TAD of Graph - Integrative task two.

TAD <Graph>

Abstract object: A graph is a data structure consisting of a set of nodes or vertices, connected to each other by edges or arcs. Each vertex can have a label or value associated with it, and each edge can have additional attributes such as weight or direction. The graph can be directed, where the edges have a specified direction, or undirected, where the edges do not have a specified direction. For this implementation of graphs we have implementations such as DFS, BFS and dijkstra.

{
inv:

- There must be no duplicate vértices. Each vertex in the graph must be unique, that is, there cannot be two vertices with the same id or label.

- There must be no duplicate edges. Each edge in the graph must be unique, which means that no two edges can connect exactly the same vertices.

- All vertices and edges of the graph must be connected. This means that any vertex in the graph must be connected through edges to other vertices.

- DFS and BFS:
    - Nodes that have been visited and no longer have neighbors to visit are considered "BLACK".
    - Unvisited nodes are kept in a "White" pending-to-visit list.
    - The nodes visited by DFS are stored in a data structure (ideally stack for DFS and queue for BFS).

- Dijkstra:
    - Minimum known distances are updated as shorter routes to visited nodes are found.
    - The nodes are marked as visited and are not processed again to avoid cycles in the graph.
    - The shortest paths from the initial node to each visited node are maintained, updating them when a shorter path is found.

}

Primitive operations:

- addVertex: (E element) → void
    - Modifier operation: Because it modifies the structure of the graph to add a new vertex.

- addEdge: (E source, E destination, double weight) → void
    - Modifier operation: Because it modifies the structure of the graph to add a new edge.

- deleteVertex: (E element) → void
    - Modifier operation: Because it modifies the structure of the graph to delete a vertex.

- searchVertex: (E element) → Vertex<E>
    - Analyzer operation: because it only looks for the vertex in the graph and returns it.

- searchEdge: (E source, E destination) → Double
    - Analyzer operation: because it only looks for the edge in the graph and returns the weight of the Edge.

- deleteEdge: (E source, E destination) → void
    - Modifier operation: Because it modifies the structure of the graph to delete an edge.

- BFS: (K sourceElement) → void
    - Analyzer operation: It does not modify the structure of the graph, it only performs a systematic path of width.

- DFS: () → void
    - Analyzer operation: It does not modify the structure of the graph, it only performs a systematic path of depth.

- Dijkstra: (K eSource, K eDestination) → Path<K>
    - Analyzer operation: It does not modify the structure of the graph, but it analyzes and determines the shortest paths.