Must Do **Binary Search Tree** problems for **Placement**

Binary Search Trees	Fina a value in a BST
Binary Search Trees	Deletion of a node in a BST
Binary Search Trees	Find min and max value in a BST
Binary Search Trees	Find inorder successor and inorder predecessor in a BST
Binary Search Trees	Check if a tree is a BST or not
Binary Search Trees	Populate Inorder successor of all nodes
Binary Search Trees	Find LCA of 2 nodes in a BST
Binary Search Trees	Construct BST from preorder traversal
Binary Search Trees	Convert Binary tree into BST
Binary Search Trees	Convert a normal BST into a Balanced BST
Binary Search Trees	Merge two BST [V.V.V>IMP]
Binary Search Trees	Find Kth largest element in a BST
Binary Search Trees	Find Kth smallest element in a BST
Binary Search Trees	Count pairs from 2 BST whose sum is equal to given value "X"
Binary Search Trees	Find the median of BST in O(n) time and O(1) space
Binary Search Trees	Count BST ndoes that lie in a given range
Binary Search Trees	Replace every element with the least greater element on its right
Binary Search Trees	Given "n" appointments, find the conflicting appointments

Binary Search Trees	Check preorder is valid or not
Binary Search Trees	Check whether BST contains Dead end
Binary Search Trees	Largest BST in a Binary Tree [V.V.V.V.V IMP]
Binary Search Trees	Flatten BST to sorted list

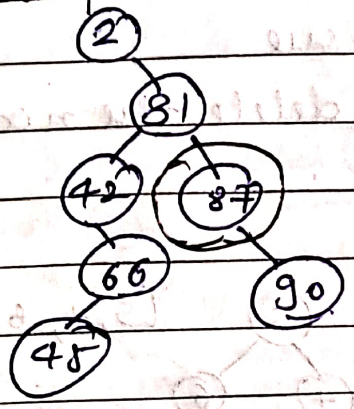
Binary Search Tree

Date 22/3/22 Page no:

Ques - Search a node in BST

Given BST and a node value X, find if the node with value 'x' is present in BST or not.

* Example :-



X = 87

Ans = 1 (true)

* Intution :-

1. Start from root
2. compare searching element (x) with root, if less than root, then recursively call left subtree.
3. else recursively call right subtree
4. if the element to search is found anywhere, return true, else return false.

Date: __/__/____ Page no: ____

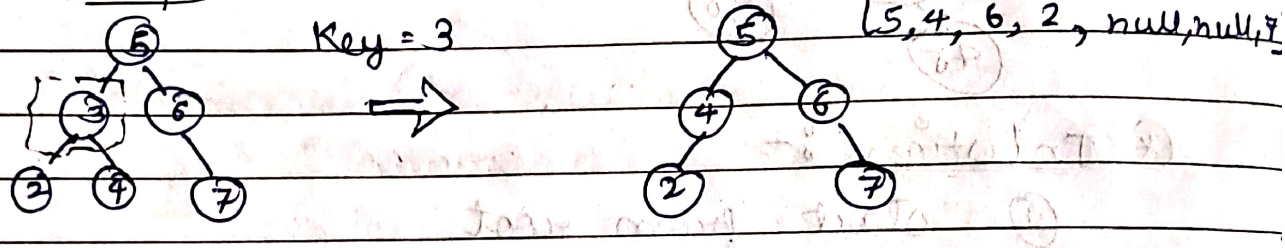
Ques- Delete node in a BST

Given root node reference of BST & a key, delete node with given key in BST. Return root node reference of BST.

Basically deletion can be divided into 2 stages:-

1) Search for node to remove
2) if the node is found, delete the node.

(*) Example :-



(*) Intution :-

We've 3 case for deleting node from BST :-

(1) The node is a leaf node :-

In this case we just remove the node & return the root, ∵ deleting any leaf node doesn't affecting remaining tree

(2) The node has one child :-

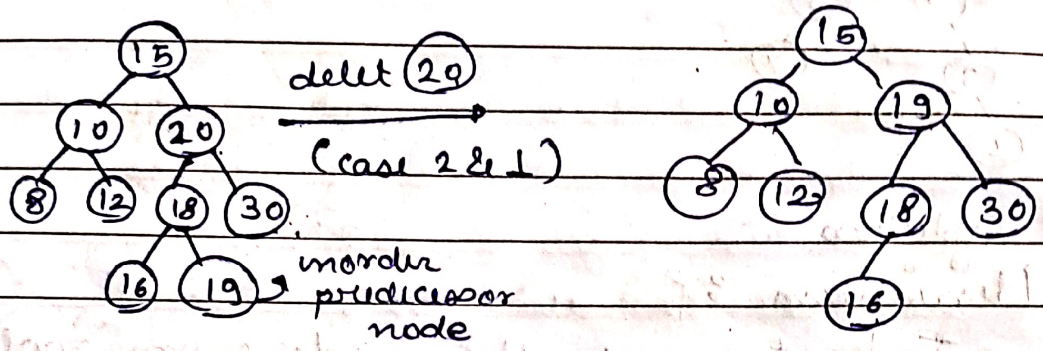
in this, we replace node with the child node and return the root.

(3) Node has 2 children :-

in this cas, in order to conserve BST properties, we need to replace the node with its inorder successor, i.e we need to replace it with either :

(1) The greatest value node in its left subtree

(2) The smallest value node in its right subtree & return the root.

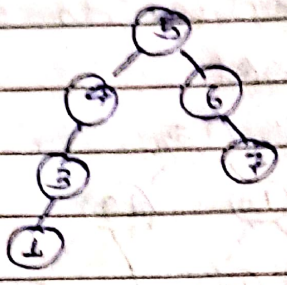


Date 23/3/22 Page no:

Ques → find Min & max. value in BST

Given BST, find mini. and max. element in BST.

Example :-



min : 1
max : 7

(a) Intuition :-

Minimum :-

Just traverse the node from root to left recursively until left is null.

The node whose left is Null is the node with min. value

Maximum :-

Just traverse the node from root to right recursively until the right is null.

The node whose right is Null is the node with max. value.

Snippit :-

(1)
```
int maxValue (struct node* node)
{
    struct node* current = node;
    while (current -> right != null)
        current = current -> right;

    return (current -> data);
}
```

(2)
```
int minValue (struct node* node)
{
    struct node* current = node;
    while (current -> left != null)
    {
        current = current -> left;
    }
    return (current -> data);
}
```

Date: / / Page no:

Ques → **Predecessor and Successor**

There is BST given, with root node with key part as integer only.
find inorder successor & predssor of given key.
If not found any return -1.

Example :-

input - T = no. of test cases, each T contain 'n' denotes no. of edges of BST.
next contain edges of BST.
last line contain BST.

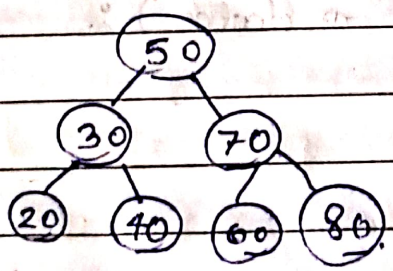
2 → T
6 → n

50,30L 30,20L 30,40R 50,70R 70,60L 70,80R

65 → key

output

predssor = 60 & Succssor = 70.



Data: __/__/____ Page no: ____

approach :-

1. If root is null, return.
2. If key is found,
   - (a) if its left subtree is not null, the predecessor will be the right most child of left subtree or left child itself.
   - (b) If its right subtree is not null, the successor will be the left most child of right subtree or right child itself.

   return.
3. If key is smaller than the root node, set the successor as root.
   search recursively into left subtree.
   - else,
     set the predecessor as root.
     search recursively into right subtree.

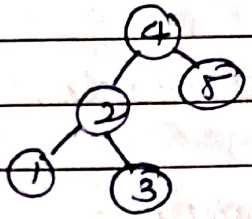
Date: / / Page no:

Ques → Check for BST

A Binary search tree has following properties :-

1. Left subtree of node contains only nodes with keys less than the node's key.
2. Right subtree of node contains only node with keys greater than the node's key.
3. Both left and right subtree also be BST.

Example :-



Ans = True

(*) Intution :-

1. Do inorder traversal of given tree and store result in temp vector.
2. This method assumes that there are no duplicate values in tree
3. Check if the temp array is sorted in ascending order, if it is, then tree is BST

Date: __/__/____ Page no: ____

Implement :-

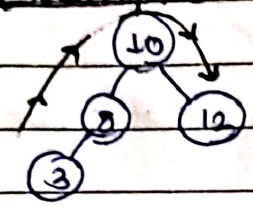
```
bool isBST(node* root)
    struct node *prev = null;
if (root)
    if (!isBST(root->left))
        return false;

if (prev != null && root->data <= prev->data)
    return false;

    prev = root;
    return isBST(root->right);

return true;
```

Date 24/3/22 Page no:

Ques- Populate inorder successor of all node

Given BT, write a func^n to populate next pointer for all nodes. The next pointer for every node should be set to pointer to inorder successor.

Example :-



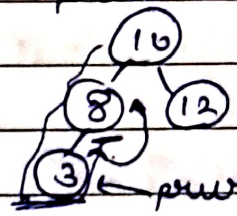
Ans = 3 → 8 (next), 8 → 10 (next), 10 → 12 (next), 12 → ⊥ (next)

Inorder - 3, 8, 10, 12

Intution :-

→ Take a prev variable as 'prev = = null'.

→ during inorder traversal, go till the bottom & when if (prev = = null) point prev = bottom element.



if (prev = null)

→ for (8) check if (prev != null) then prev → next = root; (this will point to the next successor.)

→ Then prev = root, means 8 will be prev. do the same for 10 & 12 also

Date: __/__/____ Page no: ____

Implementation :-

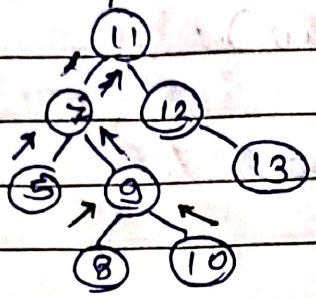
```
void solve (node * root, node * & prev)
{
    if ( ! root ) return;

    solve (root -> left , prev );
if ( prev != null ) {
        prev -> next = root ;
    }
    prev = root ;
solve ( root -> right, prev );
}

void populate next ( node * root )
{
    node * prev = NULL;
    solve ( root, prev );
}
```

Date: / / Page no:

Ques Find LCA of 2 nodes in BST

Given BST, & 2 node values. Find LCA of 2 nodes in BST

(*) Example :-



(1) $n_1 = 8$, $n_2 = 10$ ⇒ ans = LCA = 9.
∵ both 8 & 10 meeting at 9.

(2) $n_1 = 5$, $n_2 = 5$ ⇒ LCA = 7

Intution :-

Here we'll not use O(N) method.
we'll check for the 2 condition, if the given 2 condt^n becomes false then we got our LCA'

condt^n 1 - n_1 < root → data] left
n_2 < root → data

condt^n 2 - n_1 > root → data] right
n_2 > root → data

If condit^n 1 & 2 become the become false that mean 1 value is at left another at right, we found our LCA
eg.- (above example)

$n_1 = 8$ $n_2 = 10$

(1) 8 < 11, 10 < 11 ⇒ move to left
8 > 7, 10 > 7 ⇒ move to right
8 < 9, 10 > 9 ⇒ LCA = 9
↓ current node

Date: __/__/____ Page no: ____

Implement :-

```
node *  LCA ( node * root, int n1, int n2)
{
if (! root ) return null;

if (n1 < root → data and n2 < root → data)
    return LCA( root → left, n1, n2);

else if ( n1 > root → data and n2 > root → data)
    return LCA( root → right, n1, n2);

else return root;
```

Date: / / Page no:

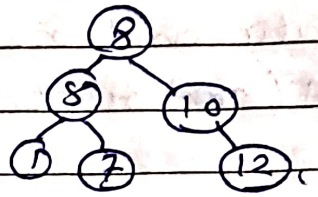
Ques - Construct BST from preorder traversal

Given a preorder traversal of BT, construct a BST.

Example :-

preorder: {8, 5, 1, 7, 10, 12}

BST -



* Intution :-

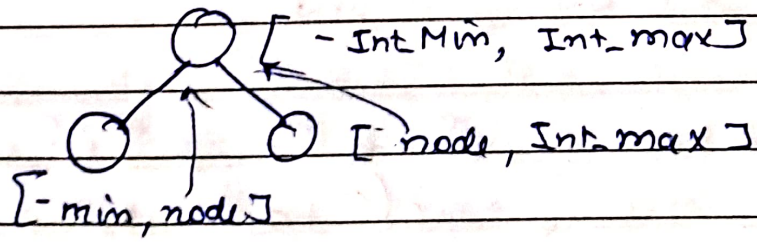
→ ∵ inorder is sorted, so firstly we'll sort the entire given data.

→ Now, you've both preorder (given) & inorder (from sorting), so now you can construct BST.

TC - $O(n \log n) + O(n)$

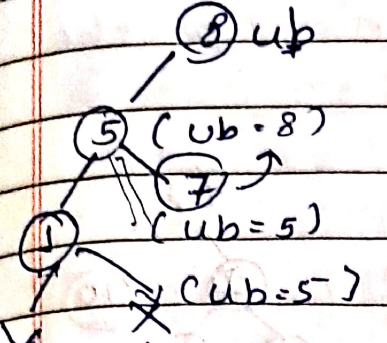
SC - $O(n)$

(*) Efficient Soln



assume upper bound ub = INT_MAX, i.e = (8).

→ ∵ preorder = root → left → right,
Here, there might be left might not be.
→ ∴ everything at left < 8 (ub).



i.e 8 < 5 (left)

→ we don't require lower bound bcoz, 7 can't be planted at 1 or 5, it is automatically checking for lb.

→ Same goes for 10 & 12.

* Implementation :-

```
TreeNode* FromPreorder(vector<int> A)  (2)
    int i = 0;
    return build(A, i, INT_MAX);
```

(3)

```
TreeNode * build(vector<int> &A, int &i, int bound)
(2)   if(i == A.size() || A[i] > bond)
        return NULL;
    TreeNode * root = new TreeNode(A[i++]);
    root → left = build(A, i, root → val);
    root → right = build(A, i, bound);
        return root;
```

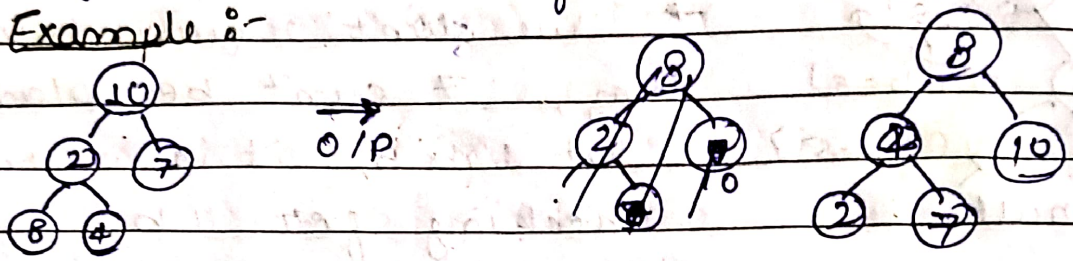
(3)

Date 25/3/22 Page no:

Ques • Convert Binary Tree to BST

Given BT, convert it into BST. It should be done in a way that it keeps the original structure of BT.

(*) Example :-



(*) Intuition :-

We can solve the above problem in 3 steps:

(1) Create a temp array arr[], that store inorder traversal of tree.

(2) Sort arr[]. In this implementation Quick sort will be used :: TC (n^2).

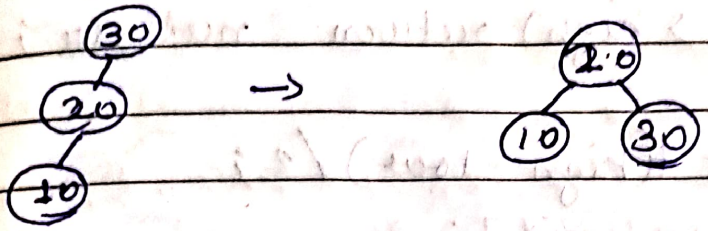
(3) Again do inorder traversal of tree and copy array elements to tree node one by one.

- TC - O(n log n)
- SC - O(n).

Q1 Convert normal BST into Balanced BST

given BST that may unbalanced, convert it into a balanced BST that has min. possible height.

* Example :-



* Intution :-

(1) Traverse given BST, in inorder & store result in an array. (TC - O(n).

Note that this array would be stored as inorder traversal of BST always produce sorted sequence.

(2) Build balance BST, from above created sorted array using recursive approch.

TC - O(n).

SC - O(n)

Date: __/__/____ Page no: ____

Snippets :-

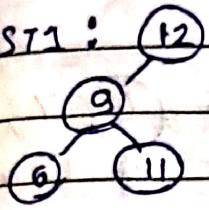
```
Node * cons(node* root, vector<node*>& common,
    int low, int high)
{
    if (low > high) return nullptr;

int mid = low + (high - low) / 2;
root = common[mid];
// Recursive approch
root->left = cons(root, common, low, mid-1);
root->right = cons(root, common, mid+1, high);

return root;
}
```

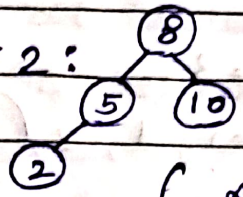
Date: / / Page no.

Ques → Merge 2 Balanced BST

Given 2 BST, write a func^n that merge 2 given balanced BST into single BST.

* Example :-

BST1 : 

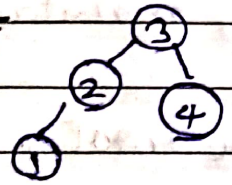
BST2 : 

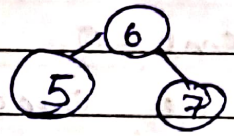
(After Merge & sort)

output : 2, 5, 6, 8, 9, 10, 11, 12

* Intution :-

1. find inorder of both trees, in an array
2. merge both array.
3. Convert normal BST to Balanced BST concept. (prev. question)

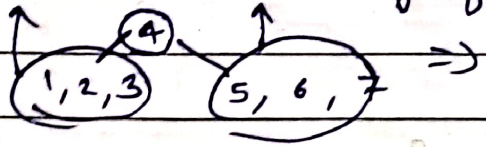
eg 

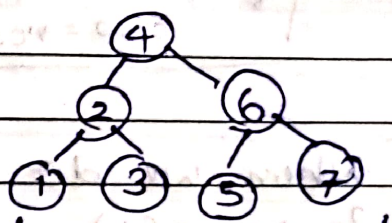


array 1 = 1, 2, 3 4 array 2 = 5, 6, 7

array 3 = 1, 2, 3, 4, 5, 6, 7

→ pro. divide array from middle





Send 1 to left and 5 to L.
3 to right 7 to R (right)

Balanced BST

TC - O(n+m)

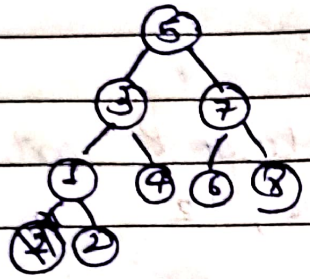
SC - O(n+m)

Date: 26/3/22 Page no:

Ques → K^{th} Largest element in BST

Given BST, write func" which will return K^{th} largest element without doing any changes in BST.

⊛ Example :-



K = 3

⇒ 1, 2, (3) 4, 5, (6) 7, 8

↑ (smallest) ↑ (largest)

Ans = 6

⊛ Intution :-

→ ∵ Inorder of any given property is always in sorted order ∵ left, root, right

→ Now to store BST, we are using extra space, to avoid it we can maintain counter, cnt = 0.

→ whenever we encounter a root, (i.e (lft root right). Inc cnt ++

Inorder ← if (cnt++ == K) ans = root

→ When we do morris traversal TC - O(n) & SC - O(1).

→ So, we use morris traversal instead of Recursive or iterative.

Date: __/__/__ Page no: ____

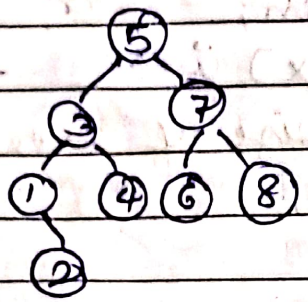
(*) Snippet :-

```
int ans; // global

void solve (node * root, int k, int & idx).
{
    if (! root) return;
    solve (root → left, k, idx);
    if (n-1<k == idx)
    {
        // n = size of tree
        ans = root → data;
        idx;
        return;
    }
    else idx++;
    solve (root → right, k, idx);
}
```

Ques → K^{th} Smallest element in BST.

Given K^{th} Smallest int in BST. find K^{th} smallest element in BST

Example



K = 3

1, 2, 3, 4, 5, 6, 7, 8

Ans = 3

* Intuition :-

→ we first find inorder traversal of given BST.

→ Maintain a counter if counter is equal index.

→ we'll get our answer.

a[] = 1, 2, 3, 4, 5, 6, 7, 8

index - 0, 2, 3, 4, 5, 6, 7

Small, largest

Eg. K = 3 (3rd Smallest / largest) ⇒ n - K Smallest / largest

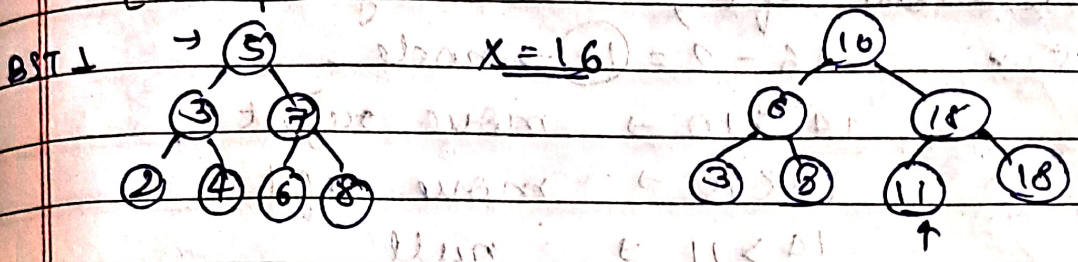
∴ 7 - 3 = 5th ans

Date: / / Page no:

Count pairs from 2 BST whose sum is equal to given value "x".

Given 2 BST with N1 & N2 distinct node. & a given value X. Task is return count of all pair whose sum is equal to X from Both the BST.

Example



16 - 5 = 11 ⇒ (5, 11)

16 - 3 = 13 X ; 16 - 2 = 14 X ; 16 - 8 = 8 ⇒ (8, 8)

return count.

* Intution :-

→ Take unordered map & store the element inside the map, on BST2.

→ Run inorder traversal on BST1. and check if (X - <element>) == map. i.e 16 - 5 = 11, present in map or not.

→ if present ↑se count (count++).

* return count.

TC - O(n+m)

SC - O(m)

Date: / / Page no.

① Optimal soln (remove extra space)

→ Find inorder of BST 1, and then check for; (x - <element>) = node, that node present in BST 2 or not.

→ now if node is < element move left, else right.

i.e ↓ 16 - 2 = (14) → node.

14 > 10 → move right.

14 < 15 → move left

14 > 11 → null

(not found)

16 - 6 = 10, ⇒ (6, 10) found.

16 - 5 = 11 → 11 > 10 → move right

11 < 15 → left

(5, 11) 11 = 11 → found

TC - O(n x height(BST2))

SC - ~~O(height)~~ ⇒ O(1)

Date: / / Page no:

* Implementation :-

```
void solve (node *root1, node* root2, int k)
{
    if (! root1) return;
solve (root1→ left, root2, k);
  if ( fetch (root2, k, - root1→ data))
        c++;
Solve (root1 → right, root2, k);
}

int countPair (node* root1, node* root2,
  int x)
{
    c = 0;
    Solve (root1, root2, x)
    return c;
}
```

27/3/22

Ques → Find median of BST in O(n) time

Given BST, find median of it.

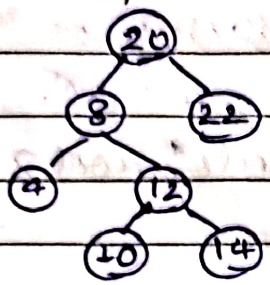
If no. of nodes are even :-

median : (n/2^th node + (n+1)/2^th node) / 2.

If no. of nodes are odd :-

median = (n+1)/2^th node.

* Example :-



median = 12

no. of nodes = 7 (odd)

* Intution :-

→ To find median, we need to find inorder of BST (because we'll get sorted order)

Then we'll find the median

→ ∵ we are not allowed to use extra space so for inorder, recursion & iterative approach can't be applied.

→ we can use Morris inorder traversal coz it doesn't require extra space.

Date: / / Page no:

Implementation :-

count no. of node in given BST, using morris Inorder.

Then perform morris Inorder one more time by counting nodes & by checking if count is equal to the median point.

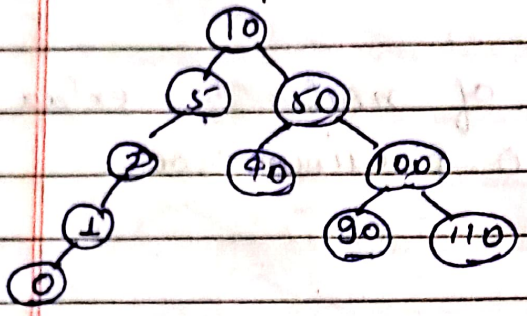
To consider even no. of nodes an extra pointer pointing to previous node is used.

Ques → Count BST That lies in given range

Given BST, and range [L, H], count no. of nodes in BST that lies in given range.

- ⊙ Value smaller than root go left side.
- ⊙ value greater or equal to root go right.

Example :-



l = 5 , h = 45

5 ≤ node , node ≤ 45

10, 5, 40 ⇒ 3 Ans.

✱ Intution :-

(1) Let say we pick (10) from above exm. 5 ≤ 10 ≤ 45, then I'll add 1 with left recursion + right recursion

⇒ 1 + (left) + right.

(2) Let say we've (2), & now check if if (x < l) → (true) (right recursion) = (∵, no use of calculating 2 & below)

(3) Let say we've (50) now check if (x > h) → (true) (left recur.). (∵, no use of calculating value greater than h).

Date: / / Page no:

Implementation :-

```
int getCount (node * root, int l, int h)
{
    if (!root) return 0;
    if (root → data >= l and root → data <= h)
        return 1 + getCount (root → left, l, h)
        + getCount (root → right, l, h);
    else if (root → data < l)
        return getCount (root → right, l, h);
    else
        return getCount (root → left, l, h);
}
```

Ques → Given n appointments, find all conflicting appointments

Given 'n' appointment, find all conflicting appointment.

⊛ Example :-

appointment = { {1,5}, {3,7}, {2,6}, {10,15}
{5,6}, {4,100} }

output → [3,7] conflict with [1,5]
[2,6] conflict with [1,5]
[5,6] " " [3,7].
[4,100] " " [1,5].

conflict means : if it conflict with any other previous app. in array.

⊛ Intution :

One by one process all appointment from the second appointment to last.

For every appo. i, check if it conflicts with i-1, i-2, ... 0.

we use Interval Tree to solve problem in O(nlogn) time.

⊛ Interval Tree

⊙ Create Interval Tree, init. with first appointment

⊙ Do the follow for all other appointment start from second one.

(a) Check if current app. conflict with any of the existing appointment in IT.

If conflicts, then print the current appointment.

(This step can be done in O(log n) time

(b) Insert current appointment in interval tree.

TC - O(log n).

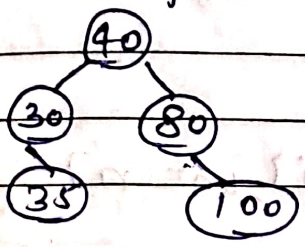
Ques: Check if given array can be represent preorder Traversal of BST.

Given an array of nos, return True if given array can represent pre-order traversal of BST else false.

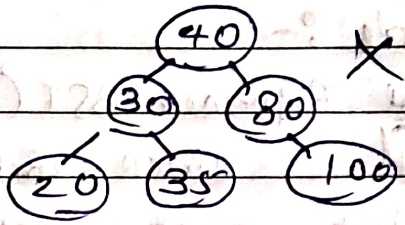
* Example :-

Input pre = {40, 30, 35, 80, 100}

ans/op = True.



Input -> {40, 30, 35, 20, 80, 100}



✗ NO (False

* Intuition :-

We can use stack to solve the prob. Here, we find next greater element & after finding next greater, if we find smaller element, return false.

(1) Create empty stack, and initialize root as INT_MIN.

Do The follow, for every element pre[i].

(a) if pre[i] is smaller than current root, return false.

(b) Keep removing ele from stack while pre[i] is greater than stack top. Make last removed item as new root (to be compared next).
At this point, pre[i] is greater than the removed root.

(c) push pre[i] to stack (all element in stack are in rising order)

* Code :-

```
bool RepresentBST(int pre[], int n)
{
    stack<int> s;
    int root = INT_MIN;
    for(int i=0; i<n; i++)
    {
        if (pre[i] < root)
            return false;
        while(!s.empty() && s.top() < pre[i])
        {
            root = s.top();
            s.pop()
        }
        s.push(pre[i]);
    }
    return true;
}
```

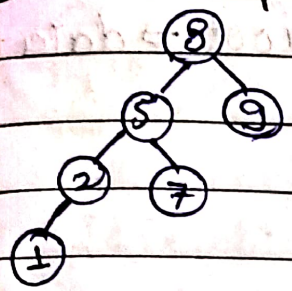
Date: / / Page no:

Check whether BST contain Dead end or not

Given BST, contain +ve int. value greater than 0. check whether BST contain dead end or not.

Dead end mean: we are not able to insert element after that node

(*) Example :-



O/p - Yes

1 is dead ent bcoz we can't insert element after that.

(*) Intution :-

→ We basically need to check if there is a leaf node with value x such that x+1 and x-1 exist in BST with expe exception of x=1.

→ For x=1, we can't insert 0 as the prob. say BST contain +ve integers.

IDEA

→ We first traverse the whole BST & store and all node in a set. we also store all leaf in a siperate hash to avoid re-traversal of BST.

→ Finally check for every leaf node x, if x-1, & x+1 present in Set or not

Recursive Approch :-

```
bool solve(node* root, int min, int max)
{
    if(!root) return false;
    if(min == max) return true;

return solve(root->left, min, root->data-1)
    || solve(root->right, root->data+1, max);
}

bool dead(node * root)
{
    return solve(root, 1, INT_MAX);
}
```

Date 30/3/22 Page no.

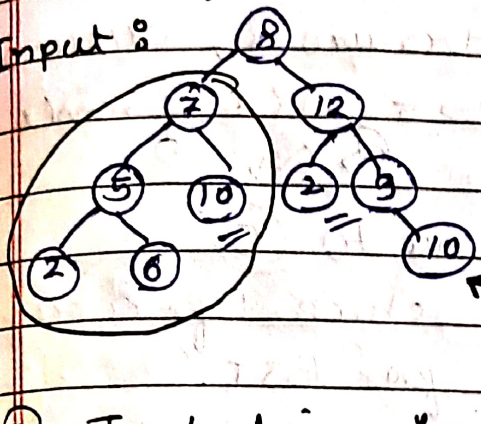
Largest BST in binary Tree

Given a bt. find the size of its largest subtree that is a BST.

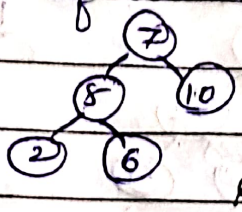
NOTE: Here, size = no. of nodes in subtree

Example :-

Input :



out of all subtree is largest BST.



Ans → 5 (size)

* Intution :- (Refer above example)

(1) Let say we pick (8) and check for the largest element in its left side, whether largest element of left is > (8), if its greater than (8) then its can't be a BST.

(2) Same thing we'll check with right side whether all the element we find the smallest element and that element is greater than (8), that mean all the elements are greater than (8) else can't be a BST.

Example

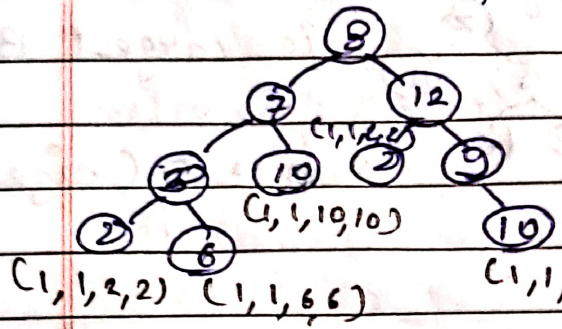
In left → compare (8) with (10)

In right → compare (8) with (2)

Approach :- (BST, size, min, max)

→ main 4 variable. (i) check whether particular node is forming BST or not. (1 and 0)
(ii) store size
(iii) minimum value, (iv) Maximum value

→ we'll use post order traversal, recursively
↓ 8 → 7 → 5 → 2 (leaf node)



→ move to (5)
check left & right for BST.
(∵ 1st variable is 1, 1 in (2) & (6) hence (5) can form BST)
(5) become 1, 3, 2, 6)

→ Move to (7)
if (7 > 6 and 7 < 10) return true
(7) become (1, 5, 2, 10),

→ Move to (9) (have left null)
Null = (1, 0, max, min). ∵ 9 > min, 9 < max.
∴, 9 become → 1, 2, 9, 10)

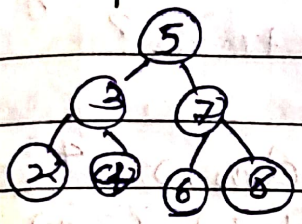
→ Move to (12)
can't form bst coz 10 < 12 on right side

Date: / / Page no:

Flatten BST to sorted list

Given BST, task is to flatten it to sorted list. Precisely, the value of each node must be lesser than the value of all the node at its right & its left node must be NULL after flattening.

Example :-



flatten - 2, 3, 4, 5, 6, 7, 8.

Intution :-

→ re-Create BST from its inorder traversal. it will tak O(N) extra space.

we can apply inorder to BST as :

→ ① Create a dummy node.
② Create variable 'prev' & make it point to the dummy node.
③ perform inorder traversal & at each step :
- set prev → right = curr
- set prev → left = NULL
- set prev = curr

Improve SC to O(H).