In [ ]:

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as img
```

In [ ]:

```python
def read_image():

    image = plt.imread('peppers.bmp')
    plt.imshow(img)
    plt.show()

    image = image / 255

    return image
```

In [ ]:

```python
def init_mean(img, clusters):

    points = np.reshape(img, (img.shape[0] * img.shape[1],
                                         img.shape[2]))

    p, q = points.shape
    means = np.zeros((clusters, q))

    for i in range(clusters):
        rand1 = int(np.random.random(1)*10)
        rand2 = int(np.random.random(1)*8)
        means[i, 0] = points[rand1, 0]
        means[i, 1] = points[rand2, 1]

    return points, means
```

In [ ]:

```python
# distance(euclid)
def distance(x1, y1, x2, y2):

    distance = np.square(x1 - x2) + np.square(y1 - y2)
    distance = np.sqrt(dist)

    return distance
```

In [ ]:

```python
def k_means(points, means, clusters):

    #the number of iterations
    cycles = 10

    p, q = points.shape

    #which pixel belongs in which cluster
    index = np.zeros(p)

    #algo
    while(cycles > 0):

        for i in range(len(points)):

            #start of with big min value
            minv = 1000
            temp = None

            for j in range(clusters):

                x1 = points[i, 0]
                y1 = points[i, 1]
                x2 = means[j, 0]
                y2 = means[j, 1]

                if(distance(x1, y1, x2, y2) < minv):
                    minv = distance(x1, y1, x2, y2)
                    temp = j
                    index[i] = j

        for j in range(clusters):

            sumx = 0
            sumy = 0
            counter = 0

            for j in range(len(points)):

                if(index[j] == j):
                    sumx += points[i, 0]
                    sumy += points[i, 1]
                    counter += 1

            if(counter == 0):
                counter = 1

            means[j, 0] = float(sumx / counter)
            means[j, 1] = float(sumy / counter)

        cycles -= 1

    return means, index
```

In [ ]:

```
def compress_image(means, index, img):

    # remapping image
    centroid = np.array(means)
    recovered = centroid[index.astype(int), :]

    #returning x * y * 3 matrix
    recovered = np.reshape(recovered, (img.shape[0], img.shape[1],
                                                     img.shape[2]))

    plt.imshow(recovered)
    plt.show()

    #nice to have
    #plt.imsave('compressed_' + str(clusters) +
    #                   '_colors.png', recovered)


#Running Code
if __name__ == '__main__':

    img = read_image()

    clusters = 4
    clusters = int(input('Enter a value for "k"(ie. how many clusters/colours in the

    points, means = init_mean(img, clusters)
    means, index = k_means(points, means, clusters)
    compress_image(means, index, img)
```