5. Doubly Linked List

Kristiāns Vinters

Fall 2023

Introduction

I solved the assignment in Go. I used Go because I want to become more familiar with it. Source code and benchmark data is available on GitHub*.

Implementation

There weren't any particular difficulties implementing doubly linked lists in Go, as Go handles references with C-like pointers. I implemented the structure in a separate `dllist` package, very similar to the `llist` package from assignment 4.

```
type DoublyLinkedListItem[T comparable] struct {
    Head T
    next *DoublyLinkedListItem[T]
    prev *DoublyLinkedListItem[T]
}

type DoublyLinkedList[T comparable] struct {
    first *DoublyLinkedListItem[T]
    last  *DoublyLinkedListItem[T]
}
```

The `Remove()` function did not pose any difficulties, as it was just a linear search to find the item then rearranging the pointers and handling the edge cases. I implemented the `Remove()` function using the `Unlink` function, as the functionality is almost the same, bar the initial search step.

```
func (l *DoublyLinkedList[T]) Unlink(item *DoublyLinkedListItem[T]) {
    if item.prev != nil {
        item.prev.next = item.next
    } else {
```

*https://github.com/Phanty133/id1021/tree/master/4-linkedlist

```
        l.first = item.next
    }

    if item.next != nil {
        item.next.prev = item.prev
    } else {
        l.last = item.prev
    }
}

func (l *DoublyLinkedList[T]) Remove(value T) {
    if l.first == nil {
        return
    }

    item := l.Find(value)

    if item == nil {
        return
    }

    l.Unlink(item)
}
```

Benchmarking

I benchmarked the linked list and array by running them 500 times with a $k = 1000$ and changing sizes $\{10, 100, 1000, 5000, 10000, 15000\}$. I generated the k random items as suggested in the assignment description.

```
a, items := PrepareDLLData(n)

kIdxs := GenRandomInts(k, 0, n)
kItems := make([]*dllist.DoublyLinkedListItem[int], k)

for j := 0; j < k; j++ {
    kItems[j] = items[kIdxs[j]]
}
```

Size	t_{LL}, ms	t_{DLL}, ms
10	0.005	0.005
100	0.052	0.003
1000	0.487	0.003
5000	2.94	0.004
10000	5.59	0.004
15000	7.72	0.005

Figure 1: Median execution times for unlinking then inserting k elements.

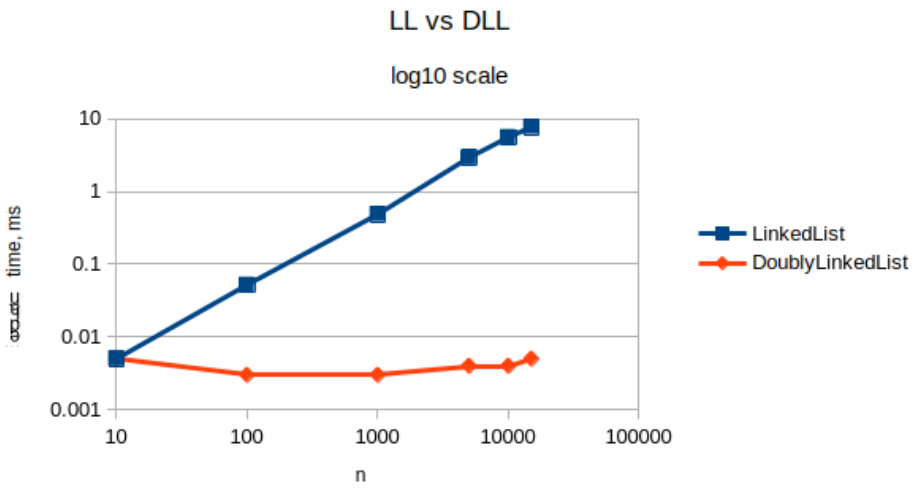

Figure 2: Median execution time

The execution time for the doubly linked list and singly linked list starts out the same, but as n grows larger, the singly linked list becomes significantly slower. The time complexity for the linked list is $O(n)$, as it has to traverse the entire list to find the item to remove. The time complexity for the doubly linked list is $O(1)$, as it has a reference to the previous item, so it can just unlink the item and insert it at the beginning of the list.