Assignment No : 07

PROGRAM

```
#include <iostream>
using namespace std;
#include <conio.h>
struct tree
{
    tree *l, *r;
    int data;
}*root = NULL, *p = NULL, *np = NULL, *q;

void create()
{
    int value,c = 0;
    while (c < 7)
    {
        if (root == NULL)
        {
            root = new tree;
            cout<<"enter value of root node\n";
            cin>>root->data;
            root->r=NULL;
            root->l=NULL;
        }
        else
        {
            p = root;
            cout<<"enter value of node\n";
            cin>>value;
            while(true)
            {
                if (value < p->data)
                {
                    if (p->l == NULL)
                    {
                        p->l = new tree;
                        p = p->l;
                        p->data = value;
                        p->l = NULL;
                        p->r = NULL;
                        cout<<"value entered in left\n";
                        break;
                    }
                    else if (p->l != NULL)
                    {
                        p = p->l;
                    }
                }
                else if (value > p->data)
                {
                    if (p->r == NULL)
                    {
                        p->r = new tree;
                        p = p->r;
                        p->data = value;
```

```
                        p->l = NULL;
                        p->r = NULL;
                        cout<<"value entered in right\n";
                    break;
                }
                        else if (p->r != NULL)
                        {
                            p = p->r;
                        }
                    }
                }
            }
            c++;
        }
}
void inorder(tree *p)
{
    if (p != NULL)
    {
        inorder(p->l);
        cout<<p->data<<endl;
        inorder(p->r);
    }
}
void preorder(tree *p)
{
    if (p != NULL)
    {
        cout<<p->data<<endl;
        preorder(p->l);
        preorder(p->r);
    }
}
void postorder(tree *p)
{
    if (p != NULL)
    {
        postorder(p->l);
        postorder(p->r);
        cout<<p->data<<endl;
    }
}
int main()
{
    create();
    cout<<"printing traversal in inorder\n";
    inorder(root);
    cout<<"printing traversal in preorder\n";
    preorder(root);
    cout<<"printing traversal in postorder\n";
    postorder(root);
    getch();
}
```

OUTPUT

```
enter value of root node

10

enter value of node

5

value entered in left

enter value of node

15

value entered in right

enter value of node

4

value entered in left

enter value of node

6

value entered in right

enter value of node

14

value entered in left

enter value of node

16

value entered in right

printing traversal in inorder

4

5

6

10

14

15

16

printing traversal in preorder

10

5

4

6

15

14

16

printing traversal in postorder

4

6

5

14

16

15

10
```

Assignment No : 08

PROGRAM

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
};

class BST {
private:
    Node* root;

    Node* insertUtil(Node* root, int val) {
        if (root == nullptr) {
            return new Node(val);
        }

        if (val < root->data) {
            root->left = insertUtil(root->left, val);
        } else if (val > root->data) {
            root->right = insertUtil(root->right, val);
        }

        return root;
    }

    int longestPathUtil(Node* root) {
        if (root == nullptr) {
            return 0;
        }

        int leftDepth = longestPathUtil(root->left);
        int rightDepth = longestPathUtil(root->right);

        return 1 + max(leftDepth, rightDepth);
    }

    int findMinUtil(Node* root) {
        if (root == nullptr) {
            cout << "Tree is empty." << endl;
            return -1; // Return some default value indicating empty tree
        }

        while (root->left != nullptr) {
            root = root->left;
        }
```

```
        return root->data;
    }

    Node* swapPointersUtil(Node* root) {
        if (root == nullptr) {
            return nullptr;
        }

        Node* temp = root->left;
        root->left = root->right;
        root->right = temp;

        swapPointersUtil(root->left);
        swapPointersUtil(root->right);

        return root;
    }

    Node* searchUtil(Node* root, int val) {
        if (root == nullptr || root->data == val) {
            return root;
        }

        if (val < root->data) {
            return searchUtil(root->left, val);
        } else {
            return searchUtil(root->right, val);
        }
    }

public:
    BST() {
        root = nullptr;
    }

    void insert(int val) {
        root = insertUtil(root, val);
    }

    int longestPath() {
        return longestPathUtil(root);
    }

    int findMin() {
        return findMinUtil(root);
    }

    void swapPointers() {
        root = swapPointersUtil(root);
    }

    bool search(int val) {
        Node* result = searchUtil(root, val);
        return result != nullptr;
    }
};

int main() {
```

```
    BST tree;

    // Inserting values into the BST
    int values[] = {5, 3, 7, 1, 4, 6, 9};
    int numValues = sizeof(values) / sizeof(values[0]);

    for (int i = 0; i < numValues; i++) {
        tree.insert(values[i]);
    }

    // Example usage of BST functionalities
    cout << "Longest path in the tree: " << tree.longestPath() << endl;
    cout << "Minimum value in the tree: " << tree.findMin() << endl;

    cout << "Swapping left and right pointers at every node..." << endl;
    tree.swapPointers();

    int searchVal = 6;
    cout << "Searching for value " << searchVal << ": ";
    if (tree.search(searchVal)) {
        cout << "Found!" << endl;
    } else {
        cout << "Not Found!" << endl;
    }

    return 0;
}
```

OUTPUT

```
Longest path in the tree: 3

Minimum value in the tree: 1

Swapping left and right pointers at every node...

Searching for value 6: Not Found!
```

Assignment No : 09

PROGRAM

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

class Graph {
private:
    int V;
    vector<vector<int>> adjList;

public:
    Graph(int vertices) {
        V = vertices;
        adjList.resize(V);
    }

    void addEdge(int src, int dest) {
        adjList[src].push_back(dest);
    }

    void DFS(int start) {
        vector<bool> visited(V, false);
        DFSUtil(start, visited);
    }

    void DFSUtil(int vertex, vector<bool>& visited) {
        visited[vertex] = true;
        cout << vertex << " ";

        for (int i = 0; i < adjList[vertex].size(); ++i) {
            int adjVertex = adjList[vertex][i];
            if (!visited[adjVertex]) {
                DFSUtil(adjVertex, visited);
            }
        }
    }

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

        visited[start] = true;
        queue.push(start);

        while (!queue.empty()) {
            int current = queue.front();
            cout << current << " ";
            queue.pop();

            for (int i = 0; i < adjList[current].size(); ++i) {
                int adjVertex = adjList[current][i];
                if (!visited[adjVertex]) {
                    visited[adjVertex] = true;
```

```
                    queue.push(adjVertex);
                }
            }
        }
    }
};

int main() {
    int V, E;
    cout << "Enter the number of vertices: ";
    cin >> V;
    cout << "Enter the number of edges: ";
    cin >> E;

    Graph graph(V);

    cout << "Enter " << E << " edges (format: source destination):" << endl;
    for (int i = 0; i < E; ++i) {
        int src, dest;
        cin >> src >> dest;
        graph.addEdge(src, dest);
    }

    int startVertex;
    cout << "Enter the starting vertex for traversal: ";
    cin >> startVertex;

    cout << "DFS traversal starting from vertex " << startVertex << ": ";
    graph.DFS(startVertex);
    cout << endl;

    cout << "BFS traversal starting from vertex " << startVertex << ": ";
    graph.BFS(startVertex);
    cout << endl;

    return 0;
}
```

OUTPUT

```
Enter the number of vertices: 4

Enter the number of edges: 5

Enter 5 edges (format: source destination):

0 1

0 2

1 2

2 3

3 0

Enter the starting vertex for traversal: 0

DFS  traversal starting from vertex 0: 0 1 2 3

BFS traversal starting from vertex 0: 0 1 2 3
```

Assignment No : 10

PROGRAM

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct Edge {
    int src, dest, weight;
};

class Graph {
private:
    int V;
    vector<Edge> edges;

public:
    Graph(int vertices) : V(vertices) {}

    void addEdge(int src, int dest, int weight) {
        Edge edge;
        edge.src = src;
        edge.dest = dest;
        edge.weight = weight;
        edges.push_back(edge);
    }

    int find(vector<int>& parent, int i) {
        if (parent[i] == -1)
            return i;
        return find(parent, parent[i]);
    }

    void unionSet(vector<int>& parent, int x, int y) {
        int xroot = find(parent, x);
        int yroot = find(parent, y);
        parent[xroot] = yroot;
    }

    void kruskalMST() {
        vector<Edge> result(V - 1);
        sort(edges.begin(), edges.end(), [](const Edge& a, const Edge& b) {
            return a.weight < b.weight;
        });

        vector<int> parent(V, -1);

        int e = 0;
        int i = 0;
        while (e < V - 1 && i < edges.size()) {
            Edge next_edge = edges[i++];
            int x = find(parent, next_edge.src);
            int y = find(parent, next_edge.dest);

            if (x != y) {
```

```
                result[e++] = next_edge;
                unionSet(parent, x, y);
            }
        }

        cout << "Minimum Spanning Tree formed by connecting offices with minimum
cost:" << endl;
        for (int j = 0; j < V - 1; ++j) {
            cout << result[j].src << " - " << result[j].dest << " : " <<
result[j].weight << endl;
        }
    }
};

int main() {
    int numOffices, numConnections;
    cout << "Enter the number of offices: ";
    cin >> numOffices;
    cout << "Enter the number of connections: ";
    cin >> numConnections;

    Graph graph(numOffices);

    cout << "Enter " << numConnections << " connections in the format: src dest cost"
<< endl;
    for (int i = 0; i < numConnections; ++i) {
        int src, dest, cost;
        cin >> src >> dest >> cost;
        graph.addEdge(src, dest, cost);
    }

    graph.kruskalMST();

    return 0;
}
```

OUTCOME

```
Enter the number of offices: 4

Enter the number of connections: 5

Enter 5 connections in the format: src dest cost

0 1 4

0 2 1

1 2 3

2 3 2

3 0 5

Minimum Spanning Tree formed by connecting offices with minimum cost:

0 - 2 : 1

2 - 3 : 2

1 - 2 : 3
```