DS2030 Data Structures and Algorithms for Data Science
Lab 1

August 12th, 2025

Lab Instructions

- Create a folder named **“DS2030_<RollNo.>”** (all letters in capital) in **“home”** directory. Eg- **DS2030_142402022**
- Name the script files in the given format **“<your_roll_no>_<Name>_Lab1.py”**
- Make sure the **folder, files, classes, functions and attributes** are named as instructed in the lab sheet.
- We will not be able to evaluate your work if the folder is not properly named or is not located in the home directory.
- Make sure to save your progress before leaving the lab.
- Do not shut down the system after completing the exam.
- You are not allowed to share code with your classmates nor allowed to use code from the internet. If we find that you have copied code from your classmate or from the Internet, you will get a straight fail grade in the course.

1 Task 1: Define the Compound Class (1 Point)

Defining the `Compound` class that encapsulates the attributes:

- `name`
- `formula`
- `source_lab`
- `ph_value` (default 7.0)
- `reactivity_score` (default 0.0)

Methods:

- `__init__`: Initialize the attributes
- `get_ph_classification()`: Returns:
  - “Acidic” if `ph_value` <7
  - “Neutral” if $7 \leq$ `ph_value` ≤ 7.5
  - “Basic” if `ph_value` >7.5
- `__str__()`: Formatted string representing the compound.

```
class Compound:
    def __init__(self, name, formula, source_lab, ph_value=7.0, reactivity_score=0.0):
        """
        Initialize the compound with the following attributes:
        - name: The compound's name.
        - formula: Chemical formula.
        - source_lab: The lab where the compound originated.
        - ph_value: pH value of the compound (default 7.0).
        - reactivity_score: Reactivity score (default 0.0).
        """
        # TO DO

    def get_ph_classification(self):
        """
        Return pH classification: Acidic, Neutral, or Basic.
        """
        # TO DO

    def __str__(self):
        """
        Return a formatted string representing the compound.
        """
        # TO DO
```

2 Task 2: Implement the Doubly Linked List (4 Points)

2.1 Node Class

Each node holds a reference to a `Compound` object and pointers to `prev` and `next` nodes.

```
class Node:
    def __init__(self, compound):
        """
        Initialize the node with a compound object.
        - compound: The compound stored in this node.
        - prev: Reference to the previous node (initialized to None).
        - next: Reference to the next node (initialized to None).
        """
        # TO DO
```

2.2 LinkedList Class

The ‘LinkedList‘ class manages the collection of nodes. You will implement methods to add nodes, sort them, and retrieve the top reactive compounds.
Methods:

- `append(compound)`: Add compound to the end of the list.
- `sort_by_reactivity()`: Sort the list by `reactivity_score` in descending order. (Use insertion sort or equivalent)
- `get_top_reactive(N)`: Returns a list of the top reactive compounds.

```
class LinkedList:
    def __init__(self):
        """
        Initialize an empty linked list.
        - head: Points to the first node in the list.
        - tail: Points to the last node in the list.
        """
        # TO DO

    def append(self, compound):
        """
        Add a new compound to the end of the list.
        - If the list is empty, set head and tail to the new node.
        - Otherwise, add the new node to the end.
        """
        # TO DO

    def sort_by_reactivity(self):
        """
        Sort the list by reactivity_score in descending order.
        - If the list is empty or has one node, do nothing.
        - Otherwise, perform insertion sort.
        """
        # TO DO

    def get_top_reactive(self, N):
        """
        Retrieve the top N reactive compounds from the list.
        - N: Number of top reactive compounds to retrieve.
        - Returns a Python list of top N reactive compounds.
        """
        # TO DO
```

3 Task 3: Simulate Batches (4 Points)

Implement a function to simulate batches, where compounds are divided into multiple batches of up to `batch_size` compounds, sorted by `reactivity_score`, and the top N reactive compounds from each batch will advance to the next round.

- Divide compounds into batches of size `batch_size`.
- Use the 'LinkedList' class to sort compounds in each batch.
- Select the top 'N' reactive compounds to advance to the next round.
- Recursively simulate batches until there is only one batch remains.

```
def simulate_batches(compounds, batch_size, N):
    """
    Simulate batches by dividing compounds into multiple batches,
    sorting each batch by reactivity_score in descending order, and recursively
    selecting top reactive compounds until one batch remains.

    - compounds: List of Compound objects.
    - batch_size: Number of compounds per batch.
    - N: Number of top reactive compounds to select from each batch.

    - Identify the base case - when the number of compounds is <= batch_size.
    - Each batch is stored as a linked list.
    """
    # base case
    # number of compounds is <= batch_size
    # - create a linkedlist of the compounds in the final batch
    # - sort the linkedlist by reactivity_score
    # - select the top 3 reactive compunds and return them as a Python List
    # TO DO

    # Divide compounds into batches in some random manner.
    # TO DO

    # initialize a Python list that will store the compounds for the next round.
    next_round_compounds = []
    # Loop through each batch while
    # - create a linkedlist of the compounds in the batch
    # - sort the linkedlist by reactivity_score
    # - select the top N reactive compounds from each batch
    # - append the N reactive compounds to the list containing next round compounds.
    # TO DO


    # Call the simulate batches function on the set of compounds participating in the
    next round.

    # TO DO
    # TO DO
```

Testing (1 Point)

Finally, you have to write test cases to ensure the correctness of your implementation. Test cases should cover the following:

- Correct insertion and sorting in the linked list.
- Correct selection of the top reactive compounds from a batch.
- Edge cases, such as when the number of compounds is less than or equal to `batch_size`.

```
def test():
    # Create a list of contestants
    compounds = [
        Compound("Hydrochloric Acid", "HCl", "Lab Alpha", ph_value=1.2, reactivity_score
    =9.1),
        Compound("Sodium Hydroxide", "NaOH", "Quantum Labs", ph_value=13.0,
    reactivity_score=8.7),
        Compound("Water", "H2O", "Lab Alpha", ph_value=7.0, reactivity_score=0.1),
        Compound("Acetic Acid", "CH3COOH", "Nova Research Center", ph_value=2.8,
    reactivity_score=6.5),
        Compound("Ammonia", "NH3", "Lab Alpha", ph_value=11.6, reactivity_score=7.9),
        Compound("Carbonic Acid", "H2CO3", "Pioneer Chemistry Lab", ph_value=5.5,
    reactivity_score=5.2),
        Compound("Sodium Chloride", "NaCl", "Quantum Labs", ph_value=7.1,
    reactivity_score=0.5),
        Compound("Calcium Hydroxide", "Ca(OH)2", "Nova Research Center", ph_value=12.4,
    reactivity_score=7.8)]

    batch_size = 4
    N = 2

    top_3 = simulate_batches(compounds, batch_size, N)

    # Print the top 3 reactive compounds details along with the pH classification
    # TO DO

# Run the test case
test()
```

Sample Output

```
Top 3 Reactive Compounds :
Acidic: Hydrochloric Acid (HCl) from Lab Alpha, pH: 1.20, Reactivity: 9.10
Basic: Sodium Hydroxide (NaOH) from Quantum Labs, pH: 13.00, Reactivity: 8.70
Basic: Ammonia (NH3) from Lab Alpha, pH: 11.60, Reactivity: 7.90
```