Date: 18.03.24
ExNo: 6

Reg No: 3122225001127
Name: Shaun Allan H

ASSIGNMENT – 6

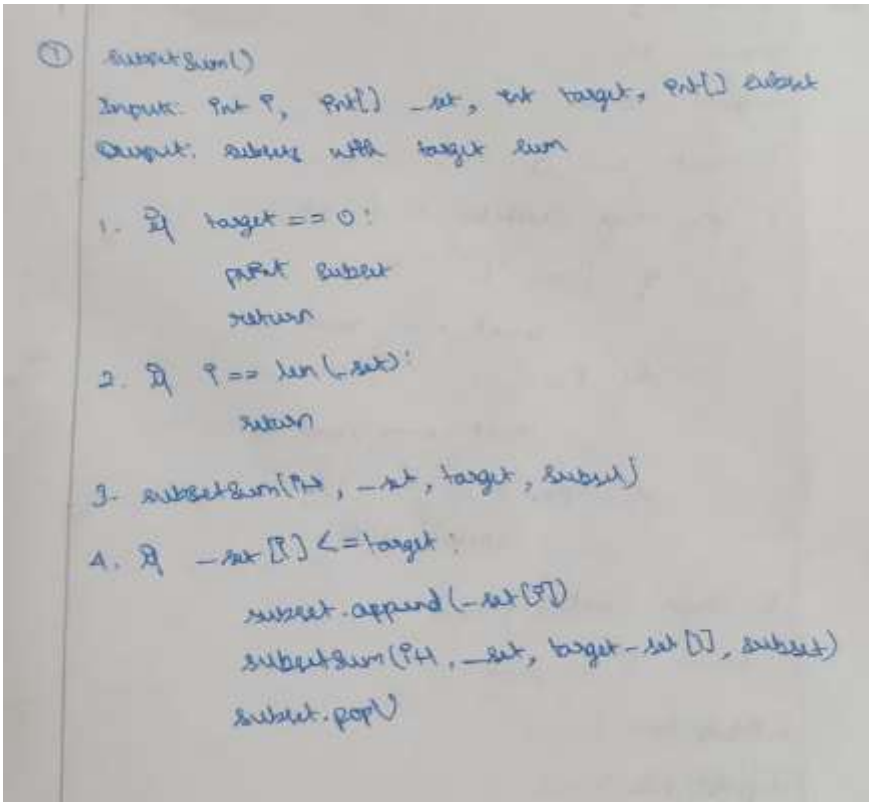
AIM:

To solve and implement the given problems using Backtracking.

Qn1:

1. Implement the backtracking algorithm for solving the Subset Sum problem.

Psuedo Code:



Date: 18.03.24
ExNo: 6

Reg No: 3122225001127
Name: Shaun Allan H

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:

```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment6> python 1.py
List:  [2, 3, 5, 6, 8, 10]
Enter sum: 10
[10], [2, 8], [2, 3, 5],
```

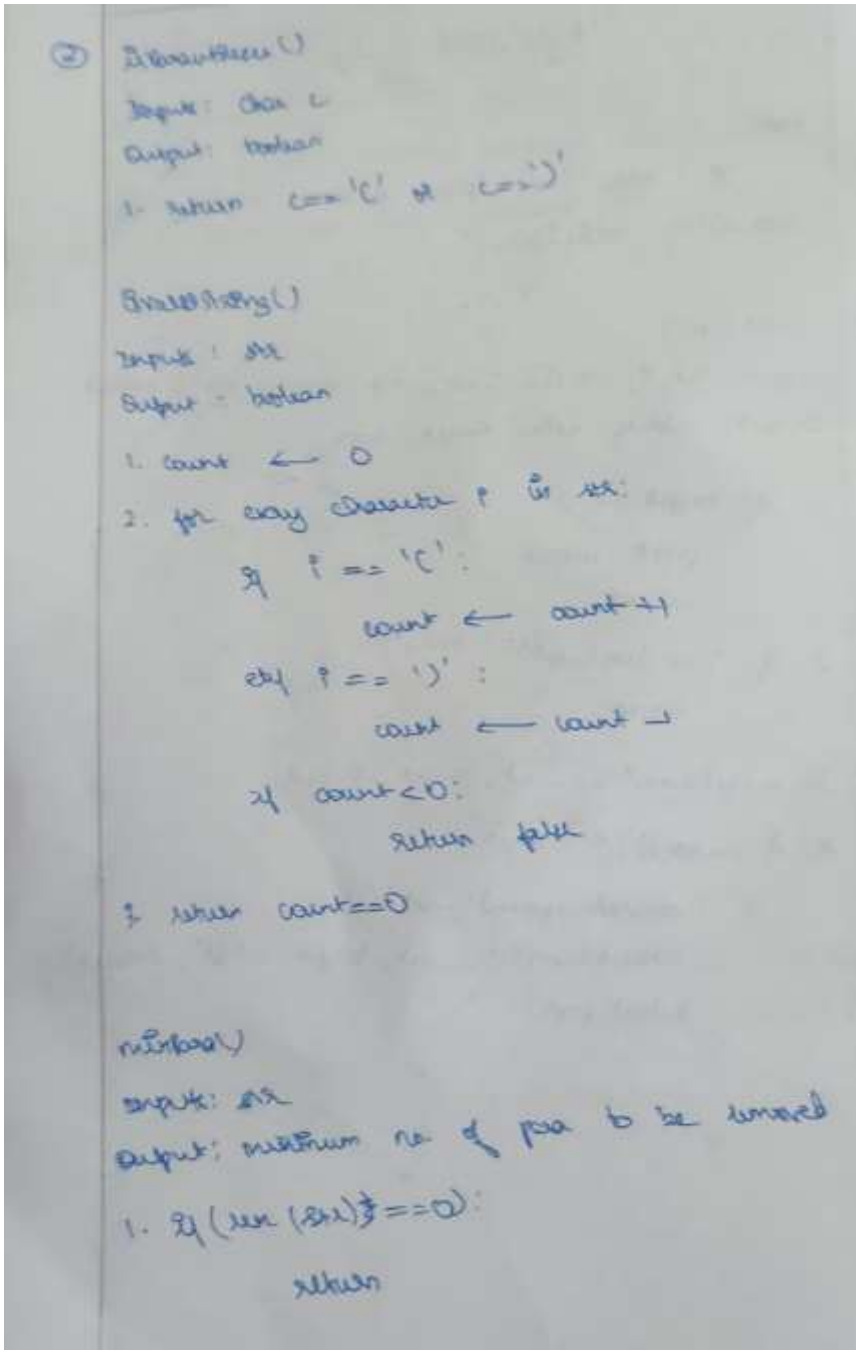
Date: 18.03.24
ExNo: 6

Reg No: 3122225001127
Name: Shaun Allan H

Qn2:

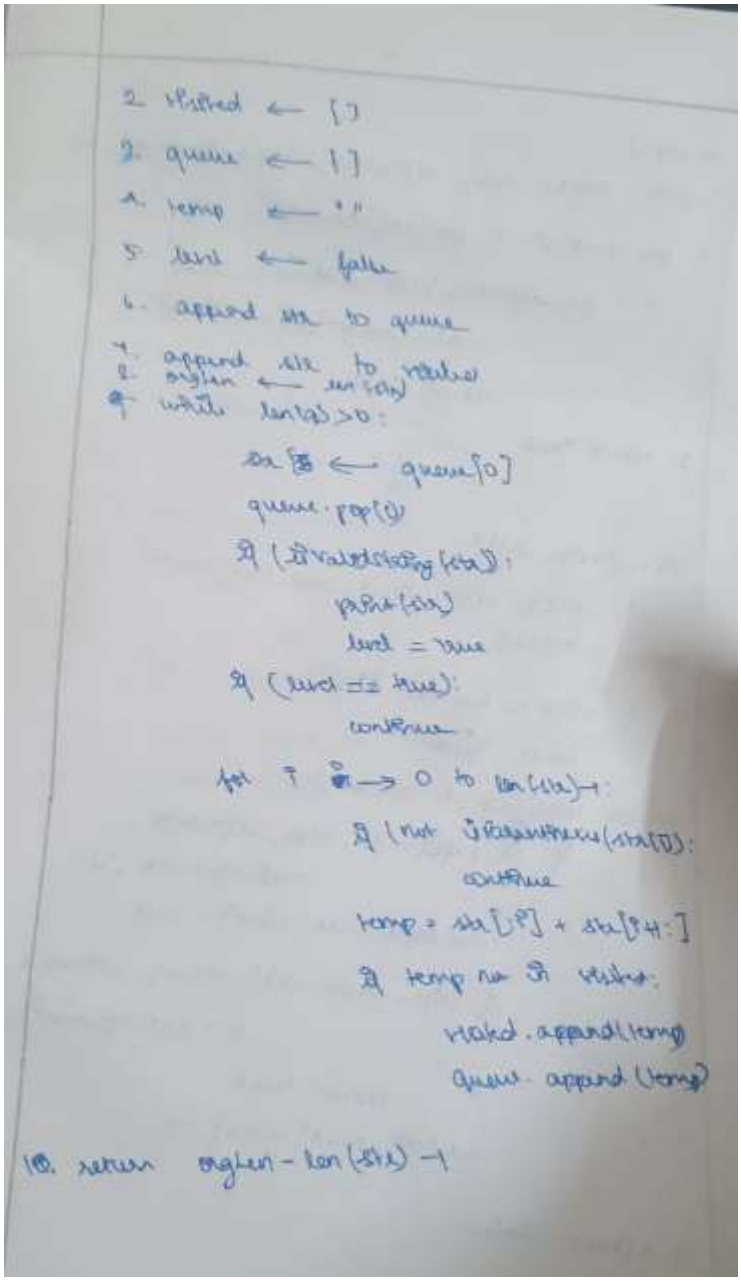
2. Given as input an expression that contains open and close parentheses and optionally some characters, write a backtracking algorithm to remove the minimum number of parentheses to make the input string valid. Implement the algorithm using Python.
Sample input = (())(()(()))(
Number of parentheses removed = 2
Output = (())()(())

Psuedo Code:



Date: 18.03.24
ExNo: 6

Reg No: 3122225001127
Name: Shaun Allan H



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}")
```

Date: 18.03.24
ExNo: 6

Reg No: 3122225001127
Name: Shaun Allan H

Output:

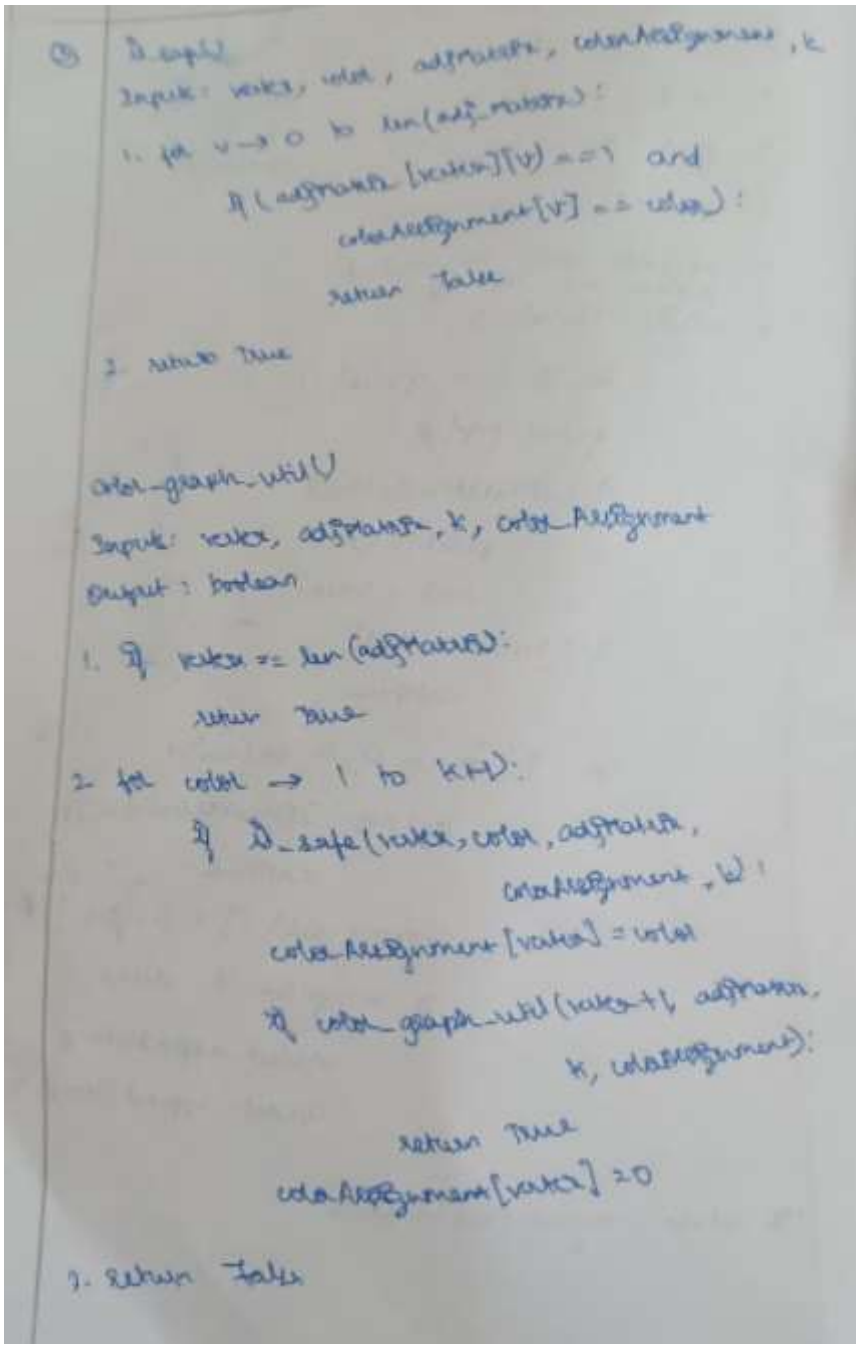
```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment6> python 2.py
String:  ()()))())
(()())
(())()
()(())
()()()
Min Para to be removed: 3

String:  ()v)
(v)
()v
Min Para to be removed: 1
```

Date: 18.03.24 Reg No: 3122225001127
ExNo: 6 Name: Shaun Allan H

Qn3:

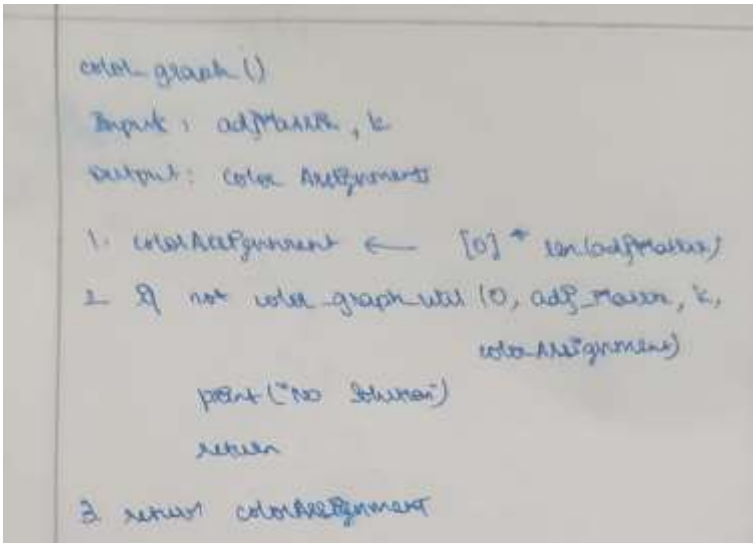
3. Given as input a graph $G = (V, E)$ and a number k, write a backtracking algorithm to colour the set V using at most k colours. If such a colouring is not possible, print "No solution exists.". Implement the algorithm using Python.

Psuedo Code:



Date: 18.03.24
ExNo: 6

Reg No: 3122225001127
Name: Shaun Allan H



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
```

Date: 18.03.24 Reg No: 3122225001127
ExNo: 6 Name: Shaun Allan H

```
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:

```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment6> python 3.py
Adjacency Matrix:
[0, 1, 1, 0]
[1, 0, 1, 1]
[1, 1, 0, 1]
[0, 1, 1, 0]
K:  3

Color assignments:
Vertex 0: Color 1
Vertex 1: Color 2
Vertex 2: Color 3
Vertex 3: Color 1
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment6> python 3.py
Adjacency Matrix:
[0, 1, 1, 0]
[1, 0, 1, 1]
[1, 1, 0, 1]
[0, 1, 1, 0]
K:  2

No solution exists.
```

Learning Outcomes:

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