1Given the strings s1 and s2 of size n and the string evil, return the number of good strings. A good string has size n, it is alphabetically greater than or equal to s1, it is alphabetically smaller than or equal to s2, and it does not contain the string evil as a substring. Since the answer can be a huge number, return this modulo 109 + 7.

Program:

```
MOD = 10**9 + 7


def good_strings_count(n, s1, s2, evil):
    def is_valid_string(s, s1, s2):
        if not (s1 <= s <= s2):
            return False
        if evil in s:
            return False
        return True

    def generate_strings(curr, s1, s2):
        if len(curr) == n:
            if is_valid_string(curr, s1, s2):
                return 1
            else:
                return 0
        count = 0
        start = 'a' if len(curr) == 0 else curr[-1]
        end = 'z'
        for char in range(ord(start), ord(end) + 1):
            count += generate_strings(curr + chr(char), s1, s2)
            count %= MOD
        return count

    return generate_strings('', s1, s2)

# Example usage:
```

```
n = 4

s1 = "abcd"

s2 = "bcde"

evil = "abc"

print(good_strings_count(n, s1, s2, evil))  # Output the number of good strings
```

2. Given a 2D integer array matrix, return the transpose of matrix. The transpose of a matrix is the matrix flipped over its main diagonal, switching the matrix's row and column indices.

Program:

```
def transpose(matrix):
    m = len(matrix)
    n = len(matrix[0]) if m > 0 else 0
    transpose_matrix = [[0] * m for _ in range(n)]

    # Fill the transpose matrix
    for i in range(m):
        for j in range(n):
            transpose_matrix[j][i] = matrix[i][j]
    return transpose_matrix
```

3. Given an integer n, return the nth digit of the infinite integer sequence [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...].

Program:

```
def find_nth_digit(n):
    if n <= 9:
        return n

    length_of_numbers = 9
    digits = 1
    start = 1

    # Calculate which range of numbers n falls into
    while n > length_of_numbers * digits:
        n -= length_of_numbers * digits
```

```
        length_of_numbers *= 10
        digits += 1
        start *= 10

    # Now n is within the range of numbers with 'digits' digits
    num = start + (n - 1) // digits
    digit_index = (n - 1) % digits

    # Convert number to string to get the digit
    return int(str(num)[digit_index])

# Example usage:
n = 11
print(find_nth_digit(n))  # Output: 0 (the 11th digit in the sequence is from number 10)
```

4. Given a sentence that consists of some words separated by a single space, and a searchWord, check if searchWord is a prefix of any word in sentence. Return the index of the word in sentence (1-indexed) where searchWord is a prefix of this word. If searchWord is a prefix of more than one word, return the index of the first word (minimum index). If there is no such word return - 1. A prefix of a string s is any leading contiguous substring of s.

Program:

```
def isPrefixOfWord(sentence, searchWord):
    words = sentence.split()

    for index, word in enumerate(words, 1):  # Enumerate starts from index 1
        if word.startswith(searchWord):
            return index

    return -1

# Example usage:
sentence = "i love eating burger"
searchWord = "burg"
```

print(isPrefixOfWord(sentence, searchWord)) # Output: 4 (index of the word "burger")

5. Given an integer array num sorted in non-decreasing order. You can perform the following operation any number of times: Choose two indices, i and j, where nums[i] < nums[j]. Then, remove the elements at indices i and j from nums. The remaining elements retain their original order, and the array is reindexed. Return the minimum length of nums after applying the operation zero or more times.

Program:

```
def min_length_after_operations(nums):
    n = len(nums)
    count_pairs = 0

    for i in range(n - 1):
        if nums[i] < nums[i + 1]:
            count_pairs += 1

    # Each valid pair reduces the length of nums by 2
    return n - 2 * count_pairs

# Example usage:
nums = [1, 2, 3, 4, 5]
print(min_length_after_operations(nums))  # Output: 1 (after removing (1, 2), (3, 4), (4, 5))
```

6. Given an array of string words, return all strings in words that is a substring of another word. You can return the answer in any order. A substring is a contiguous sequence of characters within a string

Program:

```
def substring_words(words):
    result = []

    for i in range(len(words)):
        for j in range(len(words)):
            if i != j and words[i] in words[j]:
                result.append(words[i])
                break
```

```
    return result


# Example usage:
words = ["leetcoder","leetcode","od","hamlet","am"]
print(substring_words(words))  # Output: ['leetcode', 'od', 'am']
```

7. 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 nearest_zero(mat):
    m, n = len(mat), len(mat[0])
    distances = [[float('inf')] * n for _ in range(m)]
    queue = deque()

    # Initialize queue with all cells containing 0 and set their distance to 0
    for i in range(m):
        for j in range(n):
            if mat[i][j] == 0:
                distances[i][j] = 0
                queue.append((i, j))

    # Directions for moving up, down, left, right
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    # Perform BFS
    while queue:
        i, j = queue.popleft()

        # Explore neighbors
        for di, dj in directions:
```

```
            ni, nj = i + di, j + dj
            if 0 <= ni < m and 0 <= nj < n and distances[ni][nj] == float('inf'):
                distances[ni][nj] = distances[i][j] + 1
                queue.append((ni, nj))

    return distances

# Example usage:
mat = [
    [0, 0, 0],
    [0, 1, 0],
    [1, 1, 1]
]
print(nearest_zero(mat))  # Output: [[0, 0, 0], [0, 1, 0], [1, 2, 1]]
```

8. 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}  # Base case: -1 means the start of the sequence

    for num in arr1:
        temp = defaultdict(lambda: float('inf'))
        for key in dp:
            if num > key:
                temp[num] = min(temp[num], dp[key])
            idx = bisect_right(arr2, key)
```

```
            if idx < len(arr2):
                temp[arr2[idx]] = min(temp[arr2[idx]], dp[key] + 1)
        dp = temp

    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 should be
```

9. 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):
    if len(b) == 0:
        return 0
    if len(a) == 0:
        return -1

    len_a = len(a)
    len_b = len(b)

    # Calculate the minimum length of repeated a we need to contain b
    min_length = (len_b // len_a) * len_a + len_b % len_a

    # Repeat a until its length is at least min_length
    repeated_a = ""
    k = 0
```

```
    while len(repeated_a) < min_length:
        repeated_a += a
        k += 1

        # Check if b is a substring of repeated_a
        if b in repeated_a:
            return k

    # If after sufficient repetitions b is still not a substring, return -1
    if b in repeated_a:
        return k
    else:
        return -1
```

10. 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):
    missing = len(nums)
    for i, num in enumerate(nums):
        missing ^= i ^ num
    return missing
```