EXERCISE REPORT

TEAM 15

BRUTE-FORCE

Lecturer: Son Nguyen Thanh MCS

Class: CS112.N21.KHTN

Member 1: Quan Vo Minh – 21520093

Member 2: Nhat Nguyen Viet – 21520378

Excercise: Team 4's Excercise

Date: 15/04/2023

PROBLEM:

A graph is said to be bipartite if all its vertices can be partitioned into two disjoint subsets X and Y so that every edge connects a vertex in X with a vertex in Y. (One can also say that a graph is bipartite if its vertices can be colored in two colors so that every edge has its vertices colored in different colors; such graphs are also called 2-colorable.)

1. Design a **DFS**-based algorithm for checking whether a graph is bipartite.
2. Design a **BFS**-based algorithm for checking whether a graph is bipartite.

SOLUTION:

To the definition of a bipartite graph, our problem is simply to find whether the given graph consists of 2 adjacent vertices that have the same color (i.e. 0, 1). If one vertex is colored as 1(0) and any of its adjacent vertices is colored as 1(0) then the graph is not bipartite, and vice versa.

In short, to solve the problem, we will try to color all vertices of the given graph respectively, so that all adjacent pairs of vertices have two different colors. And, we can confirm that: *It does not matter which vertex is chosen to be root and so does the color to fill the root vertex*.

Here are pseudo-codes:

Initialization:

```
# G is the graph (G = {V, E})

# Initialize a color array to store the color of each vertex
colors = [None] * G.num_vertices

# Flip the color
def revertColor(color):
    if color == 0:
        return 1
    else:
        return 0

def canColor(v, color):
    pass

def IsBipartite(G):
    for v in G.V:
        # color all vertices (not yet colored) with 0
        if colors[v] is None:
            # If conficts were found, the graph is not bipartite
            if not canColor(v, 0):
                return False

    # If all vertices have been colored and no conflicts were found, the
graph is bipartite
    return True
```

DFS-based:

```
def canColor(v, color):
    colors[v] = color
    # Iterate over all adjacent vertices of v
    for neighbor in v.neighbors:
        # If chosen vertex is not yet colored, color it with different color
with v
        if color[neighbor] is None:
            return canColor(neighbor, revertColor(color))
        # If chosen vertex is already colored with the same color with v then
Graph is not bipartite
        elif color[neighbor] == color[v]:
            return False

    return True
```

BFS-based:

```
# Init a queue to store vertices
Q = queue()

def canColor(v, color):
    colors[v] = color
    Q.push(v)

    # Loop until every vertex is colored
    while not Q.empty():
        vertex = Q.pop() # First In, First Out

        # Iterate over all adjacent vertices of the chosen vertex
        for neighbor in vertex.neighbors:
            # If chosen vertex is not yet colored, color it with different
color with v
            if color[neighbor] is None:
                color[neighbor] = revertColor(color)
                Q.push(neighbor)
            # If chosen vertex is already colored with the same color with v
then Graph is not bipartite
            elif color[neighbor] == color[v]:
                return False

    return True
```