ASSIGNMENT 24/06/2024

1 . Apply Prim's algorithm to solve the minimum spanning tree for the given graph. Also compute the total cost of all edges.

CODE :

```
import sys
V = 5
def minKey(key, mstSet):
    # Initialize min value
    min = sys.maxsize
    min_index = -1
    for v in range(V):
        if key[v] < min and not mstSet[v]:
            min = key[v]
            min_index = v
    return min_index
def printMST(parent, graph):
    print("Edge \tWeight")
    total_cost = 0
    for i in range(1, V):
        print(f"{parent[i]} - {i} \t{graph[i][parent[i]]}")
        total_cost += graph[i][parent[i]]
    print("Total cost of all edges in the MST:", total_cost)
def primMST(graph):
    key = [sys.maxsize] * V
    parent = [None] * V  # Array to store constructed MST
    key[0] = 0  # Make key 0 so that this vertex is picked as first vertex
    mstSet = [False] *
    parent[0] = -1  # First node is always the root of MST
    for cout in range(V):
            u = minKey(key, mstSet)
        mstSet[u] = True
        for v in range(V):
            if graph[u][v] > 0 and not mstSet[v] and key[v] > graph[u][v]:
                key[v] = graph[u][v]
                parent[v] = u

    printMST(parent, graph)
graph = [
    [0, 2, 0, 6, 0],
    [2, 0, 3, 8, 5],
    [0, 3, 0, 0, 7],
    [6, 8, 0, 0, 9],
    [0, 5, 7, 9, 0]
]

primMST(graph)
```

OUTPUT :

```
Edge        Weight
0 - 1       2
1 - 2       3
0 - 3       6
1 - 4       5
Total cost of all edges in the MST: 16
```

2 . To Compute the sum of Subsets for the following graph and then satisfy the given constraints.
Set S { } =(a, b, c, d, e, f, g, h, i) Values used are V{i}=(1, 2, 3…….9)

Used all values Only one time
Constraints hold such as
a + b + c = c +d + e = e + f + g = g + h + i

CODE :

```
from itertools import permutations
values = [1, 2, 3, 4, 5, 6, 7, 8, 9]
def satisfies_constraint(set_of_nums):
    a, b, c, d, e, f, g, h, i = set_of_nums
    return (a + b + c == c + d + e == e + f + g == g + h + i)
perms = permutations(values)
for perm in perms:
    if satisfies_constraint(perm):
        print("Found subset that satisfies the constraint:")
        print(perm)
        break
else:
    print("No subset found that satisfies the constraint.")
```

OUTPUT :

```
Found subset that satisfies the constraint:
(1, 7, 6, 5, 3, 9, 2, 4, 8)
```

3 . Calculate the chromatic no for the following Graph coloring.
CODE :

```
def calculate_chromatic_number(graph):
     colors = {}
    chromatic_number = 0
```

```
        def assign_color(vertex, color):
            nonlocal chromatic_number
            colors[vertex] = color
            chromatic_number = max(chromatic_number, color)
            for vertex in sorted(graph.keys()):
                used_colors = set(colors.get(neighbour, None) for neighbour in graph[vertex])
            for color in range(1, len(graph) + 1):
                if color not in used_colors:
                    assign_color(vertex, color)
                    break

    return chromatic_number
graph = {
    1: [2, 3, 4],
    2: [1, 3],
    3: [1, 2, 4],
    4: [1, 3],
    5: []
}

chromatic_number = calculate_chromatic_number(graph)
print(f"The chromatic number of the graph is: {chromatic_number}")
```

OUTPUT :

```
The chromatic number of the graph is: 3
```

4 . Consider a set S = (5, 10, 12, 13, 15, 18) and d=30. Solve it for obtaining a sum of subset.

CODE :

```
def find_subset_sum(S, target):
        def find_subset_recursive(current_index, current_subset, current_sum):
        if current_sum == target:
            subsets.append(current_subset[:])  # Found a valid subset
            return
        if current_sum > target or current_index >= len(S):
            return

        current_subset.append(S[current_index])
        find_subset_recursive(current_index + 1, current_subset, current_sum +
S[current_index])
            current_subset.pop()
        find_subset_recursive(current_index + 1, current_subset, current_sum)

    subsets = []
    find_subset_recursive(0, [], 0)
```

```
    return subsets
S = [5, 10, 12, 13, 15, 18]
target = 30
subsets = find_subse
if subsets:
    print(f"Subsets that sum up to {target}:")
    for subset in subsets:
        print(subset)
else:
    print(f"No subsets found that sum up to {target}.")
```

OUTPUT :