Sample algorithms for selected Bag ADT methods

Much of the code for this assignment is available in chapter 3 of the textbook. The algorithms I provide here should help you understand how to implement the functionality for the methods which aren't covered in the book.

1-arg/cloning constructor `LinkedBag( BagInterface<T> sourceBag )`:

```
1. initialize the instance state to that of an empty bag (hint: invoke the no-arg
   constructor)
2. 'convert' the contents of sourceBag to an array of entries (T[] sourceBagContents)
3. iterate over the entries in sourceBagContents
    a. this.add() the entry to the bag we're constructing
```

```
This approach is universal in that it will work with any implementation of BagInterface.
An alternate approach treats sourceBag more efficiently if it's a LinkedBag:
```

```
1. initialize the instance state to that of an empty bag
2. if sourceBag is a LinkedBag:
    a. traverse sourceBag's chain (same strategy as the other traversals)
        i. this.add() the data referenced by the current Node
3. otherwise // sourceBag is a different bag implementation
    a. 'convert' the contents of sourceBag to an array of entries
       (T[] sourceBagContents)
    b. iterate over the array of entries in sourceBagContents
        i. this.add() the entry to the bag we're constructing
```

BagInterface<T> difference(BagInterface<T> anotherBag):

```
1. instantiate resultBag as a new bag with its initial contents the same as ours
   (clone this)
2. 'convert' the contents of anotherBag to an array of entries
   (T[] anotherBagContents)
3. iterate over the entries in anotherBagContents
    a. resultBag.remove() each entry from resultBag (we don't care whether it
       succeeded)
4. return resultBag
```

```
As with the cloning constructor, we can make this more efficient when anotherBag is a
LinkedBag by traversing its chain of Nodes – we still use the array of entries for other
bag implementations.
```

BagInterface<T> intersection(BagInterface<T> anotherBag):

```
1. instantiate resultBag as an empty bag
2. clone anotherBag (clonedBag) (will make life easier than copying its entries to an
   array because we can use our own methods to manipulate the contents)
3. traverse our (this) chain of Nodes
     a. if clonedBag.contains() the entry our current Node references
          i. clonedBag.delete() the entry from clonedBag
         ii. resultBag.add() the entry to resultBag
4. return resultBag
```

We don’t need to concern ourselves with the implementation of anotherBag – the cloning constructor deals with that and gives us a LinkedBag which we can manipulate directly.

BagInterface<T> union(BagInterface<T> anotherBag):

```
1. clone ourself (this) to get our resultBag
2. ‘convert’ the contents of anotherBag to an array of entries
   (T[] anotherBagContents)
3. iterate over the entries in anotherBagContents
     a. resultBag.add() the entry to the bag we’re constructing
4. return resultBag
```

As with the cloning constructor and difference(), we can make this more efficient when anotherBag is a LinkedBag by traversing its chain of Nodes – we still use the array of entries for other bag implementations.