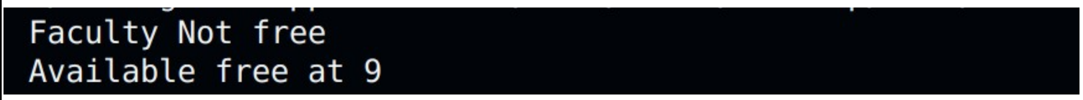
DATA STRUCTURES TUT - 7

Name-Mradul varshney
Eno-9921103137

Sol – 1 -

```
#include<bits/stdc++.h>
using namespace std;
class Lecture{
public:
int from;
int to;
string subject;
string facName;
string roomNum;
string batches;
Lecture* next;
};

Lecture* newNode(int f,int t,string sub,string fac,string num,string ba){
Lecture* temp = new Lecture;
temp->from = f;
temp->to = t;
temp->subject = sub;
temp->facName = fac;
temp->roomNum = num;
temp->batches = ba;
temp->next = NULL;
return temp;
}

void addInList(Lecture* arr[],int index,Lecture* node){
if(arr[index]==NULL){
arr[index] = node;
}
else{
Lecture* head = arr[index];
while(head->next !=NULL) head=head->next;
head->next = node;
}
}

void printList(Lecture* arr[]){
string days[] = {"MON","TUE","WED","THUR","FRI","SAT"};
for(int i = 0 ; i < 6 ; i++){
Lecture* head = arr[i];
cout<<days[i]<<"->";
while(head!=NULL){
cout<<"[ "<<head->subject<<" "<<head->from<<"-"<<head->to<<" "<<head-
>facName<<" "<<head->roomNum<<" "<<head->batches<<" ]\t";
head=head->next;
}
cout<<endl;
}
}
```

```
bool isFree(Lecture* arr[],int day,int frm,int t,string fac){
Lecture* head = arr[day];
while(head!=NULL){
if(head->facName == fac && ((frm <= head->to && frm >= head->from) || (t <= head->to
&& t >= head->from))){
return false;
}
head=head->next;
}
return true;
}

int isAvailable(string bat,int day,Lecture* arr[]){
Lecture* head = arr[day];
if(head==NULL) return 9;
int free = 9;
while(head!=NULL){
if(head->batches==bat){
free++;
}
else break;
head=head->next;
}
return free;
}

int main(){
Lecture* heads[6] = {NULL,NULL,NULL,NULL,NULL,NULL};
addInList(heads,0,newNode(9,10,"DS Lec","KSW","LT-7","F5"));
addInList(heads,0,newNode(11,12,"DS Tut","SHS","LT-3","F6"));
addInList(heads,0,newNode(15,16,"DBBW tut","AHS","LT-4","F2"));
addInList(heads,1,newNode(12,13,"DS Lec","KSW","LT-7","F5"));
addInList(heads,2,newNode(12,13,"DBBW Lec","KTR","LT-5","F7"));
addInList(heads,3,newNode(12,13,"DS Lec","PKM","LT-8","F3"));
addInList(heads,3,newNode(16,17,"DS Tut","SUK","LT-7","F8"));
addInList(heads,5,newNode(12,13,"DS Lec","SHS","LT-7","F6"));
printList(heads);
if(isFree(heads,5,12,2,"SHS")){
cout<<"Faculty is free"<<endl;
}
else{
cout<<"Faculty Not free"<<endl;
}
if(isAvailable("F6",0,heads)){
cout<<"Available free at "<<isAvailable("F6",0,heads)<<endl;
}
else{
cout<<"Batch is not free"<<endl;
}
return 0;
}
```



```
Faculty Not free
Available free at 9
```

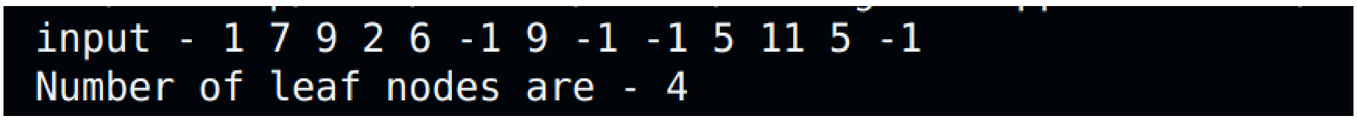
Sol – 2 -

```
function printRowWise(){
qu.enqueue(root);
while (!qu.empty()){
currNode = qu.topNode();
print(currNode->val);
if(currNode->down) qu.push(currNode->down);
qu.dequeue();
while (currNode!=NULL){
currNode = currNode->right;
if(currNode->down) qu.enqueue(currNode->down);
print(currNode->val);
}
}
}

function printColWise(){

make an Array containing the elements of the multilist in rowWisePrinting format as in
printRowWise function, say RowOrder[];

And at the same time make an array for the levels increasing from
left to right, say lvl[];

Now with the help of these two arrays make a matrix containing
elements for each lvl;

mat[lvl[j]][i] = RowOrder[j]; here i correspons to free index at
lvl[j] row of the matrix;

now print the elements from the matrix;

}
```

Sol – 3 -

```
#include<bits/stdc++.h>
using namespace std;

class TreeNode{
public:
int val;
TreeNode* left = NULL;
TreeNode* right = NULL;
TreeNode(){}
};
TreeNode* newNode(int data){
TreeNode* temp = new TreeNode;
temp->val = data;
temp->left = NULL;
temp->right = NULL;
return temp;
}
```

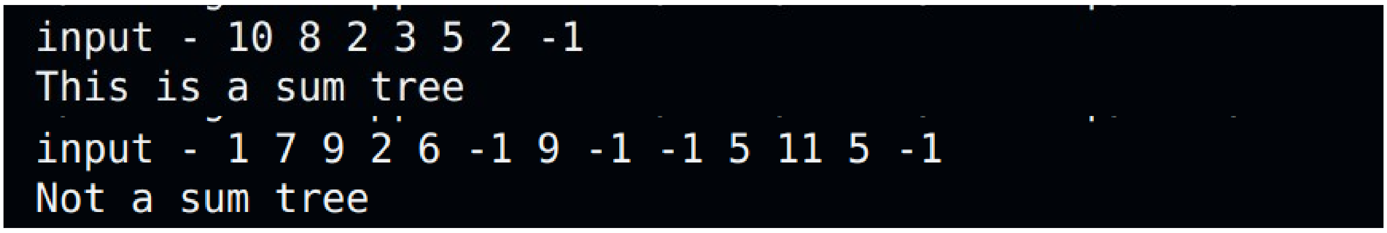
```
TreeNode* insertIterative(vector<int>& vec){
int n = vec.size();
queue<TreeNode*> qu;
if(n==0) return NULL;
vector<int> ip = vec;
TreeNode* root = newNode(vec[0]);
qu.push(root);
int i =1;
while(!qu.empty() && i < ip.size()){
TreeNode* currNode = qu.front();
qu.pop();
int leftVal = ip[i++];
if(leftVal!=-1){
currNode->left = newNode(leftVal);
qu.push(currNode->left);
}
if(i>=ip.size()) break;
int rightVal = ip[i++];
if(rightVal!=-1){
currNode->right = newNode(rightVal);
qu.push(currNode->right);
}

}
return root;
}

int countLeaf(TreeNode* root){
if(root==NULL) return 0;
if(root->left == NULL && root->right == NULL){
return 1;
}
return countLeaf(root->left) + countLeaf(root->right);
}

int main(){
vector<int> input = {1,7,9,2,6,-1,9,-1,-1,5,11,5,-1};
cout<<"input - ";
for(auto i : input){
cout<<i<<" ";
}
TreeNode* root = insertIterative(input);
cout<<"Number of leaf nodes are - ";
cout<<countLeaf(root);
return 0;
}
```



```
input - 1 7 9 2 6 -1 9 -1 -1 5 11 5 -1
Number of leaf nodes are - 4
```

Sol – 4 -

```
#include<bits/stdc++.h>
using namespace std;

class TreeNode{
public:
int val;
TreeNode* left = NULL;
TreeNode* right = NULL;
TreeNode(){}
};
TreeNode* newNode(int data){
TreeNode* temp = new TreeNode;
temp->val = data;
temp->left = NULL;
temp->right = NULL;
return temp;
}

TreeNode* insertIterative(vector<int>& vec){
int n = vec.size();
queue<TreeNode*> qu;
if(n==0) return NULL;
vector<int> ip = vec;
TreeNode* root = newNode(vec[0]);
qu.push(root);
int i =1;
while(!qu.empty() && i < ip.size()){
TreeNode* currNode = qu.front();
qu.pop();
int leftVal = ip[i++];
if(leftVal!=-1){
currNode->left = newNode(leftVal);
qu.push(currNode->left);
}
if(i>=ip.size()) break;
int rightVal = ip[i++];
if(rightVal!=-1){
currNode->right = newNode(rightVal);
qu.push(currNode->right);
}

}
return root;
}

bool isSumTree(TreeNode* root){
if(root==NULL) return true;
if(root->left && root->right) {
if(root->left->val + root->right->val == root->val) return true && isSumTree(root->left) &&
isSumTree(root->right);
else return false;
}
else if(root->left){
if(root->left->val == root->val) return true && isSumTree(root->left);
else return false;
}
else if(root->right){
if(root->right->val==root->val) return true && isSumTree(root->right);
else return false;
}
else return true;
}

int main(){
// vector<int> input = {10,8,2,3,5,2,-1};
vector<int> input = {1,7,9,2,6,-1,9,-1,-1,5,11,5,-1};
TreeNode* root = insertIterative(input);
cout<<"input - ";
for(auto i : input) cout<<i<<" ";
cout<<endl;
if(isSumTree(root)) cout<<"This is a sum tree"<<endl;
else cout<<"Not a sum tree"<<endl;
return 0;
}
```

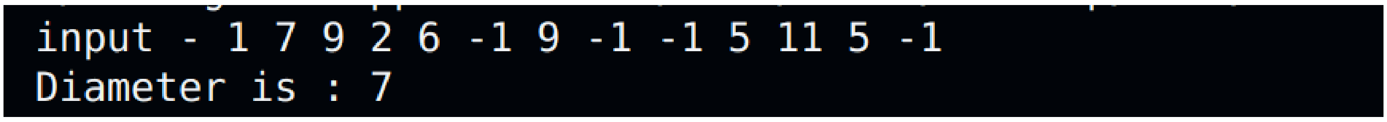


```
input - 10 8 2 3 5 2 -1
This is a sum tree
input - 1 7 9 2 6 -1 9 -1 -1 5 11 5 -1
Not a sum tree
```

Sol – 5 -

```
#include<bits/stdc++.h>
using namespace std;

class TreeNode{
public:
int val;
TreeNode* left = NULL;
TreeNode* right = NULL;
TreeNode(){}
};
TreeNode* newNode(int data){
TreeNode* temp = new TreeNode;
temp->val = data;
temp->left = NULL;
temp->right = NULL;
return temp;
}

TreeNode* insertIterative(vector<int>& vec){
int n = vec.size();
queue<TreeNode*> qu;
if(n==0) return NULL;
vector<int> ip = vec;
TreeNode* root = newNode(vec[0]);
qu.push(root);
int i =1;
while(!qu.empty() && i < ip.size()){
```

```
TreeNode* currNode = qu.front();
qu.pop();
int leftVal = ip[i++];
if(leftVal!=-1){
currNode->left = newNode(leftVal);
qu.push(currNode->left);
}
if(i>=ip.size()) break;
int rightVal = ip[i++];
if(rightVal!=-1){
currNode->right = newNode(rightVal);
qu.push(currNode->right);
}

}
return root;
}

int max(int a,int b){
int ans;
a>b ? ans = a : ans = b;
return ans;
}

int maxDepth(TreeNode* root) {
if(root==NULL) return 0;
return max(maxDepth(root->left), maxDepth(root->right)) + 1;
}
int diameterOfBinaryTree(TreeNode* root) {
if(root==NULL) return 0;
int foundInLeft = diameterOfBinaryTree(root->left);
int foundInRight = diameterOfBinaryTree(root->right);
int foundWithRoot = (maxDepth(root->left) + maxDepth(root->right));
return max(foundInLeft,max(foundInRight,foundWithRoot)) + 1;
}

int main(){
vector<int> input = {1,7,9,2,6,-1,9,-1,-1,5,11,5,-1};
cout<<"input - ";
for(auto i : input) cout<<i<<" ";
cout<<endl;
TreeNode* root = insertIterative(input);
cout<<"Diameter is : ";
cout<<diameterOfBinaryTree(root)<<endl;
return 0;
}
```

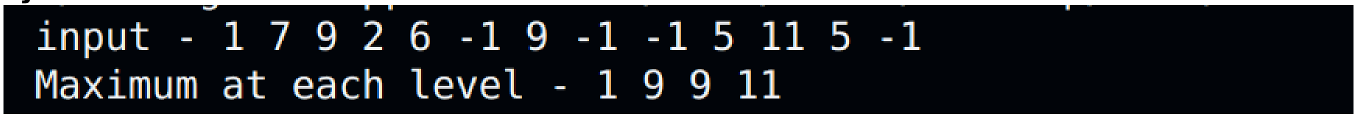


```
input - 1 7 9 2 6 -1 9 -1 -1 5 11 5 -1
Diameter is : 7
```

Sol – 6 -

```
#include<bits/stdc++.h>
using namespace std;

class TreeNode{
public:
int val;
TreeNode* left = NULL;
TreeNode* right = NULL;
TreeNode(){}
};
TreeNode* newNode(int data){
TreeNode* temp = new TreeNode;
temp->val = data;
temp->left = NULL;
temp->right = NULL;
return temp;
}

TreeNode* insertIterative(vector<int>& vec){
int n = vec.size();
queue<TreeNode*> qu;
if(n==0) return NULL;
vector<int> ip = vec;
TreeNode* root = newNode(vec[0]);
qu.push(root);
int i =1;
while(!qu.empty() && i < ip.size()){
TreeNode* currNode = qu.front();
qu.pop();
int leftVal = ip[i++];
if(leftVal!=-1){
currNode->left = newNode(leftVal);
qu.push(currNode->left);
}
if(i>=ip.size()) break;
int rightVal = ip[i++];
if(rightVal!=-1){
currNode->right = newNode(rightVal);
qu.push(currNode->right);
}

}
return root;
}

int max(int a,int b){
int ans;
a>b ? ans = a : ans = b;
return ans;
}

int maxDepth(TreeNode* root) {
if(root==NULL) return 0;
return max(maxDepth(root->left), maxDepth(root->right)) + 1;
}
```

```
int mat[100][100]; //level, value
void makeMatrix(){
for(int i=0;i<100;i++){
for(int j = 0; j<100 ; j++){
mat[i][j] = -1;
}
}
}

void insertAtLvl(int val,int lvl){
for(int i = 0 ; i < 100 ; i++){
if(mat[lvl][i]==-1){
mat[lvl][i] = val;
break;
}
}
}

void makeEachLevel(TreeNode* root,int level=0){
if(root==NULL) return;
insertAtLvl(root->val,level);
makeEachLevel(root->left,level+1);
makeEachLevel(root->right,level+1);
}

void maxAtEachLvl(int n){
int ans[n];
for (int i = 0; i < n; i++){
int maxi = 0;
for (int j = 0; j < n; j++){
maxi = max(maxi,mat[i][j]);
}
ans[i] = maxi;
}
for (int i = 0; i < n; i++){
cout<<ans[i]<<" ";
}
cout<<endl;
}

void printMatrix(int n){
for(int i=0;i<n;i++){
for(int j = 0; j<n ; j++){
cout<<mat[i][j]<<" ";
}
cout<<endl;
}
}

int main(){
makeMatrix();
vector<int> input = {1,7,9,2,6,-1,9,-1,-1,5,11,5,-1};
cout<<"input - ";
for(auto i : input) cout<<i<<" ";
cout<<endl;
TreeNode* root = insertIterative(input);
int depth = maxDepth(root);
makeEachLevel(root);
cout<<"Maximum at each level - ";
maxAtEachLvl(depth);
return 0;
}
```



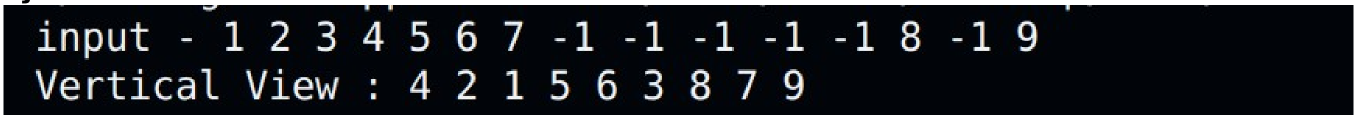
```
input - 1 7 9 2 6 -1 9 -1 -1 5 11 5 -1
Maximum at each level - 1 9 9 11
```

Sol – 7 -

```
#include<bits/stdc++.h>
using namespace std;

class TreeNode{
public:
int val;
TreeNode* left = NULL;
TreeNode* right = NULL;
TreeNode(){}
};
TreeNode* newNode(int data){
TreeNode* temp = new TreeNode;
temp->val = data;
temp->left = NULL;
temp->right = NULL;
return temp;
}

TreeNode* insertIterative(vector<int>& vec){
int n = vec.size();
queue<TreeNode*> qu;
if(n==0) return NULL;
vector<int> ip = vec;
TreeNode* root = newNode(vec[0]);
qu.push(root);
int i =1;
while(!qu.empty() && i < ip.size()){
TreeNode* currNode = qu.front();
qu.pop();
int leftVal = ip[i++];
if(leftVal!=-1){
currNode->left = newNode(leftVal);
qu.push(currNode->left);
}
if(i>=ip.size()) break;
int rightVal = ip[i++];
if(rightVal!=-1){
currNode->right = newNode(rightVal);
qu.push(currNode->right);
}

}
return root;
}
```

```
int *orderArr = new int[100];
int orderArrIndex = 0;
int *levels = new int[100];
int levelsIndex=0;
int mat[100][100]; //level, value


int max(int a,int b){
int ans;
a>b ? ans = a : ans = b;
return ans;
}

int maxDepth(TreeNode* root) {
if(root==NULL) return 0;
return max(maxDepth(root->left), maxDepth(root->right)) + 1;
}
void makeMatrix(){
for(int i=0;i<100;i++){
for(int j = 0; j<100 ; j++){
mat[i][j] = -1000;
}
}
}

void insertAtLvl(int val,int lvl){
if(lvl < 0){
lvl = 100 + lvl;
}
for(int i = 0 ; i < 100 ; i++){
if(mat[lvl][i]==-1000){
mat[lvl][i] = val;
break;
}
}
}

void printMatrix(int n){
int i = 99;
for(; i>=0 ; i--){
if(mat[i][0]==-1000){
break;
}
}i++;
for(;;i++){
if(i==100) i = 0;
if(mat[i][0]==-1000){
break;
}
for(int j = 0; j<n ; j++){
if(mat[i][j]!=-1000) cout<<mat[i][j]<<" ";
}
}
}

void makeEachLevel(TreeNode* root,int n){
```

```
for(int i = 0 ; i < n ; i++){
insertAtLvl(orderArr[i],levels[i]);
}
}

//MAKING ARRAY FOR TRAVERSAL

void putInArr(TreeNode* root, int level, int inLevels=0)
{
if (root == NULL || level < 0)
return;
if (level == 1){
orderArr[orderArrIndex++] = root->val;
levels[levelsIndex++] = inLevels;
}
else if (level > 1) {
putInArr(root->left, level - 1,inLevels-1);
putInArr(root->right, level - 1,inLevels+1);
}
}

void getLevelOrder(TreeNode* root)
{
int h = maxDepth(root);
int i;
for (i = 1; i <= h; i++)
putInArr(root, i);
}

int main(){
vector<int> input = {1,2,3,4,5,6,7,-1,-1,-1,-1,-1,8,-1,9};
// vector<int> input = {1,7,9,2,6,-1,9,-1,-1,5,11,5,-1};
cout<<"input - ";
for(auto i : input) cout<<i<<" ";
cout<<endl;
int numInp = 0;
for(auto i:input){
if(i!=-1) numInp++;
}
TreeNode* root = insertIterative(input);
int n = maxDepth(root);
makeMatrix();
getLevelOrder(root);
makeEachLevel(root,numInp);
cout<<"Vertical View : ";
printMatrix(n);cout<<endl;
return 0;
}
```



```
input - 1 2 3 4 5 6 7 -1 -1 -1 -1 -1 8 -1 9
Vertical View : 4 2 1 5 6 3 8 7 9
```