Unit-III

Greedy Method

→ It gives the optimal solution for the given problem.

→ Feasibility study is the subset of inputs with ~~satisfy~~ satisfying the constraints of the problem.

→ Optimal solution is a feasible solution which either maximises or minimizes the objective function.

Control Abstraction:- (subset paradigm) → It doesn't require sorting of i/p elements.

```
Algorithm Greedy (a, n)
// a[1:n] are the inputs
{
    sol = ϕ;
    for i=1 to n do
    {
        x = select (a);
        if feasible (sol, x) then
            sol = Union (sol, x);
    }
    return sol;
}
```

Ordering paradigm :- It requires the sorting of given i/p elements either in ascending order or descending order.

→ Greedy has the Time complexity of O(n).

Applications of Greedy:-

1. Knapsack problem.
2. Job sequencing with deadlines.
3. Single source shortest path.
4. Minimum cost spanning Tree.

1. Knapsack Problem :-

n , $W = \{w_1, w_2, w_3 \ldots w_n\}$

m , $P = \{P_1, P_2, P_3 \ldots P_n\}$

→ The objective is to fill up the bag with all the weights such that getting the maximum profit.

$X = \{x_1, x_2 \ldots x_n\}$

objective function = Maximum.

Knapsack problem

Objective fn - Maximize $\sum_{i=1}^{n} P_i x_i$ —(1)

subject to.

* $\sum_{i=1}^{n} w_i x_i \leq m$ —(2)

* $1 \leq i \leq n$, $0 \leq x_i \leq 1$ —(3)

Eg: 1 $n = 3$, $m = 20$.

$(P_1, P_2, P_3) = (25, 24, 15)$

$(w_1, w_2, w_3) = (18, 15, 10)$

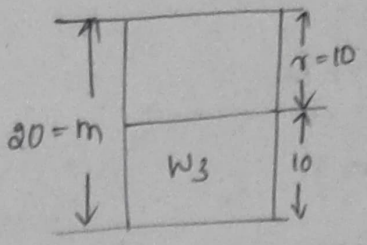
Sol: Method-1 → It will find the feasible sol by arranging increasing order (Ascending order)

After Rearranging the weights in Ascending order, then

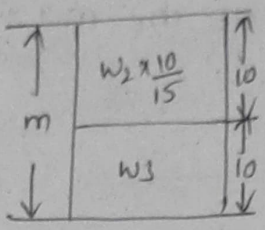
$(w_3, w_2, w_1) = (10, 15, 18)$

After rearranging the profits in Ascending order, then

$(P_3, P_2, P_1) = (15, 24, 25)$



$x_3 = 1.$



$x_2 = \frac{10}{15} = \frac{2}{3}.$

$x_3 = 1.$

$x_1 = 0.$

$X = \{x_1, x_2, x_3\}$

$X = \{0, 2/3, 1\}.$

$$\sum_{i=1}^{3} w_i x_i = w_1 x_1 + w_2 x_2 + w_3 x_3.$$

$$= 18 \times 0 + 15 \times \frac{2}{3} + 10 \times 1.$$

$$= 0 + 10 + 10$$

$$= 20. \approx m.$$

$$\sum_{i=1}^{n} P_i x_i = P_1 x_1 + P_2 x_2 + P_3 x_3$$

$$= 25 \times 0 + 24 \times \frac{2}{3} + 15 \times 1.$$

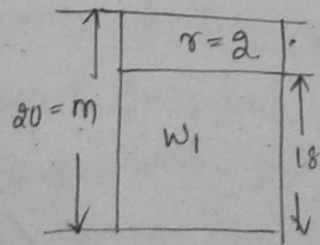
$$= 0 + 16 + 15$$

$$= 31$$

Method-ii - It rearranges the objects in the descending order the profits

$(P_1, P_2, P_3) = (25, 24, 15)$

$(w_1, w_2, w_3) = (18, 15, 10)$



$x_2 = \frac{2}{15}$

$x_1 = 1$

$x_3 = 0.$

$x = \{x_1, x_2, x_3\}$

$x = \{1, 2/15, 0\}$.

$$\sum_{i=1}^{3} p_i x_i = p_1 x_1 + p_2 x_2 + p_3 x_3.$$

$$= 25 \times 1 + 24 \times \frac{2}{15} + 15 \times 0.$$

$$= 25 + 3.2$$

$$= 28.2.$$

$$\sum_{i=1}^{3} w_i x_i = w_1 x_1 + w_2 x_2 + w_3 x_3$$

$$= 18 \times 1 + 15 \times \frac{2}{15} + 10 \times 0.$$

$$= 18 + 2$$

$$= 20. \approx m.$$

Method-iii → Method-i profit = 31

Method-ii profit = 28.2

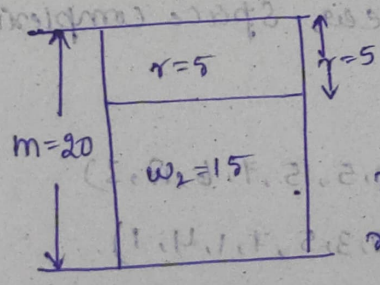
Method-iii profit = 31.5

$\frac{p_1}{w_1} = 25/18 = 1.38$ → sorting of all the p/w values

$\frac{p_2}{w_2} = 24/15 = 1.6$

$\frac{p_3}{w_3} = 15/10 = 1.5$

$$\frac{p_2}{w_2} \geq \frac{p_3}{w_3} \geq \frac{p_1}{w_1}$$



$x_2 = 1$ $x = \{x_1, x_2, x_3\}$

$x_3 = 5/10 = 1/2$. $x = \{0, 1, 1/2\}$

$x_1 = 0$.

$$\sum_{i=1}^{3} p_i x_i = p_1 x_1 + p_2 x_2 + p_3 x_3 \quad \sum_{i=1}^{3} w_i x_i = 20.$$

7.5 $\approx M$

$$= 25 \times 0 + 24 \times 1 + 15 \times \frac{1}{2}$$

$$= 24 + 7.5$$

$$= 31.5$$

* Algorithm for Knapsack:

Algorithm GreedyKnapsack (m, n)

// m is capacity of knapsack, n is n.o of objects

// w[1:N], p[1:N] are the global Arrays.

// If $\frac{P_i}{w_i} > \frac{P_{i+1}}{W_{i+1}}$ then greedy knapsack provides the optimal solution.

```
{
  for i=1 to n do
    x[i] = 0.0;
  r= m;
  for i =1 to n do.
    if (w[i] > r)
      break;.
    else
     {x[i] = 1.0;
      r=r-w[i];
     }
    if i ≤ n then
      x[i] = r/w[i];
}
```

→ Greedy knapsack algorithm takes the linear time complexity of O(n)

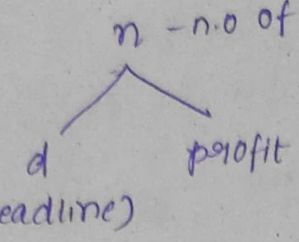
→ Greedy knapsack linear space complexity O(n)

Eg: n=7 m=15.

$(P_1 P_2 P_3 P_4 P_5 P_6 P_7) = (10, 5, 15, 7, 6, 18, 3)$

$(w_1, w_2, w_3, w_4, w_5, w_6, w_7) = (2, 3, 5, 7, 1, 4, 1)$

2. Job sequencing with Deadlines:-



→ Execution of the job must be completed within its deadline to get maximum profit.

Eg: $n=4$

$(P_1, P_2, P_3, P_4) = (100, 10, 15, 27)$

$(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

S.NO	Feasible Solution	Processing sequence	Profit value
1.	{1}	{1}	100.
2.	{2}	{2}	10.
3.	{3}	{3}	15
4	{4}	{4}	27
5.	{1,2}	{2,1}	110.
6.	{1,3}	{1,3} or {3,1}	115
7.	{1,4}	{4,1}	127
8.	{2,3}	{2,3}	25
9.	{3,4}	{4,3}	42

Optimal solution = I = {1,4}.

Total profit earned = $\sum P_i = P_1 + P_4$

$= 100 + 27$

$= 127$

→ Arranging the jobs in decreasing order of profit of the jobs.

$P_i \geq P_{i+1}$

1. $n = 5$

$(p_1, p_2, p_3, p_4, p_5) = (20, 15, 10, 5, 1)$

$(d_1, d_2, d_3, d_4, d_5) = (2, 2, 1, 3, 3)$

J	Assigned Slots	Job Considered	Action	Profit
ϕ	–	1	fits	–
{1}	[1, 2]	2	fits	20
{1, 2}	[0, 1] [1, 2]	3	can't fit	35
{1, 2}	[0, 1] [1, 2]	4	fits	35
{1, 2, 4}	[0, 1] [1, 2] [2, 3]	5	can't fits	40.

Optimal solution is $J = \{1, 2, 4\}$

Total profit earned = 40.

2. $n = 4$.

$(P_1, P_2, P_3, P_4) = (100, 10, 15, 27)$

$(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

Arrange in descending order of profit.

$(P_1, P_4, P_3, P_2) = (100, 27, 15, 10)$

$(d_1, d_4, d_3, d_2) = (2, 1, 2, 1)$

J	Assigned Slots	Job considered	Action	Profit
ϕ	–	1	fits	–
{1}	[1, 2]	4	fit	100.
{1, 4}	[0, 1] [1, 2]	3	can't fit	127
{1, 4}	[0, 1] [1, 2]	2	Can't fit	127

Optimal solution is $J = \{1, 4\}$

Total profit earned = 127.

Algorithm:-

```
Algorithm JS(d, j, n)
// d[i] ≥ 1, 1 ≤ i ≤ n are the deadlines, n ≥ 1. The objs
// are ordered such that P[1] ≥ P[2] ≥ ... P[n]. J[i]
// is the ith job in the optimal solution 1 ≤ i ≤ k
// Also at termination d[J[i]] ≤ d[J[i+1]], 1 ≤ i < k.
{
    d[0]:=J[0]:=0; //Initialize
    J[1]:=1; //include job 1
    k:=1;
    for i:=2 to n do
    {
    //consider jobs in nonincreasing order of P[i], find
    //position for i and check possibility of insertion.
    r:=k;
    While ((d[J[r]]>d[i]) and (d[J[r]]≠r)) do r:=r-1;
    if ((d[J[r]] ≤ d[i]) and (d[i] > r)) then
    {
    //insert i into J[]
    for q:=k to (r+1) step -1 do J[q+1]:=J[q];
    J[r+1]:=i; k:=k+1;
    }
    }
    return k;
}
```

Algorithm GreedyJob(d, j, n)

// J is a set of jobs that can be completed by their dead times

{

J := {1};

for i := 2 to n do

{ if (all jobs in J∪{i} can be completed by their attributes) then J := J∪{i};

}

}

→ Time complexity of Jobsequencing is O(nK)

if k=n

O(n.n) = O(n²)

* Minimum cost Spanning Trees:

G = (V, E)

graph — vertices — Edges

→ works on undirected weighted graph (input for min cost spanning tree)

→ undirected/weighted are associated with edges

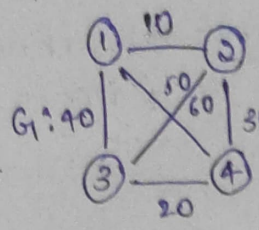
→ acyclic graph is known as tree

E' → subset of E T = (V, E')

|V| = n, |E| = e

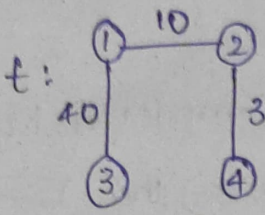
Minimum of no. of edges = E' = n − 1 < e

Eg:

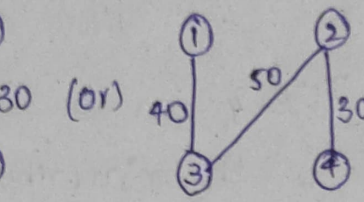


G: undirected unweighted $e=6$, $n=4$, $|e'|=3$

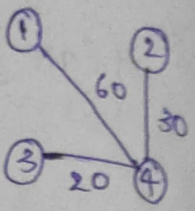
t:



(or)



(or)



→ tree formed by the graph is known spanning forest consists of spanning trees.

<u>total cost</u>: sum of weights of edges.

t_1 cost = 80

t_2 cost = 120

t_3 cost = 110

Minimum cost spanning tree is $t_1 = 80$

↓

2 conditions tested before adding edge of tree

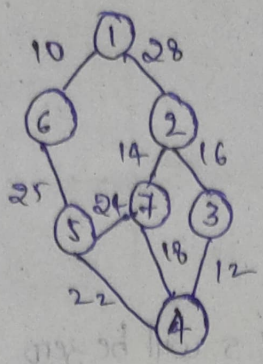
1. edge do not form cycle.
2. any edge with minimum cost must be inserted (or) minimal ↑es in cost

<u>Total cost</u>: minimize $e\sum_{i=1}^{n-1}$ cost of edge i

<u>Two types of algorithm to determine minimum cost spanning tree:</u>

1) prim's Alg. (finding nearest vertex means less weight)
2) kruskal's Alg (inserts an edge which has lowest cost)

Eg: Apply prim's Algorithm:



Adjacency Matrix (or) Cost Matrix :-

	1	2	3	4	5	6	7
1	∞	28	∞	∞	∞	10	∞
2	28	∞	16	∞	∞	∞	14
3	∞	16	∞	12	∞	∞	∞
4	∞	∞	12	∞	22	∞	18
5	∞	∞	∞	22	∞	25	24
6	10	∞	∞	∞	25	∞	∞
7	∞	14	∞	18	24	∞	∞

7x7

∞ → NO edge

cost(1,2) = 28

cost(6,5) = 25

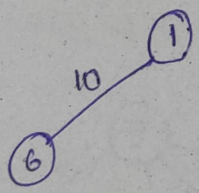
cost(i, j)

↓ ↓

row column.

Prism's Algorithm :-

Step 1 : Identify the lowest cost edge



Step 2 : Prepare near Array [n x 1]

	near	cost
1	0	—
2	1	28
3	1	∞
4	1	∞
5	6	25
6	0	—
7	1	∞

vertex no.

1 & 6 vertices are already inserted so no need to write near array.

(2,1) = 28 (2,6) = ∞

(3,1) = ∞ (3,6) = ∞

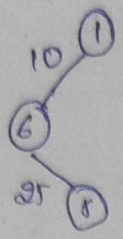
(4,1) = ∞ (4,1) = ∞

(5,1) = ∞ (5,6) = 25

(7,1) = ∞ (7,6) = ∞

Choose the lowest cost = 25

(5, 6) = 25



Step 3: After inserting vertex then near of 5 will be zero

	NEAR	
1	0	-
2	1	28
3		∞
4		22
5	0	-
6	0	-
7		24

(2,1) = 28 (3,1) = ∞

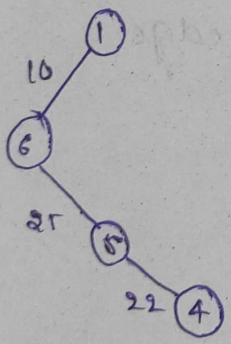
(2,5) = ∞ (3,5) = ∞

(4,1) = ∞ (7,1) = ∞

(4,5) = 22 (7,5) = 24

Minimum cost = 22

(4,5) = 22



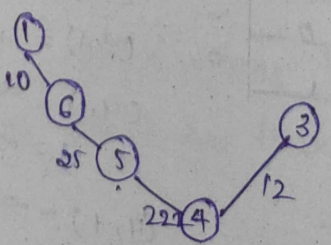
Step 4:

	NEAR	cost
1	0	-
2	1	28
3	4	12
4	0	-
5	0	-
6	0	-
7	4	18

Minimum cost = 12

(3,4) = 12

So insert (3)



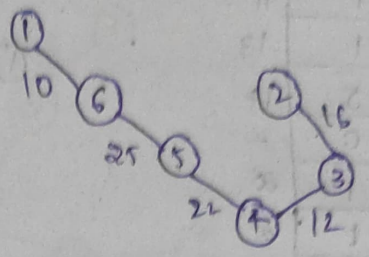
Step5:

	NEAR	COST
1	0	–
2	3	16
3	0	–
4	0	–
5	0	–
6	0	–
7	4	98

Minimum cost = 16

(2,3) = 16

Insert (2)



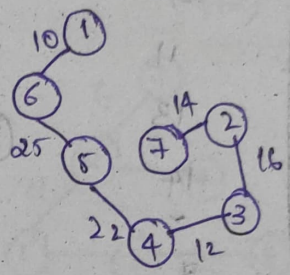
Step6:

	NEAR	COST
1	0	–
2	0	–
3	0	–
4	0	–
5	0	–
6	0	–
7		14

Minimum cost = 14

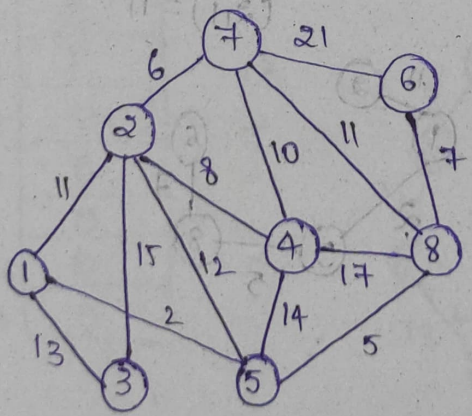
(7,2) = 14

Insert (7)

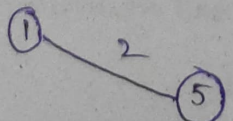


Total cost of spanning tree i.e Minimum cost = 99.

Example: 2



Step1: Minimum cost:

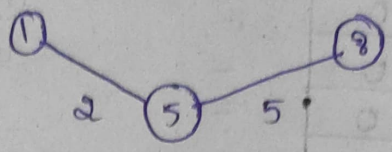


Step 2: NEAR cost

1	0	–
2	1	11
3	1	13
4	5	14
5	0	–
6	1	∞
7	1	∞
8	5	5

Minimum cost = 5

(8,5) = 5

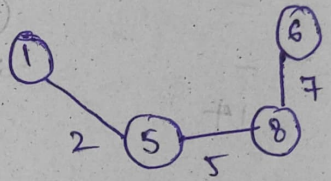


Step 3: NEAR cost

1	0	–
2	1	11
3	1	13
4	5	14
5	0	–
6	8	7
7	1	∞
8	0	–

Minimum cost = 7

(6,8) = 7

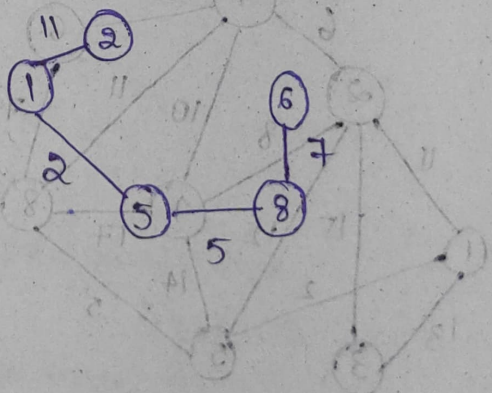


Step 4: NEAR cost

1	0	–
2	1	11
3	1	13
4	5	14
5	0	–
6	0	–
7	6	21
8	0	–

Minimum cost = 11

(2,11) = 11

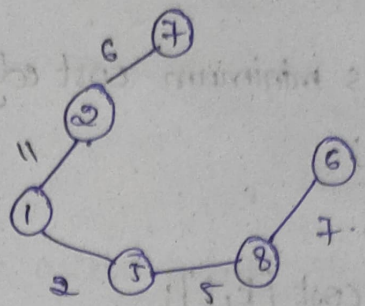


Step5: NEAR cost

	NEAR	cost
1	0	–
2	0	–
3		13
4		8
5	0	–
6	0	–
7		6
8	0	–

Minimum cost = 6

(7,2) = 6

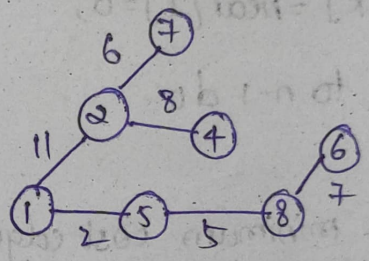


Step6: NEAR cost

	NEAR	cost
1	0	–
2	0	–
3	1	13
4	2	8
5	0	–
6	0	–
7	0	–
8	0	–

Minimum cost = 8

(4,2) = 8

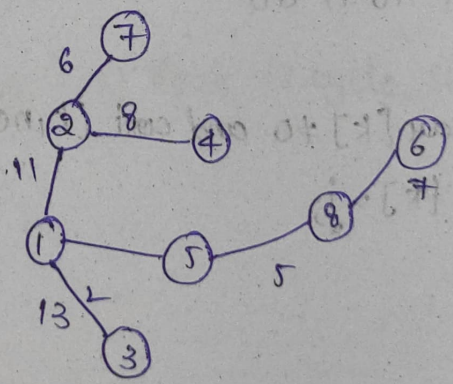


Step7: NEAR

	NEAR	cost
1	0	–
2	0	–
3	1	13
4	0	–
5	0	–
6	0	–
7	0	–
8	0	–

Minimum cost = 13

(3,1) = 13



Minimum cost spanning tree.

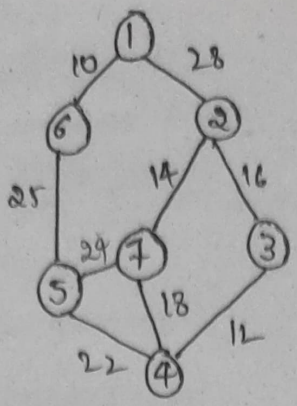
Minimum cost = 52

```
Algorithm Prims(n, E, cost, t)
// n is vertices (no); E is edge set, cost is matrix, t is spanning
{
Let (k, l) is minimum cost edge in the given graph
t[1,1] = k;
t[1,2] = l;
min cost = cost[k,l];
for i = 1 to n do
  if cost[i,l] > cost[i,k] then near[i] = k;
else near[i] = l;
  near[k] = near[l] = 0;
for i = 2 to n-1 do
{
select a minimum cost edge
(cost[j, near[j]) is minimum and near[j] ≠ 0)
t[i,1] = j;
t[i,2] = near[j];
mincost; mincost + cost[j, near[j]];
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 of prim's Algorithm is $O(n^2)$

where n is no. of vertices.

Ex:1

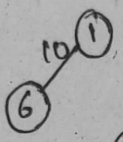


Kruskal's Algorithm

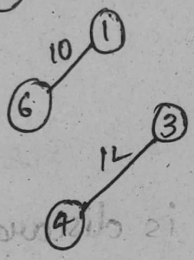
Step1: Sort all the edges cost in ascending order

10 12 14 16 18 22 25 28

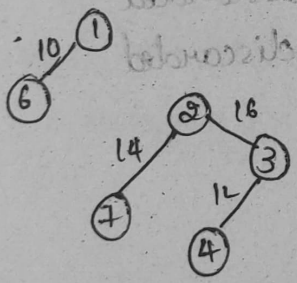
step2: Insert lost cost edge vertices with no cycle.



step3:

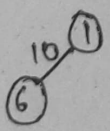


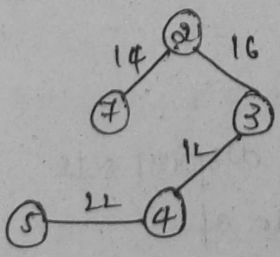
step4:



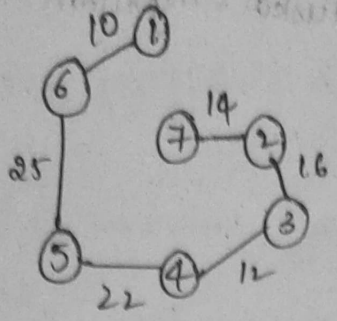
→ when we want to insert (18) there is cycle so not inserted.

→





→ 24 forms cycle so discarded.



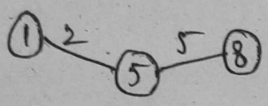
→28 forms cycle so discarded

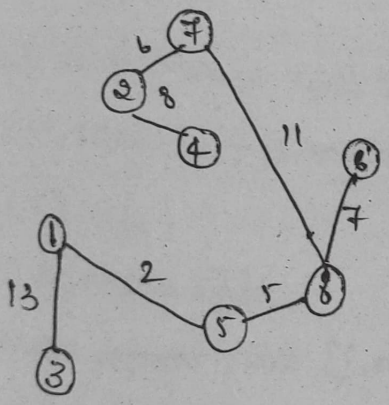
Minimum cost = 99.

Ex: 2

cost : 2, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 17, 21

Graph:





10 is discarded
12 is discarded
14 is discarded
15
17
21

minimum cost : 52

Kruskal's Algorithm:-

① min heap

{1, 2, 3} {4, 5}

parent = -1.

union - used to combine two disjoint sets.

find(i) — provides the root node of i.

heapify — to construct min heap.

```
Algorithm Kruskal(n, E, cost, t)
{
  create min heap using heapify → |E| log|E|  Time complexity.
   for i:=1 to n do
   parent[i] = -1;
    i=0; //0 edges have been inserted.
    mincost:=0.
  while(i<n-1 and heap is not empty)
  {
   Delete minimum cost edge[u,v] from minheap
   and reheapify using 'Adjust' procedure.
                          ↓
                    |E| log|E| Time
                    complexity.
    j = find(u);
    k = find(v);
   if(j != k)
   {
     i = i + 1;
    t[i,1] = u;
    t[i,2] = v;
  mincost=mincost + cost[u,v];
   union(j,k);
  }
}
if (i<n-1) then
  {
   write("No spanning tree formed");
  }
 else
   return mincost;
}
```

Single source shortest paths :

Time complexity = $O(n^2)$
Space complexity = $n^2 + n + 3$
$= O(n^2)$

I/p :- Directed, weighted graph

$cost(i,j) = +ve \quad (i,j) \in E$

$= \infty \quad (i,j) \notin E$

→ Among 'n' no of vertices one vertex should be made as source vertex

Data Structures

1. S – S is an array of size 'n'

   S[i] = True/False

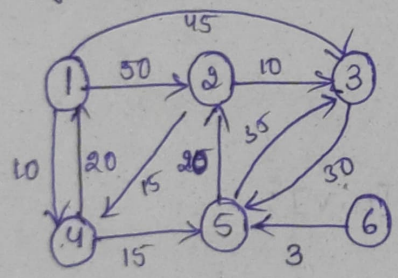
   $1 \le i \le n$

2. Dist – It is one dimensional array represents

   Dist[i] → shortest distance from source to index 'i'

3. u → adjacent vertex.

Eg Graph :-



Let us consider '1' as source vertex.

consider the shortest path.

	Path	Cost
1)	1-4	10
2)	1-4-5	25
3)	1-4-5-2	45
4)	1-3	45

↓ Increasing order of their path lengths.

Eg:2 Take the same graph and consider '5' as source vertex. consider the shortest path.

	Path	cost
1.	5-2	20
2.	5-2-3	30
3.	5-2-4	35
4.	5-2-4-1	55

1. s[5] = True s[2] = True s[3] = True
2. dist[5] = 0 dist[2] = 20 dist[3] = 30

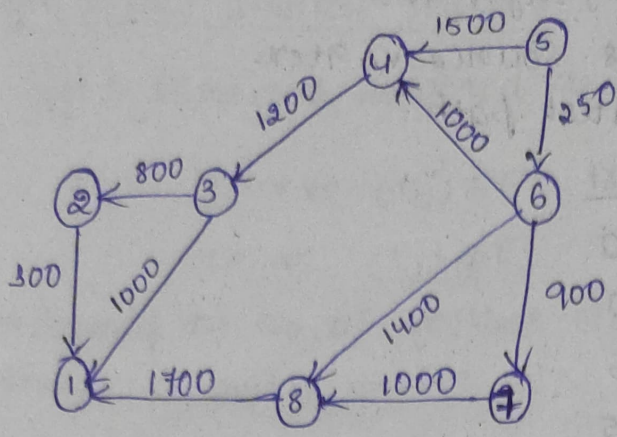
s[4] = True s[5] = True
dist[4] = 35 dist[5] = 55

```
Algorithm ShortestPaths(v, cost, dist, n)
{
 for i:=1 to n do
 {
  s[i] = false; dist[i] = cost[v,i];
 }
 s[v] = true; dist[v] := 0.0;
 for num:=2 to n-1 do
 {
  for(each w, adjacent to u with s[w]=false) do
  if(dist[w] > dist[u] + cost[u,w])) then
     dist[w] = dist[u] + cost[u,w];
 }
}
```

→ choose u from set of vertices that are in s and having minimum path length

s[u] = True;

u adjacent node is w.



Consider '5' as source vertex.

step no	S	Vertex selected	Distance of vertex from source vertex							
			1	2	3	4	5	6	7	8
—	—	—	∞	∞	∞	1500	0	250	∞	∞
1	5	6	∞	∞	∞	1250	0	250	1150	1650
2	5,6	7	∞	∞	∞	1250	0	250	1150	1650
3	5,6,7	4	∞	∞	2450	1250	0	250	1150	1650
4	5,6,7,4	8	3350	∞	2450	1250	0	250	1150	1650
5	5,6,7,4,8	3	3350	3250	2450	1250	0	250	1150	1650
6	5,6,7,4,8,3	2	3350	3250	2450	1250	0	250	1150	1650
7	5,6,7,4,8,3,2	1	3350	3250	2450	1250	0	250	1150	1650
8	5,6,7,4,8,3,2,1	-	-	-	-	-	-	-	-	-

* Source vertex is '7'

step no	S	vertex selected	Distance							
			1	2	3	4	5	6	7	8
—	—	—	∞	∞	∞	∞	∞	∞	0	1000
1	7	8	1700	∞	∞	∞	∞	∞	0	1000
2	7,8	1	-	-	-	-	-	-	-	-
3	7,8,1	-	-	-	-	-	-	-	-	-