6.11 GRAPH COLORING WITH MAXIMUM REGIONS COLORED

Question:

You and your friends are assigned the task of coloring a map with a limited number of colors. The map is represented as a list of regions and their adjacency relationships. The rules are as follows: At each step, you can choose any uncolored region and color it with any available color. Your friend Alice follows the same strategy immediately after you, and then your friend Bob follows suit. You want to maximize the number of regions you personally color. Write a function that takes the map's adjacency list representation and returns the maximum number of regions you can color before all regions are colored. Write a program to implement the Graph coloring technique for an undirected graph. Implement an algorithm with minimum number of colors. edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)] No. of vertices, n = 4.

AIM

To implement the Graph Coloring algorithm for an undirected graph with the minimum number of colors and determine the maximum number of regions you can color under turn-based coloring rules

ALGORITHM

1. Graph Representation: Represent the map using an adjacency list.
2. Graph Coloring (Greedy):
3. Assign the smallest possible color to each vertex such that no two adjacent vertices share the same color.
4. Continue until all vertices are colored.
5. Turn-based Strategy:
6. Since there are n regions and 3 players (You, Alice, Bob), you will get ⌈n/3⌉ turns if you pick optimally.
7. The maximum regions you can color = (n + 2) // 3.
8. Return Result: Output both the colored graph and the maximum number of regions you can personally color.

PROGRAM

```
def max_regions_colored(n, edges, k):
    from collections import defaultdict
    graph = defaultdict(list)
    for u, v in edges:
        graph[u].append(v)
        graph[v].append(u)

    color = [-1] * n
    turn = 0
    count = 0

    def available_colors(v):
        used = set(color[u] for u in graph[v] if color[u] != -1)
        return [c for c in range(k) if c not in used]

    while -1 in color:
        for i in range(n):
            if color[i] == -1:
                options = available_colors(i)
                if options:
                    color[i] = options[0]
                    if turn % 3 == 0:
                        count += 1
                    turn += 1
                    break
    return count

n = int(input("Enter number of vertices: "))
m = int(input("Enter number of edges: "))
edges = []
for _ in range(m):
    u, v = map(int, input("Edge: ").split())
    edges.append((u, v))
k = int(input("Enter number of colors: "))
print("Maximum regions you can color:", max_regions_colored(n, edges, k))
```

Input:

n = 4

edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]

k = 3

Output:

```
Enter number of vertices: 4
Enter number of edges: 5
Edge: 0 1
Edge: 1 2
Edge: 2 3
Edge: 3 0
Edge: 0 2
Enter number of colors: 3
Maximum regions you can color: 2
>>>
```

RESULT:

Thus, the program is successfully executed and verified to generate all unique permutations of the array.

PERFORMANCE ANALYSIS:

Time Complexity: O(n * n!) in the worst case where n is the number of elements.

Space Complexity: O(n) for recursion depth and path storage, plus O(n) for the 'used' array.