Coin Change (LeetCode #322)

Topic: Dynamic Programming — Minimum Coins to Make Amount

🔷 Problem Summary

Hume ek array `coins[]` diya hua hota hai jinme different coin denominations hoti hain.

Aur ek value `amount`.

Goal ye hai ki **minimum number of coins** use karke `amount` ko form karein.

Agar amount form hi nahi ho sakta → return **-1**.

✔️ Key Points

- Infinite supply of each coin.
- Return **fewest coins** needed.
- Agar possible nahi hai → return **1**.
- Classic **DP (Top-Down + Memoization)** problem.

🔥 Idea Behind the Solution

Har amount `n` ke liye hum yeh soch rahe hain:

> "Agar main coin arr[i] use karu, toh baaki amount n - arr[i] ko kitne coins me bana sakta hoon?"

Isliye hum har coin try karte hain → recursion + memoization.

🧠 Recursive Relation

```
f(n) = min( 1 + f(n - coin[i]) )  for all coins[i] <= n
```

- `1` because hum ek coin use kar rahe hain.
- `f(n - coin[i])` baaki amount ka answer batayega.
- Sab coins try karke **minimum** value choose karni hai.

🛑 Base Case

```
f(0) = 0     // amount 0 means 0 coins required
```

📦 Memoization

DP array:

```
dp[n] = minimum coins needed to make amount n
```

Agar dp[n] already computed hai → directly return.

💡 Why INT_MAX?

`INT_MAX` ko “impossible state” ke tarah treat kar rahe hain.

Agar koi coin amount ko represent nahi kar sakta → INT_MAX return hota.

🧾 Final Code (Top-Down DP)

```
class Solution {
public:
    int f(vector<int>& arr, int n, vector<int>& dp) {
        if (n == 0) return 0;
        if (dp[n] != -1) return dp[n];

        int curr = INT_MAX;
```

```
        for (int i = 0; i < arr.size(); i++) {
            if (n >= arr[i]) {
                int res = f(arr, n - arr[i], dp);
                if (res != INT_MAX)
                    curr = min(curr, res + 1);
            }
        }
        return dp[n] = curr;
    }

    int coinChange(vector<int>& coins, int amount) {
        vector<int> dp(amount + 1, -1);
        int ans = f(coins, amount, dp);
        return ans == INT_MAX ? -1 : ans;
    }
};
```

🧩 Examples

Example 1

```
coins = [1,2,5], amount = 11
Output = 3
Explanation: 5 + 5 + 1
```

Example 2

```
coins = [2], amount = 3
Output = -1
```

Example 3

```
coins = [1], amount = 0
Output = 0
```

🎯 Complexity

- **Time:** O(amount × number of coins)
- **Space:** O(amount) for DP + recursion stack
