# Q. Given a number x, you can do 3 different operations on x: # 1. Subtract 1 from it # 2. If it is divisible by 2, divide by 2 # 3. If it is divisible by 3, divide by 3 # Find the minimum number of steps that it takes to get to 1 using only the above operations.

*#minimum steps to 1*

*#Top down*

def minStepsToOneTD(n,dp):

    if n==1: return 0

    if n==2 or n==3: return 1

    if dp[n]!=0: return dp[n]

    div\_by\_3, div\_by\_2, less\_by\_1 = float('inf'),float('inf'),float('inf')

    if(n%3==0):

        div\_by\_3 = 1+minStepsToOneTD(n//3,dp)

    if(n%2==0):

        div\_by\_2 = 1+minStepsToOneTD(n//2,dp)

    less\_by\_1 = 1+minStepsToOneTD(n-1,dp)

    dp[n]=min(div\_by\_3, div\_by\_2, less\_by\_1)

    return dp[n]

n=7

dp=[0]\*(n+1)

print(minStepsToOneTD(n,dp))

*# Given a number x, you can do 3 different operations on x:*

*# 1. Subtract 1 from it*

*# 2. If it is divisible by 2, divide by 2*

*# 3. If it is divisible by 3, divide by 3*

*# Find the minimum number of steps that it takes to get to 1 using only the above operations.*

*#Bottom Up*

def min\_steps\_to\_one(x):

    dp = [1]\*(x+1)

    dp[1]=0

    dp[2],dp[3]=1,1

    for i in range(4,x+1):

        dp[i]=1+min(dp[i-1], dp[i//2] if i%2==0 else x, dp[i//3] if i%3==0 else x)

    return dp[x]

print(min\_steps\_to\_one(10))

# [Q. Question](https://cses.fi/problemset/task/1634/)

*#min coin, top down*

def min\_coins(n,arr,dp):

    if n==0: return 0

    if dp[n]!=0: return dp[n]

    min\_value = float('inf')

    for i in arr:

        if n-i>=0:

            min\_value = min(min\_value,1+min\_coins(n-i,arr,dp))

    dp[n]=min\_value

    return dp[n]

n=11

arr=[1,5,7]

dp=[0]\*(n+1)

print(min\_coins(n,arr,dp),dp)

*#bottom up approach*

def min\_coins(target,arr):

    dp = [float('inf')]\*(target+1)

    dp[0]=0

    for i in range(1,target+1):

        min\_val = float('inf')

        for j in arr:

            if i>=j:

                min\_val = min(min\_val, 1+dp[i-j])

        dp[i] = min\_val

    return dp[target]

target =int(input())

arr = list(map(int,input().split()))

print(min\_coins(target,arr))

Q. Find the length of longest subsequence from array which is strictly increasing.

[Question](https://cses.fi/problemset/task/1145)

*#longest subsequence*

def long\_subsequence(arr,n):

    dp=[1]\*n

    for i in range(n):

        for j in range(i):

            if arr[i]>arr[j]:

                dp[i] = max(1+dp[j], dp[i])

    return max(dp)

n = int(input())

arr=list(map(int, input().split()))

print(long\_subsequence(arr,n))

Q. [Question](https://atcoder.jp/contests/dp/tasks/dp_k)

*#K-stones*

N,K = map(int,input().split())

a=list(map(int,input().split()))

dp=[0]\*(K+1)

for i in range(1,K+1):

    for j in range(N):

        if i-a[j]>=0 and dp[i-a[j]]==0:

            dp[i]=1

if dp[K]==1:

    print('First')

else:

    print('Second')

# Q. Find the length of longest common subsequence of 2 string.

*#Longest common subsequence*

def longest\_common\_subsequence(s1,s2):

    n,m = len(s1),len(s2)

    dp = [[0 for i in range(m+1)] for j in range(n+1)]

    for i in range(n):

        for j in range(m):

            if s1[i] == s2[j]:

                dp[i+1][j+1] = dp[i][j]+1 //i+1 and j+1,dp[0][\*] represent empty string

            else: //check from diagram

                dp[i+1][j+1] = max(dp[i+1][j],dp[i][j+1])

    return dp[n][m]

print(longest\_common\_subsequence("axby","abcy"))

## Q. [k-Ordered LCS](https://www.hackerearth.com/problem/algorithm/mancunian-and-k-ordered-lcs-e6a4b8c6/)

# Q. [Dice Combination](https://cses.fi/problemset/task/1633/)

*#CSES DP PROBLEM 1. DICE COMBINATIONS*

iin=lambda: map(int,input().split())

il=lambda: list(map(int,input().split()))

ii=lambda: int(input())

mod=10\*\*9+7

n=ii()

dp=[0]\*(n+1)

dp[0]=dp[1]=1

for i in range(2,n+1):

    for j in range(1,7):

        if(i>=j):

            dp[i]+=dp[i-j]%mod

        dp[i]%=mod

print(dp[n]%mod)

*#CSES DP PROBLEM 3. Coin Combinations I*

iin=lambda: map(int,input().split())

il=lambda: list(map(int,input().split()))

ii=lambda: int(input())

mod=10\*\*9+7

n,x=iin()

arr = il()

dp=[0]\*(x+1)

dp[0]=1

for i in range(1,x+1):

    for j in arr:

        if(i>=j):

            dp[i]+=dp[i-j]%mod

        dp[i]%=mod

print(dp[x]%mod)

*#CSES DP PROBLEM 4. Coin Combinations II*

iin=lambda: map(int,input().split())

il=lambda: list(map(int,input().split()))

ii=lambda: int(input())

mod=10\*\*9+7

n,x=iin()

arr = il()

dp=[0]\*(x+1)

dp[0]=1

for j in arr:

    for i in range(1,x+1):

        if(i>=j):

            dp[i]+=dp[i-j]%mod

        dp[i]%=mod

print(dp[x]%mod)

*#CSES DP PROBLEM 4. Coin Combinations II*

iin=lambda: map(int,input().split())

il=lambda: list(map(int,input().split()))

ii=lambda: int(input())

n = ii()

count=0

while(n):

    n-=int(max(str(n)))

    count+=1

print(count)

# STRIVER SERIES

## Ninja training

<https://www.codingninjas.com/studio/problems/ninja-s-training_3621003>

def ninjaTraining(n: int, points: List[List[int]]) -> int:

    def helper(points,day,last):

        if day==n:

            return 0

        maxi=0

        for i in range(3):

            if last!=i:

                maxi = max(maxi,points[day][i]+helper(points,day+1,i))

        return maxi

    return helper(points,0,-1)

BOTTOM-UP

def ninjaTraining(n: int, points: List[List[int]]) -> int:

*# DP table to memoize the solution.*

    dp = [[-1 for \_ in range(3)] for \_\_ in range(n)]

    dp[0][0] = points[0][0]

    dp[0][1] = points[0][1]

    dp[0][2] = points[0][2]

    for i in range(1,n):

        dp[i][0] = max(dp[i - 1][1], dp[i - 1][2]) + points[i][0]

        dp[i][1] = max(dp[i - 1][0], dp[i - 1][2]) + points[i][1]

        dp[i][2] = max(dp[i - 1][0], dp[i - 1][1]) + points[i][2]

    ans= max(dp[n-1][0],dp[n-1][1],dp[n-1][2])

    return ans

Q. <https://www.codingninjas.com/studio/problems/min-cost-path_842565?leftPanelTab=0>

def getMinPathSum(matrix,n,m):

    dp=[[10000001]\*(m+1) for i in range(n+1)]

    dp[1][1]=matrix[0][0]

    for i in range(1,n+1):

        for j in range(1,m+1):

            if i==1 and j==1:

                continue

            dp[i][j]=min(dp[i-1][j],dp[i-1][j-1],dp[i][j-1])+matrix[i-1][j-1]

    return dp[n][m]