**Among all leetcode questions, I find that there are at least 5 substring search problem which could be solved by the sliding window algorithm.**  
so I sum up the algorithm template here. wish it will help you!

1. ***the template***:

public class Solution {

public List<Integer> slidingWindowTemplateByHarryChaoyangHe(String s, String t) {

//init a collection or int value to save the result according the question.

List<Integer> result = new LinkedList<>();

if(t.length()> s.length()) return result;

//create a hashmap to save the Characters of the target substring.

//(K, V) = (Character, Frequence of the Characters)

Map<Character, Integer> map = new HashMap<>();

for(char c : t.toCharArray()){

map.put(c, map.getOrDefault(c, 0) + 1);

}

//maintain a counter to check whether match the target string.

int counter = map.size();//must be the map size, NOT the string size because the char may be duplicate.

//Two Pointers: begin - left pointer of the window; end - right pointer of the window

int begin = 0, end = 0;

//the length of the substring which match the target string.

int len = Integer.MAX\_VALUE;

//loop at the begining of the source string

while(end < s.length()){

char c = s.charAt(end);//get a character

if( map.containsKey(c) ){

map.put(c, map.get(c)-1);// plus or minus one

if(map.get(c) == 0) counter--;//modify the counter according the requirement(different condition).

}

end++;

//increase begin pointer to make it invalid/valid again

while(counter == 0 /\* counter condition. different question may have different condition \*/){

char tempc = s.charAt(begin);//\*\*\*be careful here: choose the char at begin pointer, NOT the end pointer

if(map.containsKey(tempc)){

map.put(tempc, map.get(tempc) + 1);//plus or minus one

if(map.get(tempc) > 0) counter++;//modify the counter according the requirement(different condition).

}

/\* save / update(min/max) the result if find a target\*/

// result collections or result int value

begin++;

}

}

return result;

}

}

1. Firstly, here is my sliding solution this question. I will sum up the template below this code.

**2) the similar questions are:**

<https://leetcode.com/problems/minimum-window-substring/>  
<https://leetcode.com/problems/longest-substring-without-repeating-characters/>  
<https://leetcode.com/problems/substring-with-concatenation-of-all-words/>  
<https://leetcode.com/problems/longest-substring-with-at-most-two-distinct-characters/>  
<https://leetcode.com/problems/find-all-anagrams-in-a-string/>

**3) I will give my solution for these questions use the above template one by one**

**Minimum-window-substring**  
<https://leetcode.com/problems/minimum-window-substring/>

public class Solution {

public String minWindow(String s, String t) {

if(t.length()> s.length()) return "";

Map<Character, Integer> map = new HashMap<>();

for(char c : t.toCharArray()){

map.put(c, map.getOrDefault(c,0) + 1);

}

int counter = map.size();

int begin = 0, end = 0;

int head = 0;

int len = Integer.MAX\_VALUE;

while(end < s.length()){

char c = s.charAt(end);

if( map.containsKey(c) ){

map.put(c, map.get(c)-1);

if(map.get(c) == 0) counter--;

}

end++;

while(counter == 0){

char tempc = s.charAt(begin);

if(map.containsKey(tempc)){

map.put(tempc, map.get(tempc) + 1);

if(map.get(tempc) > 0){

counter++;

}

}

if(end-begin < len){

len = end - begin;

head = begin;

}

begin++;

}

}

if(len == Integer.MAX\_VALUE) return "";

return s.substring(head, head+len);

}

}

you may find that I only change a little code above to solve the question "Find All Anagrams in a String":  
change

if(end-begin < len){

len = end - begin;

head = begin;

}

to

if(end-begin == t.length()){

result.add(begin);

}

**longest substring without repeating characters**  
<https://leetcode.com/problems/longest-substring-without-repeating-characters/>

public class Solution {

public int lengthOfLongestSubstring(String s) {

Map<Character, Integer> map = new HashMap<>();

int begin = 0, end = 0, counter = 0, d = 0;

while (end < s.length()) {

// > 0 means repeating character

//if(map[s.charAt(end++)]-- > 0) counter++;

char c = s.charAt(end);

map.put(c, map.getOrDefault(c, 0) + 1);

if(map.get(c) > 1) counter++;

end++;

while (counter > 0) {

//if (map[s.charAt(begin++)]-- > 1) counter--;

char charTemp = s.charAt(begin);

if (map.get(charTemp) > 1) counter--;

map.put(charTemp, map.get(charTemp)-1);

begin++;

}

d = Math.max(d, end - begin);

}

return d;

}

}

**Longest Substring with At Most Two Distinct Characters**  
<https://leetcode.com/problems/longest-substring-with-at-most-two-distinct-characters/>

public class Solution {

public int lengthOfLongestSubstringTwoDistinct(String s) {

Map<Character,Integer> map = new HashMap<>();

int start = 0, end = 0, counter = 0, len = 0;

while(end < s.length()){

char c = s.charAt(end);

map.put(c, map.getOrDefault(c, 0) + 1);

if(map.get(c) == 1) counter++;//new char

end++;

while(counter > 2){

char cTemp = s.charAt(start);

map.put(cTemp, map.get(cTemp) - 1);

if(map.get(cTemp) == 0){

counter--;

}

start++;

}

len = Math.max(len, end-start);

}

return len;

}

}

**Substring with Concatenation of All Words**  
<https://leetcode.com/problems/substring-with-concatenation-of-all-words/>

public class Solution {

public List<Integer> findSubstring(String S, String[] L) {

List<Integer> res = new LinkedList<>();

if (L.length == 0 || S.length() < L.length \* L[0].length()) return res;

int N = S.length();

int M = L.length; // \*\*\* length

int wl = L[0].length();

Map<String, Integer> map = new HashMap<>(), curMap = new HashMap<>();

for (String s : L) {

if (map.containsKey(s)) map.put(s, map.get(s) + 1);

else map.put(s, 1);

}

String str = null, tmp = null;

for (int i = 0; i < wl; i++) {

int count = 0; // remark: reset count

int start = i;

for (int r = i; r + wl <= N; r += wl) {

str = S.substring(r, r + wl);

if (map.containsKey(str)) {

if (curMap.containsKey(str)) curMap.put(str, curMap.get(str) + 1);

else curMap.put(str, 1);

if (curMap.get(str) <= map.get(str)) count++;

while (curMap.get(str) > map.get(str)) {

tmp = S.substring(start, start + wl);

curMap.put(tmp, curMap.get(tmp) - 1);

start += wl;

//the same as https://leetcode.com/problems/longest-substring-without-repeating-characters/

if (curMap.get(tmp) < map.get(tmp)) count--;

}

if (count == M) {

res.add(start);

tmp = S.substring(start, start + wl);

curMap.put(tmp, curMap.get(tmp) - 1);

start += wl;

count--;

}

}else {

curMap.clear();

count = 0;

start = r + wl;//not contain, so move the start

}

}

curMap.clear();

}

return res;

}

}

**Find All Anagrams in a String**  
<https://leetcode.com/problems/find-all-anagrams-in-a-string/>

public class Solution {

public List<Integer> findAnagrams(String s, String t) {

List<Integer> result = new LinkedList<>();

if(t.length()> s.length()) return result;

Map<Character, Integer> map = new HashMap<>();

for(char c : t.toCharArray()){

map.put(c, map.getOrDefault(c, 0) + 1);

}

int counter = map.size();

int begin = 0, end = 0;

int head = 0;

int len = Integer.MAX\_VALUE;

while(end < s.length()){

char c = s.charAt(end);

if( map.containsKey(c) ){

map.put(c, map.get(c)-1);

if(map.get(c) == 0) counter--;

}

end++;

while(counter == 0){

char tempc = s.charAt(begin);

if(map.containsKey(tempc)){

map.put(tempc, map.get(tempc) + 1);

if(map.get(tempc) > 0){

counter++;

}

}

if(end-begin == t.length()){

result.add(begin);

}

begin++;

}

}

return result;

}

}