|  |
| --- |
| **TAD Graph** |
| Graph = {Vertices = 〈〉, Edges = 〈 }  *∀ n ∈ N*  *∀ p ^ q ∈ N*  *∀ i ^ j ^ k ^ l ∈ n*  *≠* |
| {inv*:*  *if it is simple graph the edges are unordered so, is the same as ,*  *if it is directed the edges are ordered so, is the different from ,*  *}* |
| * CreateGraph: Graph * AddVertex: Graph X Vertex Graph * AddEdge: Graph X Key (origin) X Key (end) X W (weight) Graph * RemoveVertex: Graph X Key (Vertex) Graph X Vertex * RemoveEdge: Graph X Key (origin) X Key (end) Graph * BFS: Graph X Key (origin) Graph * DFS: Graph Graph * GraphSize: Graph int |
| **CreateGraph [***Graph***] – Constructor**  *“Creates a new and empty Graph”.*  *{Pre: None}.*  *{Post: Returns a Graph without elements.}*  **AddVertex [***addVertex***] – Modifier**  *“Given an key and a value it is checked if the key is already in the graph, if it is not it adds the value to the graph in the form of a vertex.*  *{Pre: There must be a Graph, and the key must be of a compatible type}.*  *{Post: Returns the updated Graph}.*  **AddEdge [***addEdge***] - Modifier**  *“Given an origin key, end key and a weight it is first checked if the two key correspond to a vertex on the graph if that is the case it adds to the vertex or the vertexs (Depending on the type of graph), the value of the weight in their adjacent vertexs and the end vertex acts as a key to know the direction”.*  *{Pre: There must be a Graph, keys must be a compatible type, the weight must be congruent}.*  *{Post: Returns the updated Graph and the vertex with the new edge.}*  **RemoveVertex [***removeVertex***] - Modifier**  *“Removes and return the vertex with the key given, else it returns null, also it deletes the vertexs from the list of adjacencies in each vertex if the vertex removed is adjacent”.*  *{Pre: There must be a Graph}.*  *{Post: Returns the updated Graph and the vertex removed.}*  **RemoveEdge [***removeEdge***] - Modifier**  *“Removes an adjacent vertex from the list of the vertex given the origin and end key, depending on the type of graph it may remove from the end vertex also”.*  *{Pre: There must be a Graph}.*  *{Post: Returns the updated Graph and vertixes.}*  **BFS [***bfs***] – Modifier**  *“It is a travel based on amplitude, it starts from a given vertex and discover each vertex adjacent to that one and all the adjacent to those, till all the vertices are discovered and linked by a parent in a type of three, which represents all the vertices reachable from the given vertex and it represents the shortest path from the origin”.*  *{Pre: There must be a Graph, the origin or given vertex must be on a highly connected graph, else it will be discovered only one part of the graph}.*  *{Post: Returns the updated Graph, with the parent link}.*  **DFS [***dfs***] – Modifier**  *“It is a depth search that travel ordered but not uniformly, that starts at the root vertex (selecting some arbitrary vertex as the root vertex) and explores as far as possible along each path before backtracking and repeating”.*  *{Pre: There must be a non-empty Graph (As it goes from all the vertices not visited it could be not highly connected)}.*  *{Post: Returns the updated Graph, with the parent link}.*  **GraphSize [***size***] – Analyser.**  *“It returns the number of vertexs with value on the graph.”.*  *{Pre: There must be a Graph}.*  *{Post: Returns a number corresponding to the operation}.* |