LEVEL - 4

1. Imagine developing a spell-checking or plagiarism detection tool for a text editor. This tool helps in identifying typographical errors where users might have jumbled the letters of a word. One of the features of this tool is to check if two strings ‘s’ & ‘t’ are anagrams of each other.

*An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.*

Example 1:
Input: s = "astronomer", t = "moonstarer"
Output: True

Example 2:
Input: s = "rat", t = "car"
Output: False

Constraints:

1 <= s.length, t.length <= 10^5
s and t consist of lowercase English letters & space.

Test Case 1:
s = "listen"
t = "silent"
Expected Output: True
Explanation: "listen" and "silent" are anagrams because they have the same characters with the same frequencies.
Test Case 2:
s = "hello
t = "world"
Expected Output: False
Explanation: "hello" and "world" are not anagrams because they have different characters.
Test Case 3:
s = "abc"
t = "abcd"
Expected Output: False

Test Case 4:
s = "apple"
 t = "apple"
Expected Output: True
Test Case 5:
s = " "
t =
Expected Output: String 1 must be between 1 and 100000 characters
Test Case 6:
s = "Astronomer"
t = "Moon starer"
Expected Output: Invalid character in first string
Test Case 7:
s = "abc"
 t = "def"
Expected Output: False
Test Case 8:
 s = "a!b#c"
 t = "cba!#"
Expected Output: should give “Invalid character in first string” and not get the input ‘t’
Explanation: Only lowercase characters and space is permitted in the input string
Test Case 9:
s = "123@#!"
T = “abcd”
Expected Output: Invalid character in first string
Explanation: Only lowercase characters and space is permitted in the input string
Test Case 10:
s = "abcdefghijklmnopqrstuvwxyz"
t = "zyxwvutsrqponmlkjihgfedcba"
Expected Output: True
Explanation: Test with strings of maximum length to ensure performance and handling of large datasets.

2. In personalized security systems, like password managers or encryption algorithms, detecting palindromic patterns can enhance security by identifying potentially weak or predictable passwords.
Now, assume a cybersecurity specialist is designing the password manager application that alerts users if their passwords is a palindrome, which might make the system vulnerable (or) easier to crack.
Let P be a password of length n.
R → Reverse of P.
If P == R, the password is a palindrome. Hence, print "Password is Vulnerable"
If P ≠ R, the password is not a palindrome. Hence, print "Password is Non-Vulnerable"

Example 1:
Input: P = "race a car"
Output: Password is Non-Vulnerable
Explanation: "race a car" is not a palindrome.

Example 2:
Input: P = "A man a plan a canal Panama"
Output: Password is Vulnerable
Explanation: "amanaplanacanalpanama" is a palindrome.

Constraints:
P.length → n
1 <= n <= 10^7
P contains alphabets and digits
Single character & digit is by default a palindrome

Test Case 1:
Input: P='a'
Expected Output: Password is Vulnerable
Test Case 2:
Input: P='ab'
Expected Output: Password is Non-Vulnerable
Test Case 3:
Input: P='aba'
Expected Output: Password is Vulnerable
Test Case 4:

Input: P='abccba'

Expected Output: Password is Vulnerable

Test Case 5:

Input: P='abcba'

Expected Output: Password is Vulnerable

Test Case 6:

Input: P='abcdedcba'

Expected Output: Password is Vulnerable

Test Case 7:

Input: P='aabbcc'

Expected Output: Password is Non-Vulnerable

Test Case 8:

Input: P='aabbccdd'

Expected Output: Password is Non-Vulnerable

Test Case 9:

Input: P='aabbccddcceedd'

Expected Output: Password is Non-Vulnerable

Test Case 10:

Input: P='abcdefghijklmnopqrstuvwxyz0123456789'

Expected Output: Password is Non-Vulnerable

Test Case 11:

Input: P= ‘ ’

Expected Output: Password is Vulnerable

3. Imagine the list of names of employees in a company is maintained in a database, where each name consists of letters from the English alphabet. Given an employee name consisting of English letters, develop a program that returns the greatest English letter which occurs as both a lowercase and uppercase letter in the name. The returned letter should be in uppercase. If no such letter exists, return an empty string.

Hint: An English letter 'b' is greater than the letter 'a', since 'b' appears after 'a' in the English alphabet.

Example 1:
Input: s = "arRAzFif"
Output: "R"
Explanation: The letter 'R' is the greatest letter to appear in both lower and upper case. Note that 'A' and 'F' also appear in both lower and upper case, but 'R' is greater than 'F' or 'A'.
Example 2:
Input: s = "AbCdEfGhIjK"
Output: ""
Explanation: There is no letter that appears in both lower and upper case.

Constraints:
1 <= s.length <= 1000
s consists of lowercase and uppercase English letters.

Test Case 1:
Test Case 1:
Input: "aAbBcCdD"
Expected Output: "D"
Explanation: Both 'D' and 'd' are present in the string, and 'D' is the greatest letter.
Test Case 2:
Input: "abcDEFghi"
Expected Output: No common letter found
Test Case 3:
Input: "abcdefg"
Expected Output: No common letter found
Test Case 4:
Input: "ABCDEF"
Expected Output: No common letter found

Test Case 5:
Input: "aA bB!cC"
Expected Output: "C"
Test Case 6:
Input: "XYZWVU"
Expected Output: No common letter found
Test Case 7:
Input: "a"
Expected Output: No common letter found
Test Case 8:
Input: "A"
Expected Output: No common letter found
Test Case 9:
Input: "ABCDE"
Expected Output: No common letter found
Test Case 10:
Input: "abcdef"
Expected Output: No common letter found
Test Case 11:
Input: "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
Expected Output: "Z"

4. Develop a code editor for the programmers to write and edit source code. One critical feature of the editor is syntax validation, which helps programmers identify and correct errors in their code as they type. Properly matching and nesting parentheses (and), braces { and }, and brackets [and] are essential for most programming languages.

Let s be a string representing the source code written by the programmer.

With meticulous attention to detail, your program analyzes the string to verify the following:

1. **Matching Types**: Every opening bracket has a corresponding closing bracket of the same type ('(', ')', '{', '}', '[', ']').
2. **Correct Order**: Brackets must be closed in the correct order, ensuring that each opening bracket is closed before any subsequent brackets of the same type.

Example 1:
Input: s = "()[]{}"
Output: True

Example 2:
Input: s = "(]"
Output: False

Constraints:
1 <= s.length <= 104
s consists of parentheses only '()[]{}'.

Test Case 1:
Test Case 1:
Input: "()"
Expected Output: True
Test Case 2:
Input: "()[]{}"
Expected Output: True
Test Case 3:
Input: "(]"
Expected Output: False
Test Case 4:
Input: "([)]"

Expected Output: False

Test Case 5:

Input: ""

Expected Output: True

Test Case 6:

Input: "["

Expected Output: False

Test Case 7:

Input: "]"

Expected Output: False

Test Case 8:

Input: "{[()()]}"

Expected Output: True

Test Case 9:

Input: "{[(])}"

Expected Output: False

Test Case 10:

Input: "abc(def{ghi}jkl)"

Expected Output: False

Test Case 11:

Input: "([)]{}"

Expected Output: False

Test Case 12:

Input: "((((((((((((((()))))))))))))))"

Expected Output: True

5. A textile company produces fabric rolls with different patterns. The patterns are represented by strings of characters 'a', 'b', and 'c', where each character represents a specific color. The company wants to find the minimum number of fabric rolls required to store all the patterns, by applying the following operation:

- Pick a non-empty prefix from a fabric roll where all the characters in the prefix are equal.
- Pick a non-empty suffix from the same fabric roll where all the characters in this suffix are equal.
- The prefix and the suffix should not intersect at any index.
- The characters from the prefix and suffix must be the same
- Delete both the prefix and the suffix from the fabric roll.

By repeatedly applying this operation, the company can reduce the length of the fabric rolls and minimize the storage space required.

Constraints:

1 <= s.length <= 10^5
s consists of only characters 'a', 'b', 'c'.

Example 1:
Input: s = "ca"
Output: 2
Explanation: You can't remove any characters, so the string stays as is.2

Example 2:
Input: s = "cabaabac"
Output: 0
Explanation: An optimal sequence of operations is:
- Take prefix = "c" and suffix = "c" and remove them, s = "abaaba".
- Take prefix = "a" and suffix = "a" and remove them, s = "baab".
- Take prefix = "b" and suffix = "b" and remove them, s = "aa".
- Take prefix = "a" and suffix = "a" and remove them, s = "".

Test Case 1:
Input: s = "aabccabba"
Output: 3
Explanation: Take prefix = “aa” and suffix = “a” and remove them, s = “bccabb”.
Take prefix = “b” and suffix = “bb” and remove them, s = “cca”.

Test Case 2:
Input: s = ""
Output: 0
Test Case 3:
Input: s = "a"
Output: 1
Test Case 4:
Input: s = "aba"
Output: 1
Test Case 5:
Input: s = "aab"
Output: 3
Test Case 6:
Input: s = "a123@a"
Output: Invalid Input
Test Case 7:
Input: s = "hab"
Output: Invalid Input

6. Imagine a birthday party is organized in “The residency”. Develop a system that helps to determine the compatibility between the birthday person and the attendees using the FLAMES game. Here’s how you could apply the game:

Let's denote the following variable:

S1, S2 → input names (case in-sensitive strings).

C_{Common} → set of common characters between S1 and S2.

S1_new, S2_new → strings after removing C

Count → Count of elements in S1_new & S2_new

Check → {F, L, A, M, E, S}

Flames Game Procedure:

1. Take two input names S1 & S2
2. Remove C_{Common} with their respective occurrences from both strings and store the result in S1_new, S2_new
3. Get the count of the characters that are left in S1_new, S2_new
4. Start with the first letter of “Check” and eliminate up to the value of “Count”.
5. Continue eliminating in a circular manner (modular arithmetic) until one letter remains.
6. The last remaining letter in “Check” after elimination is the result of the FLAMES game for the given names S1 and S2.

Constraints:

2 <= S1.length, S2.length <= 10^2

S1, S2 → English Alphabets (Case-insensitive)

Test Case 1:

Input:

Player1 = "HEMA", Player2 = "PRIYA"

Expected Output:

Result = 'E'

Test Case 2:

Input:

Player1 = "John", Player2 = "Doe"

Expected Output:

Result = 'F'

Test Case 3:

Input:
Player1 = "", Player2 = "PRIYA"
Expected Output:
Result = ‘Please enter two different names of 2 to 100 characters long’
Test Case 4:
Input:
Player1 = "", Player2 = ""
Expected Output:
Result = ‘Please enter two different names of 2 to 100 characters long’
Test Case 5:
Input:
Player1 = "a", Player2 = "b"
Expected Output:
Result =‘Please enter two different names of 2 to 100 characters long’

7. ABC company collects customer feedback through various channels such as online surveys, emails, and social media comments. Each feedback entry is stored as a string in a dataset. The company wants to analyze the kth distinct feedback received to gain insights into less commonly expressed opinions or issues. A distinct string is defined as a string that appears only once in the feedback. If there are fewer than k distinct strings, the function should return an empty string ("").

Let's denote the following variable:

$S = [s_1, s_2, s_3, \dots, s_n]$ → Array of document strings

s_k → kth distinct string in the feedback

n → Size of the string

Example 1:

Input:

S = ["d","b","c","b","c","a"]

k = 2

Output:

"a"

Explanation:

The only distinct strings in S are "d" and "a".

"d" appears 1st, so it is the 1st distinct string.

"a" appears 2nd, so it is the 2nd distinct string.

Since k == 2, "a" is returned.

Example 2:

Input:

S = ["aaa","aa","a"]

k = 1

Output:

"aaa"

Explanation:

All strings in S are distinct, so the 1st string "aaa" is returned.

Constraints:

- n>=0
- 1 <= S.length <= 10^5

- 1<=s[i].length<=30
- s[i] consists of lowercase and uppercase English letters only
- Inputs are case sensitive

Test Case 1: Single character
S = ["d","b","c","b","c","a"]
k = 2
Expected Output: "a"
Test Case 2: String with similar letters
S = ["aaa","aa","a"]
k = 1
Expected Output: "aaa"
Test Case 3: Lesser distinct string
S = ["a","b","a", "A"]
k = 3
Expected Output: ""
Test Case 4: Empty Array
S = [ ]
k = 1
Expected Output: ""
Test Case 5: Invalid input
S = ["$", "%", "2", "*", "$", "#", "*"]
k = 2
Expected Output: “Invalid input”
Test Case 6: Case Sensitive input
S = ["Apple", "apple", "banana", "Banana", "orange", "Orange"]
k = 3
Expected Output: "banana"
Test Case 7: Distinct input
S = ["cat", "dog", "elephant", "monkey"]
k = 3
Expected Output: "elephant"
Test Case 8: Identical strings
S = ["perfect", "perfect", "perfect"]
k = 2
Expected Output: ""

8. A new social media platform called "NameVibe" aims to help users create memorable and unique usernames. One of the challenges users face is finding a username that is both unique and reflects their personality. To address this, NameVibe introduces a feature that generates all possible palindromic partitions of a user's desired username.

Example:

When a user, Alice, creates an account in “NameVibe”, she decides to use the name "madam" because it has personal significance. However, she is unsure how to make it unique while keeping it recognizable. Alice uses the palindromic partition feature to explore creative ways to structure her username.

The system generates all possible palindromic partitions of "madam":

- "m", "a", "d", "a", "m"
- "m", "ada", "m"
- "madam"

Constraints:

Input Constraints:

- Let s be a string with length n.
- n >= 1

Output Constraints:

- If n = 1, the output should be −2.
- If n > 1, the output should be all possible palindromic partitions of s.
- If n < 0, Invalid.

Test Case 1:

Input:
Please enter your string: “aab”
Output:
a
a
b
a a
No of palindrome substrings: 4

Test Case 2:

Please enter your string: “a”

Output:

```
-2
```

Test Case 3:

Input:

Please enter your string: “abcba”

Output:

```
a
b
c
b c b
a b c b a
b
a
No of palindrome substrings:  7
```

Test Case 4:(Empty String)

Input: Please enter your string: " "

Output:

```
Invalid
```

Test Case 5 :

Input:

Please enter your string: “xyz”

Output:

```
x
y
z
No of palindrome substrings:  3
```

Test Case 6 :

Input:

Please enter your string: “a#b”

Output:

```
a
#
b
No of palindrome substrings:  3
```

Test Case 7 :

Input:

Please enter your string: “a1#a2”

Output:

```
a
1
#
a
2
No of palindrome substrings:  5
```

Test Case 8 :

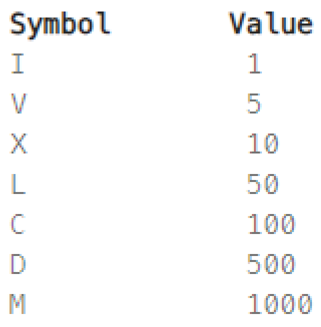
Input:

Please enter your string: “!@#$%^”

Output:

```
!
@
#
$
%
^
No of palindrome substrings:  6
```

9. Historians or researchers are analyzing ancient manuscripts or inscriptions that use Roman numerals extensively. These documents may contain dates, quantities, or other numerical data represented in Roman numerals. Converting these Roman numerals into their integer equivalents can aid in understanding historical timelines, quantities of goods or resources, or numerical data recorded in ancient texts. Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.



Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, 2 is written as II in Roman numerals, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

- I can be placed before V (5) and X (10) to make 4 and 9.
- X can be placed before L (50) and C (100) to make 40 and 90.
- C can be placed before D (500) and M (1000) to make 400 and 900.

Example 1:
Input: s = "III"
Output: 3
Explanation: III = 3.
Example 2:
Input: s = "LVIII"
Output: 58
Explanation: L = 50, V= 5, III = 3.
Example 3:
Input: s = "MCMXCIV"

Output: 1994
Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

Constraints:

- 1 <= s.length <= 15
- s contains only the characters ('I', 'V', 'X', 'L', 'C', 'D', 'M').
- It is guaranteed that s is a valid roman numeral in the range [1, 3999].

Test Case 1: Basic Case

Input:
Enter a Roman numeral: III
Output: 3

Test Case 2: Single Subtraction Case

Input:
Enter a Roman numeral: LVIII
Output: 58

Test Case 3: Multiple Subtractions

Input:
Enter a Roman numeral: MCMXCIV
Output: 1994.

Test Case 4: Simple Addition Case

Input:
Enter a Roman numeral: MVIII
Output: 1008

Test Case 5: Subtraction and Addition Combination

Input:
Enter a Roman numeral: MCMXC
Output: 1990

Test Case 6: Complex Subtractions

Input:
Enter a Roman numeral: XCIX
Output: 99

Test Case 7: Largest Single Symbol

Input:

Enter a Roman numeral: M

Output: 1000

Test Case 8: Combination of All Symbols

Input:

Enter a Roman numeral: MDCLXVI

Output: 1666

Test Case 9: Invalid input

Input:

Enter a Roman numeral: ABCD

Output: Invalid input

Test Case 10: Invalid input

Input:

Enter a Roman numeral: 12345

Output: Invalid input

10. In smart home systems, especially those that involve automated routines based on user preferences, you might need to analyze various permutations of time inputs to configure the latest possible 24-hour time schedule. For instance, if a user inputs an array of four digits for setting up timers or alarms, determining the latest 24-hour time that can be formed helps using each digit exactly once in scheduling automated events (like turning on lights or heating) in a way that best fits the user's needs. A valid 24-hour time is formatted as "HH", where HH is between 00 and 23, and MM is between 00 and 59. If no valid time can be made, return an empty string.

Let's denote the following variable:

$D = \{d_1, d_2, d_3, d_4\}$ → Set of four digits where d_i (for i=1,2,3,4) is an integer digit from the set {0,1,2,3,4,5,6,7,8,9}

HH → Hour component (00 to 23)

MM → Minutes component (00 to 59)

Example 1:

Input: D = [1, 2, 3, 4]

Output: "23:41"

Explanation: The valid 24-hour times are "12:34", "12:43", "13:24", "13:42", "14:23", "14:32", "21:34", "21:43", "23:14", and "23:41". Of these times, "23:41" is the latest.

Example 2:

Input: D = [5, 5, 5, 5]

Output: ""

Explanation: The valid 24-hour time cannot be formed

Constraints:

- The input array D contains exactly 4 digits.
- Each digit value is from 0 to 9.
- 00<=HH<=23
- 00<=MM<=59
- Each digit must be used exactly once in forming the time
- If there exists at least one valid time, return the latest possible valid time in the format "HH:MM" or an empty string if no valid time can be formed

Test Case 1: All Unique Digits, Valid Time
Input: D=[1,2,3,4]
Expected Output: "23:41"
Test Case 2: Digits Repeating, One Valid Time
Input: D=[1,1,2,3]
Expected Output: "23:11"
Test Case 3: All Same Digits
Input: D=[5,5,5,5]
Expected Output: ""
Test Case 4: All zeroes
Input: D=[0,0,0,0]
Expected Output: "00:00"
Test Case 5: Minimum Valid Digits
Input: D=[0,0,0,1]
Expected Output: "10:00"
Test Case 6: Mixed Digits, Multiple Valid Times
Input: D=[1,2,4,9]
Expected Output: "21:49"
Test Case 7: Maximum Hour Edge Case
Input: D=[1,9,2,3]
Expected Output: "23:19"
Test Case 8: Hours and Minutes Boundary
Input: D=[2,3,5,9]
Expected Output: "23:59"
Test Case 9: Invalid input
Input: D=[-2,0,1,9]
Expected Output: Invalid input
Test Case 10: Digits That Don’t Form a Valid Time
Input: D=[7,8,9,9]
Expected Output: ""

11. In text-based games or puzzles, such as crossword puzzles or word games, finding pairs of words in an array that do not share any common letters can be used to create challenging and engaging content. It can also be used in game design to develop puzzles where players must find optimal word pairs under specific constraints. Your goal is to determine the maximum product of the lengths of two such words where they do not have any letters in common.

Let's denote the following variable:

$W = [w_1, w_2, w_3, \dots, w_n]$ → set of n words

Each word w_i (for i=1,2,3,..,n) → String of letters from the English alphabet

P → Maximum product of the lengths of two words w_i *and* w_j (where i≠j) such that the two words do not share any common letters.

Example 1:
Input:Length:6
 words = ["abcw", "baz", "foo", "bar", "xtfn", "abcdef"]
Output: 16
Explanation: The two words with the maximum product of lengths that do not share any letters are "abcw" and "xtfn". Their lengths are 4 and 4, respectively, resulting in a product of 16.

Example 2:
Input:Length:4
 words = ["a", "aa", "aaa", "aaaa"]
Output: 0
Explanation: They all share the same letter “a”. There are no two unique words. So the product length is zero.

Constraints:

- $letters\,(w_i) \cap letters(w_j) = \phi$
- $max\{length(w_i) * length(w_j) \mid letters\,(w_i) \cap letters(w_j) = \phi \; and \; i \neq j\}$
- 2<=W.length<=100
- $1 <= w_i <= 10$
- Each word consists of lowercase English letters.
- Inputs are case sensitive

Test Case 1:
Input: Length:6
words = ["abcw", "baz", "foo", "bar", "xtfn", "abcdef"]
Expected Output: 16
Test Case 2:
Input:Length:7
 words = ["a", "ab", "abc", "d", "cd", "bcd", "abcd"]
Expected Output: 4
Test Case 3:
Input: Length:4
words = ["a", "aa", "aaa", "aaaa"]
Expected Output: 0
Test Case 4:
Input:Length:4
 words = ["hello", "world", "python", "java"]
Expected Output: 24
Test Case 5:
Input: Length:4
words = ["usa", "canada", "brazil", "mexico"]
Expected Output: 18
Test Case 6:
Input: Length:4
words = ["abc", "def", "ghi", "jkl"]
Expected Output: 9
Test Case 7:
Input: Length:4
words = ["apple", "orange", "grape", "banana"]
Expected Output: 0
Test Case 8:
Input:Length:4
 words = ["data", "science", "machine", "learning"]
Expected Output: 28
Test Case 9:
Input: Length:4
words = ["cat", "dog", "fish", "bird"]
Expected Output: 12

Test Case 10:
Input: Length:4
words = ["alpha", "beta", "gamma", "delta"]
Expected Output: 0

12. Assume a content moderation system is used by two user's for developing the content.
Let's denote the following variable names:

userOne → message from first user (string)
userTwo → message from second user (string)

Determine if you can construct the userOne string using the letters from the userTwo string. Each letter in the userTwo message can be used only once to construct the message for userOne.
If it is possible to construct the userOne message from userTwo message → Return True.
Otherwise, return False.

Example 1:
Input: userOne = "aa", userTwo = "ab"
Output: False

Example 2:
Input: userOne = "aa", userTwo = "aab"
Output: True

Constraints:

1 <= userOne.length, userTwo.length <= 10^2
userOne and userTwo consist of lowercase English letters.

Test Case 1:
Input: userOne = "a", userTwo = "b"
Output: False
Test Case 2:
Input: userOne = "a", userTwo = "a"
Output: True
Test Case 3:
Input: userOne = "abc", userTwo = "abccba"
Output: True
Test Case 4:
Input: userOne = "aaa", userTwo = "aab"
Output: False
Test Case 5:
Input: userOne = "ab", userTwo = "aabbcc"

Output: True

Test Case 6:

Input: userOne = "aabbcc", userTwo = "abcabc"

Output: False

Test Case 7:

Input: userOne = "abcdef", userTwo = "fedcbafedcba"

Output:False

Test Case 8:

Input: userOne = "aabbcc", userTwo = "abc"

Output: False

Test Case 9:

Input: userOne = "", userTwo = ""

Output: True

Explanation: An empty userOne can be constructed from an empty userTwo trivially.

Test Case 10:

Input: userOne = "a" * 100, userTwo = "a" * 100

Output: True

Explanation: userTwo exactly matches userOne with 100 'a' characters each.

13. You are working on a data processing project for an e-commerce company that needs to analyze product reviews. The company has a large dataset of product reviews, and they want to extract useful information from these reviews.
One of the pieces of information they want to extract is the frequency of each character in the reviews. This could also help the company improve their product offerings or customer service.
Write a program that takes a string of text as input and returns the string with the characters sorted in decreasing order based on their frequency. If there are multiple valid answers, the program should return any one of them.
Let's denote the following variable:

s → Input String (Review)

n → Length of the input string.

Example 1:
Input: s = "tree"
Output: "eert"
Explanation: 'e' appears twice while 'r' and 't' both appear once.
So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid answer.
Example 2:
Input: s = "cccaaa"
Output: "aaaccc"
Explanation: Both 'c' and 'a' appear three times, so both "cccaaa" and "aaaccc" are valid answers.
Note that "cacaca" is incorrect, as the same characters must be together.
Example 3:
Input: s = "Aabb"
Output: "bbAa"
Explanation: "bbaA" is also a valid answer, but "Aabb" is incorrect.
Note that 'A' and 'a' are treated as two different characters.

Constraints:

s - set of upper and lower case english alphabets

1 <= n <= 10^5.

Test Case 1:
Input: s = "x"
Expected Output: "x"
Test Case 2:

Input: s = "abcdef"

Expected Output: "abcdef"

Test Case 3:

Input: s = "zzzz"

Expected Output: "zzzz"

Test Case 4:

Input: s = "aabbccc"

Expected Output: "cccbbaa"

Test Case 5:

Input: s = "aabbcccddd"

Expected Output: "cccdddbbaa"

Test Case 6:

Input: s = ""

Output: ""

Test Case 7:

Input: s = "aaabbbccc"

Output: "aaabbbccc"

Test Case 8:

Input: s = "aaabbc"

Output: "aaabbc"

Test Case 9:

Input: s = "aabbcc"

Output: "aabbcc"

Test Case 10:

Input: s = "abcabc"

Output: "aabbcc"

14. A text processing project for a social media company involves a massive dataset of user posts, from which the company wants to analyze and identify trends and patterns.
One of the analyses they want to perform is to find the first non-repeating character in each post. This information could be useful for identifying a unique character the user uses in their posts.
Write a program that takes a string of text as input and returns the first non-repeating character in the string.
Let's denote the following variable:

post → input string

n → length of the input string “post”

If the input contains non-repeating characters, then the program returns the first non-repeating character.
If there are no non-repeating character, then the program returns None.

Example 1:
Input: s = "warmwelcome"
Output: a

Example 2:
Input: s = "happyanniversary"
Output: h

Example 3:
Input: s = "warmwarm"
Output: None

Constraints:

1 <= n <=10^5

post consists of only lowercase English alphabets.

Test Case 1:
Input: s = "greetings"
Output: "g"
Test Case 2:
Input 2: "Greetteerg"
Expected Output 2: None
Test Case 3:
Input 3: "aabbcc"
Expected Output 3: None

Test Case 4:
Input: s = "a"
Output: "a"
Test Case 5:
Input: s = "a" * 100000
Output: "a"
Test Case 6:
Input: s = ""
Output: None
Test Case 7:
Input: s = "abcdefghi"
Output: "a"
Test Case 8:
Input: s = "abacabad"
Output: "c"
Test Case 9:
Input: s = "aabccddeeff"
Output: "b"
Test Case 10:
Input: s = "abcdefgh"
Output: "a"

15. Imagine an online registration system for a university where students need to create a unique username. The usernames are alphanumeric strings that may contain both lowercase letters and digits. For security and administrative purposes, the system needs to perform various checks on the username, including verifying the complexity and uniqueness of the username.

One of the checks could be to determine the second largest digit in the username to ensure it meets certain criteria on further processing.

Let's denote the following variable

S → Input String (Alphanumeric)

Example 1:
Input: s = "dfa12321afd"
Output: 2
Explanation: The digits that appear in s are [1, 2, 3]. The second largest digit is 2.

Example 2:
Input: s = "abc1111"
Output: -1
Explanation: The digits that appear in s are [1]. There is no second largest digit.

Example 2:
Input: s = "a123456789b"
Expected Output: 8
Explanation: The digits are [1, 2, 3, 4, 5, 6, 7, 8, 9]. The second largest digit is 8.

Constraints:
1 <= s.length <= 10^3
s consists of only lowercase English letters and/or digits.

Test Case 1:
Input: s = "987abc"
Output: 8
Test Case 2:
Input: s = "abc112233def"
Output: 2
Test Case 3:

Input: s = "frt2222uty"

Output: -1

Test Case 4:

Input: s = "abcdefghijklmnopqrstuvwxyz"

Output: -1

Test Case 5:

Input: s = "a" * 100

Output: -1

Test Case 6:

Input: s = "a"

Output: -1

Test Case 7:

Input: s = "abc123def"

Output: 2

Test Case 8:

Input: s = "9876543210"

Expected Output: 8

Test Case 9:

Input: s = "1"

Expected Output: -1

Test Case 10:

Input: s = "a12a12"

Expected Output: 1

16. Develop a system to analyze posts for sentiment analysis and detect specific patterns of interest in user-generated content. Specifically, identify posts that contain certain keywords (patterns) within a specified distance from each other.
For instance, you are interested in finding instances where the word "happy" (pattern a) and the word "birthday" (pattern b) appear within a distance of 5 characters from each other in user posts. This can help in identifying birthday wishes. This could help in understanding context or detecting specific topics being discussed together.
Let's denote the following variables:
s → Input String
a, b → Input Patterns (set of lowercase characters)
k → Integer denoting the distance between a and b

An index i is beautiful if:
    0 <= i <= s.length - a.length
    s[i..(i + a.length - 1)] == a
There exists an index j such that:
    0 <= j <= s.length - b.length
    s[j..(j + b.length - 1)] == b
    |j - i| <= k
Finally, Return an array of all the beautiful indices in the text, sorted in ascending order.

Example 1:
Input: s = "isawsquirrelnearmysquirrelhouseohmy", a = "my", b = "squirrel", k = 15
Output: [16,33]
Explanation: There are 2 beautiful indices: [16,33].
- The index 16 is beautiful as s[16..17] == "my" and there exists an index 4 with s[4..11] == "squirrel" and |16 - 4| <= 15.
- The index 33 is beautiful as s[33..34] == "my" and there exists an index 18 with s[18..25] == "squirrel" and |33 - 18| <= 15.
Thus we return [16,33] as the result.

Example 2:
Input: s = "abcd", a = "a", b = "a", k = 4
Output: [0]
Explanation: There is 1 beautiful index: [0].
- The index 0 is beautiful as s[0..0] == "a" and there exists an index 0 with s[0..0] == "a" and |0 - 0| <= 4.
Thus we return [0] as the result.

Constraints:
1 <= k <= s.length <= 105
1 <= a.length, b.length <= 10
s, a, and b contain only lowercase English letters.

Test Case 1: Basic Case
Input:
s = "isawsquirrelnearmysquirrelhouseohmy"
a = "my"
b = "squirrel"
k = 15
Expected Output: [16, 33]

Test Case 2: Pattern at the Same Position
Input:
s = "abcd"
a = "a"
b = "a"
k = 4
Expected Output: [0]

Test Case 3: Overlapping Patterns
Input:
s = "abcabca"
a = "abc"
b = "bca"
k = 1
Expected Output: [0,3]

Test Case 4: Multiple Beautiful Indices
Input:
s = "ababab"
a = "a"
b = "b"
k = 1
Expected Output: [0, 2, 4]

Test Case 5: No Beautiful Indices
Input:
s = "abcde"
a = "a"
b = "e"
k = 1
Expected Output: []

Test Case 6: Large Distance
Input:
s = "abcdefghijklmnopqrstuvwxyz"
a = "a"
b = "z"
k = 25
Expected Output: [0]

Test Case 7: Patterns at Ends of the String
Input:
s = "abcyz"
a = "abc"
b = "yz"
k = 2
Expected Output: []

Test Case 8: Large String with No Beautiful Indices
Input:
s = "a" * 100000
a = "abc"
b = "def"
k = 100
Expected Output: []

Test Case 9: Single Character Patterns
Input:
s = "aaa"
a = "a"
b = "a"

k = 1
Expected Output: [0, 1, 2]

Test Case 10: Pattern Appearing Multiple Times
Input:
s = "abababab"
a = "ab"
b = "ba"
k = 2
Expected Output: [0, 2, 4, 6]