Double Linked List

Fredrik Berzins

Fall 2023

Introduction

This assignment will be a comparison between linked lists(one-way and double linked lists(dual-way). A double linked list is a list where you not only have one forward pointer but also a previous pointer. A reference to the previous element will come in handy when we want to remove an element in the list without traversing the list from the beginning.

Linked structure explanation

A linked structure also called a node structure is a structure made from nodes and references to other nodes. Each node can have some information and some references.

Linked list explanation

A linked list is a node structure that has one known origin node and every node links to one other node, only the origin node is known the rest are only known by reference from the node before them. Each node can hold multiple data types of information. The last node in a linked list still has a reference but it's referencing a null object.

Double linked list explanation

A double linked list is like a linked list but each node has a extra pointer to the element before it.

Implementation

Linked list implementation

The linked List is implemented by creating nodes with a null pointer from the beginning that will later be changed it also always has space for an int, this is shown in the node constructor method below.

```
private class Node {
    public int value;
    public Node next;

    public Node (int value) {
        this.value = value;
        this.next = null;
    }
}
Node first;
```

The two main functions for benchmark was link and unlink, link linked a argument node to the beginning of the linked list.

Unlink extracts a node from a list and redirects the previous node pointer. The unlink was more complicated since it has to have different cases for first, last or a node in the center of a list.

```
public void link(Node m) {
    m.next = first;
    first = m;
}
public void unlink(Node m) {
    Node n = first;
    if (m == first) {
        first = m.next;
    }
    else {
        while (n != null) {
            if (n.next == m) {
                n.next = m.next;
                m.next = null;
                break;
            }
            else {
                n = n.next;
            }
        }
    }
}
```

Double linked list implementation

The only difference in node implementation is 2 added lines of code, one to define a Node prev and one to set this value to null. This makes both Node next and prev start at null and later changed.

The difference for link is minor since it only one extra a pointer to change. On the other hand unlink is made to not have to loop thru the list instead it works with the help of prev since it can redirect pointers to ”remove” a node from a list.

```
public void link(Node m) {
    first.prev = m;
    m.next = first;
    first = m;
}
public void unlink(Node m) {
    Node n = first;
    if (m == n) {
        first = m.next;
        m.next.prev = null;
    }
    else if (m.next != null) {
        m.prev.next = m.next;
        m.next.prev = m.prev;
    }
    else{
        m.prev.next = null;
    }
}
```

Benchmarks

The testing was done with length between 100 and 1600 values(doubling each time, 100,200,400...), the amount of random cells for all test was 400. Each time test was redone 1’000 times witch included unlink and link 100 different nodes to get as accurate results as possible. This gives a min time showing avg from the group with the 100 quickest runs.

Result

The result from the benchmark is that a linked lists time will be dependent on the size, with a big o notation of O(n). While the result for double linked list is not dependent on size it is constant, this gives it a big o notation of O(1).

n	Linked list	Double linked list
100	69849	11088
200	123407	11050
400	234225	11011
800	453531	11011
1600	1199984	11101

Table 1: Shows time to unlink then link a random node in the list, in ns.

Conclusion

In conclusion double linked list is faster to work with independent of size. Their are downsides to double linked list since it has a second pointer it uses more memory that might not be used, for example when making a stack when you only need to know what the first then what the next element is. it might also be preferred to keep the node class privet, this will hinder some of the use cases of double linked list.