Trees

-> Varient of graph

-> Non linear data structure

-> acyclic graph

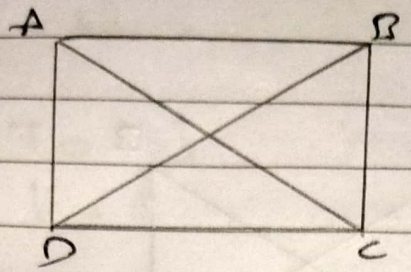
↳ There is a unique single path between any two vertices

Basic Terminologies

1. Root : starting node (A)
2. Terminal Nodes : End points of tree (leaf nodes)

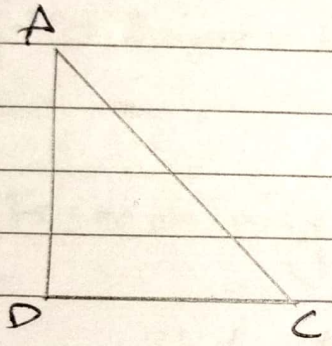
Spanning Tree

Graph बाट Tree बनाईदा सबै Vertices एक हुने



$V_G = \{A, B, C, D\}$

$E_G = \{AB, AC, AD, BC, BD, CD\}$



$V_H = \{A, C, D\}$

$E_H = \{AD, AC, DC\}$

$V_H \subseteq V_G$

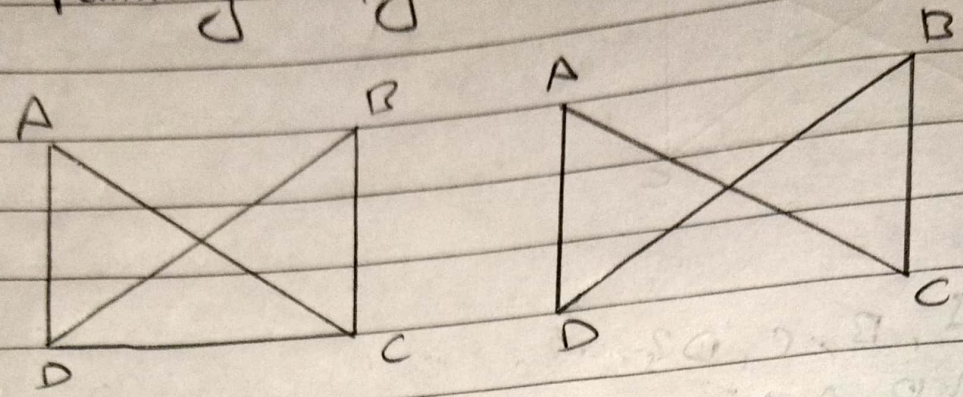
$E_H \subseteq E_G$

H is a subgraph of G

$V_H \neq V_G$

$E_H = E_G$

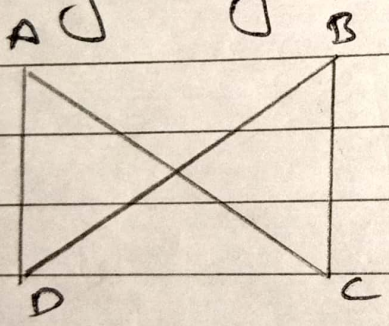
Spanning subgraph

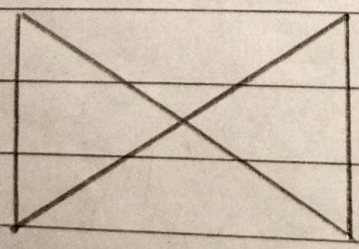


$V_H = V_G$

$E_H = E_G$

Spanning subgraph



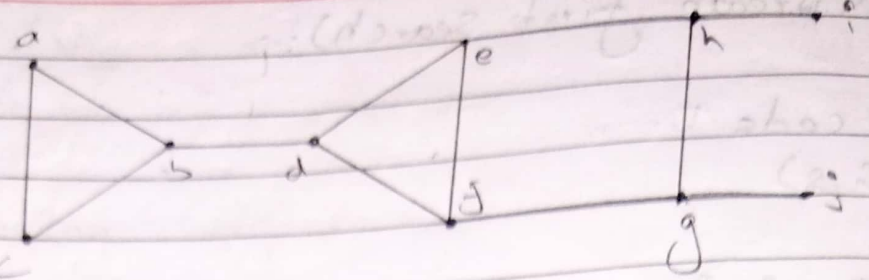


BFS (Breath first search):

Pseudo code :

```
BFS(G,s)
{
    T = {s}        // starting vertex
    L = ϕ
    Enqueue(L,s). (Source vertex को List मा राख्ने) //Insert
    while(L != ϕ)
{
  V = Dequeue(L); (List बाट पहिलो vertex झिक्ने)
                  // Delete one element from L

  For each neighbour w to v
{
      If (w ∉ L and w ∉ T)
    {
        Enqueue
        Enqueue(L,w)
        T = T ∪ {w}
    }
}
}
```



Step 1:

$T = \{a\}$

$L = \{\}$

$L = \{a\}$

$v = a, L = \{\}$

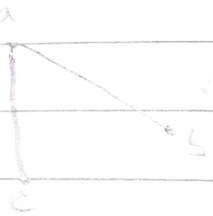
$L = \{b\}$

$T = \{a, b\}$

For another neighbour c

$L = \{b, c\}$

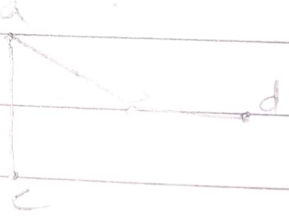
$T = \{a, b, c\}$



step 2

$V = b, L = \{c\}$

$L = \{c, d\}, T = \{a, b, c, d\}$



step 3

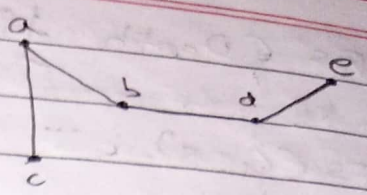
$V = c, L = \{d\}$

Both the neighbours of c are in the tree.

step 4:

$v = d$, $L = \{\}$

$L = \{e\}$, $T = \{a, b, c, d, e\}$



step 5:

$v = e$, $L = \{\}$

$L = \{e, f\}$, $T = \{a, b, c, d, e, f\}$

step 6:

$v = f$

$L = \{g\}$, $T = \{a, b, c, d, e, f, g\}$

step 7:

$v = g$

$L = \{h, j\}$, $T = \{a, b, c, d, e, f, g, h, j\}$

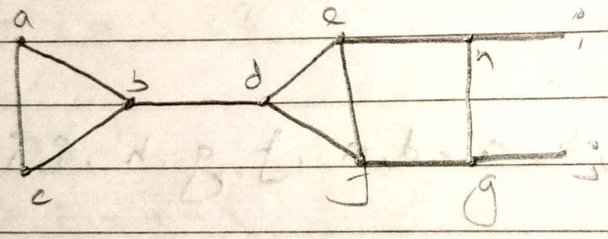
step 8:

$v = h$

$L = \{i\}$, $T = \{a, b, c, d, e, f, g, h, i, j\}$

DFS (Depth first search

```
DFS (G, s)
{
        T = {s}
        Traverse (s);
}
Traverse (v)
{
        For each w adjacent to v and not
          yet in T
        {
            T = T U {w}; // put edge {v, w} in tree
            Traverse (w);
        }
}
```



Let s = a i.e., considering a as the source vertex.

T = {a}

V = a

w = b

T = {a, b}

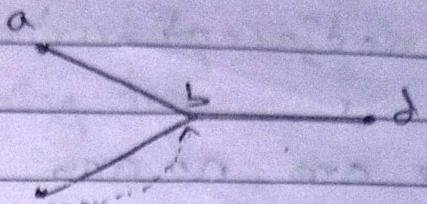
V = b

w = c

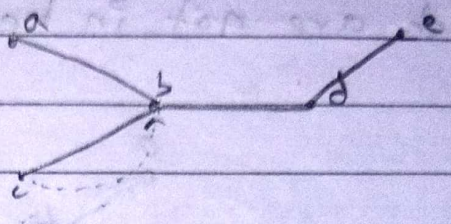
T = {a, b, c}

v = c

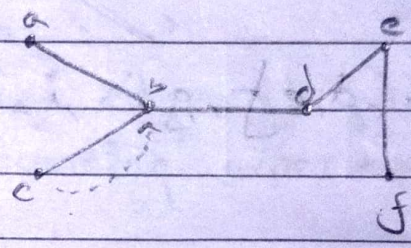
∴ There are no any adjacent vertices of c which are not in tree, we proceed to next adjacent vertex of b.



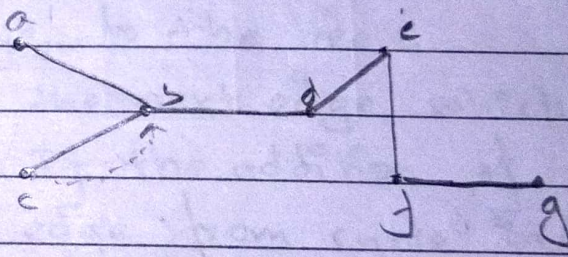
v = b
w = d
T = {a, b, c, d}



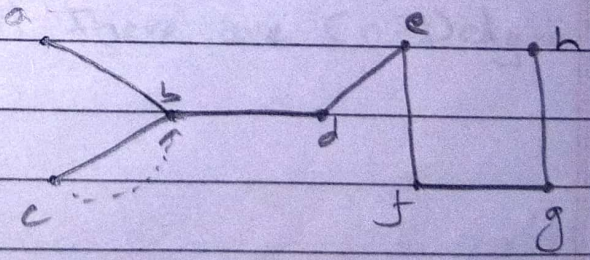
v = d
w = e
T = {a, b, c, d, e}



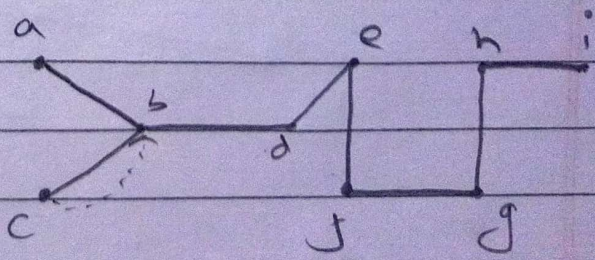
v = e
w = f
T = {a, b, c, d, e, f}



v = f
w = g
T = {a, b, c, d, e, f, g}



v = g
w = h
T = {a, b, c, d, e, f, g, h}

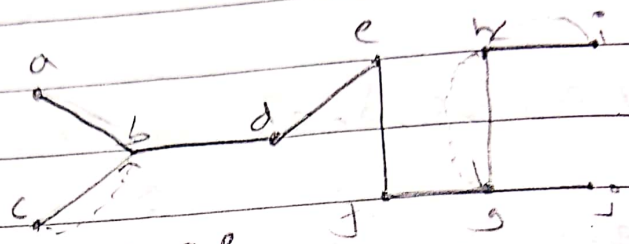


v = h
w = i
T = {a, b, c, d, e, f, g, h, i}

$v = i$

∴ There are no any adjacent vertices of i which are not in tree. We proceed to find non-visited adjacent vertex of b.

Again, there are no any adjacent vertices of h which are not in tree. We proceed to g.



$v = g$

$w = j$

$T = \{a, b, c, d, e, f, g, h, i, j\}$

Minimal spanning Tree

The spanning tree with minimum total cost made out from the given ~~big~~ weighted graph is known as minimal spanning tree

- Kruskal's Algorithm
- Prim's Algorithm

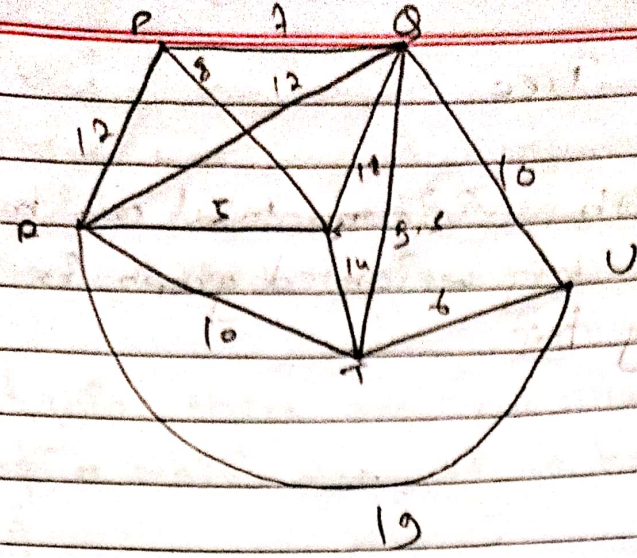
Kruskal's Algorithm

Step 1: Arrange all the edges of provided ~~un~~ weight graph in ascending order based on their weights.

Step 2: Add first edge to the tree.

Step 3: Proceed to add the next edge which was sorted in step 1. If the addition of current selected edge from cycle in the tree then discard it

Step 4: Repeat step 3 until there are (n-1) edges in tree



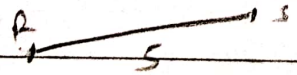
Step 1:

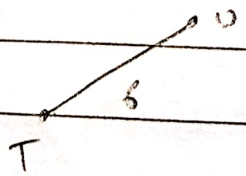
Edges	RS	TU	PQ	PS	QT	QU	RT	RQ	PR
weight	5	6	7	8	9.5	10	10	12	12

QS	RU
18	19

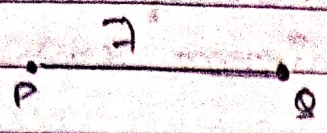
Step 2:

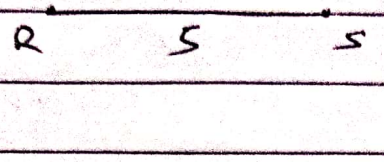
Step 3:

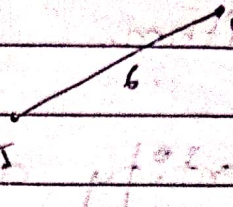




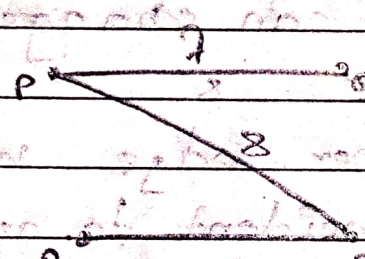
step 4:

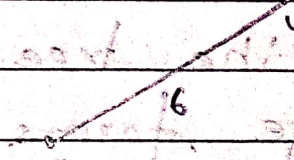




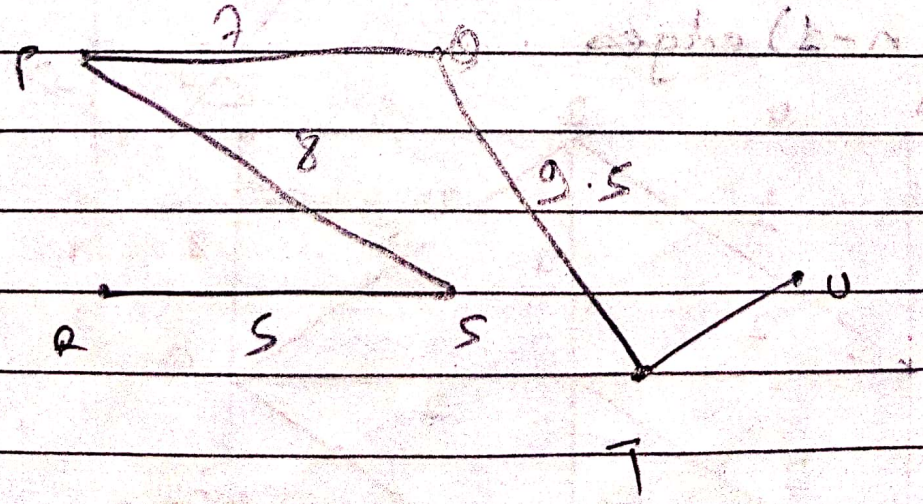


step 5:





step 6:



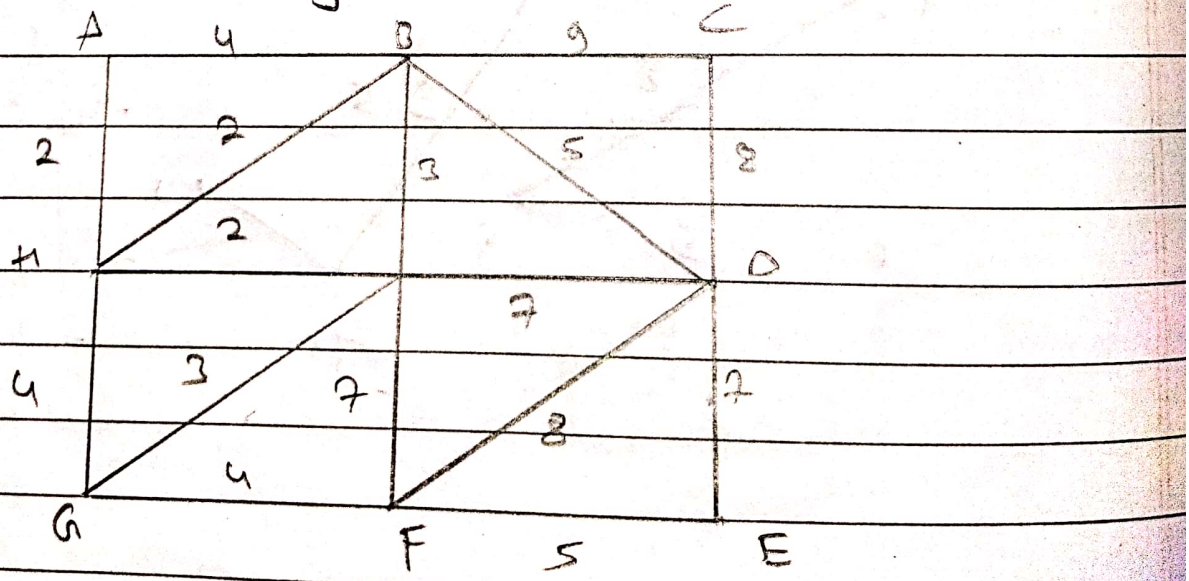
Since the tree contains (n-1) edges i.e. 5 edges, we can stop our process here. And the tree has minimum cost of
5+6+7+8+9-5 = 35.5

Prim's Algorithm

Step 1 :- Identify an edge with minimum cost and add it to tree. (If there exist more edges with same minimum cost then we can change choose any one of it).

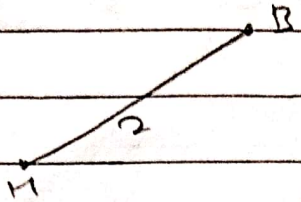
Step 2 : Identify another edge with minimum cost that is incident to any vertices in the tree. If addition of such edge forms cycle then discard it and proceed to next edge.

Step 3 : Repeat step 2 until tree contains (n-1) edges.



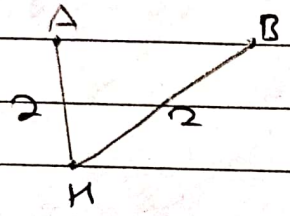
step 1:

There are those three edges with same minimum cost so, we can choose any one. So, we choose BH

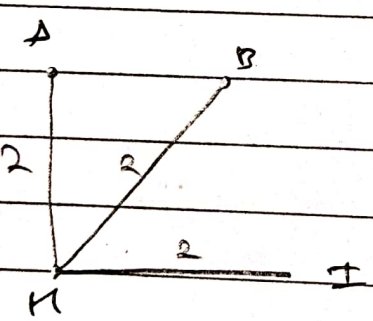


step 2:

There are two edges with same minimum cost of 2 they are AH and HI. So, we can choose any one.



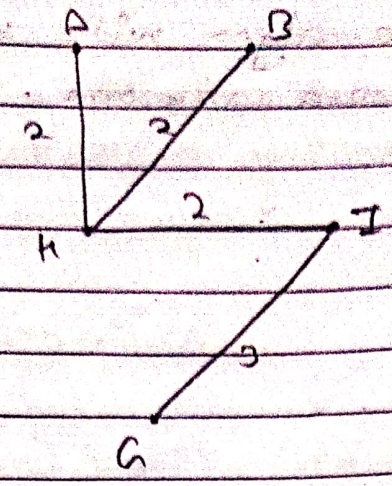
step 3:



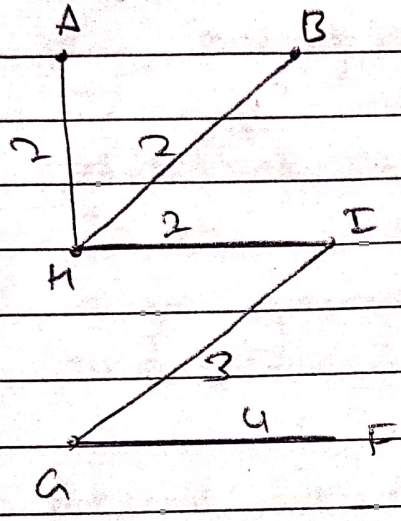
Step 4:

Since, BI forms cycle, so we cannot add this.

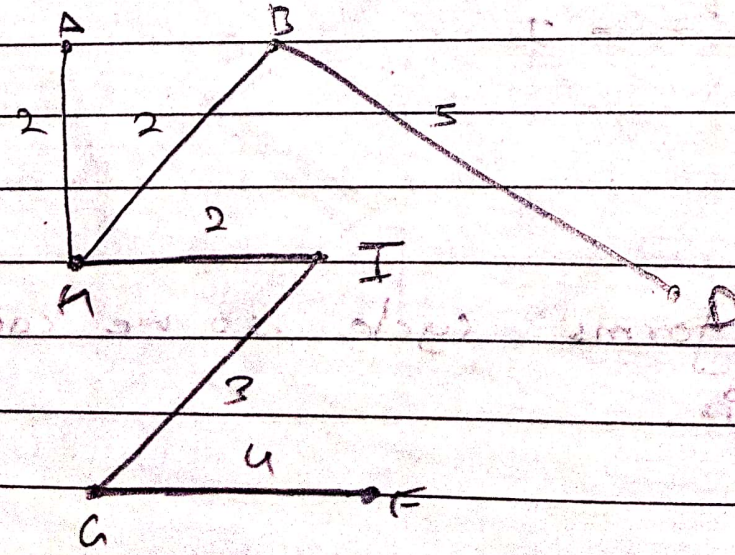
step 5 :



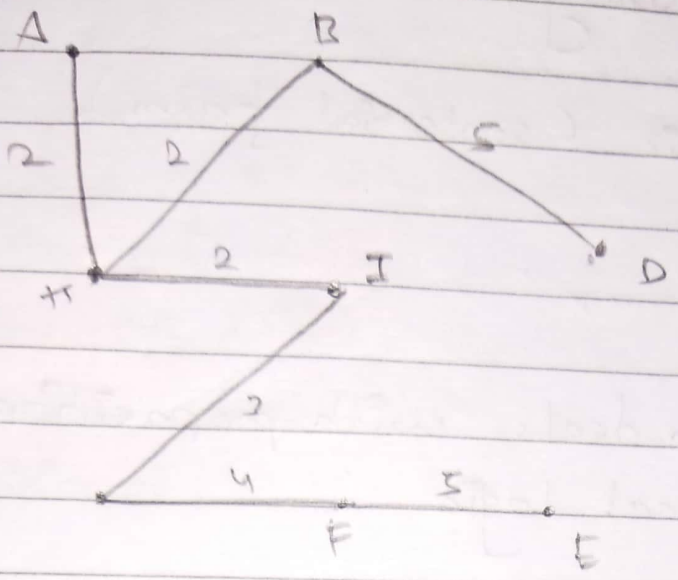
step 6 :



step 7 :-



step 8:-



step 9:-