CS 61B Spring 2019 — Linked Lists & Arrays

Discussion 3: Febuary 4, 2019

More Practice with Linked Lists

```java
public class SLList {
    private class IntNode {
        public int item;
        public IntNode next;
        public IntNode(int item, IntNode next) {
            this.item = item;
            this.next = next;
        }
    }

    private IntNode first;

    public void addFirst(int x) {
        first = new IntNode(x, first);
    }
}
```

1.1 Implement SLList.insert which takes in an integer x and an integer position. It inserts x at the given position. If position is after the end of the list, insert the new node at the end.

For example, if the SLList is $5 \rightarrow 6 \rightarrow 2$, insert(10, 1) results in $5 \rightarrow 10 \rightarrow 6 \rightarrow 2$ and if the SLList is $5 \rightarrow 6 \rightarrow 2$, insert(10, 7) results in $5 \rightarrow 6 \rightarrow 2 \rightarrow 10$. Additionally, for this problem assume that position is a non-negative integer.

```java
public void insert(int item, int position) {
```

```java
    if (first == null || position == 0) {
        addFirst(item);
        return;
    }
    IntNode currentNode = first;
    while (position > 1 && currentNode.next != null) {
        position--;
        currentNode = currentNode.next;
    }
    IntNode newNode = new IntNode(item, currentNode.next);
    currentNode.next = newNode;
}
```

1.2 Add another method to the SLList class that reverses the elements. Do this using the existing IntNode objects (you should not use **new**).

```
public void reverse() {
```

```
    if (first == null || first.next == null) {
        return;
    }

    IntNode ptr = first.next;
    first.next = null;

    while (ptr != null) {
        IntNode temp = ptr.next;
        ptr.next = first;
        first = ptr;
        ptr = temp;
    }
}
```

1.3 *Extra*: If you wrote reverse iteratively, write a second version that uses recursion (you may need a helper method). If you wrote it recursively, write it iteratively.

```
public void reverseRecur() {
    first = reverseHelper(first);
}

private IntNode reverseHelper(IntNode lst) {
    if (lst == null || lst.next == null) {
        return lst;
    } else {
        IntNode endOfReversed = lst.next;
        IntNode reversed = reverseHelper(lst.next);
        endOfReversed.next = lst;
        lst.next = null;
        return reversed;
    }
}
```

Arrays

2.1 Consider a method that inserts an `int item` into an `int[] arr` at the given position. The method should return the resulting array. For example, if `x = [5, 9, 14, 15]`, `item = 6`, and `position = 2`, then the method should return `[5, 9, 6, 14, 15]`. If `position` is past the end of the array, insert `item` at the end of the array.

Is it possible to write a version of this method that returns void and changes `arr` in place (i.e., destructively)? *Hint:* These arrays are filled meaning an array containing `n` elements will have length `n`.

No, because arrays have a fixed size, so to add an element, you need to create a new array.

Extra: Fill in the below according to the method signature:

```java
public static int[] insert(int[] arr, int item, int position) {

    int[] result = new int[arr.length + 1];
    position = Math.min(arr.length, position);
    for (int i = 0; i < position; i++) {
        result[i] = arr[i];
    }
    result[position] = item;
    for (int i = position; i < arr.length; i++) {
        result[i + 1] = arr[i];
    }
    return result;
}
```

2.2 Consider a method that destructively reverses the items in `arr`. For example calling `reverse` on an array `[1, 2, 3]` should change the array to be `[3, 2, 1]`. Write the `reverse` method:

```java
public static void reverse(int[] arr) {

    for (int i = 0; i < arr.length / 2; i++) {
        int j = arr.length - i - 1;
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}
```

2.3 *Extra*: Write a non-destructive method `replicate(int[] arr)` that replaces the number at index `i` with `arr[i]` copies of itself. For example, `replicate([3, 2, 1])` would return `[3, 3, 3, 2, 2, 1]`. For this question assume that all elements of the array are positive.

```java
public static int[] replicate(int[] arr) {
```

```java
    int total = 0;
    for (int item : arr) {
        total += item;
    }
    int[] result = new int[total];
    int i = 0;
    for (int item : arr) {
        for (int counter = 0; counter < item; counter++) {
            result[i] = item;
            i++;
        }
    }
    return result;
}
```