In [ ]:
```
sum of subsets
```

In [1]:
```
def subset(nums, target):
    def backtrack(start, path, summ):
        if summ == target:
            res.append(path[:])
            return
        if summ > target:
            return
        for i in range(start, len(nums)):
            path.append(nums[i])
            backtrack(i + 1, path,summ + nums[i])
            path.pop()
    res = []
    backtrack(0, [], 0)
    return res
nums = [6, 4, 8,2]
target = 10
res = subset(nums, target)
print("Subsets with sum", target, "are:", res)
```

```
Subsets with sum 10 are: [[6, 4], [8, 2]]
```

In [ ]:
```
graph colouring
```

In [2]:
```
def is_safe(graph, colors, v, c):
    for i in range(len(graph)):
        if graph[v][i] == 1 and colors[i] == c:
            return False
    return True
def graph_coloring_util(graph, m, colors, v, num_vertices, all_colorings):
    if v == num_vertices:
        all_colorings.append(colors[:])
        return
    for c in range(1, m + 1):
        if is_safe(graph, colors, v, c):
            colors[v] = c
            graph_coloring_util(graph, m, colors, v + 1, num_vertices, all_c
            colors[v] = 0
def graph_coloring(graph, m):
    num_vertices = len(graph)
    colors = [0] * num_vertices
    all_colorings = []
    graph_coloring_util(graph, m, colors, 0, num_vertices, all_colorings)
    unique_colorings = []
    for coloring in all_colorings:
        if coloring not in unique_colorings:
            unique_colorings.append(coloring)
    if unique_colorings[0] == unique_colorings[-1]:
        return unique_colorings[:-1]
    return unique_colorings
graph = [
    [0, 1, 1, 0],
    [1, 0, 0, 0],
    [1, 0, 0, 1],
    [0, 0, 1, 0]
]
m = 3
unique_colorings = graph_coloring(graph, m)
print("All unique colorings:")
for coloring in unique_colorings:
    print(coloring)
```

```
All unique colorings:
[1, 2, 2, 1]
[1, 2, 2, 3]
[1, 2, 3, 1]
[1, 2, 3, 2]
[1, 3, 2, 1]
[1, 3, 2, 3]
[1, 3, 3, 1]
[1, 3, 3, 2]
[2, 1, 1, 2]
[2, 1, 1, 3]
[2, 1, 3, 1]
[2, 1, 3, 2]
[2, 3, 1, 2]
[2, 3, 1, 3]
[2, 3, 3, 1]
[2, 3, 3, 2]
[3, 1, 1, 2]
[3, 1, 1, 3]
[3, 1, 2, 1]
[3, 1, 2, 3]
[3, 2, 1, 2]
[3, 2, 1, 3]
[3, 2, 2, 1]
[3, 2, 2, 3]
```

In [3]:
```
import sys
def solve_word_wrap(words, max_length):
    n = len(words)
    extras = [[0] * n for _ in range(n)]

    for i in range(n):
        extras[i][i] = max_length - len(words[i])
        for j in range(i + 1, n):
            extras[i][j] = extras[i][j - 1] - len(words[j]) - 1
    line_cost = [[0] * n for _ in range(n)]

    for i in range(n):
        for j in range(i, n):
            if extras[i][j] < 0:
                line_cost[i][j] = sys.maxsize
            elif j == n - 1:
                line_cost[i][j] = 0
            else:
                line_cost[i][j] = extras[i][j] ** 2
    total_cost = [0] * n
    p = [0] * n
    for j in range(n):
        total_cost[j] = sys.maxsize
        for i in range(j + 1):
            if total_cost[i - 1] + line_cost[i][j] < total_cost[j]:
                total_cost[j] = total_cost[i - 1] + line_cost[i][j]
                p[j] = i
    lines = []
    i = n
    while i > 0:
        start = p[i - 1]
        lines.append(" ".join(words[start:i]))
        i = start

    lines.reverse()
    return lines, total_cost, p, extras, line_cost
text = "Saveetha School of Engineering"
words = text.split()
max_length = 11
wrapped_text, total_cost, p, extras, line_cost = solve_word_wrap(words, max_

print("Wrapped Text:")
for line in wrapped_text:
    print(line)
print("\nTotal Cost Array:")
print(total_cost)
print("\nPosition Array:")
print(p)
print("\nExtras Array:")
for row in extras:
    print(row)
print("\nLine Cost Array:")
for row in line_cost:
    print(row)
```

```
Wrapped Text:
Saveetha
School of
Engineering

Total Cost Array:
[9, 34, 13, 13]

Position Array:
[0, 1, 1, 3]

Extras Array:
[3, -4, -7, -19]
[0, 5, 2, -10]
[0, 0, 9, -3]
[0, 0, 0, 0]

Line Cost Array:
[9, 9223372036854775807, 9223372036854775807, 9223372036854775807]
[0, 25, 4, 9223372036854775807]
[0, 0, 81, 9223372036854775807]
[0, 0, 0, 0]
```

In [4]:
```
def longest_palindrome(s):
    n = len(s)
    if n == 0:
        return ""
    dp = [[0] * n for _ in range(n)]

    start = 0
    max_length = 1

    for i in range(n):
        dp[i][i] = 1

    for i in range(n - 1):
        if s[i] == s[i + 1]:
            dp[i][i + 1] = 2
            start = i
            max_length = 2

    for length in range(3, n + 1):
        for i in range(n - length + 1):
            j = i + length - 1
            if s[i] == s[j] and dp[i + 1][j - 1] > 0:
                dp[i][j] = dp[i + 1][j - 1] + 2
                if dp[i][j] > max_length:
                    max_length = dp[i][j]
                    start = i

    for i in range(n):
        for j in range(i + 1, n):
            dp[i][j] = max(dp[i][j], dp[i + 1][j], dp[i][j - 1])

    print("DP Table:")
    for row in dp:
        print(row)

    return s[start:start + max_length]

s = "teeth"
print("Input string:", s)
print("The longest palindrome substring is:", longest_palindrome(s))
```

```
Input string: teeth
DP Table:
[1, 1, 2, 4, 4]
[0, 1, 2, 2, 2]
[0, 0, 1, 1, 1]
[0, 0, 0, 1, 1]
[0, 0, 0, 0, 1]
The longest palindrome substring is: teet
```

In [ ]: