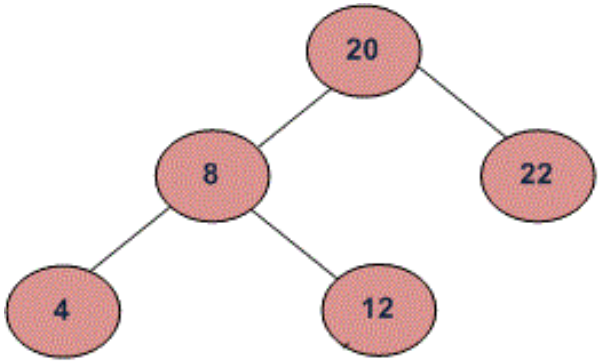
Print BST keys in the given range

Given two values k1 and k2 (where k1 < k2) and a root pointer to a Binary Search Tree. Print all the keys of tree in range k1 to k2. i.e. print all x such that k1<=x<=k2 and x is a key of given BST. Print all the keys in increasing order.

For example, if k1 = 10 and k2 = 22, then your function should print 12, 20 and 22.

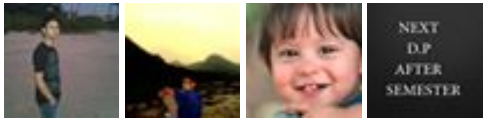


Thanks to bhasker for suggesting the following solution.

Algorithm:
1) If value of root's key is greater than k1, then recursively call in left subtree.
2) If value of root's key is in range, then print the root's key.
3) If value of root's key is smaller than k2, then recursively call in right subtree.

Implementation:

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

```c
/* A tree node structure */
struct node
{
  int data;
  struct node *left;
  struct node *right;
};

/* The functions prints all the keys which in the given range [k1..k2]
    The function assumes than k1 < k2 */
void Print(struct node *root, int k1, int k2)
{
   /* base case */
   if ( NULL == root )
      return;

   /* Since the desired o/p is sorted, recurse for left subtree first
      If root->data is greater than k1, then only we can get o/p keys
      in left subtree */
   if ( k1 < root->data )
     Print(root->left, k1, k2);

   /* if root's data lies in range, then prints root's data */
   if ( k1 <= root->data && k2 >= root->data )
     printf("%d ", root->data );

  /* If root->data is smaller than k2, then only we can get o/p keys
      in right subtree */
   if ( k2 > root->data )
     Print(root->right, k1, k2);
}

/* Utility function to create a new Binary Tree node */
struct node* newNode(int data)
{
  struct node *temp = new struct node;
  temp->data = data;
  temp->left = NULL;
  temp->right = NULL;

  return temp;
}

/* Driver function to test above functions */
int main()
{
  struct node *root = new struct node;
```


Popular Posts

```
    int k1 = 10, k2 = 25;

    /* Constructing tree given in the above figure */
    root = newNode(20);
    root->left = newNode(8);
    root->right = newNode(22);
    root->left->left = newNode(4);
    root->left->right = newNode(12);

    Print(root, k1, k2);

    getchar();
    return 0;
}
```

Output:
12 20 22

Time Complexity: O(n) where n is the total number of keys in tree.

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

Related Tpoics:

- 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

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

27 Comments GeeksforGeeks

Sort by Newest

CoderKnowledge • 8 months ago

Iterative Solution:

Approach:
1) Do Inorder Traversal.
2) while printing the nodes check whether the data within range of n1 and n2 w donot print.

psuedo Code:

Recent Comments

affiszerv 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

```
void keysInRange(struct node *root,int n1,int n2)
{
if(n1 > n2)
{
// swap n1 and n2
swap(n1,n2);
// now n1 and n2 values interchanged.
}
stack s;
while(1)
```

see more

2 ▲ | ▼ • Reply • Share ›

Guest • 8 months ago

Time Complexity: O(n) .

We can also store the values in a vector, if we want to process these values f

```
#include<cstdio>

#include<cstdlib>

#include<vector>

#include<iostream>

using namespace std;

typedef struct node

{
```

see more

• Reply • Share ›

Shuchit Khurana • 9 months ago

Would the complexity differ from their solution? Please let me know :)

• Reply • Share ›

miandfhy • 9 months ago

Do inorder traversal. At node apply the condition
Does this work?

• Reply • Share ›

sonali gupta • 10 months ago

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
typedef struct NODE
{
int info;
struct NODE *left,*right;
}node;
node *temp;
node *getnode()
{return((node *)malloc(sizeof(node)));
}
node *newNode(int x)
{
temp=getnode();
temp->info=x;
temp->left=NULL;
```

```
temp->right=NULL;
```

see more

Reply • Share ›

Kushagra Singhal • a year ago

in the second point of the also u have to call the function on the left and right s

1 Reply • Share ›

abhishek08aug • a year ago

C++ code:

```
 #include <iostream>
#include <stdlib.h>
using namespace std;

class tree_node {
  private:
    int data;
    tree_node * left;
    tree_node * right;
  public:
    tree_node() {
      left=NULL;
      right=NULL;
    }
    void set_data(int data) {
      this->data=data;
```

see more

Reply • Share ›

Jasprit Singh Chawla • a year ago

We can do INORDER Traversal and print the output.....
void BSTrange(struct node* node, int k1, int k2).
{

if(node == NULL).

return;.

BSTrange(node->left, k1, k2);.

if(k1 <= node->data && node->data <= k2).

printf("%d ", node->data);.

BSTrange(node->right, k1, k2);.

}

^ | v • Reply • Share ›

anonymous • 2 years ago

is this correct?

```
 void Print(struct node *root, int k1, int k2)
{
   /* base case */
   if ( NULL == root )
      return;

   /* Since the desired o/p is sorted, recurse for left subtree first
      If root->data is greater than k1, then only we can get o/p keys
      in left subtree */
   if ( k1 > root->data )
     Print(root->right, k1, k2);
```

```
/* if root's data lies in range, then prints root's data */
else if ( k1 <= root->data && k2 >= root->data )
 {
      Print(root->left, k1, k2);
```

see more

• Reply • Share ›

anonymous • 2 years ago

i want to know if this approach is better??

```
void Print(struct node *root, int k1, int k2)
{
/* base case */
if ( NULL == root )
return;

/* Since the desired o/p is sorted, recurse for left subtree first
If root->data is greater than k1, then only we can get o/p keys
in left subtree */
if ( k1 > root->data )
Print(root->right, k1, k2);

/* if root's data lies in range, then prints root's data */
else if ( k1 data && k2 >= root->data )
{
Print(root->left, k1, k2);
printf("%d ", root->data );
```

see more

• Reply • Share ›

I Think Sandeep It Misses Some Corner Cases

Like We have to check that k1 root value because obviously data will be printe
why missing some corner cases ? correct me if i missed something ?

[sourcecode language="C"]
/* Paste your code here (You may delete these lines if not writing code) */

 • Reply • Share ›

kartik ➦ WgpShashank • 2 years ago

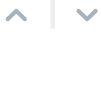
@WgpShashank: It works for all the cases. The following condition tak

```
 /* if root's data lies in range, then prints root's data */
if ( k1 <= root->data && k2 >= root->data )
  printf("%d ", root->data );
```

 • Reply • Share ›

R.Srinivasan • 3 years ago

void Print(struct node *root, int k1, int k2)
{
if(root)
{
Print(root->left,k1,k2);
if(k1 data && root->data data);
Print(root->right,k1,k2);
}
}

 • Reply • Share ›

mrn • 3 years ago

i think the code is eating up extra space as well as time(making new stack fra

instructions).In my humble opinion the best case would be non-recursive inorc checking value within it (you are also doing this checking in each stack frame) said above.

Reply • Share ›

Udit • 3 years ago

In the Algorithm:
In 1st point shouldn't there be k2 instead of k1 and similarly in the 3rd point k1
1st and 3rd points acc. to me:
1) If value of root's key is greater than k2, then recursively call in left subtree.
3) If value of root's key is smaller than k1, then recursively call in right subtree.

Plz correct me if I'm wrong

Reply • Share ›

yash khandelwal ➚ Udit • 2 years ago

I think Udit is right..
following line should be corrected
if (k1 data) // should be k2data
Print(root->left, k1, k2);
similarly, for
if (k2 > root->data) // should be k1>root->data
Print(root->right, k1, k2);

correct me if i am wrong ...

Reply • Share ›

yash khandelwal ➚ yash khandelwal • 2 years ago

sorry.!!
in line if(k1data) //should be k2data

Reply • Share ›

ajay • 3 years ago

we can do simply pre-order traversal and while printing data apply the conditio

• Reply • Share ›

vipul.mittal • 3 years ago

```
 /*i think it could also work,if it wont then tell me :where did i mi
void Print(struct node *root, int k1, int k2)
{
   /* base case */
   if (root==NULL )
      return;

   /* Since the desired o/p is sorted, recurse for left subtree first
      If root->data is greater than k1 and less than k2, then only we
     Print(root->left, k1, k2);
          int x = root->data;
      if(x>=k1 && x<=k2){
        printf("%d ", root->data );
        Print(root->right, k1, k2);
      }


}
```

• Reply • Share ›

anonymous • 3 years ago

it's O(n)

• Reply • Share ›

geek4u anonymous • 3 years ago

given range and h is the height of tree. For a balanced tree, height can
So Kartik's argument is also true for balanced BST.

Reply • Share ›

GeeksforGeeks geek4u • 3 years ago

@anonymous & @geek4u: Thanks for pointing this out. To mal
complexity to O(n).

Reply • Share ›

Jing • 3 years ago

For the middle case where (k1 <= root->data&&root->data<=k2)

Shouldn&#039t we first recursively call left tree, then print root, then recursivel
recursion in this case.

Reply • Share ›

GeeksforGeeks Jing • 3 years ago

@Jing: Please take a closer look at the program. It wiil call for the left s
call for right subtree when k2 > root->data. So when (k1 < root->data &
conditions in if statements will become true.

Reply • Share ›

vipulkv • 3 years ago

How the TC is O(k + Logn)?

Reply • Share ›

kartik vipulkv • 3 years ago

To understand this, we can take the corner cases first and then take th

1) Corner cases

a) If all the nodes are in the range [k1, k2], then k = n, the program bec

program and complexity of traversal is O(n).

b) If there are no nodes in the range [k1, k2], then all nodes will either b
range. I mean all the values would be either smaller than k1 or greater
the height of tree and complexity becomes O(Logn)

2) General Case:
The program will print k nodes and then traverses rest of the tree alon

• Reply • Share ›

Palash kartik • 2 years ago

Here is a code that trims a BST to have it's elements in a new
Use this to trim the BST and then run inorder traversal to get th
helps.

```
node* trimBST(node* root, int min, int max)
{
if(!root)return NULL;
else if(root->dataright,min,max);
else if(root->data>max) return trimBST(root->left,min,max);
else{
root->left=trimBST(root->left,min,max);
root->right=trimBST(root->right,min,max);
return root;
}
}
```

• Reply • Share ›