Digit DP | Introduction - GeeksforGeeks

Source: https://www.geeksforgeeks.org/digit-dp-introduction/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Digit DP | Introduction Last Updated : 17 Jan, 2023 Prerequisite : How to solve a Dynamic Programming Problem ? There are many types of problems that ask to count the number of integers ' x ' between two integers say ' a ' and ' b ' such that x satisfies a specific property that can be related to its digits. So, if we say G(x) tells the number of such integers between 1 to x (inclusively), then the number of such integers between a and b can be given by G(b) - G(a-1) . This is when Digit DP (Dynamic Programming) comes into action. All such integer counting problems that satisfy the above property can be solved by digit DP approach. Key Concept: Let given number x has n digits. The main idea of digit DP is to first represent the digits as an array of digits t[]. Let's say a we have t n t n-1 t n-2 ... t 2 t 1 as the decimal representation where t i (0 < i <= n) tells the i-th digit from the right. The leftmost digit t n is the most significant digit. Now, after representing the given number this way we generate the numbers less than the given number and simultaneously calculate using DP, if the number satisfy the given property. We start generating integers having number of digits = 1 and then till number of digits = n. Integers having less number of digits than n can be analyzed by setting the leftmost digits to be zero. Example Problem : Given two integers a and b . Your task is to print the sum of all the digits appearing in the integers between a and b. For example if a = 5 and b = 11, then answer is 38 (5 + 6 + 7 + 8 + 9 + 1 + 0 + 1 + 1) Constraints : 1 <= a < b <= 10^18 Now we see that if we have calculated the answer for state having n-1 digits, i.e., t n-1 t n-2 ... t 2 t 1 and we need to calculate answer for state having n digits t n t n-1 t n-2 ... t 2 t 1 . So, clearly, we can use the result of the previous state instead of re-calculating it. Hence, it follows the overlapping property. Let's think for a state for this DP Our DP state will be dp(idx, tight, sum) 1) idx It tells about the index value from right in the given integer 2) tight This will tell if the current digits range is restricted or not. If the current digit's range is not restricted then it will span from 0 to 9 (inclusively) else it will span from 0 to digit[idx] (inclusively). Example : consider our limiting integer to be 3245 and we need to calculate G(3245) index : 4 3 2 1 digits : 3 2 4 5 Unrestricted range: Now suppose the integer generated till now is : 3 1 * * (* is empty place, where digits are to be inserted to form the integer). index : 4 3 2 1 digits : 3 2 4 5 generated integer: 3 1 _ _ here, we see that index 2 has unrestricted range. Now index 2 can have digits from range 0 to 9(inclusively). For unrestricted range tight = 0 Restricted range: Now suppose the integer generated till now is : 3 2 * * ('*' is an empty place, where digits are to be inserted to form the integer). index : 4 3 2 1 digits : 3 2 4 5 generated integer: 3 2 _ _ here, we see that index 2 has a restricted range. Now index 2 can only have digits from range 0 to 4 (inclusively) For restricted range tight = 1 3) sum This parameter will store the sum of digits in the generated integer from msd to idx. Max value for this parameter sum can be 9*18 = 162, considering 18 digits in the integer State Relation: The basic idea for state relation is very simple. We formulate the dp in top-down fashion. Let's say we are at the msd having index idx. So initially the sum will be 0. Therefore, we will fill the digit at index by the digits in its range. Let's say its range is from 0 to k (k<=9, depending on the tight value) and fetch the answer from the next state having index = idx-1 and sum = previous sum + digit chosen. int ans = 0; for (int i=0; i<=k; i++) { ans += state(idx-1, newTight, sum+i) }

state(idx,tight,sum) = ans; How to calculate the newTight value? The new tight value from a state depends on its previous state. If tight value form the previous state is 1 and the digit at idx chosen is digit[idx](i.e the digit at idx in limiting integer) , then only our new tight will be 1 as it only then tells that the number formed till now is prefix of the limiting integer. // digitTaken is the digit chosen // digit[idx] is

the digit in the limiting // integer at index idx from right // previouTight is the tight value form previous // state

newTight = previousTight & (digitTake == digit[idx]) Below is the implementation of the above approach CPP // Given two integers a and b. The task is to print // sum of all the digits appearing in the // integers between a and b #include "bits/stdc++.h" using namespace std ; // Memoization for the state results long long dp [20][180][2]; // Stores the digits in x in a vector digit void getDigits (long long x , vector < int > & digit) { while (x) { digit . push_back (x % 10); x /= 10 ; } } // Return sum of digits from 1 to integer in // digit vector long long digitSum (int idx , int sum , int tight , vector < int > & digit) { // base case if (idx == -1) return sum ; // checking if already calculated this state if (dp [idx][sum][tight] != -1 and tight != 1) return dp [idx][sum][tight]; long long ret = 0 ; // calculating range value int k = (tight) ? digit [idx] : 9 ; for (int i = 0 ; i <= k ; i ++) { // calculating newTight value for next state int newTight = (digit [idx] == i) ? tight : 0 ; // fetching answer from next state ret += digitSum (idx -1 , sum + i , newTight , digit); } if (! tight) dp [idx][sum][tight] = ret ; return ret ; } // Returns sum of digits in numbers from a to b. int rangeDigitSum (int a , int b) { // initializing dp with -1 memset (dp , -1 , sizeof (dp)); // storing digits of a-1 in digit vector vector < int > digitA ; getDigits (a -1 , digitA); // Finding sum of digits from 1 to "a-1" which is passed // as digitA. long long ans1 = digitSum (digitA . size () -1 , 0 , 1 , digitA); // Storing digits of b in digit vector vector < int > digitB ; getDigits (b , digitB); // Finding sum of digits from 1 to "b" which is passed // as digitB. long long ans2 = digitSum (digitB . size () -1 , 0 , 1 , digitB); return (ans2 - ans1); } // driver function to call above function int main () { long long a = 123 , b = 1024 ; cout << "digit sum for given range : " << rangeDigitSum (a , b) << endl ; return 0 ; } Java // Java program for above approach import java.util.ArrayList ; import java.util.Arrays ; // Given two integers a and b. The task is to print // sum of all the digits appearing in the // integers between a and b public class GFG { // Memoization for the state results static long dp [][][] = new long [20][180][2] ; // Stores the digits in x in a vector digit static void getDigits (long x , ArrayList < Integer > digit) { while (x != 0) { digit . add ((int)(x % 10)); x /= 10 ; } } // Return sum of digits from 1 to integer in // digit vector static long digitSum (int idx , int sum , int tight , ArrayList < Integer > digit) { // base case if (idx == - 1) return sum ; // checking if already calculated this state if (dp [idx][sum][tight] != - 1 && tight != 1) return dp [idx][sum][tight] ; long ret = 0 ; // calculating range value int k = (tight != 0) ? digit . get (idx) : 9 ; for (int i = 0 ; i <= k ; i ++) { // calculating newTight value for next state int newTight = (digit . get (idx) == i) ? tight : 0 ; // fetching answer from next state ret += digitSum (idx - 1 , sum + i , newTight , digit); } if (tight != 0) dp [idx][sum][tight] = ret ; return ret ; } // Returns sum of digits in numbers from a to b. static int rangeDigitSum (int a , int b) { // initializing dp with -1 for (int i = 0 ; i < 20 ; i ++) for (int j = 0 ; j < 180 ; j ++) for (int k = 0 ; k < 2 ; k ++) dp [i][j][k] = - 1 ; // storing digits of a-1 in digit vector ArrayList < Integer > digitA = new ArrayList < Integer > (); getDigits (a - 1 , digitA); // Finding sum of digits from 1 to "a-1" which is // passed as digitA. long ans1 = digitSum (digitA . size () - 1 , 0 , 1 , digitA); // Storing digits of b in digit vector ArrayList < Integer > digitB = new ArrayList < Integer > (); getDigits (b , digitB); // Finding sum of digits from 1 to "b" which is // passed as digitB. long ans2 = digitSum (digitB . size () - 1 , 0 , 1 , digitB); return (int)(ans2 - ans1); } // driver function to call above function public static void main (String [] args) { int a = 123 , b = 1024 ; System . out . println ("digit sum for given range : " + rangeDigitSum (a , b)); } } // This code is contributed by Lovely Jain Python3 # Given two integers a and b. The task is to # print sum of all the digits appearing in the # integers between a and b # Memoization for the state results dp = [[[- 1 for i in range (2)] for j in range (180)] for k in range (20)] # Stores the digits in x in a list digit def getDigits (x , digit): while x : digit . append (x % 10) x //= 10 # Return sum of digits from 1 to integer in digit list def digitSum (index , sumof , tight , digit): # Base case if index == - 1 : return sumof # Checking if already calculated this state if dp [index][sumof][tight] != - 1 and tight != 1 : return dp [index][sumof][tight] ret = 0 # Calculating range value k = digit [index] if tight else 9 for i in range (0 , k + 1): # Calculating newTight value for nextstate newTight = tight if digit [index] == i else 0 # Fetching answer from next state ret += digitSum (index - 1 , sumof + i , newTight , digit) if not tight : dp [index][sumof][tight] = ret return ret # Returns sum of digits in numbers from a to b def rangeDigitSum (a , b): digitA = [] # Storing digits of a-1 in digitA getDigits (a - 1 , digitA) # Finding sum of digits from 1 to "a-1" which is passed as digitA ans1 = digitSum (len (digitA) - 1 , 0 , 1 , digitA) digitB = [] # Storing digits of b in digitB getDigits (b , digitB) # Finding sum of digits from 1 to "b" which is passed as digitB ans2 = digitSum (len (digitB) - 1 , 0 , 1 , digitB) return ans2 - ans1 a , b = 123 , 1024 print ("digit sum for given range: " , rangeDigitSum (a , b)) # This code is contributed by rupasriachanta421 C# // C# Code using System ; using System.Collections.Generic ; namespace GFG { class Program { // Memoization for the state results

static long [, ,] dp = new long [20 , 180 , 2]; // Stores the digits in x in a vector digit static void getDigits (long x , List < int > digit) { while (x != 0) { digit . Add ((int)(x % 10)); x /= 10 ; } } // Return sum of digits from 1 to integer in // digit vector static long digitSum (int idx , int sum , int tight , List < int > digit) { // base case if (idx == - 1) return sum ; // checking if already calculated this state if (dp [idx , sum , tight] != - 1 && tight != 1) return dp [idx , sum , tight]; long ret = 0 ; // calculating range value int k = (tight != 0) ? digit [idx] : 9 ; for (int i = 0 ; i <= k ; i ++) { // calculating newTight value for next state int newTight = (digit [idx] == i) ? tight : 0 ; // fetching answer from next state ret += digitSum (idx - 1 , sum + i , newTight , digit); } if (tight != 0) dp [idx , sum , tight] = ret ; return ret ; } // Returns sum of digits in numbers from a to b. static int rangeDigitSum (int a , int b) { // initializing dp with -1 for (int i = 0 ; i < 20 ; i ++) for (int j = 0 ; j < 180 ; j ++) for (int k = 0 ; k < 2 ; k ++) dp [i , j , k] = - 1 ; // storing digits of a-1 in digit vector List < int > digitA = new List < int > (); getDigits (a - 1 , digitA); // Finding sum of digits from 1 to "a-1" which is // passed as digitA. long ans1 = digitSum (digitA . Count - 1 , 0 , 1 , digitA); // Storing digits of b in digit vector List < int > digitB = new List < int > (); getDigits (b , digitB); // Finding sum of digits from 1 to "b" which is // passed as digitB. long ans2 = digitSum (digitB . Count - 1 , 0 , 1 , digitB); return (int)(ans2 - ans1); } // driver function to call above function public static void Main (String [] args) { int a = 123 , b = 1024 ; Console . WriteLine ("digit sum for given range : " + rangeDigitSum (a , b)); } } } // This code is contributed by ishankhandelwals. JavaScript // Given two integers a and b. The task is to print // sum of all the digits appearing in the // integers between a and b // Memoization for the state results let dp = []; // Stores the digits in x in a array digit function getDigits (x , digit) { while (x) { digit . push (x % 10); x = Math . floor (x / 10); } } // Return sum of digits from 1 to integer in // digit array function digitSum (idx , sum , tight , digit) { // base case if (idx === - 1) return sum ; // checking if already calculated this state if (dp [idx] && dp [idx][sum] && dp [idx][sum][tight] != null && tight !== 1) { return dp [idx][sum][tight]; } let ret = 0 ; // calculating range value let k = tight ? digit [idx] : 9 ; for (let i = 0 ; i <= k ; i ++) { // calculating newTight value for next state let newTight = digit [idx] === i ? tight : 0 ; // fetching answer from next state ret += digitSum (idx - 1 , sum + i , newTight , digit); } if (tight !== 1) { dp [idx] = dp [idx] || []; dp [idx][sum] = dp [idx][sum] || []; dp [idx][sum][tight] = ret ; } return ret ; } // Returns sum of digits in numbers from a to b. function rangeDigitSum (a , b) { // initializing dp with null dp = []; // storing digits of a-1 in digit array let digitA = []; getDigits (a - 1 , digitA); // Finding sum of digits from 1 to "a-1" which is passed // as digitA. let ans1 = digitSum (digitA . length - 1 , 0 , 1 , digitA); // Storing digits of b in digit array let digitB = []; getDigits (b , digitB); // Finding sum of digits from 1 to "b" which is passed // as digitB. let ans2 = digitSum (digitB . length - 1 , 0 , 1 , digitB); return ans2 - ans1 ; } // driver function to call above function function main () { let a = 123 , b = 1024 ; console . log ("digit sum for given range : " + rangeDigitSum (a , b)); } main (); // This code is contributed by divyansh2212 Output: digit sum for given range : 12613 Time Complexity : There are total idx*sum*tight states and we are performing 0 to 9 iterations to visit every state. Therefore, The Time Complexity will be O(10*idx*sum*tight) . Here, we observe that tight = 2 and idx can be max 18 for 64 bit unsigned integer and moreover, the sum will be max 9*18 ~ 200. So, overall we have 10*18*200*2 ~ 10^5 iterations which can be easily executed in 0.01 seconds . Space Complexity: The space complexity of this algorithm is O(d*sum*tight) as it uses a dp array of size d*sum*tight. where d is the number of digits in the number, sum is the sum of the digits and tight is a boolean value indicating whether or not the current digit is restricted to the digit in the number or not. The above problem can also be solved using simple recursion without any memoization. The recursive solution for the above problem can be found here . We will be soon adding more problems on digit dp in our future posts. Comment Article Tags: Article Tags: DSA