```
#include <iostream>
#include "console.h"
#include "basicgraph.h"
#include "simpio.h"
#include "set.h"
#include "priorityqueue.h"
#include "gwindow.h"
#include "gobjects.h"
#include "random.h"
#include "map.h"
#include <cmath>

using namespace std;

const bool GRID_GRAPH = true;

void kruskal(BasicGraph &graph, BasicGraph &mst);
void drawGraph(GWindow &gw, BasicGraph &graph, string title);
void adjustWindows(GWindow &gw1,GWindow &gw2);
void populateGraph(BasicGraph &graph,bool randomVerts = false);

int main() {
    // create two windows and stack them horizontally
    GWindow gwOrig,gwMst;

    // adjust the windows to make them look nice
    adjustWindows(gwOrig,gwMst);

    BasicGraph graph;
    string response = getLine("Do you want a random graph? (y/n)");
    if (response.size() > 0 && response[0] == 'y') {
        populateGraph(graph,true);
    } else {
        populateGraph(graph,false);
    }

    drawGraph(gwOrig,graph,"Original");

    BasicGraph mst;
    kruskal(graph,mst);
    drawGraph(gwMst, mst, "Minimum Spanning Tree");

    return 0;
}
```

```
 46 void kruskal(BasicGraph& graph, BasicGraph &mst) {
 47     // first, copy the graph and remove the edges
 48     // This is so we can populate it later with the mst edges
 49     mst = graph;
 50     mst.clearEdges();
 51     // put each vertex into a 'cluster', initially containing only itself
 52     Map<Vertex*, Set<Vertex*>* > clusters;
 53     Set<Vertex*> allVertices = graph.getVertexSet();
 54     Vector<Set<Vertex*>* > allSets;   // for freeing later
 55     for (Vertex* v : allVertices) {
 56         Set<Vertex*>* set = new Set<Vertex*>();
 57         set->add(v);
 58         clusters[v] = set;
 59         allSets.add(set);
 60     }
 61
 62     // put all edges into a priority queue, sorted by weight
 63     PriorityQueue<Edge*> pq;
 64     Set<Edge*> allEdges = graph.getEdgeSet();
 65     for (Edge* edge : allEdges) {
 66         pq.enqueue(edge, edge->cost);
 67     }
 68
 69     // repeatedly pull min-weight edge out of PQ and add it to MST if its
 70     // endpoints are not already connected
 71     Set<Edge*> mstEdges;
 72     while (!pq.isEmpty()) {
 73         Edge* e = pq.dequeue();
 74         Set<Vertex*>* set1 = clusters[e->start];
 75         Set<Vertex*>* set2 = clusters[e->finish];
 76         if (set1 != set2) {
 77             mstEdges.add(e);
 78
 79             // merge the two sets
 80             set1->addAll(*set2);
 81             for (Vertex* v : *set1) {
 82                 Set<Vertex*>* setv = clusters[v];
 83                 if (setv != set1) {
 84                     clusters[v] = set1;
 85                 }
 86             }
 87         }
 88     }
 89
 90     for (Set<Vertex*>* set : allSets) {
 91         delete set;
 92     }
 93
 94     // populate the graph with the edges
 95     // We can't add the edge pointers directly
 96     // because that would cause trouble freeing later
 97     for (Edge *edge : mstEdges) {
 98         mst.addEdge(edge->start->name,edge->end->name,edge->cost,false);
 99     }
100 }
101
```

```
102 void drawGraph(GWindow &gw, BasicGraph &graph,string title) {
103     // always start with the same random seed
104     // so we always get the same drawing for the same
105     // vertices
106     if (!GRID_GRAPH) {
107         setRandomSeed(1234);
108     }
109     // place the title there
110     gw.setTitle(title);
111     Map<Vertex*,GPoint> vertexLocations;
112     int vertCount = 0; // for the grid
113     for (Vertex *vert : graph.getVertexSet()) {
114         int x,y;
115 
116         if (!GRID_GRAPH) {
117             // randomly locate vertex on graph
118             x = randomInteger(5,gw.getWidth()-6);
119             y = randomInteger(5,gw.getHeight()-6);
120         } else {
121             // put in a staggerd grid
122             const int NUMCOLS = 3;
123             int widthApart = gw.getWidth() / (NUMCOLS + 1) - 10; // make a bit smaller for scre
124             int heightApart = gw.getHeight() / 6; // could fix this
125             x = (widthApart + ((vertCount % NUMCOLS) + (vertCount / NUMCOLS)) * widthApart);
126             y = ((vertCount / NUMCOLS + 1 + vertCount) * heightApart);
127         }
128         GPoint p(x,y);
129         vertexLocations.add(vert,p);
130         // place a dot there
131         gw.fillOval(x,y,10,10);
132 
133         // place a label
134         GLabel *gl = new GLabel(vert->name);
135         gl->setFont("SansSerif-28");
136         gw.add(gl,p.getX(),p.getY());
137         vertCount++;
138     }
139 
140     Set<Set<string>>seenAlready;
141     for (Edge *edge : graph.getEdgeSet()) {
142         Set<string>sortedEdgeSet;
143         sortedEdgeSet.add(edge->start->name);
144         sortedEdgeSet.add(edge->end->name);
145 
146         if (!seenAlready.contains(sortedEdgeSet)) {
147             GPoint startPoint = vertexLocations[edge->start];
148             GPoint endPoint = vertexLocations[edge->end];
149             gw.drawLine(startPoint,endPoint);
150 
151             // find the 1/3 point of the line (midpoint would have many
152             // overlapping points for the grid)
153             int avgX = abs((startPoint.getX()*2 + endPoint.getX())/3.0);
154             int avgY = abs((startPoint.getY()*2 + endPoint.getY())/3.0);
155             GLabel *gl = new GLabel(doubleToString(edge->cost));
156             gl->setFont("SansSerif-28");
157             gw.add(gl,avgX,avgY);
158             // add an unambiguous edge to seenAlready
159             // so we don't put same undirected edge in twice
160             seenAlready.add(sortedEdgeSet);
161         }
162     }
163 }
164 
```

```
165 void adjustWindows(GWindow &gw1,GWindow &gw2) {
166     gw1.setSize(600,700);
167     gw2.setSize(600,700);
168     gw1.repaint();
169     gw2.setLocation(gw1.getLocation().getX()+gw1.getCanvasWidth(),
170                     gw1.getLocation().getY());
171 }
172
173 void populateGraph(BasicGraph &graph, bool randomVerts) {
174     if (!randomVerts) {
175         // add some vertices and edges
176         for (int i=0; i < 5; i++) {
177             Vertex *v = new Vertex("" + charToString(i+'a'));
178             graph.addVertex(v);
179         }
180
181         // add some edges with weights
182         graph.addEdge("a","b",9,false);
183         graph.addEdge("a","c",75,false);
184         graph.addEdge("b","c",95,false);
185         graph.addEdge("b","d",19,false);
186         graph.addEdge("b","e",42,false);
187         graph.addEdge("c","d",51,false);
188         graph.addEdge("d","e",31,false);
189     } else {
190        // add 10 vertices
191        for (int i=0; i < 10; i++) {
192            Vertex *v = new Vertex("" + charToString(i+'a'));
193            graph.addVertex(v);
194        }
195        // for each vertex, add up to three random edges
196        Vector<Vertex *> allVerts;
197        for (Vertex *v : graph.getVertexSet()) {
198            allVerts.add(v);
199        }
200        for (Vertex *v : graph.getVertexSet()) {
201            Set<Vertex *>usedAlready;
202            usedAlready.add(v); // don't use yourself
203            for (int i=0;i<3;i++) {
204               int vertIndex = randomInteger(0,graph.size()-1);
205               if (usedAlready.contains(allVerts[vertIndex])) {
206                   continue; // ignore
207               }
208               Vertex *second = allVerts[vertIndex];
209               // get a random cost
210               int cost = randomInteger(1,100);
211               // create an edge
212               graph.addEdge(v,second,cost);
213               usedAlready.add(second);
214            }
215        }
216     }
217 }
218
```