Write a program to calculate the number of ways to climb stairs in Python

In this article, TipsMake.com will learn with you how to write a program to count the number of ways to climb stairs in Python.

Problem : Given n stairs, a person standing at the bottom of the stairs wants to climb to the top. This person can take 1 step or 2 steps at a time. Count the number of ways he or she can get to the top.

You can see an example in the picture. The value of n is 3 and there are 3 ways to step to the top.

Picture 1 of Write a program to calculate the number of ways to climb stairs in Python

For example:

```
Input: n = 1 Output: 1 There is only one way to climb 1 stair Input: n = 2 Outpu
```

In this article, TipsMake.com will learn with you how to write a program to count the number of ways to climb stairs in Python.

Method 1: We will use recursion to solve this problem

We can easily see the recursion in this problem. The person can get to the nth step from the n-1th step or the n-2th step. Therefore, for each of the nth steps, let's try to find the number of ways to get to the n-1 and n-2th rungs and add them to give the answer to the nth rung. With this approach, we have the following expression:

```
ways(n) = ways(n-1) + ways(n-2)
```

The above expression is actually an expression of Fibonacci numbers but one thing to note is that the value of ways(n) is equal to fibonacci(n+1).

```
ways(1) = fib(2) = 1 ways(2) = fib(3) = 2 ways(3) = fib(4) = 3
```

To better understand, let's refer to the recursion tree below:

```
Input: N = 4 fib(5) '3' /  '2' /  fib(4) fib(3) '2' /  '1' /  /  /  fib(3) fib(2
```

So we can use the function for Fibonacci numbers to find the value of ways(n). Here is sample code:

```
# Python program to count # ways to reach nth stair # Recursive function to find
? cách là = ", print countWays(s) # Contributed by Harshit Agrawal
```

Generalizing the problem

How to count the number of ways if a person can climb m stairs for a given number of m. For example m is 4, then the person can climb 1 ladder, 2 stairs, 3 stairs or 4 stairs at the same time.

To generalize the above approach, we can use the following recursive relation:

```
ways(n, m) = ways(n-1, m) + ways(n-2, m) + . ways(n-m, m)
```

In this approach, to get to the nth stair, try to climb all the stairs less n than the current stairs.

Here is sample code for this overview:

```
# A program to count the number of ways # to reach n'th stair # Recursive functio
? cách là =", countWays(s, m) # Contributed by Harshit Agrawal
```

Method 2: Memorize

We can also use dp's bottom-up approach to solve this problem.

We can create an array dp[] and initialize it with -1.

Whenever we see that a subproblem is not solved, we can call the method recursively.

Otherwise, we stop the recursion if the subproblem is solved.

The sample code is as follows:

```
# Python program to count number of # ways to reach Nth stair # A simple recursiv
?
 cách là = " + str(countWays(n))) # This code is contributed by shinjanpatra
```

Method 3: Dynamic Programming

This way uses Dynamic Programming technique to solve the problem.

We create a res[] table in a bottom-up manner using the following relationship:

```
res[i] = res[i] + res[i-j] for every (i-j) >= 0
```

Thus, the i-th index of the array will contain the number of ways needed to step to the i-th order taking into account all possible climbs (i.e. from 1 to i).

Here is sample code following this approach:

```
# A program to count the number of # ways to reach n'th stair # Recursive functio
? cách là =", countWays(s, m) # Contributed by Harshit Agrawal
```

Method 4: Using Sliding Windows

In this way, we will use the Dymanic Programming Approach more effectively.

In this approach for the ith staircase, we keep a window containing the total number of final m stairs from which we can climb the ith staircase. Instead of running an inner loop, we persist the result of the loop inside a temporary variable. We remove the elements of the previous window and add the element of the current window.

Here is sample code:

```
# Python3 program to count the number # of ways to reach n'th stair when # user c
? cách là =', countWays(n, m)) # This code is contributed by 31ajaydandge
```

Method 5: Use simple math

In this approach, we simply count the number of sets with 2. This method is only applicable if the ordering problem during step counting is not important.

```
# Here n/2 is done to count the number 2's in n # 1 is added for case where there
```

Method 6: Use the Matrix Exponentitation technique

In this way, we use Matrix Exponentitation technique to solve the problem.

The number of ways to step up the nth ladder (with consideration of the order) is equal to the sum of the number of ways to step up and turn on the n-1th ladder and the n-2th step.

This corresponds to the following repetition relationship:

```
f(n) = f(n-1) + f(n-2) f(1) = 1 f(2) = 2
```

where f(n) is the number of steps to the nth step.

Note:

```
f(1) = 1 vì ch? có 1 cách b??c lên ??nh c?u thang có 1 b?
c, n=1, {1} f(2) = 2 vì ch? có 2 cách b??c lên ??nh c?u thang có 1 b?
c, n=2, {1,1},{2}
```

It is a kind of linear recurrence relation with constant coefficients and we can solve them by matrix exponentiation method. Basically, this method finds the transformation matrix for a given recurrence relation and iteratively applies this transformation to a basis vector to find the solution.

$F(n) = C^{N-1}F(1)$

where C is the transformation matrix, F(1) is the basis vector, and F(n) is the desired solution.

Therefore, in our case the matrix C will be:

```
0 1 1 1
```

C^{N-1} can be computed using the Divide and Conquer technique, in O((K^3) Log n) where K is the size of C.

And F(1):

```
1 2
```

For example, for n=4:

$F(4) = C^3 F(1)$

$C^3 =$

```
1 2 2 3
```

Therefore, $C^3 F(1) =$

```
5 8
```

The sample code is as follows:

```
# Computes A*B # where A,B are square matrices def mul(A, B, MOD): K = len(A); C
? cách là = " , ways(n) , ""); # This code is contributed by gauravrajput1
```

Generalizing the problem

Given an array A{a1, a2, ., am} containing all valid steps, calculate the number of steps required to step up the nth, ordered ladder.

For example:

```
Input: A = [1,2] n = 4 Output: 5 Gi?i thích: Tìm s? cách ?? leo lên b?
c thang th? 4, m?i l?n có th? leo 1 ho?c 2 b??c. S?
 cách là: {1,1,1,1} {1,1,2} {1,2,1} {2,1,1} {2,2} = 5 Input: A = [2,4,5] n = 9 Ou
?i thích: Có 5 cách ?? leo lên b?c thang th? 9 v?i kh? n?ng leo 2 ho?c 4 ho
?c 5 b?c thang m?i l?n. S?
 cách là: {5,4} {4,5} {2,2,5} {2,5,2} {5,2,2} = 5
```

The number of ways to climb to the nth step is given by the following recursion relation:

Picture 2 of Write a program to calculate the number of ways to climb stairs in Python

Let K be the largest element in A.

Step 1: Calculate the basis vector F(1) (including f(1). f(K))

It can be done in O(m2K) time using the following dynamic programming approach:

Let's flip A = {2,4,5] as an example. To compute F(1) = {f(1), f(2), f(3), f(4), f(5)}, we will maintain an initial empty array and iteratively concatenate Ai with it and for each Ai we will find the number of ways to achieve [Ai-1, to Ai].

Therefore, for A = {2, 4, 5]

Let T be the original empty array:

```
Iteration 1: T = {2} n = {1,2} dp = {0,1} (Number of ways to reach n=1,2 with st
```

Note: since some values ??are already calculated (1,2 for iteration 2.) we can avoid them in the loop.

After all iterations the dp array will be: [0,1,0,2,1]

So the basis vector F(1) for A = [2,4,5] is:

```
0 1 0 2 1
```

Now that we have the basis vector F(1), it's pretty easy to compute C (the transformation matrix).

Step 2: Calculate C, transformation matrix

It's a matrix with elements Ai, i+1 = 1 and the last row contains constants.

Now, constants can be determined by the presence of that element in A.

So for A = [2,4,5] the constants will be c = [1,1,0,1,0] (Ci = 1 if K-i+1) is present in A, or 0 if 1 = i = K.

Therefore, the transformation matrix C for A = [2,4,5] is:

```
0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 1 0 1 0
```

Step 3: Calculate F(n)

To calculate F(n), the following formula is used:

$F(n) - C^{n-1}F(1)$

Now that we have C and F(1), we can use the Divide and Conquer technique to find C^{n-1} and from there find the output.

Here is sample code:

```
# Computes A*B when A,B are square matrices of equal # dimensions) def mul(A, B,
? l??
ng cách là = " ,ways(A, n)); # This code is contributed by gauravrajput1
```

Note : This approach is ideal when n is too large to use loops.

Example : You should consider this approach when 1= n = 109 and 1 = m,k = 102.

Hope this article will be useful to you.