Count Possible Decodings of a given Digit Sequence

Difficulty Level : Medium • Last Updated : 31 May, 2022

Let 1 represent 'A', 2 represents 'B', etc. Given a digit sequence, count the number of possible decodings of the given digit sequence.

Examples:

```
Input:  digits[] = "121"
Output: 3
// The possible decodings are "ABA", "AU", "LA"

Input: digits[] = "1234"
Output: 3
// The possible decodings are "ABCD", "LCD", "AWD"
```

An empty digit sequence is considered to have one decoding. It may be assumed that the input contains valid digits from 0 to 9 and there are no leading 0's, no extra trailing 0's, and no two or more consecutive 0's.

count of decodings as 0. We recur for two subproblems.
1) If the last digit is non-zero, recur for the remaining (n-1) digits and add the result to the total count.
2) If the last two digits form a valid character (or smaller than 27), recur for remaining (n-2) digits and add the result to the total count.

Following is the implementation of the above approach.

C++

```
// C++ implementation to count number of
// decodings that can be formed from a
// given digit sequence
#include <cstring>
#include <iostream>
using namespace std;

// recurring function to find
// ways in how many ways a
// string can be decoded of length
// greater than 0 and starting with
// digit 1 and greater.
int countDecoding(char* digits, int n)
{
    // base cases
    if (n == 0 || n == 1)
        return 1;
    if (digits[0] == '0')
        return 0;

    // for base condition "01123" should return 0
    // Initialize count
    int count = 0;

    // If the last digit is not 0,
    // then last digit must add
    // to the number of words
    if (digits[n - 1] > '0')
```

```
    // last two digits and recur
    if (digits[n - 2] == '1'
        || (digits[n - 2] == '2'
        && digits[n - 1] < '7'))
        count += countDecoding(digits, n - 2);

    return count;
}

// Given a digit sequence of length n,
// returns count of possible decodings by
// replacing 1 with A, 2 with B, ... 26 with Z
int countWays(char* digits, int n)
{
    if (n == 0 || (n == 1 && digits[0] == '0'))
        return 0;
    return countDecoding(digits, n);
}

// Driver code
int main()
{
    char digits[] = "1234";
    int n = strlen(digits);
    cout << "Count is " << countWays(digits, n);
    return 0;
}
// Modified by Atanu Sen
```

Java

```
// A naive recursive Java implementation
// to count number of decodings that
// can be formed from a given digit sequence

class GFG {

    // recurring function to find
    // ways in how many ways a
```

```
{
    // base cases
    if (n == 0 || n == 1)
        return 1;

    // for base condition "01123" should return 0
    if (digits[0] == '0')
        return 0;

    // Initialize count
    int count = 0;

    // If the last digit is not 0, then
    // last digit must add to
    // the number of words
    if (digits[n - 1] > '0')
        count = countDecoding(digits, n - 1);

    // If the last two digits form a number
    // smaller than or equal to 26,
    // then consider last two digits and recur
    if (digits[n - 2] == '1'
        || (digits[n - 2] == '2'
            && digits[n - 1] < '7'))
        count += countDecoding(digits, n - 2);

    return count;
}

// Given a digit sequence of length n,
// returns count of possible decodings by
// replacing 1 with A, 2 with B, ... 26 with Z
static int countWays(char[] digits, int n)
{
    if (n == 0 || (n == 1 && digits[0] == '0'))
        return 0;
    return countDecoding(digits, n);
}

// Driver code
public static void main(String[] args)
```

```
                countWays(digits, n));
    }
}

// This code is contributed by Smitha Dinesh Semwal.
// Modified by Atanu Sen
```

Python3

```
# Recursive implementation of numDecodings
def numDecodings(s: str) -> int:
    if len(s) == 0
    or (len(s) == 1
        and s[0] == '0'):
        return 0
    return numDecodingsHelper(s, len(s))


def numDecodingsHelper(s: str, n: int) -> int:
    if n == 0 or n == 1:
        return 1
    count = 0
    if s[n-1] > "0":
        count = numDecodingsHelper(s, n-1)
    if (s[n - 2] == '1'
        or (s[n - 2] == '2'
            and s[n - 1] < '7')):
        count += numDecodingsHelper(s, n - 2)
    return count


# Driver code
digits = "1234"
print("Count is ", numDecodings(digits))
# This code is contributed by Frank Hu
```

C#

```
class GFG {

    // recurring function to find
    // ways in how many ways a
    // string can be decoded of length
    // greater than 0 and starting with
    // digit 1 and greater.
    static int countDecoding(char[] digits, int n)
    {

        // base cases
        if (n == 0 || n == 1)
            return 1;

        // Initialize count
        int count = 0;

        // If the last digit is not 0, then
        // last digit must add to
        // the number of words
        if (digits[n - 1] > '0')
            count = countDecoding(digits, n - 1);

        // If the last two digits form a number
        // smaller than or equal to 26, then
        // consider last two digits and recur
        if (digits[n - 2] == '1'
            || (digits[n - 2] == '2'
                && digits[n - 1] < '7'))
            count += countDecoding(digits, n - 2);

        return count;
    }

    // Given a digit sequence of length n,
    // returns count of possible decodings by
    // replacing 1 with A, 2 with B, ... 26 with Z
    static int countWays(char[] digits, int n)
    {
        if (n == 0 || (n == 1 && digits[0] == '0'))
            return 0;
```

```
    public static void Main()
    {
        char[] digits = { '1', '2', '3', '4' };
        int n = digits.Length;
        Console.Write("Count is ");
        Console.Write(countWays(digits, n));
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
<?php
// A naive recursive PHP implementation
// to count number of decodings that can
// be formed from a given digit sequence

//recurring function to find
//ways in how many ways a
//string can be decoded of length
//greater than 0 and starting with
//digit 1 and greater.
function countDecoding(&$digits, $n)
{
    // base cases
    if ($n == 0 || $n == 1)
        return 1;

    $count = 0; // Initialize count

    // If the last digit is not 0, then last
    // digit must add to the number of words
    if ($digits[$n - 1] > '0')
        $count = countDecoding($digits, $n - 1);

    // If the last two digits form a number
    // smaller than or equal to 26, then
    // consider last two digits and recur
    if ($digits[$n - 2] == '1' ||
```

```
    return $count;
}

// Given a digit sequence of length n,
// returns count of possible decodings by
// replacing 1 with A, 2 with B, ... 26 with Z
function countWays(&$digits, $n){
  if($n==0 || ($n == 1 && $digits[0] == '0'))
     return 0;
  return countDecoding($digits, $n);
}

// Driver Code
$digits = "1234";
$n = strlen($digits);
echo "Count is " . countWays($digits, $n);

// This code is contributed by ita_c
?>
```

Javascript

```
<script>

// A naive recursive JavaScript implementation
// to count number of decodings that
// can be formed from a given digit sequence

    // recurring function to find
    // ways in how many ways a
    // string can be decoded of length
    // greater than 0 and starting with
    // digit 1 and greater.
    function countDecoding(digits, n)
    {
        // base cases
        if (n == 0 || n == 1)
        {
            return 1;
```

```
        return 0;
    }

    // Initialize count
    let count = 0;

    // If the last digit is not 0, then
    // last digit must add to
    // the number of words
    if (digits[n - 1] > '0')
    {
        count = countDecoding(digits, n - 1);
    }
    // If the last two digits form a number
    // smaller than or equal to 26,
    // then consider last two digits and recur
    if (digits[n - 2] == '1'
        || (digits[n - 2] == '2'
            && digits[n - 1] < '7'))
    {
        count += countDecoding(digits, n - 2);
    }
    return count;
}
// Given a digit sequence of length n,
// returns count of possible decodings by
// replacing 1 with A, 2 with B, ... 26 with Z
function countWays(digits, n)
{
    if (n == 0 || (n == 1 && digits[0] == '0'))
    {
        return 0;
    }
    return countDecoding(digits, n);
}

// Driver code
digits=['1', '2', '3', '4'];
let n = digits.length;
document.write("Count is ",countWays(digits, n));
```

Output:

```
Count is 3
```

The time complexity of above the code is exponential. If we take a closer look at the above program, we can observe that the recursive solution is similar to Fibonacci Numbers. Therefore, we can optimize the above solution to work in O(n) time using Dynamic Programming.

Following is the implementation for the same.

C++

```
// A Dynamic Programming based C++
// implementation to count decodings
#include <iostream>
#include <cstring>
using namespace std;

// A Dynamic Programming based function
// to count decodings
int countDecodingDP(char *digits, int n)
{
    // A table to store results of subproblems
    int count[n+1];
    count[0] = 1;
    count[1] = 1;
    //for base condition "01123" should return 0
    if(digits[0]=='0')
         return 0;
    for (int i = 2; i <= n; i++)
    {
        count[i] = 0;

        // If the last digit is not 0,
        // then last digit must add to the number of words
        if (digits[i-1] > '0')
```

```
        // then last two digits form a valid character
        if (digits[i-2] == '1' ||
              (digits[i-2] == '2' && digits[i-1] < '7') )
            count[i] += count[i-2];
    }
    return count[n];
}

// Driver program to test above function
int main()
{
    char digits[] = "1234";
    int n = strlen(digits);
    cout << "Count is " << countDecodingDP(digits, n);
    return 0;
}
// Modified by Atanu Sen
```

Java

```
// A Dynamic Programming based Java
// implementation to count decodings
import java.io.*;

class GFG
{

// A Dynamic Programming based
// function to count decodings
static int countDecodingDP(char digits[],
                           int n)
{
    // A table to store results of subproblems
    int count[] = new int[n + 1];
    count[0] = 1;
    count[1] = 1;
    if(digits[0]=='0')   //for base condition "01123" should return 0
          return 0;
    for (int i = 2; i <= n; i++)
    {
```

```
        // the number of words
        if (digits[i - 1] > '0')
            count[i] = count[i - 1];

        // If second last digit is smaller
        // than 2 and last digit is smaller
        // than 7, then last two digits
        // form a valid character
        if (digits[i - 2] == '1' ||
           (digits[i - 2] == '2' &&
            digits[i - 1] < '7'))
            count[i] += count[i - 2];
    }
    return count[n];
}

// Driver Code
public static void main (String[] args)
{
    char digits[] = {'1','2','3','4'};
    int n = digits.length;
    System.out.println("Count is " +
               countDecodingDP(digits, n));
}
}

// This code is contributed by anuj_67
// Modified by Atanu Sen
```

Python3

```
# A Dynamic Programming based Python3
# implementation to count decodings

# A Dynamic Programming based function
# to count decodings
def countDecodingDP(digits, n):

    count = [0] * (n + 1); # A table to store
```

```
        count[i] = 0;

        # If the last digit is not 0, then last
        # digit must add to the number of words
        if (digits[i - 1] > '0'):
            count[i] = count[i - 1];

        # If second last digit is smaller than 2
        # and last digit is smaller than 7, then
        # last two digits form a valid character
        if (digits[i - 2] == '1' or
           (digits[i - 2] == '2' and
            digits[i - 1] < '7') ):
            count[i] += count[i - 2];

    return count[n];

# Driver Code
digits = "1234";
n = len(digits);
print("Count is" ,
       countDecodingDP(digits, n));

# This code is contributed by mits
```

C#

```
// A Dynamic Programming based C#
// implementation to count decodings
using System;

class GFG
{

// A Dynamic Programming based
// function to count decodings
static int countDecodingDP(char[] digits,
                           int n)
```

```
    count[1] = 1;

    for (int i = 2; i <= n; i++)
    {
        count[i] = 0;

        // If the last digit is not 0,
        // then last digit must add to
        // the number of words
        if (digits[i - 1] > '0')
            count[i] = count[i - 1];

        // If second last digit is smaller
        // than 2 and last digit is smaller
        // than 7, then last two digits
        // form a valid character
        if (digits[i - 2] == '1' ||
           (digits[i - 2] == '2' &&
            digits[i - 1] < '7'))
            count[i] += count[i - 2];
    }
    return count[n];
}

// Driver Code
public static void Main()
{
    char[] digits = {'1','2','3','4'};
    int n = digits.Length;
    Console.WriteLine("Count is " +
            countDecodingDP(digits, n));
}
}

// This code is contributed
// by Akanksha Rai
// Modified by Atanu Sen
```

PHP

```php
// A Dynamic Programming based function to count decodings
function countDecodingDP($digits, $n)
{
    // A table to store results of subproblems
    $count[$n+1]=array();
    $count[0] = 1;
    $count[1] = 1;

    for ($i = 2; $i <= $n; $i++)
    {
        $count[$i] = 0;

        // If the last digit is not 0, then last digit must add to
        // the number of words
        if ($digits[$i-1] > '0')
            $count[$i] = $count[$i-1];

        // If second last digit is smaller than 2 and last digit is
        // smaller than 7, then last two digits form a valid character
        if ($digits[$i-2] == '1' || ($digits[$i-2] == '2' && $digits[$i-
            $count[$i] += $count[$i-2];
    }
    return $count[$n];
}

// Driver program to test above function
    $digits = "1234";
    $n = strlen($digits);
    echo   "Count is " , countDecodingDP($digits, $n);

#This code is contributed by ajit.
?>
```

Javascript

```
<script>

// A Dynamic Programming based Javascript
// implementation to count decodings
```

```
{

    // A table to store results of subproblems
    let count = new Array(n + 1);
    count[0] = 1;
    count[1] = 1;

    // For base condition "01123" should return 0
    if (digits[0] == '0')
          return 0;

    for(let i = 2; i <= n; i++)
    {
        count[i] = 0;

        // If the last digit is not 0,
        // then last digit must add to
        // the number of words
        if (digits[i - 1] > '0')
            count[i] = count[i - 1];

        // If second last digit is smaller
        // than 2 and last digit is smaller
        // than 7, then last two digits
        // form a valid character
        if (digits[i - 2] == '1' ||
           (digits[i - 2] == '2' &&
            digits[i - 1] < '7'))
            count[i] += count[i - 2];
    }
    return count[n];
}

// Driver Code
let digits = [ '1','2','3','4' ];
let n = digits.length;

document.write("Count is " +
               countDecodingDP(digits, n));

// This code is contributed by rag2127
```

Output:

```
Count is 3
```

Time Complexity of the above solution is O(n) and it requires O(n) auxiliary space. We can reduce auxiliary space to O(1) by using the space-optimized version discussed in the Fibonacci Number Post. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Method 3 : (Top Down DP)

Approach :

The above problem can be solved using Top down DP in the following way . One of the basic intuition is that we need to find the total number of ways to decode the given string such that each and every number in the string must lie in between the range of [1 , 26] both inclusive and without any leading 0's . Let us consider an example string .

str = "123"

If we observe carefully we can observe a pattern over here i.e., the number of ways a particular substring can be decoded depends on the number of ways the remaining string is going to be decoded . For example , we want the number of ways to decode the string with "1" as a prefix the result depends on the number of ways the remaining string, i.e., "23" can be decoded . The number of ways the string "23" can be decoded are "2" , "3" and "23" there are 2 ways in both of these cases we can just append "1" to get the number of ways the given string can be decoded with "1" as a prefix i.e., "1" , "2" , "3" and "1" , "23" . Now we have found the number of ways we can decode the given string with "1" as a

depends on the result on how the remaining string is decoded . Here the remaining string is “3” it can be decoded in only 1 way so we can just append “12” in front of the string “3” to get it i.e., “12” , “3” . So the total number of ways the given string can be decoded are 3 ways .

But we can see some of the overlapping of subproblems over here i.e., when we are computing the total number of ways to decode the string “23” we are computing the number of ways the string “3” can be decoded as well as when we are computing the number of ways the string “12” can be decoded we are again computing the number of ways the string “3” can be decoded . So we can avoid this by storing the result of every substring . Here we can identify each and every sub problem through the index of the string . So , if at any point of time if we have already computed the number of ways the substring can be decoded we can directly return the result and that leads to a lot of optimization .

Below is the C++ implementation

C++

```
#include<bits/stdc++.h>
using namespace std;

int mod = 1e9 + 7;

// function which returns the number of ways to decode the message
int decodeMessage(vector<int> &dp,int s,string &str,int n)
{
    // an empty string can also form 1 valid decoding
    if(s >= n)
        return 1;

    /*
        if we have already computed the number of
```

```
        return dp[s];

    int num,tc;
    num = tc = 0;
    for(int i=s;i<n;i++)
    {
        // generate the number
        num = num*10 + (str[i] - '0');

        // validate the number
        if(num >= 1 and num <= 26)
        {
            /*
                since the number of ways to decode any string
                depends on the result of
                how the remaining string is decoded so get the
                number of ways how the rest of the string can
                be decoded
            */
            int c = decodeMessage(dp,i+1,str,n);

            // add all the ways that the substring
            // from the current index can be decoded
            tc = (tc%mod + c%mod)%mod;
        }

        // leading 0's or the number
        // generated so far is greater than 26
        // we can just stop the process
        // as it can never be a part of our solution
        else
            break;
    }

    // store all the possible decodings and return the result
    return (dp[s] = tc);
}
int CountWays(string str)
{
    int n = str.size();
```

```
    // dp vector to store the  number of ways
    // to decode each and every substring
    vector<int> dp(n,-1);

    // return the result
    return decodeMessage(dp,0,str,n);
}
int main()
{
    string str = "1234";
    cout << CountWays(str) << endl;
    return 0;
}
```

Java

```
/*package whatever //do not write package name here */
import java.io.*;

class GFG {
  static int mod = 1000000007;

  // function which returns the number of ways to decode the message
  static int decodeMessage(int[] dp, int s, String str, int n)
  {

    // an empty string can also form 1 valid decoding
    if(s >= n)
      return 1;

    /*
        if we have already computed the number of
        ways to decode the substring return the
        answer directly
    */
    if(dp[s] != -1)
      return dp[s];

    int num,tc;
    num = tc = 0;
```

```
    // validate the number
    if(num >= 1 && num <= 26)
    {
      /*
              since the number of ways to decode any string
              depends on the result of
              how the remaining string is decoded so get the
              number of ways how the rest of the string can
              be decoded
          */
      int c = decodeMessage(dp, i + 1, str, n);

      // add all the ways that the substring
      // from the current index can be decoded
      tc = (tc%mod + c%mod)%mod;
    }

    // leading 0's or the number
    // generated so far is greater than 26
    // we can just stop the process
    // as it can never be a part of our solution
    else
      break;
  }

  // store all the possible decodings and return the result
  return (dp[s] = tc);
}
static int CountWays(String str)
{
  int n = str.length();

  // empty string can form 1 valid decoding
  if(n == 0)
    return 1;

  // dp vector to store the number of ways
  // to decode each and every substring
  int[] dp = new int[n];
  for(int i = 0; i < n; i++){
```

```
    return decodeMessage(dp,0,str,n);
  }

  // Driver Code
  public static void main(String args[])
  {
    String str = "1234";
    System.out.println(CountWays(str));
  }
}

// This code is contributed by shinjanpatra
```

Python3

```
mod = 1e9 + 7

# function which returns the number of ways to decode the message
def decodeMessage(dp, s, str, n):

    # an empty string can also form 1 valid decoding
    if(s >= n):
        return 1

        # if we have already computed the number of
        # ways to decode the substring return the
        # answer directly

    if(dp[s] != -1):
        return dp[s]

    num = 0
    tc = 0
    for i in range(s,n):
        # generate the number
        num = num*10 + (ord(str[i]) - ord('0'))

        # validate the number
        if(num >= 1 and num <= 26):
```

```
                # be decoded
            c = decodeMessage(dp, i + 1, str, n)

            # add all the ways that the substring
            # from the current index can be decoded
            tc = int((tc%mod + c%mod)%mod)

        # leading 0's or the number
        # generated so far is greater than 26
        # we can just stop the process
        # as it can never be a part of our solution
        else:
            break

    # store all the possible decodings and return the result
    dp[s] = tc
    return dp[s]

def CountWays(str):

    n = len(str)

    # empty string can form 1 valid decoding
    if(n == 0):
        return 1

    # dp vector to store the  number of ways
    # to decode each and every substring
    dp = [-1]*(n)

    # return the result
    return decodeMessage(dp, 0, str, n)

# driver code
if __name__ == "__main__" :

   str = "1234"
   print(CountWays(str))

  # This code is contributed by shinjanpatra.
```

```
// the above approach
using System;

class GFG
{

  static int mod = 1000000007;

  // function which returns the number of ways to decode the message
  static int decodeMessage(int[] dp, int s, string str, int n)
  {

    // an empty string can also form 1 valid decoding
    if(s >= n)
      return 1;

    /*
        if we have already computed the number of
        ways to decode the substring return the
        answer directly
    */
    if(dp[s] != -1)
      return dp[s];

    int num,tc;
    num = tc = 0;
    for(int i=s;i<n;i++)
    {
      // generate the number
      num = num*10 + ((int)str[i] - '0');

      // validate the number
      if(num >= 1 && num <= 26)
      {
        /*
                since the number of ways to decode any string
                depends on the result of
                how the remaining string is decoded so get the
                number of ways how the rest of the string can
                be decoded
            */
```

```
            tc = (tc%mod + c%mod)%mod;
          }

          // leading 0's or the number
          // generated so far is greater than 26
          // we can just stop the process
          // as it can never be a part of our solution
          else
            break;
        }

        // store all the possible decodings and return the result
        return (dp[s] = tc);
      }
      static int CountWays(string str)
      {
        int n = str.Length;

        // empty string can form 1 valid decoding
        if(n == 0)
          return 1;

        // dp vector to store the number of ways
        // to decode each and every substring
        int[] dp = new int[n];
        for(int i = 0; i < n; i++){
          dp[i] = -1;
        }

        // return the result
        return decodeMessage(dp,0,str,n);
      }

      // Driver Code
      public static void Main()
      {
        string str = "1234";
        Console.Write(CountWays(str));
      }
    }
```

```
<script>

const mod = 1e9 + 7;

// function which returns the number of ways to decode the message
function decodeMessage(dp,s,str,n)
{
    // an empty string can also form 1 valid decoding
    if(s >= n)
        return 1;

    /*
        if we have already computed the number of
        ways to decode the substring return the
        answer directly
    */
    if(dp[s] != -1)
        return dp[s];

    let num,tc;
    num = tc = 0;
    for(let i=s;i<n;i++)
    {
        // generate the number
        num = num*10 + (str.charCodeAt(i) - '0'.charCodeAt(0));

        // validate the number
        if(num >= 1 && num <= 26)
        {
            /*
                since the number of ways to decode any string
                depends on the result of
                how the remaining string is decoded so get the
                number of ways how the rest of the string can
                be decoded
            */
            let c = decodeMessage(dp,i+1,str,n);

            // add all the ways that the substring
            // from the current index can be decoded
```

```
            // generated so far is greater than 26
            // we can just stop the process
            // as it can never be a part of our solution
            else
                break;
        }

        // store all the possible decodings and return the result
        return (dp[s] = tc);
    }
    function CountWays(str)
    {
        let n = str.length;

        // empty string can form 1 valid decoding
        if(n == 0)
            return 1;

        // dp vector to store the  number of ways
        // to decode each and every substring
        let dp = new Array(n).fill(-1);

        // return the result
        return decodeMessage(dp,0,str,n);
    }
    // driver code

    let str = "1234";
    document.write(CountWays(str),"</br>");

    // This code is contributed by shinjanpatra.
    </script>
```

Output :

3

Time Complexity : O (N) where N is the length of the string . As we are solving each and every sub – problem only once .

Next

Program for Fibonacci numbers

RECOMMENDED ARTICLES

Page : 1 2 3

01 Count Possible Decodings of a given Digit Sequence in O(N) time and Constant Auxiliary space
09, May 20

02 Count possible decodings of a given Digit Sequence | Set 2
20, Jun 20

03 Count possible decoding of a given digit sequence with hidden characters
24, Jun 21

05 Count of N-digit numbers having digit XOR as single digit
13, Aug 20

06 Count N-digit numbers that contains every possible digit atleast once
30, Jun 21

07 Count of N digit numbers possible which satisfy the given conditions
11, Jul 19

number of times consecutively
13, Jan 21

condition
06, Feb 19

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : nitin mittal, vt_m, ukasp, Akanksha_Rai, jit_t, Mithun Kumar, atanusenstudy, fhu004, nirmitjain, avanitrachhadiya2155, rag2127, sumitgumber28, anveshkarra1234, shinjanpatra, sanjoy_62, simmytarika5

Article Tags : Amazon, Facebook, Fibonacci, Goldman Sachs, Linkedin, MakeMyTrip, Morgan Stanley, Dynamic Programming, Mathematical

Practice Tags : Morgan Stanley, Amazon, MakeMyTrip, Goldman Sachs, Linkedin, Facebook, Dynamic Programming, Mathematical, Fibonacci

Improve Article | Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

NodeJS