DIGIT-DP

Wrote this article a long ago but during solving a problem recently thought of sharing this article publicly. Hope it will help some contestants to understand the idea clearly.

Digit dp is a very easy technique and also useful to solve many dynamic programming problems. Seeing the name “Digit DP” it’s easy to guess that we are going to do something using the digits. Yes we are actually going to play with digits. Let’s explain the concept using a classical problem.

Problem

How many numbers **x** are there in the range **a** to **b**, where the digit **d** occurs exactly **k** times in **x**? There may have several solutions including number theory or combinatorics, but let’s see how we can solve this problem using digit dp.

Solve for range (zero to a)

Using digit dp we always focus on building a number satisfying all the conditions. If we finally manage to build that number then we say, yes we have got one ;-) But how we’ll build that number? For the time being let’s say **a** is zero. So we need to find the total numbers which are not greater than **b** and also satisfy the given conditions.

Building a sequence of digits

Let’s consider the number as a sequence of digits. Let’s name the sequence **sq**. Initially **sq** is empty. We’ll try to add new digits from left to right to build the sequence. In each recursive call we’ll place a digit in our current position and will call recursively to add a digit in the next position. But can we place any of the digits from **0** to **9** in our current position? Of course not, because we need to make sure that the number is not getting larger than **b**.

Information we need to place a digit at the current position

Let’s say during the building of the sequence, currently we are at position **pos**. We have already placed some digits in position from **1** to **pos-1**. So now we are trying to place a digit at current position **pos**. If we knew the whole sequence we have build so far till position **pos-1**then we could easily find out which digits we can place now. But how?

You can see that, in the sequence **sq** the left most digit is actually the most significant digit. And the significance get decreased from left to right. So if there exist any position **t** (1<=t<pos) where sq[t] < b[t] then we can place any digit in our current position. Because the sequence has already become smaller than **b** no matter which digit we place in the later positions. Note, b[t] means the digit at position **t** at number **b**.

But if there was no **t** that satisfy that condition then at position **pos**, we can't place any digit greater than b[pos]. Because then the number will become larger than **b**.

Do we really need the whole sequence?

Now imagine, do we really need that whole sequence to find if a valid **t** exist? If we placed any digit in our previous position which was smaller than its corresponding digit in **b** then couldn't we just pass the information somehow so that we can use it later? Yes, using an extra parameter **f1**(true/false) in our function we can handle that. Whenever we place a digit at position **t** which is smaller than b[t] we can make **f1** = **1** for the next recursive call. So whenever we are at any position later, we don't actually need the whole sequence. Using the value of **f1** we can know if the sequence have already become smaller than **b**.

Extra condition

So far we focused on building the sequence **sq**, but we have forgotten that there is an extra condition which is, digit **d** will have to occur exactly **k** times in sequence **sq**. We need another parameter **cnt**. **cnt** is basically the number of times we have placed digit **d** so far in our sequence **sq**. Whenever we place digit **d** in our sequence **sq** we just increment **cnt** in our next recursive call.

In the base case when we have built the whole sequence we just need to check if **cnt** is equal to **k**. If it is then we return **1**, otherwise we return **0**.

Final DP States

If we have understood everything so far then it's easy to see that we need total three states for DP memoization. At which position we are, if the number has already become smaller than **b** and the frequency of digit **d** till now.

Solve for range (a to b)

Using the above approach we can find the total valid numbers in the range **0** to **b**. But in the original problem the range was actually **a** to **b**. How to handle that? Well, first we can find the result for range **0** to **b** and then just remove the result for range **0** to **a-1**. Then what we are left off is actually the result from range **a** to **b**.

How to solve for range a to b in a single recursion?

In the above approach we used an extra parameter **f1** which helped us to make sure the sequence is not getting larger than **b**. Can't we do the similar thing so that the sequence does not become smaller than **a**? Yes of course. For that, we need to maintain an extra parameter **f2**which will say if there exist a position **t** such that sq[t] > a[t]. Depending on the value of **f2** we can select the digits in our current position so that the sequence does not become smaller than **a**. Note: We also have to maintain the condition for **f1** parallely so that the sequence remains valid.

Please check this to find the sample code of our initial approach.

Problem List

1. Investigation
2. LIDS
3. Magic Numbers
4. Palindromic Numbers
5. Chef and Digits
6. Maximum Product
7. Cantor
8. Digit Count
9. Logan and DIGIT IMMUNE numbers
10. Sanvi and Magical Numbers
11. Sum of Digits

12. Digit Sum
13. Ra-One Numbers
14. LUCIFER Number
15. 369 Numbers
16. Chef and special numbers
17. Perfect Number
18. The Great Ninja War