flaws in incorrect submissions. This process ensured our final test cases properly differentiated between correct and incorrect solutions for each problem.

Listing 2 | Prompt for generating test cases for code problem.

````
<USER>
Question:
You are given a string word and an integer k.

A substring s of word is complete if:

Each character in s occurs exactly k times.
The difference between two adjacent characters is at most 2. That is, for any two
    adjacent characters c1 and c2 in s, the absolute difference in their positions in
    the alphabet is at most 2.
Return the number of complete substrings of word.

A substring is a non-empty contiguous sequence of characters in a string.

**Input**
The first line contains a single integer t the number of test cases.
The first line of each test case contain one string word.
The second line of each test case contain one integer k.
It is guaranteed that word consists only of lowercase English letters, and the sum of |
    word| over all test cases does not exceed 10^5.

**Output**
For each test case, print one integer the number of complete substrings of word.

**Example**
Input
```
2
igigee
2
aaabbbccc
3
```

Output
```
3
6
```

Construct some random input generators to generate diverse and adversarial inputs,
    which are large enough to to make incorrect codes exceed the time limit.
</USER>

<ASSISTANT>
First construct a random input generator for large data with 10 testcases of the same
    length.

```python
import random
import string

def generate_random_string(l_min, l_max):
    length = random.randint(l_min, l_max)
    seq = ''.join(random.choices(string.ascii_lowercase, k=length))
    k = random.randint(1, length)
````
````

````
    return f"{{seq}}\n{{k}}"

t = 10 # the number of test cases
testcase = [generate_random_string(10000, 10000) for _ in range(t)]

print(t)
print("\n".join(testcase))
```

First construct a random input generator for large data with 10 testcases of different
    length.

```python
import random
import string

def generate_random_string(l_min, l_max):
    length = random.randint(l_min, l_max)
    seq = ''.join(random.choices(string.ascii_lowercase, k=length))
    k = random.randint(1, length)
    return f"{{seq}}\n{{k}}", len(seq)

t = 10 # the number of test cases
testcase = []
cum_length = 0
for i in range(t):
       # each test case should have length >= 1
       cur_testcase, cur_length = generate_random_string(1, 100000 - cum_length - (t -
           i))
       testcase.append(cur_testcase)
       cum_length += cur_length

print(t)
print("\n".join(testcase))
```

Construct a random input generator for large data with only one test case.

```python
import random
import string

def generate_random_string(l_min, l_max):
    ...

t = 1 # the number of test cases
testcase = [generate_random_string(90000, 100000) for _ in range(t)]

print(t)
print("\n".join(testcase))
```

Construct an input generator for large adversarial inputs of a string of a repeated
    single character.

```python
import random
import string

def generate_single_repeated_char(l_min, l_max):
    length = random.randint(l_min, l_max)
````

````
    char = random.choice(string.ascii_lowercase)
    seq = char * length
    k = random.randint(1, length)
    return f"{{seq}}\n{{k}}"

t = 1 # the number of test cases
testcase = [generate_single_repeated_char(90000, 100000) for _ in range(t)]

print(t)
print("\n".join(testcase))
```

Construct an input generator for large adversarial inputs of a string of two
    alternating characters.

```python
import random
import string

def generate_alternating_chars(l_min, l_max):
    length = random.randint(l_min, l_max)
    char1, char2 = random.sample(string.ascii_lowercase, 2)
    seq = ''.join(char1 if i % 2 == 0 else char2 for i in range(length))
    k = random.randint(1, length)
    return f"{{seq}}\n{{k}}"

t = 1 # the number of test cases
testcase = [generate_alternating_chars(90000, 100000) for _ in range(t)]

print(t)
print("\n".join(testcase))
```

Construct an input generator for large adversarial inputs of a string of sequential
    characters from alphabet.

```python
...
```

</ASSISTANT>

Question:
Question Description

Construct some random input generators to generate large, diverse and adversarial
    inputs, which are large enough to testing time complexity and to make incorrect
    codes exceed the time limit.

Use the format used in the above example by returning several input generators in
    different code blocks. Each of these generators prints EXACTLY ONE input directly
    into stdout.
````

In addition, we employ few-shot prompting for DeepSeek-V3 to generate responses to simple math problems, such as "1 + 1 = ?", ensuring that the outputs remain concise and appropriately structured. We provide the prompt for a simple math problem in Listing 3.

Listing 3 | Prompt for generating chain-of-thought for simple math problem.

```
## Question
How much is 5+4?
```
```