1. Activity Selection Problem

Recurrence Relation: Not typically formulated with a recurrence relation as it is primarily solved with a greedy approach.

2. Assembly Line Scheduling

Recurrence Relation: `F[i][j] = e[i] + a[i][j] + min(F[i][j-1], F[1-i][j-1] + t[1-i][j])` for `j > 1`, and `F[i][1] = e[i] + a[i][1]`.

- **Table**: 2D array with dimensions `2 x n` (2 lines, `n` stations).
- **Filling Order**: Row-major, sequentially for each station `j` from `1` to `n`, considering both assembly lines.

3. Coin Changing

Recurrence Relation: `C[n] = min{ C[n-d_i] } + 1` for all coin denominations `d_i` that are less than or equal to `n`, with base case `C[0] = 0`.

- **Table**: 1D array where `C[i]` stores the minimum number of coins needed for amount `i`.
- **Filling Order**: Column-major, from left to right (`1` to `N`), representing increasing amounts.

4. Edit Distance

Recurrence Relation: `D[i][j] = min(D[i-1][j-1] + cost(substitute), D[i-1][j] + 1, D[i][j-1] + 1)`, with base cases `D[i][0] = i` and `D[0][j] = j`.

- **Table**: 2D array with dimensions `(m+1) x (n+1)` for strings of length `m` and `n`.
- **Filling Order**: Row-major, starting from `(0,0)` to `(m,n)`, each cell representing the edit distance between substrings.

5. Longest Common Subsequence (LCS)

Recurrence Relation: `LCS[i][j] = LCS[i-1][j-1] + 1` if characters match, otherwise `LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])`, with base cases `LCS[i][0] = 0` and `LCS[0][j] = 0`.

- **Table**: 2D array with dimensions `(m+1) x (n+1)` for strings of length `m` and `n`.
- **Filling Order**: Row-major, filling each cell based on conditions, from top-left to bottom-right.

6. Fibonacci Sequence

Recurrence Relation: `F[n] = F[n-1] + F[n-2]` with base cases `F[0] = 0`, `F[1] = 1`.

- **Table**: 1D array where `F[i]` stores the `i-th` Fibonacci number.
- **Filling Order**: Column-major, filling from `F[2]` upwards to `F[n]`.

7. Chessboard Traversal (Maximal Profit)

Recurrence Relation: `P[i][j] = max(P[i-1][j], P[i][j-1]) + C[i][j]` where `C[i][j]` is the cost at cell `(i,j)`, with base case `P[0][0] = C[0][0]`.

- **Table**: 2D array with dimensions `n x n` for a chessboard of size `n`.

- **Filling Order**: Row-major, starting from `P[0][0]` and filling horizontally in each row before moving to the next row, considering only right and down moves.

In each of these problems, the structure and dimensions of the table used to store values, as well as the order in which the table is filled, are closely tied to the nature of the problem and the specific requirements of the dynamic programming approach used to solve it.