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

```
1  from LinkedList import LinkedList
2  from Node import Node
3
4
5  def union(list1, list2):
6      # Return other List if one of them is empty
7      if (list1.is_empty()):
8          return list2
9      elif (list2.is_empty()):
10         return list1
11
12     unique_values = set()
13     result = LinkedList()
14
15     start = list1.get_head()
16
17     # Traverse the first list till the tail
18     while start:
19         unique_values.add(start.data)
20         start = start.next_element
21
22     start = list2.get_head()
23
24     # Traverse the second list till the tail
25     while start:
26         unique_values.add(start.data)
27         start = start.next_element
28
```

Output 0.170s

```
22 -> 21 -> 15 -> 14 -> 8 -> 7 -> None
```

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.

Another approach would be to add all unique elements to a set. It would also work in the same time complexity, assuming that hashing is O(1) on average.

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(m + n)* where **m** is the size of the first list and **n** is the size of the second list.

Solution: Intersection

main.py

LinkedList.py

Node.py

```
from LinkedList import LinkedList
from Node import Node


def intersection(list1, list2):

    result = LinkedList()
    visited_nodes = set()  # Keep track of all the visited nod
    current_node = list1.get_head()

    # Traversing list1 and adding all unique nodes into the ha
    while current_node is not None:
        value = current_node.data
        if value not in visited_nodes:
            visited_nodes.add(value)  # Visiting current_node
        current_node = current_node.next_element

    start = list2.get_head()

    # Traversing list 2
    # Nodes which are already present in visited_nodes are add
    while start is not None:
        value = start.data
        if value in visited_nodes:
            result.insert_at_head(start.data)
        start = start.next_element
    result.remove_duplicates()
    return result


ilist1 = LinkedList()
ilist2 = LinkedList()

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

ilist2.insert_at_head(21)
ilist2.insert_at_head(14)
ilist2.insert_at_head(15)

```

```
lst = intersection(ilist1, ilist2)
lst.print_list()

```

```
Output                                                    0.627s

 14 -> 15 -> None
```

You are already familiar with this approach. We simply create a set that contains all the unique elements from `list1`. 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 *O(m + n)* where **m** is the size of the first list and **n** is the size of the second list.

Back

Challenge 12: Union & Intersection of ...

Next

Hashing Quiz: Test your understandin...

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/solution-review-union--intersection-of-linked-lists__introduction-to-hashing__data-structures-for-coding-interviews-in-python)