**Problem Summary**

Given strings s1 and s2, return true if s2 contains a permutation of s1.  
That is, if any substring of s2 is an anagram of s1.

**Input Example**

s1 = "ab"

s2 = "eidbaooo"

Expected Output: true  
→ Because "ba" is a substring of s2 and is a permutation of "ab".

**Key Concepts**

* We use two frequency arrays:
  + s1Freq: frequency of characters in s1
  + windowFreq: frequency of a current-length window (same length as s1) in s2
* We slide this window over s2 and check at each position if windowFreq == s1Freq.

**Dry Run Table**

We’ll simulate the loop for:

s1 = "ab" → len1 = 2

s2 = "eidbaooo" → len2 = 8

**Step 1: Initial Frequencies**

* s1Freq:  
  a=1, b=1, others = 0
* windowFreq (first 2 chars "ei"):  
  e=1, i=1, others = 0

→ matches(s1Freq, windowFreq) = false

**Step 2: Slide Window**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i (start idx) | Window in s2 | Action | windowFreq update | Match? |
| 0 | "ei" | Initial | e=1, i=1 | ❌ |
| 1 | "id" | Remove 'e', add 'd' | i=1, d=1 | ❌ |
| 2 | "db" | Remove 'i', add 'b' | d=1, b=1 | ❌ |
| 3 | "ba" | Remove 'd', add 'a' | b=1, a=1 | ✅ |
|  |  | windowFreq == s1Freq |  | ✅ return true |

✅ Found valid permutation at index 3 ("ba")

**Final Output**

true

**Helper Method: matches(a, b)**

Simple comparison of both arrays:

for (int i = 0; i < 26; i++)

if (a[i] != b[i]) return false;

→ O(26) = O(1)

**Time and Space Complexity**

|  |  |
| --- | --- |
| Metric | Value |
| Time | O(n \* 26) → O(n) where n = s2.length() |
| Space | O(1) — two arrays of size 26 |

# Solution

public class Solution {

    public boolean checkInclusion(String s1, String s2) {

        int len1 = s1.length(), len2 = s2.length();

        if (len1 > len2) return false;

        int[] s1Freq = new int[26];

        int[] windowFreq = new int[26];

        // Build frequency of s1

        for (char c : s1.toCharArray()) {

            s1Freq[c - 'a']++;

        }

        // Initialize window

        for (int i = 0; i < len1; i++) {

            windowFreq[s2.charAt(i) - 'a']++;

        }

        // Slide window

        for (int i = 0; i <= len2 - len1; i++) {

            if (i > 0) {

                // Slide: remove left, add right

                windowFreq[s2.charAt(i - 1) - 'a']--;

                windowFreq[s2.charAt(i + len1 - 1) - 'a']++;

            }

            if (matches(s1Freq, windowFreq)) {

                return true;

            }

        }

        return false;

    }

    // Helper to compare two frequency arrays

    private boolean matches(int[] a, int[] b) {

        for (int i = 0; i < 26; i++) {

            if (a[i] != b[i]) return false;

        }

        return true;

    }

}