Shabari K S - Self Practice - Problem Solving - IV

"

1. Remaining String

Problem statement:

Given a string S without spaces, a character ch, and an integer count, the task is to find the string after the specified character has occurred count number of times.

Example 1:

```
Input: S = "Thisisdemostring", ch = 'i',
count = 3
Output: ng
```

> Explanation: Substring of S after the 3rd occurrence of 'i' is "ng"

Example 2:

```
Input: S = "Thisisdemostri", ch = 'i',
count = 3
Output: Empty string
```

> Explanation: 'i' occurce 3rd time at last index

Constraints:

```
1 <= |S| <= 10^5
```

Analysis:

1. We need to find the index of the character ch in the string S.
2. If the index of the character ch is less than count, then return an empty string.
3. Otherwise, return the substring of S starting from the index of the character ch after count occurrences of ch.

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class RemainingStrings {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String S = sc.next();
        char ch = sc.next().charAt(0);
        int count = sc.nextInt();
        System.out.println(remainingString(S, ch, count));
    }

    private static String remainingString(String S, char ch, int count) {
        int index = -1;
        for (int i = 0; i < S.length(); i++) {
            if (S.charAt(i) == ch) {
                count--;
                if (count == 0) {
                    index = i;
                    break;
                }
            }
        }
        if (index == -1) {
            return "";
        }
        return S.substring(index + 1);
    }
}
```

Output:

```
Input: Thisisdemostring i 3
Output: ng
```

---

2. Check for subsequence

Problem statement:

Given two strings A and B, find if A is a subsequence of B.

Example 1:

```
Input:
A = AXY
```

```
B = YADXCP
Output: 0
```

> Explanation: A is not a subsequence of B as 'Y' appears before 'A'.

Example 2:

```
Input:
A = gksrek
B = geeksforgeeks
Output: 1
```

> Explanation: A is a subsequence of B.

Constraints:

```
1<= |A|,|B| <=10^4
```

Analysis:

1. Initialize two pointers i and j to 0.
2. Traverse both strings A and B.
3. If A[i] is equal to B[j], then increment i and j.
4. If i becomes equal to the length of A, then return 1.
5. If j becomes equal to the length of B, then return 0.
6. If A is a subsequence of B, then i will be equal to the length of A.

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class CheckSubsequence {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String A = sc.next();
        String B = sc.next();
        System.out.println(checkSubsequence(A, B));
    }

    private static int checkSubsequence(String A, String B) {
        int i = 0, j = 0;
        while (i < A.length() && j < B.length()) {
            if (A.charAt(i) == B.charAt(j)) {
                i++;
            }
            j++;
        }
        return i == A.length() ? 1 : 0;
    }
}
```

Output:

```
Input: gksrek geeksforgeeks
Output: 1
```

3. Longest Common Prefix in an Array

Problem statement:

Given an array of N strings, find the longest common prefix among all strings present in the array.

Example 1:

```
Input:
N = 4
arr[] = {geeksforgeeks, geeks, geek, geezer}
Output: gee
```

> Explanation: "gee" is the longest common prefix in all the given strings.

Example 2:

```
Input:
N = 2
arr[] = {hello, world}
Output: -1
```

> Explanation: There's no common prefix in the given strings.

Constraints:

```
1≤ N ≤ 10^3
1≤ |arri| ≤ 10^3
```

Analysis:

1. Initialize the longest common prefix as the first string in the array.
2. Traverse the array of strings.
3. For each string, find the common prefix with the longest common prefix.
4. Update the longest common prefix with the common prefix.
5. If the longest common prefix becomes an empty string, then return -1.

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class LongestCommonPrefix {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt();
        String[] arr = new String[N];
        for (int i = 0; i < N; i++) {
            arr[i] = sc.next();
        }
        System.out.println(longestCommonPrefix(arr));
    }

    private static String longestCommonPrefix(String[] arr) {
        String lcp = arr[0];
        for (int i = 1; i < arr.length; i++) {
            lcp = commonPrefix(lcp, arr[i]);
            if (lcp.equals("")) {
                return "-1";
            }
        }
        return lcp;
    }

    private static String commonPrefix(String s1, String s2) {
        int i = 0;
        while (i < s1.length() && i < s2.length() && s1.charAt(i) == s2.charAt(i)) {
            i++;
        }
        return s1.substring(0, i);
    }
}
```

Output:

```
Input: 4 geeksforgeeks geeks geek geezer
Output: gee
```

---

4. Reversing the vowels

Problem statement:

Given a string consisting of lowercase english alphabets, reverse only the vowels present in it and print the resulting string.

Example 1:

```
Input:
S = "geeksforgeeks"
Output: geeksforgeeks
```

> Explanation: The vowels are: e, e, o, e, e
>
> Reverse of these is also e, e, o, e, e.

######Example 2: Input: S = "practice" Output: prectica

> Explanation: The vowels are a, i, e.Reverse of these is e, i, a.

Constraints:

```
1<=|S|<=10^5
```

Analysis:

1. Initialize two pointers i and j to the start and end of the string.
2. Traverse the string from both ends.
3. If the character at i is a vowel, then increment i.
4. If the character at j is a vowel, then decrement j.
5. Swap the characters at i and j.
6. Continue till i is less than j.

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class ReverseTheVowels {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String S = sc.next();
        System.out.println(reverseVowels(S));
    }

    private static String reverseVowels(String S) {
        char[] vowels = {'a', 'e', 'i', 'o', 'u'};
        int i = 0, j = S.length() - 1;
        char[] s = S.toCharArray();
        while (i < j) {
            if (isVowel(s[i], vowels) && isVowel(s[j], vowels)) {
                char temp = s[i];
                s[i] = s[j];
                s[j] = temp;
                i++;
                j--;
            } else if (isVowel(s[i], vowels)) {
                j--;
            } else {
                i++;
            }
        }
        return new String(s);
    }

    private static boolean isVowel(char c, char[] vowels) {
        for (char vowel : vowels) {
            if (c == vowel) {
                return true;
            }
        }
        return false;
    }
}
```

Output:

```
practice
prectica
```

---

5. Integer To Roman

Problem Statement:

Given an integer A, convert it to a roman numeral, and return a string corresponding to its roman numeral version

Input 1:

```
A = 5
```

Output 1:

```
"V"
```

Input 2:

```
A = 14
```

Output 2:

```
"XIV"
```

Constraints

```
1 <= A <= 3999
```

Analysis:

1. Create two arrays, one for roman numerals and the other for their corresponding values.
2. Traverse the values array from the end.
3. If the value is less than or equal to A, then append the roman numeral to the result and subtract the value from A.
4. Continue till A becomes 0.

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class IntegerToRoman {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int A = sc.nextInt();
        System.out.println(intToRoman(A));
    }

    private static String intToRoman(int A) {
        String[] roman = {"I", "IV", "V", "IX", "X",
                          "XL", "L", "XC", "C", "CD",
                          "D", "CM", "M"};
        int[] values = {1, 4, 5, 9, 10, 40, 50, 90, 100,
                          400, 500, 900, 1000};
        StringBuilder result = new StringBuilder();
        for (int i = values.length - 1; i >= 0; i--) {
            while (A >= values[i]) {
                result.append(roman[i]);
                A -= values[i];
            }
        }
        return result.toString();
    }
}
```

Output:

```
Input: 14
Output: XIV
```

6. Minimum Number of Lamps

Problem statement:

You are given a string 'S' containing dots(.) and asterisks (*) only, where the dot represents free spaces, and the asterisk denotes lamps. A lamp can lighten up its own cell as well as its immediate neighboring cells. You need to determine the minimum number of extra lamps that have to be installed at some free spaces in the string so that the whole string will be illuminated i.e. all the indices of the string can have access to the light of some lamp.

> Note: If a lamp is present at index 'I', then it illuminates index 'I' - 1, 'I' and 'I' + 1.

If a lamp is present at index 0, then it illuminates only 0 and 1. Given that the length of the string is greater than or equal to 2. If a lamp is present at the last index, then it illuminates the last and the second last index, given that the length of the string is greater than or equal to 2. The length of each string is guaranteed to be at least 1.

Sample Input 1:

```
1
......
```

Sample Output 1:

```
2
```

> Explanation of Sample Input 1
> the given string, since no lamps are installed initially if we install one lamp at index 2, it will light up indices 1,2,3 and another lamp at index 5, it will light up index 4, 5, 6. Hence the answer is 2.

Sample Input 2:

```
3
*.
```

```
.*
*.*..
```

Sample Output 2:

```
0
0
1
```

Constraints:

```
1 <= 'T' <= 100
1 <= |'S'| <= 5 * 10 ^ 4
```

Analysis:

1. Initialize a variable count to 0.
2. Traverse the string S.
3. If the character at index i is '.', then check if the character at index i + 1 is '.'.
4. If true, then increment count and increment i by 2.
5. If the character at index i is '*', then increment i by 1.
6. Continue till the end of the string.

Code:

```
package com.self_practice;

import java.util.Scanner;
```

```
public class MinimumLamps {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int T = sc.nextInt();
        for (int i = 0; i < T; i++) {
            String S = sc.next();
            System.out.println(minimumLamps(S));
        }
    }

    private static int minimumLamps(String S) {
        int count = 0;
        for (int i = 0; i < S.length(); i++) {
            if (S.charAt(i) == '.') {
                if (i + 1 < S.length() && S.charAt(i + 1) == '.') {
                    count++;
                    i++;
                }
            }
        }
        return count;
    }
}
```

Output:

```
Input: *.*
Output: 0
```

7. Bulls and Cows

Problem Statement:

You are playing the Bulls and Cows game with your friend.You write down a secret number and ask your friend to guess what the number is. When your friend makes a guess, you provide a hint with the following info: The number of "bulls", which are digits in the guess that are in the correct position. The number of "cows", which are digits in the guess that are in your secret number but are located in the wrong position. Specifically, the non-bull digits in the guess that could be rearranged such that they become bulls. Given the secret number secret and your friend's guess guess, return the hint for your friend's guess. The hint should be formatted as "xAyB", where x is the number of bulls and y is the number of cows. Note that both secret and guess may contain duplicate digits.

Example Input

```
Input 1:
secret = "1807", guess = "7810"
Input 2:
secret = "1123", guess = "0111"
```

Example Output

```
Output 1:
"1A3B"
Output 2:
"1A1B"
```

Example Explanation

> Explanation 1:

> Bulls are connected with a '|':
>
> ```
> "1807"
>     |
> "7810"
> ```
>
> Explanation 2:
>
> Bulls are connected with a '|'
>
> ```
> "1123" "1123"
>   |  or  |
> "0111" "0111"
> ```
>
> Note that only one of the two unmatched 1s is counted as a cow since the non-bull digits can only be rearranged to allow one 1 to be a bull.

Analysis:

1. Initialize two arrays secretCount and guessCount to store the count of each digit in secret and guess.
2. Initialize two variables bulls and cows to 0.
3. Traverse the secret and guess strings.
4. If the characters at the same index are equal, then increment bulls.
5. Increment cows by the minimum of the count of the current digit in secret and guess.
6. Decrement the count of the current digit in secret and guess.
7. Return the hint in the format "xAyB".

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class BullsAndCows {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String secret = sc.next();
        String guess = sc.next();
        System.out.println(getHint(secret, guess));
    }

    private static String getHint(String secret, String guess) {
        int[] secretCount = new int[10];
        int[] guessCount = new int[10];
        int bulls = 0, cows = 0;
        for (int i = 0; i < secret.length(); i++) {
            if (secret.charAt(i) == guess.charAt(i)) {
                bulls++;
            } else {
                secretCount[secret.charAt(i) - '0']++;
                guessCount[guess.charAt(i) - '0']++;
            }
        }
        for (int i = 0; i < 10; i++) {
            cows += Math.min(secretCount[i], guessCount[i]);
        }
        return bulls + "A" + cows + "B";
    }
}
```

Output:

```
Input: 1807 7810
Output: 1A3B
```

---

8. Recursively remove all adjacent duplicates

Problem Statement:

Given a string s, remove all its adjacent duplicate characters recursively. Note: For some test cases, the resultant string would be an empty string. In that case, the function should return the empty string only.

Example 1:

```
Input:
S = "geeksforgeek"
Output: "gksforgk"
```

> Explanation:
>
> g(ee)ksforg(ee)k -> gksforgk

Example 2:

```
Input:
S = "abccbccba"
Output: ""
```

> Explanation:
>
> ab(cc)b(cc)ba->abbba->a(bbb)a->aa->(aa)->""(empty string)

Analysis:

1. Initialize a variable i to 0.
2. Traverse the string S.
3. If the character at index i is equal to the character at index i + 1, then increment i.
4. Otherwise, append the character at index i to the result.
5. Continue till the end of the string.

Code:

```
package com.self_practice;

import java.util.Scanner;

public class RemoveAdjacentDuplicates {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String S = sc.next();
        System.out.println(removeDuplicates(S));
    }
    private static String removeDuplicates(String S) {
        StringBuilder result = new StringBuilder();
        int i = 0;
        while (i < S.length()) {
            if (i + 1 < S.length() && S.charAt(i) == S.charAt(i + 1)) {
                i++;
            } else {
                result.append(S.charAt(i));
            }
            i++;
        }
        return result.toString();
    }
}
```

Output:

```
Input: geeksforgeek
Output: gksforgk
```

---

9. Convert number to words

Problem statement : You are given an Integer ‘N’ you have to convert the integer to words. For example you are given integer N = 2234 then you have to return the string “two thousand two hundred and thirty four”. Detailed explanation (Input/output format, Notes, Images)

Sample Input 1 :

```
3
1234
45821
555093
```

Sample Output 1 :

```
one thousand two hundred and thirty four
forty five thousand eight hundred and twenty one
five lakh fifty five thousand and ninety three
```

> Explanation of Sample Input 1 : For the first test case: The given integer is 1234 we can see that in words it is represented as “one thousand two hundred and thirty four”. For the second test case: The given integer is 45821; we can see that in words it is represented as “forty five thousand eight hundred and twenty-one”. For the third test case The given integer is 555093; we can see that in words it is represented as “five lakh fifty five thousand and ninety three”.

Sample Input 2 :

```
3
99999
1000
30000
```

Sample Output 2 :

```
ninety nine thousand nine hundred and ninety nine
one thousand
thirty thousand
```

Constraints :

```
1 <= T <= 100
1 <= N <= 999999999
```

Analysis:

1. Create arrays for units, tens, and hundreds.
2. Initialize a variable result to an empty string.
3. If the number is greater than 999999, then append the number in words to the result.
4. If the number is greater than 999, then append the number in words to the result.
5. If the number is greater than 99, then append the number in words to the result.
6. If the number is greater than 19, then append the number in words to the result.
7. If the number is greater than 9, then append the number in words to the result.
8. Append the number in words to the result.

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class NumberToWords {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int T = sc.nextInt();
        for (int i = 0; i < T; i++) {
            int N = sc.nextInt();
            System.out.println(numberToWords(N));
        }
    }

    private static String numberToWords(int N) {
        String[] units = {"", "one", "two", "three",
                        "four", "five", "six", "seven",
                        "eight", "nine"};
        String[] tens = {"", "ten", "twenty", "thirty",
                        "forty", "fifty", "sixty", "seventy",
                        "eighty", "ninety"};
        String[] hundreds = {"", "one hundred", "two hundred",
                            "three hundred", "four hundred", "five hundred",
                            "six hundred", "seven hundred", "eight hundred",
                            "nine hundred"};
        String[] thousands = {"", "thousand", "lakh", "crore"};
        StringBuilder result = new StringBuilder();
        int i = 0;
        while (N > 0) {
            if (N % 1000 != 0) {
                result.insert(0, thousands[i] + " ");
                result.insert(0, convertToWords(N % 1000, units, tens, hundreds));
            }
            N /= 1000;
            i++;
        }
        return result.toString().trim();
    }

    private static String convertToWords(int N, String[] units, String[] tens, String[] hund
        StringBuilder result = new StringBuilder();
        if (N >= 100) {
            result.append(hundreds[N / 100] + " ");
            N %= 100;
        }
        if (N >= 20) {
            result.append(tens[N / 10] + " ");
            N %= 10;
        }
        if (N > 0) {
            result.append(units[N] + " ");
        }
        return result.toString();
    }
}
```

Output:

```
2
1203
one thousand two hundred three
1432
one thousand four hundred thirty two
```

---

10. Last Substring in Lexicographical Order

Problem Statement:

Given a string s, return the last substring of s in lexicographical order.

Example 1:

```
Input: s = "abab"
Output: "bab"
```

> Explanation: The substrings are ["a", "ab", "aba", "abab", "b", "ba", "bab"]. The lexicographically maximum substring is "bab".

Example 2:

```
Input: s = "leetcode"
Output: "tcode
```

Analysis:

1. Initialize a variable maxChar to the first character of the string.
2. Traverse the string from the second character.
3. If the character at index i is greater than maxChar, then update maxChar.
4. Initialize a variable maxString to an empty string.
5. Traverse the string from the first character.
6. If the character at index i is equal to maxChar, then append the character to maxString.
7. Return maxString.

Code:

```
package com.self_practice;

import java.util.Scanner;

public class LastSubstring {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s = sc.next();
        System.out.println(lastSubstring(s));
    }

    private static String lastSubstring(String s) {
        char maxChar = s.charAt(0);
        for (int i = 1; i < s.length(); i++) {
            if (s.charAt(i) > maxChar) {
                maxChar = s.charAt(i);
            }
        }
        StringBuilder maxString = new StringBuilder();
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == maxChar) {
                maxString.append(s.charAt(i));
            }
        }
        return maxString.toString();
    }
}
```

Output:

```
Input: leetcode
Output: tcode
```