```
#1.throw and dice
Def count(n, m, X):
    Dp = [[0] * (X + 1) for h in range(n + 1)]
    Dp[0][0] = 1
    For i in range(1, n + 1):
        For j in range(1, X + 1):
            Dp[i][j] = 0
            For k in range(1, m + 1):
                If j – k >= 0:
                    Dp[i][j] += dp[i – 1][j – k]

    Return dp[n][X]
N = 3
M = 6
X = 8
Print(f”Number of ways to get sum {X} with {n} dice: {count(n, m, X)}”)
```

```
#2.tsp in dp
Import itertools
Def tsp(dist):
    N = len(dist)
    Dp = [[float(‘inf’)] * n for _ in range(1 << n)]
    Dp[1][0] = 0

    For mask in range(1 << n):
        For i in range(n):
            If mask & (1 << i):
                For j in range(n):
```

```
          If mask & (1 << j) and i != j:
            Dp[mask][i] = min(dp[mask][i], dp[mask ^ (1 << i)][j] + dist[j][i])
  Res = float('inf')
  For i in range(1, n):
    Final_res = min(res, dp[(1 << n) – 1][i] + dist[i][0])

  Return res
Dist = [[0, 10, 15, 20],[10, 0, 35, 25],[15, 35, 0, 30],[20, 25, 30, 0]]



Print(f"Minimum cost of visiting all cities: {tsp(dist)}")

#obst in dp
Def obst(keys, p):
  N = len(keys)
  Cost = [[0 for a in range(n + 1)] for a in range(n + 1)]
  Sump = [[0 for b in range(n + 1)] for b in range(n + 1)]

  For i in range(1, n + 1):
    Cost[i][i – 1] = 0
    Cost[i][i] = p[i – 1]
    Sump[i][i] = p[i – 1]

  For length in range(2, n + 1):
    For i in range(1, n – length + 2):
      J = i + length – 1
      Cost[i][j] = float('inf')
      Sump[i][j] = sump[i][j – 1] + p[j – 1]
      For r in range(i, j + 1):
        Ltcost = cost[i][r – 1] if r > i else 0
        Rtcost = cost[r + 1][j] if r < j else 0
        Currcost = ltcost + rtcost + sump[i][j]

        If currcost < cost[i][j]:
          Cost[i][j] = currcost

  Return cost[1][n]
Keys = [10, 20, 30, 40]
P = [0.1, 0.2, 0.4, 0.3]
Print(f"Minimum cost of Optimal BST: {obst(keys, p)}")


'''output:
Number of ways to get sum 8 with 3 dice: 21
Minimum cost of visiting all cities: inf
Minimum cost of Optimal BST: 1.7'''
```