# **Assignment 2 Part 2: Commit Owners**

**1. Data Structures we used and argumentation**

**HashMap**: Stores employee ID to full name mappings for O(1) lookup time.

**HashSet**: Contains all valid employee IDs for O(1) membership testing during substring validation.

**Dynamic Programming Arrays**: Four arrays track optimal solutions - maxCommits stores maximum commits from position i, totalWays counts all decompositions, optimalWays counts optimal decompositions, bestChoice stores the chosen employee ID.

**Precomputed Match Lists**: 2D list structure storing all valid employee ID matches starting at each position, computed once to avoid redundant work.

### **2. Pseudo-code for the Algorithm**

### **main**

findOptimalDecomposition(weld):  
 n = weld.length  
 matchesAtPosition = precomputeMatches(weld)  
 initialize DP arrays of size n+1  
   
 // base case  
 maxCommits[n] = 0  
 totalWays[n] = optimalWays[n] = 1  
   
 for i = n-1 down to 0:  
 maxCommits[i] = -1  
 totalWays[i] = optimalWays[i] = 0  
   
 for each empId in matchesAtPosition[i]:  
 nextPos = i + empId.length  
 if maxCommits[nextPos] != -1:  
 currentCommits = 1 + maxCommits[nextPos]  
 totalWays[i] += totalWays[nextPos]  
   
 if maxCommits[i] < currentCommits:  
 update optimal solution  
 else if maxCommits[i] == currentCommits:  
 optimalWays[i] += optimalWays[nextPos]  
   
 reconstruct optimal path

### **Match Precomputation**

precomputeMatches(weld):  
 n = weld.length  
 matchesAtPosition = array of empty lists[n+1]  
   
 for i = 0 to n-1:  
 maxLen = min(maxIdLength, n-i)  
 for len = 1 to maxLen:  
 candidate = weld.substring(i, i+len)  
 if employeeIds.contains(candidate):  
 matchesAtPosition[i].add(candidate)  
   
 return matchesAtPosition

### **3. Correctness of Algorithm**

**Dynamic Programming Correctness**: The recurrence relation captures the problem structure. For position i, we consider all valid employee IDs starting at i and combine with optimal solutions from the remaining substring.

**Optimal Substructure**: If an optimal solution uses employee ID at position i, then the remaining substring must also be solved optimally.

**Match Precomputation Correctness**: By checking all substrings of length 1 to maxIdLength at each position and validating against the employee ID set, we guarantee finding all valid matches.

### **4. Overall Time Complexity**

**Preprocessing**: O(M) where M is the total length of all employee IDs for reading and storing in HashSet.

**Match Precomputation**: O(N × L) where N is the weld string length and L is the maximum employee ID length.

**Main Algorithm**: O(N + Z) where Z is the total number of matches found across all positions.

**Space Complexity**: O(N + M + Z)

**Overall**: O(N × L + M), however it can be counted as O(N) as the L and M are constants.

**5. Build Version - 1.0**