**Hashing - 2:**

**Longest Palindrome:**

class Solution {

public int longestPalindrome(String s) {

if(s==null || s.length() == 0){

return 0;

}

Set<Character> set = new HashSet<>();

int count = 0;

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

if(!set.contains(c)){

set.add(c);

}

else{

count+=2;

set.remove(c);

}

}

return set.isEmpty()?count:count+1;

}

/\*\*

The time complexity is O(N).

The Space Complexity is O(1).

\*/

Contiguous subarray:

class Solution {

public int findMaxLength(int[] nums) {

if(nums == null || nums.length == 0)

return 0;

Map<Integer,Integer> sumMap = new HashMap<>();

sumMap.put(0,-1);

int rSum = 0, maxLen = 0;

for(int i = 0; i<nums.length;i++){

//calculating rSum

rSum = nums[i]==0 ?rSum -1:rSum+1;

//checking if rSum exists in map or not

if(!sumMap.containsKey(rSum)){

sumMap.put(rSum,i);

}

else{

// maxLen <> existing value

maxLen = Math.max(maxLen, i-sumMap.get(rSum));

}

}

return maxLen;

}

}

/\*\*

The Time complexity is O(N)

The space Complexity is O(N).

\*/

SubArray sum equals to K

class Solution {

public int subarraySum(int[] nums, int k) {

if(nums== null || nums.length == 0)

return 0;

Map<Integer,Integer> sumMap = new HashMap<>();

sumMap.put(0,1);

int rSum = 0, count = 0;

for(int i = 0; i<nums.length;i++){

rSum += nums[i];

int n = sumMap.getOrDefault(rSum-k, 0);

count += n;

sumMap.put(rSum, sumMap.getOrDefault(rSum,0)+1);

}

return count;

}

}