Back To Course Home

Grokking the Coding Interview: Patterns for Coding Questions

5% completed

Search Course

Solution Review: Problem Challenge 2

We'll cover the following

String Anagrams (hard)

Given a string and a pattern, find **all anagrams of the pattern in the given string**.

Anagram is actually a **Permutation** of a string. For example, “abc” has the following six anagrams:

1. abc
2. acb
3. bac
4. bca
5. cab

Back To Course Home

Search Course

6. cba

Write a function to return a list of starting indices of the anagrams of the pattern in the given string.

Example 1:

```
Input: String="ppqp", Pattern="pq"
Output: [1, 2]
Explanation: The two anagrams of the pattern in the given string are "pq" and "q
p".
```

Example 2:

```
Input: String="abbcabc", Pattern="abc"
Output: [2, 3, 4]
Explanation: The three anagrams of the pattern in the given string are "bca", "ca
b", and "abc".
```

Solution

This problem follows the **Sliding Window** pattern and is very similar to Permutation in a String (https://www.educative.io/collection/page/5668639101419520/5671464854355968/5401934796161024/). In this problem, we need to find every occurrence of any permutation of the pattern in the string. We will use a list to store the starting indices of the anagrams of the pattern in the string.

Code

Back To Course Home

Grokking the Coding Interview: Patterns for Coding Questions

5% completed

Search Course

Here is what our algorithm will look like, only the highlighted lines have changed from Permutation in a String (https://www.educative.io/collection/page/5668639101419520/5671464854355968/5401934796161024/):

Java | Python3 | C++ | JS

```
      char rightChar = str.charAt(windowEnd);
      // decrement the frequency of the matched cha
      if (charFrequencyMap.containsKey(rightChar))
        charFrequencyMap.put(rightChar, charFrequen
        if (charFrequencyMap.get(rightChar) == 0)
          matched++;
      }

      if (matched == charFrequencyMap.size()) // ha
        resultIndices.add(windowStart);

      if (windowEnd >= pattern.length() - 1) { // s
        char leftChar = str.charAt(windowStart++);
        if (charFrequencyMap.containsKey(leftChar))
          if (charFrequencyMap.get(leftChar) == 0)
            matched--; // before putting the charac
          // put the character back
          charFrequencyMap.put(leftChar, charFreque
        }
      }
    }

    return resultIndices;
  }

  public static void main(String[] args) {
    System.out.println(StringAnagrams.findStringAna
    System.out.println(StringAnagrams.findStringAna
  }
}

```

Run Save Reset

← Back To Course Home

Grokking the Coding Interview: Patterns for Coding Questions

5% completed

Search Course

Time Complexity

The time complexity of the above algorithm will be $O(N + M)$ where ‘N’ and ‘M’ are the number of characters in the input string and the pattern respectively.

Space Complexity

The space complexity of the algorithm is $O(M)$ since in the worst case, the whole pattern can have distinct characters which will go into the **HashMap**. In the worst case, we also need $O(N)$ space for the result list, this will happen when the pattern has only one character and the string contains only that character.

← Back
Problem Challenge 2

Next →
Problem Challenge 3

☑ Mark as Completed

5% completed, meet the criteria and claim your course certificate!

Claim Certificate

Report an Issue

Ask a Question (https://discuss.educative.io/tag/solution-review-problem-challenge-2__pattern-sliding-window__grokking-the-coding-interview-patterns-for-coding-questions)

Back To Course Home

Grokking the Coding Interview: Patterns for Coding Questions

5% completed

Search Course

(/collection/page/56686391014

● Fruits into Baskets (medium) (/collection/page/56686391014

● No-repeat Substring (hard) (/collection/page/56686391014

● Longest Substring with Same Letters after Replacement (hard) (/collection/page/56686391014

● Longest Subarray with Ones after Replacement (hard) (/collection/page/56686391014

○ Problem Challenge 1 (/collection/page/56686391014

○ Solution Review: Problem Challenge 1