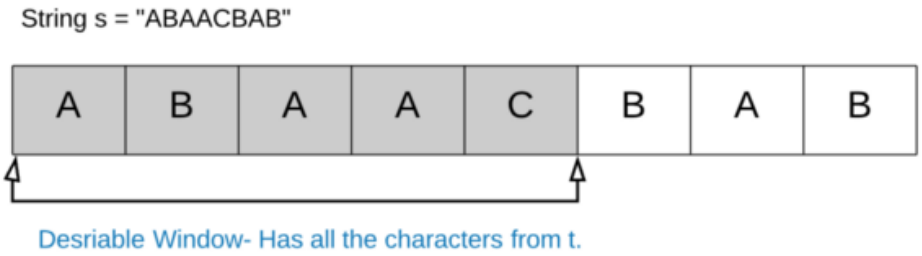
Published in Algo Shaft

Kode Shaft Follow
Aug 2, 2019 · 2 min read · Listen

Save

Solve minimum window substring problem

In this post we are gonna discuss how to solve minimum window substring problem using Sliding window principle



Problem Statement :

> *Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n).*
>
> *Example:*
>
> *Input: S = "ADOBECODEBANC", T = "ABC"*
>
> *Output: "BANC"*
>
> *Note:If there is no such window in S that covers all characters in T, return the empty string "".*
>
> *If there is such window, you are guaranteed that there will always be only one unique minimum window in S.*

Step by step solution to the problem :

As in this problem we have to ***find the minimum window ,*** first paradigm which comes to our mind to solve this is Sliding window.

Base Case:

if either s or t is null then we can't find any common window between them.

```
if(s==null || t==null) return "";
```

Main Algorithm:

first lets keep all the character count of t in a map like below code so that we can check a character's availability from this map easily in O(1).

```
int[] map=new int[128];
for(Character c:t.toCharArray())map[c]++;
```

Let's take two pointers start and end to iterate through s and initialize them with 0 along with some other variables.

```
int start = 0, end = 0, minStart = 0, minLen = Integer.MAX_VALUE, counter = t.length();
```

Now we iterate over s using end pointer and validate if current character is present in t or not by checking **map[arr[end]]** > 0. If it's present decrement the counter to signify that we got one character match and (count-1) character to be matched.Also we decrement **map[arr[end]]** to signify that we have traversed current character using end pointer.

In this process if we get **count==0**, that means we got an desired window, so replace this window as result window if this is the minimum window.

```
1
2           while(end<arr.length){
3                if (map[arr[end]] > 0) counter--;
4                map[arr[end]]--;
5                end++;
6                while(counter==0){
7                    if((end-start)<minLen){
8                        minLen=end-start;
9                        minStart=start;
10                   }
11                   map[arr[start]]++;
12                   if(map[arr[start]]>0)counter++;
13                   start++;
14               }
15          }
```

MinimumWindowSubstring.java hosted with ❤ by GitHub view raw

Two important things to note here is :

1. `if (map[arr[end]] > 0) counter – ;`
   `end++;`

this signifies we got match of char in s with char in t, so we have counter-1 char to match in the current window.

2. `if(map[arr[start]]>0)counter++;`
   `start++;`

this signifies that after restoring the actual char count in map by traversing through start pointer, if count>0 then increment counter and start as we are removing this char from current window, and moving the window forward.

The complete code for this problem can be found in https://github.com/GyanTech877/algorithms/blob/master/slidingwindow/SlidingWindowMaximum.java

Happy Learning 😎