LAB DAY-9

1.n number of dices problem:

```
def combinations(n, x, current=None, result=None):
    if current is None:
        current = []
    if result is None:
        result = []

    if n == 0:
        if x == 0:
            print(f"Found valid combination: {current}")
            result.append(current.copy())
        else:
            print(f"Combination discarded (sum mismatch): {current}")
        return

    for i in range(1, 7):
        if x - i >= 0:
            current.append(i)
            combinations(n - 1, x - i, current, result)
            current.pop()

    return result

def dice(n, x):
    return combinations(n, x)

n = 3
x = 7
combinations = dice(n, x)
```

```
print(combinations)
```

OUTPUT:

```
Combination discarded (sum mismatch): [1, 1, 1]
Combination discarded (sum mismatch): [1, 1, 2]
Combination discarded (sum mismatch): [1, 1, 3]
Combination discarded (sum mismatch): [1, 1, 4]
Found valid combination: [1, 1, 5]
Combination discarded (sum mismatch): [1, 2, 1]
Combination discarded (sum mismatch): [1, 2, 2]
Combination discarded (sum mismatch): [1, 2, 3]
Found valid combination: [1, 2, 4]
Combination discarded (sum mismatch): [1, 3, 1]
Combination discarded (sum mismatch): [1, 3, 2]
Found valid combination: [1, 3, 3]
Combination discarded (sum mismatch): [1, 4, 1]
Found valid combination: [1, 4, 2]
Found valid combination: [1, 5, 1]
Combination discarded (sum mismatch): [2, 1, 1]
Combination discarded (sum mismatch): [2, 1, 2]
Combination discarded (sum mismatch): [2, 1, 3]
Found valid combination: [2, 1, 4]
Combination discarded (sum mismatch): [2, 2, 1]
Combination discarded (sum mismatch): [2, 2, 2]
Found valid combination: [2, 2, 3]
Combination discarded (sum mismatch): [2, 3, 1]
Found valid combination: [2, 3, 2]
Found valid combination: [2, 4, 1]
Combination discarded (sum mismatch): [3, 1, 1]
Combination discarded (sum mismatch): [3, 1, 2]
Found valid combination: [3, 1, 3]
Combination discarded (sum mismatch): [3, 2, 1]
```

Found valid combination: [3, 2, 2]

Found valid combination: [3, 3, 1]

Combination discarded (sum mismatch): [4, 1, 1]

Found valid combination: [4, 1, 2]

Found valid combination: [4, 2, 1]

Found valid combination: [5, 1, 1]

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

2.Traveling salesman problem:

```
import sys


def tsp(graph, n):
    memo = {}

    def tsp_dp(current, visited):
        if visited == (1 << n) - 1:
            return graph[current][0]

        if (current, visited) in memo:
            return memo[(current, visited)]



        min_cost = sys.maxsize

        for next_city in range(n):
            if visited & (1 << next_city) == 0:
                new_visited = visited | (1 << next_city)
                cost = graph[current][next_city] + tsp_dp(next_city, new_visited)
                min_cost = min(min_cost, cost)
```

```
        memo[(current, visited)] = min_cost
        return min_cost

    start_city = 0
    initial_visited = 1 << start_city
    return tsp_dp(start_city, initial_visited)

if __name__ == "__main__":
    graph = [
        [0, 10, 15, 20],
        [10, 0, 35, 25],
        [15, 35, 0, 30],
        [20, 25, 30, 0]
    ]
    n = len(graph)

    min_cost = tsp(graph, n)
    print(f"Minimum cost of TSP: {min_cost}")
```

OUTPUT: Minimum cost of TSP: 80

3.Optimal binary search tree:

```
def obst(keys, freq):
    n = len(keys)

    cost = [[0] * n for _ in range(n)]
    root = [[0] * n for _ in range(n)]

    for i in range(n):
        cost[i][i] = freq[i]
        root[i][i] = i
```

```
    for length in range(2, n + 1):
        for i in range(n - length + 1):
            j = i + length - 1
            cost[i][j] = float('inf')

            for r in range(i, j + 1):
                c = cost[i][r - 1] if r > i else 0
                c += cost[r + 1][j] if r < j else 0
                c += sum(freq[i:j + 1])

                if c < cost[i][j]:
                    cost[i][j] = c
                    root[i][j] = r

    return root

def constructbst(keys, freq):
    n = len(keys)
    if n == 0:
        return None

    root = obst(keys, freq)

    def constructtree(i, j):
        if i > j:
            return None
        elif i == j:
            return Node(keys[i])

        r = root[i][j]
        node = Node(keys[r])
```

```
    node.left = constructtree(i, r - 1)
    node.right = constructtree(r + 1, j)
    return node

return constructtree(0, n - 1)
```