Ladder dp

10 September 2021 00:13

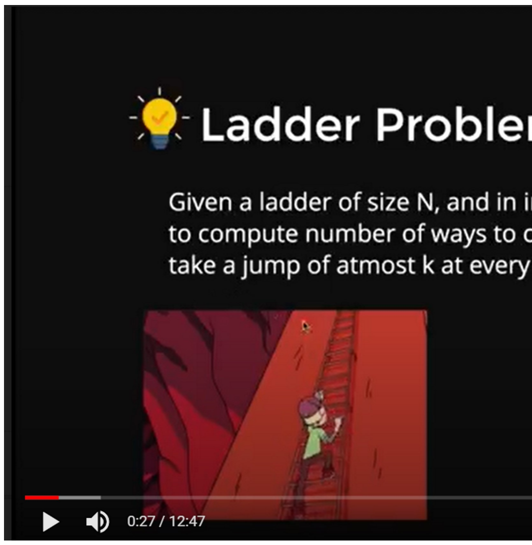
If k =3, then

Either you jump 1,2 or 3.

If Person is at "n" = It can be possible only, either he is at 1 step less, 2 step less or 3 step less.

F(n) = f(n-1) +f(n-2) + f(n-3)

For k,   F(n) = f(n-1) +f(n-2) + f(n-3) + f(n-4) +.......f(n-k)

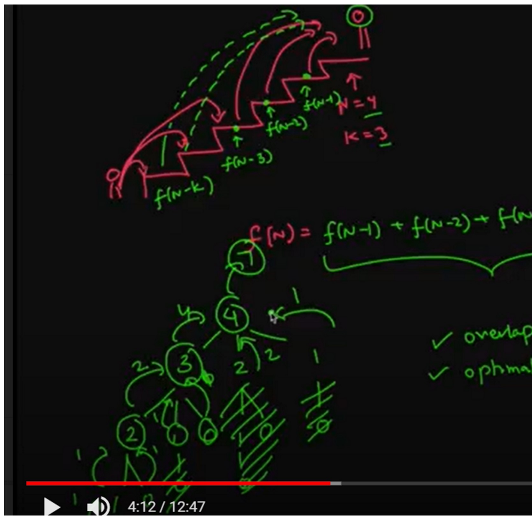


Ladder Probler

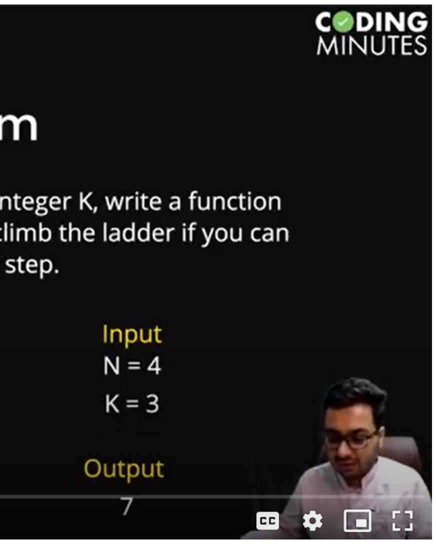
Given a ladder of size N, and in i
to compute number of ways to c
take a jump of atmost k at every

0:27 / 12:47

<u>Top Down approach</u>



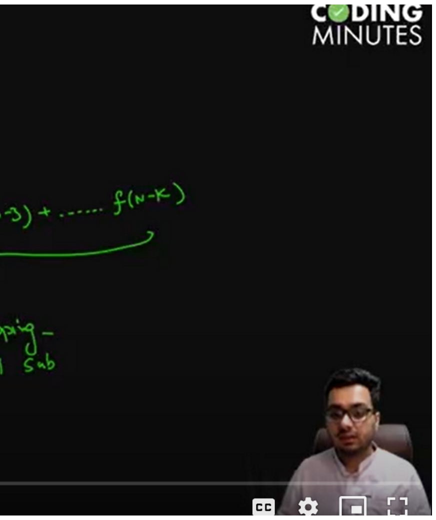
<u>Bottom up approach</u>


m

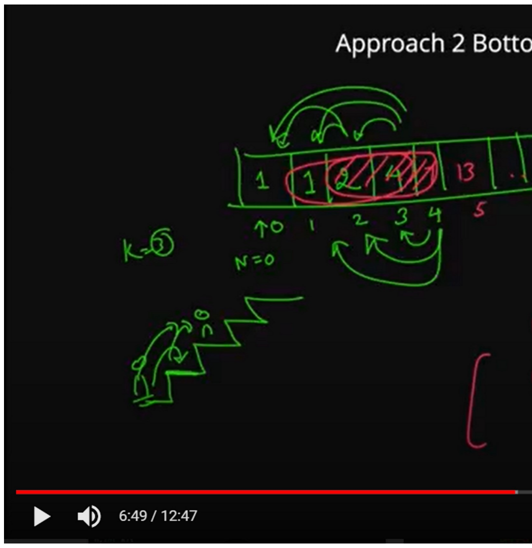
nteger K, write a function
climb the ladder if you can
step.

Input
N = 4
K = 3

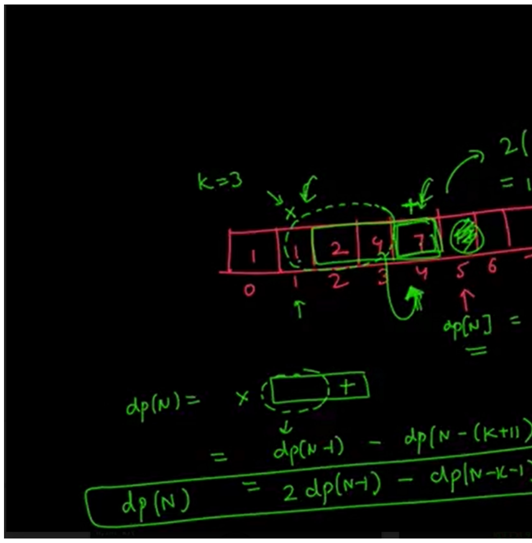
Output
7


-3) + ... f(N-K)

pping –
Sub

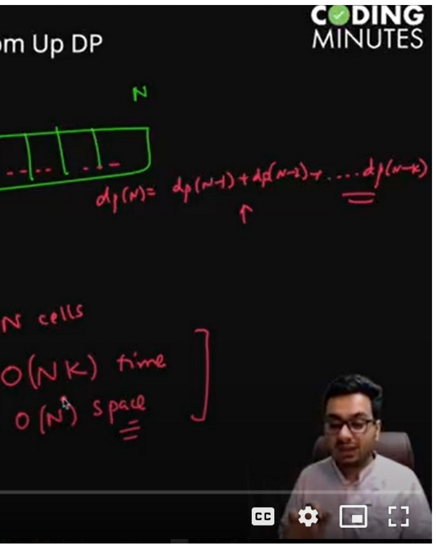


Optimisation of Bottom up b

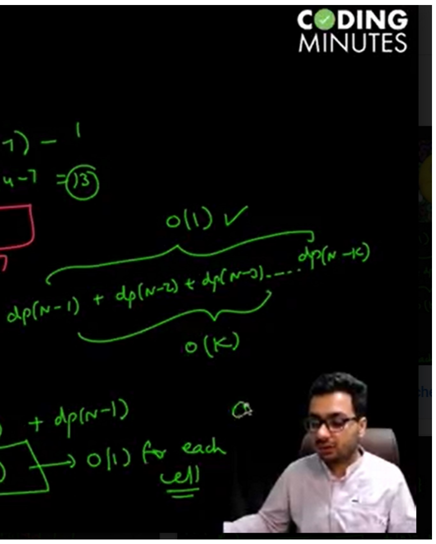


```
#include<iostream>
#include<vector>
#include<algorithm>

using namespace std;

int recursion_only(int n, int k)
```

O(2^n)



y using sliding window



```
int recursion_only(int n, int k)
{
    if(n==0) return 1;
    if(n<0)
    return 0;
    int ans=0;

    for(int i=1;i<=k;i++)
    {
        ans=ans + recursion_only(n-i,k);
    }
    return ans;
}
```

O(2^n)

The worst case is, when we at negitive stairs, answer

F(n) = f(n-1) +f(n-2) + f(n-3) + f(n-4) +.......f(n-k)

```
int dp_topdowm(int n, int k,int *dp)
{
    if(n==0) return 1;
    if(n<0)
    return 0;
    if(dp[n]!=0)
    {
        return dp[n];
    }
    int ans=0;
    for(int i=1;i<=k;i++)
    {
        ans=ans + recursion_only(n-i,k);
    }
    return dp[n]=ans;
}
```

O(n+k)

Check the answer, if the value is available in the arra

Store the new value to the array, for further refr

```
int dp_bottomup(int n, int k)
{
    vector<int>dp(n+1,0);
    dp[0]=1;
    dp[1]=1;

    for(int i=2;i<=k;i++)
    {
        dp[i]=2*dp[i-1];
    }


    for(int i=k+1;i<=n;i++)
    {
```

In bottom up - move from 0 to n......

From 2 to k, we just multiple the previous result by 2 r should be 0

y or not

ence

2 (due to analysis).

$\boxed{k = 5}$

$i_0 = \cancel{0}1$

$i_1 = 1$

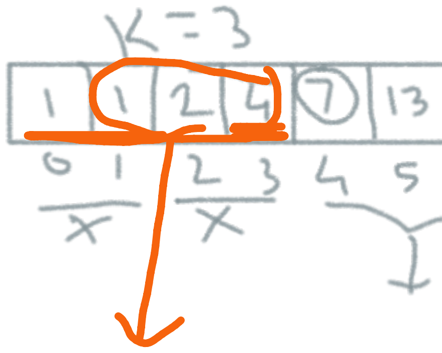
```
    for(int i=k+1;i<=n;i++)
    {
        for(int j=i-k;j<=i-1;j++)
        {
            dp[i]+=dp[j];
        }
        //cout<<"J"<<endl;
    }


    return dp[n];
}
```

From k+1 to n, we do add previous k results to the it
O(n*k)

```
int dp_bottomup_opt(int n, int k)
{
    vector<int>dp(n+1,0);
    dp[0]=1;
    dp[1]=1;

    for(int i=2;i<=k;i++)
    {
        dp[i]=2*dp[i-1];
    }


    for(int i=k+1;i<=n;i++)
    {
        dp[i]=2*dp[i-1] - dp[i-k-1];
    }


    return dp[n];
}
```

Bottom-up +sliding window.
O(n)

From 2 to k, we just multiple the previou
analysis).



We get 4 by adding 1,1,2
We want dp[i-1]+dp[i-2]+dp[i-3]

Dp[i-1] = dp[i-2]+dp[i-3]+dp[i-4]
But we don't want dp[i-4]

```
dp[i]=dp[i-1] + dp[i-1] -dp[
```

```
int main()
{
    int n;
    int k;
    cin>>n>>k;
    int dp[1000]={0};
    cout<<dp_bottomup(n,k)<<endl;
    cout<<recursion_only(n,k)<<endl;
    cout<<dp_topdowm(n,k,dp)<<endl;
    cout<<dp_bottomup_opt(n,k)<<endl;

}
```

i1 = 1

h element.

i2 = 2

i3 = 4

i5 = 8

i6 = 16 (sum of previous k)

us result by 2 (due to

Audio recording started: 21:19 10 September 2021

```
i-k-1];
```

```
}
```