**ASSIGNMENT – 6**

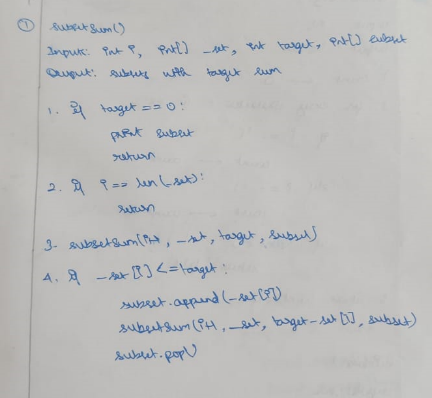
**AIM:**

To solve and implement the given problems using Backtracking.

**Qn1:**

****

**Psuedo Code:**

****

**Source Code:**

def subsetSum(i, \_set, target\_sum, subset):

    if target\_sum == 0:

        print(subset, end=", ")

        return

    if i==len(\_set):

        return

    subsetSum(i+1, \_set, target\_sum, subset)

    if \_set[i] <= target\_sum:

        subset.append(\_set[i])

        subsetSum(i+1, \_set, target\_sum-\_set[i], subset)

        subset.pop()

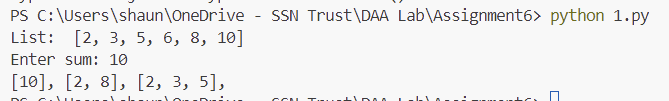
l = [2,3,5,6,8,10]

print("List: ", l)

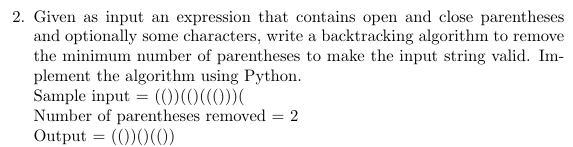
sum = int(input("Enter sum: "))

subsetSum(0, l, 10, [])

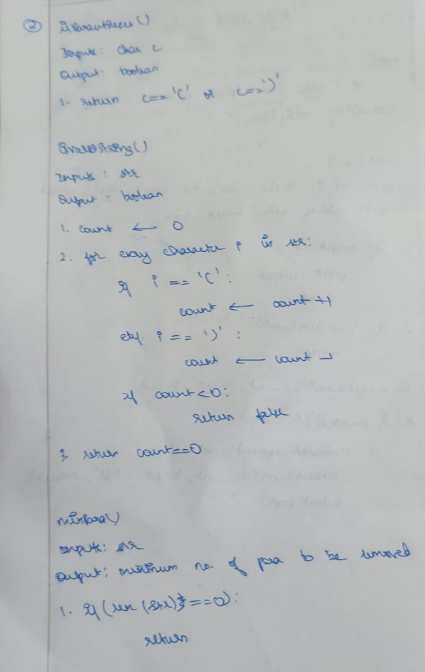
**Output:**

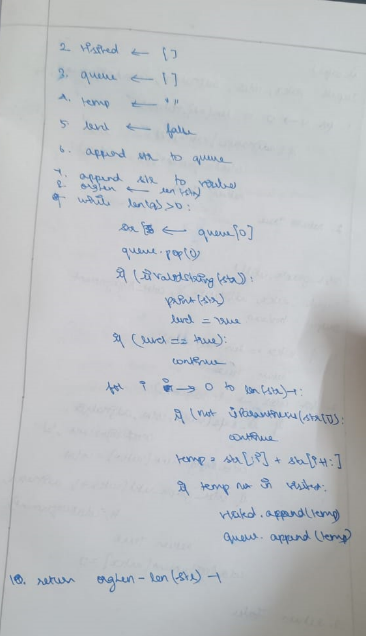
****

**Qn2:**

****

**Psuedo Code:**

****

****

**Source Code:**

def isParanthesis(c):

    return c=="(" or c==")"

def isValidString(str):

    count = 0

    for i in str:

        if i == "(":

            count += 1

        elif i == ")":

            count -= 1

        if count<0:

            return False

    return count==0

def minPara(str):

    if len(str) == 0:

        return

    visited = []

    q = []

    temp = ""

    level = False

    orgLen = len(str)

    q.append(str)

    visited.append(str)

    while len(q) > 0:

        str = q[0]

        q.pop(0)

        if isValidString(str):

            print(str)

            level = True

        if level:

            continue

        for i in range(len(str)):

            if not isParanthesis(str[i]):

                continue

            temp = str[:i] + str[i+1:]

            if temp not in visited:

                visited.append(temp)

                q.append(temp)

    return orgLen - len(str) - 1

expression = "()()))())"

print("String: ", expression)

min\_count = minPara(expression)

print(f"Min Para to be removed: {min\_count}")

print()

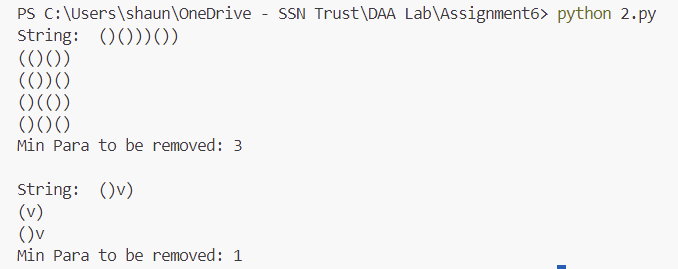
expression = "()v)"

print("String: ", expression)

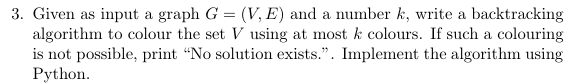
min\_count = minPara(expression)

print(f"Min Para to be removed: {min\_count}")

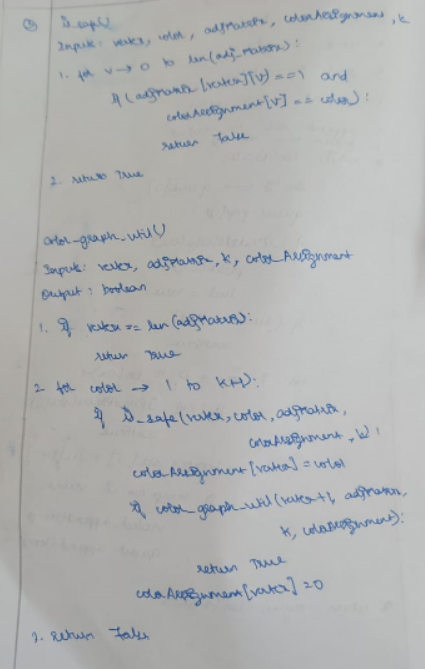
**Output:**

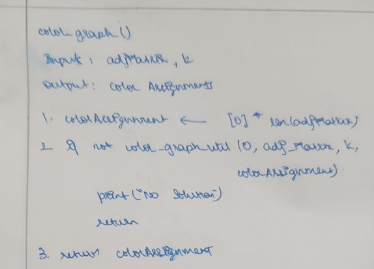
****

**Qn3:**

****

**Psuedo Code:**

****

****

**Source Code:**

def is\_safe(vertex, color, adj\_matrix, color\_assignment, k):

    for v in range(len(adj\_matrix)):

        if adj\_matrix[vertex][v] == 1 and color\_assignment[v] == color:

            return False

    return True

def color\_graph\_util(vertex, adj\_matrix, k, color\_assignment):

    if vertex == len(adj\_matrix):

        return True

    for color in range(1, k+1):

        if is\_safe(vertex, color, adj\_matrix, color\_assignment, k):

            color\_assignment[vertex] = color

            if color\_graph\_util(vertex+1, adj\_matrix, k, color\_assignment):

                return True

            color\_assignment[vertex] = 0

    return False

def color\_graph(adj\_matrix, k):

    color\_assignment = [0] \* len(adj\_matrix)

    if not color\_graph\_util(0, adj\_matrix, k, color\_assignment):

        print("No solution exists.")

        return

    print("Color assignments:")

    for vertex, color in enumerate(color\_assignment):

        print(f"Vertex {vertex}: Color {color}")

# Sample Input

V = 4

E = [(0, 1), (0, 2), (1, 2), (1, 3), (2, 3)]

k = 3

# Initialize adjacency matrix

adj\_matrix = [[0] \* V for \_ in range(V)]

for u, v in E:

    adj\_matrix[u][v] = 1

    adj\_matrix[v][u] = 1

print("Adjacency Matrix:")

print(\*adj\_matrix, sep="\n")

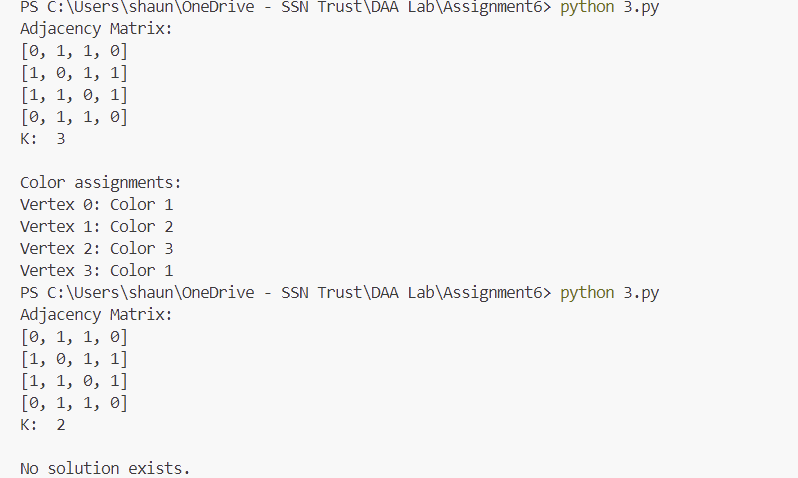
print("K: ", k)

print()

# Color the graph

color\_graph(adj\_matrix, k)

**Output:**

****

**Learning Outcomes:**

* I learnt to analyse and implement backtracking algorithms using Python.