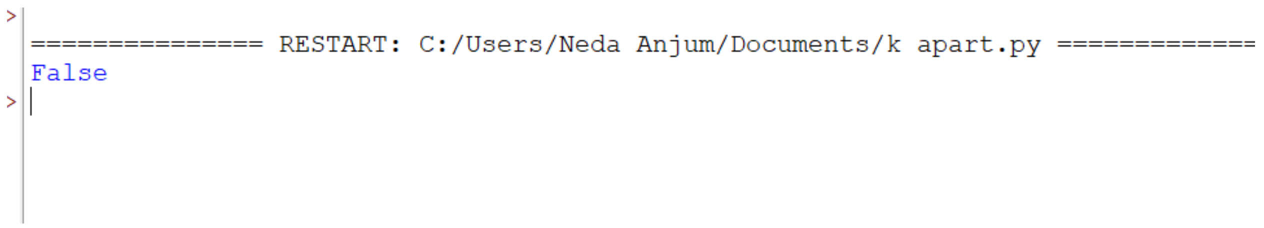
Lab(8/6/24)

1. Given an binary array num and an integer k return true if all 1's are at last k places away from each other otherwise return false

Code:

```
def k_length(nums, k):
    prev = -k - 1
    for i, num in enumerate(nums):
        if num == 1:
            if i - prev <= k:
                return False
            prev = i
    return True
nums = [1, 0, 0, 1, 0, 1]
k = 2
print(k_length(nums, k))
```

output:



```
>
=============== RESTART: C:/Users/Neda Anjum/Documents/k apart.py =============
False
>
```

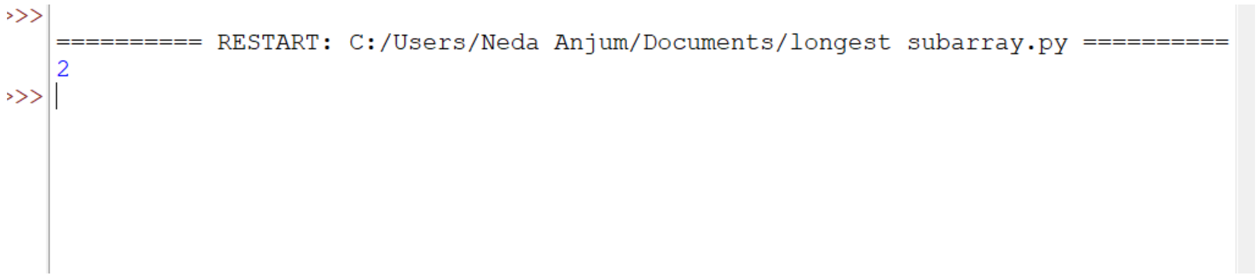
2. Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit

Given an array of integers nums and an integer limit, return the size of the longest non-empty subarray such that the absolute difference between any two elements of this subarray is less than or equal to limit.

Code:

```
from collections import deque
def longest_subarray(nums, limit):
    max_deque = deque()
    min_deque = deque()
    left = 0
    result = 0

    for right in range(len(nums)):
        while (max_deque and nums[right] > max_deque[-1]):
            max_deque.pop()
        while( min_deque and nums[right] < min_deque[-1]):
            min_deque.pop()

        max_deque.append(nums[right])
        min_deque.append(nums[right])
         while (max_deque[0] - min_deque[0] > limit):
            if( max_deque[0] == nums[left]):
                max_deque.popleft()
            if min_deque[0] == nums[left]:
                min_deque.popleft()
            left += 1
         result = max(result, right - left + 1)

    return result
nums = [8, 2, 4, 7]
limit = 4
print(longest_subarray(nums, limit))
```

output:



```
>>>
========== RESTART: C:/Users/Neda Anjum/Documents/longest subarray.py ==========
2
>>> |
```

3.Find the Kth Smallest Sum of a Matrix With Sorted Rows

You are given an m x n matrix mat that has its rows sorted in non-decreasing order and an integer k.

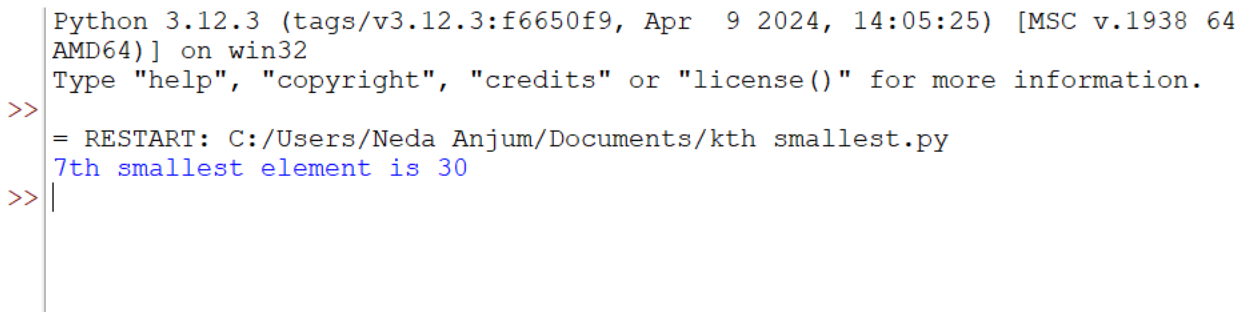
You are allowed to choose exactly one element from each row to form an array.

Return the kth smallest array sum among all possible arrays.

Code:

```
 import heapq


def kthSmallest(arr, n, k):
    pq = []
    for i in range(n):
        for j in range(n):
            heapq.heappush(pq, arr[i][j])
    c = 0
    while pq:
        temp = heapq.heappop(pq)
        c += 1
        if c == k:
            return temp
    return -1
if __name__ == "__main__":
    mat = [
        [10, 20, 30, 40],
        [15, 25, 35, 45],
        [25, 29, 37, 48],
        [32, 33, 39, 50]
    ]
    res = kthSmallest(mat, 4, 7)
    print("7th smallest element is", res)
```

output:



```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>
= RESTART: C:/Users/Neda Anjum/Documents/kth smallest.py
7th smallest element is 30
>>
```

4. Count Triplets That Can Form Two Arrays of Equal XOR

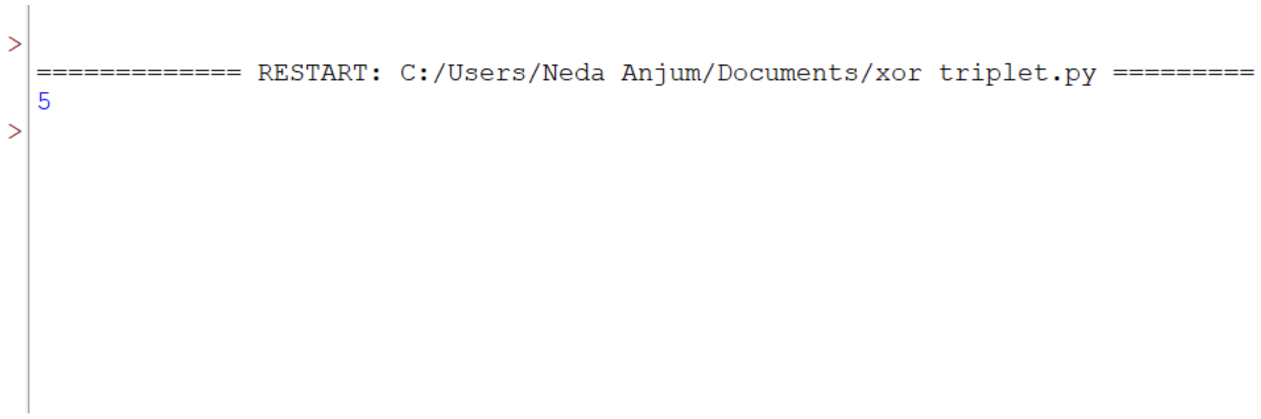
Given an array of integers arr.

We want to select three indices i, j and k where (0 <= i < j <= k < arr.length).

Code:

```
def xor_triplet(arr, n):


    ans = 0

    for i in range(n):

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

            for k in range(j, n):


                xor1 = 0

                xor2 = 0

                for x in range(i, j):

                    xor1 ^= arr[x]

                for x in range(j, k + 1):

                    xor2 ^= arr[x]

                if (xor1 == xor2):

                    ans += 1



    return ans

if __name__ == '__main__':

    arr = [1, 2, 3, 4, 5]
    n = len(arr)

    print(xor_triplet(arr, n))
```

output:



```
>
============= RESTART: C:/Users/Neda Anjum/Documents/xor triplet.py =========
5
>
```

5. Minimum Time to Collect All Apples in a Tree

Given an undirected tree consisting of n vertices numbered from 0 to n-1, which has some apples in their vertices. You spend 1 second to walk over one edge of the tree. Return the minimum time in seconds you have to spend to collect all apples in the tree, starting at vertex 0 and coming back to this vertex.

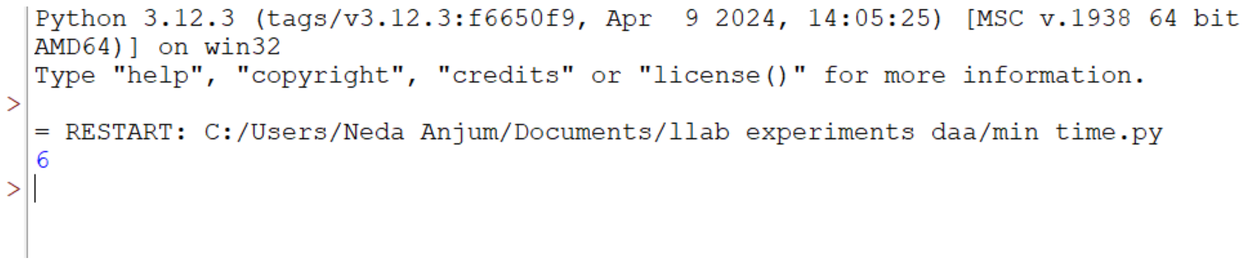
The edges of the undirected tree are given in the array edges, where edges[i] = [ai, bi] means that exists an edge connecting the vertices ai and bi. Additionally, there is a boolean array hasApple, where hasApple[i] = true means that vertex i has an apple; otherwise, it does not have any apple.

Code:

```
def minTime(n, edges, hasApple):

  tree = {}

  for u, v in edges:

    if u not in tree:

      tree[u] = []

    if v not in tree:

      tree[v] = []

    tree[u].append(v)

    tree[v].append(u)

  def dfs(node, parent):

    total_time = 0
```

```
        for child in tree[node]:
            if child != parent:
                child_time = dfs(child, node)
                if child_time > 0 or hasApple[child]:
                    total_time += child_time + 2
        return total_time

    return max(0, dfs(0, -1) - 2)
n = 7
edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]]
hasApple = [False,False,True,False,True,True,False]
print(minTime(n, edges, hasApple))
```

output:



```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>
= RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/min time.py
6
>
```

6. Number of Ways of Cutting a Pizza

Given a rectangular pizza represented as a rows x cols matrix containing the following characters: 'A' (an apple) and '.' (empty cell) and given the integer k. You have to cut the pizza into k pieces using k-1 cuts.

For each cut you choose the direction: vertical or horizontal, then you choose a cut position at the cell boundary and cut the pizza into two pieces. If you cut the pizza vertically, give the left part of the pizza to a person. If you cut the pizza horizontally, give the upper part of the pizza to a person. Give the last piece of pizza to the last person.

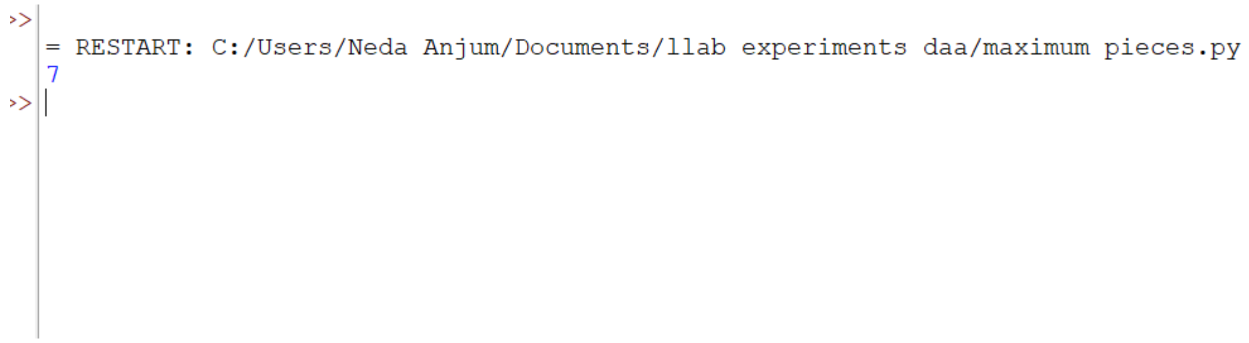
Return the number of ways of cutting the pizza such that each piece contains at least one apple. Since the answer can be a huge number, return this modulo 10^9 + 7.

Code:

```
def findMaximumPieces(n):
        return int(1 + n * (n + 1) / 2)
```

print(findMaximumPieces(3))

output:



```
>>
= RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/maximum pieces.py
7
>>
```