CPSC 319

Data Structures, Algorithms and Their Application

Department of Computer Science

Created by:

Fahim Anzum

Winter 2020

Course Information

- **Course website:**

All course materials, such as lecture slides, important dates, assignments and exercises can be found on the course website, which is located at http://pages.cpsc.ucalgary.ca/~hudsonj/CPSC319W20/

- **For Assignment and Grading:**

D2L will be used for submitting assignments and reporting grades.

https://d2l.ucalgary.ca/d2l/home

- **Tutorial Contents:**

Will be uploaded in the following site:

https://sites.google.com/view/fahimanzum/courses

N-ary Tree

- If a tree is a rooted tree in which each node has no more than N children, it is called N-ary tree
- A binary tree is a special form of a N-ary tree

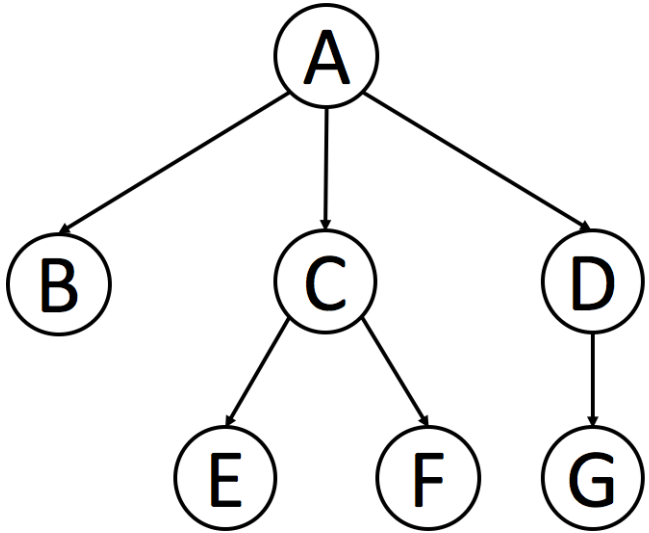
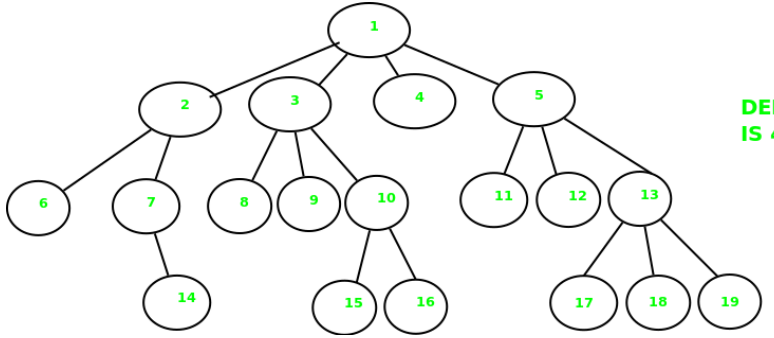


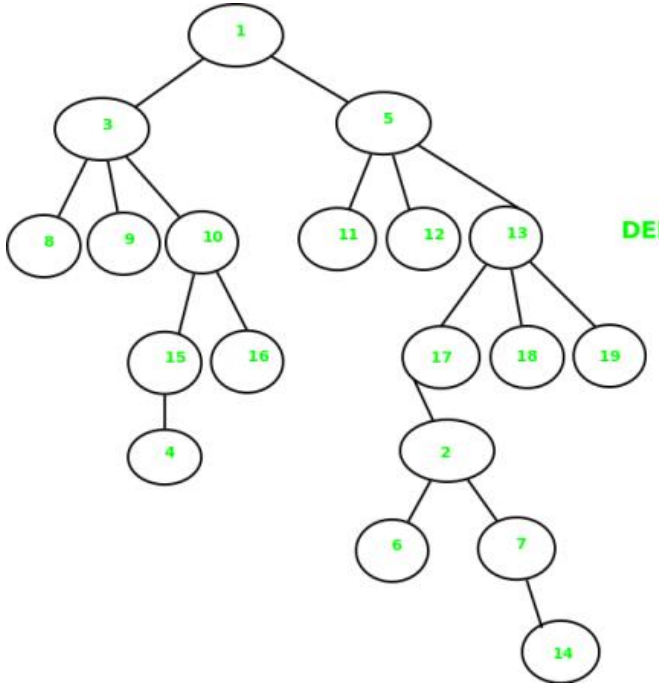
Figure: 3-ary tree

Example

- Given an N-ary tree, find depth of the tree.
- N-ary tree can be traversed just like a normal tree.
- We consider all children of a given node and recursively call that function on every node.



DEPTH OF THE N-ARY IS 4



DEPTH OF THE N-ARY IS 7

Example

- Given an N-ary tree, find depth of the tree.

```
class GFG
{
    //Structure of a node of an n-ary tree
    static class Node
    {
        char key;
        Vector<Node > child;
    };

    //Utility function to create a new tree node
    static Node newNode(int key)
    {
        Node temp = new Node();
        temp.key = (char) key;
        temp.child = new Vector<Node>();
        return temp;
    }

    //Function that will return the depth
    //of the tree
    static int depthOfTree(Node ptr)
    {
        // Base case
        if (ptr == null)
            return 0;

        // Check for all children and find
        // the maximum depth
        int maxdepth = 0;
        for (Node it : ptr.child)
            maxdepth = Math.max(maxdepth,
                                depthOfTree(it));

        return maxdepth + 1 ;
    }
```

Example

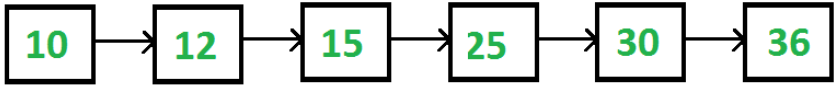
- Given an N-ary tree, find depth of the tree.

```
    //Driver Code
    public static void main(String[] args)
    {
        /* Let us create below tree
        *           A
        *        / / \ \
        *     B F D E
        *     / \ | /|\
        *     K J G C H I
        *     /\       \
        * N M          L
        */

        Node root = newNode('A');
        (root.child).add(newNode('B'));
        (root.child).add(newNode('F'));
        (root.child).add(newNode('D'));
        (root.child).add(newNode('E'));
        (root.child.get(0).child).add(newNode('K'));
        (root.child.get(0).child).add(newNode('J'));
        (root.child.get(2).child).add(newNode('G'));
        (root.child.get(3).child).add(newNode('C'));
        (root.child.get(3).child).add(newNode('H'));
        (root.child.get(3).child).add(newNode('I'));
        (root.child.get(0).child.get(0).child).add(newNode('N'));
        (root.child.get(0).child.get(0).child).add(newNode('M'));
        (root.child.get(3).child.get(2).child).add(newNode('L'));

        System.out.print(depthOfTree(root) + "\n");
    }
}
```

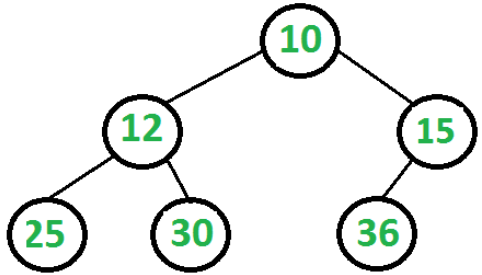
More Example

- Given, a linked list representation of complete binary tree
- We need to construct the binary tree

- If root node is stored at index i, its left and right children are stored at indices 2*i+1 and 2*i+2 respectively
- We consider all children of a given node and recursively call that function on every node.



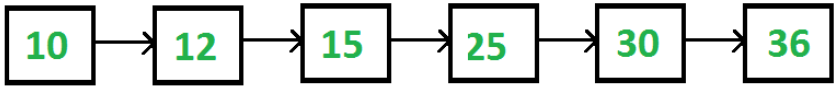
The above linked list represents following binary tree



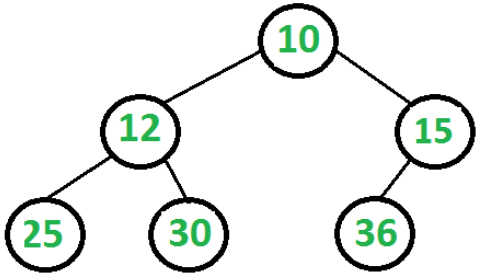
More Example

- Given, a linked list representation of complete binary tree
- We need to construct the binary tree

- 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



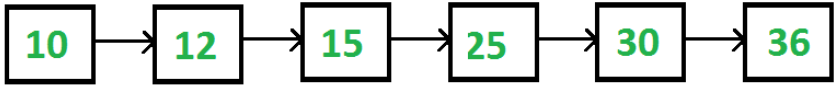
The above linked list represents following binary tree



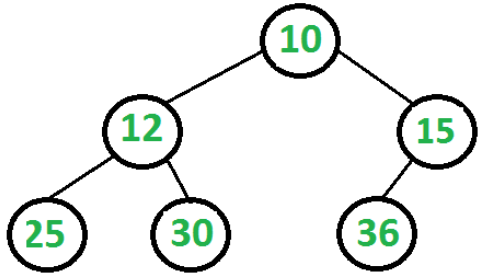
More Example

- Given, a linked list representation of complete binary tree
- We need to construct the 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



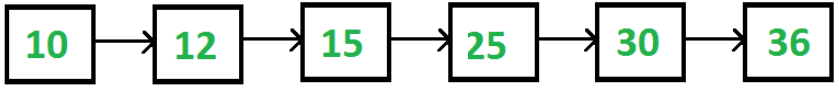
The above linked list represents following binary tree



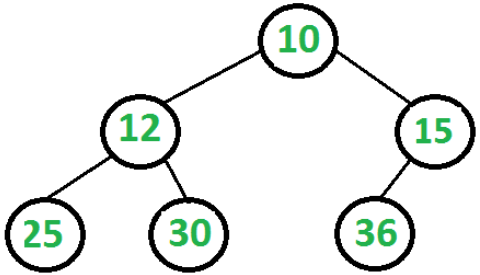
More Example

- Given, a linked list representation of complete binary tree
- We need to construct the binary tree

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.



The above linked list represents following binary tree



More Example

- Given, a linked list representation of complete binary tree
- We need to construct the binary tree

```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;
    }
}
```

More Example

- Given, a linked list representation of complete binary tree
- We need to construct the binary tree

```
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;
        }
```

More Example

- Given, a linked list representation of complete binary tree
- We need to construct the binary tree

```
// 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;
}
```

Program Output

Inorder Traversal of the constructed Binary Tree is:

25 12 30 10 36 15

```
        return node;
    }

    // Utility function to traverse the binary tree
    // after conversion
    void inorderTraversal(BinaryTreeNode node)
    {
        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);
    }
}
```

Reference

- https://leetcode.com/articles/introduction-to-n-ary-trees/
- https://www.geeksforgeeks.org/depth-n-ary-tree/
- https://www.geeksforgeeks.org/check-if-the-given-n-ary-tree-is-a-binary-tree/
- https://www.geeksforgeeks.org/given-linked-list-representation-of-complete-tree-convert-it-to-linked-representation/

UNIVERSITY OF CALGARY

Thank You

Fahim Anzum

Email: fahim.anzum@ucalgary.ca