Assignment 3

Problem 1 - Extended Features for the Single-Linked List

In class `SingleLL` of file SingleLL.py, implement the three following methods:

1. `eradicate(self, value)`

   Removes every node that contains the given value from the calling object's linked list.

2. `reverse(self)`

   Reverses the order of the nodes in the calling object's linked list.

3. `sublist(self, otherlist)`

   Checks whether the argument `otherlist` is a sublist of the calling object's linked list. In other words, all of the values in `otherlist` must occur in the linked list. Duplicates count: if a value `v` appears X times in `otherlist`, then `v` must also appear X times in the linked list for `otherlist` to be a sublist.

Hints:

- ➢ Brainstorm on paper first. Drawing how references change helps a lot when you code with linked lists.
- ➢ Watch out for corner cases!
- ➢ There is no time complexity requirement for this assignment.

Problem 2 - Extended Features for the Double-Linked List

In class `DoubleLL` of file DoubeLL.py, implement method `remove_intersection(self, otherlist)`

`remove_intersection(self, otherlist)` removes every occurence of the values in `otherlist` from the calling object's list. This includes duplicates: if a value `v` appears at least once in `otherlist` and also appears X times in the linked list, then all X occurrences of `v` must be removed from the linked list.

Problem 3 - Summer Sort

Design a new sorting algorithm, called the "*Summer Sort*", that can use no other operation than the two following ones:

`findMax(array, a, b)` returns the index of the largest value in the array between index positions `a` and `b`

`reverse(array, a, b)` reverses all the values in the array from index position `a` to index position `b`

Both of these operations are already available in the canvas code file summer_sort.py. Implement your algorithm there, and answer the two following questions (15pts):

1. What is the best-case runtime complexity of your *Summer Sort*?
2. What is the worst-case runtime complexity of your *Summer Sort*?

Problem 4 - In-Place Sorting of Sequences of Two Values

Radix Sorting can break the N.log(N) barrier in terms of runtime complexity. It assumes that the number of possible values is limited. However, it requires extra memory.

In this problem, you must design an in-place sorting algorithm, called the "*EitherOr*", that sorts random sequences of only two values a and b.

Its runtime complexity must be in O(N).

Implement your algorithm in file either_or_sort.py

Submission format

The files below constitute your coding canvas for this assignment.

1. SingleLL.py
2. DoubeLL.py
3. summer_sort.py
4. either_or_sort.py

You need to complete them, and then submit them directly (do not put them in a directory or zip) to gradescope:

https://www.gradescope.com/courses/399287/assignments/2080895