

Graph																																			
<p>Graph = {</p> <p>Vertexes = {Vertex_1 = <key, value, Adjacent = {Vertex_1 ... Vertex_n} > ... Vertex_n = <key, value,</p> <p>Adjacent = {Vertex_1 ... Vertex_n} > ,</p> <p>Edges = {Edge_1 = <weight> ... Edge_n = <weight>}</p>																																			
<p>{inv: If the graph is not directed, the sum of the degrees of all vertices in the graph is equal to twice the number of edges in the graph, moreover, at least one vertex must exist}.</p>																																			
<p>Primitive Operations:</p> <table> <tr> <td>CreateGraph</td><td></td><td>→ Graph</td></tr> <tr> <td>AddVertex</td><td>Graph X Vertex</td><td>→ Graph</td></tr> <tr> <td>AddEdge</td><td>Graph X Vertex X Vertex</td><td>→ Graph</td></tr> <tr> <td>RemoveVertex</td><td>Graph X Vertex</td><td>→ Graph</td></tr> <tr> <td>RemoveEdge</td><td>Graph X Vertex X Vertex</td><td>→ Graph</td></tr> <tr> <td>BFS</td><td>Graph</td><td>→ Graph</td></tr> <tr> <td>DFS</td><td>Graph</td><td>→ Graph</td></tr> <tr> <td>Dijkstra</td><td>Graph X Vertex</td><td>→ Vertexes = {..}</td></tr> <tr> <td>Floyd-Warshal</td><td>Graph</td><td>→ Matrix = { ... }</td></tr> <tr> <td>Kruskal</td><td>Graph</td><td>→ Graph</td></tr> <tr> <td>Prim</td><td>Graph</td><td>→ Graph</td></tr> </table>			CreateGraph		→ Graph	AddVertex	Graph X Vertex	→ Graph	AddEdge	Graph X Vertex X Vertex	→ Graph	RemoveVertex	Graph X Vertex	→ Graph	RemoveEdge	Graph X Vertex X Vertex	→ Graph	BFS	Graph	→ Graph	DFS	Graph	→ Graph	Dijkstra	Graph X Vertex	→ Vertexes = {..}	Floyd-Warshal	Graph	→ Matrix = { ... }	Kruskal	Graph	→ Graph	Prim	Graph	→ Graph
CreateGraph		→ Graph																																	
AddVertex	Graph X Vertex	→ Graph																																	
AddEdge	Graph X Vertex X Vertex	→ Graph																																	
RemoveVertex	Graph X Vertex	→ Graph																																	
RemoveEdge	Graph X Vertex X Vertex	→ Graph																																	
BFS	Graph	→ Graph																																	
DFS	Graph	→ Graph																																	
Dijkstra	Graph X Vertex	→ Vertexes = {..}																																	
Floyd-Warshal	Graph	→ Matrix = { ... }																																	
Kruskal	Graph	→ Graph																																	
Prim	Graph	→ Graph																																	

Construction operations

CreateGraph()

“Create a new graph.”

{ **pre:** TRUE }

{ **post:** Graph $\neq \{\emptyset\}$ }

Modifying operations

AddVertex(Vertex)

“Add a new vertex to the graph.”

{ **pre:** Graph = $\{\dots\}$ \wedge Vertex = $\{\dots\}$ }

{ **post:** Vertex = $\{\dots\} \in$ Graph = $\{\dots\}$ }

AddEdge(Vertex1, Vertex2)

“Add an edge between two vertices in the graph.”

{ **pre:** Graph = $\{\dots\}$ \wedge Vertex1 = $\{\dots\} \in$ Graph \wedge Vertex2 = $\{\dots\} \in$ Graph }

{ **post:** The vertices are connected, the edge is created and vertices are added to the adjacency list of the vertices \wedge Edge = $\{\dots\} \in$ Graph = $\{\dots\}$ }

RemoveVertex(Vertex)

“Remove a vertex and its incident edges from the graph.”

{ **pre:** Graph = $\{\dots\}$ \wedge Vertex = $\{\dots\} \in$ Graph }

{ **post:** Vertex = $\{\dots\} \notin$ Graph = $\{\dots\}$ \wedge the incident edges are removed }

RemoveEdge(Vertex1, Vertex2)

“Remove the edge between two vertices in the graph.”

{ **pre:** $\text{Graph} = \{\dots\} \wedge \text{Vertex1} = \{\dots\} \in \text{Graph} \wedge \text{Vertex2} = \{\dots\} \in \text{Graph} = \{\dots\} \}$

{ **post:** $\text{Edge} = \{\dots\} \notin \text{Graph} = \{\dots\} \wedge \text{the vertices are removed from the adjacency list of the vertices} \}$

Analysing operations

BFS(Vertex)

"Perform Breadth-First Search in the graph, starting from the given Vertex, explores the vertices in successive levels, beginning from the provided vertex. It visits all neighboring vertices before advancing to neighbors of neighbors, ensuring that the closer vertices to the starting vertex are explored first. The result is a graph containing all vertices reachable from the starting vertex in the order they were discovered."

{**pre:** $\text{Graph} \neq \{\emptyset\} \wedge \text{Vertex} = \{\dots\} \in \text{Graph} = \{\dots\} \}$

{**post: return** $\text{Graph} = \{\dots\} \}$

DFS(Vertex)

“Perform Depth-First Search in the graph, starting from the given Vertex, explores a graph starting from the given Vertex. It traverses as deeply as possible along each branch before backtracking. This exploration might not guarantee the closest vertices are visited first. The result is a graph containing all reachable vertices from the starting Vertex.

{**pre:** $\text{Graph} \neq \{\emptyset\} \wedge \text{Vertex} = \{\dots\} \in \text{Graph} = \{\dots\} \}$

{**post: return** $\text{Graph} = \{\dots\} \}$

Dijkstra(Vertex)

"Performs the Dijkstra's algorithm on the graph to find the shortest path from the given source vertex to all other vertices in the graph with positive weights. Creates a data structure to store the unvisited vertices and another for the visited vertices, starts from the source vertex, and explores

the adjacent vertices, updating the minimum distances from the source vertex to other vertices as shorter paths are found. The algorithm repeats until all vertices have been visited or until the shortest path to all vertices has been found."

{**pre:** Graph $\neq \{\emptyset\} \wedge \text{Vertex} = \{\dots\} \in \text{Graph} = \{\dots\} \wedge \forall e \in \text{Edges}, \text{Weight}(e) \geq 0$ }

{**post:** For each vertex, modifies the weight between the vertex and the vertex of origin, se calcula y actualiza las distancias mínimas desde el vértice de origen a todos los demás vértices en el grafo}

Floyd-Warshall()

"Performs the Floyd-Warshall algorithm on the graph to find the shortest distances between all pairs of vertices. Finds the shortest distances between all pairs of vertices in the graph, even if the graph contains negative cycles. Updates a distance matrix where initially the length of direct paths between vertices is stored, and then calculates the minimum distances considering all intermediate vertices. The result is a matrix that contains the shortest distances between all pairs of vertices in the graph."

{**pre:** Graph $\neq \{\emptyset\}$ }

{**post:** Matrix = $\{\dots\}$ }

Kruskal()

"Performs the Kruskal algorithm on the graph to find the Minimum Spanning Tree. Initially, each vertex forms its own set. Then, edges are selected in increasing order of weight. If an edge connects two vertices in different sets, it is added to the tree. The process is repeated until all vertices are in a single set or the tree contains $n-1$ edges, where n is the number of vertices. The result is a subgraph that is a tree and minimizes the sum of the weights of the selected edges."

{**pre:** Graph $\neq \{\emptyset\}$ }

{**post: return** Graph = $\{\dots\}$ }

Prim()

"Performs the Prim algorithm on the graph to find the Minimum Spanning Tree. It starts with an arbitrary vertex and then iteratively adds the closest vertex to the growing tree set, ensuring no cycles are formed. Unlike the Kruskal algorithm, this one accommodates negative weights. The outcome is a tree that spans the vertices of the original graph, aiming to minimize the total weight."

{**pre:** Graph $\neq \{\emptyset\}$ }

```
{post: return Graph = {...} }
```