Solution Review: Union & Intersection of Linked Lists

This review provides a detailed analysis of the different ways to solve the Union and Intersection of Linked Lists challenge.

We'll cover the following

Solution: Union

main.py

LinkedList.py

Node.py

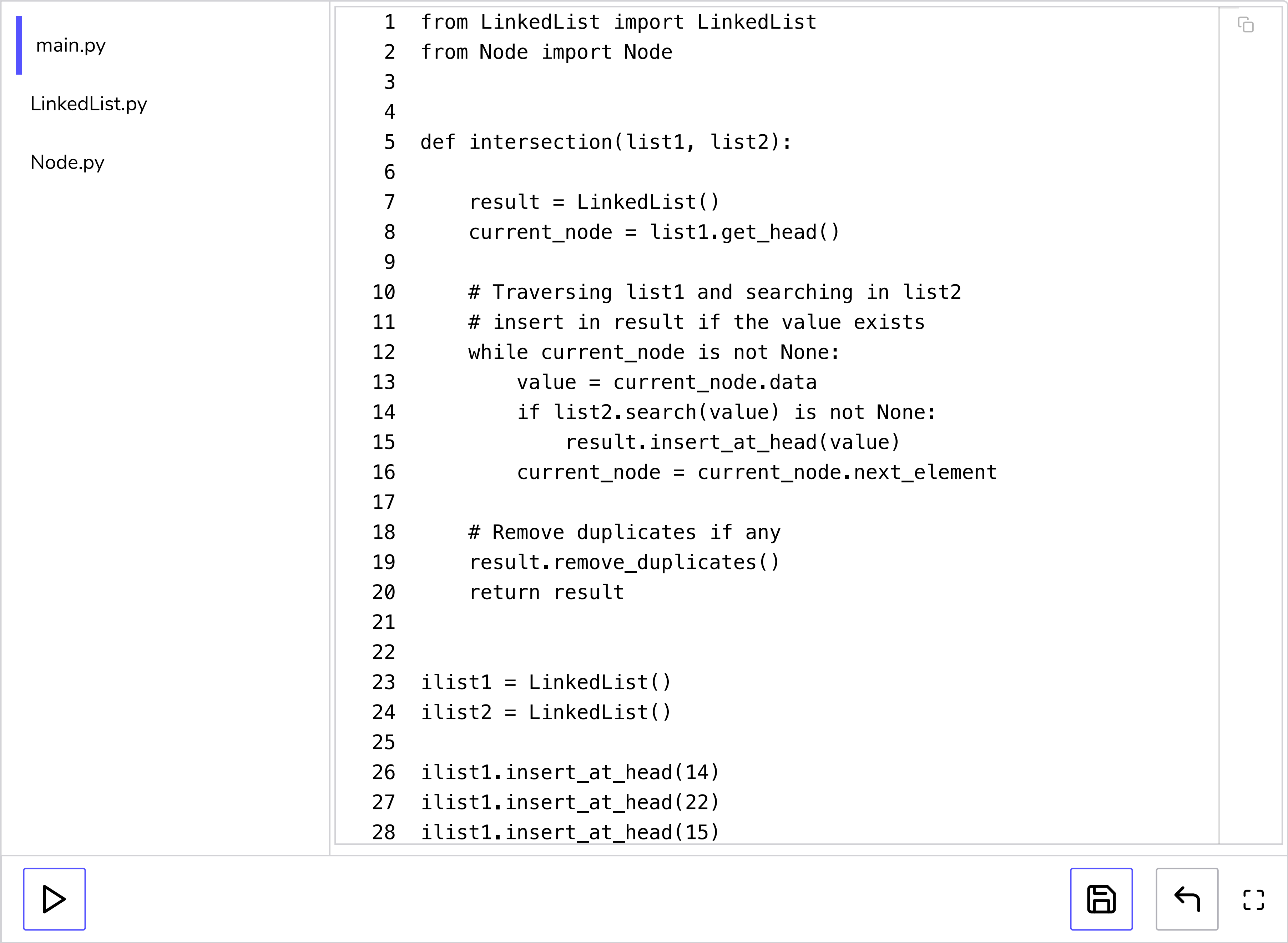
```python
from LinkedList import LinkedList
from Node import Node


def union(list1, list2):
    # Return other List if one of them is empty
    if (list1.is_empty()):
        return list2
    elif (list2.is_empty()):
        return list1

    start = list1.get_head()

    # Traverse the first list till the tail
    while start.next_element:
        start = start.next_element

    # Link last element of first list to the first element of second
    start.next_element = list2.get_head()
    list1.remove_duplicates()
    return list1


ulist1 = LinkedList()
ulist2 = LinkedList()
ulist1.insert_at_head(8)
ulist1.insert_at_head(22)
ulist1.insert_at_head(15)
```

Nothing too tricky going on here. We traverse to the tail of the first list and link it to the first node of the second list. All we have to do now is remove duplicates from the combined list.

Time Complexity

If we did not have to care for duplicates, The runtime complexity of this algorithm would be *O(m)* where **m** is the size of the first list. However, because of duplicates, we need to traverse the whole union list. This increases the time complexity to $O(l)^2$ where l = m + n. Here, **m** is the size of the first list, and **n** is the size of the second list.

Solution: Intersection



```python
from LinkedList import LinkedList
from Node import Node


def intersection(list1, list2):

    result = LinkedList()
    current_node = list1.get_head()

    # Traversing list1 and searching in list2
    # insert in result if the value exists
    while current_node is not None:
        value = current_node.data
        if list2.search(value) is not None:
            result.insert_at_head(value)
        current_node = current_node.next_element

    # Remove duplicates if any
    result.remove_duplicates()
    return result


ilist1 = LinkedList()
ilist2 = LinkedList()

ilist1.insert_at_head(14)
ilist1.insert_at_head(22)
ilist1.insert_at_head(15)
```

You are already familiar with this approach. We simply `list1` and search for all elements in `list2`. If any of these values are found in `list2`, it is added to the `result` linked list.

Since we insert at head, as shown on **line 25**, insert works in constant time.

Time Complexity

The time complexity will be $max(O(mn), O(min(m,n)^2))$ where **m** is the size of the first list and **n** is the size of the second list.

> **Note:** The solution provided above is not the optimal solution for this problem. We can write a more efficient solution using hashing. We will cover that approach in Hashing Chapter: Challenge 12 (https://www.educative.io/courses/data-structures-in-python-an-interview-refresher/q2G5YoKO4L2)

If you've made it this far, you've become very experienced in the art of linked lists. Just one more challenge to go! See you there.

← Back

Challenge 9: Union & Intersection of Li...

Next →

Challenge 10: Return the Nth node fro...

☑ Mark as Completed

Report an

Ask a Question

Report an Issue (https://discuss.educative.io/tag/solution-review-union-intersection-of-linked-lists__introduction-to-linked-lists__data-structures-for-coding-interviews-in-python)