Question - 1

Lottery Coupons

There is a lottery with *n* coupons and *n* people take part in it. Each person picks exactly one coupon. Coupons are numbered consecutively from *1* to *n*, and the *coupon[i]* has the number *i* written on it. The winner of the lottery is any person who owns a coupon with the sum of digits written on it equal to *s*. If there are multiple winners, the prize is split equally among them. Determine how many values of *s* there are where there is at least one winner and the prize is split among the most people.

For example, given the input *n = 12*, the list of ticket numbers is *coupons = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]*. The sums of the digits are *coupon_sums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3]*. The largest number of winners is *2* which will occur for tickets numbered *[1, 10], [2, 11]* and *[3, 12]*. The maximum number of possible winners occurs for any of *3* possible values of *s*, so *3* is the answer.

Function Description

Complete the function *lotteryCoupons* in the editor below. The function must return the number of ways to choose *s*, in such a way that there is at least one winner and the price is split among the most number of people.

lotteryCoupons has the following parameter(s): *n*: an integer that represents the maximum ticket number

Constraints

- $1 \leq n \leq 10^4$

▼ Input Format For Custom Testing

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer, *n*, denoting the maximum ticket number.

▼ Sample Case 0

Sample Input 0

```
3
```

Sample Output 0

```
3
```

Explanation 0

The three lottery coupons are numbered 1, 2 and 3. The sum of the digits of the coupon numbers are 1, 2 and 3 respectively. There are three ways to choose s:

Help

- When s = 1, only the person with coupon number = 1 is the winner.
- When s = 2, only the person with coupon number = 2 is the winner.
- When s = 3, only the person with coupon number = 3 is the winner.

▼ Sample Case 1

Sample Input 1

```
11
```

Sample Output 1

```
2
```

Explanation 1

The lottery coupons are numbered from 1 to 11 and the sum of the digits of each of them is 1, 2, 3, 4, 5, 6, 7, 8, 9, 1 and 2 respectively. There are two ways to choose s:

- When s = 1, there are two winners and their coupon numbers are 1 and 10.
- When s = 2, there are two winners and their coupon numbers are 2 and 11.

▼ Sample Case 2

Sample Input 2

```
22
```

Sample Output 2

```
3
```

Explanation 2

The lottery coupons are numbered from *1* to *22* and the sum of the digits of each of them is *1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 3* and *4* respectively. There are two ways to choose s:

- When s = 1, there are two winners and their coupon numbers are 1 and 10.
- When s = 2, there are three winners and their coupon numbers are 2, 11 and 20.
- When s = 3, there are three winners, numbers 3, 12 and 21.
- When s = 4, there are three winners, numbers 4, 13, and 22.
- When numbers s is in the range [5-9] there are two winning numbers
- When s = 10, there is one winning number, 19.

There are *3* ways to have the greatest number of winners, when *n* = *2, 3 or 4.*

Question - 2

Approximate Matching

Given three strings, *text, prefixString* and *suffixString*, find:

- *prefixScore:* the longest substring of *text* matching the end of *prefixString*
- *suffixScore:* the longest substring of *text* matching the beginning of *suffixString*.

Sum the lengths of those two strings to get the *textScore*. The substring of *text* that begins with the matching prefix and ends with matching suffix is the string to remember. If it is the substring with the highest *textScore*, it is the value you are looking for. If there are other substrings with equal *textScore*, return the lexicographically lowest substring.

For example, if *text = "engine"*, *prefixString = "raven"*, and *suffixString = "ginkgo"*:

- *engine* matches *raven* so *prefixScore = 2*
- *engine* matches *ginkgo* so *suffixScore = 3*
- *textScore = prefixScore + suffixScore = 2 + 3 = 5*
- The substring of *text* with the highest *textScore* is *engin*.

Function Description

Complete the function *calculateScore* in the editor below. The function must return a string denoting the non-empty substring of *text* having a maximal *textScore*. If there are multiple such substrings, choose the lexicographically smallest substring.

calculateScore has the following parameter(s):

text: a string
prefixString: a string
suffixString: a string

Constraints

- *text*, *prefixString*, and *suffixString* contain lowercase English alphabetic letters *ascii[a-z]* only.
- $1 \leq |text|, |prefixString|, |suffixString| \leq 50$.
- It is guaranteed that there will always be a substring of *text* that matches at least one of the following:
  - One or more characters at the end of *prefixString*.
  - One or more characters at the beginning of *suffixString*.

▼ Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

The first line contains a string *text*.
The next line contains a string *prefixString*.
The last line contains a string *suffixString*.

▼ Sample Case 0

Sample Input 0

```
nothing
bruno
ingenious
```

Sample Output 0

```
nothing
```

Explanation 0

- *nothing* matches *bruno* so *prefixScore = 2*
- *nothing* matches *ingenious* so *suffixScore = 3*
- *textScore = prefixScore + suffixScore = 2 + 3 = 5*

The substring of *text* with the highest *textScore* begins with the prefix *no* and ends with the suffix *ing*: *nothing*.

▼ Sample Case 1

Sample Input 1

```
ab
b
a
```

Sample Output 1

```
a
```

Explanation 1

Given *text = "ab"*, our possible substrings are *sub = "a"*, *sub = "b"*, and *sub = "ab"*.

- *sub = "a"*
  - *prefixString = "b"*: The beginning of *sub* doesn't match the end of *prefixString*, so *prefixScore = 0*.
  - *suffixString = "a"*: The last character of *sub* matches the first character of *suffixString*, so *suffixScore = 1*.
  - *textScore = prefixScore + suffixScore = 0 + 1 = 1*
- *sub = "b"*
  - *prefixString = "b"*: The first character of *sub* matches the last character of *prefixString*, so *prefixScore = 1*.
  - *suffixString = "a"*: The end of *sub* doesn't match the beginning of *suffixString*, so *suffixScore = 0*.
  - *textScore = prefixScore + suffixScore = 1 + 0 = 1*
- *sub = "ab"*
  - *prefixString = "b"*: The beginning of *sub* doesn't match the end of *prefixString*, so *prefixScore = 0*.
  - *suffixString = "a"*: The last character of *sub* doesn't match the first character of *suffixString*, so *suffixScore = 0*.
  - *textScore = prefixScore + suffixScore = 0 + 0 = 0*

Two of these have a *textScore* of *1*, so we return the lexicographically smallest one (i.e., *"a"*).