TAD<Graph>

G = (V, E), where V is a set of vertices, and E is a set of edges

{inv: There cannot be two vertexes with the same value on the Graph.}

Primitive operations:

- Graph	constructor	Graph	-> Graph
- newVertex	modifier	Graph x Value	-> Graph
- deleteVertex	modifier	Graph x Value	-> Graph
- edge	modifier	Graph x Value1 x Value2	-> Graph
- edgeWeight	modifier	Graph x Value1 x Value2 x Weight	-> Graph
- deleteEdge	modifier	Graph x Value1 x Value2	-> Graph
- BFS	analyzer	Graph x Value1	-> Graph
- DFS	analyzer	Graph	-> Graph
- dijkstra	analyzer	Graph x Value1	-> Integer[]
- floydWarshall	analyzer	Graph	-> Integer[][]
- prim	analyzer	Graph x Value1	-> List<Edges>
- kruskal	analyzer	Graph	-> List<Edges>

Graph()
Creates a new Graph
{ pre: TRUE }
{ post: Graph is created}

newVertex(G, u)
Adds vertex u to the graph G.
{ pre: TRUE }
{ pos: The vertex is added to the graph G }

deleteVertex(G, u)
Removes vertex u from the graph G.
{ pre: u must belong to the set of vertices of the graph G }
{ pos: The vertex is removed from the graph G }

edge(G, u, v)
Adds the arc or edge (u,v) to the graph G.
{ pre: u and v must belong to the set of vertices of the graph }
{ pos: An edge connecting u with v is created in the graph G }

edgeWeight(G, u, v, w)
For a valued graph, adds the arc (u,v) to the network and the cost of the edge, w.
{ pre: u and v must belong to the set of vertices of the graph }
{ pos: An weighted edge connecting u with v is created }

deleteEdge(G, u, v)
Removes edge(u,v) from the graph G.
{ pre: There must be an edge between u and v }
{ pos: The edge is removed from the graph G}

BFS(G, u)
Find the shortest path from u to each reachable vertex of the graph G.
{ pre: Graph G must not have cycles }
{ pos: The shortest distance from u to each reachable vertex has been determined.

DFS(G)
Explore in depth by visiting all the neighbors of a vertex in the graph G.
{ pre: TRUE }
{ pos: The distance and time of discovery of each vertex in the graph G is determined. }

dijkstra(G, u)
Finds the shortest path from vertex u to all other vertices in a weighted graph.
{ pre: G must be a weighted graph and all edge weights in the graph G must be non-negative. }
{ pos: The shortest path from u to all other vertices in the graph G is found. }

floydWarshall(G)
Find all the shortest distances between all pairs of vertices in the graph G.
{ pre: G must be a weighted graph and all edge weights in the graph G must be non-negative. }
{ pos: Calculates all the shortest distances between all pairs of vertices in the graph G. }

prim(G, u)
Finds the minimum spanning tree in a weighted graph from vertex u.
{ pre: G must be a connected graph and must not contain edges with negative weights.}
{ pos: The minimum spanning tree of the network G is returned. }

kruskal(G)
Finds the minimum spanning tree in a weighted graph.
{ pre: G must be a connected graph and must not contain edges with negative weights. }
{ pos: The minimum spanning tree of the network G is returned. }