# UNIT-IV

# GREEDY METHOD

Among all the algorithmic approaches, the simplest and straightforward approach is the Greedy method. In this approach, the decision is taken on the basis of current available information without worrying about the effect of the current decision in future.

## Components of Greedy Algorithm

Greedy algorithms have the following components –

**1. Candidate set** – A solution is created from this set.

**2. Selection function** – Used to choose the best candidate to be added to the solution.

**3. Feasible function** – Most problems have n inputs and its solution contains a subset of inputs that satisfies a given constraint (condition). Any subset that satisfies the constraint is called feasible solution.

**4. Optimal solution**: To find a feasible solution that either maximizes or minimizes a given objective function. A feasible solution that does this is called optimal solution.

**5. Objective function** – Used to assign a value to a solution or a partial solution.

**6. Solution function** – Used to indicate whether a complete solution has been reached.

**Control Abstraction of Greedy Method:** All of the gready problems have n inputs. It works in stages, by using some selection procedure, we arrange the input in an order and by considering one input at a time. At each stage, a decision is made regarding whether a particular input is in an optimal solution. If the inclusion of the next input into the partially constructed optimum solution will result in an infeasible solution, then this input is not added to the partial solution otherwise it is added.

```
Algorithm Greedy (a,n)
{
   solution =0; //Initialise the solution.
   for i=1 to n do
   {
      x=select(a);
      if(feasible(solution,x))then
     solution=union(solution,x);
   }
   return solution;
}
```

Feasible is a Boolean value function that determines whether X can be included into the solution vector.

The function Union combines X with The solution and updates the objective function.

A greedy algorithm will once a particular problem is chosen and the function subset, feasible & union are properly implemented.

## Applications of Greedy Method

1 : Knapsack Problems

2 : Job Sequencing with Deadlines

3 : Minimum Cost Spanning Tree.

4 : Single Source Shortest Paths.

## 1 : Knapsack Problems :

**1 :** Knapsack is an empty bag. Consider knapsack whose size is M. it contains n items with weights w1, w2, w3, ……, wn. The profits of each weight is p1, p2, p3, ……. , pn.

2 : The problem is we have to fill the bag with the help of N objects and the resulting profit has to be maximum.

3 : Formally the problem can be stated as

**Maximize $\sum P_i X_i$ subject to $W_i X_i \leq M$.**

Where xi is the fraction of object and it lies between 0 or 1.

4 : There are so many ways to solve this problem, which will give many feasible solution for which we have to find the optimal solution.

5 : But in this algorithm, it will generate only one solution which is going to be feasible as well as optimal.

### Algorithm for Knapsack Problem

```
Algorithm GreayKnapsack( m , n )
{
   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];
 }
```

**Ex: Consider the following instance of knapsack problem. N = 3, M = 20, ( P1, P2, P3 )=(25, 24, 15) , ( W1, W2, W3 ) = ( 18, 15, 10 ). Find the optimal solution for Maximum profit, Minimum weight, Maximum profit per unit weight.**

**Ans :** The feasible solutions are as given below.

**Maximum Profit:** In this case, we will place an item in the bag whose profit is maximum.

X1 = 1 complete item w1 is kept in the bag. i.e, 18 only 2 units place is left ( 20 – 18 ) = 2.

X2 = left out space / weight of item to be placed = 2 / 15.When w2 is placed no space is left in bag.

X3 = 0

**Let us compute**

$\sum P_i X_i$ = p1x1 + p2x2 + p3x3
= 25 * 1 + 24 * 2 / 15 + 15 * 0
= 25 + 3.2 + 0
= 28 .2

$\sum W_i X_i$ = w1x1 + w2x2 + w3x3
= 18 * 1 + 15 * 2 / 15 + 10 * 0
= 18 + 2 + 0
= 20

**Minimum Weight:** In this case, we will place an item in the bag whose weight is minimum.

X3 = 1 complete item w3 is kept in the bag. i.e, 10 only 10 units place is left ( 20 – 10 ) = 10.

X2 = left out space / weight of item to be placed = 10 / 15 = 2 / 3.When w2 is placed no space

is left in bag.

X1 = 0

**Let us compute**

$\sum P_i X_i$ = p1x1 + p2x2 + p3x3
= 25 * 0 + 24 * 2 / 3+ 15 * 1
= 0 + 16 + 15
= 31

$\sum W_i X_i$ = w1x1 + w2x2 + w3x3
= 18 * 0 + 15 * 2 / 3 + 10 * 1
= 0 + 10 + 10
= 20

**Maximum profit per unit weight :** whose profit per unit weight ratio is maximum.

$$\frac{P1}{W1} = \frac{25}{18} = 1.4$$

$$\frac{P2}{W2} = \frac{24}{15} = 1.6$$

$$\frac{P3}{W3} = \frac{15}{10} = 1.5$$

P2 / w2 is maximum so,X2 = 1. now remaining weight of bag is 20 – 15 =5. next maximum is p3 / w3. now we have to place third item in bag but no space is not sufficient. So we have to place 5 / 10 ( i.e, 1 / 2 ) therefore x3 = 1 / 2. The bag is already filled, so

x1 = 0
x2 = 1
x3 = 1 / 2

**Let us compute**

$\sum \mathbf{P_i X_i}$ = p1x1 + p2x2 + p3x3

= 25 * 0 + 24 * 1+ 15 * 1 / 2

= 0 + 24 + 7.5

= 31.5

$\sum \mathbf{W_i X_i}$ = w1x1 + w2x2 + w3x3

= 18 * 0 + 15 * 1 + 10 * 1 / 2

= 0 + 15 + 5

= 20

To summarize this,

| $\sum \mathbf{W_i X_i}$ | $\sum \mathbf{P_i X_i}$ |
|---|---|
| 20 | 28.2 |
| 20 | 31 |
| **20** | **31.5** |

The solution 3 gives maximum profit and hence to be optimum solution.

**Example 2: Find the optimal solution for the fractional knapsack problem making use of greedy approach. Consider n = 5, w = 60 kg,**

| Item | Weight | Profit |
|---|---|---|
| 1 | 5 | 30 |
| 2 | 10 | 40 |
| 3 | 15 | 45 |
| 4 | 22 | 77 |
| 5 | 25 | 90 |

**Solution:** The feasible solutions are as given below.

**Maximum Profit:** In this case, we will place an item in the bag whose profit is maximum.

x5 = 1 ( 60-25 = 35 )

x4 = 1 ( 35 – 22 ) = 13

x3 = 13/15

x2 = 0

x1 = 0

**Let us compute**

$\sum P_i X_i$ = p1x1 + p2x2 + p3x3 + p4x4 + p5x5

= 30*0 + 40*0 + 45*13/15 + 77*1 + 90*1

= 0 + 0 + 585/15 + 77 + 90

= 0 + 0 + 39 + 77 + 90

= 206

$\sum w_i X_i$ = w1x1 + w2x2 + w3x3 + w4x4 + w5x5

= 5*0 + 10*0 + 15*13/15 + 22*1 + 25*1

= 0 + 0 + 195/15 + 22 + 25

= 0 + 0 + 13 + 22 + 25

= 60

**Minimum Weight:** In this case, we will place an item in the bag whose weight is minimum.

x1 = 1 ( 60-5 = 55 )

x2 = 1 ( 55 – 10 = 45 )

x3 = 1 ( 45 – 15 = 30 )

x4 = 1 ( 30 – 22 = 8 )

x5 = 8 / 25

**Let us compute**

$\sum \mathbf{P_i X_i}$ = p1x1 + p2x2 + p3x3 + p4x4 + p5x5

= 30*1 + 40*1 + 45*1 + 77*1 + 90*8/25

= 30 + 40 + 45 + 77 + 720/25

= 30 + 40 + 45 + 77 + 28.8

= 220.8

$\sum \mathbf{w_i X_i}$ = w1x1 + w2x2 + w3x3 + w4x4 + w5x5

= 5*1 + 10*1 + 15*1 + 22*1 + 25*8/25

= 5 + 10 + 15 + 22 + 200/25

= 5 + 10 + 15 + 22 + 8

= 60

**Maximum profit per unit weight :** whose profit per unit weight ratio is maximum.

$$\frac{P1}{W1} = \frac{30}{5} = 6$$

$$\frac{P2}{W2} = \frac{40}{10} = 4$$

$$\frac{P3}{W3} = \frac{45}{15} = 3$$

$$\frac{P4}{W4} = \frac{77}{22} = 3.5$$

$$\frac{P5}{W5} = \frac{90}{25} = 3.6$$

P1 / w1 is maximum, so

x1 = 1 ( 60-5 = 55 )

x2 = 1 ( 55 – 10 = 45 )

x5 = 1 ( 45 – 25 = 20 )

x4 = 20/22

x3 = 0

**Let us compute**

$\sum P_i X_i$ = p1x1 + p2x2 + p3x3 + p4x4 + p5x5

= 30*1 + 40*1 + 45*0 + 77*20/22 + 90*1

= 30 + 40 + 0 + 1540/22 + 90

= 30 + 40 + 0 + 70 + 90

= 230

$\sum w_i X_i$ = w1x1 + w2x2 + w3x3 + w4x4 + w5x5

= 5*1 + 10*1 + 15*0 + 22*20/22 + 25*1

= 5 + 10 + 0 + 440/22 + 25

= 5 + 10 + 0 + 20 + 25

= 60

**To summarize this,**

| $\sum W_i X_i$ | $\sum P_i X_i$ |
|---|---|
| **60** | **206** |
| **60** | **220.8** |
| **60** | **230** |

The solution 3 gives maximum profit and hence to be optimum solution.

## 2. Job Sequencing With Deadlines-

The sequencing of jobs on a single processor with deadline constraints is called as Job Sequencing with Deadlines.

**Here-**

- You are given a set of jobs.
- Each job has a defined deadline and some profit associated with it.
- The profit of a job is given only when that job is completed within its deadline.
- Only one processor is available for processing all the jobs.
- Processor takes one unit of time to complete a job.

**Approach to Solution:**

- A feasible solution would be a subset of jobs where each job of the subset gets completed within its deadline.
- Value of the feasible solution would be the sum of profit of all the jobs contained in the subset.
- An optimal solution of the problem would be a feasible solution which gives the maximum profit.

### Algorithm for Job Sequencing With Deadlines

```
Algorithm JS(d,j,n)
{
        d[0]= J[0]=0;
        J[1]=1;
        K=1;
        for i =2 to n do
        {
                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
                {
                        for q=k to (r+1) step –1 do J [q+1]=j[q]
                        J[r+1]=i;
                        K=k+1;
               }
       }
  return k;
}
```

**Application** :

1. Job scheduling to processor
2. Flight landling or take off an airport
3. Car repair workshop.

**Procedure:**

The greedy algorithm described below always gives an optimal solution to the job sequencing problem-

**Step-01:**

- Sort all the given jobs in decreasing order of their profit.

**Step-02:**

- Check the value of maximum deadline.
- Draw a Gantt chart where maximum time on Gantt chart is the value of maximum deadline.

**Step-03:**

- Pick up the jobs one by one.
- Put the job on Gantt chart as far as possible from 0 ensuring that the job gets completed before its deadline.

**Example 1: Find the solution generated by the job sequencing with deadlines. N = 4,**

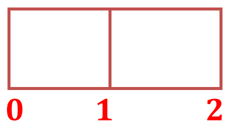
| Jobs | J1 | J2 | J3 | J4 |
|---|---|---|---|---|
| **Deadlines** | 2 | 1 | 2 | 1 |
| **Profits** | 100 | 10 | 15 | 27 |

**Solution:** Sort all the given jobs in decreasing order of their profit.

| Jobs | J1 | J4 | J3 | J2 |
|---|---|---|---|---|
| **Deadlines** | 2 | 1 | 2 | 1 |
| **Profits** | 100 | 27 | 15 | 10 |

Value of maximum deadline = 2. Maximum of two jobs will form the feasible solution.

So, draw a Gantt chart with maximum time on Gantt chart = 2 units as shown-



**Feasible solution and their profits**

| S.No | Possibility Jobs | Feasible Solution | Processing Sequence | ∑ Pi |
|---|---|---|---|---|
| **1** | J= { 1 } | Yes | 1 | 100 |
| **2** | J= { 4 } | Yes | 4 | 27 |
| **3** | J= { 3 } | Yes | 3 | 15 |
| **4** | J= { 2 } | Yes | 2 | 10 |
| **5** | J = { 1, 4 } | Yes | ( 1,4 ) reject, But (4,1 ) Accepted. | 100+27=127 |
| **6** | J = { 1, 3 } | Yes | ( 1,3 ) or (3,1 ) Accepted. | 100+15=115 |
| **7** | J = { 1, 2 } | Yes | ( 1,2 ) reject, But (2,1 ) Accepted. | 100+10=110 |
| **8** | J = { 4, 3 } | Yes | ( 4,3 ) or (3,4 ) Accepted. | 15+27=42 |
| **9** | J = { 4, 2 } | No | Reject | ------- |
| **10** | J = { 3, 2 } | Yes | ( 3,2 ) reject, But (2,3 ) Accepted. | 15+10=25 |

Job 1 is added as it has the largest profit and { 1 } is a feasible solution. Next, job 4 is considered, { 1, 4 } is also feasible. Next, job 3 is considered, { 1,3 } is also feasible solution. Next, job 2 is considered and is also feasible solution.

In this problem { 1, 4 } is the only feasible solution with maximum **∑ pi.** Therefore , J = ( 1, 4 ) is the optimal solution and the **optimal profit is 127** for the given problem instance.

**Example 2 : Find the solution generated by the job sequencing with deadlines . N = 7,**

**( p1, p2, p3, p4, p5, p6, p7 ) = (3, 5, 20, 18, 1, 6, 30 ),**

**( d1, d2, d3,d4, d5, d6, d7 ) = ( 1, 3, 4, 3, 2, 1, 2 ).**

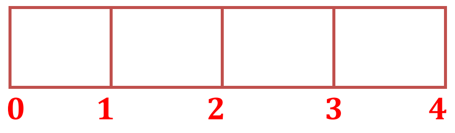
**Solution:** Sort all the given jobs in decreasing order of their profit.

(p7, p3, p4, p6, p2, p1, p5) = (30, 20, 18, 6, 5, 3, 1)

(d7, d3, d4, d6, d2, d1, d5) = (2, 4, 3, 1, 3, 1, 2)

Value of maximum deadline = 4. Maximum of four jobs will form the feasible solution.

So, draw a Gantt chart with maximum time on Gantt chart = 4 units as shown-



**Feasible solution and their profits**

| S.No | Possibility Jobs | Feasible Solution | Processing Sequence | ∑ Pi |
|---|---|---|---|---|
| **1** | J= { 1 } | Yes | 1 | 3 |
| **2** | J= { 2 } | Yes | 2 | 5 |
| **3** | J= { 3 } | Yes | 3 | 20 |
| **4** | J= { 4 } | Yes | 4 | 18 |
| **5** | J= { 5 } | Yes | 5 | 1 |
| **6** | J= { 6 } | Yes | 6 | 6 |
| **7** | J= { 7 } | Yes | 7 | 30 |
| **8** | J = { 7, 3 } | Yes | (7, 3) or (3, 7) Accepted. | 30+20=50 |
| **9** | J = { 7, 4 } | Yes | (7, 4) or (4, 7) Accepted. | 30+18=48 |

| | | | | |
|---|---|---|---|---|
| **10** | J = { 7, 6 } | Yes | ( 7,6 ) reject, But (6,7 ) Accepted. | 30+6=36 |
| **11** | J = { 7, 2 } | Yes | (7, 2) or (2, 7) Accepted. | 30+5=35 |
| **12** | J = { 7, 1 } | Yes | ( 7,1) reject, But (1,7 ) Accepted. | 30+3=33 |
| **13** | J = { 7, 5 } | Yes | (7, 5) or (5, 7) Accepted. | 30+1=31 |
| **14** | J = { 7, 3, 4 } | Yes | ( 7, 3, 4 ) or ( 7, 4, 3 ) Accepted. | 30+20+18=68 |
| **15** | J = { 7, 3, 6 } | Yes | ( 7, 3, 6 ) reject, But ( 6, 7, 3 ) accepted | 30+20+6=56 |
| **16** | J = { 7, 3, 2 } | Yes | ( 7, 3, 2 ) or ( 2, 7, 3 ) Accepted. | 30+20+5=55 |
| **17** | J = { 7, 3, 1 } | Yes | ( 7, 3, 1) reject, But ( 1, 7, 3 ) accepted | 30+20+3=53 |
| **18** | J = { 7, 3, 5 } | Yes | ( 7, 3, 5) reject, But ( 5, 7, 3 ) accepted | 30+20+1=51 |
| **19** | J = { 7, 3, 4, 6 } | Yes | ( 7, 3, 4, 6 ) reject, But ( 6, 7, 4, 3 ) accepted | 30+20+18+6=74 |
| **20** | J = { 7, 3, 4, 2 } | Yes | ( 7, 3, 4, 2 ) reject, But ( 7, 4, 2, 3 ) accepted | 30+20+18+5=73 |
| **21** | J = { 7, 3, 4, 1 } | Yes | ( 7, 3, 4, 1 ) reject, But ( 1, 7, 4, 3 ) accepted | 30+20+18+3=71 |
| **22** | J = { 7, 3, 4, 5 } | Yes | ( 7, 3, 4, 5 ) reject, But ( 5, 7, 4, 3 ) accepted | 30+20+18+1=69 |

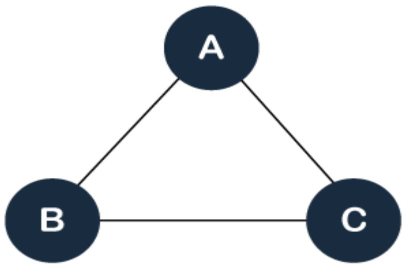
Job 7 is added as it has the largest profit and { 7 } is a feasible solution. Next, job 3 is considered, { 7, 3 } is also feasible. Next, job 4 is considered, { 7, 3, 4 } is also feasible solution. Next, job 6 is considered, { 6, 7, 4, 3 } and is also feasible solution.

In this problem { 6, 7, 4, 3 } is the only feasible solution with maximum **$\sum$ pi.** **Therefore, J = (6, 7, 4, 3 ) is the optimal solution and the optimal profit is 74 for the given problem instance.**

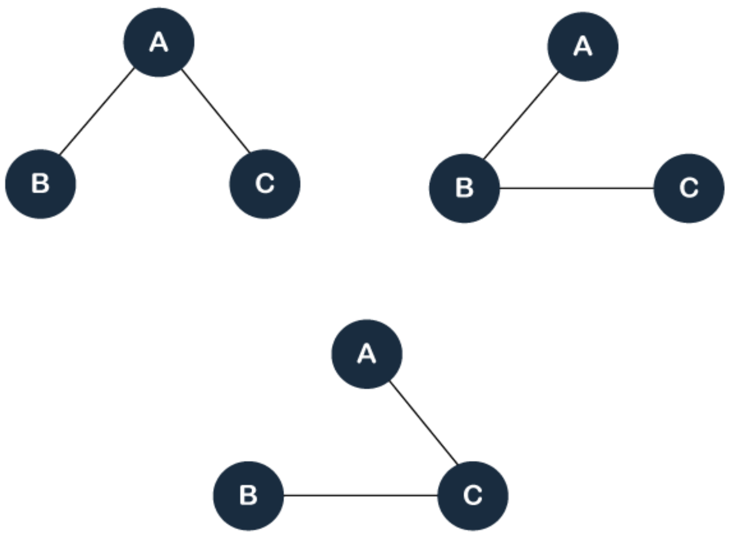
## 3. SPANNING TREES:

- Spanning tree of a graph G is a sub graph, which has all the vertices covered with minimum possible no of edges.
- Hence, a spanning tree for a graph has no cycles and it cannot be disconnected.
- If there are n vertices in the graph, then each spanning tree has n vertices and n – 1 edges.

**Example:**



**Solution:** The possible spanning trees are



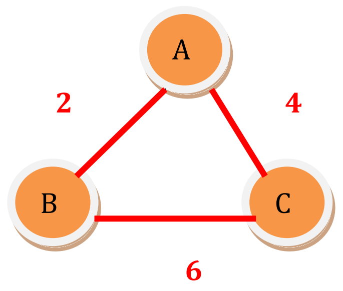
**Spanning tree can be classified into two types.**

1 : Minimum spanning tree.

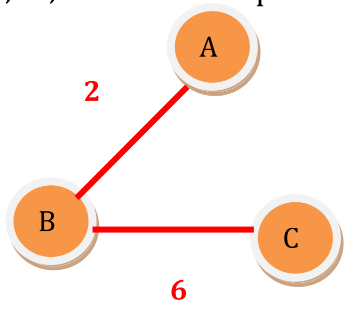
2 : Maximum spanning tree.

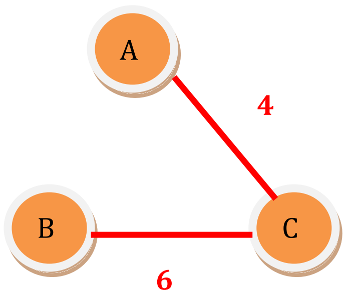
**Minimum Spanning Tree :** The spanning tree with smallest weight is called minimum spanning tree. The spanning tree with largest weight is called maximum spanning tree. The minimum and maximum spanning tree is not unique. But their weights must be same.

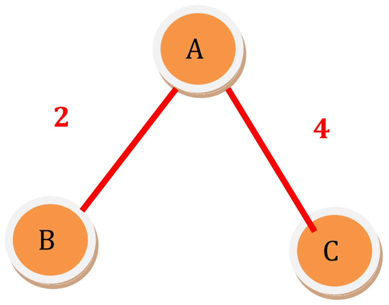
**Example:**



If the graph consist of n vertices then the possible spanning trees are $n^{n-2}$, in the above example n = 3, i.e, $3^{3-2} = 3^1 = 3$ spanning trees. These spanning trees can be



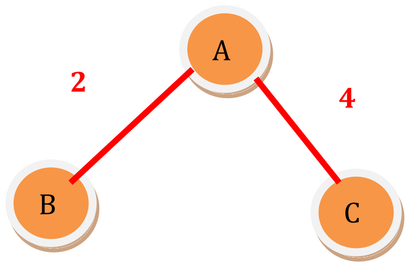




**Total Weight = 2 + 6 = 8** **Total Weight = 4+ 6 = 10** **Total Weight = 2+ 4 = 6**

( A ) ( B ) ( C )

Among these spanning trees, total cost has minimum, so it is called minimum spanning tree for this graph, the minimum spanning tree is



**Total Weight = 2+ 4 = 6**

Similarly, for N = 6, $6^{6-2} = 6^4 = 1296$ possible spanning trees, draw and find the cost of all possible spanning trees. It is a time consuming and difficult process as n value increases. So to avoid this difficulty, we will use two standard algorithms.

1 : Kruska'l Algorithm.

2 : Prim's algorithem.

## 1. Kruskal's Algorithm-

- Kruskal's Algorithm is a famous greedy algorithm.
- It is used for finding the Minimum Spanning Tree (MST) of a given graph.
- To apply Kruskal's algorithm, the given graph must be weighted, connected and undirected.

## Kruskal's Algorithm Implementation-

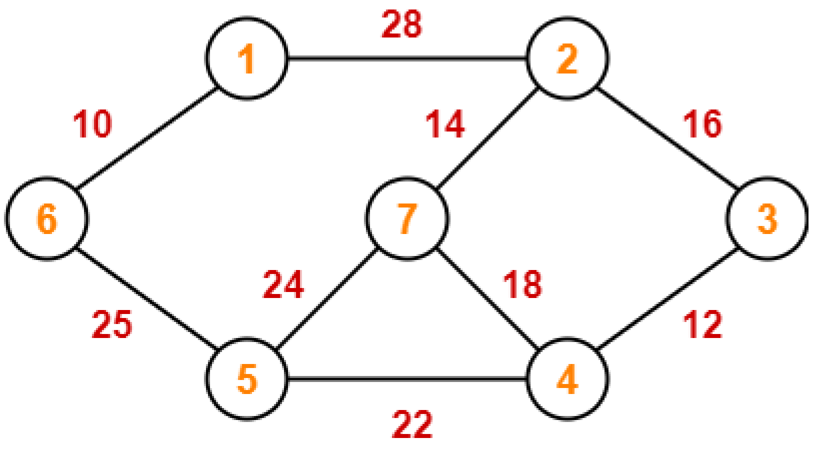
The implementation of Kruskal's Algorithm is explained in the following steps-

1. Sort all the edges in decreasing order of their weights (low weight to high weight).
2. Take the edge with the lowest weight and use it to connect the vertices of graph. If adding an edge creates a cycle, then reject that edge and go for the next least weight edge.
3. Keep adding edges until all the vertices are connected and a Minimum Spanning Tree ( MST ) is obtained.
4. If there are n vertices in the graph, then each spanning tree has n vertices and n – 1 edges.

```
Algorithm Kruskal (E, cost, n,t)
{
        for i:= 1 to n do parent[i]:= -1
        i: = 0; mincost: = 0.0;
        while (i<n-1) and (heap not empty))do
        {
                Delete a minimum cost edge (u,v) from the heaps; and 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;
}
```

**Time complexity of Kruskal's Algorithm = O(ElogV) or O(ElogE)**

**Example: Construct the minimum spanning tree (MST) for the given graph using Kruskal's Algorithm-**

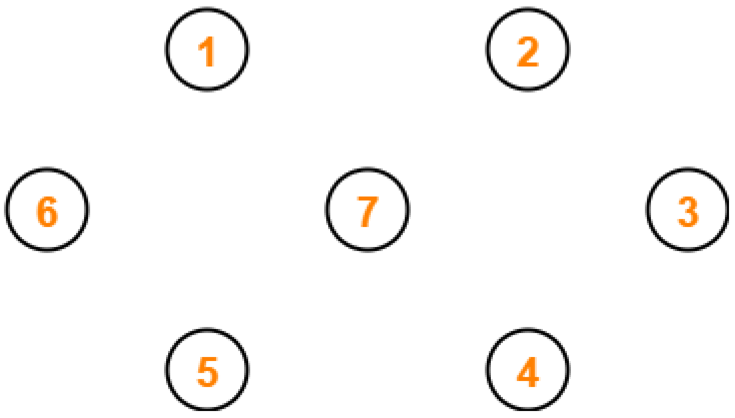


**Solution:** This graph contains 7 vertices and thus we obtain a spanning tree with 6 edges

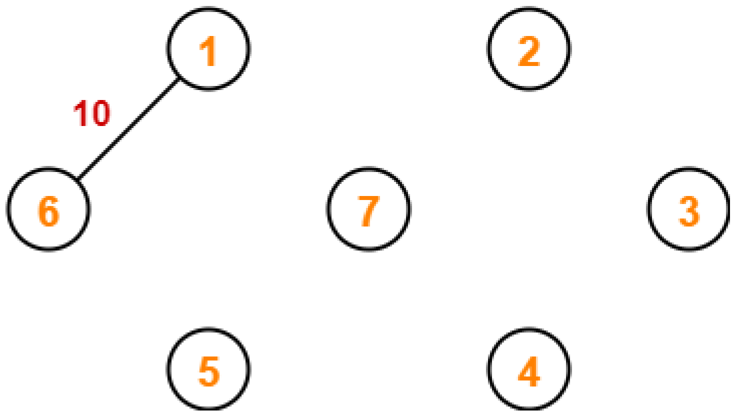
Sort all the edges in decreasing order of their weights (low weight to high weight).

| **Edge** | **(1,6)** | **(3,4)** | **(2,7)** | **(2,3)** | **(4,7)** | **(4,5)** | **(5,7)** | **(5,6)** |
|---|---|---|---|---|---|---|---|---|
| **Weight** | 10 | 12 | 14 | 16 | 18 | 22 | 24 | 25 |

**Step 1:**

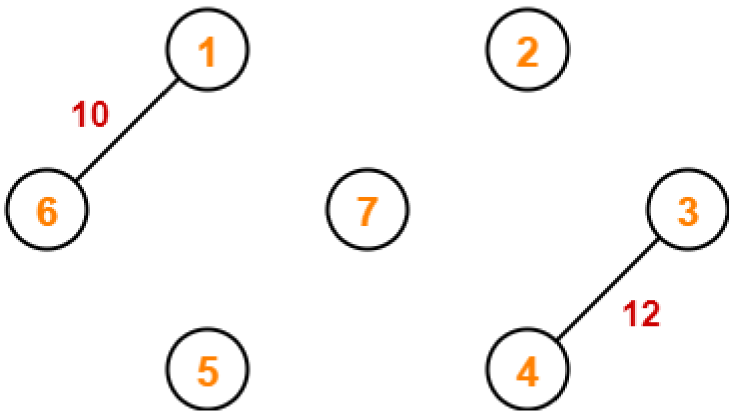


**Step 2:** The first edge in the sorted list i.e, (1,6) that has the least cost is included in the spanning tree. Then the edges 1 and 6 are merged as single set.



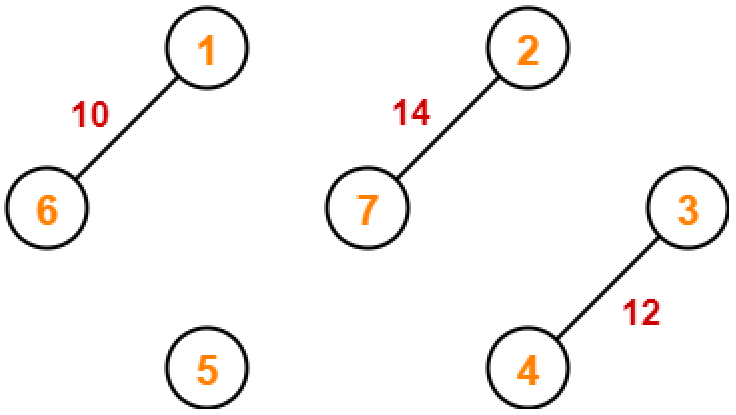
**Total weight = 10**

**Step 3:** The next edge in the sorted list i.e, (3,4) that has the least cost is included in the spanning tree. Then the edges 3 and 4 are merged as single set.



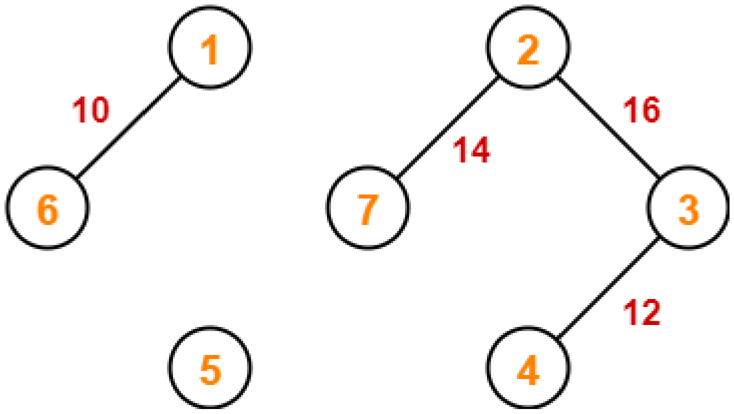
**Total weight = 10 + 12 = 22**

**Step 4:** The next edge in the sorted list i.e, (2,7) that has the least cost is included in the spanning tree. Then the edges 2 and 7 are merged as single set.



**Total weight = 10 + 12 + 14 = 36**

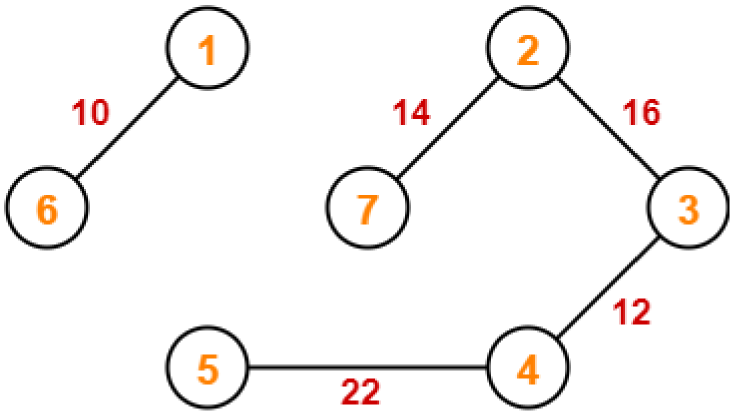
**Step 5:** The next edge in the sorted list i.e, (2,3) that has the least cost is included in the spanning tree. Then the edges 2 and 3 are merged as single set.



**Total weight = 10 + 12 + 14 + 16 = 52**

**Step 6:** The next edge in the sorted list i.e, (4,7) that has the least cost is included in the spanning tree. An edge creates a cycle, and then rejects that edge and go for the next least weight edge.

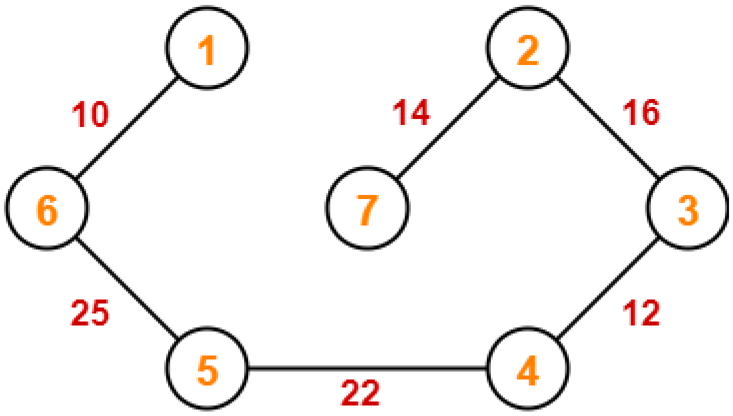
**Step 7:** The next edge in the sorted list i.e, (4,5) that has the least cost is included in the spanning tree. Then the edges 4 and 5 are merged as single set.



**Total weight = 10 + 12 + 14 + 16 + 22 = 74**

**Step 8:** The next edge in the sorted list i.e, (5,7) that has the least cost is included in the spanning tree. An edge creates a cycle, and then rejects that edge and go for the next least weight edge.

**Step 9:** The next edge in the sorted list i.e, (5,6) that has the least cost is included in the spanning tree. Then the edges 5 and 6 are merged as single set.



**Total weight = 10 + 12 + 14 + 16 + 22 + 25 = 99**

## 2.Prim’s Algorithm

- Prim’s Algorithm is a famous greedy algorithm.
- It is used for finding the Minimum Spanning Tree (MST) of a given graph.
- To apply Prim’s Algorithm, the given graph must be weighted, connected and undirected.

## Prim’s Algorithm Implementation-

The implementation of Prim’s Algorithm is explained in the following steps-

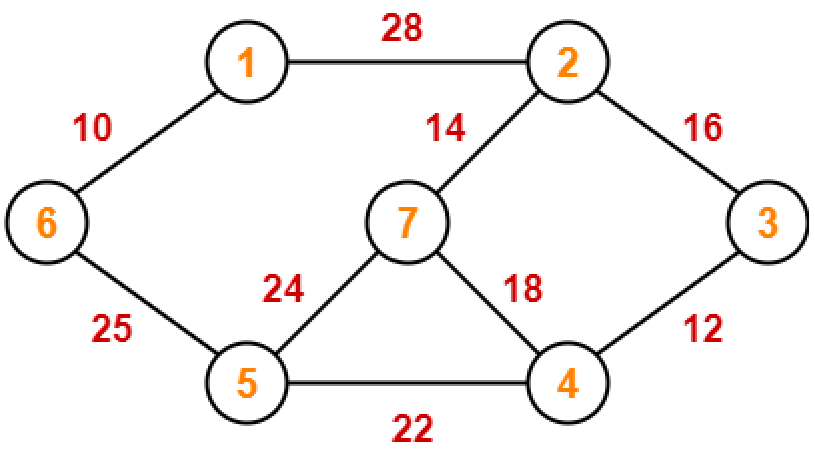
1. Randomly choose any vertex. The vertex connecting to the edge having least weight is usually selected.
2. Find all the edges that connect the tree to new vertices. Find the least weight edge among those edges and include it in the existing tree. If including that edge creates a cycle, then reject that edge and look for the next least weight edge.
3. Keep repeating step-02 until all the vertices are included and Minimum Spanning Tree (MST) is obtained.

4.If there are n vertices in the graph, then each spanning tree has n vertices and n – 1 edges.

**Algorithm :** Prim( G[0….size – 1, 0…size – 1, nodes )

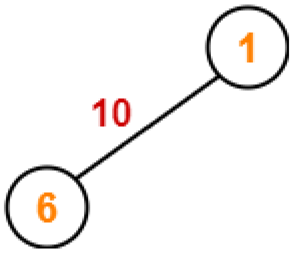
```
{
        total = 0;
        for i = 0 to nodes-1 do
        tree[i] = 0, tree[0] = 1;
        for k = 1 to nodes do
        {
                min_dist = ∞ ;
                for i = 0 to nodes-1 do
                {
                        for j = 0 to nodes-1 do
                        {
                                if ( G[i,j] AND ((tree[i] AND tree[j] ) OR (!tree[i] AND tree[j] )))then
                                {
                                        if(G[i,j] < min_dist)then
                                        {
                                                min_dist = G[ij];
                                                v1 = i;
                                                v2 = j;
                                        }
                                }
                        }
                }
                write(v1, v2, min_dist);
                tree[v1] = tree[v2] = 1;
                total = total + min_dist;
        }
        write ( “ Total path lenth is”, total);
}
```

**Example: Construct the minimum spanning tree (MST) for the given graph using Prim's Algorithm**



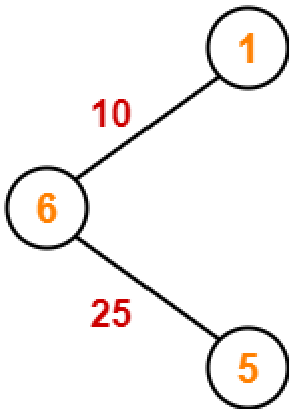
**Solution :** This graph contains 7 vertices and thus we obtain a spanning tree with 6 edges

**Step 1: The** starting node is 6. The possible edges from 6 that can be added in the minimum spanning tree 'S' are ( 6, 1 ), ( 6, 5). The costs of these edges are 10 and 25. Since, the edge ( 6, 1) has the lowest cost among the possible edges. It is included in the tree 'S'.



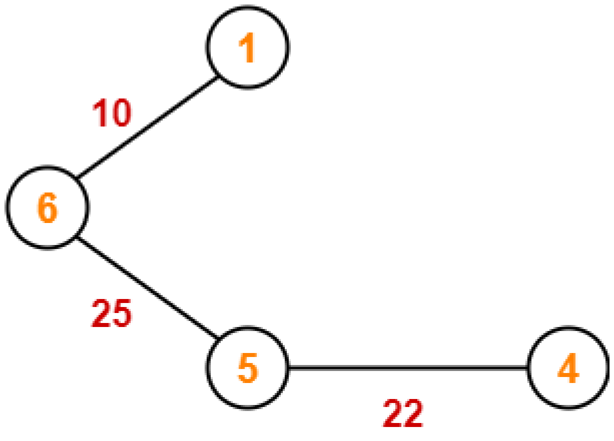
**Total Weight = 10**

**Step 2:** The nodes included in tree S are 6 and 1 . The possible edges from these nodes that can be added in the minimum spanning tree 'S' are ( 1, 2 ), ( 6, 5 ). The costs of these edges are 28 and 25. Since, the edge (6, 5) has the lowest cost among the possible edges. It is included in the tree 'S'.



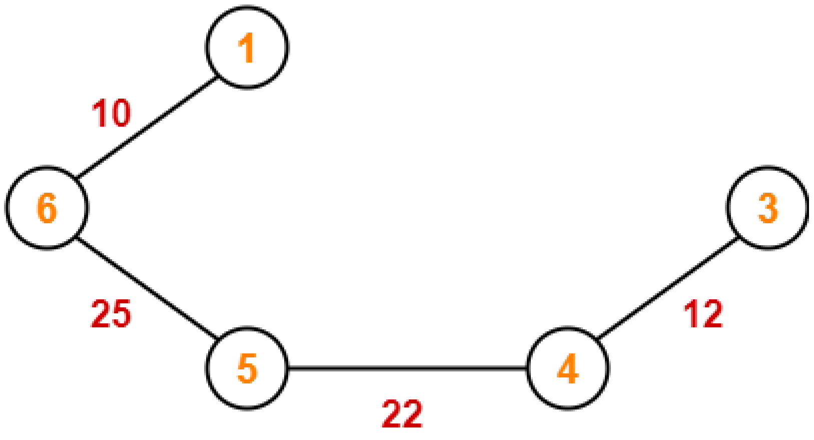
**Total Weight = 10 + 25 = 35**

**Step 3:** The nodes included in tree S are 1, 6 and 5 . The possible edges from these nodes that can be added in the minimum spanning tree 'S' are ( 1, 2 ), ( 5, 4 ) ( 5, 7). The costs of these edges are 28 , 22 and 24. Since, the edge (5, 4) has the lowest cost among the possible edges. It is included in the tree 'S'.



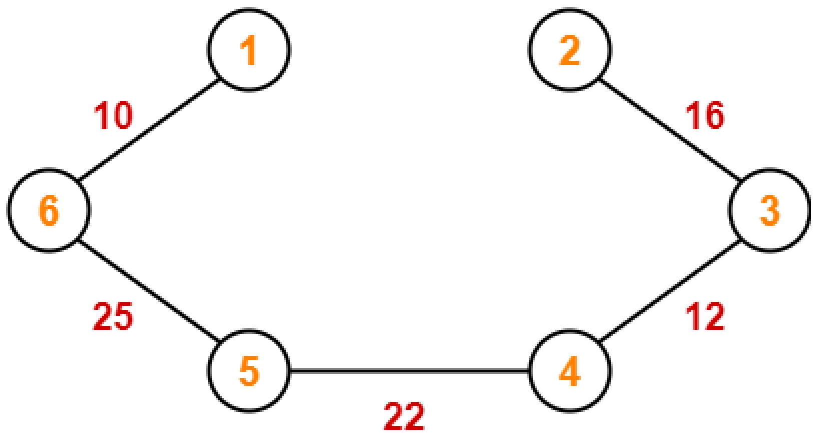
**Total Weight = 10 + 25 + 22 = 57**

**Step 4:** The nodes included in tree S are 1,6,5 and 4 . The possible edges from these nodes that can be added in the minimum spanning tree 'S' are ( 1, 2 ), ( 5, 7 ), ( 4, 3 ) ( 4, 7). The costs of these edges are 28 , 24, 12 and 18. Since, the edge (4, 3) has the lowest cost among the possible edges. It is included in the tree 'S'.



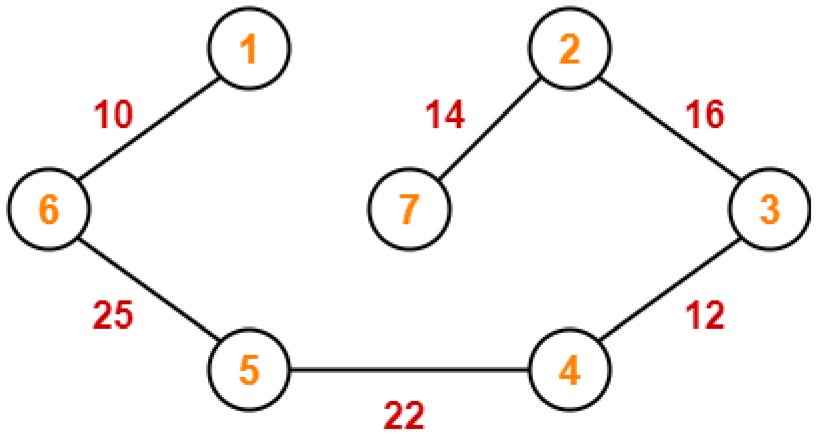
**Total Weight = 10 + 25 + 22 + 12 = 69**

**Step 5:** The nodes included in tree S are 1,6,5,4 and 3 . The possible edges from these nodes that can be added in the minimum spanning tree 'S' are ( 1, 2 ), ( 5, 7 ) ( 4, 7 ) ( 3, 2 ). The costs of these edges are 28 24, 18 and 16. Since, the edge (3, 2) has the lowest cost among the possible edges. It is included in the tree 'S'.



**Total Weight = 10 + 25 + 22 + 12 + 16 = 85**

**Step 6:** The nodes included in tree S are 1,6,5,4,3 and 2 . The possible edges from these nodes that can be added in the minimum spanning tree 'S' are ( 1, 2 ), ( 5, 7 ) ( 4, 7 ) ( 2, 7). The costs of these edges are 28, 24, 18 and 14. Since, the edge (2, 7) has the lowest cost among the possible edges. It is included in the tree 'S'.



**Total Weight = 10 + 25 + 22 + 12 + 16 + 14 = 99**

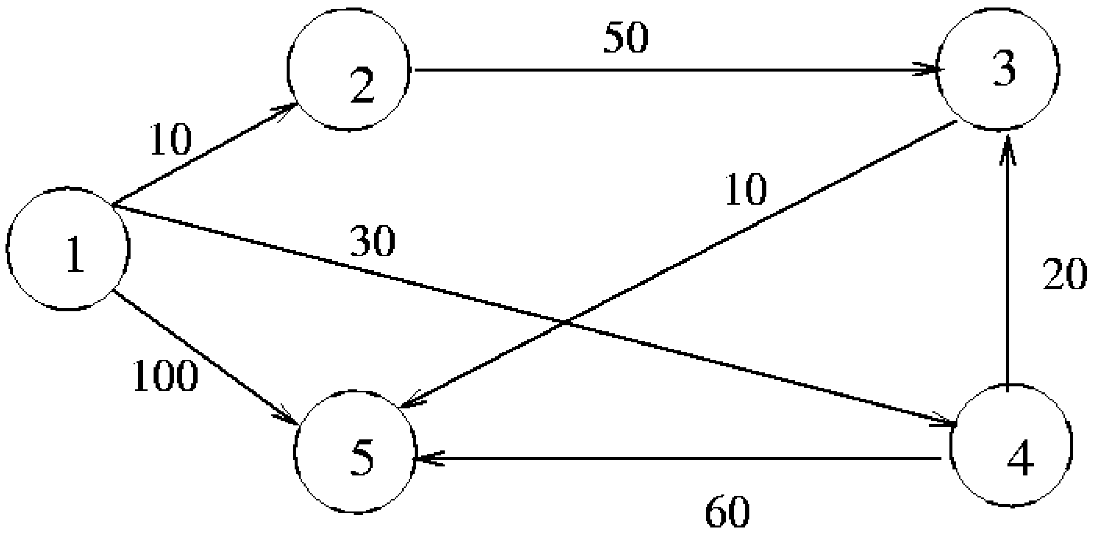
**4: Single Source Shortest Path Problem:** **L**et G = ( V, E ) be a directed graph with weighting function w for the edges of G. The starting vertex of path is called the source and the last vertex is called destination. Let v be the any other vertex which belongs to set of vertices V. the problem to determine a shortest path to given destination vertex v from source is called single source shortest path problem.

To formulate a greedy based algorithm to generate the shortest paths, we must conceive of a multistage solution to the problem and also of an optimization measure. One possibility is to construct the shortest paths one by one. As an optimization measure, we can use the sum of the lengths of all paths so far generated. This measure is to be minimized.

```
Algorithm ShortestPath( g,n,w,dist,v )
{
    for i : 1 to n do
    {
        S[i] = false;
        dist[i]=cost[v,i];
    }
    s[v]=true;
    dist[v]=0;
    for k = 2 to n – 1 do
    {
        s[u]:=true;
        for( each w adjacent to u with s[w] = false ) do
        {
            //UPDATE DISTANCES
            If( dist[w] > dist[u] + cost[u,w] ) then
                dist[w]: dist[u]+cost[u,w];
        }
    }

}
```

**Example: Consider the Graph**



**Solution :**

**Initially :**

V = { 1, 2, 3, 4, 5 }
S = { 1 }

D[2] = 10
D[3] = ∞
D[4] = 30
D[5] = 100

**Iteration 1 :**

Select u = 2, so that S = { 1,2 }

**D[w] = min (D[w], D[u] + C[u, w])**

**Select w.**

W=V-S=*{1,2,3,4,5}*-{1, 2}={3,4,5}

**D[w] = min (D[w], D[u] + C[u, w])**

**D[3] = min ( D[3], D[2] +** C[2,3])

min ( ∞, 10 + 50 )

min ( ∞, 60 ) = 60

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

min ( 30, 10 + ∞ )

min ( 30, ∞ ) = 30

**D[5] = min ( D[5], D[2] +** C[2,5])

min ( 100, 10 + ∞ )

min ( 100, ∞ ) = 100

**Iteration 2 :**

**Select** u = 4, so that S = { 1,2,4 }

**D[w] = min (D[w], D[u] + C[u, w])**

**Select w.**

W=V-S=*{1,2,3,4,5}*-{1, 2,4}={3, 5}

**D[w] = min (D[w], D[u] + C[u, w])**

**D[3] = min ( D[3], D[4] +** C[4,3])

min ( 60, 30 + 20 )

min ( 60, 50 ) = 50

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

min ( 100, 30 + 60 )

min ( 100, 90 ) = 90

**Iteration 3 :**

**Select** u = 3, so that S = { 1,2,4, 3 }

**D[w] = min (D[w], D[u] + C[u, w])**

**Select w.**

W=V-S=*{1,2,3,4,5}*-{1, 2,4, 3}={5}

**D[w] = min (D[w], D[u] + C[u, w])**

**D[5] = min ( D[5], D[3] +** C[3,5])

min ( 90, 50 + 10 )

min ( 90, 60 ) = 60

**Iteration 4 :**

**Select** u = 5, so that S = { 1,2,4, 3, 5 }

D[2] = 10
D[3] = 50
D[4] = 30
D[5] = 60

**Shortest Path from 1 to Other vertices**

| Vertex | Route | Minimum Cost |
|---|---|---|
| **1 —> 2** | 1 —> 2 | 50 |
| **1 —> 3** | 1 —> 4 —> 3 | 30 + 20 = 50 |
| **1 —> 4** | 1 —> 4 | 30 |
| **1 —> 5** | 1 —> 4 —> 3—> 5 | 30 + 20 + 10 = 60 |