| Graph ADT  Graph = { Vertex= <set of vertices/nodes>, Edges = <set of edges>, Directed = <True | False. Indicates whether the graph is directed or not>, Weighted = <True | False. Indicates whether the graph's edges will have weights between them> }  **Simple Graph**  A simple graph consists of V, a non-empty set of vertices, and E, a set of unordered pairs of distinct elements from V. These pairs are called edges.  **Multigraph**  A multigraph consists of a set V of vertices, a set E of edges, and a function f from E to . The multigraph allows multiple connections between the vertices. Edges and are called multiple or parallel edges if .  **Pseudograph**  Un pseudografo consta de un conjunto V de vértices, un conjunto E de aristas y una función f de E en . El pseudografo permite relaciones de tipo bucle.Una arista e es un bucle, o lazo, si para algún u ∈ V.  A pseudograph consists of a set V of vertices, a set E of edges, and a function f from E to . The pseudograph allows loop-type relationships. An edge e is a loop if for some u ∈ V.  **Directed Graph**  A directed graph consists of a set V of vertices and a set E of edges, which are ordered pairs of elements from V. The edges have a direction and are traversed only in the specified direction.  **Directed Multigraph**  A directed multigraph consists of a set V of vertices, a set E of edges, and a function f from E to . Edges and are called multiple or parallel edges if . In addition to having multiple edges, it has direction, so they can only be traversed in the specified direction. |
| --- |
| **Invariants**  { inv: Graph.Nodes is a set of unique elements, meaning there cannot be vertices with the same identifier (duplicate vertices). Graph.Edges is a set of ordered pairs of nodes, and for each edge (startVertex, endVertex) in Graph.Edges, both startVertex and endVertex are in Graph.Nodes. The size of the Graph must be greater than 0 (it can be initialized without vertices, but it should have space to store at least one vertex). Additionally, its adjacency matrix must be square, and its adjacency list must contain all its vertices. }  **Simple Graph**  No Loops: A simple graph does not contain loops, which means there is no edge that connects a vertex to itself.  No Parallel Edges: In a simple graph, there cannot be multiple edges connecting the same pair of vertices or nodes.  **Multigraph**  A multigraph allows multiple relationships, where and are multiple or parallel edges if:    **Pseudograph**  For an edge e to be a loop, it must satisfy for some u ∈ V.  **Directed graph**  Being a directed graph, edges have a specific direction, meaning the edges are oriented and go from a starting node to an ending node. This direction must be respected, and it's not possible to go from the ending node to the starting node.  **Directed Multigraph**  The edges y are multiple or parallel edges if , and the direction of the edge must be respected. |
| Primitive Operations:  CreateGraph: Size → Graph  ▪ AddVertex: Graph x Node → Graph  ▪ RemoveVertex: Graph x Node → Graph  ▪ AddEdge: Graph x Node x Node → Graph  ▪ RemoveEdge: Graph x Node x Node → Graph  ▪ HasEdge: Graph x Node x Node → Boolean  ▪ GetVertexes: Graph → Set of Vertexes  ▪ GetEdges: Graph → Set of Edges |

| CreateGraph(vertexAmount) "Creates a new hash table with the specified size" |
| --- |
| { pre: {tamaño > 0, createGraph = true} } |
| { post: Graph.Size = size, Graph.Vertexes is >= 0 } |

| AddVertex(K key, V value) "Adds a new vertex in the graph, which contains a key (index) and data to store (value)" |
| --- |
| { pre: {Graph = graph, key ≠ null, value ≠ null, key.alreadyExists = false} } |
| { post: Graph.Vertexes = totalVertexes+1, Graph.Vertexes contains the new vertex } |

| RemoveNode(K key) "Removes a vertex from the graph" |
| --- |
| { pre: {Graph = graph, vertex.exists ≠ null} } |
| { post: Graph.Vertexes = totalVertexes-1, Graph.Edges = totalEdges-1, Graph.Vertexes no longer contains the removed node, and any edge involving the removed vertex is removed from Graph.Edges} |

| HasEdge(startVertex, endVertex) "Checks if there is an edge between two vertex in the graph" |
| --- |
| { pre: {Graph = graph, startVertex ≠ null, endVertex ≠ null, startVertex and endVertex are in Graph.Nodes} } |
| { post: Returns true if there is an edge between startVertex and endVertex in Graph.Edges; otherwise, returns false } |

| BFS(graph, startingVertex) "Performs a Breadth-First Search on the graph starting from the specified vertex" |
| --- |
| { pre: {BFS.Graph = graph, BFS.startingVertex ≠ null } } |
| { post: Returns a list of vertices visited during the BFS traversal from the starting vertex. This could be a string that represents the BFS or level traversal order path followed during the algorithm } |

| ExecuteDFS(graph, startingVertex) "Performs a Depth-First Search on the graph starting from the specified vertex" |
| --- |
| { pre: {DFS.Graph = graph, DFS.startingVertex ≠ null } } |
| { post: Returns a list of vertices visited during the DFS traversal from the start nod vertex. This could be a string that represents the DFS or depth traversal order path followed during the algorithm } |