Assignment

BADHON DATTA PROTTOY

ROLL: B230101EC

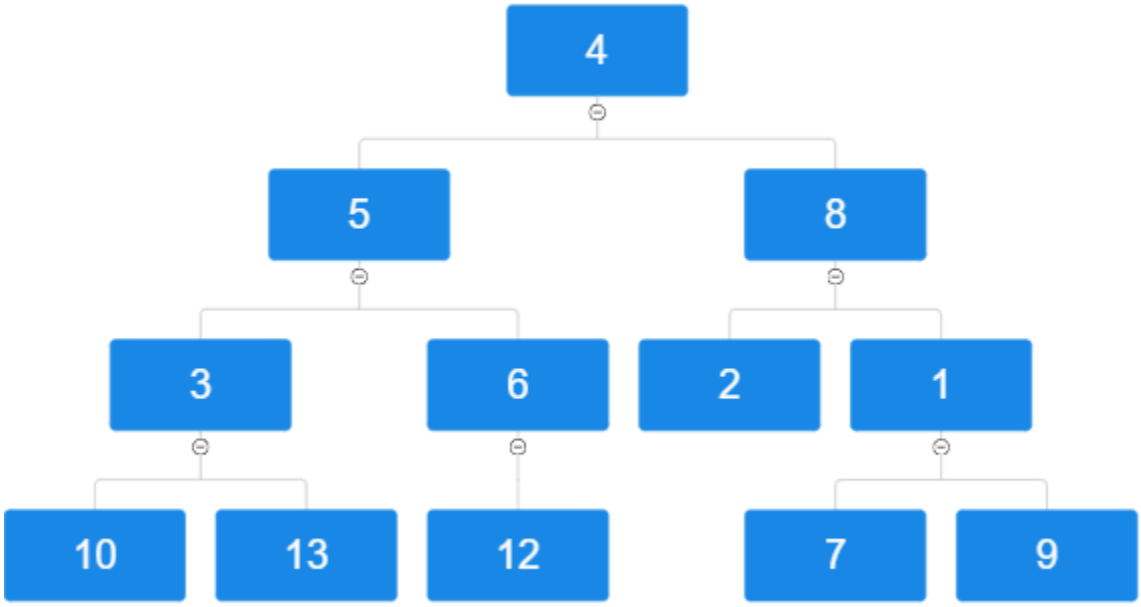
EC01

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

EC2022E Data Structures

Part A:

1. Take a binary tree and do tree traversals



(Note : This Tree is used for both the Questions)

Code:

```
#include<bits/stdc++.h>
using namespace std;
struct Node{
int data;
struct Node *left, *right;
Node(int data){
this->data = data;
left=right=NULL;
}
};
void pre_order(struct Node *node){
if(node==NULL) return;
else{
    cout<< node->data <<" ";
pre_order(node->left);
pre_order(node->right);
```

```
}
}
void in_order(struct Node *node){
if(node==NULL) return;
else{
in_order(node->left);
cout<< node->data <<" ";
in_order(node->right);
}
}
void post_order(struct Node *node){
if(node==NULL) return;
else{
post_order(node->left);
post_order(node->right);
cout<< node->data <<" ";
}
}
void level_order(struct Node *node){
if(node==NULL) return;
else{
queue<Node*> Q;
Q.push(node);
while(!Q.empty()){
Node *temp = Q.front();
Q.pop();
cout<<temp->data<<" ";
if(temp->left != NULL){
Q.push(temp->left);
}
if(temp->right != NULL){
Q.push(temp->right);
}
}
}
}
int main(){
struct Node *root = new Node(4);
root->left = new Node(5);
root->right = new Node(8);
```

```
root->left->left = new Node(3);
root->left->right = new Node(6);
root->right->left = new Node(2);
root->right->right = new Node(1);
root->left->left->left = new Node(10);
root->left->left->right = new Node(13);
root->left->right->left = new Node(12);
root->right->right->left = new Node(7);
root->right->right->right = new Node(9);
cout<<"\nPre-Order traversal : ";
pre_order(root);
cout<<"\n\nIn-Order traversal : ";
in_order(root);
cout<<"\n\nPost-Order traversal : ";
post_order(root);
cout<<"\n\nLevel-Order traversal : ";
level_order(root);
}
```

Output:

```
Pre-Order traversal : 4 5 3 10 13 6 12 8 2 1 7 9

In-Order traversal : 10 3 13 5 12 6 4 2 8 7 1 9

Post-Order traversal : 10 13 3 12 6 5 2 7 9 1 8 4

Level-Order traversal : 4 5 8 3 6 2 1 10 13 12 7 9
```

2. Construct a BST and do the following on it:
a) Insert
b) Delete
c) Search
d) Max
e) Min
f) Predecessor
g) Successor

Code:

```
#include<bits/stdc++.h>
```

```
using namespace std;
class BST{
public :
int data;
BST *left, *right;
public :
BST(){}
BST(int value){
data = value;
left = right = NULL;
}
void level_order(BST *node){
if(node==NULL) return;
else{
queue<BST *> Q;
Q.push(node);
while(!Q.empty()){
BST *temp = Q.front();
Q.pop();
cout<<temp->data<<" ";
if(temp->left != NULL){
Q.push(temp->left);
}
if(temp->right != NULL){
Q.push(temp->right);
}
}
}
}
BST* insert(BST *root,int value){
if(root==NULL){
root = new BST(value);
cout<<"Your Data is Successfully inserted "<< endl;
return root;
}
else{
if(value < root->data){
root->left = insert(root->left,value);
}
else if(value > root->data){
```

```
root->right = insert(root->right,value);
}
}
return root;
}
BST* max(BST *root){
BST *temp;
temp = root;
while(temp->right!=NULL){
temp = temp->right;
}
return temp;
}
BST* min(BST *root){
BST *temp ;
temp = root;
while(temp->left!=NULL){
temp = temp->left;
}
return temp;
}
void search(BST *root , int value){
int depth = 0;
BST *temp ;
temp = root;
while(temp!=NULL){
depth++;
if(temp->data == value){
cout<<"\nThe Value "<<value<<" is found at depth " <<depth<<endl;
return;
}
else if(temp->data > value){
temp = temp->left;
}
else if(temp->data < value){
temp = temp->right;
}
}
}
BST* deleteNode(BST *root,int value){
```

```
if(root==NULL) return NULL;
if(root->data > value){
root->left = deleteNode(root->left,value);
}
else if(root->data < value){
root->right = deleteNode(root->right,value);
}
else{
if(root->left==NULL && root->right == NULL){
    return NULL;
}
else if(root->left==NULL){
return root->right;
}
else if(root->right==NULL){
return root->left;
}
BST *temp = min(root->right);
root->data = temp->data;
root->right = deleteNode(root->right,temp->data);
}
return root;
}
int successor(BST *root ,int value){
if (root == NULL) return -1;
BST *temp ;
stack<BST *> Stack;
temp = root;
// key search
while(temp!=NULL){
if(temp->data == value){
break;
}
Stack.push(temp);
if(temp->data > value){
temp = temp->left;
}
else if(temp->data < value){
temp = temp->right;
}
```

```
}
// successor search
if(temp->right!=NULL)
{ BST *t = min(temp->right);
return t->data;
}
while(Stack.empty()== false && temp==Stack.top()->right){
temp = Stack.top();
Stack.pop();
}
return Stack.top()->data;
}
int predecessor(BST *root ,int value){
    if (root == NULL) return -1;
BST *temp ;
stack<BST *> Stack;
temp = root;
// key search
while(temp!=NULL){
if(temp->data == value){
break;
}
Stack.push(temp);
if(temp->data > value){
temp = temp->left;
}
else if(temp->data < value){
temp = temp->right;
}
}
// predecessor search
if(temp->left!=NULL)
{ BST *t = max(temp->left);
return t->data;
}
while(Stack.empty()== false && temp==Stack.top()->left){
temp = Stack.top();
Stack.pop();
}
return Stack.top()->data;
```

```
}
};
int main(){
BST *root = NULL;
BST b;
int n,value;
bool flag = true;
cout<<"Select the correct choice : "<<endl;
cout<<"0 - Quit "<<endl;
cout<<"1 - Insert "<<endl;
cout<<"2 - Level Order display "<<endl;
cout<<"3 - Search "<<endl;
cout<<"4 - Maximum"<<endl;
cout<<"5 - Minimum "<<endl;
cout<<"6 - Delete "<<endl;
cout<<"7 - Successor"<<endl;
cout<<"8 - Predecessor "<<endl;
cout<<"------------------------------"<<endl;
while(1){
cout<<"\nInsert choice : ";
cin>>n;
switch(n){
case 0:
{
    exit(0);
}
case 1: {
cout<<"Enter the Value to be Inserted : ";
cin>>value;
if(flag) {
root = b.insert(root,value);
flag = false;
}
else b.insert(root,value);
break;
}
case 2: {
cout<<"Level Order Display : ";
b.level_order(root);
cout<<endl;
```

```
break;
}
case 3: {
cout<<"Enter the Value to be Searched : ";
cin>>value;
b.search(root,value);
break;
}
case 4: {
cout<<"Maximum Value : ";
cout<<b.max(root)->data<<endl;
break;
}
case 5: {
cout<<"Minimum Value : ";
cout<<b.min(root)->data<<endl;
break;
}
case 6: {
cout<<"Enter the value to be Deleted : ";
cin>>value;
b.deleteNode(root,value);
cout<<"Value has been deleted successfully"<<endl;
break;
}
case 7: {
cout<<"Enter the value Whose Successor you want to find : ";
cin>>value;
cout<<"Successor of "<<value<<" is :"<<b.successor(root,value)<<endl;
break;
}
case 8: {
cout<<"Enter the value Whose Predecessor you want to find : ";
cin>>value;
cout<<"Predecessor of "<<value<<" is :"<<b.predecessor(root,value)<<endl;
break;
}
default:
{
cout<<"Entered invalid choice."<<endl;
```

```
break;
}
}
}
}
```

Output:

```
Select the correct choice :
0 - Quit
1 - Insert
2 - Level Order display
3 - Search
4 - Maximum
5 - Minimum
6 - Delete
7 - Successor
8 - Predecessor
------------------------------

Insert choice : 1
Enter the Value to be Inserted : 4
Your Data is Successfully inserted

Insert choice : 1
Enter the Value to be Inserted : 5
Your Data is Successfully inserted

Insert choice : 1
Enter the Value to be Inserted : 8
Your Data is Successfully inserted

Insert choice : 1
Enter the Value to be Inserted : 3
Your Data is Successfully inserted
```

```
Insert choice : 1
Enter the Value to be Inserted : 6
Your Data is Successfully inserted

Insert choice : 1
Enter the Value to be Inserted : 2
Your Data is Successfully inserted

Insert choice : 1
Enter the Value to be Inserted : 1
Your Data is Successfully inserted

Insert choice : 1
Enter the Value to be Inserted : 10
Your Data is Successfully inserted

Insert choice : 1
Enter the Value to be Inserted : 13
Your Data is Successfully inserted

Insert choice : 1
Enter the Value to be Inserted : 12
Your Data is Successfully inserted

Insert choice : 1
Enter the Value to be Inserted : 7
Your Data is Successfully inserted

Insert choice : 1
Enter the Value to be Inserted : 9
Your Data is Successfully inserted

Insert choice : 2
Level Order Display : 4 3 5 2 8 1 6 10 7 9 13 12

Insert choice : 3
Enter the Value to be Searched : 7

The Value 7 is found at depth 5
```

```
Insert choice : 4
Maximum Value : 13

Insert choice : 5
Minimum Value : 1

Insert choice : 7
Enter the value Whose Successor you want to find : 7
Successor of 7 is :8

Insert choice : 8
Enter the value Whose Predecessor you want to find : 5
Predecessor of 5 is :4

Insert choice : 6
Enter the value to be Deleted : 9
Value has been deleted successfully

Insert choice : 2
Level Order Display : 4 3 5 2 8 1 6 10 7 13 12

Insert choice :
```

Part B:

3. Implement the following graph algorithms

i) BFS

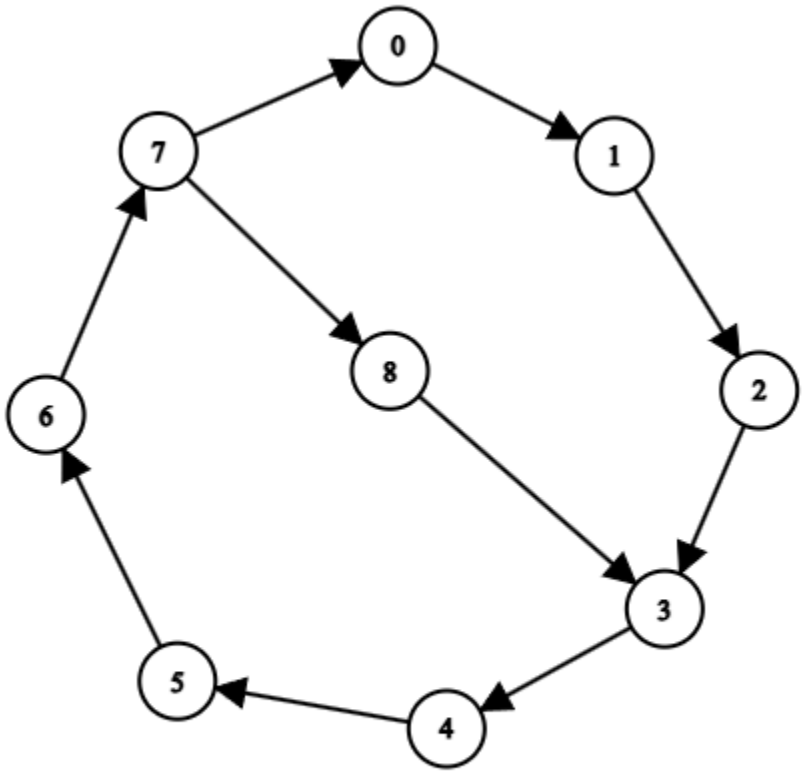
ii) DFS

iii) PRIM'S ALGORITHM

iv) KRUSKAL'S ALGORITHM

v) DJIKSTRA'S ALGORITHM

GRAPH FOR BFS AND DFS



i) BFS:

CODE:

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

```
class Graph {
    int V;
    list<int> *neighbour;

public:
    Graph(int V) {
        this->V = V;
        neighbour = new list<int>[V];
    }

    void addEdge(int v, int w) {
        neighbour[v].push_back(w);
    }

    void BFS(int s) {
        vector<bool> visited(V, false);
        list<int> queue;

        visited[s] = true;
        queue.push_back(s);

        list<int>::iterator i;
        while (!queue.empty()) {
            s = queue.front();
            cout << s << " ";
            queue.pop_front();

            for (i = neighbour[s].begin(); i != neighbour[s].end(); ++i) {
                if (!visited[*i]) {
                    visited[*i] = true;
                    queue.push_back(*i);
                }
            }
        }
    }

    ~Graph() {
        delete[] neighbour;
    }
};
```

```
int main() {
    Graph graph(9);
    graph.addEdge(0, 1);
    graph.addEdge(1, 2);
    graph.addEdge(2, 3);
    graph.addEdge(3, 4);
    graph.addEdge(4, 5);
    graph.addEdge(5, 6);
    graph.addEdge(6, 7);
    graph.addEdge(7, 0);
    graph.addEdge(7, 8);
    graph.addEdge(8, 3);

    cout << "Breadth First Traversal: ";
    graph.BFS(1);

    return 0;
}
```

Output:

```
Breadth First Traversal: 1 2 3 4 5 6 7 0 8
```

ii) DFS:

Code:

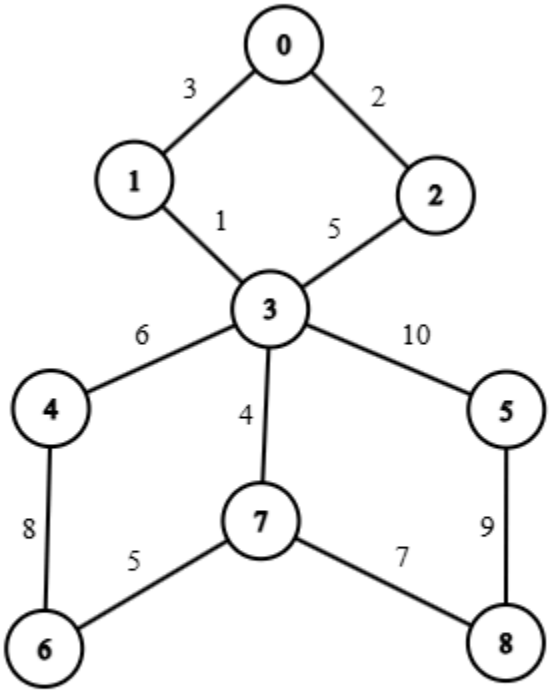
```
#include<bits/stdc++.h>
using namespace std;
class Graph
{
private:
int V;
list<int> *neighbour;
vector<bool> visited;
public:
Graph(int v){
V=v;
neighbour=new list<int>[V];
```

```
visited.resize(v,false);
}
void addEdge(int u,int v,bool undir=true){
neighbour[u].push_back(v);
if(undir){
neighbour[v].push_back(u);
}
}
void DFS(int r){
visited[r]=true;
cout<<r<<" ";
for(auto nbr:neighbour[r]){
if(!visited[nbr]){
DFS(nbr);
}
}
}
};
int main(){
    Graph graph(9);
    graph.addEdge(0, 1);
    graph.addEdge(1, 2);
    graph.addEdge(2, 3);
    graph.addEdge(3, 4);
    graph.addEdge(4, 5);
    graph.addEdge(5, 6);
    graph.addEdge(6, 7);
    graph.addEdge(7, 0);
    graph.addEdge(7, 8);
    graph.addEdge(8, 3);
cout << "Depth First Traversal "<<endl;
graph.DFS(1);
return 0;
}
```

Output:

```
Depth First Traversal
1 0 7 6 5 4 3 2 8
```

Graph used for Prim's, Kruskal's and Djikstra's algorithms



iii) PRIM'S ALGORITHM

Code:

```
#include <iostream>
#include <vector>
#include <list>
#include <climits>

#define INF INT_MAX
using namespace std;

class Graph {
private:
    int V;
    list<pair<int, int>> *l;
```

```
public:
    Graph(int v) {
        V = v;
        l = new list<pair<int, int>>[V];
    }

    void addEdge(int u, int v, int wt, bool undir = true) {
        l[u].push_back({wt, v});
        if (undir) {
            l[v].push_back({wt, u});
        }
    }

    int minkey(vector<int> &key, vector<bool> &mstSet) {
        int min_value = INF, min_idx = -1;
        for (int i = 0; i < V; i++) {
            if (!mstSet[i] && key[i] < min_value) {
                min_value = key[i];
                min_idx = i;
            }
        }
        return min_idx;
    }

    void printMST(vector<int> &parent, vector<int> &key) {
        int totalCost = 0;
        cout << "Edge \tWeight\n";
        for (int i = 1; i < V; i++) {
            cout << parent[i] << " - " << i << " \t" << key[i] << endl;
            totalCost += key[i];
        }
        cout << "Total Cost = " << totalCost << endl;
    }

    void prims() {
        vector<int> key(V, INF);
        vector<int> parent(V, -1);
        vector<bool> mstSet(V, false);

        key[0] = 0;

        for (int i = 0; i < V - 1; i++) {
            int node = minkey(key, mstSet);
            if (node == -1) break; // Safety check in case all nodes are visited
```

```
            mstSet[node] = true;

            for (auto nbr : l[node]) {
                int weight = nbr.first;
                int n = nbr.second;
                if (!mstSet[n] && weight < key[n]) {
                    parent[n] = node;
                    key[n] = weight;
                }
            }
        }
        printMST(parent, key);
    }

    ~Graph() {
        delete[] l;  // Free allocated memory
    }
};

int main() {
    Graph g(9);
    g.addEdge(0, 2, 2);
    g.addEdge(0, 1, 3);
    g.addEdge(1, 3, 1);
    g.addEdge(2, 3, 5);
    g.addEdge(3, 4, 6);
    g.addEdge(3, 5, 10);
    g.addEdge(3, 7, 4);
    g.addEdge(4, 6, 8);
    g.addEdge(6, 7, 5);
    g.addEdge(7, 8, 7);
    g.addEdge(8, 5, 9);

    cout << "PRIM'S ALGORITHM OUTPUT:\n";
    g.prims();

    return 0;
}
```

Output:

```
PRIM'S ALGORITHM OUTPUT:
Edge    Weight
0 - 1   3
0 - 2   2
1 - 3   1
3 - 4   6
8 - 5   9
7 - 6   5
3 - 7   4
7 - 8   7
Total Cost = 37
```

iv) KRUSKAL'S ALGORITHM

Code:

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

typedef pair<int, int> iPair;

class Graph {
    int V;
    vector<pair<int, iPair>> edges;

public:
    Graph(int V) { this->V = V; }

    void addEdge(int u, int v, int w) {
        edges.push_back({w, {u, v}});
    }

    int kruskalMST();
};

class DisjointSets {
    vector<int> parent, rank;

public:
    DisjointSets(int n) {
        parent.resize(n + 1);
```

```
        rank.resize(n + 1, 0);

        for (int i = 0; i <= n; i++)
            parent[i] = i;
    }

    int find(int u) {
        if (u != parent[u])
            parent[u] = find(parent[u]);
        return parent[u];
    }

    void merge(int x, int y) {
        x = find(x), y = find(y);
        if (rank[x] > rank[y])
            parent[y] = x;
        else {
            parent[x] = y;
            if (rank[x] == rank[y])
                rank[y]++;
        }
    }
};


int Graph::kruskalMST() {
    int mst_wt = 0;
    sort(edges.begin(), edges.end());

    DisjointSets ds(V);
    cout << "Edges in the MST (Kruskal):\n";

    for (auto &edge : edges) {
        int u = edge.second.first;
        int v = edge.second.second;
        int weight = edge.first;

        int set_u = ds.find(u);
        int set_v = ds.find(v);
```

```
        if (set_u != set_v) {
            cout << u << " - " << v << " (Weight: " << weight << ")\n";
            mst_wt += weight;
            ds.merge(set_u, set_v);
        }
    }

    return mst_wt;
}

// Main function
int main() {
    Graph g(9);
    g.addEdge(0, 2, 2);
    g.addEdge(0, 1, 3);
    g.addEdge(1, 3, 1);
    g.addEdge(2, 3, 5);
    g.addEdge(3, 4, 6);
    g.addEdge(3, 5, 10);
    g.addEdge(3, 7, 4);
    g.addEdge(4, 6, 8);
    g.addEdge(6, 7, 5);
    g.addEdge(7, 8, 7);
    g.addEdge(8, 5, 9);

    cout << "Minimum Spanning Tree (MST) using Kruskal's Algorithm:\n";
    int mst_wt = g.kruskalMST();

    cout << "\nTotal Weight of MST: " << mst_wt << endl;
    return 0;
}
```

Output:

```
Minimum Spanning Tree (MST) using Kruskal's Algorithm:
Edges in the MST (Kruskal):
1 - 3 (Weight: 1)
0 - 2 (Weight: 2)
0 - 1 (Weight: 3)
3 - 7 (Weight: 4)
6 - 7 (Weight: 5)
3 - 4 (Weight: 6)
7 - 8 (Weight: 7)
8 - 5 (Weight: 9)

Total Weight of MST: 37
```

v) DJIKSTRA'S ALGORITHM

Code:

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

class Graph {
private:
    int V;
    list<pair<int, int>> *l; // (weight, destination node)

public:
    Graph(int v) {
        V = v;
        l = new list<pair<int, int>>[v];
    }

    ~Graph() {
        delete[] l; // Free dynamically allocated memory
    }

    void addEdge(int u, int v, int wt, bool undir = true) {
        l[u].push_back({wt, v});
        if (undir) {
            l[v].push_back({wt, u});
        }
```

```
    }

    vector<int> dijkstra(int src) {
        vector<int> dist(V, INT_MAX);
        priority_queue<pair<int, int>, vector<pair<int, int>>,
greater<pair<int, int>>> pq;

        dist[src] = 0;
        pq.push({0, src});

        while (!pq.empty()) {
            pair<int, int> top = pq.top();
            pq.pop();

            int nodeDist = top.first;  // Extract correct distance
            int node = top.second;     // Extract correct node

            // Iterate over the neighbors
            for (auto nbrPair : l[node]) {
                int currEdge = nbrPair.first;
                int nbr = nbrPair.second;

                if (nodeDist + currEdge < dist[nbr]) {
                    dist[nbr] = nodeDist + currEdge;
                    pq.push({dist[nbr], nbr});
                }
            }
        }
        return dist;
    }
};

int main() {
    Graph g(9);
    g.addEdge(0, 2, 2);
    g.addEdge(0, 1, 3);
    g.addEdge(1, 3, 1);
    g.addEdge(2, 3, 5);
    g.addEdge(3, 4, 6);
    g.addEdge(3, 5, 10);
```

```
    g.addEdge(3, 7, 4);
    g.addEdge(4, 6, 8);
    g.addEdge(6, 7, 5);
    g.addEdge(7, 8, 7);
    g.addEdge(8, 5, 9);

    vector<int> shrtDist = g.dijkstra(0);

    cout << "Source -> Destination | Shortest Distance\n";
    for (int i = 0; i < shrtDist.size(); i++) {
        cout << "0 -> " << i << " = " << shrtDist[i] << endl;
    }

    return 0;
}
```

Output:

```
Source -> Destination | Shortest Distance
0 -> 0 = 0
0 -> 1 = 3
0 -> 2 = 2
0 -> 3 = 4
0 -> 4 = 10
0 -> 5 = 14
0 -> 6 = 13
0 -> 7 = 8
0 -> 8 = 15
```