

LAB-5:

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_apart(nums, k):
    pos = 0
    count = 0
    while pos < len(nums) and nums[pos] == 0:
        pos += 1
    for i in range(pos + 1, len(nums)):
        if nums[i] == 0:
            count += 1
        else:
            if count < k:
                return False
            count = 0
    return True

def main():
    nums = [1, 0, 0, 0, 1, 0, 0, 1, 0, 0]
    k = 2
    result = k_length_apart(nums, k)
    if result:
        print(f"true")
    else:
        print(f"false")

if __name__ == "__main__":
    main()
```

OUTPUT:

```
IDLE Shell 3.12.3
File Edit Shell Debug Options Window Help
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/bored/AppData/Local/Programs/Python/Python312/binary array.p
y
true
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/binary array.p
y
true
>>>
```

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:

```
def longestSubarray(nums, limit):
```

```
    min_queue = []
```

```
    max_queue = []
```

```
    left = 0
```

```
    right = 0
```

```
    max_length = 0
```

```
    while right < len(nums):
```

```
        while min_queue and nums[right] <= nums[min_queue[-1]]:
```

```
            min_queue.pop()
```

```
        min_queue.append(right)
```

```
        while max_queue and nums[right] >= nums[max_queue[-1]]:
```

```
            max_queue.pop()
```

```
        max_queue.append(right)
```

```
        while nums[max_queue[0]] - nums[min_queue[0]] > limit:
```

```

        left += 1

    if left > min_queue[0]:
        min_queue.pop(0)

    if left > max_queue[0]:
        max_queue.pop(0)

    max_length = max(max_length, right - left + 1)

    right += 1

return max_length

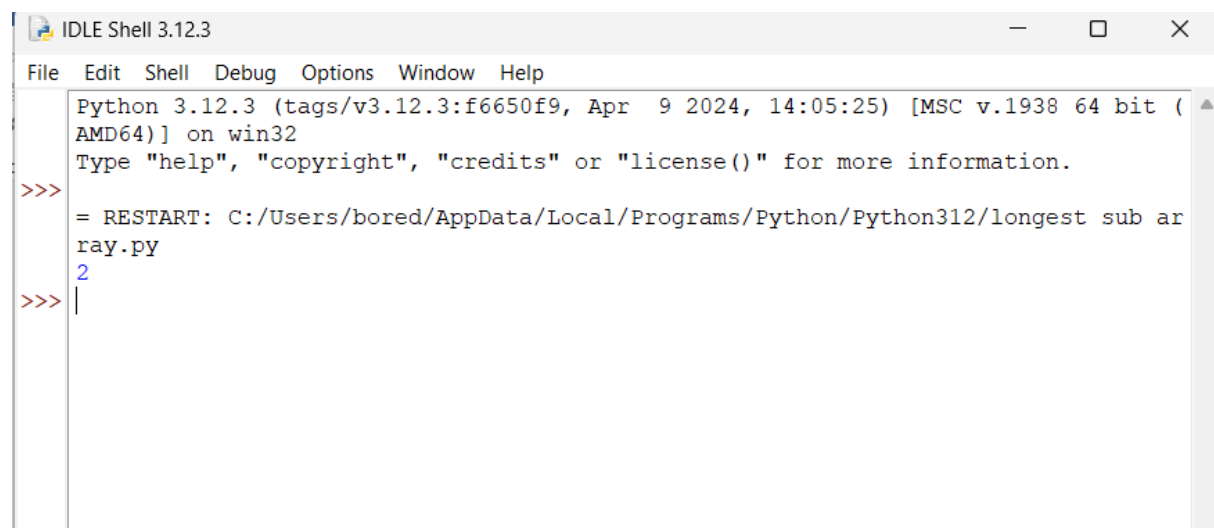
nums = [8, 2, 4, 7]

limit = 4

print(longestSubarray(nums, limit))

```

OUTPUT:



```

IDLE Shell 3.12.3
File Edit Shell Debug Options Window Help
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/bored/AppData/Local/Programs/Python/Python312/longest sub array.py
2
>>> |

```

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

You are given an $m \times n$ 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 `k`th smallest array sum among all possible arrays.

CODE:

```

import heapq

def kthSmallest(mat, k):
    m, n = len(mat), len(mat[0])

    heap = [(sum(row[0] for row in mat), [0] * m)]

```

```

for _ in range(k):
    current_sum, indices = heapq.heappop(heap)

    for i in range(m):
        if indices[i] < n - 1:
            next_indices = indices[:]
            next_indices[i] += 1

            next_sum = current_sum - mat[i][indices[i]] + mat[i][next_indices[i]]

            heapq.heappush(heap, (next_sum, next_indices))

    return current_sum

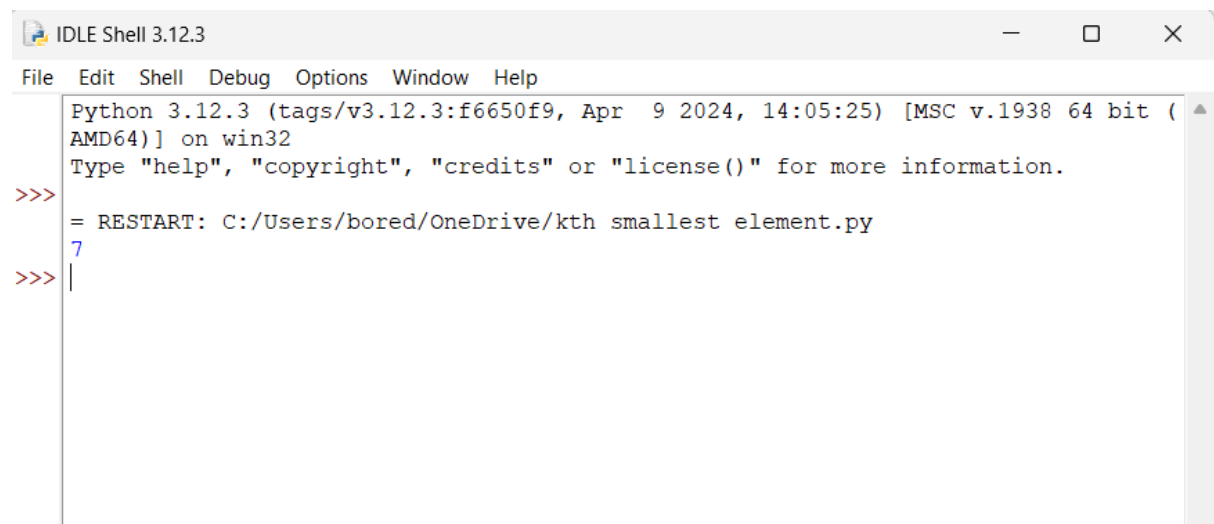
mat = [[1,3,11],[2,4,6]]

k = 5

print(kthSmallest(mat, k))

```

OUTPUT:



```

IDLE Shell 3.12.3
File Edit Shell Debug Options Window Help
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/bored/OneDrive/kth smallest element.py
>>> 7
>>> |

```

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 $(0 \leq i < j \leq k < \text{arr.length})$.

Let's define a and b as follows:

- $a = \text{arr}[i] \oplus \text{arr}[i + 1] \oplus \dots \oplus \text{arr}[j - 1]$
- $b = \text{arr}[j] \oplus \text{arr}[j + 1] \oplus \dots \oplus \text{arr}[k]$

Note that \oplus denotes the bitwise-xor operation.

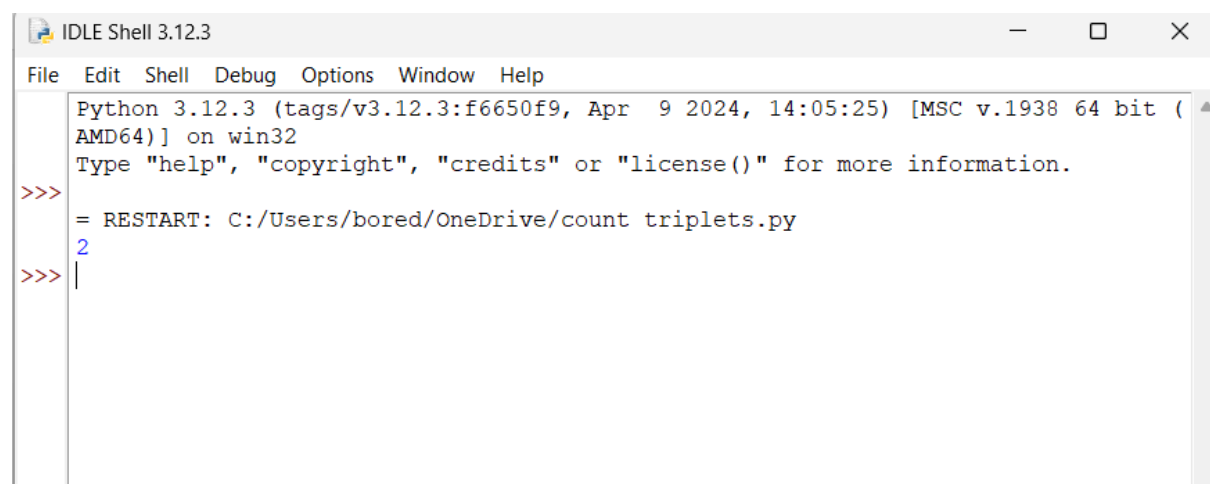
Return the number of triplets (i, j and k) Where $a == b$.

CODE:

```
def countTriplets(arr):
    n = len(arr)
    xor_prefix = [0] * (n + 1)
    xor_freq = {0: 1}
    count = 0
    for i in range(1, n + 1):
        xor_prefix[i] = xor_prefix[i - 1] ^ arr[i - 1]
    for j in range(1, n):
        for k in range(j, n):
            a = xor_prefix[j] ^ xor_prefix[0]
            b = xor_prefix[k + 1] ^ xor_prefix[j]
            if a == b:
                count += 1
    return count

arr = [2, 3, 1, 6, 7]
print(countTriplets(arr))
```

OUTPUT:



```
IDLE Shell 3.12.3
File Edit Shell Debug Options Window Help
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/bored/OneDrive/count triplets.py
2
>>> |
```

5. Minimum Time to Collect All Apples in a Tree

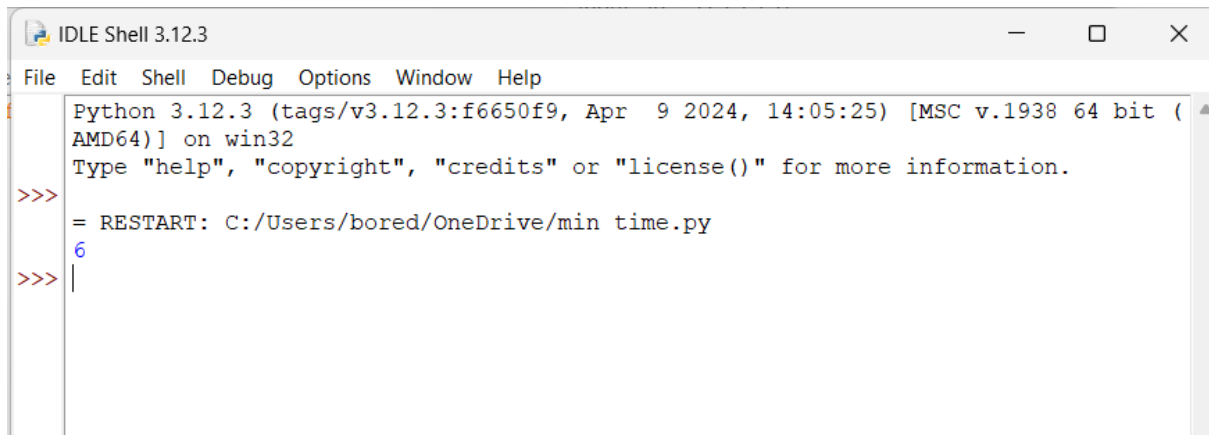
Given an undirected tree consisting of n vertices numbered from 0 to $n-1$, which has some apples in their vertices. You spend 1 second to walk over one edge of the tree. Return the minimum time in seconds you have to spend to collect all apples in the tree, starting at vertex 0 and coming back to this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



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:

MOD = $10^{**}9 + 7$

def ways(pizza, k):

 rows, cols = len(pizza), len(pizza[0])

 prefix_sum = [[0] * (cols + 1) for _ in range(rows + 1)]

 for i in range(rows - 1, -1, -1):

 for j in range(cols - 1, -1, -1):

 prefix_sum[i][j] = prefix_sum[i + 1][j] + prefix_sum[i][j + 1] - prefix_sum[i + 1][j + 1] + (pizza[i][j] == 'A')

 def dp(r, c, cuts):

 if cuts == 0:

 return int(prefix_sum[r][c] > 0)

 ways = 0

 for i in range(r + 1, rows):

```

    if prefix_sum[r][c] - prefix_sum[i][c] > 0:
        ways = (ways + dp(i, c, cuts - 1)) % MOD
for j in range(c + 1, cols):
    if prefix_sum[r][c] - prefix_sum[r][j] > 0:
        ways = (ways + dp(r, j, cuts - 1)) % MOD
return ways

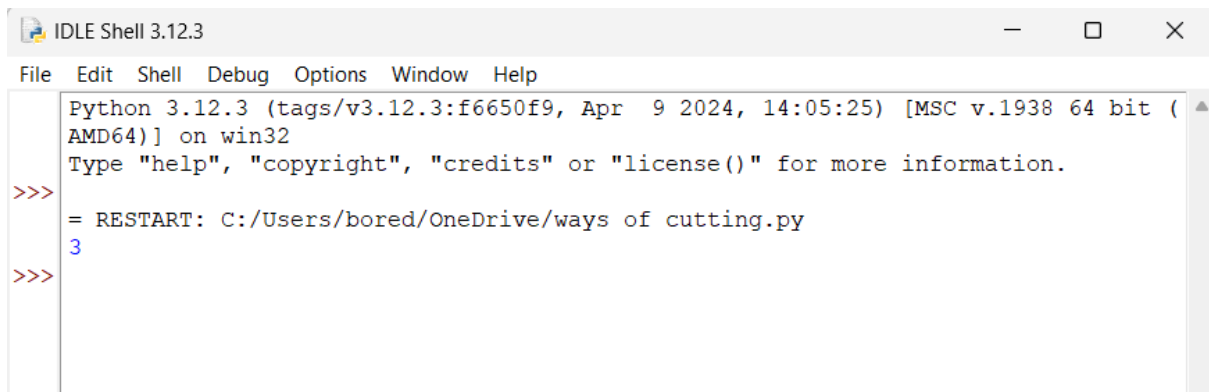
return dp(0, 0, k - 1)

pizza = ["A..", "AAA", "..."]
k = 3

print(ways(pizza, k))

```

OUTPUT:



```

IDLE Shell 3.12.3
File Edit Shell Debug Options Window Help
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/bored/OneDrive/ways of cutting.py
3
>>>

```