**ADT Design**

* Daniel José Plazas Cortés - A00400085
* Sebastian Erazo Ochoa - A00400086
* Juan Manuel Casanova Marín - A00400090

|  |  |
| --- | --- |
| **Direct Graph ADT** | **No Direct Graph ADT** |
| \*E: set of edges, V: set of vertexes\*. Graph<V,E> = {<>, <>,isWeighted} | |
| Invariant: , | Invariant: |
| **Primitive Operations:**  CreateGraph: isWeighted -> Graph  AddVertex : Graph x Vertex -> Graph  AddWeightedEdge : Graph x VertexFrom x VertexTo x Weight -> Graph  AddEdge : Graph x VertexFrom x VertexTo -> Graph  DeleteVertex : Graph x Vertex -> Graph  DeleteEdge : Graph x VertexFrom x VertexTo -> Graph  BFS : Graph -> NaryTree  DFS : Graph -> NaryTree[]  Dijkstra : Graph x Vertex -> Double[],Vertex[]  Floyd-Warshall : Graph -> Double[][]  Prim : Graph -> NaryTree<Vertex>  Kruskal : Graph -> Edge[]  Size : Vertexes -> Integer  Size : Edges -> Integer | |

|  |
| --- |
| CreateGraph : isWeighted -> Graph |
| \*Creates an empty graph weighted or not weighted (isWeighted). The param isWeighted is a boolean.\* |
| {pre:TRUE} |
| {post: Graph={<>,<>,isWeighted}} |

|  |
| --- |
| AddVertex: Graph x Vertex> Graph |
| \*Adds a vertex to the graph\* |
| {pre: Graph={<>,<>,true/false} Graph!={<>,<>,true/false}, for example Graph={<>,<>,true/false}} |
| {post: Graph={<Vertex>,<>,isWeighted} Graph{<v1,v2,Vertex>,<>,true/false}} |

|  |
| --- |
| AddEdge: Graph x VertexFrom x VertexTo -> Graph |
| \*Creates and adds an edge to the graph.\* |
| {pre: isWeighted=FALSE Graph!={<>,<>,FALSE}, for example Graph={<>,<>,FALSE} VertexFromV VertexToV} |
| {post: Graph={<>,<>,TRUE}},={VertexFrom,VertexTo} |

|  |
| --- |
| AddWeightedEdge: Graph x VertexFrom x VertexTo x Weight -> Graph |
| \*Creates and adds an weighted edge to the graph.\* |
| {pre: isWeighted =TRUE Graph!={<><>,TRUE}, for example Graph={<>,<>,TRUE} VertexFromV VertexToV} |
| {post: Graph={<>,<>,TRUE}}, ={VertexFrom,VertexTo,Weight} |

|  |
| --- |
| DeleteVertex : Graph x Vertex -> Graph |
| \*Deletes a vertex, and if an edge exists with that vertex, deletes it as well.\* |
| {pre: [Graph={<>,<>,TRUE/FALSE} Graph={<>,<>,TRUE/FALSE}] VertexV, for example Vertex= , for example } |
| {post: Graph={<>,<>,TRUE/FALSE} Graph={<>,<>}} |

|  |
| --- |
| DeleteEdge: Graph x VertexFrom x VertexTo -> Graph |
| \*Deletes an edge of the graph.\* |
| {pre: Graph={<>,<>,TRUE/FALSE} , for example } |
| {post: Graph={<>,<>,TRUE/FALSE}} |

|  |
| --- |
| BFS : Graph x Vertex -> NaryTree |
| \*Traverse all the graph and find the way that all the vertexes, since Vertex, are connected. Returns a tree of this connections without a cycle.\* |
| {pre: isWeighted=FALSE } |
| {post: Returns a n-ary tree with all the nodes that are reachable since Vertex in the graph.} |

|  |
| --- |
| DFS: Graph -> NaryTree[] |
| \*Traverse all the graph and find the ways that follow all the vertexes. If all vertexes are not connected (the graph are not related), then the algorithm returns a forest (a set of Nary Tree), otherwise returns a tree (as BFS case).\* |
| {pre: isWeighted=FALSE } |
| {post: Returns n n-ary trees with all the nodes and their ways.} |

|  |
| --- |
| Dijkstra : Graph x Vertex -> Integer[],Vertex[] |
| \*This does the same as BFS. However, this returns one more attribute of distances. This method calculates the minimum distances that can be reachable from Vertex to all the graph connected. If all the vertexes are not related, the method will not include this nodes.\* |
| {pre: isWeighted=TRUE } |
| {post: distances=Integer[], predecessors=vertexes(Vertex[])} |

|  |
| --- |
| Floyd-Warshall: Graph -> Double[][] |
| \*This method is so like DFS method. This one calculates the minimum distances between all the vertexes in the graph, no matter if this is connected or not (there’s “forest” or not). In contrast, this one accepts negative weights, DFS not.\* |
| {pre: isWeighted=TRUE} |
| {post: Matrix of distances} |

|  |
| --- |
| Prim: Graph x Vertex -> NaryTree |
| \*Calculates the way that all the vertexes relate to the minimum of weight and edges. At the end, this method creates a n-ary tree of these vertexes without a cycle.\* |
| {pre: isWeighted=TRUE } |
| {post: Returns a nary tree of vertexes.} |

|  |
| --- |
| Kruskal: Graph -> Edge[] |
| \*This method calculates the same as Prim, but uses an union-find to do it.\* |
| {pre: isWeighted=TRUE} |
| {post: Returns an array of edges.} |

|  |
| --- |
| Size: Vertexes -> Integer |
| \*Returns the number of vertexes in the graph.\* |
| {pre:TRUE, for example Graph={<>,<>,TRUE/FALSE}} |
| {post: 3} |

|  |
| --- |
| Size: Edges -> Graph |
| \*Returns the number of vertexes in the graph.\* |
| {pre:TRUE, for example Graph={<>,<>,TRUE/FALSE}} |
| {post: 4} |