Bridge course

Assignment 2 Name: A.Jyothi

Roll no.: m22s007

Date: 31-07-2022

\_------------------------------------------------------------------------------------------------------------------------------------

Section1

1. Write a program to implement a max or min (any one) heap with insert and delete functions.

#include<stdio.h>

#include<stdlib.h>

//Putting size globally for the heap.

void swap(int\* a, int\* b){

int temp = \*b;

\*b = \*a;

\*a = temp;

}

//size is nothing but the number of elements in the heap.

void insertheapify(int arr[], int size, int i){

//Given the index i where the node is inserted. Now we need to check if i's parent is greater

//than i (Heap property-> parent greater then child.)

int parent = (i-1)/2;

if(arr[parent] > 0){

if( arr[i] > arr[parent]){

swap(&arr[i],&arr[parent]);

insertheapify(arr, size, parent);

}

}

}

//Taking size as pointer as we are going to change the size value.

void insert(int arr[],int\* size, int num){

//increase the size by 1

\*size = \*size + 1;

//place the new element at the end.

arr[\*size-1] = num;

//heapify from the last element to root.

insertheapify(arr,\*size, \*size-1);

}

void deleteheapify(int arr[], int size, int i){

//Delete heapify starts from root.

//Initilly 0 for root.

int max = i;

//Find left and right child indices

int leftChild = 2\*i+1;

int rightChild = 2\*i+2;

//Left child and right child values should be less than max index value.If not swap.

if(leftChild < size && arr[leftChild]>arr[max]){

max = leftChild;

}

if(rightChild < size && arr[rightChild]> arr[max]){

max = rightChild;

}

//If max index value is not the root, then the swap the root with max index value and heapify

//recursively down the max index.

if(max!=i){

swap(&arr[max],&arr[i]);

deleteheapify(arr,size, max);

}

}

void delete(int arr[], int\*size){

//Find the last element

if(\*size == 0){

printf("Heap is empty\n");

return;

}

int l = arr[\*size-1];

//Replace the root with last element.

arr[0] = l;

\*size = \*size-1;

deleteheapify(arr,\*size, 0);

}

void printHeap(int arr[], int size){

int i =0;

printf("Heap Elements:\n");

for(int i =0; i< size; i++){

printf("%d ",arr[i]);

}

printf("\n");

}

int main(){

int arr[1000];

int size = 0;

while(1){

int op;

printf("Max Heap.\n 1. Insert\n 2. Delete\n 3.Print\n");

scanf("%d",&op);

switch(op){

case 1: int key;

printf("Please enter the value to be inserted:\n");

scanf("%d",&key);

insert(arr,&size,key);

break;

case 2: delete(arr,&size);

printf("Deleted root from Heap.");

break;

case 3: printHeap(arr,size);

break;

default: exit(1);

}

}

return(0);

}

2. Write a program to convert the max heap into the min heap in linear time.

//Converting Max heap to min heap is nothing but min heapifying an input array.

#include<stdio.h>

#include<stdlib.h>

void swap(int\*a, int\*b){

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void heapify(int arr[], int size, int i){

int smallest = i;

int leftChild = i\*2+1;

int rightChild = i\*2+2;

if(leftChild < size && arr[leftChild]< arr[smallest]){

smallest = leftChild;

}

if(rightChild < size && arr[rightChild] < arr[smallest]){

smallest = rightChild;

}

if(smallest!=i){

swap(&arr[smallest],&arr[i]);

heapify(arr,size,smallest);

}

}

void convertMaxToMinHeap(int arr[], int size){

int lastParent = (size-2)/2;

for(int i=lastParent; i>=0; i--){

heapify(arr,size,i);

}

}

void printHeap(int arr[], int size){

int i;

printf("HeapElements:\n");

for(int i =0; i <size; i++){

printf("%d ",arr[i]);

}

printf("\n");

}

int main(){

int arr[1000] = {10,5,6,2,4,3,1};

int size = 7;

convertMaxToMinHeap(arr,size);

printHeap(arr,size);

}

3. Write a program to check if an array represents min heap or not.

//Converting Max heap to min heap is nothing but min heapifying an input array.

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

void swap(int\*a, int\*b){

int temp = \*a;

\*a = \*b;

\*b = temp;

}

bool CheckParentLessThanChild(int arr[], int size, int i){

int smallest = i;

int leftChild = i\*2+1;

int rightChild = i\*2+2;

if(leftChild < size && arr[leftChild]< arr[smallest]){

smallest = leftChild;

}

if(rightChild < size && arr[rightChild] < arr[smallest]){

smallest = rightChild;

}

if(smallest!=i){

return false;

}

return true;

}

bool isMinHeap(int arr[], int size){

int lastParent = (size-2)/2;

for(int i=lastParent; i>=0; i--){

if(!CheckParentLessThanChild(arr,size,i)){

return false;

}

}

return true;

}

void printHeap(int arr[], int size){

int i;

printf("HeapElements:\n");

for(int i =0; i <size; i++){

printf("%d ",arr[i]);

}

printf("\n");

}

int main(){

int arr[1000] = {1,2,3,5,4,10,6};

int size = 7;

bool IsMin = isMinHeap(arr,7);

if(IsMin)

{

printf("Heap is MinHeap");

}

else{

printf("Heap is Not MinHeap");

}

}

4. Write a program to convert a binary search tree into a min heap

#include<iostream>

#include<vector>

#include<queue>

#include<string>

#include<utility>

using namespace std;

struct Node{

    int data;

    Node\* left;

    Node\* right;

    Node(){

    }

    Node(int data){

        this->data = data;

        this->left = nullptr;

        this->right = nullptr;

    }

};

//Inserting a node in binary search treee.

Node\* insert(Node\* root, int key){

    if(root == nullptr){

        return new Node(key);

    }

        if(key < root->data){

        root->left = insert(root->left,key);

    }

    else{

        root->right = insert(root->right, key);

    }

    return root;

}

void levelOrderTraversal(Node\* root){

    if(root == nullptr){

        return;

    }

    queue<Node\*> queue;

    queue.push(root);

    while(!queue.empty()){

        int n = queue.size();

        while(n--){

            //Get the front node

            Node\* front = queue.front();

            cout<< front->data << ' ';

            //Pop the queue

            queue.pop();

            //Insert front's left and right into queue.

            if(front->left != nullptr){

                queue.push(front->left);

            }

            if(front->right != nullptr){

                queue.push(front->right);

            }

        }

        cout << endl;

    }

}

//Traverse the tree in inorder fashion and push the node's keys in to the queue.

void inorderTraversal(Node\* root, queue<int>& keys){

    if(root == nullptr){

        return;

    }

    inorderTraversal(root->left, keys);

    keys.push(root->data);

    inorderTraversal(root->right, keys);

}

//Now again traverse the node in preOrder traversal and assign the node's data as queue's next val.

void preOrderTraversal(Node\* root, queue<int>& keys){

    if(root==nullptr){

        return;

    }

    root->data = keys.front();

    keys.pop();

    preOrderTraversal(root->left, keys);

    preOrderTraversal(root->right, keys);

}

void ConvertBinaryToHeap(Node\* root){

    if(root == nullptr){

        return;

    }

    queue<int> keys;

    //Traverse inorder and fill the keys queue in inorder way.

    inorderTraversal(root, keys);

    //Traverse preorder and fill the root's data with the queue's next value.

    preOrderTraversal(root,keys);

}

int main()

{

    Node\* root = nullptr;

    cout<< "Enter the number of nodes to be there in the Binary Search Tree." <<endl;

    int size;

    cin>>size;

    for(int i = 0; i<size; i++){

        int nodeVal;

        cout<<"Enter the "<<i <<"th Node element"<<endl;

        cin>>nodeVal;

        root = insert(root,nodeVal);

    }

    ConvertBinaryToHeap(root);

    levelOrderTraversal(root);

}

5. Write a program to implement Binary Tree using structure and pointer

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node\* left;

struct node\* right;

}\* root = NULL;

struct node\* insert(struct node\*,int);

struct node\* searchNode(struct node\*,int);

int main(){

int c;

while(1)

{

printf("Binary Search Tree:\n 1.Insert\n 2.Display Inorder\n 3. Search\n");

scanf("%d",&c);

switch(c)

{

case 1: int val;

printf("Please enter the data to be inserted:\n");

scanf("%d",&val);

root = insert(root,val);

break;

case 2: printf("Inorder traversal:\n");

inorderDisplay(root);

printf("\n");

break;

case 3: int val2;

printf("Enter the value to be searched:\n");

scanf("%d",&val2);

struct node\* found = searchNode(root,val2);

if(found == NULL){

printf("No node found\n");

}

else{

printf("Found %d\n",found->data);

}

break;

default: exit(1);

break;

}

}

}

struct node\* insert(struct node\* root, int data){

if(root == NULL){

//While traversing left or right, at one point the root becomes null, there we need to insert this.

struct node\* newNode = (struct node\*)malloc(sizeof(struct node\*));

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

else{

//Traverse left or right to the exact position where to insert and then inserts the node.

if(data < root->data){

root->left = insert(root->left,data); //Traversing to the left

}

else if( data > root->data){

root->right = insert(root->right,data); //Traversing to the right.

}

return root;

}

}

void inorderDisplay(struct node\* root){

if(root!=NULL){

inorderDisplay(root->left);

printf("%d ",root->data);

inorderDisplay(root->right);

}

}

struct node\* searchNode(struct node\* root, int data){

if(root == NULL){ //Not found case.

return NULL;

}

if(root->data == data){

return root;

}

if(root->data > data){

return searchNode(root->left,data);

}

else{

return searchNode(root->right,data);

}

}

6. Write a program to implement Binary Search Tree using structure and pointer

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node\* left;

struct node\* right;

}\* root = NULL;

struct node\* insert(struct node\*,int);

void inorderDisplay(struct node\*);

void preorderDisplay(struct node\*);

void postorderDisplay(struct node\*);

struct node\* searchNode(struct node\*,int);

int main(){

int c;

while(1){

printf("Binary Search Tree:\n 1.Insert\n 2.Display Inorder\n 3.Display PreOrder\n 4. Display PostOrder\n 5. Search\n 6.Exit\n");

scanf("%d",&c);

switch(c){

case 1: int val;

printf("Please enter the data to be inserted:\n");

scanf("%d",&val);

root = insert(root,val);

break;

case 2: printf("Inorder traversal:\n");

inorderDisplay(root);

printf("\n");

break;

case 3: printf("PreOrder traversal:\n");

preorderDisplay(root);

printf("\n");

break;

case 4: printf("PostOrder traversal:\n");

postorderDisplay(root);

printf("\n");

break;

case 5: int val2;

printf("Enter the value to be searched:\n");

scanf("%d",&val2);

struct node\* found = searchNode(root,val2);

if(found == NULL){

printf("No node found\n");

}

else{

printf("Found %d\n",found->data);

}

break;

default: exit(1);

break;

}

}

}

struct node\* insert(struct node\* root, int data){

if(root == NULL){

//While traversing left or right, at one point the root becomes null, there we need to insert this.

struct node\* newNode = (struct node\*)malloc(sizeof(struct node\*));

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

else{

//Traverse left or right to the exact position where to insert and then inserts the node.

if(data < root->data){

root->left = insert(root->left,data); //Traversing to the left

}

else if( data > root->data){

root->right = insert(root->right,data); //Traversing to the right.

}

return root;

}

}

void inorderDisplay(struct node\* root){

if(root!=NULL){

inorderDisplay(root->left);

printf("%d ",root->data);

inorderDisplay(root->right);

}

}

void preorderDisplay(struct node\* root){

if(root!=NULL){

printf("%d ",root->data);

preorderDisplay(root->left);

preorderDisplay(root->right);

}

}

void postorderDisplay(struct node\* root){

if(root!= NULL){

postorderDisplay(root->left);

postorderDisplay(root->right);

printf("%d ", root->data);

}

}

struct node\* searchNode(struct node\* root, int data){

if(root == NULL){ //Not found case.

return NULL;

}

if(root->data == data){

return root;

}

if(root->data > data){

return searchNode(root->left,data);

}

else{

return searchNode(root->right,data);

}

}

7. Verify if a Binary tree is symmetric around its center i.e: left subtree = right subtree at every node.

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

struct node{

int data;

struct node\* left;

struct node\* right;

}\* root = NULL;

bool isMirrorImage(struct node\* root1, struct node\* root2){

if(root1== NULL && root2 == NULL){

return true;

}

if(root1 && root2 && root1->data == root2->data && isMirrorImage(root1->left,root2->right) &&

isMirrorImage(root1->right,root2->left) ){

return true;

}

return false;

}

//Is Symmetric in the sense the tree should be divided in to two parts each part is mirror of other.

bool isSymmetric(struct node\* root){

return isMirrorImage(root,root);

}

struct node\* createNode(int data){

struct node\* newNode = (struct node\*)malloc(sizeof(struct node\*));

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

int main(){

root = createNode(5);

root->left = createNode(3);

root->right = createNode(3);

root->left->left = createNode(7);

root->left->right = createNode(9);

root->right->left = createNode(9);

root->right->right = createNode(7);

if(isSymmetric(root)){

printf("Binary Tree is symmetric around root");

}

else{

printf("Binary tree is not symmetric around root\n");

}

}

8. Create two functions named Encoding and decoding. In Encoding function pass the pointer to the root of binary tree and return the string representation of Binary tree. In Decoding function pass the string and return the root of Binary tree.

#include<iostream>

#include<vector>

#include<queue>

#include<string>

#include<utility>

#include<sstream>

using namespace std;

struct Node{

    int data;

    Node\* left;

    Node\* right;

    Node(){

    }

    Node(int data){

        this->data = data;

        this->left = nullptr;

        this->right = nullptr;

    }

};

//Inserting a node in binary search treee.

Node\* insert(Node\* root, int key){

    if(root == nullptr){

        return new Node(key);

    }

        if(key < root->data){

        root->left = insert(root->left,key);

    }

    else{

        root->right = insert(root->right, key);

    }

    return root;

}

string encode(Node\* root){

    if(!root){

        return "";

    }

    string s="";

    queue<Node\*> q;

    q.push(root);

    while(!q.empty()){

        Node\* current = q.front();

        q.pop();

        if(current==NULL){

            s.append("#,");

        }

        else{

            s.append(to\_string(current->data)+',');

        }

        if(current!= NULL){

            q.push(current->left);

            q.push(current->right);

        }

    }

    cout<< "Encoded String "<< s;

    return s;

}

Node\* decode(string input){

    if(input.size() == 0){

        return NULL;

    }

    stringstream s(input);

    string str;

    getline(s,str,',');

    Node\* root = new Node(stoi(str));

    queue<Node\*> q;

    q.push(root);

    while(!q.empty()){

        Node\* node = q.front();

        q.pop();

        getline(s,str,',');

        if(str == "#"){

            node->left = NULL;

        }

        else{

            Node\* leftNode = new Node(stoi(str));

            node->left = leftNode;

            q.push(leftNode);

        }

        getline(s,str,',');

        if(str == "#"){

            node->right = NULL;

        }

        else{

            Node\* rightNode = new Node(stoi(str));

            node->right = rightNode;

            q.push(rightNode);

        }

    }

    return root;

}

void inorder(Node\* root){

    if(root== NULL){

        return;

    }

    inorder(root->left);

    cout<< root->data << " ";

    inorder(root->right);

}

int main(){

    Node\* root = nullptr;

    cout<< "Enter the number of nodes to be there in the Binary Search Tree." <<endl;

    int size;

    cin>>size;

    for(int i = 0; i<size; i++){

        int nodeVal;

        cout<<"Enter the "<<i <<"th Node element"<<endl;

        cin>>nodeVal;

        root = insert(root,nodeVal);

    }

    string s = encode(root);

    Node\* node = decode(s);

    cout<< "Inorder traversal of Decoded Tree:";

    inorder(node);

}

9. Write a program to construct Binary tree from Inorder and preorder.

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node\* left;

struct node\* right;

};

int search(int arr[] , int start, int end, char value){

for(int i = start; i<=end; i++){

if( arr[i] == value){

return i;

}

}

}

struct node\* createNode(int data){

struct node\* node = (struct node\*)malloc(sizeof(struct node\*));

node->data = data;

node->left = NULL;

node->right = NULL;

return node;

}

struct node\* buildTree(int inor[], int preor[], int instart, int inend){

static int preindex = 0;

if(instart > inend){

return NULL;

}

struct node\* newNode = createNode(preor[preindex++]);

//if this node has no children, then this will be the case

if(instart == inend){

return newNode;

}

//Find the index of the same element in the inorder

int inorderIndex = search(inor,instart,inend,newNode->data);

//Need to build the left tree from the inindex left side and right tree from inindex right side

newNode->left = buildTree(inor,preor,instart,inorderIndex-1);

newNode->right = buildTree(inor,preor,inorderIndex+1, inend);

return newNode;

}

void inorder(struct node\* root){

if(root== NULL){

return;

}

inorder(root->left);

printf("%d ",root->data);

inorder(root->right);

}

int main(){

int inor[] = { 4,2,5,1,6,3};

int preor[] = { 1,2,4,5,3,6};

int len = sizeof(inor) / sizeof(inor[0]);

struct node\* root = buildTree(inor,preor, 0, len-1);

printf("Inorder traversal:");

inorder(root);

}

10. Write a program to check if all the leafs of a Binary tree are at the same level or not ?

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

struct node{

int data;

struct node\* left;

struct node\* right;

};

struct node\* createNode(int data){

struct node\* node = (struct node\*)malloc(sizeof(struct node\*));

node->data = data;

node->left = NULL;

node->right = NULL;

return node;

}

//Inserting a node in binary search treee.

struct node\* insert(struct node\* root, int key){

if(root == NULL){

return createNode(key);

}

if(key < root->data){

root->left = insert(root->left,key);

}

else{

root->right = insert(root->right, key);

}

return root;

}

void inorder(struct node\* root){

if(root== NULL){

return;

}

inorder(root->left);

printf("%d ",root->data);

inorder(root->right);

}

bool CheckRecursive(struct node\* root, int currentLevel, int\* firstLeafLevel){

if(root == NULL){

return true;

}

//if leaf is found.

if(root->left == NULL && root->right ==NULL){

//If encountering the leaf for the first time then assign the current level to

//firstleaflevel

if(\*firstLeafLevel == 0){

\*firstLeafLevel = currentLevel;

return true;

}

return currentLevel == \*firstLeafLevel;

}

return CheckRecursive(root->left, currentLevel+1, firstLeafLevel) && CheckRecursive(root->right,

currentLevel+1, firstLeafLevel);

}

bool checkIfAllLeavesAtSameLevel(struct node\* root){

int currentlevel = 0;

int firstLeafLevel = 0;

return CheckRecursive(root,currentlevel,&firstLeafLevel);

}

int main(){

struct node\* root = NULL;

printf("Enter the number of nodes to be there in the Binary Search Tree.\n");

int size;

scanf("%d",&size);

for(int i = 0; i<size; i++){

int nodeVal;

printf("Enter the %dth node data:",i);

scanf("%d",&nodeVal);

root = insert(root,nodeVal);

}

if(checkIfAllLeavesAtSameLevel(root)){

printf("All leaves are at same level\n");

}

else{

printf("Leaves are not at same level");

}

}

Section 2

1. using namespace std;

#include<vector>

#include<list>

#include<iostream>

class Graph{

    int vertices;

    vector<list<char>> adjacents;

    public :

    Graph(int v);  // Constructor

    // function to add an edge to graph

    void addEdge(char v, char w);

    // prints BFS traversal from a given source s

    void BFS(char startingindex);

    void DFS(char startingindex, vector<bool> visited);

};

Graph::Graph(int v){

        this->vertices = v;

        adjacents.resize(v);

    }

void Graph::addEdge(char v, char w){

        adjacents[v-'A'].push\_back(w);

    }

void Graph::BFS(char startingindex){

        vector<bool> visited;

        visited.resize(this->vertices,false);

        //Take a queue and push the starting index. and put visited is true for that index

        list<char> queue;

        visited[startingindex-'A'] = true;

        queue.push\_back(startingindex);

        while(!queue.empty()){

            char value = queue.front();

            cout << value << " ";

            queue.pop\_front();

            for(auto adjacent: adjacents[value-'A']){

                if(!visited[adjacent-'A']){

                    visited[adjacent-'A'] = true;

                    queue.push\_back(adjacent);

                }

            }

        }

    }

void Graph:: DFS(char startingindex, vector<bool> visited){

    visited[startingindex-'A'] = true;

    cout << startingindex << " ";

    for(auto adjacent: adjacents[startingindex-'A']){

        if(!visited[adjacent-'A']){

            DFS(adjacent,visited);

        }

    }

}

int main(){

    Graph g(16);

    g.addEdge('D','A');

    g.addEdge('K','E');

    g.addEdge('E','A');

    g.addEdge('C','A');

    g.addEdge('A','B');

    g.addEdge('A','F');

    g.addEdge('A','G');

    g.addEdge('G','A');

    g.addEdge('G','F');

    g.addEdge('F','I');

    g.addEdge('I','G');

    g.addEdge('H','G');

    g.addEdge('I','H');

    g.addEdge('H','I');

    g.addEdge('H','I');

    g.addEdge('I','J');

    cout << "BFS: ";

    g.BFS('A');cout<<endl;

    vector<bool> visited;

    visited.resize(16,false);

    cout<< "DFS: ";

    g.DFS('A',visited);

    cout<<endl;

}