ĐẠI HỌC QUỐC GIA TPHCM
Trường Đại học Công nghệ Thông tin
Khoa Khoa học Máy tính

BÀI TẬP BRUTE FORCE

Thành viên 1: 21520117 - Phan Trường Trí
Thành viên 2: 21520373 - Nguyễn Đức Nhân
Lớp: CS112.N21.KHTN
Giảng viên: Nguyễn Thanh Sơn

Mục lục

1 Đề bài: Tower of Hanoi

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.)

2 Bài giải:

2.1 Design a DFS-based algorithm for checking whether a graph is bipartite

- **Giải thuật:**

  1. Khởi tạo 1 mảng $visited[]$ có n phần tử chứa thông tin đỉnh i đã được thăm hay chưa. Khởi tạo 1 mảng $colors[]$ có n phần tử chứa thông tin đỉnh i có màu gì. Mảng $visited[]$ là 0 và mảng $colors[]$ được khởi tạo tất cả bằng -1.
  2. Thực hiện thuật toán DFS với mọi node.
  3. Nếu đỉnh u chưa được thăm trước đó thì gán màu của đỉnh u là màu ngược lại với màu của đỉnh v (với v là đỉnh ở bước DFS hiện tại và u là đỉnh liền kề của v). Sau đó bắt đầu duyệt DFS tại u
  4. Nếu trong quá trình duyệt DFS ta bắt gặp một đỉnh kề với đỉnh hiện tại nhưng có cùng màu thì ta kết luận đồ thị không phải đồ thị 2 phía.
  5. Nếu duyệt quả tất cả các đỉnh và đều thỏa thì ta có thể kết luận đồ thị trên là đồ thị 2 phía.

- **Mã giả:**

```
    def dfs(vertex, c):
        visited[vertex] = True
        color[vertex] = c

        for adjacent in G[vertex]:
            if not visited[adjacent]:
                dfs(adjacent, !color[vertex])
            elif color[adjacent] == color[vertex]:
                return False
        return True

    function isBipartite(G):
        #n is the number of vertices in G
        visited = array of size n, initialized with 0
        color = array of size n, initialized with -1

        for vertex in range(n):
            if not visited[vertex]:
                if not dfs(vertex, 0):
                    return False

        return True

```

Độ phức tạp của thuật toán giảm còn $O(n)$.

2.2 Design a DFS-based algorithm for checking whether a graph is bipartite

- **Giải thuật:**

1. Khởi tạo 1 mảng $visited[]$ có n phần tử chứa thông tin đỉnh i đã được thăm hay chưa. Khởi tạo 1 mảng $colors[]$ có n phần tử chứa thông tin đỉnh i có màu gì. Mảng $visited[]$ là 0 và mảng $colors[]$ được khởi tạo tất cả bằng -1.
2. Chọn 1 đỉnh u và đặt $colors[u] = 0$ sau đó thêm u vào hàng đợi.
3. Trong khi hàng đợi vẫn còn đỉnh trong đấy. Lấy đỉnh đầu tiên được nạp vào hàng đợi ra và thêm các đỉnh liền kề của đỉnh này vào hàng đợi.
4. Với mỗi đỉnh kề, nếu đỉnh này chưa được thăm trước đó thì ta sẽ gán $colors[v] =!c$ (v là đỉnh kề của đỉnh đang xét còn c là

màu của đỉnh đang xét). Còn nếu đỉnh này đã được thăm, ta kiểm tra đỉnh này có cùng màu với đỉnh đang xét không. Nếu có thì đồ thị không phải đồ thị 2 phía.

5. Sau khi hoàn thành BFS mà thõa điều kiện trên thì đồ thị là đồ thị 2 phía.

- **Mã giả:**

```
function isBipartite(G):
    #n is the number of vertices in G
    visited = array of size n, initialized with 0
    color = array of size n, initialized with -1

    for vertex in range(n):
        if visited[vertex]:
            continue
        color[vertex] = 0
        queue.push(vertex)

        while queue not empty:
            current = queue.pop(0)
            visited[current] = 1

            for adjacent in G[current]:
                if not visited[adjacent]:
                    color[adjacent] = !color[current]
                    queue.append(adjacent)
                elif color[adjacent] == color[current]:
                    return False

    return True

```

Độ phức tạp của thuật toán giảm còn $O(V + E)$.