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: **00002** Thời gian làm bài: 70 phút *(không kể thời gian chép/phát đề)*
Được sử dụng tài liệu khi làm bài.

---

**Họtên:……..**Trần Phương Nam…….**Lớp**:……21TCLC_DT3……..**MSSV**:……102210218……
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*): Cho bài toán như sau: Trong các lâu đài cổ người ta thường xây dựng các đường hầm bí mật để thoát hiểm trong các trường hợp khẩn cấp. Các đường hầm chỉ có thể vào từ một cửa vào duy nhất tại phòng Trung tâm và thoát ra ở rất nhiều cửa ra. Các cửa ra đều nằm ở rìa lâu đài, do vậy, nếu thoát ra được rìa lâu đài thì coi như đã thoát hiểm. Để ngụy trang, người ta cho đào nhiều nhánh hầm cụt và cửa vào giả. Ngoài ra, để tăng khả năng thoát hiểm, người ta còn xây dựng các đường hầm giao nhau tại một số vị trí. Để nghiệm thu công trình, chủ lâu đài cần kiểm tra xem từ phòng trung tâm có thể thoát hiểm qua hệ thống đường hầm hay không. Hãy sử dụng thuật toán **DFS** giúp chủ lâu đài kiểm tra hệ thống trên. Biết rằng lâu đài là một hình vuông được chia lưới ô vuông gồm n dòng, n cột. Trên đồ hoạ, ô ở dòng i cột j được ghi số 1 nếu có đường hầm, số 0 nếu không có (ô ở góc trên trái có toạ độ (0,0)). 2 ô chỉ có thể thông nhau nếu chúng có chung cạnh.

Dữ liệu nhập vào từ tập tin văn bản "bfs_dfs.csv" gồm:

- Dòng đầu chứa 3 số nguyên dương n < 30, D và C ( trong đó D, C là dòng và cột của phòng trung tâm).

- n dòng tiếp theo, mỗi dòng chứa n số là các số ở các vị trí tương ứng trên họa đồ.

Kết quả tìm được ghi ra tập tin văn bản "bfs_dfs_out.csv". Dòng đầu chứa số m là số ô phải đi qua, nếu không thoát được thì m = -1. Trong trường hợp thoát được, m dòng tiếp theo: mỗi dòng chứa 2 số là số hiệu dòng cột của các ô phải đi qua theo đúng trình tự của một cách thoát hiểm.

| Ví dụ: bfs_dfs.csv | Tệp bfs_dfs_out.csv(lưu ý đây chỉ là một phương án) |
|---|---|
| **4 2 1**<br>**0 1 1 0**<br>**1 0 0 1**<br>**1 1 1 1**<br>**0 1 1 0** | **3**<br>**2 1**<br>**2 2**<br>**3 2** |

a) (*1 điểm*)Viết hàm **DFS** để giải quyết bài toán trên

**# Trả lời:** Dán code vào bên dưới

```
def dfs(n, start_row, start_col, grid):
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    stack = [(start_row, start_col, [(start_row, start_col)])]
    visited = set()

    while stack:
        x, y, path = stack.pop()
        if x in [0, n-1] or y in [0, n-1]:
```

```
            return path  # Escape found
        visited.add((x, y))
        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if is_valid_move(nx, ny, n, grid, visited):
                stack.append((nx, ny, path + [(nx, ny)]))
    return -1  # No escape found
```

b) (*1 điểm*)Viết chương trình hoàn thiện cho bài toán trên

**# Trả lời:** Dán code vào bên dưới

```
import csv
from google.colab import files
from io import StringIO

def read_input(file_content):
    reader = csv.reader(file_content)
    first_line = next(reader)
    n = int(first_line[0])
    start_row = int(first_line[1])
    start_col = int(first_line[2])
    grid = []
    for line in reader:
        grid.append(list(map(int, line)))
    return n, start_row, start_col, grid

def write_output(path, path_list):
    with open(path, 'w', newline='') as file:
        writer = csv.writer(file)
        if path_list == -1:
            writer.writerow([-1])
        else:
            writer.writerow([len(path_list)])
            for position in path_list:
                writer.writerow(position)

def is_valid_move(x, y, n, grid, visited):
    return 0 <= x < n and 0 <= y < n and grid[x][y] == 1 and (x, y) not in
visited

def dfs(n, start_row, start_col, grid):
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    stack = [(start_row, start_col, [(start_row, start_col)])]
    visited = set()

    while stack:
        x, y, path = stack.pop()
        if x in [0, n-1] or y in [0, n-1]:
            return path  # Escape found
        visited.add((x, y))
        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if is_valid_move(nx, ny, n, grid, visited):
```

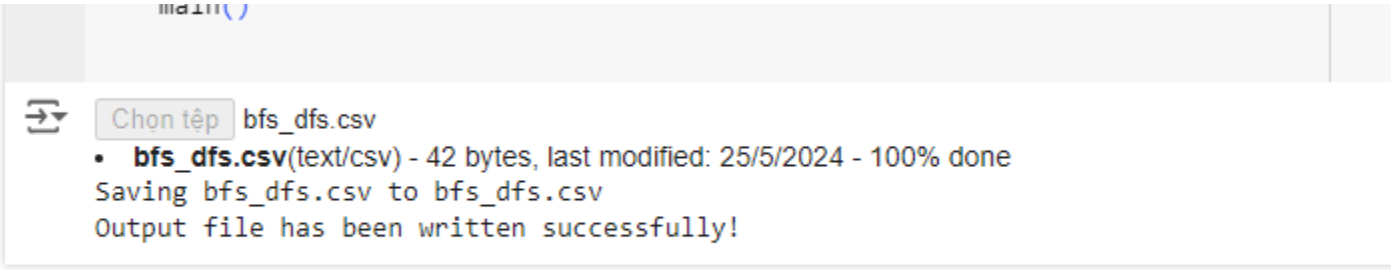
```
                stack.append((nx, ny, path + [(nx, ny)]))
    return -1  # No escape found

def main():
    uploaded = files.upload()
    for fn in uploaded.keys():
        input_content = uploaded[fn].decode('utf-8').splitlines()
        n, start_row, start_col, grid = read_input(input_content)
        path = dfs(n, start_row, start_col, grid)
        output_file = 'bfs_dfs_out.csv'
        write_output(output_file, path)
        print("Output file has been written successfully!")

if __name__ == "__main__":
    main()
```

c) (*1 điểm*)Kết quả thực thi trên tệp "bfs_dfs_out.csv

**# Trả lời:** Dán kết quả vào bên dưới và kèm lời giải thích




| A | B |
|---|---|
| 3 | |
| 2 | 1 |
| 2 | 2 |
| 2 | 3 |

***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) *(2 điểm)* Hãy viết chương trình phân loại hoa trên cơ sở dùng Logistic Regression kết hợp với lớp softmax.

**# Trả lời:** Dán code vào đây

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import torch
from torch.utils.data import DataLoader, TensorDataset
import torch.nn as nn
import torch.optim as optim
from torch.nn.functional import cross_entropy
from IPython.display import display
import ipywidgets as widgets
import io

# Define the IrisClassifier model
class IrisClassifier(nn.Module):
    def __init__(self):
        super(IrisClassifier, self).__init__()
        self.fc1 = nn.Linear(4, 10)  # Lớp ẩn với 10 nơ-ron
        self.fc2 = nn.Linear(10, 3)  # Lớp đầu ra với 3 loài hoa

    def forward(self, x):
        x = torch.relu(self.fc1(x))  # Hàm kích hoạt ReLU cho lớp ẩn
        x = self.fc2(x)
        return x

# Function to handle file upload
def upload_file():
    upload_widget = widgets.FileUpload(accept='.csv', multiple=False)
    display(upload_widget)

    def on_upload_change(change):
        if len(change['new']) > 0:
            uploaded_file = list(change['new'].values())[0]
            content = uploaded_file['content']
            data = pd.read_csv(io.BytesIO(content), header=None)
            process_data(data)

    upload_widget.observe(on_upload_change, names='value')
    return upload_widget

def process_data(data):
    X = data.iloc[:, :-1].values  # Các đặc trưng
    y = data.iloc[:, -1].values  # Nhãn

    # Mã hóa nhãn
    label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(y)

    # Chia dữ liệu thành tập huấn luyện và kiểm tra
    X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.2, random_state=42)
```

```
    # Chuyển đổi dữ liệu sang tensor
    X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
    X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
    y_train_tensor = torch.tensor(y_train, dtype=torch.long)
    y_test_tensor = torch.tensor(y_test, dtype=torch.long)

    # Tạo DataLoader
    train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
    test_dataset = TensorDataset(X_test_tensor, y_test_tensor)

    train_loader = DataLoader(train_dataset, batch_size=10, shuffle=True)
    test_loader = DataLoader(test_dataset, batch_size=10, shuffle=False)

    model = IrisClassifier()

    # Huấn luyện mô hình
    def train_model(num_epochs):
        optimizer = optim.Adam(model.parameters(), lr=0.01)
        criterion = nn.CrossEntropyLoss()
        model.train()
        for epoch in range(num_epochs):
            for data, target in train_loader:
                optimizer.zero_grad()
                output = model(data)
                loss = criterion(output, target)
                loss.backward()
                optimizer.step()
            print(f"Epoch {epoch+1}, Loss: {loss.item()}")

    train_model(100)

    # Đánh giá mô hình
    def evaluate_model():
        model.eval()
        correct = 0
        with torch.no_grad():
            for data, target in test_loader:
                output = model(data)
                _, predicted = torch.max(output.data, 1)
                correct += (predicted == target).sum().item()
        accuracy = 100 * correct / len(test_loader.dataset)
        print(f"Độ chính xác trên tập kiểm tra: {accuracy:.2f}%")

    evaluate_model()

# Khởi động widget để tải lên file
upload_file()
```

**# Trả lời:** Dán kiến trúc mạng ( yêu cầu kiến trúc chứa ít nhất 1 lớp ẩn) và giải thích làm thể nào để phân loại?

**Kiến trúc mạng:**
**Lớp input (4 neurons):** Đầu vào của mạng là các đặc trưng của hoa Iris, bao gồm độ dài và độ rộng của lá đài và cánh hoa (tổng cộng 4 đặc trưng).
**Lớp ẩn (1 hidden layer, 10 neurons):** Mạng có một lớp ẩn với 10 nơ-ron. Các nơ-ron trong lớp ẩn này sẽ học các biểu diễn phức tạp của dữ liệu đầu vào và trích xuất các đặc trưng quan trọng để phân loại.
**Hàm kích hoạt:** Hàm kích hoạt ReLU được sử dụng sau lớp ẩn để tạo ra không gian phi tuyến tính và giúp mạng học được các đặc trưng phức tạp.
**Lớp output (3 neurons):** Lớp đầu ra có 3 nơ-ron tương ứng với 3 loài hoa Iris. Mỗi nơ-ron ở lớp này đại diện cho xác suất của một loài hoa cụ thể.
**Phương pháp phân loại:**
**Huấn luyện mạng nơ-ron**: Dữ liệu huấn luyện gồm các điểm dữ liệu có các đặc trưng của hoa Iris và nhãn tương ứng (loài hoa).
**Chuyển đổi dữ liệu thành tensor**: Dữ liệu đầu vào được chuyển đổi thành tensor và được chia thành các batch để đưa vào mạng nơ-ron.
**Feedforward (lan truyền tiến):** Dữ liệu được đưa qua mạng theo chiều tiến để tính toán đầu ra của mạng.
**Tính toán hàm mất mát (loss):** So sánh đầu ra dự đoán với nhãn thực tế để tính toán độ lỗi của mạng, thông qua hàm CrossEntropyLoss.
**Backpropagation (lan truyền ngược):** Sử dụng giải thuật lan truyền ngược để tính gradient của hàm mất mát theo các tham số của mạng.
**Cập nhật tham số:** Các tham số của mạng được cập nhật theo hướng giảm gradient để giảm thiểu độ lỗi.
**Lặp lại quá trình huấn luyện**: Quá trình từ bước 3 đến bước 6 được lặp lại nhiều lần (trong ví dụ này là 100 epochs) để mạng học được càng nhiều từ dữ liệu huấn luyện.
Đánh giá mô hình: Mô hình được đánh giá bằng cách tính độ chính xác trên tập kiểm tra, để biết mức độ mà mô hình có thể phân loại đúng loài hoa Iris từ dữ liệu mới.
Trong quá trình huấn luyện và đánh giá, mạng nơ-ron cố gắng học và trích xuất các mối quan hệ phức tạp giữa các đặc trưng của hoa Iris để phân loại chúng vào các loài khác nhau dựa trên dữ liệu huấn luyện có sẵn.

b) *(2 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

```
Epoch 28, Loss: 0.31681448221206665
Epoch 29, Loss: 0.1507626175880432
Epoch 30, Loss: 0.2282474786043167
Epoch 31, Loss: 0.25214684009552
Epoch 32, Loss: 0.1990242451429367
Epoch 33, Loss: 0.21847763657569885
Epoch 34, Loss: 0.22489699721336365
Epoch 35, Loss: 0.21764333546161652
Epoch 36, Loss: 0.2157040536403656
Epoch 37, Loss: 0.07897452265024185
Epoch 38, Loss: 0.05917034670710564
Epoch 39, Loss: 0.11455176770687103
Epoch 40, Loss: 0.2155505120754242
Epoch 41, Loss: 0.09499235451221466
Epoch 42, Loss: 0.4049762785434723
Epoch 43, Loss: 0.15877732634544373
Epoch 44, Loss: 0.09527299553155899
Epoch 45, Loss: 0.10489039123058319
Epoch 46, Loss: 0.1786135882139206
```

```
Epoch 47, Loss: 0.23176345229148865
Epoch 48, Loss: 0.13649539649486542
Epoch 49, Loss: 0.1268441081047058
Epoch 50, Loss: 0.15134048461914062
Epoch 51, Loss: 0.05968227982521057
Epoch 52, Loss: 0.1350713074207306
Epoch 53, Loss: 0.17797328531742096
Epoch 54, Loss: 0.0848010927438736
Epoch 55, Loss: 0.11930517852306366
Epoch 56, Loss: 0.15875141322612762
Epoch 57, Loss: 0.16850003600120544
Epoch 58, Loss: 0.05963335558772087
Epoch 59, Loss: 0.0801016315817833
Epoch 60, Loss: 0.08557742834091187
Epoch 61, Loss: 0.09725917875766754
Epoch 62, Loss: 0.15598145127296448
Epoch 63, Loss: 0.2224723845720291
Epoch 64, Loss: 0.09008383005857468
Epoch 65, Loss: 0.17811259627342224
Epoch 66, Loss: 0.27150923013687134
Epoch 67, Loss: 0.27025097608566284
Epoch 68, Loss: 0.07322807610034943
Epoch 69, Loss: 0.22830970585346222
Epoch 70, Loss: 0.11766520887613297
Epoch 71, Loss: 0.03659989312291145
Epoch 72, Loss: 0.13918724656105042
Epoch 73, Loss: 0.11944518983364105
Epoch 74, Loss: 0.07868851721286774
Epoch 75, Loss: 0.14618249237537384
Epoch 76, Loss: 0.11578875780105591
Epoch 77, Loss: 0.07672325521707535
Epoch 78, Loss: 0.1323893964290619
Epoch 79, Loss: 0.04051869362592697
Epoch 80, Loss: 0.1878976970911026
Epoch 81, Loss: 0.10723626613616943
Epoch 82, Loss: 0.20323173701763153
Epoch 83, Loss: 0.028999026864767075
Epoch 84, Loss: 0.14281979203224182
Epoch 85, Loss: 0.07792486995458603
Epoch 86, Loss: 0.07815499603748322
Epoch 87, Loss: 0.23812666535377502
Epoch 88, Loss: 0.18149688839912415
Epoch 89, Loss: 0.14532794058322906
Epoch 90, Loss: 0.19521529972553253
Epoch 91, Loss: 0.12859687209129333
Epoch 92, Loss: 0.031310223042964935
Epoch 93, Loss: 0.09474093466997147
Epoch 94, Loss: 0.03820539265871048
Epoch 95, Loss: 0.2571340799331665
Epoch 96, Loss: 0.022570032626390457
Epoch 97, Loss: 0.10133077204227448
Epoch 98, Loss: 0.01595381833612919
Epoch 99, Loss: 0.09144783020019531
Epoch 100, Loss: 0.17703405022621155
Độ chính xác trên tập kiểm tra: 93.3%
```

**# Trả lời:** Dán kết quả nhãn ứng với 30 mẫu dữ liệu

```
Dự đoán nhãn: [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
2 2 2]
```

***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 chứa thuật toán *k*-means để phân cụm

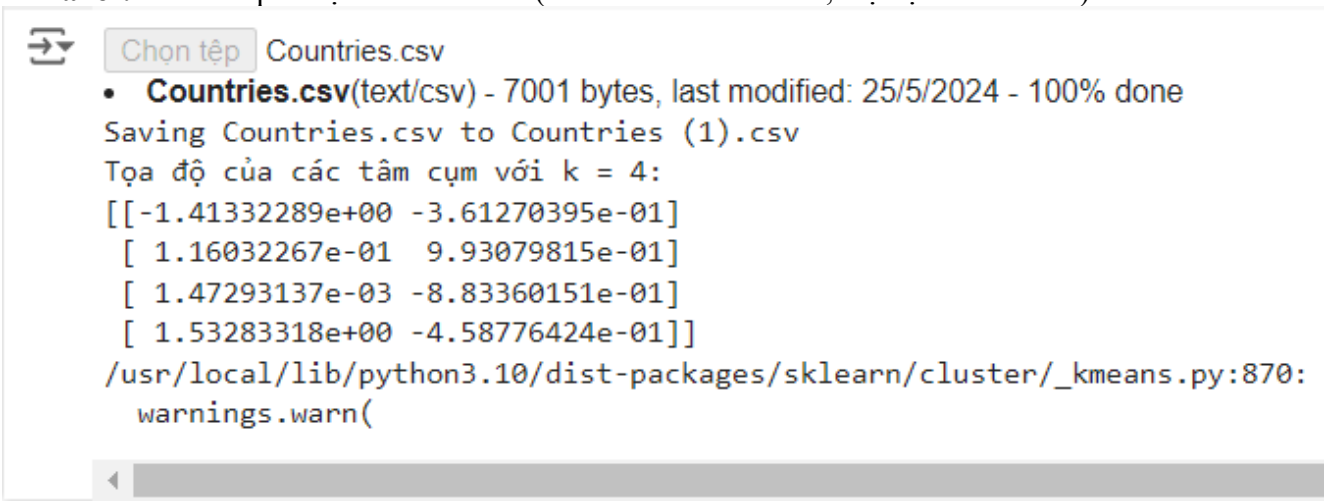
# **Trả lời:** Dán code về hàm

```
# a. Xây dựng hàm chứa thuật toán K-Means để phân cụm
def k_means_clustering(data, n_clusters):
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(data)

    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans.fit(scaled_data)

    return kmeans.labels_, kmeans.cluster_centers_
```

# **Trả lời:** Dán kết quả thực thi với **k = 4** ( chỉ cần đưa ra số tâm, toạ độ của các tâm)



```
Chọn tệp Countries.csv
• Countries.csv(text/csv) - 7001 bytes, last modified: 25/5/2024 - 100% done
Saving Countries.csv to Countries (1).csv
Tọa độ của các tâm cụm với k = 4:
[[-1.41332289e+00 -3.61270395e-01]
 [ 1.16032267e-01  9.93079815e-01]
 [ 1.47293137e-03 -8.83360151e-01]
 [ 1.53283318e+00 -4.58776424e-01]]
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
  warnings.warn(
```

b) *(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

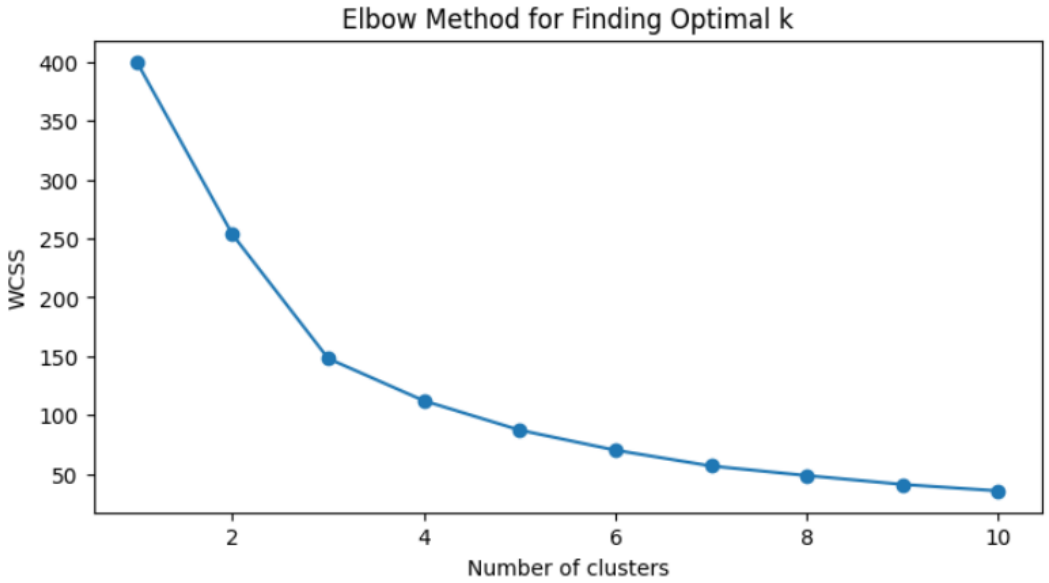
```
# b. Xây dựng hàm để khảo sát việc lựa chọn k
def find_optimal_k(data, max_k=10):
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(data)

    wcss = []  # Within-cluster sums of squares
    for k in range(1, max_k+1):
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(scaled_data)
        wcss.append(kmeans.inertia_)

    plt.figure(figsize=(8, 4))
    plt.plot(range(1, max_k+1), wcss, marker='o')
    plt.title('Elbow Method for Finding Optimal k')
    plt.xlabel('Number of clusters')
    plt.ylabel('WCSS')
    plt.show()
```

Giải thích cách lựa chọn k:
Chuẩn hóa dữ liệu: Chuyển đổi dữ liệu thành dạng chuẩn hóa để đảm bảo rằng các đặc tính có đơn vị đo khác nhau sẽ không ảnh hưởng đến quá trình phân cụm.
Tính toán WCSS cho các giá trị k từ 1 đến max_k: WCSS là tổng bình phương khoảng cách từ

các điểm dữ liệu đến tâm cụm của chúng.
Vẽ biểu đồ Elbow: Biểu đồ hiển thị giá trị WCSS cho mỗi giá trị k. Điểm gấp khúc trên biểu đồ (elbow point) là nơi mà WCSS giảm dần chậm hơn khi số cụm tăng lên, điều này chỉ ra số cụm tối ưu.
**# Trả lời:** Dán kết quả thi với k vừa khảo sát được ( chỉ cần đưa ra số tâm, toạ độ của các tâm)




```
Tọa độ của các tâm cụm với k = 4:
[[-1.41332289e+00 -3.61270395e-01]
 [ 1.16032267e-01  9.93079815e-01]
 [ 1.47293137e-03 -8.83360151e-01]
 [ 1.53283318e+00 -4.58776424e-01]]
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n
  warnings.warn(
```

c) *(1 điểm)* Xây dựng chương trình hoàn thiện và thực thi với dữ liệu đã cho

**# Trả lời**: Dán code hoàn thiện

```
from google.colab import files
uploaded = files.upload()


import pandas as pd

file_path = 'Countries.csv'
data = pd.read_csv(file_path)
print(data.head())

# Tiếp tục với các bước phân cụm K-Means
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# a. Xây dựng hàm chứa thuật toán K-Means để phân cụm
def k_means_clustering(data, n_clusters):
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(data)
```

```
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans.fit(scaled_data)

    return kmeans.labels_, kmeans.cluster_centers_

# b. Xây dựng hàm để khảo sát việc lựa chọn k
def find_optimal_k(data, max_k=10):
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(data)

    wcss = []  # Within-cluster sums of squares
    for k in range(1, max_k+1):
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(scaled_data)
        wcss.append(kmeans.inertia_)

    plt.figure(figsize=(8, 4))
    plt.plot(range(1, max_k+1), wcss, marker='o')
    plt.title('Elbow Method for Finding Optimal k')
    plt.xlabel('Number of clusters')
    plt.ylabel('WCSS')
    plt.show()

# c. Xây dựng chương trình hoàn thiện và thực thi với dữ liệu đã cho
def main():
    # Đọc dữ liệu
    data = pd.read_csv(file_path)

    # Lựa chọn cột dữ liệu để phân cụm
    features = data[['Longitude', 'Latitude']]

    # Khảo sát số lượng cụm k
    find_optimal_k(features)

    # Số cụm k tối ưu dựa trên phương pháp Elbow
    optimal_k = 4  # Giả sử chúng ta chọn 4 cụm từ biểu đồ Elbow

    # Thực hiện phân cụm với k tối ưu
    labels, cluster_centers = k_means_clustering(features, optimal_k)

    # Thêm nhãn cụm vào dữ liệu ban đầu
    data['Cluster'] = labels

    # Xuất dữ liệu đã phân cụm
    output_path = 'Countries_with_clusters.csv'
    data.to_csv(output_path, index=False)
    print(f'Phân cụm hoàn tất. Kết quả đã được lưu vào {output_path}')

# Thực thi chương trình
main()
```

Đà Nẵng, ngày 14  tháng 05 năm 2023

**GIẢNG VIÊN BIÊN SOẠN ĐỀ THI**

**TRƯỞNG BỘ MÔN**
(đã duyệt)