```
public class LinkedBag<T> implements
Bag<T> {

   private class Node {
      private T data;
      private Node next;
   }

   private Node head;

   public LinkedBag() {
      head = null;
   }

   public int getCurrentSize() {
      int size = 0;
      Node current = head;
      while (current != null) {
         size++;
         current = current.next;
      }
      return size;
   }

   public boolean isEmpty()
   {
      return (head == null);
   }

   public boolean add(T newEntry)
   {
      Node newItem = new Node();
      newItem.data = newEntry;
      newItem.next = head;
      head = newItem;
      return true;
   }

   public int getFrequencyOf(T anEntry) {
      int frequency = 0;
      Node current = head;
      while (__________________) {
         if (__________________) {
            frequency++;
         }
         current = ________________
      }
      return frequency;
   }
```

```
   public boolean remove(T anEntry)
   {
      Node current = head;

      while (_________________)
      {
         // When we find what we
         // are looking for,
         // Remove by swapping
         // data from first position in
         if (_________________) {
            current.data =____________;
           //After copying, move the head
            //node to node after the
            //current head
            head = _________________
            return true;
         }
         current = _________________;
      }
      return false;
   }


// We aren't going to implement grab()
// for a linked bag in class.
// One way of doing so would be to:
// 1. Get the current size of the bag
// 2. Choose a random number <size
// 3. Traverse the bag keeping track of
// where you are
// 4. Perform the same operation as
// remove() at the right index
// This is O(n) (because getCurrentSize()
//is O(n). We then pay another O(n) to
// iterate through the bag
// How could we implement (random) grab
// without storing the size and without
// iterating through the bag twice?

public T grab() {




}


}
```