# 🚀 DSA Patterns Course 2025 — Day 5 (Two Pointer Pattern)

**A1-Level Notes — First Principles + Full Hinglish + Interview-Perfect**

---

## ✨ Ye Notes Kyu Special Hain?

Ye notes **First Principles Thinking** se banaye gaye hain — matlab:

- Har problem ko bilkul **scratch se** samjhaya gaya hai.
- Har line **kyu likhi ja rahi hai**, uska reason diya hua hai.
- Intuition is level ki hai ki **beginner bhi padhe to samajh jaye**, aur
- Interviewer ko explain karne layak clarity bhi ho.

## 🥇 Problem 1: Strobogrammatic Number (LC 256 – Easy)

### 🧠 First-Principle Intuition

#### ❓ Problem kya bol raha hai?

Agar tum ek number ko **180° rotate** karo aur woh phir bhi **same** dikhe to woh strobogrammatic number hota hai.

#### ❓ Kaunse digits rotate hoke valid bante hain?

- 0 → 0 (same hi dikhta hai)
- 1 → 1
- 8 → 8
- 6 → 9 (flip hota hai)
- 9 → 6 (flip hota hai)

#### ❌ Invalid digits: 2,3,4,5,7

Rotate hone par koi valid form nahi banate.

#### ❓ Logic simple kya hai?

Ye **palindrome check** jaisa hi hai, bas simple same-same compare nahi karte.
Left wale digit ko **rotate** karke right wale digit se match karna hota hai.

#### ❓ Two Pointer kyu?

Kyuki hum beginning aur end se characters check kar rahe hain.

---

## 🧪 Dry Run (Simple Example)

Input → "619"

- 6 rotate → 9 = matches right
- 1 rotate → 1 = matches → Output: **true**

---

## ✅ C++ Code (Comments)

```
class Solution {
public:
    bool isStrobogrammatic(string num) {
        // Ye map batata hai ki kaunsa digit rotate hoke kya banega
        unordered_map<char,char> rot = {
            {'0','0'}, {'1','1'}, {'8','8'},
            {'6','9'}, {'9','6'}
        };

        int l = 0, r = num.size() - 1;
        while (l <= r) {
            char L = num[l], R = num[r];

            // Agar left wala digit rotate hi nahi ho sakta → invalid number
            if (!rot.count(L)) return false;

            // Rotate hone ke baad left ka digit right se match hona chahiye
            if (rot[L] != R) return false;

            l++;
            r--;
        }
        return true;
    }
};
```

---

## 🔍 Complexity (Explanation)

### Time → O(n)

- Two pointers ek hi baar puri string traverse karte hain.
- Har step me constant kaam hota hai.

### Space → O(1)

- Map fixed size ka hai (5 entries).
- Extra memory nahi lagti.

---

# 🥈 Problem 2: Append Characters to Make Subsequence (LC 2486 – Medium)

## 🧠 First-Principle Intuition

### ❓ Problem kya keh raha?

`t` ko `s` ka **subsequence** banana hai.
Agar kuch part match nahi hota, woh end me **append** karna padega.

### ❓ Matching kaise hoga?

- `i` → `s` ke andar move karega
- `j` → `t` ke andar
- Jab match mile → dono pointers aage
- Jab match na mile → sirf `i` aage

`t` ka jo part match na ho → wahi append karoge.

## 🧪 Dry Run

s = "coaching"
t = "coding"

`t` me se sirf "co" match hota hai.
Remaining → "ding" → length = **4**.

## ✅ C++ Code (Comments)

```
class Solution {
public:
    int appendCharacters(string s, string t) {
        int i = 0, j = 0;
        int n = s.size(), m = t.size();

        while (i < n && j < m) {
            // Match mil gaya → t ka next char dhundho
            if (s[i] == t[j]) j++;

            // s to hamesha aage badhega
            i++;
        }

        // t ka kitna part match nahi hua → wahi append hoga
        return m - j;
    }
};
```

---

## 🔍 Complexity

**Time → O(n + m)**

- `s` poora ek baar scan hota hai → O(n)
- `t` ka pointer sirf match hone par aage badhta hai → max O(m)

**Space → O(1)**

- Bas do pointers use ho rahe hain.

---

# 🥉 Problem 3: Lowest Common Ancestor III (LC 1650 – Medium)

## 🧠 First-Principles Intuition

### ❓ Yeh problem special kaise hai?

Har node ke paas **parent pointer** diya hua hai.
Matlab koi bhi node se root tak ja sakte ho:

```
node → parent → parent → ... → root
```

Yeh ek **linked list** jaisa lagta hai.

### ❓ P aur Q ka LCA kaise milega?

- P se root tak ek path → linked list
- Q se root tak ek path → linked list

Dono paths ki **length alag ho sakti hai**.
Isko solve karne ka best logic:

### 🔥 Two-Pointer Reset Trick (Linked List Intersection Technique)

- A = P se start
- B = Q se start
- Jab A root se upar (NULL) jaye → usko Q par le aao
- Jab B NULL jaye → usko P par le aao

Dono **equal distance** travel karenge → same node par milenge → wahi LCA.

---

## 🧪 Dry Run

P = 7
Q = 4

Paths:
P → 7 → 2 → 5 → 3
Q → 4 → 2 → 5 → 3

Meet point = **2** = LCA.

---

## ✅ C++ Code

```cpp
class Solution {
public:
    Node* lowestCommonAncestor(Node* p, Node* q) {
        Node* a = p;
        Node* b = q;

        // Jab tak dono same node par nahi aa jate
        while (a != b) {
            // Agar a upar tak pahuch gaya → Q se start kara do
            a = (a == nullptr) ? q : a->parent;

            // Agar b upar tak pahuch gaya → P se start kara do
            b = (b == nullptr) ? p : b->parent;
        }

        // Jaha dono mil gaye → wahi LCA
        return a;
    }
};
```

---

## 🔍 Complexity

### Time → O(h_p + h_q)

- P se root tak height = h_p
- Q se root tak height = h_q
- Dono pointers milkar at max h_p + h_q distance cover karenge
- Balanced tree: h ≈ logN
- Skew tree: worst-case O(N)

### Space → O(1)

- Sirf do pointers use ho rahe hain.

---

# 📌 Day 5 Final Summary

| Problem | Pattern | Time | Space | Main Idea |
|---|---|---|---|---|
| Strobogrammatic | Two Pointer + Map | O(n) | O(1) | Rotate(L) = R check |
| Append Characters | Greedy Two Pointer | O(n+m) | O(1) | t ka jo part match na ho → append |
| LCA III | Two Pointer Reset | O(h) | O(1) | Paths = linked lists, meet = LCA |

---

## 🎯 Final Takeaway

Two Pointer sirf array/string ke liye nahi hota.
Tree + parent pointer problems me bhi **super clean** solution deta hai.
LCA III two-pointer trick → **industry-level smart approach** hai.

---