Double linked list

Laasyasree Desu

Fall 2023

1 Introduction

This assignment involved implementing a double linked list and different methods such as insertion, deletion, finding the length, and finding an element. The double linked list was then compared to a single linked list for insertion and deletion.

2 Linked list

A linked list is a data structure used for the storage of elements. Linked lists operate through references where elements are organized in a series of cells, each containing an element and a reference to the next cell. The initial cell acts as the starting point, and all subsequent cells point to the following element in the sequence. The last cell's pointer is set to null since there are no more elements to reference beyond it. To create the list a for loop was added which iterated n times, n being the length of the list. During each iteration, the pointer of the current element was directed to the next added element, and the current element itself was updated accordingly. An example of this code can be seen below:

```
public class list {
    int head;
    list next;


public list(int element, list after) {
    head = element;
    next = after;
    }
```

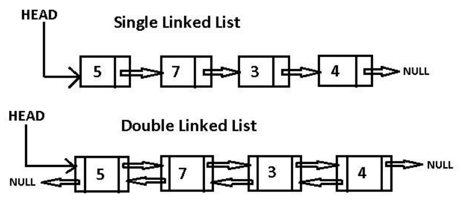

Figure 1: Visual representation of single and double linked list taken from *"Data Structures In The Real World — Linked List"* by Christopher Webb, 2011, Medium.

3 Double linked list

A double linked list shares several similarities with a single linked list, as previously discussed, with one crucial distinction – it incorporates two references or pointers within each cell instead of one. These two pointers serve distinct purposes: one points to the previous cell in the sequence, while the other points to the next cell. This feature enables double linked lists with bidirectional traversal capabilities, enabling navigation in both forward and backward directions within the list. In terms of the terminal cells of the double linked list, the first cell's previous pointer and the last cell's next pointer are both set to null. This is because no other cells are present before or after. A visual representation of single and double linked list can be seen above.

This data-structure was also implemented with a for loop that iterates through the list. For every for loop the the new cell would be added while updating the next and the previous pointers (to null and the previous element).

```
public class doublelist {
    int head;
    doublelist next;
    doublelist prev;


public doublelist(int element, doublelist before, doublelist after) {
    head = element;
    prev = before;
    next = after;
    }
```

3.1 Methods

After creating the linked list, different methods of altering and using the list were also implemented. The methods included adding an element/cell, deletion, finding the length of the list and finding a specific element.

3.1.1 Adding

This method involved adding an element (cell) to the beginning of the list or to the end. Adding to the beginning was done by getting the cell to be added as a parameter first. The cells ”next” pointer was set to point at the ”head” or the first element. The ”previous” pointer of new cell was set to null, and the new cell was updated as the first cell. The method to add at the end was implemented similarly, the list was traversed to get to the last cell and its null pointer was set to point at the new element instead. The new elements previous and next pointers were then updated. To add in between the list, the list would be traversed to the position in which to add the cell (take as a parameter), the previous and next pointers of the cell before and after would be updated.

3.1.2 Deleting

The deleting or removing method can be broken into three parts - deleting the first cell, deleting the last cell or deleting in between the list. If the cell to be removed is the first one, then the first cell would be updated to the next cell. If the cell in question is at the end, then the previous cells ”next” pointer is updated to null. If the cell is in the middle of the list, then next cells ”previous” pointer is updated to cell before, and the previous cells ”next” pointer is updated to the cell after, essentially removing the links to the cell being removed.

3.1.3 Finding

Like the deleting method, to find an element a for loop was created to traverse through the list while checking if the element in question was present in the list (if-statement). If the element was found the method returned true. Otherwise at the end of the loop it returned false.

3.1.4 Length

In order to determine the length of the linked list, a for loop was created to traverse the list, incrementing a counter with each iteration. Upon the completion of this loop, the resulting count was returned as the length of the list.

4 Benchmark

In this task a single linked list and a double linked were compared using two methods, add and remove. For this comparison, random elements were added and random positions were given to remove certain elements. This was done on both list types.

List size	Time taken (µs)
100	700
200	1500
400	3700
800	9300
1600	15200

Table 1: Average time taken to add and remove elements from a singly linked list

The results show a linear relationship between the list size and the time taken add and remove elements. This gives the time complexity O(n), as the time depends on the length of the list. This can also seen in the algorithm as to be able to remove an element, it ha to first be present in the array, then the previous cell had to be accessed to be able to update the pointer. This indicates that the process would dependant on the size of the array.

List size	Time taken (µs)
100	96
200	127
400	115
800	148
1600	133

Table 2: Average time taken to add and remove elements from a doubly linked list

The results show a relatively consistent time measurement for adding and deleting elements in a double linked list. This results in the time constant time complexity O(1), as the time taken is independent of the list size. The results also show a significantly faster run-time for this data-structure compared to the single linked list. This increased efficiency can be attributed to the features of the double linked list. The previous pointers present in the list make accessing the cell to be removed much more efficient, as it is accessed directly compared to traversing the entire list (single linked).

5 Sources Cited

1. Webb, C. (2017). Data structures in the real world - linked list. Retrieved from https://medium.com/journey-of-one-thousand-apps/data-structures-in-the-real-world-508f5968545a