**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**
**ĐẠI HỌC KHOA HỌC TỰ NHIÊN**


# BÁO CÁO ĐỒ ÁN
# PHÂN HOẠCH ĐỒ THỊ
# VÀ
# BÀI TOÁN TÔ MÀU

**Giảng viên:**

PGS.TS: Nguyễn Đình Thúc

**Thực hiện đề tài:**

22C15018: Phạm Minh Thạch
22C15045: Nguyễn Thị Hoàng Trang

**TP.HCM, ngày 26 tháng 03 năm 2023**

# MỤC LỤC

# GIỚI THIỆU

Xin chào quý thầy cô, ở đồ án này, nhóm em sẽ trình bày gồm 2 nội dung, bao gồm:

- Thuật toán phân hoạch trên đồ thị.
- Giải bài toán tô màu bằng phương pháp phân hoạch đồ thị.

# 1. THUẬT TOÁN PHÂN HOẠCH TRÊN ĐỒ THỊ

## 1.1. Mô tả

Một thành phần liên thông của một đồ thị vô hướng là một đồ thị con trong đó giữa bất kì hai đỉnh nào đều có đường đi đến nhau, và không thể nhận thêm bất kì một đỉnh nào mà vẫn duy trì tính chất trên. Thành phần liên thông có nhiều ứng dụng trong các lĩnh vực như:

- Xử lý ảnh: gom nhóm các pixel trong cùng một vật thể thông qua sự tương đồng về màu sắc.
- Mạng xã hội: xác định những người dùng có mối liên hệ với nhau thông qua các mối quan hệ chung hoặc sở thích chung.
- Máy học: gom nhóm những điểm dữ liệu ở gần nhau hoặc có nhiều đặc trưng tương tự nhau.

## 1.2. Cài đặt

Trong đoạn chương trình bên dưới, hàm *connected_components* nhận vào một đồ thị và trả về danh sách các thành phần liên thông trong đồ thị. Hàm *dfsvisit* nhận vào một đỉnh và trả về thành phần liên thông xuất phát từ đỉnh đó.

```
def connected_components(self):                  self: graph được lưu ở dạng danh sách
      for aVertex in self:                       đỉnh kề
          aVertex.setVisited(False)
          aVertex.setPrev(-1)
      components = []                            components: các thành phần liên thông
      for aVertex in self:
          print(aVertex)                         In đỉnh hiện tại và các đỉnh liền kề
          if aVertex.getVisited() == False:
              component = []
              self.dfsvisit(aVertex, component)  Hàm dfsvisit: tìm thành phần liên thông
                                                 xuất phát từ đỉnh vertex
              components.append(component)       Thêm thành phần liên thông tìm được vào
                                                 kết quả
      return components

def dfsvisit(self, startVertex, component):      Hàm dfsvisit: tìm thành phần liên thông
      startVertex.setVisited(True)               xuất phát từ một đỉnh cho trước

      component.append(startVertex.getId())      Thêm đỉnh vào thành phần liên thông

      for nbr in startVertex.getConnections():   Duyệt qua các đỉnh kề của và thêm các
          if nbr.getVisited() == False:          đỉnh kề vào thành phần liên thông
              nbr.setPrev(startVertex)
              self.dfsvisit(nbr, component)
      startVertex.setVisited(True)
```

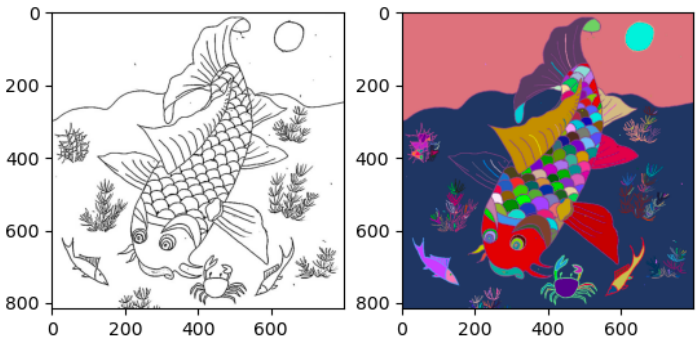
### 1.3. Độ phức tạp thuật toán

Thuật toán sẽ duyệt qua tất cả các đỉnh và tất cả các cạnh của đồ thị, vì vậy độ phức tạp sẽ là O(V+E), trong đó V là số đỉnh của đồ thị và E là cạnh của đồ thị.

## 2. BÀI TOÁN TÔ MÀU

### 2.1. Mô tả

Bài toán mà ta giải quyết tương tự như bài toán bé tập tô màu, ta nhận được một bức ảnh trắng đen. Các nét vẽ sẽ chia bức ảnh thành các khoảng trắng tách biệt nhau. Nhiệm vụ của chúng ta là tô màu vào các khoảng trắng, sao cho mỗi khoảng trắng sẽ được tô bởi chỉ một màu. Màu sắc trong quá trình tô sẽ được chọn ngẫu nhiên.




### 2.2. Input

Input là một bức ảnh trắng đen dạng RGB.

### 2.3. Output

Output là một bức ảnh màu dạng RGB.

### 2.4. Phương pháp cài đặt

Ta sẽ tiến hành tô màu bức ảnh theo các bước như sau:

*Bước 1: chuyển ảnh từ dạng RGB về dạng grayscale*

Ảnh ban đầu có dạng RGB, mỗi điểm ảnh sẽ có 3 giá trị gồm Red, Green và Blue. Để thuận tiện cho việc tính toán, ta sẽ chuyển ảnh về dạng grayscale, mỗi điểm ảnh chỉ có một giá trị là gray.

*Bước 2: Chuyển ảnh từ dạng grayscale về dạng ma trận nhị phân*

Trong bài toán này, chúng ta có hai đối tượng chính, đó là nét vẽ và khoảng trắng. Để thuận tiện cho việc gán nhãn, ta sẽ chuyển ảnh về dạng ma trận nhị phân. Khi điểm ảnh là nét vẽ thì ô tương ứng trên ma trận sẽ có giá trị 1, khi điểm ảnh là khoảng trắng ô sẽ có giá trị 0.

*Bước 3: Thực hiện gán nhãn cho các thành phần liên thông trên ma trận nhị phân*

Trên ma trận nhị phân, mỗi ô sẽ có cạnh chung hoặc đỉnh chung với 8 ô xung quanh nó. Hai ô được gọi là *giao nhau* nếu chúng có cạnh chung hoặc đỉnh chung.

Để đưa bài toán về dạng đồ thị, ta xem mỗi ô như một node của đồ thị. Hai ô liền kề nhau nếu chúng giao nhau và có cùng giá trị. Một thành phần liên thông sẽ bao gồm các ô liền kề nhau.

Để thực hiện gán nhãn, ta lần lượt duyệt qua các ô, nếu ô đó có giá trị 0 và vẫn chưa gán nhãn, ta sẽ thực hiện gán nhãn cho thành phần liên thông xuất phát từ ô đó. Thuật toán dừng lại khi ta duyệt qua tất cả các ô.

*Bước 4: Thực hiện tô màu theo ma trận đã gán nhãn*

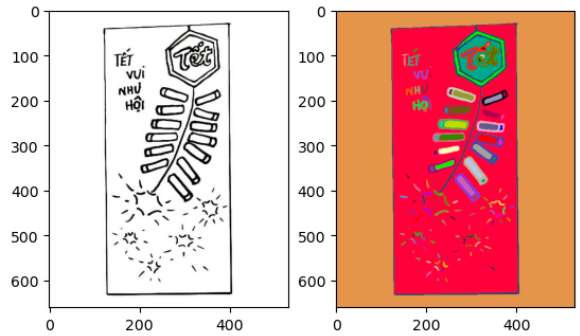
Đầu tiên, ta tạo một ma trận có kích thước H x W x 3. Ma trận này sau đó sẽ được chuyển đổi thành ảnh màu RGB.

Giả sử, sau bước gán nhãn, ta có được số nhãn cần gán là K. Ta sẽ tạo ngẫu nhiên K màu tương ứng với K nhãn. Ta duyệt các ô trên ma trận nhãn, căn cứ vào giá trị của ô để xác định giá trị màu sắc cho ô tương ứng trên ma trận mới được tao ra.

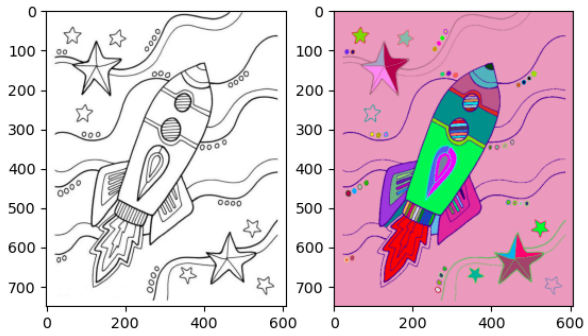
*Bước 5: Chuyển đổi ma trận về dạng RGB*
Sau khi đã hoàn thành việc điền các giá trị màu sắc cho các ô trên ma trận, ta chuyển đổi ma trận sang dạng ảnh RGB.

### 2.5. Một số kết quả của chương trình

Bên dưới là hai kết quả của chương trình. Ảnh trắng đen là input, ảnh màu là output.




*Thực hiện tô màu trên một ảnh thiệp Tết*



*Thực hiện tô màu trên một ảnh tàu vũ trụ*

### 2.6. Một số vấn đề gặp phải và cách giải quyết

Nếu ta sử dụng Depth First Search để cài đặt thao tác gán nhãn, thì khi chạy trên những ảnh có kích thước lớn ta sẽ phải lỗi *maximum recursion depth exceeded* vì chương trình gọi đệ quy quá nhiều lần. Để giải quyết vấn đề này, nhóm đã chuyển sang cài đặt thao tác gán nhãn bằng thuật toán Breadth First Search.

### 2.7. Mã nguồn

Mã nguồn của chương trình tô màu được lưu tại: *https://github.com/thachmpham/graph-color* .

## 3. NGUỒN THAM KHẢO

- https://en.wikipedia.org/wiki/Graph_coloring
- https://en.wikipedia.org/wiki/Connected-component_labeling