**TWO SUM**

class Solution:

def twoSum(self, nums: List[int], target: int) -> List[int]:

hashmap = {}

for i in range(len(nums)):

hashmap[nums[i]] = i

for i in range(len(nums)):

complement = target - nums[i]

if complement in hashmap and hashmap[complement] != i:

return [i, hashmap[complement]]

# If no valid pair is found, return an empty list

return []

**ADD TWO NUMBERS**

# Definition for singly-linked list.

# class ListNode:

# def \_\_init\_\_(self, val=0, next=None):

# self.val = val

# self.next = next

class Solution:

def addTwoNumbers(

self, l1: Optional[ListNode], l2: Optional[ListNode]

) -> Optional[ListNode]:

dummyHead = ListNode(0)

curr = dummyHead

carry = 0

while l1 != None or l2 != None or carry != 0:

l1Val = l1.val if l1 else 0

l2Val = l2.val if l2 else 0

columnSum = l1Val + l2Val + carry

carry = columnSum // 10

newNode = ListNode(columnSum % 10)

curr.next = newNode

curr = newNode

l1 = l1.next if l1 else None

l2 = l2.next if l2 else None

return dummyHead.next

**LONGEST SUBSTRING**

class Solution:

def lengthOfLongestSubstring(self, s: str) -> int:

n = len(s)

maxLength = 0

charSet = set()

left = 0

for right in range(n):

if s[right] not in charSet:

charSet.add(s[right])

maxLength = max(maxLength, right - left + 1)

else:

while s[right] in charSet:

charSet.remove(s[left])

left += 1

charSet.add(s[right])

return maxLength

**MEDIAN OF TWO ARRAY SORTED LIST**

class Solution:

def findMedianSortedArrays(self, nums1: List[int], nums2: List[int]) -> float:

# Merge the two sorted arrays

merged = sorted(nums1 + nums2)

length = len(merged)

# Check if the total length is even or odd

if length % 2 == 0:

# If even, return the average of the two middle elements

return (merged[length // 2 - 1] + merged[length // 2]) / 2

else:

# If odd, return the middle element

return merged[length // 2]

**LONGEST PALINDROME**

class Solution:

def longestPalindrome(self, s: str) -> str:

if len(s) <= 1:

return s

Max\_Len=1

Max\_Str=s[0]

for i in range(len(s)-1):

for j in range(i+1,len(s)):

if j-i+1 > Max\_Len and s[i:j+1] == s[i:j+1][::-1]:

Max\_Len = j-i+1

Max\_Str = s[i:j+1]

return Max\_Str

**ZIGZAG CONVERSION**

class Solution:

def convert(self, s: str, numRows: int) -> str:

if numRows == 1 or numRows >= len(s):

return s

idx, d = 0, 1

rows = [[] for \_ in range(numRows)]

for char in s:

rows[idx].append(char)

if idx == 0:

d = 1

elif idx == numRows - 1:

d = -1

idx += d

for i in range(numRows):

rows[i] = ''.join(rows[i])

return ''.join(rows)

**REVERSE AN INTEGER**

class Solution:

def reverse(self, x: int) -> int:

ans = int(str(abs(x))[::-1]) \* (-1 if x < 0 else 1)

return ans if -2\*\*31 <= ans < 2\*\*31 else 0

**STRING TO INTEGER**

class Solution(object):

def myAtoi(self, s):

"""

:type s: str

:rtype: int

"""

i = 0

n = len(s)

while i < n and s[i] == ' ':

i += 1

sign = 1

if i < n and (s[i] == '-' or s[i] == '+'):

sign = -1 if s[i] == '-' else 1

i += 1

result = 0

while i < n and s[i].isdigit():

result = result \* 10 + int(s[i])

if result \* sign > 2\*\*31 - 1:

return 2\*\*31 - 1

if result \* sign < -2\*\*31:

return -2\*\*31

i += 1

return result \* sign

**PALINDROME NUMBER**

class Solution:

def isPalindrome(self, x: int) -> bool:

if x < 0:

return False

reverse = 0

xcopy = x

while x > 0:

reverse = (reverse \* 10) + (x % 10)

x //= 10

return reverse == xcopy

**REGULAR EXPRESSION MATCHING**

class Solution(object):

def isMatch(self, s, p):

"""

:type s: str

:type p: str

:rtype: bool

"""

m, n = len(s), len(p)

dp = [[False] \* (n + 1) for \_ in range(m + 1)]

dp[0][0] = True

for j in range(1, n + 1):

if p[j - 1] == '\*':

dp[0][j] = dp[0][j - 2]

for i in range(1, m + 1):

for j in range(1, n + 1):

if p[j - 1] in {s[i - 1], '.'}:

dp[i][j] = dp[i - 1][j - 1]

elif p[j - 1] == '\*':

dp[i][j] = dp[i][j - 2]

if p[j - 2] in {s[i - 1], '.'}:

dp[i][j] = dp[i][j] or dp[i - 1][j]

return dp[m][n]

**CONTAINER WITH WATER**

class Solution {

public:

int maxArea(vector<int>& height) {

int left = 0, right = height.size() - 1;

int maxWater = 0;

while (left < right) {

int h = min(height[left], height[right]);

int w = right - left;

maxWater = max(maxWater, h \* w);

// Move the pointer pointing to the shorter line

if (height[left] < height[right])

left++;

else

right--;

}

return maxWater;

}

};

**3SUM**

class Solution {

public:

vector<vector<int>> threeSum(vector<int>& nums) {

vector<vector<int>> result;

sort(nums.begin(), nums.end());

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

if (i > 0 && nums[i] == nums[i - 1]) continue; // skip duplicates

int left = i + 1;

int right = nums.size() - 1;

while (left < right) {

int sum = nums[i] + nums[left] + nums[right];

if (sum == 0) {

result.push\_back({nums[i], nums[left], nums[right]});

left++;

right--;

// skip duplicates

while (left < right && nums[left] == nums[left - 1]) left++;

while (left < right && nums[right] == nums[right + 1]) right--;

}

else if (sum < 0) {

left++;

}

else {

right--;

}

}

}

return result;

}

};

**4SUM**

class Solution {

public:

vector<vector<int>> fourSum(vector<int>& nums, int target) {

vector<vector<int>> result;

int n = nums.size();

sort(nums.begin(), nums.end());

for (int i = 0; i < n - 3; ++i) {

if (i > 0 && nums[i] == nums[i - 1]) continue; // skip duplicates

for (int j = i + 1; j < n - 2; ++j) {

if (j > i + 1 && nums[j] == nums[j - 1]) continue; // skip duplicates

int left = j + 1;

int right = n - 1;

while (left < right) {

long long sum = (long long)nums[i] + nums[j] + nums[left] + nums[right];

if (sum == target) {

result.push\_back({nums[i], nums[j], nums[left], nums[right]});

++left;

--right;

while (left < right && nums[left] == nums[left - 1]) ++left;

while (left < right && nums[right] == nums[right + 1]) --right;

}

else if (sum < target) {

++left;

}

else {

--right;

}

}

}

}

return result;

}

};

**VALID PARENTHESES**

class Solution {

public:

bool isValid(string s) {

stack<char> st;

unordered\_map<char, char> bracketMap = {

{')', '('},

{']', '['},

{'}', '{'}

};

for (char c : s) {

if (c == '(' || c == '[' || c == '{') {

st.push(c);

} else {

if (st.empty() || st.top() != bracketMap[c]) {

return false;

}

st.pop();

}

}

return st.empty();

}

};