TAD<Graph>

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

{inv: $\forall(V_i , V_j) \rightarrow (i \neq j)$ = An already existing vertex can't be added.}

Operations:

•	Graph	constructor		→ Graph
•	insertVertex	modifier	Graph x key x value	→ Graph
•	deleteVertex	modifier	Graph x key	→ Graph
•	insertEdge	modifier	Graph x key x key x weight	→ Graph
•	deleteEdge	modifier	Graph x key x key	→ Graph
•	adjacent	analyzer	Graph x key x key	→ boolean
•	bfs	analyzer	Graph x key	→ Graph
•	dfs	analyzer	Graph	→ Graph
•	dijkstra	analyzer	Graph x key x key	→ int[]
•	floydWarshall	analyzer	Graph	→ int[][]
•	prim	analyzer	Graph	→ ArrayList<Integer>
•	kruskal	analyzer	Graph x ArrayList<Integer> x ArrayList<Integer>	→ Arraylist<Integer>

Graph()
"Creates a new graph"
{pre: TRUE}
{pos: Creates a graph}

insertVertex(G, k, value)
"Adds a new vertex in the graph *G*"
{pre: G = {} ∧ the new vertex must not belong to the vertex set}
{post: The vertex has been added to the graph *G*}

deleteVertex(G, k)
"Deletes a vertex with the specified key of the graph *G*"
{pre: *k* must be a key of a vertex in the set of vertices of the graph *G*}
{pos: The vertex is removed from the graph *G*}

insertEdge(G, k1, k2, weight)
"Adds an edge between the vertexes with keys k1 and k2 with the specified weight to the graph *G*"
{pre: k1 and k2 keys must belong to vertexes in the set of vertexes of the graph *G*}
{pos: A weighted edge connecting the vertexes with keys *k1* and *k2* has been created in the graph G}

deleteEdge(G, k1, k2)
"Deletes the edge between the vertexes with keys *k1* and *k2* of the graph *G*"
{pre: There must be an edge between the vertexes with keys *k1* and *k2*}
{pos: The edge is removed from the graph G}

adjacent(G, k1, k2)
"Returns true if vertexes with keys *k1* and *k2* form an edge"
{pre: There must be an edge between the vertexes with keys *k1* and *k2*}
{pos: true if vertexes with keys *k1* and *k2* form an edge. False otherwise}

bfs(G, k)
"Explores the graph G starting on the vertex with key *k* and carries on with all its neighbors"
{pre: k1 must belong to a vertex in the set of vertexes of the graph *G*}
{post: All nodes reachable from the source vertex}

dfs(G)
"Explores all the graph *G* starting in the first vertex"
{pre: TRUE}
{post: All vertexes visited during the DFS traversal}

dijkstra(G, k1, k2)
"Returns the shortest path between the vertexes with keys *k1* and *k2*"
{pre: *G* must be an undirected or directed weighted graph without negative cycles}
{post: Shorter distances from one source node to all are returned}

floydWarshall(G)
"Returns the shortest path between all the pair of vertexes"
{pre: *G* must be a weighted graph without negative cycles.}
{post: All shortest distances between all pairs of nodes are returned}

prim(G)
"Creates a minimal spanning tree from an initial node."
{pre: G must be undirected and connected graph with non-negative edge weights}
{post: get a minimum spanning tree connecting all vertexes of the graph G}

kruskal(G)
"Creates a minimal spanning tree with no cycles and minimal weight."
{pre: G must be undirected and connected graph with non-negative edge weights}
{post: get a minimum spanning tree of the graph G that connects vertices without cycles and with the minimum weight}