1.Given an m x n binary matrix mat, return the distance of the nearest 0 for each cell. The distance between two adjacent cells is 1.

Program:

from collections import deque

def updateMatrix(mat):

m, n = len(mat), len(mat[0])

dist = [[float('inf')] \* n for \_ in range(m)]

queue = deque()

# Enqueue all cells with 0s and set their distance to 0

for i in range(m):

for j in range(n):

if mat[i][j] == 0:

dist[i][j] = 0

queue.append((i, j))

# Directions array for moving up, down, left, right

directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

# Perform BFS

while queue:

x, y = queue.popleft()

for dx, dy in directions:

new\_x, new\_y = x + dx, y + dy

if 0 <= new\_x < m and 0 <= new\_y < n and dist[new\_x][new\_y] > dist[x][y] + 1:

dist[new\_x][new\_y] = dist[x][y] + 1

queue.append((new\_x, new\_y))

return dist

# Example usage:

mat = [

[0, 0, 0],

[0, 1, 0],

[1, 1, 1]

]

print(updateMatrix(mat))

2. Given two integer arrays arr1 and arr2, return the minimum number of operations (possibly zero) needed to make arr1 strictly increasing. In one operation, you can choose two indices 0 <= i < arr1.length and 0 <= j < arr2.length and do the assignment arr1[i] = arr2[j]. If there is no way to make arr1 strictly increasing, return -1.

Program:

from bisect import bisect\_right

from collections import defaultdict

def makeArrayIncreasing(arr1, arr2):

arr2 = sorted(set(arr2))

dp = {-1: 0} # Using -1 as a placeholder for the initial state

for num in arr1:

new\_dp = defaultdict(lambda: float('inf'))

for key in dp:

if num > key:

new\_dp[num] = min(new\_dp[num], dp[key])

idx = bisect\_right(arr2, key)

if idx < len(arr2):

new\_dp[arr2[idx]] = min(new\_dp[arr2[idx]], dp[key] + 1)

dp = new\_dp

if dp:

return min(dp.values())

return -1

# Example usage:

arr1 = [1, 5, 3, 6, 7]

arr2 = [1, 3, 2, 4]

print(makeArrayIncreasing(arr1, arr2)) # Output: 1

3. Given two strings a and b, return the minimum number of times you should repeat string a so that string b is a substring of it. If it is impossible for b​ to be a substring of a after repeating it, return -1. Notice: string "abc" repeated 0 times is "", repeated 1 time is "abc" and repeated 2 times is "abcabc"

Program:

def repeatedStringMatch(a, b):

# Calculate the minimum number of repeats needed

min\_repeats = -(-len(b) // len(a)) # Equivalent to math.ceil(len(b) / len(a))

# Check if b is a substring of a repeated min\_repeats or min\_repeats + 1 times

for i in range(2):

if b in (a \* (min\_repeats + i)):

return min\_repeats + i

# If b is not found, return -1

return -1

# Example usage:

a = "abcd"

b = "cdabcdab"

print(repeatedStringMatch(a, b)) # Output: 3

4. Given an array nums containing n distinct numbers in the range [0, n], return the only number in the range that is missing from the array

Program:

def missingNumber(nums):

n = len(nums)

xor\_all = 0

xor\_nums = 0

for i in range(n + 1):

xor\_all ^= i

for num in nums:

xor\_nums ^= num

return xor\_all ^ xor\_nums

# Example usage:

nums = [3, 0, 1]

print(missingNumber(nums)) # Output: 2

5. You are given an n x n integer matrix grid.Generate an integer matrix maxLocal of size (n - 2) x (n - 2) such that: maxLocal[i][j] is equal to the largest value of the 3 x 3 matrix in grid centered around row i + 1 and column j + 1. In other words, we want to find the largest value in every contiguous 3 x 3 matrix in grid. Return the generated matrix.

Program:

def largestLocal(grid):

n = len(grid)

maxLocal = [[0] \* (n - 2) for \_ in range(n - 2)]

for i in range(n - 2):

for j in range(n - 2):

max\_val = float('-inf')

for k in range(i, i + 3):

for l in range(j, j + 3):

max\_val = max(max\_val, grid[k][l])

maxLocal[i][j] = max\_val

return maxLocal

# Example usage:

grid = [

[9, 9, 8, 1],

[5, 6, 2, 6],

[8, 2, 6, 4],

[6, 2, 2, 2]

]

print(largestLocal(grid))