Dhaka International University

Department of Computer Science and Engineering

B.Sc. in Computer Science and Engineering

Course Code: CSE-301 Course Title: Algorithm Design & Analysis

Assignment

Name: MD. SIDDIQUL ISLAM LABIB **Role**: 11 **Department:** B. Sc. (CSE)

Semester: 7 th **Reg No:** 119415

1. Define dynamic programming and explain its significance in problem-solving algorithms.

Dynamic Programming (DP) is a problem-solving technique used to efficiently solve complex problems by breaking them down into simpler overlapping subproblems, solving each subproblem only once, and storing their solutions for future use.

Significance in Problem-Solving Algorithms:

- **Efficiency**: DP reduces the computational complexity by storing the results of subproblems (known as memorization), which prevents the need to recompute them. This leads to significant time savings, especially for problems with a lot of overlapping subproblems.
- **Optimal Solutions**: DP ensures that the solution to the problem is optimal by considering all possible subproblem solutions and combining them to form the best possible answer.
- **Wide Applicability**: DP can be applied to a variety of problems in fields like computer science, operations research, and economics, including shortest path problems, resource allocation, and sequence alignment in bioinformatics.

2. Discuss the key characteristics of problems suitable for dynamic programming solutions.

Key Characteristics of Problems Suitable for Dynamic Programming Solutions:

1. **Overlapping Subproblems**:
    - The problem can be broken down into smaller subproblems that are reused multiple times in the overall problem. Instead of solving the same subproblem repeatedly, DP solves it once and stores the result for future use.
    - **Example**: In the Fibonacci sequence, calculating F(n) requires calculating F(n-1) and F(n-2), which in turn require further smaller Fibonacci numbers. These subproblems overlap significantly.
2. **Optimal Substructure**:
    - The optimal solution to the problem can be constructed from the optimal solutions of its subproblems. This means that solving each subproblem optimally will lead to an optimal solution for the entire problem.
    - **Example**: In the shortest path problem, the shortest path from a starting point to a destination can be found by combining the shortest paths of subroutes.
3. **Memoization or Tabulation Feasibility**:
    - The problem allows for storing intermediate results (memoization) or building solutions iteratively (tabulation). This storage of results is crucial for avoiding redundant computations and ensuring efficiency.
    - **Example**: In DP, a table or dictionary is often used to store the results of subproblems, which are then used to build the final solution.

3. What are the two main approaches to solving dynamic programming problems?

Explain each with examples.

Top-Down Approach (Memoization):

- **Explanation**: In this approach, the problem is solved by breaking it down into smaller subproblems recursively. The results of these subproblems are stored (or memoized) so that they do not need to be recomputed if encountered again. This approach works by starting with the original problem and solving smaller subproblems as needed.
- **Example**:
    - **Fibonacci Sequence**: To find F(n), you compute F(n-1) and F(n-2) recursively. If the value of F(n-1) is needed again, it is retrieved from memory rather than recalculated

4. Implement the Fibonacci sequence using dynamic programming. Provide both

recursive and dynamic programming solutions.

Here are two approaches: the recursive solution with memoization (Top-Down approach) and the iterative solution with tabulation (Bottom-Up approach).

1. Recursive Solution with Memoization (Top-Down Approach)

This method uses recursion and stores the results of subproblems to avoid redundant calculations.

Explanation:

- The function checks if the Fibonacci number for n has already been computed. If not, it calculates it by recursively calling the function for n-1 and n-2, and stores the result in the memo dictionary.
- This avoids redundant calculations, making the function efficient.

5. Explain the 0/1 knapsack problem and its variations.

Problem Definition:

- The 0/1 Knapsack problem is a classic optimization problem in which you are given a set of items, each with a weight and a value. Your goal is to determine the most valuable combination of items to include in a knapsack (bag) without exceeding a specified weight limit.
- **0/1** means that you can either include the whole item in the knapsack (1) or exclude it entirely (0); you cannot take fractional parts of an item.

Formal Statement:

- **Input**:
    - A list of n items, where each item i has a weight w[i] and a value v[i].
    - A knapsack with a maximum weight capacity W.
- **Output**:
    - A selection of items such that the total weight does not exceed W, and the total value is maximized.

Example:

- Suppose you have three items:
    - Item 1: weight = 2 kg, value = $3
    - Item 2: weight = 3 kg, value = $4
    - Item 3: weight = 4 kg, value = $5
- If the knapsack's weight capacity is 5 kg, the best combination is to take Item 1 and Item 2, for a total value of $7.