Day 25: Solve Recursion Problems

Abhinav Yadav

"Recursion is the process of defining something in terms of itself."
— Anonymous

---

1 Introduction

Recursion is a programming technique where a function calls itself in order to solve a problem. It can often simplify code by breaking down complex problems into smaller subproblems. However, recursive functions must have a **base case** (to stop the recursion) and a **recursive case** (to continue breaking down the problem).

In this document, we will explore three classic examples of recursion:

- Factorial of a number.
- Fibonacci series up to n.
- Tower of Hanoi.

We will explain the recursion depth, base case, and provide step-by-step recursion trees to help understand the process.

2 Factorial of a Number

The factorial of a number n, denoted as $n!$, is the product of all positive integers less than or equal to n. It is defined as:

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 1$$

For $n = 0$, the factorial is defined as $0! = 1$.

2.1 Recursive Definition

The recursive function for calculating factorial is:

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \times \text{factorial}(n-1) & \text{if } n > 0 \end{cases}$$

2.2 Code Implementation

```java
import java.util.Scanner;

public class Factorial {
    // Recursive function to calculate factorial
    public static int factorial(int n) {
        if (n == 0) {
            return 1; // Base case
        } else {
            return n * factorial(n - 1); // Recursive case
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int num = scanner.nextInt();

        System.out.println("Factorial of " + num + " is " +
           factorial(num));

        scanner.close();
    }
}
```

2.3 Recursion Tree Example for Factorial of 5

The recursion tree for calculating 5! is shown below:

$$\text{factorial}(5) = 5 \times \text{factorial}(4)$$
$$\text{factorial}(4) = 4 \times \text{factorial}(3)$$
$$\text{factorial}(3) = 3 \times \text{factorial}(2)$$
$$\text{factorial}(2) = 2 \times \text{factorial}(1)$$
$$\text{factorial}(1) = 1 \times \text{factorial}(0)$$
$$\text{factorial}(0) = 1 \quad \text{(Base case)}$$

Thus, the factorial of 5 is calculated as:

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

3 Fibonacci Series

The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones. It is defined as:

$$F(0) = 0, \quad F(1) = 1$$
$$F(n) = F(n-1) + F(n-2) \quad \text{for } n \geq 2$$

3.1 Recursive Definition

The recursive function for calculating the n^{th} Fibonacci number is:

$$\text{fibonacci}(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \text{fibonacci}(n-1) + \text{fibonacci}(n-2) & \text{if } n > 1 \end{cases}$$

3.2 Code Implementation

```java
import java.util.Scanner;

public class Fibonacci {
    // Recursive function to calculate Fibonacci number
    public static int fibonacci(int n) {
        if (n == 0) {
            return 0; // Base case
        } else if (n == 1) {
            return 1; // Base case
        } else {
            return fibonacci(n - 1) + fibonacci(n - 2); // Recursive case
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int num = scanner.nextInt();

        System.out.println("Fibonacci number at position " + num + " is " + fibonacci(num));

        scanner.close();
    }
}
```

3.3 Recursion Tree Example for Fibonacci(5)

The recursion tree for calculating $F(5)$ is shown below:

$$F(5) = F(4) + F(3)$$
$$F(4) = F(3) + F(2)$$
$$F(3) = F(2) + F(1)$$
$$F(2) = F(1) + F(0)$$
$$F(1) = 1 \quad \text{(Base case)}$$

$$F(0) = 0 \quad \text{(Base case)}$$

Thus, the Fibonacci number at position 5 is:

$$F(5) = 5$$

4 Tower of Hanoi

The Tower of Hanoi is a classic puzzle where you need to move a stack of disks from one pole to another, following certain rules:

- Only one disk can be moved at a time.
- A disk can only be placed on top of a larger disk or an empty pole.
- The objective is to move all disks from the source pole to the destination pole.

4.1 Recursive Definition

The recursive function to solve the Tower of Hanoi is:

$$\text{tower}(n, A, B, C) = \begin{cases} \text{Move disk 1 from A to C} & \text{if } n = 1 \\ \text{tower}(n-1, A, C, B) \quad \text{Move disk } n \text{ from A to C} & \\ \text{tower}(n-1, B, A, C) & \text{if } n > 1 \end{cases}$$

4.2 Code Implementation

```java
import java.util.Scanner;

public class TowerOfHanoi {
    // Recursive function to solve Tower of Hanoi
    public static void towerOfHanoi(int n, char source, char auxiliary, char destination) {
        if (n == 1) {
            System.out.println("Move disk 1 from " + source + " to " + destination);
            return;
        }
        towerOfHanoi(n - 1, source, destination, auxiliary);
        System.out.println("Move disk " + n + " from " + source + " to " + destination);
        towerOfHanoi(n - 1, auxiliary, source, destination);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of disks: ");
        int n = scanner.nextInt();

```

```
        // A, B, C represent the source, auxiliary, and
           destination pegs respectively
        towerOfHanoi(n, 'A', 'B', 'C');

        scanner.close();
    }
}
```

4.3 Recursion Tree Example for Tower of Hanoi (3 disks)

The recursion tree for solving the Tower of Hanoi with 3 disks is shown below:

$$\text{towerOfHanoi}(3, A, B, C) = \text{towerOfHanoi}(2, A, C, B)$$
$$\text{towerOfHanoi}(2, A, C, B) = \text{towerOfHanoi}(1, A, B, C)$$
$$\text{Move disk 1 from A to C}$$
$$\text{Move disk 2 from A to B}$$
$$\text{towerOfHanoi}(1, C, A, B)$$
$$\text{Move disk 1 from C to B}$$
$$\text{Move disk 3 from A to C}$$
$$\text{towerOfHanoi}(2, B, A, C) = \text{towerOfHanoi}(1, B, C, A)$$
$$\text{Move disk 1 from B to A}$$
$$\text{Move disk 2 from B to C}$$
$$\text{towerOfHanoi}(1, A, B, C)$$
$$\text{Move disk 1 from A to C}$$

Thus, the moves are displayed as:

Move disk 1 from A to C, Move disk 2 from A to B, Move disk 1 from C to B, Move disk 3 from A to

5 Conclusion

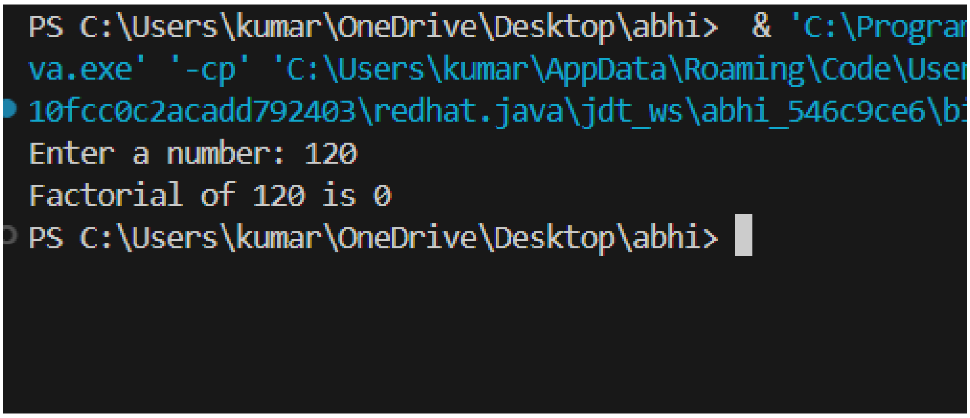
Recursion is a powerful technique that simplifies complex problems by breaking them into smaller subproblems. The depth of recursion depends on the size of the problem, and the base case ensures that recursion terminates. Understanding recursion trees is crucial to visualizing the step-by-step process and recognizing the pattern in recursive calls.

6 Outputs of Recursive Factorial, Fibonacci No. and Tower of Hanoi

6.1 Output of Factorial of a number

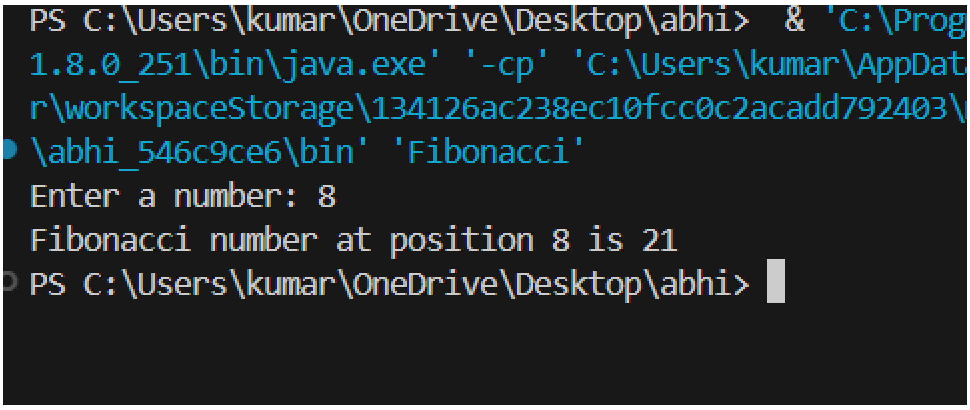
6.2 Output of Fibonacci Series

6.3 Output for Tower of Hanoi



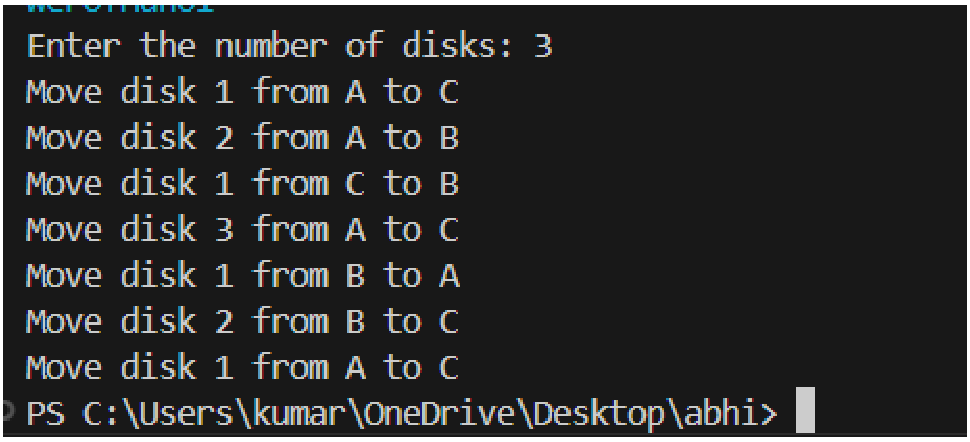
```
PS C:\Users\kumar\OneDrive\Desktop\abhi>  & 'C:\Progra
va.exe' '-cp' 'C:\Users\kumar\AppData\Roaming\Code\Use
10fcc0c2acadd792403\redhat.java\jdt_ws\abhi_546c9ce6\b
Enter a number: 120
Factorial of 120 is 0
PS C:\Users\kumar\OneDrive\Desktop\abhi>
```

Figure 1: 6.1 Output



```
PS C:\Users\kumar\OneDrive\Desktop\abhi>  & 'C:\Prog
1.8.0_251\bin\java.exe' '-cp' 'C:\Users\kumar\AppDat
r\workspaceStorage\134126ac238ec10fcc0c2acadd792403\
\abhi_546c9ce6\bin' 'Fibonacci'
Enter a number: 8
Fibonacci number at position 8 is 21
PS C:\Users\kumar\OneDrive\Desktop\abhi>
```

Figure 2: 6.2 Output



```
Enter the number of disks: 3
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
PS C:\Users\kumar\OneDrive\Desktop\abhi>
```

Figure 3: 6.3 Output