Leetcode problems

1→(1)Two sum

```
class Solution:
    def twoSum(self, nums, target):
        a=[]
        for i in range(len(nums)):
            for j in range(i+1,len(nums)):
                if nums[i]+nums[j]==target:
                    a.append(i)
                    a.append(j)
                    return a
        return twoSum(nums,target)
```

2→(9)palindrome number

```
class Solution:
    def isPalindrome(self, x: int) :
        m=x
        temp=0
        while x>0:
            rem=x%10
            temp=temp*10+rem
            x=x//10
        if m==temp:
            return True
        else:
            return False
```

3→(13)Roman to Integer

```
class Solution:
    def romanToInt(self, s: str):
        m = {
            'I': 1,
            'V': 5,
            'X': 10,
            'L': 50,
            'C': 100,
            'D': 500,
            'M': 1000
        }

        ans = 0

        for i in range(len(s)):
            if i < len(s) - 1 and m[s[i]] < m[s[i+1]]:
                ans -= m[s[i]]
            else:
                ans += m[s[i]]

        return ans
```

4→(14)Longest common prefix

```
class Solution:
    def longestCommonPrefix(self, strs):
        strs=sorted(strs)
        if len(strs)==1:
            return strs[0]
        s=strs[0]
        ans=""
        flag=0
```

```
        q=0
        for i in s:
            for j in strs:
                if i!=j[q]:
                    return ans
            else:
                ans=ans+i
                q+=1
        return ans
```

5→(20)valid parenthesis

```
class Solution:
    def isValid(self, s: str) -> bool:
        a=[]
        for i in s:
            if i=="(" or i=="{" and i=="[":
                a.append(i)
            elif (i==")" or i=="}" or i=="]") and  len(a)==0:
                   return False
            else:
                if i==")" and a[-1]=="(":
                    a.pop()
                elif i=="}" and a[-1]=="{":
                    a.pop()
                elif i=="]" and a[-1]=="[":
                    a.pop()
                elif i=="}" and a[-1]=="{":
                    a.pop()
                else:
                    a.append(i)
        if len(a)==0:
            return True
        else:
```

```
            return False
```

6→(28)Find the index of the first occurrence in the string

```
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        if len(haystack)<len(needle):
            return -1
        for i in range(len(haystack)-len(needle)+1):
            if haystack[i:i+len(needle)]==needle:
                return i
        return -1
```

7→(58)Length of last word

```
class Solution:
    def lengthOfLastWord(self, s: str) -> int:
        a=s.strip().split(" ")
        if not s:
            return 0
        return len(a[-1])
```

8→(66)Plus one

```
class Solution:
    def plusOne(self, digits: List[int]) -> List[int]:
        m=len(digits)
        for i in range(m-1,-1,-1):
            if digits[i]<9:
                digits[i]+=1
                return digits
```

```
            digits[i]=0
        return [1]+digits
```

9→(69)Sqrt(x)

```
class Solution:
    def mySqrt(self, x: int) -> int:
        i=1
        while i**2<x:
           i+=1
        if i**2==x:
            return i
        else:
            return (i+x//i)//2
```

10→(3110)Score of a string

```
class Solution:
    def scoreOfString(self, s: str) -> int:
        m=0
        for i in range(len(s)-1):
            m=m+abs(ord(s[i])-ord(s[i+1]))
        return m
```

11→(1108)Defanging of an IP address

```
class Solution:
    def defangIPaddr(self, address: str) -> str:
        a=""
        for i in address:
            if i==".":
                a=a+"[.]"
```

```
            else:
                a=a+i
        return a
```

12→(2011)Find value of variable after performing operation

```
class Solution:
    def finalValueAfterOperations(self, operations: List[str]) 
        x=0
        for i in operations:
            if i[0]=="+" or i[-1]=="+":
                x+=1
            else:
                x-=1
        return x
```

13→(771)Jewels and stones

```
class Solution:
    def numJewelsInStones(self, jewels: str, stones: str) -> int
        count=0
        for i in jewels:
            for j in stones:
                if i==j:
                    count=count+1
        return count
```

14→(2942)Find words containing characters

```
class Solution:
    def findWordsContaining(self, words: List[str], x: str) -> L
        index=[]
```

```
        for i in range(0,len(words)):
            if x in words[i]:
                index.append(i)
        return index
```

15→(1678)Goal parser interpretation

```
class Solution:
    def interpret(self, command: str) -> str:
        a=""
        for i in range(len(command)):
            if command[i]=="G":
                a=a+"G"
            elif command[i]=="(" and command[i+1]==")":
                a=a+"o"
            elif command[i]=="(" and command[i+1]=="a":
                    a=a+"al"
        return a
```

16→(2114)Maximum no.of words present in a sentence

```
class Solution:
    def mostWordsFound(self, sentences: List[str]) -> int:
        count=[]
        for i in sentences:
           l=i.split(" ")
           count.append(len(l))
        return max(count)
```

17→(1221)Split a string in balanced strings

```
class Solution:
    def balancedStringSplit(self, s: str) -> int:
        l=0
        r=0
        count=0
        for i in s:
            if i=='R':
                r+=1
            else:
                l+=1
            if l==r:
                count+=1
                l=0
                r=0
        return count
```

18→(1662)Check if two strings arrays are equivalent

```
class Solution:
    def arrayStringsAreEqual(self, word1: List[str], word2: List
                if "".join(word1)=="".join(word2):
                    return True
                else:
                    return False
```

19→(1773)Count items matching a rule

```
class Solution:
    def countMatches(self, items: List[List[str]], ruleKey: str,
        count=0
        if ruleKey=="type":
```

```
            for i in items:
                if i[0]==ruleValue:
                    count+=1
        elif ruleKey=="color":
            for i in items:
                if i[1]==ruleValue:
                    count+=1
        elif ruleKey=="name":
            for i in items:
                if i[2]==ruleValue:
                    count+=1
        return count
```

20→(1816)Truncate sentence

```
class Solution:
    def truncateSentence(self, s: str, k: int) -> str:
        m=""
        l=s.split( )
        for i in range(k):
            if i==k-1:
                m=m+l[i]
                return m
            m=m+l[i]+" "
```

21→(1528)Shuffle string

```
class Solution:
    def restoreString(self, s: str, indices: List[int]) -> str:
        m=""
        indices=list(indices)
        for i in range(len(s)):
            b=indices.index(i)
```

```
            m+=s[b]
        return m
```

22→(2325)Decode the message

```
lass Solution:
    def decodeMessage(self, key: str, message: str) -> str:
        l1=""
        l2="abcdefghijklmnopqrstuvwxyz"
        for i in key:
           if i not in l1 and i!=" ":
              l1=l1+i
        ans=""
        for i in message:
            if i!=" ":
              a=l1.index(i)
              ans=ans+l2[a]
            else:
                ans=ans+" "
        return ans
```

23→(2108)Find fist palindromic string in the array

```
class Solution:
    def firstPalindrome(self, words: List[str]) -> str:
        for i in words:
            if i[::]==i[::-1]:
                return i
        else:
                return ""
```

24→(2194)Cells in a range of an excel sheet

```
class Solution:
    def cellsInRange(self, s: str) -> List[str]:
        i=s
        a=ord(i[0])
        b=ord(i[3])
        c=int(i[1])
        d=int(i[4])
        l=[]
        for i in range(a,b+1):
            for j in range(c,d+1):
                s=chr(i)+str(j)
                l.append(s)
        return l
```

25→(1614)Maximum nesting depth of the parenthesis

```
class Solution:
    def maxDepth(self, s: str) -> int:
        c=0
        max=0
        for i in s:
            if i=="(":
                c+=1
            if max<c:
                max=c
            if i==")":
                c-=1
        return max
```

26→(709)To lower case

```
class Solution:
    def toLowerCase(self, s: str) -> str:
        s1=""
        for i in range(len(s)):
            s1=s1+s[i].lower()
        return s1
```

27→(2810)Faulty keyboard

```
class Solution:
    def finalString(self, s: str) -> str:
        temp=""
        for i in s:
            if i!="i":
                temp=temp+i
            else:
                temp=temp[::-1]
        return temp
```

28→(1832)Check if the sentence is pangram

```
class Solution:
    def checkIfPangram(self, sentence: str) -> bool:
        a="abcdefghijklmnopqrstuvwxyz"
        for i in a:
            if i not in sentence:
              return False
        else:
          return True
```

29→(557)Reverse words in a string III

```
class Solution:
    def reverseWords(self, s: str) -> str:
        a=""
        t=s.split(" ")
        for i in range(len(t)):
            if i==len(t)-1:
                a=a+t[i][::-1]
            else:
                a=a+t[i][::-1]+" "
        return a
```

30→(1684)Count the number of consistent strings

```
class Solution:
    def countConsistentStrings(self, allowed: str, words: List[s
        c=0
        allowed=set(allowed)
        for i in words:
            for j in i:
                if j not in allowed:
                    c+=1
                    break
        return len(words)-c
```

31→(1859)Sorting the sentence

```
class Solution:
    def sortSentence(self, s: str) -> str:
        a = s.split()
        b = [0] * len(a)
        for i in a:
```

```
            b[int(i[-1]) - 1] = i[0:-1]
        return " ".join(b)
```

32→(2828)Check if a string is a acronym of words

```
class Solution:
    def isAcronym(self, words: List[str], s: str) -> bool:
        if len(words)!=len(s):
            return False
        for i in range(len(s)):
            if s[i]!=words[i][0]:
                return False
        else:
             return True
```

33→(804)Unique morse code words

```
class Solution:
    def uniqueMorseRepresentations(self, words: List[str]) -> in
        morse=[".-","-...","-.-.","-..",".","..-.","--.","....",
        l=[]
        for j in words:
            ans=""
            for i in j:
                ans=ans+morse[ord(i)-97]
            l.append(ans)
        q=set(l)
        return len(q)
```

34→(2744)Find maximum number of string pairs

```
class Solution:
    def maximumNumberOfStringPairs(self, words: List[str]) -> in
        pairs=0
        s=set()
        for i in words:
            if i in s:
                pairs+=1
            else:
                s.add(i[::-1])
        return pairs
```

35→(2000)Reverse prefix of word

```
class Solution:
    def reversePrefix(self, word: str, ch: str) -> str:
        index=0
        for i in range(len(word)):
            if word[index]==ch:
                return word[index::-1]+word[index+1:]
            index+=1
        return word
```

36→(2418)Sort the people

```
class Solution:
    def sortPeople(self, names: List[str], heights: List[int]) -
        a=[]
        for i in range(len(names)):
            a.append([heights[i],names[i]])
        a.sort(reverse=True)
        b=[]
        for i in range(len(names)):
```

```
            b.append(a[i][1])
        return b
```

37→(344)Reverse string

```
class Solution:
    def reverseString(self, s: List[str]) -> None:
        return s.reverse()
```

38→(13)Roman to integer

```
class Solution:
    def romanToInt(self, s: str):
        m = {
            'I': 1,
            'V': 5,
            'X': 10,
            'L': 50,
            'C': 100,
            'D': 500,
            'M': 1000
        }

        ans = 0

        for i in range(len(s)):
            if i < len(s) - 1 and m[s[i]] < m[s[i+1]]:
                ans -= m[s[i]]
            else:
                ans += m[s[i]]

        return ans
```

39→(70)Climbing stairs

```
class Solution:
    def climbStairs(self, n: int) -> int:
        ways=0
        while n:
            if ways=="1" or ways=="2":
                return ways
            else:
                return 0
```

40→(26)Remove duplicates from sorted array

```
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        s=set(nums)
        l=list(s)
        l.sort()
        for i in range(len(l)):
            nums[i]=l[i]
        return len(l)
```

41→(27)Remove Element

```
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
        index=0
        for i in range(len(nums)):
            if nums[i]!=val:
                nums[index]=nums[i]
```

```
                index+=1
        return index
```

42→(225)Implement stack using queues

```
class MyStack:

    def __init__(self):
        self.q1=[]
        self.q2=[]

    def push(self, x: int) -> None:
        self.q2.append(x)

    def pop(self) -> int:
        for i in range(len(self.q2)):
            self.q2.append(self.q2.pop())
        a=self.q2.pop()
        for i in range(len(self.q1)):
            self.q1.append(self.q1.pop())
        return a

    def top(self) -> int:
        return self.q2[-1]

    def empty(self) -> bool:
        if len(self.q2)==0:
            return True
        else:
            return False
```

43→(232)Implement queue using stacks

```
class MyQueue:

    def __init__(self):
        self.s1=[]
        self.s2=[]


    def push(self, x: int) -> None:
        self.s1.append(x)

    def pop(self) -> int:
        for i in range(len(self.s1)):
            self.s2.append(self.s1.pop())
        a=self.s2.pop()
        for i in range(len(self.s2)):
            self.s1.append(self.s2.pop())
        return a

    def peek(self) -> int:
        return self.s1[0]

    def empty(self) -> bool:
        if len(self.s1)==0:
            return True
        else:
            return False
```

44→(599)Minimum index sum of two lists

```
class Solution:
    def findRestaurant(self, list1: List[str], list2: List[str]
        sum=[]
        ans=[]
        for i in range(0,len(list1)):
```

```
            for j in range(0,len(list2)):
                if list1[i]==list2[j]:
                    sum.append(i+j)
        m=min(sum)
        for i in range(0,len(list1)):
            for j in range(0,len(list2)):
                if list1[i]==list2[j] and i+j==m:
                    ans.append(list1[i])
        return ans
```

45→(2315)Count Asterisks

```
class Solution:
    def countAsterisks(self, s: str) -> int:
        n=""
        c=0
        t=0
        for i in s:
            if i=="|":
                t=t+1
            elif t%2==0:
                c+= i=="*"
        return c
```

63→(121)Best time to buy and sell stock

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        buy_price=prices[0]
        profit=0
        for i in prices:
            if i<buy_price:
                buy_price=i
```

```
            profit=max(profit,i-buy_price)
        return profit
```

64→(136)single number

```
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        n=0
        for i in nums:
            n^=i
        return n
```

65→(144)Binary tree preorder traversal

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def preorderTraversal(self, root: Optional[TreeNode]) -> Lis
        s=[]
        def order(root,s):
          if root:
            s.append(root.val)
            order(root.left,s)
            order(root.right,s)
        order(root,s)
        return s
```

66→(15)3Sum

```
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        b=[]
        nums.sort()
        for i in range(0,len(nums)-2):
            j=i+1
            k=len(nums)-1
            while j<k:
                a=[]
                if nums[i]+nums[j]+nums[k]==0:
                    a.append(nums[i])
                    a.append(nums[j])
                    a.append(nums[k])
                    b.append(a)
                k-=1
                j+=1
        return b
```