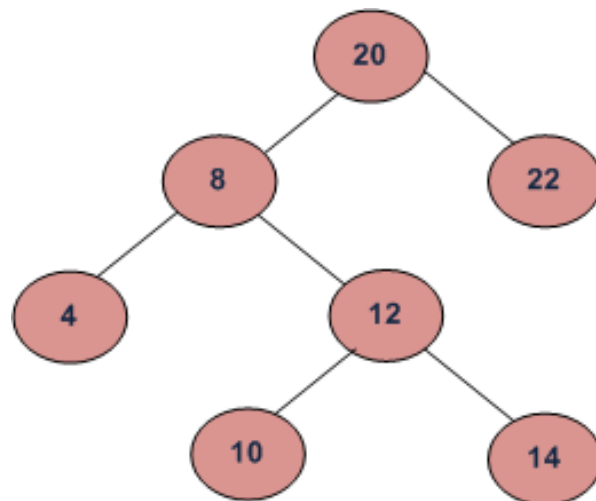


Inorder Successor in Binary Search Tree

In Binary Tree, Inorder successor of a node is the next node in Inorder traversal of the Binary Tree. Inorder Successor is NULL for the last node in Inorder traversal.

In Binary Search Tree, Inorder Successor of an input node can also be defined as the node with the smallest key greater than the key of input node. So, it is sometimes important to find next node in sorted order.



In the above diagram, inorder successor of 8 is 10, inorder successor of 10 is 12 and inorder successor of 14 is 20.

Method 1 (Uses Parent Pointer)

In this method, we assume that every node has parent pointer.

The Algorithm is divided into two cases on the basis of right subtree of the input node being empty or not.

Google™ Custom Search



GeeksforGeeks



52,731 people like GeeksforGeeks.



Interview Experiences

Advanced Data Structures

Dynamic Programming

Greedy Algorithms

Backtracking

Pattern Searching

Divide & Conquer

Mathematical Algorithms

Recursion

Geometric Algorithms

Input: *node*, *root* // *node* is the node whose Inorder successor is needed.

output: *succ* // *succ* is Inorder successor of *node*.

1) If right subtree of *node* is not *NULL*, then *succ* lies in right subtree. Do following.

Go to right subtree and return the node with minimum key value in right subtree.

2) If right subtree of *node* is *NULL*, then *succ* is one of the ancestors. Do following.

Travel up using the parent pointer until you see a node which is left child of its parent. The parent of such a node is the *succ*.

Implementation

Note that the function to find InOrder Successor is highlighted (with gray background) in below code.

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

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
{
    int data;
    struct node* left;
    struct node* right;
    struct node* parent;
};

struct node * minValue(struct node* node);

struct node * inOrderSuccessor(struct node *root, struct node *n)
{
    // step 1 of the above algorithm
    if( n->right != NULL )
        return minValue(n->right);

    // step 2 of the above algorithm
    struct node *p = n->parent;
    while(p != NULL && n == p->right)
    {
        n = p;
        p = p->parent;
    }
    return p;
}
```

ITT Tech - Official Site

itt-tech.edu

Associate, Bachelor Degree
Programs Browse Programs Now &
Learn More.

Popular Posts

[All permutations of a given string](#)

[Memory Layout of C Programs](#)

[Understanding "extern" keyword in C](#)

[Median of two sorted arrays](#)

[Tree traversal without recursion and without stack!](#)

[Structure Member Alignment, Padding and Data Packing](#)

[Intersection point of two Linked Lists](#)

[Lowest Common Ancestor in a BST.](#)

[Check if a binary tree is BST or not](#)

[Sorted Linked List to Balanced BST](#)

```

/* Given a non-empty binary search tree, return the minimum data
   value found in that tree. Note that the entire tree does not need
   to be searched. */
struct node * minValue(struct node* node) {
    struct node* current = node;

    /* loop down to find the leftmost leaf */
    while (current->left != NULL) {
        current = current->left;
    }
    return current;
}

/* Helper function that allocates a new node with the given data and
   NULL left and right pointers. */
struct node* newNode(int data)
{
    struct node* node = (struct node*)
        malloc(sizeof(struct node));

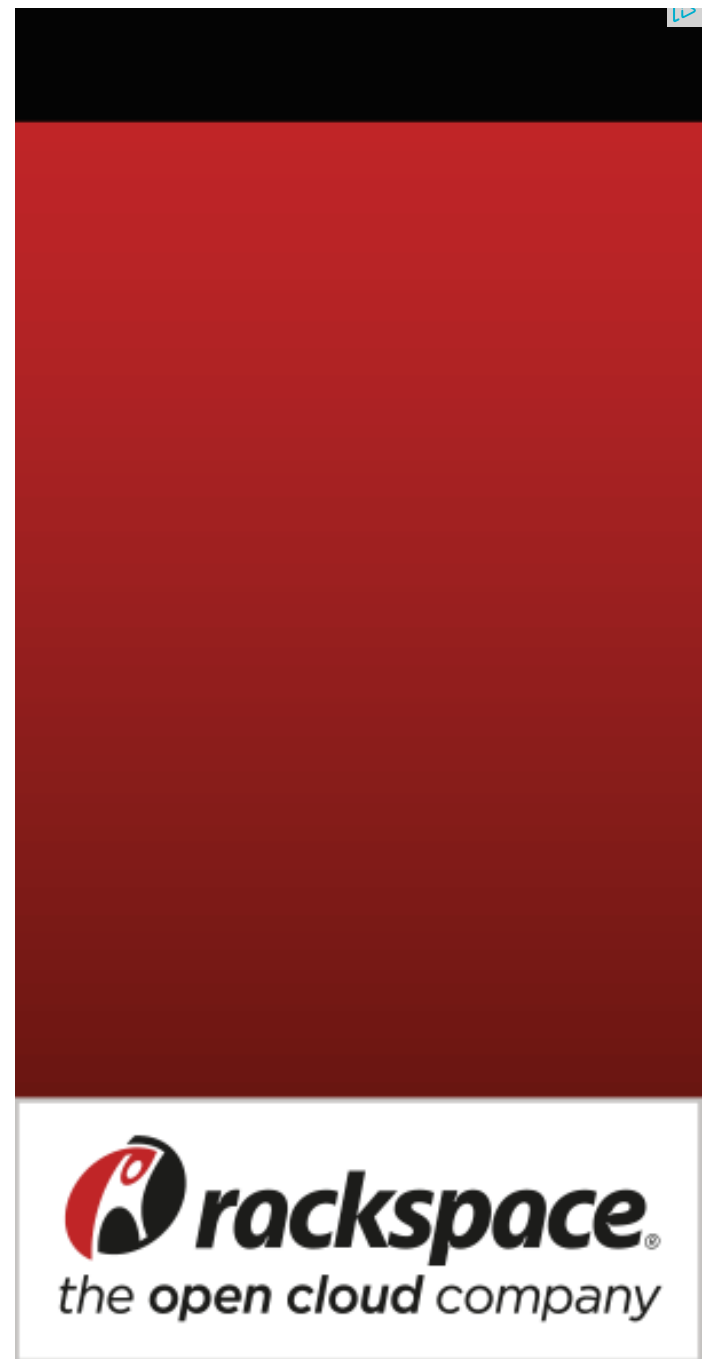
    node->data    = data;
    node->left    = NULL;
    node->right   = NULL;
    node->parent  = NULL;

    return (node);
}

/* Give a binary search tree and a number, inserts a new node with
   the given number in the correct place in the tree. Returns the new
   root pointer which the caller should then use (the standard trick to
   avoid using reference parameters). */
struct node* insert(struct node* node, int data)
{
    /* 1. If the tree is empty, return a new,
       single node */
    if (node == NULL)
        return (newNode(data));
    else
    {
        struct node *temp;

        /* 2. Otherwise, recur down the tree */
        if (data <= node->data)
        {
            temp = insert(node->left, data);
            node->left = temp;
        }
    }
}

```





Recent Comments

affizerv Your example has two 4s on row 3, that's why it...

[Backtracking | Set 7 \(Sudoku\)](#) · 43 minutes ago

RVM Can someone please elaborate this Qs from above...

[Flipkart Interview | Set 6](#) · 1 hour ago

Vishal Gupta I talked about as an Interviewer in general,...

[Software Engineering Lab, Samsung Interview | Set 2](#) · 1 hour ago

@meya Working solution for question 2 of 4f2f round....

[Amazon Interview | Set 53 \(For SDE-1\)](#) · 1 hour ago

sandeep void rearrange(struct node *head) {...

Given a linked list, reverse alternate nodes and append at the end · 3 hours ago

Neha I think that is what it should return as, in...

Find depth of the deepest odd level leaf node · 3 hours ago

AdChoices

[► Binary Tree](#)

[► Java Tree](#)

[► Node](#)

```

        temp->parent= node;
    }
    else
    {
        temp = insert(node->right, data);
        node->right = temp;
        temp->parent = node;
    }

    /* return the (unchanged) node pointer */
    return node;
}

/* Driver program to test above functions*/
int main()
{
    struct node* root = NULL, *temp, *succ, *min;

    //creating the tree given in the above diagram
    root = insert(root, 20);
    root = insert(root, 8);
    root = insert(root, 22);
    root = insert(root, 4);
    root = insert(root, 12);
    root = insert(root, 10);
    root = insert(root, 14);
    temp = root->left->right->right;

    succ = inOrderSuccessor(root, temp);
    if(succ != NULL)
        printf("\n Inorder Successor of %d is %d ", temp->data, succ->data)
    else
        printf("\n Inorder Successor doesn't exist");

    getchar();
    return 0;
}

```

Output of the above program:

Inorder Successor of 14 is 20

Time Complexity: O(h) where h is height of tree.

Method 2 (Search from root)

Method 2 (Search from root)

Parent pointer is NOT needed in this algorithm. The Algorithm is divided into two cases on the basis of right subtree of the input node being empty or not.

Input: *node*, *root* // *node* is the node whose Inorder successor is needed.

output: *succ* // *succ* is Inorder successor of *node*.

- 1) If right subtree of *node* is not *NULL*, then *succ* lies in right subtree. Do following.
Go to right subtree and return the node with minimum key value in right subtree.
- 2) If right subtree of *node* is *NULL*, then start from root and use search like technique. Do following.
Travel down the tree, if a node's data is greater than root's data then go right side, otherwise go to left side.

```
struct node * inOrderSuccessor(struct node *root, struct node *n)
{
    // step 1 of the above algorithm
    if( n->right != NULL )
        return minValue(n->right);

    struct node *succ = NULL;

    // Start from root and search for successor down the tree
    while (root != NULL)
    {
        if (n->data < root->data)
        {
            succ = root;
            root = root->left;
        }
        else if (n->data > root->data)
            root = root->right;
        else
            break;
    }

    return succ;
}
```

Thanks to [R.Srinivasan](#) for suggesting this method.

Time Complexity: $O(h)$ where h is height of tree.

References:

<http://net.pku.edu.cn/~course/cs101/2007/resource/Intro2Algorithm/book6/chap13.htm>

AdChoices

► [Tree Root](#)

► [Java Array](#)

► [Null Pointer](#)

AdChoices

► [Java XML](#)

► [XML Data](#)

► [C++](#)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Explore New Frontiers in Data.

HPCC Systems is Big Data Processing and Analytics
Open Source. Proven. Trusted.

 LexisNexis® [Learn More](#) 

Related Topics:

- [Print a Binary Tree in Vertical Order | Set 2 \(HashMap based Method\)](#)
- [Print Right View of a Binary Tree](#)
- [Red-Black Tree | Set 3 \(Delete\)](#)
- [Construct a tree from Inorder and Level order traversals](#)
- [Print all nodes at distance k from a given node](#)
- [Print a Binary Tree in Vertical Order | Set 1](#)
- [Interval Tree](#)
- [Check if a given Binary Tree is height balanced like a Red-Black Tree](#)



5



Tweet 2



4

Writing code in comment? Please use ideone.com and share the link here.

