B1

#include <iostream>

#include <vector>

using namespace std;

struct *TreeNode*

{

    int val;

*TreeNode*\* left;

*TreeNode*\* right;

    TreeNode(int *data*){

        val = *data*;

        left = NULL;

        right = NULL;

    }

};

*TreeNode*\* buildTree(int *N*, vector<pair<int, int>>& *edges*) {

    vector<*TreeNode*\*> nodes(*N*);

}

int height(*TreeNode*\* *root*) {

    if(*root* == NULL)

        return 0;

    int leftHeight = height(*root*->left);

    int rightHeight = height(*root*->right);

    return max(leftHeight,rightHeight) + 1;

}

void preorder(*TreeNode*\* *root*, vector<int>& *result*) {

    if (*root* == NULL) {

        return;

    }

*result*.push\_back(*root*->val);

    preorder(*root*->left, *result*);

    preorder(*root*->right, *result*);

}

void postorder(*TreeNode*\* *root*, vector<int>& *result*) {

    if (*root* == nullptr) {

        return;

    }

    postorder(*root*->left, *result*);

    postorder(*root*->right, *result*);

*result*.push\_back(*root*->val);

}

void inorder(*TreeNode*\* *root*, vector<int>& *result*) {

    if (*root* == nullptr) {

        return;

    }

    inorder(*root*->left, *result*);

*result*.push\_back(*root*->val);

    inorder(*root*->right, *result*);

}

bool isBinaryTree(*TreeNode*\* *root* ) {

    if (*root* == nullptr) {

        return true;

    }

    if (!isBinaryTree(*root*->left) || !isBinaryTree(*root*->right)) {

        return false;

    }

    return true;

}

int main() {

   int n,m;

   cin>>n>>m;

   vector<*TreeNode*\*> nodes(n + 1);

   for (int i = 1; i <= n; i++) {

        nodes[i] = **new** *TreeNode*(i);

    }

   for(int i=0;i<m;i++){

    int u,v;

    cin>>u>>v;

    if (nodes[u]->left == nullptr) {

            nodes[u]->left = nodes[v];

        } else {

            nodes[u]->right = nodes[v];

        }

   }

*TreeNode*\* root = nodes[1];

    vector<int> result;

    cout << height(root) << endl;

    preorder(root, result);

    for (int num : result) {

        cout << num << " ";

    }

    cout << endl;

    result.clear();

    postorder(root, result);

    for (int num : result) {

        cout << num << " ";

    }

    cout << endl;

    result.clear();

    if (isBinaryTree(root)) {

        inorder(root,result);

    } else {

        cout << "NOT BINARY TREE";

    }

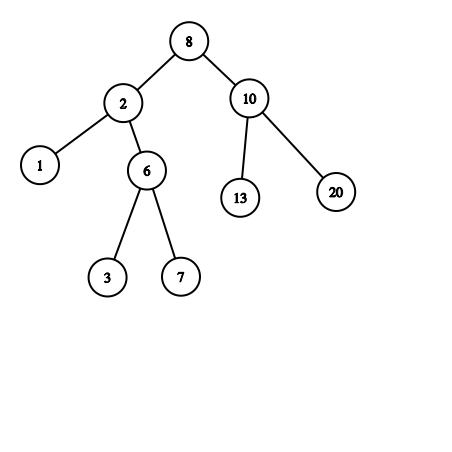
    cout << endl;

    return 0;

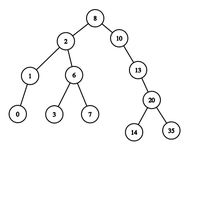
}

B2

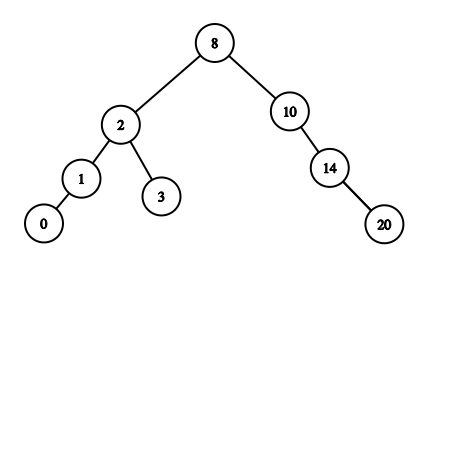
a



B



C



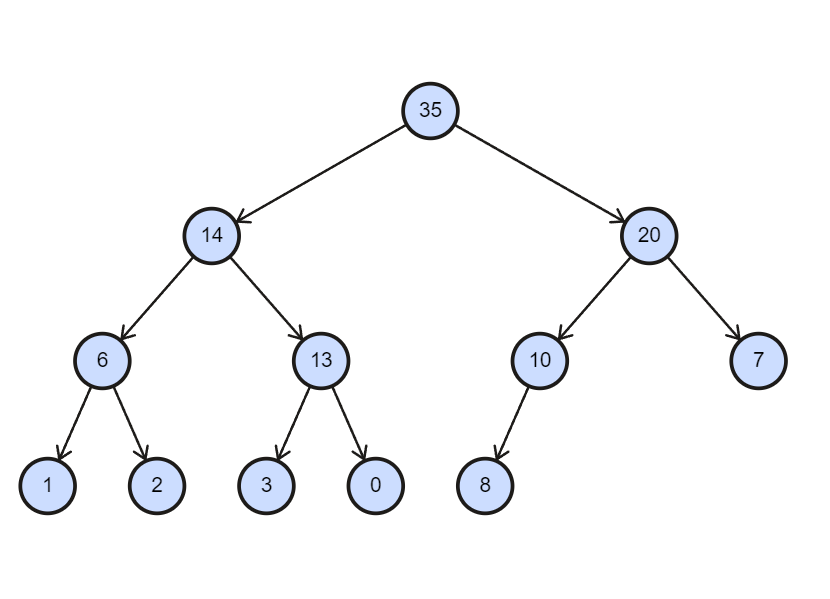
B3

a

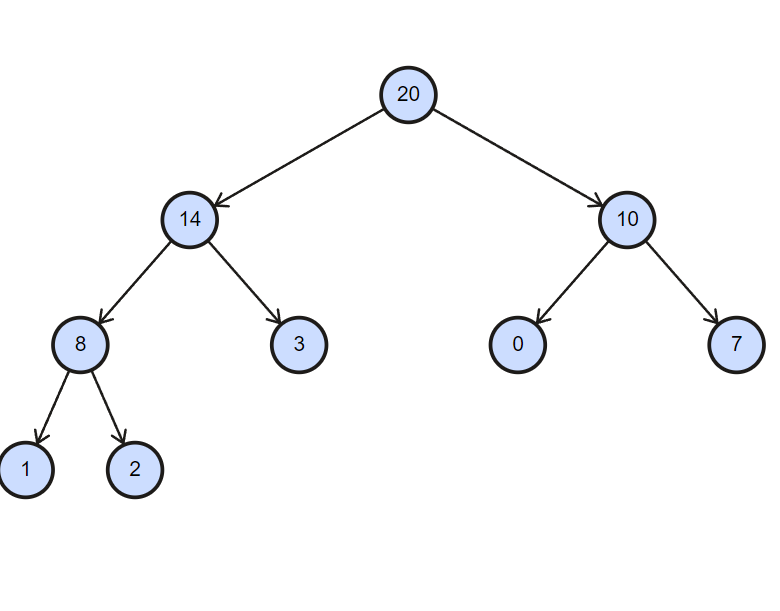
A diagram of a network

Description automatically generated

B



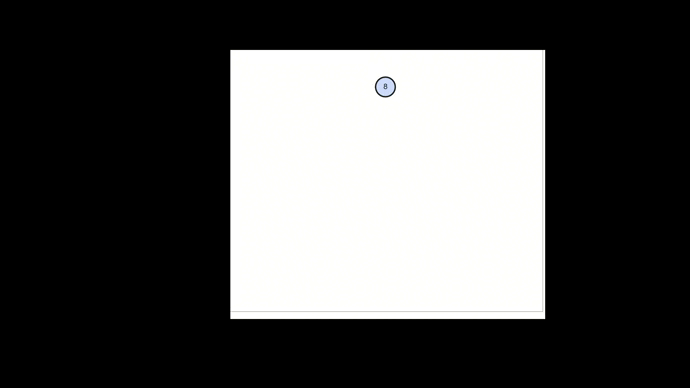
c



B4

8,5,9,4,18,15,17,1,7,14

a



b

function removeMax(root):

if root is NULL:

// The tree is empty, nothing to remove.

return root

parent = NULL

current = root

// Traverse to the rightmost node (maximum element).

while current.right is not NULL:

parent = current

current = current.right

// Check if the maximum element is a leaf node.

if current.left is NULL:

// Case a: Maximum element has no right child (leaf node).

if parent is not NULL:

parent.right = NULL

else:

// The maximum element was the root.

root = NULL

else:

// Case b: Maximum element has a right child.

parent.right = current.left

current.left = NULL

delete current

return root

c

function removeMin(root):

if root is NULL:

return root

parent = NULL

current = root

// Traverse to the leftmost node (minimum element).

while current.left is not NULL:

parent = current

current = current.left

// Check if the minimum element is a leaf node.

if current.right is NULL:

// Case a: Minimum element has no right child (leaf node).

if parent is not NULL:

parent.left = NULL

else:

// The minimum element was the root.

root = NULL

else:

// Case b: Minimum element has a right child.

parent.left = current.right

current.right = NULL

delete current

return root

B5

A diagram of a network

Description automatically generated

B

A screen shot of a diagram

Description automatically generated

C

function removeMax(root):

if root is NULL:

return root

parent = NULL

current = root

// Traverse to the rightmost node (minimum element).

while current.right is not NULL:

parent = current

current = current.right

// Check if the max element is a leaf node.

if current.right is NULL:

// Case a: max element has no left child (leaf node).

if parent is not NULL:

parent.right = NULL

else:

// The max element was the root.

root = NULL

else:

// Case b: max element has a child.

parent.right = current.left

current.left = NULL

delete current

return root