Find It's type and solve it & make note of it

Post Title

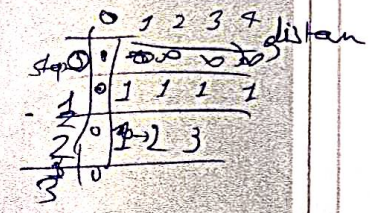
Count number of ways to cover a distance

Given a distance 'dist', count total number of ways to cover the distance with 1, 2 and 3 steps.

Examples:

```
Input: n = 3
Output: 4
Explanation:
Below are the four ways
 1 step + 1 step + 1 step
 1 step + 2 step
 2 step + 1 step
 3 step

Input: n = 4
Output: 7
Explanation:
Below are the four ways
 1 step + 1 step + 1 step + 1 step
 1 step + 2 step + 1 step
 2 step + 1 step + 1 step
 1 step + 1 step + 2 step
 2 step + 2 step
 3 step + 1 step
 1 step + 3 step
```



Recursive solution

- **Approach:** There are n stairs, and a person is allowed to next step, skip one position or skip two positions. So there are n positions. The idea is standing at the ith position the person can move by i+1, i+2, i+3 position. So a recursive function can be formed where at current index i the function is recursively called for i+1, i+2 and i+3 positions.
  There is another way of forming the recursive function. To reach position i, a person has to jump either from i-1, i-2 or i-3 position where i is the starting position.

- **Algorithm:**
  1. Create a recursive function (*count(int n)*) which takes only one parameter.
  2. Check the base cases. If the value of n is less than 0 then return 0, and if value of n is equal to zero then return 1 as it is the starting position.
  3. Call the function recursively with values n-1, n-2 and n-3 and sum up the values that are returned, i.e. *sum = count(n-1) + count(n-2) + count(n-3).*
  4. Return the value of *sum.*
- **Implementation:**

C++ | Java | Python3 | C# | PHP | Javascript

```
// A naive recursive Java program to count number
// of ways to cover a distance with 1, 2 and 3 steps
import java.io.*;

class GFG
{
    // Function returns count of ways to cover 'dist'
    static int printCountRec(int dist)
    {
        // Base cases
        if (dist<0)
            return 0;
        if (dist==0)
            return 1;

        // Recur for all previous 3 and add the results
        return printCountRec(dist-1) +
               printCountRec(dist-2) +
               printCountRec(dist-3);
    }

    // driver program
    public static void main (String[] args)
    {
        int dist = 4;
        System.out.println(printCountRec(dist));
    }
}

// This code is contributed by Pramod Kumar
```

Output:

```
7
```

- **Complexity Analysis:**
  - **Time Complexity:** $O(3^n)$.
    The time complexity of the above solution is exponential, a close upper bound is $O(3^n)$. From each state 3, a recursive function is called. So the upper bound for n states is $O(3^n)$.
  - **Space complexity:** O(1).
    No extra space is required.

Efficient solution

- **Approach:** The idea is similar, but it can be observed that there are n states but the recursive function is called 3 ^ n times. That means that some states are called repeatedly. So the idea is to store the value of states. This can be done in two ways.
  - The first way is to keep the recursive structure intact and just store the value in a HashMap and whenever the function is called, return the value store without computing (Top-Down Approach).

- The second way is to take an extra space of size n and start computing values of states from 1, 2 .. to n, i.e. compute values of i, i+1, i+2 and then use them to calculate the value of i+3 (Bottom-Up Approach).
- Overlapping Subproblems in Dynamic Programming.
- Optimal substructure property in Dynamic Programming.
- Dynamic Programming(DP) problems

- **Algorithm:**
  1. Create an array of size n + 1 and initialize the first 3 variables with 1, 1, 2. The base cases.
  2. Run a loop from 3 to n.
  3. For each index i, compute value of ith position as *dp[i] = dp[i-1] + dp[i-2] + dp[i-3].*
  4. Print the value of dp[n], as the Count of number of ways to cover a distance.
- **Implementation:**

C++ | Java | Python3 | C# | PHP | Javascript

```
// A Dynamic Programming based Java program
// to count number of ways to cover a distance
// with 1, 2 and 3 steps
import java.io.*;

class GFG
{
    // Function returns count of ways to cover 'dist'
    static int printCountDP(int dist)
    {
        int[] count = new int[dist+1];

        // Initialize base values. There is one way to
        // cover 0 and 1 distances and two ways to
        // cover 2 distance
        count[0] = 1;
        if(dist >= 1)
            count[1] = 1;
        if(dist >= 2)
            count[2] = 2;

        // Fill the count array in bottom up manner
        for (int i=3; i<=dist; i++)
            count[i] = count[i-1] + count[i-2] + count[i-3];

        return count[dist];
    }

    // driver program
    public static void main (String[] args)
    {
        int dist = 4;
        System.out.println(printCountDP(dist));
    }
```

```
}


// This code is contributed by Pramod Kumar
```

Output :

```
7
```

- **Complexity Analysis:**
  - **Time Complexity:** O(n).
    Only one traversal of the array is needed. So Time Complexity is O(n)
  - **Space complexity:** O(n).
    To store the values in a DP O(n) extra space is needed.

More Optimal Solution

Approach: Instead of using array of size n+1 we can use array of size 3 because for calculating no of ways for a particular step we need only last 3 steps no of ways.

Algorithm:

1. Create an array of size 3 and initialize the values for step 0,1,2 as 1,1,2 (Base cases).
2. Run a loop from 3 to n(dist).
3. For each index compute the value as ways[i%3] = ways[(i-1)%3] + ways[(i-2)%3] + ways[(i-3)%3] and store its value at i%3 index of array ways. If we are computing value for index 3 then the computed value will go at index 0 because for larger indices(4 ,5,6.....) we don't need the value of index 0.
4. Return the value of ways[n%3].

Output :

```
7
```

Time Complexity : O(n)

Space Complexity : O(1)

This article is contributed by Vignesh Venkatesan. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Words: 709
Characters: 3894