Iterative Method to find Height of Binary Tree

There are two conventions to define height of Binary Tree

1) Number of nodes on longest path from root to the deepest node.

2) Number of edges on longest path from root to the deepest node.

In this post, the first convention is followed. For example, height of the below tree is 3.

Example Tree

Recursive method to find height of Binary Tree is discussed here. How to find height without recursion? We can use level order traversal to find height without recursion. The idea is to traverse level by level. Whenever move down to a level, increment height by 1 (height is initialized as 0). Count number of nodes at each level, stop traversing when count of nodes at next level is 0.

Following is detailed algorithm to find level order traversal using queue.

```
Create a queue.
Push root into the queue.
height = 0
Loop
        nodeCount = size of queue

        // If number of nodes at this level is 0, return height
        if nodeCount is 0
                return Height;
        else
                increase Height

        // Remove nodes of this level and add nodes of
        // next level
        while (nodeCount > 0)
                pop node from front
                push its children to queue
                decrease nodeCount
       // At this point, queue has nodes of next level
```

Following is the implementation of above algorithm.

C++

```
/* Program to find height of the tree by Iterative Method */
#include <iostream>
#include <queue>
using namespace std;

// A Binary Tree Node
struct node
{
    struct node *left;
    int data;
    struct node *right;
};

// Iterative method to find height of Bianry Tree
int treeHeight(node *root)
{
    // Base Case
    if (root == NULL)
        return 0;

    // Create an empty queue for level order tarversal
    queue<node *> q;

    // Enqueue Root and initialize height
    q.push(root);
    int height = 0;

    while (1)
    {
        // nodeCount (queue size) indicates number of nodes
        // at current lelvel.
        int nodeCount = q.size();
        if (nodeCount == 0)
            return height;

        height++;

        // Dequeue all nodes of current level and Enqueue all
        // nodes of next level
        while (nodeCount > 0)
        {
            node *node = q.front();
            q.pop();
            if (node->left != NULL)
                q.push(node->left);
            if (node->right != NULL)
                q.push(node->right);
```

```
            nodeCount--;
        }
    }
}

// Utility function to create a new tree node
node* newNode(int data)
{
    node *temp = new node;
    temp->data = data;
    temp->left = NULL;
    temp->right = NULL;
    return temp;
}

// Driver program to test above functions
int main()
{
    // Let us create binary tree shown in above diagram
    node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);

    cout << "Height of tree is " << treeHeight(root);
    return 0;
}
```

Java

```
// An iterative java program to find height of binary tree

import java.util.LinkedList;
import java.util.Queue;

// A binary tree node
class Node {

    int data;
    Node left, right;

    Node(int item) {
        data = item;
        left = right;
    }
}

class BinaryTree {
```

```
    static Node root;

    // Iterative method to find height of Bianry Tree
    int treeHeight(Node node) {
        // Base Case
        if (node == null) {
            return 0;
        }

        // Create an empty queue for level order tarversal
        Queue<Node> q = new LinkedList();

        // Enqueue Root and initialize height
        q.add(node);
        int height = 0;

        while (1 == 1) {

            // nodeCount (queue size) indicates number of nodes
            // at current lelvel.
            int nodeCount = q.size();
            if (nodeCount == 0) {
                return height;
            }

            height++;

            // Dequeue all nodes of current level and Enqueue all
            // nodes of next level
            while (nodeCount > 0) {
                Node newnode = q.peek();
                q.remove();
                if (newnode.left != null) {
                    q.add(newnode.left);
                }
                if (newnode.right != null) {
                    q.add(newnode.right);
                }
                nodeCount--;
            }
        }
    }

    // Driver program to test above functions
    public static void main(String args[]) {

        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
```

```
        System.out.println("Height of tree is " + tree.treeHeight(root));

    }
}

// This code has been contributed by Mayank Jaiswal
```

Python

```
# Program to find height of tree by Iteration Method

# A binary tree node
class Node:

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

# Iterative method to find height of Binary Tree
def treeHeight(root):

    # Base Case
    if root is None:
        return 0

    # Create a empty queue for level order traversal
    q = []

    # Enqueue Root and Initialize Height
    q.append(root)
    height = 0

    while(True):

        # nodeCount(queue size) indicates number of nodes
        # at current level
        nodeCount = len(q)
        if nodeCount == 0 :
            return height

        height += 1

        # Dequeue all nodes of current level and Enqueue
        # all nodes of next level
        while(nodeCount > 0):
            node = q[0]
```

```
            q.pop(0)
            if node.left is not None:
                q.append(node.left)
            if node.right is not None:
                q.append(node.right)

            nodeCount -= 1


# Driver program to test above function
# Let us create binary tree shown in above diagram
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)

print "Height of tree is", treeHeight(root)

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

Output:

```
Height of tree is 3
```

Time Complexity: O(n) where n is number of nodes in given binary tree.

This article is contributed by **Rahul Kumar**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

28 Comments Category: Queue Trees

Related Posts:

- Minimum time required to rot all oranges
- How to efficiently implement k Queues in a single array?
- An Interesting Method to Generate Binary Numbers from 1 to n
- Construct Complete Binary Tree from its Linked List Representation
- Find the first circular tour that visits all petrol pumps
- Implement Stack using Queues
- Find the largest multiple of 3
- Implement LRU Cache

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.
jeffwa
I'm confused what the decrementing the nodecount on the dequeue does when the while loop doesn't depend on it, and nodecount is then recalculated anyways. It seems to be unnecessary.
http://about.me/tg9963 GOPI GOPINATH
we had 2 while loops….second while loop depends on nodecount..its the end condition if nodecount==0
http://www.phoenixuser.blogspot.com GOPI GOPINATH
Here is the implementation of iterative method to find height of a binary tree with linked list queue

```
#include
#include
#include
struct Treenode
{
int data;
struct Treenode * left;
struct Treenode *right;
};
struct Treenode* newnode(int data)
{
struct Treenode* temp=(struct Treenode *)malloc(sizeof(struct Treenode));
temp->data=data;
temp->left=NULL;
temp->right=NULL;
return temp;
}
struct List
{
struct Treenode* node;
struct List *next;
};
struct queue
{
struct List * front;
struct List *rear;
};
struct queue* createqueue()
{
struct queue* ptr= (struct queue *)malloc(sizeof(struct queue));
if(!ptr)return NULL;
```

```
ptr->front= NULL;
ptr->rear=NULL;
return ptr;
}
void enqueue(struct queue* q,struct Treenode *root )
{
struct List *newnode=(struct List *)malloc(sizeof(struct List));
newnode->node=root;
newnode->next=NULL;
if(q->rear==NULL)
{
q->rear=newnode;
}
else
{
q->rear->next=newnode;
q->rear=newnode;
}
if(q->front==NULL)
q->front=q->rear;
}
int isempty(struct queue* q)
{
return (q->front==NULL);
}
struct Treenode* dequeue(struct queue* q)
{
struct Treenode* dq;
struct List *temp;
if(isempty(q))
{
printf("No elements in queue");
return NULL;
}
else
{
temp=q->front;
dq=q->front->node;
q->front=q->front->next;
free(temp);
}
return dq;
}
struct Treenode* get_top(struct queue *q)
{
return q->front->next->node;
}
void find_height(struct Treenode *root)
{
struct Treenode* temp;
struct queue* q=createqueue();
enqueue(q,root);
/* Having a NULL to determine the end of each level */
enqueue(q,NULL);
int level=0;
while(!isempty(q))
{
temp=dequeue(q);
if(temp==NULL)
{
if(!isempty(q))
enqueue(q,NULL);
level++;
}
else
{
if(temp->left)
{
enqueue(q,temp->left);
}
if(temp->right)
{
```

```
enqueue(q,temp->right);
}
}
}
printf("%d",level);
}
int main()
{
struct Treenode* root=newnode(5);
root->left =newnode(1);
root->right =newnode(7);
root->left->right =newnode(4);
root->left->left =newnode(9);
root->right->left =newnode(10);
root->right->right =newnode(3);
find_height(root);
return 0;
}
```

Guest
Here is the solution for finding the height (or depth) of a binary tree without recursion (queue implementation).
http://ideone.com/e.js/ndP4PS
isha
as you have discussed here that we find the height of a tree by the Number of edges on longest path from root to the deepest node then according this what should be the height of a tree 2 or 3 for above example????
anonymous
The usual convention says that the height of such a tree should be 2. The number of edges are counted as the height.
The only problem with this is that, when you write the recursive function for height, if you want it to be the number of edges, you would have to give the base case as
if(!root)
return -1;
That is, if we count it as the number of edges, then both, a tree with one node has a height of 0, and an empty tree as -1.
anonymous
The usual convention says that the height of such a tree should be 2. The number of edges are counted as the height.
The only problem with this is that, when you write the recursive function for height, if you want it to be the number of edges, you would have to give the base case as
if(!root)
return -1;
That is, if we count it as the number of edges, then both, a tree with one node has a height of 0, and an empty tree as -1.
Nitin Sharma

```
/*HEIGHT OF TREE WITHOUT LEVEL ORDER TRAVERSAL*/
#include
#include
typedef struct node
{
int value;
struct node *left,*right;
}node;
node* newnode(int n)
{
node *tmp;
tmp = (node*)calloc(1,sizeof(node));
if(tmp==NULL)
{
printf("Memory Underflow.n");
exit(0);
}
tmp->value=n;
tmp->left=NULL;
tmp->right=NULL;
return tmp;
}
void main()
{
node *stack[10],*tmp,*root;
int top=-1,max=0,over=0;
root=newnode(1);
root->left=newnode(2);
root->right=newnode(3);
```

```
root->left->left=newnode(4);
root->left->right=newnode(5);
stack[++top]=NULL;
while(1)
{
while(root)
{
stack[++top]=root;
root=root->left;
}
if(maxright==NULL || stack[top]==tmp)
{
if(stack[top]==NULL)
{
over=1;
break;
}
top–;
}
if(over==1)
{
break;
}
root=stack[top]->right;
stack[top]=tmp;
}
printf(“Height of tree is : %dn”,max);
}
```

Patil

Here is C implementation.

```
int treeHeight(mynode *root)
{
if(root == NULL)
return 0;
mynode *queue[20];
int height,front,rear;
height=0;
front = 0;
rear = 1;
queue[rear] = root;
while(1)
{
int nodeCount = (rear-front);
if(nodeCount == 0)
return height;
else
height++;
while(nodeCount > 0)
{
root = queue[++front];
if(root->left)
queue[++rear] = root->left;
if(root->right)
queue[++rear] = root->right;
nodeCount–;
}
}
}
```

12rad

Java Implementation:

```
        public static int getHeightOFtree_Iterative(Node root){

                Deque<Node> a = new LinkedList<Node>();
                int height = 0;

                int nodesinCurrentLevel =0;

                if(root == null){
                        return height;
                }
```

```
                a.add(root);
                height ++;

                nodesinCurrentLevel++;
                int nodeinNextLevel = 0;


                while(!a.isEmpty()){
                        Node removed = a.poll();
                        nodesinCurrentLevel --;

                        if(removed!=null){
                                List l = removed.getChildren();
                                l.removeAll(Collections.singleton(null));
                                nodeinNextLevel = l.size();
                                System.out.println("noide in nex tleve is "+nodeinNextLevel);
                                a.addAll(l);
                        }

                        if(nodesinCurrentLevel ==0){
                                nodesinCurrentLevel = nodeinNextLevel;
                                height++;
                        }
                }



                return height;
        }
```

ankur jain

```
#include<stdio.h>
#include<stdlib.h>
#include<iostream>
#include<vector>
#include<set>
#include<map>
#include<string>
#define input freopen("input.txt","r",stdin)
#define output freopen("out.txt","w",stdout)
//a=a+b-(b=a);
using namespace std;
/*
struct tree
{
int data;
string s;
int arr[]
};*/
string alpha[] = {"", "a", "b", "c", "d", "e",
"f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r",
"s", "t", "u", "v", "w", "x", "v", "z"};
void create(int d,string s,int arr[],int n)
{
//printf(" %d %d ",d,n);cout<<s<<endl;
if (n==0)
{
cout<<s<<endl;
return ;
}
d=arr[0];
//printf(" 1() %d %d ",d,n-1);cout<<s+alpha[d]<<endl;
create (d,s+alpha[d],arr+1,n-1);
if (n > 1)
{
d=arr[0]*10+arr[1];
//printf(" 2-> %d %d ",d,n);cout<<s<<endl;
if (d < 27)
{
//s=s+alpha[d];
//printf(" 2() %d %d ",d,n);cout<<s+alpha[d]<<endl;
create (d,s+alpha[d],arr+2,n-2);
```

```
}
}
}
void printAllInterpretations(int arr[],int n)
{
// printf(" -> %d %d ",0,n );cout<<" "<<endl;
create(0,"",arr,n);
}
int main()
{
int arr[] = {1, 2, 2, 1};
int n=sizeof(arr)/sizeof(arr[0]);
printAllInterpretations(arr,n);
}
```

Akshay Jindal

Here's the c implementation tested for the above tree
My approach uses a stack based Iterative inorder traversal
In my approach a node will have 2 extra fields
1.parent(to traverse upwards)
2.visited
visited—>Here's what it means
1.node->visited=0 —> It means that the node has been unvisited yet
2.node->visited=1 —> It means that the node has been visited but its left and right child are unvisited (time to push the node into the stack)
3.node->visited=2 —> It means that the node has been visited and its left child has also been visited (pop the left child from the stack)
4.node->visited=3 —> it means that the node has been visited and its left and right childs have been visited (pop the right child from the stack)
Works perfectly well but quite a long one, suggest some optimization for this method.

```
#include<stdio.h>

//Initially all the visited fields of the node is 0
struct tree_node
{
 int visited;
 struct tree_node *left;
 struct tree_node *right;
 struct tree_node *parent;
}
typedef tree_node Node;

void inorder(Node *root)
{
 p=root;
 if(root==NULL)
  return;

 else
  {
   while(1)
   {
    if(p->visited==0)
    {
     while(p->left!=NULL)
     {
      p->visited=1;
      push(p);
      p=p->left;
     }
     p->visited=1;push(p);
    }

    if(p->visited==1)
    {
     printf("%d",p->data);
     p->visited=2;
    }

    if(p->visited==2)
    {
     if(p->left!=NULL)
```

```
       pop(p->left);
     p->visited=3;
     if(p->right!=NULL)
      p=p->right;
    }

    if(p->visited==3)
    {
     if(p->right!=NULL)
       pop(p->right);
     p=p->parent;
     if(p==NULL)
       break;
    }

   } //close of while
 }
}
```

Akshay Jindal
The above code is the for traversal. Here comes the main part i.e. calculating the height of the tree.Did a slight modification in the section starting from line 13

```
if(p->visited==0)
    {
     while(p->left!=NULL)
     {
      p->visited=1;
      top=push(p);
      p=p->left;
     }
     p->visited=1;push(p);
     if(max<top)
       max=top;
}//close of if
```

Coder

```
public void HeightOfTree(struct node *root)
{
        struct Queue *Q = createQueue();
        int level = 1;

        if(!root)
          return;

        Enqueue(Q,root);
        Enqueue(Q,NULL);

        while(!IsEmpty(Q))
        {
                root = Dequeue(Q);

                // Indicates level completion.
                if(root == NULL)
                {
                        if(!IsEmpty(Q))
                        {
                                Enqueue(NULL);
                        }
                        level++;
                }
                else
                {
                        if(root->left)
                                Enqueue(Q,root->left);
                        if(root->right)
                                Enqueue(Q,root->right);
                }
        }

        printf("\n height of the tree is [%d]",level);
```

```
}
```

noobie
level must be initiated with value 0 bcoz u r incrementing it after the completion of every level. this way u'll end up displaying +1 levels.
kush

```
int height(tree *root)
{
        int max=-1;
        tree *arr[10000];int top=-1,hr[10000],h=0;
        while(1)
        {
                while(root)
                {
                        ++top;
                        arr[top]=root;
                        root=root->left;
                        hr[top]=++h;
                }
                tree *temp=arr[top];
                while(!(temp->right))
                {
                        temp=arr[top];
                        h=hr[top];
                        if(max<h)max=h;
                        top--;
                        if(top==-1)return max;
                }
                root=temp->right;

        }
        return max;


}
```

Nitin Sharma
I think your algorithm will go in infinite loop…..lets see this example
1
1->left =2
1->right=3
2->left=4
2->right=5
now your algorithm will go in infinite loop in switching from 2 to 5 and 5 to 2 and it will switch infinitely……
kush

```
int height(tree *root)
{
int max=-1;
tree *arr[10000];int top=-1,hr[10000],h=0;
while(1)
{
while(root)
{
++top;
arr[top]=root;
root=root->left;
hr[top]=++h;
}
tree *temp=arr[top];
while(!(temp->right))
{
temp=arr[top];
h=hr[top];
if(maxright;
}
return max;
}
```

AMIT
If we just want to find height,we can do any other traversal like iterative inorder with stack and add level info inside the stack node,so with same time complexity,space complexity can be reduced to o(height of tree)
MANISH

Hi Amit,
Isn't if you do iterative inorder traversal, then your time complexity will be O(n)?
AMIT
yes,time complexity of both level order traversal and inorder traversal with stack is o(n)
but the space complexity of level order traversal is o(n) while inorder with stack is o(height,logn if we consider it as a balanced Binary tree)
Nikhil Agrawal

```
public void iterativeHeight(Node root)
{
    int height=0;
    Node t=new Node(-1);
    if(root==null)
        System.out.println("Height="+height);

    Queue<Node> s=new LinkedList<>();
    s.add(root);
    s.add(t);

    while(!s.isEmpty())
    {
        Node tt=(Node) s.remove();

        if(tt.value==-1)
        {
            height++;
            s.add(tt);

            Node justNext=(Node) s.peek();

            if(justNext.value==-1)
                break;
        }
        else
        {
            if(tt.left!=null)
                s.add(tt.left);

            if(tt.right!=null)
                s.add(tt.right);
        }


    }
    System.out.println("Iterative height="+height);
}
```

Devarshi
why dont we simply to the DFS.
Anon_001
Because topic is to solve iteratively .
Devarshi
ohh!!…thanks.
http://shashank7s.blogspot.com Shashank
you mean dfs can't be implemented iteratively ?
FYI we can 😉

```
/* Paste your code here (You may delete these lines if not writing code) */
```

AMIT
Exactly..its better to perform iterative inorder or preorder or similar thing…which can reduce space complexity