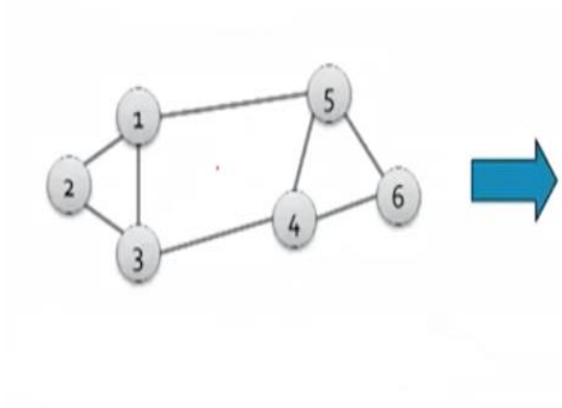
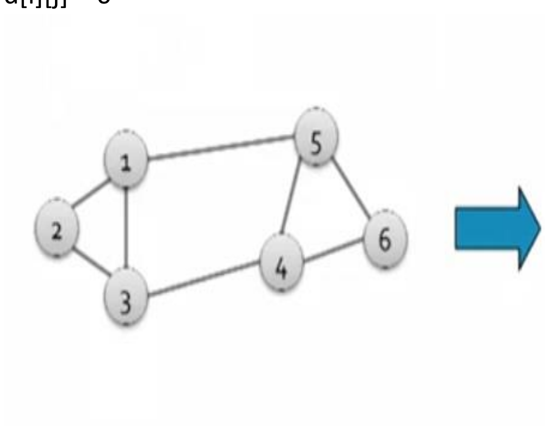
VIDA GHARAVIAN

- # basing graph theory notions:
  - Adjacency matrix (A)
    - Given a graph with n vertices and m nodes, the adjacency matrix is a square n*n matrix with the property:
    - A[i][j] = 1 if there is an edge between node i and node j, 0 otherwise



	1	2	3	4	5	6
1	0	1	1	0	1	0
2	1	0	1	0	0	0
3	1	1	0	1	0	0
4	0	0	1	0	1	1
5	1	0	0	1	0	1
6	0	0	0	1	1	0

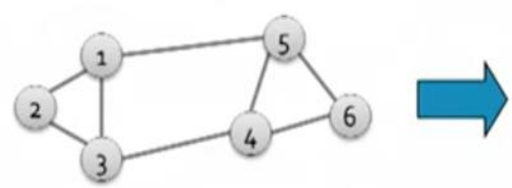
  - Degree matrix (D)
    - The degree matrix is a n*n diagonal matrix with the property
    - d[i][i] = the number of adjacent edges in node i or the degree of node i
    - d[i][j] = 0



	1	2	3	4	5	6
1	3	0	0	0	0	0
2	0	2	0	0	0	0
3	0	0	3	0	0	0
4	0	0	0	3	0	0
5	0	0	0	0	3	0
6	0	0	0	0	0	2

  - Laplacian matrix (L)
    - The laplacian matrix is a n*n matrix defined as: L = D –A

- Its eigen values are positive real numbers and the eigen vectors are real and orthogonal (the dot product of the 2 vectors is 0)



	1	2	3	4	5	6
1	3	-1	-1	0	-1	0
2	-1	2	-1	0	0	0
3	-1	-1	3	-1	0	0
4	0	0	-1	3	-1	-1
5	-1	0	0	-1	3	-1
6	0	0	0	-1	-1	2

- Conductance
- A measure of the connectivity of a group to the rest of the network relative to the density of the group (the number of edges that point outside the cluster divided by the sum of the degrees of the nodes in the cluster). The lower the conductance, the better the cluster.
- Calculating the eigen values and eigen vectors of A with x (n dimensional vector with the values of the nodes): A * x = lambda * x
- $$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \qquad y_i = \sum_{j=1}^{n} A_{ij} x_j = \sum_{(i,j) \in E} x_j$$

* Get input:

```python
def read_image() -> list:
    im = Image.open('flower.jpg', 'r')
    return list(im.getdata())
```

* Compute adjacency matrix:

```python
def adjacency_matrix(data: list) -> list:
    A = []
    for element in data:
        a = []
        for element2 in data:
            e = np.exp(-sum((np.power((element[i] - element2[i]), 2)) for i in 
range(len(element2))))
            a.append(e)
```

```
        A.append(a)
    return A
```

- degree matrix:

```
def degree_matrix(A: list) -> list:
    count = 0
    D = []
    for element in A:
        row = []
        sum = 0
        for i in range(len(element)):
            row.append(0)
            sum += element[i] if i != count else 0

        row[count] = sum
        D.append(row)
        count += 1
    return D
```

- laplacian matrix:

```
def laplacian_matrix(D, A):
    L = []
    for i in range(len(A)):
        row = []
        for j in range(len(A[0])):
            row.append(D[i][j] - A[i][j])
        L.append(row)
    return L
```

- Compute the eigen values (lambda) and eigen vectors (x) such that L* x = lambda*x

```
def compute_lambda(L):
    w, v = LA.eig(np.array(L))
    m = np.argmax(w)
    v = v[:, np.argsort(w)]
    w = w[np.argsort(w)]
    return v.real
```

- Find clusters in this subspace using various clustering algorithms, such as k-means
  - ```
    def ploting(v):
        kmeans = KMeans(n_clusters=8)
        kmeans.fit(v[:, 1:8])
        colors = kmeans.labels_
        return colors
    ```

- complete code :

- ```
  import numpy as np
  from PIL import Image
  from numpy import linalg as LA
  from sklearn.cluster import KMeans



  def read_image() -> list:
      im = Image.open('flower.jpg', 'r')
  ```

```
    return list(im.getdata())


def adjacency_matrix(data: list) -> list:
    A = []
    for element in data:
        a = []
        for element2 in data:
            e = np.exp(-sum((np.power((element[i] - element2[i]), 2)) for i in
range(len(element2))))
            a.append(e)
        A.append(a)
    return A


def degree_matrix(A: list) -> list:
    count = 0
    D = []
    for element in A:
        row = []
        sum = 0
        for i in range(len(element)):
            row.append(0)
            sum += element[i] if i != count else 0

        row[count] = sum
        D.append(row)
        count += 1
    return D


def laplacian_matrix(D, A):
    L = []
    for i in range(len(A)):
        row = []
        for j in range(len(A[0])):
            row.append(D[i][j] - A[i][j])
        L.append(row)
    return L


def conductance(A):
    pass


def compute_lambda(L):
    w, v = LA.eig(np.array(L))
    m = np.argmax(w)
    v = v[:, np.argsort(w)]
    w = w[np.argsort(w)]
    return v.real
```

```
def ploting(v):
    kmeans = KMeans(n_clusters=8)
    kmeans.fit(v[:, 1:8])
    colors = kmeans.labels_
    return colors


def get_centers(colors: np.ndarray, data):
    centers = []
    clusters = []
    for i in range(colors.size):
        if not clusters.__contains__(colors[i]):
            clusters.append(colors[i])
            centers.append(data[i])
    return centers


def graph_clustering(data):
    A = adjacency_matrix(data)
    D = degree_matrix(A)
    L = laplacian_matrix(D, A)
    return compute_lambda(L)


def make_image(colours,clusters,img):
    new_image = []
    for x in range(img.size[1]):
        new_image_row = []
        for i in range(img.size[0]):
            new_image_row.append(clusters[colours[x * (img.size[0]) + i]])
        new_image.append(new_image_row)
    new_image=np.asarray(new_image, dtype=np.uint8)
    new_image = Image.fromarray(new_image, 'RGB')
    new_image.save('flower_out2.jpg')
    new_image.show()

img = Image.open('flower.jpg', 'r')
data: list = read_image()
print(len(data))
v = graph_clustering(data[:10])
colors: np.ndarray = ploting(v)
centers = get_centers(colors, data)
counter = 1

while counter < int(len(data) / 10):
    new_data = []
    new_data.append(data[counter * 10:(counter + 1) * 10])
    new_data = np.append(new_data, centers).reshape(18, 3).tolist()
    v = graph_clustering(new_data)
    new_colors = ploting(v)
    colors = np.append(colors, new_colors[:10])
```

```
    counter += 1
print(colors.size)
make_image(colors,clusters=centers,img=img)
```

- **input:**
- 
- Output:
- 
- References:

- https://towardsdatascience.com/spectral-graph-clustering-and-optimal-number-of-clusters-estimation-32704189afbe
- https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.eig.html
- https://lpsa.swarthmore.edu/MtrxVibe/EigMat/MatrixEigen.html#:~:text=Any%20value%20of%20%CE%BB%20for,v%2D%CE%BB%C2%B7v%3D0
- https://www.researchgate.net/figure/An-example-graph-with-n-8-and-m-8-The-conductance-of-several-clusters-are-shown_fig2_301817085