import csv

import sys

from util import Node, StackFrontier, QueueFrontier

# Maps names to a set of corresponding person\_ids

names = {}

# Maps person\_ids to a dictionary of: name, birth, movies (a set of movie\_ids)

people = {}

# Maps movie\_ids to a dictionary of: title, year, stars (a set of person\_ids)

movies = {}

def load\_data(directory):

"""

Load data from CSV files into memory.

"""

# Load people

with open(f"{directory}/people.csv", encoding="utf-8") as f:

reader = csv.DictReader(f)

for row in reader:

people[row["id"]] = {

"name": row["name"],

"birth": row["birth"],

"movies": set()

}

if row["name"].lower() not in names:

names[row["name"].lower()] = {row["id"]}

else:

names[row["name"].lower()].add(row["id"])

# Load movies

with open(f"{directory}/movies.csv", encoding="utf-8") as f:

reader = csv.DictReader(f)

for row in reader:

movies[row["id"]] = {

"title": row["title"],

"year": row["year"],

"stars": set()

}

# Load stars

with open(f"{directory}/stars.csv", encoding="utf-8") as f:

reader = csv.DictReader(f)

for row in reader:

try:

people[row["person\_id"]]["movies"].add(row["movie\_id"])

movies[row["movie\_id"]]["stars"].add(row["person\_id"])

except KeyError:

pass

def main():

if len(sys.argv) > 2:

sys.exit("Usage: python degrees.py [directory]")

directory = sys.argv[1] if len(sys.argv) == 2 else "large"

# Load data from files into memory

print("Loading data...")

load\_data(directory)

print("Data loaded.")

source = person\_id\_for\_name(input("Name: "))

if source is None:

sys.exit("Person not found.")

target = person\_id\_for\_name(input("Name: "))

if target is None:

sys.exit("Person not found.")

path = shortest\_path(source, target)

if path is None:

print("Not connected.")

else:

degrees = len(path)

print(f"{degrees} degrees of separation.")

path = [(2, 3), (4, 5)] + path

for i in range(degrees):

person3 = people[path[i][1]]["name"]

person5 = people[path[i + 1][1]]["name"]

movie = movies[path[i + 1][0]]["title"]

print(f"{i + 1}: {person3} and {person5} starred in {movie}")

def shortest\_path(source, target):

"""

Returns the shortest list of (movie\_id, person\_id) pairs

that connect the source to the target.

If no possible path, returns None.

"""

# TODO

raise NotImplementedError

def person\_id\_for\_name(name):

"""

Returns the IMDB id for a person's name,

resolving ambiguities as needed.

"""

person\_ids = list(names.get(name.lower(), set()))

if len(person\_ids) == 0:

return None

elif len(person\_ids) > 1:

print(f"Which '{name}'?")

for person\_id in person\_ids:

person = people[person\_id]

name = person["name"]

birth = person["birth"]

print(f"ID: {person\_id}, Name: {name}, Birth: {birth}")

try:

person\_id = input("Intended Person ID: ")

if person\_id in person\_ids:

return person\_id

except ValueError:

pass

return None

else:

return person\_ids[0]

def neighbors\_for\_person(person\_id):

"""

Returns (movie\_id, person\_id) pairs for people

who starred with a given person.

"""

movie\_ids = people[person\_id]["movies"]

neighbors = set()

for movie\_id in movie\_ids:

for person\_id in movies[movie\_id]["stars"]:

neighbors.add((movie\_id, person\_id))

return neighbors

if \_\_name\_\_ == "\_\_main\_\_":

main()