RECURSION ASSIGNMENT

Advanced Recursive Problem Solving

Instructions

1. Solve all problems using recursion only (no loops unless necessary for printing).
2. Provide time complexity analysis for each problem.
3. Draw the recursion tree for at least TWO problems.
4. Implement in C++ or Python.

Problems

1. Combinations (nCk)

Generate all combinations of size k from a given set. **vector<vector<int>> Combinations(vector<int> & V, int k)**

Input: set = {1,2,3,4,5}, k = 2 **Output:** {1,2},{1,3},{1,4},{1,5},{2,3},{2,4},{2,5},{3,4},{3,5},{4,5} **Explanation:** We fix the first element and combine it with every element to its right: Fix 1 → pair with 2,3,4,5 → {1,2},{1,3},{1,4},{1,5} Fix 2 → pair with 3,4,5 → {2,3},{2,4},{2,5} Fix 3 → pair with 4,5 → {3,4},{3,5} Fix 4 → pair with 5 → {4,5} Total = 4+3+2+1 = 10 = C(5,2)	**Input:** set = {1,2}, k = 3 **Output:** No output (k > n is impossible) **Explanation:** You cannot choose 3 items from a set of only 2 items. C(2,3) = 0 → the recursion produces no output. Always validate k <= n before calling the recursive function.
Input: set = {A,B,C,D}, k = 2 **Output:** {A,B},{A,C},{A,D},{B,C},{B,D},{C,D}	**Input:** set = {1,2,3}, k = 3 **Output:** {1,2,3}

2. Binary Strings

vector<string> Generate01Strings(int n)

Generate all binary strings of length n.

Input: n = 2 **Output:** 00, 01, 10, 11 **Explanation:** At each position we branch into two choices: place '0' or place '1'. Level 0 (pos 0): branch into 0__ and 1__ Level 1 (pos 1): each branch into _0 and _1 Result: 00, 01, 10, 11 → 2^2 = 4 strings	**Input:** n = 3 **Output:** 000, 001, 010, 011, 100, 101, 110, 111 **Explanation:** Each string is 3 characters long, with 2^3 = 8 strings total. The recursion goes 3 levels deep, branching 2x at each level. Leaves (depth=3): 000, 001, 010, 011, 100, 101, 110, 111 Notice: left subtree always starts with 0, right subtree starts with 1.

3. Balanced Parentheses

vector<string> GenerateAllParenthesis(int n)

Generate all valid parentheses with n pairs.

Input: n = 2 **Output:** (()), ()() **Explanation:** Rules: open < n → can add '(' | close < open → can add ')' Start: open=0, close=0, string='' Add '(' → open=1, close=0 → string='(' Add '(' → open=2, close=0 → string='((' Add ')' → open=2, close=1 → string='(()' Add ')' → open=2, close=2 → RESULT: '(())' Add ')' → open=1, close=1 → string='()' Add '(' → open=2, close=1 → string='()(' Add ')' → open=2, close=2 → RESULT: '()()' Total: 2 strings = C_2 = 2	**Input:** n = 3 **Output:** ((())), (()()), (())(), ()(()), ()()() **Explanation:** The 5 valid strings for n=3 are: 1. ((())) — all opens then all closes 2. (()()) — nested and sequential 3. (())() — one pair nested, one sequential 4. ()(()) — one sequential, one pair nested 5. ()()() — all sequential Count = 5 = C_3 (Catalan number) ✓

Hint: Consider n = 3 i.e. 6 character long bit string where 0's are equal to 1's and from left to right the number of 0's>=1's. Now replace all the 0's with '(' and 1's as ')'.

4. Subsequences

Generate all subsequences of a string.

vector<string> GenerateAllSubsequences(string s)

Input: s = "ab" **Output:** "", "a", "b", "ab"	**Input:** s = "abc" **Output:** "", "a", "b", "c", "ab", "ac", "bc", "abc"

5. N-Queens

vector<vector<vector<int>>> NQueens(int n)

Place N queens so that none attack each other. There are how many solutions for 8x8, report all solutions.

Input: n = 4 **Output:** [2,4,1,3], [3,1,4,2] — 2 solutions **Explanation:** Board layout (Q = queen, . = empty): Row 1: . Q . . (queen at column 2) Row 2: . . . Q (queen at column 4) Row 3: Q . . . (queen at column 1) Row 4: . . Q . (queen at column 3)	**Input:** n = 2v **Output:** No Solution **Explanation:** On a 2×2 board, any queen placement blocks all remaining squares. Place Q at (1,1): row 2, col 1, and both diagonals are blocked. Place Q at (1,2): row 2, col 2, and both diagonals are blocked.

6. Determinant (Cofactor Expansion)

Compute determinant recursively.

int Determinant(vector<vector<int>> M)

$$\begin{vmatrix} 2 & 1 & 3 \\ 4 & 0 & 5 \\ 6 & 7 & 8 \end{vmatrix}$$

We expand along the **first row** (cofactor expansion).

$$= 2\begin{vmatrix} 0 & 5 \\ 7 & 8 \end{vmatrix} - 1\begin{vmatrix} 4 & 5 \\ 6 & 8 \end{vmatrix} + 3\begin{vmatrix} 4 & 0 \\ 6 & 7 \end{vmatrix}$$

Input: [[4,7], [2,6]] **Output:** 10 **Explanation:** det([[4,7],[2,6]]) = 4 × det([[6]]) − 7 × det([[2]]) = 4 × 6 − 7 × 2 = 24 − 14 = 10 Base case: det of a 1×1 matrix = that single element. Sign pattern for first row: +, −, +, −, ...	**Input:** [[1,2,3],[4,5,6],[7,8,0]] **Output:** 27 **Explanation:** Expand along row 1: +1 × det([[5,6],[8,0]]) = +1 × (5×0 − 6×8) = +1 × (−48) = −48 −2 × det([[4,6],[7,0]]) = −2 × (4×0 − 6×7) = −2 × (−42) = +84 +3 × det([[4,5],[7,8]]) = +3 × (4×8 − 5×7) = +3 × (−3) = −9 Total = −48 + 84 − 9 = 27 ✓

7. Tower of Hanoi

Move n disks from source to destination. **void TOH(string src, string helper, string dst, int n)**

Input: n=2 **Output:** 3 moves (A→B, A→C, B→C) **Explanation:** Goal: move 2 disks from A to C. Step 1: Move disk 1 (small) from A → B (free up disk 2) Step 2: Move disk 2 (large) from A → C (place large disk) Step 3: Move disk 1 (small) from B → C (stack small on large)	**Input:** n = 3 **Output:** 7 moves **Explanation:** 1. Move disk 1: A → C 2. Move disk 2: A → B 3. Move disk 1: C → B 4. Move disk 3: A → C 5. Move disk 1: B → A 6. Move disk 2: B → C 7. Move disk 1: A → C

8. Grid Paths

void generatePaths(int row, int col, int m, int n, string path)

- Print all paths from top-left to bottom-right of an mxn matrix with all zeroes.
- What if we have restricted positions and now we have nxn matrix.

Input: 2×2 grid **Output:** RD, DR (2 paths) **Explanation:** Grid (S = start, E = end): S . . E Path 1: move Right then Down → RD Path 2: move Down then Right → DR From any cell: move Right if col < max, move Down if row < max. Each path has length (rows−1) + (cols−1) = 1 + 1 = 2 moves. Total paths = C(2,1) = 2 ✓	**Input:** 3×3 grid **Output:** RRDD, RDRD, RDDR, DRRD, DRDR, DDRR (6 paths) **Explanation:** Each path needs exactly 2 Rights and 2 Downs (4 moves total). Total = C(4,2) = 6 paths: RRDD, RDRD, RDDR, DRRD, DRDR, DDRR Think of it as: in which 2 of the 4 positions do we place 'R'? The remaining 2 positions automatically get 'D'.

9. Subset Sum

Find all subsets whose sum equals target.

Input: set={3,1,4,2,6}, target=7 **Output:** {1,6}, {3,4}, {1,2,4} **Explanation:** We explore include/exclude for each element: Include 3: Include 4 → sum = 7 ✓ → {3,4} Include 1, include 2 → 3+1+2 = 6 ≠ 7 Include 1: Include 6 → sum = 7 ✓ → {1,6} Include 2, include 4 → 1+2+4 = 7 ✓ → {1,2,4} Final answers: {3,4}, {1,6}, {1,2,4}	**Input:** set={1,2,5}, target=11 **Output:** No solution **Explanation:** set = {1,2,5}, target = 11 Maximum possible sum = 1 + 2 + 5 = 8 < 11 No subset can ever reach 11 → output: No solution Optimization tip: prune the recursion early if the sum of all remaining elements cannot possibly reach the remaining target.

10. String Permutations

void permute(string s, int index)

Generate all permutations of a string.

Input: s = "ab" **Output:** "ab", "ba" (2 permutations)	**Input:** s = "cat" **Output:** "cat", "cta", "act", "atc", "tca", "tac" (6 permutations)