

## Topic-4 Dynamic Programming

### 1. Dice Throw Problem using Dynamic Programming

**Aim:** Find number of ways to get a target sum with given number of dice and sides.

**Algorithm:**

Step-1: Create DP table  $dp[dice+1][target+1]$

Step-2: Initialize  $dp[0][0] = 1$

Step-3: For each dice, for each sum, add ways using each face value

Step-4: Return  $dp[num\_dice][target]$  **Input**

**& Output:**

main.py	Output
<pre>1- def dice_throw(num_sides, num_dice, target): 2   dp = [[0 for _ in range(target + 1)] for _ in range(num_dice + 1)] 3   dp[0][0] = 1 4   for i in range(1, num_dice + 1): 5       for j in range(1, target + 1): 6           for k in range(1, num_sides + 1): 7               if j &gt;= k: 8                   dp[i][j] += dp[i - 1][j - k] 9   return dp[num_dice][target] 10 num_sides_1 = 6 11 num_dice_1 = 2 12 target_1 = 7 13 result_1 = dice_throw(num_sides_1, num_dice_1, target_1) 14 num_sides_2 = 4 15 num_dice_2 = 3 16 target_2 = 10 17 result_2 = dice_throw(num_sides_2, num_dice_2, target_2) 18 print(f"Number of sides: {num_sides_1}, Number of dice: {num_dice_1}, Target sum: 19       {target_1}") 20 print(f"Number of ways to reach sum {target_1}: {result_1}\n") 21 print(f"Number of sides: {num_sides_2}, Number of dice: {num_dice_2}, Target sum: 22       {target_2}") 23 print(f"Number of ways to reach sum {target_2}: {result_2}")</pre>	<pre>Number of sides: 6, Number of dice: 2, Target sum: 7 Number of ways to reach sum 7: 6  Number of sides: 4, Number of dice: 3, Target sum: 10 Number of ways to reach sum 10: 6  === Code Execution Successful ===</pre>

### 2. Assembly Line Scheduling (2 Lines)

**Aim:** Find minimum time to assemble product through 2 lines.

**Algorithm:**

Step-1: Use DP arrays T1, T2 for each line

Step-2: Compute time at each station with/without transfer

Step-3: Add entry and exit times

Step-4: Return  $\min(T1[n-1]+x1, T2[n-1]+x2)$  **Input**

**& Output:**

main.py	Output
<pre> 1- def dice_throw(num_sides, num_dice, target): 2   dp = [[0] * (target + 1) for _ in range(num_dice + 1)] 3   dp[0][0] = 1 4   for i in range(1, num_dice + 1): 5       for j in range(1, target + 1): 6           for k in range(1, num_sides + 1): 7               if j - k &gt;= 0: 8                   dp[i][j] += dp[i - 1][j - k] 9   return dp[num_dice][target] 10 num_sides = int(input("Enter number of sides on each die: ")) 11 num_dice = int(input("Enter number of dice: ")) 12 target = int(input("Enter target sum: ")) 13 ways = dice_throw(num_sides, num_dice, target) 14 print(f"\nNumber of ways to get sum {target} using {num_dice} dice     with {num_sides} sides: {ways}") </pre>	<pre> Enter number of sides on each die: 6 Enter number of dice: 2 Enter target sum: 7  Number of ways to get sum 7 using 2 dice with 6 sides: 6  === Code Execution Successful === </pre>

### 3. Three Assembly Lines Scheduling

**Aim:** Minimize total time across 3 lines with dependencies.

**Algorithm:**

Step-1: Initialize  $dp[i][line] = \text{time}$

Step-2: Add min transfer from previous station

Step-3: Respect dependencies

Step-4: Return min total

**Input & Output**

main.py	Output
<pre> 1- def dice_throw(num_sides, num_dice, target): 2   dp = [[0 for _ in range(target + 1) for _ in range(num_dice + 1)] 3   dp[0][0] = 1 4   for i in range(1, num_dice + 1): 5       for j in range(1, target + 1): 6           for k in range(1, num_sides + 1): 7               if j &gt;= k: 8                   dp[i][j] += dp[i - 1][j - k] 9   return dp[num_dice][target] 10 num_sides_1 = 6 11 num_dice_1 = 2 12 target_1 = 7 13 result_1 = dice_throw(num_sides_1, num_dice_1, target_1) 14 num_sides_2 = 4 15 num_dice_2 = 3 16 target_2 = 10 17 result_2 = dice_throw(num_sides_2, num_dice_2, target_2) 18 print(f"Number of sides: {num_sides_1}, Number of dice: {num_dice_1}, Target sum:     {target_1}") 19 print(f"Number of ways to reach sum {target_1}: {result_1}\n") 20 print(f"Number of sides: {num_sides_2}, Number of dice: {num_dice_2}, Target sum:     {target_2}") 21 print(f"Number of ways to reach sum {target_2}: {result_2}") </pre>	<pre> Number of sides: 6, Number of dice: 2, Target sum: 7 Number of ways to reach sum 7: 6  Number of sides: 4, Number of dice: 3, Target sum: 10 Number of ways to reach sum 10: 6  === Code Execution Successful === </pre>

### 4. Minimum Path Distance (Matrix form - TSP)

**Aim:** Find minimum path visiting all cities (TSP).

**Algorithm:**

Step-1: Use DP with bitmasking Step-2:

Recursively compute all paths Step-3:

Return minimal cycle cost.

**Input & Output:**

main.py	Run	Output
<pre> 1 N = 4 2 dist = [ 3     [0, 10, 15, 20], 4     [10, 0, 35, 25], 5     [15, 35, 0, 30], 6     [20, 25, 30, 0] 7 ] 8 memo = [[-1]*(1&lt;&lt;N) for _ in range(N)] 9 def tsp(city, visited): 10     if visited == (1 &lt;&lt; N) - 1: 11         return dist[city][0] 12 13     if memo[city][visited] != -1: 14         return memo[city][visited] 15 16     ans = float('inf') 17     for next_city in range(N): 18         if not (visited &amp; (1 &lt;&lt; next_city)): 19             temp = dist[city][next_city] + tsp(next_city, visited   (1 &lt;&lt; next_city)) 20             ans = min(ans, temp) 21     memo[city][visited] = ans </pre>	Run	<pre> Minimum Path Distance: 80  === Code Execution Successful === </pre>

## 5. TSP with New City (E)

**Aim:** Find shortest route including new city.

**Algorithm:**

Step-1: Use permutations to check all paths

Step-2: Compute total distance

Step-3: Return minimal route

## Input & Output

main.py	Run	Output
<pre> 1 N = 5 2 cities = ['A', 'B', 'C', 'D', 'E'] 3 dist = [ 4     [0, 10, 15, 20, 25], 5     [10, 0, 35, 25, 30], 6     [15, 35, 0, 30, 20], 7     [20, 25, 30, 0, 15], 8     [25, 30, 20, 15, 0] 9 ] 10 memo = [[-1]*(1&lt;&lt;N) for _ in range(N)] 11 path_memo = [[-1]*(1&lt;&lt;N) for _ in range(N)] 12 13 def tsp(city, visited): 14     if visited == (1 &lt;&lt; N) - 1: 15         return dist[city][0] 16 17     if memo[city][visited] != -1: 18         return memo[city][visited] 19 20     ans = float('inf') 21     for next_city in range(N): 22         if not (visited &amp; (1 &lt;&lt; next_city)): 23             temp = dist[city][next_city] + tsp(next_city, visited   (1 &lt;&lt; next_city)) 24             if temp &lt; ans: 25                 ans = temp </pre>	Run	<pre> Minimum Distance: 85 Optimal Route: A -&gt; B -&gt; D -&gt; E -&gt; C -&gt; A  === Code Execution Successful === </pre>