Linked List Algorithms

Lecture Question

Task: Write reduce for our linked list

- Write a method in the datastructures.LinkedListNode class (from the repo) named reduce that:
  - Takes a function of type (A, A) => A
  - Returns A
  - Combines all the elements of the list into a single value by applying the provided function to all elements
  - If the list has size 1, return that element without calling the provided function

Example:
If head stores a reference to the List(4, 6, 2)

```
head.reduce((a: Int, b: Int) => a + b) == 12
```

Find a Value

- Navigate through the list one node at a time
    - Check if the node contains the value
    - If it doesn't, move to the next node
    - If the end of the list is reached, the list does not contain the element

```
def find(toFind: A): LinkedListNode[A] = {
  if (this.value == toFind) {
    this
  } else if (this.next == null) {
    null
  } else {
    this.next.find(toFind)
  }
}
```

Find - Recursion v. Iteration

```
def findIterative(toFind: A): LinkedListNode[A] = {
  var node = this
  while (node != null) {
    if (node.value == toFind) {
      return node
    }
    node = node.next
  }
  null
}
```

```
def find(toFind: A): LinkedListNode[A] = {
  if (this.value == toFind) {
    this
  } else if (this.next == null) {
    null
  } else {
    this.next.find(toFind)
  }
}
```

ForEach

- Call a function on each node of the list

```
def foreach(f: A => Unit): Unit = {
  f(this.value)
  if(this.next != null) {
    this.next.foreach(f)
  }
}
```

Map

- Apply a function to each element of the list
    - Return a new list containing the return values of the function

```scala
def map(f: A => A): LinkedListNode[A] = {
  val newValue = f(this.value)
  if (this.next == null) {
    new LinkedListNode[A](newValue, null)
  } else {
    new LinkedListNode[A](newValue, this.next.map(f))
  }
}
```

Map Usage

- Map function exists in the builtin list
- Used to transform every element in a list

```
val numbers: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
val numbersSquared = numbers.map((n: Int) => n * n)
println(numbersSquared)
```

```
List(1, 4, 9, 16, 25, 36, 49, 64, 81, 100)
```

Map - Change Type

- Can change the type of the returned list with a second type parameter
- A could be equal to B if the you don't want to change the type
- Example: You want to divide a list of Ints by 2 and have to return a list of Doubles to avoid truncation

```
def map[B](f: A => B): LinkedListNode[B] = {
  val newValue = f(this.value)
  if (this.next == null) {
    new LinkedListNode[B](newValue, null)
  } else {
    new LinkedListNode[B](newValue, this.next.map(f))
  }
}
```

Lecture Question

Task: Write reduce for our linked list

- Write a method in the datastructures.LinkedListNode class (from the repo) named reduce that:
    - Takes a function of type (A, A) => A
    - Returns A
    - Combines all the elements of the list into a single value by applying the provided function to all elements
    - If the list has size 1, return that element without calling the provided function

Example:
If head stores a reference to the List(4, 6, 2)

```
head.reduce((a: Int, b: Int) => a + b) == 12
```