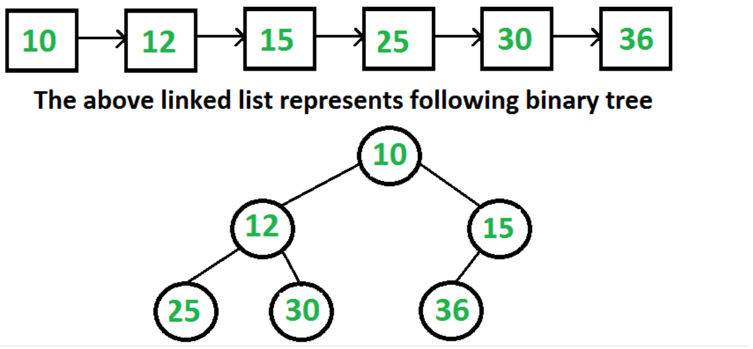
Construct Complete Binary Tree from its Linked List Representation

Given Linked List Representation of Complete Binary Tree, construct the Binary tree. A complete binary tree can be represented in an array in the following approach.

If root node is stored at index i, its left, and right children are stored at indices 2*i+1, 2*i+2 respectively.
Suppose tree is represented by a linked list in same way, how do we convert this into normal linked representation of binary tree where every node has data, left and right pointers? In the linked list representation, we cannot directly access the children of the current node unless we traverse the list.



We are mainly given level order traversal in sequential access form. We know head of linked list is always is root of the tree. We take the first node as root and we also know that the next two nodes are left and right children of root. So we know partial Binary Tree. The idea is to do Level order traversal of the partially built Binary Tree using queue and traverse the linked list at the same time. At every step, we take the parent node from queue, make next two nodes of linked list as children of the parent node, and enqueue the next two nodes to queue.

1. Create an empty queue.
2. Make the first node of the list as root, and enqueue it to the queue.
3. Until we reach the end of the list, do the following.
.........**a.** Dequeue one node from the queue. This is the current parent.
.........**b.** Traverse two nodes in the list, add them as children of the current parent.
.........**c.** Enqueue the two nodes into the queue.

Below is the code which implements the same in C++.

C++

```
// C++ program to create a Complete Binary tree from its Linked List
// Representation
#include <iostream>
#include <string>
#include <queue>
using namespace std;

// Linked list node
struct ListNode
{
    int data;
    ListNode* next;
};

// Binary tree node structure
struct BinaryTreeNode
{
    int data;
    BinaryTreeNode *left, *right;
};
```

```
// Function to insert a node at the beginning of the Linked List
void push(struct ListNode** head_ref, int new_data)
{
    // allocate node and assign data
    struct ListNode* new_node = new ListNode;
    new_node->data = new_data;

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

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

// method to create a new binary tree node from the given data
BinaryTreeNode* newBinaryTreeNode(int data)
{
    BinaryTreeNode *temp = new BinaryTreeNode;
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// converts a given linked list representing a complete binary tree into the
// linked representation of binary tree.
void convertList2Binary(ListNode *head, BinaryTreeNode* &root)
{
    // queue to store the parent nodes
    queue<BinaryTreeNode *> q;

    // Base Case
    if (head == NULL)
    {
        root = NULL; // Note that root is passed by reference
        return;
    }

    // 1.) The first node is always the root node, and add it to the queue
    root = newBinaryTreeNode(head->data);
    q.push(root);

    // advance the pointer to the next node
    head = head->next;

    // until the end of linked list is reached, do the following steps
    while (head)
    {
        // 2.a) take the parent node from the q and remove it from q
        BinaryTreeNode* parent = q.front();
        q.pop();

        // 2.c) take next two nodes from the linked list. We will add
        // them as children of the current parent node in step 2.b. Push them
        // into the queue so that they will be parents to the future nodes
        BinaryTreeNode *leftChild = NULL, *rightChild = NULL;
        leftChild = newBinaryTreeNode(head->data);
        q.push(leftChild);
        head = head->next;
        if (head)
        {
            rightChild = newBinaryTreeNode(head->data);
            q.push(rightChild);
            head = head->next;
        }

        // 2.b) assign the left and right children of parent
        parent->left = leftChild;
        parent->right = rightChild;
    }
}

// Utility function to traverse the binary tree after conversion
void inorderTraversal(BinaryTreeNode* root)
void inorderTraversal(BinaryTreeNode* root)
{
    if (root)
    {
        inorderTraversal( root->left );
        cout << root->data << " ";
        inorderTraversal( root->right );
    }
}

// Driver program to test above functions
int main()
{
    // create a linked list shown in above diagram
    struct ListNode* head = NULL;
    push(&head, 36);  /* Last node of Linked List */
    push(&head, 30);
    push(&head, 25);
    push(&head, 15);
    push(&head, 12);
    push(&head, 10); /* First node of Linked List */

    BinaryTreeNode *root;
    convertList2Binary(head, root);

    cout << "Inorder Traversal of the constructed Binary Tree is: \n";
    inorderTraversal(root);
    return 0;
}
```

Java

```
// Java program to create complete Binary Tree from its Linked List
// representation

// importing necessary classes
import java.util.*;

// A linked list node
class ListNode
{
    int data;
    ListNode next;
    ListNode(int d)
    {
        data = d;
        next = null;
    }
}

// A binary tree node
class BinaryTreeNode
{
    int data;
    BinaryTreeNode left, right = null;
    BinaryTreeNode(int data)
    {
        this.data = data;
        left = right = null;
    }
}

class BinaryTree
{
    ListNode head;
    BinaryTreeNode root;

    // Function to insert a node at the beginning of
    // the Linked List
    void push(int new_data)
    {
        // allocate node and assign data
        ListNode new_node = new ListNode(new_data);

        // link the old list off the new node
        new_node.next = head;

        // move the head to point to the new node
        head = new_node;
    }

    // converts a given linked list representing a
    // complete binary tree into the linked
    // representation of binary tree.
    BinaryTreeNode convertList2Binary(BinaryTreeNode node)
    {
        // queue to store the parent nodes
        Queue<BinaryTreeNode> q =
                       new LinkedList<BinaryTreeNode>();

        // Base Case
        if (head == null)
        {
            node = null;
            return null;
        }

        // 1.) The first node is always the root node, and
        //     add it to the queue
        node = new BinaryTreeNode(head.data);
        q.add(node);

        // advance the pointer to the next node
        head = head.next;

        // until the end of linked list is reached, do the
        // following steps
        while (head != null)
        {
            // 2.a) take the parent node from the q and
            //      remove it from q
            BinaryTreeNode parent = q.peek();
            BinaryTreeNode pp = q.poll();

            // 2.c) take next two nodes from the linked list.
            // We will add them as children of the current
            // parent node in step 2.b. Push them into the
            // queue so that they will be parents to the
            // future nodes
            BinaryTreeNode leftChild = null, rightChild = null;
            leftChild = new BinaryTreeNode(head.data);
            q.add(leftChild);
            head = head.next;
            if (head != null)
            {
                rightChild = new BinaryTreeNode(head.data);
                q.add(rightChild);
                head = head.next;
            }

            // 2.b) assign the left and right children of
            //      parent
            parent.left = leftChild;
            parent.right = rightChild;
        }

        return node;
    }

    // Utility function to traverse the binary tree
    // after conversion
    void inorderTraversal(BinaryTreeNode node)
    {
        if (node != null)
        if (node != null)
        {
            inorderTraversal(node.left);
            System.out.print(node.data + " ");
            inorderTraversal(node.right);
        }
    }

    // Driver program to test above functions
    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();
        tree.push(36); /* Last node of Linked List */
        tree.push(30);
        tree.push(25);
        tree.push(15);
        tree.push(12);
        tree.push(10); /* First node of Linked List */
        BinaryTreeNode node = tree.convertList2Binary(tree.root);

        System.out.println("Inorder Traversal of the"+
                        " constructed Binary Tree is:");
        tree.inorderTraversal(node);
    }
}
// This code has been contributed by Mayank Jaiswal
```

Python

```
# Python program to create a Complete Binary Tree from
# its linked list representation

# Linked List node
class ListNode:

        # Constructor to create a new node
        def __init__(self, data):
            self.data = data
            self.next = None

# Binary Tree Node structure
class BinaryTreeNode:

    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# Class to convert the linked list to Binary Tree
class Conversion:

    # Constructor for storing head of linked list
    # and root for the Binary Tree
    def __init__(self, data = None):
        self.head = None
        self.root = None

    def push(self, new_data):

        # Creating a new linked list node and storing data
        new_node = ListNode(new_data)

        # Make next of new node as head
        new_node.next = self.head

        # Move the head to point to new node
        self.head = new_node

    def convertList2Binary(self):
        # Queue to store the parent nodes
        q = []

        # Base Case
        if self.head is None:
            self.root = None
            return

        # 1.) The first node is always the root node,
        # and add it to the queue
        self.root = BinaryTreeNode(self.head.data)
        q.append(self.root)

        # Advance the pointer to the next node
        self.head = self.head.next

        # Until th end of linked list is reached, do:
        while(self.head):

            # 2.a) Take the parent node from the q and
            # and remove it from q
            parent = q.pop(0) # Front of queue

            # 2.c) Take next two nodes from the linked list.
            # We will add them as children of the current
            # parent node in step 2.b.
            # Push them into the queue so that they will be
            # parent to the future node
            leftChild= None
            rightChild = None

            leftChild = BinaryTreeNode(self.head.data)
            q.append(leftChild)
            self.head = self.head.next
            if(self.head):
                rightChild = BinaryTreeNode(self.head.data)
                q.append(rightChild)
                self.head = self.head.next

            #2.b) Assign the left and right children of parent
            parent.left = leftChild
            parent.right = rightChild

    def inorderTraversal(self, root):
        if(root):
            self.inorderTraversal(root.left)
            print root.data,
            self.inorderTraversal(root.right)

# Driver Program to test above function

# Object of conversion class
conv = Conversion()
conv.push(36)
conv.push(30)
conv.push(25)
conv.push(15)
conv.push(12)
conv.push(10)

conv.convertList2Binary()

print "Inorder Traversal of the contructed Binary Tree is:"
conv.inorderTraversal(conv.root)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
Inorder Traversal of the constructed Binary Tree is:
25 12 30 10 36 15
```

Time Complexity: Time complexity of the above solution is O(n) where n is the number of nodes.