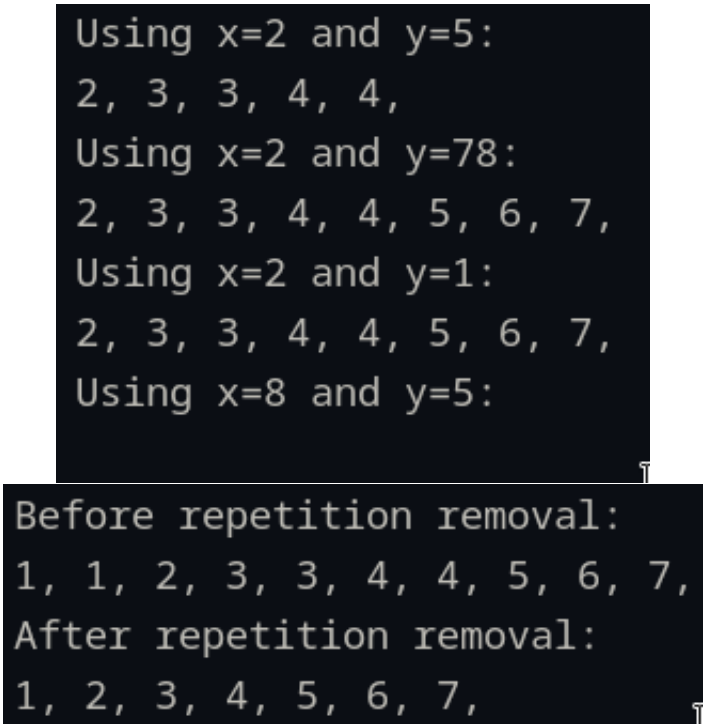
Assignment 2: Linked Lists and Arrays
Part 1: Linked List Iterator
CS3305/W01 Data Structures

Casey Hampson

August 27, 2024

Program Output

Just to make extra sure it is understood: the output for $x = 8$ and $y = 5$ has nothing below it, which is what is intended. Also, I misread the instructions and didn't notice we could use Java's built-in LinkedList, so I just made my own. Good practice!



Source Code

```
// Name: Casey Hampson
// Class: CS 3305/W01
// Term: Fall 2024
// Instructor: Sharon Perry
// Assignment: 2 - Part 1 Iterator


class Node {
    public int x;
    public Node next_node;

    Node(int _x) {
        this.x = _x;
        this.next_node = null;
    }
}

class LinkedList {
    public Node head;

    LinkedList() {
        this.head = null;
    }

    void Add(int x) {
        // create the node
        Node node = new Node(x);

        // if this is the first node, we just make it the head node
        if (this.head == null) {
            this.head = node;
            return;
        }

        // otherwise, we need to grab the final node and make this new one the next node
        Node current = this.head;
        while (current.next_node != null) current = current.next_node;

        current.next_node = node;
    }

    void Remove(Node node) {
        Node current = this.head;
        while (current != null) {
            // if the next node is the one we want to remove,
            // then we basically just create a skip over it
            // by making the current node's nextnode the one after the one we want to remove
            // a little confusing!
            if (!(current.next_node == null) && (current.next_node.equals(node))) {
                current.next_node = current.next_node.next_node;
            }
            current = current.next_node;
```

```
        }
    }

    // this just prints all the values in the list.
    // used for printing before/after repetition removal step
    void PrintAll() {
        Node current = this.head;
        while (current != null) {
            System.out.printf("%d, ", current.x);
            current = current.next_node;
        }
        System.out.println();
    }


    void PrintRange(int min, int max) {
        System.out.printf("Using x=%d and y=%d:\n", min, max);

        Node current = this.head;
        boolean x_found = false;
        while (current != null) {
            if (current.x == min) x_found = true;
            if (x_found) {
                // place break this first so that we don't include y, as per the instructions
                if (current.x == max) break;
                System.out.printf("%d, ", current.x);
            }
            current = current.next_node;
        }
        System.out.println();
    }


    void RemoveRepetitions() {
        Node current = this.head;
        while (current != null) {
            // start our second loop on the next node;
            // otherwise, the current node is always equal to the current node, so it would be removed
            Node num = current.next_node;
            while (num != null) {
                if (num.x == current.x) this.Remove(num);
                num = num.next_node;
            }
            current = current.next_node;
        }
    }
}


public class P1 {

    // i like abstracting this stuff away to make the main function as clean as possible
    // i could have made this a overloaded constructor, but using a class's method
    // from inside its constructor was making VSCode throw lots of warnings,
```

```
    // and I didn't want to recode the Add function again, basically
    private static LinkedList LinkedList_FromArr(int arr[]) {
        LinkedList list = new LinkedList();
        for (int i: arr) {
            list.Add(i);
        }
        return list;
    }

    public static void main(String[] args) {
        LinkedList list = LinkedList_FromArr(new int[]{1, 1, 2, 3, 3, 4, 4, 5, 6, 7});

        list.PrintRange(2,5);
        list.PrintRange(2,78);
        list.PrintRange(2,1);
        list.PrintRange(8,5);

        System.out.printf("Before repetition removal:\n");
        list.PrintAll();
        System.out.printf("After repetition removal:\n");
        list.RemoveRepetitions();
        list.PrintAll();
    }
}
```