Longest Substring Without Repeating Characters

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 1 in C++

```
class Solution {
public:
    int countuniquechar(const char *s, int len)
    {
        int i, cnt;
        std::map<char,int> chrMap;
        for( i=cnt=0; i<len; i++)
        {
            if (chrMap[s[i]]) return i;
            chrMap[s[i]]=i+1;
        }
        return i;
    }
    int lengthOfLongestSubstring(string s) {
        const char *buff=s.c_str();
        int str_len=s.length();
        int substr_len=str_len;
        int cur_len;
        int max_len=0;
        for( int i=0; i<str_len; i++)
        {
            cur_len=countuniquechar(buff+i,substr_len--);
            if (cur_len>max_len) max_len=cur_len;
            if (max_len>str_len) break;
        }
        return max_len;
    }
};
```

Solution 2 in Java : Brute Force - O(n^3)

```
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;
    }
}
```

Solution 2 in Java - Sliding Window - O(2n)

```
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;
    }
}
```

Solution 3 in Java - Hashmap - O(2n)

```
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;
    }
}
```