LAB-11

1. subsets generation:

Coding:

```
1 usage
def backtracking_sum_of_subsets(nums):
    def backtrack(index, current_subset, current_sum):
        nonlocal total_sum
        if index == len(nums):
            print(f"Subset: {current_subset}, Sum: {current_sum}")
            total_sum += current_sum
            return
        backtrack(index + 1, current_subset + [nums[index]], current_sum + nums[index])
        backtrack(index + 1, current_subset, current_sum)
    total_sum = 0
    backtrack( index: 0,  current_subset: [],  current_sum: 0)
    print(f"Total sum of all subset sums: {total_sum}")
    return total_sum
nums = [1, 2, 3]
backtracking_sum_of_subsets(nums)
```

Output:

```
Subset: [1, 2, 3], Sum: 6
Subset: [1, 2], Sum: 3
Subset: [1, 3], Sum: 4
Subset: [1], Sum: 1
Subset: [2, 3], Sum: 5
Subset: [2], Sum: 2
Subset: [3], Sum: 3
Subset: [], Sum: 0
Total sum of all subset sums: 24
```

2.Longest palindromic subsequence:

Coding:

```
1 usage
def longest_palindromic_subsequence(s):
    n = len(s)
    dp = [[0] * n for _ in range(n)]
    sequences = [["" for _ in range(n)] for _ in range(n)]
    for i in range(n):
        dp[i][i] = 1
        sequences[i][i] = s[i]
    max_length = 1
    max_sequence = s[0]
    for length in range(2, n + 1):
        for i in range(n - length + 1):
            j = i + length - 1
            if s[i] == s[j]:
                if length == 2:
                    dp[i][j] = 2
                    sequences[i][j] = s[i] + s[j]
```

```
                dp[i][j] = 2
                sequences[i][j] = s[i] + s[j]
            else:
                dp[i][j] = dp[i + 1][j - 1] + 2
                sequences[i][j] = s[i] + sequences[i + 1][j - 1] + s[j]
            if dp[i][j] > max_length:
                max_length = dp[i][j]
                max_sequence = sequences[i][j]
        else:
            if dp[i + 1][j] > dp[i][j - 1]:
                dp[i][j] = dp[i + 1][j]
                sequences[i][j] = sequences[i + 1][j]
            else:
                dp[i][j] = dp[i][j - 1]
                sequences[i][j] = sequences[i][j - 1]
    return max_sequence
s = "banana"
```

```
s = "banana"
print(longest_palindromic_subsequence(s))
```

Output:

```
anana

Process finished with exit code 0
```

3.subsets generation:

Coding:

```
def generate_subsets(nums):
    def backtrack(start, path):
        result.append(path)
        for i in range(start, len(nums)):
            backtrack(i + 1, path + [nums[i]])
    result = []
    backtrack( start: 0,  path: [])
    return result
nums = [1, 2, 3]
print(generate_subsets(nums))
```

Output:

```
[[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3], [3]]

Process finished with exit code 0
```

4.GRAPH COLOURING:

CODING:

```python
def graph_coloring(graph, max_colors):
    def is_safe(v, color, colors):
        for i in range(len(graph)):
            if graph[v][i] == 1 and color == colors[i]:
                return False
        return True
    def backtrack(v, colors):
        if v == len(graph):
            print(colors[:])
            return
        for color in range(1, max_colors + 1):
            if is_safe(v, color, colors):
                colors[v] = color
                backtrack(v + 1, colors)
                colors[v] = 0
    colors = [0] * len(graph)
    backtrack( v: 0, colors)
graph = [[0, 1, 1, 1],
         [1, 0, 1, 0],
         [1, 1, 0, 1],
         [1, 0, 1, 0]]
max_colors = 3
graph_coloring(graph, max_colors)

```

OUTPUT:

```
C:\Users\vinot\PycharmProjects\pythonProje
[1, 2, 3, 2]
[1, 3, 2, 3]
[2, 1, 3, 1]
[2, 3, 1, 3]
[3, 1, 2, 1]
[3, 2, 1, 2]

Process finished with exit code 0
```