Construct tree from ancestor matrix

Given an ancestor matrix mat[n][n] where Ancestor matrix is defined as below.

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

Construct a Binary Tree from given ancestor matrix where all its values of nodes are from 0 to n-1.

1. It may be assumed that the input provided the program is valid and tree can be constructed out of it.
2. Many Binary trees can be constructed from one input. The program will construct any one of them.

Examples:

```
Input: 0 1 1
       0 0 0
       0 0 0
Output: Root of one of the below trees.
    0               0
  /   \     OR     /   \
 1     2          2     1

Input: 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
Output: Root of one of the below trees.
      5              5               5
   /    \           / \            /   \
  1      2   OR    2   1    OR    1     2  OR ....
 /  \   /        /    /  \       / \    /
0   4  3        3    0    4     4   0  3
There are different possible outputs because ancestor
matrix doesn't store that which child is left and which
is right.
```

This problem is mainly reverse of below problem.

Construct Ancestor Matrix from a Given Binary Tree

We strongly recommend you to minimize your browser and try this yourself first.
Observations used in the solution:

1. The rows that correspond to leaves have all 0's
2. The row that corresponds to root has maximum number of 1's.
3. Count of 1's in i'th row indicates number of descendants of node i.

The idea is to construct the tree in **bottom up manner**.
1) Create an array of node pointers node[].

2) Store row numbers that correspond to a given count. We have used multimap for this purpose.

3) Process all entries of multimap from smallest count to largest (Note that entries in map and multimap can be traversed in sorted order). Do following for every entry.
.....a) Create a new node for current row number.
.....b) If this node is not a leaf node, consider all those descendants of it whose parent is not set, make current node as its parent.

4) The last processed node (node with maximum sum) is root of tree.

Below is C++ implementation of above idea. Following are steps.

```
// Given an ancestor matrix for binary tree, construct
// the tree.
#include <bits/stdc++.h>
using namespace std;

# define N 6

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

/* 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);
}

// Constructs tree from ancestor matrix
Node* ancestorTree(int mat[][N])
{
    // Binary array to determine weather
    // parent is set for node i or not
    int parent[N] = {0};

    // Root will store the root of the constructed tree
    Node* root = NULL;

    // Create a multimap, sum is used as key and row
    // numbers are used as values
    multimap<int, int> mm;

    for (int i = 0; i < N; i++)
    {
        int sum = 0; // Initialize sum of this row
        for (int j = 0; j < N; j++)
            sum += mat[i][j];

        // insert(sum, i) pairs into the multimap
        mm.insert(pair<int, int>(sum, i));
    }

    // node[i] will store node for i in constructed tree
    Node* node[N];

    // Traverse all entries of multimap.  Note that values
    // are accessed in increasing order of sum
    for (auto it = mm.begin(); it != mm.end(); ++it)
    {
      // create a new node for every value
      node[it->second] = newNode(it->second);

      // To store last processed node. This node will be
      // root after loop terminates
      root = node[it->second];

      // if non-leaf node
      if (it->first != 0)
      {
        // traverse row 'it->second' in the matrix
        for (int i = 0; i < N; i++)
        {
           // if parent is not set and ancestor exits
           if (!parent[i] && mat[it->second][i])
           {
             // check for unoccupied left/right node
             // and set parent of node i
```

```
            if (!node[it->second]->left)
              node[it->second]->left = node[i];
            else
              node[it->second]->right = node[i];

            parent[i] = 1;
          }
        }
      }
    }
    return root;
}

/* Given a binary tree, print its nodes in inorder */
void printInorder(Node* node)
{
    if (node == NULL)
        return;
    printInorder(node->left);
    printf("%d ", node->data);
    printInorder(node->right);
}

// Driver program
int main()
{
    int mat[N][N] = {{ 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 }
    };

    Node* root = ancestorTree(mat);

    cout << "Inorder traversal of tree is \n";
    printInorder(root);

    return 0;
}
```

Output:

```
Inorder traversal of tree is
0 1 4 5 3 2
```

Note that we can also use an array of vectors in place of multimap. We have used multimap for simplicity. Array of vectors would improve performance as inserting and accessing elements would take O(1) time.