Digit DP | Introduction

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 $\mathbf{t_n t_{n-1} t_{n-2} \dots t_2 t_1}$ as the decimal representation where $\mathbf{t_i}$ **(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 to 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., $\mathbf{t_{n-1}\ t_{n-2} \dots t_2\ t_1}$ and we need to calculate answer for state having n digitd$\mathbf{t_n\ t_{n-1}\ t_{n-2} \dots t_2\ t_1}$. 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 unrestricted 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 = 167, 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 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])
```

C++ code for the above implementation:

```
// 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
long long 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++)
    {
        // caclulating 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;
}
```

Run on IDE

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**.

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.

This article is contributed by **Nitish Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Practice Tags : Dynamic Programming

Article Tags : Competitive Programming Dynamic Programming

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content. Login to Improve this Article

Recommended Posts:

Bitmasking and Dynamic Programming | Set 1 (Count ways to assign unique cap to every person)
Program for Fibonacci numbers
How to solve a Dynamic Programming Problem ?
Digit DP | Introduction
Minimum cost path from source node to destination node via an intermediate node
Count number of right triangles possible with a given perimeter
Program to find last two digits of 2^n
Count unique subsequences of length K
Queries to find maximum product pair in range with updates
Sum of all prime divisors of all the numbers in range L-R

(Login to Rate)

4.1 Average Difficulty : **4.1/5.0**
Based on **28** vote(s)

Basic | Easy | Medium | Hard | Expert

Add to TODO List

Mark as DONE

Feedback | Add Notes | Improve Article

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!