Unit-3: Greedy Algorithms

🔹 What is a Greedy Algorithm?

- Greedy algorithms are used for **optimization problems** (maximize profit, minimize cost, etc.).
- At each step, they make the **locally best choice** (the one that looks best at that moment).
- They are **short-sighted** → do not worry about future consequences.
- Work only for problems with the **Greedy Choice Property** (local choice leads to global solution).

🔹 Characteristics

1. A **candidate set** of possible choices.
2. Once a candidate is chosen/rejected, it is not reconsidered.
3. Uses four functions:
    - **Solution function** → Checks if solution is complete.
    - **Feasible function** → Checks if partial solution is valid.
    - **Selection function** → Selects the best candidate.
    - **Objective function** → Measures the value of a solution.

🔹 Problems Solved with Greedy

1. Make Change Problem

- Goal: Make a given amount using minimum coins.
- Always pick the **largest coin** less than or equal to remaining amount.
- Example: To make 28 with {10, 5, 2, 1} → pick 10 + 10 + 5 + 2 + 1 = 5 coins.

---

2. Minimum Spanning Tree (MST)

A spanning tree connects all vertices with minimum total edge weight.

(a) Kruskal's Algorithm

- Sort edges by weight.
- Add edge if it does not form a cycle.
- Stop when n-1 edges are included.
- Time Complexity: O(E log E).

(b) Prim's Algorithm

- Start from any vertex.
- Keep adding the **smallest edge** that connects the tree to a new vertex.
- Continue until all vertices are included.
- Time Complexity: $O(V^2)$ with adjacency matrix.

---

3. Dijkstra's Algorithm (Single Source Shortest Path)

- Finds shortest path from one source to all other nodes (non-negative weights).
- Steps:
    1. Start with source.
    2. Update distances to all neighbors.
    3. Pick unvisited vertex with smallest distance.
    4. Repeat until all vertices visited.
- Time Complexity: $O(V^2)$ (or $O((V+E) \log V)$ with priority queue).

---

4. Fractional Knapsack Problem

- Given items with value & weight, capacity W.
- Can take fractions of items.
- Choose items based on **value/weight ratio** (highest first).
- Time Complexity: O(n log n) (due to sorting).

---

5. Activity Selection Problem

- Select maximum number of activities that don't overlap.
- Sort activities by **finishing time**.
- Always pick earliest finishing activity compatible with previous one.

---

6. Job Scheduling with Deadlines

- Jobs with deadlines and profits.
- Goal: Maximize profit while meeting deadlines.
- Sort jobs by profit (descending).
- Schedule each job as late as possible before deadline.

---

7. Huffman Coding (Data Compression)

- Builds optimal **prefix codes**.
- Frequently used characters → shorter codes.
- Steps:
    1. Build a priority queue with frequencies.
    2. Extract two smallest nodes, combine.
    3. Repeat until one node remains (root of Huffman Tree).
- Time Complexity: O(n log n).