3. Longest Substring Without Repeating Characters

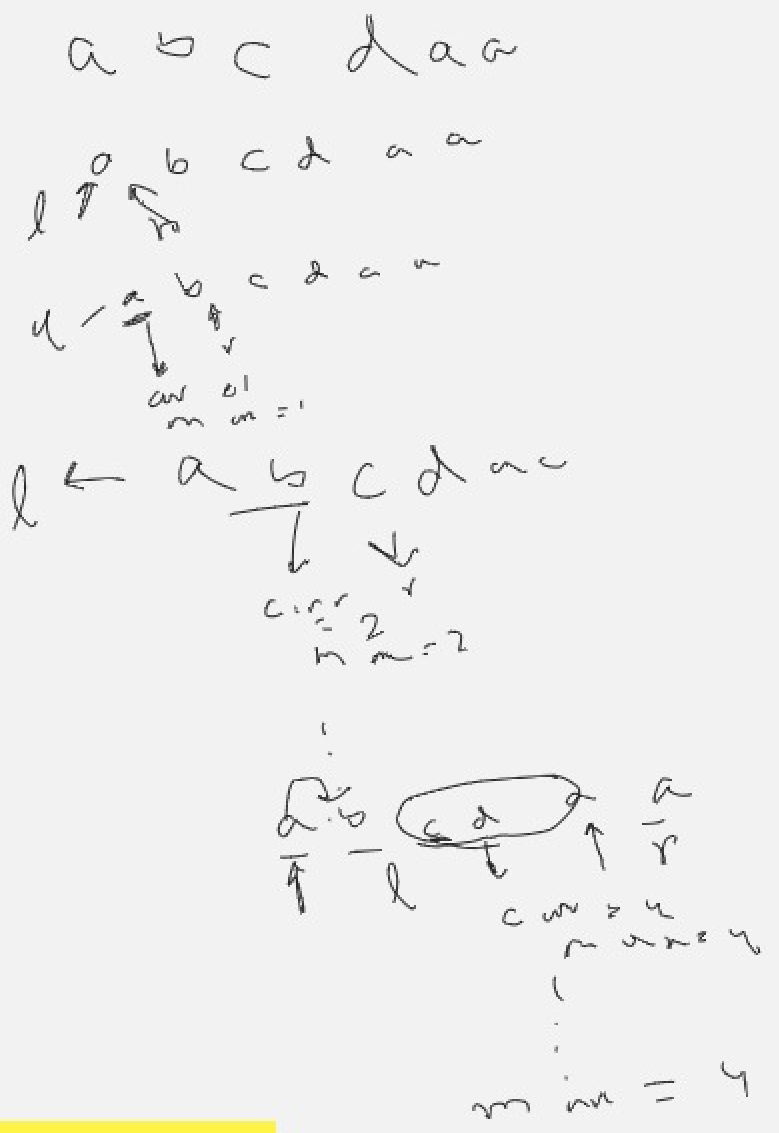
This is a sliding window technique problem. The technique involves maintaining an optimization in a fixed or variable size window which is continuous in nature. (Subarray or substring).

In the problem, given a string we need to find the largest substring of unique character. So for example: In abcdaa, abcd is the largest substring of unique characters whose length is 4.

To solve this problem, first let us find the brute force way to solve it, we run a nested loop inside another loop, and form substring length until we get a character equal to the outer loop character. this is a O(n^2) solution.

But using sliding window we eliminate the need for two loops. This can be solved in n time complexity. The solution is as follows:

1. Maintain left and right pointer.
2. Keep the last occurred index of a character in an unordered map. (Hashmap key value pair).
3. We run a for loop on the string.
4. Inside we check if we dont find the key in the hashmap already, then we increase the current_length by 1 and put the key into the hashmap with value as the index.
5. Else if we find the key in the hashmap, if the key is already present with a range of l, we do update l to 1 plus the last occurance of the current key and set value for current key in the key value pair.
6. Keep a max and current length count and return the max_len after the end of the function.



The code:

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        int l = 0;
        int r = 0;
        int max_len = 0;
        int curr_len = 0;
        unordered_map<char, int> pos_map;
        for(int i=0;i<s.size();i++){
            if(pos_map.find(s[i]) != pos_map.end()&&pos_map[s[i]]>=l){

                l = pos_map[s[i]]  + 1;
                pos_map[s[i]] = i;
                r++;
                curr_len = r - l;
                max_len = max(max_len,curr_len);
            }
                else{
                pos_map[s[i]] = i;
                r++;
                curr_len = r-l;
                max_len = max(max_len,curr_len);
                }

            }
             return max_len;
        }
};
```