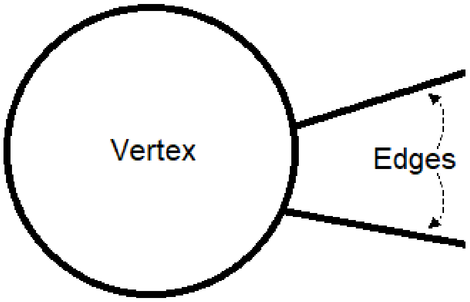
The ADT Graph

ADT Vertex

Representation:



Vertex = {value = <Object>, edgeList = <List>}

{inv: value ≠ NIL, edgeList.size ≥ 0 }

Primitive Operations:

createVertex	Value	➔	Vertex
addEdge	Vertex x Edge	➔	Vertex
removeEdge	Vertex x Edge	➔	Vertex
getValue	Vertex	➔	Value
getEdges	Vertex	➔	List
isAdjacent	Vertex x Vertex	➔	Boolean

createVertex(val)
“Creates a new Vertex, with its given value.”
{pre: TRUE}
{post: vertex={val, edgeList} }

addEdge(vert, edg)
“Connects this vertex to a new edge.”
{pre: vert ≠ NIL, edg ∈ Edge, (edg.vertex1 = NIL ∧ edg.vertex2 ≠ NIL) ∨ (edg.vertex1 ≠ NIL ∧ edg.vertex2 = NIL)}
{post: edg ∈ vert.edgeList}

removeEdge(vert, edg)
“Disconnects this vertex from an edge.”
{pre: vert ≠ NIL, edg ∈ vert.edgeList, edg.vertex1 = vert v edg.vertex2 = vert}
{post: edg ∉ vert.edgeList}

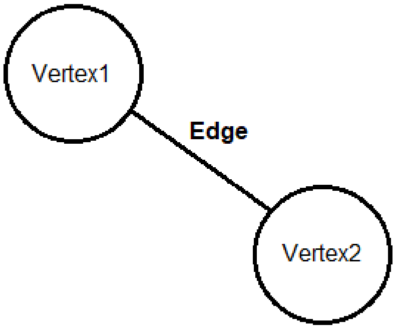
getValue (vert)
“Returns the value of this Vertex”
{pre: vert ≠ NIL}
{post: <value>}

getEdges (vert)
“Returns all of the edges this vertex is connected to.”
{pre: vert ≠ NIL}
{post: <edgeList>}

isAdjacent(vert1, vert2)
“Determines whether a pair of vertexes are adjacent or not.”
{pre: vert1 ≠ NIL, vert1.edgeList.size > 0, vert2 ≠ NIL, vert2.edgeList.size > 0}
{post: FALSE if (edg.vert1 = vert2 or edg.vert2 = vert2) and edg ∈ vert1.edgeList; TRUE otherwise}

ADT Edge

Representation:



Edge = {Vertex1 = <Vertex>, Vertex2 = <Vertex>, Weight = <Integer>, Directed = <Boolean>}

{inv: Vertex1 ≠ NIL, Vertex2 ≠ NIL, Weight ≥ 0 }

Primitive Operations:

createEdge	Vertex x Vertex x Integer x Boolean	→	Edge
isWeighted	Edge	→	Boolean
getWeight	Edge	→	Integer
isDirected	Edge	→	Boolean
getVertex1	Edge	→	Vertex
getVertex2	Edge	→	Vertex

createEdge(v1,v2, w, d)

"Creates a new Edge and connects two vertexes to it. Also determines its weight and if its either directed or not."

{pre: TRUE}

{post: edge={v1, v2, w, d}

isWeighted(ed)

"Determines whether an edge is weighted or not."

{pre: ed ≠ NIL}

{post: TRUE if ed.Weight >0; FALSE otherwise}

getWeight (ed)

"Determines the weight of this edge."

{pre: ed ≠ NIL}

{post: <Weight>}

isDirected(ed)

"Determines whether an edge is directed or not, in which case it'll be directed from ed.Vertex1 to ed.Vertex2"

{pre: ed ≠ NIL}

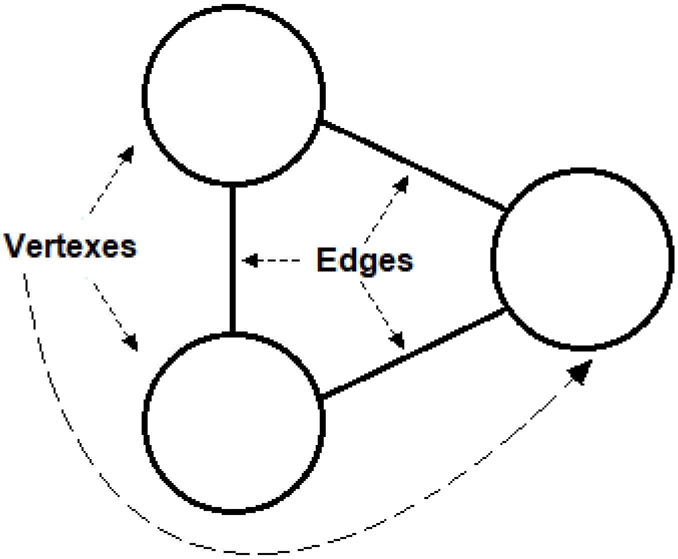
{post: <Directed>}

getVertex1(ed)
“Returns the first vertex this edge is connected to.”
{pre: ed ≠ NIL}
{post: <Vertex1>}

getVertex2(ed)
“Returns the second vertex this edge is connected to.”
{pre: ed ≠ NIL}
{post: <Vertex2>}

ADT Graph

Representation:



Graph = {V, E}, where V is a set of Vertexes and E is a set of Edges

{inv: V.size ≥ 0, E.size ≥ 0}

Primitive Operations:

createGraph		➔	Graph
isWeighted	Graph	➔	Boolean
isDirected	Graph	➔	Boolean
isRelated	Graph	➔	Boolean
addVertex	Graph x Vertex	➔	Graph
addEdge	Graph x Edge	➔	Graph
removeVertex	Graph x Vertex	➔	Graph
removeEdge	Graph x Edge	➔	Graph
getNumberOfEdges	Graph	➔	Integer
getNumberOfVertexes	Graph	➔	Integer
areConnected	Graph x Vertex x Vertex	➔	Boolean
getWeightMatrix	Graph	➔	$A = \{a_{ij}\}$
getDirectionMatrix	Graph	➔	$A = \{a_{ij}\}$

createGraph()

"Creates a new Graph and initializes its components."

{pre: TRUE}

{post: graph={V, E}, V = {}, E = {} }

isWeighted(gr)

"Determines whether a Graph is weighted or not."

{pre: TRUE}

{post: TRUE if at least one x∈gr.E is weighted; FALSE otherwise}

isDirected (gr)
“Determines whether a Graph is directed or not.”
{pre: TRUE}
{post: TRUE if at least one x∈gr.E is directed; FALSE otherwise}

isRelated (gr)
“Determines whether a Graph is related or not.”
{pre: TRUE}
{post: TRUE if there are paths to arrive from an arbitrary Vertex to every other vertex in the graph; FALSE if at least one Vertex is not reachable by any path from any arbitrary vertex.}

addVertex (gr, vert)
“Adds a new Vertex in the graph.”
{pre: TRUE}
{post: vert ∈ gr.V}

addEdge (gr, ed)
“Adds a new Edge in the Graph.”
{pre: TRUE}
{post: ed ∈ gr.E}

removeVertex(gr, vert)
“Removes a given vertex from the graph, provided it already exists in the Graph.”
{pre: vert ∈ gr.V}
{post: <vert> and gr.V reduces its size in one}

removeEdge (gr, ed)
“Removes a given edge from the graph, provided it already exists in the Graph.”
{pre: ed ∈ gr.E}
{post: <ed> and gr.E reduces its size in one}

getNumberOfEdges (gr)
“Retrieves the number of edges in this graph.”
{pre: TRUE}
{post: <gr.E.size> }

getNumberOfVertexes (gr)
“Retrieves the number of vertexes in this graph.”
{pre: TRUE}
{post: <gr.V.size>}

areConnected(gr, v1, v2)
“Determines whether a pair of vertexes are adjacent (connected by, at least, one edge) to each other or not”
{pre: gr.V.size ≥ 1, v1 ∈ gr.V, v2 ∈ gr.V}
{post: TRUE if there’s at least one e ∈ gr.E, (e.getV1 = v1 and e.getV2 = v2) or (e.getV1 = v2 and e.getV2 = v1); FALSE otherwise}

getWeightMatrix (gr)
“Returns the weight matrix of this graph.”
{pre: TRUE}
{post: A = [a_{ij}], where i and j are vertexes, and a_{ij} is the weight of the edge that connects them both, or ∞ if there is no such edge}

getDirectionMatrix (gr)
“Returns the direction matrix of this graph.”
{pre: TRUE}
{post: A = [a_{ij}], where i and j are vertexes, and a_{ij} is 1 if there is a edge that connects from vertex i to vertex j, or 0 otherwise}