Lecture 9: Singly Linked Lists

- Singly Linked Lists

Some slides adopted from Algorithms 4th Edition and Oracle tutorials
Singly Linked Lists

- Dynamic linear data structures.
- In contrast to sequential data structures, linked data structures use pointers/links/references from one object to another.
Recursive Definition of Singly Linked Lists

- A singly linked list is either empty (null) or a node having a reference to a singly linked list.
- **Node**: is a data type that holds any kind of data and a reference to a node.
Node

private class Node {
    Item item;
    Node next;
}

Node
Standard Operations

- `SinglyLinkedList()`: Constructs an empty singly linked list.
- `isEmpty()`: Returns true if the singly linked list does not contain any item.
- `size()`: Returns the number of items in the singly linked list.
- `Item get(int index)`: Returns the item at the specified index.
- `add(Item item)`: Inserts the specified item at the head of the singly linked list.
- `add(int index, Item item)`: Inserts the specified item at the specified index.
- `Item remove()`: Retrieves and removes the head of the singly linked list.
- `Item remove(int index)`: Retrieves and removes the item at the specified index.
SinglyLinkedList(): Constructs an empty SLL

SinglyLinkedList<String> sll = new SinglyLinkedList<String>();

first = null

n = 0

What should happen?
sll.add("CS062");
add(Item item): Inserts the specified item at the head of the singly linked list.

sll.add("CS062")

n=1

sll.add("ROCKS");

What should happen?
add(Item item): Inserts the specified item at the head of the singly linked list

sll.add("ROCKS")

n=2

What should happen?

sll.add("!");
**SINGLY LINKED LISTS**

`add(Item item)`:
Inserts the specified item at the head of the singly linked list

```
sll.add("!");
n=3
```

```
What should happen?
sll.add(1,"?");
```

Diagram:
- Head/Beginning/Front/First
- `!` node connected to `ROCKS` node, which is connected to `CS062` node.
add(int index, Item item): Adds item at the specified index

sll.add(1, "?"), n=4

What should happen?

sll.remove();
remove(): Retrieves and removes the head of the singly linked list

What should happen?

sll.remove(1);

n=3
remove(int index): Retrieves and removes the item at the specified index

sll.remove(1)
n=2
Our own implementation of Singly Linked Lists

- We will follow the textbook style.
  - It does not offer a class for this so we will build our own.
- We will work with generics because we don’t want to offer multiple implementations.
- We will use an inner class Node and we will keep track of how many elements we have in our singly linked list.
public class SinglyLinkedList<Item> implements Iterable<Item> {
    private Node first; // head of the singly linked list
    private int n; // number of nodes in the singly linked list

    /**
     * This nested class defines the nodes in the singly linked list with a value
     * and pointer to the next node they are connected.
     */
    private class Node {
        Item item;
        Node next;
    }
}
Retrieve item from specified index

```java
/**
 * Returns item at the specified index.
 *
 * @param index the index of the item to be returned
 * @return the item at specified index
 */
public Item get(int index) {
    rangeCheck(index);

    Node finger = first;
    // search for index-th element or end of list
    while (index > 0) {
        finger = finger.next;
        index--;
    }
    return finger.item;
}
```
Insert item at head of singly linked list

```java
/**
 * Inserts the specified item at the head of the singly linked list.
 * @param item the item to be inserted
 */
public void add(Item item) {
    // Save the old node
    Node oldfirst = first;

    // Make a new node and assign it to head. Fix pointers.
    first = new Node();
    first.item = item;
    first.next = oldfirst;

    n++; // increase number of nodes in singly linked list.
}
```
Check if index is $\geq 0$ and $< n$

```java
/**
 * A helper method to check if an index is in range $0 \leq index < n$
 *
 * @param index the index to check
 */
private void rangeCheck(int index) {
    if (index >= n || index < 0)
        throw new IndexOutOfBoundsException("Index " + index + " out of bounds");
}
```
/**
 * Inserts the specified item at the specified index.
 *
 * @param index the index to insert the node
 * @param item the item to insert
 */
public void add(int index, Item item) {
    rangeCheck(index);

    if (index == 0) {
        add(item);
    } else {

        Node previous = null;
        Node finger = first;
        // search for index-th position
        while (index > 0) {
            previous = finger;
            finger = finger.next;
            index--;
        }
        // create new value to insert in correct position.
        Node current = new Node();
        current.next = finger;
        current.item = item;
        // make previous value point to new value.
        previous.next = current;

        n++;
    }
}
Retrieve and remove head

```java
/**
 * Retrieves and removes the head of the singly linked list.
 */

public Item remove() {
    Node temp = first;
    // Fix pointers.
    first = first.next;

    n--;

    return temp.item;
}
```
Retrieve and remove element from a specific index

```java
/**
 * Retrieves and removes the item at the specified index.
 *
 * @param index            the index of the item to be removed
 * @return the item previously at the specified index
 */
public Item remove(int index) {
    rangeCheck(index);

    if (index == 0) {
        return remove();
    } else {
        Node previous = null;
        Node finger = first;
        // search for value indexed, keep track of previous
        while (index > 0) {
            previous = finger;
            finger = finger.next;
            index--;
        }
        previous.next = finger.next;
        n--;
        // finger's value is old value, return it
        return finger.item;
    }
}
```
Lecture 9: Singly Linked Lists

- Singly Linked Lists
Readings:

- **Textbook:**
  - Chapter 1.3 (Page 142-146)

- **Textbook Website:**

Practice Problems:

- 1.3.18-1.3.27