Linked List - Problems

Relevel
by Unacademy

Problems to be Covered

Problem-1: Sort the triplet
Problem-2: Swap the Pair
Problem-3: Intersection in Linked Lists
Problem-4: Linked List to BST
Problem-5: K-Reversing

Problem-1 Sort the triplet

Given a linked list with nodes having values, 0,1 and 2. Your task would be to sort the linked list i.e. the sorted list should contain all the 0 data nodes to the head of the linked list, 2 towards the tail of the linked list and 1 in the middle of the list.

Input:
1,2,2,2,1,1,0,0,1,2

Output:
0,0,1,1,1,1,2,2,2,2

Expected:
Time Complexity: O(N)
Auxiliary Space Complexity: O(1)

Approach:
Given the linked list, we can keep track of the count of each data in the linked list and then update the linked list accordingly

1. Count occurrence of 0,1 and 2 in the linked list
2. Update the first zeroCount nodes with value 0
3. Update the next oneCount of nodes with value 1
4. Update the last twoCount of nodes with value 2

Code:

```
segregate(){
        let sp = this.head;
        let zeroCount =0;
        let oneCount = 0;
        let twoCount = 0;
        //Count the number of each item
        while(sp) {
            if(sp.data === 0)
zeroCount++;
            else if(sp.data === 1)
oneCount++;
            else twoCount++;
            sp = sp.next;
        }
        //Set the pointer to head node
        let curr = this.head;
        //Update the first zeroCount
nodes with value 0
while(zeroCount>0) {
            curr.data = 0;
            curr = curr.next;
            zeroCount--;
        }
        //Update the next oneCount
nodes with value 1
          while(oneCount>0) {
            curr.data = 1;
            curr = curr.next;
            oneCount--;
        }
        //Update the remaining twoCount
node with value 2
            while(twoCount>0) {
            curr.data = 2;
            curr = curr.next;
            twoCount--;
        }
    }
```

Problem-2: Swap the Pair

Write a javascript program to swap the consecutive nodes of the singly linked list instead of the data.

Input:
1,2,3,4,5,6

Output:
2,1,4,3,6,5

Explaination:
For the given input 1,2,3,4,5,6 the pair will be formed as
(1,2) (3,4) (5,6) and the swap of these pair will be
(2,1) (4,3) (6,5) hence the result will be
2,1,4,3,6,5

Expected:
Time Complexity: O(N)
Auxiliary Space Complexity: O(1)

Approach:
Solution to this problem is pretty straight forward, Iterate over the linked list, take two consecutive nodes at a time and perform swapping by updating the next values of the nodes.

Code:

```
pairWiseSwap()
         {
         //Check if there exits atleast
a pair of node
        if ( !this.head ||
!this.head.next ) {
            return this.head;
        }

        let currentNode = this.head;
        let prevNode = null;
while ( currentNode && currentNode.next ) {
               //Select the nodes to swap
            const next = currentNode.next;
            const nextToNext = next.next;
                              //Change the
address of the selected nodes
            next.next = currentNode;
            currentNode.next = nextToNext;
            //Update the pointer

            if ( prevNode ) {
                prevNode.next = next;
            } else {
                this.head = next;
            }
            prevNode = currentNode;
            currentNode = nextToNext;
        }

    }
```

Problem-3: Intersection in Linked Lists

Given two linked list, with unique nodes value independently, you need to write a program that take in two list and return the linked with the intersecting nodes.
Note: The order of the nodes in the intersecting list should be maintained as that of the actual list 1.

Input:
1,5,8,9,12
1,4,5,6,7,9,10,12

Output:
1,5,9,12

Expected:
Time Complexity: O(m+n)
Auxiliary Space Complexity: O(m+n)

Approach:
Iterate over the nodes of the first linked list, and at each node, check if the value exists in the second node or not. If yes then create a new node and add it to the resultant list.

Code:

```
function isPresent(head, data)
    {
        let temp = head;
        while (temp != null) {
            if (temp.data == data)
                return true;
            temp = temp.next;
        }
        return false;
    }

function
getIntersection(head1,head2,result)
    {
        let temp = head1;
```

```
    while(temp!=null){
          if
(isPresent(head2,temp.data)){
                let node = new
Node(temp.data)

result.addNode(node);
           }
          temp=temp.next;
        }
    }
```

Output:

```
1,5,9,12
```

Problem-4: Linked List to BST

Given a sorted linked list, say L, your task is to convert it into a balanced binary tree keeping the time complexity as O(n).
Note: In a balance binary tree the difference between the height of the left and the right subtree is never greater than 1.

Input:
5 4 1 0 -3 -4 -5

Output:
0 4 5 1 -4 -3 -5

Explaination:
On converting the linked list 5 4 1 0 -3 -4 -5 to binary tree, we will get:

Expected:
Time Complexity: O(n)
Auxiliary Space Complexity: O(n)

Approach:

We can perform the construction of the BST by building the leaf and then going towards the root. So, what we need to is that we will iterate the linked list, and at position we will take the node and insert it in the BST so as to make sure the time complexity is O(n).

Overall the steps that can be followed to solve this problem would be:

1. Count the number of nodes in linked list
2. Take n/2 nodes and construct the left subtree
3. Once the left subtree is created then allocate the memory for the creation of root.
4. Now construct the right subtree
5. Link the right subtree with the root

Code:

```
class Node{
    constructor(data){
        this.data = data;
        this.next = null;
    }
}

class TreeNode{
         constructor(data){
        this.data = data;
        this.right = null;
        this.left = null;
    }

}
```

```
class LinkedList{
    constructor(head=null){
    this.head=null
     this.length=0;
  }
addNode(node){
         if(this.head==null){
         this.head=node;
               return;
         }
         let temp = this.head;
         while(temp.next!=null)
         temp=temp.next;
         temp.next = node;
      this.length+=1;
    }
```

```
printList(){
      let data= []
      let temp = this.head;
      while(temp!=null){
        data.push(temp.data);
        temp = temp.next;
        }
      console.log(...data);
    }
}
```

```
function sortedListToBSTRecur(n)
{
    if (n <= 0)
          return null;
    var left = sortedListToBSTRecur(parseInt(n /
2));
    var root = new TreeNode(l1.head.data);
    root.left = left;
    l1.head = l1.head.next;
    root.right = sortedListToBSTRecur(n -
parseInt(n / 2) - 1);
    return root;
}

function preOrder(node)
{
    if (node == null)
        return;
    console.log(node.data + " ");
    preOrder(node.left);
    preOrder(node.right);
}
```

```
values=[5,4,1,0,-3,-4,-5]

let l1 = new LinkedList();
for(let
idx=0;idx<values.length;idx++
){
    let node = new
Node(values[idx]);
    l1.addNode(node);
}

bst =
sortedListToBSTRecur(l1.lengt
h);
preOrder(bst)
```

Problem-5: K-Reversing

Given a linked list, you need to reverse the nodes in the linked list for every k group. K is an integer which determine the group size. All the nodes in the group will be reversed.

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

Output:
3,2,1,6,5,4,8,7

Explaination:
Group of size 3 for the list 1,2,3,4,5,6,7,8 is
(1,2,3),(4,5,6) & (7,8) and the reverse of these groups will be:
(3,2,1), (6,5,4) & (8,7)

Expected:
Time Complexity: O(n)
Auxiliary Space Complexity: O(n/k)

Approach:
To solve the problem we can take a temporary stack to store the k nodes, So, idea would be to iterate the linked list and keep appending the node to the stack till the time k nodes are encountered or the list comes to an end.
Then take the nodes in the stack and update the linked list it.

Code:

```
kReverse(k) {
    let tempStack = [];
    let current = this.head;
    let prev = null;
    while (current != null) {
        let count = 0;
        while (current != null && count
< k) {
            tempStack.push(current);
            current = current.next;
            count++;
        }
        while (tempStack.length > 0) {
            if (prev == null) {
                prev = tempStack.pop();
                this.head = prev;
            } else {
                prev.next =
tempStack.pop();
                prev = prev.next;
            }
        }
    }
    prev.next = null;
}
```

Thank You!