1.def array(arr):

return all(c in [0, 1] for c in arr)

def check(binaryarray):

if not array(binaryarray) or len(binaryarray) < 3:

return False

return binaryarray[-3:] == [0, 0, 1]

binaryarray = [1, 0, 0, 1, 0, 1, 1]

result = check(binaryarray)

print("True" if result else "False")

2. from collections import deque

def subarray(nums, limit):

max\_d=deque()

min\_d=deque()

left=0

maxlen = 0

for right in range(len(nums)):

while max\_d and nums[max\_d[-1]]<=nums[right]:

max\_d.pop()

max\_d.append(right)

while min\_d and nums[min\_d[-1]]>=nums[right]:

min\_d.pop()

min\_d.append(right)

while nums[max\_d[0]]-nums[min\_d[0]]>limit:

left+=1

if max\_d[0]<left:

max\_d.popleft()

if min\_d[0]<left:

min\_d.popleft()

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

return maxlen

print(subarray([8,2,4,7], 4))

3.. from collections import deque

def subarray(nums, limit):

max\_d=deque()

min\_d=deque()

left=0

maxlen = 0

for right in range(len(nums)):

while max\_d and nums[max\_d[-1]]<=nums[right]:

max\_d.pop()

max\_d.append(right)

while min\_d and nums[min\_d[-1]]>=nums[right]:

min\_d.pop()

min\_d.append(right)

while nums[max\_d[0]]-nums[min\_d[0]]>limit:

left+=1

if max\_d[0]<left:

max\_d.popleft()

if min\_d[0]<left:

min\_d.popleft()

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

return maxlen

print(subarray([8,2,4,7], 4))

4. def count\_triplets(arr):  
 n =len(arr)  
 xor\_prefix=[0]\*(n+1)  
 count={}  
 result=0  
  
 for i in range(1, n + 1):  
 xor\_prefix[i] = xor\_prefix[i - 1] ^ arr[i - 1]  
  
 for i in range(n):  
 for k in range(i + 1, n):  
 xor\_i\_k = xor\_prefix[i] ^ xor\_prefix[k + 1]  
 if xor\_i\_k == 0:  
 result += k - i  
 return result  
  
print(count\_triplets([1, 1, 1, 1, 1]))

5. def min\_time\_to\_collect\_apples(n, edges, has\_apple):  
  
 graph = {i: [] for i in range(n)}  
 for edge in edges:  
 graph[edge[0]].append(edge[1])  
 graph[edge[1]].append(edge[0])  
  
  
 def dfs(node):  
 total\_time = 0  
 for neighbor in graph[node]:  
 if not visited[neighbor]:  
 visited[neighbor] = True  
 time = dfs(neighbor)  
 if time > 0 or has\_apple[neighbor]:  
 total\_time += 2 + time  
 return total\_time  
  
 visited = [False] \* n  
 visited[0] = True  
 return dfs(0)  
print(min\_time\_to\_collect\_apples(7,[[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]], [False,False,True,False,True,True,False])) # Output: 8

6….def ways\_to\_cut\_pizza(pizza, k):  
 MOD = 10 \*\* 9 + 7  
 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')  
  
  
 dp = [[[0] \* k for \_ in range(cols)] for \_ in range(rows)]  
  
  
 for i in range(rows):  
 dp[i][cols - 1][0] = 1  
 for j in range(cols):  
 dp[rows - 1][j][0] = 1  
  
  
 for slices in range(1, k):  
 for i in range(rows - 1, -1, -1):  
 for j in range(cols - 1, -1, -1):  
 for x in range(i + 1, rows):  
 if prefix\_sum[i][j] - prefix\_sum[x][j] > 0:  
 dp[i][j][slices] += dp[x][j][slices - 1]  
 dp[i][j][slices] %= MOD  
 for y in range(j + 1, cols):  
 if prefix\_sum[i][j] - prefix\_sum[i][y] > 0:  
 dp[i][j][slices] += dp[i][y][slices - 1]  
 dp[i][j][slices] %= MOD  
 return dp[0][0][k - 1]  
print(ways\_to\_cut\_pizza(["A..", "AAA", "..."], 3))