Borrow Movies (20 pts)

Background

A town has **two libraries**, each offering several **movie CDs** for borrowing.
The list of available movie titles in each library is stored in two text files:

- `l1.txt` — contains the list of movies in **Library 1**
- `l2.txt` — contains the list of movies in **Library 2**

Each line in the text file represents the title of one movie.
For example:

```
Avatar
Inception
Titanic
```

Write a Python program that performs the following tasks using **set operations** and **file reading**.

1. **`def movies_in_library(file_path):`**
   - **Input:** a string `file_path` — the path to a text file (e.g., `"l1.txt"` or `"l2.txt"`).
   - **Output:** a **set** containing all movie titles in that library.
   - **Hint:** strip newline characters when reading each line.

   Example:

   ```python
   print('Output:', movies_in_library("l1.txt"))
   # Output: {'Avatar', 'Inception', 'Titanic'}
   ```

2. **`def movies_in_both_l1_and_l2():`**
   - **Output:** a **set** of movies available in **both libraries**.
   - Use the results of `movies_in_library("l1.txt")` and `movies_in_library("l2.txt")` .

   Example:

   ```python
   # Suppose l1.txt = {'Avatar', 'Inception', 'Titanic'}
   # and l2.txt = {'Inception', 'Interstellar'}
   print('Output:', movies_in_both_l1_and_l2())
   # Output: {'Inception'}
   ```

3. `def movies_in_only_one_library():`

- **Output:** a **set** of movies that are available in **exactly one** library (i.e., not shared between both).

  Example:

  ```
  # Suppose l1.txt = {'Avatar', 'Inception', 'Titanic'}
  # and l2.txt = {'Inception', 'Interstellar'}
  print('Output:', movies_in_only_one_library())
  # Output: {'Avatar', 'Titanic', 'Interstellar'}
  ```

4. `def all_available_movies_in_town():`
   - **Output:** a **set** of **all unique movies** available in either library.
   - (This is the **union** of the two libraries' movie sets.)

   Example:

   ```
   # Suppose l1.txt = {'Avatar', 'Inception', 'Titanic'}
   # and l2.txt = {'Inception', 'Interstellar'}
   print('Output:', all_available_movies_in_town())
   # Output: {'Avatar', 'Inception', 'Titanic', 'Interstellar'}
   ```

Warehouse and Storage Management System (30 pts)

You are tasked with building a **storage management system** that tracks multiple **warehouses**, their **inventory**, and helps locate the **nearest warehouse** that has a given item in stock.

The system will consist of two main classes:

- `Warehouse` — represents a single warehouse and manages its inventory.
- `StorageManagementSystem` — manages multiple warehouses and coordinates queries across them.

`Warehouse` Class (20 pts)

This class represents an individual warehouse, storing its **location** and **inventory**.

Attributes:

- `name` — warehouse name (e.g., `"Warehouse A"`)
- `location` — a tuple `(x, y)` representing coordinates on a map
- `inventory` — a dictionary mapping item names to quantities, e.g.:

  ```
  {"apple": 120, "banana": 80}
  ```

Methods:

- `__init__(self, name, location, file_path=None)`
  Initializes a warehouse with a name, a location tuple, and optionally loads inventory data from a text file.
  If `file_path` is provided, load the inventory from the file.
  Each line in the file will contain an item and quantity, separated by a comma:

  ```
  apple,120
  banana,80
  orange,50
  ```

  If `file_path` is not provided, initialize the inventory as an empty dictionary.
- `def add_item(self, item, quantity):`
  Adds the specified quantity of an item to the warehouse inventory.
  If the item doesn't exist, add it to the dictionary.
- `def remove_item(self, item, quantity):`
  Removes a quantity of an item from the warehouse inventory.
  - If the item does not exist or there isn't enough stock, raise:

    ```
    raise ValueError("Not enough stock or item does not exist.")
    ```

- `def get_quantity(self, item) -> int:`
  Returns the current quantity of a specific item.
  Returns `0` if the item is not in stock.

`StorageManagementSystem` Class (10 pts)

This class manages multiple warehouses and provides a centralized interface to query and manage them.

Attributes:

- `warehouses` — a list of `Warehouse` objects.

Methods:

- `def add_warehouse(self, warehouse: Warehouse):`
  Adds a new warehouse to the system.
- `def nearest_warehouse_with_item(self, item, current_location):`
  Finds the **nearest warehouse** (by Euclidean distance) that has the given item in stock.

    - If no warehouse has the item, return `None` .

  Example:

```
system.nearest_warehouse_with_item("apple", (10, 20))
# Output: ("Warehouse B", 15.3)
# meaning Warehouse B is 15.3 units away and has the item
```

Example File Format (`w1.txt`)

```
apple,120
banana,80
orange,50
```

Example Usage

```
# Initialize warehouses
w1 = Warehouse("Warehouse A", (0, 0), file_path="w1.txt")
w2 = Warehouse("Warehouse B", (10, 15))
w2.add_item("apple", 30)
w2.add_item("mango", 10)

# Create system
system = StorageManagementSystem()
system.add_warehouse(w1)
system.add_warehouse(w2)

# Query
print(system.nearest_warehouse_with_item("apple", (5, 5)))
# ('Warehouse B', 11.18)
```

Hints

- Use the Euclidean distance formula:

$$\text{distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$