Doubly Linked Lists

Dhruba Das

Spring Fall 2023

Introduction

This assignment has a simple objective: link each cell of the list such that its linked to both the cells before and after it. This is done so by creating another pointer which points to the cell before it. We then benchmark two new methods: Unlink and Insert.

Cell Structure

The cell structure is essentially the same as that of the singly linked list, except it now has another pointer which points to the previous cell, if there is one. Below is the code snippet:

```
public class Cell {
        int value;
        Cell prev;
        Cell next;

        public Cell(int val) {
            this.value = val;
        }
    }
```

A constructor is then made to create the linked list; the code is given below:

```
public DList(int n) {
            cellArray = new Cell[n];
            for(int i = 0; i < n; i++){
                add(i);
                cellArray[i] = first;
            }
        }
```

As it can be seen above, every time a cell is added to the list, it's also added to a array which contains all the cells in the list. This is required in order to use the Unlink and Insert methods during benchmarking.

Implementation of the methods

For each method, it's essentially the same as that of the singly linked list, except *remove* and *add*. Below is the code snippet for both:

```
public void remove(int item){
            if(first.value == item){
                first = first.next;
                first.prev = null;
            }
            Cell current = first;
            while(current.next != null){
                if(current.value == item){
                    current.next.prev = current.prev;
                    current.prev.next = current.next;
                    return;
                }
            current = current.next;
            }
            if(current.value == item && current.next == null){
                current = current.prev;
                current.next = null;
            }
        }
...
public void add(int item) {
            //Create a new Cell
            Cell newC = new Cell(item);

            //if list is empty,  newC is first
            if(first == null) {
                first = newC;
            }
            else {
                //add newC to the beginning of list.
                first.prev = newC;
                //newNode->previous set to tail
                newC.next = first;
                //newNode becomes new tail
                first = newC;
```

```
            //tail's next point to null
            newC.prev = null;
        }
    }
```

Unlink method

The concept of this was a little hard for me to grasp at first, but from what I understood, instead of removing an item from the list, we completely remove links of the cell itself. Below is the code snippet for the doubly linked list:

```
public void unlink(Cell gone){
        if(first == gone){
            first = first.next;
        }
        if(gone.next != null){
            gone.next.prev = gone.prev;
        }
        if(gone.prev != null){
            gone.prev.next = gone.next;
        }
    }
```

Next is the code snippet for the implementation of this method in the singly linked list:

```
public void unlink(Node gone){
        if(first == gone){
            first = first.tail;
        }

        Node current = first;
        while(current.tail != null && current.tail != gone){
            current = current.tail;
        }
        if(current.tail == gone){
            current.tail = gone.tail;
        }
    }
```

As seen above, the implementation is a little different from that of the doubly linked list, since we don't have access to the previous cell.

Insert method

This method was quite easy to implement, as it's essentially the add method in concept, but instead of an item, we add a cell. Below is the code snippet for the doubly linked list:

```
public void insert(Cell newC){
            newC.next = first;
            newC.prev = null;
            first = newC;
        }
```

Next is the code snippet for the implementation of this method in the singly linked list:

```
public void insert(Node newC){
            newC.tail = first;
            first = newC;
        }
```

As seen above, we do not have to worry about the *prev* pointer in the singly linked list implementation of the method.

Benchmarks

I have represented the difference in performances between unlinking and inserting a thousand cells two linked lists, in a table below (S is for singly; D is for doubly linked list):

Size(n)	**S (ms)**	**D (ms)**
1600	6967	154
3200	10000	67
6400	20326	61
12800	40457	61
25600	80623	78
51200	169486	79
102400	321864	55

Table 1: (For each size, unlink and insert is carried out for 1000 cells; the min. time (in microseconds) is taken)

Conclusion

Doubly linked lists are much more efficient when it comes to accessing the contents of the data structure, when compared to singly linked lists; this is because the traversal time is much less when we want to retrieve a cell/item, as we don't have to traverse from the beginning every single time; we can simply move backwards to the previous cells.