Digit Sum

Find the number of integers between 1 and K (inclusive) satisfying the following condition, modulo 10^9 + 7:

The sum of the digits in base ten is a multiple of D.

Constraints

All values in input are integers.

1 ≤ K < 10^10000
1 ≤ D ≤ 100

Sample Input 1
30
4

Sample Output 1
6

Those six integers are:
4, 8, 13, 17, 22 and 26.

Problem Breakdown

Lets understand the problem first

You are given:

K (a huge number, up to 10,000 digits long!)
D (1 ≤ D ≤ 100)

Count how many integers x satisfy:

1 ≤ x ≤ K
(sum of digits of x) % D == 0

Return the count mod 1e9+7.

Because K has up to 10,000 digits, you cannot convert it to an integer type.
So we treat K as a string.

Digit DP was made for this.

DP State

`dp[pos][tight][sumMod]`

Meaning:

- pos → which digit of K we're at (0...len-1)
- tight
  - 1 → prefix of our number is exactly equal to prefix of K
  - 0 → we are already below K → can place digits 0..9 freely
- sumMod
  - current digit sum modulo D

This is enough to determine future outcomes.

We do not need a prevDigit, because no adjacency condition.

Transition

At pos:

```
limit = tight ? K[pos] : 9

for d = 0..limit:
   nextTight = tight && (d == limit)
   nextSum = (sumMod + d) % D
```

The answer is:

```
dp[0][1][0] - 1
```

(because we subtract the number 0)

Code

```
#include <bits/stdc++.h>
using namespace std;

static const long long MOD = 1000000007;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    string K;
    int D;
    cin >> K >> D;

    int n = K.size();

    // dp[pos][tight][sumMod]
    static long long dp[10005][2][105];
    memset(dp, 0, sizeof(dp));

    dp[0][1][0] = 1;

    for (int pos = 0; pos < n; pos++) {
        int digitLimit = K[pos] - '0';
        for (int tight = 0; tight <= 1; tight++) {
            for (int sumMod = 0; sumMod < D; sumMod++) {

                long long val = dp[pos][tight][sumMod];
                if (val == 0) continue;

                int limit = (tight ? digitLimit : 9);

                for (int d = 0; d <= limit; d++) {
                    int nextTight = (tight && (d == limit));
                    int nextSum = (sumMod + d) % D;

                    dp[pos + 1][nextTight][nextSum] =
                        (dp[pos + 1][nextTight][nextSum] + val) % MOD;
                }
            }
        }
    }

    long long ans = (dp[n][0][0] + dp[n][1][0]) % MOD;

    // subtract the zero number (0..0 all digits)
    ans = (ans - 1 + MOD) % MOD;

    cout << ans << "\n";

    return 0;
}
```