Construct Ancestor Matrix from a Given Binary Tree

Given a Binary Tree where all values are from **0** to **n-1**. Construct an ancestor matrix **mat[n][n]**. Ancestor matrix is defined as below.

```
mat[i][j] = 1 if i is ancestor of j
mat[i][j] = 0, otherwise
```

Examples:

```
Input: Root of below Binary Tree.
          0
        /   \
       1     2
Output: 0 1 1
        0 0 0
        0 0 0

Input: Root of below Binary Tree.
           5
        /    \
       1      2
      /  \    /
     0    4  3
Output: 0 0 0 0 0 0
        1 0 0 0 1 0
        0 0 0 1 0 0
        0 0 0 0 0 0
        0 0 0 0 0 0
        1 1 1 1 1 0
```

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

The idea is to traverse the tree. While traversing, keep track of ancestors in an array. When we visit a node, we add it to ancestor array and consider corresponding row in adjacency matrix. We mark all ancestors in its row as 1. Once a node and all its children are processed, we remove the node from ancestor array.

Below is C++ implementation of above idea.

```
// C++ program to construct ancestor matrix for
// given tree.
#include<bits/stdc++.h>
using namespace std;
#define MAX 100

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

// Creating a global boolean matrix for simplicity
bool mat[MAX][MAX];

// anc[] stores all ancestors of current node.  This
// function fills ancestors for all nodes.
// It also returns size of tree.  Size of tree is
// used to print ancestor matrix.
int ancestorMatrixRec(Node *root, vector<int> &anc)
{
    /* base case */
    if (root == NULL) return 0;;

    // Update all ancestors of current node
```

```
    int data = root->data;
    for (int i=0; i<anc.size(); i++)
       mat[anc[i]][data] = true;

    // Push data to list of ancestors
    anc.push_back(data);

    // Traverse left and right subtrees
    int l = ancestorMatrixRec(root->left, anc);
    int r = ancestorMatrixRec(root->right, anc);

    // Remove data from list the list of ancestors
    // as all descendants of it are processed now.
    anc.pop_back();

    return l+r+1;
}

// This function mainly calls ancestorMatrixRec()
void ancestorMatrix(Node *root)
{
    // Create an empty ancestor array
    vector<int> anc;

    // Fill ancestor matrix and find size of
    // tree.
    int n = ancestorMatrixRec(root, anc);

    // Print the filled values
    for (int i=0; i<n; i++)
    {
        for (int j=0; j<n; j++)
            cout << mat[i][j] << " ";
        cout << endl;
    }
}

/* Helper function to create a new node */
Node* newnode(int data)
{
    Node* node = new Node;
    node->data = data;
    node->left = node->right = NULL;
    return (node);
}

/* Driver program to test above functions*/
int main()
{
    /* Construct the following binary tree
                5
              /   \
            1      2
          /  \    /
         0    4  3    */
    Node *root        = newnode(5);
    root->left        = newnode(1);
    root->right       = newnode(2);
    root->left->left  = newnode(0);
    root->left->right = newnode(4);
    root->right->left = newnode(3);

    ancestorMatrix(root);

    return 0;
}
```

Output:

```
0 0 0 0 0 0
1 0 0 0 1 0
0 0 0 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0
1 1 1 1 1 0
```

Time complexity of above solution is $O(n^2)$.

How to do reverse – construct tree from ancestor matrix?

Construct tree from ancestor matrix