**Department of Computer Science and Engineering**

**Chandpur Science and Technology University**

|  |  |
| --- | --- |
| Course Code: CSE 2201 | Credits: 1.50 |
| Course Name: Algorithm Design and Analysis | Semester: 2-2 |

**Lab 06**

**Implementation of Fibonacci Series using Recursion and Dynamic Programming**

1. **Learning Objectives**

By the end of this lab, students should be able to:

* Implement the Fibonacci series using a purely recursive approach.
* Implement the Fibonacci series using dynamic programming (both memoization and tabulation techniques).
* Analyze and compare the time complexities of recursive and dynamic programming approaches for the Fibonacci series.
* Understand the concept of overlapping subproblems and optimal substructure, which are key to dynamic programming.

1. **Lesson Fit:**

Prerequisite: C/C++, Data Structure

1. **Theory Recap:**

### 

The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones, usually starting with 0 and 1. The sequence is: 0,1,1,2,3,5,8,13,21,…

Mathematically, it can be defined as:

* F(0)=0
* F(1)=1
* F(n)=F(n−1)+F(n−2) for n>1

### Recursive Approach:

A direct translation of the mathematical definition leads to a recursive function.

**Example:** To calculate F(5): F(5)=F(4)+F(3) F(4)=F(3)+F(2) F(3)=F(2)+F(1) F(2)=F(1)+F(0)

Notice that F(3) and F(2) are calculated multiple times. This leads to a very inefficient solution due to **overlapping subproblems**.

**Time Complexity:** The time complexity of this approach is exponential, specifically O(2n). This is because the number of function calls roughly doubles with each increase in n.

### Dynamic Programming Approach:

Dynamic Programming is an optimization technique used to solve complex problems by breaking them down into simpler subproblems. It is applicable to problems that have two properties:

1. **Overlapping Subproblems:** The same subproblems are solved multiple times.
2. **Optimal Substructure:** The optimal solution to the problem can be constructed from the optimal solutions of its subproblems.

For the Fibonacci sequence, both properties hold true.

#### **Memoization (Top-Down DP):**

This approach is essentially recursion with a cache (or "memo"). When a function is called, it first checks if the result for the given input has already been computed and stored in the cache. If it has, the stored value is returned directly. Otherwise, the function computes the result, stores it in the cache, and then returns it.

**Process:**

1. Initialize a data structure (e.g., an array or hash map) to store computed results, often filled with a special value (like -1) to indicate "not computed yet".
2. In the recursive function, check if the result for the current n is already in the cache. If yes, return it.
3. Otherwise, compute the result, store it in the cache, and then return.

**Time Complexity:** O(n). Each Fibonacci number up to n is computed only once. **Space Complexity:** O(n) for the memoization table and O(n) for the recursion stack space.

#### **Tabulation (Bottom-Up DP):**

This approach solves the problem iteratively, typically using a loop. It builds up the solution from the base cases to the desired result. Instead of recursion, it fills a table (array) with solutions to subproblems in a specific order.

**Process:**

1. Initialize a table (e.g., an array dp) of size n+1.
2. Fill in the base cases: dp[0] = 0, dp[1] = 1.
3. Iterate from i=2 to n, calculating dp[i] = dp[i-1] + dp[i-2].
4. The final result is dp[n].

**Time Complexity:** O(n). It involves a single loop iterating n−1 times.

**Space Complexity:** O(n) for the DP table. (This can be further optimized to O(1) space since only the two previous values are needed at any point).

**Example:**

**Lab 6 Activity List**

# Experiment # 1: Implementation of Fibonacci Series using Recursion and Dynamic Programming

**Tasks:**

* Write pseudocode for the Fibonacci Series using Recursion and Dynamic Programming.
* Implement the algorithm in C/C++/Python.
* Test on various number of data.
* Compare the result with the known optimal solution (where applicable).

**📊 Time Complexity:**

**🧮 Space Complexity:**

### 📊 Comparison Table (Empirical)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Coins Used (Greedy) | Minimum Coins (Optimal) | Greedy Correct? |
|  |  |  |  |
|  |  |  |  |

**Report:**

The report should cover the following

Name of the Experiment

1. Objective
2. Algorithm
3. Theoretical Solution of given problem
4. Practical Work:
   1. Pseudocode
   2. Source Code in C/CPP/Python
5. Analysis Table

| Algorithm | Best Case | Worst Case | Avg Case | Space |

|----------------|-----------|------------|----------|--------|

1. Observations
2. Challenges
3. Conclusion

📸 Attachments:

- Screenshot of program output.

- Manual step count snapshots.

- Complexity graph (drawn or plotted).