Untitled4

December 26, 2023

[ ]:
```
#Recursion Assignment

#Q.1 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?


#The recursive solution to the Tower of Hanoi involves breaking down the␣
 ↪problem into subproblems. The basic idea is to move n-1 disks from the␣
 ↪source rod to an auxiliary rod, then move the nth disk from the source rod␣
 ↪to the destination rod, and finally, move the n-1 disks from the auxiliary␣
 ↪rod to the destination rod.

class GFG :
        {
    # Java recursive function to solve tower of hanoi puzzle
    static void towerOfHanoi( int n, char from_rod, char to_rod, char aux_rod)
    {
        if (n == 1)
        {
            System.out.println("Move disk 1 from rod " +  from_rod + " to rod "␣
 ↪+ to_rod);
            return;
        }
        towerOfHanoi(n-1, from_rod, aux_rod, to_rod);
        System.out.println("Move disk " + n + " from rod " +  from_rod + " to␣
 ↪rod " + to_rod);
        towerOfHanoi(n-1, aux_rod, to_rod, from_rod);
    }

    //  Driver method
    public static void main(String args[])
    {
        int n = 4; // Number of disks
        towerOfHanoi(n, \'A\', \'C\', \'B\');  // A, B and C are names of rods
    }
```

```
}n this program, the towerOfHanoi method is a recursive function that takes the␣
 ↪number of disks (n) and the source, auxiliary, and destination rods as␣
 ↪parameters. The base case (n == 1) is when there is only one disk to move,␣
 ↪and in that case, the program prints the movement of the disk from the␣
 ↪source rod to the destination rod. Otherwise, it recursively moves n-1 disks␣
 ↪from the source rod to the auxiliary rod, then moves the nth disk from the␣
 ↪source rod to the destination rod, and finally, recursively moves the n-1␣
 ↪disks from the auxiliary rod to the destination rod.

#The recursive nature of the algorithm handles the breakdown of the problem␣
 ↪into smaller subproblems, and each recursive call represents a step in␣
 ↪solving the Tower of Hanoi for a smaller number of disks.
```

[ ]:
```
'''
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')
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')
'''

public class EditDistance {
    public static void main(String[] args) {
        String word1 = "horse";
        String word2 = "ros";
        int result1 = minDistance(word1, word2);
        System.out.println("Example 1: " + result1);

        String word3 = "intention";
        String word4 = "execution";
        int result2 = minDistance(word3, word4);
        System.out.println("Example 2: " + result2);
    }
```

```
    public static int minDistance(String word1, String word2) {
        int m = word1.length();
        int n = word2.length();

        // Create a two D array to store the minimum operations
        int[][] dp = new int[m + 1][n + 1];

        // Initialize the first row and column
        for (int i = 0; i <= m; i++) {
            dp[i][0] = i;
        }
        for (int j = 0; j <= n; j++) {
            dp[0][j] = j;
        }

        // Build the matrix
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (word1.charAt(i - 1) == word2.charAt(j - 1)) {
                    dp[i][j] = dp[i - 1][j - 1];
                } else {
                    dp[i][j] = 1 + Math.min(dp[i - 1][j - 1], Math.min(dp[i][j␣
 ↪- 1], dp[i - 1][j]));
                }
            }
        }

        // The bottom-right cell contains the minimum number of operations
        return dp[m][n];
    }
}
```

[ ]:
```
#Q. 3 Print the max value of the array [ 13, 1, -3, 22, 5].

public class MaxValueInArray {
    public static void main(String[] args) {
        int[] array = {13, 1, -3, 22, 5};

        int maxValue = findMaxValue(array);

        System.out.println("The maximum value in the array is: " + maxValue);
    }

    public static int findMaxValue(int[] array) {
        // Check if the array is not empty
        if (array == null || array.length == 0) {
```

```
            throw new IllegalArgumentException("Array is empty or null");
        }

        // Initialize max with the first element of the array
        int max = array[0];

        // Iterate through the array to find the maximum value
        for (int i = 1; i < array.length; i++) {
            if (array[i] > max) {
                max = array[i];
            }
        }

        return max;
    }
}

#Output
#The maximum value in the array is: 22
```

[ ]:
```
#Q.4 Find the sum of the values of the array [92, 23, 15, -20, 10].

public class SumOfArray {
    public static void main(String[] args) {
        int[] array = {92, 23, 15, -20, 10};

        int sum = findSum(array);

        System.out.println("The sum of the values in the array is: " + sum);
    }

    public static int findSum(int[] array) {
        // Check if the array is not empty
        if (array == null || array.length == 0) {
            throw new IllegalArgumentException("Array is empty or null");
        }

        // Initialize sum to 0
        int sum = 0;

        // Iterate through the array to find the sum
        for (int value : array) {
            sum += value;
        }

        return sum;
    }
```

```
}

#Output
#The sum of the values in the array is: 120

'''Q.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'''

public class ArmstrongNumber {
    public static void main(String[] args) {
        int input1 = 153;
        int input2 = 134;

        System.out.println(input1 + " is an Armstrong number: " +␣
 ↪isArmstrong(input1));
        System.out.println(input2 + " is an Armstrong number: " +␣
 ↪isArmstrong(input2));
    }

    public static boolean isArmstrong(int number) {
        // Convert the number to a string to find the total digits
        String numberStr = Integer.toString(number);
        int totalDigits = numberStr.length();

        // Calculate the sum of each digit raised to the power of total digits
        int sum = 0;
        int originalNumber = number;
        while (number > 0) {
            int digit = number % 10;
            sum += Math.pow(digit, totalDigits);
            number /= 10;
        }

        // Check if the sum is equal to the original number
        return sum == originalNumber;
    }
}

#Output
```

```
'''153 is an Armstrong number: true
134 is an Armstrong number: false''
```