<C*deVita/>
The TCS Global Coding Contest

Abhishek Kumar

05 Hr 50 Min 49 Sec
Hover mouse for exact Contest end time

- Guidelines
- Coding Area
- Public Testcase Submissions
- Private Testcase Submissions
- Unevaluated Submissions
- Feedback Form
- Dashboard
- Graphs

Coding Area

A B C D E F G H

ONLINE EDITOR (E)

Distinct SubString

— Problem Description

Given a string and a set of rules, process the string in such a way, that it produces some unique substrings. Print all the longest possible substrings with distinct characters, following the below mentioned rules:

- String should be processed from left to right
- Substring should be longest possible consecutive stream of different characters (longest substring). As soon as a repeating character is encountered, consider the already processed characters as the longest substring achieved in that pass
- Once a pass is over, the next pass begins from the next index that was used in the previous pass. Simply put, first pass will always begin from first character, second pass will begin from second character, so on and so forth
- Just to make it explicit - a subsequent pass cannot process characters whose index is smaller than the beginning index of the current pass. For example, second pass cannot process first character, third pass cannot process first and second characters, so on and so forth
- A longest substring can be of two types - a *Proper substring* and an *Improper substring*
- A proper substring is one which satisfies two criteria

    1. All characters in a proper substring are fully contained in a previously identified longest substring and in same order

    2. Also, indices of proper substring are those that are completely overlapped by previously identified longest substring and in same order

        - E.g. if string is "abcdc", the first longest substring is "abcd" and its indices are from 1 to 4 (assume 1-based index)

        - Now, "bcd" is a proper substring of "a**bcd**" because not only "bcd" is completely contained in "a**bcd**", but also indices of "bcd" are 2 to 4, which completely lie in the range 1 - 4

- An Improper substring is one which satisfies only first criteria above.
    - E.g. Consider string "abcdc**bcd**", the first longest substring is "abcd" and its indices are from 1 to 4 (assume 1-based index)
    - Now, "**bcd**" at index 6 to 8 is an improper substring of "abcd" because even though it satisfies the first criteria of proper substring, it does not satisfy the second criteria
- In order to capture all possible longest substrings, ignore Proper substrings and retain Improper substrings
- Additional restrictions that exist on which longest substrings can be retained are as follows
    - o When a longest substring is broken by a repeating character which is same as the first character of the substring, then two rules apply
        - If that substring is the first longest substring then retain that substring
        - Else, ignore that substring
        - Example - Consider string "abcdabca" - first longest substring "abcd" which is broken by 'a', is retained. However, second pass which generates second longest substring "bcda" is to be ignored because the substring starts and is also broken by the same character 'b'. Same logic applies for third longest substring "cdab", which is to be ignored
- When processing is complete, arrange all longest possible substring(s) in lexicographical order and print it as output

— Constraints

0 < N <= 10000

String consist of all lowercase characters {a - z}

— Input

First line contains an integer N which denotes the length of string

Second line contains the actual string

— Output

Print all the longest possible substring(s) with distinct characters, delimited by space character, while maintaining the rules mentioned above

— Time Limit

1

— Examples

Example 1

Input

8

abcdcbac

Output

abcd bac dcba

Explanation:

String: abcdcbcd

For illustration purpose and better understanding, substrings of interest are marked in **bold** and are underlined

- In the first pass, longest substring is "abcd" (**abcd**cbac). Since there is 'c' which is already in the substring, we retain only "abcd" and move forward.
- In the second pass, longest substring is "bcd" (a**bcd**cbac). Since the next character is 'c' and "bcd" is a proper substring of previously captured longest substring "abcd", we discard this substring.
- In the third pass, longest substring is "cd" (ab**cd**cbac). Since the next character is 'c' and "cd" is already a proper substring of "abcd", we discard this substring also. Also, substring "cd" is discarded because, the starting character of this substring and the first repetitive character is same. Hence "cd" is discarded because of two rules.
- In the fourth pass, longest substring is "dcba" (abc**dcba**c). Since the next character is 'c' and this substring is not contained in any previously retained longest substring, we retain "dcba" and move forward.
- In the fifth pass, longest substring is "cba" (abcd**cba**c), Since the next character is 'c' and "cba" is a proper substring of "dcba", we discard this substring.
- In the sixth pass, longest substring is "bac" (abcdc**bac**). We consider this substring because it is not a part of any previously retained longest substrings.
- In the seventh pass, longest substring is "ac" (abcdcb**ac**). Since "ac" is a proper substring of "bac", we discard this substring.
- In the eighth pass, longest substring is "c" (abcdcba**c**). Since "c" is a proper substring of "bac", we discard this substring too.

Since the length of the string is 8, we made eight passes over the string to compute longest substrings. Notice that in the current pass, all the characters whose index is lower than the index of the beginning character from where the current pass begins, are not considered.

Hence, final output with retained longest substrings sorted in lexicographical order is - "abcd bac dcba".

Example 2

Input

8

abcdcbcd

Output

abcd bcd dcb

Explanation:

String: abcdcbcd

For illustration purpose and better understanding, substrings of interest are marked **bold** and underlined

- In the first pass, longest substring is "abcd" (**abcd**cbcd). Since there is 'c' which is already in the substring, we retain only "abcd" and move forward.
- In the second pass, longest substring is "bcd" (a**bcd**cbcd). Since the next character is 'c' and "bcd" is a proper substring of previously captured longest substring "abcd", we discard this substring.
- In the third pass, longest substring is "cd" (ab**cd**cbcd). Since the next character is 'c' and "cd" is already a proper substring of "abcd", we discard this substring also.
- In the fourth pass, longest substring is "dcb" (abc**dcb**cd). Since the next character is 'c' and this substring is not contained in any previously retained longest substring, we retain "dcb" and move forward.
- In the fifth pass, longest substring is "cb" (abcd**cb**cd), Since the next character is 'c' and also "cb" is a proper substring of "dcb", we discard this substring.

- In the sixth pass, longest substring is "bcd" (abcdc**bcd**). We consider this substring because although it is a part of previously retained longest substring "abcd", it is an Improper substring. Hence, we retain this substring.
- In the seventh pass, longest substring is "cd" (abcdcb**cd**). Since "cd" is a proper substring of "bcd", we discard this substring.
- In the eighth pass, longest substring is "d" (abcdcbc**d**). Since "d" is a proper substring of "bcd", we discard this substring too.

Since the length of the string is 8, we made eight passes over the string to compute longest substrings. Notice that in the current pass, all the characters whose index is lower than the index of the beginning character from where the current pass begins, are not considered.

Hence, final output with retained longest substrings sorted in lexicographical order is - "abcd bcd dcb".

Upload Solution [Question : E]

☐ I, **abhishek kumar** confirm that the answer submitted is my own.

☐ Took help from online sources (attributions)

Choose a File ...