DS2030 Data Structures and Algorithms for Data Science
Week 1 - Practice Problem

`August 12th, 2025`

Objective

The objective of this practice set is to help you become familiar with the concepts of doubly linked lists and recursion in Python. You will work through exercises that involve implementing a doubly linked list and using recursion to simulate a contest in which contestants compete in heats. The goal is to reinforce your understanding of these data structures and techniques through hands-on problem solving.

Background Information

A linked list is a linear data structure where elements are not stored in contiguous memory locations. Instead, each element, called a node, contains a reference to the next node in the sequence. In a doubly linked list, each node contains a reference to both the previous and next nodes, allowing for traversal in both directions.
This lab will involve:

- Implementing a 'Contestant' class to hold contestant data.
- Implementing a doubly linked list using 'Node' and 'LinkedList' classes.
- Writing functions to manipulate the linked list, including sorting contestants by their performance time.
- Simulating a contest where contestants compete in heats, with the top performers advancing to subsequent rounds, culminating in a final round to determine the top three winners.

Task 1: Define the Contestant Class

Begin by defining a 'Contestant' class. This class will encapsulate the contestant's attributes such as name, chest number, country, and time.

Instructions

- Define the '__init__' method to initialize the contestant's attributes.
- Implement the '__str__' method to return a formatted string representing the contestant.

Starter Code

```python
class Contestant:
    def __init__(self, name, chest_number, country, time=0):
        """
        Initialize the contestant with the following attributes:
        - name: The contestant's name.
        - chest_number: The contestant's chest number.
        - country: The country the contestant represents.
        - time: The time taken by the contestant in the heat (default is 0).
        """
        # TO DO

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

Task 2: Implement the Doubly Linked List

Now, implement a doubly linked list by defining ‘Node‘ and ‘LinkedList‘ classes.

Node Class

The ‘Node‘ class represents each element in the linked list. Each node should hold a reference to a ‘Contestant‘ object and have pointers to the previous and next nodes.

Starter Code

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

LinkedList Class

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

Instructions

- Implement the ‘append‘ method to add a new contestant to the list. The contestant is added to the end of the list.
- Implement the ‘sort_by_time‘ method to sort the list based on the contestant’s time.
- Implement the ‘get_top_performers‘ method to retrieve the top M performers from the list.

Starter Code

```
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.
        """
        self.head = None
        self.tail = None

    def append(self, contestant):
        """
        Add a new contestant to the end of the list.
        - contestant: The contestant object to add to the list.
        - If the list is empty, set the head and tail to the new node.
        - Otherwise, add the new node to the end of the list.
        """
```

```python
        # TO DO

    def sort_by_time(self):
        """
        Sort the list based on the time attribute of the contestants.
        - If the list is empty do nothing.
        - Otherwise perform insertion sort on the list
        """
        # TO DO

    def get_top_performers(self, M):
        """
        Retrieve the top M performers from the list.
        - M: The number of top performers to retrieve. The function returns a Python list of top performers.
        """
        # TO DO
```

Task 3: Simulate Heats

Next, you will implement a function to simulate heats. Contestants will be divided into multiple heats, with each heat containing up to 'N' contestants. Assign random times to each contestant, and the top 'M' performers will advance to the next round.

Instructions

- Divide contestants into heats of size 'N'.
- Assign random times to each contestant.
- Use the 'LinkedList' class to sort contestants in each heat.
- Select the top 'M' performers to advance to the next round.
- Recursively simulate heats until there is only one heat remaining.

Starter Code

```python
import random

def simulate_heats(contestants, N, M):
    """
    Simulate heats by dividing contestants into multiple heats, assigning random times,
    and recursively selecting the top performers until only one heat remains.
    - contestants: Python List of Contestant objects.
    - N: Number of contestants per heat.
    - M: Number of top performers to select from each heat.
    - Identify the base case - when the number of contestants is <= N.
    - Each heat is stored as a linked list.
    """
    # base case
    # number of contestants is <= N
    # - create a linkedlist of the contestants in the final heat
    # - sort the linkedlist by time
    # - select the top 3 performers and return them as a Python List
    # TO DO

    # Divide contestants into heats in some random manner.
    # TO DO

    # initialize a Python list that will store the contestants for the next round.
    next_round_contestants = []

    # Loop through each heat while
    # - assigning random times to each contestant in the heat
    # - create a linkedlist of the contestants in the heat
    # - sort the linkedlist by time
    # - select the top M performers from each heat
    # - append the top M performers to the list containing next round contestants.
    # TO DO


    # Call the simulate heats function on the set of contestants participating in the
    next level of heats.

    # TO DO
```

Task 4: Testing

Finally, you will 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 top performers from a heat.
- Edge cases, such as when the number of contestants is less than or equal to 'N'.

Sample Test Case

```
def test():
    # Create a list of contestants
    contestants = [
        Contestant("Contestant A", 1, "Country A"),
        Contestant("Contestant B", 2, "Country B"),
        Contestant("Contestant C", 3, "Country C"),
        Contestant("Contestant D", 4, "Country D"),
        Contestant("Contestant E", 5, "Country E"),
        Contestant("Contestant F", 6, "Country F"),
        Contestant("Contestant G", 7, "Country G"),
        Contestant("Contestant H", 8, "Country H")
    ]

    N = 4  # Number of contestants per heat
    M = 2  # Number of top performers to select from each heat

    # Run the final selection process
    winners = simulate_heats(contestants, N, M)
    print("Final Selection:")
    medals = ["Gold", "Silver", "Bronze"]
    for i, winner in enumerate(winners):
        print(f"{medals[i]}: {winner}")

# Run the test case
test()
```

1 References

1. M Goodrich, R Tamassia, and M. Goldwasser, "Data Structures and Algorithms in Python", 1st edition, Wiley, 2013.