ASSIGNMENT 2

Q1] Solve the following knapsack problem (Also write Algorithm)

$n = 5$, $m = 100$, $(P_1, P_2, P_3, P_4, P_5) = (20, 30, 66, 40, 60)$
$(w_1, w_2, w_3, w_4, w_5) = (10, 20, 30, 40, 50)$

[Explain the mathematical formulation of knapsack problem]

* Feasible solution

$(x_1, x_2, x_3, x_4, x_5)$	$\Sigma w_i x_i$	$\Sigma p_i x_i$	Strategy
$(0, 0, 1, 1/2, 1)$	100	146	Max profit
$(1, 1, 1, 1, 0)$	100	156	Least weight
$(1, 1, 1, 0, 4/5)$	100	164	Decreasing order of $p_i w_i$

* Max profit

start according to profits $\Rightarrow P_3 > P_5 > P_4 > P_2 > P_1$
$w_3 > w_5 > w_4 > w_2 > w_1$

- $w_3 < 100$ added to bag.
- $(m - w_3) = 70$ is still capacity remaing
- $w_5 < m$ added to bag
- $(70 - 50) = 20$ capacity still present in bag
- w_4 is not completely added only $1/2$ is added and capacity of bag is full.

* <u>Least weight</u>

Start according to increasing order of weights $\Rightarrow w_1 < w_2 < w_3 < w_4$

- w_1 added to bag completly, capacity remaing = 90
- w_2 added to bag completly, capacity remaing = 70
- w_3 added to bag completly, capacity remaing = 40
- w_4 added to bag completly, capacity remaing = 0
- w_5 no place remaing/available in bag, It is full

* <u>Decreasing order of P_i/w_i</u>

$P_i w_i = (2, 1.5, 2.2, 1, 1.2)$

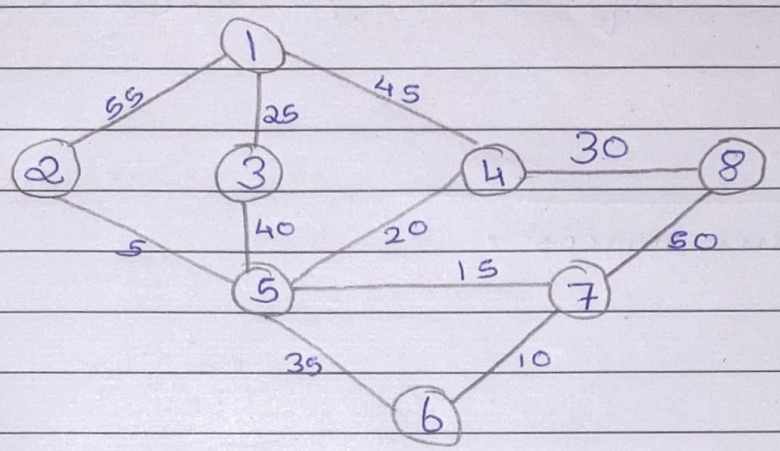
Decreasing order $w_3 > w_1 > w_2 > w_5 > w_4$

- w_3 added completly, capacity remaing = 70
- w_1 added completly, capacity remaing = 60
- w_2 added completly, capacity remaing = 40
- w_5 added 4/5, capacity remaing = 0
- w_4 no place available in bag, It is full.

$\therefore$ $\boxed{\text{Total profit} = 164}$

Q2] Find the minimum cost spanning tree for the following graph using prim's algo & Kruskal's Algo

[Also write Prim's and Kruskal's Algo]



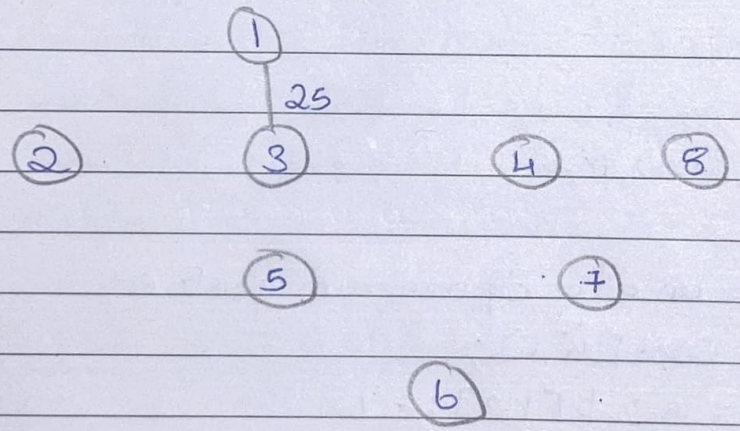
* Prim's Algo

```
Algorithm Prim (E, cost, n, t)
{
    let (K, L) be an edge of minimum cost E;
    mincost := cost[K, L];
    t[1,1] := K; t[1,2] := L
    for i := 1 to n do // initialize near
    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
    {
        // find n-2 additional edges for t
        let J be index s.t near[J] ≠ 0 and
        cost[J, near[J]] is minimum;
```

```
        t[i,1]:=j;
        t[i,2]:=near[j];
        mincost:=mincost+cost[[i],near[j]];
        near[j]:=0
        for k:=1 to n do //update near[]
        if((near[k]≠0)and(cost(k,near[k]]>cost
                                        [k,j]))then
                                        near[k]:=j
    }
    return mincost;
}
```

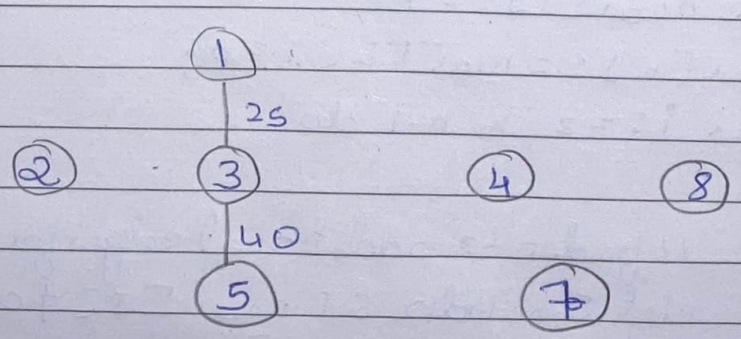
→ Graph

(1)



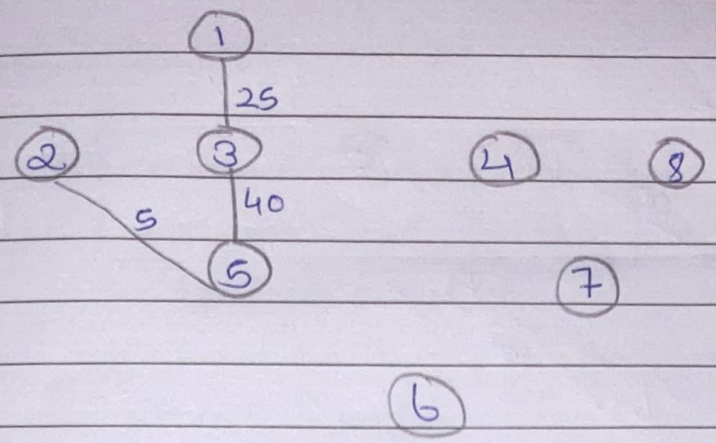
mincost = 25

(2)



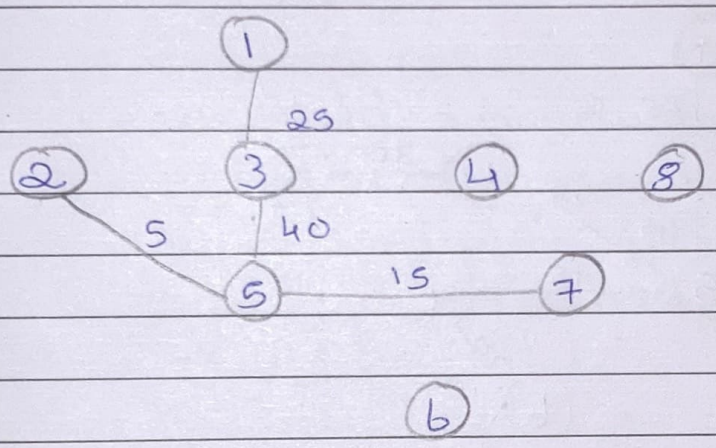
mincost = 25+40 = 65

(3)



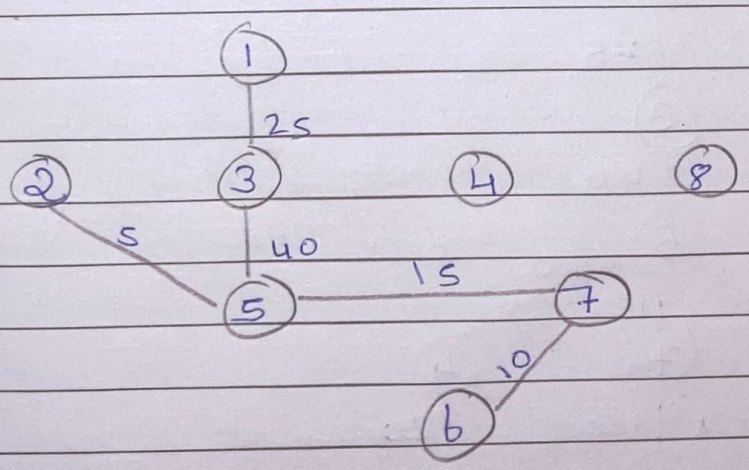
mincost = 25+40+5 = 70

(4)



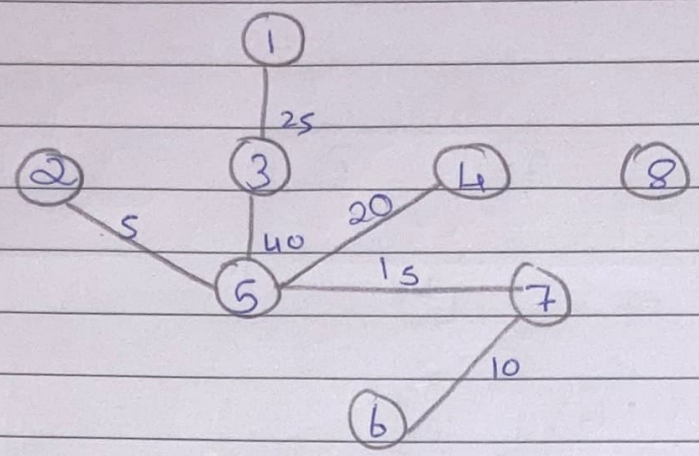
mincost = 70 + 15 = 85

(5)



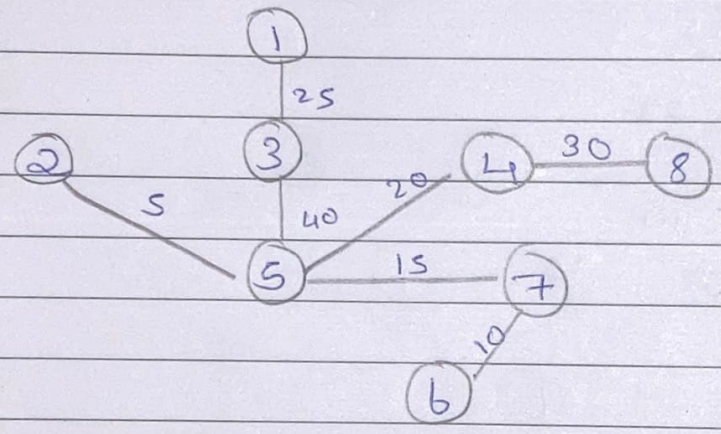
mincost = 70 + 10 + 15 = 95

(6)

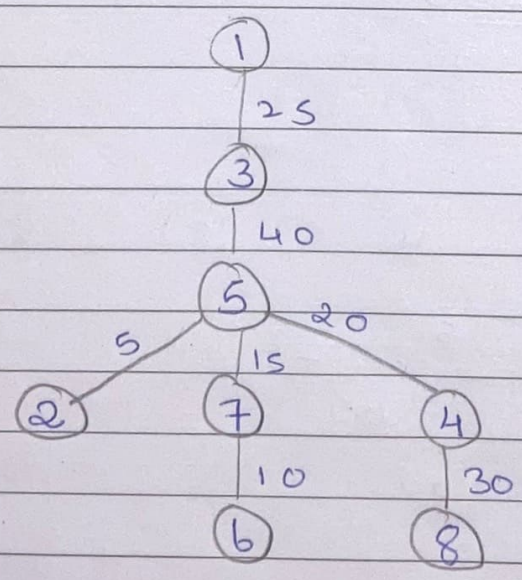


mincost = 95 + 20 = 115

(7)



mincost = 95 + 20 + 30 = 145



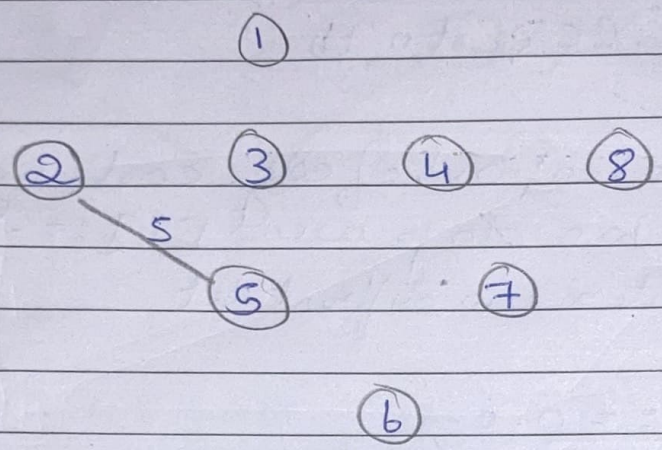
Kruskal's Algo

```
Algorithm Kruskal (E, cost, n, t)
{
    Construct a heap out of edge cost using Heapify.
    for i := 1 to n do parent [i] := -1;
    //Each vertex is in diffrent set
    i := 0;
    mincost := 0.0;
    while ((i < n-1) and (heap not empty)) do
    {
        Delete a minimum cost edge (u,v) from heap
        & reheapify using adjust;
        j := Find(u) ; k := find(v);
        if (j ≠ k) then
        {
            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");
    else return mincost;
}
```

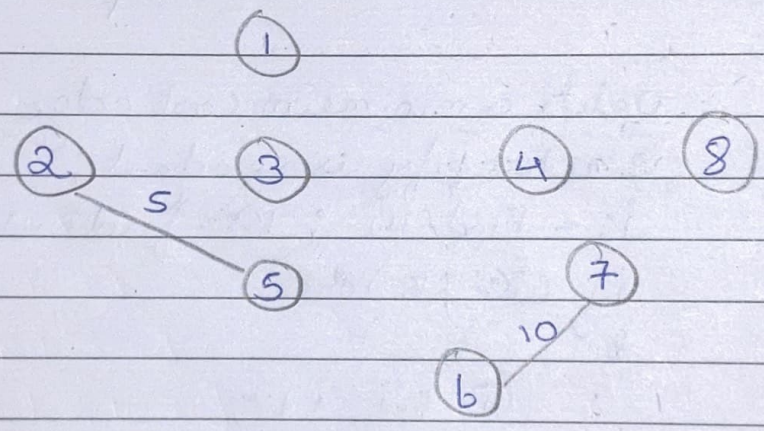
Page No :
Date : / /

=> Graph

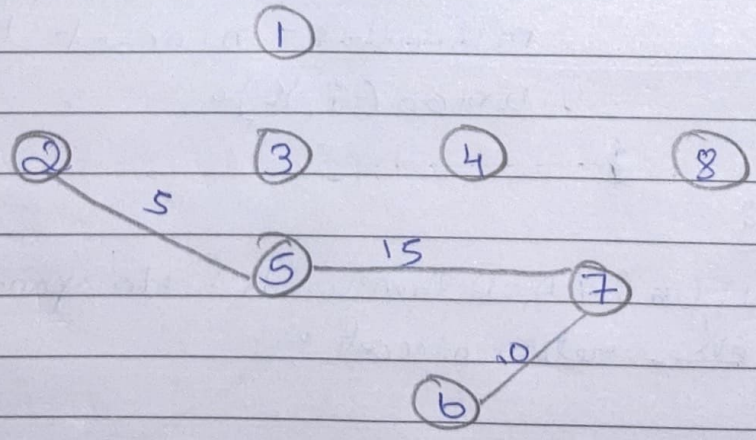
(1)



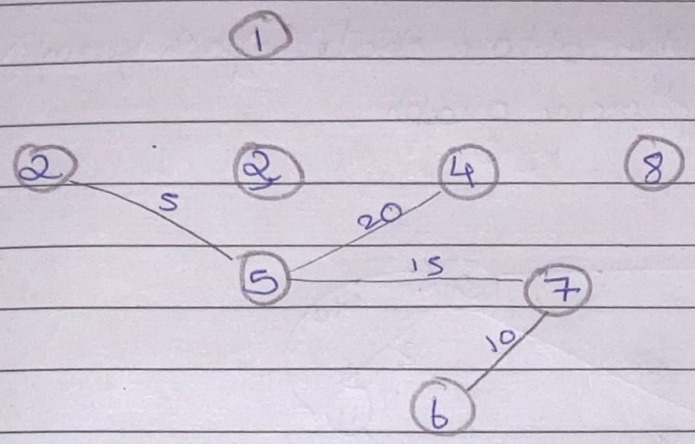
(2)



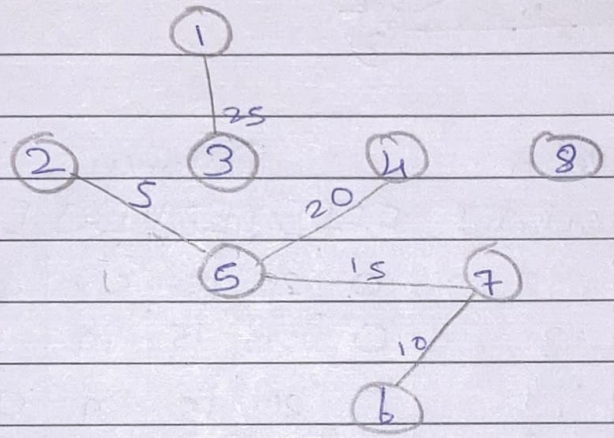
(3)



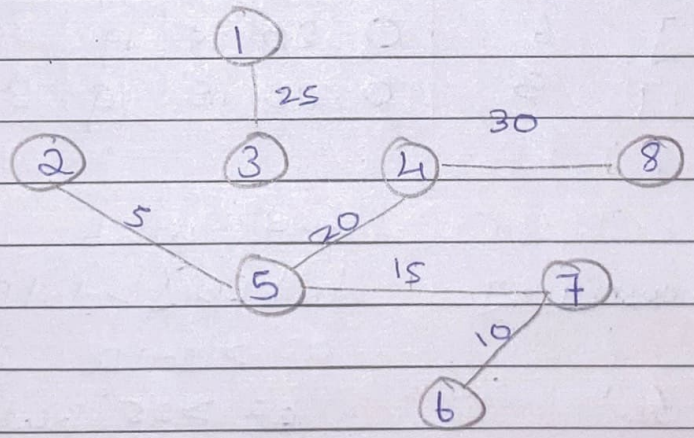
4)



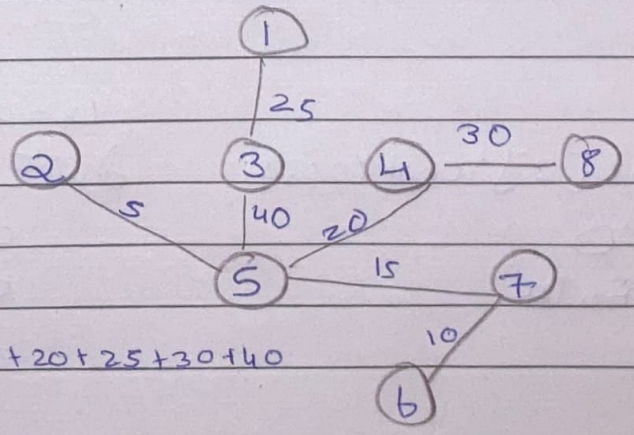
5)



6)



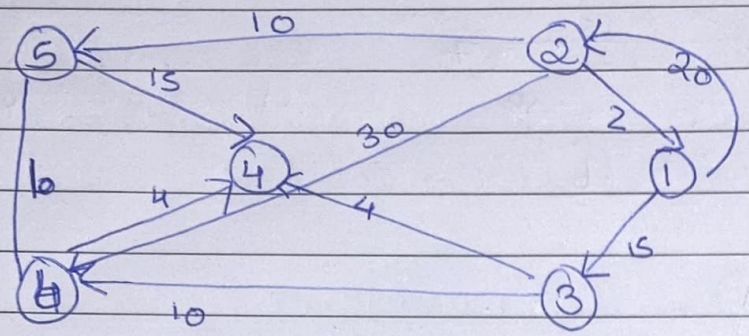
7)



mncost = 5 + 10 + 15 + 20 + 25 + 30 + 40

= 145

Q3] using Dijkstra's Algorithm obtain shortest path from Vertex 1 to all remaining vertices of following graph.



Iteration	S	Vertex selected	Distance [1]	[2]	[3]	[4]	[5]	[6]
Initial	----	----	0	20	15	∞	∞	∞
1	[1]	3	0	20	15	19	∞	25
2	[1,3]	4	0	20	15	19	∞	25
3	[1,3,4]	2	0	20	15	19	30	25
4	[1,3,4,2]	6	0	20	15	19	30	25
5	[1,3,4,2,6]	5	0	20	15	19	30	25

u = 3 w = 4

dist[4] > dist[3] + cost[3,4]

~~20~~ > 15 + 4

∞ > 19 True

u = 3 w = 6

dist[6] > dist[3] + cost[3,6]

∞ > 15 + 10

∞ > 25 True

w = 4 w = ∅

u = 2 w = 6

dist[6] > dist[2] + cost(2,6)

25 > 20 + 30

25 > 50 False

u = 2 w = 5

dist[5] > dist[2] + cost(2,5)

∞ > 20 + 10

∞ > 30 true

u = 6 w = 5

u = 6 w = 4

dist[5] > dist[6] + cost(6,5)
30 > 25 + 10
30 > 35 false

dist[4] > dist[6] + cost(6,4)
19 > 25 + 4
> 29 false

Q4] Explain Greedy approach towards algorithm design by starting Greedy method algorithm

- Greedy method is perhaps the most straight forward design technique

- This method suggest that one can devise an algorithm that works in stages, considering one i/p at a time

- At each stage a decision is made regarding whether particular i/p is an optimal solution.

- This is done considering i/p's in a order determined by some selection procedure.

- If the inclusion of next i/p, the partial constructed optimal solution, will result in non-feasible solution

- The this i/p is not added to partial solution, the selection procedure is itself based on some optimization measures.

- This version of greedy is called subset paradigm.

- The function select an i/p from a[] and removes it.

- The selected I/P values are assigned to x.

- Feasible is a Boolean valued function that determines whether x can be included into solution vector.

- The union combines x with solution and updates the objective function.

- The function greedy describes essential way that a greedy algo will look once a particular ~~part~~ problem is choosen and functions select, feasible & union are properly implemented

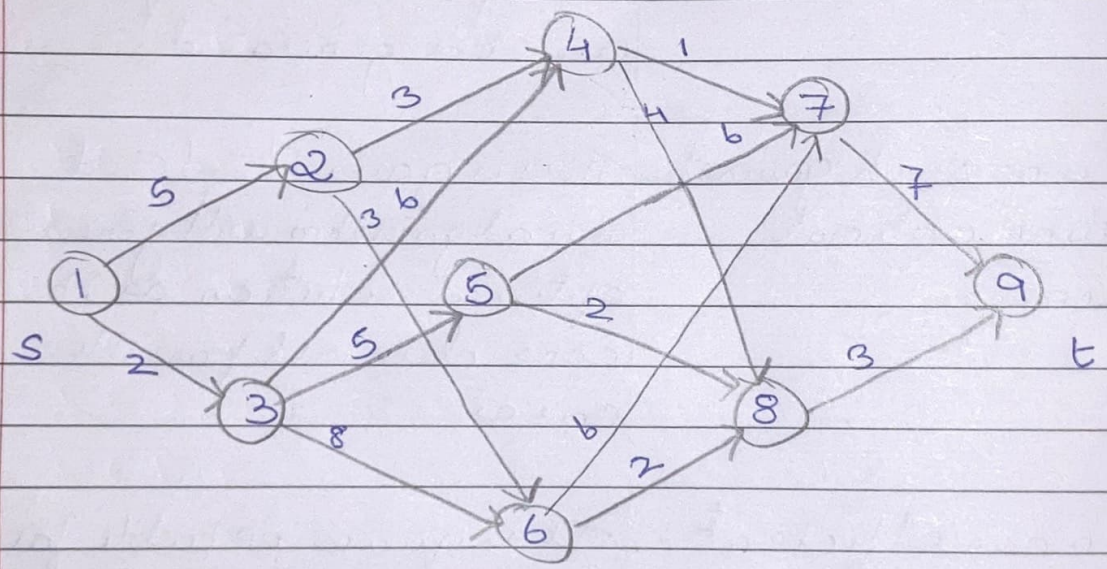
```
Algorithm Greedy (a,n)
// a[1:n] contains n inputs
{
    Solution := 0;
    for i := 1 to n do
    {
        x := select (a);
        if feasible (solution, x) then
        Solution := union (solution, x);
    }
    return solution
}
```

Q5] State the difference between Greedy method and Dynamic programming approach.

Greedy Method	Dynamic Programming
- Only 1 decision sequence is ever generated	In dynamic programming many decision sequence are ever generated.
- There is no such gaurantee of getting optimal sequences	This is gaurantee that this algorithm will generate optimal solution as it considers all possible cases.
- It is more efficient in terms of memory as it never looks back	It requires DP table for memorisation and increase memory complexity.
Eg: Fractional Knapsack	Eg: 0/1 Knapsack or multistage graph min cost

Q6] Find the minimum cost path from S to t in the following multistage graph using forward approach and Backward approach

[Also write the Algorithm]



* Forward Approach

1. Initial L
Cost (4, 7) = 7
Cost (4, 8) = 3

2. K = 3 i.e. V3

→ cost (3,4) = min {c(4,7) + cost (4,7), c(4,8) + cost 4, 8)}
= min {1 + 7, 4 + 3}
= min {8, 7}
min = 7 d = 8

$\rightarrow$ $Cost(3,5) = \min \begin{Bmatrix} C(5,7) + cost(4,7) \\ C(5,8) + cost(4,8) \end{Bmatrix}$

$= \min\{6+7, 2+3\}$

$= \min\{13, 5\}$

$min = 5 \quad l = 8$

$\rightarrow$ $Cost(3,6) = \min \begin{Bmatrix} C(6,7) + cost(4,7) \\ C(6,8) + cost(4,8) \end{Bmatrix}$

$= \min\{6+7, 2+3\}$

$= \min\{13, 5\}$

$min = 5 \quad l = 8$

3. <u>For $k = 2$</u>

$\rightarrow$ $Cost(2,2) = \min \begin{Bmatrix} C(2,4) + cost(3,4) \\ C(2,6) + cost(3,6) \end{Bmatrix}$

$= \min\{3+7; 3+5\}$

$= \min\{10, 8\}$

$min = 8 \quad l = 6$

$\rightarrow$ $cost(2,3) = \min \begin{Bmatrix} C(3,4) + cost(3,4) \\ C(3,5) + cost(3,5) \end{Bmatrix}$

$= \min\{6+7, 5+5\}$

$= \min\{13, 10\}$

$min = 10 \quad l = 5$

4) <u>For K = 1</u>

$\rightarrow$ $Cost(1,1) = \min \begin{Bmatrix} c(1,2) + cost(2,2) \\ c(1,2) + cost(2,3) \end{Bmatrix}$

$= \min\{5+8, 2+10\}$

$= \min\{13, 12\}$

$\min = 12 \qquad l = 3$

$d(3,4) = 8$
$d(3,5) = 8$
$d(3,6) = 8$
$d(2,2) = 6$
$d(2,3) = 5$
$d(1,1) = 3$

$V_2 = d(1,1) = 3$
$V_3 = d(2,3) = 5$
$V_4 = d(3, d(2,3)) = d(3,5) = 8$

The minimum cost path from 1 to 9 is

$1 \rightarrow 3 \rightarrow 5 \rightarrow 8 \rightarrow 9$

Page No :
Date

```
Algorithm FGraph (G, k, n, P)
{
    cost[n] := 0.0;
    for i := n-1 to 1 step -1 do
    {
        // compute cost[i]
        Let r be a vertex such that (i, r) is an edge of G and
        c[i, r] + cost[r] is minimum;
        cost[i] := c[i, r] + cost[r];
        d[i] := r;
    }
    // Find minimum cost path
    P[1] := 1;
    P[k] := n;
    for i := 2 to k-2 do P[i] := d[P[i-1]];
}
```

<u>Backward approach</u>

1. Initial L
   Cost(2,2) = 5
   Cost(2,3) = 2

2. <u>For k = 3</u> i.e. V3

→ cost = min{cost(2,2) + c(2,4); cost(2,3) + c(3,4)}
= min{5+3, 2+6}
= min{8, 8}

m = 8 L = 3 @ 2

$\rightarrow$ $cost(3,5) = min\{cost(2,3) + c(3,5)\}$
$= min\{2 + 5\}$

$min = 7 \quad L = 3$

$\rightarrow$ $cost(3,6) = min\{cost(2,2) + c(2,6), cost(2,3) + c(3,6)\}$
$= min\{5+3, 2+8\}$
$= min\{8, 10\}$

$min = 8 \quad L = 2$

2. <u>For $k=4$</u>

$\rightarrow$ $cost(4,7) = min\{cost(3,4) + c(4,7), cost(3,5) + c(5,7),$
$cost(3,5) + c(6,7)\}$
$= min\{8+1, 7+6, 8+6\}$
$= min\{5, 13, 14\}$

$min = 9 \quad L = 5$

$\rightarrow$ $cost(4,8) = min\{cost(8,4) + c(4,8), cost(3,5) + c(5,8),$
$cost(3,6) + c(6,8)\}$
$= min\{8+4, 7+2, 8+2\}$
$= min\{12, 9, 10\}$

$min = 9 \quad L = 5$

3) <u>For k=5</u>

$cost(5,9) = min(cost(4,7)+c(7,9), cost(4,8)+c(8,9))$
$= min\{9+7, 9+3\}$
$= min\{16, 12\}$

min = 12 l = 8

mincost path $s \to v_1 \to v_2 \to v_3 \to v_4 \to t$

$v_4 = d(5,9) = 8$
$v_3 = d(4,8) = 5$
$v_2 = d(3,5) = 3$

$S = 1 \to 3 \to 5 \to 8 \to 9$

```
Algorithm BGraph(G, k, n, p) //same as F graph
{
    bcost[1] = 0.0;
    for j := 2 to n do
    {
        // compute bcost[j]
        let r be such that <r, j> is an edge and bcost[r]
        + c[r, j] is minimum;
        bcost[j] := bcost[r] + c[r, j];
        d[j] := r;
    }
    //Find a minum-cost path
    p[1] := 1; p[k] := n;
    for j := k-1 to 2 do p[j] := d[p[j+1]];
}
```