ASSIGNMENT(24-06-24)

1.PRIMS ALGORITHM:

```
import heapq


def prim(graph, start):
    mst = []
    visited = set()
    minheap = [(0, start, None)]
    totalcost = 0

    while minheap:
        cost, vertex, fromvertex = heapq.heappop(minheap)
        if vertex in visited:
            continue
        visited.add(vertex)
        if fromvertex is not None:
            mst.append((fromvertex, vertex, cost))
            totalcost += cost

        for neighbor, weight in graph[vertex]:
            if neighbor not in visited:
                heapq.heappush(minheap, (weight, neighbor, vertex))

    return mst, totalcost

graph = {
    'A': [('B', 1), ('C', 3)],
    'B': [('A', 1), ('C', 3), ('D', 6)],
    'C': [('A', 3), ('B', 3), ('D', 4), ('E', 2)],
    'D': [('B', 6), ('C', 4), ('E', 5)],
    'E': [('C', 2), ('D', 5)]
```

```
}

mst, totalcost = prim(graph, 'A')
print("Edges in MST:", mst)
print("Total cost:", totalcost)
```

2.SUM OF SUBSETS:

```
def valid(solution):
    a, b, c, d, e, f, g, h, i = solution
    return (a + b + c == c + d + e == e + f + g == g + h + i)

def findsolution():
    values = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    from itertools import permutations
    for perm in permutations(values):
        if valid(perm):
            return perm
    return None

solution = findsolution()
if solution:
    print("Solution found:", solution)
else:
    print("No solution found")
```

3.CHROMATIC NUMBER:

```
def greedy(graph):
    colors = {}
    for vertex in graph:
        avcolors = set(range(len(graph)))
        for neighbor in graph[vertex]:
```

```
            if neighbor in colors:
                avcolors.discard(colors[neighbor])
        colors[vertex] = min(avcolors)
    return colors

graph = {
    'A': ['B', 'C'],
    'B': ['A', 'C', 'D'],
    'C': ['A', 'B', 'D'],
    'D': ['B', 'C']
}

colors = greedy (graph)
res= max(colors.values()) + 1
print("Colors assigned:", colors)
print("Chromatic number:", res)
```

4.SUM OF SUBSET:

```
def subset(nums, target):
    def backtrack(start, path, target):
        if target == 0:
            result.append(path)
            return
        for i in range(start, len(nums)):
            if nums[i] > target:
                continue
            backtrack(i + 1, path + [nums[i]], target - nums[i])

    result = []
    nums.sort()
    backtrack(0, [], target)
```

```
    return result


S = [5, 10, 12, 13, 15, 18]
d = 30
res= subset(S, d)
print("Subsets that sum to", d, ":", res)
```