SOEN 331 Assignment 1

David Bastien (26948553) and Michael Bilinsky (26992358)

**ALGEBRAIC SPECIFICATION:**

Graph ADT

**Spec:** Graph(Element)

**Sort:** Graph

**Imports:** **N**0, Boolean, Edge, Vertex, Set

**Description:** Graph operations are *newGraph* which returns a new empty Graph, *vertices* which returns a set of the vertices present in the graph, *edges* which returns a set of the edges present in a graph, *countAllVertices* which returns the number of vertices in a graph, *countAllEdges* which returns the number of edges in a graph, *getEdge* which returns the edge connecting two vertices defined in the operation, *incidentEdges* which returns a set of edges linking to a vertex defined in the operation, *opposite* which returns the second vertex of an edge other than the vertex defined in the operation, *endVertices* which returns a set of the two vertices connected to an edge defined in the operation, *areAdjacent* which returns a Boolean value determining whether two vertices defined in the operation are part of the same edge, *insertVertex* which adds a vertex to the defined graph, *removeVertex* which removes a vertex defined in the operation, *insertEdge* which adds an edge and its two vertices to a graph, *removeEdge* which removes an edge defined in the operation from a graph, *getEdgeElem* which returns the element contained in a specified edge, and, finally, *replaceEdgeElem* which changes the element of an edge with another element defined in the operation.

**Signature of Operations:**

1. newGraph 🡪 Graph;
2. vertices: Graph 🡪 Set;
3. edges: Graph 🡪 Set;
4. countAllVertices: Graph 🡪 **N**0;
5. countAllEdges: Graph 🡪 **N**0;
6. getEdge: Graph, Vertex v, Vertex w 🡪 Edge;
7. incidentEdges: Graph, Vertex v 🡪 Set;
8. opposite: Graph, Edge e, Vertex v 🡪 Vertex;
9. endVertices: Graph, Edge e 🡪 Set;
10. areAdjacent: Graph, Vertex v, Vertex w 🡪 Boolean;
11. insertVertex: Graph, Vertex v 🡪 Graph;
12. removeVertex: Graph, Vertex v 🡪 Graph;
13. insertEdge: Graph, Vertex v, Vertex w, Element x 🡪 Graph;
14. removeEdge: Graph, Vertex v, Vertex w 🡪 Graph;
15. getEdgeElem: Graph, Edge e 🡪 Element;
16. replaceEdgeElem: Graph, Edge e, Element x 🡪 Graph;

**Variables:**

g: Graph; v, w: Vertex; e: Edge; x, y: Element; n: **N**0

**Axioms:**

1. isEmpty(vertices(newGraph())) = true;
2. isEmpty(edges(newGraph())) = true;
3. countAllVertices(newGraph()) = 0;
4. countAllEdges(newGraph()) = 0;
5. countAllVertices(insertVertex(insertVertex(newGraph(), v), w)) = 2;
6. countAllVertices(insertVertex(insertVertex(newGraph(), v), v)) = 1;
7. countAllVertices(removeVertex(insertVertex(g, v), v)) = countAllVertices(g);
8. countAllEdges(removeEdge(insertEdge(g, v, w, x), v, w, x)) = countAllEdges(g);
9. if(not(getEdge(g, v, w) == undefined)) then areAdjacent(g, v, w) = true;
10. incidentEdges(g, w) = if((opposite(g, e1, v1) == w) AND (opposite(g, e2, v2) == w)) then {e1, e2};
11. opposite(g, e, v) = if(getEdge(g, v, w) == e) then w;
12. endVertices(g, e) = if(getEdge(g, v, w) == e) then {v, w};
13. getEdgeElem(g, e) = if(countAllEdges(insertEdge(g, v, w, x)) == countAllEdges(g) + 1) then x;
14. if(getEdgeElem(g, e) == x) then replaceEdgeElem(g, e, y) = if(getEdgeElem(g, e) == y) then g;

**ALGEBRAIC SPECIFICATION:**

Directed Graph ADT

**Spec:** Directed Graph(Element)

**Sort:** DirectedGraph

**Extend:** Graph(Element)

**Description:** A new description of *newDirectedGraph* returns a new empty DirectedGraph. There are also new operations. There is *InsertDirectedEdge* which adds an edge to a graph where the first defined vertex is the origin of the edge and the second defined vertex is the end point, *incomingEdgesOf* which returns a set of all the edges that are directed towards the defined vertex, *inDegreeOf* which returns the number of edges entering a defined vertex, *outDegreeOf* which returns the number of edges leaving a defined vertex, and, finally, *outgoingEdgesOf* which returns a set of all the edges that are directed away from the defined vertex.

**Signature of Operations:**

1. newDirectedGraph 🡪 DirectedGraph;
2. insertDirectedEdge: DirectedGraph, Vertex v, Vertex w, Element e 🡪 DirectedGraph;
3. incomingEdgesOf: DirectedGraph, Vertex v 🡪 Set;
4. inDegreeOf: DirectedGraph, Vertex v 🡪 **N**0;
5. outDegreeOf: DirectedGraph, Vertex v 🡪 **N**0;
6. outgoingEdgesOf: DirectedGraph, Vertex v 🡪 Set;

**Variables:**

g: DirectedGraph; v, w: Vertex; e: Edge; x, y: Element; n: **N**0

**Axioms:**

1. isEmpty(vertices(newDirectedGraph())) = true;
2. isEmpty(edges(newDirectedGraph())) = true;
3. countAllVertices(newDirectedGraph()) = 0;
4. countAllEdges(newDirectedGraph()) = 0;
5. incomingEdgesOf(g, v) = if((incidentEdges(g, v) == {e1, e2}) AND (outgoingEdgesOf(g, v) == {e2})) then {e1};
6. inDegreeOf(g,v) = if((incidentEdges(g,v) == {e1,e2}) AND (incomingEdgesOf(g, v) == {e1})) then 1;
7. outDegreeOf(g,v) =if((incidentEdges(g,v) == {e1,e2}) AND (outgoingEdgesOf(g,v) == {e2})) then 1;
8. outgoingEdgesOf(g, v) = if((incidentEdges(g, v) == {e1, e2}) AND (incomingEdgesOf(g, v) == {e1})) then {e2};
9. inDegreeOf(g, v) = if((incidentEdges(g, v) == {e1, e2}) AND (incomingEdgesOf(g, v) == {e1, e2})) then 2;
10. outDegreeOf(g, v) = if((incidentEdges(g, v) == {e1, e2}) AND (outgoingEdgesOf(g, v) == {e1, e2})) then 2;