CS 61B Fall 2024

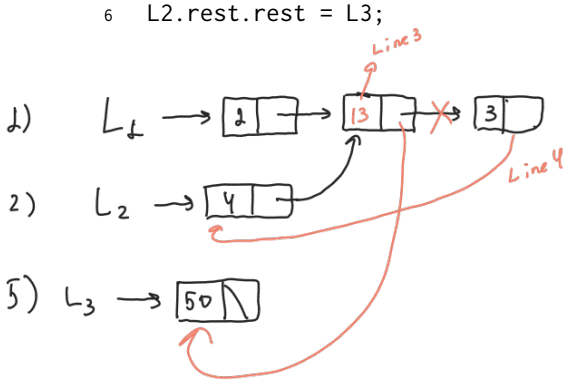
Arrays, Linked Lists

Exam-Level 02: September 9, 2024

1 Boxes and Pointers

Draw a box and pointer diagram to represent the IntLists `L1`, `L2`, and `L3` after each statement.

```
1  IntList L1 = IntList.list(1, 2, 3);
2  IntList L2 = new IntList(4, L1.rest);
3  L2.rest.first = 13;
4  L1.rest.rest.rest = L2;
5  IntList L3 = IntList.list(50);
6  L2.rest.rest = L3;
```



2 Interweave

Implement `interweave`, which takes in an `IntList lst` and an integer `k`, and *destructively* interweaves `lst` into `k IntLists`, stored in an array of `IntLists`. Here, destructively means that instead of creating new `IntList` instances, you should focus on modifying the pointers in the existing `IntList lst`.

Specifically, we require:

- It is the **same** length as the other lists. You may assume the `IntList` is evenly divisible.
- The first element in `lst` is put in the first index of the array of IntLists. The second element is put in the second index. This goes on until the array is traversed, and then we wrap around to put elements in the first index of the array.
- Its ordering is consistent with the ordering of `lst`, i.e. items in earlier in `lst` must **precede** items that are later.

For instance, if `lst` contains the elements `[6, 5, 4, 3, 2, 1]`, and `k = 2`, then the method should return an array of `IntList`, `[6, 4, 2]` at index 0, and `[5, 3, 1]` at index 1.

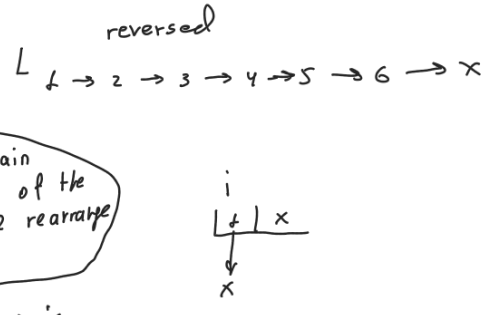
In the beginning, we reversed the `IntList` lst destructively, because it's usually easier to build `IntList` backwards.

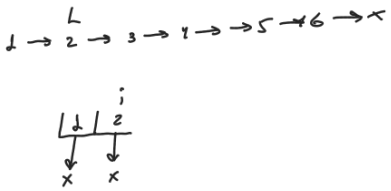
Hint: The elements in the array should track the head of the small `IntList` that they are building.

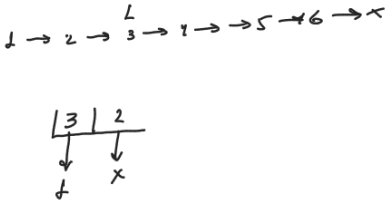
```
public static IntList[] interweave(IntList lst, int k) {
    IntList[] array = new IntList[k];
    int index = k - 1;
    IntList L = reverse(lst); // Assume reverse is implemented correctly

    while (_____L != null_____) {

        IntList prevAtIndex = _____array[index]_____;

        IntList next = _____L.rest_____;

        _____array[index] = L_____;

        _____array[index].rest = prevAtIndex_____;

        L = _____L.rest_____;

        index -= 1;

        if (_____index < 0_____) {

            _____index = k - 1_____;
        }
    }
    return array;
}
```

This is to maintain link to the rest of the IntList, while we rearrange pointers

In the first case, this is null







3 Remove Duplicates

Using the simplified `DLList` class defined on the next page, implement the `removeDuplicates` method.

`removeDuplicates` should remove all duplicate items from the **DLList**. For example, if our initial list is `[8, 4, 4, 6, 4, 10, 12, 12]`, our final list should be `[8, 4, 6, 10, 12]`. You may **not** assume that duplicate items are grouped together, or that the list is sorted!

```
public class DLList {
    Node sentinel;
    public DLList() { // ... }

    public class Node { int item; Node prev; Node next; }

    public void removeDuplicates() {

        Node ref = sentinel.next;
        Node checker;

        while (ref != sentinel) {

            checker = ref.next;

            while (checker != sentinel) {

                if (ref.item == check.item) {

                    Node checkerPrev = checker.prev;
                    Node checkerNext = checker.next;

                    checkerPrev.next = checker.next;

                    checkerNext.prev = checker.prev    (maybe =ref ?)
                }

                checker = checkerNext;
            }

            ref = ref.next;
        }
    }
}
```

This list is circular, pointing back at sentinel