Length of the smallest sub-string consisting of maximum distinct characters

Tuesday, December 14, 2021 8:50 AM

Smallest window that contains all characters of string itself

- Difficulty Level : Hard
- Last Updated : 05 Jul, 2021

Given a string, find the smallest window length with all distinct characters of the given string. For eg. str = “aabcbcdbca”, then the result would be 4 as of the smallest window will be “dbca” .

Examples:

```
Input: aabcbcdbca
Output: dbca
Explanation:
Possible substrings= {aabcbcd, abcbcd,
bcdbca, dbca....}
Of the set of possible substrings 'dbca'
is the shortest substring having all the
distinct characters of given string.
Input: aaab
Output: ab
Explanation:
Possible substrings={aaab, aab, ab}
Of the set of possible substrings 'ab'
is the shortest substring having all
the distinct characters of given string.
```

Recommended: Please solve it on “***PRACTICE***” first, before moving on to the solution.

Solution: Above problem states that we have to find the smallest window that contains all the distinct characters of the given string even if the smallest string contains repeating elements.

For example, in “aabcbcdb”, the smallest string that contains all the characters is “abcbcd”.

Method 1: This is the Brute Force method of solving the problem using HashMap.

- **Approach :** For solving the problem we first have to find out all the distinct characters present in the string. This can be done using a **HashMap**. The next thing is to generate all the possible substrings. This follows by checking whether a substring generated has all the required characters(stored in the hash_map) or not. If yes, then compare its length with the minimum substring length which follows the above constraints, found till now.
  HashMap: HashMap is a part of Java’s collection since Java 1.2. It provides the basic implementation of the Map interface of Java. It stores the data in (Key, Value) pairs. To access a value one must know its key. HashMap is known as HashMap because it uses a technique called Hashing. Hashing is a technique of converting a large String to small String that represents the same String. A shorter value helps in indexing and faster searches. HashSet also uses HashMap internally. It internally uses a link list to store key-value pairs already explained in HashSet in detail and further articles.

- **Algorithm :**

1. Store all distinct characters of the given string in a hash_map.
2. Take a variable count and initialize it with value 0.
3. Generate the substrings using two pointers.

4. **Now check whether generated substring is valid or not-:**

1. As soon we find that the character of the substring generated has not been encountered before, increment count by **1**.
2. We can use a visited array of **max_chars** size to find whether the current character has been encountered before or not.
3. If count is equal to equal to size of hash_map the substring generated is valid
4. If it is a valid substring, compare it with the minimum length substring already generated.

- **Pseudo Code:**

```
maphash_map;
for ( i=0 to str.length())
hash_map[str[i]]++;//finding all distinct characters of string
minimum_size=INT_MAX
Distinct_chars=hash_map.size()
for(i=0 to str.length())
count=0;
sub_str="";
visited[256]={0};
 for(j=i to n)
   sub_str+=str[j]
   if(visited[str[j]]==0)
   count++
   visited[str[j]]=1;
   if(count==Distinct_chars)
   end loop
if(sub_str.length()<minimum_size&&
count==Distinct_chars)
ans=sub_str;

return ans
```

- **Implementation:**

From <https://www.geeksforgeeks.org/smallest-window-contains-characters-string/>

Java

```java
import java.io.*;
import java.util.*;

// Java program to find the smallest
// window containing all characters
// of a pattern.
class GFG
{

    // Function to find smallest window containing
    // all distinct characters
    public static String findSubString(String str)
    {
        int n = str.length();

        // Count all distinct characters.
        int dist_count = 0;
        HashMap<Character, Integer> mp = new HashMap<>();
        for (int i = 0; i < n; i++)
        {
```

```
            if (mp.containsKey(str.charAt(i)))
            {
                Integer a = mp.get(str.charAt(i));
                mp.put(str.charAt(i),a+1);
            }
          else
          {
                mp.put(str.charAt(i), 1);
            }
        }
        dist_count = mp.size();
        int size = Integer.MAX_VALUE;
        String res = "";

        // Now follow the algorithm discussed in below
        for (int i = 0; i < n; i++)
        {
            int count = 0;
            int visited[] = new int[256];
            for(int j = 0; j < 256; j++)
              visited[j] = 0;
            String sub_str = "";
            for (int j = i; j < n; j++)
            {
                if (visited[str.charAt(j)] == 0)
                {
                    count++;
                    visited[str.charAt(j)] = 1;
                }
                sub_str += str.charAt(j);
                if (count == dist_count)
                    break;
            }
            if (sub_str.length() < size && count == dist_count)
            {
                res = sub_str;
                size=res.length();
            }
        }
        return res;
    }

  // Driver code
    public static void main (String[] args)
    {
        String str = "aabcbcdbca";
        System.out.println("Smallest window containing all distinct"+
                " characters is: "+ findSubString(str)) ;
    }
}

// This code is contributed by Manu Pathria
```

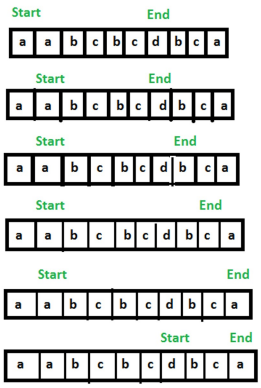
Output

```
Smallest window containing all distinct characters is: dbca
```

- **Complexity Analysis:**
- **Time Complexity:** O(N^2).
  This time is required to generate all possible sub-strings of a string of length “N”.
- **Space Complexity:** O(N).
  As a **hash_map** has been used of size N.

Method 2: Here we have used Sliding Window technique to arrive at the solution. This technique shows how a nested for loop in few problems can be converted to single for loop and hence reducing the time complexity.

- **Approach:** Basically a window of characters is maintained by using two pointers namely **start** and **end**.
  These **start** and **end** pointers can be used to shrink and increase the size of window respectively. Whenever the window contains all characters of given string, the window is shrinked from left side to remove extra characters and then its length is compared with the smallest window found so far.
  If in the present window, no more characters can be deleted then we start increasing the size of the window using the **end** until all the distinct characters present in the string are also there in the window. Finally, find the minimum size of each window.



- **Algorithm :**

5. Maintain an array **(visited)** of maximum possible characters (256 characters) and as soon as we find any in the string, mark that index in the array **(this is to count all distinct characters in the string).**
6. Take two pointers **start** and **end** which will mark the start and end of window.
7. Take a **counter=0** which will be used to count distinct characters in the window.
8. Now start reading the characters of the given string and if we come across a character which has not been visited yet increment the counter by **1**.
9. If the **counter** is equal to total number of distinct characters, Try to shrink the window.
10. **For shrinking the window -:**
11. If the **frequency** of character at start pointer is **greater than 1** increment the pointer as it is redundant.
12. Now compare the length of present window with the minimum window length.

- **Implementation:**

From <https://www.geeksforgeeks.org/smallest-window-contains-characters-string/>

Java

```java
// Java program to find the smallest window containing
// all characters of a pattern.
import java.util.Arrays;
public class GFG {

    static final int MAX_CHARS = 256;

    // Function to find smallest window containing
    // all distinct characters
    static String findSubString(String str)
```

```
{
    int n = str.length();

    // if string is empty or having one char
    if (n <= 1)
        return str;

    // Count all distinct characters.
    int dist_count = 0;

    boolean[] visited = new boolean[MAX_CHARS];
    Arrays.fill(visited, false);
    for (int i = 0; i < n; i++) {
        if (visited[str.charAt(i)] == false) {
            visited[str.charAt(i)] = true;
            dist_count++;
        }
    }

    // Now follow the algorithm discussed in below
    // post. We basically maintain a window of
    // characters that contains all characters of given
    // string.
    int start = 0, start_index = -1;
    int min_len = Integer.MAX_VALUE;

    int count = 0;
    int[] curr_count = new int[MAX_CHARS];
    for (int j = 0; j < n; j++) {
        // Count occurrence of characters of string
        curr_count[str.charAt(j)]++;

        // If any distinct character matched,
        // then increment count
        if (curr_count[str.charAt(j)] == 1)
            count++;

        // if all the characters are matched
        if (count == dist_count) {
            // Try to minimize the window i.e., check if
            // any character is occurring more no. of
            // times than its occurrence in pattern, if
            // yes then remove it from starting and also
            // remove the useless characters.
            while (curr_count[str.charAt(start)] > 1) {
                if (curr_count[str.charAt(start)] > 1)
                    curr_count[str.charAt(start)]--;
                start++;
            }

            // Update window size
            int len_window = j - start + 1;
            if (min_len > len_window) {
                min_len = len_window;
                start_index = start;
            }
        }
    }
    // Return substring starting from start_index
    // and length min_len
    return str.substring(start_index,
```

```
                                start_index + min_len);
    }

    // Driver code
    public static void main(String args[])
    {
        String str = "aabcbcdbca";
        System.out.println(
            "Smallest window containing all distinct"
            + " characters is: " + findSubString(str));
    }
}
// This code is contributed by Sumit Ghosh
```

From <https://www.geeksforgeeks.org/smallest-window-contains-characters-string/>