```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <queue>
using namespace std;


struct Edge // Struct to use with priority queue
{
    int vertex1;
    int vertex2;
    int weight;
    int pathlength;
    Edge()
    {
        vertex1 = 0;
        vertex2 = 0;
        weight = 0;
        pathlength = 0;
    }
};

class Graph
{
private:
    int adj[100][100];
    bool vertices[100];

public:
    Graph();
    void addVertex(int vertex);
    void addEdge(int vertex1, int vertex2, int weight);
    void removeEdge(int vertex1, int vertex2);
    void removeVertex(int vertex);
    bool isEdge(int vertex1, int vertex2);
    int edgeWeight(int vertex1, int vertex2);
    string toString();

    Graph minSpanningTree(Graph T, int X);
    int treeSum(Graph T);
    Graph shortestPathTree(Graph T, int X);
    void findPath(Graph& path, Graph& SPT, int vertex1, int vertex2,
        bool& visited);
    Graph path(Graph& SPT, int vertex1, int vertex2);
    int pathLength(Graph path);
    void readGraph(string filename);
};

Graph::Graph()
{
    for(int i=0; i<100; i++)
    {
        vertices[i] = false;
    }
    for(int i=0; i<100; i++)
        for(int j=0; j<100; j++)
        {
            adj[i][j] = 0;
            adj[j][i] = 0;
        }
}

void Graph::addVertex(int vertex)
{
    vertices[vertex] = true;
}

void Graph::addEdge(int vertex1, int vertex2, int weight)
{
    adj[vertex1][vertex2] = weight;
    adj[vertex2][vertex1] = weight;
}

void Graph::removeEdge(int vertex1, int vertex2)
{
    adj[vertex1][vertex2] = 0;
    adj[vertex2][vertex1] = 0;
}

void Graph::removeVertex(int vertex)
{
    vertices[vertex] = false;
    for(int i=0; i<100; i++)
    {
        if(adj[vertex][i] != 0)
        adj[vertex][i] = 0;
        if(adj[i][vertex] != 0)
        adj[i][vertex] = 0;
    }
}

bool Graph::isEdge(int vertex1, int vertex2)
{
    if(adj[vertex1][vertex2] != 0 || adj[vertex2][vertex1] != 0)
    return true;
    else
    return false;
}

int Graph::edgeWeight(int vertex1, int vertex2)
{
    return adj[vertex1][vertex2];
}

string Graph::toString()
{
    int i, j;
    string outputstring = "";
    string edges, verts;
    for(i=0; i<100; i++)
    {
        if(vertices[i] == true)
        {
            verts += to_string(i) + ", ";
        }
    }
    verts.erase(verts.end()-2, verts.end());
    for(i=0; i<100; i++)
        for(j=0; j<100; j++)
        {
            if(adj[i][j] != 0 && i < j)
            {
                edges += " " + to_string(i) + "-" + to_string(j) + "(" +
                to_string(adj[i][j]) + ")" + ",";
            }
        }
        edges.pop_back();
    outputstring = "{" + verts + ":" + edges + "}";
    return outputstring;
}

//Comparator function for the priority queue that compares edge weights.
struct compare
{
    bool operator()(const Edge& l, const Edge& r) const
    {
        return l.weight > r.weight;
    }
};

//Comparator function for the priority queue that compares path length.
struct compare2
{
    bool operator()(const Edge& l, const Edge& r) const
    {
        return l.pathlength > r.pathlength;
    }
};


Graph Graph::minSpanningTree(Graph T, int X)
{ // Using Prim's Algorithm
    Graph MST;
    Edge set;
    bool inMST[100];
    priority_queue <Edge, vector<Edge>, compare> pq;
    for(int i=0; i<100; i++)
    // Finds all edges from X and puts them in the queue
    {
        inMST[i] = false;
        if(T.isEdge(X,i))
        {
            set.vertex1 = X;
            set.vertex2 = i;
```

```
                set.weight = T.edgeWeight(X,i);
                pq.push(set);
            }
        }
    while(!pq.empty())
    // Checks the top of the queue , which is the least weight,
    // and adds to graph if vertex 2 has not been visited
    {
        int u = pq.top().vertex1;
        int v = pq.top().vertex2;
        int z = pq.top().weight;
        pq.pop();
        inMST[u] = true;
        if(inMST[v] == false)// If true then vertex 2 is in the graph
        {
            MST.addVertex(u);
            MST.addVertex(v);
            MST.addEdge(u,v,z);
            inMST[v] = true;
            for(int i=0; i<100; i++)
            {
                if(T.isEdge(v, i))
                {
                    set.vertex1 = v;
                    set.vertex2 = i;
                    set.weight = T.edgeWeight(v,i);
                    pq.push(set);
                }
            }
        }
    }
    return MST;
}

int Graph::treeSum(Graph T)
{
    int sum = 0;
    for(int i=0; i<100; i++)
        for(int j=0; j<100; j++)
        {
            if(T.isEdge(i, j) && i<j)
            sum += T.edgeWeight(i,j);
        }
    return sum;
}

Graph Graph::shortestPathTree(Graph T, int X)
{
    Graph SPT;
    Edge set;
    bool inSPT[100];
    set.pathlength = 0;
    priority_queue <Edge, vector<Edge>, compare2> pq;
    for(int i=0; i<100; i++)
    // Finds all edges from X and puts them in the queue
    {
        inSPT[i] = false;
        if(T.isEdge(X,i))
        {
            set.vertex1 = X;
            set.vertex2 = i;
            set.weight = T.edgeWeight(X,i);
            set.pathlength = T.edgeWeight(X,i);
            pq.push(set);
        }
    }
    while(!pq.empty())
    // Checks the top of the queue, which is the shortest path,
    // and adds to graph if vertex 2 has not been visited
    {
        int u = pq.top().vertex1;
        int v = pq.top().vertex2;
        int z = pq.top().weight;
        pq.pop();
        inSPT[u] = true;
        if(inSPT[v] == false) // If true then vertex 2 is in the graph
        {
            SPT.addVertex(u);
            SPT.addVertex(v);
            SPT.addEdge(u, v, z);
            inSPT[v] = true;
            for(int i=0; i<100; i++)
            {
                if(T.isEdge(v, i))
                {
                    set.vertex1 = v;
                    set.vertex2 = i;
                    set.weight = T.edgeWeight(v,i);
                    set.pathlength = T.edgeWeight(u,v) + T.edgeWeight(v,i);
                    pq.push(set);
                }
            }
        }
    }
    return SPT;
}

void Graph::findPath(Graph& path, Graph& SPT, int X, int Y, bool& visited)
{
    // Base case, if Y is found, add the edge and vertexes
    if(SPT.isEdge(X,Y)) {
        visited = true;
        path.addVertex(X);
        path.addEdge(X, Y, SPT.edgeWeight(X,Y));
        return;
    }
    // Check for edges between X and j, and adds the j vertexes to the graph
    for(int j = 0; (j<100 && !visited); j++) {
        if(SPT.isEdge(X, j) && path.vertices[j] == false){
            path.addVertex(j);
            findPath(path, SPT, j, Y, visited);
        // Recursively traverses the SPT and if does not find Y, visited is not
        // updated and then recursively removes the vertices not in the path
            if(visited) {
                path.addEdge(X, j, SPT.edgeWeight(X, j));
            }
            else {
                path.removeVertex(j);
            }
        }
    }
}

Graph Graph::path(Graph& SPT, int X, int Y)
{
    Graph path;
    path.addVertex(X);
    bool visited = false;
    findPath(path, SPT, X, Y, visited);
    return path;
}

int Graph::pathLength(Graph path)
{
    int sum = 0;
    for(int i=0; i<100; i++)
        for(int j=0; j<100; j++)
        {
            if(path.isEdge(i, j) && i<j)
            sum += path.edgeWeight(i,j);
        }
    return sum;
}

void Graph::readGraph(string filename)
{
    ifstream inf;
    inf.open(filename);
    char ch;
    int a,b,c;
    while(!inf.eof())
    {
        inf >> ch >> a >> b >> c >> ch >> ws;
        addVertex(a);
        addVertex(b);
        addEdge(a,b,c);
    }
}

int main()
{
    Graph g, MST, SPT, Path;
    string filename;
    filename = "graph10.txt";
    g.readGraph(filename);
    cout << endl;

    // Output for graph 10
    MST = g.minSpanningTree(g, 0);
    string s = MST.toString();
    for(int i = 0; i <= 100; i++) cout << "\u2014";
    cout << endl;
    cout << setw(70) << "***  Min Spanning Tree for Graph 10  ***" <<endl;
    for(int i = 0; i <= 100; i++) cout << "\u2014";
    cout << endl << endl;
    cout << s << endl << endl;
    cout << "TREE SUM = " << g.treeSum(MST) << endl << endl << endl;


    SPT = g.shortestPathTree(g, 1);
    string s1 = SPT.toString();
    for(int i = 0; i <= 100; i++) cout << "\u2014";
    cout << endl;
    cout << setw(70) << "***  Shortest Path Tree for Graph 10  ***" <<endl;
    for(int i = 0; i <= 100; i++) cout << "\u2014";
    cout << endl << endl;
    cout << s1 << endl << endl;
    cout << "TREE SUM = " << g.treeSum(SPT) << endl << endl << endl;


    Path = g.path(SPT, 1, 7);
    string s2 = Path.toString();
    for(int i = 0; i <= 100; i++) cout << "\u2014";
    cout << endl;
    cout << setw(70) << "***  Path from Vertex 1 to Vertex 7  ***" << endl;
    for(int i = 0; i <= 100; i++) cout << "\u2014";
    cout << endl << endl;
    cout << s2 << endl << endl;
    cout << "TOTAL LENGTH FROM VERTEX 1 TO VERTEX 7 = " << Path.treeSum(Path)
     << endl << endl << endl;


    // Output for graph 100
    filename = "graph100.txt";
    g.readGraph(filename);
    g.minSpanningTree(g, 0);

    MST = g.minSpanningTree(g, 3);
    string s3 = MST.toString();
    for(int i = 0; i <= 100; i++) cout << "\u2014";
    cout << endl;
    cout << setw(70) << "*** Min Spanning Tree for Graph 100 ***" <<endl;
    for(int i = 0; i <= 100; i++) cout << "\u2014";
    cout << endl << endl;
    cout << s3 << endl << endl;
    cout << "TREE SUM = " << g.treeSum(MST) << endl << endl << endl;


    SPT = g.shortestPathTree(g, 2);
    string s4 = SPT.toString();
    for(int i = 0; i <= 100; i++) cout << "\u2014";
    cout << endl;
    cout << setw(70) << "*** Shortest Path Tree for Graph 100 ***" <<endl;
    for(int i = 0; i <= 100; i++) cout << "\u2014";
    cout << endl << endl;
    cout << s4 << endl << endl;
    cout << "TREE SUM = " << g.treeSum(SPT) << endl << endl << endl;


    Path = g.path(SPT, 2, 87);
    string s5 = Path.toString();
    for(int i = 0; i <= 100; i++) cout << "\u2014";
    cout << endl;
    cout << setw(70) << "***  Path from Vertex 2 to Vertex 87  ***" << endl;
    for(int i = 0; i <= 100; i++) cout << "\u2014";
    cout << endl << endl;
    cout << s5 << endl << endl;
    cout << "TOTAL LENGTH FROM VERTEX 2 TO VERTEX 87 = " << Path.treeSum(Path)
     << endl << endl << endl;

    return 0;
}
```