THỰC HÀNH ĐẠI SỐ TUYẾN TÍNH

TÀI LIỆU PHỤC VỤ SINH VIÊN NGÀNH KHOA HỌC DỮ LIỆU

Nhóm Giảng viên biên soạn: TS. Hoàng Lê Minh – Khưu Minh Cảnh – Hoàng Thị Kiều Anh – Lê Công Hiếu – Trần Ngọc Việt – Nguyễn Công Nhựt – Đỗ Đình Thủ - Nguyễn Thị Thanh Bình – Trần Kim Mỹ Vân – Phạm Trọng Nghĩa – Nguyễn Hữu Trí Nhật – Lê Thị Ngọc Huyên – Nguyễn Thị Tuyết Mai – Nguyễn Viết Cường – Huỳnh Thái Học và các Giảng viên khác.

TP.HCM - Năm 2020

MỤC LỤC

CHƯƠNG 7: KHÔNG GIAN VECTOR VÀ ÁNH XẠ TUYẾN TÍNH (PHẦN 1)

Mục tiêu:

- *Một số ứng dụng không gian vector và ánh xạ tuyến tính*
- *Thử nghiệm sử dụng các gói xử lý ảnh/đồ thị: PIL, skimage, mathplotlib,…*
- *Ứng dụng trị riêng, vector riêng: Giới thiệu thuật toán tìm hạng của trang web.*

Nội dung chính:

Trong bài này, sinh viên sẽ được giới thiệu một số khái niệm nền tảng của không gian vector và ứng dụng của chúng.

1. Giới thiệu một số ứng dụng của tích vector (dot product)

Nhắc lại, tích vector của một ma trận và một vector là một ma trận có số dòng bằng số dòng vector và số cột là số dòng ma trận. Ví dụ: xét tích ma trận – vector sau:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 10 & 0 \end{bmatrix}, V = [3, -1]$$

Tích vector $A * V = \big[[1,2] * [3,-1], [3,4] * [3,-1], [10,0] * [3,-1]\big] = [1,5,30]$

Lệnh thể hiện trong Python:

>>> import numpy as np

>>> signals = **np.array**([[1,2],[3,4],[10,0]])

>>> sample = **np.array**([3,-1])

>>> np.**inner**(signals, sample)

…… # Sinh viên điền kết quả

1.1.Ứng dụng 1 – Nguyên lý tìm nốt nhạc trong chuỗi âm thanh (Audio search)

Âm thanh được lưu dưới dạng số và thường là dữ liệu bảng số. Với một chuỗi âm thanh đưa vào, người ta cũng lưu trữ dưới dạng chuỗi số.

Ví dụ 1: Cần tìm chuỗi tín hiệu âm thanh [0, 1, -1] trong chuỗi âm **[0, 0, -1, 2, 3, -1, 0, 1, -1, -1]**:

Giải: Chúng ta thực hiện phép tích **dot product** giữa ma trận tín hiệu và vector như sau:

$$\begin{bmatrix} 0 & 0 & -1 \\ 0 & -1 & 2 \\ -1 & 2 & 3 \\ 2 & 3 & -1 \\ 3 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix} * [0, 1, -1]^T$$

Sinh viên thực hiện các lệnh Python như sau:

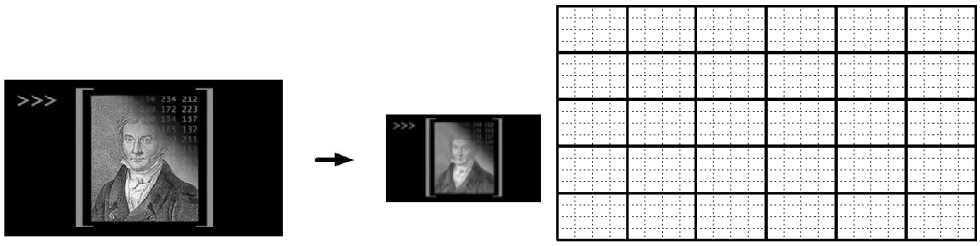
```
>>> import numpy as np
>>> A = np.array([0,0,-1,2,3,-1,0,1,-1,-1])
>>> search_vector = np.array([0,1,-1])
>>> len(A),  len(search_vector)  # nghĩa là = (10, 3)
…… # Sinh viên điền  kết quả
>>> B = np.array([1])
>>> B = np.resize(B, (len(A)-len(search_vector)+1, len(search_vector)))
>>> B = np.asmatrix(B)
>>> for i in range(len(A)-len(search_vector)+1):  # so dong
        for j in range(len(search_vector)):  # so cot
                B[i,j] = A[i+j]
…… # Sinh viên điền  kết quả
>>> C = np.inner(B, search_vector)
>>> for i in range(len(A)-len(search_vector)+1):          # tìm vị trí của vector vừa tìm thấy
        if ( C[0,i] == np.inner(search_vector, search_vector) ):
                print (i, B[i])
…… # Sinh viên điền  kết quả
```

1.2.Ứng dụng 2 – Tạo ảnh mẫu và làm mờ ảnh

Giới thiệu thư viện PIL và phương pháp làm mờ ảnh:

Nguyên lý:

- [Downsampling] Kỹ thuật tạo ảnh mẫu:

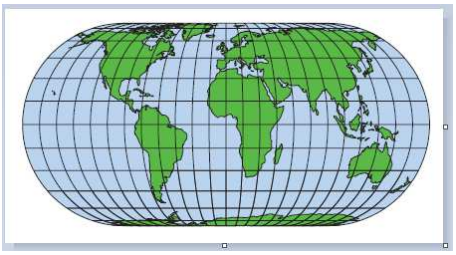


- Mỗi pixel có giá trị low-res của hình ảnh nằm trong một lưới của ảnh có high-res.
- Giá trị cường độ của các pixel low-res là ***trung bình cường độ*** của các giá trị high-res pixel tương ứng.
- ***Giá trị trung bình có thể tính bằng dot-product (tích trong).***
- Tính toán tích trong của các pixel low-res. Và thay thế 1 ô lưới bằng giá trị 1 pixel.

Ví dụ: nếu chúng ta giảm ảnh ¼ nghĩa là chúng ta sẽ thay thế 1 giá trị trung bình của lưới 2x2 bằng 1 giá trị.

Với gói xử lý **PIL** (Python Image Library), thuật toán thay đổi kích thước được cài đặt sẵn. Người sử dụng chỉ cần thông số kích thước ảnh mới là việc tính toán khác sẽ do gói phần mềm. Sinh viên có thể làm quen với các câu lệnh như sau:

Giả định trong desktop có một tập tin ảnh là **'C:/traidat.PNG'**. Sinh viên có thể thay đổi ảnh khác có trên trong máy. Lưu ý: nên chọn tập tin có kích thước nhỏ.



Khi đó, chúng ta thực hiện các lệnh sau:

>>> from **PIL** import **Image**

>>> img = Image.**open**('C:/traidat.PNG')

>>> img.**height** # xem chiều cao của ảnh

>>> img.**width** # chiều rộng của ảnh

>>> img.**mode** # xem kiểu ảnh. Thường là 'RGBA', với kiểu ảnh có chữ 'P' chúng ta phải thêm một lệnh xử lý như sau:

>>> img = img.**convert**("RGB") # convert it to RGB (để chuyển về dạng RGB)

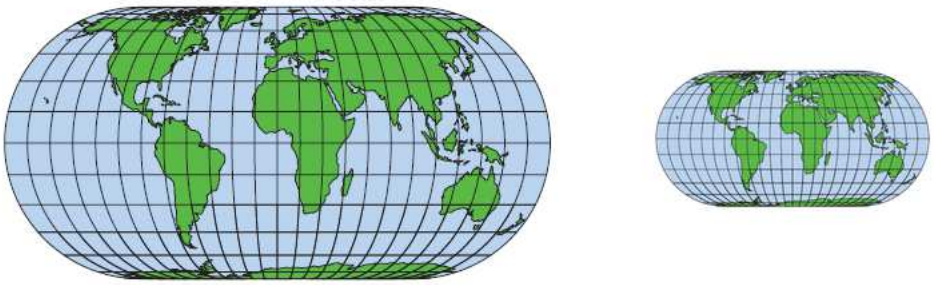
>>> new_width = int(img.width / 2) # giảm ½ chiều rộng

>>> new_height = int(img.height / 2) # giảm ½ chiều cao

>>> new_img = **img.resize((new_width, new_height),Image.ANTIALIAS)**

>>> new_img.**save**('C:/traidat_small.PNG')

Sau đó, sinh viên xem tập tin vừa tạo thành.



- [**Image blur**] Kỹ thuật làm mờ ảnh:

- Để làm mờ ảnh, thay thế các pixel có cường độ cao bằng các giá trị trung bình.
- ***Giá trị trung bình có thể tính bằng cách tính tích trong.***
- Dạng tính này là tích matrix-vector. Ví dụ: một ma trận làm mờ được cho như sau:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Nghĩa là giá trị trung tâm sẽ bằng tổng trung bình 8 giá trị xung quanh và chính nó.

Sinh viên thực hiện các lệnh sau:

>>> from PIL import Image, ImageDraw

>>> input_image = **Image.open**('C:/traidat.PNG')

>>> input_pixels = input_image.**load**() # đọc các pixel(điểm ảnh). GV giải thích khái niệm pixel

>>> box_kernel = [[1 / 9, 1 / 9, 1 / 9],

[1 / 9, 1 / 9, 1 / 9],

[1 / 9, 1 / 9, 1 / 9]]

Dưới đây là xác định vị trí bắt đầu và vị trí kết thúc. Lưu ý: pixel ban đầu của ảnh không thể làm mờ theo kỹ thuật này. Vì ít nhất phải là pixel ở tọa độ bên trong khuôn:

>>> kernel = box_kernel

>>> offset = len(kernel) // 2

Tiếp theo, chúng ta tạo sẵn ảnh (khuôn):

>>> output_image = **Image.new**("RGB", input_image.size)

>>> draw = **ImageDraw.Draw**(output_image)

Thực thi dòng lệnh thay thế giá trị điểm ảnh (pixel) bằng giá trị mờ. Lưu ý đoạn code trên: do mỗi điểm ảnh được tạo thành từ 3 thành tố ảnh (kênh màu): R (red - đỏ), G (green – lục), B (blue – lam) nên mỗi yếu tố cũng cần phải được lấy giá trị trung bình.

Cụ thể, mỗi điểm ảnh được xác định bằng 3 thông số. Trong đoạn mã bên dưới, các giá trị của từng kênh màu được lưu trong mảng acc. Tuần tự:

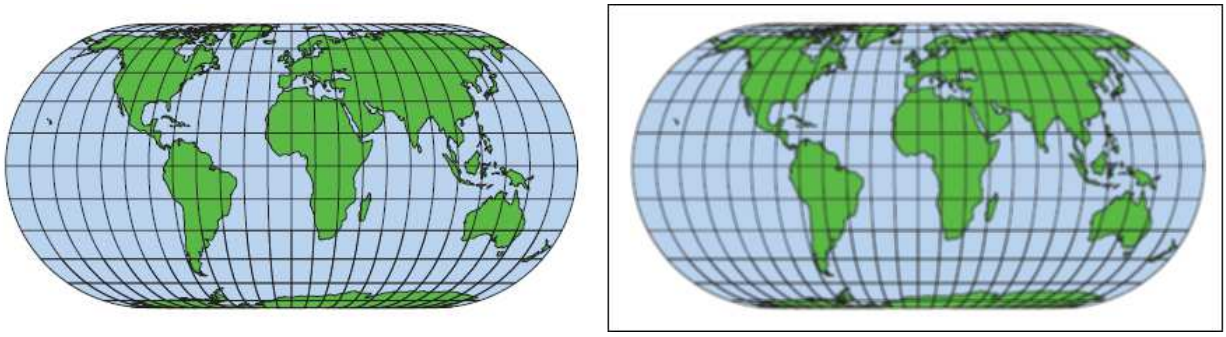
- acc[0]: màu đỏ.
- acc[1]: màu lục.
- acc[2]: màu lam.

```
>>> for x in range(offset, input_image.width - offset):
        for y in range(offset, input_image.height - offset):
                acc = [0, 0, 0]
                for a in range(len(kernel)):
                        for b in range(len(kernel)):
                                xn = x + a - offset
                                yn = y + b - offset
                                pixel = input_pixels[xn, yn]
                                acc[0] += pixel[0] * kernel[a][b]
                                acc[1] += pixel[1] * kernel[a][b]
                                acc[2] += pixel[2] * kernel[a][b]
                draw.point((x, y), (int(acc[0]), int(acc[1]), int(acc[2])))
```

Và cuối cùng là lưu tập tin ảnh đã xử lý:

>>> output_image.save('C:/traidat_lammo.PNG'**)**

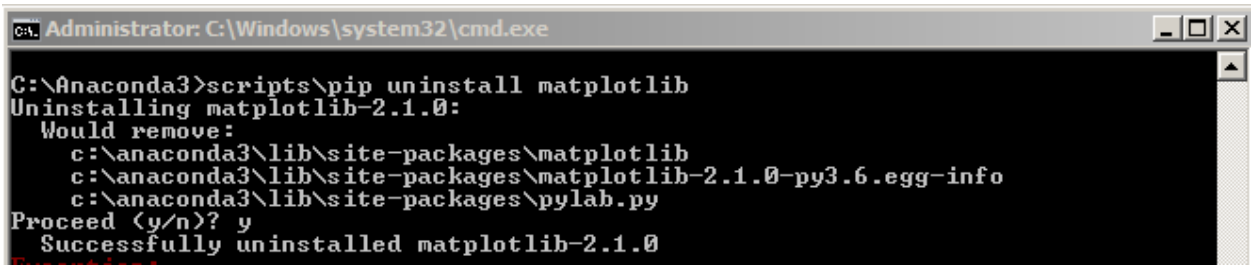
Sau đó, xem lại kết quả (ảnh trái trước khi làm mờ và ảnh phải đã làm mờ):



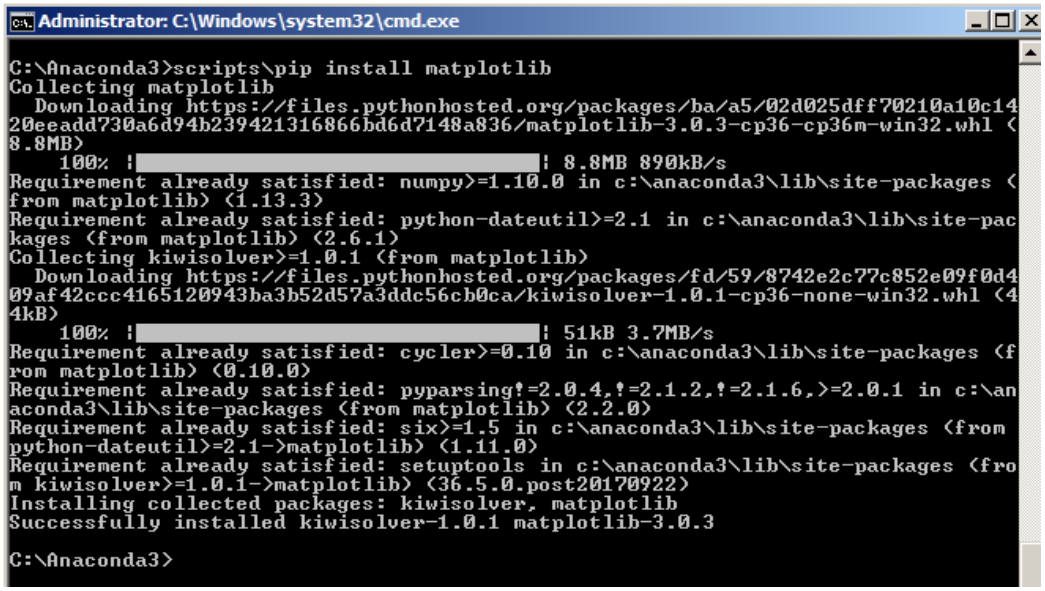
Bài tập 3: Bên trên, sinh viên sử dụng hàm **resize** để định lại kích thước ảnh (thư viện này của PIL). Với bài thực tập về làm mờ ảnh, sinh viên được tiếp cận phương pháp đọc và xử lý các pixel trên ảnh thành các array. Từ đó, sinh viên hãy tự viết lại hàm **resize**.

Lưu ý: Bên cạnh gói xử lý ảnh **PIL**, thư viện **matplotlib** còn hỗ trợ chúng ta các hàm xử lý ảnh. Tuy nhiên, trong một số phiên bản Anaconda được cài đặt, chúng ta phải gỡ và cài đặt lại gói phần mềm matplotlib. Sinh viên có thể thử nghiệm tại nhà.

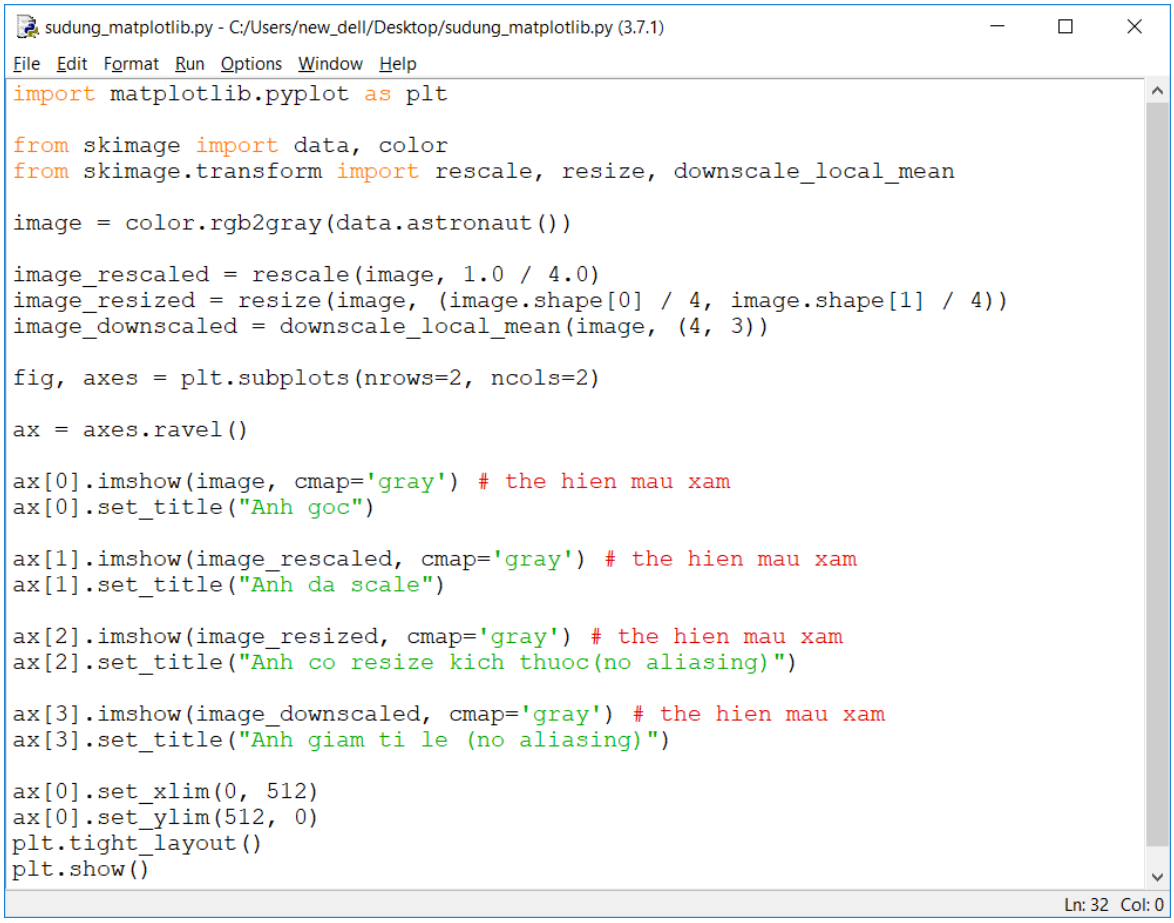
- Gỡ cài đặt:



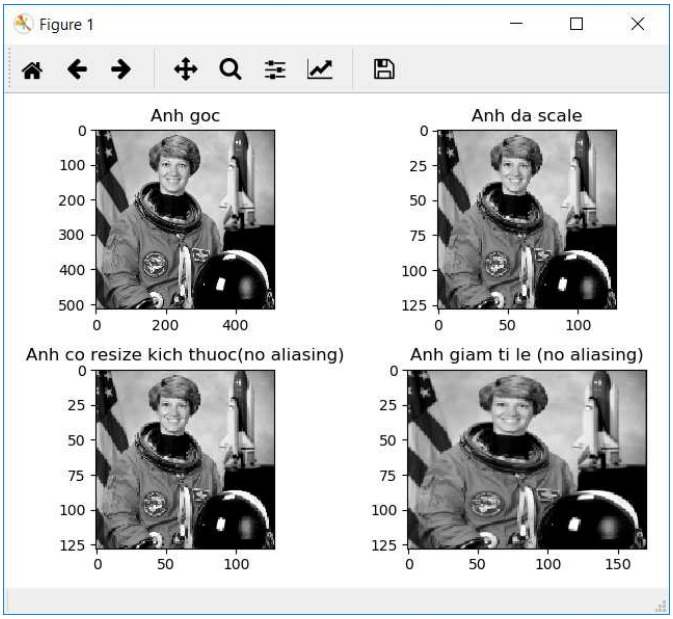
- Cài đặt lại gói: **pip install matplotlib** nếu máy chưa cài đặt gói này:



Dưới đây là một đoạn mã điển hình sử dụng matplotlib trong xử lý ảnh (sinh viên dễ dàng tìm thấy trên mạng để thử nghiệm):



Và kết quả:

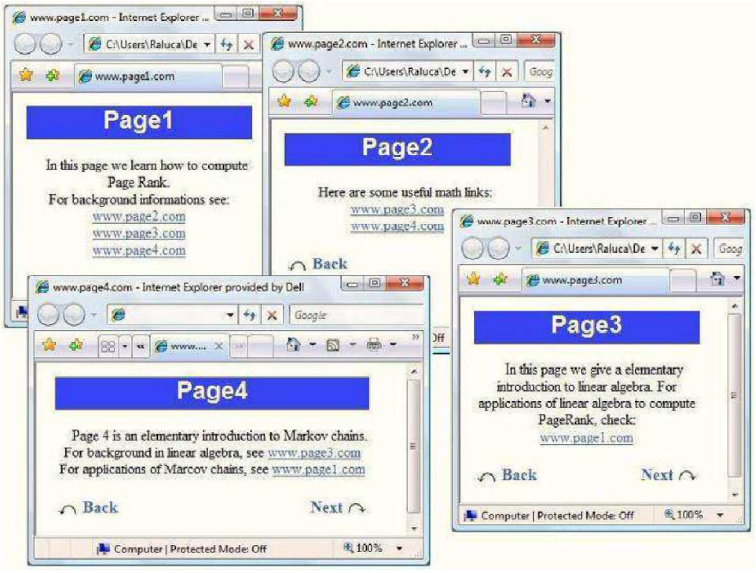


2. Ứng dụng: Xếp hạng các trang web với trị riêng và vector riêng

Phần này giới thiệu một ứng dụng của trị riêng và vector riêng. Đó là thuật toán xác định "hạng" của trang web được Page và Brin đề xuất năm 1998. Theo đó, thuật toán Pagerank do Page và Brin đề xuất năm 1998 và được sử dụng trong lõi của hệ thống tìm kiếm của Google. Mục tiêu của thuật toán là ước lượng mức độ phổ biến/quan trọng của một trang web dựa trên tính liên kết nối trên web. Một số giả định được sử dụng là: (i) trang có nhiều liên kết đến sẽ quan trọng hơn trang có ít liên kết đến; (ii) một trang có liên kết từ một trang nổi tiếng/quan trọng thì sẽ quan trọng.

Trong bài viết này, chúng ta xem xét đến thuật toán từ góc nhìn ma trận. Trên thực tế, số lượng web rất lớn, do vậy, các minh họa ở đây chỉ lấy số web là rất nhỏ.

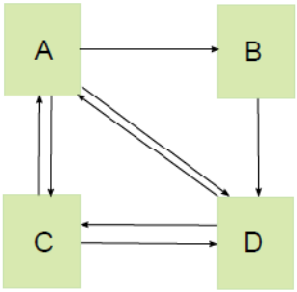
Về bản chất đại số, đây là một ứng dụng của giá trị riêng, vector riêng đối với một ma trận đặc biệt. Ma trận này hình thành từ đồ thị kết nối và thiết lập theo định hướng ma trận cột ngẫu nhiên (ma trận có các phần tử dương và tổng cột bằng 1). Vì với định lý Perron-Frobenius, chúng ta luôn tìm được giá trị riêng và trong đó 1 giá trị riêng là 1 cùng với các giá trị riêng có trị tuyệt đối nhỏ hơn 1.



2.1.Đơn giản hóa thuật toán Pagerank

Sự kết nối giữa các trang thể hiện bằng một đồ thị bao gồm: nốt là trang web và cạnh (mũi tên) là sự kết nối giữa hai trang giả định A và B và cụ thể là liên kết từ trang A đến trang B. Số lượng các liên kết "đi ra" (out-going) là tham số quan trọng của một trang web. Chúng ta sẽ sử dụng khái niệm "out-degree of a node" để chỉ số lượng các liên kết đi ra từ một trang web. Đồ thị này gọi là đồ thị web graph. Nhắc lại: mỗi "nốt" (node) trong đồ thị là một "trang" (page) và hai khái niệm được sử dụng như một trong phần trình bày bên dưới.

Xét ví dụ 1 dưới đây: Giả định chúng ta có 4 trang web là A, B, C và D và các kết nối như hình:



Theo hình trên, giả sử chúng ta có 4 trang: trang A chứa liên kết trang B, trang C và cả trang D. Trang B chỉ chứa liên kết đến trang D… Khi đó, chúng ta sẽ có số lượng các liên kết như sau: L(A) = 3, L(B) = 1 (do từ B chỉ có ra D) và L(C) = L(D) = 2 (từ C ra A và D; D ra A và C).

Gọi N là số lượng các trang cần xét, trường hợp này N=4. Chúng ta có thể dựng một ma trận NxN với các vị trí (i, j) được định nghĩa như sau:

$$a_{ij} = \begin{cases} \frac{1}{L(j)} nếu\ có\ liên\ kết\ từ\ j\ đến\ i \\ 0, trường\ hợp\ còn\ lại \end{cases}$$

Khi đó, ma trận $\mathbb{A}$ sẽ là:

$$\mathbb{A} = \begin{bmatrix} 0 & 0 & 1/2 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1 & 1/2 & 0 \end{bmatrix}$$

```
>>> import numpy as np
>>> A = np.array([[0,0,1.0/2, 1.0/2],
                  [1.0/3,0,0,0],
                  [1.0/3,0,0,1.0/2],
                  [1.0/3,1.0,1.0/2,0]])
>>> x = np.array([1,1,1,1])
```

Lưu ý: ma trận trên có tính chất là các phần tử đều không âm và tổng mỗi cột bằng 1 vì đó là tỉ lệ % kết nối. Tổng quát hơn, các ma trận có tính chất các phần tử không âm và tổng mỗi cột bằng 1 thì chúng ta gọi đó là các cột ngẫu nhiên (column-stochastic). Tóm lại, ma trận $\mathbb{A}$ là ma trận cột ngẫu nhiên với giả định mỗi trang đều có 1 trang kết nối.

Từ đó, chúng ta phát biểu thuật toán Pagerank đơn giản như sau:

Khởi tạo x là một vector cột $N \times 1$ với các phần tử không âm và thực hiện tính x bằng tích (phép nhân) $\mathbb{A}x$ đến khi tích hội tụ, nghĩa là lặp liên tục:

$$\boldsymbol{x_{mới}} = \mathbb{A}\boldsymbol{x}$$

Khi x hội tụ, chúng ta gọi vector x là vector pagerank của ma trận $\mathbb{A}$.

Thực hành: Sinh viên hãy thực hiện tích của ma trận $\mathbb{A}$ với vector x như sau:

$$x = \begin{bmatrix} x_A \\ x_B \\ x_C \\ x_D \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

```
>>> x = np.dot(A, x)
>>> print (x)
```

……

Hoặc:

```
>>> x = np.array([1.0, 1.0, 1.0, 1.0])
>>> for i in range(10):
        x = np.dot(A, x)
        print (i+1, x)
```

……

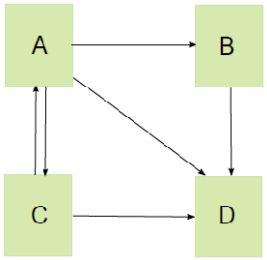
Bảng mô tả các lần lặp (sinh viên tự điền vào các giá trị tại các vị trí …):

Lần lặp	x_A	x_B	x_C	x_D
0	1	1	1	1
1	1	0.3333	0.8333	1.8333
2	…	…	…	…
3	…	…	…	…
4	…	…	…	…
5	…	…	…	…
6	…	…	…	…
7	…	…	…	…
8	…	…	…	…
9	…	…	…	…
10	1.20005466	0.39991641	1.06706131	1.33296762

Chúng ta có thể quan sát được rằng, sau 10 lần lặp, trang … có giá trị cao nhất, chúng ta gọi đó là hạng của một trang web (pagerank). Và đến ngày nay phương pháp này vẫn được sử dụng để xếp hạng các trang web.

2.2. [Đọc thêm] Xử lý dangling node trong thuật toán Pagerank

Một node gọi là **dangling** node nếu nó không tồn tại liên kết ra ngoài, nghĩa là bậc kết nối bên ngoài bằng 0. Ví dụ node D trong hình bên dưới:



Theo cách xây dựng trên, ma trận $\mathbb{A}$ tương ứng là:

$$\mathbb{A} = \begin{bmatrix} 0 & 0 & 1/2 & \mathbf{0} \\ 1/3 & 0 & 0 & \mathbf{0} \\ 1/3 & 0 & 0 & \mathbf{0} \\ 1/3 & 1 & 1/2 & \mathbf{0} \end{bmatrix}$$

Ma trận $\mathbb{A}$ trên không phải là ma trận ngẫu nhiên do có 1 dòng có tổng khác 1. Do đó, khi thực hiện đoạn mã trên, chúng ta sẽ có kết quả sau:

```
>>> A = np.array([[0,0,1.0/2, 0.0],
                  [1.0/3,0,0,0],
                  [1.0/3,0,0,0.0],
                  [1.0/3,1.0,1.0/2,0]])
>>> for i in range(5):
        x = np.dot(A, x)
        print (i+1, x)
```

…….

Sinh viên điền vào bảng sau:

Lần lặp	x_A	x_B	x_C	x_D
0	1	1	1	1
1	0.5	0.333333	0.83333	1.83333
2	…	…	…	…
3	…	…	…	…
4	…	…	…	…
5	0.0139	0.0093	0.0098	0.0509

Như vậy, sau các lần lặp, vector bị phân kì và tiến đến 0. Để khắc phục vấn đề này, một hiệu chỉnh nhỏ trên thuật toán được áp dụng, đó là thay thế cột toàn giá trị 0 ở node D bằng cột mang giá trị $1/N$. Việc lựa chọn sẽ thỏa điều kiện ma trận trở thành ma trận ngẫu nhiên. Về mặt ý nghĩa, chúng ta có thể hiểu là trang web D có thể liên kết đến các trang web khác theo xác suất là như nhau và bằng 1/N(=1/4 trong trường hợp này, do xét đồ thị 4 node).

$$\begin{bmatrix} 1/N & 1/N & 1/N & 1/N \end{bmatrix}^T = \begin{bmatrix} 1/4 & 1/4 & 1/4 & 1/4 \end{bmatrix}^T$$

Nghĩa là ma trận $\mathbb{A}$ sẽ được định nghĩa và thay bằng ma trận hiệu chỉnh $\overline{\mathbb{A}}$ (với N là số lượng trang web) là:

$$\bar{a}_{ij} = \begin{cases} \dfrac{1}{L(j)}, nếu\ có\ liên\ kết\ từ\ j\ sang\ i \\ \dfrac{1}{N}, nếu\ node\ j\ là\ dạng\ dangling\ node \\ 0, trường\ hợp\ còn\ lại \end{cases}$$

Khi đó, ma trận $\overline{\mathbb{A}}$ sẽ là:

$$\overline{\mathbb{A}} = \begin{bmatrix} 0 & 0 & 1/2 & \mathbf{1/4} \\ 1/3 & 0 & 0 & \mathbf{1/4} \\ 1/3 & 0 & 0 & \mathbf{1/4} \\ 1/3 & 1 & 1/2 & \mathbf{1/4} \end{bmatrix}$$

```
>>> x = np.array([1.0, 1.0, 1.0, 1.0])
>>> A = np.array([[0,0,1.0/2, 1/4.0],
                  [1.0/3,0,0,1/4.0],
                  [1.0/3,0,0,1/4.0],
                  [1.0/3,1.0,1.0/2,1/4.0]])
>>> for i in range(7):
        x = np.dot(A, x)
        print (i+1, x)
```

…… # sinh viên điền kết quả vào bảng bên dưới:

Lần lặp	x_A	x_B	x_C	x_D
0	1	1	1	1
1	0.75	0.5833	0.5833	2.0833
2	…	…	…	…
3	…	…	…	…
4	…	…	…	…
5				
6				
7				

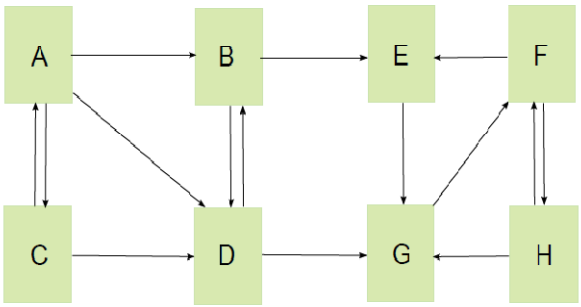
>>> np.**max**(x)

Sau khi tính toán, chúng ta cũng được kết quả node D là node có giá trị lớn nhất……. điều này có nghĩa là node D được xếp hạng cao nhất, hạng 1.

2.3. [Đọc thêm] Xử lý nhánh web reducible

Còn gọi là các thành phần không kết nối hoặc thành phần không liên thông (disconnected components).

Xét đồ thị web gồm 8 node như hình bên dưới:



$$\begin{bmatrix} 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/3 & 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1/2 \\ 0 & 0 & 0 & 1/2 & 1 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 \end{bmatrix}$$

Với đồ thị web trên, chúng ta hoàn toàn không có các dangling node. Tuy vậy, với nhánh các trang E, F, G và H thì lại không có kết nối với các nhánh bên ngoài. Ma trận $\mathbb{A}$ của đồ thị web được xây dựng như hình trên (bên phải).

Với ma trận trên, chúng ta có thể tính toán các giá trị vector x của từng trang web:

```
>>> A = np.array([[0.0,   0.0, 1/2.0,   0.0, 0.0, 0.0, 0.0, 0.0],
                  [1.0/3, 0.0, 1/2.0,   0.0, 0.0, 0.0, 0.0, 0.0],
                  [1/3.0,  0.0,   0.0,  0.0, 0.0, 0.0, 0.0, 0.0],
                  [1.0/3,1/2.0,1/2.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                  [0.0  ,1/2.0,  0.0, 0.0, 0.0,1/2.0, 0.0,0.0],
                  [0.0, 0.0, 0.0,0.0, 0.0, 0.0, 1.0,1/2.0],
                  [0.0, 0.0, 0.0,1/2.0, 1.0, 0.0, 0.0, 1/2.0],
                  [0.0, 0.0, 0.0,0.0, 0.0,1/2.0, 0.0, 0.0]])
>>> N = 8
>>> x = np.array([1.0/N, 1.0/N, 1.0/N, 1.0/N, 1.0/N, 1.0/N, 1.0/N, 1.0/N])
```

>>> for i in range(7):

x = np.dot(A, x)

print (i+1, x)

Lần lặp	x_A	x_B	x_C	x_D	x_E	x_F	x_G	x_H
0	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
1								
2								
…								
7	0.001?	0.0053?	0.0002?	0.0063?	0.1739?	0.3549?	0.2912?	0.1681?

Để xử lý, thuật toán bổ sung thêm như sau:

$$\mathcal{M} = d\bar{\mathbb{A}} + \frac{1-d}{N}\begin{bmatrix} 1 & 1 \dots & 1 \\ \dots & 1 \dots & \dots \\ 1 & 1 \dots & 1 \end{bmatrix}$$

Trong đó, d là một hệ số (hằng) "trộn". Google sử dụng giá trị $d = 0.85$.

Sinh viên tính toán ma trận $\mathcal{M}$ bằng Python như đoạn mã dưới đây:

>>> d = 0.85

>>> N = 8

>>> x = np.array([1.0/N, 1.0/N, 1.0/N, 1.0/N, 1.0/N, 1.0/N, 1.0/N, 1.0/N])

>>> A = np.array(……..)

>>> M = d*A + ((1-d)/N)* **np.ones([N,N])**

……

>>> for i in range(7):

x = np.dot(A, x)

print (i+1, x)

Lần lặp	x_A	x_B	x_C	x_D	x_E	x_F	x_G	x_H
0	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
1								
2								
3								
4								

5								
6								
7								

Trên đây là mô tả sơ lược về phương pháp để tính toán hạng của các trang web và các vấn dề liên quan như: đánh giá mức độ thân thiết trong quan hệ bạn bè trên mạng xã hội. Sinh viên có thể tự tìm hiểu thêm về kiến thức toán đại số và thống kê có liên quan đến nội dung trên theo các địa chỉ tham khảo:

- http://pi.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture3/lecture3.html
- https://en.wikipedia.org/wiki/PageRank

BÀI TẬP CHƯƠNG 7

Câu 1: Thực hiện cài đặt cấu hình làm mờ bằng Gauss (thay đổi ma trận **box_kernel**).

Lưu ý: lọc (làm mờ) Gaussian trong các phần mềm xử lý ảnh là một kỹ thuật dạng này. Ma trận ("nhân") làm mờ ảnh theo Gauss là:

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Câu 2: Thực hiện cài đặt phương pháp "rút xương" (tô đậm cạnh) cho các hình ảnh theo 2 "nhân" (ma trận) như sau:

a. $\begin{pmatrix} 0 & -1 & 0 \\ -1 & \mathbf{4} & -1 \\ 0 & -1 & 0 \end{pmatrix}$

b. $\begin{pmatrix} -1 & -1 & -1 \\ -1 & \mathbf{8} & -1 \\ -1 & -1 & -1 \end{pmatrix}$

Lưu ý: các "nhân" đều là những hàm có tổng tuyến tính bằng 0 hoặc 1.