|  |
| --- |
| TAD <Graph > |
| Graph={ArrayList, Matriz}  where arraylist is the list and the Matrix is the adjacency matriz |
| Inv: ∀v, v ≠ null ∧ |V| > 0  Inv: {vertex diferent from null and vertices>0} |
| Primitive Operations:  AddVertex(vertex): -> Void  FindVertex(data): -> Vertex  RemoveVertex(vertex): -> Void  AddEdge(source, destination, weight): -> Void  RemoveEdge(source, destination): -> Void  DFS(startVertex): -> Void  BFS(startVertex): -> Void Floyd(): ->double[][]  Dijkstra(startVertex, endVertex): -> ArrayList<Vertex<T>>  FloydWarshall(): -> int[][]  PrimAL(startVertex): -> AdjacencyListGraph<T>  KruskalAL(): -> AdjacencyListGraph<T>  RemoveAllEdges(): -> Void  GetVertices(): -> ArrayList<Vertex<T>> |

**AddVertex**

|  |
| --- |
| AddVertex(vertex): -> Void  {pre: vertex}  {pos: vertices.contains(vertex)}  “Creates a new vertex and adds it to the graph” |

**FindVertex**

|  |
| --- |
| FindVertex(data): -> Vertex  {pre: data}  {pos: returns the vertex with the specified data}  “Finds a vertex by its data” |

**RemoveVertex**

|  |
| --- |
| RemoveVertex(vertex): -> Void  {pre: vertices.contains(vertex)}  {pos: !vertices.contains(vertex)}  “Removes a vertex from the graph” |

**AddEdge**

|  |
| --- |
| AddEdge(source, destination, weight): -> Void  {pre: source, destination, weight}  {pos: Edge between source and destination with the specified weight is added}  “Adds an edge between two vertices with a given weight” |

**RemoveEdge**

|  |
| --- |
| RemoveEdge(source, destination): -> Void  {pre: source, destination}  {pos: Edge between source and destination is removed}  “Removes an edge between two vertices” |

**dfs**

|  |
| --- |
| DFS(startVertex): -> Void  {pre: startVertex}  {pos: Depth-First Search is performed starting from the specified vertex}  “Performs Depth-First Search starting from a given vertex” |

**bfs**

|  |
| --- |
| BFS(startVertex): -> Void  {pre: startVertex}  {pos: Breadth-First Search is performed starting from the specified vertex}  “Performs Breadth-First Search starting from a given vertex” |

**Dijkstra**

|  |
| --- |
| Dijkstra(startVertex, endVertex): -> ArrayList<Vertex<T>>  {pre: startVertex, endVertex}  {pos: Returns the shortest path between startVertex and endVertex}  “Applies Dijkstra's algorithm to find the shortest path between two vertices” |

**FloydWarshall**

|  |
| --- |
| FloydWarshall(): -> int[][]  {pre: true}  {pos: Returns the matrix of all-pair shortest paths}  “Applies Floyd-Warshall algorithm to find all-pair shortest paths” |

**Prim**

|  |
| --- |
| PrimAL(startVertex): -> AdjacencyListGraph<T>  {pre: startVertex}  {pos: Returns a graph representing the minimum spanning tree}  “Applies Prim's algorithm to generate a minimum spanning tree using an adjacency list representation” |

**Kruskal**

|  |
| --- |
| KruskalAL(): -> AdjacencyListGraph<T>  {pre: true}  {pos: Returns a graph representing the minimum spanning tree}  “Applies Kruskal's algorithm to generate a minimum spanning tree using an adjacency list representation” |

**RemoveAllEdges**

|  |
| --- |
| RemoveAllEdges(): -> Void  {pre: true}  {pos: All edges are removed from the graph}  “Removes all edges from the graph” |

**GetVertices**

|  |
| --- |
| GetVertices(): -> ArrayList<Vertex<T>>  {pre: true}  {pos: Returns the list of vertices in the graph}  “ Gets the list of vertices in the graph” |