ASSIGNMENT – 2

Name : Keerthiga Veni

Reg numb : 192324001

Course code : CSA0664

**11. To Return the maximum amount of water a container can store.**

class Solution:

def maxArea(self, height: List[int]) -> int:

left = 0

right = len(height) - 1

maxArea = 0

while left < right:

currentArea = min(height[left], height[right]) \* (right - left)

maxArea = max(maxArea, currentArea)

if height[left] < height[right]:

left += 1

else:

right -= 1

return maxArea

**12. Given an integer, convert it to a roman numeral.**

class Solution:

def intToRoman(self, num: int) -> str:

ones = ["", "I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX"]

tens = ["", "X", "XX", "XXX", "XL", "L", "LX", "LXX", "LXXX", "XC"]

hrns = ["", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC", "CM"]

ths = ["", "M", "MM", "MMM"]

return ths[num // 1000] + hrns[(num % 1000) // 100] + tens[(num % 100) // 10] + ones[num % 10]

sol = Solution()

number = 1987 # You can change this number to test other cases

result = sol.intToRoman(number)

print(f"The Roman numeral for {number} is {result}")

**13. Given a roman numeral, convert it to an integer.**

class Solution:

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

translations = {

"I": 1,

"V": 5,

"X": 10,

"L": 50,

"C": 100,

"D": 500,

"M": 1000

}

number = 0

s = s.replace("IV", "IIII").replace("IX", "VIIII")

s = s.replace("XL", "XXXX").replace("XC", "LXXXX")

s = s.replace("CD", "CCCC").replace("CM", "DCCCC")

for char in s:

number += translations[char]

return number

sol = Solution()

roman\_numeral = "MCMLXXXVII" # You can change this Roman numeral to test other cases

result = sol.romanToInt(roman\_numeral)

print(f"The integer for Roman numeral {roman\_numeral} is {result}")

**14. To find the longest common prefix string amongst an array of strings.**

class Solution:

def longestCommonPrefix(self, strs):

if not strs:

return ""

min\_length = min(len(s) for s in strs)

low, high = 1, min\_length

while low <= high:

mid = (low + high) // 2

if self.isCommonPrefix(strs, mid):

low = mid + 1

else:

high = mid - 1

return strs[0][: (low + high) // 2]

def isCommonPrefix(self, strs, length):

prefix = strs[0][:length]

for s in strs:

if not s.startswith(prefix):

return False

return True

sol = Solution()

print(sol.longestCommonPrefix(["flower", "flow", "flight"]))

print(sol.longestCommonPrefix(["dog", "racecar", "car"]))

print(sol.longestCommonPrefix(["interspace", "internet", "interval"]))

print(sol.longestCommonPrefix(["", "b", "c"]))

print(sol.longestCommonPrefix(["a"]))

**15. Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.**

class Solution:

def threeSum(self, nums):

nums.sort

result = []

for i in range(len(nums) - 2):

if i > 0 and nums[i] == nums[i - 1]:

continue

left, right = i + 1, len(nums) - 1

while left < right:

total = nums[i] + nums[left] + nums[right]

if total < 0:

left += 1

elif total > 0:

right -= 1

else:

result.append([nums[i], nums[left], nums[right]])

while left < right and nums[left] == nums[left + 1]:

left += 1

while left < right and nums[right] == nums[right - 1]:

right -= 1

left += 1

right -= 1

return result

sol = Solution()

print(sol.threeSum([-1, 0, 1, 2, -1, -4]))

print(sol.threeSum([0, 1, 1]))

print(sol.threeSum([0, 0, 0]))

print(sol.threeSum([-2, 0, 1, 1, 2]))

**16. Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to target. Return the sum of the three integers.**

class Solution:

def threeSumClosest(self, nums, target):

nums.sort()

closest\_sum = float('inf')

for i in range(len(nums) - 2):

left, right = i + 1, len(nums) - 1

while left < right:

current\_sum = nums[i] + nums[left] + nums[right]

if current\_sum == target:

return current\_sum

if abs(current\_sum - target) < abs(closest\_sum - target):

closest\_sum = current\_sum

if current\_sum < target:

left += 1

else:

right -= 1

return closest\_sum

sol = Solution()

print(sol.threeSumClosest([-1, 2, 1, -4], 1)) # Output: 2

print(sol.threeSumClosest([0, 0, 0], 1)) # Output: 0

print(sol.threeSumClosest([1, 1, -1, -1, 3], -1)) # Output: -1

print(sol.threeSumClosest([1, 2, 5, 10, 11], 12)) # Output: 13

**17. Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order.**

class Solution:

def letterCombinations(self, digits):

if not digits:

return []

phone\_map = {

"2": "abc", "3": "def", "4": "ghi", "5": "jkl",

"6": "mno", "7": "pqrs", "8": "tuv", "9": "wxyz"

}

result = []

def backtrack(index, path):

if index == len(digits):

result.append("".join(path))

return

possible\_letters = phone\_map[digits[index]]

for letter in possible\_letters:

path.append(letter)

backtrack(index + 1, path)

path.pop()

backtrack(0, [])

return result

sol = Solution()

print(sol.letterCombinations("23"))

print(sol.letterCombinations(""))

print(sol.letterCombinations("2"))

print(sol.letterCombinations("234"))

**18. Given an array nums of n integers, return an array of all the unique quadruplets**

class Solution:

def fourSum(self, nums, target):

nums.sort()

result = []

n = len(nums)

for i in range(n - 3):

if i > 0 and nums[i] == nums[i - 1]:

continue

for j in range(i + 1, n - 2):

if j > i + 1 and nums[j] == nums[j - 1]:

continue

left, right = j + 1, n - 1

while left < right:

total = nums[i] + nums[j] + nums[left] + nums[right]

if total == target:

result.append([nums[i], nums[j], nums[left], nums[right]])

while left < right and nums[left] == nums[left + 1]:

left += 1

while left < right and nums[right] == nums[right - 1]:

right -= 1

left += 1

right -= 1

elif total < target:

left += 1

else:

right -= 1

return result

sol = Solution()

print(sol.fourSum([1, 0, -1, 0, -2, 2], 0)) # Output: [[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]

print(sol.fourSum([2, 2, 2, 2, 2], 8)) # Output: [[2, 2, 2, 2]]

print(sol.fourSum([0, 0, 0, 0], 0)) # Output: [[0, 0, 0, 0]]

print(sol.fourSum([-3, -1, 0, 2, 4, 5], 2)) # Output: [[-3, -1, 2, 4]]

**19. Remove Nth Node From End of List**

class ListNode:

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

self.val = val

self.next = next

class Solution:

def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:

dummy = ListNode(0, head)

first = dummy

second = dummy

for \_ in range(n + 1):

first = first.next

while first:

first = first.next

second = second.next

second.next = second.next.next

return dummy.next

def create\_linked\_list(arr):

if not arr:

return None

head = ListNode(arr[0])

current = head

for value in arr[1:]:

current.next = ListNode(value)

current = current.next

return head

def print\_linked\_list(head):

current = head

result = []

while current:

result.append(current.val)

current = current.next

print(result)

sol = Solution()

head = create\_linked\_list([1, 2, 3, 4, 5])

new\_head = sol.removeNthFromEnd(head, 2)

print\_linked\_list(new\_head)

head = create\_linked\_list([1])

new\_head = sol.removeNthFromEnd(head, 1)

print\_linked\_list(new\_head)

head = create\_linked\_list([1, 2])

new\_head = sol.removeNthFromEnd(head, 1)

print\_linked\_list(new\_head)

**20. Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.**

class Solution:

def isValid(self, s: str) -> bool:

stack = []

mapping = {')': '(', '}': '{', ']': '['}

for char in s:

if char in mapping:

top\_element = stack.pop() if stack else '#'

if mapping[char] != top\_element:

return False

else:

stack.append(char)

return not stack

sol = Solution()

print(sol.isValid("()"))

print(sol.isValid("()[]{}"))

print(sol.isValid("(]"))