Lab 6 Doubly Linked List

Weichen Chai

Fall 2022

Introduction

This lab report seeks to answer the questions presented in "Doubly linked list" for ID1021. In this assignment, the goal is to analyse the advantage of disadvantages of the doubly array, as well as how to implement them, and to benchmark the speed of this array and compare it with our previous lab, where we discussed the speed of singly linked lists.

Background

Similar to the previous lab, we will be working with linked lists, which are structures that are linked to each other using forward pointers, an element can use this pointer to point towards the next element, however, the element being pointed to is unaware of the reference. In addition to singly linked list, doubly linked lists will include a previous pointer, which allows us to return to the previous element. This difference makes it possible to execute functions on an element without traversing through the entire list, which is an advantage compared to singly linked lists as it is more efficient. The draw back of a doubly linked list is that it requires much more memory space due to the fact that more references have to be included.

Tasks

In this task, we were asked to make a benchmark which keeps a list of n elements, and performs k add and remove operations in the doubly linked list. The selection for add and remove operations are random, in the end, doubly linked list should be compared to singly linked list after the same operation is performed on them.

In order to compare the two different types of linked list, some code which randomly chooses indexes in the range of 0 to n-1 was given, where the variable k determines the amount of variables we want to perform was set to 100 times for this report.

Benchmark

Array Size	Doubly Linked List(Nanoseconds)	Singly Linked List(Nanoseconds)
100	10.2	22.7
200	9.8	34.2
300	10.3	47.3
400	10.3	62.8
500	9.9	77.6
1000	11.1	148.2
2000	10.6	286.5
5000	10.5	701.4

In the table above, we have performed the remove function first, which removes one variable at random, then added it back to the first position in the linked list. it is visible that the doubly linked list is more or less of constant time complexity, making it O(1), and singly linked list follows a O(n) trend. The reason that doubly linked lists are way faster is due to the fact that we receive a reference to the node we should remove and we can jump directly to it when deleting. The codes for the doubly linked list is as follows:

```
public void deleteDoubly(Node D)
    {
        // If node is the head of the list
        //set the next node to be the head instead
        if (head == D) {
            head = D.next;
        }

        if (D.next != null) {
            D.next.prev = D.prev;
        }

        if (D.prev != null) {
            D.prev.next = D.next;
        }
.
.
.
```

In the task, we were made to think about special cases for deletion, and there are only 4 cases in which deletion can occur, the first part shows that

when the head is removed,head is set to the next node, the two if statements after that indicates what happens if the node deleted is not the last node, and not the first node respectively. The codes shown for doubly deletion means that it is efficient at removing, as it only requires to access the node that was given to it, then changing the reference, making it take an extremely short amount of time and giving it the time complexity of O(1).

A difference between Singly linked list is that it doesn't have a previous reference, making it unable to jump to the node of its choosing, and therefore, functions mostly like a search algorithm, which we know from the previous tests, have the time complexity of O(n). It will read through the entire list as the previous nodes needs to be known and the traversal will take O(n).

Since the requirement in the task is to add first, the time complexity will not be so different for doubly and singly linked lists, as stated before, singly linked lists will have to traverse through the list, so its time to add an element will be O(n), however, the element added will be in the first position, making it O(1). Time complexity for doubly linked lists, as stated in deletion process, will be O(1) as well, of course, adding with a doubly linked list may be slightly inefficient, since the program requires for it to update the previous pointer to the first node in the list, with the codes shown below.

```
public void addDoubly(int v) {
        Node node = new Node(v, null, null);

        if (head == null)
            tail = node;
        else {
            node.next = head;
            head.prev = node;
        }

        head = node;
    }
```

Through the analysis of addition and deletion process comparing singly linked list to doubly linked list, we can see that the addition process for doubly light take slightly longer than singly, due to the fact that it will need to change its previous reference, both times complexities for the add to first function is O(1), however, the deletion process for doubly will be much quicker, as it will only need to access one node to change the reference, whereas singly linked lists will have to traverse through the entire list to find the node that is required to be removed. Making deletion's time complexity

for singly linked list to be O(n) and O(1) for doubly linked list. When we perform the operation to remove a variable at random first, and then add it back to first position. Its clear that for doubly, the time complexity is O(1) because both the addition operation and the deletion operation are of O(1).

for singly linked lists on the other hand, it will be O(n) because even though its addition operation has the complexity of O(1), its deletion operation still has to traverse through the list, making that operation have the time complexity of O(n).