Leetcode 76. Minimum Window Substring

Raymond Ruan Jul 27, 2019 · 4 min read

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"
```

Approach: Sliding Window

The question asks us to return the minimum window from the string SS which has all the characters of the string TT. Let us call a window `desirable` if it has all the characters from TT.

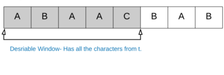
We can use a simple sliding window approach to solve this problem.

In any sliding window-based problem we have two pointers. One right pointer whose job is to expand the current window and then we have the left pointer whose job is to contract a given window. At any point in time only one of these pointers move and the other one remains fixed.

The solution is pretty intuitive. We keep expanding the window by moving the right pointer. When the window has all the desired characters, we contract (if possible) and save the smallest window till now.

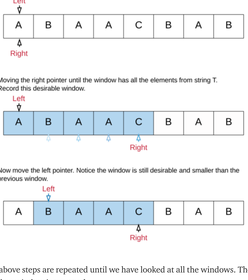
The answer is the smallest desirable window.

For eg. `S = "ABAACBAB" T = "ABC"`. Then our answer window is `"ACB"` and shown below is one of the possible desirable windows.



Algorithm

1. We start with two pointers, left and right initially pointing to the first element of the string SS.
2. We use the right pointer to expand the window until we get a desirable window i.e. a window that contains all of the characters of TT.
3. Once we have a window with all the characters, we can move the left pointer ahead one by one. If the window is still a desirable one we keep on updating the minimum window size.
4. If the window is not desirable anymore, we repeat step $\backslash; 2 step 2$ onwards.



The above steps are repeated until we have looked at all the windows. The smallest window is returned.

```
def minWindow(self, s, t):
 """
 :type s: str
 :type t: str
 :rtype: str
 """


if not t or not s:
 return ""


# Dictionary which keeps a count of all the unique characters in t.
 dict_t = Counter(t)

# Number of unique characters in t, which need to be present in the
desired window.
 required = len(dict_t)

# left and right pointer
 l, r = 0, 0

# formed is used to keep track of how many unique characters in t are
present in the current window in its desired frequency.
 # e.g. if t is "AABC" then the window must have two A's, one B and
one C. Thus formed would be = 3 when all these conditions are met.
 formed = 0

# Dictionary which keeps a count of all the unique characters in the
current window.
 window_counts = {}

# ans tuple of the form (window length, left, right)
 ans = float("inf"), None, None


while r < len(s):

# Add one character from the right to the window
 character = s[r]
 window_counts[character] = window_counts.get(character, 0) + 1

# If the frequency of the current character added equals to the
desired count in t then increment the formed count by 1.
 if character in dict_t and window_counts[character] ==
dict_t[character]:
 formed += 1

# Try and contract the window till the point where it ceases to be
'desirable'.
 while l <= r and formed == required:
 character = s[l]

# Save the smallest window until now.
 if r - l + 1 < ans[0]:
 ans = (r - l + 1, l, r)

# The character at the position pointed by the `left` pointer is no
longer a part of the window.
 window_counts[character] -= 1
 if character in dict_t and window_counts[character] <
dict_t[character]:
 formed -= 1

# Move the left pointer ahead, this would help to look for a new
window.
 l += 1


# Keep expanding the window once we are done contracting.
 r += 1
 return "" if ans[0] == float("inf") else s[ans[1] : ans[2] + 1]
```

Complexity Analysis

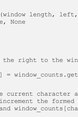
- Time Complexity: $O(|S| + |T|)$ $O(|S|+|T|)$ where $|S|$ and $|T|$ represent the lengths of strings SS and TT. In the worst case we might end up visiting every element of string SS twice, once by left pointer and once by right pointer. $|T|+|T|$ represents the length of string T.
- Space Complexity: $O(|S| + |T|)$ $O(|S|+|T|)$. $|S|$ $|S|$ when the window size is equal to the entire string SS. $|T|+|T|$ when TT has all unique characters.

Jul 27, 2019

Leetcode 84. Largest Rectangle in Histogram

Problem:

Given n non-negative integers representing the histogram's bar height where the width of each bar is 1, find the area of largest rectangle in the histogram.



Above is a histogram where width of each bar is 1, given height = `[2,1,5,6,2,3]`.

Read more · 3 min read