SLIDING WINDOW

Leetcode 121. Best Time to Buy and Sell Stock

<https://leetcode.com/problems/best-time-to-buy-and-sell-stock/description/>

Algorithm: 2 pointers, DS: Array

* Initialize 3 variables, max, left and right. Left starts at 0, right starts at 1.
* Calculate the current profit (arr[left] - arr[right]). Compare with max.
* If arr[left] is greater than arr[right], increment right by one, set left = right.
* Return max.

3. Longest Substring Without Repeating Characters

<https://leetcode.com/problems/longest-substring-without-repeating-characters/description/>

DS: Hashset

* Initialize a set of chars, initialize a left var.
* Iterate through the string, if we encounter a char that’s already in the set, remove the elements before it and the element too. Then add the element
* Get the max of elements in the set via: i-left +1.

4. Longest Repeating Character Replacement

https://leetcode.com/problems/longest-repeating-character-replacement/

DS: Dictionary

* Initialize a dictionary of chars, left, max frequency, and result variables
* Everytime we insert or update a value in the dictionary, use the maxf to keep track of the max value in dict.Values
* Check if the num of chars to replace is greater than k. If it is, remove the element at s[left] and then left++;
* Compare max and the window size. Return the max at the end

5. Permutation in String

<https://leetcode.com/problems/permutation-in-string/description/>

DS: Array

* Check the edge case: s1 length > s2 length?
* Initialize two arrays, then populate the arrays with the content of the first window in s1 and s2
* Get the count of equal elements in both arrays
* Increase and decrease the window size: Get the left and right chars
* If the count is 26, return true
* Else, increase the right and compare the count of equal chars in arr2 and arr1
* Decrease the left and compare the count of equal chars in arr1 and arr2.
* Return true if count ==26;

76. Minimum Window Substring

<https://leetcode.com/problems/minimum-window-substring/description/>

DS = Dictionary

* Edge case
* Initialize 2 dictionaries for S and T. Populate the dictT.
* Initialize variables for have, need, left, result, and arr.
* Iterate through S

\* At each insertion in dictS, if dictT contains char and dict[char] == dictS[char], have++;

\* Compare have and need, while they are equal:

compare the current window size and res, update res accordingly

Update arr

\* Decrement S[left] in dict S

\*Compare s[left] in dictT and dictS, update have accordingly.

\* Increment left.

Return res

BASIC IDEA: Keep adding to the dictionary till we find a complete match between have and need. Once we have a complete match, decrement the left most chars of S in dictS until have and need are no longer equal. Then start adding to dictS again.

* Keep track of the window size
* Keep track of the have and need values

239. Sliding Window Maximum

<https://leetcode.com/problems/sliding-window-maximum/description/>

Data Structure: LinkedList

Basic Idea:

\*Iterate through the array

While there are elements in the queue and the first value at the index in the queue isn't part of the current window, remove it.

While there are elements in the queue and the current value is greater than the last value in the queue, remove the last value to maintain decreasing order

Add the index of the current value to the end of the queue

If the window size is equal to k, it means we have enough values, so add the max item in the queue (which is the first value) to the result.

Window Size = i-k+1;

if the index is less than the window size (k-1), then the number isn't part of the queue

//we're adding the indexes rather than the numbers so that we can use the index to determine if the number at that index is part of the window