Beispiel: 1D Array

Die Integer Arrays A, B:

int[] A = new int[n];
int[] B = new int[m];

Seien A, B zwei Integer Arrays. Wir wollen herausfinden, wie oft der Subarray B im Array A vorkommt. Vorkommen bedeutet dass ein Index i wobei $0 \leq i < n$ existiert, so dass $A_i = B_0, A_{i+1} = B_1, ..., A_{i+m-1} = B_{m-1}$. Das bedeutet also dass B in A vorkommt, wenn alle Elemente von B aufeinander folgend in A vorkommen.

Es gilt: $i < j$

Es ist zu beachten, dass wenn zwei Index i, j existieren, welche die obige Bedingung erfüllen und $j \leq i + m - 1$ gilt, dann wird der Subarray B nur einmal gezählt. Einfach ausgedrückt: Es darf keine Überlappungen geben.

Beispiel

int[] A = new int[] {1, 1, 1, 2, 1, 1, 2, 1};
int[] B = new int[] {1, 1, 2};
B kommt in A zweimal vor

int[] A = new int[] {1, 1, 1};
int[] B = new int[] {1, 1};
B kommt in A einmal vor

Code (Lösung)

```
public static boolean arrayStartsWith(int[] A, int[] B, int offset) {
    if (A.length < B.length || offset > A.length - B.length) {
        return false;
    }
    for (int j = 0; j < B.length; j++) {
        if (A[offset+j] != B[j]) {
            return false;
        }
    }
    return true;
}

public static int countSubArray(int[] A, int[] B) {
    int occurrences = 0;
    if (B.length > A.length) {
        return occurrences;
    }

    int i = 0;
    while (i <= A.length - B.length) {
        if (arrayStartsWith(A, B, i)) {
            occurrences++;
            i += B.length;
        } else {
            i++;
        }
    }

    return occurrences;
}
```

Überprüfe die Bounds

Wende den beschriebenen Algorithmus an

Überprüfe die Bounds

$i + m - 1 < i + m$

Schaue bei jedem Element, ob B ein Subarray ist. Falls B ein Subarray ist, überspringe den Subarray B im Array A