

1. Patterns

Problem: *Diamond Pattern (Star Pattern)*

Write a program that prints a diamond pattern using stars (*). The diamond should have n rows in the upper half and $n-1$ rows in the lower half. The width of the diamond should be $2n - 1$ stars in the center.

Test Cases:

Input: $n = 3$

Output:

```
*  
***  
*****  
***  
*
```

Input: $n = 4$

Output:

```
*  
***  
*****  
*****  
***  
*
```

Input: `n = 5`

Output:

```
*  
***  
*****  
*****  
*****  
****  
***  
*
```

Input: `n = 2`

Output:

```
*  
***  
*
```

2 *Find the Majority Element*

Given an array of size `n`, find the majority element. The majority element is the element that appears more than $n/2$ times. You need to do this in $O(n)$ time and $O(1)$ space.

Test Cases:

1. **Input:** `[3, 3, 4, 2, 4, 4, 2, 4, 4]`

Output: `4`

2. **Input:** `[1, 1, 1, 1, 2, 2, 2]`

Output: `1`

3. **Input:** [1, 2, 3, 4, 5]
Output: -1 (No majority element)

4. **Input:** [2, 2, 2, 3, 3, 2, 2]
Output: 2

3 Longest Subarray with Sum Equal to K

Given an array of integers and a target sum k , find the length of the longest subarray with a sum equal to k .

Test Cases:

1. **Input:** arr = [1, -1, 5, -2, 3], k = 3

Output: 4

Explanation: The subarray [1, -1, 5, -2] has sum 3 and length 4.

2. **Input:** arr = [1, 1, 1, 1, 1], k = 3

Output: 3

3. **Input:** arr = [1, 2, 3, 4, 5], k = 9

Output: 2

4. **Input:** arr = [-1, 2, 3, 1, -2, 1], k = 3

Output: 3

4 Longest Substring Without Repeating Characters

Given a string, find the length of the longest substring without repeating characters.

Test Cases:

1. Input: "abcabcbb"

Output: 3

Explanation: The longest substring without repeating characters is "abc".

2. Input: "bbbbbb"

Output: 1

Explanation: The longest substring without repeating characters is "b".

3. Input: "pwwkew"

Output: 3

Explanation: The longest substring without repeating characters is "wke".

4. Input: "abcde"

Output: 5

Explanation: The longest substring without repeating characters is "abcde".

5 Anagram Substring Search

Given a string `s` and a string `p`, return the start indices of all the substrings of `s` that are anagrams of `p`.

Test Cases:

1. **Input:** `s = "cbaebabacd"`, `p = "abc"`

Output: `[0, 6]`

Explanation: The substrings "cba" and "bac" are anagrams of "abc".

2. **Input:** `s = "abab"`, `p = "ab"`

Output: `[0, 1, 2]`

Explanation: The substrings "ab", "ba", and "ab" are anagrams of "ab".

3. **Input:** `s = "aaaaaaaaaa"`, `p = "aa"`

Output: `[0, 1, 2, 3, 4, 5, 6, 7, 8]`

Explanation: All substrings of length 2 are anagrams of "aa".

4. **Input:** `s = "abcd"`, `p = "xyz"`

Output: `[]`

Explanation: No substring of "abcd" is an anagram of "xyz".