|  |
| --- |
| ADT GRAPH |
| GRAPH = set of vertices and edges |
| (Inv: There always have to be at least one edge) |
| Primitive operations:   * Graph - ->Graph * insertVertex: Graph x Value ->Graph * insertEdge: Graph x Vertex1 x Vertex2 ->Graph * searchVertex: Graph x Value ->Vertex * searchEdge: Graph x Vertex1 x Vertex2 ->Edge * bfs Graph ->Graph * dfs Graph ->Graph * dijkstra Graph ->Graph * floydWarshall Graph ->Graph * prim Graph ->Graph * kruskal Graph ->Graph |

ESPECIFICACION DETALLADA DE LAS OPERACIONES:

|  |
| --- |
| <Grafo> Construction |
| “<Se crea el grafo>” |

|  |
| --- |
| <insertVertex> Modifier |
| “<Se agrega un vértice, pidiendo como entrada el elemento o valor que desea guardar dentro de él y el grafo donde desea guardar el vértice>” |

|  |
| --- |
| <insertEdge > Modifier |
| “<Se agrega una arista, pidiendo como entrada los dos vértices con los que se desea realizar la conexión y el grafo donde desea agregar la arista>” |

|  |
| --- |
| <searchVertex> Analyzer |
| “<Se busca un vértice, pidiendo como entradas un valor, el programa busca este valor en todo el grafo y cuando lo encuentra devuelve el nodo que lo contiene>” |

|  |
| --- |
| <searchEdge> Analyzer |
| “<Se busca una arista, se pide como entrada los dos nodos que están conectados y el programa busca en todos los nodos dicha conexión, el programa devuelve la arista |

|  |
| --- |
| <bfs> Analyzer |
| “<Given a graph G = (V, E) and an origin vertex s, the BFS systematically  explores the edges of G to discover each vertex that is reachable from s >” |

|  |
| --- |
| <dfs> Analyzer |
| “<expand each node that it locates, on a recurring basis, on a specific path.  When there are no more nodes left to visit on that path, it returns, so  which repeats the same process with each of the siblings of the node already  indicted >” |

|  |
| --- |
| <djikstra> Analyzer |
| “<find the path of minimum length between two nodes >” |

|  |
| --- |
| <floydWarshall> Analyzer |
| “<compares all possible paths between each  pair of vertices of the graph and find the shortest path between two nodes>” |

|  |
| --- |
| <prim> Analyzer |
| “<build minimal weight spanning trees >” |

|  |
| --- |
| <kruskal> Analyzer |
| “<build minimal weight spanning trees >” |