Recursively remove all adjacent duplicates

Given a string, recursively remove adjacent duplicate characters from string. The output string should not have any adjacent duplicates. See following examples.

```
Input:  azxxzy
Output: ay
First "azxxzy" is reduced to "azzy". The string "azzy" contains duplicates,
so it is further reduced to "ay".

Input: geeksforgeeg
Output: gksfor
First "geeksforgeeg" is reduced to "gksforgg". The string "gksforgg" contains
duplicates, so it is further reduced to "gksfor".

Input: caaabbbaacdddd
Output: Empty String

Input: acaaabbbacdddd
Output: acac
```

A simple approach would be to run the input string through multiple passes. In every pass remove all adjacent duplicates from left to right. Stop running passes when there are no duplicates. The worst time complexity of this method would be O(n^2).

We can remove all duplicates in O(n) time.
1) Start from the leftmost character and remove duplicates at left corner if there are any.
2) The first character must be different from its adjacent now. Recur for string of length n-1 (string

Google™ Custom Search

GeeksforGeeks

52,731 people like GeeksforGeeks.

Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion
Geometric Algorithms

without first character).
3) Let the string obtained after reducing right substring of length n-1 be *rem_str*. There are three possible cases
……..**a)** If first character of *rem_str* matches with the first character of original string, remove the first character from *rem_str*.
……..**b)** Else if the last removed character in recursive calls is same as the first character of the original string. Ignore the first character of original string and return *rem_str*.
……..**c)** Else, append the first character of the original string at the beginning of *rem_str*.
4) Return *rem_str*.

Following is C++ implementation of the above algorithm.

```
#include <iostream>
#include <string.h>
using namespace std;

// Recursively removes adjacent duplicates from str and returns new
// string. las_removed is a pointer to last_removed character
char* removeUtil(char *str, char *last_removed)
{
    // If length of string is 1 or 0
    if (str[0] == '\0' || str[1] == '\0')
        return str;

    // Remove leftmost same characters and recur for remaining string
    if (str[0] == str[1])
    {
        *last_removed = str[0];
        while (str[1] && str[0] == str[1])
            str++;
        str++;
        return removeUtil(str, last_removed);
    }

    // At this point, the first character is definiotely different fro
    // adjacent. Ignore first character and recursively remove characte
    // remaining string
    char* rem_str = removeUtil(str+1, last_removed);

    // Check if the first character of the rem_string matches with the
    // first character of the original string
    if (rem_str[0] && rem_str[0] == str[0])
    {
        *last_removed = str[0];
```

Popular Posts

All permutations of a given string
Memory Layout of C Programs
Understanding “extern” keyword in C
Median of two sorted arrays
Tree traversal without recursion and without stack!
Structure Member Alignment, Padding and Data Packing
Intersection point of two Linked Lists
Lowest Common Ancestor in a BST.
Check if a binary tree is BST or not
Sorted Linked List to Balanced BST

```
        return (rem_str+1); // Remove first character
    }

    // If remaining string becomes empty and last removed character
    // is same as first character of original string.  This is needed
    // for a string like "acbbcddc"
    if (rem_str[0] == '\0' && *last_removed == str[0])
         return rem_str;

    // If the two first characters of str and rem_str don't match, app
    // first character of str before the first character of rem_str.
    rem_str--;
    rem_str[0] = str[0];
    return rem_str;
}

char *remove(char *str)
{
    char last_removed = '\0';
    return removeUtil(str, &last_removed);
}

// Driver program to test above functions
int main()
{
    char str1[] = "geeksforgeeg";
    cout << remove(str1) << endl;

    char str2[] = "azxxxzy";
    cout << remove(str2) << endl;

    char str3[] = "caaabbbaac";
    cout << remove(str3) << endl;

    char str4[] = "gghhg";
    cout << remove(str4) << endl;

    char str5[] = "aaaacddddcappp";
    cout << remove(str5) << endl;

    char str6[] = "aaaaaaaaaa";
    cout << remove(str6) << endl;

    char str7[] = "qpaaaaadaaaaadprq";
    cout << remove(str7) << endl;

    char str8[] = "acaaabbbacdddd";
```

```
    cout << remove(str8) << endl;

    char str9[] = "acbbcddc";
    cout << remove(str9) << endl;

    return 0;
}
```

Output:

```
gksfor
ay

g
a

qrq
acac
a
```

Time Complexity: The time complexity of the solution can be written as T(n) = T(n-k) + O(k) where n is length of the input string and k is the number of first characters which are same. Solution of the recurrence is O(n)

Thanks to **Prachi Bodke** for suggesting this problem and initial solution. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Recent Comments

affiszerv Your example has two 4s on row 3, that's why it...

Backtracking | Set 7 (Sudoku) · 13 minutes ago

RVM Can someone please elaborate this Qs from above...

Flipkart Interview | Set 6 · 33 minutes ago

Vishal Gupta I talked about as an Interviewer in general,...

Software Engineering Lab, Samsung Interview | Set 2 · 33 minutes ago

@meya Working solution for question 2 of 4f2f round....

Amazon Interview | Set 53 (For SDE-1) · 1 hour ago

sandeep void rearrange(struct node *head) {...

Given a linked list, reverse alternate nodes and append at the end · 2 hours ago

Neha I think that is what it should return as, in...

Find depth of the deepest odd level leaf node · 2 hours ago

Related Tpoics:

- Printing Longest Common Subsequence
- Suffix Array | Set 2 (nLogn Algorithm)
- Rearrange a string so that all same characters become d distance away
- Find the first non-repeating character from a stream of characters
- Dynamic Programming | Set 33 (Find if a string is interleaved of two other strings)
- Remove “b” and “ac” from a given string
- Dynamic Programming | Set 29 (Longest Common Substring)
- Write your own atoi()

28 Tweet 3 1

Writing code in comment? Please use **ideone.com** and share the link here.