Dynamic Programming Patterns (with C++ Snippets)

1D DP

```
vector<int> dp(n+1, 0);
dp[0] = base_case;
for (int i = 1; i <= n; i++) {
    dp[i] = function_of(dp[i-1], dp[i-2], ...);
}
```

Practice Problems:

- Leetcode 70 - Climbing Stairs

- Leetcode 198 - House Robber

- Leetcode 746 - Min Cost Climbing Stairs

- GFG - Friends Pairing Problem

- Leetcode 91 - Decode Ways

2D Grid DP

```
vector<vector<int>> dp(m, vector<int>(n, 0));
dp[0][0] = grid[0][0];  // or 1
for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
        if (i > 0) dp[i][j] = min(dp[i][j], dp[i-1][j] + grid[i][j]);
        if (j > 0) dp[i][j] = min(dp[i][j], dp[i][j-1] + grid[i][j]);
    }
}
```

Practice Problems:

- Leetcode 64 - Minimum Path Sum

- Leetcode 62 - Unique Paths

- Leetcode 63 - Unique Paths II

- GFG - Gold Mine Problem

- Leetcode 221 - Maximal Square

Dynamic Programming Patterns (with C++ Snippets)

1D DP

```
vector<int> dp(n+1, 0);
dp[0] = base_case;
for (int i = 1; i <= n; i++) {
    dp[i] = function_of(dp[i-1], dp[i-2], ...);
}
```

Practice Problems:

- Leetcode 70 - Climbing Stairs

- Leetcode 198 - House Robber

- Leetcode 746 - Min Cost Climbing Stairs

- GFG - Friends Pairing Problem

- Leetcode 91 - Decode Ways

2D Grid DP

```
vector<vector<int>> dp(m, vector<int>(n, 0));
dp[0][0] = grid[0][0];  // or 1
for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
        if (i > 0) dp[i][j] = min(dp[i][j], dp[i-1][j] + grid[i][j]);
        if (j > 0) dp[i][j] = min(dp[i][j], dp[i][j-1] + grid[i][j]);
    }
}
```

Practice Problems:

- Leetcode 64 - Minimum Path Sum

- Leetcode 62 - Unique Paths

- Leetcode 63 - Unique Paths II

- GFG - Gold Mine Problem

- Leetcode 221 - Maximal Square

3D Knapsack DP

```
vector<vector<int>> dp(n+1, vector<int>(W+1, 0));
for (int i = 1; i <= n; i++) {
    for (int w = 0; w <= W; w++) {
        dp[i][w] = dp[i-1][w];
        if (w >= wt[i-1]) {
            dp[i][w] = max(dp[i][w], dp[i-1][w - wt[i-1]] + val[i-1]);
        }
    }
}
```

Practice Problems:

- Leetcode 416 - Partition Equal Subset Sum

- Leetcode 494 - Target Sum

- GFG - 0/1 Knapsack

- Leetcode 1049 - Last Stone Weight II

- GFG - Subset Sum Problem

4D Partition DP

```
vector<vector<int>> dp(k, vector<int>(n, 1e9));
for (int j = 0; j < n; j++) dp[0][j] = getCost(0, j);

for (int i = 1; i < k; i++) {
    for (int j = i; j < n; j++) {
        for (int p = i - 1; p < j; p++) {
            int currCost = getCost(p + 1, j);
            dp[i][j] = min(dp[i][j], max(dp[i - 1][p], currCost));
        }
    }
}
```

Practice Problems:

- Leetcode 410 - Split Array Largest Sum

- Leetcode 1335 - Minimum Difficulty of a Job Schedule

- GFG - Painter's Partition Problem

- Codeforces 1041D - Glider

- AtCoder - ABC 208F

5D Subsequence DP

```
vector<vector<int>> dp(m+1, vector<int>(n+1, 0));
for (int i = 1; i <= m; i++) {
    for (int j = 1; j <= n; j++) {
        if (A[i-1] == B[j-1])
            dp[i][j] = 1 + dp[i-1][j-1];
        else
            dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
    }
}
```

Practice Problems:

- Leetcode 1143 - Longest Common Subsequence

- Leetcode 516 - Longest Palindromic Subsequence

- Leetcode 392 - Is Subsequence

- Leetcode 115 - Distinct Subsequences

- GFG - Longest Repeating Subsequence

6D Bitmask DP

```
int dp[1<<n];
dp[0] = 0;
for (int mask = 1; mask < (1<<n); mask++) {
    for (int i = 0; i < n; i++) {
        if (mask & (1<<i)) {
            dp[mask] = min(dp[mask], dp[mask ^ (1<<i)] + cost[i]);
        }
    }
}
```

Practice Problems:

- Leetcode 847 - Shortest Path Visiting All Nodes

- Leetcode 698 - Partition to K Equal Sum Subsets

- Leetcode 473 - Matchsticks to Square

- Leetcode 1349 - Maximum Students Taking Exam

- GFG - Travelling Salesman Problem