**Task «A+B=C»**

To solve this problem, we can apply the method of dynamic programming. We introduce the following signs.

Let *C '* be the number consisting of the last k digits of the number C.

Let s[*k*][*i*][*j*][0]  be the number of ways to decompose the number *C’* into the sum of the numbers *A’* and *B’*, each consisting of k digits, with *A '* starting with the digit *i* (0≤*i*≤9) and *B’* starting with the digit *j* (0≤*j*≤9). We also consider sums in which the terms begin with zero, for example, 25 = 03 + 22.

Next, we add one to the number *C '*on the left and denote by *s*[*k*][*i*][*j*][1] the number of ways to decompose the new number into the same sum as discussed above (we can say that here, when adding, "1 is transferred to the highest rank"). Summarizing the above, *s*[*k*][*i*][*j*][*p*]  is the number of ways to represent the number *C’* as a sum with the transfer of p to the highest digit (0 ≤ i ≤ 10, 0≤*i*≤10, 0≤*j*≤10, 0≤*p*≤1, 1≤*k* ≤|*C*|  and |*C|* denotes the length of the original number – n).

With this in mind, it is not difficult to come to the conclusion that the answer to the problem will be equal to the sum of all *s*[|*C*|][*i*][*j*][0] for all *i*and*j* other than zero (components cannot begin with zero). Now it remains only to calculate *s*[*k*][*i*][*j*][*p*]]. Let's show how it can be done.

Let *t* be the *k*th digit of *C*. Consider three cases.

1) *(i + j) = (10p + t),*that is, *(i + j)* is equal to either *t*or *(10 + t)* depending on the value of *p*. In this case, there is no shift to the right, and one only needs to sum *s*[*k* – 1][*x*][*y*][0] for all *0 ≤ x ≤ 9*and*0 ≤ y ≤ 9* except for those for which *i = x*or*j = y*, that is, two adjacent digits are equal.

2) *(i + j) = (10p + t-1)*. In this case, 1 is transferred to the right digit, and now you need to sum *s*[*k* – 1][*x*][*y*][1] (again except for those for which *i = x* or *j = y*).

3) in other cases, *s*[*k*–1][*x*][*y*][1]=0, since it is impossible to get the digit *t* from the numbers starting with *i* and *j*.

The algorithm implementing the described solution has a linear complexity with respect to the length of the number *C,* but the constant will be quite large: for each length of the number *k*, you need to calculate 10 \* 10 \* 2 = 200 numbers *s*[*k*][*i*][*j*][*p*], and to calculate each of them, you need about 10 \* 10 = 100 additions. But in fact, most of the values of*s*[*k*][*i*][*j*][*p*] will be zero (see the third case discussed above), and only about 40 non-zero such values will need to be calculated. As a result, the asymptotic complexity of the algorithm will be about 4000×*N*..

Note that the required number of pairs of beautiful numbers *A* and *B* can be very large, so the answer in the problem is proposed to output modulo (109+7). All calculations must also be done on this module, so that the calculation process does not overflow the type used.