Linked List 2

Friday, 14 May 2021 8:05 PM

Q1: 3 → 6 → 2 → 5 → 9 → 4

output: 4 9 5 2 6 3

Solution:

Iterate on Linked List & insert data in a vector/array } step1:

| 3 | 6 | 2 | 5 | 9 | 4 | } step2:

Iterate from right to left & print

TC: O(N) SC: O(N)

Recursion

Steps:

Assumption: print the linked list in reverse order

Main logic:

Base Condn: if head == NULL : stop:

```
void print R(Node *head){
    if(head == NULL) return
    print R(head->next);
    cout << head->data;
}
```

Main Logic

print R with 3

7 → [3 → 2 → 9 → 10] as per assumption

{10 9 2 3} 7

if we print 7

Final output: 10 9 2 3 7

TC: O(N)

Recursive Relation:

T(N) = T(N-1) + O(1)

→ Generalized Expression:

= T(N-k) + O(k)

k = N : T(0) + O(N)

O(1)

: O(N)

k: Tree Depth, Space Complexity: O(N)

2Q) Given linked list in sorted order, & (x) Insert x such that order is entire SLL in sorted

Ex1: 1 → 3 → 5 → 6 → 14 → 15 → 18, x = 9

Ex2: 3 → 4 → 7 → 12 → 18, x = 14

Ex3: -4 → 3 → 5 → 7 → 10 → 18 → 25, x = 6

```
Node* Insert(Node *h, int x){
    Node *t = new Node(x);
    Node *tmp = h;
    Edge Case 1
    if(h == NULL || h->data > x) { t->next = h; return t }
    while(h->next != NULL && h->next->data < x)
        h = h->next
    Edge Case 2
    t->next = h->next
    h->next = t;
    return tmp;
}
```

Debug1: x = 7, 3 → 4 → 6, BTE

h->next->data < 7 : h = h->next

h->next->data < 7 : h = h->next

h->next->data < 7 } BTE run time error

NULL->data

Before we access h->next->data we need to check if h->next != NULL

Debug2: At at start

(Avoid using extra space)

1) Can I use BS & find first element less than x, * [In case of linked list not possible]

2) Iterate on LinkedList & stop at a node which is ≥ x. (Here need to get prev Node)

→ We will get prev Node & update it only fly (2-Pointers)

Note:

Before we access h->data or h->next } Runtime

we need to check if h != NULL

Before we access h->next->data, h->next->next

we need to check if h->next != NULL

3rd: Given an SLL & x Delete all occurence of x

Ex1: 8 → 6 → 3 → 9 → 3 → 3 → 3 → 4 → 5 → 3 → 3, Delete

x = 3 : 8 → 6 → 9 → 4 → 5

Ex2: 8 → 3 → 9 → 3 → 9

x = 9 : 8 → 3 → 3

No extra space:

```
Node* Deleteall(Node *h, int x){
    if(h == NULL) return h
    Node *tmp = h;
    while(h->next != NULL){
        if(h->next->data == x){
            h->next = h->next->next
        }
        else{
            h = h->next
        }
    }
    if(tmp->data == x)
        return tmp->next
    else return tmp
}
```

Ex: 3 → 3 → 3 → 8 → 9 → 3 → 3

output: 8 9

output based on code: 3 8 9

After code, if x exists at will only be at starting node

```
Node* deleteall(Node *h, int x){
    Fn: Given a LinkedList delete all occurence of x & return headnode of new LinkedList
    if(h == NULL) return NULL;
    if(h->data == x){
        return deleteall(h->next, x)   // Head of new LinkedList
    }
    else{
        h->next = deleteall(h->next, x)
        return h;
    }
}
```

Ex: x = 3, 3 → 8 9 3 4 3

8 → 9 → 4, return 8

Ex: 2 8 9 3 3 5

3 h: 8 → 9 → 5

4) Given a sorted SSLL, delete repeated Elements

Ex: 3 → 3 → 3 → 5 → 5 → 5 → 5 → 8 → 8

output: 3 → 5 → 8

Ex2: 3 → 3 → 6 → 9 → 9 → 9 → 10

output: 3 → 6 → 9 → 10

Why sorting is important?

: Because of sorting grouping happens

: Adjacent things come together

```
void deleteRep(Node *h){
    if(h == NULL) return;
    while(h->next != NULL)   // till we reach last Node
        if(h->data == h->next->data){
            h->next = h->next->next
        }
        else{
            h = h->next;
        }
}
```

5th: Given a LinkedList reverse the Linked List & return new headNode

Ex: h: 7 → 3 → 2 → 9 → 10

h: 10 → 9 → 2 → 3 → 7

Sol: Can be solved with extra space. TC: O(N) SC: O(N)

Approach:

h = NULL

h2: 10 → 9 → 2 → 3 → 7 → NULL

Idea: If we add nodes from start, then linked list is reversed

```
h2 = NULL
while(h != NULL){
    tmp = h;
    h = h->next
    tmp->next = h2
    h2 = tmp
}
return h2;
```

TC: O(N), SC: O(1)

Very Helpful

```
Node* LLRR(Node *h){
    if(h == NULL || h->next == NULL) return h
    Assumption: Reverse the linked list & return new headNode
    Node (h2) = LLRR(h->next)
    h->next->next = h
    h->next = NULL
    return h2;
}
```

Ex: h: 3 → NULL

Node h2 = LLRR(NULL)

h2 = NULL

h->next->next = h

NULL

h: 7 → 8 → 4 → 2 → 1 → 10 → 11, return 11

h->next (Reverse the linked list & return head 11)

h->next->next = h

8 7

6th) Given a linked list reverse the first k Nodes

Ex: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9, k = 4 : 4 → 3 → 2 → 1 → 5 → 6 → 7 → 8 → 9

Ex2: 1 → 2 → 3, k = 4 : 3 → 2 → 1

Ex: 5 → 6 → 7 → 8 → 9, k = 4 : 4 → 3 → 2 → 1 → NULL

h2, tmp2

```
h2 = NULL;   tmp2 = h;
int cnt = 0;
while(h != NULL && cnt < k){
    tmp = h
    h = h->next;
    tmp->next = h2
    h2 = tmp;
    cnt++;
}
tmp2->next = h
```

7th) Given a linked list reverse all k groups

Ex: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8, k = 4 : 4 → 3 → 2 → 1 → 8 → 7 → 6 → 5

k = 3 : 3 → 2 → 1 → 6 → 5 → 4 → 8 → 7

If k = 2 :

output: 2 1 4 3 6 5 8 7

Swap Adjacent Nodes

```
Node* reverseK(Node *h, int k){
    if(h == NULL) return h;
    Ass: Reverse all k groups & return head Node
    h2 = NULL;   tmp2 = h;
    int cnt = 0;
    while(h != NULL && cnt < k){
        tmp = h
        h = h->next;
        tmp->next = h2
        h2 = tmp;
        cnt++;
    }
    tmp2->next = reverseK(h, k)
    return h2;
}
```

1 → 2 → 3 → 4 → 5 → 6 → 7 → 8

3 → 2 → 1 → 6 → 5 → 4 → 8 → 7

TC: O(N) SC: O(N/k)