Unit - IV

Greedy Method

→ It is used to solve the optimization problems.

→ difference in Dynamic programming and Greedy Method is

→ DP → Over lapping Subproblems.

greedy → no Overlapping Subproblems.

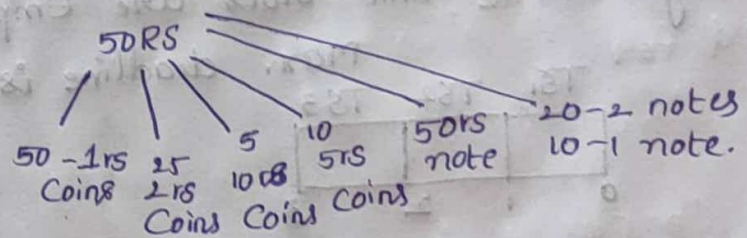
→ DP – Optimal Solution is guarantee.

greedy → no guarantee for Optimal Solution.

→ Set of all possible solutions are feasible solution.

→ optimal solution → The feasible solution which maximizes or minimizes the given Constrains.

Ex: 50 rs from your friends with min no. of notes.



→ The optimal solution is 50 rs notes.

Applications of Greedy method:

1. fractional knapsack problem.
2. Job sequencing with deadlines.
3. minimum Cost Spanning Trees.
   1. prims algorithm
   2. Kruskals Algorithm.
4. Dijkstra's Algorithm or Single Source Shortest path.

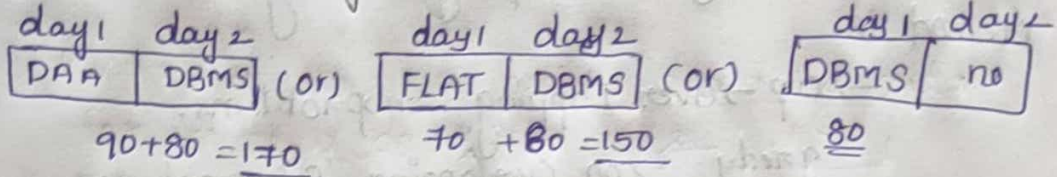
Job Sequencing with deadlines :

→ Set of n jobs given each job deadline given profit.

→ get the max profit by executing the jobs, on or before the deadline.

Ex :

	Deadlines	profit
DAA	- 1 day	90M
FLAT	- 1 day	70M
DBMS	- 2 days	80M.



90+80 = 170 70 + 80 = 150 80

170, 150, 80 → Feasible Solutions.

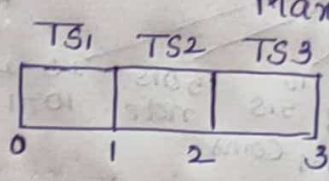
→ optimal solution is 170.

Ex ① :

n	D	P
1	3	100
2	1	10
3	2	15
4	1	27

Sol: Initially all the time slots are empty

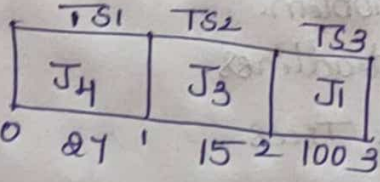
Max. deadline is 3



Arrange the jobs in profit decreasing order.

3 1 2 1 → deadlines.

J_1 J_4 J_3 J_2 → Jobs



100 + 15 + 27 = 142

Max. profit = 142.

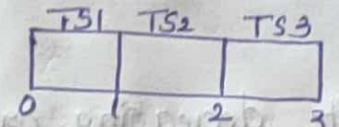
Job sequency is J_4, J_3, J_1

Ex-2 :

	n	D	P
$J_1 \rightarrow$	1	2	60
$J_2 \rightarrow$	2	1	100
$J_3 \rightarrow$	3	3	20
$J_4 \rightarrow$	4	2	40
$J_5 \rightarrow$	5	1	20

Sol: Initially all the time slots are empty

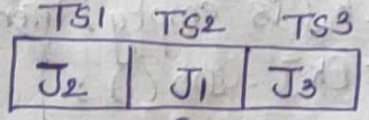
→ maximum deadline is 3,



Arrange the jobs in profit-decreasing order

$\underline{J_2}$ $\underline{J_1}$ $\underline{J_4}$ $\underline{J_3}$ $\underline{J_5}$ → Jobs.

1 2 2 3 1 → deadlines



100 + 60 + 20 = 180.

max profit = 180

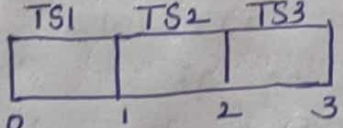
Job sequency is J_2, J_1, J_3.

Ex :-3 ⇒

n	D	P
1	2	100
2	1	19
3	2	27
4	1	25
5	3	15

Sol: Initially all the time slots are Empty.

→ max deadline is 3



Arrange the jobs in profit - decreasing order.

$\underline{J_1}$ $\underline{J_3}$ $\underline{J_4}$ $\underline{J_2}$ $\underline{J_5}$ → Jobs.

2 1 2 1 3 → deadlines.

TS1	TS2	TS3
J3	J1	J5
27	100	15

$27 + 100 + 15 = 142.$

Max profit = 142

Job sequency is J_3, J_1, J_5

Algorithm For Job Sequencing deadlines :

```
Algorithm JobSequencing ( )
{
    // Arrange Jobs in profit decreasing order.
    for i=1 to n do          (n: no. of Jobs.)
    k=min (d max , deadline(i));
    while (k>=1) do
        if (timeslot[k] is empty) then.
            timeslot[k] = Job (i) ;
            break ;
        end if
        k = k-1 ;
    end while ;
end for
}.
```

Time Complexity :

Time Complexity for Job Sequencing with deadlines. $O(n^2)$

Ex-2 :

n	D	P
1	2	60
2	1	100
3	3	20
4	2	40
5	1	20

Tracing

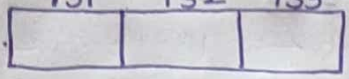
Tracing of Job Sequence Algorithm:

Step-1 : Arrange the Job profits in decreasing order

P ⇒	100	60	40	20	20
n ⇒	2	1	4	3	5
D ⇒	1	2	2	3	1

Job Sequence → J_2, J_1, J_4, J_3, J_5

Time Slots :



Algorithm JobSequencing

{

for i=1 to n do

i=1 // J2

k = min (3,1) ⇒ k=1

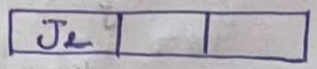
while (1>=1) T

if (timeslot [1] is Empty) T

timeslot [1] = Job(1)

// timeslot [1] = J2

break;



i=2 // J1

k = min (3,2) ⇒ k=2

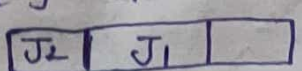
while (2>=1) T

if (timeSlot[2] is empty) T

timeslot[2] = Job(2)

// timeslot[2] = J1

break;



i=3 // J4

k = min (3, 2) ⇒ k=2

while (2>=1) T

if (time slot [2] is empty) F

{

}

k = 2-1 = 1

while (1>=1) T

if (time slot [1] is empty) F

{

}

k = 1-1 = 0

while (0>=1) F

J2	J1	

i=4 // J3

k = min (3, 3) ⇒ k = 3

while (3>=1) T

if (timeslot [3] is empty) T

timeslot [3] = Job(4)

//timeslot [3] = J3

break;

J2	J1	J3

i=5 // J5

k = min (3, 1) ⇒ k=1

while (1>=1) T

if (timeslot [1] is empty) F

{

}

k = 1-1 = 0

while (0>=1) F

i=6 , F

∴ Job sequence is

TS1	TS2	TS3.
J2	J1	J3

Fractional knapsack problem :

Given weights and values of n items, we need to put these items in a knapsack of capacity W to get the maximum total value in the knapsack

In the 0-1 knapsack problem, we are not allowed to break items. We either take the whole item or don't take it

In Fractional knapsack, we can break items for maximizing the total value of knapsack. This problem in which we can break an item is also called the fractional knapsack problem.

Ex-1 : 0, 1, 0 → 1

n	p	w
1	25	18
2	24	15
3	15	10

M = 20

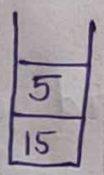
sol:

$\frac{P}{W}$	1	2	3
	1.38	1.6	1.5

We have to Arrange the objects in $\frac{P}{W}$ decreasing order

2	3	1
15	10	18
24	15	25

We have to place the 2nd object into knapsack



$5 - 5 = 0 \rightarrow 15 \times \frac{1}{2} = 7.5$

$20 - 15 = 5 \rightarrow 24$

Max profit = 7.5 + 24

= 31.5

$x_1 = 0 \quad x_2 = 1 \quad x_3 = 0.5$

Ex-2:

n	W	P
1	18	30
2	15	21
3	10	18

$m = 20$.

Sol: Arrange the objects in P/w dec order

P/w →	30/18	21/15	18/10
	1.6	1.4	1.8

1.8	1.6	1.4
3	1	2

10
10

$30 \times 0.55 = 16.67$

18

$\frac{10}{18} = 0.55$

$20 - 10 = 10$

34.67

max profit = 34.67.

Algorithm for Fractional knapsack problem:

```
Algorithm greedyknapsack (m,n)
{
    // P,W > P/W dec order
    for i=1 to n do
        x[i] = 0.0;
    U = m;
    for i=1 to n do
    {
        if (W[i] > U) then
        break;
        x[i] = 1.0;
        U = U - W[i]
    }
    if (i<=n) then
        x[i] = U/W[i];
}
```

Time Complexity :

Time complexity for fractional knapsack problem is $O(n)$

Ex-3 : find an optimal solution for $n=7$, $m=15$ and

$P[1,7] = \{10, 5, 15, 7, 6, 8, 3\}$

$w[1,7] = \{1, 3, 5, 7, 1, 4, 1\}$

n	w	P
1	2	10
2	3	5
3	5	15
4	7	17
5	1	6
6	4	18
7	1	3

$m=7$

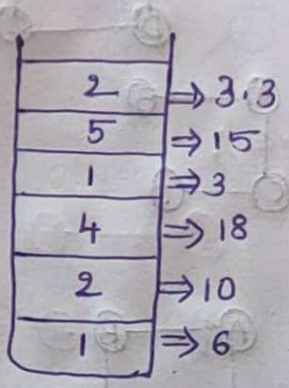
$\frac{P}{w}$:	1	2	3	4	5	6	7
	5	16	3	1	6	4.5	3

$\frac{P}{w}$:	6	5	4.5	3	3	1.6	1

object : 5 1 6 7 3 2 4

profit $= 3.3 + 15 + 3 + 18 + 10 + 6$

$= 55.3$



$\frac{2}{3} = 0.66 \times 5 = 3.3$

(or)

$1.6 \times 2 = 3.2$.

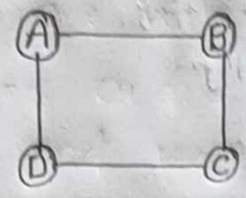
objects :	1	2	3	4	5	6	7
X	1	0.66	1	0	1	1	1

Spanning Tree :-

It is a Connected undirected graph without any cycle or any loops

→ If spanning Contains 'n' vertices. It Contains 'n-1' Edges.

Ex :



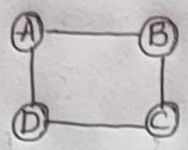
4-vertices
3-edges.

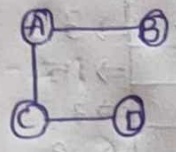
If 'n' no. of vertices then n^{n-2} Spanning Trees are possible.

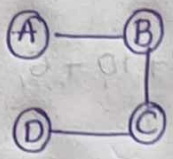
Ex :- $n=4$

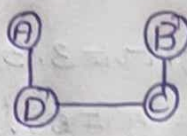
no. of possible spanning 'n' $= 4^{4-2} = 4^2 = 16$

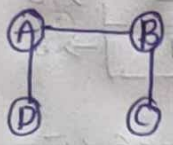
Ex :-



① 

② 

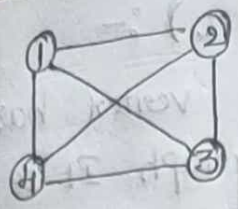
③ 

④ 

Complete graphs :

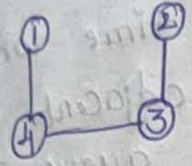
Every vertex is Connected with every vertex in the graph is Called Complete graph.

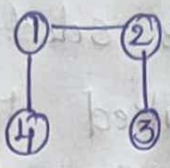
Ex:

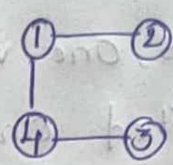


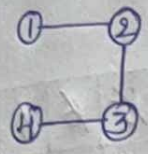
4-vertices
3-edges
Then Spanning trees are 16

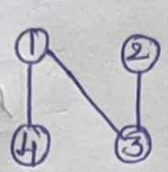
$4^{4-2} = 4^2 \Rightarrow 16$

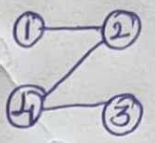
(1)


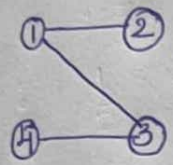
(2)


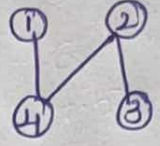
(3)


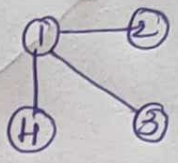
(4)


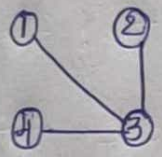
(5)


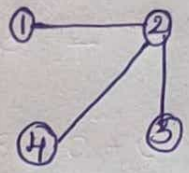
(6)


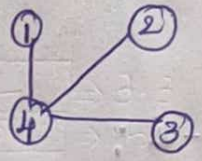
(7)


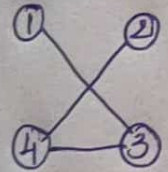
(8)


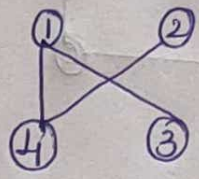
(9)


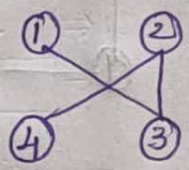
(10)


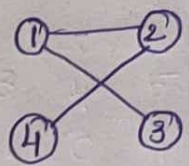
(11)


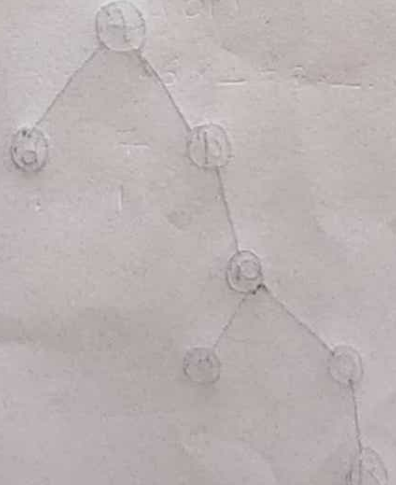
(12)


(13)


(14)


(15)


(16)


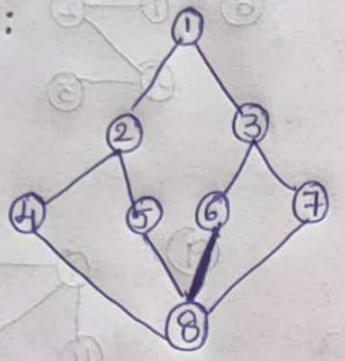


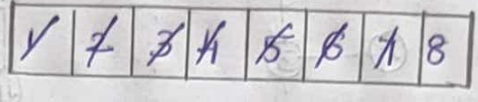
→ BFS (Breadth first spanning tree) :-

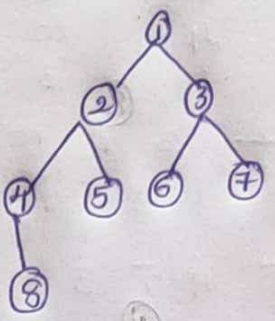
Breadth first search is a vertex based technique for finding a shortest path in graph. It is uses a Queue data structure which follows first in first out.

⇒ In BFS, one vertex is selected at a time when it is visited and marked then its adjacent are visited and ~~Marked~~ stored in the queue. It is slower than DFS.

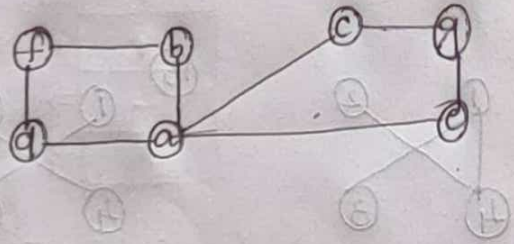
Ex-1:

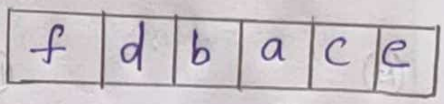


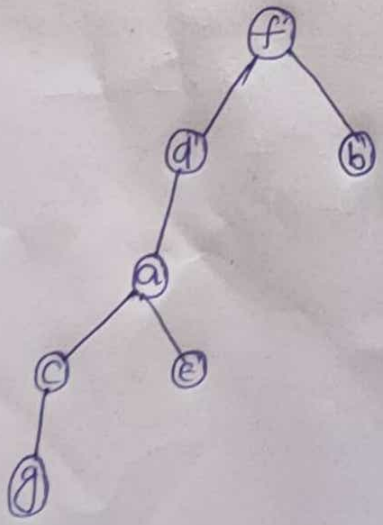




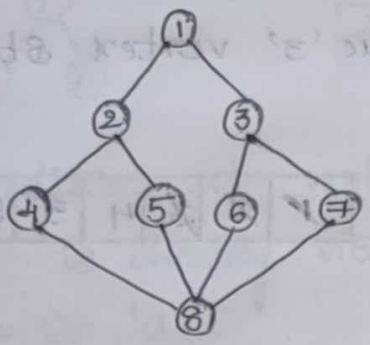
Ex-2:







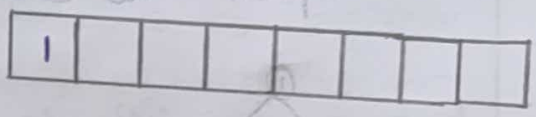
Example - 1



Step-1 : visit vertex '1'

→ push into Queue

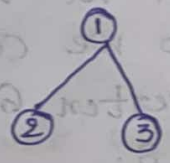
①

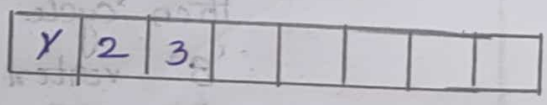


Step-2 : visit all adjacent vertexes of '1' i.e, 2 and 3

→ push into Queue.

→ One '1' vertex is already visited with respective all adjacent vertex, then it strike off it



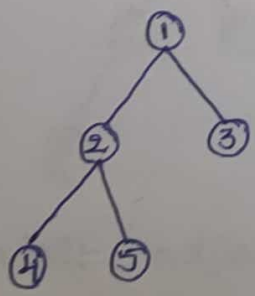


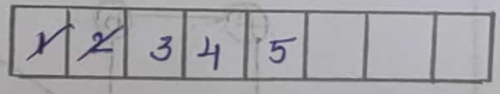
Step-3 : visit adjacent vertexes for vertex '2'

Because it is the next element into the Queue i.e, 4 and

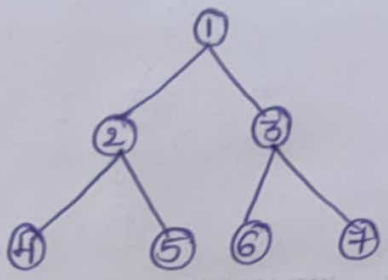
→ push into Queue

→ vertex '2' adjacent vertexes are visited So we strike off it.



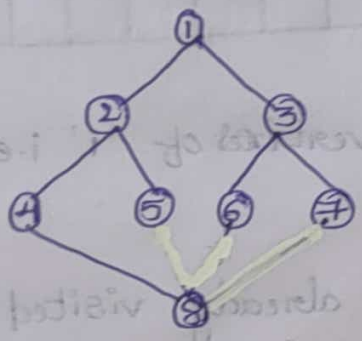


Step-4 : visit all adjacent of 3 vertex i.e, 6 and 7.
→ push into Queue '3' vertex & strike off it.



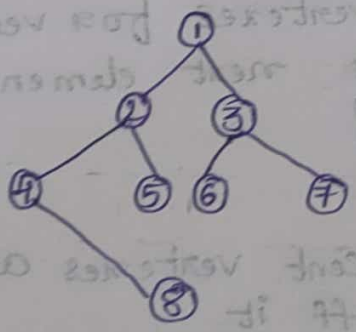
1	2	3	4	5	6	7	

Step-5 : visit all adjacent of 4 vertex. i.e., 8
→ push into Queue '4' Strike off.



1	2	3	4	5	6	7	8

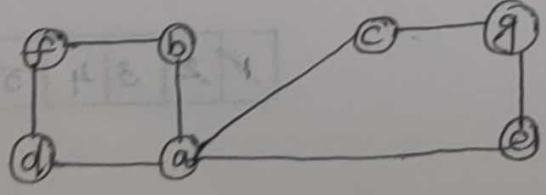
Step-6 : visit adjacent vertex of 5 i.e. 8 is already in the tree again we visit 8 then cycle is formed. So we can't visit '8' vertex similarly vertex 6 and 7 also have adjacent i.e., 8.



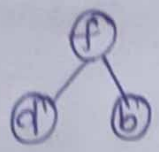
1	2	3	4	5	6	7	8

Queue is empty

Example-2 :

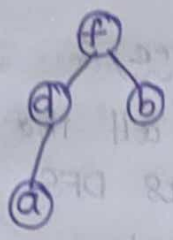


visit f



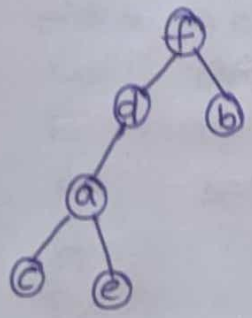
f	d	b

visit adjacent of vertex d' i.e f and a
– f is already visited. so now we need to visit 'a'



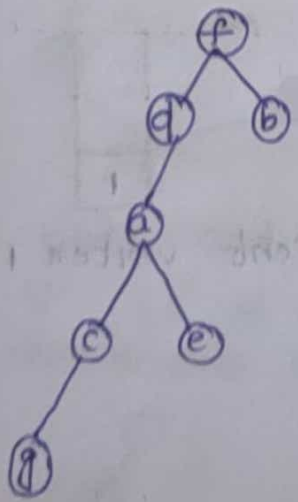
f	d	b	a

visit adjacent vertex of 'b' i.e. a and f d is already visited. so then visit the adjacent of 'a' i.e, c and e



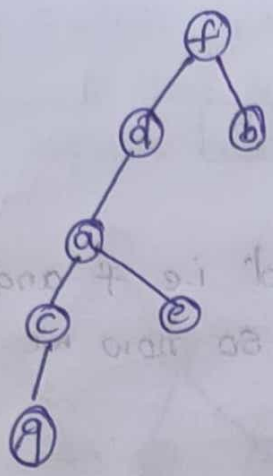
f	d	b	a	c	e

visit adjacent vertex of 'c' i.e, a and g a is already visited



f	d	b	a	c	e	g

visit adjacent vertex of 'g' i.e, c and e both are all visited so.

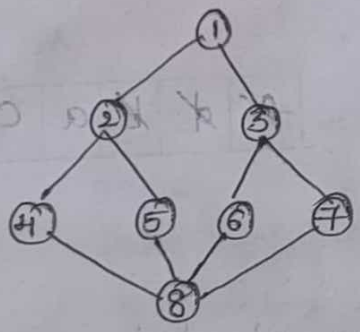


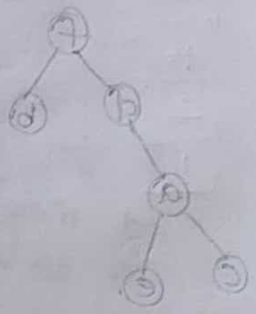
DFS (Depth first spanning tree):

In BFS we visit all the adjacent vertices to a vertex whereas DFS we only visit one adjacent vertex of a vertex.

* Height of the tree is increased
* stack.

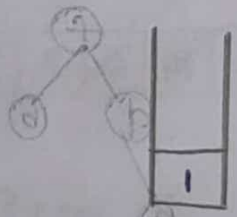
Example - 1 :



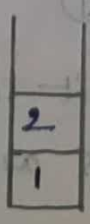


Steps :

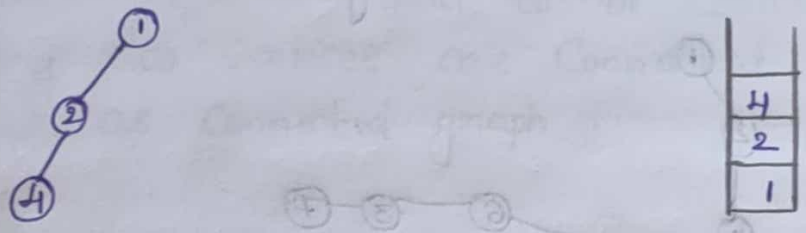
① visit vertex '1'
→ push onto a stack



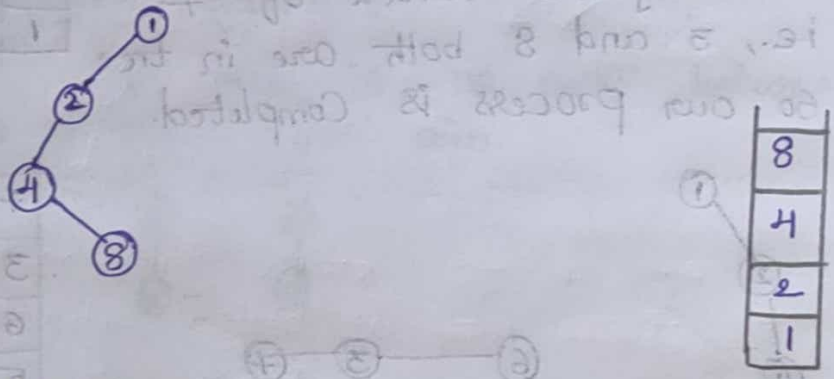
② visit & added the adjacent vertex 1 either 2 or 3. Here 2 taken
→ push onto a stack



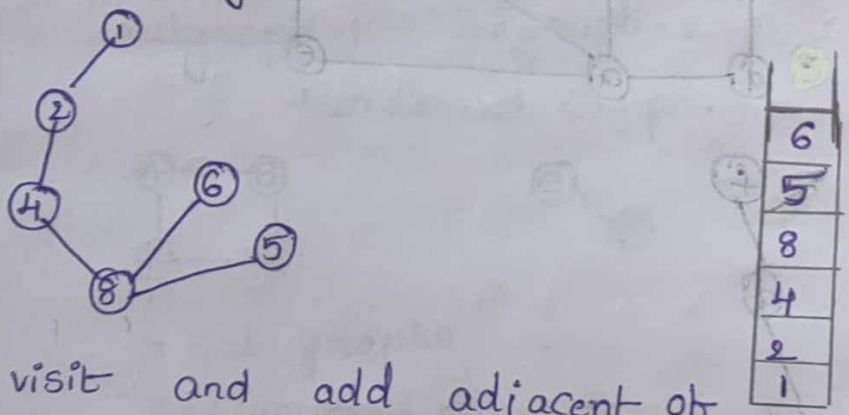
③ visit and added the adjacent of 2 i.e., 4 and 5 either 4 or 5. Here '4'



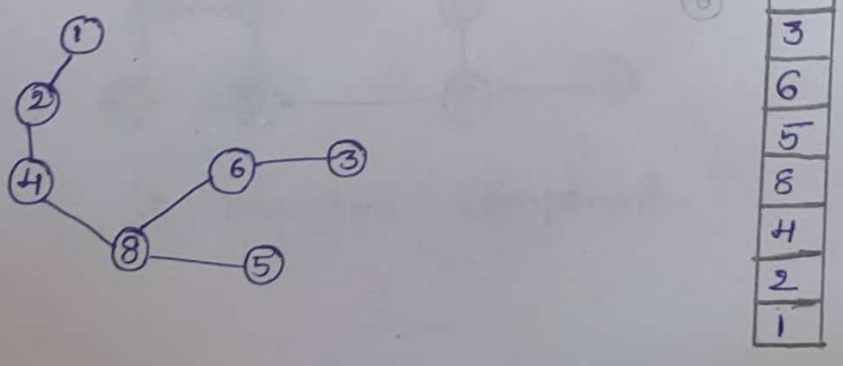
④ visit and add adjacent vertex of 4 i.e., 8
→ push onto a Stack



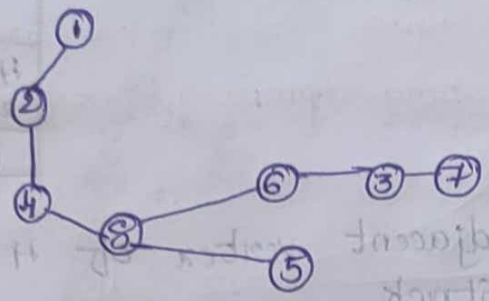
⑤ visit adjacent vertex of 8. i.e., 5, 6, 7 either 6 or 7 or 5. If we visit 5, adjacent vertex of 5 is 2 and 8 both are already in tree so no possible to visit. So we follow backtrack i.e., 8. The adjacent vertex of '8' is 4, 5, 6, 7, 4 and 5 are already in Spanning tree. So either 6 or 7. If 6

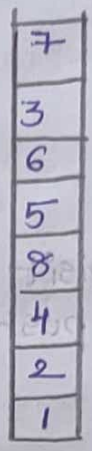


⑥ visit and add adjacent of '6' i.e, 3 and 8 already in tree so 3
→ push onto stack

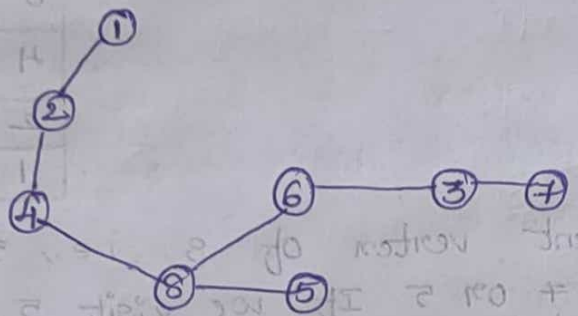


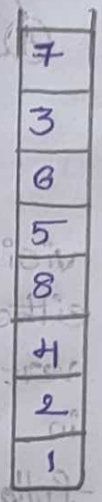
⑦ visit adjacent of '3' i.e., 1 and 6 and ⑦ 1 and 6 already in tree so 7



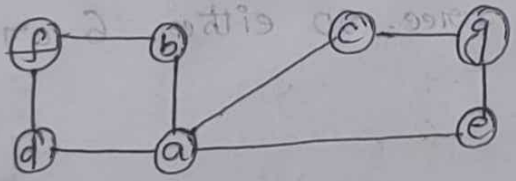


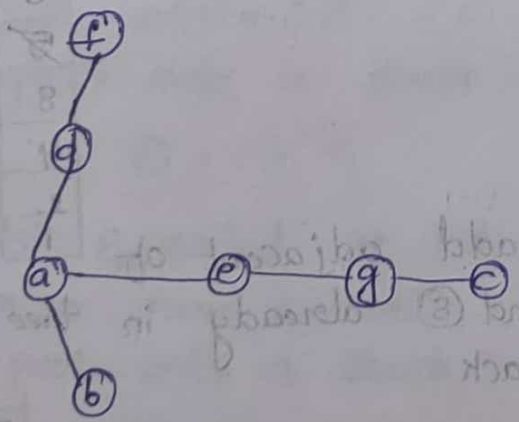
⑧ visit adjacent vertex of 7 i.e., 3 and 8 both are in tree so our process is completed.





Example – 2 :

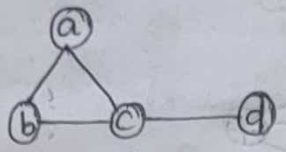




Connected Graph :

A graph is said to be Connected in which any two vertices are Connected by path is called as Connected graph.

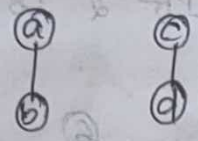
Ex :



Disconnected Graph :

If there is no path between two pairs to reach our aim.

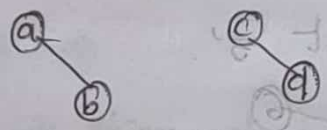
Ex :



Connect Component :-

A Connected Component of an undirected graph is a subgraph in which any two vertices are connected by path is known as Component.

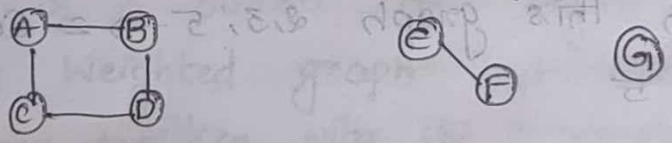
Ex ① :



Subgraph-1 Subgraph-2

two Connect Component.

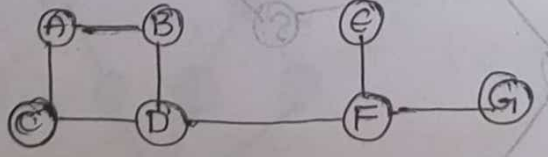
Ex ② :



3 Sub graphs

3- Connected Components

Ex ③ :

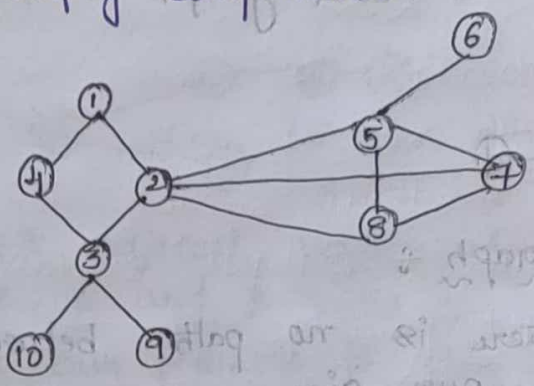


1- Connected Component.

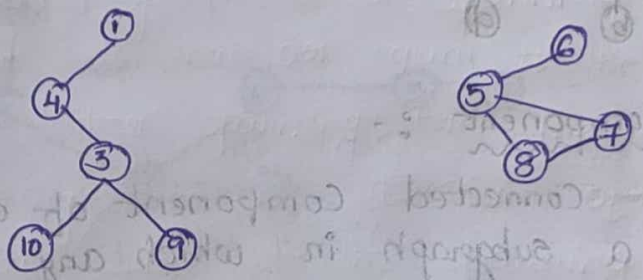
Articulation point :

It is a vertex whose deletion disconnects the graph into two or more no of non-empty Components.

Ex :

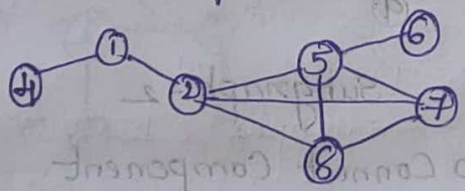


Articulation point '2', If 2 is deleted



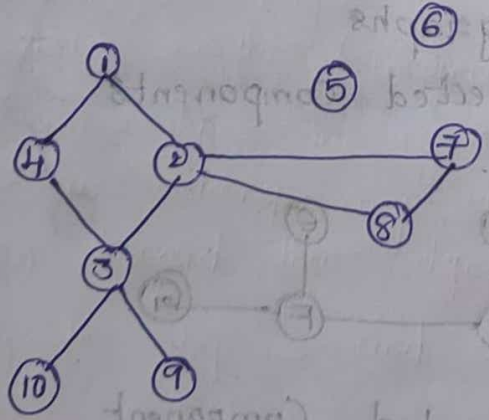
2 Connected Components

Articulation point at '3'



3 Connected Components

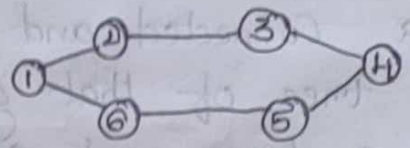
So in this graph 2, 3, 5 → 2 Connected At 5



Biconnected Components :

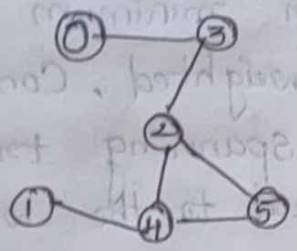
If a graph said to be Biconnected if it doesn't contain any articulation point, then it is called Biconnected components.

Ex:-



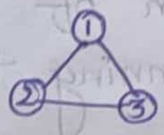
It doesnot contain any articulation points.

Ex :-

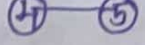


2 is articulation points.

It is not a biconnected component



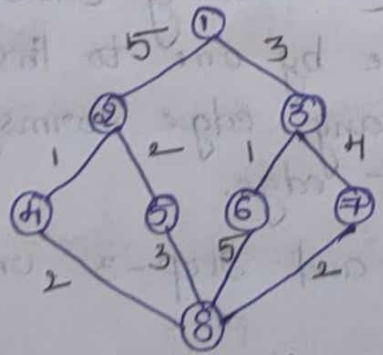
It is Biconnected Component.



It is Biconnected Component.

Minimum Cost Spanning Tree (MST) :-

MST is a subset of edges of a Connected Weighted graph that Connects all the vertex together with the minimum possible total edge weight.



How to Construct the minimum Cost Spanning tree.

There are two techniques to find out.

1. Kruskal's algorithm
2. prims algorithm.

Minimum Spanning Tree:

Given a connected and undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A minimum spanning tree (MST) or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree with a weight less than or equal to the weight of every other spanning tree. The weights of a spanning tree is the sum of weights given to each edge of the spanning tree.

A minimum spanning tree has 'n-1' edges where n is the no.of vertices in the given graph.

We have two techniques for find the MST

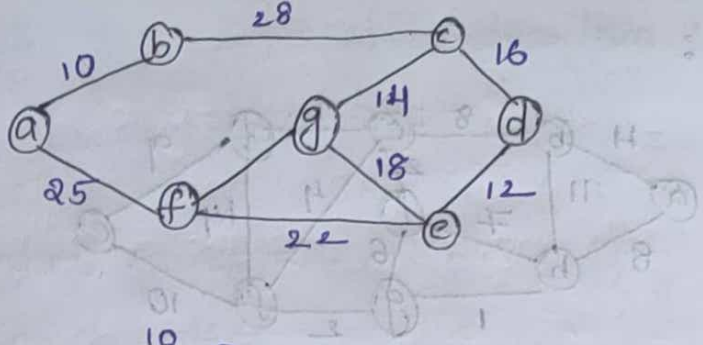
(1) Kruskal's algorithm.

(2) prims Algorithm.

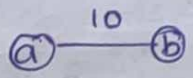
(1) Kruskal's Algorithm:

1. Sort all the edges in non-decreasing order of their weight. [Increasing order]
2. Add the edges one by one to the spanning tree
3. If an adding any edge forms a cycle then discard that edge.
4. Repeat step-2 and step-3 until n vertex 'n-1' edges.

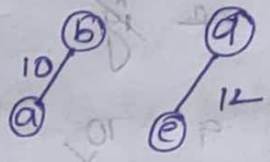
Ex :



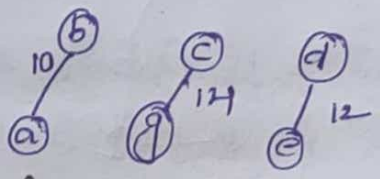
Step-1 :



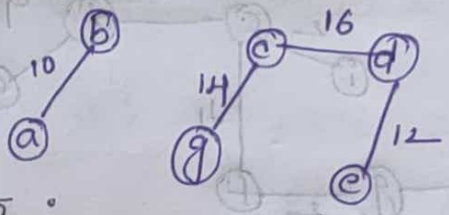
Step-2 :



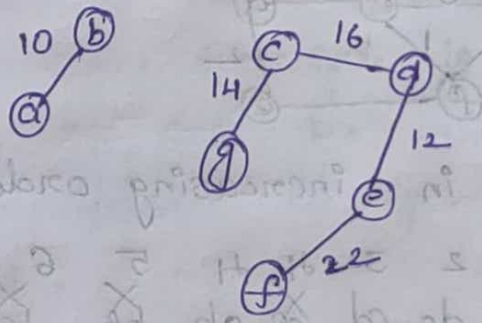
Step-3 :



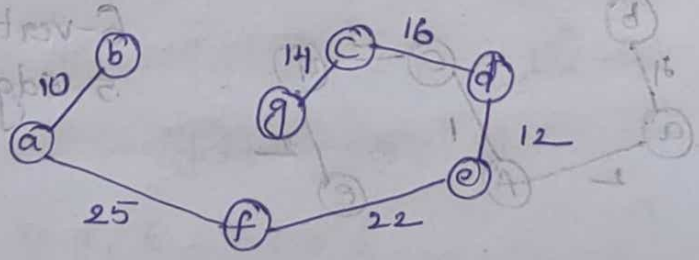
Step-4 :



Step-5 :



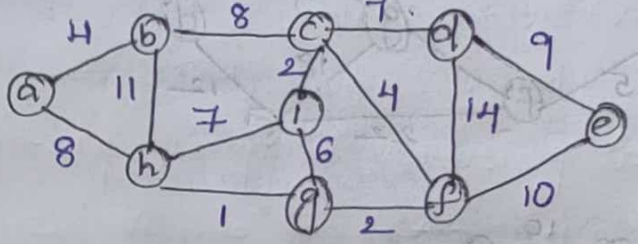
Step-6 :



Min cost = 10 + 14 + 16 + 12 + 22 + 25

= 99

Ex-2 :



1 2 2 4 4 6

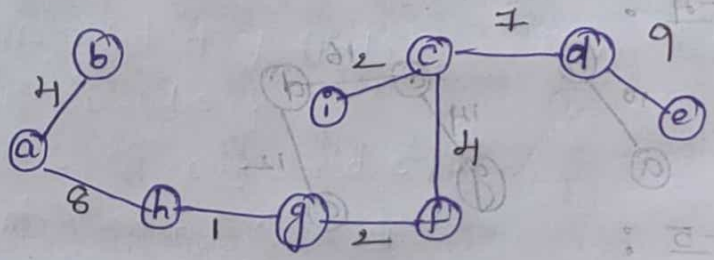
hg fg ci ab cf ig

~~7~~ 7 8 ~~8~~ 9 10

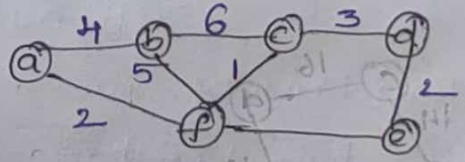
~~hi~~ cd ah ~~bc~~ de ef

11 14

bh df



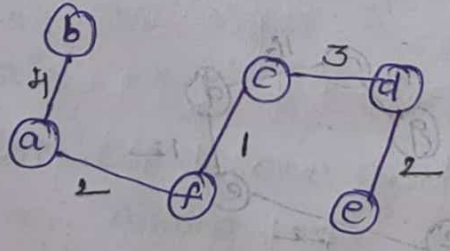
Ex :



Sort the edges in increasing order

Edges : 1 2 2 3 4 4 5 6

cf af de cd ~~ef~~ ab ~~bf~~ ~~bc~~



6-vertices

5-edges

MinCost = 12

Algorithm for Kruskal's Algorithm :

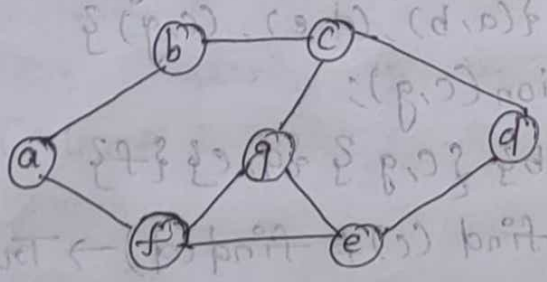
```
Algorithm kruskal's (V, E)
{
   // where V is the set of vertices and E is
   the set of edges.
   A = ∅ ; // A temporary set
           of MST
   for each v∈V do // v is set of verbex in one
   set sort E by weight increasing order
   for each (v1, v2) ∈ E
   {
      if find (v1) != find (v2) then
      A = A ∪ {(v1, v2)}; // A is spanning tree
      union (v1, v2) ;
   }
   return A ;
}
```

Note :- If both the vertices belongs to the same set cycle is formed.

If both the vertices belongs to the different set cycle is not formed.

Tracing :

Ex :



Here V = 7, E = 7-1 = 6

V is vertices

E is Edges

```
Algorithm kruskal's (V, E)
{
   A = ∅ ;
```

for each $v \in V$ do

Make-disjoint set(v);

$\Rightarrow$ {a}, {b}, {c}, {d}, {e}, {f}, {g}

Sort E by weight increasing order.

10 12 14 16 18 22 24 25 28

(a,b) (d,e) (c,g) (c,d) (e,g) (e,f) (f,g) (a,f) (b,c)

for each $(v_1, v_2) \in E$

{

(a,b). if find (a) ≠ find (b) → Truee // find → returns the set numbers

A = ∅ ∪ {v_1, v_2};

A = {(a,b)};

Union(v_1, v_2); // Union is used to Continue.

{a,b}, {c} {d} {e} {f} {g}

(d,e). if find (d) ≠ find (e) → True

A = A ∪ {(d,e)}

A = {(a,b)} ∪ {(d,e)}

A = {(a,b), (d,e)}

Union (d,e);

{a,b} {c}, {d,e} {f} {g}

(c,g). If find (c) ≠ find (g) → True

A = A ∪ {(c,g)}

= {(a,b), (d,e)} ∪ {(c,g)}

= {(a,b), (d,e), (c,g)}

Union (c,g);

{a,b} {c,g} {d,e} {f}

(c,d). If find (c) ≠ find (d) → True

A = A ∪ {(c,d)}

= {(a,b), (d,e), (c,g), (c,d)}

union (c,d);

{a,b} {c,d,e,g} {f}

(e,g). If find (e) ≠ find (g) → false.

(e,f) · If find (e) ≠ find (f) → True

A = A ∪ {e,f}

= {(a,b), (d,e), (c,g), (c,d)} ∪ {(e,f)}

= {(a,b), (d,e), (c,g), (c,d), (e,f)}.

Union (e,f);

⇒ {a,b} {c, d, e, f, g}

(f,g) · If find (f) ≠ find (g) → false

(a f) · If find (a) ≠ find (f) → True

A = A ∪ {(a,f)}

= {(a,b), (d,e) (c,g), (c,d) (e,f) ∪ {(a,f)}

= {(a,b), (d,e), (c,g), (c,d), (e,f) (a,f)}

Union (a,f);

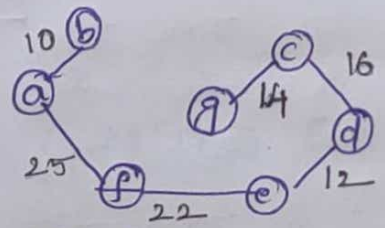
⇒ {a, b, c, d, e, f, g}

(b,c) ⇒ If find (b) ≠ find (c) → false

So the final spanning tree A is.

∴ A = {(a,b) (d,e), (c,g), (c,d), (e,f), (a,f)}

Graph:



Minimum cost = 99.

Time Complexity:

Time Complexity of the Kruskal's algorithm is O (E log e) where E is the no. of edges in the graph.

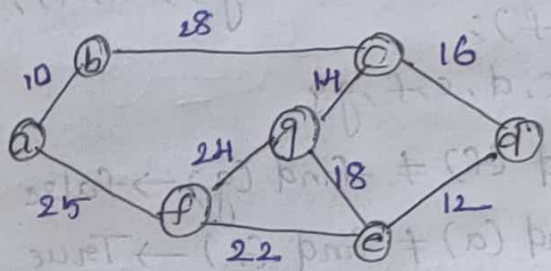
Prim's Algorithm:

prim's Algorithm is a minimum Spanning tree algorithm that takes a graph as input and find the subset of the edges of that graph which.

*form a tree that includes every vertex.

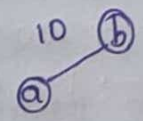
* has the minimum sum of weights among all the trees that can be formed from the graph.

Ex:

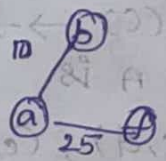


Every time taken adjacent verten of that verten.

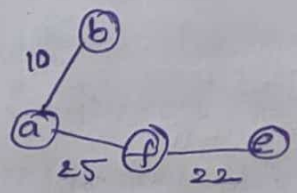
Step-1:



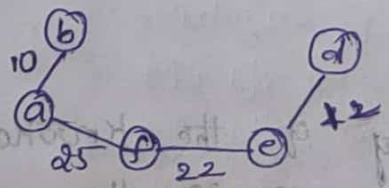
Step-2:



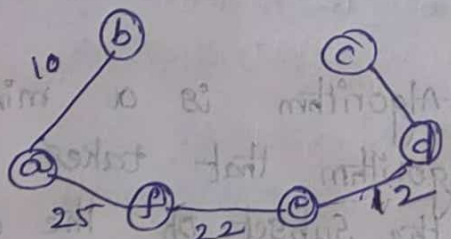
Step-3:



Step-4:



Step-5:



Adjacent

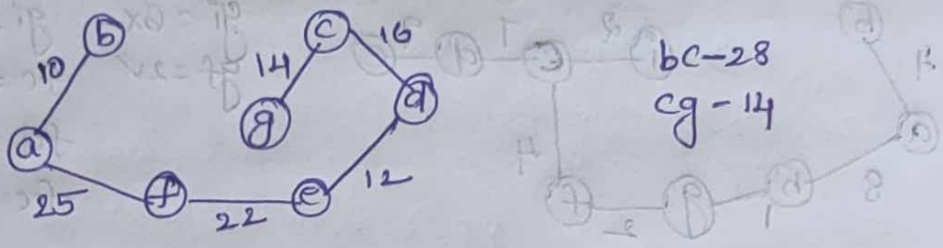
bc - 28

fg - 24

eg - 18

dc - 16

Step-6 :



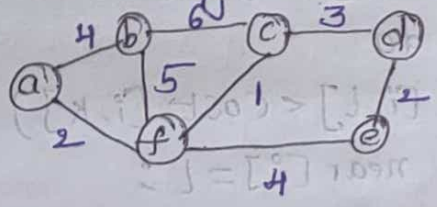
Minimum Cost = 10+25+22+12+16+14

= 99.

It is a spanning tree with 7 vertices and 6 edges

If the cost is repeated then more than one MST are possible - But minimum cost is same but edges are change.

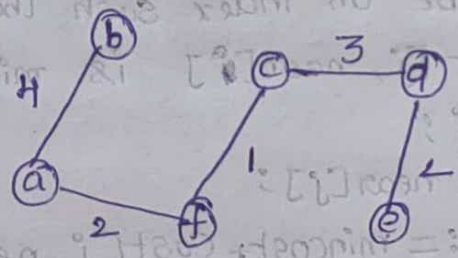
Ex :



V → 6

E → ?

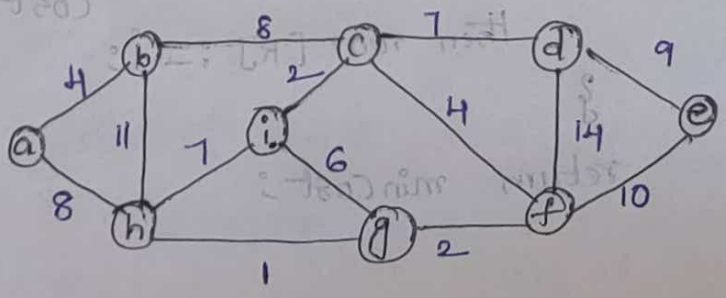
1	2	2	3	4	4	5	6
(c,f)	(a,f)	(e,d)	(e,d)	(a,b)	(e,f)	(b,f)	(b,c)

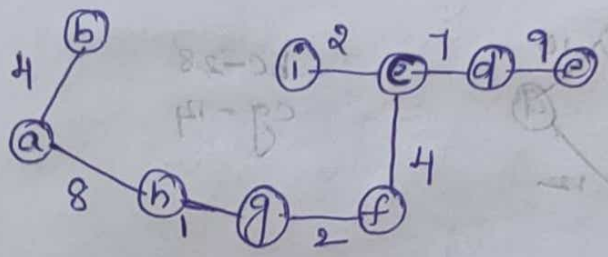


MinCost = 4+2+1+3+2

= 12

Ex :





hi = 7x
ha = 8x
gi = 6x
gf = 2✓

hi = 7x
ha = 8x
gi = 6x
fc = 4
fd = 14 x
fe = 10 x

Min cost = 37

Algorithm for prim's Algorithm :

```
Algorithm prim's (E, cost, n, t)
{
    let [k, l] be an edge of minimum cost in E:
    mincost := cost [k, l];
    t [1, 1] := k ;  t[1, 2] = l;
    for i=1 to n do
        if (cost [i, l] < cost [i, k]) then
                near [i] = l;
        else
                near [i] = k;

    near [k] := near [l] = 0;
    For i=2 to n-1 do.
    {
     let j be an index such that near[j] ≠ 0
     and cost [j, near [j] is minimum;
    t[i, 1] := j;
    t[i, 2] := near [j];
    mincost := mincost + cost[j, near [j]);
    near [j] := 0;
    for k := 1 to n do
        if ((near [k] ≠ 0 and (cost [k, near [k]]>
                                cost [k, j] ))
        then near [k] := j;
    }
    return mincost;
}
```

Time Complexity :

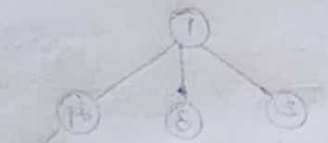
The time complexity for prim's Algorithm is $O(n^2)$

Disjoint sets :

Two sets are said to be disjoint sets if they have no Element in Common

Ex : $S_1 = \{1, 2, 3\}$

$S_2 = \{4, 5\}$



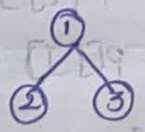
$S_1 \cap S_2 = \emptyset \rightarrow$ called disjoint sets.

Application of Disjoint Sets :

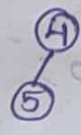
1. Union
2. Find

1. Union :

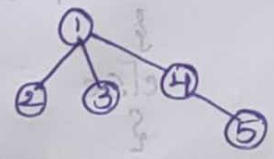
$S_1 = \{1, 2, 3\} \Rightarrow$



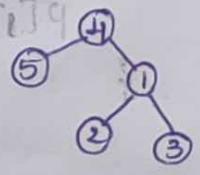
$S_2 = \{4, 5\} \Rightarrow$



$\Rightarrow S_1 \cup S_2 = \{1, 2, 3, 4, 5\} \Rightarrow$



(or)



Algorithm for Simple union :

```
Algorithm union (i,j)
{
    p[i] = j;
}
```

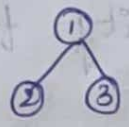
find :

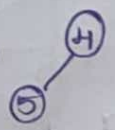
* find operation returns the root of the tree.

Algorithm for Simple find :

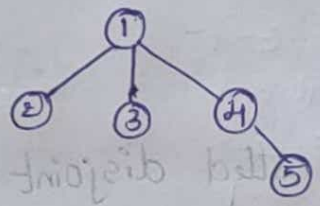
```
Algorithm Simplefind (i)
{
    while (p[i] ≥ 0)
    {
        i = p[i]
    }
    return i;
}
```

Weighted union :-

$S_1 = \{1,2,3\} \Rightarrow$  $\Rightarrow$ weight = 3

$S_2 = \{4,5\} \Rightarrow$  $\Rightarrow$ weight = 2

Weighted union = $\{S_1 \cup S_2\}$



Algorithm for weighted union :

```
Algorithm weighted union (i,j)
{
    temp = p[i] + p[j]
    if [p[i] > p[j]] then
    {
        p[i] = j;
        p[j] = temp;
    }
    else
    {
        p[j] = i;
        p[i] = temp;
    }
}
```

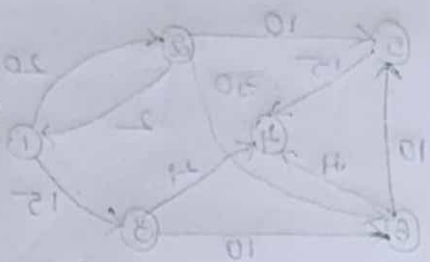
Collaspe find :

All the Elements has to point out the root and return the root node.

Algorithm for collapsing find :

```
Algorithm Collapsing find [i]
{
    N = i
    while (p(r) > 0)
        r = p(r);
    while (i ≠ r) do
```

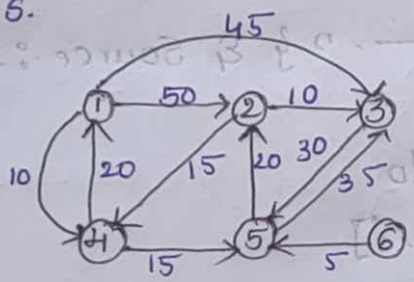
```
{
    s=p[i];
    p[i]=r;
    i=s;
}
return s;
}
```



Dijkstra's Algorithm (or) single source shortest path problem :

Here a weighted directed graph will be given. We have to take one node as the source and we have to find out the shortest path from the source node to the remaining node's.

Ex :



Source = 1

⇒ 1-4

1-4-5

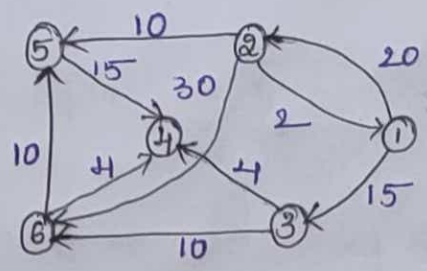
1-4-5-2

1-3

1-6

Source	path	Dest	Cost
1	1-4-5-2	2	45
1	1-3	3	45
1	1-4	4	10
1	1-4-5	5	25
1		6	8

Ex:



⇒ 1-3
1-3-4
1-2
1-2-5
1-3-6

Path	Dest	Cost
1-2	2	20
1-3	3	15
1-3-4	4	19
1-2-5	5	30
1-3-6	6	25

Algorithm for Dijkstra's problem:

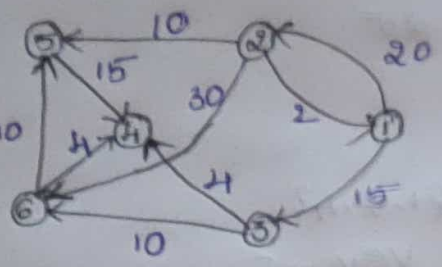
```
Algorithm Dijkstra's (C)
{
    let V = {1, 2, ——, n} & Source : 1;
    S = {1};
    for i=2 to n do
        D[i] = C[1, i]
    for i=1 to n do
    {
        choose a vertex w∈V-S such that
        D[w] is minimum;
        S = S∪{w};
        for each vertex v∈V-S
            D[v] = min (D[v], D[w] + C[w,v])
    }
}
```

Time Complexity:

The time Complexity for the Dijkstra's Algorithm is $O(n^2)$

Tracing :-

Ex:



Sol: Algorithm Dijkstra's ( )

{

let $V = \{1, 2, 3, 4, 5, 6\}$ and source = 1 ;

$S = \{1\}$

~~for~~ for i = 2 to 6 do

$D[2] = C(1,2) \Rightarrow D[2] = 20$

$D[3] = C(1,3) \Rightarrow D[3] = 15$

$D[4] = C(1,4) \Rightarrow D[4] = \infty$

$D[5] = C(1,5) \Rightarrow D[5] = \infty$

$D[6] = C(1,6) \Rightarrow D[6] = \infty$

→ for i = 1 ~~to~~ 6 do → True

2	3	4	5	6
20	15	∞	∞	∞

w = 3 ;

$S = S \cup \{3\} = S\{1,3\}$

→ for each vertex $v \in V - S$

2	4	5	6

$D[2] = \min(D[2], D[3] + (3,2))$

$= \min(20, 15+\infty)$

$= \min(20)$

$D[4] = \min(D[4], D[5] + (3,4))$

$= \min(\infty, 15+4)$

$= \min(19)$

$D[5] = \min(D(5), D[3] + (3,5))$

$= \min(\infty, 15+\infty)$

$= \min(\infty)$

$D[6] = \min(D[6], D[3] + (3,6))$

$= \min(\infty, 15+10)$

$= \min(25)$

→ for i=2 to 6 do → True

	2	4	5	6
w	20	19	∞	25

So w = 4

S = S ∪ {4} → S = {1, 3, 4}

for each vertex v ∈ V − S

2, 5, 6

D[2] = min (D[2], D[4] + C(4, 2))
= min (20, ∞ + ∞)
= min (20)

D[5] = min (D[5], D(4) + C(4, 5))
= min (∞, ∞ + ∞)
= min (∞)

D[6] = min (D(6), D(4) + D(4, 6))
= min (∞, ∞ + ∞)
= min (∞)

→ for i=3 to 6 do → True

	2	5	6
w	20	∞	∞

So w = 2

S = S ∪ {2} ⇒ S = {1, 2, 3, 4}

for each vertex v ∈ V − S

5	6
∞	∞

D[5] = min [D[5], D(2) + D(2, 5)]
= min (∞, 20 + 10)
= min (30)

D[6] = min (D(6), D(2) + D(2, 6))
= min (∞, 20 + 30)
= min (50)

→ for i=4 to 6 do → True

	5	6
w	30	50

So $w=5$

$S = S \cup \{5\} \Rightarrow S = \{1, 2, 3, 4, 5\}$

for each vertex $v \in V-S$

6

$D[6] = \min(D(6), D(5) + D(5,6))$

$= \min(\infty, \infty + \infty)$

$= \min(\infty)$

→ for $i=5$ to 6 do → True

6

∞

So $w = 6$

$S = S \cup \{6\} \Rightarrow S = \{1, 2, 3, 4, 5, 6\}$

for each vertex $v \in V-S$ → false

→ for $i=6$ to 6 do → true

but no vertices.

→ for $i=7$ to 6 do → false

So,

$D[2] = (20)$ → direct path

$D[3] = (15)$ → direct path

$D[4] = 19$ → $3 \to 4$ (via 3)

$D[5] = \min(\underset{3\to5}{\infty}, \underset{4\to5}{\infty}, \underset{2\to5}{30})$

$= \min(\underset{2\to5}{30})$

$= 30$ (via 2)

$D[6] = \min(\underset{3\to6}{25}, \underset{4\to6}{\infty}, \underset{2\to6}{50}, \underset{5\to6}{\infty})$

$= \min(\underset{3\to6}{25})$

$= 25$ (via 3)

Source	path	Dest	Cost
1	1→2	2	20
1	1→3	3	15
1	1→3→4	4	19
1	1→2→5	5	30
1	1→3→6	6	25