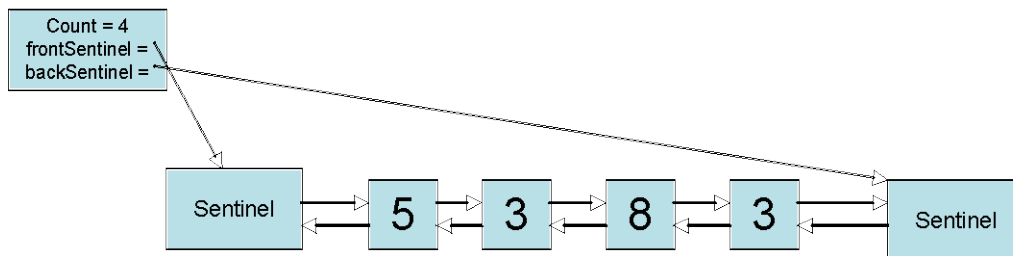


Worksheet 22: Constructing a Bag using a Linked List

In Preparation: Read Chapter 8 to learn more about the Bag data abstraction. If you have not done so already, complete Worksheets 17 and 18 to learn about the basic features of the linked list.

In this lesson we continue developing the LinkedList data structure started in Worksheet 19. In the earlier worksheet you implemented operations to add and remove values from either the front or the back of the container. Recall that this implementation used a sentinel at both ends and double links. Because we want to quickly determine the number of elements in the collection, the implementation also maintained an integer data field named count, similar to the count field in the dynamic array bag.



Also recall that adding or removing elements is a problem that you have already solved. Adding a new value at either end was implemented using a more general internal function, termed `addLink`:

```
void _addLink (struct LinkedList * lst, struct dLink * lnk, TYPE e);
```

Similarly removing a value, from either the front or the back, used the following function:

```
void _removeLink (struct linkedList * lst, struct dLink * lnk);
```

To create a bag we need three operations: `add`, `contains`, and `remove`. The `add` operation can simply add the new value to the front, and so is easy to write. The `contains` method must use a loop to cycle over the chain of links. Each element is tested against the argument, using the `EQ` (equal to) macro. If any are equal, then the Boolean value `true` is returned. Otherwise, if the loop terminates without finding any matching element, the value `False` is returned.

The `remove` method uses a similar loop. However, this time, if a matching value is found, then the method `removeLink` is invoked. The method then terminates, without examining the rest of the collection.

Complete the implementation of the `ListBag` based on these ideas:

Worksheet 22: Constructing a Bag using a Linked List NAME:

```
struct dLink {
    TYPE value;
    struct dLink * next;
    struct dLink * prev;
};

struct linkedList {
    struct dLink * frontSentinel;
    struct dLink * backSentinel;
    int size;
};

/* the following functions were written in earlier lessons */
void linkedlistInit (struct linkedList *lst);
void linkedListFree (struct linkedList *lst);
void _addLink (struct linkedList * lst, struct dLink * lnk, TYPE e);
void _removeLink (struct linkedList * lst, struct dLink * lnk);

void linkedListAdd (struct linkedList * lst, TYPE e)
{ _addLink(lst, lst->frontSentinel->next, e); }

/* you must write the following */

int linkedListContains (struct linkedList *lst, TYPE e) {

}

void linkedListRemove (struct linkedList *lst, TYPE e) {

}

}
```

1. What were the algorithmic complexities of the methods addLink and removeLink that you wrote back in linked list for Deque?

2. Given your answer to the previous question, what are the algorithmic complexities of the three principle Bag operations?