

java anagrams hackerrank solution

Java anagrams Hackerrank solution is a popular problem that challenges developers to determine if two strings are anagrams of each other. An anagram is a word or phrase formed by rearranging the letters of a different word or phrase, using all the original letters exactly once. In this article, we will explore the concept of anagrams, break down the problem statement, and provide a comprehensive solution using Java. Additionally, we will discuss various approaches and optimizations that can be applied.

Understanding Anagrams

Anagrams are interesting puzzles in the world of linguistics and computer science. They showcase how letters can be rearranged to form new words. For example, the words "listen" and "silent" are anagrams because they contain the same letters in different orders.

To determine if two strings are anagrams, we need to ensure:

1. Both strings have the same length.
2. Both strings contain the same characters in the same frequency.

Problem Statement

The typical problem statement for the Java anagrams Hackerrank challenge is as follows:

Given two strings, determine if they are anagrams of each other. The strings can contain uppercase and lowercase letters, and the comparison should be case-insensitive.

Input

- Two strings (s1 and s2).

Output

- Return "Anagrams" if the strings are anagrams of each other; otherwise, return "Not Anagrams".

Approaches to Solve the Problem

There are several approaches to determine if two strings are anagrams. Here are a few common techniques:

1. Sorting: Sort both strings and compare them.
2. Counting Characters: Count the frequency of each character and compare the counts.
3. Using HashMaps: Utilize a HashMap to store character counts.

We will discuss each approach in detail.

Approach 1: Sorting

The simplest way to check if two strings are anagrams is to sort both strings and compare them. If they are identical after sorting, then they are anagrams.

Steps:

1. Convert both strings to lowercase (to ensure case insensitivity).
2. Remove any non-alphabetical characters if needed.
3. Sort both strings.
4. Compare the sorted strings.

Java Implementation:

```
```java
import java.util.Arrays;

public class AnagramChecker {
 public static String areAnagrams(String s1, String s2) {
 // Convert to lowercase
 s1 = s1.toLowerCase();
 s2 = s2.toLowerCase();

 // Sort both strings
 char[] charArray1 = s1.toCharArray();
 char[] charArray2 = s2.toCharArray();

 Arrays.sort(charArray1);
 Arrays.sort(charArray2);

 // Compare sorted strings
 if (Arrays.equals(charArray1, charArray2)) {
 return "Anagrams";
 } else {
```

```

return "Not Anagrams";
}
}

public static void main(String[] args) {
System.out.println(areAnagrams("Listen", "Silent")); // Anagrams
System.out.println(areAnagrams("Hello", "World")); // Not Anagrams
}
}
```

```

Approach 2: Counting Characters

Another efficient approach is to count the frequency of each character in both strings. If the counts match for every character, the strings are anagrams.

Steps:

1. Convert both strings to lowercase.
2. Create an array of size 26 (for each letter of the English alphabet).
3. Iterate through the first string and increment the corresponding index in the count array.
4. Iterate through the second string and decrement the corresponding index in the count array.
5. If all counts are zero, the strings are anagrams.

Java Implementation:

```

```java
public class AnagramChecker {
public static String areAnagrams(String s1, String s2) {
// Convert to lowercase
s1 = s1.toLowerCase();
s2 = s2.toLowerCase();

// Early exit if lengths are different
if (s1.length() != s2.length()) {
return "Not Anagrams";
}

int[] charCount = new int[26];

// Increment counts for s1 and decrement for s2
for (int i = 0; i < s1.length(); i++) {

```

```

charCount[s1.charAt(i) - 'a']++;
charCount[s2.charAt(i) - 'a']--;
}

// Check if all counts are zero
for (int count : charCount) {
 if (count != 0) {
 return "Not Anagrams";
 }
}
return "Anagrams";
}

public static void main(String[] args) {
 System.out.println(areAnagrams("Listen", "Silent")); // Anagrams
 System.out.println(areAnagrams("Hello", "World")); // Not Anagrams
}
}
'''

```

## Approach 3: Using HashMaps

Another method involves using HashMaps to store character frequencies. This is particularly useful if the strings can contain characters beyond the English alphabet.

Steps:

1. Convert both strings to lowercase.
2. Create a HashMap for each string to store character counts.
3. Populate the HashMaps with character frequencies.
4. Compare the two HashMaps.

Java Implementation:

```

'''java
import java.util.HashMap;

public class AnagramChecker {
 public static String areAnagrams(String s1, String s2) {
 // Convert to lowercase
 s1 = s1.toLowerCase();
 s2 = s2.toLowerCase();

```

```

// Early exit if lengths are different
if (s1.length() != s2.length()) {
 return "Not Anagrams";
}

HashMap charCountMap = new HashMap<>();

// Populate the map with counts from s1
for (char c : s1.toCharArray()) {
 charCountMap.put(c, charCountMap.getOrDefault(c, 0) + 1);
}

// Decrease the counts using s2
for (char c : s2.toCharArray()) {
 if (!charCountMap.containsKey(c)) {
 return "Not Anagrams";
 }
 charCountMap.put(c, charCountMap.get(c) - 1);
 if (charCountMap.get(c) < 0) {
 return "Not Anagrams";
 }
}

return "Anagrams";
}

public static void main(String[] args) {
 System.out.println(areAnagrams("Listen", "Silent")); // Anagrams
 System.out.println(areAnagrams("Hello", "World")); // Not Anagrams
}
}
...

```

## Conclusion

In summary, the Java anagrams Hackerrank solution can be approached from various angles, each with its advantages and trade-offs. The sorting method is the simplest but can be less efficient for large strings due to its  $O(n \log n)$  complexity. The character counting method provides a linear  $O(n)$  solution, making it more efficient for larger datasets. Using HashMaps offers flexibility for strings with a wider range of characters.

Understanding these methods not only helps in solving the anagram problem but also enhances your problem-solving skills in competitive programming. The key takeaway is to choose an approach based on the problem constraints to optimize performance effectively.

## Frequently Asked Questions

### What is the basic approach to solving the Java Anagrams problem on HackerRank?

The basic approach involves checking if two strings are anagrams by sorting both strings and comparing them. If the sorted strings are equal, they are anagrams.

### What data structures can be used to efficiently check for anagrams in Java?

You can use a frequency array or a HashMap to count the occurrences of each character in the strings. If the counts match for both strings, they are anagrams.

### How can I optimize the anagram check for large input strings in Java?

To optimize for large strings, you can use a fixed-size array of 26 integers (for lowercase letters) to count character frequencies, which reduces the time complexity compared to sorting.

### What are common pitfalls to avoid when implementing an anagram solution in Java?

Common pitfalls include not handling case sensitivity, ignoring non-alphabetic characters, and not accounting for different string lengths before checking for anagrams.

### Can you provide a sample Java code snippet for checking anagrams on HackerRank?

Sure! Here's a simple example:

```
```java
public static boolean areAnagrams(String str1, String str2) {
    if (str1.length() != str2.length()) return false;
    int[] charCount = new int[26];
    for (char c : str1.toCharArray()) charCount[c - 'a']++;
    for (char c : str2.toCharArray()) charCount[c - 'a']--;
}
```

```
for (int count : charCount) if (count != 0) return false;
return true;
}
'''
```

[Java Anagrams Hackerrank Solution](#)

Java Anagrams Hackerrank Solution

Related Articles

- [jew in the modern world](#)
- [joey pigza loses control report](#)
- [john deere 300b backhoe manual](#)

[Back to Home](#)