CCSCNE-2025: Programming Contest

April 4th, 2025

1. Mail Delivery

A mailman is delivering mail in a neighborhood, where houses are arranged in an n × n grid. The mailman starts at the post office, located at the top-left corner of the grid at position (0,0). The mailman receives a sequence of movement commands in a single line, where:

- The first value represents n (the size of the neighborhood grid).
- The following characters represent movement directions for delivering mail:
  - 'U' → Moves Up (one row up).
  - 'D' → Moves Down (one row down).
  - 'L' → Moves Left (one column left).
  - 'R' → Moves Right (one column right).

It is guaranteed that the mailman will always be within the bounds of the grid. After processing all movements, the program should output the final house number where the mailman stops. The program should be able to read multiple lines of input, processing each line separately. The last line of input will be a -1 and should not be processed.

Neighborhood Numbering Pattern: Each house in the neighborhood is uniquely numbered, starting from 0 at the post office. The numbering increases from left to right across each row. When reaching the end of a row, the numbering continues at the beginning of the next row directly below. For example, if n = 3, the house numbers in the neighborhood is arranged as follows:

```
0  1  2
3  4  5
6  7  8
```

For the input 3 RDDLU, the movement of mailman for delivery is as follows:

Start at post office 0

```
*  1  2
3  4  5
6  7  8
```

Moves Right

```
0  *  2
3  4  5
6  7  8
```

Moves Down

```
0  1  2
3  *  5
6  7  8
```

Moves Down

```
0  1  2
3  4  5
6  *  8
```

Moves Left

```
0  1  2
3  4  5
*  7  8
```

Moves Up

```
0  1  2
*  4  5
6  7  8
```

Thus, output is 3.

Input

```
3 RDDLU
4 DDRRUULL
2 D
-1
```

Output

```
3
0
2
```

2. Arcade Games

You are at an arcade with x game tokens. Each token comes in a case. When you use a token to play a game, the token is spent, but you are left with an empty case.

The arcade offers a special exchange deal: You can trade y empty cases at the arcade counter to receive one additional game token (with its case).

Your task is to determine the maximum number of games you can play before you completely run out of tokens and can no longer exchange empty cases for new tokens.

Each test case consists of a single line containing two integers:

x The initial number of tokens (each in its case)

y The number of empty cases needed to exchange for one new token

For each test case, output a single integer – the maximum number of games that can be played. The end of input will be signaled by a -1.

Input

```
7 3
9 3
5 5
17 4
-1
```

Output

```
10
13
6
22
```

3. Base Conversion

The balanced signed-digit base 10 number system has all the standard base 10 digits plus 9 additional digits defined as follows: $\bar{1} = -1, \bar{2} = -2, \bar{3} = -3, \bar{4} = -4, \bar{5} = -5, \bar{6} = -6, \bar{7} = -7, \bar{8} = -8,$ and $\bar{9} = -9$. In all other ways, this number system works like standard base 10 notation, e.g., $5\bar{8}2 = 5\times10^2+(-8)\times10^1+2\times10^0 = 500 - 80 + 2 = 422$. This yields an infinite number of ways to represent most numbers. For example, $16 = 16, 2\bar{4}, 1\bar{8}\bar{4}, 1\bar{9}6, \ldots$ Your task is to write a program that accepts as input any number $0 \leq x < 99$ in base 10 and a number $1 \leq n \leq 10$ as input and prints out the number of representations of x exist with n or fewer digits in the base system described above.

Please note that the judges will use a timeout when running your program. Your program must solve *all* the test cases within this timeout or you will not receive credit for this problem.

Input

Your program must accept multiple test cases. Your program will end when the sentinel "−1" is entered on a line by itself. Each input test case, on a line by itself, will consist of two integers separated by a single space. The first integer is the *up to* two digit number to find representations of and the second integers is how many digits can be used in the construction of the representations. You should not double count representations that are the same except that they have a different number of leading zeros.

```
16 6
16 10
-1
```

Output

For each input test case, the number of representations of the number can be made using the base system proposed by this problem.

```
10
18
```

4. Approaching Perfection

A perfect tree is one where all of the internal nodes have the maximum number of children and all of the leaves are on the same level. Although binary trees are the most common type of tree, it is sometimes useful for a tree's nodes to have more than 2 children. In these k-ary trees each internal node can have up to k children.

Your goal is given the number of nodes that will be in the tree, to choose the type of tree that will be "most perfect", meaning it would have the fewest missing leaves to make it perfect.

Input

1. The number of nodes will be a positive integer, with each test case on a s eparate line.
2. The end of input will be signaled with a -1

```
55
23
84
88532
65000
-1
```

Output

1. For each test case output the number of nodes followed by two dashes and then the value of k that makes the tree most perfect. (no whitespace between the numbers)
2. The following constraints apply:
   - k must be greater than or equal to 2.
   - k must be less than a third of the number of nodes.
   - If there are two values of k that are equal in perfection, then choose the smaller one.

```
55--7
23--2
84--4
88532--3
65000--255
```

5. Hidden Key

Alice and Bob want to exchange messages, but they do not want to have just a single key that can get compromised. The protocol that they come up with is:

1. There is an initial secret key that they share.
2. The key will be present in the plaintext of the first message sent.
3. The sender will choose a random integer as the offset. The plaintext will be enciphered by shifting all of the letters of the message by the offset, i.e., a Caeser cipher.
4. The receipient will not know the offset, but by knowing the key they can deduce the offset from the ciphertext.
5. The word following the secret key in the plaintext will be the new secret key for the next message.

Input

1. The plaintext will only use lowercase letters, numbers, whitespace, and punctuation.
2. Only the lowercase letters should be enciphered. Numbers, whitespace, and punctuation should not be changed.
3. A test case will begin with the initial secret key on a line by itself. The secret key will be a separate word and not a substring of a word.
4. Following lines will be ciphertext to be decoded.
5. A test case will end with a line with a single period on it.
6. The end of the test cases will be signaled with a line with two periods on it.

```
welcome
qyfwigy ni nby 2025
jlialuggcha wihnymn
aiix fowe
.
difficult
nbyly uly nqi xczzcwofn nbcham
ch wigjonyl mwcyhwy:
hugcha,
wuwbcha,
uhx izz vs ihy yllilm.
.
..
```

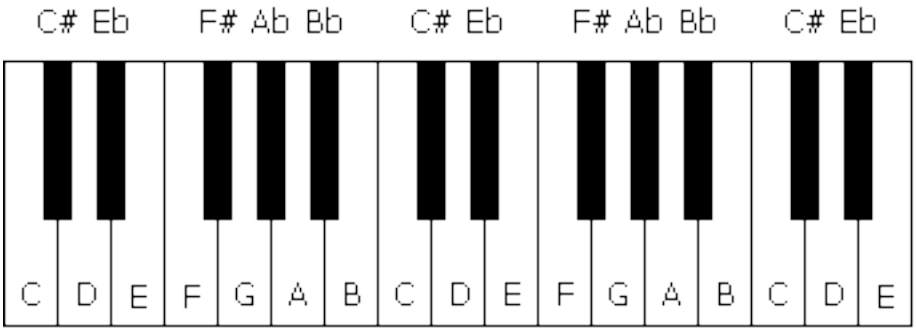
Output

1. Output each line of the decoded messages.
2. Separate test cases by a blank line.

```
welcome to the 2025
programming contest
good luck

there are two difficult things
in computer science:
naming,
caching,
and off by one errors.
```

6. Musical Scales



Background Information:

A major musical scale is a sequence of notes formed by a series of whole steps and half steps Starting with a musical note, you progress to the next note in a major scale using the following pattern:

whole, whole, half, whole, whole, whole half

A half step moves to the next key (whether it is black or white). A whole step moves to the second next key. A musical note is labelled with a capital letter [A-G] with a possible sharp(#),double sharp(##), flat(b), or double flat(bb) modifier. A sharp indicates a half step higher (key to the right); a flat indicates a half step lower (key to the left). Double sharps are a whole step higher; double sharps are a whole step lower. In the diagram shown, the white keys are labeled A through G which repeats. The black note between a C note and D note is both C# and Db. Also, E# is the same as F; Cb is the same as B.

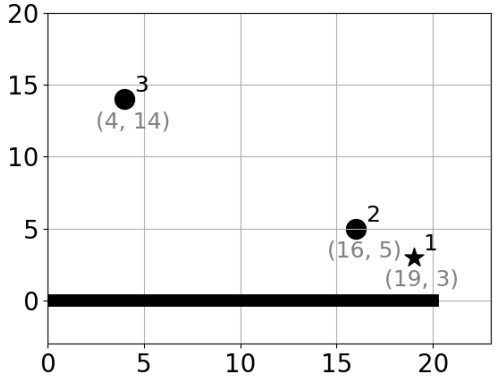
Your program will take legal notes (a letter between A and G with an optional sharp or flat modifier), and you will print the 8 note major scale sequence, starting and ending with the input note. The starting and ending note letters will be the only ones that are the same note letter. Programming Problem: Input: A legal key, represented by a letter A through G, followed by an optional # or b. The end of input will be a period on a line by itself. Output: The musical major scale, with each of the 8 notes separated by a space

Input

```
C
Db
B#
.
```

Output

```
C D E F G A B C
Db Eb F Gb Ab Bb C Db
B# C## D## E# F## G## A## B#
```



A graphical representation of the first sample input/output

7. Cartesian Race

Imagine n people on a cartesian coordinate plane think it might be fun to have a race. They decide on some *finite* line segment $\overline{AB}$ to count as the finish line. Since they are lazy, they decide that a "winner" of the race is anyone who *would have* reached the point along the line closest to themselves at the same time as or before anyone else *could have* reached the same point. Observe that there could be more than one "winner" by this definition. Everyone runs at the same speed and travels in perfectly straight lines. Your task is to write a program that finds all of the "winners" of this race.

Input

Your program must accept multiple test cases. Your program will end when the sentinel "-1" is entered on a line by itself. No output should be generated for the sentinel character.

1. All input values will be integers between 0 and 20, inclusively.
2. Each test case as a series of $2 \leq p \leq 22$ points, separated by a single space: $A\ B\ C_1\ C_2\ \ldots C_{p-1}\ C_p$.
3. The first two points A and B define the finish line; the remaining $n = p - 2$ points define the starting points of all the runners.
4. Each point, e.g., A will consist of two integers separated by a single space, e.g., $A_x\ A_y$.
5. Each test case will be on a line by itself.

```
0 0 20 0 19 3 16 5 4 14
15 15 5 5 4 14 14 4 4 4
-1
```

Output

For each test case, print out the positions within the input of each of the winners, separated by single spaces. For example, if the runners starting at points C_5, C_{12}, C_{13} and C_{17} are all of the winners, then your program must print `5 12 13 17` on a line by itself. Notice these positions are 1-based, not 0-based.

```
1
1 2 3
```