**Longest Substring Without Repeating Characters**

주어진 문자열에서 반복되지 않는 제일 긴 substring을 찾아 그 문자의 길이를 리턴하는 문제입니다.

1. 각 캐릭터를 돌면서 hash map 에 해당 문자가 존재 하는지 확인합니다
2. 존재하면 그 문자를 hash map에서 삭제 합니다
   1. 삭제와 동시에 max counter를 –1합니다
3. 존재하지 않으면 그 문자를 hash map에 삽입합니다
   1. 삽입과 동시에 max counter를 ++1 합니다
4. 마지막 문자에서 max counter 를 리턴합니다

Graphical user interface, text

Description automatically generated with medium confidence

**Container with most water**

주어진 배열의 각 요소는 컨테이너의 높이를 의미합니다. 컨테이너에서 제일 큰 부피를 찾는 문제입니다.

A picture containing chart

Description automatically generated

* Key takeaways
  + We should have two pointers (left, right).
    - Left points to the first element of the array
    - Right points to the last element of the array
    - Doing so, we maximize our width
  + Then, compare heights of the left and right pointers
    - Move the pointer which is smaller to the direction
    - If same, move either one

Graphical user interface, text, application

Description automatically generated

**Three sum**

주어진 배열에서 합이 0이 되는 세 개의 element를 찾는 문제입니다. 단, 중복 되는 triplet 가질수 없습니다.

[-3, 3, 3, 4, -3, 1, 2, 2, -2, -1, 1, 0]

- 정렬을 해야 하는 이유

- 같은 자리에 같은 값이 들어 올수 있기때문에 정렬을 하면 duplicate triplet을 avoid할 수 있다.

- temp > 0, right—하는 이유

- 배열이 정렬되어 있기 때문에 temp가 작아 지기 위해서는 right pointer descrease

- temp === 0, while loop이유

- to avoid duplicate triplet, we don’t need to compute same element that was already computed

Text, timeline

Description automatically generated

Remove Nth Node from End of List

Given the head of a linked list, remove the nth node from the end of the list and return its head.

Input: 1-> 2 -> 3 -> 4 -> 5 , n = 2

Output: 1 -> 2 -> 3 -> 5

Explanation

1 -> 2 -> 3 -> 4 -> 5

L,R

We gonna loop n time to shift right pointer

1 -> 2 -> 3 -> 4 -> 5

L R

Then, we gonna shift both pointer to next while right pointer hits the end

1 -> 2 -> 3 -> 4 -> 5

L R

1 -> 2 -> 3 -> 4 -> 5

L R

1 -> 2 -> 3 -> 4 -> 5 -> null

L R

The left pointer exactly points to the 2nd node from end of the list. Now, we just need to get its previous element and link it to nth node’s next node. However, we face the issue that there is no way to get the previous node. In order to solve this, we should introduce a dummy node and set left pointer to point the dummy node. But we maintain right pointer to point the head. Then, we shift right pointer to nth time.

Dummy -> 1 -> 2 -> 3 -> 4 -> 5

L R

Dummy -> 1 -> 2 -> 3 -> 4 -> 5

L R

Dummy -> 1 -> 2 -> 3 -> 4 -> 5

L R

Dummy -> 1 -> 2 -> 3 -> 4 -> 5 -> null

L R

Now, we link 3 to 5.

Graphical user interface, text, application

Description automatically generated

**Valid Parentheses**

* If stack is empty, it is valid input because stack always have corresponding open bracket

Graphical user interface, text, application

Description automatically generated

Merge K Sorted Lists

Input: [[1 -> 4 -> 5], [1 -> 3 -> 4], [2 -> 6]]

Output: [1 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6]

Time complexity: O(nlogk) because we are taking a list and divide them by 2(logk) for n times. That’s nlogkGraphical user interface, application

Description automatically generated with medium confidenceTimeline

Description automatically generated with low confidence

Search in Rotated Sorted Array

* Here the number of rotation on the array doesn’t matter. That is just trick to make you confused.
* No matter how many times we rotate the array, there always have at least one uniformly increasing subarray.
* Our purpose it not to find the pivot element but only to find the target exists in the array or not.
* Why we look for uniformly increasing sorted array?
  + Because binary search comes with a precondition that array must be sorted
* Otherwise, discard the sorted half and keep examining the unsorted half.
* Since we are portioning the array in half at each step, this gives us O(log n) runtime complexity.

Group Anagram

* Two strings are anagrams if and only if their sorted strings are equal.

Graphical user interface, text, application

Description automatically generated

Maximum Subarray

* If sum of current subarray sum is less than current element, reset current subarray sum to the current element because obviously current element is greater than previous subarray. Why would we keep tracking the previous sub?
  + However, we should be very careful that we should also check current subarray sub is less than 0. If we don’t check this will fail.
  + [1,2]
    - Should be 3 but without checking less than 0 will give us 2.

Graphical user interface, text, application, email

Description automatically generated

Jump Game

Graphical user interface, text, application

Description automatically generated

Greedy solution

* For loop
  + Once you reached to the last index, you update that the goal to current index because current index proves that can reached to the last index (so now we treat current index as last index)
* lastPos
  + lastPos === 0 means starting index of 0 can reach to the last index
* if
  + we wanna know if we are at the position i + nums[i] which is the jump of the length. We want to know can this jump starting at position i taking its maximum jump is greater than or equal to the lastPos(goal). Because if it is, we can reach the goal

Graphical user interface, text, application

Description automatically generated

Merge Interval

* We need to first sort them based on start value. We don’t care about the end value because we want to go through our intervals starting at the start value and going to the next start value. This allows us to detect if two intervals are overlapped.
* If we don’t sort them, we should iterate our output every time we check the current intervals.
* [[1, 3], [8, 10], [11, 12], [2, 6]]
  + If not sorted, we iterate back to [1,3] at the interval [2, 6]

Text

Description automatically generated

Insert intervals

Graphical user interface, text, application, email

Description automatically generated

Unique Paths

* Detail in the note

Graphical user interface, text, application

Description automatically generated

Climbing stairs

* On the other mac