**ASSIGNMENT 1- CSA0666**

**1.** **Two Sum Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target. You may assume that each input would have exactly one solution, and you may not use the same element twice. You can return the answer in any order.**

def two\_sum(nums, target):

num\_to\_index = {}

for i, num in enumerate(nums):

complement = target - num

if complement in num\_to\_index:

return [num\_to\_index[complement], i]

num\_to\_index[num] = i

print(two\_sum([2, 7, 11, 15], 9))

print(two\_sum([3, 2, 4], 6))

print(two\_sum([3, 3], 6))

**2. Add Two Numbers You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list. You may assume the two numbers do not contain any leading zero, except the number 0 itself.**

class ListNode:

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

self.val = val

self.next = next

def addTwoNumbers(l1, l2):

dummy\_head = ListNode(0)

current = dummy\_head

carry = 0

while l1 or l2 or carry:

val1 = l1.val if l1 else 0

val2 = l2.val if l2 else 0

# Compute the new sum and the new carry

total = val1 + val2 + carry

carry = total // 10

new\_val = total % 10

# Append the new value to the result list

current.next = ListNode(new\_val)

current = current.next

# Move to the next nodes

if l1:

l1 = l1.next

if l2:

l2 = l2.next

return dummy\_head.next

# Helper function to convert list to linked list

def list\_to\_linkedlist(lst):

dummy = ListNode(0)

current = dummy

for number in lst:

current.next = ListNode(number)

current = current.next

return dummy.next

# Helper function to convert linked list to list

def linkedlist\_to\_list(node):

result = []

while node:

result.append(node.val)

node = node.next

return result

# Test cases

l1 = list\_to\_linkedlist([2, 4, 3])

l2 = list\_to\_linkedlist([5, 6, 4])

result = addTwoNumbers(l1, l2)

print(linkedlist\_to\_list(result)) # Output: [7, 0, 8]

l1 = list\_to\_linkedlist([0])

l2 = list\_to\_linkedlist([0])

result = addTwoNumbers(l1, l2)

print(linkedlist\_to\_list(result)) # Output: [0]

l1 = list\_to\_linkedlist([9, 9, 9, 9, 9, 9, 9])

l2 = list\_to\_linkedlist([9, 9, 9, 9])

result = addTwoNumbers(l1, l2)

print(linkedlist\_to\_list(result)) # Output: [8, 9, 9, 9, 0, 0, 0, 1]

3. **Longest Substring without Repeating Characters Given a string s, find the length of the longest substring without repeating characters.**

def lengthOfLongestSubstring(s):

char\_index = {}

max\_length = 0

start = 0

for end, char in enumerate(s):

if char in char\_index and char\_index[char] >= start:

start = char\_index[char] + 1

char\_index[char] = end

max\_length = max(max\_length, end - start + 1)

return max\_length

# Test cases

print(lengthOfLongestSubstring("abcabcbb")) # Output: 3

print(lengthOfLongestSubstring("bbbbb")) # Output: 1

print(lengthOfLongestSubstring("pwwkew")) # Output: 3

print(lengthOfLongestSubstring("")) # Output: 0

print(lengthOfLongestSubstring("dvdf")) # Output: 3

4. **Median of Two Sorted Arrays Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).**

def findMedianSortedArrays(nums1, nums2):

# Ensure nums1 is the smaller array

if len(nums1) > len(nums2):

nums1, nums2 = nums2, nums1

m, n = len(nums1), len(nums2)

imin, imax, half\_len = 0, m, (m + n + 1) // 2

while imin <= imax:

i = (imin + imax) // 2

j = half\_len - i

if i < m and nums1[i] < nums2[j - 1]:

imin = i + 1

elif i > 0 and nums1[i - 1] > nums2[j]:

imax = i - 1

else:

if i == 0:

max\_of\_left = nums2[j - 1]

elif j == 0:

max\_of\_left = nums1[i - 1]

else:

max\_of\_left = max(nums1[i - 1], nums2[j - 1])

if (m + n) % 2 == 1:

return max\_of\_left

if i == m:

min\_of\_right = nums2[j]

elif j == n:

min\_of\_right = nums1[i]

else:

min\_of\_right = min(nums1[i], nums2[j])

return (max\_of\_left + min\_of\_right) / 2.0

# Test cases

print(findMedianSortedArrays([1, 3], [2])) # Output: 2.0

print(findMedianSortedArrays([1, 2], [3, 4])) # Output: 2.5

print(findMedianSortedArrays([0, 0], [0, 0])) # Output: 0.0

print(findMedianSortedArrays([], [1])) # Output: 1.0

print(findMedianSortedArrays([2], [])) # Output: 2.0

5. **Longest Palindromic Substring Given a string s, return the longest palindromic substring in s.**

def longestPalindrome(s: str) -> str:

if not s:

return ""

start, end = 0, 0

def expandAroundCenter(s, left, right):

while left >= 0 and right < len(s) and s[left] == s[right]:

left -= 1

right += 1

return left + 1, right - 1

for i in range(len(s)):

# Check for odd-length palindromes

l1, r1 = expandAroundCenter(s, i, i)

# Check for even-length palindromes

l2, r2 = expandAroundCenter(s, i, i + 1)

if r1 - l1 > end - start:

start, end = l1, r1

if r2 - l2 > end - start:

start, end = l2, r2

return s[start:end + 1]

# Test cases

print(longestPalindrome("babad")) # Output: "bab" or "aba"

print(longestPalindrome("cbbd")) # Output: "bb"

print(longestPalindrome("a")) # Output: "a"

print(longestPalindrome("ac")) # Output: "a" or "c"

**6. Zigzag Conversion**

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

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

return s

# Create an array to hold the strings for each row

rows = [''] \* numRows

current\_row = 0

direction = -1 # Used to change direction from down to up and vice versa

for char in s:

rows[current\_row] += char

# Change direction if we are at the top or bottom row

if current\_row == 0 or current\_row == numRows - 1:

direction \*= -1

current\_row += direction

# Combine all rows to get the final string

return ''.join(rows)

# Test cases

print(convert("PAYPALISHIRING", 3)) # Output: "PAHNAPLSIIGYIR"

print(convert("PAYPALISHIRING", 4)) # Output: "PINALSIGYAHRPI"

print(convert("A", 1)) # Output: "A"

**7. Reverse Integer**

def reverse(x: int) -> int:

INT\_MAX = 2\*\*31 - 1

INT\_MIN = -2\*\*31

sign = -1 if x < 0 else 1

x = abs(x)

reversed\_x = 0

while x != 0:

digit = x % 10

x = x // 10

if reversed\_x > (INT\_MAX - digit) // 10:

return 0

reversed\_x = reversed\_x \* 10 + digit

return sign \* reversed\_x

# Test cases

print(reverse(123)) # Output: 321

print(reverse(-123)) # Output: -321

print(reverse(120)) # Output: 21

print(reverse(0)) # Output: 0

print(reverse(1534236469)) # Output: 0

**8. String to Integer (atoi)**

def myAtoi(s: str) -> int:

INT\_MAX = 2\*\*31 - 1

INT\_MIN = -2\*\*31

i, n = 0, len(s)

# Skip leading whitespaces

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

i += 1

if i == n:

return 0

# Check for sign

sign = 1

if s[i] == '-':

sign = -1

i += 1

elif s[i] == '+':

i += 1

# Read digits and convert to integer

result = 0

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

digit = int(s[i])

# Check for overflow and clamp to INT\_MAX or INT\_MIN

if result > (INT\_MAX - digit) // 10:

return INT\_MAX if sign == 1 else INT\_MIN

result = result \* 10 + digit

i += 1

return sign \* result

# Test cases

print(myAtoi("42")) # Output: 42

print(myAtoi(" -42")) # Output: -42

print(myAtoi("4193 with words"))# Output: 4193

print(myAtoi("words and 987")) # Output: 0

print(myAtoi("-91283472332")) # Output: -2147483648

**9. Palindrome Number**

def isPalindrome(x: int) -> bool:

# Negative numbers are not palindromes

if x < 0:

return False

# Convert the number to a string

s = str(x)

# Check if the string is equal to its reverse

return s == s[::-1]

# Test cases

print(isPalindrome(121)) # Output: True

print(isPalindrome(-121)) # Output: False

print(isPalindrome(10)) # Output: False

**10. Regular Expression Matching**

def isMatch(s: str, p: str) -> bool:

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

# dp[i][j] will be True if s[0:i] matches p[0:j]

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

dp[0][0] = True # Empty string matches empty pattern

# Initialize the table for patterns with '\*' that can match an empty string

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

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

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

# Fill the DP table

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

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

if p[j - 1] == '.' or p[j - 1] == s[i - 1]:

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

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

# Check zero occurrence of the element before '\*'

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

# Check one or more occurrences if preceding element matches current character in s

if p[j - 2] == '.' or p[j - 2] == s[i - 1]:

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

return dp[m][n]

# Test cases

print(isMatch("aa", "a")) # Output: False

print(isMatch("aa", "a\*")) # Output: True

print(isMatch("ab", ".\*")) # Output: True

print(isMatch("aab", "c\*a\*b")) # Output: True

print(isMatch("mississippi", "mis\*is\*p\*.")) # Output: False