|  |
| --- |
| TAD Graph<T> |
| Graph<T>: HashTable<K,V>, K = vertexKey, V = Vertex<T> ∧ Vertex<T> = {Edges = {E1 , E2, …. En} } ∧ adjacencyMatrix[ ][ ] = ΣE for Vertex<T>I ∧ Edges.weight = F(Vertex<T>I, Vertex<T>j) → R ∧ weightMatrix[ ][ ]ij = E.weight |
| {Inv.:  - weightMatrixif = Edge.weightij, if {Vi, VJ} ∈ Edges,  Else, ∞.     * adjacencyMatrixij = 0 if Vertex<T> if Edges = { ∅ } * HashTable<K,V>.size = Constant. * if Graph<T>.Edge Origin and destination is irrelevant, then graphGrade = 2 x Edges = Σ Vertex<T>.grade.   Vertex<T>.grade = n  } |
| Primitive Operations:   * addVertex: HashTable<K,V> x Vertex<T> → HashTable <K,V>. * addEdge: Vertex<T> x Vertex<T> x Weight → Edge * getVertex: HashTable<K, V> x Key → Vertex<T> * dfsRoute: HashTable<K,V> x Vertex -> List<Vertex<T>> * dijkstraRoute : HashTable<K,V> -> Vertex<T> -> Map<Vertex<T>, Double> |

|  |
| --- |
| addVertex()  “Add a new vertex to the hash table.”  {pre: Vertex<T>.data ≠ null }  {post: HashTable<K, V> → newHashTable<K, V>} |

|  |
| --- |
| addEdge()  “Creates a new edge that connects two vertexes in the hash table.”  {pre: Vertex<T>i ∧ Vertex<T>j ∈ HashTable<K,V>  Edge.weight > = 0  }  {post: edge = new Edge(Vi,VJ) } |

|  |
| --- |
| getVertex()  “Searches for an existing vertex in the hash table”  {pre: HashTable<K,V> ≠ null }  {post: if Vertex<T> ∈ HashTable<K,V>, then returns Vertex<T>  Else, returns null.  } |

|  |
| --- |
| dfsRoute()  “Perform a Depth-First Search (DFS) traversal on the graph represented by the hash table, starting from a given vertex, and return a list of vertices in the order they are visited during the traversal.”  {pre: Vertex<T> ∈ HashTable<K,V>,  If Vertex<T>i+1 is adjacent to Vertex<T>I, then.  List.add(Vertex<T>)  }  {post: returns List<Vertex<T>>, } |

|  |
| --- |
| dijkstraRoute()  “ This algorithm is executed on the graph starting from the given vertex. The algorithm maintains a priority queue of vertices to explore, initializing the distance from the starting vertex to itself as 0 and the distances to all other vertices as infinity.”  {pre: Vertex<T> ∈ HashTable<K,V>,  Vertex<T>.Edges.weight ≠ null  }  {post: List<Vertex<T>> (for Vertex<T> in List, Weight is minimum  } |