|  |
| --- |
| **TAD<GRAFO >** |
| GRAFO = {V, E}  V: vertices  E: Aristas |
| Inv = ∀v ∀w ∀G (v∈G ^ w∈G ^ v=w) → (v y w son el mismo vértice)  v y w son vértices  G es un grafo  Se lee como: todo vértice v y w pertenecientes a un grafo G, si estos vértices son iguales entonces v y w hacen referencia al mismo vértice. |
| **Main operations**  **Builder → CreateGraph()**: → Graph  **Modifier → InsertVertex (**): Graph x Key x Value → Graph  **Modifier →RemoveVertex ():** Graph x Key → Graph  **Modifier → InsertEdge (**): Graph x Key x Key x Weight → Graph  **Modifier →RemoveEdge ()**: Graph x Key x Key x Weight → Graph  **Analyzer → BFS():** Graph\_1 x Key → Graph\_2  **Analyzer → DFS():** Graph\_1 x1 Key → Graph\_2  **Analyzer → Dijkstra():** Graph x Key → ArrayList<Integer>  **Analyzer → FloydWarshall():** Graph x → int[][]  **Analyzer → Prim():** Graph\_1 x Key x→ Graph\_2  **Analyzer → Kruskal():** Graph\_1 x → Graph\_2  **Analyzer → Contains():** Graph x Key → Boolean  **Analyzer → Contains():** Graph x Key\_1 x Key\_2 → boolean |

|  |
| --- |
| **Graph**()  “Crea un nuevo grafo”  {pre : True}  {pos : new Graph} |

|  |
| --- |
| **InsertVertex**(Graph G, Key K, Value V)  “Inserta un vértice al grafo dado”  {pre : G ≠ nill ^ el vértice no esté en el grafo}  {pos : Se añade el nuevo vértice} |

|  |
| --- |
| **deleteVertex**(Graph G, Key K)  “Elimina un vértice al grafo dado”  {pre : En G hay un vértice con llave K}  {pos : Se elimina el vértice del grafo} |

|  |
| --- |
| **InsertEdge(**Graph G, Key K1, Key K2, Weight W)  “Se añade una arista al grafo dado”  {pre : En G hay un vértice con llave K1 ^ En G hay un vértice con llave K2}  {pos : Se añade la nueva arista} |

|  |
| --- |
| **deleteEdge**(Graph G, Key K1, Key K2)  “Se elimina una arista del grafo dado”  {pre : En G hay un vértice con llave K1 ^ En G hay un vértice con llave K2 ^ hay una arista en G que tiene las llaves K1 y K2}  {pos : Se elimina la arista del grafo} |

|  |
| --- |
| **BFS** (Graph G, Key K)  “Se recorre el grafo en profundidad partiendo del vértice con llave k”  {pre : k es la llave de un vértice del grafo}  {pos : Se retorna un nuevo grafo con todos los vértices accesibles desde el vértice con llave k} |

|  |
| --- |
| **DFS** (Graph G, Key K)  “Se recorre el grafo en anchura partiendo del vértice con llave k”  {pre : k es la llave de un vértice del grafo}  {pos : Se retorna un nuevo grafo con todos los vértices accesibles desde el vértice con llave k} |

|  |
| --- |
| **Dijkstra** (Graph G, Key K)  “Da una lista con los pesos mínimos para llegar del nodo con llave k a cualquier otro”  {pre : k es la llave de un vértice del grafo}  {pos : Retorna una lista de enteros que representa los pesos mínimos para llegar del nodo con llave k hasta los demás} |

|  |
| --- |
| **FloydWarshall**(Graph G, Key K)  “Da una matriz de tamaño n \* n, siendo n la cantidad de vértices, que contiene los pesos mínimos entre cada par de vértices”  {pre : k es la llave de un vértice del grafo}  {pos : Retorna una matriz con los pesos mínimos entre cada par de vértices} |

|  |
| --- |
| Prim(Graph G, Key K)  “Genera el árbol de expansión mínimo a partir de un nodo raíz con llave K”  {pre : k es la llave de un vértice del grafo}  {pos : Retorna el subárbol de expansión mínimo} |

|  |
| --- |
| Kruskal (Graph G)  “Genera el árbol de expansión mínimo a partir de las aristas con menor peso”  {pre : TRUE}  {pos : Retorna el subárbol de expansión mínimo} |

|  |
| --- |
| **Contains** (Graph G, Key k)  “Evalúa si existe en el grafo G un vértice con llave K”  {pre : TRUE}  {pos : Retorna True si lo encuentra, sino, retorna False} |

|  |
| --- |
| **Contains** (Graph G, Key k)  “Evalúa si existe en el grafo G un vértice con llave K”  {pre : TRUE}  {pos : Retorna True si lo encuentra, sino, retorna False} |

|  |
| --- |
| **Contains** (Graph G, Key k1, Key2)  “Evalúa si existe en el grafo G una arista que lleve del nodo con llave k1 a otro con llave k2”  {pre : TRUE}  {pos : Retorna True si lo encuentra, sino, retorna False} |