Implementation of a simple undirected 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 $i = 0,\dots,$ *numVertices* − 1, *vertex*[*i*] contains the vertex with index *i*; for $i = 0,\dots,$ *numVertices* − 1, $j = 0,\dots,$ *numVertices* − 1 *edge*[*i, j*] contains edge *edge* iff *edge* is an edge between the vertices with indices *i* and *j*

Initialization

$numVertices \leftarrow 0$
$numEdges \leftarrow 0$

Algorithms

```
elements():
   output: collection of elements stored in positions of graph
col ← empty collection
for $i = 0,\dots, numVertices - 1$ do
      add element of vertex $vertex[i]$ to col
for $i = 0,\dots, numVertices - 1$ do
      for $j = 0,\dots, i - 1$ do
            if $edge[i,j]$ contains an edge then
                  add element of edge $edge[i,j]$ to col return col
```

```
positions():
   output: collection of positions of graph
col ← empty collection
for $i = 0,\dots, numVertices - 1$ do
      add vertex $vertex[i]$ to col
for $i = 0,\dots, numVertices - 1$ do
      for $j = 0,\dots, i - 1$ do
            if $edge[i,j]$ contains an edge then
                  add edge $edge[i,j]$ 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 $i = 0, \ldots, numVertices - 1$ do
    add $vertex[i]$ to $col$
return $col$

edges():
  output: collection of the edges of the graph
$col \leftarrow$ 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 $col$
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 $vertex$ $col \leftarrow$ empty collection
$i \leftarrow$ index of $vertex$
for $j = 0,..., numVertices - 1$ do
    if $edge[i, j]$ contains an edge then
        add $vertex[j]$ to $col$
return $col$

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

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

removeVertex($vertex$):
  input: vertex to be removed
  postcondition: $vertex$ and edges incident on $vertex$ have been removed from graph
$i \leftarrow$ index of $vertex$
for $j = 0,..., numVertices - 1$ do
    if $edges[i, j]$ contains an edge then
        degree of $vertex[j] \leftarrow$ degree of $vertex[j] - 1$
    $edges[i, j] \leftarrow edges[numVertices - 1, j]$
for $j = 0,..., numVertices - 1$ do
    $edges[j, i] \leftarrow edges[j, numVertices - 1]$
$vertex[i] \leftarrow vertex[numVertices - 1]$
```

```
set index of $vertex[i]$ to $i$
$numVertices \leftarrow numVertices - 1$
$numEdges \leftarrow numEdges -$ degree of $vertex$
```