|  |
| --- |
| **Graph<T> ADT** |
| Graph is an ordered pair G = (V, E) comprising:   * Problema del camino más corto - Wikipedia, la enciclopedia libreV, a set of vertices. * , a set of edges, which are unordered pairs of vertices. |
| {Inv: } |

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | Primitive Operations:  Graph  addVertex  isInGraph  searchVertex  deleteVertex  addEdge  addEdge  deleteEdge  areAdjacent  getAdjVertices  getEdgeWeight  setEdgeWeight  bfs  dfs  prim  kruskal  dijkstra  floydWarshall  getVertices  getTotalVertices  getTotalEdges  isDirected  isWeighted | Boolean x Boolean  Graph x Vertex  Graph x T  Graph x T  Graph x Vertex  Graph x Vertex x Vertex x Double  Graph x Vertex x Vertex  Graph x Vertex x Vertex  Graph x Vertex x Vertex  Graph x Vertex  Graph x Vertex x Vertex  Graph x Vertex x Vertex x Double  Graph x Vertex  Graph  Graph x Vertex  Graph  Graph x Vertex  Graph  Graph  Graph  Graph  Graph  Graph | → Graph  → Graph  → Boolean  → Vertex  → Graph  → Graph  → Graph  → Graph  → Boolean  → List<Vertex>  → Double  → Graph  → Graph  → Graph  → Graph  → List<Edge>  → Graph  → Double[][]  → List<Vertex>  → Integer  → Integer  → Boolean  → Boolean | |

|  |
| --- |
| **Graph(directed, weighted)**  “Builds a new graph which is either directed or not and weighted or not”.  {pre:}  {post: Graph = {V = {null}, E = {null}, directed, weighted} |
| **addVertex(v)**  “Adds a new vertex to the graph”.  {pre: v g.V}  {post: v g.V} |
| **addEdge(x, y)**  “Adds an edge with weight 1 from x to y and from y to x if the graph is not directed”.  {pre: x, y g.V}  {post: e = (x, y, 1) g.E. If g.directed = false, e’ = (y, x, 1) ) g.E} |
| **addEdge(x, y, w)**  “Adds an edge with weight w from x to y and from y to x if the graph is not directed”.  {pre: x, y g.V, g.weighted = true, w > 0}  {post: e = (x, y, w) g.E. If g.directed = false, e’ = (y, x, w) ) g.E} |
| **deleteVertex(v)**  “Deletes the vertex v from the graph”.  {Pre: v g.V}  {Post: v g.V. For every incident vertex to v g.E} |
| **deleteEdge (x, y)**  “Deletes the edge from x to y from the graph”.  {Pre: x,y g.V, (x, y, \*) g.E}  {Post: e = (x, y, \*) g.E. If g.directed = false, e’ = (y, x, \*) ) g.E} |
| **getAdjVertices(x)**  “Returns a list of adjacent vertices to x. There is an edge from x to every vertex v from the list”.  {Pre: x g.V}  {Post: vertices = : ∀, (x, , \*) g.E} |
| **getTotalVertices()**  “Returns the total number of vertices in the graph”.  {Pre:}  {Post: n = size(g.V)} |
| **getTotalEdges()**  “Returns the total number of edges in the graph”.  {Pre:}  {Post: n = size(g.E)} |
| **areAdjacent(x, y)**  “Returns true if there is an edge from x to y. Returns false if there is not.”  {Pre: x, y g.V}  {Post: true if and only if (x, y, \*) g.E} |
| **isInGraph(t)**  “Returns true if there is a vertex with value t in the graph. Returns false if there is not.”  {Pre:}  {Post: true if and only if x g.V: value(x) = t} |
| **getEdgeWeight(x, y)**  “Returns the weight of the edge from x to y”.  {Pre: x, y g.V, (x, y, \*) g.E}  {Post: weight = (x, y).w} |
| **setEdgeWeight(x, y, w)**  “Sets a weight w from x to y”.  {Pre: x, y g.V, (x, y, \*) g.E, w > 0}  {Post: (x, y, w) g.E} |
| **isDirected()**  “Returns true if the graph is directed. Returns false if it’s not.”  {Pre:}  {Post: g.directed} |
| **isWeighted()**  “Returns true if the graph is weighted. Returns false if it’s not.”  Pre:  Post: g.weighted |
| **bfs(x)**  “Performs the Breadth First Search algorithm”.  {Pre: x g.V}  {Post: ∀u g.V, added attributes u.pred y u.d} |
| **dfs()**  “Performs the Depth First Search algorithm”.  {Pre:}  {Post: ∀u g.V, added attributes u.pred, u.d y u.f} |
| **dijkstra(x)**  “Perform the Dijkstra algorithm.”  {Pre: x g.V, g there is no negative weight}  Post: ∀u g.V, added attributes u.pred y u.d} |
| **floydwarshall()**  “Performs the Floyd-Warshall algorithm”.  {Pre:}  {Post: returns the matrix of weights m where m[i,j] is the minimum distance from vi to vj} |
| **prim(x)**  “Performs the Prim algorithm”.  {Pre: x g.V, g is not directed}  {Post: ∀u g.V, added attributes u.pred y u.d} |
| **kruskal(x)**  “Performs the Kruskal algorithm on a graph g”.  {Pre:}  {Post: , where g.E are edges which form a minimum spanning tree.} |