Hack around the Lock

`hack.c`, `hack.C`, `hack.java`, `hack.p`

For your travel to the ACM-ICPC World Finals in Egypt, you have bought a brand new luggage. The luggage has a numeric lock composed of K wheels with digits $0 \dots 9$ on each of them. Each combination of the values on the wheels can be represented as a K-digit number (possibly with leading zeros that count as digits). Only one combination opens the luggage.

As a little bit paranoid programmer, you wrote a sophisticated random number generator to choose the secret number and you can feel safe now because nobody will ever be able to guess it easily.

Unfortunately, after arriving to the hotel, you realize "nobody" does also include you. Sad story, isn't it? Instead of starting to frenetically rotate the wheels trying to find the right combination, you decided to write another program that will help you. Given an initial numeric combination, the program should find fastest way to try all other possible combinations — as we all know, the correct combination will always be the one we try at the very last.

It is possible to rotate one wheel at a time only. After changing the digit by one, you can check if the combination is correct. You also know that the initial combination is *not* correct (otherwise, you wouldn't need the whole thing at all, would you?). So, the overall time needed to open the luggage is proportional to the number of *steps*, each step corresponding to the rotation of some wheel by one. To make the things worse, it is impossible to turn a wheel directly from 0 to 9 or vice versa — you need 9 steps instead.

Input Specification

The input contains several instances; input of each instance is a line with a single decimal number N, which is the initial combination. The number may have leading zeros, which are counted into its number of digits K, $1 \leq K \leq 7$. For example, "`007`" is considered to be a 3-digit number. The last instance is followed by a line containing "`-1`".

Output Specification

For each instance, print two lines: the first one containing the minimum possible number of steps S. The second line must contain a sequence of S K-digit numbers. Every possible K-digit number (except for the initial one) must appear at least once in the sequence and each two consecutive numbers must differ in exactly one digit by exactly one (in the digit value).

Sample Input

```
5
00
-1
```

Output for Sample Input

```
13
6 7 8 9 8 7 6 5 4 3 2 1 0
99
10 20 30 40 50 60 70 80 90 91 81 71 61 51 41 31 21 11 01
  02 12 22 32 42 52 62 72 82 92 93 83 73 63 53 43 33 23 13 03
  04 14 24 34 44 54 64 74 84 94 95 85 75 65 55 45 35 25 15 05
  06 16 26 36 46 56 66 76 86 96 97 87 77 67 57 47 37 27 17 07
  08 18 28 38 48 58 68 78 88 98 99 89 79 69 59 49 39 29 19 09
```

The last line of the sample output above is cut into several lines to fit into the page width, but it should be a single line in the real output.