31. 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.

Input: mat = [[0,0,0],[0,1,0],[0,0,0]] ; Output: [[0,0,0],[0,1,0],[0,0,0]]

Input: mat = [[0,0,0],[0,1,0],[1,1,1]] ;Output: [[0,0,0],[0,1,0],[1,2,1]]

Program:

```
from collections import deque
def updateMatrix(mat):
    rows, cols = len(mat), len(mat[0])
    dist = [[float('inf')] * cols for _ in range(rows)]
    queue = deque()
    for r in range(rows):
        for c in range(cols):
            if mat[r][c] == 0:
                dist[r][c] = 0
                queue.append((r, c))
    directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]
    while queue:
        r, c = queue.popleft()
        for dr, dc in directions:
            nr, nc = r + dr, c + dc
            if 0 <= nr < rows and 0 <= nc < cols:
                if dist[nr][nc] > dist[r][c] + 1:
                    dist[nr][nc] = dist[r][c] + 1
                    queue.append((nr, nc))
    return dist
mat1 = [[0,0,0],[0,1,0],[0,0,0]]
```

```
mat2 = [[0,0,0],[0,1,0],[1,1,1]]

print(updateMatrix(mat1))

# Output: [[0,0,0],[0,1,0],[0,0,0]]

print(updateMatrix(mat2))

# Output: [[0,0,0],[0,1,0],[1,2,1]]
```

32. 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.

Example 1: Input: arr1 = [1,5,3,6,7], arr2 = [1,3,2,4] Output: 1

Explanation: Replace 5 with 2, then arr1 = [1, 2, 3, 6, 7]

Program:

```
def min_operations(arr1, arr2):
    n = len(arr1)
    arr2.sort()
    dp = {-1: 0}
    for i in range(n):
        new_dp = {}
        for key in dp:
            if arr1[i] > key:
                new_dp[arr1[i]] = min(new_dp.get(arr1[i], float('inf')), dp[key])
            while arr2 and arr2[0] <= key:
                arr2.pop(0)
            if arr2:
                new_dp[arr2[0]] = min(new_dp.get(arr2[0], float('inf')), dp[key] + 1)
        dp = new_dp
```

```
    if dp:
        return min(dp.values())
    return -1
arr1 = [1, 5, 3, 6, 7]
arr2 = [1, 3, 2, 4]
print(min_operations(arr1, arr2))
# Output: 1
```

33. 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". Example 1: Input: a = "abcd", b = "cdabcdab" ; Output: 3

Explanation: We return 3 because by repeating a three times "abcdabcdabcd", b is a substring of it

Program:

```
def min_repeats_v2(a, b):
    if b in a:
        return 1
    for i in range(1, len(b) + 1):
        if b == a[:i] * (len(b) // i) + a[:len(b) % i]:
            return len(b) // i + (len(b) % i != 0)
    return -1
a = "abcd"
b = "cdabcdab"
result = min_repeats_v2(a, b)
print(result)
# Output: 3
```

34. 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.

Example 1: Input: nums = [3,0,1] ; Output: 2

Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

Program:

```
def missing_number(nums):
    n = len(nums)
    total_sum = n * (n + 1) // 2
    array_sum = sum(nums)
    return total_sum - array_sum
nums = [3, 0, 1]
print(missing_number(nums))
# Output: 2
```

35. 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.

Input: grid = [[9,9,8,1],[5,6,2,6],[8,2,6,4],[6,2,2,2]] Output: [[9,9],[8,6]]

Explanation: The diagram above shows the original matrix and the generated matrix. Notice that each value in the generated matrix corresponds to the largest value of a contiguous 3 x 3 matrix in grid.

Program:

```
def generate_max_local(grid):
    n = len(grid)
    max_local = [[max(grid[i-1][j-1], grid[i-1][j], grid[i-1][j+1],
                  grid[i][j-1], grid[i][j], grid[i][j+1],
```

```
                grid[i+1][j-1], grid[i+1][j], grid[i+1][j+1])
            for j in range(1, n-1)]
        for i in range(1, n-1)]
    return max_local
grid = [[9, 9, 8, 1], [5, 6, 2, 6], [8, 2, 6, 4], [6, 2, 2, 2]]
result = generate_max_local(grid)
print(result)
```