TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA Công Nghệ Thông Tin

ĐỀ THI VÀ BÀI LÀM

Tên học phần: **Trí tuệ nhân tạo**
Mã học phần: Hình thức thi: *Tự luận có giám sát*
Đề số: **Đ0001** Thời gian làm bài: 75 phút *(không kể thời gian chép/phát đề)*
Được sử dụng tài liệu khi làm bài.

Họ tên: Trần Hoàng Phúc Huy. **Lớp:** 22T-DT1……..**MSSV**:102220067

Sinh viên làm bài trực tiếp trên tệp này, lưu tệp với định dạng MSSV_HọTên.pdf và nộp bài thông qua MSTeam:

Câu 1 (*3 điểm*): Một lâu đài cổ có hệ thống đường hầm bí mật, với một cửa vào duy nhất tại phòng trung tâm và nhiều cửa ra ở rìa lâu đài. Để đánh lạc hướng, hệ thống có thêm các nhánh hầm cụt và cửa giả. Hai ô hầm chỉ nối với nhau nếu có chung cạnh. Hãy giúp chủ lâu đài kiểm tra khả năng thoát hiểm từ phòng trung tâm đến rìa lâu đài bằng thuật toán A*, với hàm chi phí:

- f(x) = g(x) + h(x), trong đó:
  - g(x): chi phí từ điểm bắt đầu đến ô hiện tại.
  - h(x): khoảng cách Manhattan đến rìa lâu đài.

Dữ liệu đầu vào (file “A_in.csv”):

- Dòng 1: ba số nguyên dương n , D, C — kích thước lâu đài và tọa độ phòng trung tâm.
- Dòng 2 đến n+1: ma trận n x n, mỗi ô là:
  - `1`: có đường hầm (đi được),
  - `0`: không có (không đi được).

Kết quả đầu ra (file “A_out.csv”):

- Nếu không thoát được: ghi `-1`.
- Nếu thoát được:
  - Dòng đầu: số ô phải đi qua (m).
  - Tiếp theo m dòng: tọa độ các ô (dòng, cột) theo thứ tự đường đi từ phòng trung tâm đến một cửa ra.

Dữ liệu minh hoạ

A_in.csv	**A_out.csv**
5,2,2	6
0,0,1,0,0	2,2
0,1,1,1,0	1,2
0,0,1,0,0	1,1

1,1,1,0,0 0,0,1,1,1	2,1 3,1 3,0

a) Xác định hàm h(x)

 # Trả lời: Minh hoạ giải thích hàm
- Với vị trí (x, y) hiện tại ,tính khoảng cách còn lại đến 4 biên: trên, dưới, trái, phải
- Dùng độ đo Manhattan: abs(x1 - x2) + abs(y1 - y2)
- Chọn hướng đi có khoảng cách còn lại đến biên là ngắn nhất

 # Trả lời: Dán code hàm h(x):

```
# Tính khoảng cách Manhattan giữa hai điểm
def distance_function(x1, y1, x2, y2):
   return abs(x1 - x2) + abs(y1 - y2)

# Tính heuristic cho một ô
# Heuristic là khoảng cách ngắn nhất từ ô hiện tại đến biên của ma trận
def calculate_heuristic(x, y, n):
   if x == 0 or x == n - 1 or y == 0 or y == n - 1:
       return 0

   min = float('inf')

   # Tính khoảng cách đến 4 cạnh của ma trận
   # Lấy giá trị nhỏ nhất làm heuristic

   #Biên phải
   for k in range(n):
       min = min(min, distance_function(x, y, 0, k))

   #Biên trái
   for k in range(n):
       min = min(min, distance_function(x, y, n - 1, k))

   #Biên trên
   for k in range(n):
       min = min(min, distance_function(x, y, k, 0))

   #Biên dưới
   for k in range(n):
```

```
        min = min(min, distance_function(x, y, k, n - 1))

    return min
```

b) Viết chương trình hoàn thiện cho bài toán trên

 # Trả lời: Dán code vào bên dưới

```
import heapq

def distance_function(x1, y1, x2, y2):
    return abs(x1 - x2) + abs(y1 - y2)

def calculate_heuristic(x, y, n):
    if x == 0 or x == n - 1 or y == 0 or y == n - 1:
        return 0
    min_distance = float('inf')

    for k in range(n):
        min_distance = min(min_distance, distance_function(x, y, 0, k))


    for k in range(n):
        min_distance = min(min_distance, distance_function(x, y, n - 1, k))


    for k in range(n):
        min_distance = min(min_distance, distance_function(x, y, k, 0))


    for k in range(n):
        min_distance = min(min_distance, distance_function(x, y, k, n - 1))

    return min_distance


def solve_maze(n, start_x, start_y, maze):

    open_list = []
    closed_list_g = {}

    g = 0
```

```
    h = calculate_heuristic(start_x, start_y, n)
    f = g + h
    heapq.heappush(open_list, (f, g, start_x, start_y, None))
    closed_list_g[(start_x, start_y)] = (0, None)


    dx = [-1, 1, 0, 0]
    dy = [0, 0, -1, 1]

    while open_list:
        f, g, x, y, parent = heapq.heappop(open_list)
        if x == 0 or x == n - 1 or y == 0 or y == n - 1:

            path = []
            current = (x, y)
            while parent is not None:
                path.append(current)
                current = parent
                parent = closed_list_g.get(current, (None, None))[1]
            path.append(current)
            return path[::-1]


        for i in range(4):
            new_x, new_y = x + dx[i], y + dy[i]


            if 0 <= new_x < n and 0 <= new_y < n and maze[new_x][new_y] == 1:
                new_g = g + 1


                if (new_x, new_y) not in closed_list_g or new_g <
closed_list_g[(new_x, new_y)][0]:
                    new_h = calculate_heuristic(new_x, new_y, n)
                    new_f = new_g + new_h
                    heapq.heappush(open_list, (new_f, new_g, new_x, new_y, (x,
y)))
                    closed_list_g[(new_x, new_y)] = (new_g, (x, y))

    return []
n, start_x, start_y = 0, 0, 0
```

```
maze = []
with open("A_in.csv", "r") as f_in:
    lines = f_in.readlines()
    print(lines[0].strip().split(","))
    n_str, start_x_str, start_y_str = lines[0].strip().split(",")
    n = int(n_str)
    start_x = int(start_x_str)
    start_y = int(start_y_str)
    for i in range(1, n + 1):
        row = list(map(int, lines[i].strip().split(",")))
        maze.append(row)
path = solve_maze(n, start_x, start_y, maze)

with open("A_out.csv", "w") as f_out:
    if not path:
        f_out.write("-1\n")
    else:
        f_out.write(f"{len(path)}\n")
        for x, y in path:
            f_out.write(f"{x},{y}\n")
```

 # Trả lời: Giải thích chương trình

- Sử dụng hàm độ đo Manhattan để tính khoảng cách còn lại cần đi
- Hàm h(x) lấy hướng đi với khoảng cách đến 1 trong 4 cạnh còn lại bé nhất
- Ở thuật toán A*: Thực hiện thêm node đầu tiên vào trong cây, với toạ độ của đề bài, g(x) = 0 và h(x) tính theo toạ độ hiện tại, không có bước đi kề đó
- Tiến hành vòng lặp:
    - Lấy node gần nhất trong heap tree
    - Kiểm tra điều kiện thắng: nếu toạ độ (x, y) ứng với toạ độ của các điểm là cạnh của lâu đài (x = 0 || x = n -1, y = 0 || y - 1) thì là kết thúc bài toán, kết luận có đường đi ngắn nhất
    - Nếu không, lấy vị trí 4 điểm gần với điểm hiện tại nhất; với mỗi điểm:
        - Xét có thể đi được hay không; nếu không, bỏ qua
        - Xét nếu đã nằm trong heap tree hay không; nếu không, bỏ qua
        - Xét nếu di chuyển đến ô tiếp theo có tối ưu hay không: new_f = new_g + new_h
        - Thực hiện thêm các thông tin về x, y, f, g và node cha vào trong heap tree, tiếp tục vòng lặp
    - Nếu đã lặp hết các trường hợp mà không ra kết quả -> -1

c) Kết quả thực thi trên tệp "A_in.csv"

 # Trả lời: Dán kết quả kết quả A_out.csv vào bên dưới

```
7
5,5
5,6
5,7
```

```
5,8
6,8
7,8
7,9
```

Câu 2 (*4 điểm*): Cho tập dữ liệu input.csv với 75 mẫu dữ liệu, mỗi mẫu có 4 đặc trưng (chiều dài đài hoa, chiều rộng đài hoa, chiều dài cánh hoa, chiều rộng cánh hoa) và tên loài hoa tương ứng.

a) *(1 điểm) Xây dựng hàm mục tiêu (hàm mất mát) cho bài toán*

 # Trả lời: Dán hàm mất mát vào đây:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\log(p_{ic})$$

- N: số mẫu
- C: số loại hoa
- y(ic) = 1 nếu đúng nhãn, = 0 nếu sai nhãn
- p(ic): độ chính xác khi đoán nhãn dựa trên đặc trưng (probability)

 # Trả lời: Dán code của hàm loss:

```
def cross_entropy_loss(self, y_true, y_pred):
        epsilon = 1e-15
        y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
        return -np.mean(np.sum(y_true * np.log(y_pred), axis=1))
```

Sử dụng elipson rất bé để tránh đường hợp y_pred = 0 -> lỗi

b) (2 điểm) Hãy viết chương trình phân loại hoa trên.

 # Trả lời: Dán code vào đây

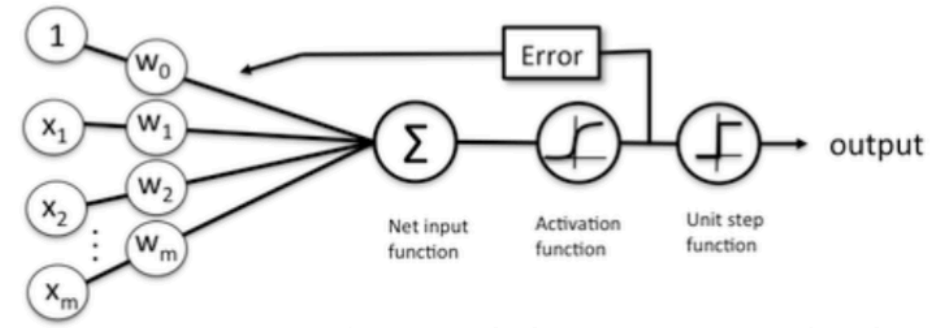
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report

def train_test_split(X, y, test_size=0.2, random_state=42):
    np.random.seed(random_state)
```

```
    n_samples = len(X)
    indices = np.arange(n_samples)
    np.random.shuffle(indices)

    split_idx = int(n_samples * (1 - test_size))

    train_indices = indices[:split_idx]
    test_indices = indices[split_idx:]
    X_train = X[train_indices]
    X_test = X[test_indices]
    y_train = y[train_indices]
    y_test = y[test_indices]

    return X_train, X_test, y_train, y_test

class LogisticRegression:
    def __init__(self, learning_rate=0.01, num_iterations=1000):
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations
        self.weights = None
        self.bias = None
        self.classes = None
        self.loss_history = []

    def cross_entropy_loss(self, y_true, y_pred):
        epsilon = 1e-15
        y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
        return -np.mean(np.sum(y_true * np.log(y_pred), axis=1))

    def softmax(self, z):
        exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
        return exp_z / np.sum(exp_z, axis=1, keepdims=True)

    def one_hot_encode(self, y):
        n_samples = len(y)
        n_classes = len(self.classes)
        one_hot = np.zeros((n_samples, n_classes))
        for i, label in enumerate(y):
            one_hot[i, np.where(self.classes == label)[0][0]] = 1
        return one_hot
```

```
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.classes = np.unique(y)
        n_classes = len(self.classes)


        self.weights = np.random.randn(n_features, n_classes) * 0.01
        self.bias = np.zeros(n_classes)


        y_one_hot = self.one_hot_encode(y)


        for _ in range(self.num_iterations):
            linear_pred = np.dot(X, self.weights) + self.bias
            predictions = self.softmax(linear_pred)

            dw = (1/n_samples) * np.dot(X.T, (predictions - y_one_hot))
            db = (1/n_samples) * np.sum(predictions - y_one_hot, axis=0)


            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db
            loss = self.cross_entropy_loss(y_one_hot, predictions)
            self.loss_history.append(loss)

    def predict(self, X):
        linear_pred = np.dot(X, self.weights) + self.bias
        predictions = self.softmax(linear_pred)
        return self.classes[np.argmax(predictions, axis=1)]


def load_data(input_file):
    data = pd.read_csv(input_file, header=None)
    X = data.iloc[:, :4].values  # đặc trưng 4 cột đầu
    y = data.iloc[:, 4].values   # nhãn ở cột cuối
    return X, y

X, y = load_data('input_2.csv')
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)



scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression(learning_rate=0.1, num_iterations=1000)
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)
print("\nModel Evaluation on Test Set:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

output_data = pd.read_csv('output_2.csv', header=None).values
output_scaled = scaler.transform(output_data)
predictions = model.predict(output_scaled)

with open('predictions.txt', 'w') as f:
    for pred in predictions:
        f.write(f"{pred}\n")
```

 # Trả lời: Dán kiến trúc mạng và giải thích làm thế nào để phân loại?



Schematic of a logistic regression classifier.

Mô hình sử dụng hồi quy logistic đa lớp (multiclass logistic regression)

- Khi dự đoán một bông hoa mới, quá trình diễn ra như sau:
- Đầu tiên, tính toán giá trị tuyến tính bằng cách nhân ma trận đặc trưng với trọng số và cộng bias
- Sau đó, áp dụng hàm softmax để chuyển đổi các giá trị thành xác suất
- Cuối cùng, chọn lớp có xác suất cao nhất làm dự đoán

Quá trình train này lặp đi lặp lại cho tới khi đủ epoch
Model sau đó được sử dụng để nhận các đặc trưng và cho ra nhãn có xác suất cao nhất ứng với các đặc trưng trên.

c) *(1 điểm)* Hãy thực thi chương trình và cho biết nhãn của 30 mẫu dữ liệu trong output.csv

 # Trả lời: Dán code thực thi thành công

```
Model Evaluation on Test Set:
Accuracy: 0.9333333333333333

Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         3
Iris-versicolor       0.83      1.00      0.91         5
 Iris-virginica       1.00      0.86      0.92         7

       accuracy                           0.93        15
      macro avg       0.94      0.95      0.94        15
   weighted avg       0.94      0.93      0.93        15
```

 # Trả lời: Dán kết quả nhãn ứng với 30 mẫu dữ liệu
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
Iris-versicolor
Iris-versicolor
Iris-versicolor
Iris-versicolor
Iris-versicolor
Iris-versicolor
Iris-versicolor
Iris-versicolor
Iris-versicolor
Iris-versicolor
Iris-versicolor
Iris-versicolor
Iris-versicolor
Iris-virginica

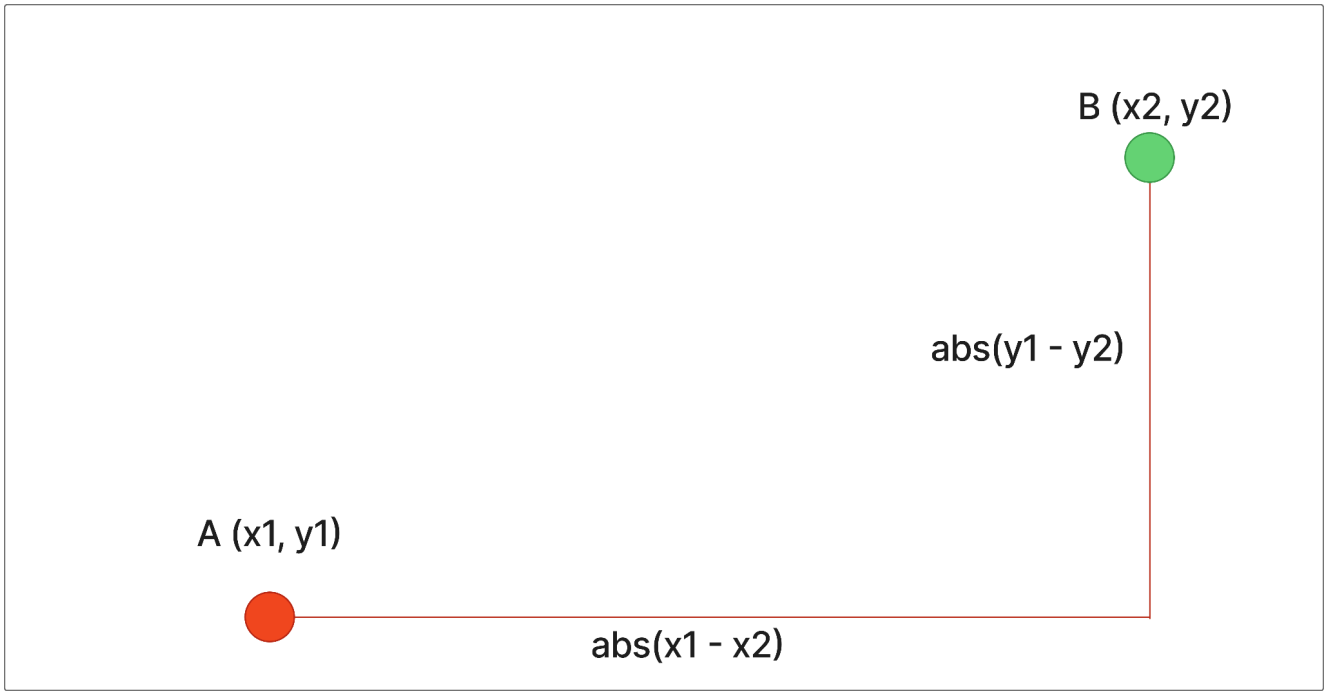
Iris-virginica
Iris-virginica
Iris-virginica
Iris-virginica
Iris-versicolor
Iris-virginica

Câu 3 (3 *điểm*): Cho tập dữ liệu Countries.csv. Hãy viết chương trình phân cụm bằng thuật toán *k*-means

a) *(1 điểm) Xây dựng hàm đo khoảng cách sử dụng độ đo* Manhattan

 # Trả lời: Minh hoạ tính khoảng cách:

- Sử dụng hàm độ đo Manhattan, với khoảng cách của 2 điểm A(x1, y1) và B(x2, y2 được tính): distance = abs(x1 - x2) + abs(y1 - y2)



 # Trả lời: Dán code hàm tính khoảng cách:

```
def distance_function(x1, y1, x2, y2):
    return np.abs(x1 - x2) + np.abs(y1 - y2)
```

b) (1 điểm) Xây dựng hàm chứa thuật toán *k*-means để phân cụm

 # Trả lời: Dán code về hàm

```
def distance_function(x1, y1, x2, y2):
    return np.abs(x1 - x2) + np.abs(y1 - y2)

def initialize_centroids(X, K):
    m, n = X.shape
    k_rand = np.ones((K, n))
    k_rand = X[np.random.choice(m, K, replace=False)]
    return k_rand
```

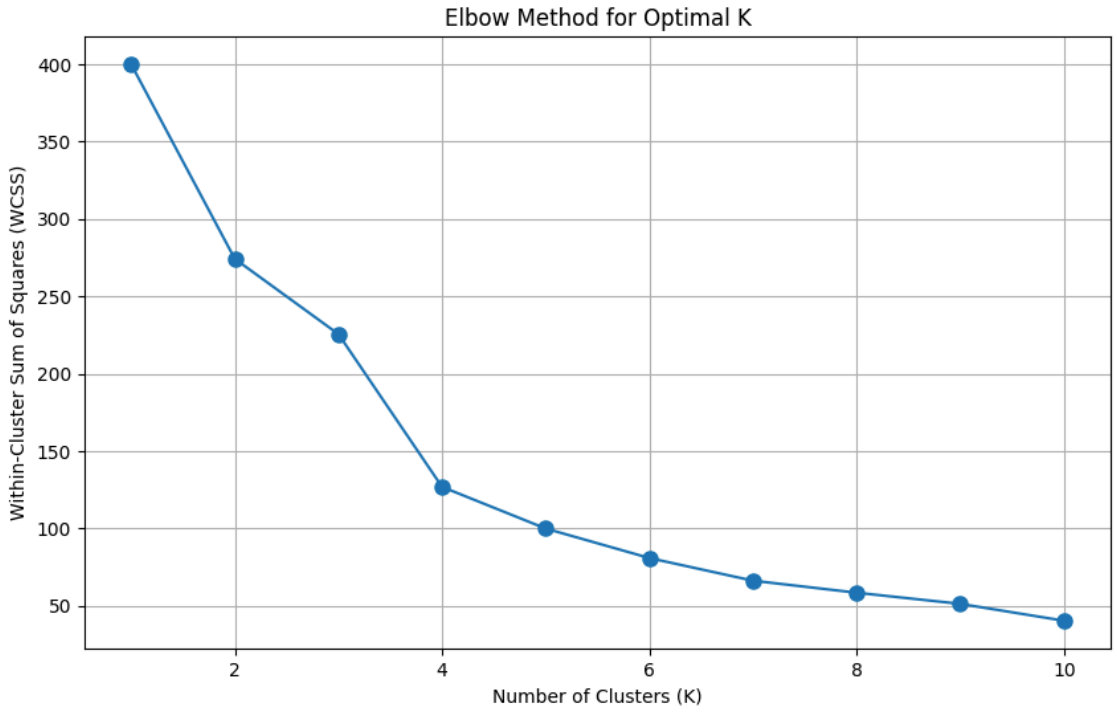
```
def find_closest_centroids(X, centroids):
    m = len(X)
    c = np.zeros(m)
    for i in range(m):
        distances = np.array([distance_function(X[i][0], X[i][1], centroid[0],
centroid[1])
                              for centroid in centroids])
        c[i] = np.argmin(distances)
    return c

def compute_means(X, idx, K):
    m, n = X.shape
    centroids = np.zeros((K, n))
    for k in range(K):
        points_belong_k = X[np.where(idx == k)]
        centroids[k] = np.mean(points_belong_k, axis=0)
    return centroids



def calculate_wcss(X, centroids, idx):
    wcss = 0
    for i in range(len(X)):
        cluster_idx = int(idx[i])
        wcss += np.sum((X[i] - centroids[cluster_idx]) ** 2)
    return wcss

def find_k_means(X, K, max_iters=10):
    m, n = X.shape
    centroids = initialize_centroids(X, K)
    centroids_history = np.zeros((max_iters, K, n))

    for _ in range(max_iters):
        idx = find_closest_centroids(X, centroids)
        centroids = compute_means(X, idx, K)

    return centroids, idx

centroids, idx = find_k_means(X_scaled, optimal_k, max_iters=100)
```

c) *(1 điểm)* Xây dựng hàm để khảo sát việc lựa chọn k

 # Trả lời: Dán code về hàm và giải thích cách lựa chọn k phù hợp

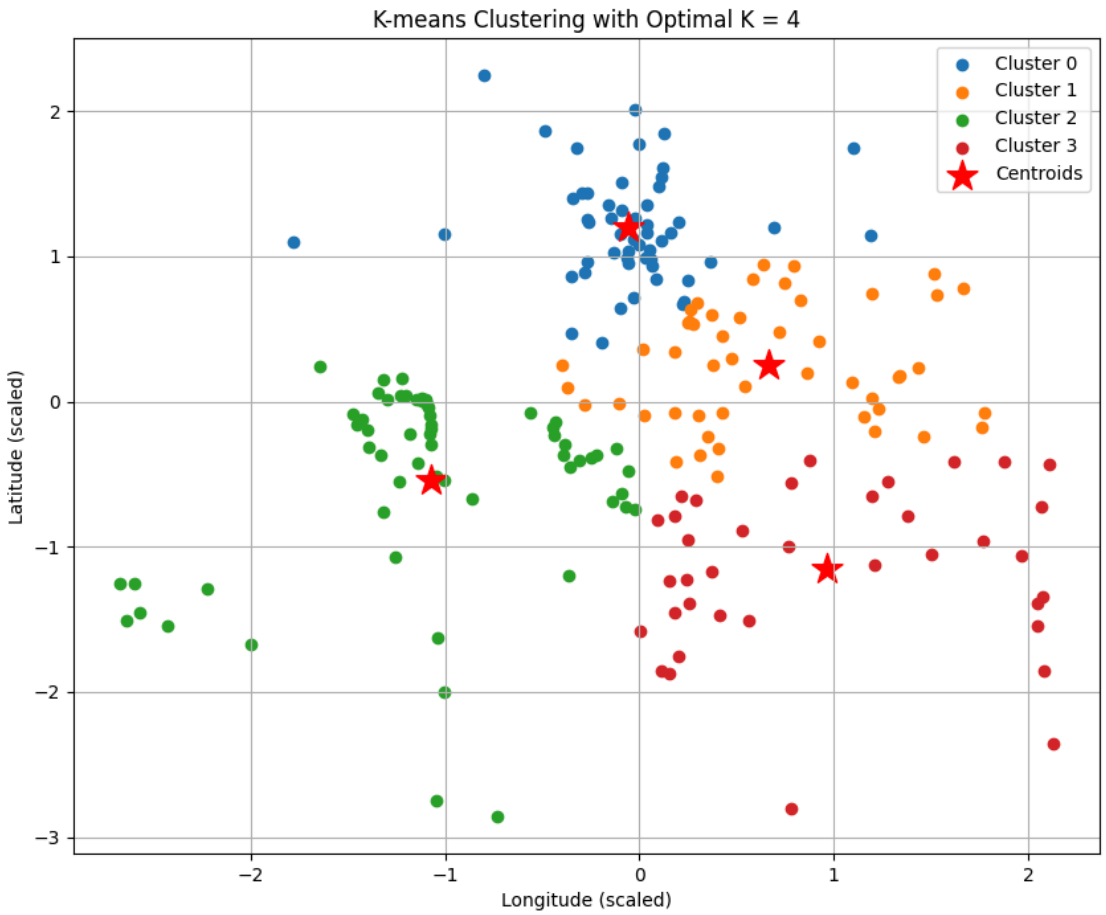
```
def find_optimal_k(X, k_range, max_iters=10):
    wcss_values = []
    for k in k_range:
        centroids, idx = find_k_means(X, k, max_iters)
        wcss = calculate_wcss(X, centroids, idx)
        wcss_values.append(wcss)
        print(f"K = {k}, WCSS = {wcss:.2f}")
```

- Để chọn K tối ưu, cần khảo sát mức độ tương qua của K với WCSS (tổng bình phương khoảng cách từ các điểm lân cận tới các điểm phân cụm)
    - K quá nhỏ: WCSS quá lớn, phân cụm không có ý nghĩa
    - K quá lớn: WCSS quá nhỏ, phân cụm quá rời rạc, tốn chi phí tính toán
- Với mỗi giá trị K, ta cần tính WCSS của toàn bộ các điểm phân cụm tới các điểm khác. Chọn K mà WCSS giảm sâu nhất (thuyết khuỷu tay)



Có thể thấy với K = 4 -> WCSS giảm đáng kể nhất -> chọn K = 4

 # Trả lời: Dán kết quả thi với k(lưu ý có giải thích và bình luận):
Với K = 4, chọn được 4 điểm mốc để phân cụm dữ liệu từ dataset




```
name,Longitude,Latitude,Cluster
China,103.8190735,36.56176546,1.0
CÃ´te d'Ivoire,-5.5692157,7.6284262,2.0
Cameroon,12.73964156,5.69109849,2.0
Dem. Rep. Congo,23.64396107,-2.87746289,3.0
Congo,15.21965762,-0.83787463,2.0
Cook Is.,-159.7872422,-21.21927288,2.0
Colombia,-73.08114582,3.91383431,2.0
Comoros,43.68253968,-11.87783444,3.0
Cape Verde,-23.9598882,15.95523324,2.0
Costa Rica,-84.19208768,9.97634464,2.0
Cuba,-79.01605384,21.62289528,2.0
CuraÃ§ao,-68.97119369,12.19551675,2.0
Cayman Is.,-80.91213321,19.42896497,2.0
N. Cyprus,33.5684813,35.26277486,0.0
Cyprus,33.0060022,34.91667211,0.0
Czech Rep.,15.31240163,49.73341233,0.0
Germany,10.38578051,51.10698181,0.0
Djibouti,42.5606754,11.74871806,1.0
```

```
Dominica,-61.357726,15.4394702,2.0
Denmark,10.02800992,55.98125296,0.0
Dominican Rep.,-70.50568896,18.89433082,2.0
Algeria,2.61732301,28.15893849,0.0
Ecuador,-78.75201922,-1.42381612,2.0
Egypt,29.86190099,26.49593311,1.0
Eritrea,38.84617011,15.36186618,1.0
Spain,-3.64755047,40.24448698,0.0
Estonia,25.54248537,58.67192972,0.0
Ethiopia,39.60080098,8.62278679,1.0
Finland,26.2746656,64.49884603,0.0
Fiji,165.4519543,-17.42858032,3.0
Falkland Is.,-59.35238956,-51.74483954,2.0
France,-2.76172945,42.17344011,0.0
Faeroe Is.,-6.88095423,62.05385403,0.0
Micronesia,153.2394379,7.45246814,3.0
Gabon,11.7886287,-0.58660025,2.0
United Kingdom,-2.86563164,54.12387156,0.0
Georgia,43.50780252,42.16855755,0.0
Guernsey,-2.57239064,49.46809761,0.0
Ghana,-1.21676566,7.95345644,2.0
Guinea,-10.94066612,10.43621593,2.0
Gambia,-15.39601295,13.44965244,2.0
Guinea-Bissau,-14.94972445,12.04744948,2.0
Eq. Guinea,10.34137924,1.70555135,2.0
Greece,22.95555794,39.07469623,0.0
Grenada,-61.68220189,12.11725044,2.0
Greenland,-41.34191127,74.71051289,0.0
Guatemala,-90.36482009,15.69403664,2.0
Guam,144.7679102,13.44165626,1.0
Guyana,-58.98202459,4.79378034,2.0
Hong Kong,114.1138045,22.39827737,1.0
Heard I. and McDonald Is.,73.5205171,-53.08724656,3.0
Honduras,-86.6151661,14.82688165,2.0
Croatia,16.40412899,45.08047631,0.0
Haiti,-72.68527509,18.93502563,2.0
Hungary,19.39559116,47.16277506,0.0
Indonesia,117.2401137,-2.21505456,3.0
Isle of Man,-4.53873952,54.22418911,0.0
India,79.6119761,22.88578212,1.0
Indian Ocean Ter.,104.851898,-10.6478515,3.0
```

```
Br. Indian Ocean Ter.,72.44541229,-7.33059751,3.0
Ireland,-8.13793569,53.1754487,0.0
Iran,54.27407004,32.57503292,1.0
Iraq,43.74353149,33.03970582,1.0
Iceland,-18.57396167,64.99575386,0.0
Israel,35.00444693,31.46110101,1.0
Italy,12.07001339,42.79662641,0.0
Jamaica,-77.31482593,18.15694878,2.0
Jersey,-2.12689938,49.21837377,0.0
Jordan,36.77136104,31.24579091,1.0
Japan,138.0308956,37.59230135,1.0
Siachen Glacier,77.18011865,35.39236325,1.0
Kazakhstan,67.29149357,48.15688067,0.0
Kenya,37.79593973,0.59988022,3.0
Kyrgyzstan,74.54165513,41.46221943,1.0
Cambodia,104.9069433,12.72004786,1.0
Kiribati,-45.61110513,0.86001503,2.0
St. Kitts and Nevis,-62.68755265,17.2645995,2.0
Korea,127.8391609,36.38523983,1.0
Kosovo,20.87249811,42.57078707,0.0
Kuwait,47.58700459,29.33431262,1.0
Lao PDR,103.7377241,18.50217433,1.0
Lebanon,35.88016072,33.92306631,1.0
Liberia,-9.32207573,6.45278492,2.0
Libya,18.00866169,27.03094495,1.0
Saint Lucia,-60.96969923,13.89479481,2.0
Liechtenstein,9.53574312,47.13665835,0.0
Sri Lanka,80.70108238,7.61266509,3.0
Lesotho,28.22723131,-29.58003188,3.0
Lithuania,23.88719355,55.32610984,0.0
Luxembourg,6.07182201,49.76725361,0.0
Latvia,24.91235983,56.85085163,0.0
Macao,113.5093212,22.22311688,1.0
St-Martin,-63.05972851,18.08888611,2.0
Morocco,-8.45615795,29.83762955,0.0
Monaco,7.40627677,43.75274627,0.0
Moldova,28.45673372,47.19498804,0.0
Madagascar,46.70473674,-19.37189587,3.0
Maldives,73.45713004,3.7287092,3.0
Mexico,-102.5234517,23.94753724,2.0
Marshall Is.,170.3397612,7.00376358,3.0
```

```
Macedonia,21.68211346,41.59530893,0.0
Mali,-3.54269065,17.34581581,1.0
Malta,14.40523316,35.92149632,0.0
Myanmar,96.48843321,21.18566599,1.0
Montenegro,19.23883939,42.78890259,0.0
Mongolia,103.0529977,46.82681544,0.0
N. Mariana Is.,145.6196965,15.82927563,1.0
Mozambique,35.53367543,-17.27381643,3.0
Mauritania,-10.34779815,20.25736706,1.0
Montserrat,-62.18518546,16.73941406,2.0
Mauritius,57.57120551,-20.27768704,3.0
Malawi,34.28935599,-13.21808088,3.0
Malaysia,109.6976228,3.78986846,3.0
Namibia,17.20963567,-22.13032568,3.0
New Caledonia,165.6849237,-21.29991806,3.0
Niger,9.38545882,17.41912493,1.0
Norfolk Island,167.9492168,-29.0514609,3.0
Nigeria,8.08943895,9.59411452,2.0
Nicaragua,-85.0305297,12.84709429,2.0
Niue,-169.8699468,-19.04945708,2.0
Netherlands,5.28144793,52.1007899,0.0
Norway,15.34834656,68.75015572,0.0
Nepal,83.9158264,28.24891365,1.0
Nauru,166.9325682,-0.51912639,3.0
New Zealand,171.4849235,-41.81113557,3.0
Oman,56.09166155,20.60515333,1.0
Pakistan,69.33957937,29.9497515,1.0
Panama,-80.11915156,8.51750797,2.0
Pitcairn Is.,-128.317042,-24.36500535,2.0
Peru,-74.38242685,-9.15280381,2.0
Philippines,122.8839325,11.77536778,1.0
Palau,134.4080797,7.28742784,3.0
Papua New Guinea,145.2074475,-6.46416646,3.0
Poland,19.39012835,52.12759564,0.0
Puerto Rico,-66.47307604,18.22813055,2.0
Dem. Rep. Korea,127.1924797,40.15350311,1.0
Portugal,-8.50104361,39.59550671,0.0
Paraguay,-58.40013703,-23.22823913,2.0
Palestine,35.19628705,31.91613893,1.0
Fr. Polynesia,-144.9049439,-14.72227409,2.0
Qatar,51.18479632,25.30601188,1.0
```

```
Romania,24.97293039,45.85243127,0.0
Russia,96.68656112,61.98052209,0.0
Rwanda,29.91988515,-1.99033832,3.0
W. Sahara,-12.21982755,24.22956739,1.0
Saudi Arabia,44.53686271,24.12245841,1.0
Sudan,29.94046812,15.99035669,1.0
S. Sudan,30.24790002,7.30877945,1.0
Senegal,-14.4734924,14.36624173,2.0
Singapore,103.8172559,1.35876087,3.0
S. Geo. and S. Sandw. Is.,-36.43318388,-54.46488248,2.0
Saint Helena,-9.54779416,-12.40355951,2.0
Solomon Is.,159.6328767,-8.92178022,3.0
Sierra Leone,-11.79271247,8.56329593,2.0
El Salvador,-88.87164469,13.73943744,2.0
San Marino,12.45922334,43.94186747,0.0
Somaliland,46.25198395,9.73345496,1.0
Somalia,45.70714487,4.75062876,1.0
St. Pierre and Miquelon,-56.30319779,46.91918789,0.0
Serbia,20.78958334,44.2215032,0.0
SÃ£o TomÃ© and Principe,6.72429658,0.44391445,2.0
Suriname,-55.9123457,4.13055413,2.0
Slovakia,19.47905218,48.70547528,0.0
Slovenia,14.80444238,46.11554772,0.0
Sweden,16.74558049,62.77966519,0.0
Swaziland,31.4819369,-26.55843045,3.0
Sint Maarten,-63.05713363,18.05081728,2.0
Seychelles,55.47603279,-4.66099094,3.0
Syria,38.50788204,35.02547389,1.0
Turks and Caicos Is.,-71.97387881,21.83047572,2.0
Chad,18.64492513,15.33333758,1.0
Togo,0.96232845,8.52531356,2.0
Thailand,101.0028813,15.11815794,1.0
Tajikistan,71.01362631,38.5304539,1.0
Turkmenistan,59.37100021,39.11554137,1.0
Timor-Leste,125.8443898,-8.82889162,3.0
Tonga,-174.8098734,-20.42843174,2.0
Trinidad and Tobago,-61.26567923,10.45733408,2.0
Tunisia,9.55288359,34.11956246,0.0
Turkey,35.16895346,39.0616029,0.0
Taiwan,120.9542728,23.7539928,1.0
Tanzania,34.81309981,-6.27565408,3.0
```

```
Uganda,32.36907971,1.27469299,3.0
Ukraine,31.38326469,48.99656673,0.0
Uruguay,-56.01807053,-32.79951534,2.0
United States,-112.4616737,45.6795472,0.0
Uzbekistan,63.14001528,41.75554225,1.0
Vatican,12.43387177,41.90174985,0.0
St. Vin. and Gren.,-61.20129695,13.22472269,2.0
Venezuela,-66.18184123,7.12422421,2.0
British Virgin Is.,-64.47146992,18.52585755,2.0
U.S. Virgin Is.,-64.80301538,17.95500624,2.0
Vietnam,106.299147,16.6460167,1.0
Vanuatu,167.6864464,-16.22640909,3.0
Wallis and Futuna Is.,-177.3483483,-13.88737039,2.0
Samoa,-172.1648506,-13.75324346,2.0
Yemen,47.58676189,15.90928005,1.0
South Africa,25.08390093,-29.00034095,3.0
Zambia,27.77475946,-13.45824152,3.0
Zimbabwe,29.8514412,-19.00420419,3.0
```

Đà Nẵng, ngày 20 tháng 05 năm 2025

GIẢNG VIÊN BIÊN SOẠN ĐỀ THI

TRƯỞNG BỘ MÔN
(đã duyệt)