Q1

class Solution:

def minimumDeleteSum(self, s1: str, s2: str) -> int:

dp = collections.defaultdict(int)

for i in range(len(s1)):

for j in range(len(s2)):

if s1[i] == s2[j]:

dp[i, j] = dp[i - 1, j - 1] + ord(s1[i])

else:

dp[i, j] = max(dp[i - 1, j], dp[i, j - 1])

total\_ascii\_sum = sum(map(ord, s1)) + sum(map(ord, s2))

return total\_ascii\_sum - 2 \* dp[i, j]

Q2

class Solution:

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

leftmin = leftmax = 0

for c in s:

if c == "(":

leftmax += 1

leftmin += 1

if c == ")":

leftmax -= 1

leftmin = max(0, leftmin-1)

if c == "\*":

leftmax +=1

leftmin = max(0, leftmin-1)

if leftmax < 0:

return False

if leftmin == 0:

return True

Q3

class Solution:

def minDistance(self, word1: str, word2: str) -> int:

@lru\_cache(None)

def calc(i, j):

if i == len(word1):

return len(word2) - j

elif j == len(word2):

return len(word1) - i

if word1[i] == word2[j]:

# We move on without deleting anything

return calc(i+1, j+1)

# We delete from word1 or word2

return 1 + min(calc(i+1, j), calc(i, j+1))

return calc(0, 0)

Q4

class Solution:

def tree2str(self, root: Optional[TreeNode]) -> str:

def dfs(node):

if node is None:

return ''

left = dfs(node.left)

right = dfs(node.right)

s = f'{node.val}'

if left:

s += f'({left})'

if right:

if not left:

s += '()'

s += f'({right})'

return s

return dfs(root)

Q5

class Solution:

def compress(self, chars: List[str]) -> int:

n=len(chars)

left,right,k=0,0,0

while left<n:

left=right

while right<n and chars[right]==chars[left]:

right+=1

chars[k]=chars[left]

k+=1

if (right-left)>1:

for l in str(right-left):

chars[k]=l

k+=1

left=right

return k

Q6

class Solution(object):

def findAnagrams(self, s, p):

n = len(s)

m = len(p)

p = Counter(p)

ans = []

window = None

for i in range(0,n-m+1):

if i == 0:

window = Counter(s[:m])

else:

window[s[i-1]] -= 1

window[s[i+m-1]] += 1

if len(window - p) == 0:

ans.append(i)

return ans

Q7

class Solution(object):

def decodeString(self, s):

stack = []; curNum = 0; curString = ''

for c in s:

if c == '[':

stack.append(curString)

stack.append(curNum)

curString = ''

curNum = 0

elif c == ']':

num = stack.pop()

prevString = stack.pop()

curString = prevString + num\*curString

elif c.isdigit():

curNum = curNum\*10 + int(c)

else:

curString += c

return curString

Q8

class Solution:

def buddyStrings(self, string: str, goal: str) -> bool:

left, right = 0, len(string) - 1

if len(string) != len(goal):

return False

if string == goal and len(set(string)) < len(string):

return True

difference = []

for i in range(len(string)):

if string[i] != goal[i]:

difference.append((string[i], goal[i]))

if len(difference) == 2 and difference[0] == difference[-1][::-1]:

return True

return False