DSArecursion

February 5, 2024

[4]:
```
# 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?
def tower_of_hanoi(n,source,auxilary,destination):
    if n== 1:
        print(f"Move disk1 from {source} to {destination}")
        return
    tower_of_hanoi(n-1,source,destination,auxilary)
    print(f"move disk {n} from {source} to {destination}")
    tower_of_hanoi(n-1,auxilary,source,destination)

n=3
tower_of_hanoi(n,'source','auxilary','destination')
```

```
Move disk1 from source to destination
move disk 2 from source to auxilary
Move disk1 from destination to auxilary
move disk 3 from source to destination
Move disk1 from auxilary to source
move disk 2 from auxilary to destination
Move disk1 from source to destination
```

[5]:
```
# 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')
# Example 2:
# Input: word1 = "intention", word2 = "execution"
# Output: 5
# Explanation:
# intention -> inention (remove 't')
```

```python
# inention -> enention (replace 'i' with 'e')
# enention -> exention (replace 'n' with 'x')
# exention -> exection (replace 'n' with 'c')
# exection -> execution (insert 'u')


def minDistance(word1, word2):
        """
        :type word1: str
        :type word2: str
        """
        m = len(word1)
        n = len(word2)

        # Create a 2D DP table to store the minimum edit distance
        dp = [[0 for _ in range(n + 1)] for _ in range(m + 1)]

        # Fill the base cases
        for i in range(m + 1):
            dp[i][0] = i  # If word2 is empty, we need to insert i characters␣
 ↪from word1
        for j in range(n + 1):
            dp[0][j] = j  # If word1 is empty, we need to insert j characters␣
 ↪from word2

        # Fill the DP table
        for i in range(1, m + 1):
            for j in range(1, n + 1):
                if word1[i - 1] == word2[j - 1]:
                    # No change needed, so the distance remains the same
                    dp[i][j] = dp[i - 1][j - 1]
                else:
                    # Minimum of the three operations: insert, delete, replace
                    dp[i][j] = min(dp[i - 1][j] + 1,  # Insert
                                   dp[i][j - 1] + 1,  # Delete
                                   dp[i - 1][j - 1] + 1)  # Replace

        return dp[m][n]

# Example usage
word1 = "horse"
word2 = "ros"
print(minDistance(word1, word2))  # Output: 3

word1 = "intention"
word2 = "execution"
print(minDistance(word1, word2))  # Output: 5
```

```
3
5
```

[6]:
```
# Q. 3 Print the max value of the array [ 13, 1, -3, 22, 5].
arr=[13,1,-3,22,5]
max(arr)
```

[6]:
```
22
```

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

[7]:
```
120
```

[9]:
```
# # Q.5Given 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(n):
  """
  This function checks if a number is an Armstrong number.

  Args:
    n: The number to check.

  Returns:
    True if the number is an Armstrong number, False otherwise.
  """

  # Convert the number to a string and get the number of digits.
  num_str = str(n)
  num_digits = len(num_str)

  # Calculate the sum of each digit raised to the power of the number of digits.
  sum_of_digits = 0
  for digit in num_str:
    sum_of_digits += int(digit) ** num_digits

  # Return True if the sum is equal to the original number, False otherwise.
```

```
  return sum_of_digits == n

# Test cases
print(is_armstrong(153))  # True
print(is_armstrong(134))  # False
```

```
True
False
```

[ ]: