```cpp
/*
Filename: p11.cpp
Author(s): Zachary Rea and Parker Ross
Date:20 April 2023
Description: The cpp for p11
*/
#include "P11.h"
#include <iostream>
#include <iomanip>

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;
    delete q;
}
//*************************************************************************
//Private Functions
//*************************************************************************
//Function to return the mapping of x,y
//Written by Zach
int Graph::ind(int x, int y) const{
    int rc = 0;
    if (directed) {
        rc = x * n + y;
    } else {
        rc = x<y?x*n+y:y*n+x;
    }
    return rc;
}
//*************************************************************************
//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;
}
//*************************************************************************
```

```
70  //Function to populate lambda with the distances to all other nodes
71  //Written by Parker and Zach, modified by Parker
72  void Graph::dijkstra(int s) {
73          for (int i = 0; i < vCount; i++) {
74              lambda[i] = INFINITE;
75              set[i] = Y;
76          }
77          lambda[s] = 0;
78          set[s] = X;
79          for (int i = 0; i < vCount; i++) {
80              int key = vidToLabel(s);
81              if (isEdge(vidToLabel(s),key)) {
82                  lambda[i] = a[ind(s,labelToVid(key))];
83              }
84          }
85          lambda[s] != 0;
86          int minV;
87          while(minLambdaY(minV)) {
88              set[minV] = X;
89              for (int i = 0; i < vCount; i++) {
90                  int key  = vidToLabel(s);
91                  if (isEdge(vidToLabel(minV),key)) {
92                      lambda[i] = min(a[ind(minV,i)]+lambda[minV],lambda[i]);
93                  }
94              }
95          }
96
97  }
98  //*************************************************************************
99  //Function to return the minimum value of Y in lambda
100 //Written by Zach modified by Parker
101 bool Graph::minLambdaY(int &minV) {
102     bool rc = false;
103     minV = -1;
104     int minVal = INFINITE;
105         for (int i = 0; i < vCount; i++) {
106             if ((set[i] == Y) && (lambda[i] <= minVal)) {
107                 minV = i;
108                 minVal = lambda[i];
109                 rc = true;
110             }
111         }
112     return rc;
113 }
114 //*************************************************************************
115 //Function to check if directed graph is cyclic
116 //Written by Zach
117 bool Graph::isCyclicDirected() {
118     bool rc = false;
119     for (int i = 1; i < vCount; i++) {
120         if (isPath(vidToLabel(i),vidToLabel(i))) {
121             rc = true;
122         }
123     }
124     return rc;
125 }
126 //*************************************************************************
127 //Function to check if undirected graph is cyclic
128 //Written by Zach
129 bool Graph::isCyclicUndirected() {
130     bool rc = false;
131     for (int i = 1; i < vCount; i++) {
132         if (isPath(vidToLabel(i),vidToLabel(i))) {
133             rc = true;
134         }
135     }
136     return rc;
137 }
138 //*************************************************************************
```

```
139   //Public Functions
140   //************************************************************************
141   //Function to create a vertex
142   //Written by Zach
143   bool Graph::createV(int label) {
144       bool rc = false;
145       bool check = isV(label);
146       if ((vCount < n) && (!check)) {
147           labels->add(label);
148           vCount++;
149           rc = true;
150       }
151       return rc;
152   }
153   //************************************************************************
154   //Function to add an edge between two vertices
155   //Written by Zach modified by Parker
156   bool Graph::addEdge(int uLabel, int vLabel, int weight) {
157       bool rc = false;
158       bool checku = isV(uLabel);
159       bool checkv = isV(vLabel);
160       bool checkw = isEdge(uLabel,vLabel);
161       if ((!checkw) && (weight > 0)) {
162           bool createu = false;
163           bool createv = false;
164           if (!checku) {
165               createu = createV(uLabel);
166           } else {
167               createu = true;
168           }
169           if (!checkv) {
170               createv =  createV(vLabel);
171           } else {
172               createv = true;
173           }
174           if ((createu) && (createv)) {
175               a[ind(labelToVid(uLabel),labelToVid(vLabel))] = weight;
176               eCount++;
177               rc = true;
178           }
179       }
180       return rc;
181   }
182   //************************************************************************
183   //Function to delete an edge between two vertices
184   //Written by Zach
185   bool Graph::deleteEdge(int uLabel, int vLabel, int &weight) {
186       bool rc = false;
187       bool checkw = isEdge(uLabel,vLabel);
188       if (checkw) {
189           weight = a[ind(labelToVid(uLabel),labelToVid(vLabel))];
190           a[ind(labelToVid(uLabel),labelToVid(vLabel))] = 0;
191           eCount--;
192           rc = true;
193       }
194       return rc;
195   }
196   //************************************************************************
197   //Function to clear the graph
198   //Written by Parker
199   void Graph::clear() {
200       int ecount = 0;
201       int vcount = 0;
202           for (int i = 0; i < n*n ; i++){
203               a [i] = 0;
204           }
205           labels->clear ();
206   }
207   //************************************************************************
```

```
208  //Function to check if there is an edge between two vertices
209  //Written by Parker
210  bool Graph::isEdge(int uLabel, int vLabel) const{
211      bool rc = false;
212      int uVid = labelToVid(uLabel);
213      int vVid = labelToVid(vLabel);
214      if ((uVid >= 0) && (vVid >= 0)){
215          rc = a[ind(uVid,vVid)] > 0;
216      }
217      return rc;
218  }
219  //*************************************************************************
220  //Function to check if there is a vertex at a given location
221  //Written by Zach modified by Parker
222  bool Graph::isV(int label) const {
223      return (labels->getIndex(label) != -1);
224  }
225  //*************************************************************************
226  //Function to return the in degree of a given label
227  //Written by Zach
228  int Graph::inDegree(int label) const{
229      int inDeg = labelToVid(label);
230      int rc = -1;
231      if ((inDeg >= 0) && (directed)) {
232          rc = 0;
233          for (int i = 0; i < vCount; i++) {
234              if (a[ind(i, inDeg)]) {
235                  rc++;
236              }
237          }
238      }
239      return rc;
240  }
241  //*************************************************************************
242  //Function to return the out degree of a given label
243  //Written by Parker
244  int Graph::outDegree(int label) const{
245      int outDeg = labelToVid(label);
246      int rc = -1;
247      if ((outDeg >= 0) && (directed)) {
248          rc = 0;
249          for (int i = 0; i < vCount; i++) {
250              if (a[ind(outDeg, i)]) {
251                  rc++;
252              }
253          }
254      }
255      return rc;
256  }
257  //*************************************************************************
258  //Function to return the number of vertices possible
259  //Written by Parker
260  int Graph::sizeV() const{
261      return n;
262  }
263  //*************************************************************************
264  //Function to return the number of vertices
265  //Written by Parker
266  int Graph::sizeUsedV() const{
267      return vCount;
268  }
269  //*************************************************************************
270  //Function to return the number of edges
271  //Written by Parker
272  int Graph::sizeE() const{
273      return eCount;
274  }
275  //*************************************************************************
276  //Function to print the contents of the graph
```

```
277  //Written by Zach
278  void Graph::printIt() {
279      int r, c;
280
281      cout << "Graph info:\n";
282      cout << "  Graph size = " << n << endl;
283      cout << "  vCount = " << vCount << endl;
284      cout << "  eCount = " << eCount << endl;
285      cout << "\nGraph contents:\n";
286      for (r = 0; r < vCount; r++) {
287          int key;
288          labels->readAt(r,key);
289      cout << "  Node(" << r << "," << key << "):";
290      for (c = 0; c < vCount; c++) {
291          if (c > r) {
292              cout << "  " << a[ind(r,c)];
293          } else {
294              cout << "  0";
295          }
296      }
297      cout << endl;
298      }
299
300      cout << "Degree table (normal, in, out)\n";
301
302      for (r = 0; r < vCount; r++) {
303          int key;
304          labels->readAt(r,key);
305      cout << "  Node(" << r << "," << key << "):";
306      cout << "     " << degree(key) << ", " << inDegree(key) <<
307      ", " << outDegree(key) << endl;
308      }
309  }
310  //*************************************************************************
311  //Function to do a breadth first print
312  //Written by Zach modified by Parker
313  void Graph::bfPrint(int label) const {
314      iQ *q = new iQ(vCount);
315      bool *mark = new bool [vCount];
316      for (int i = 0; i < vCount; i++) {
317          mark[i] = false;
318      }
319              int vid = labelToVid(label);
320              q->enq(vid);
321              int j = 0;
322              int current;
323              while (q->count() > 0) {
324                  q->deq(current);
325                  int lab = vidToLabel(current);
326                  cout << "\t\t\t Item " << j << " is (" << current << "," <<
327                  lab << ")\n";
328                  j++;
329                  for (int i = 0; i < vCount; i++) {
330                      if (isEdge((lab), vidToLabel(i))) {
331                          if (mark[i] == false) {
332                              mark[i] = true;
333                              q->enq(i);
334                          }
335                      }
336                  }
337              }
338          }
339  //*************************************************************************
340  //Function to tell if there is a path between two nodes
341  //Written by Zach modified by Parker
342  bool Graph::isPath(int ulabel, int vlabel) const{
343      iQ *q = new iQ(vCount);
344      bool rc = false;
345      if (directed) {
```

```
346         if ((isEdge(ulabel,vlabel)) && (ulabel == vlabel)) {
347             rc = true;
348         }
349     }
350     bool check1 = isV(ulabel);
351     bool check2 = isV(vlabel);
352     if ((check1 && check2) && (ulabel != vlabel)) {
353         int vid = labelToVid(ulabel);
354         bool *mark = new bool [vCount];
355         for (int i = 0; i < vCount; i++) {
356             mark[i] = false;
357         }
358         q->enq(vid);
359         mark[vid] = true;
360         int current;
361         while (q->count() > 0) {
362             q->deq(current);
363             if (vidToLabel(current) == vlabel) {
364                 rc = true;
365                 break;
366             }
367             for (int i = 0; i < vCount; i++) {
368                 int lab = vidToLabel(current);
369                 if (isEdge(lab, vidToLabel(i))) {
370                     if (mark[i] == false) {
371                         mark[i] = true;
372                         q->enq(i);
373                     }
374                 }
375             }
376         }
377     }
378     return rc;
379 }
380 //*************************************************************************
381 //Function to print the paths of the graph
382 //Written by Parker
383 void Graph::printPaths() const{
384     for (int i = 0; i < vCount; i++) {
385         for (int j = 0; j < vCount; j++) {
386             if (isPath(vidToLabel(i), vidToLabel(j))) {
387                 int key = vidToLabel(i);
388                 cout << key << " does have a path to ";
389                 key = vidToLabel(j);
390                 cout << key << endl;
391             } else {
392                 int key = vidToLabel(i);
393                 cout << key << " does not have a path to ";
394                 key = vidToLabel(j);
395                 cout << key << endl;
396             }
397         }
398     }
399 }
400 //*************************************************************************
401 //Function to return the distance between two nodes
402 //Written by Zach
403 bool Graph::dijkstra(int sLabel, int dLabel, int &distance) {
404     bool rc = false;
405     distance = -1;
406     if ((isV(sLabel) && isV(dLabel))) {
407         dijkstra(labelToVid(sLabel));
408         distance = lambda[(labelToVid(dLabel))];
409         rc = true;
410     }
411     return rc;
412 }
413 //*************************************************************************
414 //Function to return the degree of a vertex in undirected graph
```

```
415   //Written by Zach
416   int Graph::degree(int label) {
417       int rc = -1;
418       if ((!directed) && (isV(label))) {
419           rc = 0;
420           for (int i = 0; i < vCount; i++) {
421               if ((i != labelToVid(label)) && (isEdge(label,vidToLabel(i)))) {
422                   rc++;
423               }
424           }
425       }
426       return rc;
427   }
428   //*****************************************************************************
429   //Function to call the cyclic checkers respectively
430   //Written by Zach
431   bool Graph::isCyclic() {
432       return (directed?isCyclicDirected():isCyclicUndirected());
433   }
434   //*****************************************************************************
435   //Non-Member Functions
436   //*****************************************************************************
437   //Function to return the min of two values
438   //Written by Zach
439   int min(int x, int y) {
440       int rc = 0;
441       if (x < y) {
442           rc = x;
443       } else {
444           rc = y;
445       }
446       return rc;
447   }
448   //*****************************************************************************
449   //Function to return the max of two values
450   //Written by Zach
451   int max(int x, int y) {
452       int rc = 0;
453       if (x > y) {
454           rc = x;
455       } else {
456           rc = y;
457       }
458       return rc;
459   }
```