# In-place conversion of Sorted DLL to Balanced BST

Given a Doubly Linked List which has data members sorted in ascending order. Construct a Balanced Binary Search Tree which has same data members as the given Doubly Linked List. The tree must be constructed in-place (No new node should be allocated for tree conversion)

Examples:

```
Input:  Doubly Linked List 1 <--> 2 <--> 3
Output: A Balanced BST
     2
   /  \
  1    3


Input: Doubly Linked List 1 <--> 2 <-->3 <--> 4 <-->5 <--> 6 <--> 7
Output: A Balanced BST
        4
      /   \
     2     6
   /  \   / \
  1   3  4   7

Input: Doubly Linked List 1 <--> 2 <--> 3 <--> 4
Output: A Balanced BST
      3
    /  \
   2    4
 /
1

Input:  Doubly Linked List 1 <--> 2 <--> 3 <--> 4 <--> 5 <--> 6
Output: A Balanced BST
      4
    /   \
   2     6
 /  \   /
1   3  5
```

The Doubly Linked List conversion is very much similar to this Singly Linked List problem and the method 1 is exactly same as the method 1 of previous post. Method 2 is also almost same. The only difference in method 2 is, instead of allocating new nodes for BST, we reuse same DLL nodes. We use prev pointer as left and next pointer as right.

**Method 1 (Simple)**

Following is a simple algorithm where we first find the middle node of list and make it root of the tree to be constructed.

```
1) Get the Middle of the linked list and make it root.
2) Recursively do same for left half and right half.
       a) Get the middle of left half and make it left child of the root
          created in step 1.
       b) Get the middle of right half and make it right child of the
          root created in step 1.
```

Time complexity: O(nLogn) where n is the number of nodes in Linked List.

**Method 2 (Tricky)**

The method 1 constructs the tree from root to leaves. In this method, we construct from leaves to root. The idea is to insert nodes in BST in the same order as the appear in Doubly Linked List, so that the tree can be constructed in O(n) time complexity. We first count the number of nodes in the given Linked List. Let the count be n. After counting nodes, we take left n/2 nodes and recursively construct the left subtree. After left subtree is constructed, we assign middle node to root and link the left subtree with root. Finally, we recursively construct the right subtree and link it with root.

While constructing the BST, we also keep moving the list head pointer to next so that we have the appropriate pointer in each recursive call.

Following is C implementation of method 2. The main code which creates Balanced BST is highlighted.

```
#include<stdio.h>
#include<stdlib.h>

/* A Doubly Linked List node that will also be used as a tree node */
struct Node
{
    int data;

    // For tree, next pointer can be used as right subtree pointer
    struct Node* next;

    // For tree, prev pointer can be used as left subtree pointer
    struct Node* prev;
};


// A utility function to count nodes in a Linked List
int countNodes(struct Node *head);

struct Node* sortedListToBSTRecur(struct Node **head_ref, int n);

/* This function counts the number of nodes in Linked List and then calls
```

```
   sortedListToBSTRecur() to construct BST */
struct Node* sortedListToBST(struct Node *head)
{
    /*Count the number of nodes in Linked List */
    int n = countNodes(head);

    /* Construct BST */
    return sortedListToBSTRecur(&head, n);
}

/* The main function that constructs balanced BST and returns root of it.
       head_ref -->  Pointer to pointer to head node of Doubly linked list
       n  --> No. of nodes in the Doubly Linked List */
struct Node* sortedListToBSTRecur(struct Node **head_ref, int n)
{
    /* Base Case */
    if (n <= 0)
        return NULL;

    /* Recursively construct the left subtree */
    struct Node *left = sortedListToBSTRecur(head_ref, n/2);

    /* head_ref now refers to middle node, make middle node as root of BST*/
    struct Node *root = *head_ref;

    // Set pointer to left subtree
    root->prev = left;

    /* Change head pointer of Linked List for parent recursive calls */
    *head_ref = (*head_ref)->next;

    /* Recursively construct the right subtree and link it with root
      The number of nodes in right subtree  is total nodes - nodes in
      left subtree - 1 (for root) */
    root->next = sortedListToBSTRecur(head_ref, n-n/2-1);

    return root;
}

/* UTILITY FUNCTIONS */
/* A utility function that returns count of nodes in a given Linked List */
int countNodes(struct Node *head)
{
    int count = 0;
    struct Node *temp = head;
    while(temp)
    {
        temp = temp->next;
        count++;
    }
    return count;
}

/* Function to insert a node at the beginging of the Doubly Linked List */
void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node =
            (struct Node*) malloc(sizeof(struct Node));

    /* put in the data  */
    new_node->data  = new_data;

    /* since we are adding at the begining,
      prev is always NULL */
```

```
    new_node->prev = NULL;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* change prev of head node to new node */
    if((*head_ref) !=  NULL)
      (*head_ref)->prev = new_node ;

    /* move the head to point to the new node */
    (*head_ref)    = new_node;
}

/* Function to print nodes in a given linked list */
void printList(struct Node *node)
{
    while(node!=NULL)
    {
        printf("%d ", node->data);
        node = node->next;
    }
}

/* A utility function to print preorder traversal of BST */
void preOrder(struct Node* node)
{
    if (node == NULL)
        return;
    printf("%d ", node->data);
    preOrder(node->prev);
    preOrder(node->next);
}

/* Drier program to test above functions*/
int main()
{
    /* Start with the empty list */
    struct Node* head = NULL;

    /* Let us create a sorted linked list to test the functions
     Created linked list will be 7->6->5->4->3->2->1 */
    push(&head, 7);
    push(&head, 6);
    push(&head, 5);
    push(&head, 4);
    push(&head, 3);
    push(&head, 2);
    push(&head, 1);

    printf("\n Given Linked List ");
    printList(head);

    /* Convert List to BST */
    struct Node *root = sortedListToBST(head);
    printf("\n PreOrder Traversal of constructed BST ");
    preOrder(root);

    return 0;
}
```

Run on IDE

Time Complexity: O(n)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



48 Comments Category: Linked Lists

## Related Posts:

- Merge two sorted linked lists such that merged list is in reverse order
- Compare two strings represented as linked lists
- Rearrange a given linked list in-place.
- Sort a linked list that is sorted alternating ascending and descending orders?
- Select a Random Node from a Singly Linked List
- Merge Sort for Doubly Linked List
- Point to next higher value node in a linked list with an arbitrary pointer
- Swap nodes in a linked list without swapping data



Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.