

# shortest substring hackerrank solution

Shortest substring Hackerrank solution is a common problem faced by many aspiring programmers in coding competitions and interviews. The challenge typically involves finding the shortest substring within a given string that contains all the characters of another string. This problem is not only an excellent test of one's string manipulation skills but also a great way to practice the sliding window technique, which is a crucial concept in algorithm design.

In this article, we will explore the problem in detail, discuss various approaches to solve it, and provide a step-by-step explanation of the optimal solution. By the end, you should have a thorough understanding of how to tackle the shortest substring problem effectively.

## Understanding the Problem

To approach the shortest substring Hackerrank solution, it's essential first to understand the requirements of the problem. The typical statement is as follows:

- You are given two strings, `s` and `t`.
- Your task is to find the minimum length substring of `s` that contains all characters (including duplicates) of `t`.

### Example

Consider the following example:

- Input: `s = "ADOBECODEBANC"`, `t = "ABC"`
- Output: `"BANC"`

In this case, the substring `"BANC"` from `"ADOBECODEBANC"` is the shortest one that contains all the

characters `A`, `B`, and `C` from string `t`.

## Key Concepts

To solve this problem effectively, there are several key concepts that one should grasp:

### 1. Character Frequency

Understanding the frequency of characters in both strings is critical. You will need to keep track of how many times each character appears in both `s` and `t`. This can be efficiently done using a hash map or an array.

### 2. Sliding Window Technique

The sliding window technique is a method for iterating through a list or string while maintaining a subset of elements. This can help in dynamically adjusting the window size to find the shortest substring that meets the criteria.

### 3. Two Pointers

Using two pointers is an efficient way to represent the current substring in the sliding window. One pointer will denote the start of the window, and the other will denote the end.

# Approach to Solve the Problem

Now that we understand the problem and the concepts involved, let's delve into a structured approach to solving the shortest substring problem.

## Step 1: Initialize Variables

You will need the following variables:

- Two hash maps (or arrays) to keep track of character frequencies in `t` and the current window in `s`.
- Two pointers `left` and `right` to represent the current window in `s`.
- A variable to track the number of unique characters in `t` that need to be matched.

## Step 2: Expand the Right Pointer

Start with the right pointer at the beginning of the string `s` and expand it to include characters until the window contains all characters of `t`.

## Step 3: Contract the Left Pointer

Once the current window contains all characters of `t`, move the left pointer to minimize the window size while still containing all characters. Update the minimum length and substring whenever a smaller window is found.

## Step 4: Repeat

Continue expanding and contracting until the right pointer has traversed the entire string `s`.

## Step 5: Return the Result

After iterating through the string, return the shortest substring found.

# Implementation

Let's walk through an example implementation of the shortest substring Hackerrank solution in Python:

```
```python
from collections import Counter

def min_window(s: str, t: str) -> str:
    if not t or not s:
        return ""

    dict_t = Counter(t)
    required = len(dict_t)

    l, r = 0, 0
    formed = 0
    window_counts = {}

    ans = float("inf"), None, None

    while r < len(s):
        char = s[r]
        window_counts[char] = window_counts.get(char, 0) + 1

        if char in dict_t and window_counts[char] == dict_t[char]:
            formed += 1

        while l <= r and formed == required:
            char = s[l]
            if r - l + 1 < ans[0]:
```

```
ans = (r - l + 1, l, r)
```

```
window_counts[char] -= 1
```

```
if char in dict_t and window_counts[char] < dict_t[char]:
```

```
formed -= 1
```

```
l += 1
```

```
r += 1
```

```
return "" if ans[0] == float("inf") else s[ans[1]: ans[2] + 1]
```

Example Usage

```
s = "ADOBECODEBANC"
```

```
t = "ABC"
```

```
print(min_window(s, t)) Output: "BANC"
```

```
...
```

Explanation of the Code

- The `Counter` class from the `collections` module is used to count the frequency of characters in `t`.
- The `required` variable tracks how many unique characters from `t` need to be matched.
- The `window_counts` dictionary keeps a count of characters in the current window.
- The outer loop expands the right pointer while the inner loop contracts the left pointer to find the shortest valid substring.
- Finally, the result is returned based on the minimum length found.

## Challenges and Edge Cases

When implementing the shortest substring Hackerrank solution, you may encounter several challenges:

- Empty Strings: Both `s`` and `t`` can be empty. Ensure to handle these cases gracefully.
- No Valid Substring: It is possible that no valid substring exists. Your algorithm should return an appropriate result in such cases.
- Character Case Sensitivity: Be mindful of whether the problem specifies case sensitivity (i.e., `'a'` vs. `'A'`).

## Performance Analysis

The time complexity of this solution is  $O(n)$ , where  $n$  is the length of string `s``. This is because each character is processed at most twice (once by the right pointer and once by the left pointer). The space complexity is  $O(m)$ , where  $m$  is the size of the character set (the number of unique characters in `t``).

## Conclusion

The shortest substring Hackerrank solution is a classic problem that blends string manipulation with efficient algorithms. By mastering the sliding window technique and character frequency counting, you can tackle similar challenges effectively. Practice implementing this solution and consider variations of the problem to further enhance your skills.

In competitive programming and coding interviews, being able to demonstrate a clear understanding of algorithms and data structures is invaluable. The shortest substring problem is just one of many that can help you prepare for your next coding challenge.

## Frequently Asked Questions

## **What is the 'Shortest Substring' problem on HackerRank?**

The 'Shortest Substring' problem on HackerRank requires finding the shortest substring of a given string that contains all the unique characters of that string.

## **What are the key steps to solve the 'Shortest Substring' problem?**

Key steps include identifying unique characters, using two pointers to maintain a sliding window, and checking if the current window contains all unique characters.

## **What data structures are commonly used in the 'Shortest Substring' solution?**

Commonly used data structures include hash maps or dictionaries to keep track of character counts, and arrays or lists to manage the current substring.

## **How do you optimize the solution for the 'Shortest Substring' problem?**

Optimization can be achieved by minimizing the window size as soon as all unique characters are found, and using the sliding window technique to avoid unnecessary checks.

## **Can the 'Shortest Substring' problem be solved in linear time?**

Yes, the problem can be solved in linear time  $O(n)$  by using the sliding window technique combined with hash tables for efficient character counting.

## **What are some common pitfalls to avoid when solving the 'Shortest Substring' problem?**

Common pitfalls include not correctly identifying unique characters, failing to update the window size efficiently, and not handling edge cases like an empty string or strings with fewer unique characters than required.

# **Shortest Substring Hackerrank Solution**

Shortest Substring Hackerrank Solution

## **Related Articles**

- [sign language fist to chin](#)
- [short history of racism](#)
- [significant figures practice worksheet answer key](#)

[Back to Home](#)