VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY
COMPUTER SCIENCE

EXERCISE REPORT
COMPLETE SEARCH - BRUTE FORCE

Lecturer: Son Nguyen Thanh McS

Class: CS112.N21.KHTN

Members: Hoang Ha Van - 21520033
Anh Vo Thi Phuong - 21522883

Date: 17/04/2023

Contents

1. 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.

2. Solution

Algorithm for checking whether a graph is bipartite:

- Use a color[] array which stores -1, 0 or 1 for every vertex (with -1 if this vertex hasn't been colored, 0 if vertex has colored 0, 1 if vertex has colored 1).

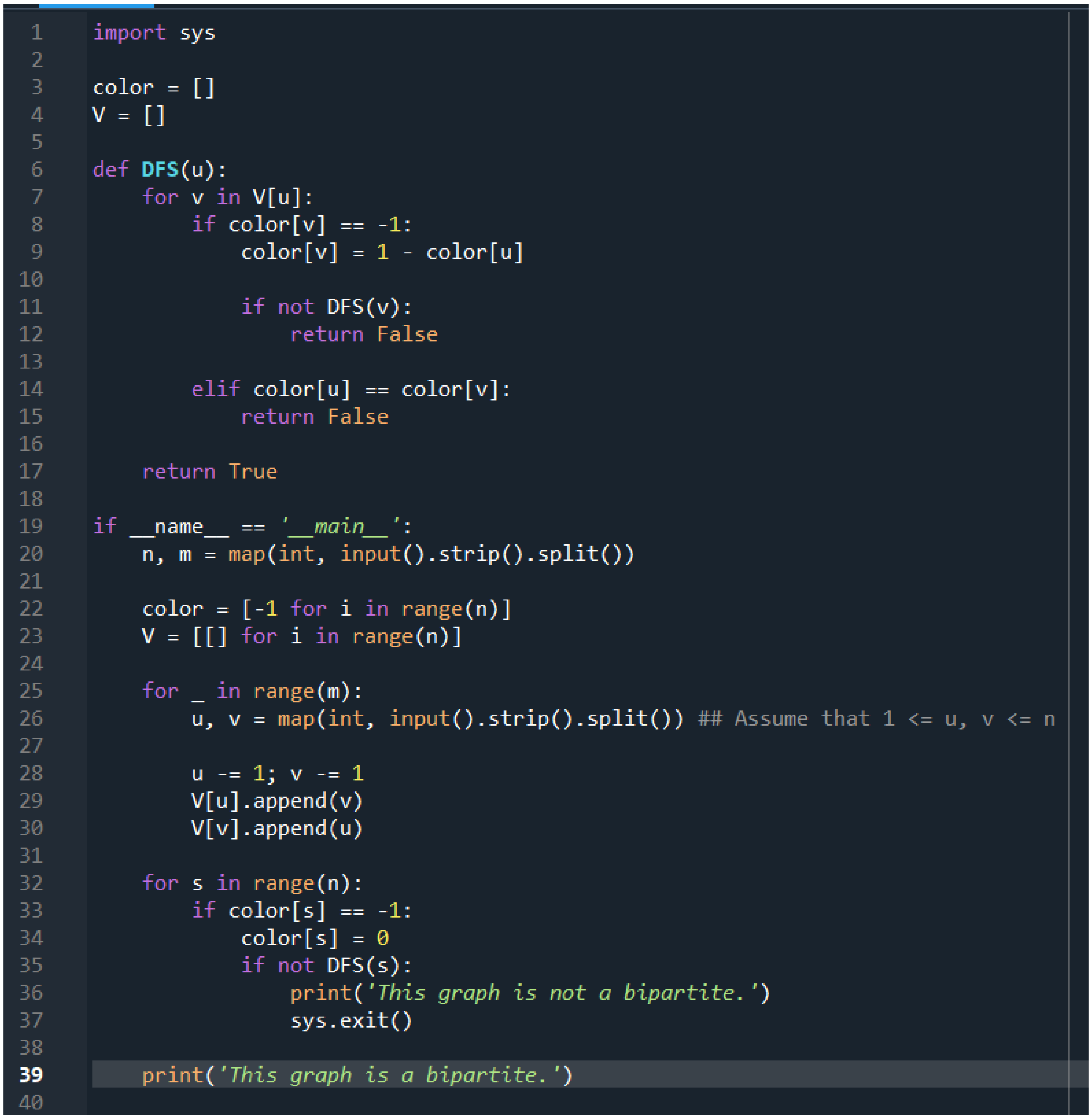
- From each vertex s which hasn't been colored yet (color[s] == -1), set color[s] = 0 or 1 and call the function (DFS or BFS). For BFS, create a queue and push v into it.

- If vertex v which adjacent to u hasn't colored (color[v] == -1), set color[v] = 1 - color[u] and push v into the queue (for BFS) or call DFS again for v (for DFS).

- If at any point, color[u] is equal to color[v], then the graph is not bipartite.

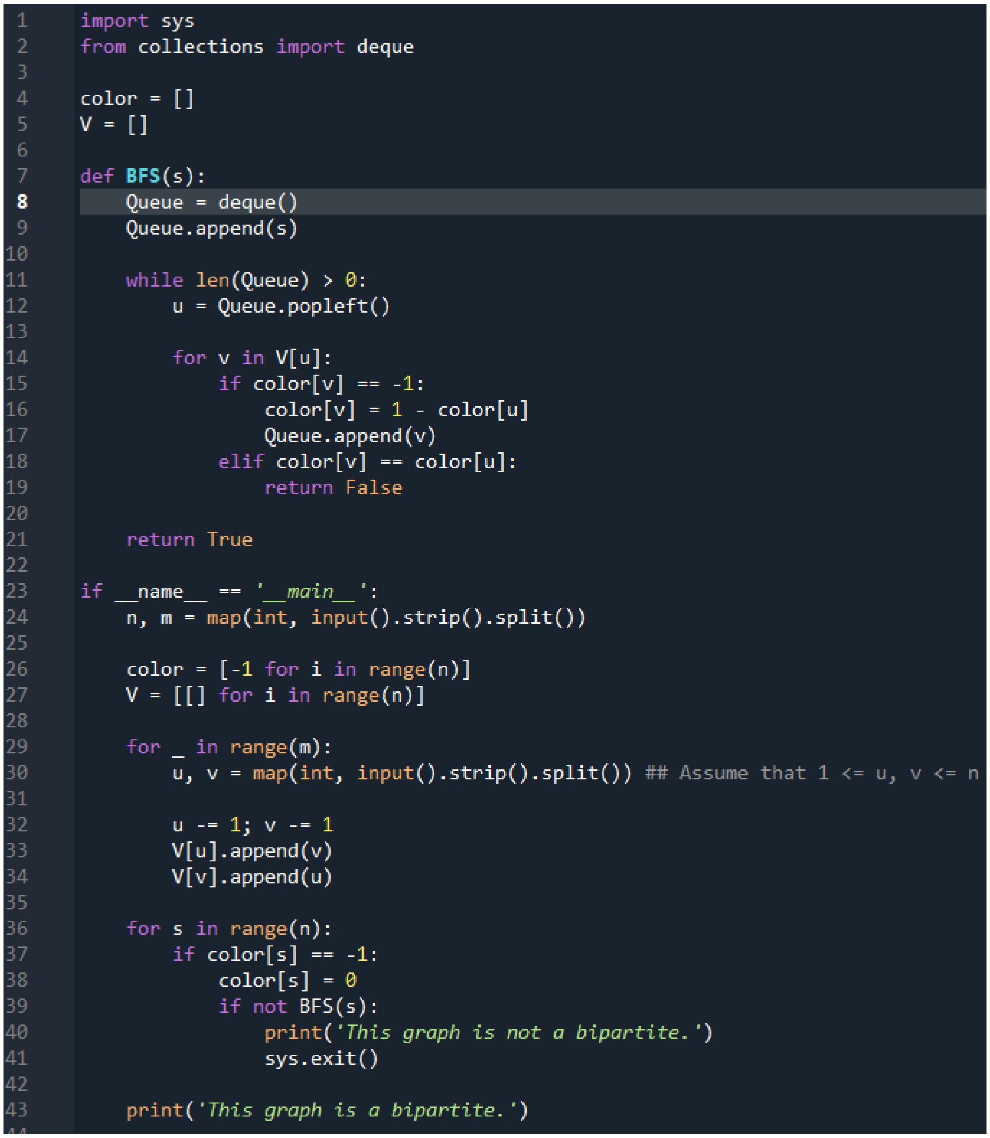
- Modify the function such that it returns a boolean value at the end.

DFS implementation:



```python
import sys

color = []
V = []

def DFS(u):
    for v in V[u]:
        if color[v] == -1:
            color[v] = 1 - color[u]

            if not DFS(v):
                return False

        elif color[u] == color[v]:
            return False

    return True

if __name__ == '__main__':
    n, m = map(int, input().strip().split())

    color = [-1 for i in range(n)]
    V = [[] for i in range(n)]

    for _ in range(m):
        u, v = map(int, input().strip().split()) ## Assume that 1 <= u, v <= n

        u -= 1; v -= 1
        V[u].append(v)
        V[v].append(u)

    for s in range(n):
        if color[s] == -1:
            color[s] = 0
            if not DFS(s):
                print('This graph is not a bipartite.')
                sys.exit()

    print('This graph is a bipartite.')

```

BFS implementation:



```python
import sys
from collections import deque

color = []
V = []

def BFS(s):
    Queue = deque()
    Queue.append(s)

    while len(Queue) > 0:
        u = Queue.popleft()

        for v in V[u]:
            if color[v] == -1:
                color[v] = 1 - color[u]
                Queue.append(v)
            elif color[v] == color[u]:
                return False

    return True

if __name__ == '__main__':
    n, m = map(int, input().strip().split())

    color = [-1 for i in range(n)]
    V = [[] for i in range(n)]

    for _ in range(m):
        u, v = map(int, input().strip().split()) ## Assume that 1 <= u, v <= n

        u -= 1; v -= 1
        V[u].append(v)
        V[v].append(u)

    for s in range(n):
        if color[s] == -1:
            color[s] = 0
            if not BFS(s):
                print('This graph is not a bipartite.')
                sys.exit()

    print('This graph is a bipartite.')
```