GREEDY ALGORITHMS

- A greedy algorithm is a problem-solving technique that makes a sequence of choices by selecting the best possible option available at each step.
- The key characteristic of a greedy algorithm is that it makes these choices in a locally optimal way, with the hope that these local choices will lead to a globally optimal solution.

Working

To understand how a greedy algorithm works, let's break it down into simple steps:

- **Make a choice** – At each step, pick the best available option based on a specific criterion.
- **Proceed to the next step** – Move forward and repeat the process until the problem is solved.
- **Check the final outcome** – The algorithm arrives at a solution that is either optimal or close to optimal.

Problems solved with greedy algorithm

- Coin change problem
- Fractional knapsack
- Activity Selection
- 0/1 Knapsack
- Dijkstra's shortest path algorithm
- Huffman Encoding
- MST (Prim's & Kruskal's Algorithm)

REAL LIFE EXAMPLES

Coin change problem

- The Coin Change problem is a classic example of a greedy algorithm in action. The task is: given a set of coin denominations, determine the minimum number of coins required to make a specific amount.

Greedy Approach:

- To solve this, the greedy algorithm picks the largest coin denomination first and reduces the target amount by that coin's value.
- This process repeats until the entire amount is made up using the least number of coins possible.

Example

- Imagine we have coin denominations of {1, 5, 10, 25}, and we need to make 30.
- Start with the largest coin (25). Select one 25-Rs coin.
- Remaining amount: 30 - 25 = 5
- Next, choose the largest coin less than or equal to the remaining amount (5). Select one 5-Rs coin.
- Remaining amount: 5 - 5 = 0
- In this case, the greedy algorithm successfully uses just two coins (one 25-Rs coin and one 5-Rs coin) to make 30

Python Code

```
def coin_change_min(coins, amount):
    # Initialize dp array with "infinity"
    dp = [float('inf')] * (amount + 1)
    dp[0] = 0  # Base case: 0 coins to make amount 0
    for coin in coins:
```

```
        for x in range(coin, amount + 1):
            dp[x] = min(dp[x], dp[x - coin] + 1)
    return dp[amount] if dp[amount] != float('inf') else -1
# Example
coins = [1, 2, 5]
amount = 11
print("Minimum coins:", coin_change_min(coins, amount))
# Output: 3 (11 = 5 + 5 + 1)
```

Activity selection problem

- The activity selection problem is an example of a greedy algorithm where the maximum number of non-overlapping activities are selected from the given activity set. A person can complete one activity at a time. The activities are given in the form of their starting and completion times.
- We are given n activities, each with:
    - a start time
    - a finish time
- We need to select the maximum number of activities that can be performed by a single person, assuming one activity at a time.

Key Idea of the Greedy Technique

- The greedy strategy here is:

    Always pick the activity that finishes earliest (among the remaining ones).

Why?

- If we choose the activity that finishes earliest, it leaves the most room for other activities to fit afterward.

- This ensures the maximum number of non-overlapping activities can be chosen.

Algorithm Steps

- Sort all activities by their finish times (earliest finishing activity comes first).
- Select the first activity (since it finishes earliest).
- For each next activity:
  - If its start time ≥ finish time of the last selected activity, then select it.
  - Otherwise, skip it.
- Continue until all activities are checked.

Example

Input:

arr[] = {{5,9},{1,2},{3,4},{0,6},{5,7},{8,9}}

Sorted Activities: {1,2}, {3,4},{0,6},{5,7},{5,9},{8,9}

Select First activity {1,2}

Continue selecting

- Next, **A2 (3–4)** → valid (starts after 2) ✅
- Next, **A4 (5–7)** → valid (starts after 4) ✅
- Next, **A5 (8–9)** → valid (starts after 7) ✅
- Skip **A6 (5–9)** and **A3 (0–6)** because they overlap ❌

Final Selected Set:(A1: 1–2), (A2: 3–4), (A4: 5–7), (A5: 8–9)

Time Complexity

- Overall: O(n log n)