Find Height of Binary Tree represented by Parent array

A given array represents a tree in such a way that the array value gives the parent node of that particular index. The value of the root node index would always be -1. Find the height of the tree.
Height of a Binary Tree is number of nodes on the path from root to the deepest leaf node, the number includes both root and leaf.

```
Input: parent[] = {1 5 5 2 2 -1 3}
Output: 4
The given array represents following Binary Tree
         5
        /  \
       1    2
      /    / \
     0    3   4
         /
        6


Input: parent[] = {-1, 0, 0, 1, 1, 3, 5};
Output: 5
The given array represents following Binary Tree
          0
        /   \
       1     2
      / \
     3   4
    /
   5
  /
 6
```

Source: Amazon Interview experience | Set 128 (For SDET)

We strongly recommend to minimize your browser and try this yourself first.

A **simple solution** is to first construct the tree and then find height of the constructed binary tree. The tree can be constructed recursively by first searching the current root, then recurring for the found indexes and making them left and right subtrees of root. This solution takes $O(n^2)$ as we have to linearly search for every node.

An **efficient solution** can solve the above problem in O(n) time. The idea is to first calculate depth of every node and store in an array depth[]. Once we have depths of all nodes, we return maximum of all depths.
1) Find depth of all nodes and fill in an auxiliary array depth[].
2) Return maximum value in depth[].

Following are steps to find depth of a node at index i.
1) If it is root, depth[i] is 1.
2) If depth of parent[i] is evaluated, depth[i] is depth[parent[i]] + 1.
3) If depth of parent[i] is not evaluated, recur for parent and assign depth[i] as depth[parent[i]] + 1 (same as above).

Following is the implementation of above idea.

C++

```
// C++ program to find height using parent array
#include <iostream>
using namespace std;

// This function fills depth of i'th element in parent[].  The depth is
// filled in depth[i].
void fillDepth(int parent[], int i, int depth[])
{
    // If depth[i] is already filled
    if (depth[i])
        return;

    // If node at index i is root
    if (parent[i] == -1)
    {
        depth[i] = 1;
        return;
    }

    // If depth of parent is not evaluated before, then evaluate
    // depth of parent first
    if (depth[parent[i]] == 0)
        fillDepth(parent, parent[i], depth);

    // Depth of this node is depth of parent plus 1
    depth[i] = depth[parent[i]]  + 1;
```

```
}

// This function returns height of binary tree represented by
// parent array
int findHeight(int parent[], int n)
{
    // Create an array to store depth of all nodes/ and
    // initialize depth of every node as 0 (an invalid
    // value). Depth of root is 1
    int depth[n];
    for (int i = 0; i < n; i++)
        depth[i] = 0;

    // fill depth of all nodes
    for (int i = 0; i < n; i++)
        fillDepth(parent, i, depth);

    // The height of binary tree is maximum of all depths.
    // Find the maximum value in depth[] and assign it to ht.
    int ht = depth[0];
    for (int i=1; i<n; i++)
        if (ht < depth[i])
            ht = depth[i];
    return ht;
}

// Driver program to test above functions
int main()
{
    // int parent[] = {1, 5, 5, 2, 2, -1, 3};
    int parent[] = {-1, 0, 0, 1, 1, 3, 5};

    int n = sizeof(parent)/sizeof(parent[0]);
    cout << "Height is " << findHeight(parent, n);
    return 0;
}
```

Java

```
// Java program to find height using parent array
class BinaryTree {

    // This function fills depth of i'th element in parent[].  The depth is
    // filled in depth[i].
    void fillDepth(int parent[], int i, int depth[]) {

        // If depth[i] is already filled
        if (depth[i] != 0) {
            return;
```

```
        }

        // If node at index i is root
        if (parent[i] == -1) {
            depth[i] = 1;
            return;
        }

        // If depth of parent is not evaluated before, then evaluate
        // depth of parent first
        if (depth[parent[i]] == 0) {
            fillDepth(parent, parent[i], depth);
        }

        // Depth of this node is depth of parent plus 1
        depth[i] = depth[parent[i]] + 1;
    }

    // This function returns height of binary tree represented by
    // parent array
    int findHeight(int parent[], int n) {

        // Create an array to store depth of all nodes/ and
        // initialize depth of every node as 0 (an invalid
        // value). Depth of root is 1
        int depth[] = new int[n];
        for (int i = 0; i < n; i++) {
            depth[i] = 0;
        }

        // fill depth of all nodes
        for (int i = 0; i < n; i++) {
            fillDepth(parent, i, depth);
        }

        // The height of binary tree is maximum of all depths.
        // Find the maximum value in depth[] and assign it to ht.
        int ht = depth[0];
        for (int i = 1; i < n; i++) {
            if (ht < depth[i]) {
                ht = depth[i];
            }
        }
        return ht;
    }

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

        BinaryTree tree = new BinaryTree();
```

```
        // int parent[] = {1, 5, 5, 2, 2, -1, 3};
        int parent[] = new int[]{-1, 0, 0, 1, 1, 3, 5};

        int n = parent.length;
        System.out.println("Height is  " + tree.findHeight(parent, n));
    }
}
```

Python

```
# Python program to find height using parent array

# This functio fills depth of i'th element in parent[]
# The depth is filled in depth[i]

def fillDepth(parent, i , depth):

    # If depth[i] is already filled
    if depth[i] != 0:
        return

    # If node at index i is root
    if parent[i] == -1:
        depth[i] = 1
        return

    # If depth of parent is not evaluated before,
    # then evaluate depth of parent first
    if depth[parent[i]] == 0:
        fillDepth(parent, parent[i] , depth)

    # Depth of this node is depth of parent plus 1
    depth[i] = depth[parent[i]] + 1

# This function reutns height of binary tree represented
# by parent array
def findHeight(parent):
    n = len(parent)
    # Create an array to store depth of all nodes and
    # initialize depth of every node as 0
    # Depth of root is 1
    depth = [0 for i in range(n)]

    # fill depth of all nodes
    for i in range(n):
        fillDepth(parent, i, depth)
```

```
    # The height of binary tree is maximum of all
    # depths. Find the maximum in depth[] and assign
    # it to ht
    ht = depth[0]
    for i in range(1,n):
        ht = max(ht, depth[i])

    return ht

# Driver program to test above function
parent = [-1 , 0 , 0 , 1 , 1 , 3 , 5]
print "Height is %d" %(findHeight(parent))

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

Output:

```
Height is 5
```

Note that the time complexity of this programs seems more than O(n). If we take a closer look, we can observe that depth of every node is evaluated only once.

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

Leave a comment Category: Trees

Related Posts:

- Check sum of Covered and Uncovered nodes of Binary Tree
- Lowest Common Ancestor in a Binary Tree | Set 2 (Using Parent Pointer)
- Construct a Binary Search Tree from given postorder
- BFS vs DFS for Binary Tree
- Maximum difference between node and its ancestor in Binary Tree
- Inorder Non-threaded Binary Tree Traversal without Recursion or Stack
- Check if leaf traversal of two Binary Trees is same?
- Closest leaf to a given node in Binary Tree