Count the number of nodes of a tree in O(1) space

Edit

In this article, we would see how to traverse the whole tree in O(1) space. We know how to traverse the tree using recursion or using data structures like stack and queue. But the problem is these data structures consume O(n) space (Why O(n):- Because we need to traverse all the elements so they need to be required to be stored in specific data structures). Recursive functions use something called "the call stack" which consumes O(n) space. You can learn even more about time complexity by clicking here

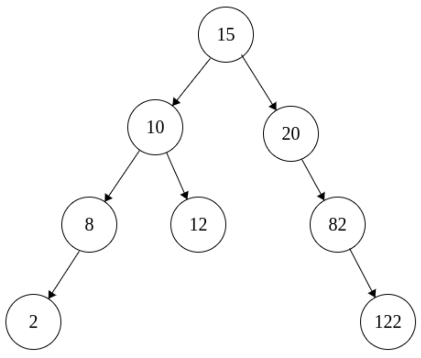
So now the only option left with us that is to think of doing changes to the links. Let's see how to accomplish this task in O(1) space (constant space).

```
Approach :
1)Make a pointer which points to current node
2)Continue steps above till the current node pointer is not null
3)If the left of the pointer is NULL , then increment count and move to right.
4)If left pointer is not null , make another temporary pointer to one left of current pointer and move towa
   it's not  null
```

Dry Run:

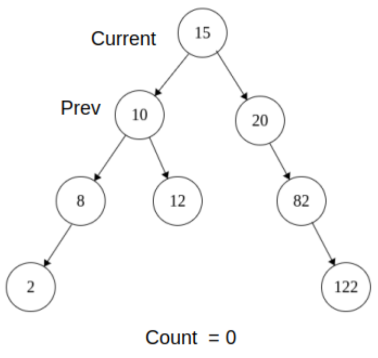
Let's take an example and find the number of nodes

Example :



Tree

1) Lets take 2 variables , current = 15 and prev = NULL and count = 0 .



Tree

2) We should continue the following process until the current node is NULL

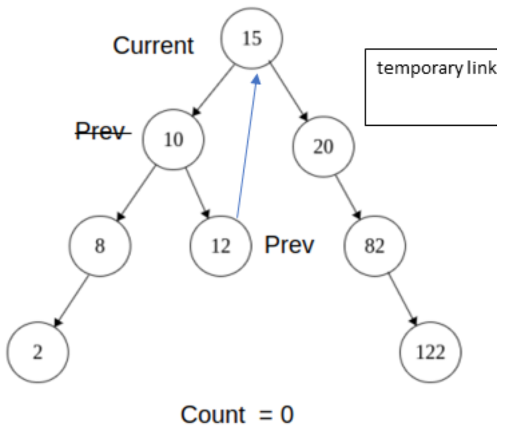
3) If Current -> left != NULL , prev = 10

Now iterate prev to right until prev -> right != NULL && prev -> right != current

so now prev = 12

Now prev -> right = NULL, therefore just make a temporary link to current i.e make a temporary right link from 12 to 15 , and move current to left

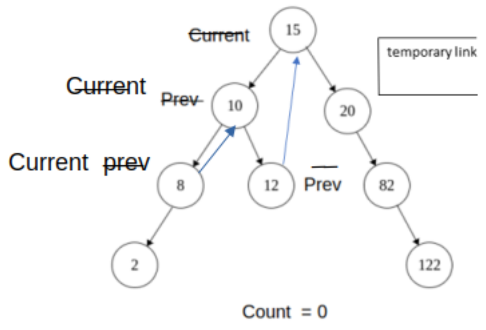
Current = 10



Tree

4) Again repeat step 3) , Now prev = 8 and its right is NULL, so again make a temporary right link from 8 to 10 and move current to left

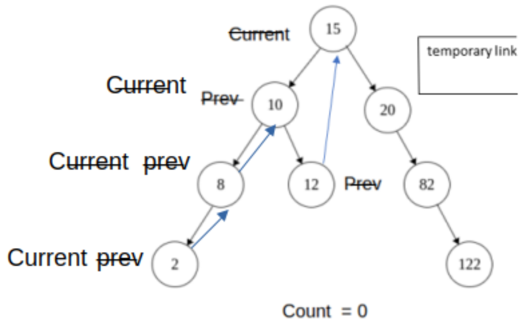
Current = 8



Tree

5)Again repeat step 3) , Now prev = 2 and its right is NULL, so again make a temporary right link from 2 to 8 and move current to left

Current = 2

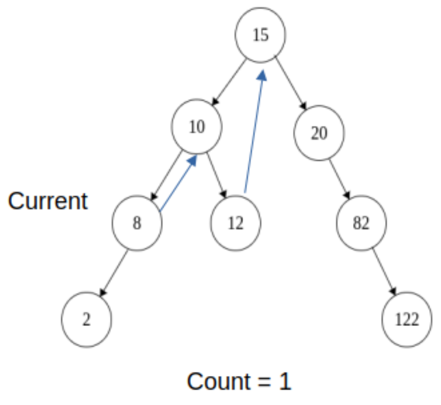


Tree

w current -> left == NULL, So increment count and move current to its right

nt = 1

current = 8 (We have a temporary pointer so we are able to go back)



Tree

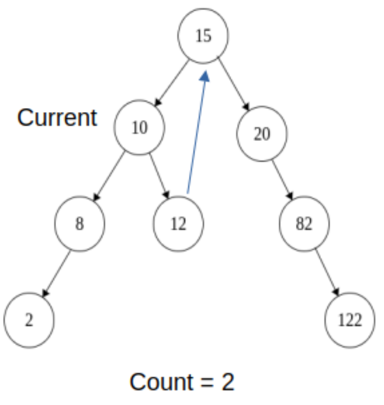
7)Again repeat step 3) , Now prev = 2 and its right is NULL, now when we iterate in loop prev -> right != curr, we stop when prev -> right = NULL

i.e prev = 2 , so make prev -> right = NULL and increment the count,move current to current -> right

The temporary link from 2 -> 8 is removed

Current = 10

Count = 2



Tree

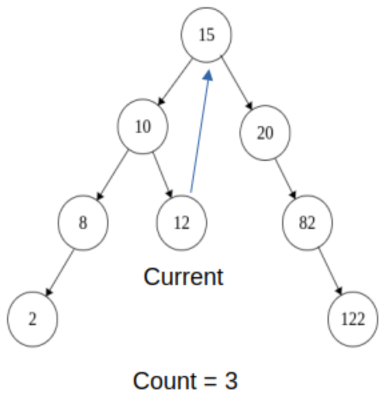
8)Again repeat step 3) , Now prev = 8 and its right is NULL, now when we iterate in loop prev -> right != curr, we stop when prev -> right = NULL

i.e prev = 8 , so make prev -> right = NULL and increment the count,move current to current -> right

The temporary link from 8 -> 10 is removed

Current = 12

Count = 3

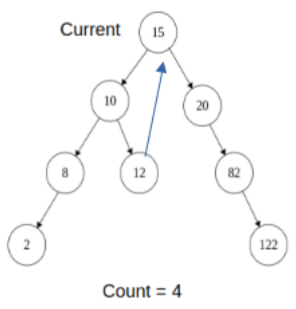


Tree

9)Now current -> left is NULL, increment count and moves current to current -> right

Current = 15

Count = 4



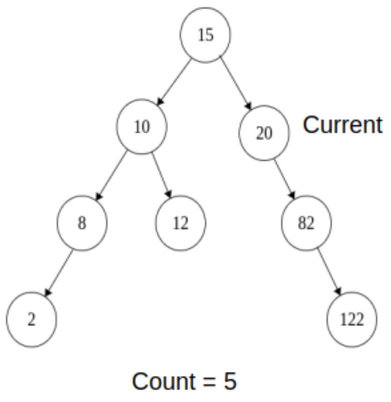
10)Again repeat step 3), Now prev = 10 and its right is NULL, now when we iterate in loop prev -> right != curr, we stop when prev -> right = NULL

i.e prev = 12 , so make prev -> right = NULL and increment the count,move current to current -> right

The temporary link from 12 -> 15 is removed

Current = 20

Count = 5

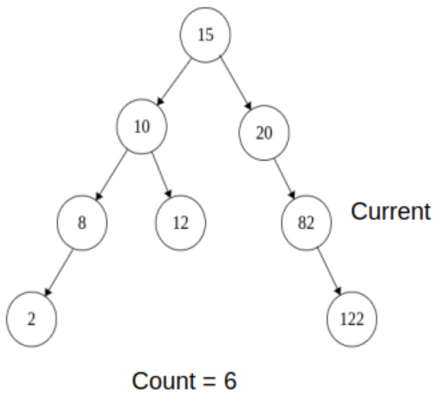


Tree

11)Now current -> left is NULL, increment count and moves current to current -> right

Current = 20

Count = 6

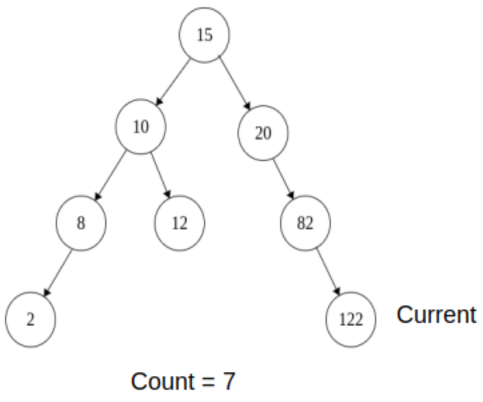


Tree

12)Now current -> left is NULL, increment count and moves current to current -> right

Current = 82

Count = 7

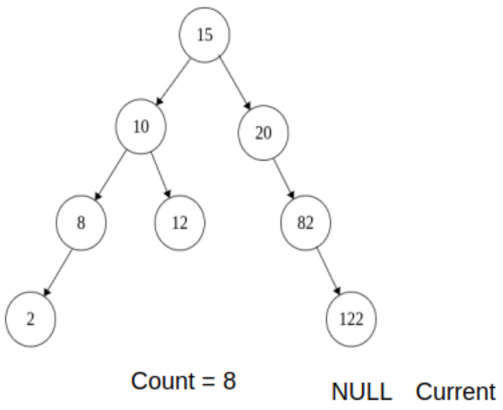


Tree

13)Now current -> left is NULL, increment count and moves current to current -> right

Current = 122

Count = 8



Tree

14)Now the current is NULL, so stop the loop .

In this way, we have found the count of nodes in a binary search tree in O(1) space

Code :

C

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

typedef struct tree {
    int data;
    struct tree* left; // Making structure with left , right
                       // pointer and data
    struct tree* right;
} tree;
typedef tree* Tree;

oid initTree(Tree* tnode)
{
    *tnode = NULL; // Initing the node to NULL
}

void insertIntoTree(Tree* tnode, int data)
{
    Tree newnode = (tree*)malloc(sizeof(tree));
    newnode->left = NULL; // Allocating space for tree and
                          // setting data
```

```
        newnode->right = NULL;
        newnode->data = data;

        if (!*tnode) {
            *tnode = newnode; // If tree is empty , make the
                              // newnode as root
            return;
        }
        Tree current = *tnode, prev = NULL;
        while (current) {
            prev = current;
            if (current->data
                == data) // If duplicate data is being inserted
                         // , discard it
                return;

            else if (current->data > data)
                current
                    = current->left; // If data to be inserted
                                     // is less than tnode ->
                                     // data then go left

            else
                current
                    = current->right; // If data to be inserted
                                      // is more than tnode ->
                                      // data then go right
        }

        if (prev->data > data)
            prev->left = newnode;

        else
            prev->right = newnode;

        return;
    }

    int getCountOfNodes(Tree tnode)
    {

        int count = 0; // Initialize count to 0
        Tree curr = tnode;
        while (curr != NULL) {
            if (curr->left
                == NULL) { // If current node is NULL then print
                           // data of the current node and move
                           // pointer to right
                // printf("%d ",curr -> data);
                count++; // Increment count
                curr = curr->right;
            }
            else {
                Tree prev
                    = curr->left; // Store left child of current
                                  // in some variable
                while (prev->right && prev->right != curr)
                    prev = prev->right; // Iterate until prev ->
                                        // right != NULL or prev
                                        // -> right != current
                                        // node

                if (prev->right
                    == NULL) { // If this is last node then ,
                               // make a temporary pointer to
```

```
                                  // root (tnode)
                        prev->right
                            = curr; // Move current pointer to left
                        curr = curr->left;
                    }
                    else {
                        prev->right
                            = NULL; // If prev -> next == current
                                    // node then , there is temporary
                                    // link present , remove it i.e
                                    // make NULL

                        // printf("%d ",curr -> data); //Print the
                        // data
                        count++;

                        curr = curr->right; // Move current to right
                    }
                }
            }
            return count;
        }
        int main()
        {

            Tree tree1; // Making Tree and initing
            initTree(&tree1);

            insertIntoTree(&tree1, 15);
            insertIntoTree(&tree1, 10);
            insertIntoTree(&tree1, 20);
            insertIntoTree(&tree1,
                           8); // Inserting elements into the tree
            insertIntoTree(&tree1, 12);
            insertIntoTree(&tree1, 82);
            insertIntoTree(&tree1, 122);
            insertIntoTree(&tree1, 2);

            int count = getCountOfNodes(tree1);
            printf("Count of nodes in O(1) space : %d \n", count);

            return 0;
        }
```

Output

```
Count of nodes in O(1) space : 8
```

Time Complexity: O(n) (We visit each node at most once, so the time complexity is an order of n i.e n)

Space complexity: O(1) (We just use some temporary pointer variables to make changes in links, no additional data structure or recursion is used)

Advantage :

1)We can traverse the tree in O(1) space.

can find the number of nodes, height, etc in constant space

Limitations :

1)We couldn't change links if we are not allowed to make changes to the tree.

sarvesh123305
Checkout this author's Contributed articles

C Programs CS – Placements C Language Advanced Data Structure Binary Search Tree

Author's Thread

Comments

Paragraph

Start Writing

Add Comment

GeeksforGeeks Team 2023-04-14 - 00:06:21
Dear Author,

Kindly write the problem statement clearly. So, that we can review it.
Reply