ESO207 Programming Assignment 3

Arpit Kumar, Roll no.: 200189

Kumar Arpit, Roll no.: 200532

1. **Pseudo Code**

```
fun Bipartite(g)                                        //'g' is the input graph
{
    color[V]                                            //Creating an array to store the color of a vertex which is either 0 or 1
    for i from 0 to V-1                                 //Initializing the color of each vertex to -1
        color[i]=-1
    if(modifiedDFS(g,color,0,0)==true)                  //Calls the subroutine modifiedDFS from vertex 0 and the color to be assigned to it as 0
        V1[V], V2[V]                                    //Creating 2 arrays to store the partitions
        for i from 0 to g->V-1                          //Looping through the array to add a vertex to either V1 or V2
            if(color[i]==0)
                V1.append(i)
            else
                V2.append(i)
        return (V1,V2)
    else
        return false                                    //Returning false if g is not bipartite
}

fun modifiedDFS(g,color[],vertex,c)                     //A modified DFS routine. Note that color kind of plays the role of mark[] as well.
{
    color[vertex]=c
    temp=g->adjLists[vertex]                            //Creating a copy of the pointer at linked list of the vertex 'vertex' in the graph g
    while(temp!=NULL)
        if(color[temp->vertex]==-1)                     //If -1 then the node has not been visited
            if(dfs(g,color,temp->vertex,(c+1)%2)==0)    //Call the function recursively with the other color
                return false                            //If the result of call was zero then it wasn't possible to color the graph with two colors. Thus the sequence of return 0 is passed down the recursion and finally to Bipartite.
        else if(color[temp->vertex]==c)
            return false                                //If color same as its parent then 0 is returned instantly.
        temp=temp->next                                 //Proceeds to the next neighboring vertex of temp
```

```
    return true                                        //If it comes out of the while loop without returning zero then the subgraph can be successfully colored using two colors and true is returned.
}


//Structures used:
struct node
{
    long long int vertex;
    struct node* next;
};
struct Graph
{
    long long int V;
    struct node** adjLists;
};
```

2. **Uniqueness:**

   In the given case, graph G is connected. Thus, in this case, the partition V1 and V2 is unique (assuming the order V1 and V2 does not matter.)

   This can be seen from the method used to solve the given problem i.e., the two coloring itself. Assuming we know that a bipartite connected graph can always be colored using two colors say A and B then if we choose any vertex in V for partition V1 and assigns it the color say A then all its neighbors will be assigned color B and their neighbors A and so on. Since the graph is connected, all vertices will be assigned a color. Since the coloring is unique or complementary for different choices of the first vertex, the partition is unique.

   However, when a graph is unconnected it has ***always*** *has* more than one way of partitioning it into V1 and V2 if it is bipartite. This can be seen as follow:

   Suppose the graph has n components, let a1 and a2 be any two of them.

   Since G is bipartite, a1 and a2 themselves are bipartite as well.

   Say x1 and x2 are bipartite partitions of a1 and y1 and y2 of a2. Let z1 and z2 be two bipartitions of the remaining n-2 components (again we know that the union of all those components must be bipartite as well assuming a third component otherwise they both are empty.)

   We then partition G as $x1 \cup y1 \cup z1$and $x2 \cup y2 \cup z2$ once and $x1 \cup y2 \cup z1$ and $x2 \cup y1 \cup z2$ again.

   Note that these partitions must be different as G was not connected and thus had more than one component.

   Hence, we prove by contradiction that a non-connected bipartite graph can always be partitioned in more than one way.

3. **Implementation:**

   [On Hackerrank]