```cpp
/*
Filename: p10.cpp
Author(s): Zachary Rea and Parker Ross
Date: April 13 2023
Description: The cpp for p10
*/
#include "P10.h"
#include <iostream>

using namespace std;

//Graph class
//*************************************************************************
//Constructors and Destructors
//*************************************************************************
//Constructor
//Written by Zach
Graph::Graph(int n, bool directed) {
    this->n = n;
    this->directed = directed;
    labels = new intList(n);
    a = new int[n*n];
    iQ *q = new iQ(vCount);
    lambda = new int[n];
    set = new int[n];
    clear();
}
//*************************************************************************
//Destructor
//Written by Zach modified by Parker
Graph::~Graph() {
    delete[] a;
    delete labels;
}
//*************************************************************************
//Private Functions
//*************************************************************************
//Function to return the mapping of x,y
//Written by Zach
int Graph::ind(int x, int y) const{
    return x * n + y;
}
//*************************************************************************
//Function to return the vertex id of a given label
//Written by Zach modified by Parker
int Graph::labelToVid(int label) const{
    int rc = -1;
    if (isV(label)) {
        rc = labels->getIndex(label);
    }
    return rc;
}
//*************************************************************************
//Function to return the label of a given vid
//Written by Zach
int Graph::vidToLabel(int vid) const{
    int rc = -1;
    labels->readAt(vid, rc);
    return rc;
}
//*************************************************************************
//Function to populate lambda with the distances to all other nodes
//Written by Parker and Zach
void Graph::dijkstra(int s) {
    if (isV(vidToLabel(s))) {
        for (int i = 0; i < vCount; i++) {
            lambda[i] = INFINITE;
            set[i] = Y;
        }
```

```
70          lambda[s] = 0;
71          set[s] = X;
72          for (int i = 0; i < vCount; i++) {
73              int key;
74              labels->readAt(i,key);
75              if (isEdge(vidToLabel(s),key)) {
76                  lambda[i] = a[ind(s,labelToVid(key))];
77              }
78          }
79
80          int minV;
81          while(minLambdaY(minV)) {
82              set[minV] = X;
83              for (int i = 0; i < vCount; i++) {
84                  int key;
85                  labels->readAt(i,key);
86                  if (isEdge(vidToLabel(minV),key)) {
87                      lambda[i] = min(a[ind(minV,labelToVid(key))]+lambda[minV],lambda[i]);
88                  }
89              }
90          }
91      }
92  }
93  //*************************************************************************
94  //Function to return the minimum value of Y in lambda
95  //Written by Zach
96  bool Graph::minLambdaY(int &minV) {
97      bool rc = false;
98      minV = -1;
99      if (set) {
100         for (int i = 0; i < vCount; i++) {
101             if (set[i] == Y) {
102                 if (minV == -1) {
103                     minV = i;
104                 }
105                 minV = min(i,minV);
106                 rc = true;
107             }
108         }
109     }
110     return rc;
111 }
112 //*************************************************************************
113 //Public Functions
114 //*************************************************************************
115 //Function to create a vertex
116 //Written by Zach
117 bool Graph::createV(int label) {
118     bool rc = false;
119     bool check = isV(label);
120     if ((vCount < n) && (!check)) {
121         labels->add(label);
122         vCount++;
123         rc = true;
124     }
125     return rc;
126 }
127 //*************************************************************************
128 //Function to add an edge between two vertices
129 //Written by Zach modified by Parker
130 bool Graph::addEdge(int uLabel, int vLabel, int weight) {
131     bool rc = false;
132     bool checku = isV(uLabel);
133     bool checkv = isV(vLabel);
134     bool checkw = isEdge(uLabel,vLabel);
135     if ((!checkw) && (weight > 0)) {
136         bool createu = false;
137         bool createv = false;
138         if (!checku) {
```

```
            createu = createV(uLabel);
        } else {
            createu = true;
        }
        if (!checkv) {
            createv =  createV(vLabel);
        } else {
            createv = true;
        }
        if ((createu) && (createv)) {
            a[ind(labelToVid(uLabel),labelToVid(vLabel))] = weight;
            eCount++;
            rc = true;
        }
    }
    return rc;
}
//*****************************************************************************
//Function to delete an edge between two vertices
//Written by Zach
bool Graph::deleteEdge(int uLabel, int vLabel) {
    bool rc = false;
    bool checkw = isEdge(uLabel,vLabel);
    if (checkw) {
        a[ind(labelToVid(uLabel),labelToVid(vLabel))] = 0;
        eCount--;
        rc = true;
    }
    return rc;
}
//*****************************************************************************
//Function to clear the graph
//Written by Parker
void Graph::clear() {
    int ecount = 0;
    int vcount = 0;
        for (int i = 0; i < n*n ; i++){
            a [i] = 0;
        }
        labels->clear ();
}
//*****************************************************************************
//Function to check if there is an edge between two vertices
//Written by Parker
bool Graph::isEdge(int uLabel, int vLabel) const{
    bool rc = false;
    int uVid = labelToVid(uLabel);
    int vVid = labelToVid(vLabel);
    if ((uVid >= 0) && (vVid >= 0)){
        rc = a[ind(uVid,vVid)] > 0;
    }
    return rc;
}
//*****************************************************************************
//Function to check if there is a vertex at a given location
//Written by Zach modified by Parker
bool Graph::isV(int label) const {
    return (labels->getIndex(label) != -1);
}
//*****************************************************************************
//Function to return the in degree of a given label
//Written by Parker
int Graph::inDegree(int label) const{
    int inDeg = labelToVid(label);
    int rc = -1;
    if (inDeg >= 0) {
        rc = 0;
        for (int i = 0; i < vCount; i++) {
            if (a[ind(i, inDeg)]) {
```

```
                rc++;
            }
        }
    }
    return rc;
}
//*************************************************************************
//Function to return the out degree of a given label
//Written by Parker
int Graph::outDegree(int label) const{
    int outDeg = labelToVid(label);
    int rc = -1;
    if (outDeg >= 0) {
        rc = 0;
        for (int i = 0; i < vCount; i++) {
            if (a[ind(outDeg, i)]) {
                rc++;
            }
        }
    }
    return rc;
}
//*************************************************************************
//Function to return the number of vertices possible
//Written by Parker
int Graph::sizeV() const{
    return n;
}
//*************************************************************************
//Function to return the number of vertices
//Written by Parker
int Graph::sizeUsedV() const{
    return vCount;
}
//*************************************************************************
//Function to return the number of edges
//Written by Parker
int Graph::sizeE() const{
    return eCount;
}
//*************************************************************************
//Function to print the contents of the graph
//Written by Zach
void Graph::printIt() const{
    int r, c;

    cout << "Graph info:\n";
    cout << "  Graph size = " << n << endl;
    cout << "  vCount = " << vCount << endl;
    cout << "  eCount = " << eCount << endl;
    cout << "\nGraph contents:\n";
    for (r = 0; r < vCount; r++) {
        int key;
        labels->readAt(r,key);
    cout << "  Node(" << r << "," << key << "):";
    for (c = 0; c < vCount; c++) {
        cout << " " << a[ind(r,c)];
    }
    cout << endl;
    }

    cout << "Degree table (in, out)\n";

    for (r = 0; r < vCount; r++) {
        int key;
        labels->readAt(r,key);
    cout << "  Node(" << r << "," << key << "):";
    cout << " " << inDegree(key) << ", " << outDegree(key) << endl;
    }
```

```
277   }
278   //*************************************************************************
279   //Function to do a breadth first print
280   //Written by Zach
281   void Graph::bfPrint(int label) const {
282       iQ *q = new iQ(vCount);
283       int *traversal = new int[vCount];
284       int *mark = new int[vCount];
285       for (int i = 0; i < vCount; i++) {
286           traversal[i] = 0;
287           mark[i] = 0;
288       }
289       for (int i = 0; i < vCount; i++) {
290           if (traversal[i] == 0) {
291               int vid = labelToVid(label);
292               q->enq(vid);
293               int j = 0;
294               int current = 0;
295               while (q->count() > 0) {
296                   q->deq(current);
297                   int lab = vidToLabel(current);
298                   cout << "\t\t\tItem " << j << " is (" << current << "," <<
299                   lab << ")\n";
300                   j++;
301                   for (int i = 0; i < vCount; i++) {
302                       if (isEdge((lab), vidToLabel(i))) {
303                           if (mark[i] == 0) {
304                               mark[i] = 1;
305                               q->enq(i);
306                           }
307                       }
308                   }
309               }
310               for (int j = 0; j < vCount; j++) {
311                   if (mark[j] == 1) {
312                       traversal[j] = 1;
313                   }
314               }
315           }
316       }
317   }
318   //*************************************************************************
319   //Function to tell if there is a path between two nodes
320   //Written by Zach
321   bool Graph::isPath(int ulabel, int vlabel) const{
322       iQ *q = new iQ(vCount);
323       bool rc = false;
324       bool check1 = isV(ulabel);
325       bool check2 = isV(vlabel);
326       if ((check1 && check2) && (vlabel != ulabel)) {
327           int vid = labelToVid(ulabel);
328           int *mark = new int[vCount];
329           for (int i = 0; i < vCount; i++) {
330               mark[i] = 0;
331           }
332           q->enq(vid);
333           mark[vid] = 1;
334           int x = 0;
335           int current = 0;
336           while (q->count() > 0) {
337               q->deq(current);
338               if (vidToLabel(current) == vlabel) {
339                   rc = true;
340                   break;
341               }
342               for (int i = 0; i < vCount; i++) {
343                   int lab = vidToLabel(current);
344                   if (isEdge(lab, vidToLabel(i))) {
345                       if (mark[i] == 0) {
```

```
                        mark[i] = 1;
                        q->enq(i);
                    }
                }
            }
        }
    }
    return rc;
}
//*************************************************************************
//Function to print the paths of the graph
//Written by Parker
void Graph::printPaths() const{
    for (int i = 0; i < vCount; i++) {
        for (int j = 0; j < vCount; j++) {
            if (isPath(vidToLabel(i), vidToLabel(j))) {
                int key = 0;
                labels->readAt(i,key);
                cout << key << " does have a path to ";
                labels->readAt(j,key);
                cout << key << endl;
            } else {
                int key = 0;
                labels->readAt(i,key);
                cout << key << " does not have a path to ";
                labels->readAt(j,key);
                cout << key << endl;
            }
        }
    }
}
//*************************************************************************
//Function to return the distance between two nodes
//Written by Zach
bool Graph::dijkstra(int sLabel, int dLabel, int &distance) {
    bool rc = false;
    distance = -1;
    if ((isV(sLabel) && isV(dLabel))) {
        dijkstra(labelToVid(sLabel));
        distance = lambda[(labelToVid(dLabel))];
        rc = true;
    }
    return rc;
}
//*************************************************************************
//Non-Member Functions
//*************************************************************************
//Function to return the min of two values
//Written by Zach
int min(int x, int y) {
    int rc = 0;
    if (x < y) {
        rc = x;
    } else {
        rc = y;
    }
    return rc;
}
//*************************************************************************
//Function to return the max of two values
//Written by Zach
int max(int x, int y) {
    int rc = 0;
    if (x > y) {
        rc = x;
    } else {
        rc = y;
    }
    return rc;
```

```
415   }
416
417   /* Professor, /* Professor,
418   we understand that the output is incorrect,
419   we hope to apply corrections to dijkstra by the time p11 is due. */
```