Minimum window substring

30 April 2022 17:23

https://leetcode.com/problems/minimum-window-substring/

Question is - We have two string - S1 --> long string
S2 --> target string

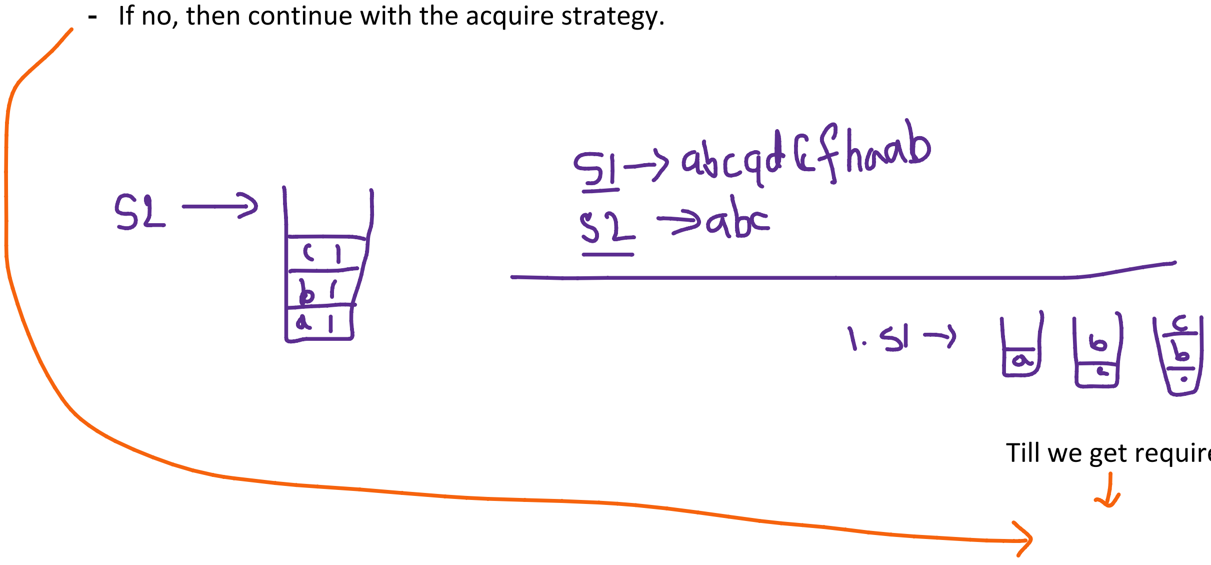
Select the min. Substring (which contain other unwanted elements), which meetup our target string.

Input: s = "ADOBECODEBANC", t = "ABC"
Output: "BANC"
Explanation: The minimum window substring "BANC" includes 'A', 'B', and 'C' from string t.

Approach - 1. create hashmap for target string i.e map2.

2. travel the whole long string s1...
3. add the new elements to the long string hashmap i.e map1 (Acquire strategy).
4. check whether map1== map2

- If yes, then count the freq. Of map1. If it is greater then previous one, then placed it as curr_min.
  - start releasing the elements from the starting of the current substring, till the substring become in valid(map1!=map2).
  And check while relasing the elements, if its size is lesser than update it.
- If no, then continue with the acquire strategy.



Till we get required target in it.

```
#include<bits/stdc++.h>
using namespace std;

// check the map1 and map2 is valid or not??????????
```

```
bool isvalid(unordered_map<char,int>map1, unordered_map<char,int>map2)
{
   for(auto i : map2)   // for all elements of map2
   {

       if(map1[i.first]<i.second)  // check the frequency of present element of map2 is smaller or not. If
greater then false.
       {

       return false;
       }
   }
   return true;  // otherwise it is true.
}


int main()
{
   string s1,s2;

   cin>>s1;
   cin>>s2;


   unordered_map<char,int> map1; // map1 for s1- long string
   unordered_map<char,int> map2;  // map2 for s2- target string

   //create hashmap for target string
   for(int i=0;i<s2.size();i++)
   {
      if(map2.count(s2[i])>0)
      {
         map2[s2[i]]++;
      }
      else
      {
         pair<char,int>p(s2[i],1);
         map2.insert(p);
      }
   }


   int count=INT_MAX;

   //travel the whole long string s1...
   for(int i=0;i<s1.size();i++)
   {
      if(map1.count(s1[i])>0)
            {
               map1[s1[i]]++;
            }
         else                                  //add the new elements to the longstring hashmap
            {
                  pair<char,int>p(s1[i],1);
                  map1.insert(p);
                }

        //then check that is the current hashmap1 is valid or not
        if(isvalid(map1,map2))
        {
            int sum=0;
            for(auto i:map1)              // count the total frequency of the map1.
              {
                 sum=sum+i.second;
              }
          if(count>sum)                 // if previous count is max, then sum -> update it, because we want min.
string
          {
             count=sum;

          }

          //start release the starting part of the current substring, till that current substring becomes
invalid.
          int j=i-sum+1;

          while(isvalid(map1,map2))
          {
             int sum=0;
            for(auto i:map1)
              {
                 sum=sum+i.second;            // check after releasing element, whether the current substring is
less size then previous one or not.
              }
          if(count>sum)
          {
             count=sum;


          }
             map1[s1[j]]--;    // release the starting element of current substring

             j++;   // then increase the index
          }

        }


    }
    cout<<count<<endl;    // print min. size of the required string.

}
```