recursion

October 3, 2024

Recursion Assignment

[ ]:
```
# Can you explain the logic and working of the Tower of Hanoi algorithm by writing a Java program? How does the recursion work, and how are the movements of disks between rods accomplished?
Logic of Tower of Hanoi
Problem Definition: You have three rods: A (source), B (auxiliary), and C (destination). You need to move n disks from rod A to rod C, following these rules:

Only one disk can be moved at a time.
Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
No larger disk may be placed on top of a smaller disk.
Base Case: If there is only one disk, simply move it from the source rod to the destination rod.

Recursive Case: For n disks:

Move the top n-1 disks from the source rod (A) to the auxiliary rod (B), using the destination rod (C) as a temporary holding area.
Move the nth (largest) disk directly from the source rod (A) to the destination rod (C).
Move the n-1 disks from the auxiliary rod (B) to the destination rod (C), using the source rod (A) as a temporary holding area.

public class TowerOfHanoi {

    // Recursive function to solve the Tower of Hanoi
    public static void solveTowerOfHanoi(int n, char source, char auxiliary, char destination) {
        // Base case: Only one disk
        if (n == 1) {
            System.out.println("Move disk 1 from rod " + source + " to rod " + destination);
            return;
        }
```

```
        // Move n-1 disks from source to auxiliary, using destination as␣
 ↪temporary
        solveTowerOfHanoi(n - 1, source, destination, auxiliary);

        // Move the nth disk from source to destination
        System.out.println("Move disk " + n + " from rod " + source + " to rod␣
 ↪" + destination);

        // Move the n-1 disks from auxiliary to destination, using source as␣
 ↪temporary
        solveTowerOfHanoi(n - 1, auxiliary, source, destination);
    }

    public static void main(String[] args) {
        int n = 3; // Number of disks
        System.out.println("The sequence of moves involved in the Tower of␣
 ↪Hanoi are:");
        solveTowerOfHanoi(n, 'A', 'B', 'C'); // A, B and C are names of rods
    }
}
Explanation of the Code
Function solveTowerOfHanoi:

Takes four parameters: the number of disks (n), the source rod (source), the␣
 ↪auxiliary rod (auxiliary), and the destination rod (destination).
If n is 1, it simply prints the move for the single disk.
For more than one disk, it first recursively calls itself to move n-1 disks to␣
 ↪the auxiliary rod.
Then, it prints the move of the nth disk to the destination rod.
Finally, it recursively calls itself again to move the n-1 disks from the␣
 ↪auxiliary rod to the destination rod.
Main Method:

Sets the number of disks (n = 3 in this case) and calls the solveTowerOfHanoi␣
 ↪function with the appropriate parameters.
How Recursion Works
Each recursive call breaks down the problem into smaller subproblems (moving␣
 ↪n-1 disks).
The base case ensures that when there is only one disk, it can be moved␣
 ↪directly.
The sequence of function calls creates a stack of operations that unwind when␣
 ↪the base case is reached, allowing for the completion of all moves.
Movements of Disks
The movements are accomplished through print statements showing which disk is␣
 ↪moved from which rod to which rod, adhering to the rules of the game.
```

[1]:
```
# 2. Q.2 Given two strings word1 and word2, return the minimum number of␣
 ↪operations required to convert word1 to word2.
# Example 1:
# Input: word1 = "horse", word2 = "ros"
# Output: 3
# Explanation:
# horse -> rorse (replace 'h' with 'r')
# rorse -> rose (remove 'r')
# rose -> ros (remove 'e')
def min_distance(word1, word2):
    m, n = len(word1), len(word2)
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(m + 1):
        dp[i][0] = i
    for j in range(n + 1):
        dp[0][j] = j

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if word1[i - 1] == word2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            else:
                dp[i][j] = min(
                    dp[i - 1][j] + 1,
                    dp[i][j - 1] + 1,
                    dp[i - 1][j - 1] + 1
                )

    return dp[m][n]

word1 = "horse"
word2 = "ros"
output = min_distance(word1, word2)
print(output)
```

```
3
```

[3]:
```
# Example 2:

# Input: word1 = "intention", word2 = "execution"
# Output: 5
# Explanation:
# intention -> inention (remove 't')
# inention -> enention (replace 'i' with 'e')
# enention -> exention (replace 'n' with 'x')
# exention -> exection (replace 'n' with 'c')
```

```
# exection -> execution (insert 'u')

def min_distance(word1, word2):
    m, n = len(word1), len(word2)
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(m + 1):
        dp[i][0] = i
    for j in range(n + 1):
        dp[0][j] = j

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if word1[i - 1] == word2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            else:
                dp[i][j] = min(
                    dp[i - 1][j] + 1,
                    dp[i][j - 1] + 1,
                    dp[i - 1][j - 1] + 1
                )

    return dp[m][n]

word1 = "intention"
word2 = "execution"
output = min_distance(word1, word2)
print(output)
```

```
5
```

[5]:
```
# 3 Print the max value of the array [ 13, 1, -3, 22, 5].
arr = [13, 1, -3, 22, 5]
max_value = max(arr)
print(max_value)  # Output: 22
```

```
22
```

[7]:
```
# 4.Find the sum of the values of the array [92, 23, 15, -20, 10]
arr = [92, 23, 15, -20, 10]
total_sum = sum(arr)
print(total_sum)
```

```
120
```

[9]:
```
# 5 Given a number n. Print if it is an armstrong number or not.An armstrong␣
 ↪number is a number if the sum of every digit in that number raised to the␣
 ↪power of total digits in that number is equal to the number.
```

```
# Example : 153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153 hence 153 is an armstrong number. (Easy)
# Input1 : 153
# Output1 : Yes
# Input 2 : 134
# Output2 : No

def is_armstrong_number(n):
    str_n = str(n)
    num_digits = len(str_n)

    sum_of_powers = sum(int(digit) ** num_digits for digit in str_n)

    return sum_of_powers == n

input1 = 153
output1 = "Yes" if is_armstrong_number(input1) else "No"
print(f"Input: {input1}, Output: {output1}")

input2 = 134
output2 = "Yes" if is_armstrong_number(input2) else "No"
print(f"Input: {input2}, Output: {output2}")
```

```
Input: 153, Output: Yes
Input: 134, Output: No
```

[ ]:
```
#
```

[ ]:
```
#
```

[ ]:
```
#
```

[ ]:
```
#
```

[ ]:
```
#
```

[ ]:
```
#
```

[ ]:
```
#
```

[ ]:
```
#
```