b'

Amazon Interview Experience

- Difficulty Level :\nMedium
- Last Updated :\n16 Oct, 2018

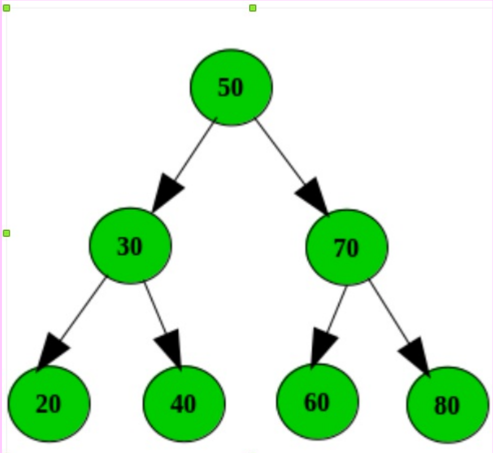
Round: I
An Array arr={7, 7, 8, 8, 9, 1, 1, 4, 2, 2} has numbers appearing twice or once. Duplicates appear side by side everytime. Might be few numbers can be occur one time and just assume this is a right rotating array (just say an array can rotate k times towards right). Aim is to identify numbers that occured once in array.

```
#include <stdio.h>
\xc2\xa0\xc2\xa0
int main()
{
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0int a[] = { 7, 7, 8, 8, 9, 1, 1, 4, 2, 2 }, i = 1, m = 10;
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// int a[]={7, 8, 8, 9, 1, 1, 4, 2, 2, 7}, i=1, m=10;
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0if (a[0] == a[m - 1]) {
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0i = 2;
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0m = m - 1;
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0}
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0else
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0printf("%d\\n", a[m - 1]); // For cases like { 7, 7, 8, 8, 9, 1, 1, 4, 2, 3 }, a[] = { 7, 7, 8, 8, 9, 1, 1, 4, 4,
2 }
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0for (; i < m; i++)
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0if (a[i] == a[i - 1])
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0i++;
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0else
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0printf("%d\\n", a[i - 1]);
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0return 0;
}
```

Input:
7, 7, 8, 8, 9, 1, 1, 4, 2, 2
Output:
9
4

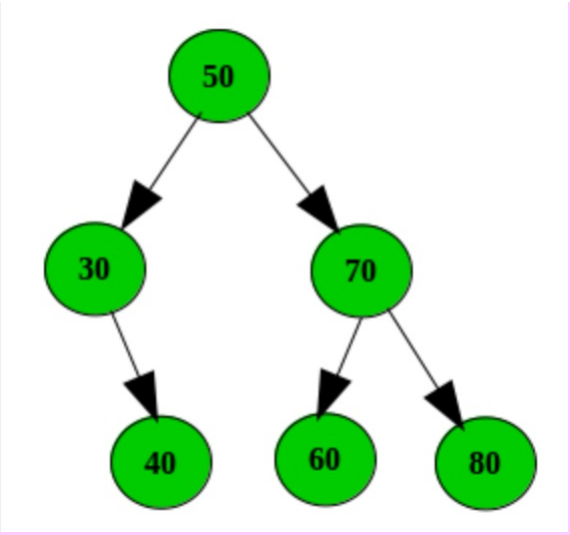
Round II:
In second round a question asked was asked regarding BST. Sum of key elements in individual path of BST (just say this sum as path_ weight) and threshold path weight is given as input. If any of the pathweight is less than oversold path weight then that should be deleted from the tree.

Example :
Input:
Consider the below tree as input and threshold path weight is 110.



Output:
Below are the no of paths the input tree can make
path 1 : 50 -> 30 -> 20, Sum = 100
path 1 : 50 -> 30 -> 40, Sum = 120
path 1 : 50 -> 70 -> 60, Sum = 180
path 1 : 50 -> 70 -> 80, Sum = 200

In the current scenario path is less than the threshold path weight (100 < 110), So we have to destroy the path 1.
Tree after destroying the path 1.



Program for above task:

```
#include <stdio.h>
#include <stdlib.h>
\xc2\xa0\xc2\xa0
struct node {
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0int key;
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0struct node *left, *right;
};
\xc2\xa0\xc2\xa0
struct node* newnode(int element)
{
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0struct node* temp = (struct node*)malloc(sizeof(struct node));
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0temp->key = element;
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0temp->left = temp->right = NULL;
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0return temp;
}
\xc2\xa0\xc2\xa0
struct node* insert(struct node* root, int element)
{
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0if (root == NULL)
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0return newnode(element);
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0if (element < root->key)
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0root->left = insert(root->left, element);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0else if (element > root->key)
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0root->right = insert(root->right, element);
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0return root;
}
\xc2\xa0\xc2\xa0
void inorder(struct node* mynode)
{
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0if (mynode != NULL) {
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0inorder(mynode->left);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0printf("%d\\t", mynode->key);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0inorder(mynode->right);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0}
}
\xc2\xa0\xc2\xa0
struct node* minValueNode(struct node* node)
{
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0struct node* current = node;
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0/* loop down to find the leftmost leaf */
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0while (current->left != NULL)
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0current = current->left;
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0return current;
}
\xc2\xa0\xc2\xa0
struct node* deleteNode(struct node* root, int key)
{
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// base case
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0if (root == NULL)
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0return root;
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// If the key to be deleted is smaller than the root\'s key,
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// then it lies in left subtree
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0if (key < root->key)
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0root->left = deleteNode(root->left, key);
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// If the key to be deleted is greater than the root\'s key,
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// then it lies in right subtree
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0else if (key > root->key)
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0root->right = deleteNode(root->right, key);
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// if key is same as root\'s key, then This is the node
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// to be deleted
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0else {
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// node with only one child or no child
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0if (root->left == NULL) {
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0struct node* temp = root->right;
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0free(root);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0return temp;
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0}
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0else if (root->right == NULL) {
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0struct node* temp = root->left;
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0free(root);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0return temp;
```

```
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0}
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// node with two children: Get the inorder successor (smallest
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// in the right subtree)
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0struct node* temp = minValueNode(root->right);
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// Copy the inorder successor\'s content to this node
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0root->key = temp->key;
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// Delete the inorder successor
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0root->right = deleteNode(root->right, temp->key);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0}
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0return root;
}
\xc2\xa0\xc2\xa0
/** Function : checkpathCost(int pathweight, int thresoldPathWeight,\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc
node *currentNode, struct node *root)
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0This function will sum the all individual elements in a every\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0path and lets say it as pathweight and if the sum less than\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0key then the path will be deleted.
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0Arguments:\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0pathweight\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0 : Sum of all elements in a path. As we are
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc
in recursive mode we the sum of\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc
elements.
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0thresoldPathWeight : Threshold value to compare the path weight
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0currentNode\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0 : CUrrent node in the tree.
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0root\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0 : root of the tree, used while
node.\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0Output : will delete paths which not satisfy the condition\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0(pathweight<thresoldPathWeight).\xc2\xa0\xc2\xa0\xc2\xa0
**/
checkpathCost(int pathweight, int thresoldPathWeight,
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0struct node* currentNode, struct node* root)
{
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// Add the current key to ath weight.
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0pathweight = pathweight + currentNode->key;
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// Check the conditionh if current node is a leaf node.
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0if ((currentNode->left == NULL) && (currentNode->right == NULL)) {
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// If pathweight is less that thresoldPathWeight delete the path
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0if (pathweight < thresoldPathWeight) {
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// we need to delete the node, so there is no necessity
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// in maintaining the current->key in pathweight.
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0pathweight = pathweight - currentNode->key;
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0printf("need to Delete %d.\\n", currentNode->key);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0deleteNode(root, currentNode->key);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0checkpathCost(0, thresoldPathWeight, root, root);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0}
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0else
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// If currentNode->key satisfies the condition
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// (pathweight > thresoldPathWeight), need to backtrack
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// back to check with other paths and no necessity in
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// maintaining the current->key in pathweight.
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0pathweight = pathweight - currentNode->key;
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0}
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// If currentNode is not a leaf traverse all the
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0// possible paths.
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0if (currentNode->left != NULL) {
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0checkpathCost(pathweight, thresoldPathWeight,
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0cu
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0}
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0if (currentNode->right != NULL) {
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0checkpathCost(pathweight, thresoldPathWeight,
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0cu
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0}
}
\xc2\xa0\xc2\xa0
int main()
{
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0int thresoldPathWeight = 110;
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0/* Let us create following BST
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa050
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0/\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0 \\
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa030\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0 70
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0/\xc2\xa0 \\\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0 /\xc2\xa0 \\
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa020\xc2\xa0\xc2\xa0 40\xc2\xa0 60\xc2\xa0\xc2\xa0 80 */
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0struct node* root = NULL;
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0root = insert(root, 50);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0root = insert(root, 30);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0root = insert(root, 20);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0root = insert(root, 40);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0root = insert(root, 70);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0root = insert(root, 60);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0root = insert(root, 80);
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0printf("Inorder traversal of the given tree \\n");
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0inorder(root);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0printf("\\n");
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0checkpathCost(0, thresoldPathWeight, root, root);
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0printf("Inorder traversal of the given tree \\n");
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0inorder(root);
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0printf("\\n");
\xc2\xa0\xc2\xa0
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0return 0;
}
```

In addition to the existing code for BST in geeksforgeeks and below function will implement the above task.
https://www.geeksforgeeks.org/binary-search-tree-set-2-delete/

My Personal Notes\n*arrow_drop_up*

Add your personal notes her

Save

'