|  |
| --- |
| **TAD Graph<T>** |
| Graph = {V = {v1, v2, …, vn}, E = {e1 = (vi1, vj1, w1), e2 = (vi2, vj2, w2), em = (vim, vjm, wm)}, directed, weighted}  https://upload.wikimedia.org/wikipedia/commons/thumb/a/a2/Directed.svg/160px-Directed.svg.png = vk = ek |
| Inv:   1. ∀ek ∈ E, vik ∈ V ∧ vjk ∈ V, wk > 0 2. directed = false ⇒ () 3. weighted = false ⇒ ∀ek ∈ E, wk = 1 |
| Operaciones básicas   * Graph Boolean, Boolean → Graph * addVertex Graph x Vertex → Graph * addEdge Graph x Vertex x Vertex → Graph * addEdge Graph x Vertex x Vertex x Double → Graph * removeVertex Graph x Vertex → Graph * removeEdge Graph x Vertex x Vertex → Graph * getNeighbors Graph x Vertex → List<Vertex> * getNumberOfVertices Graph → Integer * getNumberOfEdges Graph → Integer * areAdjacent Graph x Vertex x Vertex → Boolean * isInGraph Graph x T → Boolean * getEdgeWeight Graph x Vertex x Vertex → Double * setEdgeWeight Graph x Vertex x Vertex x Double → Graph * getVertices Graph → List<Vertex> * getVertex Graph x T → Vertex * isDirected Graph → Boolean * isWeighted Graph → Boolean * bfs Graph x Vertex → Graph * dfs Graph → Graph |

Operaciones

|  |
| --- |
| **Graph(Boolean directed, Boolean weighted)**  “Crea un nuevo grafo que puede o no ser dirigido o ponderado”  Pre:  Post: Graph = {V={}, E={}, directed, weighted} |
| **addVertex(Graph g, Vertex v)**  “Inserta un vértice en el grafo”  Pre: v g.V  Post: v g.V |
| **addEdge(Graph g, Vertex x, Vertex y)**  “Añade una arista de peso 1 que va de x a y. Si el grafo no es dirigido, también la añade de y a x”  Pre: x, y g.V  Post: e = (x, y, 1) g.E. Si g.directed = false, e’ = (y, x, 1) ) g.E |
| **addEdge(Graph g, Vertex x, Vertex y, Double w)**  “Añade una arista de peso w que va de x a y. Si el grafo no es dirigido, también la añade dé y a x”  Pre: x, y g.V, g.weighted = true, w > 0  Post: e = (x, y, w) g.E. Si g.directed = false, e’ = (y, x, w) ) g.E |
| **removeVertex(Graph g, Vertex v)**  “Elimina a v del grafo”  Pre: v g.V  Post: v g.V. Todos los vértices que son incidentes con v g.E |
| **removeEdge(Graph g, Vertex x, Vertex y)**  “Elimina la arista que va de x a y en el grafo”  Pre: x,y g.V, (x, y, \*) g.E  Post: e = (x, y, \*) g.E. Si g.directed = false, e’ = (y, x, \*) ) g.E |
| **getNeighbors(Graph g, Vertex x)**  “Devuelve los vértices v tal que hay una arista desde x hasta v”  Pre: x g.V  Post: vertices = {v1,v2,…,vn} : ∀vi, (x, vi, \*) g.E. |
| **getNumberOfVertices(Graph g)**  “Devuelve el número de vértices en el grafo”  Pre:  Post: n = size(g.V) |
| **getNumberOfEdges(Graph g)**  “Devuelve el número de aristas en el grafo”  Pre:  Post: n = size(g.E) |
| **areAdjacent(Graph g, Vertex x, Vertex y)**  “Devuelve si hay una arista de x a y”  Pre: x,y g.V  Post: true si y solo si (x, y, \*) g.E. |
| **isInGraph(T val)**  “Devuelve si hay un vértice con el valor dado en el grafo”  Pre:  Post: true si y solo si x g.V : value(x) = val. |
| **getEdgeWeight(Graph g, Vertex x, Vertex y)**  “Devuelve el peso de la arista que va de x a y”  Pre: x,y g.V, (x, y, \*) g.E  Post: peso = (x, y).w |
| **setEdgeWeight(Graph g, Vertex x, Vertex y, Double w)**  “Cambia el peso de la arista que va de x a y”  Pre: x,y g.V, (x, y, \*) g.E, w > 0  Post: (x, y, w) g.E |
| **getVertices(Graph g)**  “Devuelve la lista de vértices del grafo”  Pre:  Post: {v1,v2,…,vn} = g.V |
| **getVertex(T val)**  “Devuelve, si existe, el vértice con el valor dado en el grafo”  Pre:  Post: x g.V : value(x) = val. NIL si no existe. |
| **isDirected(Graph g)**  “Devuelve si el grafo es dirigido”  Pre:  Post: g.directed |
| **getWeighted(Graph g)**  “Devuelve si el grafo es ponderado”  Pre:  Post: g.weighted |
| **bfs(Graph g, Vertex s)**  “Realiza el algoritmo Breadth First Search, ajustando información para los vértices del grafo”  Pre: s g.V  Post: ∀u g.V, añade atributos u.pred y u.d, que corresponden a los añadidos por el algoritmo Breadth First Search |
| **dfs(Graph g, Vertex s)**  “Realiza el algoritmo Depth First Search, ajustando información para los vértices del grafo”  Pre:  Post: ∀u g.V, añade atributos u.pred, u.d y u.f, que corresponden a los añadidos por el algoritmo Depth First Search |

**Diseños de casos de pruebas unitarias**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba 1: | | | | |
| Clase | Método | Escenario | Entrada | Resultado |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |