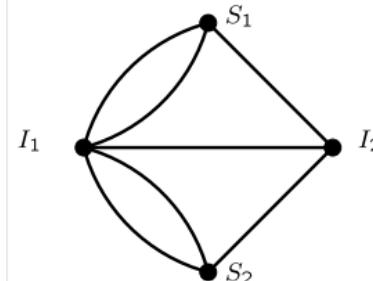
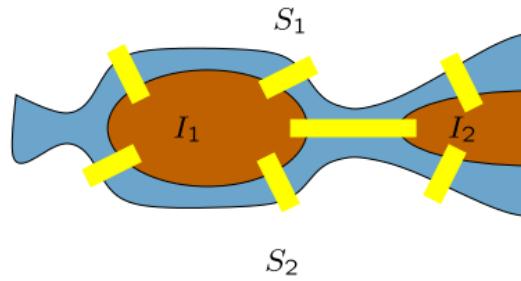


LÝ THUYẾT ĐỒ THỊ

Duyệt đồ thị & Ứng dụng

Phạm Nguyên Khang
BM. Khoa học máy tính, CNTT
pnkhang@cit.ctu.edu.vn



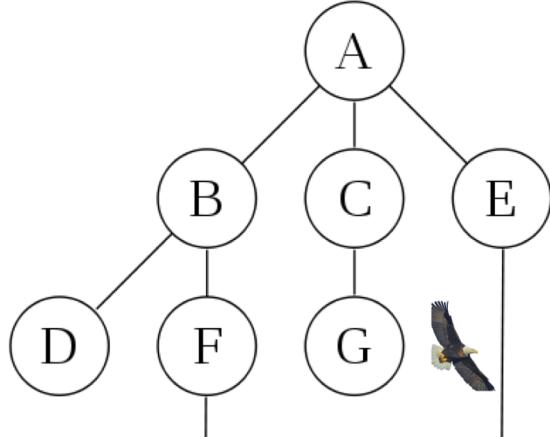
Cần Thơ, 8/2021

Trong tuần 3

- Tính liên thông của đồ thị
 - Liên thông của đồ thị vô hướng
 - Liên thông mạnh của đồ thị có hướng
- Ứng dụng duyệt đồ thị
 - Kiểm tra tính liên thông/liên thông mạnh (Tarjan)
 - Tìm bộ phận liên thông

Dựng cây duyệt đồ thị

- Trong quá trình duyệt đồ thị, có 2 bước chính
 - Duyệt đỉnh u
 - Xem xét duyệt các đỉnh kề v của nó
 - Mỗi quan hệ “cha – con” giữa các đỉnh => cây duyệt đồ thị



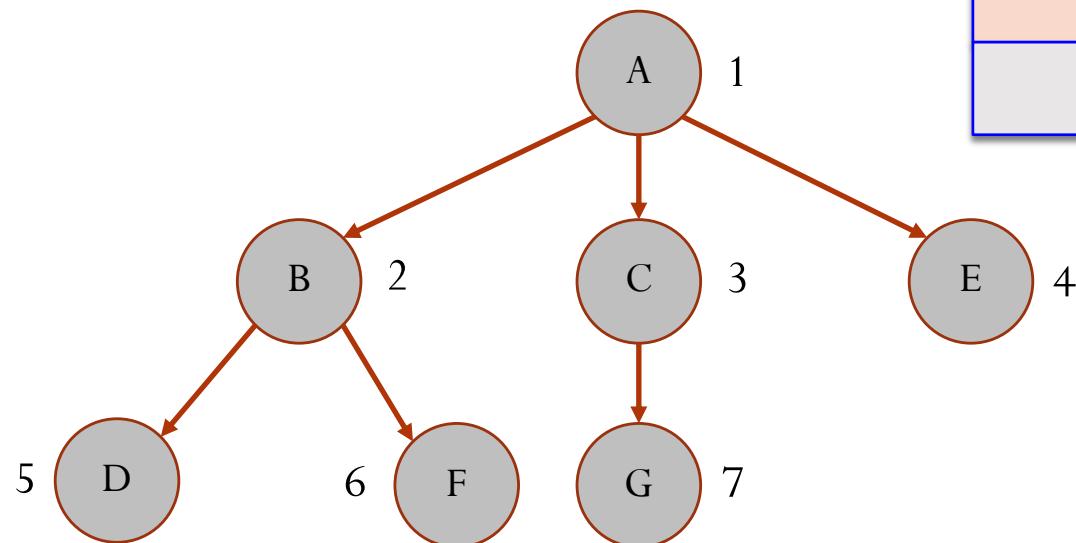
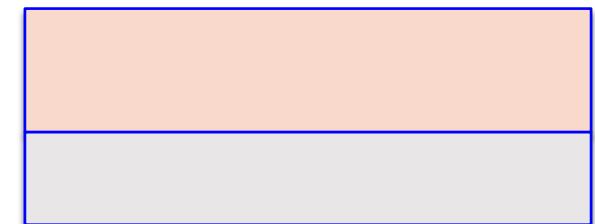
Đưa 1 đỉnh **s** bất kỳ vào **hàng đợi** (vd: đỉnh 1)

while **hàng đợi** chưa rỗng **do**

Lấy đỉnh ở đầu **hàng đợi** ra => gọi là đỉnh **u**
Làm gì đó trên **u**

for các đỉnh kề **v** của **u** **do**

if (**v** chưa duyệt) Đưa **v** vào **hàng đợi**



Hàng đợi rỗng
=> Kết thúc

Thứ tự duyệt:
A, B, C, E, D, F, G

Well done!
Thank you, Scrat



Dựng cây duyệt đồ thị

- Tổ chức hàng đợi/ngăn xếp có khả năng lưu:
 - Đỉnh sắp sửa duyệt và đỉnh cha của nó (đỉnh đưa nó vào hàng đợi/ngăn xếp)

```
typedef struct {  
    int u; //đỉnh sẽ được duyệt  
    int p; //đỉnh cha của u  
} ElementType;
```

```
typedef struct {  
    int front, rear;  
    ElementType data[MAX_SIZE];  
} Queue;
```

```
int parent[MAX_N];
```

B	C	E			
A	A	A			

Đưa 1 đỉnh **(s, -1)** vào **hàng đợi**

while **hàng đợi** chưa rỗng **do**

Lấy phần tử ở đầu **hàng đợi** ra => **(u, p)**

Cho u là con của p (**parent[u] = p;**)

for các đỉnh kề v của u **do**

if (v chưa duyệt) Đưa **(v, u)** vào **hàng đợi**

Duyệt theo chiều sâu (đệ quy)

```
void DFS(u) {  
    if (u đã duyệt)  
        return;  
    //Duyệt u  
    Đánh dấu u đã duyệt  
  
    for (các đỉnh kề v của u)  
        DFS(v);  
}
```

```
void DFS(u) {  
  
    //Duyệt u  
    Đánh dấu u đã duyệt  
  
    for (các đỉnh kề v của u)  
        if (v chưa duyệt)  
            DFS(v);  
}
```

Dựng cây duyệt đồ thị

- Duyệt theo chiều sâu dùng đệ quy (DFS đệ quy)
 - Thêm một đối số p vào hàm DFS

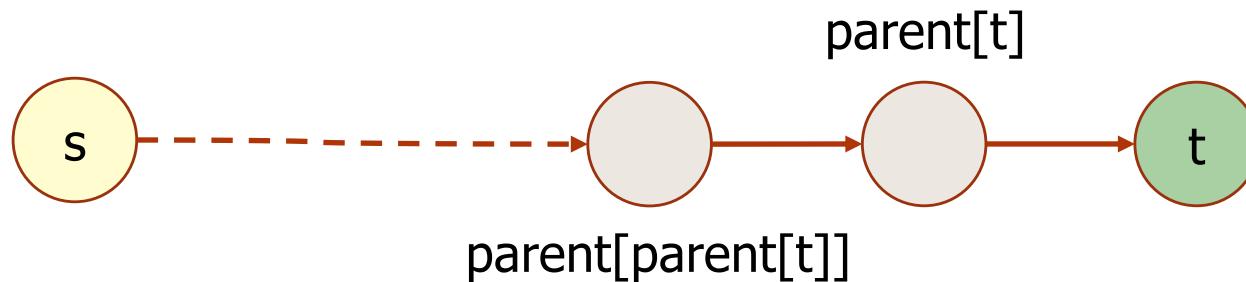
```
int parent[MAX_N];
```

```
void DFS(int u, int p) {  
    if (u đã duyệt)  
        return;  
    //Duyệt u, đánh dấu u đã duyệt  
    //Cho p là cha của u  
    parent[u] = p;  
  
    for (các đỉnh kề v của u)  
        DFS(v, u);  
}
```

```
void DFS(int u, int p) {  
    //Duyệt u, đánh dấu u đã duyệt  
    //Cho p là cha của u  
    parent[u] = p;  
  
    for (các đỉnh kề v của u)  
        if (v chưa duyệt)  
            DFS(v, u);  
}
```

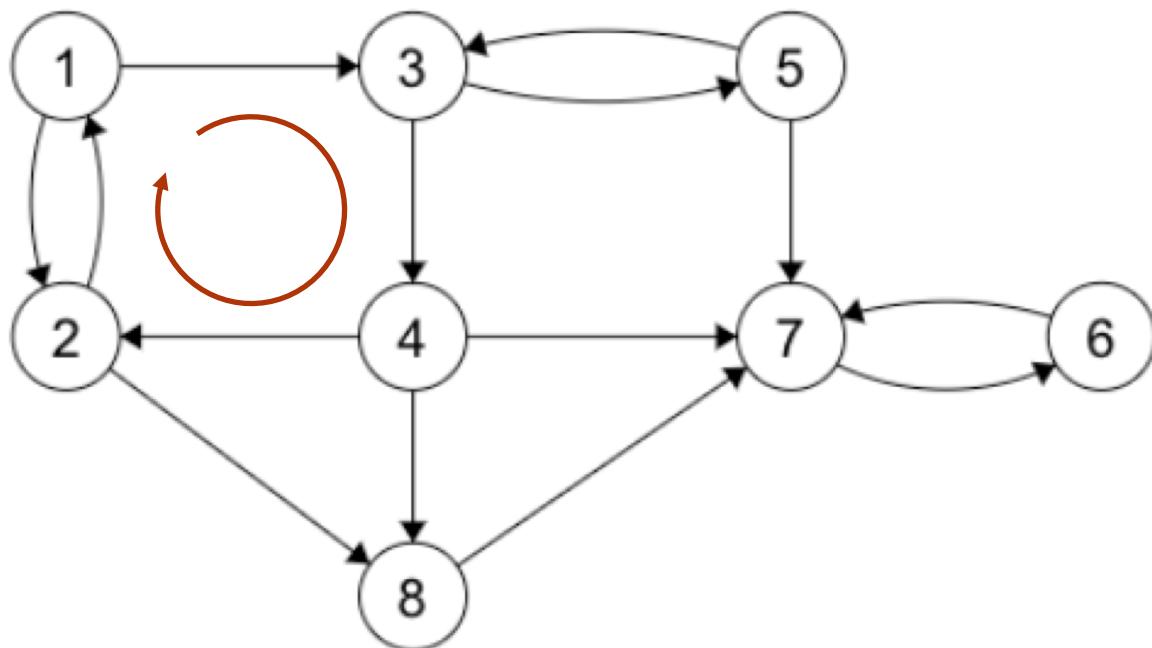
Tìm đường đi trên đồ thị

- Tìm đường đi từ s đến t.
 - Duyệt đồ thị từ đỉnh s và dựng cây trong quá trình duyệt
 - Nếu trong quá trình duyệt, đỉnh t được duyệt => Có đường đi
 - Từ t lần ngược lại theo $\text{parent}[t]$, $\text{parent}[\text{parent}[t]]$... sẽ đến s.
 - Nếu t không được duyệt => KHÔNG có đường đi từ s đến t.



Kiểm tra đồ thị chứa chu trình

- Nhắc lại: *chu trình (cycle) là đường đi (path) có đỉnh đầu trùng đỉnh cuối*



Vd: $1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1$

Kiểm tra đồ thị chứa chu trình

- Nhắc lại: *chu trình (cycle) là đường đi (path) có đỉnh đầu trùng đỉnh cuối*

	A	B	C	D	E
1	=C4				
2		=A1 + 2			
3					
4			=B2 * 2		
5					
6					
7					
8					
9					

Error
Circular dependency detected.
To resolve with iterative
calculation, see File >
Spreadsheet Settings.

Phụ thuộc vòng (circular dependency) trong Excel

Kiểm tra đồ thị chứa chu trình

- Nhắc lại: *chu trình (cycle) là đường đi (path) có đỉnh đầu trùng đỉnh cuối*
- Làm thế nào để phát hiện đồ thị chứa chu trình?

Kiểm tra đồ thị chứa chu trình

- Nhắc lại: *chu trình (cycle) là đường đi (path) có đỉnh đầu trùng đỉnh cuối*
- Làm thế nào để phát hiện đồ thị chứa chu trình?
 - Kiểm tra xem có đường đi nào có đỉnh đầu trùng đỉnh cuối

Kiểm tra đồ thị chứa chu trình

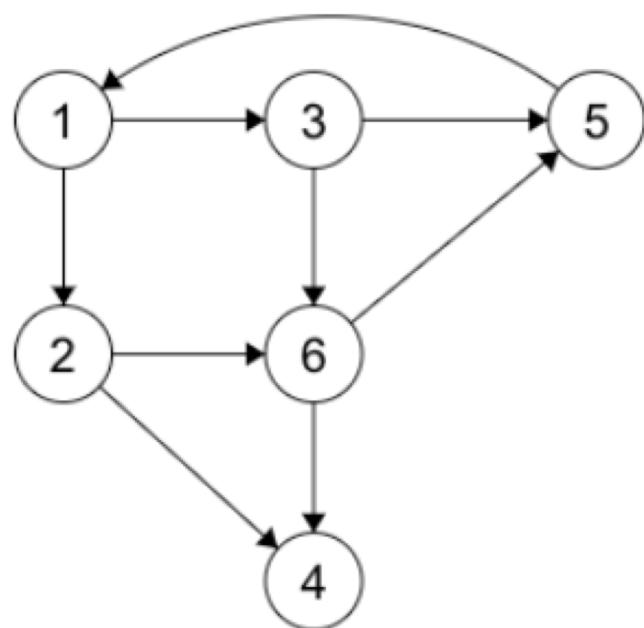
- Nhắc lại: *chu trình (cycle) là đường đi (path) có đỉnh đầu trùng đỉnh cuối*
- Làm thế nào để phát hiện đồ thị chứa chu trình?
 - Kiểm tra xem có đường đi nào có đỉnh đầu trùng đỉnh cuối
 - Kiểm tra như thế nào?

Kiểm tra đồ thị chứa chu trình

- Nhắc lại: *chu trình (cycle) là đường đi (path) có đỉnh đầu trùng đỉnh cuối*
- Làm thế nào để phát hiện đồ thị chứa chu trình?
 - Kiểm tra xem có đường đi nào có đỉnh đầu trùng đỉnh cuối
 - Kiểm tra như thế nào?
 - Kiểm tra trong quá trình duyệt đồ thị

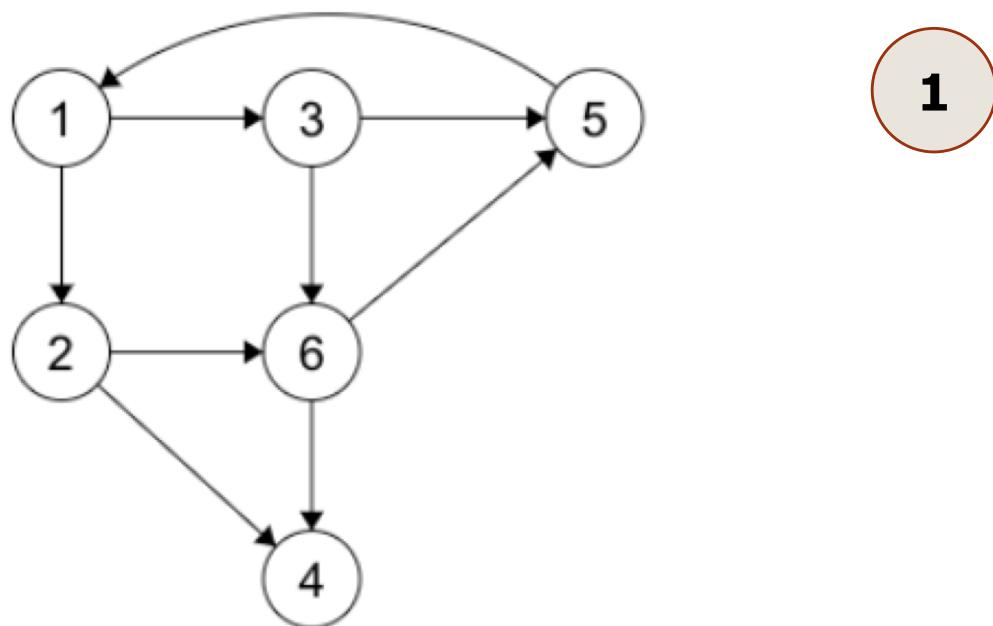
Kiểm tra đồ thị chứa chu trình

- DFS đệ quy



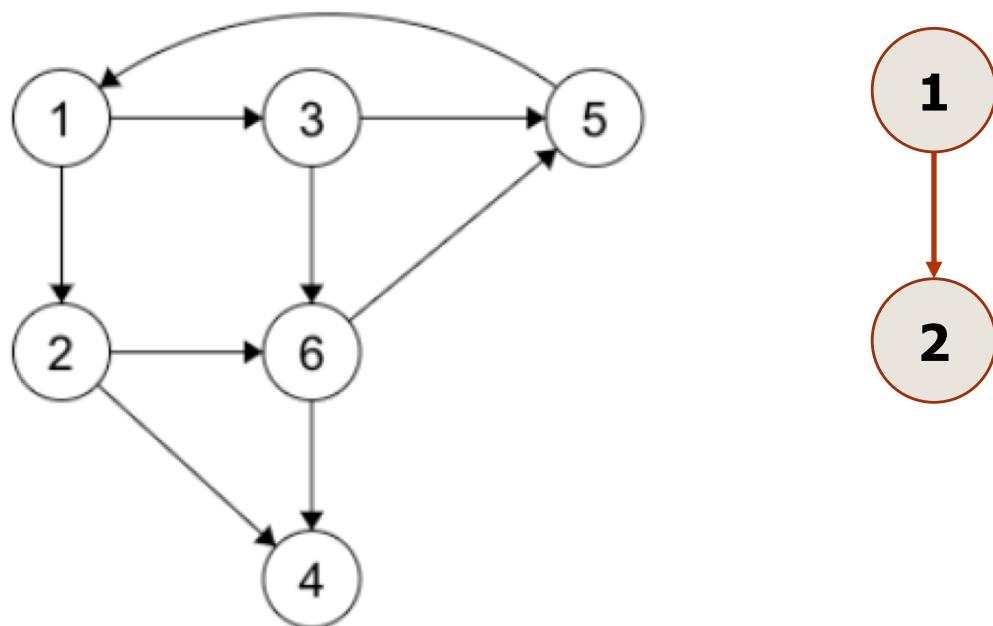
Kiểm tra đồ thị chứa chu trình

- DFS đệ quy



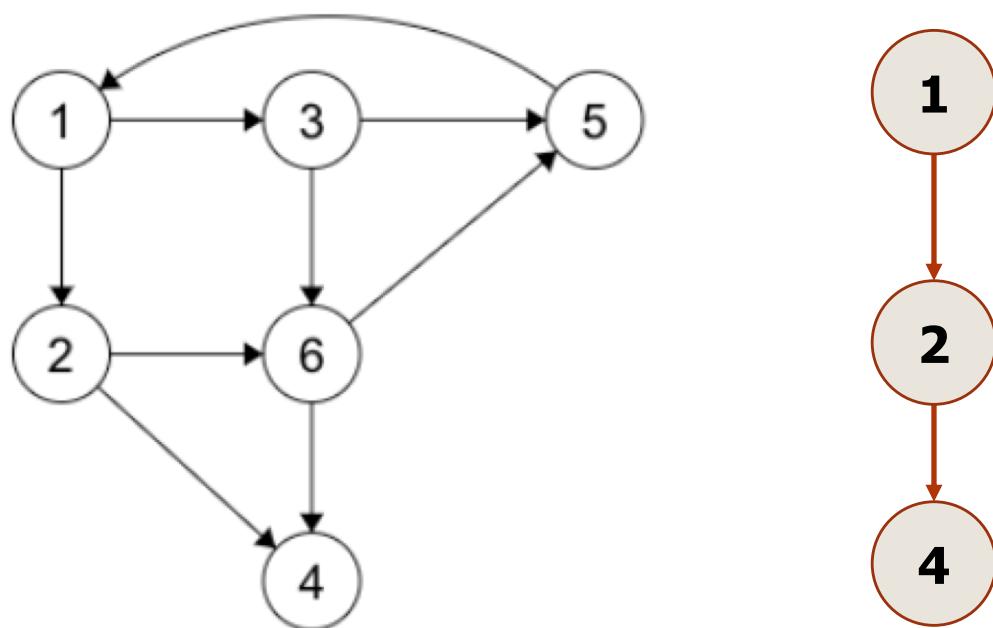
Kiểm tra đồ thị chứa chu trình

- DFS đệ quy



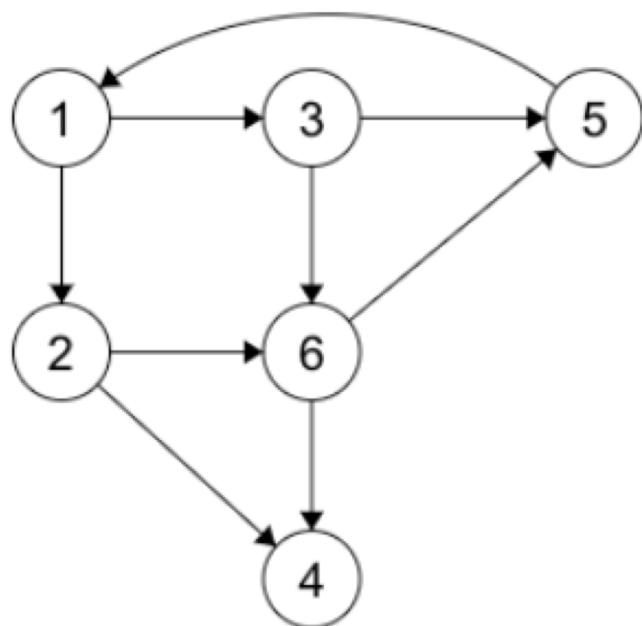
Kiểm tra đồ thị chứa chu trình

- DFS đệ quy



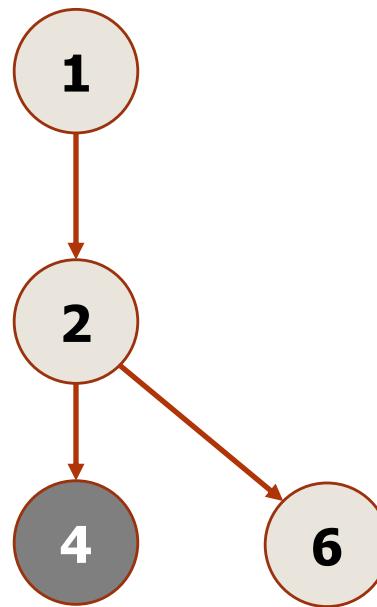
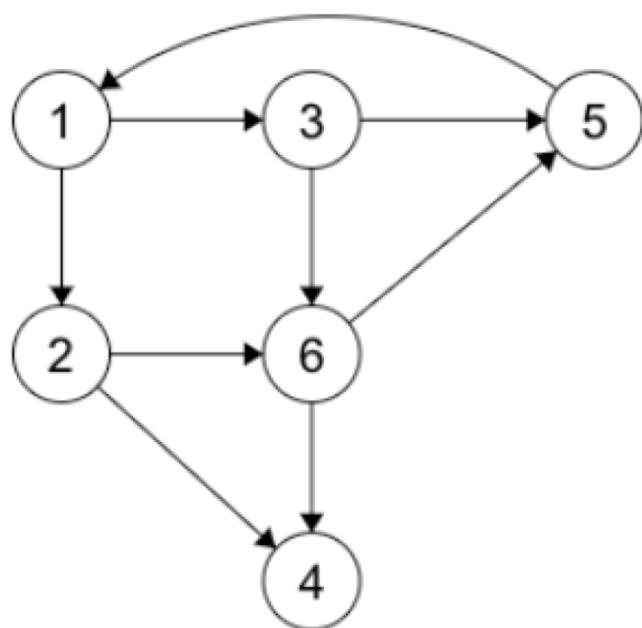
Kiểm tra đồ thị chứa chu trình

- DFS đệ quy



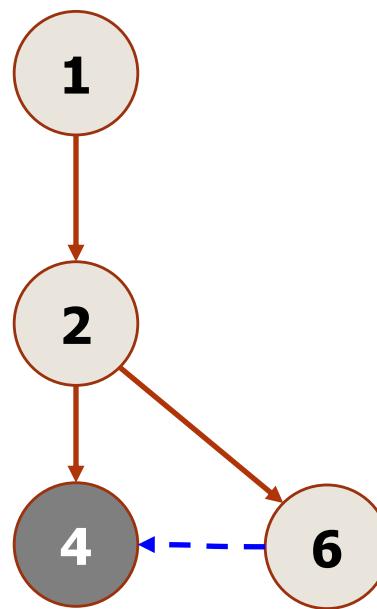
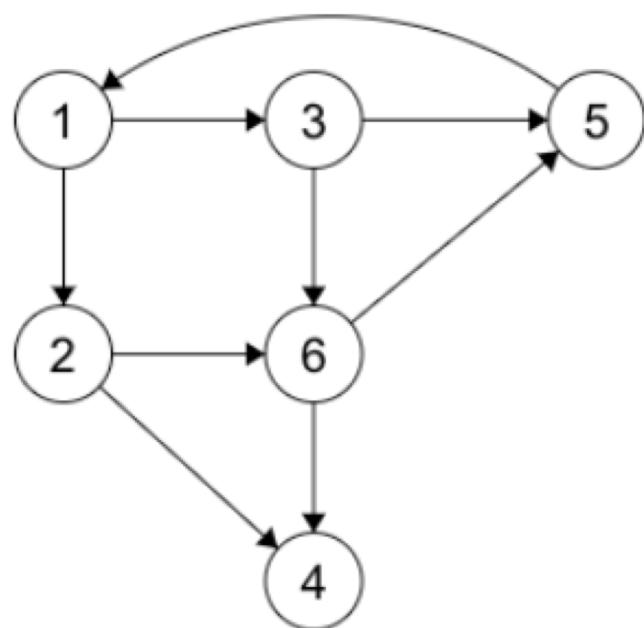
Kiểm tra đồ thị chứa chu trình

- DFS đệ quy



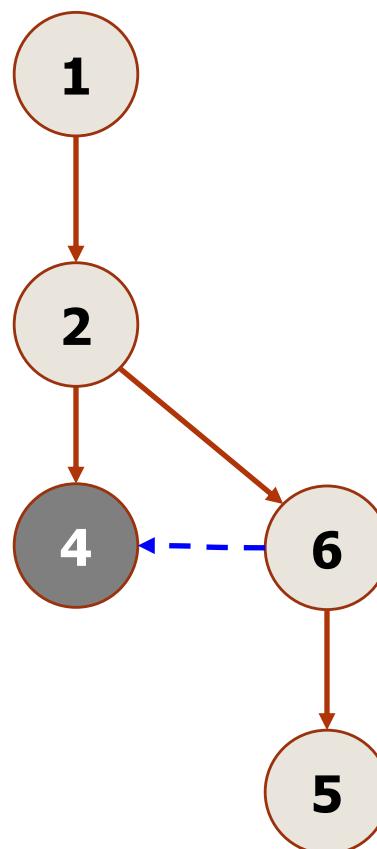
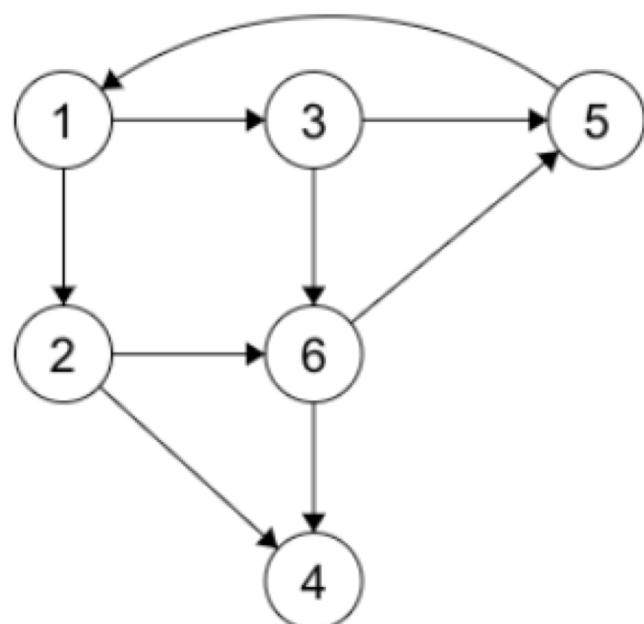
Kiểm tra đồ thị chứa chu trình

- DFS đệ quy



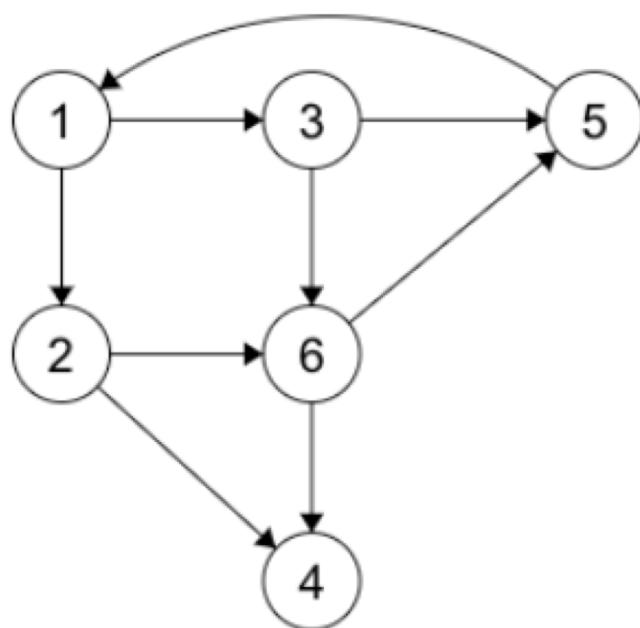
Kiểm tra đồ thị chứa chu trình

- DFS đệ quy

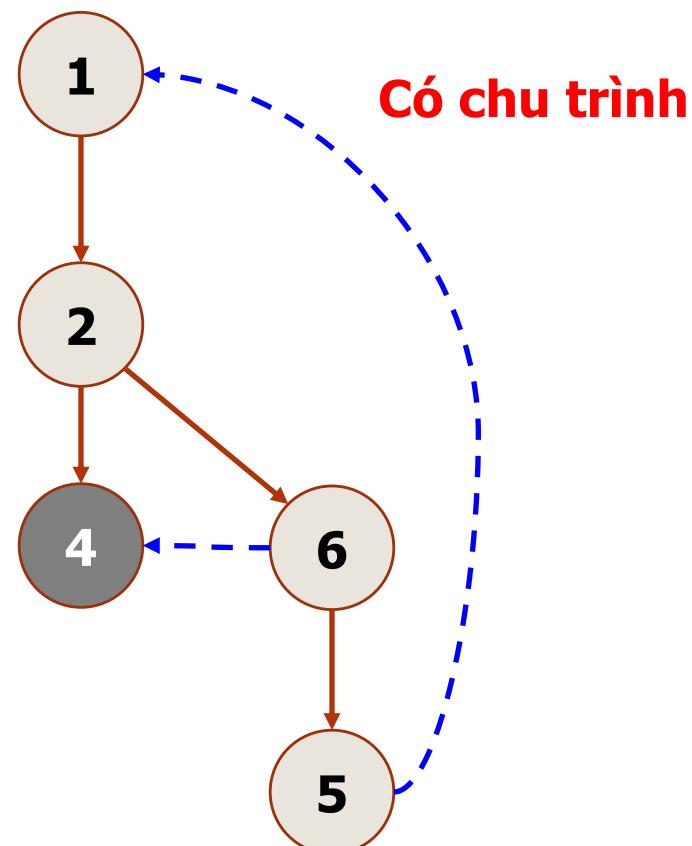


Kiểm tra đồ thị chứa chu trình

- DFS đệ quy



Phát hiện có đường đi:
 $1 \rightarrow 2 \rightarrow 6 \rightarrow 5 \rightarrow 1$

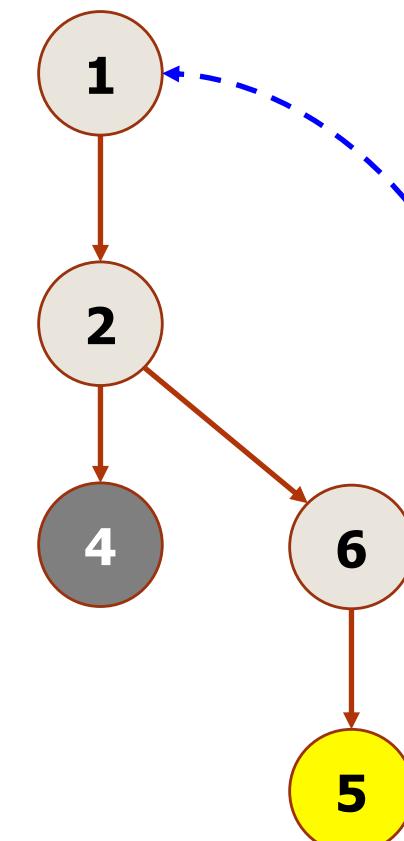
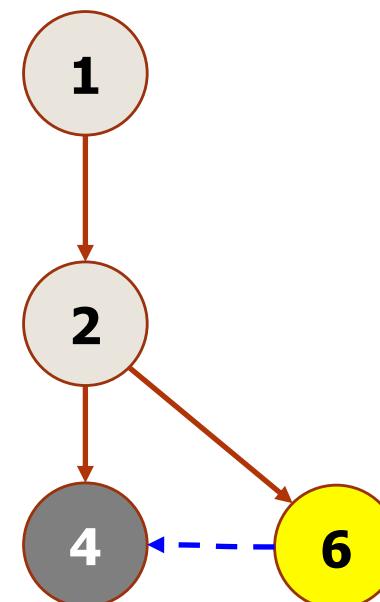


Kiểm tra đồ thị chứa chu trình

- DFS đệ quy

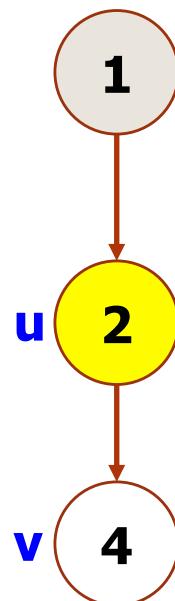


Tạo chu trình

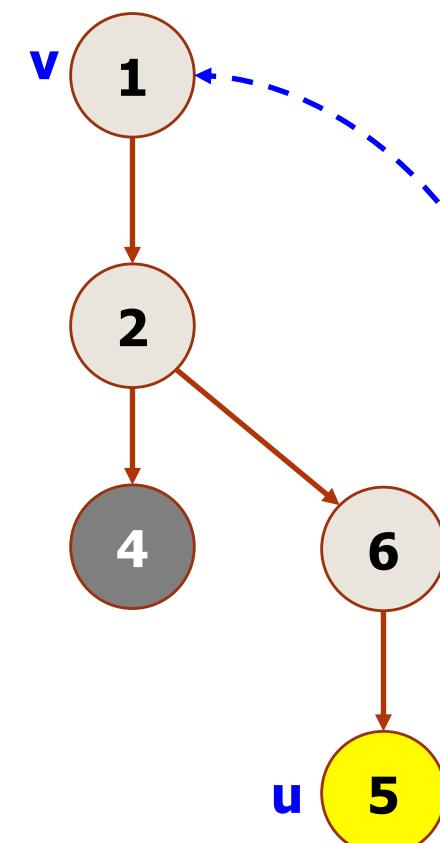
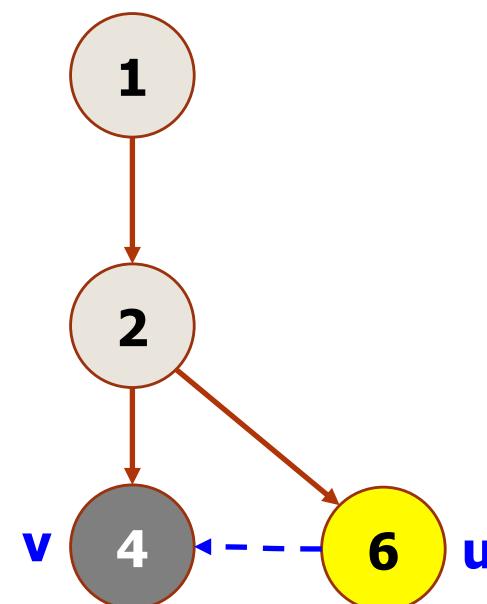


Kiểm tra đồ thị chứa chu trình

- DFS đệ quy

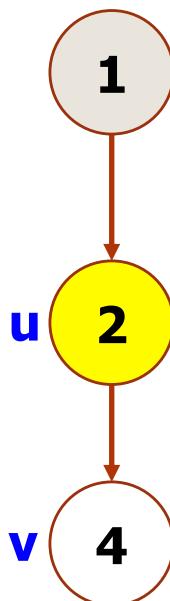


Tạo chu trình

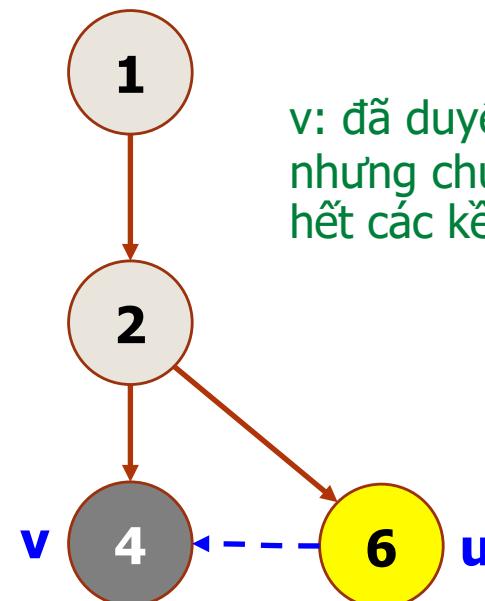


Kiểm tra đồ thị chứa chu trình

- DFS đệ quy

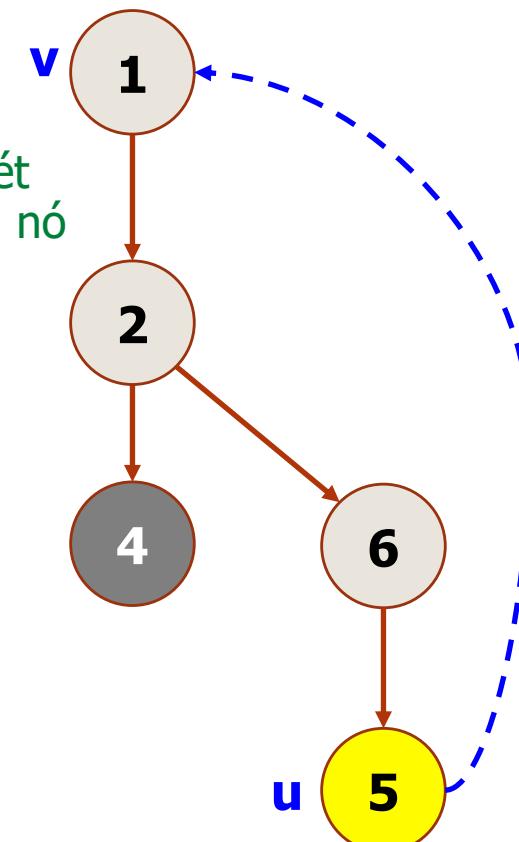


v: chưa duyệt



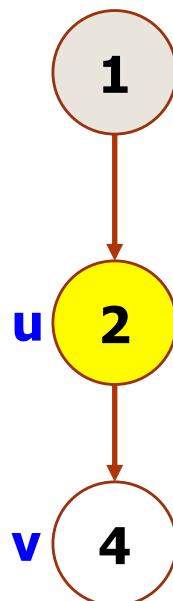
v: đã duyệt, và
xét hết các kề
của nó

Tạo chu trình

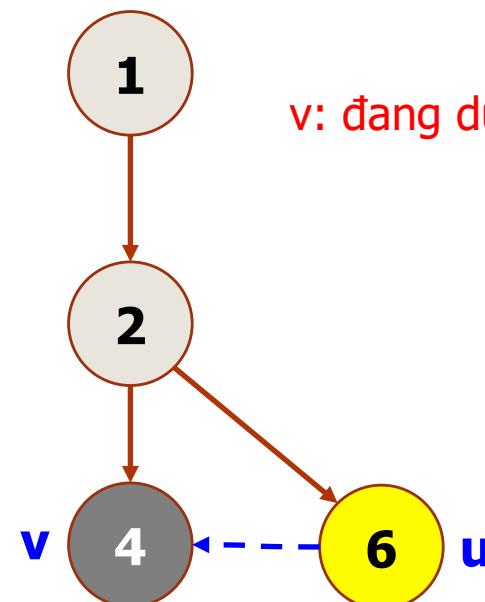


Kiểm tra đồ thị chứa chu trình

- DFS đệ quy

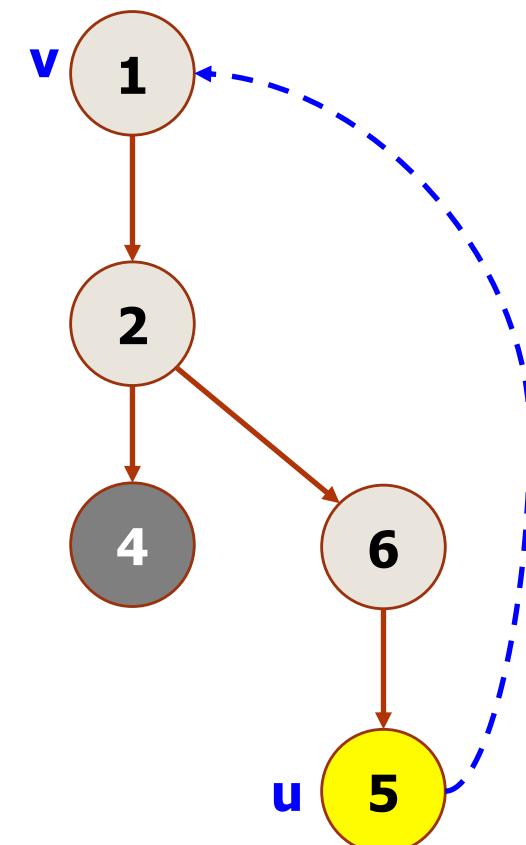


v: chưa duyệt



v: đã duyệt xong

Tạo chu trình



Kiểm tra đồ thị chứa chu trình

- Thuật toán kiểm tra đồ thị chứa chu trình
 - Trạng thái của đỉnh:
 - Chưa duyệt: chưa đụng tới nó (**TRẮNG/WHITE**)
 - Đang duyệt: đã duyệt nó nhưng chưa duyệt hết kề của nó (**XÁM/GRAY**)
 - Đã duyệt xong: duyệt nó và đã duyệt hết các kề của nó (**ĐEN/BLACK**)
 - Thuật toán:
 - DFS + tô màu các đỉnh trong quá trình duyệt
 - Nếu khi xét 1 đỉnh kề v của u mà v có màu xám \Rightarrow tạo chu trình

Kiểm tra đồ thị chứa chu trình

- Kiểm tra đồ thị CÓ HƯỚNG chứa chu trình

```
void DFS(u) {  
    color[u] = GRAY;           //1. Đang duyệt u  
  
    for (các đỉnh kề v của u) //2. Xét kề  
        if (v có màu trắng)   //2a. v chưa duyệt  
            DFS(v);  
        else if (v có màu xám) //2b. v còn đang duyệt  
            has_cycle = 1;      //tạo chu trình  
        else {}               //2c. v đã duyệt xong, bỏ qua  
  
    color[u] = BLACK;          //3. Duyệt xong u  
}
```

Kiểm tra đồ thị chứa chu trình

- Kiểm tra trên toàn bộ đồ thị
 - Vòng lặp (u từ 1 đến n) + DFS(u)

```
int color[MAX_N];
int is_cycle;

void DFS(int u) {
    ...
}
```

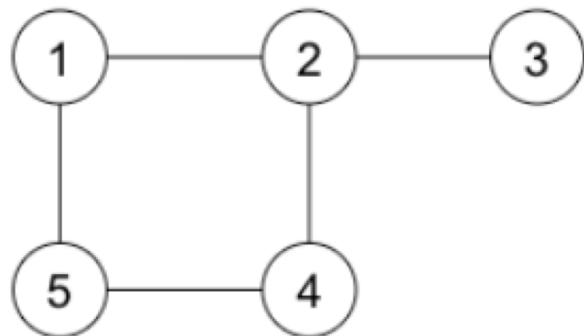
```
int main() {
    ...
    for (u = 1; u <= n; u++)
        color[u] = WHITE;
    has_circle = 0;

    for (u = 1; u <= n; u++)
        if (color[u] == WHITE)
            DFS(u);

    if (has_cicrcle)
        //Chứa chu trình
    else
        //Không chứa chu trình
    ...
}
```

Kiểm tra đồ thị chứa chu trình

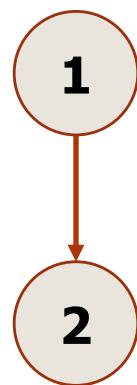
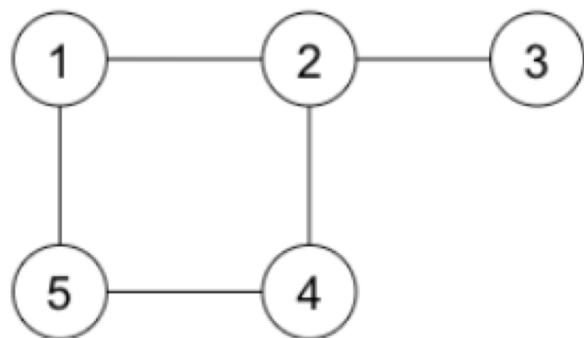
- Đồ thị vô hướng



1

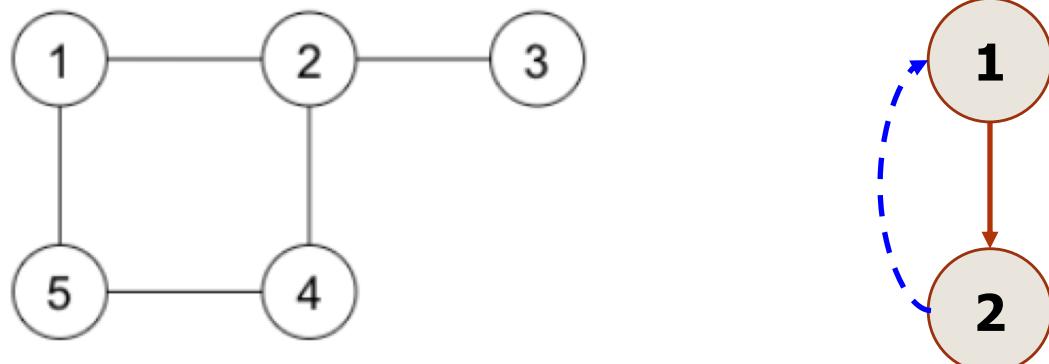
Kiểm tra đồ thị chứa chu trình

- Đồ thị vô hướng



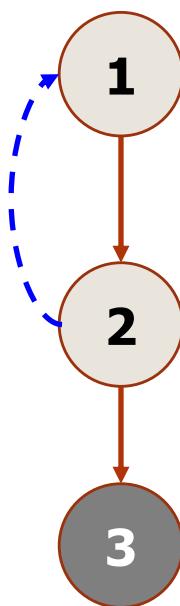
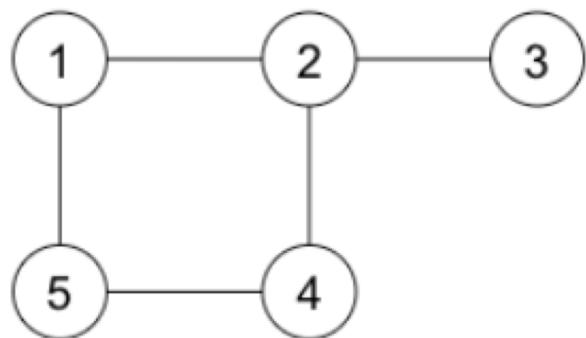
Kiểm tra đồ thị chứa chu trình

- Đồ thị vô hướng



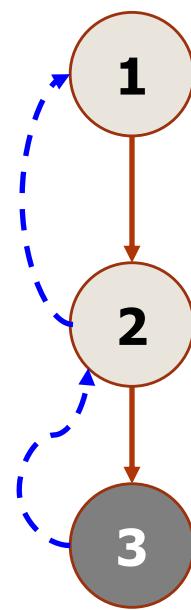
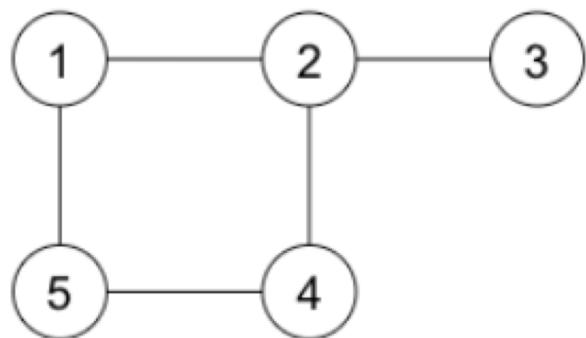
Kiểm tra đồ thị chứa chu trình

- Đồ thị vô hướng



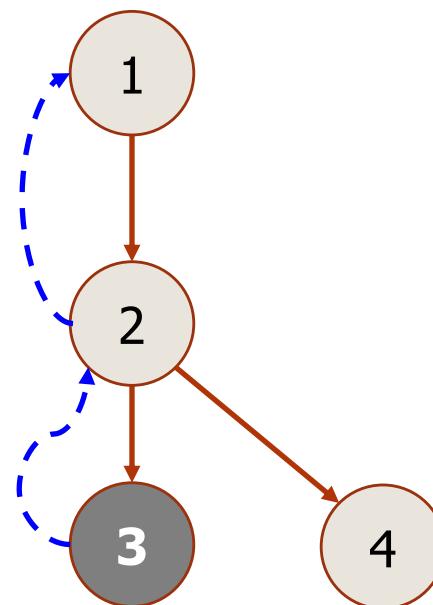
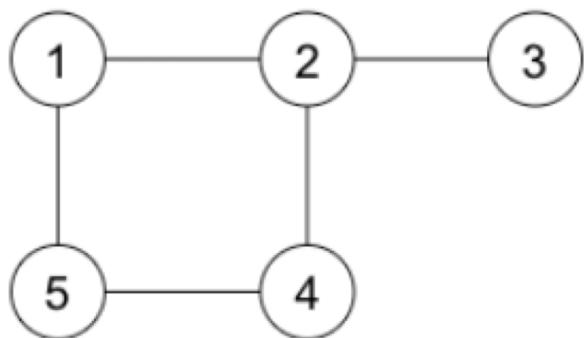
Kiểm tra đồ thị chứa chu trình

- Đồ thị vô hướng



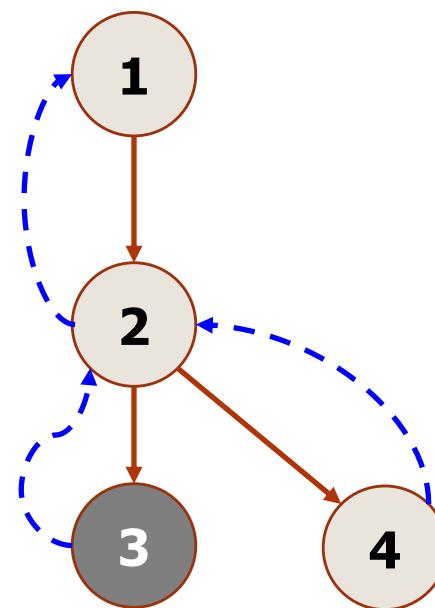
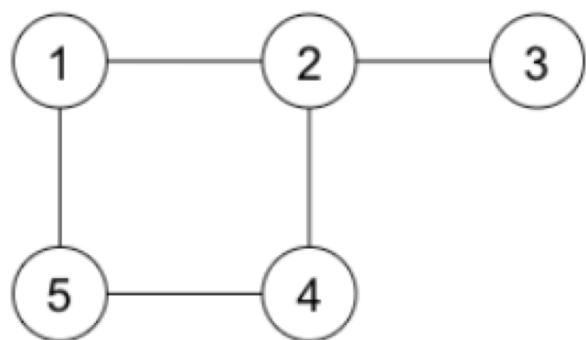
Kiểm tra đồ thị chứa chu trình

- Đồ thị vô hướng



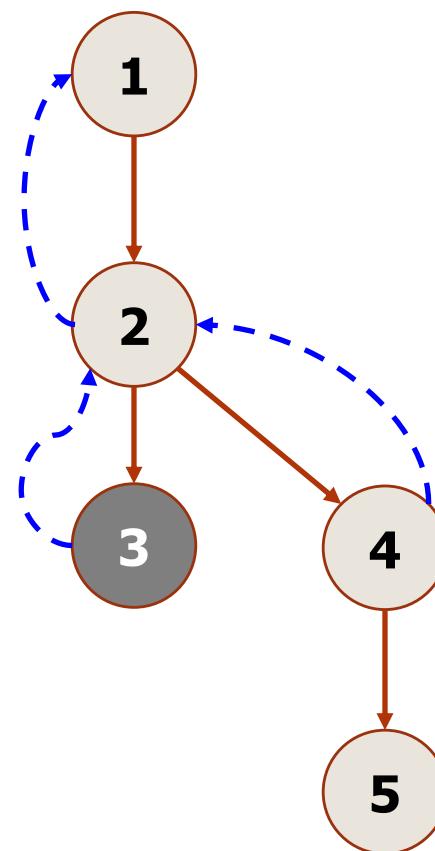
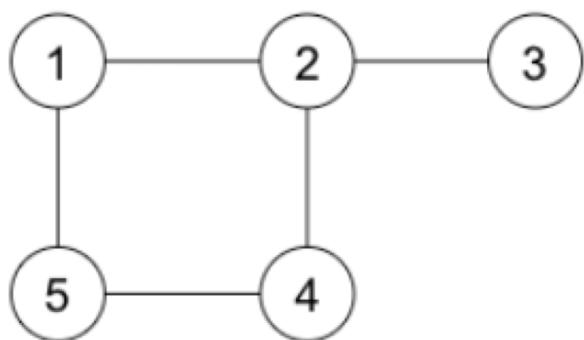
Kiểm tra đồ thị chứa chu trình

- Đồ thị vô hướng



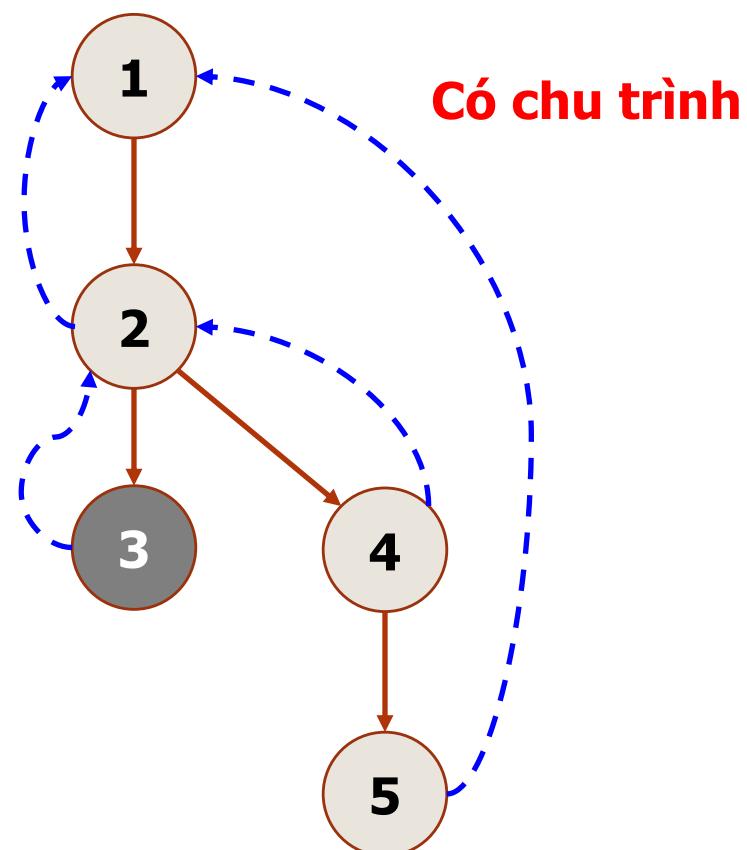
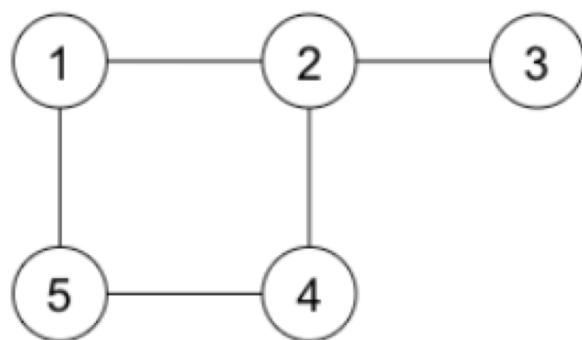
Kiểm tra đồ thị chứa chu trình

- Đồ thị vô hướng



Kiểm tra đồ thị chứa chu trình

- Đồ thị vô hướng

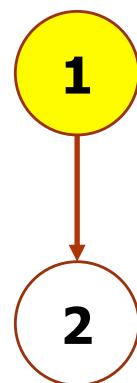


Phát hiện có đường đi:
 $1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 1$

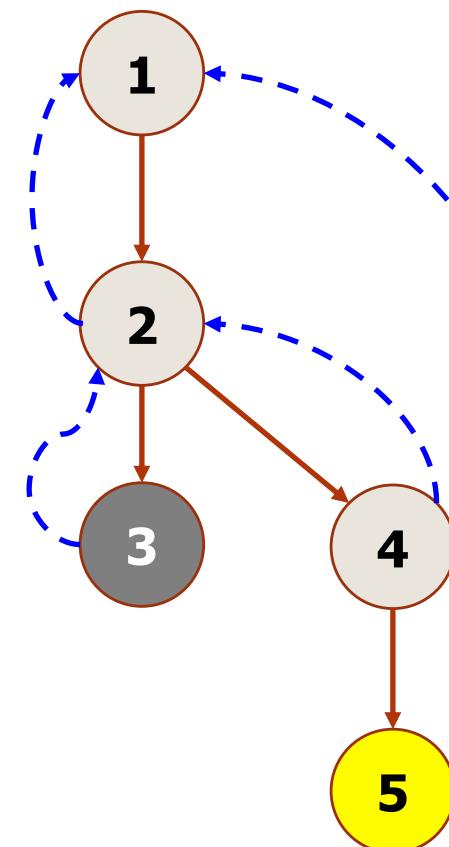
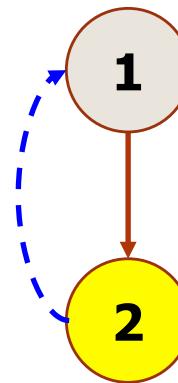
Có chu trình

Kiểm tra đồ thị chứa chu trình

- Đồ thị vô hướng

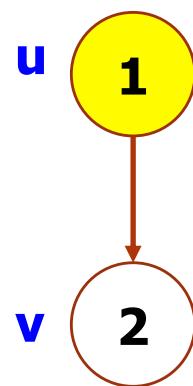


Tạo chu trình

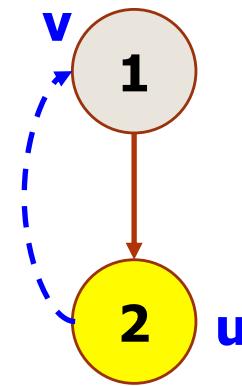


Kiểm tra đồ thị chứa chu trình

- Đồ thị vô hướng

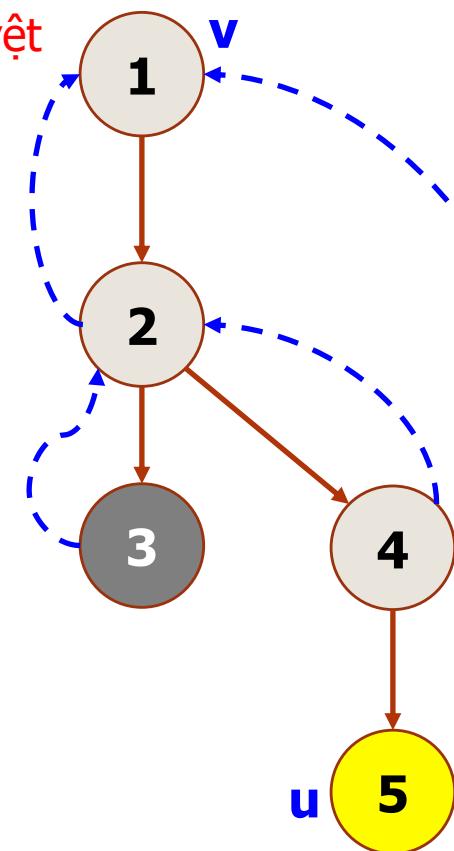


v: chưa duyệt



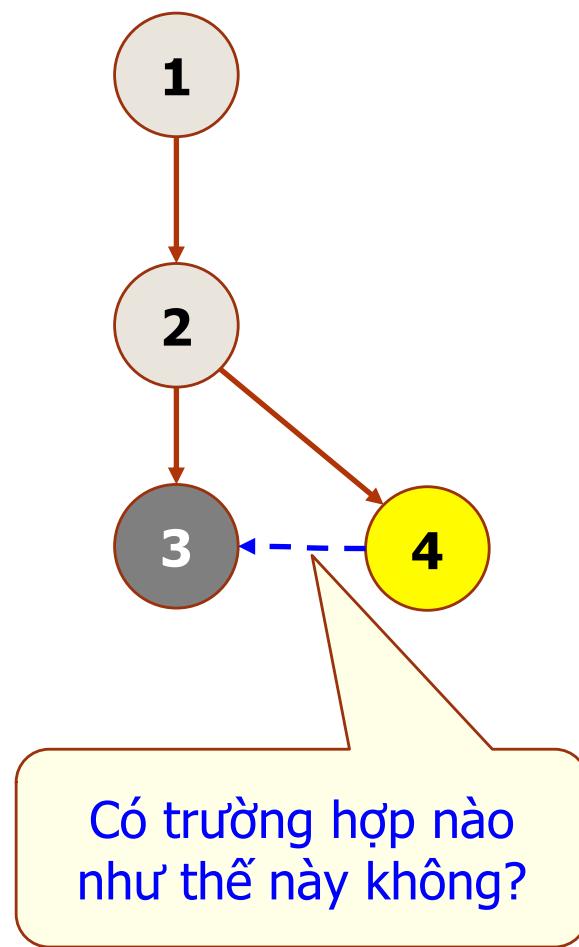
v: đang duyệt, v
là cha của u

Tạo chu trình



Kiểm tra đồ thị chứa chu trình

- Đồ thị vô hướng



Kiểm tra đồ thị chứa chu trình

- Kiểm tra đồ thị VÔ HƯỚNG chứa chu trình

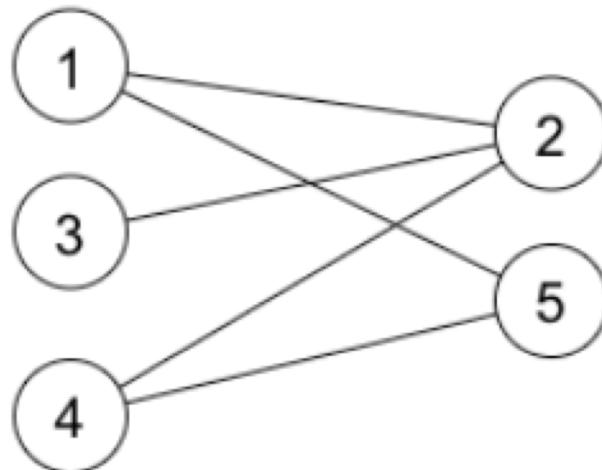
```
void DFS(u, p) {
    color[u] = GRAY; //1. Đang duyệt u

    for (các đỉnh kề v của u) //2. Xét kề
        if (v == p) { //2a. v là cha của p
            //bỏ qua
        } else if (v có màu trắng) //2b. v chưa duyệt
            DFS(v, u); //duyệt v với u là cha của v
        else if (v có màu xám) //2c. v đang duyệt
            is_cycle = 1; //tạo chu trình
        else {} //2d. v đã duyệt xong (KHÔNG có)

    color[u] = BLACK; //3. Duyệt xong u
}
```

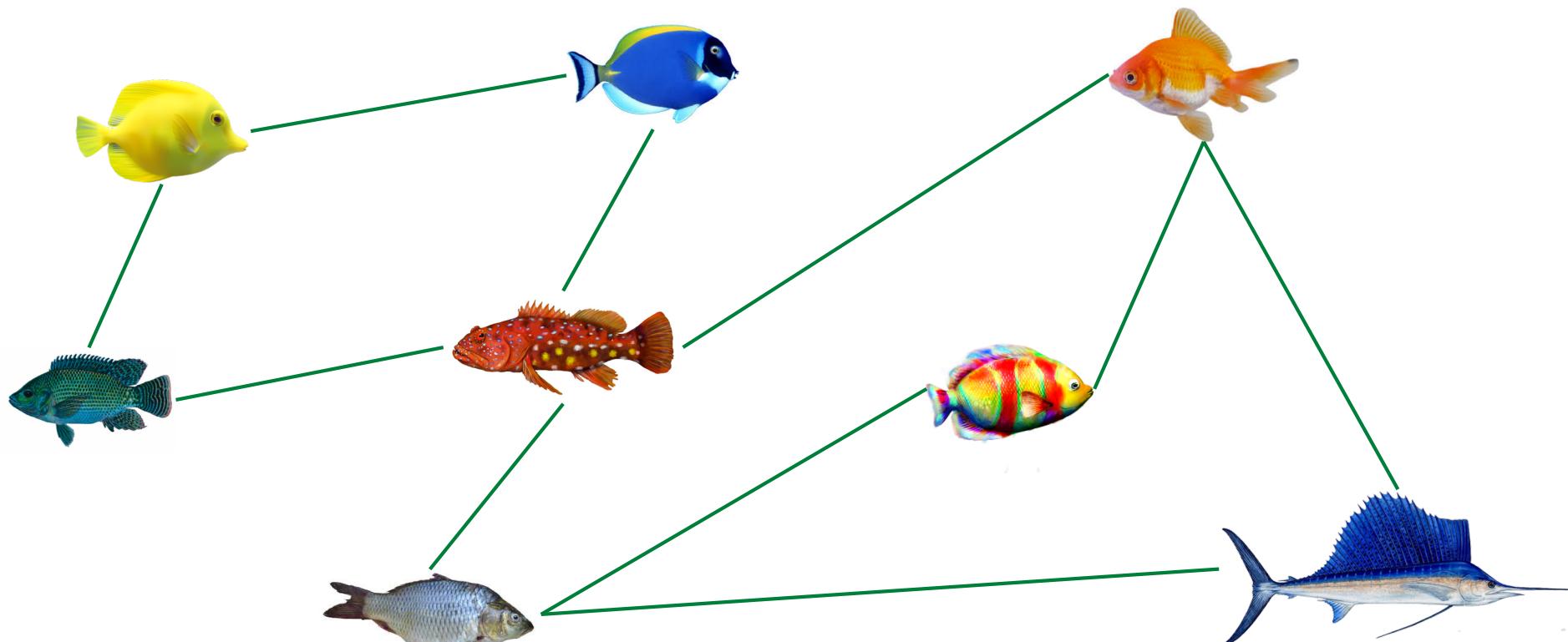
Kiểm tra đồ thị phân đôi

- Đồ thị phân đôi (đồ thị hai phần, đồ thị hai bên, đồ thị hai phía, ...)
 - Đồ thị VÔ HƯỚNG
 - Có thể chia các đỉnh của đồ thị thành hai phần (không giao nhau) sao cho các đỉnh trong mỗi phần KHÔNG kề nhau



Kiểm tra đồ thị phân đôi

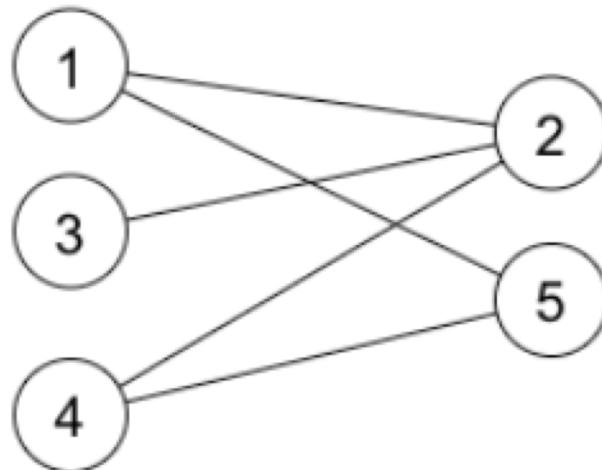
- Một số cặp cá sẽ đá nhau khi thả vào chung 1 hồ
- Ta chỉ có 2 hồ chứa cá
- Cặp các có đường nối sẽ đá nhau



Kiểm tra đồ thị phân đôi

- Thuật toán:

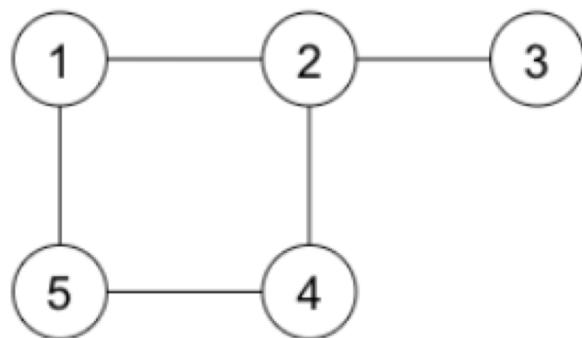
- Duyệt đồ thị + tô màu các đỉnh bằng 1 trong 2 màu (XANH, ĐỎ). Hai đỉnh kề nhau có màu khác nhau.



Kiểm tra đồ thị phân đôi

- Tô màu

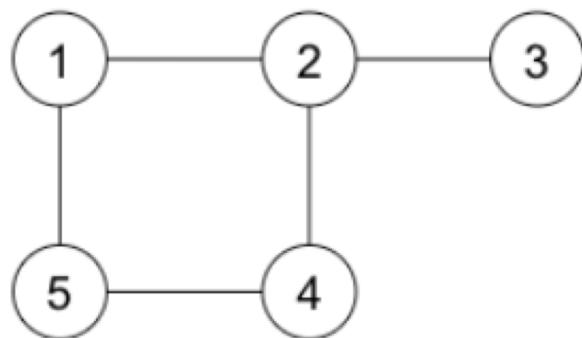
1 chưa có màu



Kiểm tra đồ thị phân đôi

- Tô màu

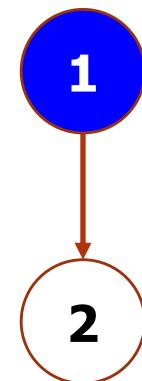
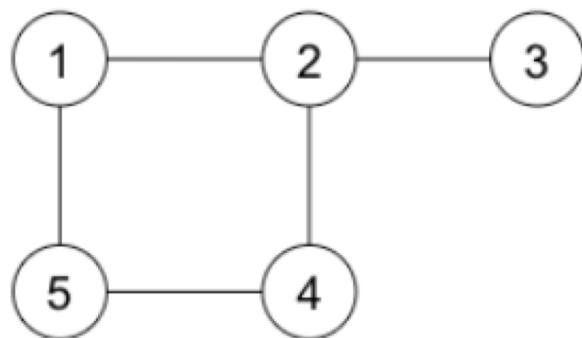
1 chưa có màu, tô màu
XANH cho nó



Kiểm tra đồ thị phân đôi

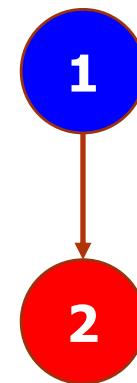
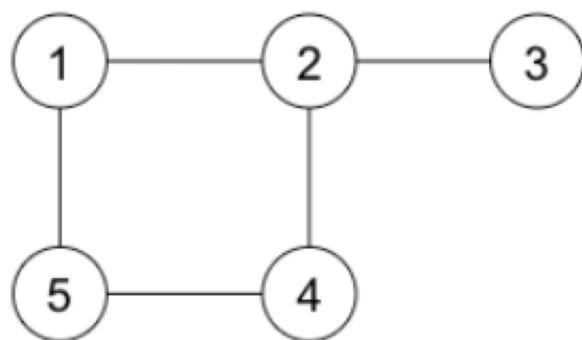
- Tô màu

2 chưa có màu



Kiểm tra đồ thị phân đôi

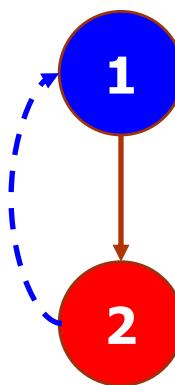
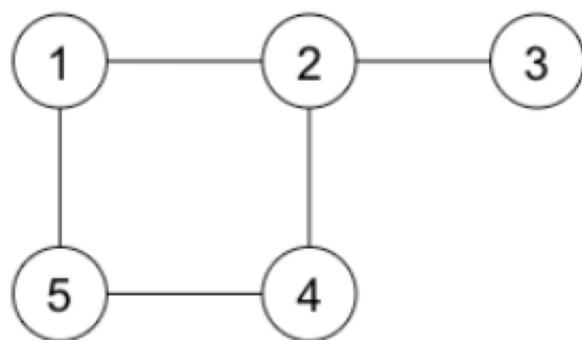
- Tô màu



2 chưa có màu, tô màu
ĐỎ cho nó

Kiểm tra đồ thị phân đôi

- Tô màu

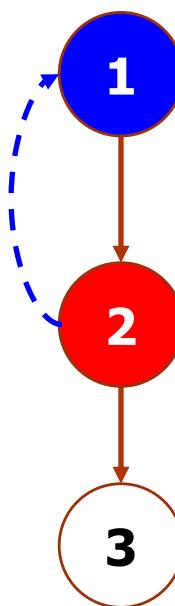
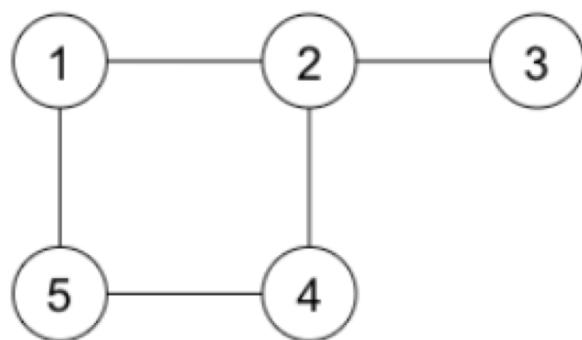


1 đã có màu XANH khác
với màu ĐỎ của 2

Kiểm tra đồ thị phân đôi

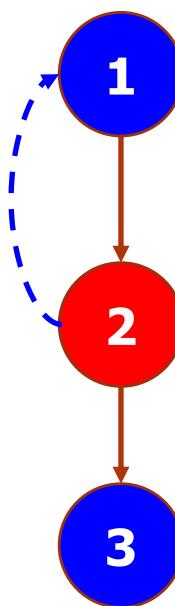
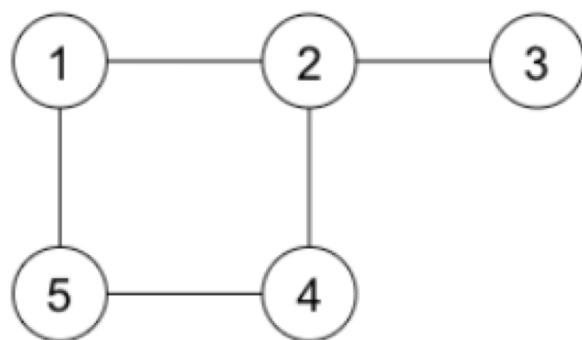
- Tô màu

3 chưa có màu



Kiểm tra đồ thị phân đôi

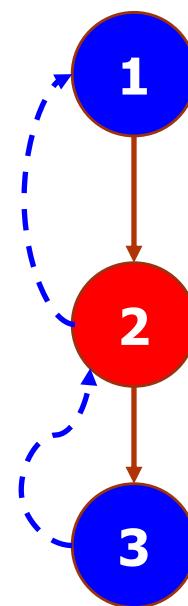
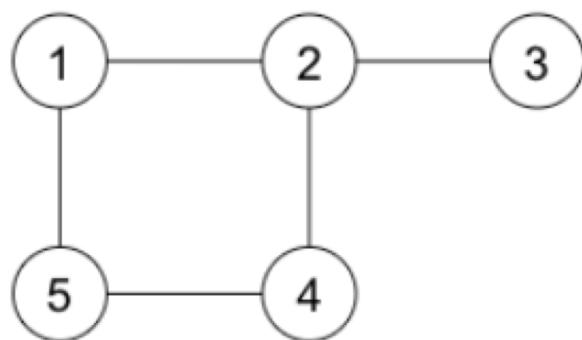
- Tô màu



3 chưa có màu, tô màu
XANH cho nó

Kiểm tra đồ thị phân đôi

- Tô màu

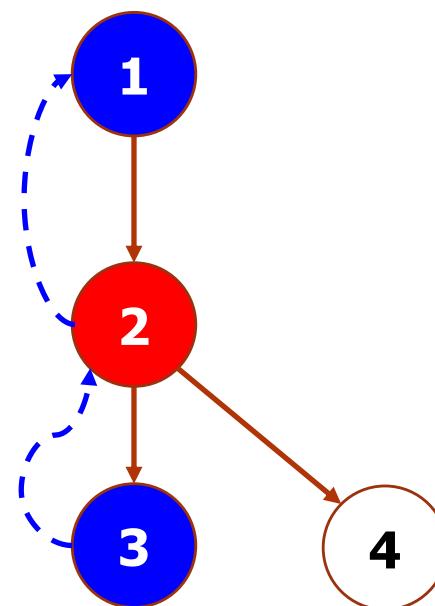
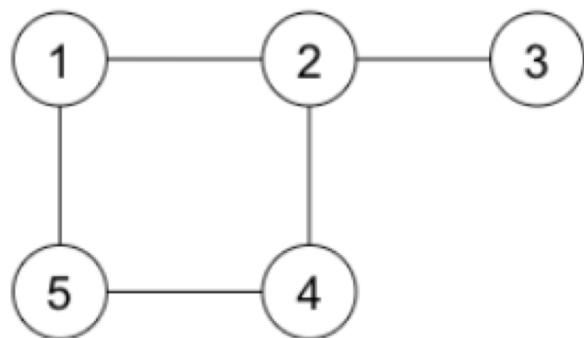


2 đã có màu ĐỎ khác
với màu XANH của 3

Kiểm tra đồ thị phân đôi

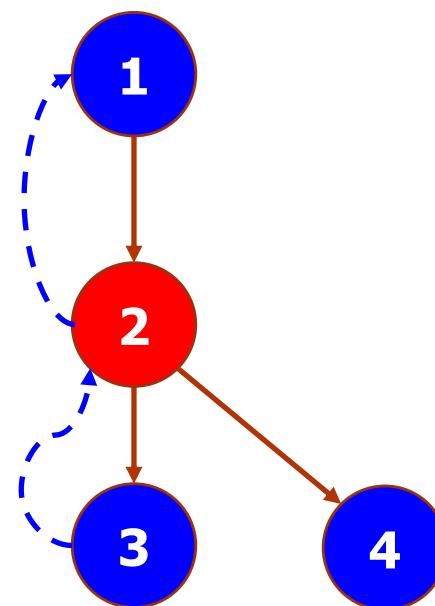
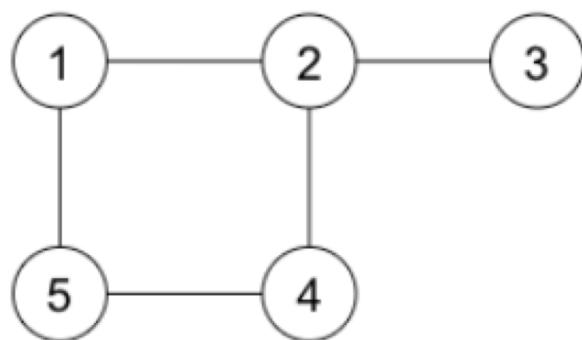
- Tô màu

4 chưa có màu



Kiểm tra đồ thị phân đôi

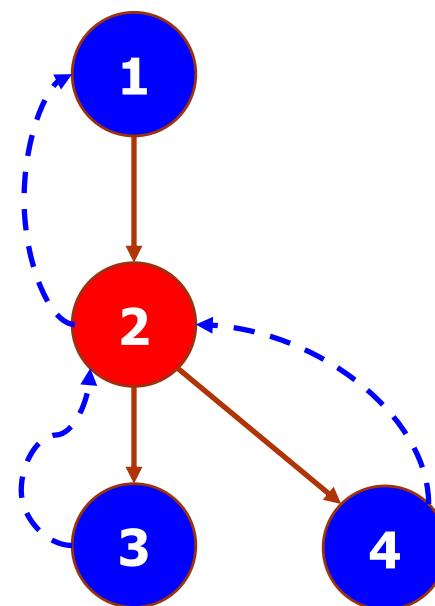
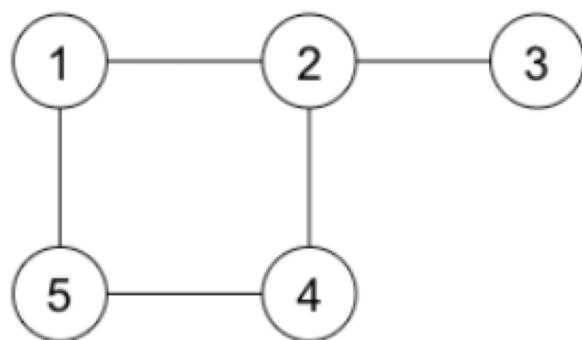
- Tô màu



4 chưa có màu, tô màu
XANH cho nó

Kiểm tra đồ thị phân đôi

- Tô màu

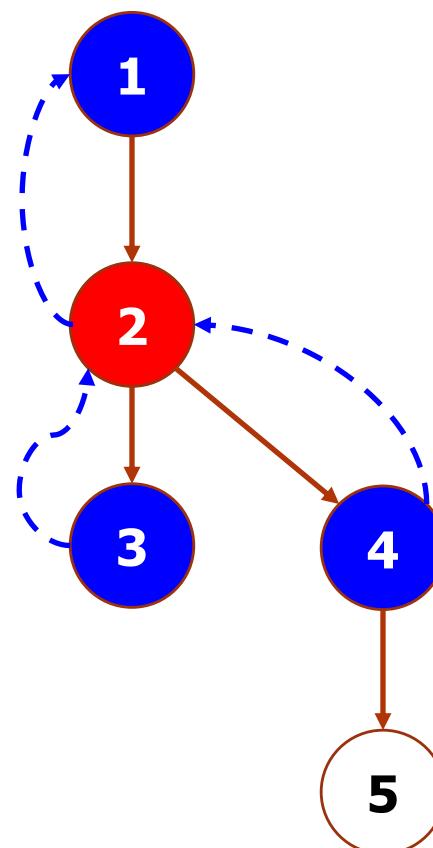
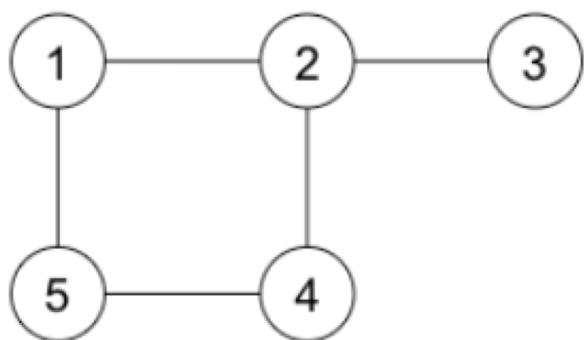


2 đã có màu ĐỎ khác
với màu XANH của 4

Kiểm tra đồ thị phân đôi

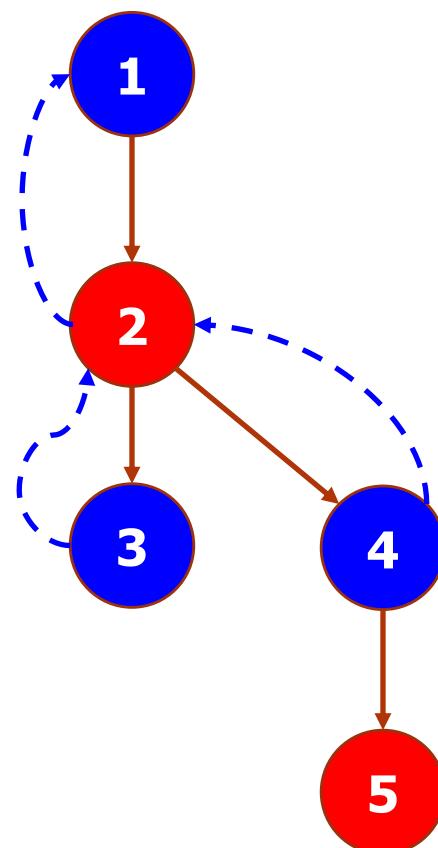
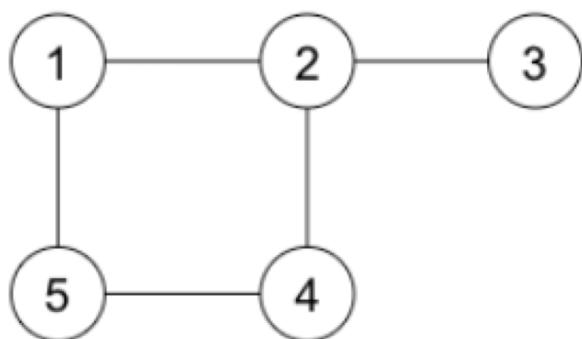
- Tô màu

5 chưa có màu



Kiểm tra đồ thị phân đôi

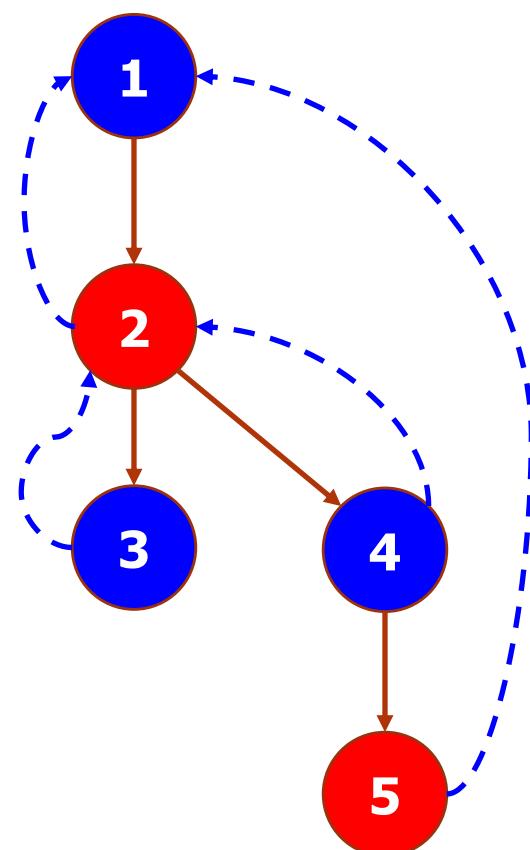
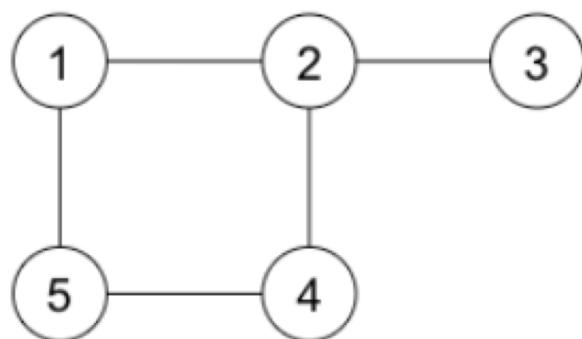
- Tô màu



5 chưa có màu, tô màu
ĐỎ cho nó

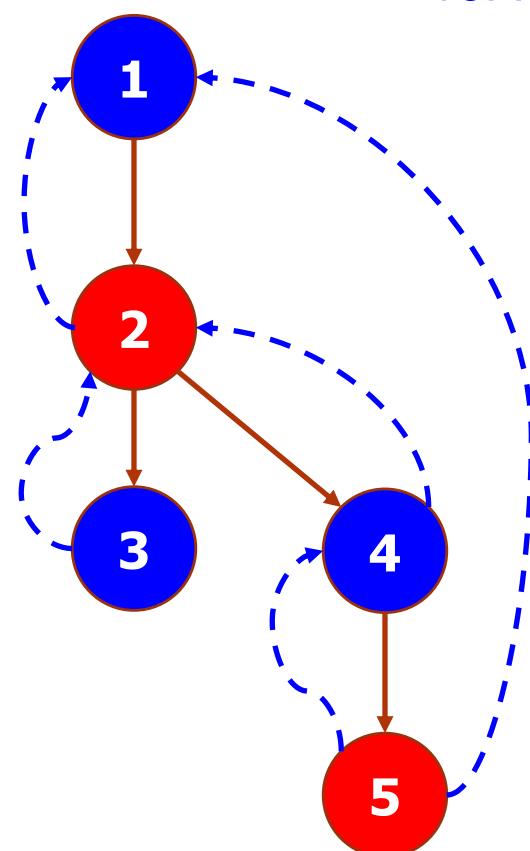
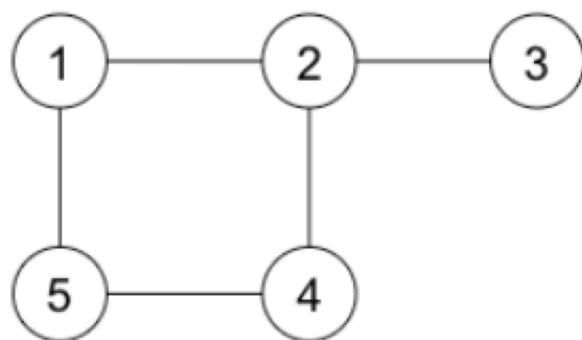
Kiểm tra đồ thị phân đôi

- Tô màu



Kiểm tra đồ thị phân đôi

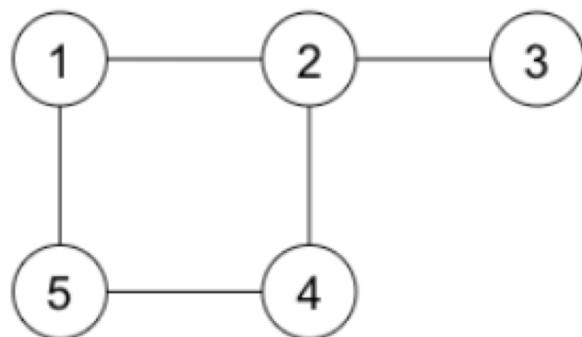
- Tô màu



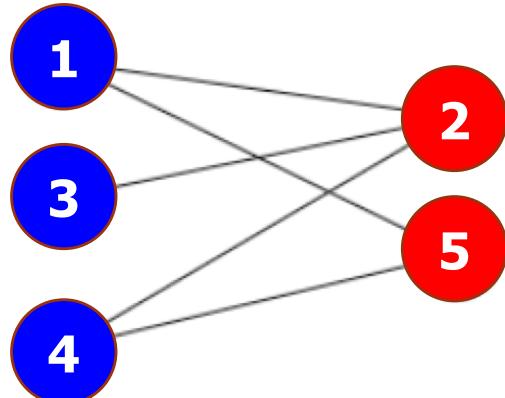
4 đã có màu XANH khác
với màu ĐỎ của 5

Kiểm tra đồ thị phân đôi

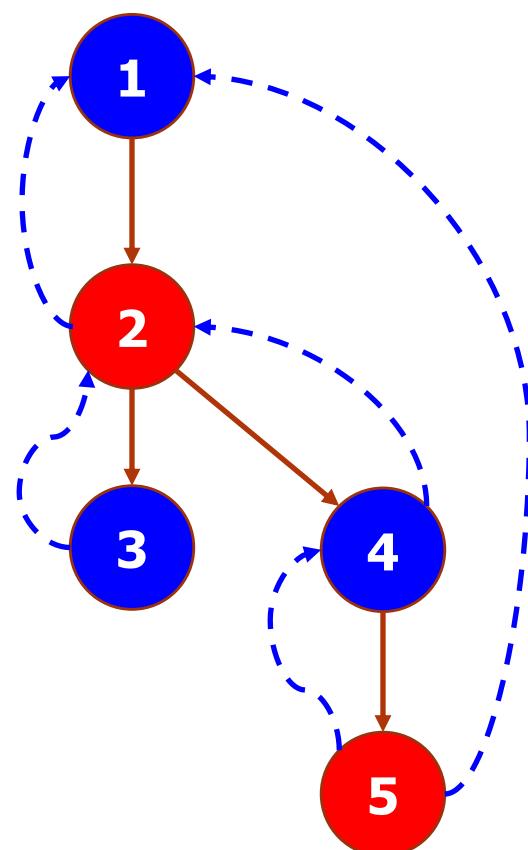
- Tô màu



Là đồ thị phân đôi



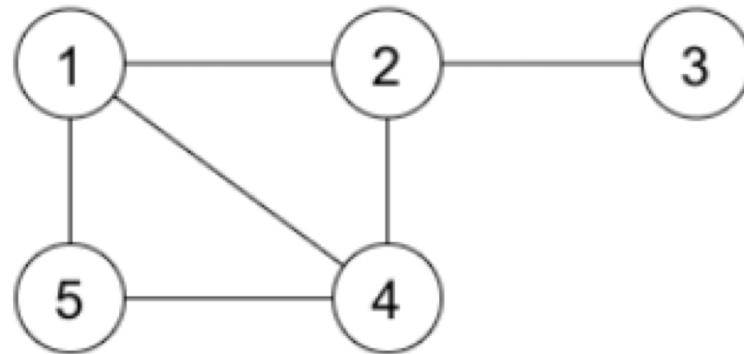
62



Tô màu thành công

Kiểm tra đồ thị phân đôi

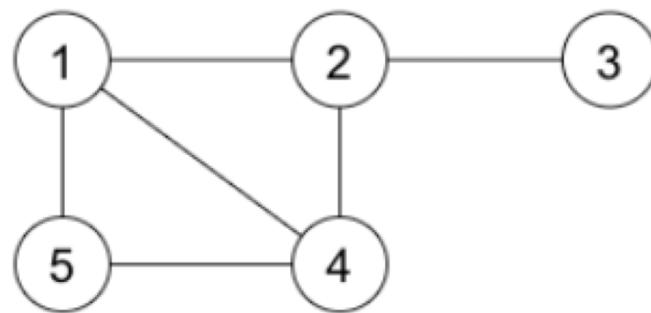
- Kiểm tra đồ thị này có phải là đồ thị phân đôi không



Kiểm tra đồ thị phân đôi

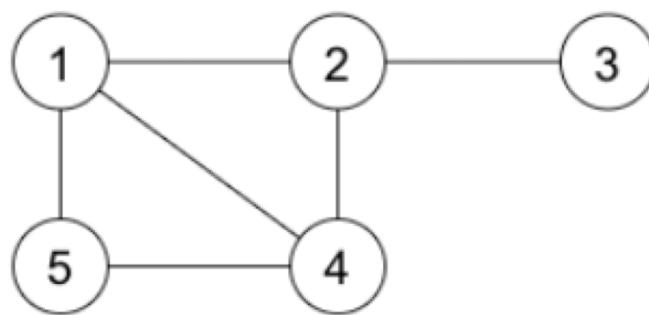
- Tô màu

1 chưa có màu



Kiểm tra đồ thị phân đôi

- Tô màu

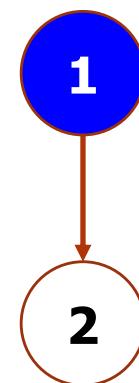
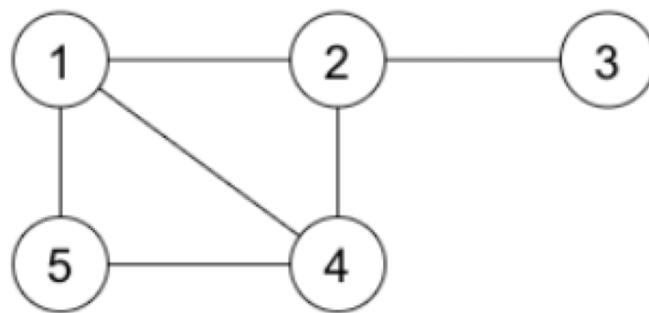


1 chưa có màu, tô màu
xanh cho nó

Kiểm tra đồ thị phân đôi

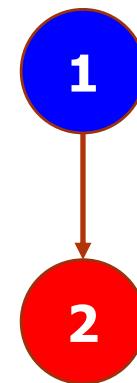
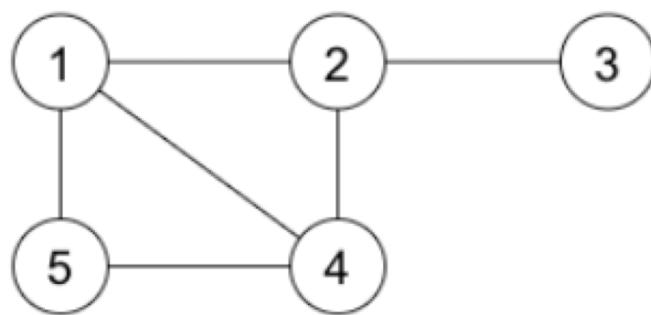
- Tô màu

2 chưa có màu



Kiểm tra đồ thị phân đôi

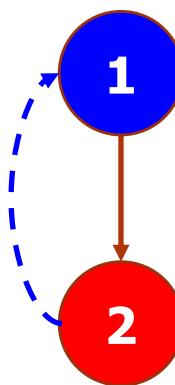
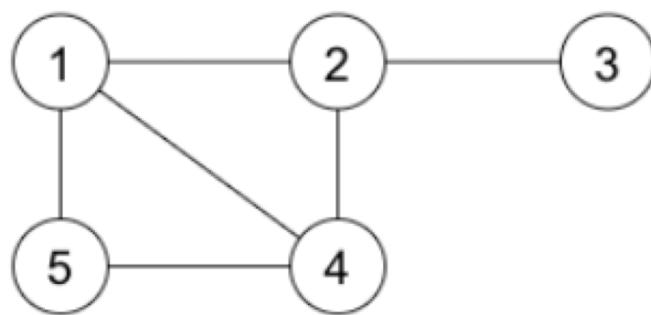
- Tô màu



2 chưa có màu, tô màu
ĐỎ cho nó

Kiểm tra đồ thị phân đôi

- Tô màu

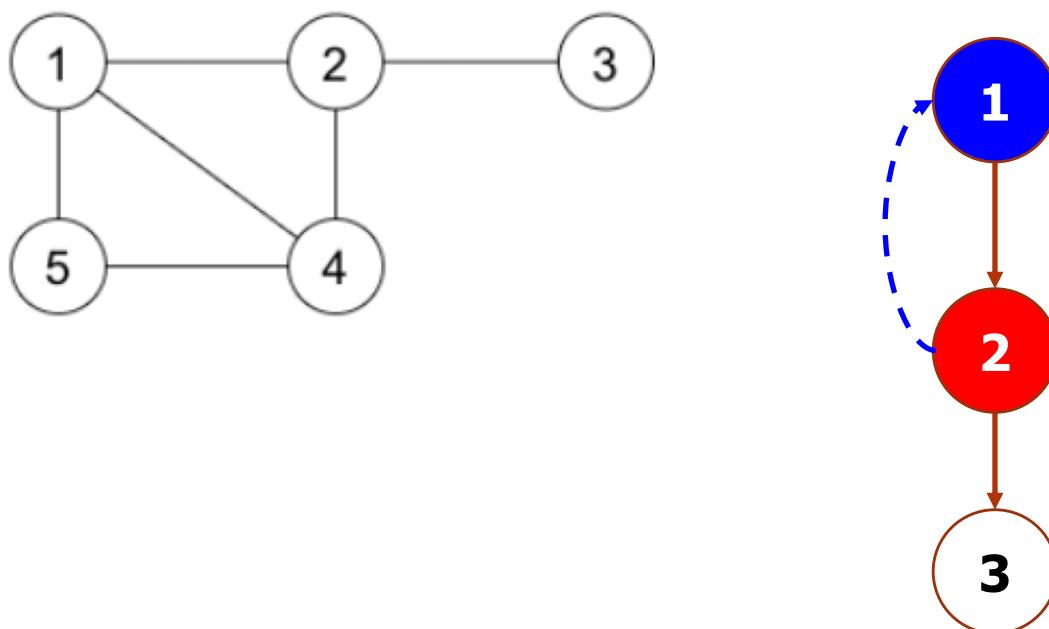


1 đã có màu XANH khác
với màu ĐỎ của 2

Kiểm tra đồ thị phân đôi

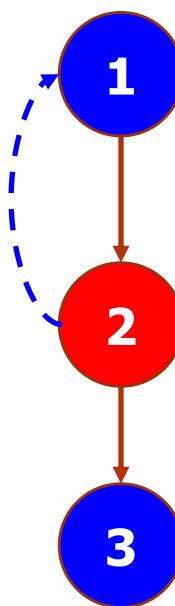
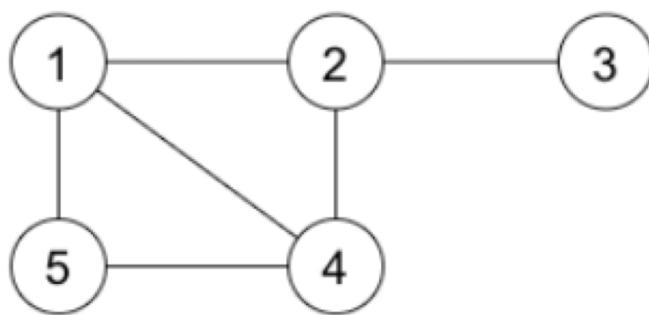
- Tô màu

3 chưa có màu



Kiểm tra đồ thị phân đôi

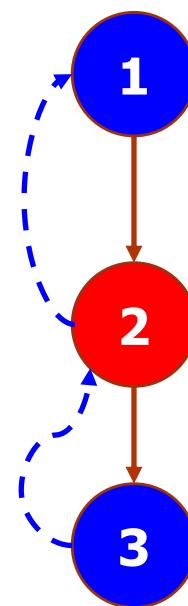
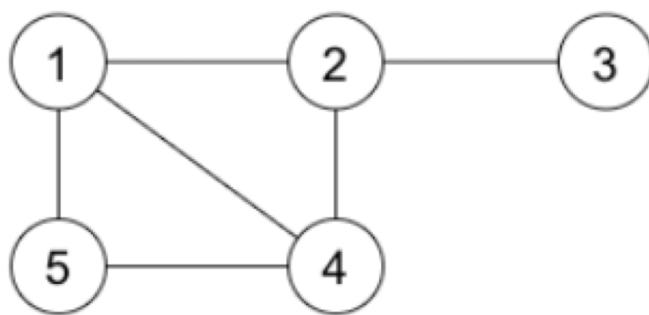
- Tô màu



3 chưa có màu, tô màu
XANH cho nó

Kiểm tra đồ thị phân đôi

- Tô màu

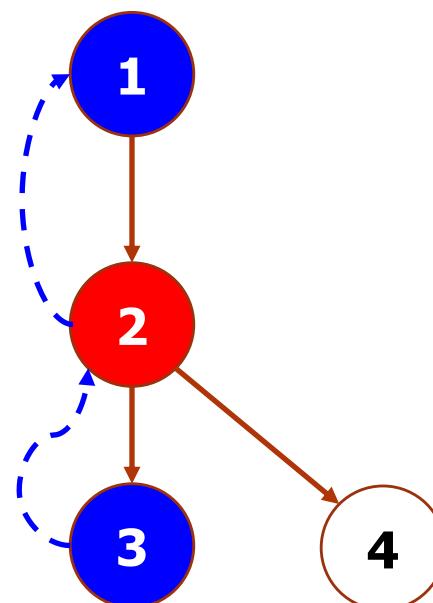
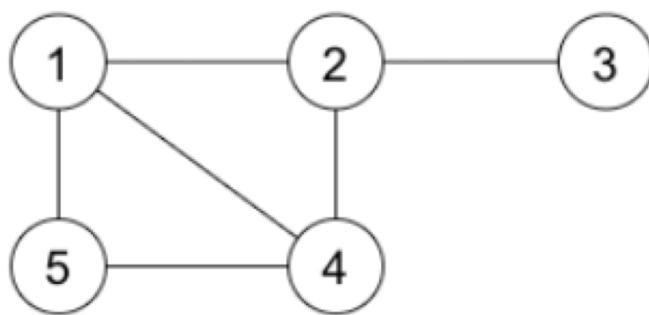


2 đã có màu ĐỎ, khác
với màu XANH của 3

Kiểm tra đồ thị phân đôi

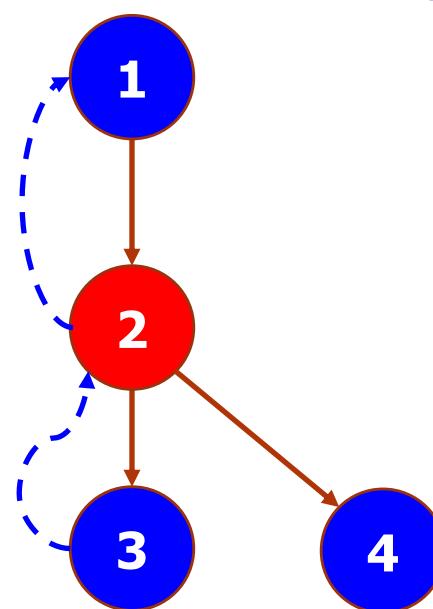
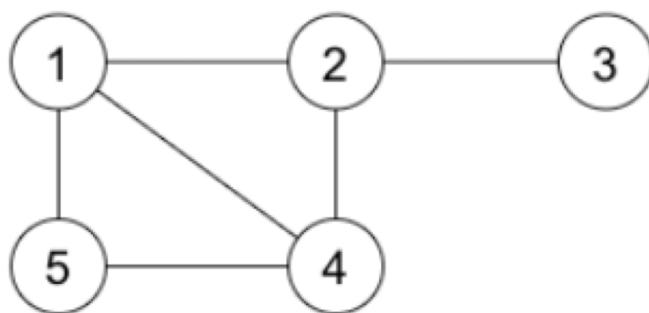
- Tô màu

4 chưa có màu



Kiểm tra đồ thị phân đôi

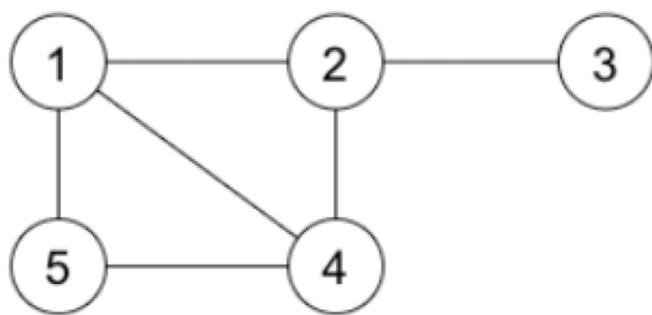
- Tô màu



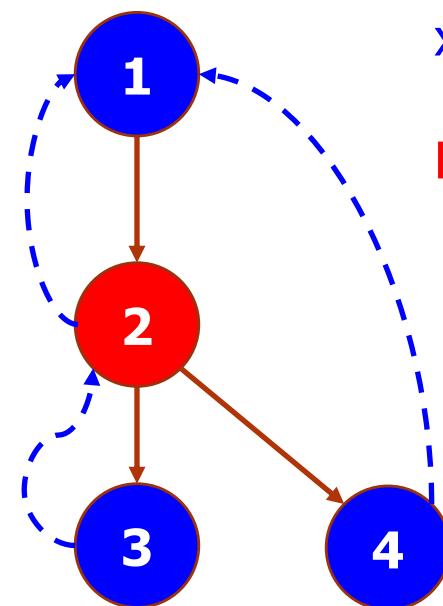
4 chưa có màu, tô màu
XANH cho nó

Kiểm tra đồ thị phân đôi

- Tô màu



KHÔNG là đồ thị phân đôi

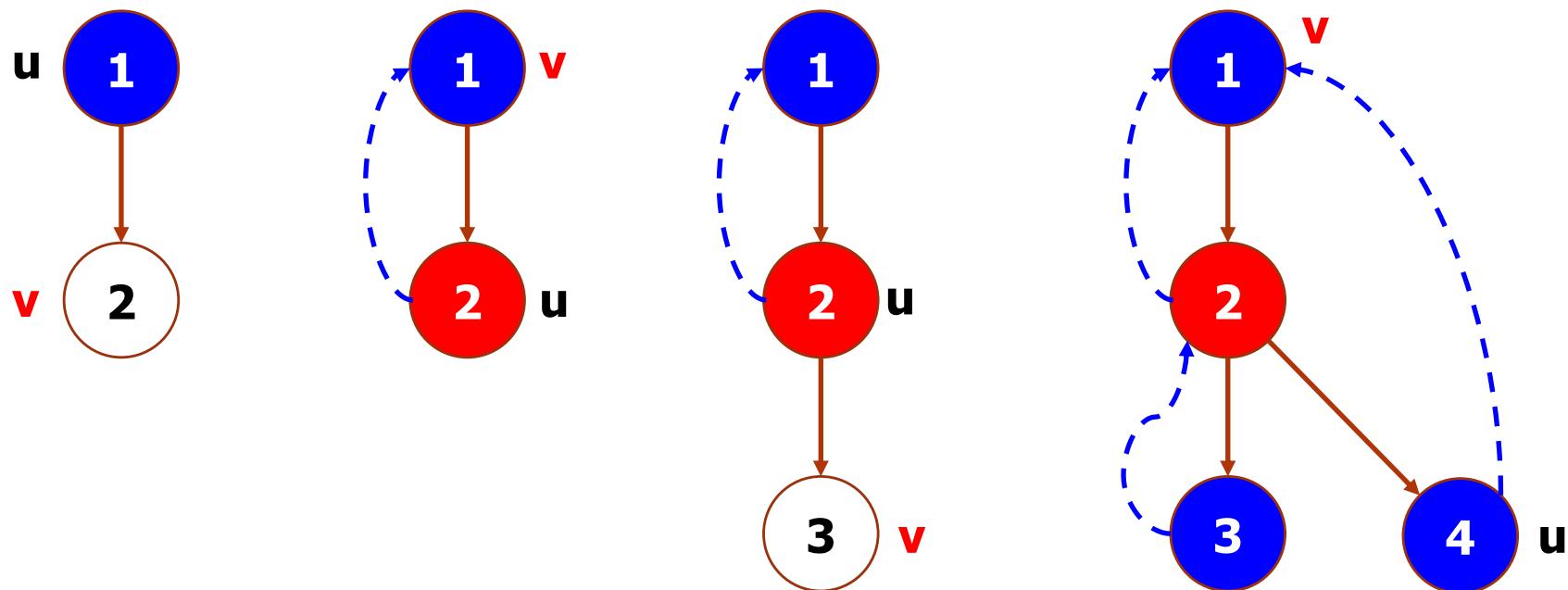


1 đã có màu XANH,
nhưng lại trùng với màu
xanh của 4

Không thể tô

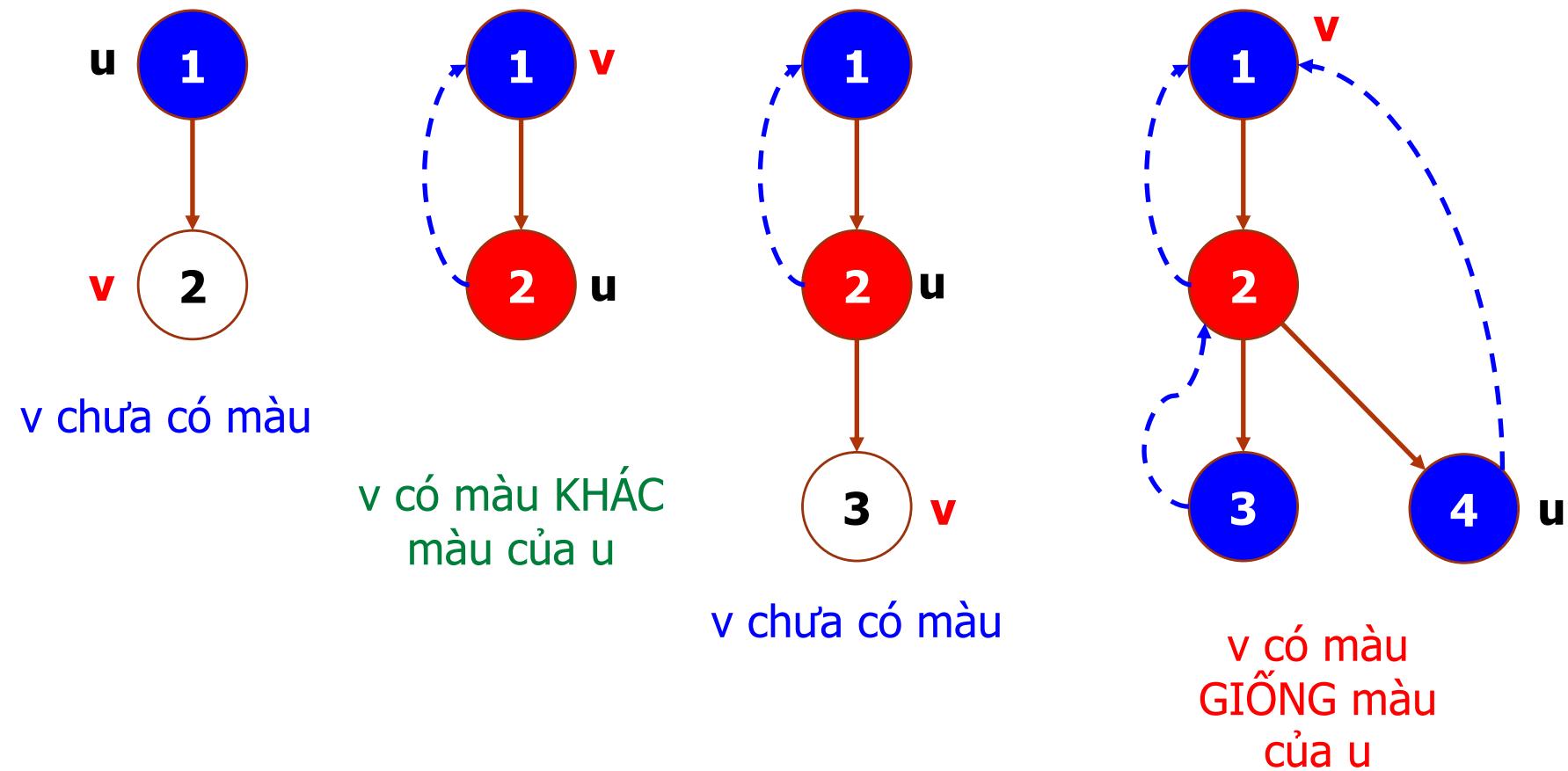
Kiểm tra đồ thị phân đôi

- Quan sát



Kiểm tra đồ thị phân đôi

- Quan sát



Kiểm tra đồ thị phân đôi

- Có 3 trường hợp xảy ra khi xét v
 - v chưa có màu => tô màu ngược lại với màu của u
 - v có màu GIỐNG với u => KHÔNG TÔ ĐƯỢC
 - v có màu KHÁC với u => bỏ qua
- Màu “ngược lại”
 - Nếu đặt S = XANH + ĐỎ
 - Thì
 - Ngược của XANH = ĐỎ = S - XANH
 - Ngược của ĐỎ = XANH = S - ĐỎ

Kiểm tra đồ thị phân đôi

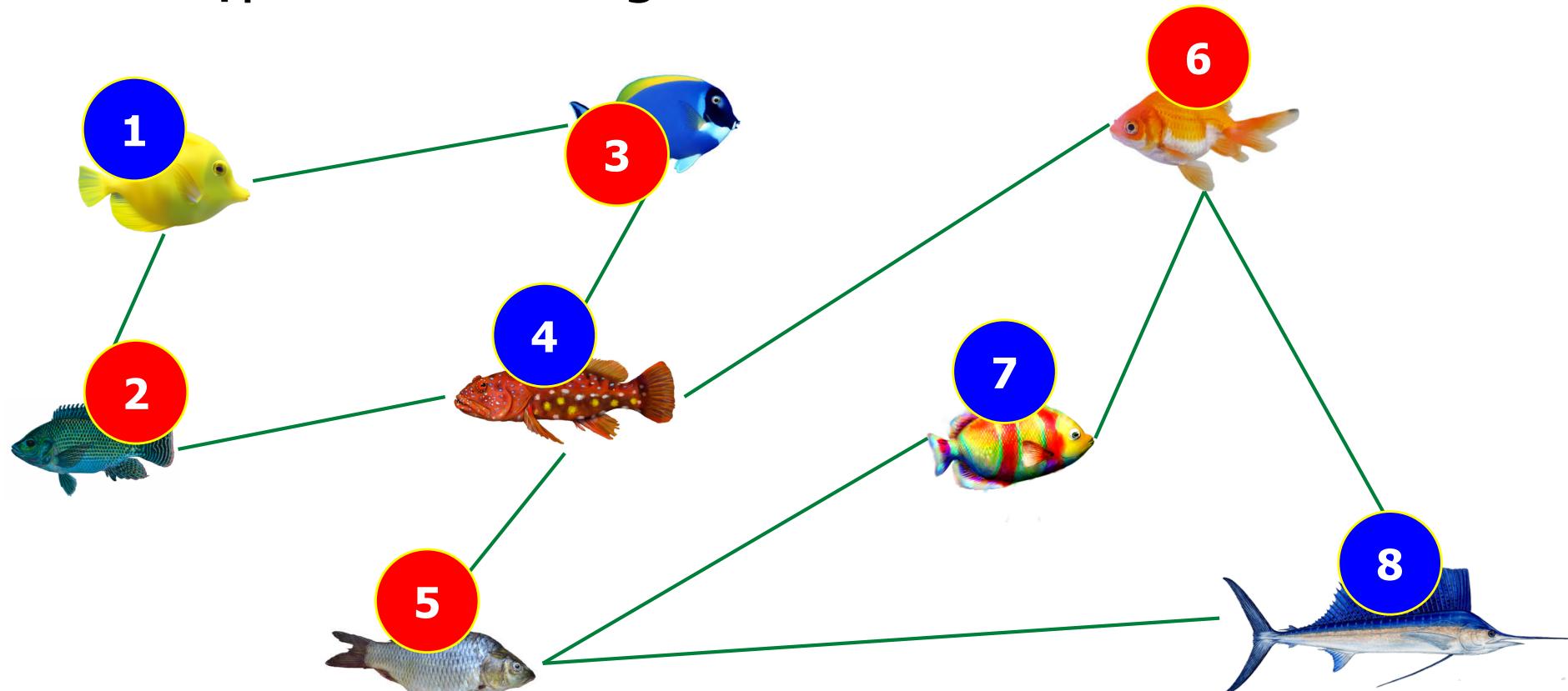
- Thuật toán

```
#define NO_COLOR -1  
#define BLUE      1  
#define RED       2
```

```
void colorize(u, c) {  
    color[u] = c;                                //1. Đang duyệt u  
  
    for (các đỉnh kề v của u)                    //2. Xét kề  
        if (v == NO_COLOR) {                      //2a. v chưa có màu  
            colorize(v, 3 - c);                  //tô màu màu ngược với c  
        } else if (color[v] == color[u])           //2b. màu v giống màu u  
            conflict = 1;                         //Đụng độ, không tô được  
        else {}                                 //2c. Màu khác nhau, bỏ qua  
}
```

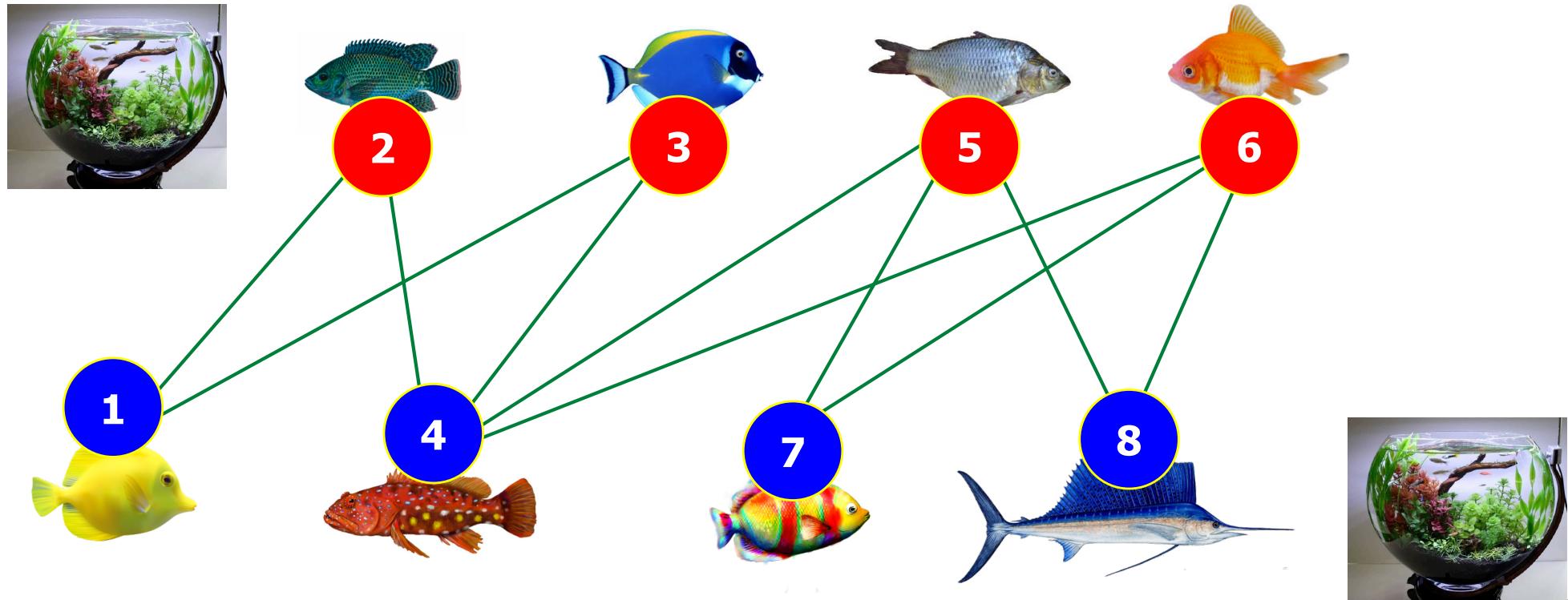
Kiểm tra đồ thị phân đôi

- Một số cặp cá sẽ đá nhau khi thả vào chung 1 hồ
- Ta chỉ có 2 hồ chứa cá
- Cặp các có đường nối sẽ đá nhau



Kiểm tra đồ thị phân đôi

- Một số cặp cá sẽ đá nhau khi thả vào chung 1 hồ
- Ta chỉ có 2 hồ chứa cá
- Cặp các có đường nối sẽ đá nhau



Hướng dẫn thực hành buổi 2

- Ôn lại bài thực hành buổi 1
 - Biểu diễn đồ thị bằng phương pháp ma trận kề
 - Khai báo cấu trúc Graph
 - Hàm init_graph()
 - Hàm add_edge()
 - Kết hợp đọc dữ liệu từ bàn phím + tập tin
 - freopen() + scanf()
 - Phương pháp liệt kê các đỉnh kề (không lặp lại) của 1 đỉnh
 - for + if + adjacent()

Hướng dẫn thực hành buổi 2

- Cài đặt các thuật toán duyệt đồ thị
 - BFS dùng hàng đợi, DFS dùng ngăn xếp
 - Xem lại học phần CTDL để biết cách cài đặt hàng đợi và ngăn xếp. Chọn 1 phương pháp đơn giản nhất để cài đặt, ví dụ: sử dụng mảng
 - Kiểu dữ liệu cho phần tử:
 - Số nguyên: duyệt bình thường (lưu các đỉnh)
 - Cấu trúc (2 số nguyên): nếu dựng cây
 - Chỉ cần cài đặt ĐỦ các hàm cần dùng
 - `make_null()`: khởi tạo rỗng
 - `empty()`: kiểm tra rỗng
 - `top()/front()`: xem phần tử trên đỉnh ngăn xếp/đầu hàng đợi
 - `push()/enqueue()`: thêm phần tử vào ngăn xếp/hàng đợi
 - `pop()/dequeue()`: xoá phần tử trên đỉnh ngăn xếp/đầu hàng đợi

Hướng dẫn thực hành buổi 2

- Cài đặt các thuật toán duyệt đồ thị
 - BFS dùng hàng đợi, DFS dùng ngăn xếp
 - Xem lại lý thuyết về các thuật toán duyệt đồ thị (Slides 1)
 - Các biến toàn cục:
 - `int mark[MAX_N]`: dùng để đánh dấu 1 đỉnh đã duyệt chưa
 - `Stack S`: ngăn xếp dành cho DFS
 - `Queue Q`: hàng đợi dành cho BFS
 - Sẽ có lợi khi ta muốn duyệt toàn bộ đồ thị bằng cách gọi BFS/DFS nhiều lần

Hướng dẫn thực hành buổi 2

- Cài đặt các thuật toán duyệt đồ thị
 - BFS dùng hàng đợi, DFS dùng ngăn xếp

```
void BFS(Graph *pG, int s) {  
    enqueue(&Q, s); //1. Đưa s vào hàng đợi  
  
    while (!empty(&Q)) { //2. Lặp đến khi Q rỗng  
        int u = front(&Q); //2a. Lấy pt đầu hàng đợi,  
        dequeue(&Q); //rồi bỏ nó ra khỏi Q Luôn  
        if (mark[u] == 1) //2b. KTra đã duyệt chưa  
            continue; // duyệt rồi thì bỏ qua  
        mark[u] = 1; //2c. Đánh dấu duyệt u  
        for (int v = 1; v <= pG->n; v++)  
            if (adjacent(pG, u, v)) //2d. Xét các đỉnh kề của u  
                enqueue(&Q, v);  
    }  
}
```

Hướng dẫn thực hành buổi 2

- Cài đặt các thuật toán duyệt đồ thị
 - Sử dụng BFS/DFS duyệt từ đỉnh s

```
int main() {  
    Graph G;  
    int s;  
    //Đọc dữ liệu  
    ...  
    make_null(&Q);           //1. Khởi tạo hàng đợi rỗng  
    for (int u = 1; u <= pG->n; u++) //2. Khởi tạo mảng mark  
        mark[u] = 0;  
    BFS(&G, s);            //3. Gọi BFS để duyệt từ s  
    ...                    //4. Kiểm tra mảng mark  
}
```

Hướng dẫn thực hành buổi 2

- Cài đặt các thuật toán duyệt đồ thị
 - Duyệt cả đồ thị để đếm số BPLT

```
int main() {
    //Khai báo, đọc dữ liệu
    ...
    make_null(&Q);                      //1. Khởi tạo hàng đợi rỗng
    for (int u = 1; u <= pG->n; u++) //2. Khởi tạo mảng mark
        mark[u] = 0;
    int cnt = 0;                         //3. Khởi tạo biến đếm cnt
    for (int u = 1; u <= pG->n; u++) //4. Tìm đỉnh chưa duyệt
        if (mark[u] == 0) {            // Nếu u chưa duyệt, duyệt nó
            BFS(&G, u); cnt++;       //5. Mỗi lần duyệt được 1 BPLT
        }
}
```

Hướng dẫn thực hành buổi 2

- Cài đặt các thuật toán duyệt đồ thị
 - DFS đệ quy: không cần CTDL ngăn xếp

```
void DFS(Graph *pG, int u) {  
    mark[u] = 1;                                //1. Duyệt u  
  
    for (int v = 1; v <= pG->n; v++) //2. Xét các đỉnh kề của u  
        if (adjacent(pG, u, v))  
            if (mark[v] == 0)  
                DFS(pG, v);  
}
```

Hướng dẫn thực hành buổi 2

- Cài đặt thuật toán kiểm tra đồ thị (CÓ HƯỚNG) chứa chu trình
 - Định nghĩa 3 màu: WHITE, GRAY, BLACK
 - Các biến toàn cục
 - `int color[MAX_N]`: lưu màu của các đỉnh
 - `int has_circle`: có chu trình hay không
 - Cài đặt hàm `DFS(Graph *pG, int u)` theo lý thuyết
 - Sử dụng `for + DFS ()` để kiểm tra trên toàn đồ thị

Hướng dẫn thực hành buổi 2

- Cài đặt thuật toán kiểm tra đồ thị (VÔ HƯỚNG) chứa chu trình
 - Định nghĩa 3 màu
 - Các biến toàn cục
 - `int color[MAX_N]`: lưu màu của các đỉnh
 - `int has_circle`: có tạo thành chu trình không
 - Cài đặt hàm `DFS(Graph *pG, int u, int p)` theo lý thuyết
 - Có bổ sung tham số `p` để kiểm tra `v == p` không

Hướng dẫn thực hành buổi 2

- Cài đặt thuật toán kiểm tra phân đôi
 - Định nghĩa 3 màu
 - Các biến toàn cục
 - `int color[MAX_N]`: lưu màu của các đỉnh
 - `int conflict`: lưu sự đụng độ khi tô màu
 - Cài đặt hàm `colorize(Graph *pG, int u, int c)` theo lý thuyết
 - Sử dụng `for + colorize()` để kiểm tra trên toàn bộ đồ thị

Hướng dẫn thực hành buổi 2

- Cài đặt thuật toán Tarjan
 - Xem lại lý thuyết (Slides 2)
 - Các biến toàn cục
 - `int num[]`: lưu chỉ số của các đỉnh
 - `int min_num[]`: lưu chỉ số nhỏ nhất của các đỉnh
 - `int k`: chỉ số, tăng tự động
 - `Stack S`: lưu các đỉnh thuộc về 1 BPTL mạnh
 - Cài đặt hàm `SCC(Graph *pG, int u)` theo lý thuyết
 - Để duyệt toàn đồ thị sử dụng `for + SCC()`

Hướng dẫn thực hành buổi 2

- Cài đặt các thuật toán duyệt đồ thị
 - Duyệt cả đồ thị để đếm số BPLT mạnh

```
int main() {
    //Khai báo, đọc dữ liệu
    ...
    make_null(&Q);                      //1. Khởi tạo hàng đợi rỗng
    for (int u = 1; u <= pG->n; u++) //2. Khởi tạo mảng num
        num[u] = -1;
    int k = 1;                           //3. Khởi tạo k
    for (int u = 1; u <= pG->n; u++) //4. Tìm đỉnh chưa duyệt
        if (num[u] == -1) {           // Nếu u chưa duyệt, duyệt nó
            SCC(&G, u);             //5. Đếm số SCC trong hàm SCC
        }
}
```