1.Dice throw problem

```python
def findWays(n, m, X):
    dp = [[0] * (X + 1) for _ in range(n + 1)]
    dp[0][0] = 1

    for i in range(1, n + 1):
        for j in range(1, X + 1):
            for k in range(1, m + 1):
                if j >= k:
                    dp[i][j] += dp[i - 1][j - k]

    return dp[n][X]

# Example usage
n = 2
m = 6
X = 7
print(findWays(n, m, X))  # Output: 6
```

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

=== Code Execution Successful ===
```

```python
import random

def roll_dice(n):
    """Return a list of integers with length `n`.

    Each integer in the returned list is a random number between 1 and 6,
        inclusive."""
    roll_results = []
    for _ in range(n):
        roll = random.randint(1, 6)
        roll_results.append(roll)
    return roll_results

# Example usage
n = 2
m = 6
X = 7
print(roll_dice(n))
```

```
[5, 2]

=== Code Execution Successful ===
```

2.Subset sum

main.py

Save Run

```python
def subset_sum(set_nums, target_sum):
    n = len(set_nums)
    dp = [[False] * (target_sum + 1) for _ in range(n + 1)]

    for i in range(n + 1):
        dp[i][0] = True

    for i in range(1, n + 1):
        for j in range(1, target_sum + 1):
            if j < set_nums[i - 1]:
                dp[i][j] = dp[i - 1][j]
            else:
                dp[i][j] = dp[i - 1][j] or dp[i - 1][j - set_nums[i - 1]]

    result = []
    for i in range(n, 0, -1):
        if dp[i][target_sum]:
            subset = []
            j = target_sum
            for k in range(i, 0, -1):
                if j >= set_nums[k - 1] and dp[k][j]:
                    subset.append(set_nums[k - 1])
                    j -= set_nums[k - 1]
            result.append(subset[::-1])

    return result
```

Output

```
[[2, 5], [3, 4]]

=== Code Execution Successful ===
```

main.py

Save Run

```python
def find_subsets(set_nums, target_sum):
    result = []
    backtrack(set_nums, target_sum, 0, [], result)
    return result

def backtrack(set_nums, target_sum, start, subset, result):
    if target_sum == 0:
        result.append(subset[:])
        return

    for i in range(start, len(set_nums)):
        if set_nums[i] > target_sum:
            return

        subset.append(set_nums[i])
        backtrack(set_nums, target_sum - set_nums[i], i + 1, subset, result)
        subset.pop()

# Example usage
set_nums = [1, 2, 3, 4, 5]
target_sum = 7
print(find_subsets(set_nums, target_sum))
```

Output

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

=== Code Execution Successful ===
```

3. Assembly line scheduling

```python
def assembly_line_scheduling(a, t, e, x):
    n = len(a)
    f = [[0] * (n + 1) for _ in range(2)]
    f[0][1] = e[0] + a[0][1]
    f[1][1] = e[1] + a[1][1]

    for j in range(2, n + 1):
        f[0][j] = min(f[0][j-1] + a[0][j], f[1][j-1] + t[1][j-1] + a[0][j])
        f[1][j] = min(f[0][j-1] + a[1][j], f[1][j-1] + t[0][j-1] + a[1][j])

    return min(f[0][n] + x[0], f[1][n] + x[1])

# Example usage
a = [[7, 9, 3, 4, 8, 4], [8, 5, 6, 4, 5, 7]]
t = [[2, 3, 1, 3, 4], [2, 1, 2, 2, 3]]
e = [5, 10]
x = [12, 15]

print(assembly_line_scheduling(a, t, e, x))  # Output: 34
```

```
29

=== Code Execution Successful ===
```

```python
def assembly_line_scheduling_greedy(a, t, e, x):
    n = len(a)
    total_time = 0
    current_line = 0

    for i in range(n):
        if a[current_line][i] < a[1 - current_line][i]:
            total_time += a[current_line][i]
        else:
            total_time += a[1 - current_line][i]
            current_line = 1 - current_line

    return total_time + e[current_line] + x[current_line]

# Example usage
a = [[7, 9, 3, 4, 8, 4], [8, 5, 6, 4, 5, 7]]
t = [[2, 3, 1, 3, 4], [2, 1, 2, 2, 3]]
e = [5, 10]
x = [12, 15]

print(assembly_line_scheduling_greedy(a, t, e, x))  # Output: 34
```

```
37

=== Code Execution Successful ===
```

4. Longest Palindromic subsequence

```python
def longestPalinSubseq(S):
    R = S[::-1]

    # dp[i][j] will store the length of the longest
    # palindromic subsequence for the substring
    # starting at index i and ending at index j
    dp = [[0] * (len(R) + 1) for _ in range(len(S) + 1)]

    # Filling up DP table based on conditions discussed
    # in the above approach
    for i in range(1, len(S) + 1):
        for j in range(1, len(R) + 1):
            if S[i - 1] == R[j - 1]:
                dp[i][j] = 1 + dp[i - 1][j - 1]
            else:
                dp[i][j] = max(dp[i][j - 1], dp[i - 1][j])

    # At the end, DP table will contain the LPS
    # So just return the length of LPS
    return dp[len(S)][len(R)]


# Driver code
s = "GEEKSFORGEEKS"
print("The length of the LPS is", longestPalinSubseq(s))
```

```
The length of the LPS is 5

=== Code Execution Successful ===
```

main.py

Save Run

```python
# A space optimized solution for assembly
# line scheduling in Python3
def carAssembleTime(a, t, e, x):

    n = len(a[0])

    # Time taken to leave first station
    # in line 1
    first = e[0] + a[0][0]

    # Time taken to leave first station
    # in line 2
    second = e[1] + a[1][0]

    for i in range(1, n):
        up = min(first + a[0][i],
                 second + t[1][i] + a[0][i])
        down = min(second + a[1][i],
                 first + t[0][i] + a[1][i])

        first, second = up, down

    first += x[0]
    second += x[1]

    return min(first, second)
```

Output

```
35

=== Code Execution Successful ===
```