TAD Graph

Graph = {ArrayList<Vertex<T>> = <vertices>, boolean= <directed>, int = <time>}

{inv: <∀ i, j ∈ Graph.Vertices, i ≠ j>
<∀ (i, j) ∈ Graph.Edges, i ∧ j ∈ Graph.Vertices>
<∀ (i, j) ∉ Graph.Edges, i ∧ j ∈ Graph.Vertices, i = j>
<∀ (i, j) ∈ Graph.Edges, (k, l) ∈ Graph.Edges, (i, j) ≠ (k, l)> }

Primitive Operations

- addVertex: <Vertex> + <Graph> → Graph
- addEdge: <Edge> + <Graph> → Graph
- removeVertex: <Vertex> + <Graph> → Graph
- removeEdge: <Edge> + <Graph> → Graph
- BFS: <Vertex> → Graph
- DFS: <Vertex> → Graph
- dijkstra: <Vertex> → Graph
- floydWarshall: <Vertex> → Graph
- isDirected: → Boolean
- getVertices: <Graph> → int

addVertex(T vertex)
"Adds a vertex to the network with the specified value"
{pre: len(Graph.Vertices) = n}
{post:len(Graph.Vertices) = n + 1}

addEdge(T source, T destination, int weight)
"Adds a directed or undirected edge with a weight between two existing vertices in the network"
{pre: len(Graph.Edges) = n}
{post:len(Graph.Edges) = n + 1}

removeVertex(T vertex)
"Removes a vertex from the network, along with all adjacent edges"
{pre: len(Graph.Vertices) = n}
{post:len(Graph.Vertices) = n - 1}

removeEdge(T source, T destination)
"Removes an edge between two existing vertices in the network"
{pre: len(Graph.Edges) = n}
{post:len(Graph.Edges) = n - 1}

BFS(T source)
"Performs a BFS path from the specified source vertex"
{pre: source ∈ Graph.Vertices}
{post: All vertices are visited in the correct order}

DFS(T source)
"Performs a DFS path from the specified source vertex"
{pre: source ∈ Graph.Vertices}
{post: All vertices are visited in the correct order}

dijkstra(T source)
"Applies Dijkstra algorithm to find the shortest paths from the specified source vertex to all other vertices of the network"
{pre: source ∈ Graph.Vertices}
{post:The shortest path from the source vertex to all other vertices in the graph has been found}

floydWarshall()
"Applies the Floyd-Warshall algorithm to find the shortest distances between all pairs of vertices in the graph"
{pre: Graph must be a weighted and connected network. It must not contain negative loops}
{post:The shortest distances between all pairs of vertices in the graph have been calculated}

isDirected()
"Returns True if the graph is directed, otherwise it returns false."
{pre: Graph.directed ≠ Null}
{post: True or false is obtained}

getVertices()
"Returns the array of vertices of the graph"
{pre: Graph.Vertices ≠ Null}
{post:The list of vertices is obtained}