**TAD TABLES**

**By:** **Juan Felipe Jojoa Crespo A00382042**

**Felipe Rojas Prado A00383819**

**JOSEPH VERDESOTO VELEZ A00395664**

| TAD <Graph> |
| --- |
|  |
| Graph = {vertices, edges}  Invariant: Each vertex in the graph has a unique identifier. |
| create\_graph() -> Graph  add\_vertex(vertex) -> None  remove\_vertex(vertex) -> None  add\_edge(vertex\_start, vertex\_end) -> None  remove\_edge(vertex\_start, vertex\_end) -> None  get\_vertices() -> List  get\_edges() -> List  are\_adjacent(vertex1, vertex2) -> boolean  get\_adjacent\_vertices(vertex) -> List  get\_vertex\_degree(vertex) -> int  is\_connected() -> boolean  is\_cyclic() -> boolean  is\_complete() -> boolean  is\_tree() -> boolean  get\_shortest\_path(vertex\_start, vertex\_end) -> List  get\_minimum\_spanning\_tree() -> Graph  get\_connected\_components() -> List |

| create\_graph() -> Graph (Constructor): |
| --- |
| Creates a new empty graph. |
| Precondition: None. |
| postcondition: Returns a new empty graph. |

| add\_vertex(vertex) -> None (Modifier): |
| --- |
| Adds a vertex to the graph. |
| precondition: The vertex does not exist in the graph. |
| postcondition: Adds the vertex to the graph. |

| remove\_vertex(vertex) -> None (Modifier): |
| --- |
| Removes a vertex from the graph along with all its associated edges. |
| Precondition: The vertex exists in the graph. |
| postcondition: Removes the vertex and any associated edges from the graph. |

| add\_edge(vertex\_start, vertex\_end) -> None (Modifier): |
| --- |
| Adds an edge between two existing vertices in the graph. |
| precondition: The start and end vertices exist in the graph. |
| postcondition: Adds an edge between the start and end vertices. |

| remove\_edge(vertex\_start, vertex\_end) -> None (Modifier): |
| --- |
| Removes an edge between two existing vertices in the graph. |
| precondition: There is an edge between the start and end vertices. |
| postcondition: Removes the edge between the start and end vertices. |

| get\_vertices() -> List (Analyzer): |
| --- |
| Returns a list of all vertices present in the graph. |
| Precondition: None. |
| postcondition: Returns a list of all vertices present in the graph. |

| get\_edges() -> List (Analyzer): |
| --- |
| Returns a list of all edges present in the graph. |
| Precondition: None. |
| postcondition: Returns a list of all edges present in the graph. |

| are\_adjacent(vertex1, vertex2) -> boolean (Analyzer): |
| --- |
| Checks if two vertices are adjacent, i.e., if there is an edge connecting them. |
| precondition: The vertices exist in the graph. |
| postcondition: Returns True if the vertices are adjacent (there is an edge between them), otherwise returns False. |

| get\_adjacent\_vertices(vertex) -> List (Analyzer): |
| --- |
| Returns a list of vertices adjacent to a given vertex. |
| precondition: The vertex exists in the graph. |
| postcondition: Returns a list of vertices adjacent to the given vertex in the graph. |

| get\_vertex\_degree(vertex) -> int (Analyzer): |
| --- |
| Returns the degree of a vertex, i.e., the number of edges incident on it. |
| precondition: The vertex exists in the graph. |
| postcondition: Returns the degree of the vertex, i.e., the number of incident edges on it. |

| is\_connected() -> boolean (Analyzer): |
| --- |
| Checks if the graph is connected, i.e., if there is a path between every pair of vertices. |
| Precondition: None. |
| postcondition: Returns True if the graph is connected (there is a path between every pair of vertices), otherwise returns False. |

| is\_cyclic() -> boolean (Analyzer): |
| --- |
| Checks if the graph contains at least one cycle, i.e., a sequence of edges forming a loop. |
| Precondition: None. |
| postcondition: Returns True if the graph contains at least one cycle, otherwise returns False. |

| is\_complete() -> bool (Analyzer): |
| --- |
| Checks if the graph is complete, i.e., if every pair of vertices is connected by an edge. |
| Precondition: None. |
| postcondition: Returns True if the graph is complete (all pairs of vertices are connected by an edge), otherwise returns False. |

| is\_tree() -> bool (Analyzer): |
| --- |
| Checks if the graph is a tree, i.e., a connected graph without cycles. |
| Precondition: None. |
| postcondition: Returns True if the graph is a tree (a connected graph without cycles), otherwise returns False. |

| get\_shortest\_path(vertex\_start, vertex\_end) -> List (Analyzer): |
| --- |
| Returns the shortest path between two vertices using Dijkstra's algorithm. |
| precondition: The start and end vertices exist in the graph. |
| postcondition: Returns a list of vertices that form the shortest path between the start and end vertices in the graph, using Dijkstra's algorithm. |

| get\_minimum\_spanning\_tree() -> Graph (Analyzer): |
| --- |
| Returns a minimum spanning tree of the graph using Prim's or Kruskal's algorithm. |
| Precondition: None. |
| postcondition: Returns a new graph that represents the minimum spanning tree of the original graph, using Prim's or Kruskal's algorithm. |

| get\_connected\_components() -> List (Analyzer): |
| --- |
| Returns a list of sets, where each set represents a connected component of the graph. |
| Precondition: None. |
| postcondition: Returns a list of sets, where each set represents a connected component of the graph. |