Count Possible Decodings of a given Digit Sequence

A top secret message containing letters from A-Z is being encoded to numbers using the following mapping:

'A' -> 1

'B' -> 2

...

'Z' -> 26

You are an FBI agent. You have to determine the total number of ways that message can be decoded.  
**Note:** 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 If there are leading 0’s, extra trailing 0’s and two or more consecutive 0’s then it is an invalid string.

**Example :**  
Given encoded message "123",  it could be decoded as "ABC" (1 2 3) or "LC" (12 3) or "AW"(1 23).  
So total ways are 3.

**Input:**  
First line contains the test cases T.  1<=T<=1000  
Each test case have two lines  
First is length of string N.  1<=N<=40  
Second line is string S of digits from '0' to '9' of N length.

**Example:  
Input:**  
2  
3  
123  
4  
2563  
**Output:**  
3  
2

**Tip: concept like fibonacii numbers**

int find(string s,int i,int &flag)

{

int n=s.length();

// cout<<"-----i: "<<i<<" ---\n";

if(i==n)

return 1;

if(s[i]=='0' && (i==0 || s[i-1]=='0' || s[i-1]>'2'))

{

//cout<<"f1\n";

flag=1;

return 0;

}

int l=0,r=0;

if(s[i]>'0')

l=find(s,i+1,flag);

if(i<n-1)

{

string tem=s.substr(i,2);

string t1="27",t2="0";

// watch(tem);

if(tem<t1 && tem[0]!='0')

{

// watch(tem);

r=find(s,i+2,flag);

}

}

// cout<<"i l r: "<<i<<" "<<l<<" "<<r<<"\n";

// cout<<"-----\n";

return l+r;

}

int32\_t main(){

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL);

cout.tie(NULL);

int t;

cin>>t;

while(t--){

int n,i,j,flag=0;

string s;

cin>>n;

cin>>s;

int ans=find(s,0,flag);

if(flag==1)

cout<<"0";

else

see(ans);

hh

}

}

Editorial:

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.

This problem is recursive and can be broken in sub-problems. We start from end of the given digit sequence. We initialize the total count of decodings as 0. We recur for two subproblems.  
1) If the last digit is non-zero, recur for remaining (n-1) digits and add the result to 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 total count.

|  |
| --- |
| // A naive recursive Java implementation  // to count number of decodings that  // can be formed from a given digit sequence    class GFG {    // Given a digit sequence of length n,  // returns count of possible decodings by  // replacing 1 with A, 2 woth B, ... 26 with Z  static int countDecoding(char[] digits, int n)  {      // 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;  }    // Driver program to test above function  public static void main(String[] args)  {      char digits[] = {'1', '2', '3', '4'};      int n = digits.length;      System.out.printf("Count is %d", countDecoding(digits, n));  }  }      // This code is contributed by Smitha Dinesh Semwal.  // Modified by Atanu Sen |

**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](https://www.geeksforgeeks.org/program-for-nth-fibonacci-number/). Therefore, we can optimize the above solution to work in O(n) time using [Dynamic Programming](https://www.geeksforgeeks.org/tag/dynamic-programming/).  
Following is implementation for the same.

|  |
| --- |
| // 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')              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 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 |

**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 space optimized version discussed in the [Fibonacci Number Post](https://www.geeksforgeeks.org/program-for-nth-fibonacci-number/).