**SHUBHAM GARG**

**9919103057**

**BATCH: F2**

**LAB ASSIGNMENT**

**BINARY TREES**

**(20-25 April 2020)**

**Q1. WAP for a Binary Search Tree that returns the maximum and minimum values in it. (using recursion)**

#include <stdio.h>

#include <stdlib.h>

struct Node

{

struct Node \*lchild;

int data;

struct Node \*rchild;

}\*root=NULL;

void Insert(int key)

{

struct Node \*t=root;

struct Node \*r=NULL,\*p;

if(root==NULL)

{

p=(struct Node \*)malloc(sizeof(struct Node));

p->data=key;

p->lchild=p->rchild=NULL;

root=p;

return;

}

while(t!=NULL)

{

r=t;

if(key<t->data)

t=t->lchild;

else if(key>t->data)

t=t->rchild;

else

return;

}

p=(struct Node \*)malloc(sizeof(struct Node));

p->data=key;

p->lchild=p->rchild=NULL;

if(key<r->data)

r->lchild=p;

else

r->rchild=p;

}

void maxmin()

{

struct Node \*t=root,\*r;

while(t!=NULL)

{

r=t;

t=t->lchild;

}

printf("The Minimum Value is %d \n",r->data);

t=root;

r=NULL;

while(t!=NULL)

{

r=t;

t=t->rchild;

}

printf("\n The Maximum Value is %d \n",r->data);

}

int main()

{

struct Node \*temp;

Insert(50);

Insert(10);

Insert(40);

Insert(20);

Insert(45);

Insert(60);

Insert(40);

maxmin();

return 0;

}

**Q2. Implement the following functions on Binary Trees:**

**a) Count the number of nodes**

**b) Count the number of leaves**

**c) Count the number of right children**

**d) Find the height of the tree**

#include <stdio.h>

#include <stdlib.h>

#include "Queue.h"

#include "Stack.h"

struct Node \*root=NULL;

void Treecreate()

{

struct Node \*p,\*t;

int x;

struct Queue q;

create(&q,100);

printf("Eneter root value ");

scanf("%d",&x);

root=(struct Node \*)malloc(sizeof(struct Node));

root->data=x;

root->lchild=root->rchild=NULL;

enqueue(&q,root);

while(!isEmpty(q))

{

p=dequeue(&q);

printf("eneter left child of %d ",p->data);

scanf("%d",&x);

if(x!=-1)

{

t=(struct Node \*)malloc(sizeof(struct Node));

t->data=x;

t->lchild=t->rchild=NULL;

p->lchild=t;

enqueue(&q,t);

}

printf("eneter right child of %d ",p->data);

scanf("%d",&x);

if(x!=-1)

{

t=(struct Node \*)malloc(sizeof(struct Node));

t->data=x;

t->lchild=t->rchild=NULL;

p->rchild=t;

enqueue(&q,t);

}

}

}

int count(struct Node \*root)

{

if(root)

return count(root->lchild)+count(root->rchild)+1;

return 0;

}

int height(struct Node \*root)

{

int x=0,y=0;

if(root==0)

return 0;

x=height(root->lchild);

y=height(root->rchild);

if(x>y)

return x+1;

else

return y+1;

}

int leafnodes(struct node\* q)

{

int cou = 0;

if(q != NULL)

{

leafnodes(q->lchild);

if((q->lchild == NULL) && (q->rchild == NULL))

{

cou++;

}

leafnodes(q->rchild);

}

return cou;

}

int rchildren(struct node \*q)

{

if(!q)

return 0;

if(q->rchild)

return 1 + rchildren(q->rchild) + rchildren(q->lchild);

else

return rchildren(q->lchild);

}

int main()

{

Treecreate();

printf(“%d”,count(root));

printf(“%d”,height(root));

printf(“%d”,leafnodes(root));

printf(“%d”,rchildren(root));

return 0;

}

“Queue.h”

struct Node

{

struct Node \*lchild;

int data;

struct Node \*rchild;

};

struct Queue

{

int size;

int front;

int rear;

struct Node \*\*Q;

};

void create(struct Queue \*q,int size)

{

q->size=size;

q->front=q->rear=0;

q->Q=(struct Node \*\*)malloc(q->size\*sizeof(struct Node \*));

}

void enqueue(struct Queue \*q,struct Node \*x)

{

if((q->rear+1)%q->size==q->front)

printf("Queue is Full");

else

{

q->rear=(q->rear+1)%q->size;

q->Q[q->rear]=x;

}

}

struct Node \* dequeue(struct Queue \*q)

{

struct Node\* x=NULL;

if(q->front==q->rear)

printf("Queue is Empty\n");

else

{

q->front=(q->front+1)%q->size;

x=q->Q[q->front];

}

return x;

}

int isEmpty(struct Queue q)

{

return q.front==q.rear;

}

**Q3. WAP to check if all the leaves of a Binary Tree are at same level or not.**

#include <stdio.h>

#include <stdlib.h>

#include "Queue.h"

struct Node \*root=NULL;

void Treecreate()

{

struct Node \*p,\*t;

int x;

struct Queue q;

create(&q,100);

printf("Eneter root value ");

scanf("%d",&x);

root=(struct Node \*)malloc(sizeof(struct Node));

root->data=x;

root->lchild=root->rchild=NULL;

enqueue(&q,root);

while(!isEmpty(q))

{

p=dequeue(&q);

printf("eneter left child of %d ",p->data);

scanf("%d",&x);

if(x!=-1)

{

t=(struct Node \*)malloc(sizeof(struct Node));

t->data=x;

t->lchild=t->rchild=NULL;

p->lchild=t;

enqueue(&q,t);

}

printf("enter right child of %d ",p->data);

scanf("%d",&x);

if(x!=-1)

{

t=(struct Node \*)malloc(sizeof(struct Node));

t->data=x;

t->lchild=t->rchild=NULL;

p->rchild=t;

enqueue(&q,t);

}

} }

int check( int level, int lLevel)

{

if (root == NULL)  return 1;

if (root->lchild == NULL && root->rchild == NULL)

{

if (lLevel == 0)

{

lLevel = level;

return 1;

}

return (level == lLevel);

}

return check(root->lchild, level+1, lLevel) &&  check(root->rchild, level+1, lLevel);

}

int main()

{

Treecreate();

int x=check(1,1);

if(x)

printf(“Leaves are at same Level”);

else

printf(“Leaves are NOT at same Level”);

return 0;

}

“Queue.h”

struct Node

{

struct Node \*lchild;

int data;

struct Node \*rchild;

};

struct Queue

{

int size;

int front;

int rear;

struct Node \*\*Q;

};

void create(struct Queue \*q,int size)

{

q->size=size;

q->front=q->rear=0;

q->Q=(struct Node \*\*)malloc(q->size\*sizeof(struct Node \*));

}

void enqueue(struct Queue \*q,struct Node \*x)

{

if((q->rear+1)%q->size==q->front)

printf("Queue is Full");

else

{

q->rear=(q->rear+1)%q->size;

q->Q[q->rear]=x;

}

}

struct Node \* dequeue(struct Queue \*q)

{

struct Node\* x=NULL;

if(q->front==q->rear)

printf("Queue is Empty\n");

else

{

q->front=(q->front+1)%q->size;

x=q->Q[q->front];

}

return x;

}

int isEmpty(struct Queue q)

{

return q.front==q.rear;

}

**Q4. WAP to check if a Binary Tree has duplicate values or not.**

#include <stdio.h>

#include <stdlib.h>

#include "Queue.h"

struct Node \*root=NULL;

void Treecreate()

{

struct Node \*p,\*t;

int x;

struct Queue q;

create(&q,100);

printf("Eneter root value ");

scanf("%d",&x);

root=(struct Node \*)malloc(sizeof(struct Node));

root->data=x;

root->lchild=root->rchild=NULL;

enqueue(&q,root);

while(!isEmpty(q))

{

p=dequeue(&q);

printf("eneter left child of %d ",p->data);

scanf("%d",&x);

if(x!=-1)

{

t=(struct Node \*)malloc(sizeof(struct Node));

t->data=x;

t->lchild=t->rchild=NULL;

p->lchild=t;

enqueue(&q,t);

}

printf("enter right child of %d ",p->data);

scanf("%d",&x);

if(x!=-1)

{

t=(struct Node \*)malloc(sizeof(struct Node));

t->data=x;

t->lchild=t->rchild=NULL;

p->rchild=t;

enqueue(&q,t);

}

} }

int check( unordered\_set<int> &s)

{

if (root == NULL)

return 0;

if (s.find(root->data) != s.end())

return 1;

s.insert(root->data);

return check(root->lchild, s) ||  check(root->rchild, s);

}

int checkDup()

{

unordered\_set<int> s;

return check( s);

}

int main()

{

Treecreate();

int x=checkDup();

if(x)

printf(“Duplicates are there”);

else

printf(“NO Duplicates”);

return 0;

}

“Queue.h”

struct Node

{

struct Node \*lchild;

int data;

struct Node \*rchild;

};

struct Queue

{

int size;

int front;

int rear;

struct Node \*\*Q;

};

void create(struct Queue \*q,int size)

{

q->size=size;

q->front=q->rear=0;

q->Q=(struct Node \*\*)malloc(q->size\*sizeof(struct Node \*));

}

void enqueue(struct Queue \*q,struct Node \*x)

{

if((q->rear+1)%q->size==q->front)

printf("Queue is Full");

else

{

q->rear=(q->rear+1)%q->size;

q->Q[q->rear]=x;

}

}

struct Node \* dequeue(struct Queue \*q)

{

struct Node\* x=NULL;

if(q->front==q->rear)

printf("Queue is Empty\n");

else

{

q->front=(q->front+1)%q->size;

x=q->Q[q->front];

}

return x;

}

int isEmpty(struct Queue q)

{

return q.front==q.rear;

}