

CSA0619-DAA(DAY 2)

1.

```
main.py  Run  Output  Clear
1 MOD = 1000000007
2 def findPaths(m, n, N, i, j):
3     dp = [[0] * n for _ in range(m)]
4     dp[i][j] = 1
5     result = 0
6     for steps in range(N):
7         temp = [[0] * n for _ in range(m)]
8         for r in range(m):
9             for c in range(n):
10                if dp[r][c] > 0:
11                    if r == 0:
12                        result = (result + dp[r][c]) % MOD
13                    else:
14                        temp[r-1][c] = (temp[r-1][c] + dp[r][c]) % MOD
15
16                    if r == m-1:
17                        result = (result + dp[r][c]) % MOD
18                    else:
19                        temp[r+1][c] = (temp[r+1][c] + dp[r][c]) % MOD
20
21                if c == 0:
22                    result = (result + dp[r][c]) % MOD
```

6

=== Code Execution Successful ===

2.

```
main.py  Run  Output  Clear
1 def rob_linear(nums):
2     prev, curr = 0, 0
3     for num in nums:
4         prev, curr = curr, max(curr, prev + num)
5     return curr
6 def rob(nums):
7     if len(nums) == 1:
8         return nums[0]
9     money1 = rob_linear(nums[:-1])
10    money2 = rob_linear(nums[1:])
11    return max(money1, money2)
12 nums = [2, 3, 2]
13 print(rob(nums))
14
```

3

=== Code Execution Successful ===

CSA0619-DAA(DAY 2)

3.

main.py	Output
<pre>1 - def climbStairs(n): 2 - if n == 1: 3 - return 1 4 - if n == 2: 5 - return 2 6 - prev2 = 1 7 - prev1 = 2 8 - for i in range(3, n+1): 9 - curr = prev1 + prev2 10 - prev2 = prev1 11 - prev1 = curr 12 - return prev1 13 print(climbStairs(4)) 14 print(climbStairs(3)) 15</pre>	<pre>5 3 === Code Execution Successful ===</pre>

4.

main.py	Output
<pre>1 - def uniquePaths(m, n): 2 - dp = [[1] * n for _ in range(m)] 3 - for i in range(1, m): 4 - for j in range(1, n): 5 - dp[i][j] = dp[i-1][j] + dp[i][j-1] 6 - return dp[m-1][n-1] 7 print(uniquePaths(7, 3)) 8</pre>	<pre>28 === Code Execution Successful ===</pre>

CSA0619-DAA(DAY 2)

5.

main.py	Output
<pre>1 def largeGroupPositions(s): 2 result = [] 3 n = len(s) 4 start = 0 5 for i in range(1, n + 1): 6 if i == n or s[i] != s[start]: 7 if i - start >= 3: 8 result.append([start, i - 1]) 9 start = i 10 return result 11 s = "abbxxxxxzy" 12 print(largeGroupPositions(s)) 13</pre>	<pre>[[3, 6]] === Code Execution Successful ===</pre>

6.

main.py	Output
<pre>1 def gameOfLife(board): 2 if not board: 3 return 4 rows, cols = len(board), len(board[0]) 5 directions = [(-1, -1), (-1, 0), (-1, 1), 6 (0, -1), (0, 1), 7 (1, -1), (1, 0), (1, 1)] 8 next_state = [[0] * cols for _ in range(rows)] 9 for r in range(rows): 10 for c in range(cols): 11 live_neighbors = 0 12 for dr, dc in directions: 13 nr, nc = r + dr, c + dc 14 if 0 <= nr < rows and 0 <= nc < cols: 15 live_neighbors += board[nr][nc] 16 if board[r][c] == 1: 17 if live_neighbors < 2: 18 next_state[r][c] = 0 19 elif live_neighbors <= 3: 20 next_state[r][c] = 1 21 else: 22 next_state[r][c] = 0 23 else: 24 if live_neighbors == 3:</pre>	<pre>[0, 0, 0] [1, 0, 1] [0, 1, 1] [0, 1, 0] === Code Execution Successful ===</pre>

CSA0619-DAA(DAY 2)

7.

main.py

Share

Run

```
1 def champagneTower(poured, query_row, query_glass):
2     tower = [[0] * k for k in range(1, 102)]
3     tower[0][0] = poured
4     for r in range(query_row):
5         for c in range(r + 1):
6             overflow = (tower[r][c] - 1.0) / 2.0
7             if overflow > 0:
8                 tower[r + 1][c] += overflow
9                 tower[r + 1][c + 1] += overflow
10    return min(1, tower[query_row][query_glass])
11 poured = 1
12 query_row = 1
13 query_glass = 1
14 print(champagneTower(poured, query_row, query_glass))
```

Output

Clear

0

=== Code Execution Successful ===