Python:

1]

Problem Statement:

N-Queens Puzzle: Given an integer \( n \), return all distinct solutions to the n-queens puzzle where \( n \) queens are placed on an \( n \times n \) chessboard such that no two queens threaten each other.

Sample Input/Output:

Input: \( n = 4 \)

Output: ([[".Q..", "...Q", "Q...", "..Q."], ["..Q.", "Q...", "...Q", ".Q.."]]\)

Explanation: Two distinct solutions to the 4-queens puzzle on a 4x4 chessboard are provided where each queen is placed such that no two queens threaten each other.

**Output:**

def solveNQueens(n):

def is\_safe(board, row, col):

# Check if there is any queen in the same column

for i in range(row):

if board[i][col] == 'Q':

return False

# Check upper left diagonal

for i, j in zip(range(row-1, -1, -1), range(col-1, -1, -1)):

if board[i][j] == 'Q':

return False

# Check upper right diagonal

for i, j in zip(range(row-1, -1, -1), range(col+1, n)):

if board[i][j] == 'Q':

return False

return True

def backtrack(row, board):

if row == n:

result.append(["".join(row) for row in board])

return

for col in range(n):

if is\_safe(board, row, col):

board[row][col] = 'Q'

backtrack(row+1, board)

board[row][col] = '.'

result = []

board = [['.' for \_ in range(n)] for \_ in range(n)]

backtrack(0, board)

return result

# Test the function with the provided sample input

n = 4

print(solveNQueens(n))

**Output:**

[['.Q..', '...Q', 'Q...', '..Q.'], ['..Q.', 'Q...', '...Q', '.Q..']]

2]

Question:

Question: Given an array of non-negative integers representing maximum jump lengths, starting from the first index, what is the minimum number of jumps needed to reach the last index?

Sample Input:

[2, 3, 1, 1, 4]

Sample Output:

2

Explanation:

To reach the last index, jump from index 0 to index 1 (jump of length 2) and then from index 1 to index 4 (jump of length 3), totaling 2 jumps.

def minJumps(nums):

n = len(nums)

if n <= 1:

return 0

# Initialize an array to store the minimum number of jumps needed to reach each index

jumps = [float('inf')] \* n

jumps[0] = 0

for i in range(1, n):

for j in range(i):

if j + nums[j] >= i: # If we can reach index i from index j

jumps[i] = min(jumps[i], jumps[j] + 1)

break

return jumps[n - 1]

# Test the function with the provided sample input

nums = [2, 3, 1, 1, 4]

print(minJumps(nums)) # Output: 2

3]

Question: Implement a wildcard pattern matching algorithm that supports '?' and '\*' characters.

Sample Input:

- Pattern: "a\*b?c"

- Text: "adceb"

Sample Output:

- True

Explanation:

The pattern "a\*b?c" matches the text "adceb" as '\*' can match any sequence of characters and '?' can match any single character.

def isMatch(pattern, text):

# Initialize a 2D DP table to store the matching status

dp = [[False] \* (len(text) + 1) for \_ in range(len(pattern) + 1)]

# Empty pattern and empty text match

dp[0][0] = True

# Handle '\*' at the beginning of the pattern

for i in range(1, len(pattern) + 1):

if pattern[i - 1] == '\*':

dp[i][0] = dp[i - 1][0]

# Fill in the DP table

for i in range(1, len(pattern) + 1):

for j in range(1, len(text) + 1):

if pattern[i - 1] == '\*':

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

elif pattern[i - 1] == '?' or pattern[i - 1] == text[j - 1]:

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

return dp[len(pattern)][len(text)]

# Test the function with the provided sample input

pattern = "a\*b?c"

text = "adbeb"

print(isMatch(pattern, text)) # Output: True

4]

Question: Given an elevation map represented by n non-negative integers, where the width of each bar is 1, calculate the amount of water that can be trapped after raining.

Sample Input:

```

Input: [0,1,0,2,1,0,1,3,2,1,2,1]

```

Sample Output:

```

Output: 6

```

Explanation: In the given elevation map, the amount of water trapped between bars is 6 units.

def trap(height):

if not height:

return 0

left\_max = [0] \* len(height)

right\_max = [0] \* len(height)

# Calculate the maximum height from the left for each bar

left\_max[0] = height[0]

for i in range(1, len(height)):

left\_max[i] = max(left\_max[i - 1], height[i])

# Calculate the maximum height from the right for each bar

right\_max[len(height) - 1] = height[len(height) - 1]

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

right\_max[i] = max(right\_max[i + 1], height[i])

# Calculate the amount of water trapped for each bar

water\_trapped = 0

for i in range(len(height)):

water\_trapped += min(left\_max[i], right\_max[i]) - height[i]

return water\_trapped

# Test the function with the provided sample input

elevation\_map = [0,1,0,2,1,0,1,3,2,1,2,1]

print(trap(elevation\_map)) # Output: 6

SQL:

1] List the customer names and their corresponding segment names for customers whose postal codes start with '9':

SELECT c.customer\_name, s.segment\_name

FROM customers c

JOIN segments s ON c.segment\_id = s.segment\_id

WHERE c.postal\_code LIKE '9%';

2] List the customer names and their corresponding states for customers in the 'Corporate' segment:

SELECT c.customer\_name, c.state

FROM customers c

JOIN segments s ON c.segment\_id = s.segment\_id

WHERE s.segment\_name = 'Corporate';

3] List the customer names and their corresponding cities for customers in the 'Home Office' segment:

SELECT c.customer\_name, c.city

FROM customers c

JOIN segments s ON c.segment\_id = s.segment\_id

WHERE s.segment\_name = 'Home Office';