UNIT-IV

GREEDY METHOD

Greedy Method: General method, applications - Job sequencing with dead lines, Knapsack problem, Minimum cost spanning trees, Single source shortest path problem.

Greedy Method / Greedy Algorithm:-

A Greedy Algorithm is a method that solves an optimization problem by making a series of choices, each of which looks best at the current step, without considering the overall problem or the impact of that choice on future steps.

Key characteristics:-

- The algorithm makes locally optimal choices at each step, hoping these will lead to a globally optimal solution.
- It does not revisit or revise decisions once made.
- Greedy algorithms are often fast and easy to implement, but they do not always guarantee an optimal solution for every problem

Advantages:-

- Simpler logic and easier implementation
- Generally faster due to not exploring all possibilities.

⊛ Disadvantages :-

⊛ May not always give the optimal solution.

⊛ only works for problems meeting specific properties.

Applications :-

1. Job sequencing with dead lines
2. Knapsack problem
3. Minimum cost spanning trees
4. Single source shortest path problem.

① Job Sequencing with deadlines (Greedy Method) :-

Job Sequencing with deadlines using the greedy method finds the optimal subset of jobs to maximize profit by selecting jobs in descending order of profit and scheduling each as late as possible before its deadline.

problem overview :-

Given several jobs, each with a profit and a deadline, the task is to schedule jobs on a single machine (one per time slot) so that jobs are finished before their deadlines, earning the associated profit.

steps :-

① Sort all jobs in descending order of their profit so jobs with the highest profit are Considered first

(2) For each job, try to schedule it in the latest available slot before its deadline.
(3) If the slot is free, assign the job; if not check earlier slots. If no slots are free, skip the job.
(4) Continue until all jobs are considered.

⇒ Complete the jobs in deadline so that profit is maximum.

Example problem:-

Tasks	T_1	T_2	T_3	T_4	T_5
profits	20	15	10	5	1
Deadline	2	2	1 (x)	3	3

(Assume each task will take one hour to complete). Max. time → 3hours

T_2 — 1st (15) T_1 — 2nd (20) T_4 — 3rd

T_2, T_1, T_4

$20+15+5 = 40 = \text{profit}$

Example 2:-

Tasks/ Jobs	T_1	T_2	T_3	T_4	T_5	T_6	T_7
profits	35	30	25	20	15	12	5
deadlines	3	4	4	2	3	1	2

max time = 4 hours

T_4 — 1st hr (20) T_3 — 2nd hr (25) T_1 — 3rd hr (35) T_2 — 4th hr (30)

profit = 20 + 25 + 35 + 30 = 110

② Knapsack problem :-

The Knapsack problem with the greedy method refers to solving the fractional Knapsack problem by always picking items based on the highest value-to-weight ratio, ensuring the maximum profit within the given capacity.

Fractional vs. 0-1 Knapsack :-

* The greedy method provides an optimal solution only for the fractional Knapsack where items can be broken into parts and fractions can be taken.
* For the 0-1 Knapsack (items cannot be split), greedy does not guarantee optimality.

Greedy Algorithm Steps :-

* Calculate value-to-weight ratio for all items.
* Sort items in descending order based on this ratio.
* Pick items with the highest ratio first;
  → If the current item fits fully, include it.
  → If not, include as much of it as possible (fractional part) to fill the remaining capacity.

Example :-

objects	1	2	3	4	5	6	7
profits(p)	10	5	15	7	6	18	3
weights(w)	2	3	5	7	1	4	1
(p/w) ratio	5	1.3	3	1	6	4.5	3

Capacity of Knapsack - 15 units

O_1	O_2	O_3	O_4	O_5	O_6	O_7
(1)	(2/3)	(1)	(0)	(1)	(1)	(1)

$$\text{profit} = 1\times10 + \frac{2}{3}\times5 + 1\times15 + 0\times7 + 1\times6 + 1\times18 + 1\times3$$

$$= 10 + 3.33 + 15 + 6 + 18 + 3$$

$$= 55.33$$

$15-1=14$
$14-2=12$
$12-4=8$
$8-5=3$
$3-1=2$
$2-2=0$

③ Minimum * Cost * Spanning * Trees:-

A minimum Cost spanning tree (also Knocon as a minimum spanning tree or MST) is a Subset of edges in a Connected, weighted, undirected graph that Connects all the vertices together without any cycles, and with the minimum possible total edge weight or cost. The "cost" here refers to the Sum of the weights of the edges included in the spanning tree, where the edge weights can represent distance, money, time or any quantity to be minimized.

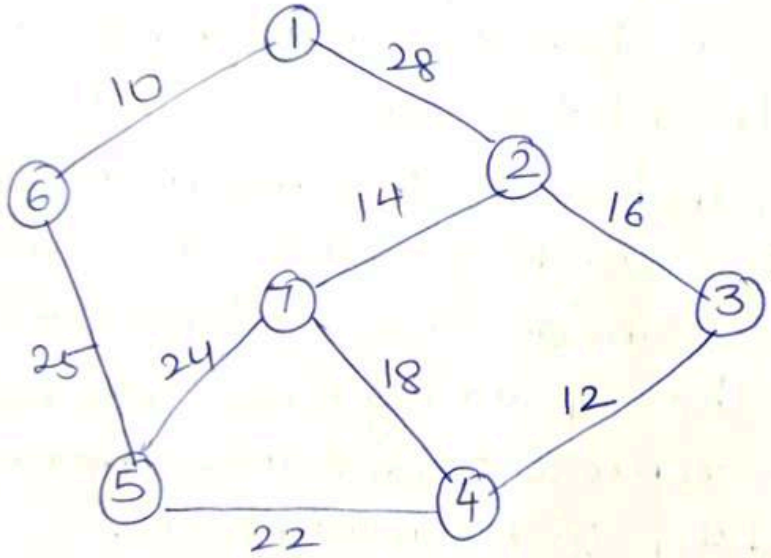
Key points about Minimum Cost Spanning Tree.

- It spans all the vertices in the graph.
- It has no cycles (is a tree)
- It minimizes the total edge weight sum among all possible spanning trees.
- There can be multiple MSTs if more than one tree has the same minimum total cost.
- Common use cases include minimizing costs for network design such as laying cables, pipelines or roads.

Algorithms to find MST

* **Prim's Algorithm:** A greedy algorithm that grows the MST by starting from an arbitrary vertex and repeatedly adding the smallest edge connecting the growing MST to a new vertex.
* **Kruskal's Algorithm:** A greedy algorithm that sorts all edges by weight and adds them one by one to the MST, skipping those that form cycles, untill all vertices are connected.

Example :-

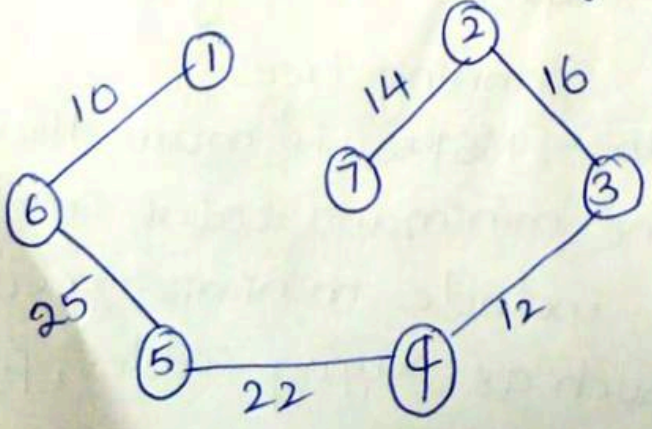


Vertices = 7

Prim's Algorithm

* First select minimum cost edge
* Next minimum Costedge, but they should be Connected to previously selected edges.

Vertices = 7
Edges = 6



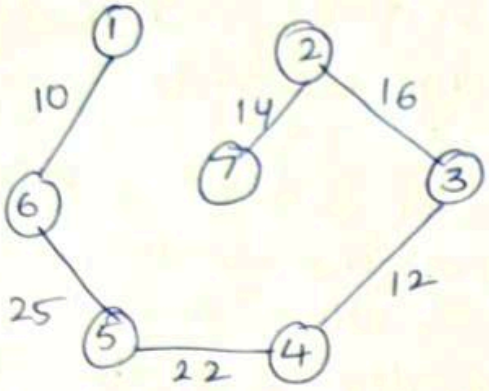
$10 + 25 + 22 + 12 + 16 + 14$
$= 99$

Minimum Cost = 99

Kruskal's Algorithm

(*) Always select minimum cost edge

(*) No need of connection



$10+25+22+12+16+14 = 99$

Minimum Cost = 99.

Example for Minimum Cost Spanning tree :-

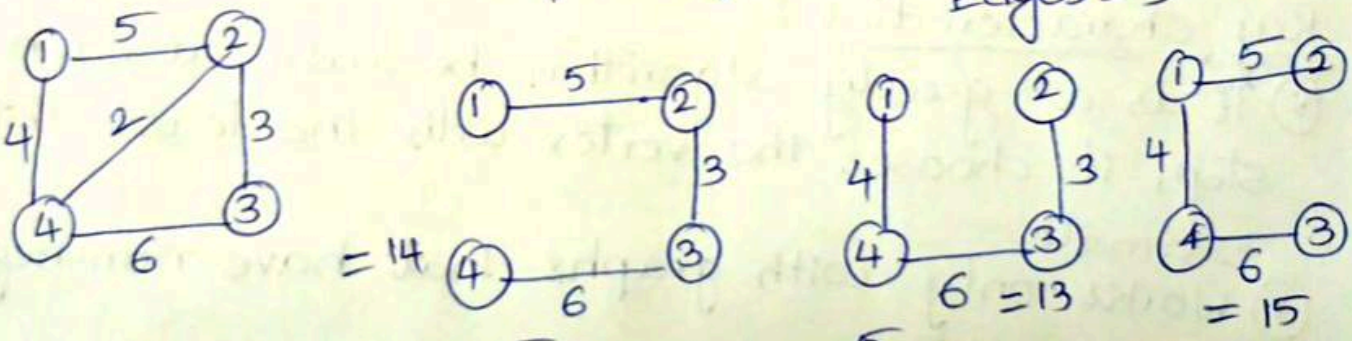
Spanning tree : Subset of a graph which has same number of vertices from original graph and (n-1) edges.

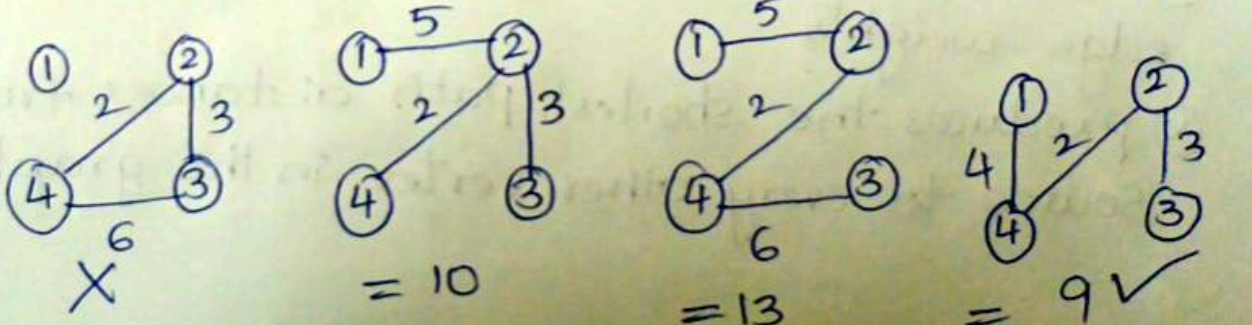
n → no. of vertices.

→ Not possible in non-Connected graph

Spanning tree Vertices (n) = 4

Edges = 3





When Comparing, Minimum Cost spanning tree = 9

Single Source shortest path problem:-

The single source shortest path greedy method is most commonly known as Dijkstra's algorithm. It finds the shortest paths from a given source vertex to all other vertices in a weighted graph with non-negative edge weights.

Working principle:-

- The algorithm maintains a set of vertices whose shortest distance from the source is already known (called the "visited" set).
- Initially, only the source vertex is in this set with a distance of zero.
- At each step, it selects the vertex with the smallest tentative distance that is not yet visited (greedy choice)
- It then updates the distances of the neighboring vertices based on the selected vertex.
- This process repeats until the shortest distances to all vertices have been determined.

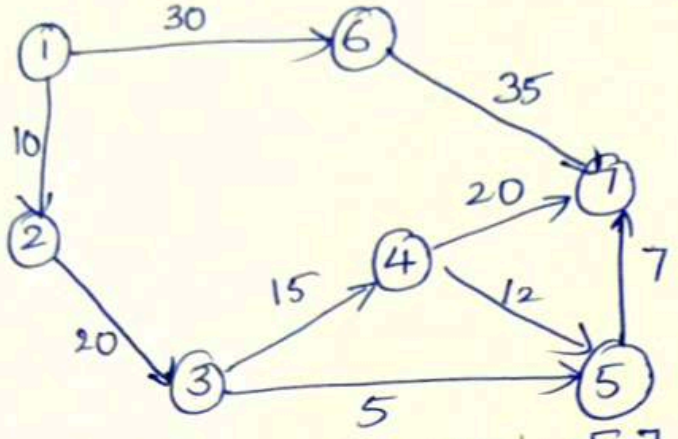
Key characteristics :-

- It is a greedy algorithm because, at each step, it chooses the vertex with the least distance estimate.
- Works only with graphs that have non-negative edge weights.
- produces the shortest path distances from the source to every other vertex in the graph.

(*) Time Complexity can be $O(n^2)$ with simple data structures or improved to $O((n+e)\log n)$ using priority queues (heaps). where n is the number of vertices and e the number of edges.

Example :-

lets assume (1) as source vertex



Selected vertex	visited set	d[2]	d[3]	d[4]	d[5]	d[6]	d[7]
		10	∞	∞	∞	30	∞
1	{1}	(10)	10+20	∞	∞	30	∞
2	{1,2}	(10)	(30)	30+15 (45)	30+5 (35)	30 (30)	∞
3	{1,2,3}	(10)	(30)	45	35	(30)	30+35 =65
6	{1,2,3,6}	(10)	(30)	45	(35)	(30)	35+7=42
5	{1,2,3,6,5}	(10)	(30)	45	(35)	(30)	(42)
7	{1,2,3,6,5,7}	(10)	(30)	45	(35)	(30)	(42)
4	{1,2,3,6,5,7,4}	(10)	(30)	(45)	(35)	(30)	(42)

Minimum Cost path = {1, 2, 3, 6, 5, 7, 4}.