Implementation of a simple graph with an adjacency matrix

Variables

numVertices: integer
numEdges: integer
vertex: array of vertices
edge: two dimensional array of edges
For each vertex, we keep track of the element associated with the vertex and the degree, the in-degree and the out-degree of the vertex, and its index, that is, a 5-tuple [*element*, *degree*, *in-degree*, *out-degree*, *index*]. For each edge, we keep track of the element associated with the edge, whether the edge is directed and the end vertices of the edge, that is, a 4-tuple [*element*, *directed?*, $vertex_1$, $vertex_2$] where $vertex_1$ is the origin of the edge and $vertex_2$ is the destination if the edge is directed.
invariant: for i = 0,..., *numVertices* − 1, *vertex*[i] contains the vertex with index i; for i = 0,..., *numVertices* − 1, j = 0,..., *numVertices* − 1 *edge*[i, j] contains edge *edge* iff *edge* is an edge between the vertices with indices i and j

Initialization

numVertices ← 0
numEdges ← 0

Algorithms

elements():
  output: collection of elements stored in positions of graph
col ← empty collection
for i = 0,..., *numVertices* − 1 **do**
    add element of vertex *vertex*[i] to *col*
for i = 0,..., *numVertices* − 1 **do**
    for j = 0,..., i − 1 **do**
        if *edge*[i, j] contains an edge **then**
            add element of edge *edge*[i, j] to *col*
        if *edge*[j, i] contains a directed edge **then**
            add element of edge *edge*[j, i] to *col*
return *col*

positions():
  output: collection of positions of graph
col ← empty collection
for i = 0,..., *numVertices* − 1 **do**
    add vertex *vertex*[i] to *col*
for i = 0,..., *numVertices* − 1 **do**
    for j = 0,..., i − 1 **do**
        if *edge*[i, j] contains an edge **then**
            add edge *edge*[i, j] to *col*
        if *edge*[j, i] contains a directed edge **then**
            add edge *edge*[j, i] to *col*
return *col*

numVertices():
  output: number of vertices of the graph
return *numVertices*

numEdges():
  output: number of edges of the graph
return *numEdges*

```
vertices():
   output: collection of the vertices of the graph
col ← empty collection
for i = 0, ..., numVertices − 1 do
     add vertex[i] to col
return col
```

```
edges():
   output: collection of the edges of the graph
col ← empty collection
for i = 0,..., numVertices − 1 do
     for j = 0,..., i − 1 do
          if edge[i, j] contains an edge then
               add edge edge[i, j] to col
          if edge[j, i] contains a directed edge then
               add edge edge[j, i] to col
return col
```

```
aVertex():
   precondition: the graph is nonempty
   output: a vertex of the graph
return vertex[0]
```

```
adjacentVertices(vertex):
   input: vertex the adjacent vertices of which are returned
   output: collection of vertices adjacent to vertex col ← empty collection
i ← index of vertex
for j = 0,..., numVertices − 1 do
     if edge[i, j] contains an edge then
          add vertex[j] to col
for j = 0,..., numVertices − 1 do
     if edge[j, i] contains a directed edge then
          add vertex[j] to col
return col
```

```
incidentEdges(vertex):
   input: vertex whose incident edges are returned
   output: collection of edges incident on vertex
i ← index of vertex
col ← empty collection
for j = 0,..., numVertices − 1 do
     if edge[i, j] contains an edge then
          add edge[i, j] to col
     if edge[j, i] contains a directed edge then
          add edge[j, i] to col
return col
```

```
areAdjacent(first, second):
   input: vertices
   output: first and second are adjacent?
i ← index of first
j ← index of second
return edge[i, j] contains an edge or edge[j, i] contains an edge
```

```
removeVertex(vertex):
   input: vertex to be removed
   postcondition: vertex and edges incident on vertex have been removed from graph
```

```
i ← index of vertex
for j = 0,..., numVertices − 1 do
    if edges[i, j] contains an edge then
        updateDegrees(edges[i, j])
    edges[i, j] ← edges[numVertices − 1, j]
for j = 0,..., numVertices − 1 do
    if edges[j, i] contains a directed edge then
        updateDegrees(edges[j, i])
    edges[j, i] ← edges[j, numVertices − 1]
vertex[i] ← vertex[numVertices − 1]
set index of vertex[i] to i
numVertices ← numVertices − 1
```

```
updateDegrees(edge):
  input: edge
  postcondition: degrees of the end vertices of edge have been updated
(first, second) ← end vertices of edge
degree of first ← degree of first − 1
degree of second ← degree of second − 1
if edge is directed then
    outdegree of first ← outdegree of first − 1
    indegree of second ← indegree of second − 1
```