Longest substring without repeating characters

BRUTE FORCE :

Generating all substring and using set to find whether they contain duplicates or not.

```
#include <bits/stdc++.h>
int uniqueSubstrings(string input)
{
    //Write your code here
    int n=input.size();
    string str;
    int maxlen=0;
    for(int i=0;i<n;i++)
    {
        str="";
        unordered_set<char> s;
        for(int j=i;j<n;j++)
        {
            str+=input[j];
            s.insert(input[j]);
            if(str.size()==s.size())
                maxlen=max(maxlen,(int)str.size());
        }
    }
    return maxlen;
}
```

- Time Complexity : O(N^2)
- Space Complexity : O(N)

Better Approach : two pointer and sliding window

We initially check whether the element at right pointer is present in the set or not if it is already present that means it is a duplicate element and we keep incrementing left and erasing from the set till we get the duplicate element erased. Once it is erased we compare the current length with maxlen and store it in maxlen using right-left+1. It there are no duplicates then we simply keep moving right and storing maxlen.

```
#include <bits/stdc++.h>
int uniqueSubstrings(string input)
{
    //Write your code here
    unordered_set<char> s;
    int n=input.size(),left=0,right=0,maxlen=0;
    while(right<n)
    {
        while(s.find(input[right])!=s.end() && left<=right)
        {
            s.erase(input[left]);
            left++;
        }
        maxlen=max(maxlen,right-left+1);
        s.insert(input[right]);
        right++;
    }
    return maxlen;
}
```

- Time Complexity : O(2N) left traverse N times and right traverses N times.
- Space Complexity : O(N)

Optimal Approach :

Instead of storing unique in set we store the char along with index in map as soon as we encounter a repeating elem we directly set left to that repeating elem index +1 but we are not deleting the elements we are only updating indexing what if we have already updated out left and the repeating char is no more in our window so to tackle this we simply use max of left and index+1 to store the starting of window which will give us the correct position of left

Ex: we already have b at index 2 but our current left is pointing to 4 so the value in left would be set to left=max(2,4)=4 and in this way we avoid using duplicates in the substring.

```
#include <bits/stdc++.h>
int uniqueSubstrings(string input)
{
    //Write your code here
    unordered_map<char,int> m;
    int n=input.size(),left=0,right=0,maxlen=0;
    while(right<n)
    {
        if(m.find(input[right])!=m.end())
        {
            left=max(m[input[right]]+1,left);
        }
        maxlen=max(maxlen,right-left+1);
        m[input[right]]=right;
        right++;
    }
    return maxlen;
}
```

- Time Complexity : reduced from O(2N) to O(N)
- Space Complexity : O(N)