Python:

1]

Question:

Create a Python function to calculate the maximum sum of a path in a triangle of numbers.

Sample Input:

```

triangle = [

[3],

[7, 4],

[2, 4, 6],

[8, 5, 9, 3]

]

```

Sample Output:

```

Maximum path sum: 23

```

Explanation:

The function should take a triangle of numbers as input, where each number represents a node in the triangle. The function should find the path from the top to the bottom of the triangle that maximizes the sum of the numbers along the path. For the provided sample input, the maximum path sum is 23, following the path 3 -> 7 -> 4 -> 9.

def maximum\_path\_sum(triangle):

if not triangle:

return 0

# Start from the second last row and move upwards

for i in range(len(triangle) - 2, -1, -1):

for j in range(len(triangle[i])):

# For each number, add the maximum of the two adjacent numbers from the row below

triangle[i][j] += max(triangle[i + 1][j], triangle[i + 1][j + 1])

# The top of the triangle will have the maximum sum

return triangle[0][0]

# Sample Input

triangle = [

[3],

[7, 4],

[2, 4, 6],

[8, 5, 9, 3]

]

# Calculate and print the maximum path sum

print("Maximum path sum:", maximum\_path\_sum(triangle))

2]

Question:

Write a Python program to solve the subset sum problem using dynamic programming.

Sample Input:

- Target sum: 9

- List of numbers: [3, 34, 4, 12, 5, 2]

Sample Output:

- True (since there exists a subset [3, 4, 2] that sums up to 9)

Explanation:

The subset sum problem is to find whether there exists a subset of the given set of numbers that adds up to a given target sum. This Python program utilizes dynamic programming to efficiently solve this problem by building a 2D table to store subproblem solutions and iteratively computing the solutions. If a subset sum equals the target sum, it returns True; otherwise, it returns False.

def subset\_sum(target\_sum, num\_list):

dp = [[False] \* (target\_sum + 1) for \_ in range(len(num\_list) + 1)]

dp[0][0] = True

for i in range(1, len(num\_list) + 1):

for j in range(target\_sum + 1):

if j < num\_list[i - 1]:

dp[i][j] = dp[i - 1][j]

else:

dp[i][j] = dp[i - 1][j] or dp[i - 1][j - num\_list[i - 1]]

return dp[len(num\_list)][target\_sum]

# Sample Input

target\_sum = 9

num\_list = [3, 34, 4, 12, 5, 2]

# Check if there exists a subset with the given sum

print(subset\_sum(target\_sum, num\_list))

3]

Question:

Implement a Python function to perform the Boyer-Moore algorithm for string searching.

Sample Input:

```python

text = "ABAAABCD"

pattern = "ABC"

```

Sample Output:

```

Pattern found at index 4

```

Explanation:

The Boyer-Moore algorithm is an efficient string searching algorithm that works by scanning the pattern from right to left and skipping characters based on precomputed tables. In this example, the pattern "ABC" is found in the text "ABAAABCD" starting at index 4.

def boyer\_moore(text, pattern):

n = len(text)

m = len(pattern)

last = {char: i for i, char in enumerate(pattern)}

i = m - 1

j = m - 1

while i < n:

if text[i] == pattern[j]:

if j == 0:

return i

else:

i -= 1

j -= 1

else:

i += m - min(j, 1 + last.get(text[i], -1))

j = m - 1

return -1

# Sample Input

text = "ABAAABCD"

pattern = "ABC"

# Find and print the index where the pattern is found

index = boyer\_moore(text, pattern)

if index != -1:

print("Pattern found at index", index)

else:

print("Pattern not found")

4]

Question:

Implement the Burrows-Wheeler transform algorithm in Python.

Sample Input:

Input: "banana"

Sample Output:

Output: "annb$aa"

Explanation:

The Burrows-Wheeler transform rearranges a string into a sequence of characters that are easier to compress. In this example, the input string "banana" gets transformed into "annb$aa".

def burrows\_wheeler\_transform(input\_string):

rotations = [input\_string[i:] + input\_string[:i] for i in range(len(input\_string))]

sorted\_rotations = sorted(rotations)

bwt\_transform = ''.join(rotation[-1] for rotation in sorted\_rotations)

return bwt\_transform

# Sample Input

input\_string = "banana"

# Obtain and print the Burrows-Wheeler Transform

bwt\_result = burrows\_wheeler\_transform(input\_string)

print("Output:", bwt\_result)

SQL:

1] Find the customers who have placed orders for products with prices that are prime numbers.

SELECT DISTINCT c.customer\_id, c.customer\_name

FROM customers c

JOIN orders o ON c.customer\_id = o.customer\_id

JOIN order\_details od ON o.order\_id = od.order\_id

JOIN products p ON od.product\_id = p.product\_id

WHERE p.price > 1 AND p.price NOT IN (

SELECT DISTINCT price

FROM products

WHERE price > 1 AND NOT EXISTS (

SELECT 1

FROM numbers n

WHERE n.number < price AND price % n.number = 0

)

);

2] List the customers along with the earliest and latest order dates.

SELECT c.customer\_id, c.customer\_name, MIN(o.order\_date) AS earliest\_order\_date, MAX(o.order\_date) AS latest\_order\_date

FROM customers c

JOIN orders o ON c.customer\_id = o.customer\_id

GROUP BY c.customer\_id, c.customer\_name;

3] Retrieve the customers who have ordered products from categories with names containing specific keywords.

SELECT DISTINCT c.customer\_id, c.customer\_name

FROM customers c

JOIN orders o ON c.customer\_id = o.customer\_id

JOIN order\_details od ON o.order\_id = od.order\_id

JOIN products p ON od.product\_id = p.product\_id

JOIN categories cat ON p.category\_id = cat.category\_id

WHERE LOWER(cat.category\_name) LIKE '%keyword1%' OR LOWER(cat.category\_name) LIKE '%keyword2%';