Linked lists

Nariman Haidar

Fall term 2022

1 Introduction:

In this assignment we are going to write a simple linked structures and find out their properties. And then we are going to compare the time to append time between the two single linked list and two arrays. Then we are going to write a code to linked list stack and dynamic array stack and describe the difference in execution time for the array implementation and the linked list implementation.

2 Method:

A linked list:

The simplest linked structures is a linked list. Every element in a linked list is called a node. It consists of two fields data and a pointer next which points to the next node in the list. The first node in the linked list is called a head and the last node is called a tail, and in the last node the pointer is points to the null. To implement the code in our assignment I started with write the first class and that is node and in this node there is one construes that have two argument one to value and one to next. Then I started to write a class linked list. I wrote five functions one to add new node and one to remove a node and one to append two lists and one to display and the main function. In the add function first I put a new node that have next was null and next I set the head is the new node so this is the top. In the function remove first I check if it is empty and then I made the head pointer to the next node in the list. In the append function I used the assignments code and the append function makes if we have two lists and we want them together so we use the append function to make them in one list. In the display function I just wrote it to display the lists that we wants to display. In the main function I wrote first random list to the first list with the fixed size and then I wrote another list with not fixed size and the append both of the lists and then I wrote some bench mark to measure the time to append those of lists and then I wrote the first list is not fixed size and the second list is a fixed size and then I use the same bench mark to measure the time. The time complexity to the linked list is o(n) because of the elements in the linked list are not contiguous, so every element access need a O(n). The add function in the linked list is different because if we wants to add at the first so we needs o(1) because we just have to do the new node next reference to the head and then the head reference to the new node and it is constant time. but if we need to add the new node at the end or in the mid so we need to go through all the needs or some of them in the list to put the new node and it takes o(n). The remove function is the same as the add function if we want remove the first node so it takes o(1) but if we wants to remove from the last or from the mid so it takes o(n) for the same reason. The append function if we want to put the first list at the first so it takes o(1), but if we want to put the first list in the last so it takes o(n). because it should pass every element to get the Right place.

code:In this code we can see how we can implement the code to add and remove a node.

```
public void add(Node newNode) {
                newNode.next = head;
                head = newNode;
                size++;
        }
        public void remove() {
                if (isEmpty()) {
                        System.out.println("The list is empty ");
                }
                head = head.next;
                size--;
        }
```

Append Array:

To make an append array first I wrote a function that to append two arrays in this function that has two arguments the first array and the second array, Then I made an array that has the size to both of two arrays and then for loop to get elements from the first array and then another for loop to get the elements from the other array. Then I wrote some bench mark in the main to measure the time that it takes to append two arrays together. The complexity time to append two arrays is O(n) add to the beginning or end.

code:In this code we can see how we can implement the code to append two arrays.

```
public void appendArrays(int[] arr1, int[] arr2) {
        int[] result = new int[arr1.length + arr2.length];
        int i = 0;
        for (int k : arr1) {
            result[i] = k;
            i++;
        }
        for (int k : arr2) {
            result[i] = k;
            i++;
            System.out.println(Arrays.toString(result));
        }
    }
```

table

a	b	Linked List	Array
10	100	0	1
100	100	1	1
1000	100	2	9
10000	100	17	7
100000	100	273	61

Table 1: This is the time out put of append two lists the list 1 is not fixed and the list 2 is fixed. And append two arrays

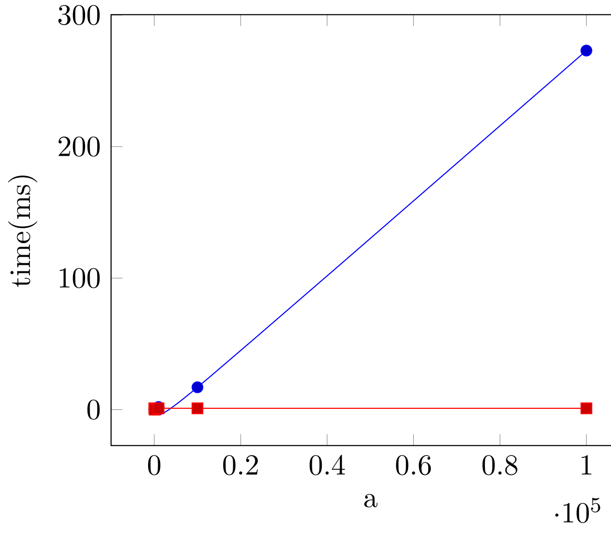
table

a	b	Linked List	Array
100	10	1	1
100	100	0	1
100	1000	1	10
100	10000	1	14
100	100000	1	195

Table 2: This is the time out put of append two lists the list 1 is fixed and the list 2 is not fixed. And append two arrays

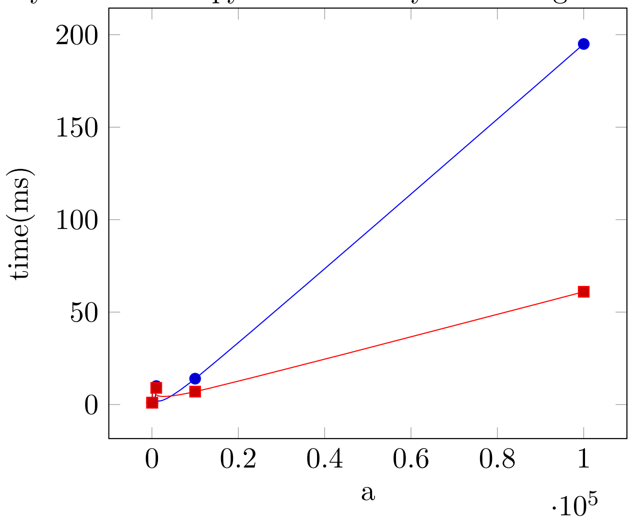
Graph: Linked List

When the list 1 is not fixed and it is growing up and the list 2 is fixed so the complexity time to this is o(n) because it must go through all the elements to append the another list so it takes n time. But when the the list 1 is fixed and the list 2 is not fixed so the complexity time is o(1) because it takes O(n) to find the right position and then the list 1 has the same length so it always going to be O(1).



Graph: Array

When the list 1 is not fixed and it is growing up and the list 2 is fixed so the complexity time to this is o(n) and also when the the list 1 is fixed and the list 2 is not fixed the complexity time is o(n) because it always makes a copy of both arrays returning a new array so it takes o(n).



Compare between Linked List and Array:

I change my benchmark of the linked list so that it shows the cost of building a list of 1000 items and benchmark of the array of the 1000 elements. Time to array was 20 ms. Time to linked list 148 ms. In conclusion array is faster than linked list because when we create an element in the array we just need one place to this element but when we create a node we need two places on for element and one for the reference and it takes more time.

Stack Linked List:

In the stack linked list I wrote three important functions one to push the nodes in to the stack and one to pop the nodes from the stack and one to peek and one to display the stack. In the push first create new node temp and allocate memory then check if stack is full then inserting an element that is going to be the top of stack then put top reference into temp link then update top reference. The peek function that function to return top element in a stack. Then pop function to pop top element from the stack first check for stack is empty then update the top pointer to point to the next node. In the linked list stack implementation the complexity time is O(1), so it is constant because every time we push an element, we'll perform exactly the same number of operations, add element, and change two pointers.

Stack dynamic Array:

In the dynamic array stack the complexity time of the push is O(nlogn) because every time we are going to do the push operation the algorithm needs to shift all the items in the array in order to add the new element to the beginning of the array.