**Longest Palindromic Substring**

class Solution {  
 public String longestPalindrome(String s) {  
 int n = s.length();  
 boolean dp[][] = new boolean[n][n];  
 int maxLength =1;  
 for (int i = 0; i < n-1; i++) {  
 dp[i][i] = true;  
 }  
 int start = 0;  
 int end = 0;  
 for (int i = 0; i < n-1; i++) {  
 if (s.charAt(i) == s.charAt(i+1)){  
 dp[i][i+1] = true;  
 start = i;  
 end = i+1;  
 maxLength = 2;  
 }  
 }  
 for (int k = 3; k <=n ; k++) {  
 for (int i = 0; i < n-k+1; i++) {  
 int j = i+k-1;  
 if (dp[i+1][j-1]== true && s.charAt(i) == s.charAt(j)){  
 dp[i][j] = true;  
 if (k>maxLength){  
 maxLength = k;  
 start = i;  
 end = j;  
 }  
 }  
 }  
 }  
 return s.substring(start,end+1);  
 }  
}

**Valid Parentheses**

* Problem: Determine if the parentheses in a string are valid.

class Solution {  
 public boolean isValid(String s) {  
 Stack<Character> stk = new Stack<>();  
 for (int i=0;i<s.length();i++) {  
 char ch = s.charAt(i);  
 if (ch == '[' || ch == '{' || ch == '(') {  
 stk.push(ch);  
 }  
 else {  
 if (stk.isEmpty()) {  
 return false;  
 }  
 if (stk.peek() == '[' && ch == ']'  
 || stk.peek() == '{' && ch == '}'  
 || stk.peek() == '(' && ch == ')') {  
 stk.pop();  
 }  
 else {  
 return false;  
 }  
 }  
 }  
 return stk.isEmpty();  
 }  
}

**Longest Substring Without Repeating Characters**

* Problem: Find the length of the longest substring without repeating characters.

class Solution {  
 public int lengthOfLongestSubstring(String s) {  
 if (s.length() == 0) return 0;  
 HashMap<Character,Integer> map = new HashMap<>();  
 int len = 0;  
 int left = 0;  
 int right = 0;  
 while (right < s.length()){  
 if (map.containsKey(s.charAt(right))){  
 int val = map.get(s.charAt(right));  
 if (val == s.length()-1){  
 left = val;  
 }  
 left = Math.max(left,val+1);  
 }  
 map.put(s.charAt(right),right);  
 len = Math.max(len,right-left+1);  
 right++;  
 }  
 return len;   
 }  
}

**Check Anagrams**

* Problem: Check if two strings are permutations of each other.

class Solution {  
 public boolean isAnagram(String s, String t) {  
 HashMap<Character,Integer> hm1 = new HashMap<>();  
 for (int i=0;i<s.length();i++){  
 char ch = s.charAt(i);  
 hm1.put(ch,hm1.getOrDefault(ch,0) +1);  
 }  
 for (int i=0;i<t.length();i++){  
 char ch = t.charAt(i);  
 if (hm1.get(ch) != null){  
 if (hm1.get(ch) == 1){  
 hm1.remove(ch);  
 }  
 else {  
 hm1.put(ch,hm1.get(ch)-1);  
 }  
 }  
 else {  
 return false;  
 }  
 }  
 return hm1.isEmpty();  
 }  
}