Algorithms and Data Structures Assignment

Matthew Henderson 2031944H

Add

Insert an element *element* into the binary search tree bag.

```
1. Crete a new CountedElement elem from element
2. Set parent to null and curr to the root note of the binary search tree
3. Repeat
    a. If curr is null
        i. Replace the null reference (Either the root, parent's left child or parents
           right child) with a reference to a newly created node with element elem
        ii. Increment the size of the bag
        iii. Terminate
    b. If curr is equal to elem
        i. Increment the count of elem
        ii. Terminate
    c. Set parent to curr
    d. If elem is less than curr's element
        i. Set curr to curr's left child
    e. Else if elem is more than curr's element
        i. Set curr to curr's right child
```

Delete

Delete an element *element* from the binary search tree bag.

```
1. Create a new CountedElement elem from element
2. Set parent to null and curr to the root node of the binary search tree
3. Repeat
    a. If curr is null
        i. Terminate
    b. If curr's element is equal to elem and curr's element count is 1
        i. Delete the topmost element from the subtree with topmost node curr and
           get a link to the remaining subtree del
        ii. Replace the link to curr with del
        iii. Decrement the size of the bag
        iv. Terminate
    c. Else if curr's element is equal to elem and curr's element count is more than 1
        i. Decrement curr's element count
        ii. Terminate
    d. Set parent to curr
    e. If elem is less than curr's element
        i. Set curr to curr's left child
    f. Else if elem is more than curr's element
        i. Set curr to curr's right child
```

Delete the topmost element from a subtree which has topmost node *top.*

```
1. If top's left child is null
    a. Terminate and return top's right child
2. Else if top's right child is null
    a. Terminate and return top's left child
3. Else
    a. Set top's element to the leftmost element in the subtree of top's right child
    b. Delete the leftmost element in the subtree of top's right child
    c. Terminate and return top
```

Delete the leftmost element from a subtree that has topmost node *top.*

```
1. If top's left child is null
    a. Terminate and return top's right child
2. Else
    a. Set parent to top and curr to top's left node
    b. Repeat while curr's left child is not null
        i. Set parent to curr and set curr to parents left child
    c. Set parent's left child to curr's right child
    d. Terminate and return top
```

Get the leftmost element of a subtree with topmost node *top.*

```
1. Set curr to top
2. Repeat while curr's left child is not null
    a. Set curr to curr's left child
3. Terminate and return curr's element
```

Iterator

Returns an *iterator* to visit each node of the binary search tree bag in order. This is modelled as a LinkedStack.

Create a new in-order iterator

```
1. Create a new LinkedStack called track with type Node<E>
2. Set curr to the root node
3. Repeat while curr is not null
    a. Push curr track
    b. Set curr to curr's left child
```

Get the next element from the iterator

```
1. Pop the top node from track and call it place
2. Set curr to places's right child
3. Repeat while curr is not null
    a. Push curr to track
```

   b. Set *curr* to *curr’s* left child

4. Terminate and return *place’s* element