Longest Substring Without Repeating Characters

Problem

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

example 1:

Input: “abcabcbb”
Output: 3
Explanation: The answer is “abc”, with the length of 3.

example 2:

Input: “bbbbb”
Output: 1
Explanation: The answer is “b”, with the length of 1.

example 3:

Input: “pwwkew”
Output: 3
Explanation: The answer is “wke”, with the length of 3.
Note that the answer must be a substring, “pwke” is a subsequence and not a substring.

Solution

Approach 1: Brute Force

Intuition

Check all the substring one by one to see if it has no duplicate character.

Algorithm

Suppose we have a function boolean allUnique(String substring) which will return true if the characters in the substring are all unique, otherwise false.
Now fill the missing parts:

- To enumerate all substrings of a given string, we enumerate the start and end indices of them. Suppose the start and end indices are i and j, respectively. Then we have $0 \leq i \leq j \leq n$, thus using two nested loops with i from 0 to $n-1$ and j from $i+1$ to n, we can enumerate all th esubstrings of s.
- To check if one string has duplicate characters, we can use a set. We iterate through all the characters in the string and put them into the set one by one. Before putting one character, we check if the set already contains it. If so, we return false. After the loop, we return true.

Java

```
public class Solution {
```

```
    public int lengthOfLongestSubstring(String s) {
        int n = s.length();
        int ans = 0;
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j <= n; j++)
                if (allUnique(s, i, j)) ans = Math.max(ans, j - i);
        return ans;
    }

    public boolean allUnique(String s, int start, int end) {
        Set<Character> set = new HashSet<>();
        for (int i = start; i < end; i++) {
            Character ch = s.charAt(i);
            if (set.contains(ch)) return false;
            set.add(ch);
        }
        return true;
    }
}
```

Approach 2: Sliding Window

Algorithm

By using HashSet as a sliding window, checking if a character in the current can be done in $O(1)$.
We use HashSet to store the characters in current window $[i, j)$. Then we slide the index j to the right. If it is not in the HashSet, we slide j further. Doing so until $s[j]$ is already in the HashSet and found the maximum size of such substring. Do this for all i, we got the answer.

Java

```
public class Solution {
    public int lengthOfLongestSubstring(String s) {
        int n = s.length();
        Set<Character> set = new HashSet<>();
        int ans = 0, i = 0, j = 0;
        while (i < n && j < n) {
            // try to extend the range [i, j]
            if (!set.contains(s.charAt(j))){
                set.add(s.charAt(j++));
                ans = Math.max(ans, j - i);
            }
            else {
                set.remove(s.charAt(i++));
            }
        }
        return ans;
    }
}
```

Python

```
class Solution:
    def lengthOfLongestSubstring(self, s):
        """
        :type s: str
        :rtype: int
        """
        if s == "":
            return 0
        List = []
```

```
        Length = 0
        Max = 0
        for ch in s:
            if ch not in List:
                List.append(ch)
                Length += 1
            else:
                while ch in List:
                    del List[0]
                List.append(ch)
                Length = len(List)
            if Length > Max:
                Max = Length

        return Max
```

Approach 3: Sliding Window Optimized

Algorithm

The above solution requires at most 2n steps. In fact, it could be optimized to require only n steps. Instead of using a set to tell if a character exists or not, we could define a mapping of the characters to its index. Then we can skip the characters immediately when we found a repeated character.
The reason is that if $s[j]$ have a duplicate in the range $[i, j)$ with index j', we don't need to increase ii little by little. We can skip all the elements in the range $[i, j']$ and let i to be $[j' + 1]$ directly.
Java

```
public class Solution {
    public int lengthOfLongestSubstring(String s) {
        int n = s.length(), ans = 0;
        Map<Character, Integer> map = new HashMap<>(); // current index of character
        // try to extend the range [i, j]
        for (int j = 0, i = 0; j < n; j++) {
            if (map.containsKey(s.charAt(j))) {
                i = Math.max(map.get(s.charAt(j)), i);
            }
            ans = Math.max(ans, j - i + 1);
            map.put(s.charAt(j), j + 1);
        }
        return ans;
    }
}
```