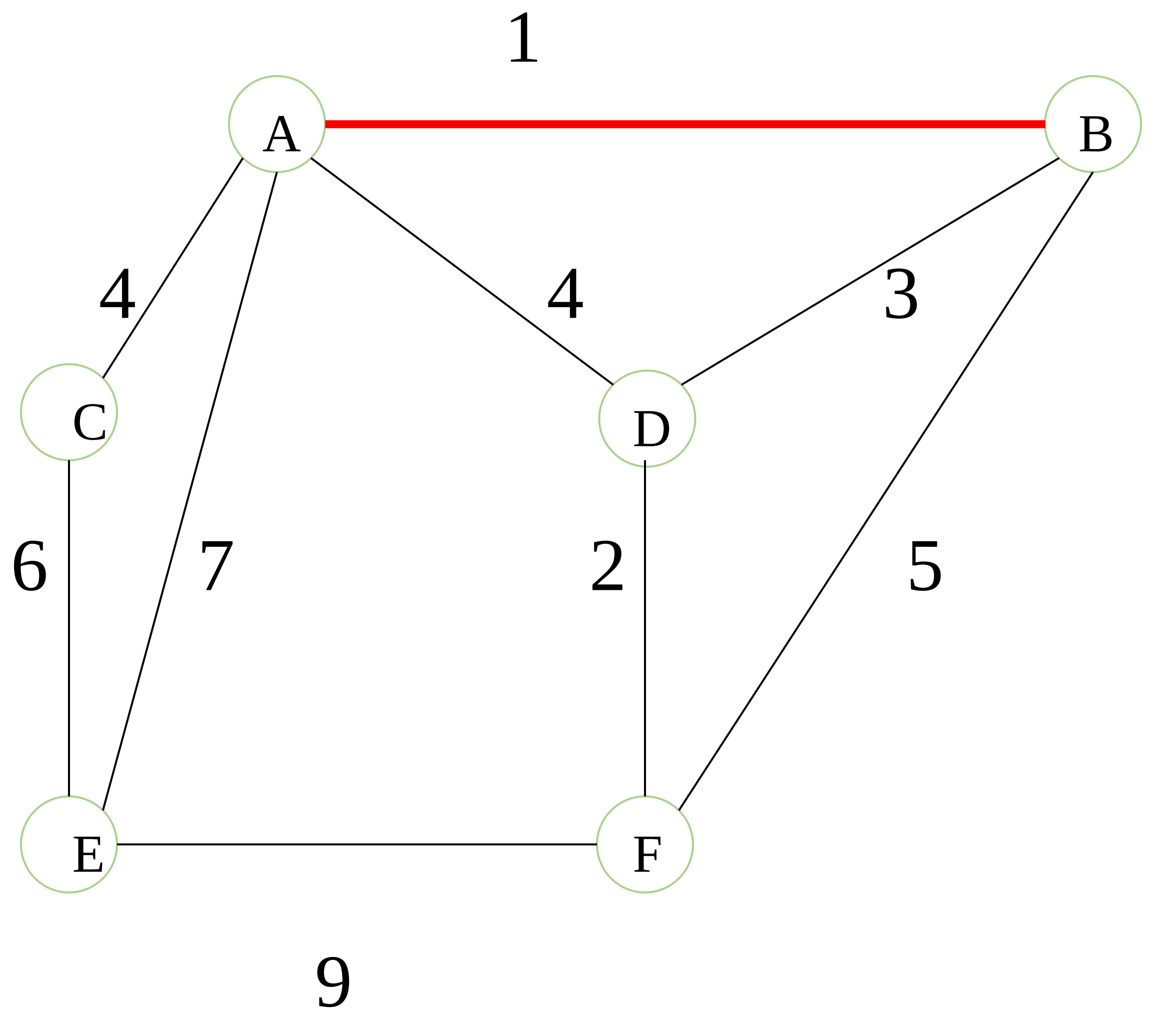
Sort the edges.



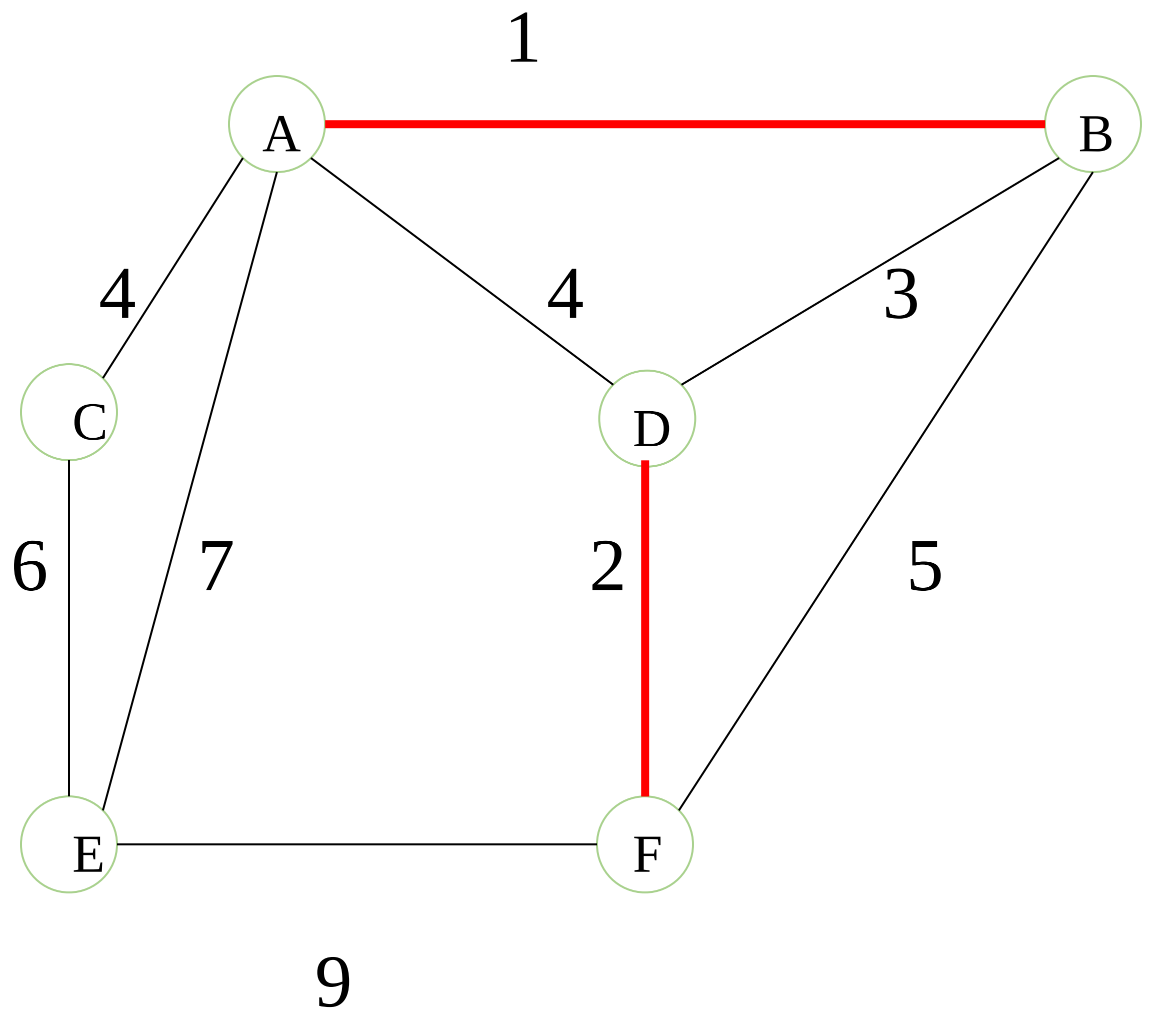
Sort the edges

AB	1
DF	2
BD	3
AC	4
AD	4
BF	5
CE	6
AE	7
EF	9

Select edge with smallest cost -- AB



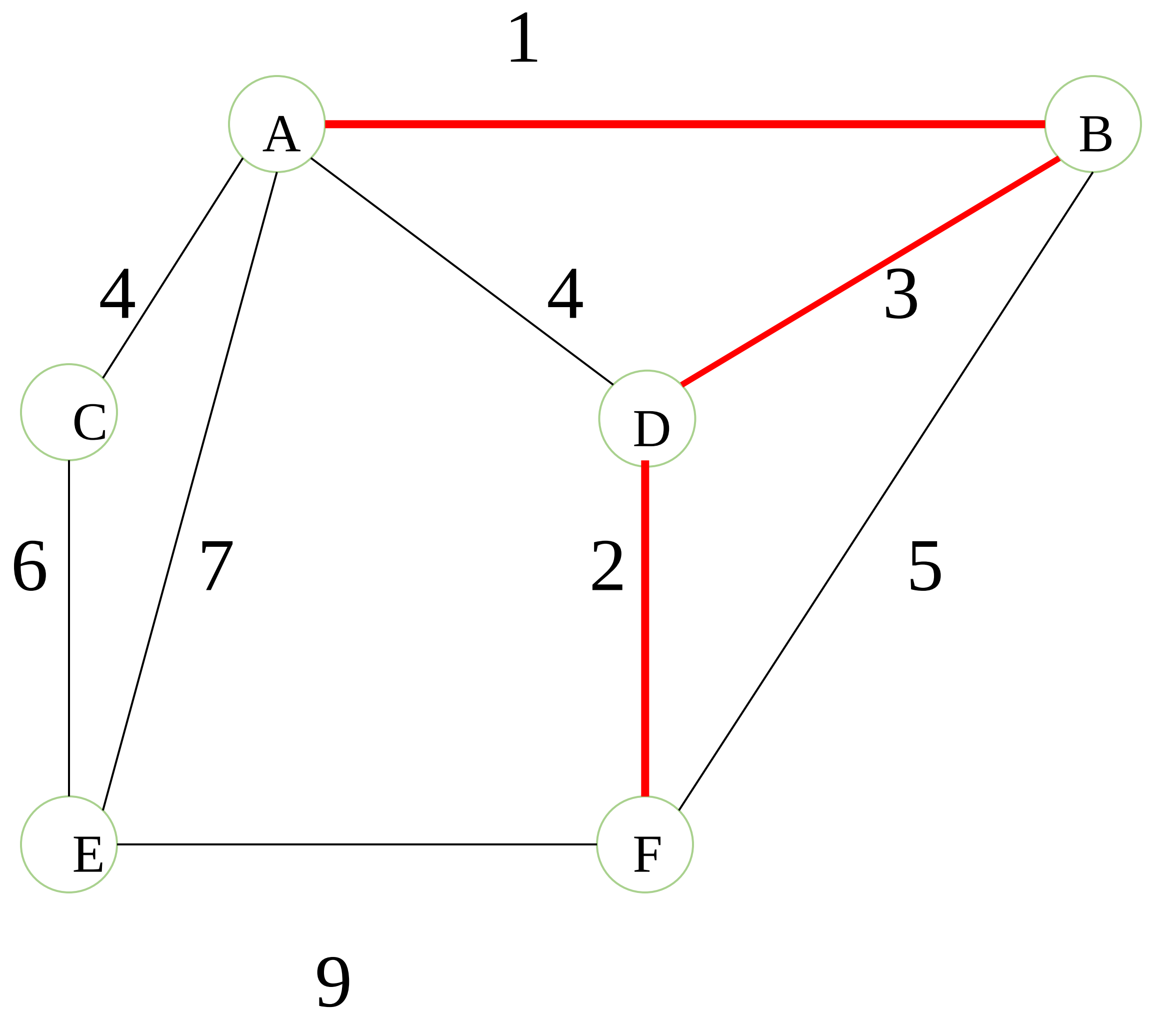
Next select DF.



Sort the edges

AB	1
DF	2
BD	3
AC	4
AD	4 XX
BF	5 XX
CE	6
AE	7
EF	9

Next select BD.



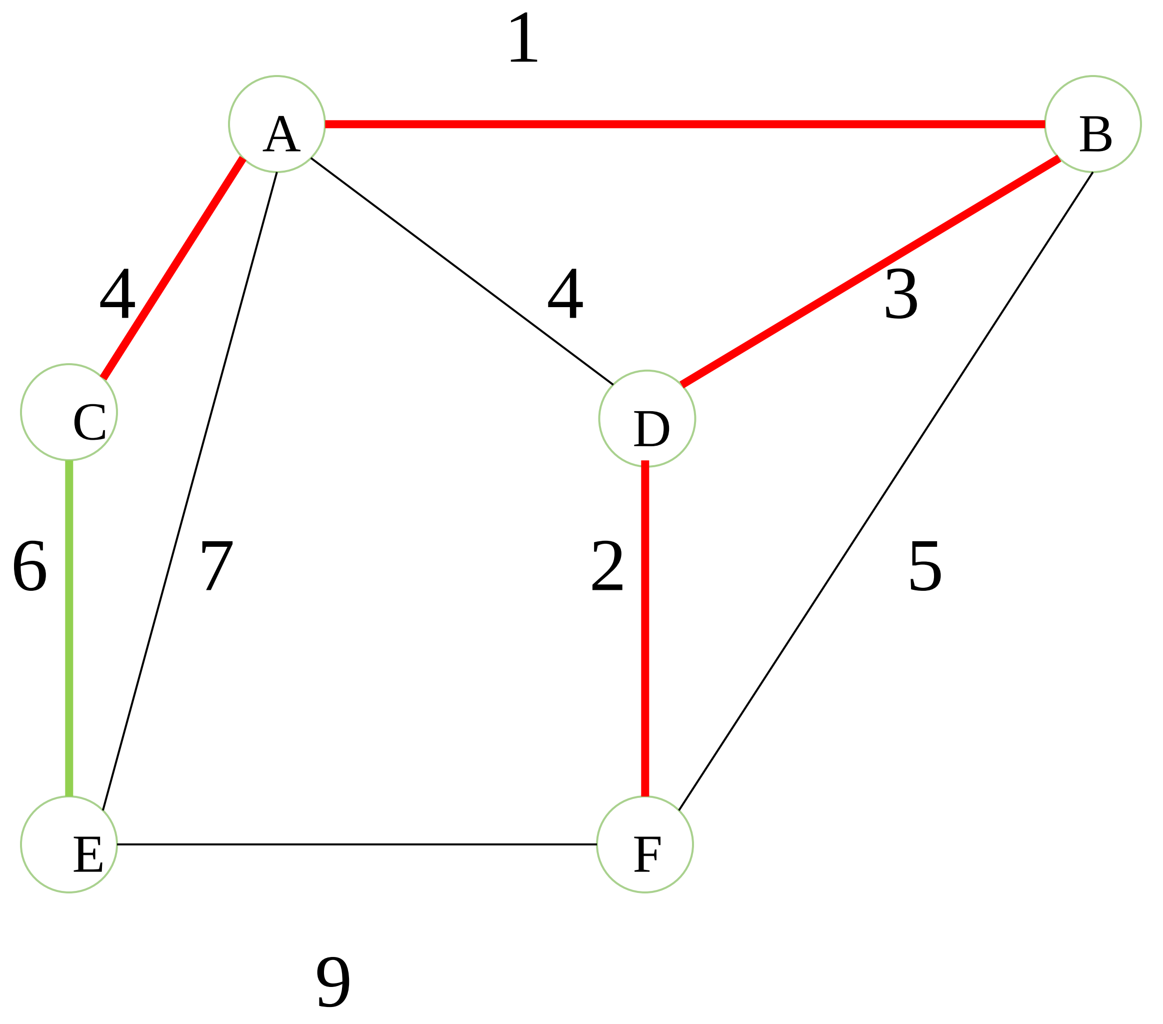
Sort the edges

AB 1
DF 2
BD 3

AC 4
AD 4 XX
BF 5 XX

CE 6
AE 7
EF 9

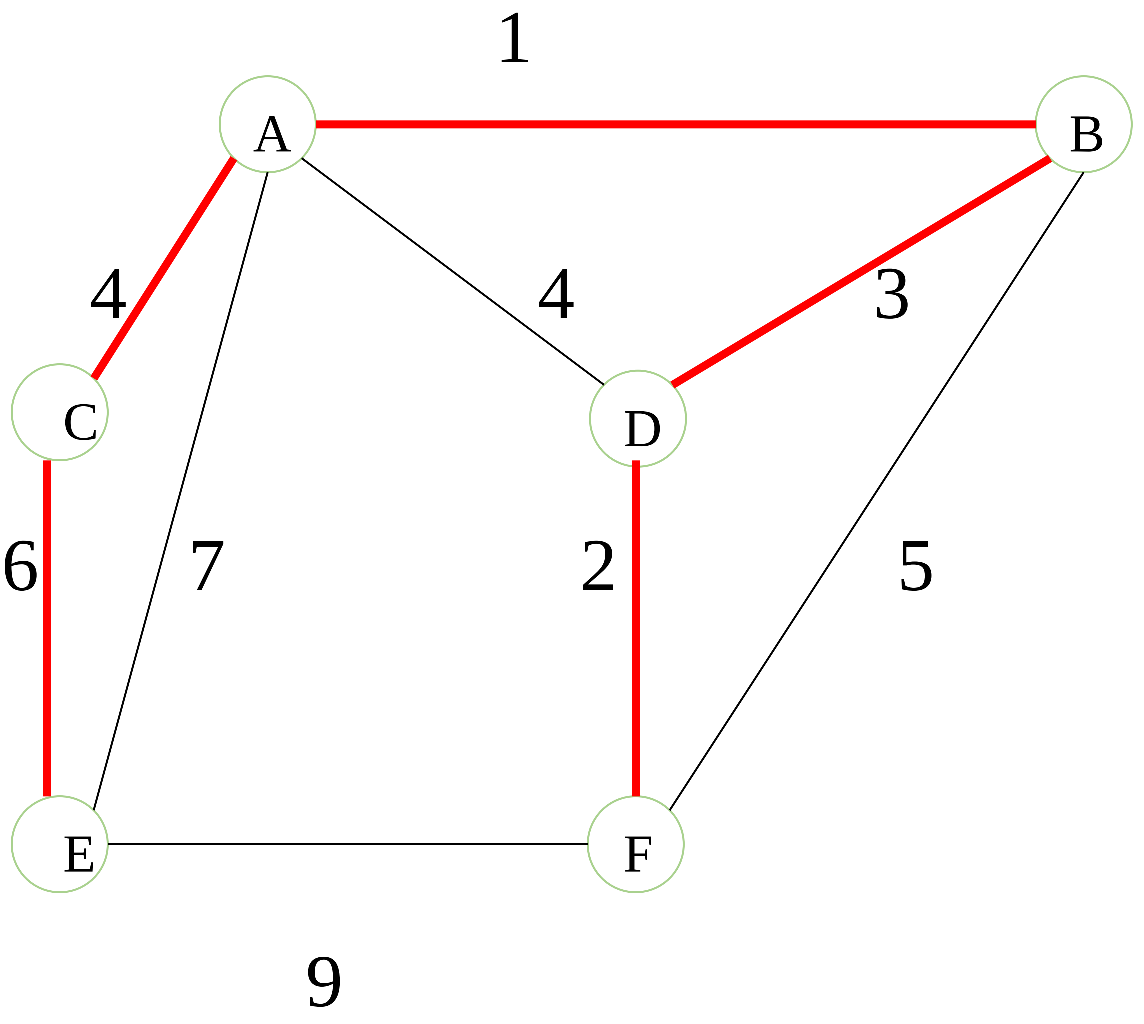
Next select AC, as AD creates a cycle and is rejected.



Sort the edges

AB	1
DF	2
BD	3
AC	4
AD	4 XX
BF	5 XX
CE	6
AE	7
EF	9

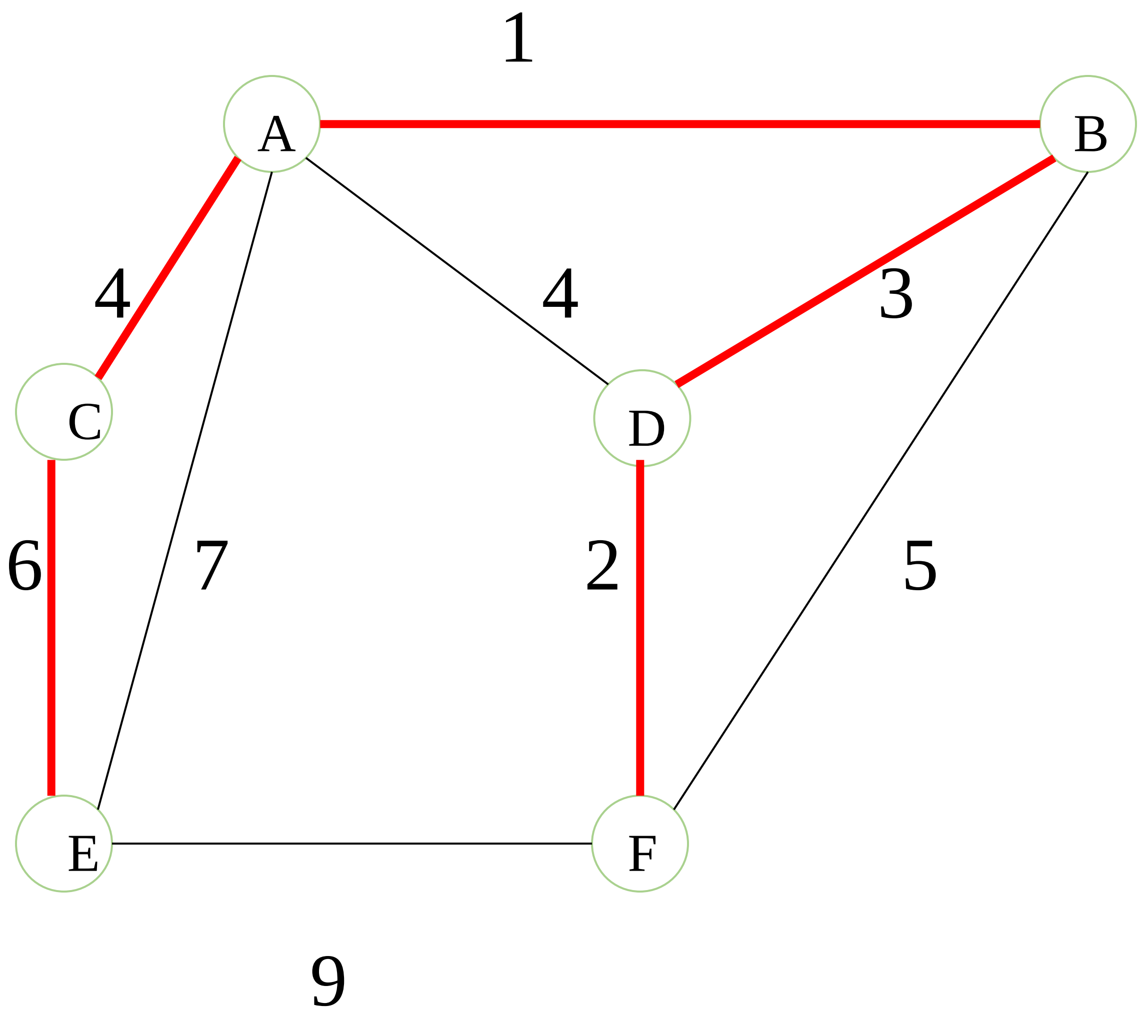
BF is rejected as it creates a cycle. Next select CE.



Sort the edges

AB	1
DF	2
BD	3
AC	4
AD	4 XX
BF	5 XX
CE	6
AE	7
EF	9

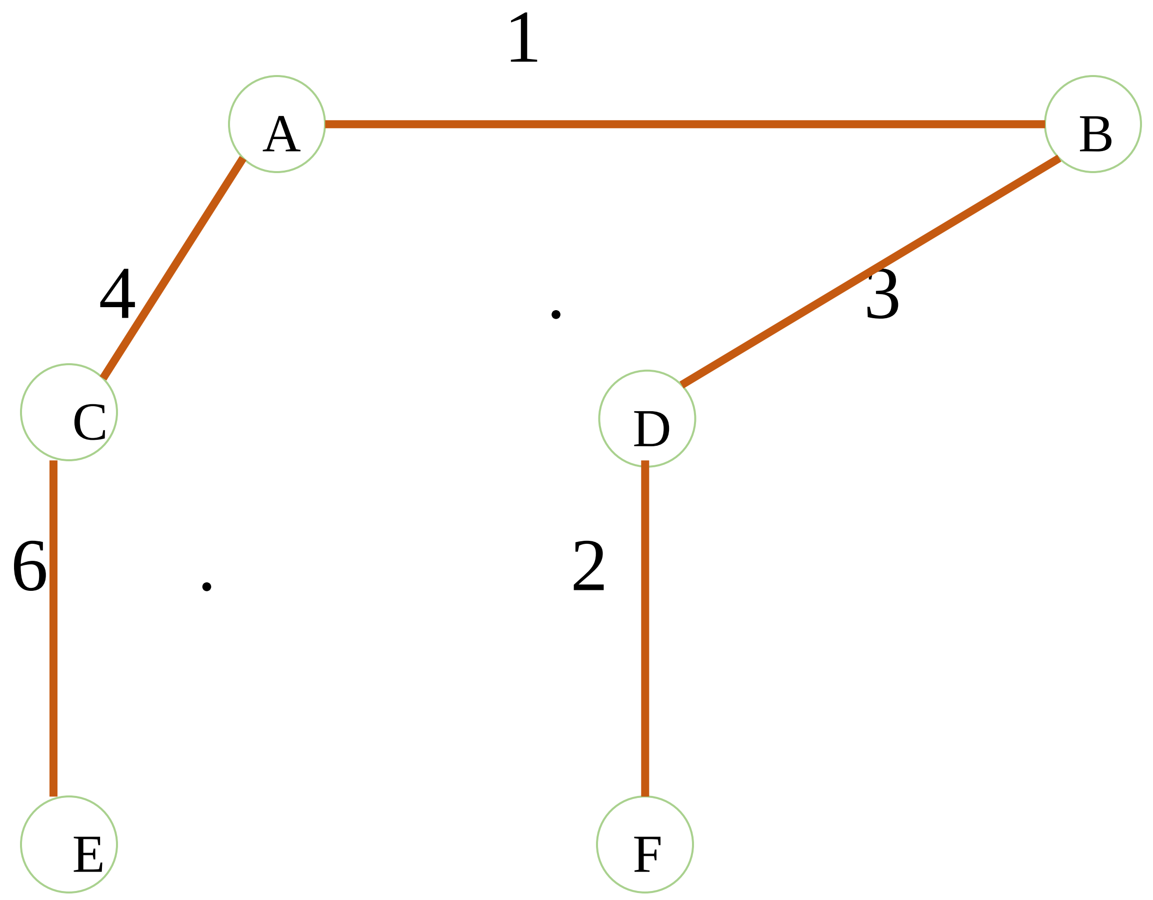
Both AE and EF are rejected as they create cycles. All edges have been considered.

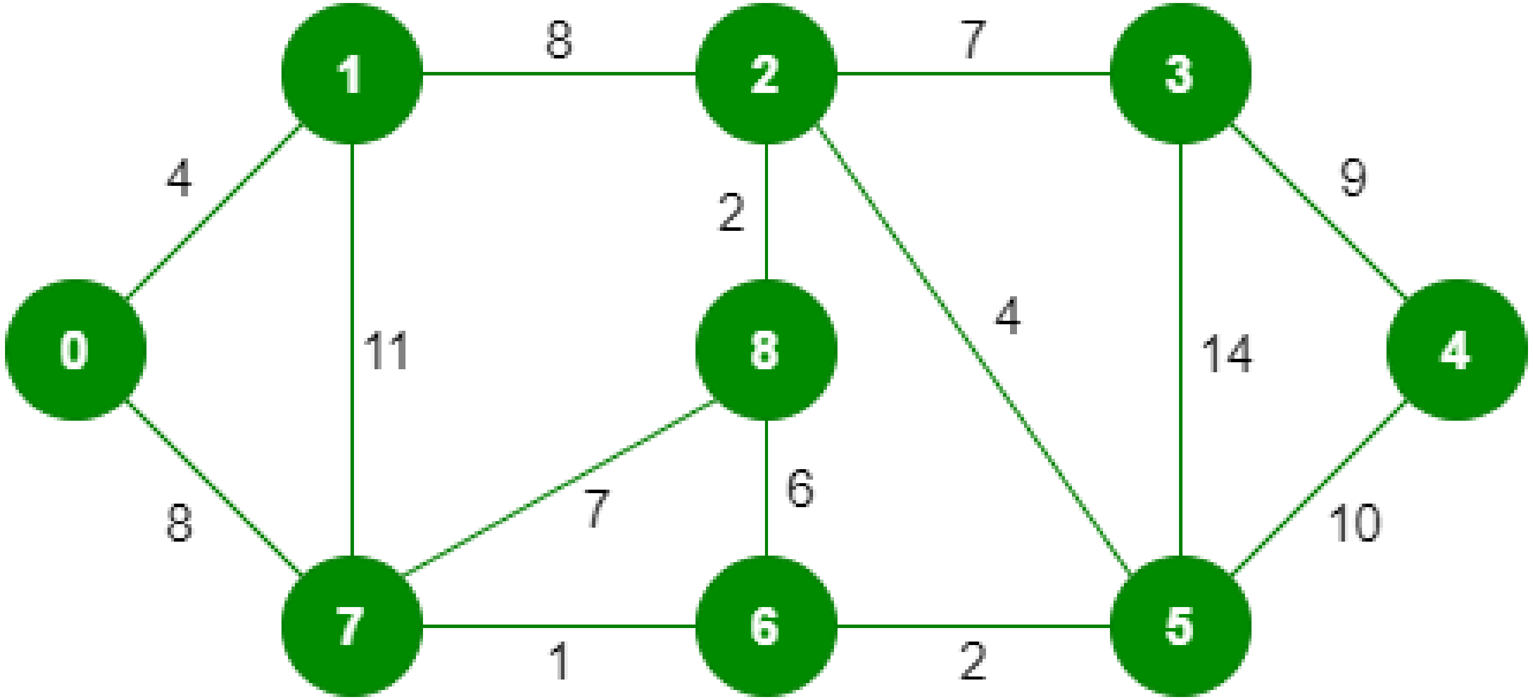


Sort the edges

AB	1
DF	2
BD	3
AC	4
AD	4 XX
BF	5 XX
CE	6
AE	7
EF	9

We got (6 – 1) edges on the tree. This is the MST. Total weight : 16 which is minimum cost





1
2
3
0
8
4
7
6
5
8
7
4
2
9
11
4
14
8
7
6
10
1
2