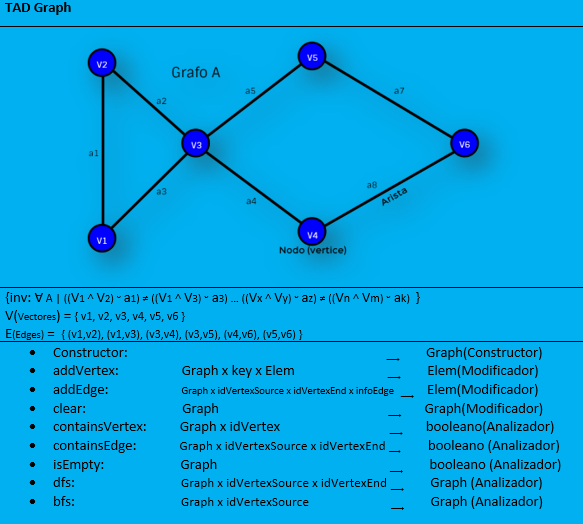


**Algoritmos y Estructuras de Datos**

**Proyecto Final: primer entrega.**

*Nicolás Martinez, Cristian Molina y Juan Manuel Castillo.*

**1. Especificacion del TAD:**

****

|  |
| --- |
| **addVertex( *V element, K key* )**    **“Agrega un vértice (nodo) al grafo”**    **{pre: Se debe introducir como parámetro el elemento y su índice a agregar.}**    **{post: Se agrega el vértice en caso de que no exista otro con los mismos valores. De no ser así; se lanza una excepción informando el error .}** |

|  |
| --- |
| **addEdge( *K idVertexSource, K idVertexEnd, A infoEdge*)**    **“Se agrega una arista(Edge) al grafo”**    **{pre:Se debe introducir como parámetro el vértice de origen y el de destino final.}**    **{post: Se agrega la arista que une el camino entre dos vértices en caso de que existan, de no ser así; se lanzará una excepción como también si la arista ya existe.}** |

|  |
| --- |
| **clear()**    **“Remueve o elimina todos los vértices, y por tanto, aristas que componen el grafo.”**    **{pre: Debe existir el grafo para la implementación del método.}**    **{post: Se han eliminado los elementos (vértices y aristas) del grafo.}** |

|  |
| --- |
| **containsVertex(*K idVertex*)**    **“Prueba la existencia de un vértice en el grafo”**    **{pre:Se introduce por parámetro el índice del vértice a ser buscado.}**    **{post: Verdadero si el vértice existe y falso en caso contrario.}** |

|  |
| --- |
| **containsEdge( *K idVertexSource, K idVertexEnd* )**      **“Prueba la existencia de una arista que une dos vértices en el grafo”**    **{pre:Se introduce por parámetro el índice del vértice de origen y el de llegada.}**    **{post: Verdadero si la arista existe y falso en caso contrario.}** |

|  |
| --- |
| **isEmpty()**    **“Determina si un grafo se encuentra vacío.”**    **{pre: Debe existir el grafo para la implementación del método.}**    **{post: true si se encuentra vacío y falso en caso contrario.}** |

|  |
| --- |
| **dfs( *K idVertexSource*)**    **“Busca un camino entre dos vértices, desde el origen al destino. El orden es importante al ser un grafo dirigido, marcando los caminos ya recorridos.”**    **{pre: Se debe introducir como parámetro el vértice donde iniciará la búsqueda}**    **{post: Lista con los vértices que componen el camino ordenados en el orden a visitar, o lista vacía de no existir camino.}** |

|  |
| --- |
| **bfs( *K idVertexSource*)**    **“Realiza la búsqueda desde el nodo de origen dado por parámetro y se exploran todos los nodos conectados por medio de una arista. Marcando con un color característico los ya visitados, repitiendo el mismo paso hasta recorrer todo el grafo.”**  **{pre: Se debe introducir como parámetro el nodo de origen donde partirá la búsqueda del grafo}**    **{post: Lista con los vértices que componen el camino ordenados en el orden a visitar, o lista vacía de no existir camino.}** |

**2. DIAGRAMA DE CLASES:**

**3. DIAGRAMA DE OBJETOS:**

**4. DISEÑO DE PRUEBAS UNITARIAS:**

***Scene1():*** *Un nodo llamado Medellín y otro nodo llamado Cali con arista correspondiente de 419 (Medellín-Cali) que representa los kilómetros en distancia de un vértice a otro.*

***Scene2():*** *Un nodo llamado Cartagena con arista de 629 (Cartagena-Medellín) que representa los kilómetros de un vértice a otro.*

***Scene3():*** *Un nodo llamado Pereira y otro llamado Cali los cuales no tienen conexión con alguno de los vértices agregados anteriormente.*

|  |  |  |  |
| --- | --- | --- | --- |
| Método | Escenario | Entrada | Salida |
| **addVertexTest()** | *Scene1()*  *Scene2()*  *Scene3()* | Ninguna. | Se prueba que se agreguen correctamente los tres nodos propuestos en los 2 primeros escenarios. En el 3ro deberá lanzar una excepción ya que el nodo “Cali” ya existe. |
| **addEdgeTest()** | *Scene1()*  *Scene2()*  *Scene3()* | Ninguna. | Se prueba que se agreguen correctamente las aristas de los escenarios 1 y 2. El escenario 3 deberá lanzar una excepción al no contar con aristas indicará que no existe. |
| **clearTest()** | *Scene1()*  *Scene2()*  *Scene3()* | Ninguna. | Se prueba que se pueda borrar los nodos y aristas almacenados anteriormente. |
| **containsVertexTest()** | *Scene1()*  *Scene2()*  *Scene3()* | Ninguna. | Se prueba que por medio de un booleano afirme si hay o no vértices (nodos) en el grafo) |
| **containsEdgeTest()** | Scene1()  Scene2()  Scene3() | Ninguna. | Se prueba que por medio de un booleano se confirme si posee o no aristas como lo es en el caso 1-2 y lo contrario en el 3. |
| **isEmptyTest()** | *Scene1()*  *Scene2()*  *Scene3()* | Ninguna. | Se prueba por medio de un booleano si el grafo se encuentra vacío. |
| **dfs()** | *Scene1()*  *Scene2()*  *Scene3()* | Ninguna. | Se prueba que haya un camino entre dos vértices, desde el origen al destino, el orden es importante al ser un grafo dirigido. |

***5. EL PROBLEMA A SOLUCIONAR:***