Lowest Common Ancestor

Write a C program to construct a Binary Search Tree. Then input two nodes and find their Lowest Common Ancestor. You are exp to fill in the code for the following function: findLCA() : For finding the lowest common ancestor between two nodes n1 and n2.

Input Format

Input Format:

Number of elements in the tree(n)

Element 1 to be placed in the tree

Element 2 to be placed in the tree

.

.

.

Element n to be placed in the tree

n1

n2

Constraints

Constraints:

Number of nodes in the tree must be greater than 2.

A node cannot be an ancestor of itself.

n1 and n2 must not be a number present in the root node.

All the elements should be distinct

Output Format

The least common ancestor

Sample Input 0

```
6
10
5
13
1
6
11
1
11
```

Sample Output 0

```
10
```

Explanation 0

A binary search tree comprising 6 elements is constructed. The lowest common ancestor of 1 and 11 is then found to be 10.

Contest ends in an hour

Submissions: 18
Max Score: 10
Difficulty: Medium

Rate This Challenge:
☆☆☆☆☆

More

C

```c
#include<stdio.h>
#include<stdlib.h>

typedef struct node
{
    int info;
    struct node* left;
    struct node* right;
}NODE;

NODE* constructTree(NODE *root,int ele);
NODE* findLCA(NODE* root, int n1, int n2,NODE* prev);
NODE* destroyTree(NODE* root);

int main()
{
NODE *root=NULL;

    int n;
    scanf("%d",&n);

    for(int i=0;i<n;i++)
    {
        int ele;
        scanf("%d",&ele);
        root=constructTree(root,ele);
    }

    int n1,n2;
    scanf("%d %d",&n1,&n2);
    NODE* lca=findLCA(root,n1,n2,root);
    printf("%d",lca->info);
    destroyTree(root);
    return 0;
}

NODE* constructTree(NODE *root,int ele)
{
    NODE* node = (NODE*)malloc(sizeof(NODE));
    node->info = ele;
    node->left = NULL;
    node->right = NULL;

    if(root == NULL)
    {
        root = node;
    }
    else
    {
        NODE* p = root;
        NODE* q= NULL;
        while(p!=NULL)
        {
            if(node->info < p->info)
            {
                q=p;
                p = p->left;
            }
            else
            {
                q=p;
                p = p->right;
            }
        }
        if(node->info < q->info)
        {
            q->left = node;
        }
        else
        {
            q->right = node;
        }
    }
    return root;
}

NODE* destroyTree(NODE* root)
{
    if (root != NULL)
    {
        root->left=destroyTree(root->left);
        root->right=destroyTree(root->right);
//        printf("Freed %d\n",root->info);
        free(root);
    }
    return NULL;
}

NODE* findLCA(NODE* root, int n1, int n2,NODE* prev)
{
    if((root->info<n1)^(root->info<n2)){
        return root;
    }
    else if((n1<root->info) && (n2<root->info)){
        return findLCA(root->left, n1, n2, root);
    }
    else if((n1>root->info) && (n2>root->info)){
        return findLCA(root->right, n1, n2, root);
    }
    else if(root->info==n1 || root->info==n2){
        return prev;
    }
    else{
        return root;
    }
}
```

Line: 106 Col: 5

Upload Code as File   Test against custom input   Run Code   Submit Code

Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
6
10
5
13
1
6
11
```

```
1
11
```

Your Output (stdout)

```
10
```

Expected Output

```
10
```