Linked lists

Isak Wilkens TIDAB

Fall 2022

Introduction

This is a report on the fifth assignment "Linked lists" on the course "ID2021" TIDAB, KTH. The assignment is to do several tasks with linked lists and then also analyze this in comparison to similar tests done with an array. A linked list is a more complex structure in comparison to an array that uses pointers to keep track of the next element in the list. Each element in the list is called a node and keeps track of its own value and a pointer reference to the next element in the list.

Task 1

In the first task the code for a linked list is given and the task is to use an append operation in two different ways. Firstly a linked list of varying size is to be added to a linked list of fixed size and bench-marked then secondly the same is to be done but the other way around where a fixed list is added to a linked list of varying size. To be able to make linked lists and test them a few methods were added. A method to create linked lists of any size was added with the following code:

```
public static LinkedList createList(int nElements) {
       Random rnd = new Random();
       LinkedList temp = new LinkedList(rnd.nextInt(300), null);
       for(int i = 0; i < nElements; i++){
           temp.append(new LinkedList(rnd.nextInt(300), null));
       }
       return temp;
   }
```

A print method was added to see if the linked list was created and worked correctly:

```
  public void print(LinkedList b){
       LinkedList nxt = this;
```

```
        int i = 1;
        while (nxt.tail != null){
            System.out.println("Element " + i + ": " + nxt.head);
            i++;
            nxt = nxt.tail;
        }
    }
```

Table 1 shows the mean value from all the measurements on the two different tasks. As we can see the run-time seems to be dependent on the linked list that is being appended to and not the list that is being appended. This is because the program needs to go through the whole list until it finds a null-pointer and thus finding the end of the linked list where the append operation is to be done linking the two lists together. In the case of appending to a fixed list we get constant time complexity of O(1) for the run-time and in the case of appending a fixed and having to go through a varying size of list we get a linear dependency of O(n).

Varied array size(n)	**256**	**512**	**1024**	**2048**	**4096**	**8192**
Append varied to fixed	120	200	250	230	550	700
Append fixed to varied	600	4400	5200	6900	7015	14700

Table 1: Mesurements of the run-time as a function of linked list size. Run-time in nano seconds.

Task 2

The next task is to compare the operation of appending a linked list to appending an array with another array. This test is done with varying sizes of linked list and array, in this test the linked list that is being appended to is being varied since the other version will just be constant and not as interesting of a comparison. The following code was added for an operation that appends arrays:

```
    public static int[] appendArray (int[] arrayA, int[] arrayB) {
        int[] tempArray = new int[arrayB.length + arrayA.length];
        int index = 0;
        for (int i=0; i < arrayA.length; i++){
            tempArray[index] = arrayA[i];
            index++;
        }
        for (int j=0; j < arrayB.length; j++){
            tempArray[index] = arrayB[j];
            index++;
        }
        return tempArray;
    }
```

Table 2 shows the results from the comparison what we can see is that both increase with the size of the array or list but the array append operation is faster in every case but not with a very large margin. Both follow the time complexity of O(n) with regards to run time as a function of list or array size.

Varied array size(n)	**256**	**512**	**1024**	**2048**	**4096**	**8192**
Linked list	4000	4500	5000	6700	6700	14700
Array	2200	3500	4000	4400	5800	8000

Table 2: Mesurements of the run-time as a function of linked list size. Run-time in nano seconds.

Task 3

The third task was to bench-mark allocating an array and a linked list of varying sizes to compare the two.

Table 3 shows the results for this which shows that it is much more costly to a build a linked list than it is to build an array of the same size:

Varied array size(n)	**10**	**100**	**1000**	**10000**
Linked list	1100	13600	$7,25 * 10^5$	$7,8 * 10^7$
Array	850	2500	13400	$1,1 * 10^5$

Table 3: Mesurements of the run-time as a function of linked list size. Run-time in nano seconds.

Task 4

The final task was to implement a stack in a similar way to the previous exercise *A calculator* but this time using linked lists. The code below shows this implementation:

```
public class Stack {
    LinkedList element = null;
    public void push(int item){
        element = new LinkedList(item, element);
    }

    public int pop(){
        int temp = this.element.head();
        element = this.element.tail();
        return temp;
    }
}
```

This implementation of a stack required a lot less code and the execution time seemed faster. The code here has a lot less conditional statements and only uses up as much memory as it needs for each element and its pointer. This means that we do not need to think about how much space we need to allocate or decrease in a dynamic stack. The system of having a linked list with nodes seem to work very well for a list that is purposed to be used as a stack. However it would use more memory since on top of storing the element in the node we also need to store the pointer for the next value.

Conclusion

In conclusion a linked list has many interesting features and is sometimes a great choice for handling data. In many cases its unique features with pointers make them easy to manage and implement. Some implementations such as a stack get very clear and straight forward code. Linked lists are however not always superior to other list types such as the array and often take up more memory and execution time. The sorting and searching algorithms we have previously studied in this course does not make much sense on linked lists and additional features would have to be added to keep track of the lists size and index.