MAHARISHI INTERNATIONAL UNIVERSITY

Assignment Three

NOVEMBER 2, 2020

KASAHUN TEHONE
IDNO: 111705

Assignment 3

A. Design a pseudo code algorithm to take a Sequence and remove all duplicate elements from the Sequence. Is the algorithm the same for both a List and a Sequence? Explain. Analyze your algorithm twice, once assuming it is a Sequence and once assuming it is a List. Which ADT is a better choice for this problem? Implement your choice in JavaScript.

Solution

It is not the same for list and sequence because the time complexity of list in O (n^2) and sequence takes O (n^3) time complexity because removing element from linked list take O(1) time but removing element from the sequence takes O(n^2) times.

Pseudo code and time complexity analysis

Remove duplicate using Recursive call based on sequence

Algorithm removeDup (L) ---- O(1)
p ← L.First() ---- O(n)
while(! L.last(p)) do ---- O(n)
e ← p.element() ----
Return Removehelper(e, L.after(p)) ---- O(n)

Algorithm remove helper (e, p, L) ---- O(1)
if L.isLast(p) then ----
if (e == p.element() then ---- O(n
L.remove(p) ----
break;
q ← L.after(p) ---- O(1)
if(e == p.element()) then ----
L.remove(p) ---- O(n)
remove helper (e, q, L) ----
T(n) = O(n)

Solution for Remove Duplicate based on List

Algorithm Dup (L) ---- O(1)
p ← L.first() ---- O(n)
while (¬L.isLast(p)) ---- O(n)
e ← p.element() ---- O(n^2)
while(¬L.isLast(p)) do ---- O(n^2)
q ← L.after(p) ---- O(n^2)
If (e == p.element()) ---- O(n^2)
L.Remove (p) ---- O(n^2)
p ← q ----
If (e == p.element() then
L.remove (p) ---- O(n)
T(n): O(n^2)

Source Code

```
function removeDuplicate(L) {
    let p = L.first()
    while (!L.isLast(p)) {
        let e = p.element()
        remove_Duplicate_Helper(e, L.after(p), L)
        if (!L.isLast(p)) {
            p = L.after(p)
        }

    }

}

function remove_Duplicate_Helper(e, p, L) {
    while (!L.isLast(p)) {
        let q = L.after(p)
        if (e === p.element()) {
            L.remove(p)}
             p = q

    }


    if (e === p.element()) {
        L.remove(p)}

}

console.log("before duplicate is not removed")
tst2.print()

console.log("after duplicate is removed is removed");

removeDuplicate(tst2)
tst2.print()
```

B. Design an algorithm, is Permutation (A, B) that takes two sequences A and B and Determines whether or not they are permutations of each other, i.e., same elements

But possibly occurring in a different order. Hint: A and B may contain duplicates,

Thus if A contains three x's, then B must also contain exactly three x's.

What is the worst-case time complexity of your algorithm? Justify your answer.

Implement your algorithm in JavaScript using either the Sequence or the List

Program provided.

pseudo code

Total time complexity we be O(n^2)

Recursive way - list based sorted

Algorithm removeDuplicate (L)
p ← L.First()
while (!L.isLast(p)) do
e ← p.element()
removeHelp (e, L.after(p))
IF (!L.isLast(p)) then
p ← L.after(p)

Algorithm removeHelp (e, p, L)
Output: Remove all occurrence of e in List that occur from position p to L
IF (L.isLast then
if e == p.element(p)
L.remove(p)

else
q ← L.after(p)
if e == p.element() then
L.remove(p)

removeHelp (e, q, L) // call

SOURCE CODE implementation \

```
function permutation_fun(A, B) {
    let p = A.first();
    if (A.size() != B.size()) {
        return false;
    }
    else {
        while (!A.isLast(p)) {
            let a = p.element();
            let q = B.first();
            let haselement = false;
            if (!(a == q.element())) {
                while (!B.isLast(q)) {
                    q = B.after(q);
                    let b = q.element();
                    if (a == b) {
                        haselement = true;
                        break;
                    }
                    // if (!haselement) {
                    //     return false;
                    // }


                }

            }
            else {
                haselement = true;
                p = A.after(p);
            }




        }

    }



}
```

```
let A = new Sequence(5)
console.log("a data created")
A.insertFirst(12)
A.insertLast(22)
A.insertBefore(A.first(), 122)
A.insertBefore(A.first(), 1233)
A.insertBefore(A.first(),234)
//A.insertBefore(A.first(),123)
A.print();




let B = new Sequence(5);
console.log("b data created")
B.insertFirst(40)
B.insertLast(2000)
B.insertAfter(B.first(), 120)
B.insertAfter(B.first(), 150)
B.insertAfter(B.first(), 1900)
//B.insertBefore(A.first(),123)

B.print()
// let kar=new permutation ()
console.log(permutation_fun(A,B))
```