Algorithm: DFS BiPartite
Input: A simple connected undirected graph $G = (V,E)$
Output: A partition of $V(G)$.
Is (V_0, V_1) if G is bipartite and $(V(G), \phi)$ otherwise.

```
Initialize a stack S.
Initialize an array Color[1..n] with value -1.
// -1 not colored. 0, 1 two different colors
Pick a starting vertex s and Color(s) ← 0.
S.push(s)
while S ≠ ∅ do
    v ← S.peek()
    if there a vertex w vertex adjacent to v that is not colored then
        w← next vertex adjacent to v to be colored.
        if all colored vertices adjacent to w has the same color as v then
            Color[w] = (Color[v] + 1) % 2
            Push w onto S
        else   // color conflict. G is not bipartite
                return (V(G), ϕ).
    else
        S.pop()
// coloring completed.
V0 ← all vertices of G colored 0
V1 ← all vertices of G colored 1
return (V0, V1)
```

Algorithm: BFS Bipartite
Input: A simple connected undirected graph G = (V,E)
Output: A partition of V(G).
Is (V_0, V_1) if G is bipartite and (V(G), ϕ) otherwise.

```
Initialize a queue Q
Initialize an array Color[1..n] with value -1.
Pick a starting vertex s and Color(s) ← 0.
Q.add(s)
while Q ≠ ∅ do
    v ← Q.dequeue()
    for all vertex w adjacent to v that is not colored
        if all colored vertices adjacent to w has the same color as v then
            Color[w] = (Color[v] + 1) % 2
            Q.add(w)
        else   // color conflict. G is not bipartite
            return (V(G), ϕ).
// coloring completed.
V0 ← all vertices of G colored 0
V1 ← all vertices of G colored 1
return (V0, V1)
```