TAD WeightedDirectedGraph= graph

graph = {Hash Map }

List = {elements: <$Vertex_1$, $Vertex_2$, $Vertex_3$, …, $Vertex_n$>}

List= {elements: <$Edge_1$, $Edge2_2$, $Edge_3$, …, $Edge_n$ >}

{ inv: Cada vértice en adjacencyList tiene una lista de aristas adyacentes no nula. }

Primitive functions:

• addVertex	Value	➔ graph
• addEdge:	sourceData X destData X weight	➔ graph
• removeVertex:	Value	➔ graph
• removeEdge:	sourceData X destData	➔ graph
• bfs:	StartVertex	➔ list
• dfs:	startVetex	➔ List
• Dfsvisit	Vertex X map X colors X list	➔ graph

WeightedDirectedGraph ()

"Creates a new WeightedDirectedGraph with 0 elements."

{pre: TRUE}

{post: graph = {table: [ ] } }

addVertex ()

"Creates a new vertex in the list "

{pre: value }

{post: list[$Vertex_1$, $Vertex_2$, $Vertex_3$, …, $Vertex_n$]

addEdge ()

"Creates a new edge in the list "

{pre: sourceData ,destData, weight ^ sourceData !0 Null ^ destData != Null}

{post: list[$Edge_1$, $Edge_2$, $Edge_3$, …, $Edge_n$]

removeEdge ()

"Delete from list a real Edge"

{pre: SourceData, DestData }

{post: list[$Edge_1$, $Edge_2$, $Edge_3$, …, $Edge_{n-1}$]

removeVertex ()

"Delete from list a real Edge"

{pre: SourceData, DestData }

{post: list[$Edge_1$, $Edge_2$, $Edge_3$, …, $Edge_{n-1}$]

bfs ()

"The main purpose of the method is to explore and visit all vertices reachable from the start vertex."

{pre: startVertex}

{post:Value {$vertex_n$}]

dfs ()

“main function is to store the values of the vertex visited in the correct order..”

{pre: startVertex}

{post:Value {list[Vertex] }]

dfsvisit ()

“is to perform a recursive depth visit starting from a given vertex in a directed weighted graph”

{pre: Vertex,color, result}

TAD WeightedDirectedGraphMatrix= graphMatrix		
graphMatriz = {matrix[capacity Initial] [capacity Initial] }		
{ inv: The size of the adjacency matrix and the array of vertex must be equal to the current number of vertices in the graph.}		
Primitive functions:		
• addVertex	Value	➔ graphMatrix
• addEdge:	sourceData X destData X weight	➔ graphMatrix
• removeVertex:	Value	➔ graphMatrix
• removeEdge:	sourceData X destData	➔ graphMatrix
• bfs:	StartVertex	➔ list
• dfs:	startVetex	➔ List

WeightedDirectedGraph ()

“Creates a new matrix with 0 elements.”

{pre: capicity }

{post: graphMatrix = {[10][10] }

addVertex ()

“adds a new vertex to the graph ensuring there is enough capacity in the vertex array and keeping the vertex counter updated.”

{pre: value }

{post: vertex = {[Vertex $_{n+1}$] }

AddEdge ()

“adds a weighted edge to the weighted directed graph, provided the source and destination vertices exist in the graph.”

{pre: sourceData, destData , weight}

{post: matrix = {[n+1=sourceData][n+1=destData] }

removeEdge ()

“removes the edge between the source vertex and the destination vertex in the weighted directed graph, provided both vertices exist in the graph.”

{pre: sourceData,DestData }

{pos:[Edge $_{n-1}$]=default Weight

removeVertex ()

“removes a vertex and all associated edges from the weighted directed graph. Both the vertex array and the adjacency matrix are updated to reflect the removal of the vertex and the corresponding connections.”

{pre: Value }

{pos: matrix[Vertex $_{n-1}$][Vertex $_{n-1}$]

dfs ()

“performs a depth traversal on the weighted directed graph starting at a starting vertex, visiting all vertices reachable from that vertex and returning the order of visit in the form of a list.”

{pre: startVertex}

{pos:list= [value vertex $_{n}$]

bfs ()

“performs a breadth-wise traversal on the weighted directed graph starting at a starting vertex, visiting all vertices reachable from that vertex and returning the order of visit in the form of a list..”

{pre: startVertex}

{pos:list= [value vertex $_{n}$]