All patterns of subsequences

Theory

1. All Subsequences (Power Set style)

- **Definition:** Subsequence formed by including or excluding each element.
- **Count:** 2^n.
- **Example:** Input = `[1,2,3]` → Output = `[[], [1], [2], [3], [1,2], [1,3], [2,3], [1,2,3]]`.
- **Pattern:** Brute force generation using recursion/backtracking or bitmask.
- **Related to:** Power set problems.

2. Non-empty Subsequences

- **Definition:** All subsequences except the empty one.
- **Count:** 2^n - 1.
- **Example:** Input = `[1,2]` → Output = `[[1], [2], [1,2]]`.
- **Pattern:** Same as all subsequences, just ignore empty set.
- **Related to:** Product/Sum problems where empty subsequence is invalid.

3. Fixed Length Subsequences (k-subsequences)

- **Definition:** Subsequences having exactly length kk.
- **Count:** {n}{k}.
- **Example:** Input = `[1,2,3]`, k=2 → Output = `[[1,2], [1,3], [2,3]]`.
- **Pattern:** Choose k elements while preserving order.
- **Related to:** Combinations.

4. Distinct Subsequences (Handling Duplicates)

- **Definition:** Unique subsequences when the input has duplicates.
- **Count:** No fixed formula (depends on frequency of duplicates).
- **Example:** Input = `"aaa"` → Raw subsequences = `["", "a", "a", "a", "aa", "aa", "aa", "aaa"]` → Distinct = `["", "a", "aa", "aaa"]`.
- **Pattern:** Use hashing, set, or DP to remove duplicates.
- **Related to:** LeetCode 90 (Subsets II), Distinct Subsequences problems.

5. Lexicographic Subsequences

- **Definition:** Subsequences arranged in dictionary order.
- **Count:** Same as all subsequences (2^n), just ordered differently.
- **Example:** Input = `"abc"` → Output = `["a","ab","abc","ac","b","bc","c"]`.
- **Pattern:** Generate all subsequences then sort lexicographically.
- **Related to:** Lexicographic ordering problems (e.g., smallest subsequence).

6. Constraint-based Subsequences

- **Definition:** Subsequences must satisfy specific conditions.

- **Count:** Depends on condition (no general formula).
- **Example:** Input = `[1,2,3]`, target sum=3 → Valid subsequences = `[[1,2],[3]]`.
- **Pattern:** Backtracking/DP with condition check.
- **Related to:** Subset sum, knapsack, palindrome subsequences.

7. Optimization Subsequences (Max/Min)

- **Definition:** Subsequences chosen to maximize/minimize some metric.
- **Count:** Not about count, but best subsequence according to metric.
- **Example:** Input = `[1,-2,3,4]` → Max sum subsequence = `[1,3,4]` with sum=8.
- **Pattern:** DP/Greedy approaches.
- **Related to:** LIS (Longest Increasing Subsequence), Maximum sum subsequence.

8. Counting Subsequences

- **Definition:** Count subsequences matching a property without listing them.
- **Count:** Depends on problem (calculated via DP).
- **Example:** Input string = `"rabbbit"`, target = `"rabbit"` → Output = `3` subsequences match.
- **Pattern:** Dynamic Programming counting states.
- **Related to:** Distinct Subsequences, Counting DP.

9. Special Patterns of Subsequences

- **Definition:** Special ways of representing/generating subsequences.
- **Count:** Still 2^n, just different representation.
- **Example:**
  - Input = `[1,2,3]` → Bitmask `101` → Subsequence `[1,3]`.
- **Pattern:**
  - Continuous subsequences (actually subarrays/substrings).
  - Bitmask representation (binary → include/exclude).
  - Recursive backtracking (classic include/exclude recursion).
  - Iterative building (expand subsequences step by step).
- **Related to:** Subarrays, Bitmask DP, Iterative construction problems.