**Binary Tree**

#include<iostream>

#include<bits/stdc++.h>

using namespace std;

class Node{

public:

int val;

Node\* left;

Node\* right;

Node(int x){

this->val = x;

left = NULL;

right = NULL;

}

};

void insertNode(Node\* root,int val){

queue<Node\*>q;

q.push(root);

while(!q.empty()){

Node\* curr =q.front();

q.pop();

if(curr->left == NULL){

curr->left = new Node(val);

return;

}

else{

q.push(curr->left);

}

if(curr->right == NULL){

curr->right = new Node(val);

return;

}

else{

q.push(curr->right);

}

}

}

void preorder(Node\*root){

if(root){

cout<<root->val<<" ";

preorder(root->left);

preorder(root->right);

}

}

int internalNodes(Node\* root){

if(root==NULL)return 0;

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

return 0;

}

return internalNodes(root->left)+internalNodes(root->right)+1;

}

int leafNodes(Node\* root){

if(root == NULL) return 0;

if(root->left == NULL && root->right == NULL)return 1;

return leafNodes(root->left)+leafNodes(root->right);

}

void inorder(Node\*root){

stack<Node\*>s;

Node\* curr = root;

while(curr!=NULL || !s.empty()){

while(curr != NULL){

s.push(curr);

curr = curr->left;

}

curr = s.top();

s.pop();

cout<<curr->val<<" ";

curr = curr->right;

}

}

void levelOrder(Node\* root){

queue<Node\*>q;

q.push(root);

while(!q.empty()){

int levelsize = q.size();

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

Node\* curr = q.front();

q.pop();

cout<<curr->val<<" ";

if(curr->left)q.push(curr->left);

if(curr->right)q.push(curr->right);

}cout<<endl;}

}

int height(Node\* root){

if(root == NULL)return 0;

int left = height(root->left);

int right = height(root->right);

return max(left,right)+1;

}

// 7 5 8 3 2 4 7 9

int main(){

int n;

cout<<"Enter no of nodes do you want to insert : ";

cin>>n;

int val;

cout<<"Enter all nodes : ";

cin>>val;

Node\* root = new Node(val);

for(int i= 0 ;i<n-1;i++){

cin>>val;

insertNode(root,val);

}

cout<<endl<<"The inorder traversal is "<<endl;

inorder(root);

cout<<endl<<"The preorder traversal is :"<<endl;

preorder(root);

cout<<endl;

cout<<endl<<"The no. of internal nodes are "<<internalNodes(root)<<endl;

cout<<endl<<"The no. of leaf nodes are "<<leafNodes(root)<<endl;

cout<<endl<<"The height of the tree is "<<height(root)<<endl;

cout<<endl<<"Level wise printing of tree "<<endl;

levelOrder(root);

return 0;}

**Binary Search Tree**

#include<iostream>

#include<stack>

using namespace std;

class Node{

public:

int value;

Node\* left;

Node\* right;

Node(){

this->left = NULL;

this->right = NULL;

}

};

Node\* createRoot(Node\* root){

cout<<"Enter data for root : ";

cin>>root->value;

return root;

}

void insertNode(Node\*root,Node\* node){

if(root->value>node->value){

if(root->left == NULL){

root->left = node;

return;

}

else{

insertNode(root->left,node);

}

}

else{

if(root->right == NULL){

root->right = node;

return;

}

else{

insertNode(root->right,node);

}

}

}

void DescendingOrder(Node\*root){

stack<Node\*>s;

Node\* curr = root;

while(!s.empty() || curr !=NULL){

while(curr!=NULL){

s.push(curr);

curr = curr->right;

}

curr = s.top();

s.pop();

cout<<curr->value<<" ";

curr = curr->left;

}

}

void AscendingOrder(Node\*root){

stack<Node\*>s;

Node\* curr = root;

while(!s.empty() || curr !=NULL){

while(curr!=NULL){

s.push(curr);

curr = curr->left;

}

curr = s.top();

s.pop();

cout<<curr->value<<" ";

curr = curr->right;

}

}

int maxElement(Node\* root){

while(root->right != NULL){

root = root->right;

}

return root->value;

}

int minElement(Node\* root){

while(root->left != NULL){

root = root->left;

}

return root->value;

}

int main(){

int n;

cout<<"Enter No. of nodes do you want to insert : ";

cin>>n;

Node\* root = new Node;

root = createRoot(root);

cout<<"Enter remaining "<<(n-1)<<"nodes"<<endl;

int value;

for(int i=0;i<n-1;i++){

Node\* node = new Node;

cin>>node->value;

insertNode(root,node);

}

cout<<endl<<endl<<"Ascending Order : "<<endl;

AscendingOrder(root);

cout<<endl<<endl<<"Descending Order : "<<endl;

DescendingOrder(root);

cout<<endl<<endl;

cout<<"Finding the max and min element from the tree : "<<endl;

cout<<endl;

cout<<"Maximum element is "<<maxElement(root);

cout<<endl<<endl;

cout<<"Minimum element is "<<minElement(root);

return 0;

}

// 9 15 13 18 8 14 16 19 4 9**Dictionary**

#include<iostream>

#include<stack>

#include<string>

using namespace std;

class Node{

public:

string value;

Node\* left;

Node\* right;

Node(){

this->left = NULL;

this->right = NULL;

}

};

Node\* createRoot(Node\* root){

cout<<"Enter data for root : ";

cin>>root->value;

return root;

}

void insertNode(Node\*root,Node\* node){

if(root->value>node->value){

if(root->left == NULL){

root->left = node;

return;

}

else{

insertNode(root->left,node);

}

}

else{

if(root->right == NULL){

root->right = node;

return;

}

else{

insertNode(root->right,node);

}

}

}

void DescendingOrder(Node\*root){

stack<Node\*>s;

Node\* curr = root;

while(!s.empty() || curr !=NULL){

while(curr!=NULL){

s.push(curr);

curr = curr->right;

}

curr = s.top();

s.pop();

cout<<curr->value<<" ";

curr = curr->left;

}

}

void AscendingOrder(Node\*root){

stack<Node\*>s;

Node\* curr = root;

while(!s.empty() || curr !=NULL){

while(curr!=NULL){

s.push(curr);

curr = curr->left;

}

curr = s.top();

s.pop();

cout<<curr->value<<" ";

curr = curr->right;

}

}

string maxElement(Node\* root){

while(root->right != NULL){

root = root->right;

}

return root->value;

}

string minElement(Node\* root){

while(root->left != NULL){

root = root->left;

}

return root->value;

}

Node\* minimum(Node\*root){

while(root->left){

root = root->left;

}

return root;

}

Node\* remove(Node\*root, string value){

if(root == NULL) {return NULL;

cout<<"node not found";

}

if(value<root->value){

root->left = remove (root->left,value);

}

else if(value>root->value){

root->right = remove(root->right,value);

}

else{

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

delete root;

return NULL;

}

else if(root->left == NULL){

Node\* temp = root->right;

delete root;

return temp;

}

else if(root->right == NULL){

Node\* temp = root->left;

delete root;

return temp;

}

else{

Node\* temp = minimum(root->right);

root->value = temp->value;

root->right = remove(root->right,value);

}

}

return root;

}

int main(){

int n;

cout<<"Enter No. of nodes do you want to insert : ";

cin>>n;

Node\* root = new Node;

root = createRoot(root);

cout<<"Enter remaining "<<(n-1)<<"nodes"<<endl;

string value;

for(int i=0;i<n-1;i++){

Node\* node = new Node;

cin>>node->value;

insertNode(root,node);

}

cout<<endl<<endl<<"Ascending Order : "<<endl;

AscendingOrder(root);

cout<<endl<<endl<<"Descending Order : "<<endl;

DescendingOrder(root);

cout<<endl<<endl;

cout<<"Finding the max and min element from the tree : "<<endl;

cout<<endl;

cout<<"Maximum element is "<<maxElement(root);

cout<<endl<<endl;

cout<<"Minimum element is "<<minElement(root);

while(true){cout<<"Enter element to delete " ;

cin>>value;

root = remove(root,value);

cout<<endl<<endl<<"Ascending Order : "<<endl;

AscendingOrder(root);

}

return 0;

}

// 9 15 13 18 8 14 16 19 4 9

// carrot dog banana giraffe frog elephant apple

**Threaded Binary Tree**

#include<iostream>

using namespace std;

class Node{

public:

int value;

Node\* left,\*right;

bool lthread,rthread;

};

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

Node\* ptr = root;

Node\* par = NULL;

while(ptr!=NULL){

if(key == ptr->value){

cout<<"duplicate value"<<endl;

return root ;

}

par = ptr;

if(key < ptr->value){

if(ptr->lthread == false){

ptr = ptr->left;

}

else break;

}

else{

if(ptr->rthread == false){

ptr = ptr->right;

}

else break;

}

}

// Creating new node

Node\* temp = new Node;

temp->value = key;

temp->lthread = true;

temp->rthread = true;

if(par == NULL){

root = temp;

temp->left = NULL;

temp->right = NULL;

}

else if(key < par->value){

temp->left = par->left;

temp->right = par;

par->lthread = false;

par->left = temp;

}

else{

temp->left= par;

temp->right = par->right;

par->rthread = false;

par->right = temp;

}

return root;

}

Node\* inOrderSuccessor(Node\* root){

if(root->rthread == true){

return root->right;

}

root = root->right;

while(root->lthread == false){

root = root->left;

}

return root;

}

void inOrder(Node\* ptr){

if(ptr == NULL) cout<<"Tree is empty ";

while(ptr->lthread == false){

ptr = ptr -> left;

}

while(ptr!=NULL){

cout<<ptr->value<<" ";

ptr = inOrderSuccessor(ptr);

}

}

void preOrder(Node\* ptr) {

if (ptr == NULL) return;

cout << ptr->value << " ";

if (ptr->lthread == false) preOrder(ptr->left);

if (ptr->rthread == false) preOrder(ptr->right);

}

void postOrder(Node\* ptr) {

if (ptr == NULL) return;

if (ptr->lthread == false) postOrder(ptr->left);

if (ptr->rthread == false) postOrder(ptr->right);

cout << ptr->value << " ";

}

int main(){

Node\* root = NULL;

int n,value;

cout<<"Enter no. of nodes do you want to insert : ";

cin>>n;

cout<<"Enter all nodes to insert : "<<endl;

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

cin>>value;

root = insert(root,value);}

cout<<endl<<"InOrder Sequence"<<endl;

inOrder(root);

cout<<endl<<endl<<"PreOrder Sequence: "<<endl;

preOrder(root);

cout<<endl<<endl<<"PostOrder Sequence: "<<endl;

postOrder(root);

return 0;

}

// 9 15 13 18 8 14 16 19 4 9

**Implementation of heap data structure**

#include<iostream>

using namespace std;

class heap{

public:

int arr[100];

int size;

heap(){

arr[0]=-1;

size = 0;

}

void insert(int value){

size = size + 1;

int index = size;

arr[index] = value;

while(index>1){

int parent = index/2;

if(arr[parent]<arr[index]){

swap(arr[parent],arr[index]);

index = parent;

}

else{

return;

}

}

}

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

int largest = i;

int left = 2\*i;

int right = 2\*i + 1;

if(left<=n && arr[left]>arr[largest]){

largest = left;

}

if(right<= n && arr[right]>arr[largest]){

largest = right;

}

if(largest != i){

swap(arr[largest],arr[i]);

heapify(arr,n,largest);

}

}

void heapSort(int arr[],int n){

while(n>1){

swap(arr[1],arr[n]);

n--;

heapify(arr,n,1);

}

}

void print(){

for(int i=1;i<=size;i++){

cout<<arr[i]<<" ";

}

}

};

int main(){

heap h;

cout<<"Enter no. of elements do you want to insert :";

int n;

cin>>n;

cout<<"Enter all elements : "<<endl;

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

int value;

cin>>value;

h.insert(value);

}

h.print();

cout<<endl;

cout<<"the sorted array is :"<<endl;

h.heapSort(h.arr,n);

h.print();

return 0;

}

**BFS Traversal**

#include<iostream>

#include<queue>

using namespace std;

class objArray{

public:

int vertex;

int cost;

};

class Graph{

public:

int arr[10][10]={};

int visited[10]={};

void create(int e){

cout<<"Enter vertices for which edge is visited : "<<endl;

for(int i=0;i<e;i++){cout<<"Enter vertices : ";

int v1,v2;

cin>>v1>>v2;

int cost;

cout<<"Enter the cost between "<<v1<<" and "<<v2<<" : ";

cin>>cost;

arr[v1][v2] = cost;

arr[v2][v1] = cost;}

}

void print(int v){

cout<<"The adjacency matrix is "<<endl;

for(int i=1;i<=v;i++){

for(int j=1;j<=v;j++){

cout<<arr[i][j]<<" ";

}

cout<<endl;

}

}

void bfs\_traversal(int v){

int vertex;

cout<<"Enter vertex to start bfs : "<<endl;

cin>>vertex;

queue<int> q;

q.push(vertex);

visited[vertex] = 1;

while(!q.empty()){

int curr = q.front();

cout<<curr<<" ";

q.pop();

for(int i = 1; i <= v; i++){

if(arr[curr][i] != 0 && !Visited(visited, i)){

q.push(i);

visited[i] = 1;

}

}

}

}

bool Visited(int visited[],int n){

if(visited[n]!=0) return true;

else return false;

}

};

int main(){

cout<<"Enter no. of vertices ";

int v;

cin>>v;

cout<<"Enter no. of edges ";

int e;

cin>>e;

Graph g;

g.create(e);

g.print(v);

g.bfs\_traversal(v);

cout<<endl<<endl;

return 0;

}

/\*

vertices = 6

edges = 7

1 5 1 1 4 2 1 3 1 1 2 8 3 5 2 3 6 1 3 4 1

\*/

**ROUGH GRAPH PROGRAM**

#include <iostream>

#include <string>

#include <vector>

#include <queue>

#include <unordered\_set>

#include <limits>

using namespace std;

class DLLNode {

public:

string vertex;

DLLNode \*next;

DLLNode \*down;

int dist;

int degree;

DLLNode() {

this->next = nullptr;

this->down = nullptr;

this->degree = 0;

this->dist = 0;

}

};

class AdjList {

public:

void create(DLLNode \*&head);

void display(DLLNode \*head);

void checkDeg\_conn(DLLNode \*head);

void findMST(DLLNode \*head); // Function to find Minimum Spanning Tree

private:

struct Edge {

string src;

string dest;

int weight;

};

struct CompareEdge {

bool operator()(const Edge &a, const Edge &b) {

return a.weight > b.weight;

}

};

};

void AdjList::create(DLLNode \*&head) {

int v;

cout << "Enter the no. of vertices : ";

cin >> v;

head = new DLLNode(); // Initialize head

DLLNode \*tempHead = head;

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

tempHead->down = new DLLNode();

DLLNode \*temp = tempHead->down;

tempHead = tempHead->down;

cout << "Enter vertex " << i + 1 << " : ";

cin >> temp->vertex;

cout << "Enter no. of Adjacent : ";

int a;

cin >> a;

for (int j = 0; j < a; j++) {

DLLNode \*curr = new DLLNode();

temp->next = curr;

temp = curr;

cout << "Enter data : ";

cin >> curr->vertex;

cout<<"Enter dist : ";

cin>>curr->dist;

}

}

}

void AdjList::display(DLLNode \*head) {

DLLNode \*curr = head->down;

while (curr != NULL) {

cout << curr->vertex;

DLLNode \*temp = curr->next;

while (temp != NULL) {

cout << "----->" << temp->vertex;

curr->degree++;

temp = temp->next;

}

cout << endl << "\n";

curr = curr->down;

}

}

void AdjList::checkDeg\_conn(DLLNode \*head) {

DLLNode \*curr = head->down;

bool flag = true;

cout << "----- Displaying Degree -----\n" << endl;

while (curr != NULL) {

cout << curr->vertex << " - " << curr->degree << endl;

if (curr->degree == 0) flag = false;

curr = curr->down;

}

if (flag) cout << "\n" << endl << "The graph is connected " << endl;

else cout << endl << "The graph is not connected " << endl;

}

void AdjList::findMST(DLLNode \*head) {

unordered\_set<string> visited;

priority\_queue<Edge, vector<Edge>, CompareEdge> minHeap;

DLLNode \*curr = head->down;

// Start with the first vertex

visited.insert(curr->vertex);

// Add all edges from the first vertex to the minHeap

DLLNode \*temp = curr->next;

while (temp != nullptr) {

minHeap.push({curr->vertex, temp->vertex, 1}); // Assuming all edges have weight 1

temp = temp->next;

}

// MST will have V-1 edges

for (int i = 1; i < visited.size(); ++i) {

Edge minEdge;

do {

if (minHeap.empty()) {

cerr << "Graph is not connected" << endl;

return;

}

minEdge = minHeap.top();

minHeap.pop();

} while (visited.find(minEdge.dest) != visited.end());

cout << "Edge: " << minEdge.src << " - " << minEdge.dest << ", Weight: " << minEdge.weight << endl;

visited.insert(minEdge.dest);

// Add new edges from the newly visited vertex

DLLNode \*temp = head->down;

while (temp != nullptr && temp->vertex != minEdge.dest) {

temp = temp->down;

}

if (temp != nullptr) {

DLLNode \*adjacent = temp->next;

while (adjacent != nullptr) {

if (visited.find(adjacent->vertex) == visited.end()) {

minHeap.push({temp->vertex, adjacent->vertex, 1}); // Assuming all edges have weight 1

}

adjacent = adjacent->next;

}

}

}

}

int main() {

AdjList l;

DLLNode \*Head = nullptr;

l.create(Head);

cout << "\n\n";

l.display(Head);

cout << "\n\n";

l.checkDeg\_conn(Head);

cout << "\nMinimum Spanning Tree:" << endl;

l.findMST(Head);

return 0;

}

/\*5 nashik 2 pune mumbai pune 2 nashik mumbai mumbai 4 pune nashik nagpur delhi nagpur 1 mumbai delhi 1 mumbai

5 zero 3 one 3 three 7 four 8 one 3 zero 3 three 4 two 1 three 3 zero 7 two 2 four 3 two 2 one 1 three 2 four 2 zero 8 three 3

\*/

**OBST**

#include <bits/stdc++.h>

using namespace std;

int dp[100][100];

int optCost(int freq[], int i, int j) {

    // Base case

    if (j < i) {

        return 0;

    }

    if (i == j) {

        return freq[i];

    }

    if (dp[i][j] != -1) {

        return dp[i][j];

    }

    int fsum = 0;

    for (int k = i; k <= j; k++) {

        fsum += freq[k];

    }

    int minCost = INT\_MAX;

    for (int r = i; r <= j; r++) {

        int cost = optCost(freq, i, r - 1) + optCost(freq, r + 1, j) + fsum;

        minCost = min(cost, minCost);

    }

    dp[i][j] = minCost;

    return dp[i][j];

}

int main() {

    int n;

    cout << "Enter ";

    cin >> n;

    int keys[n];

    int freq[n];

    for(int i  = 0;i<n;i++)

    {

        cin>>keys[i];

        cout<<"enter frequency :";

        cin>>freq[i];

    }

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            dp[i][j] = -1;

        }

    }

    cout << optCost(freq, 0, n - 1) << endl;

    return 0;

}