Python:

1]

Question:

Given an n x n 2D matrix representing an image, rotate the image by 90 degrees (clockwise).

Sample Input:

Matrix before rotation:

```

[

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

```

Sample Output:

Matrix after rotation:

```

[

[7, 4, 1],

[8, 5, 2],

[9, 6, 3]

]

```

Short Explanation:

To rotate the matrix by 90 degrees clockwise, we can perform the rotation layer by layer. Starting from the outermost layer and moving towards the inner layers, we perform a series of swaps to rotate each element in the matrix. This involves four swaps for each element to move it to its new position after rotation. By iterating through each layer of the matrix and performing these swaps, we can achieve the desired rotation.

def rotate\_image(matrix):

n = len(matrix)

# Transpose the matrix

for i in range(n):

for j in range(i, n):

matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

# Reverse each row

for i in range(n):

matrix[i].reverse()

# Sample input matrix

matrix = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

# Rotate the matrix

rotate\_image(matrix)

# Print the rotated matrix

print("Matrix after rotation:")

for row in matrix:

print(row)

2]

Question:

Given an array of integers, find a peak element. An element is a peak element if it is greater than or equal to its neighbors. Assume the array has no duplicates and may contain multiple peaks. Return the index of any peak element if it exists.

Sample Input:

Input: [1, 2, 3, 1]

Output: 2

Explanation: In the input array, 3 is a peak element because it is greater than its neighbors (2 and 1).

Input: [1, 2, 1, 3, 5, 6, 4]

Output: 5 or 4

Explanation: In the input array, both 6 and 5 are peak elements because they are greater than their neighbors (1 and 3).

Input: [1, 2, 3, 4]

Output: 3

Explanation: In the input array, 4 is a peak element because it is greater than its neighbor (3).

Short Explanation:

To find a peak element in an array, iterate through the array and compare each element with its neighbors. If an element is greater than or equal to its neighbors, it is a peak element. Return the index of any peak element found.

def find\_peak\_element(nums):

left, right = 0, len(nums) - 1

while left < right:

mid = left + (right - left) // 2

# Check if mid element is a peak

if nums[mid] > nums[mid + 1]:

right = mid

else:

left = mid + 1

return left

# Example usage:

nums = [1, 2, 3, 1]

print("Index of a peak element:", find\_peak\_element(nums))

3]

Question:

Given a directed graph, perform topological sorting on the graph. Implement a function that takes in the number of vertices and a list of directed edges, and returns a list representing the topological ordering of the vertices. If the graph contains a cycle, return an empty list.

Sample Input:

Number of vertices: 6

Directed Edges: [(5, 2), (5, 0), (4, 0), (4, 1), (2, 3), (3, 1)]

Sample Output:

[5, 4, 2, 3, 1, 0]

Explanation:

The given directed graph has vertices numbered from 0 to 5. The edges represent the dependencies between the vertices. The topological sorting of the graph is [5, 4, 2, 3, 1, 0]. In a topological sorting, for every directed edge u -> v, vertex u comes before vertex v in the ordering.

If the graph contains a cycle, the function should return an empty list as there is no valid topological ordering.

For example, if the graph has edges [(0, 1), (1, 2), (2, 0)], it forms a cycle (0 -> 1 -> 2 -> 0), and thus, the topological sorting is not possible, so the function should return an empty list.

def topological\_sort(num\_vertices, edges):

# Initialize in-degree for each vertex

in\_degree = [0] \* num\_vertices

# Create adjacency list representation of the graph

adj\_list = [[] for \_ in range(num\_vertices)]

for u, v in edges:

adj\_list[u].append(v)

in\_degree[v] += 1

# Initialize queue for BFS and result list

queue = []

result = []

# Add vertices with in-degree 0 to the queue

for i in range(num\_vertices):

if in\_degree[i] == 0:

queue.append(i)

# Perform BFS

while queue:

u = queue.pop(0)

result.append(u)

# Decrease in-degree of adjacent vertices

for v in adj\_list[u]:

in\_degree[v] -= 1

if in\_degree[v] == 0:

queue.append(v)

# Check for cycle

if len(result) != num\_vertices:

return []

return result

# Sample Input

num\_vertices = 6

edges = [(5, 2), (5, 0), (4, 0), (4, 1), (2, 3), (3, 1)]

# Sample Output

print("Topological Ordering:", topological\_sort(num\_vertices, edges))

4]

Question:

Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Sample Input:

Input: ["2", "1", "+", "3", "\*"]

Output: 9

Explanation: ((2 + 1) \* 3) = 9

Sample Input:

Input: ["4", "13", "5", "/", "+"]

Output: 6

Explanation: (4 + (13 / 5)) = 6

Short Explanation:

Reverse Polish Notation (RPN) is a mathematical notation in which every operator follows all of its operands. To evaluate an expression in RPN, we iterate through the tokens (numbers and operators) from left to right. When encountering a number, we push it onto a stack. When encountering an operator, we pop the required number of operands from the stack, perform the operation, and push the result back onto the stack. After processing all tokens, the final result will be left on the stack.

4]

Question:

Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Sample Input:

Input: ["2", "1", "+", "3", "\*"]

Output: 9

Explanation: ((2 + 1) \* 3) = 9

Sample Input:

Input: ["4", "13", "5", "/", "+"]

Output: 6

Explanation: (4 + (13 / 5)) = 6

Short Explanation:

Reverse Polish Notation (RPN) is a mathematical notation in which every operator follows all of its operands. To evaluate an expression in RPN, we iterate through the tokens (numbers and operators) from left to right. When encountering a number, we push it onto a stack. When encountering an operator, we pop the required number of operands from the stack, perform the operation, and push the result back onto the stack. After processing all tokens, the final result will be left on the stack.

SQL:

1] Find the top 5 segments with the highest average age among customers:

SELECT Segment, AVG(Age) AS AvgAge

FROM Customers

GROUP BY Segment

ORDER BY AvgAge DESC

LIMIT 5;

2] List the customers who have the same postal code but different cities:

SELECT c1.CustomerID, c1.Name, c1.PostalCode, c1.City

FROM Customers c1

JOIN Customers c2 ON c1.PostalCode = c2.PostalCode

WHERE c1.CustomerID <> c2.CustomerID

AND c1.City <> c2.City;

3] Find the customers whose age is greater than the average age of customers in their respective segments:

SELECT c.CustomerID, c.Name, c.Age, c.Segment

FROM Customers c

JOIN (

SELECT Segment, AVG(Age) AS AvgAge

FROM Customers

GROUP BY Segment

) AS AvgAgePerSegment ON c.Segment = AvgAgePerSegment.Segment

WHERE c.Age > AvgAgePerSegment.AvgAge;