# Dynamic Programming (DP)

Ansari Abrar

July 25, 2024

## 1 Introduction:

Dynamic Programming (DP) is a method used to solve complex problems by breaking them down into simpler subproblems. It solves each subproblem only once and stores the results to avoid redundant computations.

DP focuses on two main principles:

1. Avoiding redundant computations by following a specific pattern to reach the final result.
2. Discarding useless information.

## 2 Process:

The process of DP can be visualized with the following steps:

$$\begin{tikzcd}
\boxed{\text{Identify subproblems}} \arrow[r] & \boxed{\text{Store solutions}} \arrow[r] & \boxed{\text{Build up solutions}} \arrow[r] & \boxed{\text{Avoid redundancy}} \arrow[r] & \boxed{\text{Efficient result}}
\end{tikzcd}$$

## 3 Approaches:

1. **Top-Down Approach (Memoization):** Here, we start with the final solution and recursively break it down into smaller subproblems. To avoid redundant calculations, we store the results of solved subproblems in a memoization table.
    - Suitable when the number of subproblems is large and many of them are reused.
2. **Bottom-Up Approach (Tabulation):** Here, we start with the smallest subproblems and gradually build up to the final solution. We store the results of solved subproblems in a table to avoid redundant calculations.
    - Suitable when the number of subproblems is small and the optimal solution can be directly computed from the solutions to smaller sub-problems.

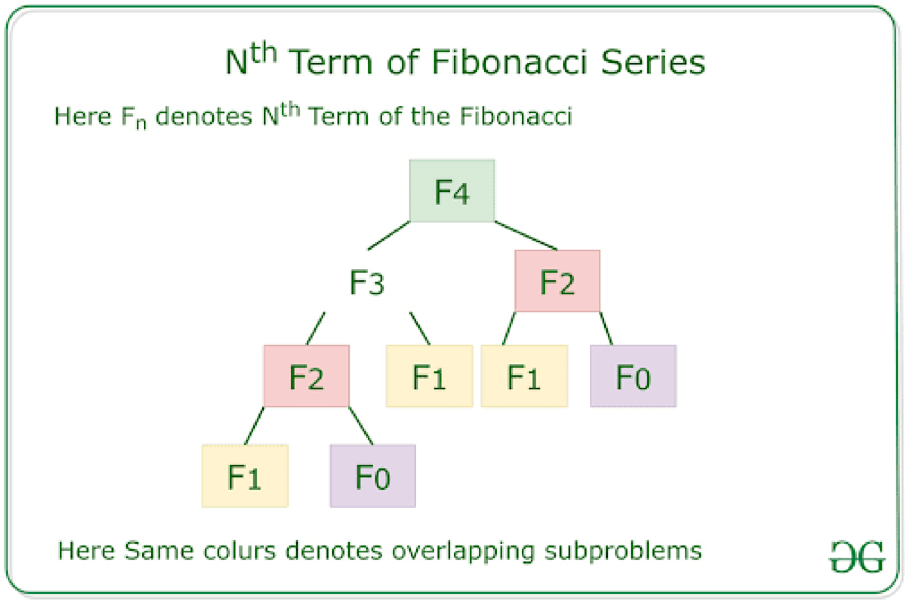

Nth Term of Fibonacci Series
Here Fn denotes Nth Term of the Fibonacci
F4
F3
F2
F2
F1
F1
F0
F1
F0
Here Same colurs denotes overlapping subproblems

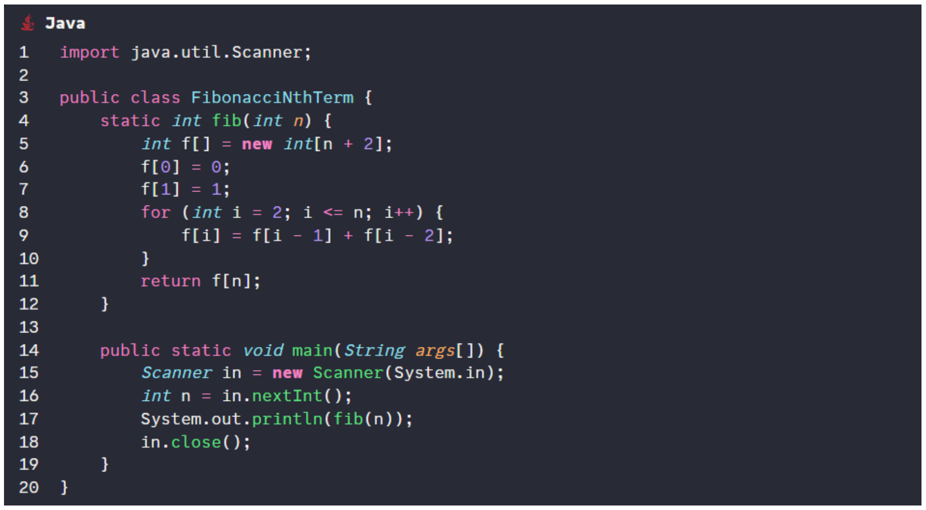
Figure 1: Fibonacci Number tree(Credit: GFG)

## Example:

**Factorial:** *Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . .*

- Subproblems: F(0), F(1), F(2), F(3), . . .
- Store Solutions: Create a table to store the values of F(n) as they are calculated.
- Build Up Solutions: For F(n), look up F(n-1) and F(n-2) in the table and add them.
- Avoid Redundancy: The table ensures that each subproblem (e.g., F(2)) is solved only once.

Yes, it's recursion but shorter and more efficient as the program doesn't recalculate the existing calculations again. In conclusion, we can efficiently calculate the Fibonacci sequence without having to recompute subproblems with DP. Here's the code using DP:



Java

```java
import java.util.Scanner;

public class FibonacciNthTerm {
    static int fib(int n) {
        int f[] = new int[n + 2];
        f[0] = 0;
        f[1] = 1;
        for (int i = 2; i <= n; i++) {
            f[i] = f[i - 1] + f[i - 2];
        }
        return f[n];
    }

    public static void main(String args[]) {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        System.out.println(fib(n));
        in.close();
    }
}
```

Figure 2: Code for Fibonacci using DP in Java