```cpp
#include <iostream>
#include <vector>

/// This header declares prirority_queue
#include <queue>

using namespace std;

class Edge {
public:
    int node1;
    int weight;

    Edge(int node1, int weight) {
        this->node1 = node1;
        this->weight = weight;
    };
};

struct EdgeComparator {
    bool operator()(const Edge& e0, const Edge& e1) {
        return e0.weight > e1.weight;
    };
};

void prepareSpace();
void insertData();

vector< vector<Edge> > vNode;
vector<bool> vVisit;
vector<Edge> vMst;
int weightSum;

void mst(int src) {
    weightSum = 0;
    priority_queue<Edge, vector<Edge>, EdgeComparator> pq;

    while(true) {
        vVisit[src] = true;

        /// Add edges to PriorityQueue
        int nNeighbors = vNode[src].size();
        for(int i = 0; i < nNeighbors; ++i) {
            Edge e = vNode[src][i];
            int id = e.node1;
            if(vVisit[id] == false) {
                pq.push(e);
            }
        }

        /// Poll the priority queue for minimum-weight edge
        Edge minEdge(-1, -1);
        while(pq.size() > 0) {
            Edge e = pq.top();
            pq.pop();

            /// Check if edge connects outside of super node.
            if(vVisit[e.node1] == false) {
                minEdge = e;
                break;
            }
        }

        if(minEdge.node1 < 0) { /// No valid edge left, finish
            break;
        } else {
            weightSum += minEdge.weight;
            vMst.push_back(minEdge);
            src = minEdge.node1;
        }
    }

    cout << "Edge found = " << vMst.size() << endl;
    cout << "Weight sum = " << weightSum;
}

int main() {
    prepareSpace();
    insertData();
    mst(0);

    return 0;
}

void prepareSpace() {
    vNode.resize(5);
    vVisit.resize(5);
}

void insertData() {
    vNode[0].push_back(Edge(1, 4));

    vNode[0].push_back(Edge(1, 4));
    vNode[0].push_back(Edge(2, 4));
    vNode[0].push_back(Edge(3, 6));
    vNode[0].push_back(Edge(4, 6));

    vNode[1].push_back(Edge(0, 4));
    vNode[1].push_back(Edge(2, 2));

    vNode[2].push_back(Edge(0, 4));
    vNode[2].push_back(Edge(1, 2));
    vNode[2].push_back(Edge(3, 8));

    vNode[3].push_back(Edge(0, 6));
    vNode[3].push_back(Edge(2, 8));
    vNode[3].push_back(Edge(4, 9));

    vNode[4].push_back(Edge(0, 6));
    vNode[4].push_back(Edge(3, 9));

    std::fill(vVisit.begin(), vVisit.end(), false);
}
```