Báo cáo Project 1

1. THÔNG TIN SINH VIÊN

Quách Trung Tín - 21130061

2. THỐNG KÊ MỨC ĐỘ HOÀN THÀNH

STT	Các chức năng	Mức độ hoàn thành	Sinh viên thực hiện
1	Checkered Board	100%	Quách Trung Tín
2	Color Correction	70%	Quách Trung Tín
3	Rotate Image	90%	Quách Trung Tín
4	Corner Line	100%	Quách Trung Tín
5	Gradient	100%	Quách Trung Tín
6	Letter	100%	Quách Trung Tín
7	Color Separate	100%	Quách Trung Tín
8	Find secret by subtract	100%	Quách Trung Tín

3. PHÂN TÍCH VÀ MÔ TẢ THUẬT TOÁN

a. Checkered Board

- Mô tả thuật toán:

1. Khởi tạo Kích Thước Bàn Cờ và Ô Vuông:
    - chessboard_size = 8: Xác định kích thước bàn cờ, ở đây là 8x8 ô vuông.
    - square_size = 50: Xác định kích thước của mỗi ô vuông, mỗi ô có chiều dài và chiều rộng là 50 pixel.
2. Tạo Ảnh Trống:
    - image = np.zeros((chessboard_size * square_size, chessboard_size * square_size, 3), dtype=np.uint8):
        - Tạo một hình ảnh trống có kích thước phù hợp với bàn cờ, với số lượng pixel là square_size * chessboard_size cho cả chiều dài và chiều rộng.
        - np.zeros tạo ra một mảng toàn giá trị 0 (tức là màu đen) với 3 kênh màu (RGB), mỗi kênh có giá trị từ 0 đến 255.
3. Duyệt Qua Từng Ô Vuông:
    - for row in range(chessboard_size) và for col in range(chessboard_size):
        - Vòng lặp qua từng hàng và cột của bàn cờ.
4. Kiểm Tra Ô Vuông Là Trắng Hay Đen:
    - if (row + col) % 2 == 0:

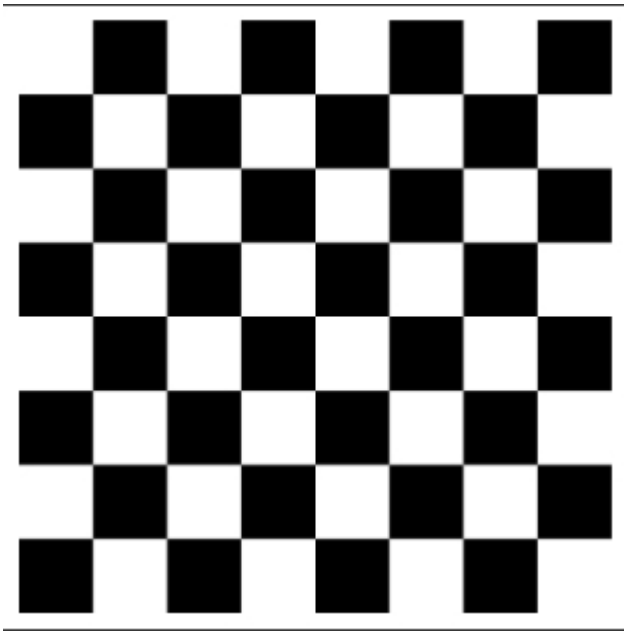
        - Điều kiện này kiểm tra xem tổng của chỉ số hàng và chỉ số cột có chia hết cho 2 không. Nếu có, ô vuông sẽ được tô màu trắng, nếu không thì giữ màu đen.
5. Xác Định Tọa Độ Và Tô Màu Ô Vuông:
    - top_left_x và top_left_y: Xác định tọa độ của góc trên bên trái của ô vuông hiện tại.
    - bottom_right_x và bottom_right_y: Xác định tọa độ của góc dưới bên phải của ô vuông hiện tại.
    - image[top_left_y:bottom_right_y, top_left_x:bottom_right_x] = [255, 255, 255]:
        - Tô màu ô vuông bằng màu trắng (giá trị [255, 255, 255] cho kênh RGB).
6. Hiển Thị Bàn Cờ:
    - plt.imshow(image) hiển thị hình ảnh bàn cờ vừa tạo.
    - plt.axis('off') tắt hiển thị trục tọa độ..

Kết quả là một hình ảnh bàn cờ 8x8 với các ô vuông đen và trắng xen kẽ.

- Code:

```
# Câu 1
# Kích thước bàn cờ (số ô vuông)
chessboard_size = 8
# Kích thước mỗi ô vuông (tính bằng pixel)
square_size = 50

# Tạo ảnh trống với kích thước phù hợp, ban đầu ảnh là màu đen (tất cả giá trị là 0)
# Ảnh có 3 kênh màu (RGB), mỗi ô vuông có chiều cao và chiều rộng bằng square_size * chessboard_size
img = np.zeros((chessboard_size * square_size, chessboard_size * square_size, 3), dtype=np.uint8)

# Duyệt qua từng ô vuông của bàn cờ
for row in range(chessboard_size):
    for col in range(chessboard_size):
        # Kiểm tra xem ô vuông hiện tại là trắng hay đen
        if (row + col) % 2 == 0:
            # Xác định tọa độ của góc trên bên trái và góc dưới bên phải của ô vuông
            top_left_x = col * square_size
            top_left_y = row * square_size
            bottom_right_x = (col + 1) * square_size
            bottom_right_y = (row + 1) * square_size

            # Vẽ ô vuông trắng (màu [255, 255, 255]) lên ảnh
            img[top_left_y:bottom_right_y, top_left_x:bottom_right_x] = [255, 255, 255]

plt.imshow(img)
plt.axis('off')
```

- Kết quả:



b.Color Correction

- Mô tả thuật toán:

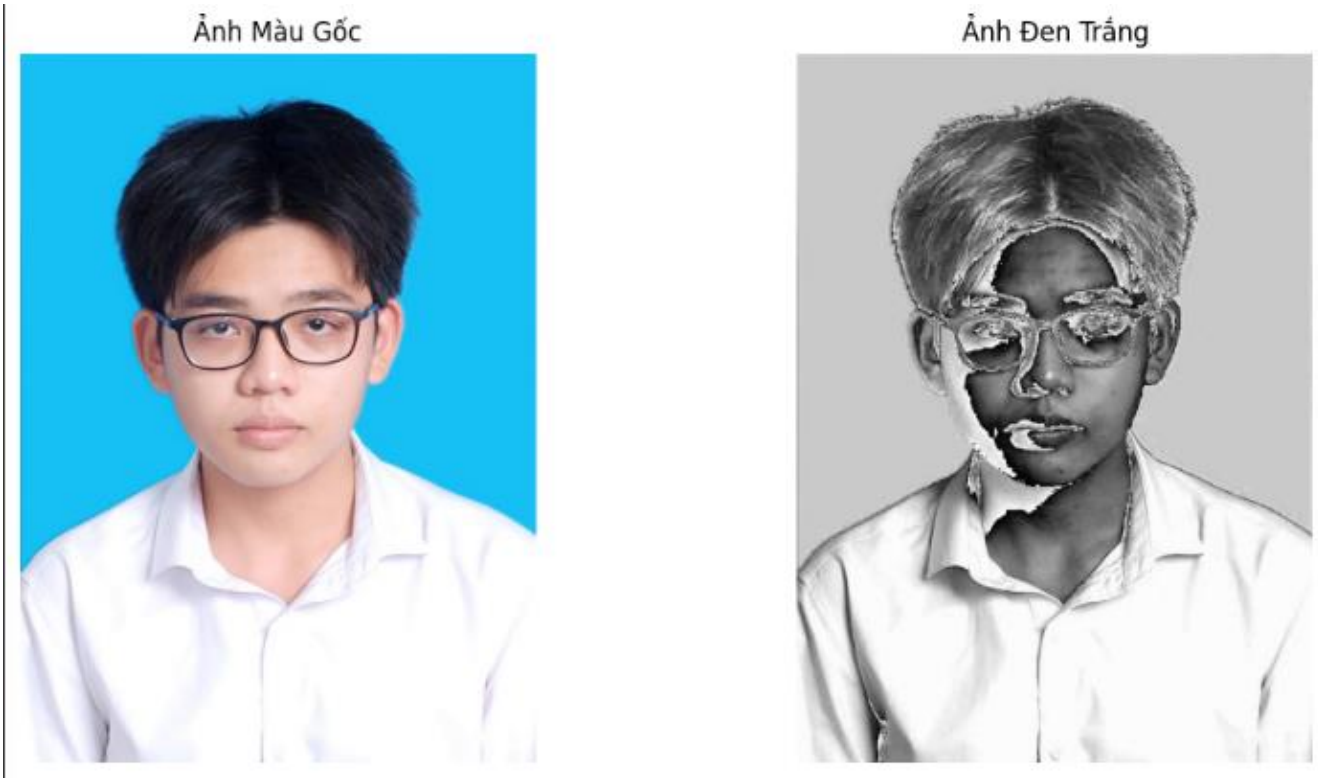
1. Lấy Kích Thước Ảnh:
    - height, width, channels = image.shape: Lấy kích thước của ảnh (chiều cao, chiều rộng) và số kênh màu (thường là 3 cho RGB) từ ảnh gốc.
2. Tạo Mảng Cho Ảnh Đen Trắng:
    - image_gray = np.zeros((height, width), dtype=np.uint8): Tạo một mảng trống (ban đầu là màu đen) có cùng kích thước với ảnh gốc để lưu trữ ảnh đen trắng.
3. Chuyển Đổi Ảnh Màu Sang Ảnh Đen Trắng:
    - Vòng lặp for i in range(height) và for j in range(width):
        - Duyệt qua từng pixel của ảnh gốc.
    - r = image[i, j, 0], g = image[i, j, 1], b = image[i, j, 2]: Lấy giá trị của ba kênh màu (R, G, B) tại vị trí pixel (i, j).
    - gray_value = int((r + g + b) / 3): Tính giá trị độ sáng của pixel bằng cách lấy trung bình cộng của các giá trị R, G, B.
    - image_gray[i, j] = gray_value: Gán giá trị độ sáng này cho pixel tương ứng trên ảnh đen trắng.
4. Hiển Thị Ảnh Gốc Và Ảnh Đen Trắng:
    - plt.figure(figsize=(12, 6)): Tạo một khung hình với kích thước 12x6 inch để hiển thị ảnh.
    - plt.subplot(1, 2, 1) và plt.subplot(1, 2, 2): Tạo hai ô (subplot) để hiển thị ảnh gốc và ảnh đen trắng.
    - plt.imshow(image) và plt.imshow(image_gray, cmap='gray'): Hiển thị ảnh gốc và ảnh đen trắng. cmap='gray' dùng để đảm bảo ảnh đen trắng được hiển thị với tông màu xám.
    - plt.title("..."): Đặt tiêu đề cho từng ảnh.
    - plt.axis('off'): Tắt hiển thị trục tọa độ.

Kết quả cuối cùng là hai ảnh được hiển thị cạnh nhau: ảnh gốc ở bên trái và ảnh đen trắng được tạo ra từ ảnh gốc ở bên phải.

- Code:

```python
# Câu 2
image_path = 'C:\\Users\\Trung Tin\\Tài liệu\\IUH\\Xử lý ảnh\\Labs\\Pictures\\image.jpg'
image = plt.imread(image_path)
if image is None:
    print("Hình ảnh không đọc được. Vui lòng kiểm tra lại đường dẫn.")
else:
    # Lấy chiều cao, chiều rộng và số kênh màu của ảnh
    height, width, channels = image.shape

    # Tạo một mảng mới có cùng chiều cao và chiều rộng với ảnh ban đầu để lưu ảnh đen trắng
    # Ban đầu ảnh đen trắng sẽ là màu đen (tất cả giá trị là 0)
    image_gray = np.zeros((height, width), dtype=np.uint8)

    # Duyệt qua từng pixel của ảnh
    for i in range(height):
        for j in range(width):
            # Lấy giá trị của từng kênh màu R, G, B
            r = image[i, j, 0]
            g = image[i, j, 1]
            b = image[i, j, 2]

            # Tính giá trị độ sáng bằng cách trung bình cộng các giá trị R, G, B
            gray_value = int((r + g + b) / 3)

            # Gán giá trị độ sáng cho pixel tương ứng trên ảnh đen trắng
            image_gray[i, j] = gray_value

    # Hiển thị ảnh gốc và ảnh đen trắng
    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.imshow(image)
    plt.title("Ảnh Màu Gốc")
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(image_gray, cmap='gray')
    plt.title("Ảnh Đen Trắng")
    plt.axis('off')
```

- Kết quả:



c. Rotate Image

- Mô tả thuật toán:

1. Lấy Kích Thước Ảnh:
    - (h, w) = image.shape[:2]: Lấy chiều cao (h) và chiều rộng (w) của ảnh từ mảng ảnh đã đọc.
2. Xác Định Góc Xoay và Tâm Ảnh:
    - angle = 180: Xác định góc xoay là 180 độ.
    - center = (w // 2, h // 2): Tính toán tọa độ tâm của ảnh bằng cách lấy trung điểm của chiều rộng và chiều cao.
3. Tính Toán Ma Trận Xoay:
    - angle_rad = np.deg2rad(angle): Chuyển đổi góc xoay từ độ sang radian.
    - cos_a = np.cos(angle_rad) và sin_a = np.sin(angle_rad): Tính toán giá trị cosine và sine của góc xoay.
4. Khởi Tạo Ảnh Mới:
    - new_w và new_h: Tính toán kích thước mới của ảnh sau khi xoay để đảm bảo ảnh không bị cắt xén.
    - h: Chiều cao ban đầu của hình chữ nhật.
    - w: Chiều rộng ban đầu của hình chữ nhật.
    - sin_a: Giá trị của sin(góc xoay).
    - cos_a: Giá trị của cos(góc xoay).

Sau khi hình chữ nhật được xoay đi một góc a, kích thước mới của hình chữ nhật không còn giữ nguyên. Để đảm bảo rằng hình chữ nhật mới có thể chứa toàn bộ hình chữ nhật cũ sau khi xoay, bạn cần tính toán lại kích thước mới của nó (new_w và new_h).

- new_w: Chiều rộng mới của hình chữ nhật sau khi xoay. Nó được tính bằng tổng của:
    - abs(h * sin_a): Phần chiều cao ban đầu nhân với giá trị sin của góc xoay, đại diện cho phần chiều cao sau khi xoay chiếu xuống trục ngang.
    - abs(w * cos_a): Phần chiều rộng ban đầu nhân với giá trị cos của góc xoay, đại diện cho phần chiều rộng sau khi xoay chiếu xuống trục ngang.
- new_h: Chiều cao mới của hình chữ nhật sau khi xoay. Nó được tính bằng tổng của:

- abs(h * cos_a): Phần chiều cao ban đầu nhân với giá trị cos của góc xoay, đại diện cho phần chiều cao sau khi xoay chiếu xuống trục dọc.
- abs(w * sin_a): Phần chiều rộng ban đầu nhân với giá trị sin của góc xoay, đại diện cho phần chiều rộng sau khi xoay chiếu xuống trục dọc.

Việc dùng hàm abs() là để đảm bảo rằng các giá trị này luôn là số dương, vì chiều rộng và chiều cao không thể là số âm.

- rotated_image = np.zeros((new_h, new_w, 3), dtype=np.uint8): Tạo một ảnh mới (ban đầu là màu đen) với kích thước đủ lớn để chứa ảnh sau khi xoay.

5. Tính Tâm Mới Của Ảnh Sau Khi Xoay:
   - new_center = (new_w // 2, new_h // 2): Tính toán tâm của ảnh mới sau khi xoay, dùng để dịch chuyển tọa độ pixel.
6. Xoay Ảnh:
   - for y in range(h) và for x in range(w): Duyệt qua từng pixel của ảnh gốc.
   - new_x và new_y: Tính toán tọa độ của pixel mới sau khi xoay bằng cách áp dụng công thức ma trận xoay lên tọa độ pixel gốc.

Biến số:

- x và y: Tọa độ ban đầu của điểm cần xoay.
- center: Tọa độ của tâm xoay ban đầu.
- new_center: Tọa độ của tâm mới sau khi xoay.
- cos_a và sin_a: Lần lượt là giá trị của hàm cosin và sin của góc xoay a.

Công thức:

- new_x = int((x - center[0]) * cos_a + (y - center[1]) * sin_a + new_center[0])
  - x - center[0]: Tính khoảng cách từ điểm ban đầu đến tâm xoay theo trục X.
  - (x - center[0]) * cos_a + (y - center[1]) * sin_a: Tính tọa độ X mới sau khi xoay điểm quanh tâm xoay.
  - + new_center[0]: Cộng thêm vào tọa độ X của tâm mới để dịch chuyển điểm đến vị trí mới.
- new_y = int(-(x - center[0]) * sin_a + (y - center[1]) * cos_a + new_center[1])
  - y - center[1]: Tính khoảng cách từ điểm ban đầu đến tâm xoay theo trục Y.
  - -(x - center[0]) * sin_a + (y - center[1]) * cos_a: Tính tọa độ Y mới sau khi xoay điểm quanh tâm xoay.
  - + new_center[1]: Cộng thêm vào tọa độ Y của tâm mới để dịch chuyển điểm đến vị trí mới.
  - Kiểm tra nếu tọa độ mới nằm trong phạm vi của ảnh mới, sau đó gán giá trị pixel từ ảnh gốc sang vị trí tương ứng trên ảnh xoay.

7. Hiển Thị Ảnh Sau Khi Xoay:
   - plt.imshow(rotated_image): Hiển thị ảnh đã xoay.
   - plt.axis('off'): Tắt hiển thị trục tọa độ.

Kết quả cuối cùng là một hình ảnh xoay 180 độ, được tính toán và gán pixel bằng tay thay vì sử dụng các hàm thư viện có sẵn.

- Code:

```python
# Câu 3
image_path = 'C:\\Users\\Trung Tin\\Tài liệu\\IUH\\Xử lý ảnh\\Labs\\Pictures\\image.jpg'
image = plt.imread(image_path)
if image is None:
    print("Hình ảnh không đọc được. Vui lòng kiểm tra lại đường dẫn.")
else:
    (h, w) = image.shape[:2]

    # Xác định góc xoay và tính toán tâm của hình ảnh
    angle = 180
    center = (w // 2, h // 2)
    angle_rad = np.deg2rad(angle)
    cos_a = np.cos(angle_rad)
    sin_a = np.sin(angle_rad)

    # Khởi tạo ảnh mới với kích thước đủ lớn để chứa ảnh sau khi xoay
    new_w = int(abs(h * sin_a) + abs(w * cos_a))
    new_h = int(abs(h * cos_a) + abs(w * sin_a))
    rotated_image = np.zeros((new_h, new_w, 3), dtype=np.uint8)

    # Tâm mới của ảnh sau khi xoay
    new_center = (new_w // 2, new_h // 2)

    # Duyệt qua từng pixel và gán giá trị cho ảnh mới sau khi xoay
    for y in range(h):
        for x in range(w):
            new_x = int((x - center[0]) * cos_a + (y - center[1]) * sin_a + new_center[0])
            new_y = int(-(x - center[0]) * sin_a + (y - center[1]) * cos_a + new_center[1])

            if 0 <= new_x < new_w and 0 <= new_y < new_h:
                rotated_image[new_y, new_x] = image[y, x]

    plt.imshow(rotated_image)
    plt.axis('off')
```

- Kết quả:

d. Corner Line

- Mô tả thuật toán:

1. Kết Nối Google Drive:
    - drive.mount('/content/drive'): Kết nối Google Colab với Google Drive để truy cập và đọc các tệp từ Drive.
2. Đọc Ảnh Từ Google Drive:
    - image = plt.imread('/content/drive/My Drive/Colab Notebooks/Xu ly anh/image.jpg'): Đọc ảnh từ Google Drive bằng cách sử dụng đường dẫn đầy đủ tới tệp ảnh.
3. Tạo Bản Sao Của Ảnh:
    - image = np.copy(image): Tạo một bản sao của ảnh gốc để tránh thay đổi trực tiếp ảnh ban đầu.
4. Xác Định Độ Dày Của Đường Thẳng:
    - line_thickness = 15: Xác định độ dày của đường thẳng là 15 pixel.
5. Xác Định Điểm Bắt Đầu Vẽ Đường Thẳng:
    - start_point = image.shape[1] // 6: Xác định điểm bắt đầu của đường thẳng. Nó được tính bằng cách chia chiều rộng của ảnh cho 6, giúp đường thẳng bắt đầu từ một phần sáu chiều rộng của ảnh từ bên trái.
6. Vẽ Đường Thẳng:
    - Hai vòng lặp for i in range(line_thickness) và for j in range(min(image.shape[0], start_point)):
        - Vòng lặp đầu tiên for i in range(line_thickness) lặp qua độ dày của đường thẳng.
        - Vòng lặp thứ hai for j in range(min(image.shape[0], start_point)) lặp qua các pixel từ trên xuống dưới, giới hạn bởi chiều cao của ảnh và điểm bắt đầu.
    - if start_point - j - i >= 0: Điều kiện kiểm tra để đảm bảo rằng chỉ số tọa độ x (cột) không bị âm.
    - image[j, start_point - j - i] = [0, 0, 0]: Gán màu đen [0, 0, 0] cho các pixel dọc theo đường thẳng được vẽ, với độ dày xác định.
7. Hiển Thị Ảnh Sau Khi Vẽ Đường Thẳng:
    - plt.imshow(image): Hiển thị ảnh sau khi đã vẽ đường thẳng màu đen.
    - plt.axis('off'): Tắt hiển thị trục tọa độ.

Kết quả cuối cùng là một hình ảnh với một đường thẳng màu đen được vẽ từ phía trên bên trái, kéo dài theo một góc nghiêng với độ dày nhất định.

- Code:

```
# Câu 4
image_path = 'C:\\Users\\Trung Tin\\Tài liệu\\IUH\\Xử lý ảnh\\Labs\\Pictures\\image.jpg'
image = plt.imread(image_path)
if image is None:
    print("Hình ảnh không đọc được. Vui lòng kiểm tra lại đường dẫn.")
else:
# Tạo một bản sao của ảnh
    image = np.copy(image)
    # Xác định độ dày của đường thẳng
    line_thickness = 15
    # Xác định điểm bắt đầu
    start_point = image.shape[1] // 6

    for i in range(line_thickness):
        for j in range(min(image.shape[0], start_point)):
            if start_point - j - i >= 0:
                image[j, start_point - j - i] = [0, 0, 0]
    plt.imshow(image)
    plt.axis('off')
```

- Kết quả:

e. Gradient

- Mô tả thuật toán:

1. Đặt Kích Thước Hình Ảnh:
    - width, height = 255, 255: Xác định chiều rộng và chiều cao của hình ảnh, ở đây cả hai đều được đặt là 255 pixel.

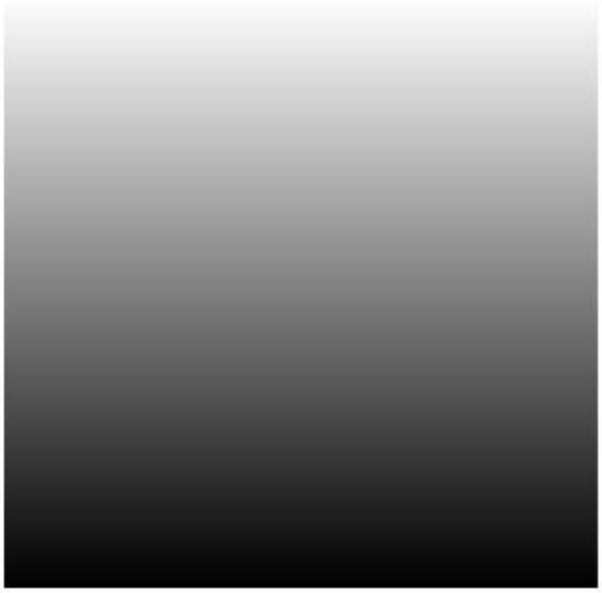
2. Tạo Ma Trận Hình Ảnh:
    - gradient_image = np.zeros((height, width), dtype=np.uint8): Tạo một ma trận hình ảnh có kích thước xác định, ban đầu tất cả các giá trị đều là 0 (màu đen). Mỗi phần tử trong ma trận là một giá trị độ xám 8-bit (từ 0 đến 255), đại diện cho mức độ sáng của một pixel.
3. Tạo Gradient Từ Trắng Đến Đen:
    - for y in range(height): Duyệt qua từng hàng của hình ảnh.
    - gray_value = int(255 * (1 - y / height)): Tính giá trị độ xám cho mỗi hàng. Giá trị này bắt đầu từ 255 (trắng) ở đầu hình ảnh và giảm dần đến 0 (đen) ở cuối hình ảnh. Công thức 255 * (1 - y / height) đảm bảo rằng giá trị này giảm tuyến tính từ 255 đến 0 khi y thay đổi từ 0 đến height.
    - gradient_image[y, :] = gray_value: Gán giá trị độ xám này cho toàn bộ hàng y trong hình ảnh.
4. Hiển Thị Hình Ảnh:
    - plt.imshow(gradient_image, cmap='gray'): Hiển thị hình ảnh đã tạo ra. cmap='gray' được sử dụng để đảm bảo rằng hình ảnh được hiển thị dưới dạng ảnh xám (gradient).
    - plt.axis('off'): Tắt hiển thị trục tọa độ.

Kết quả cuối cùng là một hình ảnh 255x255 pixel với gradient đơn sắc dọc từ trắng ở trên cùng đến đen ở dưới cùng. Mỗi hàng của hình ảnh có một màu xám đồng nhất, và màu sắc này tối dần từ trên xuống dưới.

- Code:

```python
# Câu 5
# Đặt kích thước hình ảnh
width, height = 256, 256

# Tạo một ma trận hình ảnh với kích thước đã định
gradient_image = np.zeros((height, width), dtype=np.uint8)

# Tạo gradient từ trắng đến đen
for y in range(height):
    # Tính giá trị xám từ 255 (trắng) đến 0 (đen)
    gray_value = int(255 * (1 - y / height))
    gradient_image[y, :] = gray_value

# Hiển thị hình ảnh
plt.imshow(gradient_image, cmap='gray')
plt.axis('off')
```

- Kết quả:



f. Letter

- Mô tả thuật toán:

1. Khởi Tạo Hình Ảnh Trắng:
    - height, width = 500, 1000: Xác định chiều cao và chiều rộng của hình ảnh.
    - image = np.ones((height, width), dtype=np.uint8) * 255: Tạo một hình ảnh có kích thước xác định, toàn bộ các pixel đều có giá trị 255 (trắng).
2. Hàm Vẽ Đường Thẳng:
    - draw_line(img, start_point, end_point, color, thickness): Hàm này vẽ một đường thẳng lên hình ảnh img từ điểm bắt đầu start_point đến điểm kết thúc end_point với màu color và độ dày thickness.
3. Hàm Vẽ Chữ "T":

    - draw_T(img): Hàm này vẽ chữ "T" lên hình ảnh img.
        - draw_line(img, (100, 100), (300, 100), color, thickness): Vẽ đường ngang trên cùng của chữ "T".
        - draw_line(img, (200, 100), (200, 400), color, thickness): Vẽ đường thẳng đứng của chữ "T".
4. Hàm Vẽ Chữ "I":
    - draw_I(img): Hàm này vẽ chữ "I" lên hình ảnh img.
        - draw_line(img, (400, 190), (400, 400), color, thickness): Vẽ đường thẳng đứng của chữ "I".
5. Hàm Vẽ Chữ "N":
    - draw_N(img): Hàm này vẽ chữ "N" lên hình ảnh img.
        - draw_line(img, (600, 100), (600, 400), color, thickness): Vẽ đường thẳng đứng đầu tiên của chữ "N".
        - draw_line(img, (600, 100), (800, 400), color, thickness): Vẽ đường chéo của chữ "N".
        - draw_line(img, (800, 100), (800, 400), color, thickness): Vẽ đường thẳng đứng thứ hai của chữ "N".
6. Hàm Vẽ Dấu Sắc:
    - draw_DauSac(img): Hàm này vẽ dấu sắc lên hình ảnh img.
        - draw_line(img, (400, 100), (540, 10), color, thickness): Vẽ đường chéo của dấu sắc từ điểm (400, 100) đến điểm (540, 10).
7. Vẽ Chữ "TÍN":
    - Các hàm draw_T(image), draw_I(image), draw_N(image), và draw_DauSac(image) được gọi tuần tự để vẽ các chữ và dấu sắc lên hình ảnh.
8. Hiển Thị Hình Ảnh:
    - plt.imshow(image, cmap='gray'): Hiển thị hình ảnh đã được vẽ với các chữ "TÍN" và dấu sắc.
    - plt.axis('off'): Tắt hiển thị trục tọa độ.

Kết quả cuối cùng là một hình ảnh có nền trắng với chữ "TÍN" được vẽ bằng các đường thẳng màu đen và độ dày xác định. Các ký tự và dấu sắc được tạo ra bằng cách sử dụng các đường thẳng cơ bản.

- Code:

```python
# Câu 6
# Kích thước hình ảnh
height, width = 500, 1000

# Tạo hình ảnh trắng
image = np.ones((height, width), dtype=np.uint8) * 255

def draw_line(img, start_point, end_point, color, thickness):
    cv2.line(img, start_point, end_point, color, thickness)

def draw_T(img):
    color = (0, 0, 0)
    thickness = 20
    # Đường ngang trên cùng
    draw_line(img, (100, 100), (300, 100), color, thickness)
    # Đường thẳng đứng
    draw_line(img, (200, 100), (200, 400), color, thickness)
    return img

def draw_I(img):
    color = (0, 0, 0)
    thickness = 20
    # Đường thẳng đứng
    draw_line(img, (400, 190), (400, 400), color, thickness)
    return img

def draw_N(img):
    color = (0, 0, 0)
    thickness = 20
    # Đường thẳng đứng 1
    draw_line(img, (600, 100), (600, 400), color, thickness)
    # Đường chéo
    draw_line(img, (600, 100), (800, 400), color, thickness)
    # Đường thẳng đứng 2
    draw_line(img, (800, 100), (800, 400), color, thickness)
    return img

def draw_DauSac(img):
    color = (0, 0, 0)
    thickness = 20
    # Đường chéo
    draw_line(img, (400, 100), (540, 10), color, thickness)
    return img
```

- Kết quả:

g. Color Separate

- Mô tả thuật toán:

1. Khởi tạo:
    - Đầu tiên, thuật toán kết nối với Google Drive để truy cập hình ảnh từ đó.
    - Sau khi đọc hình ảnh, thuật toán kiểm tra xem hình ảnh có được đọc thành công hay không.
    - Nếu hình ảnh được đọc thành công, các thông số về chiều cao (height), chiều rộng (width), và số kênh màu (channels) của hình ảnh được xác định.
2. Khởi tạo mảng kết quả:
    - Một mảng result_image mới được tạo với cùng kích thước và số kênh màu như hình ảnh ban đầu. Mảng này sẽ lưu kết quả sau khi áp dụng thuật toán tách vùng trắng.
3. Xác định ngưỡng màu trắng (threshold):
    - Một ngưỡng (threshold) được đặt là 160. Mục đích của ngưỡng này là để xác định liệu một pixel có được coi là màu trắng hay không. Nếu tất cả các giá trị kênh màu R, G, và B của một pixel đều lớn hơn ngưỡng này, thì pixel đó được coi là màu trắng.
4. Duyệt qua từng pixel của hình ảnh:
    - Thuật toán sử dụng hai vòng lặp lồng nhau để duyệt qua từng pixel trong hình ảnh.
    - Đối với mỗi pixel, giá trị của ba kênh màu R, G, và B được trích xuất.
5. Kiểm tra màu sắc và phân loại:
    - Thuật toán kiểm tra xem giá trị của ba kênh màu R, G, và B có lớn hơn ngưỡng hay không.
    - Nếu cả ba kênh màu đều lớn hơn ngưỡng:
        - Pixel được coi là thuộc vật thể màu trắng và giá trị của pixel đó được giữ nguyên trong result_image.
    - Nếu bất kỳ kênh màu nào không vượt qua ngưỡng:
        - Pixel đó không được coi là màu trắng và giá trị của nó trong result_image được gán là [0, 0, 0], tức là màu đen.
6. Hiển thị kết quả:
    - Cuối cùng, thuật toán hiển thị cả hình ảnh gốc và hình ảnh sau khi đã áp dụng thuật toán tách vùng trắng.
    - Hình ảnh kết quả (result_image) chỉ giữ lại các vùng màu trắng (hoặc gần trắng), còn các vùng khác được chuyển thành màu đen.

Ứng dụng: Thuật toán này hữu ích trong việc phát hiện và tách các vùng sáng hoặc màu trắng trong các hình ảnh, thường được áp dụng trong xử lý ảnh để phân loại hoặc trích xuất các đối tượng có độ sáng cao trên nền tối hoặc các màu sắc khác.

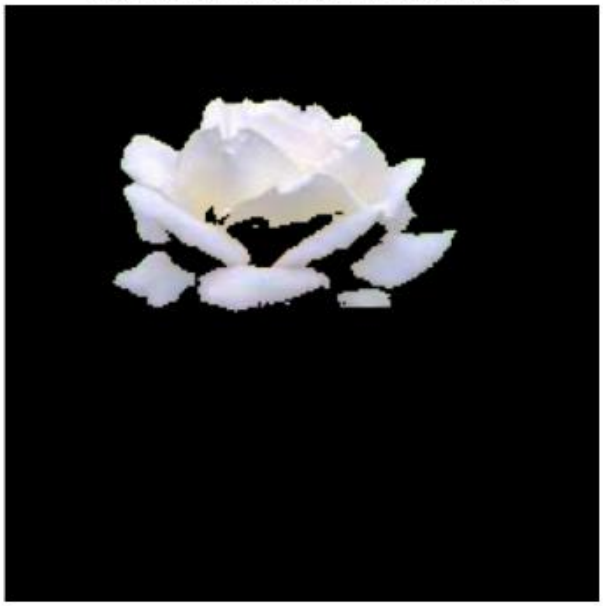
- Code:

```python
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive

drive.mount('/content/drive')
image = plt.imread('/content/drive/My Drive/Colab Notebooks/Xu ly anh/whiteRose.jfif')
if image is None:
    print("Hình ảnh không đọc được. Vui lòng kiểm tra lại đường dẫn.")
else:
    # Lấy chiều cao, chiều rộng và số kênh màu của ảnh
    height, width, channels = image.shape

    # Tạo một mảng mới có cùng chiều cao, chiều rộng và số kênh màu với ảnh ban đầu để lưu ảnh kết quả
    result_image = np.zeros((height, width, channels), dtype=np.uint8)

    # Đặt ngưỡng cho màu trắng
    threshold = 160

    # Duyệt qua từng pixel của ảnh
    for i in range(height):
        for j in range(width):
            # Lấy giá trị của từng kênh màu R, G, B
            r = image[i, j, 0]
            g = image[i, j, 1]
            b = image[i, j, 2]

            # Kiểm tra xem pixel đó có phải là màu trắng (R, G, B đều lớn hơn ngưỡng)
            if r > threshold and g > threshold and b > threshold:
                # Nếu pixel này được coi là thuộc vật thể màu trắng, giữ nguyên giá trị RGB của nó
                result_image[i, j] = image[i, j]
            else:
                # Ngược lại, gán giá trị là 0 (màu đen)
                result_image[i, j] = [0, 0, 0]
```

- Kết quả:

Ảnh Màu Gốc

Ảnh Sau Khi Tách Vật Thể Màu Trắng



h. Find secret by subtract

- Mô tả thuật toán:

1. Khởi tạo:
    - Đầu tiên, thuật toán kết nối với Google Drive để truy cập hình ảnh từ đó.
    - Hình ảnh được đọc từ đường dẫn cụ thể trong Google Drive.
    - Nếu hình ảnh được đọc thành công, các thông số về chiều cao (height), chiều rộng (width), và số kênh màu (channels) của hình ảnh được xác định.
2. Khởi tạo mảng kết quả:
    - Một mảng result_image mới được tạo với cùng kích thước và số kênh màu như hình ảnh ban đầu. Mảng này sẽ lưu kết quả sau khi áp dụng thuật toán tách vùng màu tối.
3. Đặt ngưỡng cho màu sắc (threshold):
    - Một ngưỡng (threshold) được đặt là 150. Mục đích của ngưỡng này là để xác định liệu một pixel có thuộc phần màu tối (thường là màu đen hoặc gần đen) của văn bản hay không.
4. Duyệt qua từng pixel của hình ảnh:
    - Thuật toán sử dụng hai vòng lặp lồng nhau để duyệt qua từng pixel trong hình ảnh.
    - Đối với mỗi pixel, giá trị của ba kênh màu R, G, và B được trích xuất.
5. Kiểm tra màu sắc và phân loại:
    - Nếu tất cả các giá trị màu R, G, và B của pixel nhỏ hơn ngưỡng:
        - Pixel đó được coi là thuộc phần màu tối của văn bản, và giá trị của nó trong result_image được gán là [0, 0, 0] (màu đen).
    - Nếu bất kỳ kênh màu nào vượt qua ngưỡng:
        - Pixel đó được coi là không thuộc phần màu tối, và giá trị của nó trong result_image được gán là [255, 255, 255] (màu trắng).
6. Hiển thị kết quả:
    - Thuật toán hiển thị cả hình ảnh gốc và hình ảnh sau khi đã áp dụng thuật toán tách màu.
    - Ảnh kết quả (result_image) chỉ giữ lại các vùng màu tối của văn bản, còn các vùng khác được chuyển thành màu trắng.

Ứng dụng:

- Thuật toán này hữu ích trong việc làm nổi bật các phần màu tối của văn bản hoặc các đối tượng tối trong hình ảnh trên nền sáng, giúp dễ dàng phát hiện và đọc các thông điệp ẩn hoặc các chi tiết quan trọng.
- Nó có thể được sử dụng trong các ứng dụng như nhận dạng ký tự quang học (OCR), phân tích văn bản trên ảnh, hoặc bất kỳ trường hợp nào cần làm rõ các chi tiết tối trên nền sáng.

- Code:

```python
# Câu 7
image_path = 'C:\\Users\\Trung Tin\\Tài liệu\\IUH\\Xử lý ảnh\\Labs\\Pictures\\whiteRose.jpg'
image = plt.imread(image_path)
if image is None:
    print("Hình ảnh không đọc được. Vui lòng kiểm tra lại đường dẫn.")
else:
    # Lấy chiều cao, chiều rộng và số kênh màu của ảnh
    height, width, channels = image.shape

    # Tạo một mảng mới có cùng chiều cao, chiều rộng và số kênh màu với ảnh ban đầu để lưu ảnh kết quả
    result_image = np.zeros((height, width, channels), dtype=np.uint8)

    # Đặt ngưỡng cho màu trắng
    threshold = 160

    # Duyệt qua từng pixel của ảnh
    for i in range(height):
        for j in range(width):
            # Lấy giá trị của từng kênh màu R, G, B
            r = image[i, j, 0]
            g = image[i, j, 1]
            b = image[i, j, 2]

            # Kiểm tra xem pixel đó có phải là màu trắng (R, G, B đều lớn hơn ngưỡng)
            if r > threshold and g > threshold and b > threshold:
                # Nếu pixel này được coi là thuộc vật thể màu trắng, giữ nguyên giá trị RGB của nó
                result_image[i, j] = image[i, j]
            else:
                # Ngược lại, gán giá trị là 0 (màu đen)
                result_image[i, j] = [0, 0, 0]

    # Hiển thị ảnh gốc và ảnh kết quả
    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.imshow(image)
    plt.title("Ảnh Màu Gốc")
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(result_image)
    plt.title("Ảnh Sau Khi Tách Vật Thể Màu Trắng")
    plt.axis('off')
```

- Kết quả:

Ảnh Gốc

Ảnh Sau Khi Tách