Problem statement

Find the count of numbers in range 'L' to 'R', where R>=L and L>=0 such that the sum of the digits of number equals 'S'

Brute Force approach
The brute force approach is quite simple, simply iterate over the numbers from L to R, and for each number say 'x', check if the sum of digits of x equals S.

Pseudocode:

```
//  The function takes the number cur and sum S as input
function check(cur, S)

        //  Initializing sum to 0
        sum = 0

        //  Finding sum of digits of cur
        while cur > 0
                dig = cur % 10
                sum = sum + dig
                cur = cur / 10

        return sum == S

//  The function takes the range L and R and sum S as input
function count(L, R, S)

        //  Initializing ans to 0
        ans = 0
        for cur = L to R
                //  Checking if the sum of digits of number cur is equal to S
                if check(cur)
                        ans++

        return ans
```

Time Complexity: O((R-L+1) * T), where T is the time taken by function check() in the above code, T is $\log_{10}$(cur).

The drawback of this approach is that it's not optimal for large ranges, say(L = 1 and R = 10^{18}), then iterating 10^{18} times and checking for each number the given property cannot be done under justified time constraints.

Digit DP approach

Let's try to solve the problem using digit dp, it is clear that the property that the digits of a number must follow is "The sum of digits of the numbers equals S".

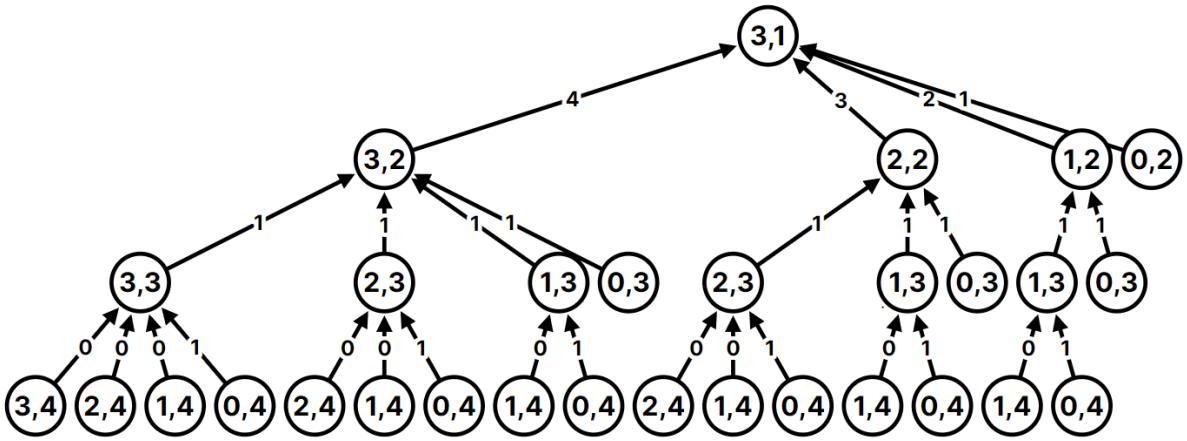
First, let's try to solve the problem for range 0 to R.

- **Range from 0 to R**

  Let us consider a number as a sequence of digits denoted by **seq**. Initially, **seq** is empty, we will try adding digits to **seq** from the most significant position to least significant position(left to right) recursively. In each recursive call, we will place a digit **d(0 to 9)** in the current position, let us denote the current position by **pos** where pos = 1 represents the most significant position**,** and call recursively to add a digit in the next position i.e pos + 1.

  Hence let us denote our recursive function by f(S,X) which returns us the count of numbers less than R, where S is the current sum and X is the current position, so f(S,X) = $\sum_{i=0}^{9}$ f(S - i,X + 1).

  Following is a figure that finds the count of all the numbers less than 1000, having the sum of digits 3, where we can clearly observe that repeated recursive calls are being made.



  What digits can we place in the current position 'pos'?

  Certainly, we cannot place any digit from 0 to 9 in our current position **pos** as we need to make sure that the number we are trying to build does not exceed **R**.

  As we are building the sequence from most significant position to least significant position(left to right) so if there exists any position **prev**(prev < pos) such that seq[prev] < R[prev], then we can place any digit from 0 to 9 at our current position **pos**.

However, if there exists no such position i.e seq[prev] = R[prev] ∀ 1<=prev<pos, then we can place digit from 0 to R[pos] at our current position.

How to keep track of the whole sequence we have built so far and determine whether such prev exists?
We do not need the whole sequence we have built so far; instead, we can maintain a boolean variable(true or false), let's denote it by **b**, whose truth value will determine if **prev** exists. Whenever we place a digit **d** such that d<R[pos], we can mark b as 'true' and pass this information in the next recursive calls. In other words, if **b** is 'false', then there exists no such position i.e seq[prev] = R[prev] ∀ 1<=prev<pos.

- **Range from L to R**
  Let us denote the count of numbers from 0 to N having the sum of digits equal to S by **count[N]**. It is easier to observe that the count of digits from L to R will be **count[R] - count[L-1]**.

- **DP States**
  It can be clearly observed that there are three states of our dp. The first state is our current position **pos;** second, the sum **S** we need to achieve, and last is a boolean value **b** to tell if the number has become smaller than R already(the upper bound for which we need to find the count).

Pseudocode:

```
/*
        The function takes input
            1. pos -> The current position
            2. S -> sum to achieve
            3. b -> boolean
            4. R -> It represents the sequence of the upper range for example, if the
               upper bound of the range is 123, then R = [1,2,3] (One Based Indexed)
            5. n -> The max length of the sequence(No of digits in R)
            6. dp -> The dp table which is a 3-D array i.e dp[pos][S][b] which is initialized
               to -1.
*/
function count(pos, S, b, R, n, dp)

        //  Base Case
        if S == 0 and pos == n + 1
                return 1

        if pos == n + 1 or S < 0
                return 0
```

```
//   If we have already calculated the answer, then return dp[pos][S][b]
if dp[pos][S][b] != -1
        return dp[pos][S][b]

//   Initializing totcnt to 0
totcnt = 0

//   Check if boolean 'b' is true then we can place any digit at pos
if b equals true
        for dig = 0 to 9
                totcnt = totcnt + count(pos + 1, S - dig, b, R, n, dp)
else
        for dig = 0 to R[pos]
                /*
                        If we place a dig less the R[pos], then we can set b as true
                        for the next recursive call
                */
                if dig < R[pos]
                        b = true
                        totcnt = totcnt + count(pos + 1, S - dig, b, R, n, dp)
                //   Otherwise go to the next recursive call with b as false
                else
                        totcnt = totcnt + count(pos + 1, S - dig, b, R, n, dp)

return dp[pos][S][b] = totcnt
```

Time Complexity: O(n*S*2*10), as we have three states representing the sum S, the position and a boolean variable with iteration through each possible digit in every recursive call in worst case.