main() function:

```
//============================================================================
// Name        : 21118_DSA_A07.cpp
// Author      : Shubham (Roll No: 21118)
//============================================================================

#include <iostream>

#include "mst.h"
using namespace std;

int main() {
	while (1) {
		int n, m, u, v, wt, opt;
		cout << "\n\nEnter number of vertices and edges.\n";
		cin >> n >> m;
		mst mymst(n, m);
		cout << "Enter edges and weights:\n";
		for (int i = 0; i < m; i++) {
			cin >> u >> v >> wt;
			mymst.addEdge(u, v, wt);
		}

		cout << "How do you want to calculate mst:\n";
		cout << "\t1 for Prim's Algorithm\n";
		cout << "\t2 for Kruskal's Algorithm\n";
		cin >> opt;

		if (opt == 1)
			cout << mymst.prims_mst_wt() << endl;
		else if (opt == 2)
			cout << mymst.kruskals_mst_wt() << endl;
		else
			cout << "INVALID CHOICE.\n";

		cout << "Do you want to calculate again?(0 or 1)\n";
		cin >> opt;
		if (opt == 0)
			break;
	}

	return 0;
}
```

mst.h header file(contain declaration of mst class):

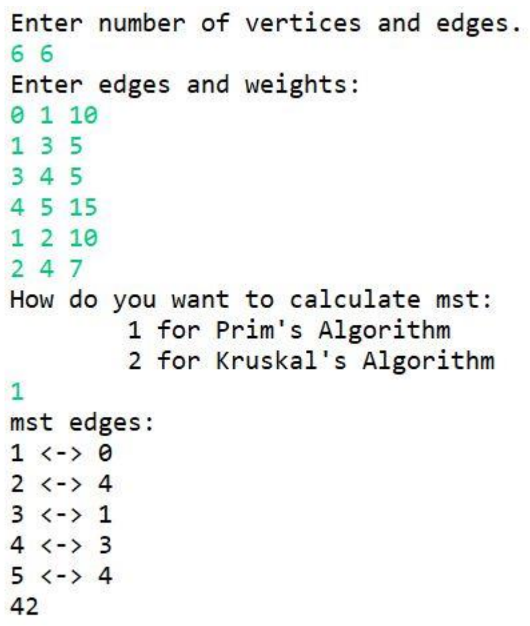
```
/*
 * kruskals_mst.h
 *
 *  Created on: 25-May-2021
 *      Author: Shubham
 */

#ifndef MST_H_
#define MST_H_

class Edge {
```

```
private:
	int u, v, wt;
public:
	Edge() {u=0, v=0, wt=0;}
	Edge(int x, int y, int w);
	friend class mst;
};

class mst {
private:
	int vertices, edges;
	int** adj_list;
	int* par; // to keep track of root node of mst in disjoint mst's (kruskal's 
algo)

	int find_mst(int u);
	void union_mst(int u1, int u2);
public:
	mst(int n, int m);
	void printGraph();
	void addEdge(int u, int v, int w);
	int prims_mst_wt();
	int kruskals_mst_wt();
	~mst();
};



#endif /* MST_H_ */
```

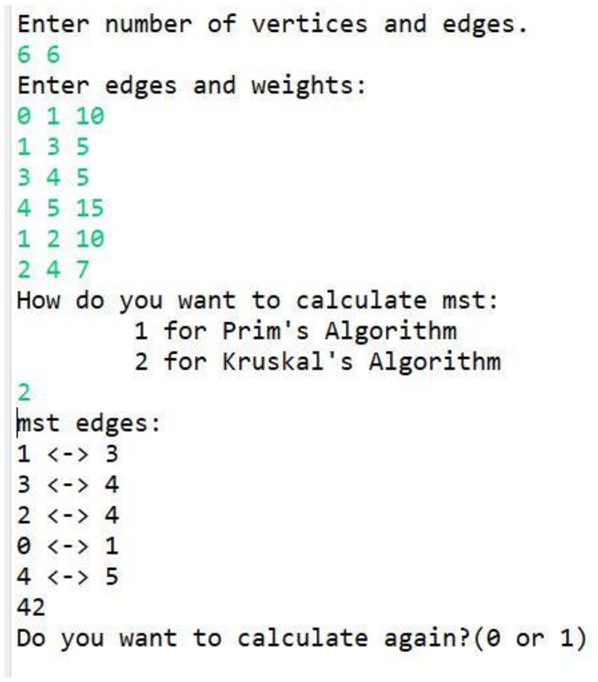
mst.cpp file(contains implementation of mst class):

```
/*
 * mst.cpp
 *
 *  Created on: 25-May-2021
 *      Author: Shubham
 */

#include <iostream>
#include <algorithm> // for sort function

#include "mst.h"
using namespace std;

Edge::Edge(int x, int y, int w) {
	u = x, v = y, wt = w;
}

// find mst of vertex u
int mst::find_mst(int u) {
	while (par[u] != u)
		u = par[u];
	return u;
}

// take union of mst's represented by u1 and u2
void mst::union_mst(int u1, int u2) {
	int x = find_mst(u1), y = find_mst(u2);
	if (x != y)
		par[x] = y;
}

mst::mst(int n, int m) {
	vertices = n, edges = m;
	par = new int[n];
	adj_list = new int*[n];
	for (int i = 0; i < n; i++) {
		adj_list[i] = new int[n];
		par[i] = i;
	}
	for (int i = 0; i < n; i++)
		for (int j = 0; j < n; j++)
			adj_list[i][j] = 0;
}

void mst::addEdge(int u, int v, int w) {
	adj_list[u][v] = w;
	adj_list[v][u] = w;
}

int mst::prims_mst_wt() {
	bool mst_vert[vertices] = { 0 };
	int vert_wt[vertices], mst_wt = 0, parent_vert[vertices];
	for (int i = 0; i < vertices; i++)
		vert_wt[i] = 10000, parent_vert[i] = -1;

	vert_wt[0] = 0;
	while (1) {
		int v = -1;
		for (int i = 0; i < vertices; i++)
			if (!mst_vert[i] && (v == -1 || vert_wt[i] < vert_wt[v]))
				v = i;
		if (v == -1)
			break;

		mst_vert[v] = 1;
		if (vert_wt[v] == 10000)
			vert_wt[v] = 0;
		mst_wt += vert_wt[v];
		for (int j = 0; j < vertices; j++)
			if (adj_list[v][j] && !mst_vert[j]
					&& (adj_list[v][j] < vert_wt[j])) {
				vert_wt[j] = adj_list[v][j];
				parent_vert[j] = v;
			}
	}

	cout << "mst edges:\n";
	for (int i = 0; i < vertices; i++)
		if (parent_vert[i] != -1)
			cout << i << " <-> " << parent_vert[i] << '\n';

	return mst_wt;
}

int mst::kruskals_mst_wt() {
	int mst_wt = 0;

	Edge edge_list[edges];
	int cnt = 0;
	for (int i = 0; i < vertices; i++)
		for (int j = i + 1; j < vertices; j++)
			if (adj_list[i][j])
				edge_list[cnt++] = Edge(i, j, adj_list[i][j]);

	// sort function from c++ standard template library with custom comparator
(using lambda fn)
	sort(edge_list, edge_list + edges, [](Edge e1, Edge e2) {
		return e1.wt < e2.wt;
	});

	int edge_cnt = 0, i = 0, mst_edges[vertices - 1][2];
	while (edge_cnt < vertices - 1 && i < edges) {
		Edge cur = edge_list[i++];
		int u_par = find_mst(cur.u), v_par = find_mst(cur.v);
		if (u_par != v_par) {
			mst_edges[edge_cnt][0] = cur.u, mst_edges[edge_cnt][1] =
cur.v;
			mst_wt += cur.wt, edge_cnt++;
			union_mst(u_par, v_par);
		}
	}

	cout << "mst edges:\n";
	for (int i = 0; i < edge_cnt; i++)
		cout << mst_edges[i][0] << " <-> " << mst_edges[i][1] << endl;

	return mst_wt;
}

void mst::printGraph() {
	cout << "Adjacency matrix representation of graph: \n";
	for (int i = 0; i < vertices; i++, cout << '\n')
		for (int j = 0; j < vertices; j++)
			cout << adj_list[i][j] << " ";
}

mst::~mst() {
	for (int i = 0; i < vertices; i++)
		delete[] adj_list[i];
	delete[] adj_list;
	delete[] par;
}
```

TESTCASE 1:

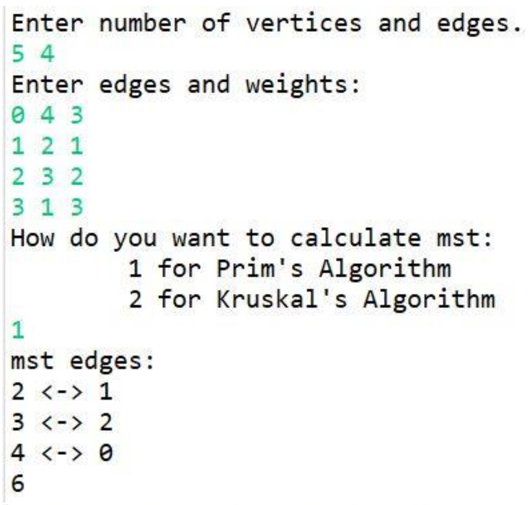


```
Enter number of vertices and edges.
6 6
Enter edges and weights:
0 1 10
1 3 5
3 4 5
4 5 15
1 2 10
2 4 7
How do you want to calculate mst:
        1 for Prim's Algorithm
        2 for Kruskal's Algorithm
1
mst edges:
1 <-> 0
2 <-> 4
3 <-> 1
4 <-> 3
5 <-> 4
42
```

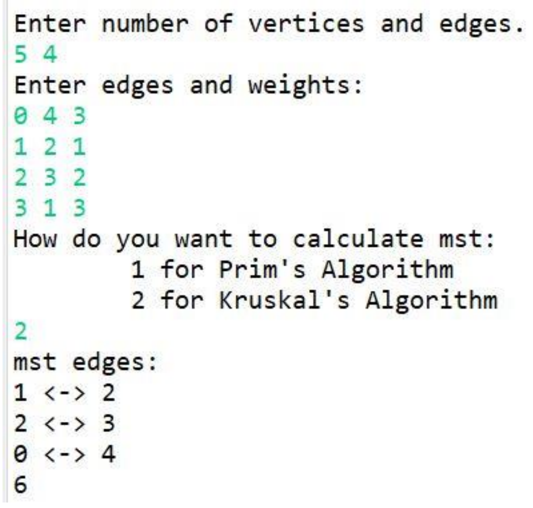


```
Enter number of vertices and edges.
6 6
Enter edges and weights:
0 1 10
1 3 5
3 4 5
4 5 15
1 2 10
2 4 7
How do you want to calculate mst:
        1 for Prim's Algorithm
        2 for Kruskal's Algorithm
2
mst edges:
1 <-> 3
3 <-> 4
2 <-> 4
0 <-> 1
4 <-> 5
42
Do you want to calculate again?(0 or 1)
```

TESTCASE 2:



```
Enter number of vertices and edges.
5 4
Enter edges and weights:
0 4 3
1 2 1
2 3 2
3 1 3
How do you want to calculate mst:
        1 for Prim's Algorithm
        2 for Kruskal's Algorithm
1
mst edges:
2 <-> 1
3 <-> 2
4 <-> 0
6
```



```
Enter number of vertices and edges.
5 4
Enter edges and weights:
0 4 3
1 2 1
2 3 2
3 1 3
How do you want to calculate mst:
        1 for Prim's Algorithm
        2 for Kruskal's Algorithm
2
mst edges:
1 <-> 2
2 <-> 3
0 <-> 4
6
```