**TAD Proyecto**

|  |
| --- |
| **TAD Graph** |
|  |
| **Invariante:** El peso de las aristas es un número mayor o igual a 0. |
| **Operaciones primitivas:**   * addNode : Object T → Node * connect : integer initial node, integer final node and integer weight → connection * processBFS : integer source and object BiConsumer → Tree * processDFS : integer source and object BiConsumer → Tree * dijkstras : integer source → List * vertex : → List * floydWharshall : → matrix of integers * prim : → Array of integers * minKey : Array of integers , Array of booleans → integer minimun index * kruskalMST : matrix of integers → * union : integer a, integer b and Array of integers → * find : integer toFind and Array of integers → integer found |

|  |
| --- |
| addNode(T value)  \*Agrega un nodo a el grafo\* |
|
| {pre: value }  {post: Node= } |

|  |
| --- |
| connect(int i, int j, int weight)  \*Crea una arista que conecta dos nodos\* |
| {pre: Node i, Node j and Weight }  {post: }  weight |

|  |
| --- |
| processBFS(int source, BiConsumer<Node, Integer> consumer)  \*Genera un árbol que muestra todos los nodos alcanzables desde el nodo de origen mediante una búsqueda de anchura\* |
| {pre: Node source and consumer}  {post: } |

|  |
| --- |
| processDFS(int source, BiConsumer<Node, Integer> consumer)  \*Genera un árbol que muestra todos los nodos alcanzables desde el nodo de origen mediante una búsqueda a profundidad\* |
| {pre: Node source and consumer}  {post: } |

|  |
| --- |
| dijkstras(int source)  \*Genera una lista del camino más corto desde el nodo dado\* |
| {pre: Node}  {post: } |

|  |
| --- |
| vertex()  \*Busca todos los vertices de un grafo\* |
| {pre:}  {post: lista de nodos que se encuentran en el grafo} |

|  |
| --- |
| floydWarshall()  \*crea una matriz con el peso de los caminos minimos entre todos los nodos\* |
| {pre:}  {post: } |

|  |
| --- |
| prim()  \*crea un arreglo con el peso que hay entre los caminos de un nodo específico a todos los demás\* |
| {pre:}  {post: } |

|  |
| --- |
| minKey(int key[], Boolean mstSet[])  \*busca el valor minimo \* |
| {pre:Node key}  {post: Valor mínimo encontrado} |

|  |
| --- |
| kruskalMST(int cost[][])  \*Busca el bosque de pesos minimos para los vertices del grafo \* |
| {pre: Graph!=vacio}  {post: integer con el costo minimo total} |

|  |
| --- |
| union(int i, int j, int[] parent)  \*une par de vertices, haciendo uno el padre del otro \* |
| {pre:Node a, Node b}  {post: el padre del nodo a va a ser el padre del nodo b } |

|  |
| --- |
| find(int i, int[] parent)  \*Busca el padre del nodo en un arreglo de padres \* |
| {pre:Node i}  {post: Node parent} |