Date: 11.04.2024
Ex. No: 6

Name: Mugilkrishna D U
Reg. No.: 3122225001073

UCS 2403 Design & Analysis of Algorithms

Assignment 6

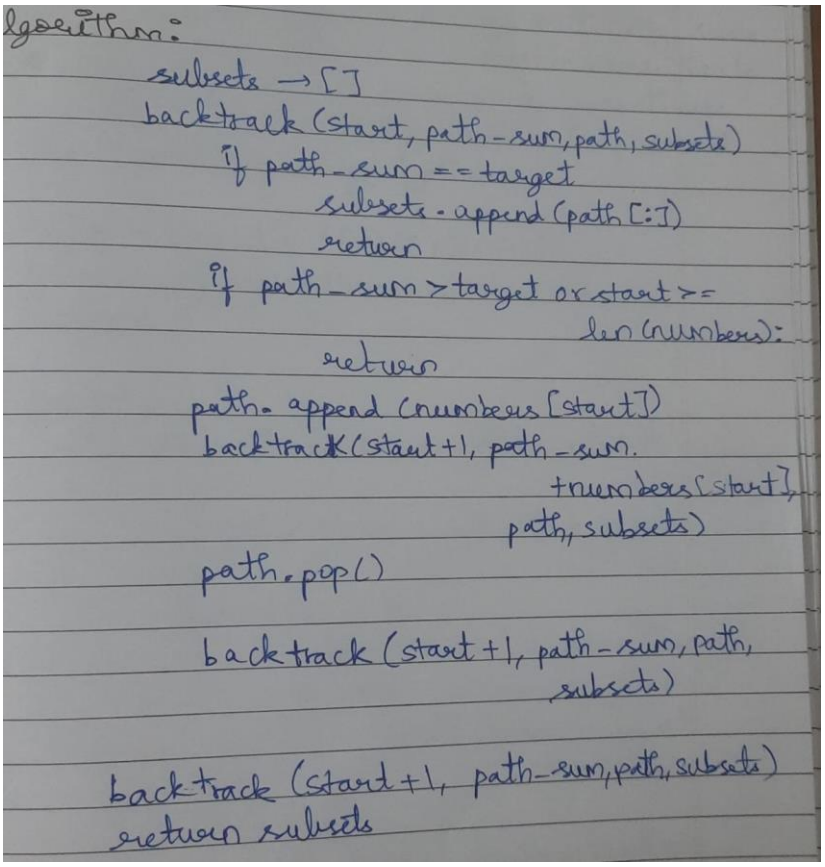
Date of Exercise: 11.04.2024

Aim: To gain understanding and proficiency on the backtracking algorithm

Question 1:

Implement the backtracking algorithm for solving the Subset Sum problem.

Algorithm:



Code:

```
def subset_sum_backtracking(numbers, target):
    subsets = []

    def backtrack(start, path_sum, path, subsets):
        if path_sum == target:
            subsets.append(path[:])  # found a valid subset
            return
```

```
        if path_sum > target or start >= len(numbers):
            return

        # Include the current element
        path.append(numbers[start])
        backtrack(start + 1, path_sum + numbers[start], path, subsets)
        path.pop()

        # Exclude the current element
        backtrack(start + 1, path_sum, path, subsets)

    backtrack(0, 0, [], subsets)
    return subsets

numbers = []
size = int(input("Enter the number of elements in the set: "))

for i in range(size):
    ele = int(input("Enter element " + str(i + 1) + ": "))
    numbers.append(ele)

target = int(input("\nEnter the target sum: "))

print("\nThe given set is: ", numbers)
print("Target: ", target)
print("\nSubsets:", subset_sum_backtracking(numbers, target))
```

Output:

```
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local
/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIG
N AND ANALYSIS OF ALGORITHMS/LAB/6.1.py"
Enter the number of elements in the set: 5
Enter element 1: 1
Enter element 2: 2
Enter element 3: 3
Enter element 4: 4
Enter element 5: 5

Enter the target sum: 6

The given set is:  [1, 2, 3, 4, 5]
Target:  6

Subsets: [[1, 2, 3], [1, 5], [2, 4]]
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local
/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIG
N AND ANALYSIS OF ALGORITHMS/LAB/6.1.py"
Enter the number of elements in the set: 6
Enter element 1: 23
Enter element 2: 45
Enter element 3: 6
Enter element 4: 7
```

```
Enter element 5: 33
Enter element 6: 40

Enter the target sum: 46

The given set is:  [23, 45, 6, 7, 33, 40]
Target:  46

Subsets: [[6, 7, 33], [6, 40]]
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local
/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIG
N AND ANALYSIS OF ALGORITHMS/LAB/6.1.py"
Enter the number of elements in the set: 3
Enter element 1: 1
Enter element 2: 2
Enter element 3: 3

Enter the target sum: 8

The given set is:  [1, 2, 3]
Target:  8

Subsets: []
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode>
```

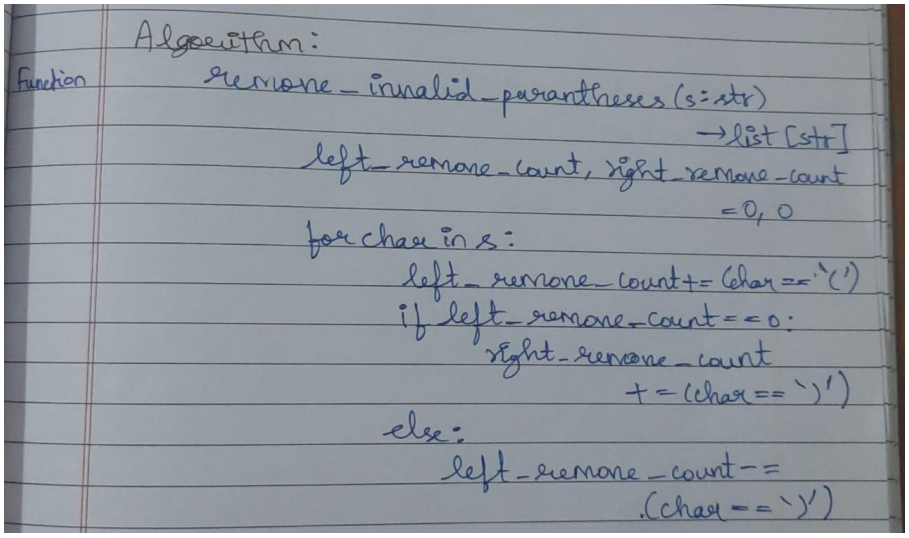
Time Complexity:

The Time Complexity of the subset sum backtracking algorithm is O(2^n)

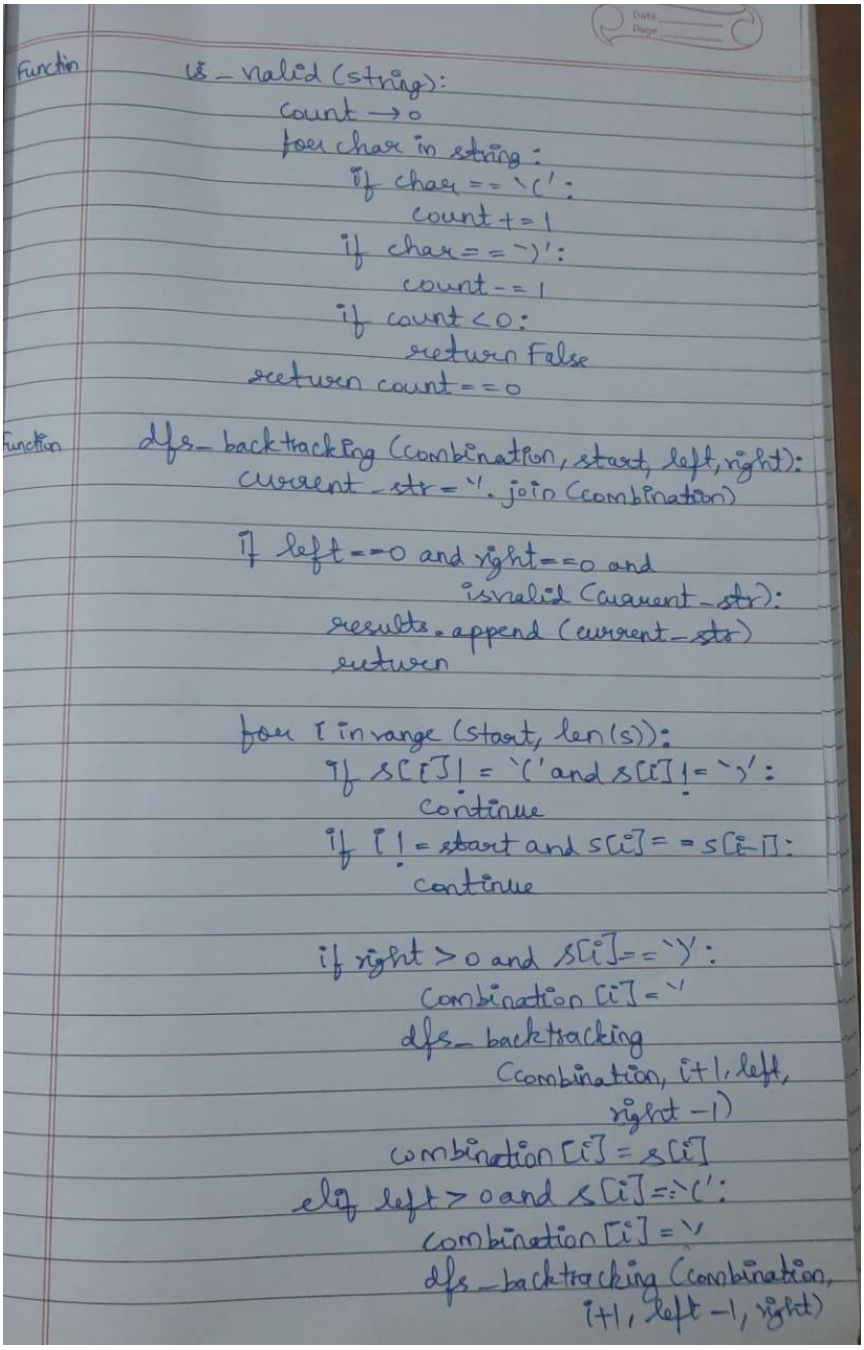
Question 2:

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 = (())()(()

Algorithm:



Name: Mugilkrishna D U
Reg. No.: 3122225001073



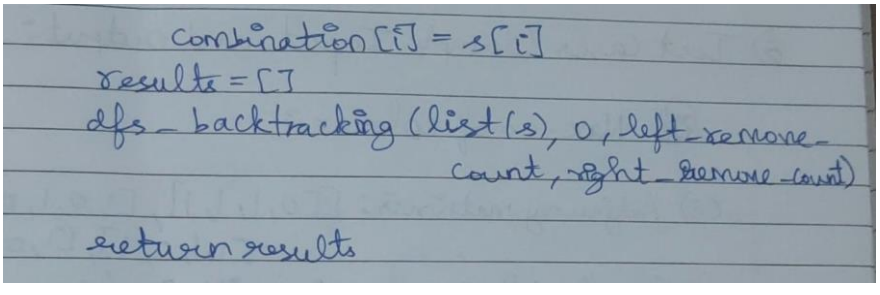
```
Function is_valid(string):
    count → 0
    for char in string:
        if char == '(':
            count += 1
        if char == ')':
            count -= 1
        if count < 0:
            return False
    return count == 0

Function dfs_backtracking(combination, start, left, right):
    current_str = ''.join(combination)

    if left == 0 and right == 0 and
                    isvalid(current_str):
        results.append(current_str)
        return

    for i in range(start, len(s)):
        if s[i] != '(' and s[i] != ')':
            continue
        if i != start and s[i] == s[i-1]:
            continue

        if right > 0 and s[i] == ')':
            combination[i] = ''
            dfs_backtracking
                (combination, i+1, left,
                    right - 1)
            combination[i] = s[i]
        elif left > 0 and s[i] == '(':
            combination[i] = ''
            dfs_backtracking(combination,
                    i+1, left - 1, right)
```



```
combination[i] = s[i]
results = []
dfs_backtracking(list(s), 0, left_remove_
                    count, right_remove_count)

return results
```

Name: Mugilkrishna D U
Reg. No.: 3122225001073

Code:

```
def remove_invalid_parentheses(s: str) -> list[str]:
    left_remove_count, right_remove_count = 0, 0
    for char in s:
        left_remove_count += (char == '(')
        if left_remove_count == 0:
            right_remove_count += (char == ')')
        else:
            left_remove_count -= (char == ')')

    # Function to check if the string is in a valid form
    def is_valid(string):
        count = 0
        for char in string:
            if char == '(':
                count += 1
            if char == ')':
                count -= 1
            if count < 0:
                return False
        return count == 0

    # Backtracking DFS function
    def dfs_backtracking(combination, start, left, right):
        current_str = ''.join(combination)

        if left == 0 and right == 0 and is_valid(current_str):
            results.append(current_str)
            return

        for i in range(start, len(s)):
            if s[i] != '(' and s[i] != ')':
                continue
            if i != start and s[i] == s[i - 1]:
                continue

            # Remove invalid ")" before "("
            if right > 0 and s[i] == ')':
                combination[i] = ''
                dfs_backtracking(combination, i + 1, left, right - 1)
                combination[i] = s[i]
            elif left > 0 and s[i] == '(':
                combination[i] = ''
                dfs_backtracking(combination, i + 1, left - 1, right)
                combination[i] = s[i]
```

```
    results = []
    dfs_backtracking(list(s), 0, left_remove_count, right_remove_count)

    return results

s = input("Enter a string with parentheses: ")
result = remove_invalid_parentheses(s)
if result:
    print("Valid combinations after removing invalid parentheses:")
    for res in result:
        print(res)
else:
    print("No valid combinations exist after removing invalid parentheses.")
```

Output:

```
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local
/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIG
N AND ANALYSIS OF ALGORITHMS/LAB/6.2.py"
Enter a string with parentheses: ((())(()(()))(
Valid combinations after removing invalid parentheses:
((())(()(()))
((())(()(()))
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local
/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIG
N AND ANALYSIS OF ALGORITHMS/LAB/6.2.py"
Enter a string with parentheses: (()
Valid combinations after removing invalid parentheses:
()
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local
/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIG
N AND ANALYSIS OF ALGORITHMS/LAB/6.2.py"
Enter a string with parentheses: ((()))()
Valid combinations after removing invalid parentheses:
((()))()
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode>
```

Time Complexity:

The Time Complexity of the backtracking DFS function is O(2^n) as it explores all possible comniantions of removing the parentheses.

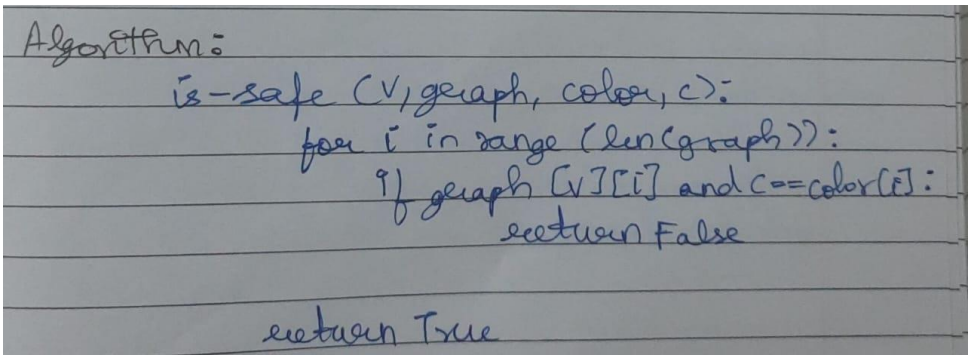
Question 3:

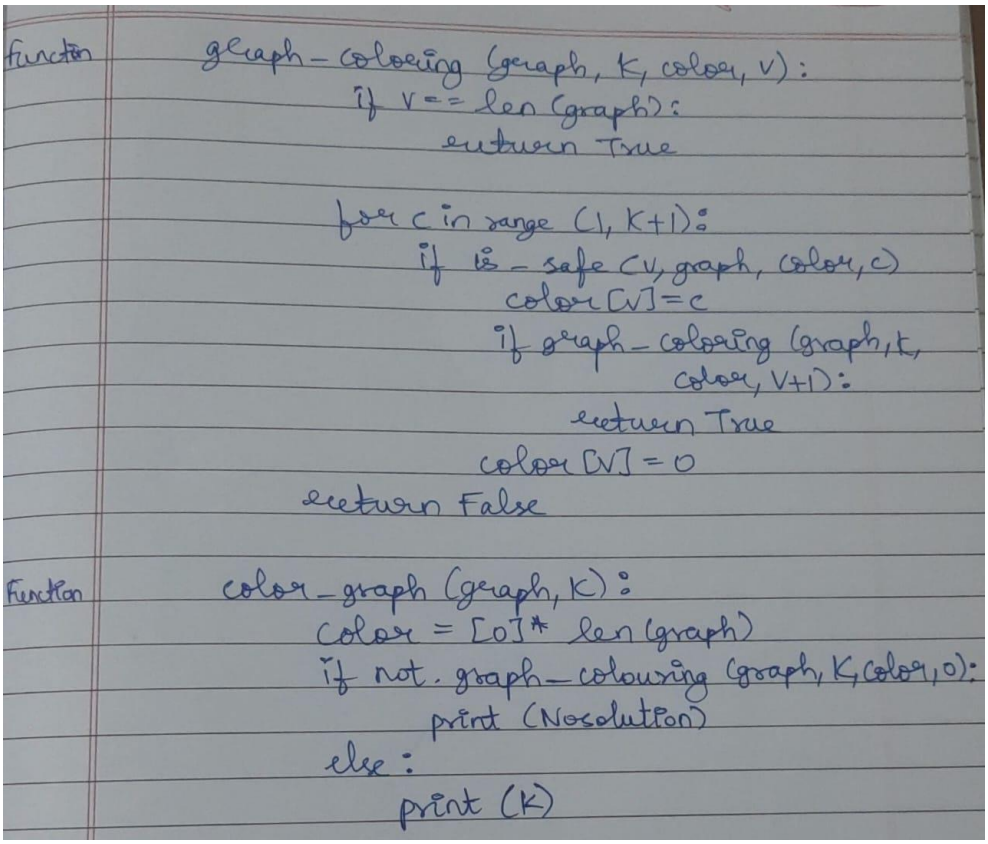
Given as input a graph G = (V, E) and a number k, write a backtracking algorithm to colour the set V using at most k colours. If such a colouring is not possible, print “No solution exists.”. Implement the algorithm using Python.

Date: 11.04.2024
Ex. No: 6

Name: Mugilkrishna D U
Reg. No.: 3122225001073

Algorithm:





Code:

```
def is_safe(v, graph, color, c):
    for i in range(len(graph)):
        if graph[v][i] and c == color[i]:
            return False
    return True

def graph_coloring(graph, k, color, v):
    if v == len(graph):
        return True
```

Name: Mugilkrishna D U
Reg. No.: 3122225001073

```
    for c in range(1, k + 1):
        if is_safe(v, graph, color, c):
            color[v] = c
            if graph_coloring(graph, k, color, v + 1):
                return True
            color[v] = 0
    return False

def color_graph(graph, k):
    color = [0] * len(graph)
    if not graph_coloring(graph, k, color, 0):
        print("No solution exists.")
    else:
        print("\nGraph coloring possible with at most", k, "colors.")
        print("Coloring:", color)


graph = [
    [0, 1, 1, 1],
    [1, 0, 1, 0],
    [1, 1, 0, 1],
    [1, 0, 1, 0]
]
k = 3  # Number of colors

print("Adjacency matrix is: ", graph)
print("The maximum number of colours is: ", k)

color_graph(graph, k)
```

Output:

```
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local
/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIG
N AND ANALYSIS OF ALGORITHMS/LAB/6.3.py"
The maximum number of colours is:  3

Adjacency matrix is:  [[0, 1, 1, 1], [1, 0, 1, 0], [1, 1, 0, 1], [1, 0, 1, 0]]
Graph coloring possible with at most 3 colors.
Coloring: [1, 2, 3, 2]

Adjacency matrix is:  [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1], [1, 0, 1, 0]]
Graph coloring possible with at most 3 colors.
Coloring: [1, 2, 1, 2]

Adjacency matrix is:  [[0, 1, 1, 1], [1, 0, 1, 1], [1, 1, 0, 1], [1, 1, 1, 0]]
No solution exists.
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode>
```

Time Complexity:

The Time Complexity of the graph coloring algorithm is O(k^n) where k is the maximum number of colors that can be used.

Learning Outcome:

Upon completing this exercise, I have understood the applications of backtracking algorithm in the subset sum problem and the valid parentheses problem as well as to find the solution for the graph colouring problem