UNIT – III

Greedy Method

Objective:

- To **Apply efficient algorithmic design paradigms**

Syllabus:

Greedy method: General method, knapsack problem, Job sequencing with deadlines, Minimum cost spanning trees.

Learning Outcomes:

At the end of the course, Students will be able to

- Apply Greedy method to Solve the knapsack problem & Job sequencing with deadlines
- Design Minimum cost spanning trees.

Learning Material

Introduction:

Greedy algorithms are typically used to solve an ***optimization problem.*** An Optimization problem is one in which, we are given a set of input values, which are required to be either maximized or minimized w. r. t. some constraints or conditions.

- In Greedy technique, the solution is constructed through a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached.
- In short, while making a choice there should be a Greed for the optimum solution.

Feasible Solution: For solving a particular problem there exists n inputs and we need to obtain a subset that satisfies some constraints. Then any subset that satisfies these constraints is called Feasible Solution.

Optimal Solution: From a set of feasible solution, particular solution that satisfies or nearly satisfies the objectives of the function such a solution is called Optimal Solution.

Objective Function: A feasible solution that either minimizes or maximizes a given objective function is called Objective function.

Characteristics of greedy algorithm:

- Used to solve optimization problem
- Most general, straightforward method to solve a problem.
- Easy to implement, and if exist, are efficient.
- Always makes the choice that looks best at the moment. That is, it makes a ***locally optimal choice in the hope that this choice will lead to a overall globally optimal solution.***
- Once any choice of input from C is rejected then it never considered again.
- Do not always yield an optimal solution; but for many problems they do.

General Method:

In Greedy method following activities are performed

1. First we select some solution from input domain
2. Then we check whether the solution is feasible or not
3. From the set of feasible solutions, particular solution that satisfies or nearly satisfies the objective of the function. Such solution is called optimal solution

4. As Greedy method works in stages. At each stage only one input is considered at each time. Based on this input it is decided whether particular input gives the optimal solution or not.

Algorithm for Greedy General Method is given below

```
Algorithm Greedy($a, n$)
// $a[1 : n]$ contains the $n$ inputs.
{
    $solution := \emptyset$; // Initialize the solution.
    for $i := 1$ to $n$ do
    {
        $x :=$ Select($a$);
        if Feasible($solution$, $x$) then
            $solution :=$ Union($solution, x$);
    }
    return $solution$;
}
```

Applications of Greedy Method:

There are various problems that can be solved using Greedy approach:

1. Knapsack Problem
2. Job Sequencing with Deadlines
3. Minimum Spanning Trees
4. Single source shortest path algorithm

Knapsack Problem:

The fractional knapsack problem is defined as:

- Given a list of n objects say$\{I_1, I_2, \ldots, I_n\}$ and a Knapsack (or bag).
- Capacity of Knapsack is M.
- Each object has I_i a weight W_i and a profit of P_i .

- If a fraction x_i (where x_i ε {0,…,1}) of an object I_i is placed into a knapsack then a profit of P_ix_i is earned

The **problem** (or Objective) is to fill a knapsack (up to its maximum capacity M) which maximizes the total profit earned.

Mathematically:

$$Maximize\ (the\ profit) \sum_{i=1}^{n} p_i x_i\ ;\ subjected\ to\ the\ constraints$$

$$\sum_{i=1}^{n} w_i x_i \leq \text{M and } x_i \in \{0,\ldots,1\}, 1 \leq \text{i} \leq \text{n}$$

Note that the value of x_i will be any value between 0 and 1 (inclusive). If any object I_i is completely placed into a knapsack then its value is 1 $(i.e. x_i = 1)$, if we do not pick (or select) that object to fill into a knapsack then its value is 0 $(i.e. x_i = 0)$. Otherwise if we take a fraction of any object then its value will be any value between 0 and 1.

To understand this problem, consider the following instance of a knapsack problem:

$Number\ of\ objects;\ n = 3$
Capacity of Knapsack; M=20
$(p_1, p_2, p_3) = (25,24,15)$
$(w_1, w_2, w_3) = (18,15,10)$

To solve this problem, Greedy method may apply any one of the following strategies:

- From the remaining objects, select the object with maximum profit that fit into the knapsack.
- From the remaining objects, select the object that has minimum weight and also fits into knapsack.
- From the remaining objects, select the object with maximum $^{p_i}/_{w_i}$ that fits into the knapsack.

Let us apply all above 3 approaches on the given knapsack instance:

Approach	(x_1, x_2, x_3)	$\sum_{i=1}^{3} w_i x_i$	$\sum_{i=1}^{3} p_i x_i$
1	$(1, \frac{2}{15}, 0)$	18+2+0=20	28.2
2	$< 0, \frac{2}{3}, 1 >$	0+10+10=20	31.0
3	$< 0, 1, \frac{1}{2} >$	0+15+5=20	31.5

Approach 1: (selection of object in decreasing order of profit):

In this approach, we select those object first which has maximum profit, then next maximum profit and so on. Thus we select 1st object (since its profit is 25, which is maximum among all profits) first to fill into a knapsack, now after filling this object ($w_1 = 18$) into knapsack remaining capacity is now 2 (i.e. 20-18=2). Next we select the 2nd object, but its weight w_2=15, so we take a fraction of this object (i.e. $x_2 = \frac{2}{15}$). Now knapsack is full (i.e. $\sum_{i=1}^{3} w_i x_i = 20$) so 3rd object is not selected. Hence we get total profit $\sum_{i=1}^{3} p_i x_i = 28$ units and the solution set $(x_1, x_2, x_3) = (1, \frac{2}{15}, 0)$

Approach 2: (Selection of object in increasing order of weights).
In this approach, we select those object first which has minimum weight, then next minimum weight and so on. Thus we select objects in the sequence 2nd then 3rd then 1st. In this approach we have total profit $\sum_{i=1}^{3} p_i x_i = 31.0$ units and the solution set $(x_1, x_2, x_3) = (0, \frac{2}{3}, 1)$.

Approach 3: (Selection of object in decreasing order of the ratio p_i/w_i).
In this approach, we select those object first which has maximum value of p_i/w_i, that is we select those object first which has maximum profit per unit weight .
Since $(p_1/w_1, p_2/w_2, p_3/w_3)$=(1.3, 1.6, 1.5). Thus we select 2nd object first , then 3rd object then 1st object. In this approach we have total profit $\sum_{i=1}^{3} p_i x_i = 31.5$ units and the solution set $(x_1, x_2, x_3) = (0, 1, \frac{1}{2},)$.

Thus from above all 3 approaches, it may be noticed that

- Greedy approaches **do not always yield an optimal solution**. In such cases the greedy method is frequently the basis of a heuristic approach.
- Approach3 **(**Selection of object in decreasing order of the ratio p_i/w_i) gives a optimal solution for knapsack problem.

Running time of Knapsack (fractional) problem:

Sorting of n items (or objects) in decreasing order of the ratio P_i/W_i takes O(n log n) time.

Since this is the lower bound for any comparison based sorting algorithm. Line 6 of ***Greedy Fractional-Knapsack*** takes O(n) time. Therefore, the total time including sort is O(n log n)

Example: 1: Find an optimal solution for the knapsack instance n=7 and M=15 ,

$(p_1, p_2 \dots, p_7) = (10, 5, 15, 7, 6, 18, 3)$
$(w_1, w_2, \dots, w_7) = (2, 3, 5, 7, 1, 4, 1)$

Solution:

Greedy algorithm gives a optimal solution for knapsack problem if you select the object in decreasing order of the ratio p_i/w_i . That is we select those object first which has maximum value of the ratio p_i/w_i ; $for\ all\ i = 1,..,7$. This ratio is also called profit per unit weight .

Since $\left(\frac{p_1}{w_1}, \frac{p_2}{w_2}, \dots, \frac{p_7}{w_7}\right) = (5,1.67,3,1,6,4.5,3)$. Thus we select 5th object first , then 1st object, then 3rd (or 7th) object, and so on.

Approach	$(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$	$\sum_{i=1}^{7} w_i x_i$	$\sum_{i=1}^{7} p_i x_i$
Selection of object in decreasing order of the ratio p_i/w_i	$(1, \frac{2}{3}, 1,0,1,1,1)$	1+2+4+5+1+2 =15	6+10+18+15+3+3.33 =55.33

Job Sequencing with Deadlines:

Consider that there are n jobs that are to be executed. At any time t=1, 2, 3…. Only exactly one job is to be executed. The profiles Pi are given. These profiles are gained by corresponding jobs. For obtaining feasible solution we should take care that the jobs get completed within their given deadlines.

We will follow following rules to obtain the feasible solution

- Each job takes one unit of time
- If job starts before or at its deadline, profit is obtained, otherwise no profit.

- Goal is to schedule jobs to maximize the total profit.
- Consider all possible schedules and compute the minimum total time in the system.
- The optimal solution is a feasible solution with maximum profit.

The algorithm for job sequencing is as given below

```
Algorithm JS(d, j, n)
// d[i] ≥ 1, 1 ≤ i ≤ n are the deadlines, n ≥ 1. The jobs
// 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 feasibility 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;
}
```

<u>Example:</u>

Solve the following job sequencing problem using Greedy algorithm.

Let N=4 profits associated with jobs (P1, P2,P3,P4)=(100,10,15,27) and Deadline associated with jobs(d1, d2, d3, d4) = (2, 1, 2, 1).

Solution:

Let Profits associated with jobs (P1, P2,P3,P4)=(100,10,15,27) and Deadline associated with jobs(d1, d2, d3, d4) = (2, 1, 2, 1).

Step 1: We will arrange the profits Pi in descending order. Then corresponding deadlines will appear

Profit	100	27	15	10
Job	P1	P4	P3	P2
Deadline	2	1	2	1

Step 2: Create an Array which stores the jobs. Initially array will be

1	2	3	4
0	0	0	0

Step 3: Add ith job in array at the index denoted by its deadline $d_{i.}$ First job is P1. The deadline for this job is 2. Hence insert P1 in array at 2nd index.

1	2	3	4
	P1		

Step 4: Next job is P4. Insert it at index 1.

1	2	3	4
P4	P1		

Step 5: Next job is P3 with deadline 2. But the array 2nd location is already occupied. Hence discard P3. Similarly job P2 will also be discarded.

Step 6: Thus final optimal sequence which we will obtain will be P1 – P4. The maximum profit will be 127.

MINIMUM COST SPANNING TREE (MCST):

Definition: **(Spanning tree):** Let G=(V,E) be an undirected connected graph. A ***subgraph*** T=(V,E') of G is a spanning tree of G if and only if T is a tree (i.e. no cycle exist in T) and contains **all the vertices** of G.

Definition: **(Minimum cost Spanning tree):**

Suppose G is a **weighted connected graph**. A ***weighted graph*** is one in which every edge of G is assigned some positive weight (or length). A graph G is having several spanning tree.

A ***minimum cost spanning tree*** (MCST) of a weighted connected graph G is that spanning tree whose sum of length (or weight) of all its edges is minimum, among all the possible spanning tree of G.

Application of spanning tree:

- *Spanning trees are widely used in designing an efficient network.*
- Another application of MCST is in the designing of efficient routing algorithm.

To find a MCST of a given graph G, one of the following algorithms is used:

1. Kruskal's algorithm

2. Prim's algorithm

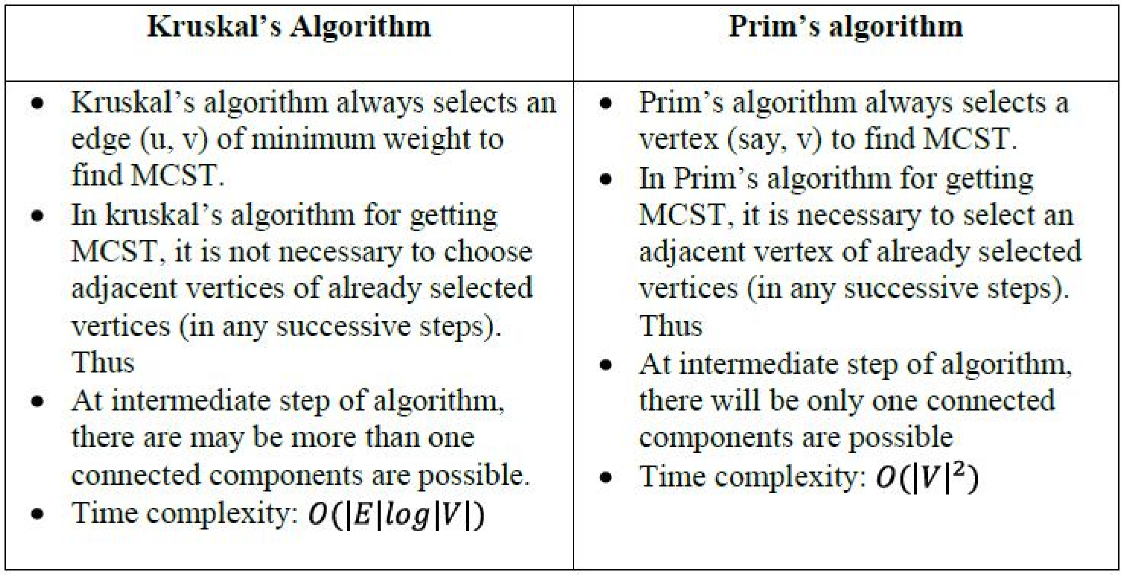
These two algorithms use Greedy approach. A greedy algorithm selects the edges one by- one in some given order. The next edge to include is chosen according to some optimization criteria. The simplest such criteria would be to choose an edge (u, v) that results in a minimum increase in the sum of the costs (or weights) of the edges so for included.

In General for constructing a MCST:

- We will build a set *A* of edges that is always a subset of some MCST.
- Initially, *A* has no edges (i.e. empty set).
- At each step, an edge (u, v) is determined such that *A* ∪ {*(u, v)*} is also a subset of a MCST. This edge *(u, v)* is called a ***safe edge.***
- At each step, we always add only ***safe edges*** to set A.
- **Termination:** when all *safe* edges are added to *A,* we stop. Now *A* contains a edges of spanning tree that is also an MCST.

***Difference* between *kruskal's* and *Prim's* algorithm:**

A main ***difference*** between ***kruskal's*** and ***Prim's*** algorithm to solve MCST problem is that the order in which the edges are selected.



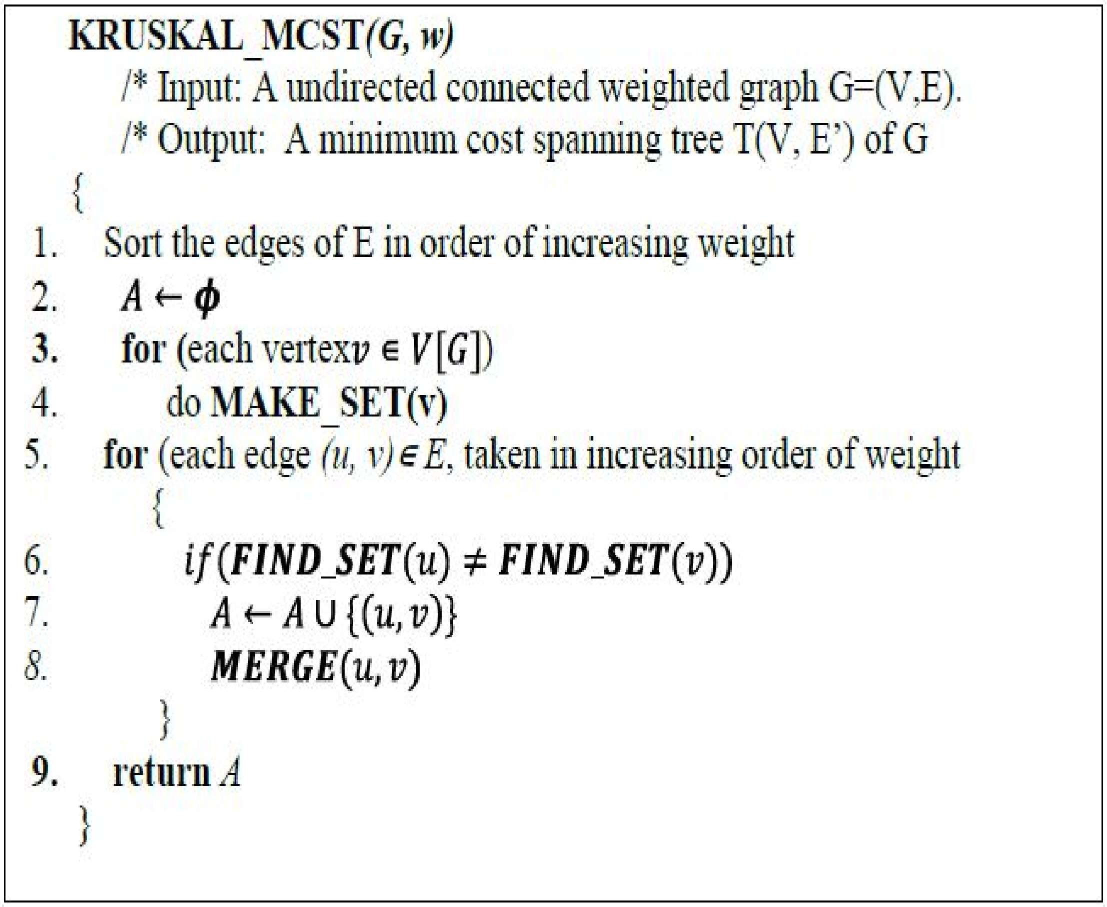
Kruskal's Algorithm	Prim's algorithm
• Kruskal's algorithm always selects an edge (u, v) of minimum weight to find MCST. • In kruskal's algorithm for getting MCST, it is not necessary to choose adjacent vertices of already selected vertices (in any successive steps). Thus • At intermediate step of algorithm, there are may be more than one connected components are possible. • Time complexity: $O(\|E\|log\|V\|)$	• Prim's algorithm always selects a vertex (say, v) to find MCST. • In Prim's algorithm for getting MCST, it is necessary to select an adjacent vertex of already selected vertices (in any successive steps). Thus • At intermediate step of algorithm, there will be only one connected components are possible • Time complexity: $O(\|V\|^2)$

Kruskal's Algorithm:

Let is a connected, weighted graph. Kruskal's algorithm finds a minimum-cost spanning tree (MCST) of a given graph G. It uses a ***greedy approach*** to find MCST, because at each step it adds an edge of least Possible weight to the set A. In this algorithm,

- First we examine the edges of G in order of increasing weight.
- Then we select an edge *(u, v)*ε *E* of minimum weight and checks whether its end points belongs to same component or different connected components.
- If *u* and *v* belongs to different connected components then we add it to set A, otherwise it is rejected because it create a cycle.
- The algorithm stops, when only one connected components remains (i.e. all the vertices of G have been reached).

The Kruskal's algorithm is given below:



```
KRUSKAL_MCST(G, w)
    /* Input: A undirected connected weighted graph G=(V,E).
    /* Output: A minimum cost spanning tree T(V, E') of G
{
1.   Sort the edges of E in order of increasing weight
2.   $A \leftarrow \phi$
3.   for (each vertex $v \in V[G]$)
4.       do MAKE_SET(v)
5.   for (each edge $(u, v) \in E$, taken in increasing order of weight
       {
6.       $if(FIND\_SET(u) \neq FIND\_SET(v))$
7.         $A \leftarrow A \cup \{(u, v)\}$
8.         $MERGE(u, v)$
       }
9.   return A
}
```

Kruskal's algorithm works as follows:

- First, we sorts the edges of E in order of increasing weight
- We build a set *A* of edges that contains the edges of the MCST. Initially A is empty.
- At line 3-4, the function **MAKE_SET(v),** make a new set {v} for all vertices of G. For a graph with n vertices, it makes n components of disjoint set such as {1},{2},... and so on.
- In line 5-8: An edge *(u, v)*ε *E,* of minimum weight is added to the set A, if and only if it joins two nodes which belongs to ***different*** components (to check this we use a **FIND_SET ()**function, which returns a same integer **Greedy Techniques**
  value, if u and v belongs to same components (In this case adding (u,v) to A creates a cycle), otherwise it returns a different integer value).
- If an edge added to A then the two components containing its end points are merged into a single component.
- Finally the algorithm stops, when there is just a single component.

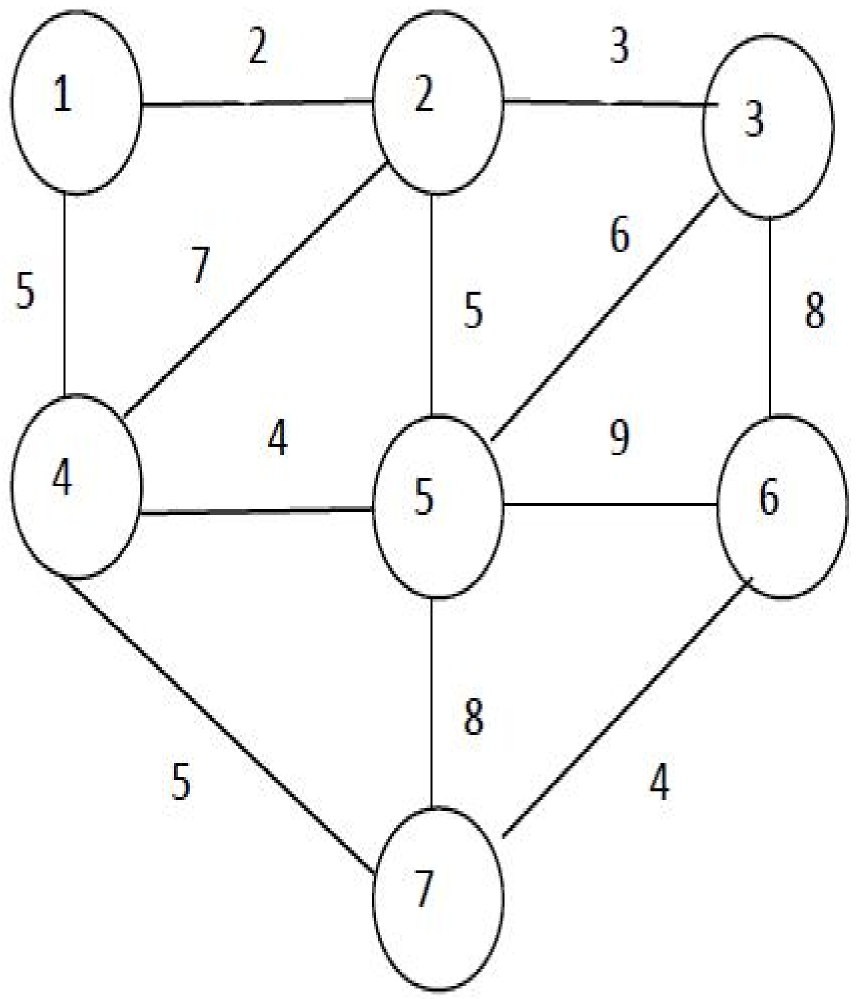
Analysis of Kruskal's algorithm:

Let $|V| = n, \textit{the number of vertices}$ and

$|E| = a, \textit{the number of Edges}$

1. Sorting of edges requires $O(|E|log|E|) = O(aloga)$ *time*.
2. Since, in any graph, minimum number of edges is $(n-1)$ and maximum number of edges (when graph is complete) is $n(n-1)/2$. Hence $n-1 \leq a \leq n(n-1)/2$. Thus $O(aloga) = O(alog\,(n(n-1)/2)) = O(alogn)$
3. Initializing n-disjoint set (in line 3-4) using **MAKE_SET** will requires O(n) time.
4. There are at most $2a$ **FIND_SET** operations (since there are a edges and each edge has 2 vertices) and $(n-1)$ **MERGE** operations. Thus we requires $O((2a+(n-1)logn)$ time.
5. At worst, O(a) time for the remaining operations.
6. For a connected graph, we know that $\boldsymbol{a \geq (n-1)}$. So the total time for Kruskal's algorithm is $O((2a+(n-1)logn) = O(alogn) = \boldsymbol{O(|E|\log|V|)}$

Example: Apply Kruskal’s algorithm on the following graph to find minimum-costspanning tree (MCST).

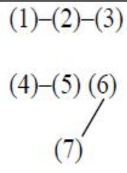
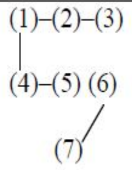
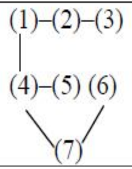


Solution:

First, we sorts the edges of G=(V,E) in order of increasing weights as:

Edges	(1,2)	(2,3)	(4,5)	(6,7)	(1,4)	(2,5)	(4,7)	(3,5)	(2,4)	(3,6)	(5,7)	(5,6)
weights	2	3	4	4	5	5	5	6	7	8	8	9

The kruskal's Algorithm proceeds as follows:

STEP	EDGE CONSIDERED	CONNECTED COMPONENTS	SPANNING FORESTS (A)
Initialization	___	{1}{2}{3}{4}{5}{6}{7} (using line 3-4)	(1) (2) (3) (4) (5) (6) (7)
1.	(1, 2)	{1, 2},{3},{4},{5},{6},{7}	(1) – (2) (3) (4) (5) (6) (7)
2.	(2, 3)	{1,2,3},{4},{5},{6},{7}	(1)–(2)–(3) (4) (5) (6) (7)
3.	(4, 5)	{1,2,3},{4,5},{6},{7}	(1)–(2)–(3) (6) (7) (4)–(5)
4.	(6, 7)	{1,2,3},{4,5},{6,7}	
5.	(1, 4)	{1,2,3,4,5},{6,7}	
6.	(2, 5)	Edge (2,5) is rejected, because its end point belongs to same connected component, so create a cycle.	
7.	(4, 7)	{1,2,3,4,5,6,7}	

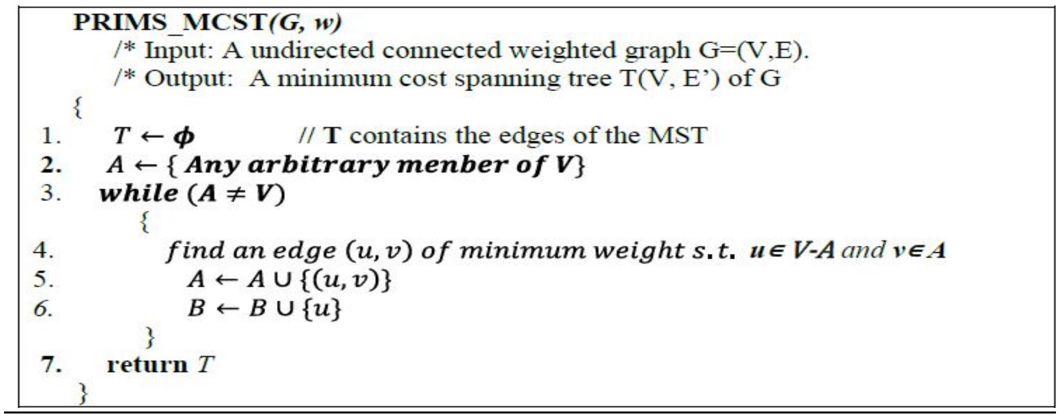
Total Cost of Spanning tree, T = 2+3+5+4+5+4=23

Prim’s Algorithm:

PRIM’s algorithm has the property that the edges in the set A (this set A contains the edges of the minimum spanning tree, when algorithm proceed step-by step) always form a single tree, i.e. at each step we have only one connected component.

- We begin with one starting vertex (say v) of a given graph G(V,E).
- Then, in each iteration, we choose a minimum weight edge *(u, v)* connecting a vertex *v* in the set A to the vertices in the set (V - A) . That is, we always find an edge *(u, v)* of minimum weight such that *v*ε *A* and *u*ε *V-A*. Then we modify the set A by adding *u* i.e. A ← A U {u}.
- This process is repeated until ≠V , i.e. until all the vertices are not in the set A.

Following pseudo-code is used to constructing a MCST, using PRIM’s algorithm:



```
PRIMS_MCST(G, w)
    /* Input: A undirected connected weighted graph G=(V,E).
    /* Output: A minimum cost spanning tree T(V, E’) of G
  {
1.     $T \leftarrow \phi$            // T contains the edges of the MST
2.     $A \leftarrow \{ Any\ arbitrary\ menber\ of\ V \}$
3.     while ($A \neq V$)
         {
4.         find an edge $(u, v)$ of minimum weight s.t. $u \in V-A$ and $v \in A$
5.           $A \leftarrow A \cup \{(u, v)\}$
6.           $B \leftarrow B \cup \{u\}$
         }
7.     return T
  }
```

PRIM's algorithm works as follows:

1. Initially the set A of nodes contains a single arbitrary node (i.e. starting vertex) and the set T of edges are empty.
2. At each step PRIM's algorithm looks for the shortest possible edge (u,v)such That ***u***ε V-A *and* ***v***ε ***A***
3. In this way the edges in T form at any instance a minimal spanning tree for the nodes in A. We repeat this process until A≠V.

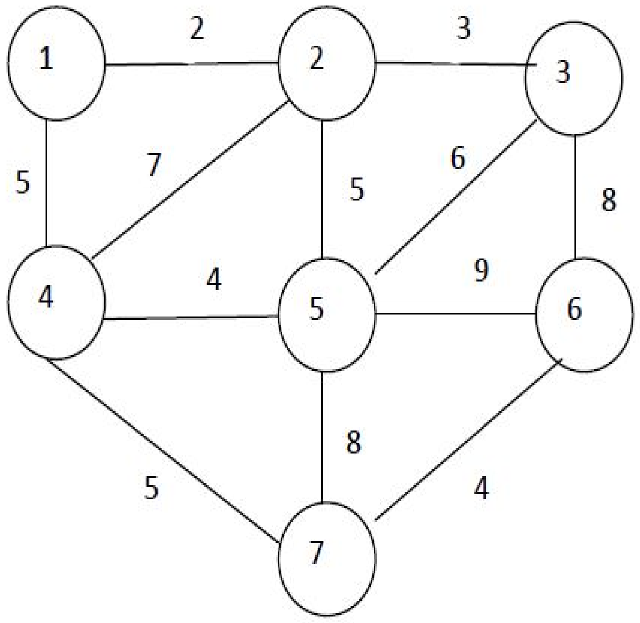
Time complexity of PRIM's algorithm:

Running time of PRIM's algorithm can be calculated as follow:

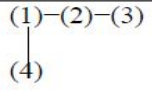
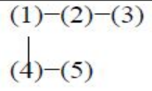
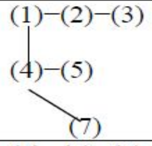
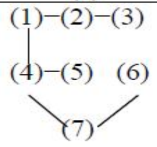
- While loop at line-3 is repeated | V | -1 = (n-1) times.
- For each iteration of while loop, the inside statements will require O(n)time.
- So the overall time complexity is $O(n^2)$.

Example: Apply PRIM's algorithm on the following graph to find minimum-cost spanning tree

(MCST).



Solution: In PRIM's, First we select an arbitrary member of V as a starting vertex (say 1), then the algorithm proceeds as follows:

STEP	**EDGE CONSIDERED (4, v)**	**CONNECTED COMPONENTS (Set A)**	**SPANNING FORESTS (set T)**
Initialization	—	{1}	(1)
1	(1,2)	{1,2}	(1)−(2)
2	(2,3)	{1,2,3}	(1)−(2)−(3)
3	(1,4)	{1,2,3,4}	
4.	(4,5)	{1,2,3,4,5}	
5	(4,7)	{1,2,3,4,5,7}	
6	(6,7)	{1,2,3,4,5,6,7}	

Total Cost of the minimum spanning tree = 2+3+5+4+5+4 = 23

UNIT – III

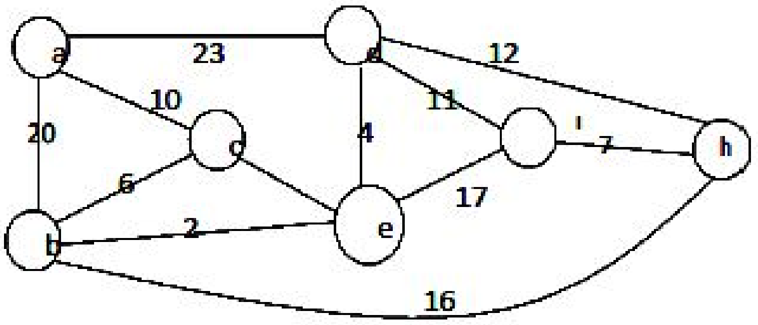
Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

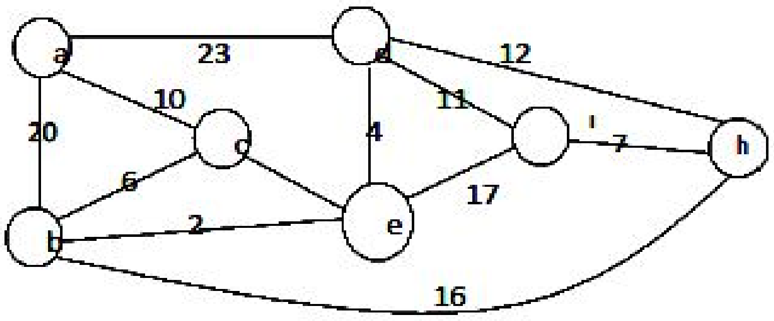
1. Which of the following standard algorithms is not a Greedy algorithm?
   A) Dijkstra's shortest path algorithm [ ]
   B) Prim's algorithm
   C) Kruskal algorithm
   D) Bellmen Ford Shortest path algorithm
2. Which of the following is/are the operations performed by kruskal's algorithm?
   i) sort the edges of G in increasing order by length [ ]
   ii) keep a subgraph S of G initially empty
   iii) builds a tree one vertex at a time
   A) i, and ii only B) ii and iii only C) i and iii only D) All i, ii and iii
3. Greedy job scheduling with deadlines algorithm time complexity is [ ]
   A) O(n) B) Ω(n log n) C) O (n^2 log n) D) O (n log n)
4. _____________turns out that one can find the shortest paths from a given source to all points in a graph in the same time. [ ]
   A) Kruskal's algorithm B) Prim's algorithm
   C) Dijkstra algorithm D) Bellman ford algorithm
5. How do you determine the cost of a spanning tree? [ ]
   A) By the sum of the costs of the edges of the tree
   B) By the sum of the costs of the edges and vertices of the tree
   C) By the sum of the costs of the vertices of the tree
   D) By the sum of the costs of the edges of the graph
6. The result of prim's algorithm is a total time bound of [ ]
   A) O(logn) B) O(n logn) C) O(n^2) D) O(n^3)

7. In Knapsack problem, the best strategy to get the optimal solution, where Pi, Wi is the Profit, Weight associated with each of the Xith object respectively is to [ ]
   A) Arrange the values Pi/Wi in ascending order
   B) Arrange the values Pi/Xi in ascending order
   C) Arrange the values Pi/Wi in descending order
   D) Arrange the values Pi/Xi in descending order
8. The time complexity of Greedy knapsack algorithm is [ ]
   A) O(logn) B) O(n logn) C) O(n^2) D) O(n)
9. Prim's algorithm is based on ____________ method. [ ]
   A) Divide and conquer method B) Dynamic programming
   C) Greedy method D) Branch and bound
10. Cost of minimum spanning tree using Prim's method is [ ]



A) 40 B) 39 C) 41 D) 47

11. Cost of minimum spanning tree using Kruskal's method is [ ]



A) 40 B) 39 C) 41 D) 47

12. Which of the following is not a feasible solution in the case of job sequence problem? [ ]

item :	1	2	3	4
profit :	100	10	15	27
deadline :	2	1	2	1

A) (1,4) B) (2,4) C) (4,3) D) (1,2)

13. Find the optimal solution for the following job sequence problem.

item :	1	2	3	4	5
profit :	20	15	10	5	1
deadline :	2	2	3	3	3

[ ]

A) (1,3,4) B) (1,2,4) C) (4,2,3) D) (1,5,2)

14. Find the optimal solution for the following job sequence problem.

item :	1	2	3	4	5	6	7	
profit :	3	5	20	18	1	6	30	
deadline :		1	3	4	3	2	1	2

A) (1,5,6,4) B) (2,3,1,7) C) (7,6,4,3) D) (1,2,3,4) [ ]

15. Which is the optimal solution to the fractional knapsack problem with capacity of Knapsack is 20 [ ]

item :	1	2	3	
profit :	25	24	15	
weight	:	18	15	10

A) 498 B) 499 C) 480 D) 485

16. Which is the optimal solution to the fractional knapsack problem with capacity of Knapsack is 10 [ ]

Item :	1	2	3	4	5	
Profit :	12	32	40	30	50	
Weight	:	4	8	2	6	1

A) 345 B) 354 C) 384 D) 350

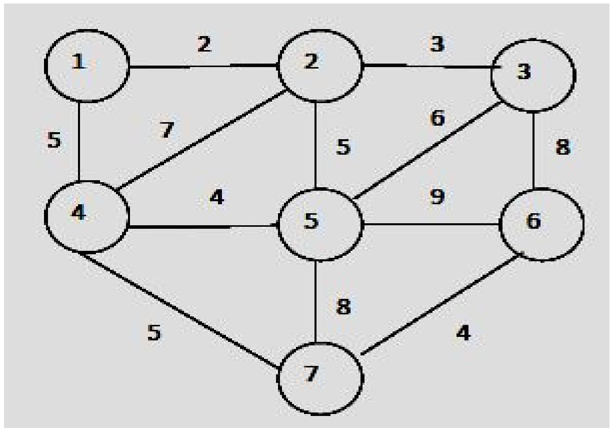
17. If the graph is represented as an adjacency matrix then the time complexityof Kruskal's algorithm is (V-set of vertices, E – set of edges) [ ]

A) $O(E \log E)$ B) $O(V \log V)$ C) $O(V^2)$ D) $O(\log E)$

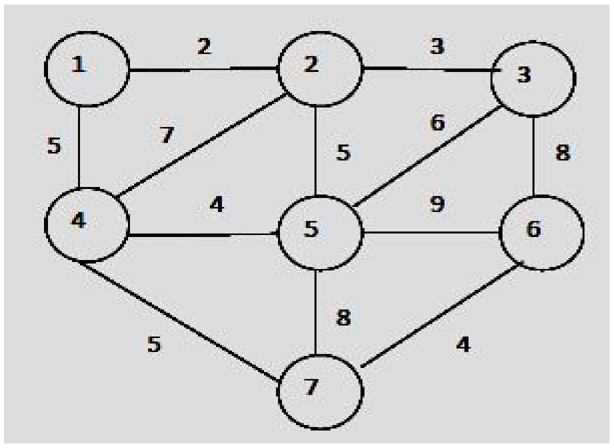
SECTION-B

SUBJECTIVE QUESTIONS

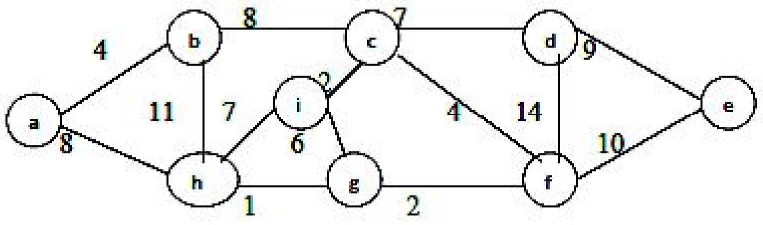
1. Differentiate between divide and conquer and Greedy method.
2. What is Greedy method? Write the control abstraction for it.
3. Explain the knapsack problem with appropriate example.
4. Explain the Job Sequencing with deadlines problem with appropriate example.
5. Differentiate between prim's and Kruskal's algorithms.
6. Explain Prim's algorithm with suitable example.
7. Illustrate Kruskal's method with an example.
8. Write the Greedy knapsack algorithm and analyze its time complexity.
9. Write Prim's algorithm and analyze its time complexity.
10. Write Kruskal's algorithm and analyze its time complexity.
11. Apply Kruskal's algorithm on the following graph to find minimum-cost spanning Tree(MCST).



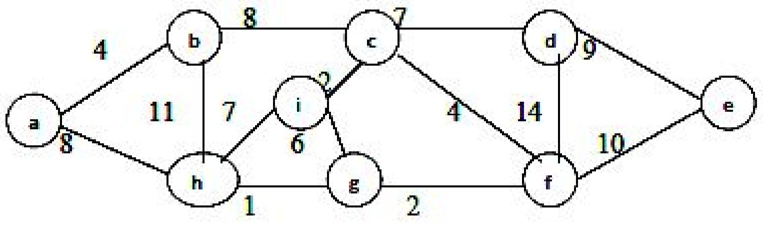
12. Apply Prim's algorithm on the following graph to find minimum-cost spanning tree (MCST).



13. Are the Minimum spanning tree of any graph is unique? Apply Prim's algorithm to find a minimum cost spanning tree for the following. (*a* is a starting vertex).



14. Apply Kruskal's algorithm to find a minimum cost spanning tree for the following. (*a* is a starting vertex).



15. Find the optimal solution to the knapsack instance n=5, M=10 (P1, P2, …. , P5) = (12, 32, 40, 30, 50) and (W1, W2, …., W5) = (4, 8, 2, 6, 1).

16. Let S={a, b, c, d, e, f, g} be a collection of objects with Profit-Weight values as follows: a:(12,4), b:(10,6), c:(8,5), d:(11,7), e:(14,3), f:(7,1) and g:(9,6). What is the optimal solution to the *fractional* knapsack problem for S, assuming we have a knapsack that can hold objects with total weight of 18? What is the *complexity* of this method?

SECTION-C

QUESTIONS AT THE LEVEL OF GATE

1. Let G be a complete undirected graph on 4 vertices, having 6 edges with weights being 1, 2, 3, 4, 5, and 6. The maximum possible weight that a minimum weight spanning tree of G can have is ________

   [GATE 2016]

2. G = (V,E) is an undirected simple graph in which each edge has a distinct weight, and e is a particular edge of G. Which of the following statements about the minimum spanning trees (MSTs) of G is/are TRUE? [GATE 2016]

   [ ]

   I. If e is the lightest edge of some cycle in G, then every MST of G includes e

   II. If e is the heaviest edge of some cycle in G, then every MST of G excludes e

   A) I only B) II only C) both I and II D) neither I nor II

3. Consider a complete undirected graph with vertex set {0, 1, 2, 3, 4}. Entry Wij in the matrix W below is the weight of the edge {i, j}.

   [ ]

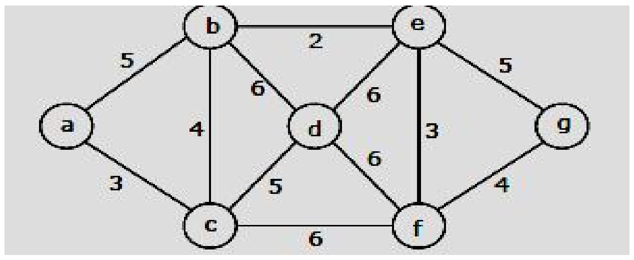
$$W = \begin{pmatrix} 0 & 1 & 8 & 1 & 4 \\ 1 & 0 & 12 & 4 & 9 \\ 8 & 12 & 0 & 7 & 3 \\ 1 & 4 & 7 & 0 & 2 \\ 4 & 9 & 3 & 2 & 0 \end{pmatrix}$$

   [GATE 2010]

   What is the minimum possible weight of a spanning tree T in this graph such that vertex 0 is a leaf node in the tree T?

   A) 7 B) 8 C) 9 D) 10

4. Consider the following graph: [GATE 2009]



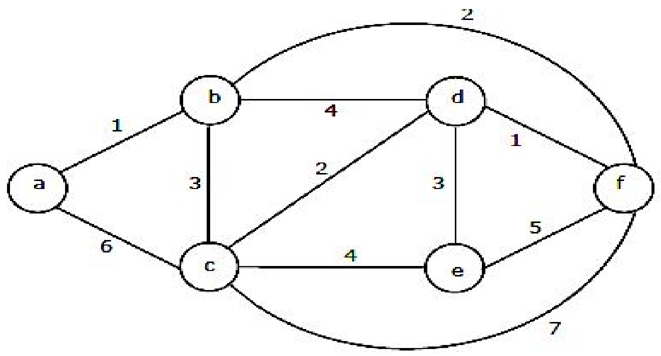
Which one of the following is NOT the sequence of edges added to the minimum

spanning tree using Kruskal's algorithm? [ ]

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

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

5. Consider the following graph:



Which one of the following cannot be the sequence of edges added, in that order, to a minimum spanning tree using Kruskal's algorithm?

[ ]

A) <a, b>, <d, f>, <b, f>, <d, c>, <d, e>

B) <a, b>, <d, f>, <d, c>, <b, f>, <d, e>

C) <d, f>, <a, b>, <d, c>, <b, f>, <d, e>

D) <d, f>, <a, b>, <b, f>, <d, e>, <d, c>