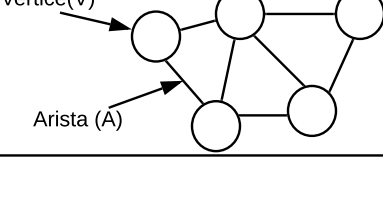


TAD Graph

$\{\text{Inv: } G = (V, A) \wedge G \neq \{\} \wedge V_n \neq V_m\}$
Grafo \rightarrow Grafo EsDirigido: Grafo \rightarrow booleano AgregarVertice: Grafo x Vertice \rightarrow Grafo AgregarArista: Grafo x Arista \rightarrow Grafo ConsultarPeso: Grafo \rightarrow entero EliminarVertice: Grafo x Vertice \rightarrow Grafo EliminarArista: Grafo x Arista \rightarrow Grafo ObtenerVertices: Grafo \rightarrow Vertices bfs: Vertice \rightarrow Grafo dfs: Vertice \rightarrow Grafo prim: Vertice \rightarrow Grafo kruskal: \rightarrow Grafo dijkstra: Vertice x Vertice \rightarrow Grafo floydWarshall: \rightarrow Matriz de numeros

Grafo() "Crea un nuevo Grafo con al menos un vertice" {Pre: Grafo \neq ()} {Post: Grafo = (V, A)}
--

EsDirigido(Grafo) "Verifica si el Grafo es dirigido o no" {Pre: } {Post: true false}
--

AgregarVertice(Grafo, V) "Agrega un nuevo vertice al Grafo" {Pre: } {Post: Grafo}

AgregarArista(Grafo, A) "Agrega una nueva arista al Grafo" {Pre: } {Post: Grafo}
--

ConsultarPeso(Grafo) "Consuta el peso del Grafo" {Pre: } {Post: Entero}

EliminarVertice() "Elimina un vertice si este existe" {Pre: Vertice \wedge Grafo.V>1 } {Post: Grafo}
--

EliminarArista() "Elimina un arista si este existe" {Pre: Arista} {Post: Grafo}

ObtenerVertices() "Obtiene todos los vertices del grafo" {Pre: } {Post: un arreglo de Vertices}

bfs(Vertice) "realiza un recorrido de todo el grafo empezando por los vertices del menor nivel" {Pre: } {Post: un arbol en forma de grafo con la misma forma como se recorrio}
--

dfs(Vertice) "realiza un recorrido de todo el grafo empezando a profundidad" {Pre: } {Post: un arbol en forma de grafo con la misma forma como se recorrio}

prim(Vertice) "realiza un recorrido desde Vertice y va recorriendo las menores aristas adyacentes de los Vertices conectados y va agregando mas Vertices" {Pre: Grafo no es dirigido \wedge Grafo es ponderado} {Post: un arbol en forma de grafo con la misma forma como se recorrio}
--

kruskal() "crea un nuevo grafo que se va generando a partir de las aristas de menor costo hasta que quede un grafo con todos los vertices que se pudieron conectar sin generar un ciclo" {Pre: Grafo no es dirigido \wedge Grafo es ponderado} {Post: un arbol o un bosque en forma de grafo con la misma forma como se recorrio}

dijkstra(Vertice) "encuentra el costo minimo de un Vertice a otro" {Pre: Grafo es ponderado} {Post: un entero que es el costo minimo}

floydWarshall() "encuentra el costo minimo entre cada par de Vertices" {Pre: Grafo es ponderado} {Post: una matriz de los costos ente cada par de vertices}
