UNDERSTANDING RECURSION WITH THE TOWER OF HANOI PROBLEM IN JAVA

Executive Summary

This case study breaks down the recursive logic behind the Tower of Hanoi problem—a classic example for teaching recursion and stack memory behavior. Implemented in Java, the solution includes step-by-step explanations of each recursive call, visual flow representation, and a dry-run for 3-disk and 4-disk scenarios. Students are guided to visualize recursion depth, base cases, and the call stack, making it ideal for both exam preparation and logic-building assignments.

1. Introduction

Recursion is a fundamental but often misunderstood concept in programming. Tower of Hanoi provides a structured and visual method for learning recursive problem-solving. The challenge is to move a stack of disks from one rod to another, obeying three rules:

1. Only one disk can be moved at a time.
2. A larger disk may not be placed on a smaller disk.
3. Only the top disk may be moved.

2. Problem Definition

The task: Move n disks from **source rod A** to **destination rod C** using **auxiliary rod B**.

Key challenges students face:

- Misunderstanding how recursive calls build upon smaller subproblems
- Difficulty tracing the control flow and call returns
- Confusion around parameters and how the function progresses

3. Java Code and Logic Explanation

java

CopyEdit

```
public class TowerOfHanoi {
    static void solveHanoi(int n, char source, char auxiliary, char destination) {
        if (n == 1) {
            System.out.println("Move disk 1 from " + source + " to " + destination);
            return;
        }

        solveHanoi(n - 1, source, destination, auxiliary);
        System.out.println("Move disk " + n + " from " + source + " to " + destination);
        solveHanoi(n - 1, auxiliary, source, destination);
    }

    public static void main(String[] args) {
        int n = 3;
        solveHanoi(n, 'A', 'B', 'C');
    }
}
```

Explanation:

- **Base Case**: When n == 1, move the disk directly.
- **Recursive Case**:
    1. Move n-1 disks from source to auxiliary.
    2. Move the nth disk to destination.
    3. Move n-1 disks from auxiliary to destination.

Each step relies on the assumption that a smaller version of the problem can be solved recursively.

4. Dry Run: 3 Disks

plaintext

CopyEdit

solveHanoi(3, A, B, C)

├ solveHanoi(2, A, C, B)

│ ├ solveHanoi(1, A, B, C) → Move disk 1 from A to C

│ └ Move disk 2 from A to B

│ └ solveHanoi(1, C, A, B) → Move disk 1 from C to B

└ Move disk 3 from A to C

└ solveHanoi(2, B, A, C)

├ solveHanoi(1, B, C, A) → Move disk 1 from B to A

└ Move disk 2 from B to C

└ solveHanoi(1, A, B, C) → Move disk 1 from A to C

Total Moves = $2^n - 1 = 7$

5. Stack Memory Illustration

For n = 3, the call stack will grow to a maximum of 3 nested calls. When each recursive branch completes, the stack unwinds in reverse order, completing each move.

6. Evaluation and Learning Outcomes

Common Mistakes Addressed:

- Students often reverse parameters in recursive calls.

- Forget to print the move after the first recursive call.
- Misplace the base case.

Pedagogical Benefit:

- Improves understanding of recursion depth
- Builds confidence in writing nested logic
- Develops debugging skills using print statements for tracing

7. Enhancements

- Add step counter using a global variable
- Create a GUI version using Java Swing
- Allow user input for number of disks via console or dialog
- Store steps in a list for export or animation

8. Conclusion

The Tower of Hanoi problem provides a structured, recursive problem that teaches critical thinking and control flow analysis. By implementing and tracing it in Java, students gain not just syntax practice but deep logic development skills.

9. References

- Cormen, T.H., Leiserson, C.E., Rivest, R.L. (2009). *Introduction to Algorithms*
- Oracle Java Documentation (Recursion and Stack Frames)
- Towers of Hanoi – MIT OpenCourseWare (CS Introduction)
- GeeksforGeeks: Tower of Hanoi Algorithm Explanation