#include <iostream>

#include <set>

#include <string>

#include <stack>

#include <tuple>

using namespace std;

struct tree {

int x;

tree \* l;

tree \* r;

};

void read\_value(tree \* T) {

if(T != NULL) {

cerr << T->x << endl;

read\_value(T->l);

read\_value(T->r);

}

else {

cerr << "An empty tree was encountered." << endl;

}

}

/\*

\* The maximum number of distinct values that appear on a path starting at the

\* root of the tree.

\*/

int maxNumOfDistVal = 0;

set<int> distinct\_values;

stack<int> stk;

int seqOfLinkedAdjNodes(tree \* top) {

if(top != NULL) {

if(distinct\_values.empty() || (distinct\_values.find(top->x)) == distinct\_values.end()) {

pair<set<int>::iterator, bool> dist\_val\_it = distinct\_values.insert(top->x);

if(dist\_val\_it.second) {

cerr << top->x << " sucessfully inserted." << endl;

}

else {

cerr << "Something prevented the value from being inserted." << endl;

}

int lAdjNode = seqOfLinkedAdjNodes(top->l);

int rAdjNode = seqOfLinkedAdjNodes(top->r);

if((lAdjNode == 0) && (rAdjNode == 0)) {

cerr << "Number of distinct values on this path is " << distinct\_values.size() << endl;

if((distinct\_values.size()) > maxNumOfDistVal) {

maxNumOfDistVal = distinct\_values.size();

cerr << "Max number of distinct values is now " << distinct\_values.size() << endl;

}

}

int numOfElemRemoved = distinct\_values.erase(top->x);

cerr << "Number of elements erased from the set: " << numOfElemRemoved << endl;

cerr << "Size of the recorded set: " << distinct\_values.size() << endl;

}

}

else {

cerr << "Application encountered an empty tree." << endl;

return 0;

}

}

bool treeContainsNR(tree \*root, int item) {

/\*

\* Return true if "item" is one of the items in the binary-sort tree

\* to which the root points. Return false if not.

\*/

tree \* runner; // For "running" down the tree.

runner = root; //Start at the root node.

while(true) {

if(runner == NULL) {

// We've fallen off the tree without finding the item.

return false;

}

else if(item == runner->x) {

// We've found the item!

return true;

}

else if(item < runner->x) {

// If the item occurs, it must be in the left subtree.

// So, advance the runner down one level to the left.

runner = runner->l;

}

else {

// If the item occurs, it must be in the right subtree.

// So, advance the runner down one level to the right.

runner = runner->r;

}

} // end while

}

int main()

{

tree \* first = NULL;

tree \* second = NULL;

tree \* third = NULL;

tree \* fourth = NULL;

tree mid\_left = {

2,

first,

second,

};

tree mid\_right = {

3,

third,

fourth,

};

tree top = {

1,

&mid\_left,

&mid\_right,

};

//std::set<int> distinct\_values; //global

//read\_value(&top);

seqOfLinkedAdjNodes(&top);

return 0;

}