Construct Binary Tree from given Parent Array representation

Given an array that represents a tree in such a way that array indexes are values in tree nodes and array values give the parent node of that particular index (or node). The value of the root node index would always be -1 as there is no parent for root. Construct the standard linked representation of given Binary Tree from this given representation.

```
Input: parent[] = {1, 5, 5, 2, 2, -1, 3}
Output: root of below tree
          5
        /  \
       1    2
      /    / \
     0    3   4
         /
        6
Explanation:
Index of -1 is 5.  So 5 is root.
5 is present at indexes 1 and 2.  So 1 and 2 are
children of 5.
1 is present at index 0, so 0 is child of 1.
2 is present at indexes 3 and 4.  So 3 and 4 are
children of 2.
3 is present at index 6, so 6 is child of 3.


Input: parent[] = {-1, 0, 0, 1, 1, 3, 5};
Output: root of below tree
         0
       /   \
      1     2
     / \
    3   4
   /
  5
 /
6
```

Expected time complexity is O(n) where n is number of elements in given array.

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

A **Simple Solution** to recursively construct by first searching the current root, then recurring for the found indexes (there can be at most two 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 use extra space. An array created[0..n-1] is used to keep track of created nodes.

createTree(parent[], n)

1. Create an array of pointers say created[0..n-1]. The value of created[i] is NULL if node for index i is not created, else value is pointer to the created node.
2. Do following for every index i of given array
   createNode(parent, i, created)

createNode(parent[], i, crated[])

1. If created[i] is not NULL, then node is already created. So return.
2. Create a new node with value 'i'.
3. If parent[i] is -1 (i is root), make created node as root and return.
4. Check if parent of 'i' is created (We can check this by checking if created[parent[i]] is NULL or not.
5. If parent is not created, recur for parent and create the parent first.

6. Let the pointer to parent be p. If p->left is NULL, then make the new node as left child. Else make the new node as right child of parent.

Following is C++ implementation of above idea.

C++

```
// C++ program to construct a Binary Tree from parent array
#include<bits/stdc++.h>
using namespace std;

// A tree node
struct Node
{
    int key;
    struct Node *left, *right;
};

// Utility function to create new Node
Node *newNode(int key)
{
    Node *temp = new Node;
    temp->key  = key;
    temp->left  = temp->right = NULL;
    return (temp);
}

// Creates a node with key as 'i'.  If i is root, then it changes
// root.  If parent of i is not created, then it creates parent first
void createNode(int parent[], int i, Node *created[], Node **root)
{
    // If this node is already created
    if (created[i] != NULL)
        return;

    // Create a new node and set created[i]
    created[i] = newNode(i);

    // If 'i' is root, change root pointer and return
    if (parent[i] == -1)
    {
        *root = created[i];
        return;
    }

    // If parent is not created, then create parent first
    if (created[parent[i]] == NULL)
        createNode(parent, parent[i], created, root);

    // Find parent pointer
    Node *p = created[parent[i]];

    // If this is first child of parent
    if (p->left == NULL)
        p->left = created[i];
    else // If second child
        p->right = created[i];
}

// Creates tree from parent[0..n-1] and returns root of the created tree
Node *createTree(int parent[], int n)
{
    // Create an array created[] to keep track
    // of created nodes, initialize all entries
    // as NULL
    Node *created[n];
    for (int i=0; i<n; i++)
        created[i] = NULL;

    Node *root = NULL;
    for (int i=0; i<n; i++)
        createNode(parent, i, created, &root);

    return root;
}

//For adding new line in a program
inline void newLine(){
    cout << "\n";
}

// Utility function to do inorder traversal
void inorder(Node *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        cout << root->key << " ";
        inorder(root->right);
    }
}

// Driver method
int main()
{
    int parent[] =  {-1, 0, 0, 1, 1, 3, 5};
    int n = sizeof parent / sizeof parent[0];
    Node *root = createTree(parent, n);
    cout << "Inorder Traversal of constructed tree\n";
    inorder(root);
    newLine();
}
```

Java

```
// Java program to construct a binary tree from parent array

// A binary tree node
class Node
{
    int key;
    Node left, right;

    public Node(int key)
    {
        this.key = key;
        left = right = null;
    }
}

class BinaryTree
{
    Node root;

    // Creates a node with key as 'i'.  If i is root, then it changes
    // root.  If parent of i is not created, then it creates parent first
    void createNode(int parent[], int i, Node created[])
    {
        // If this node is already created
        if (created[i] != null)
            return;

        // Create a new node and set created[i]
        created[i] = new Node(i);

        // If 'i' is root, change root pointer and return
        if (parent[i] == -1)
        {
            root = created[i];
            return;
        }

        // If parent is not created, then create parent first
        if (created[parent[i]] == null)
            createNode(parent, parent[i], created);

        // Find parent pointer
        Node p = created[parent[i]];

        // If this is first child of parent
        if (p.left == null)
            p.left = created[i];
        else // If second child

            p.right = created[i];
    }

    /* Creates tree from parent[0..n-1] and returns root of
       the created tree */
    Node createTree(int parent[], int n)
    {
        // Create an array created[] to keep track
        // of created nodes, initialize all entries
        // as NULL
        Node[] created = new Node[n];
        for (int i = 0; i < n; i++)
            created[i] = null;

        for (int i = 0; i < n; i++)
            createNode(parent, i, created);

        return root;
    }

    //For adding new line in a program
    void newLine()
    {
        System.out.println("");
    }

    // Utility function to do inorder traversal
    void inorder(Node node)
    {
        if (node != null)
        {
            inorder(node.left);
            System.out.print(node.key + " ");
            inorder(node.right);
        }
    }

    // Driver method
    public static void main(String[] args)
    {

        BinaryTree tree = new BinaryTree();
        int parent[] = new int[]{-1, 0, 0, 1, 1, 3, 5};
        int n = parent.length;
        Node node = tree.createTree(parent, n);
        System.out.println("Inorder traversal of constructed tree ");
        tree.inorder(node);
        tree.newLine();
    }
}

// This code has been contributed by Mayank Jaiswal(mayank_24)
```

Python

```
# Python implementation to construct a Binary Tree from
# parent array

# A node structure
class Node:
    # A utility function to create a new node
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

""" Creates a node with key as 'i'. If i is root,then
    it changes root. If parent of i is not created, then
    it creates parent first
"""
def createNode(parent, i, created, root):

    # If this node is already created
    if created[i] is not None:
        return

    # Create a new node and set created[i]
    created[i] = Node(i)

    # If 'i' is root, change root pointer and return
    if parent[i] == -1:
        root[0] = created[i] # root[0] denotes root of the tree
        return

    # If parent is not created, then create parent first
    if created[parent[i]] is None:
        createNode(parent, parent[i], created, root )

    # Find parent pointer
    p = created[parent[i]]

    # If this is first child of parent
    if p.left is None:
        p.left = created[i]
    # If second child
    else:
        p.right = created[i]


# Creates tree from parent[0..n-1] and returns root of the
# created tree
def createTree(parent):
    n = len(parent)

    # Create and array created[] to keep track
    # of created nodes, initialize all entries as None
    created = [None for i in range(n+1)]

    root = [None]
    for i in range(n):
        createNode(parent, i, created, root)

    return root[0]

#Inorder traversal of tree
def inorder(root):
    if root is not None:
        inorder(root.left)
        print root.key,
        inorder(root.right)

# Driver Method
parent = [-1, 0, 0, 1, 1, 3, 5]
root = createTree(parent)
print "Inorder Traversal of constructed tree"
inorder(root)
```

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

Output:

```
Inorder Traversal of constructed tree
6 5 3 1 4 0 2
```

Similar Problem: Find Height of Binary Tree represented by Parent array