| ***TAD Graph*** | | |
| --- | --- | --- |
| ***Graph<T> adjacencyGraph={adjacencyList<Vertex>t={v1,v2,v3}}*** | | |
| ***{inv: Vertex<T>!= NIL, T != NIL }*** | | |
| ***Primitive Operations*** | | |
| ***Graph*** | ***Graph→Graph*** | |
| ***addVertex*** | ***Graph x T → boolean*** | |
| ***addEdge*** | ***Graph x Vertex<T> → boolean*** | |
| ***searchValue*** | ***Graph x Vertex<T> → boolean*** | |
| ***searchVertex*** | ***Graph x T → Vertex<T>*** | |
| ***deleteVertex*** | ***Graph x T → boolean*** | |
| ***toString*** | ***Graph → String*** | |
| ***easierSolutionDijkstra*** | ***Graph Vertexx Vertex → List<Vertex>*** | |

| ***easierSolutionDijkstra***  ***(Vertex<T>, Vertex<T>)*** | |
| --- | --- |
| ***“This Method uses other auxiliary methods to find the shortest path between a given node and final node”*** | |
| ***{pre: vertex<T> !=NIl}*** | |
| ***{post: List with all the vertices that creates the shortest path between both nodes}*** | |

| ***Graph()*** | |
| --- | --- |
| *“Creates a graph with an adjacency list ”* | |
| *{pre:-}* | |
| *{post: Graph <T>mA= {adjacencyList ={}}}* | |

| ***addVertex(T)*** | |
| --- | --- |
| *“Add a new Vertex<T>* ***ver*** *to the adjacencyList , given a content T”* | |
| *{pre:adjacencyList ={∅} OR adjacencyList={v1,v2…vn}}* | |
| *{post: adjacencyList={****ver****} or adjacencyList={v1,v2…****ver****} }* | |

| ***addEdge(Vertex<T>, Vertex<T>)*** | |
| --- | --- |
| *“Add each vertex to each other adjacencyList of vertex creating a connection called edge”* | |
| *{pre: vertex1 ៱ vertex2 ∊ adjacencyList}* | |
| *{post: vertex1=adjacencyListVertex={v2} AND vertex2=adjacencyListVertex={v1}}* | |

| ***searchValue(Vertex<T>)*** | |
| --- | --- |
| *“Searches if a vertex exist in the adjacencyList”* | |
| *{pre: adjacencyList ≠ NIL}* | |
| *{post: true if vertex ∊ adjacencyList }* | |

| ***searchVertex(T)*** | |
| --- | --- |
| *“Searches a vertex in the adjacencyList given its content ”* | |
| *{pre: adjacencyList ≠ NIL, vertex ∊ adjacencyList}* | |
| *{post: Vertex<T> found}* | |

| ***deleteVertex(T)*** | |
| --- | --- |
| *“Deletes a vertex from the adjacencyList given its content”* | |
| *{pre:adjacencyList={v1,v2,v3,.......vn}}* | |
| *{post: adjacencyList={v2,v3,.......vn}}* | |

| ***toString()*** | |
| --- | --- |
| *“Convert the graph information into a readable string format”* | |
| *{pre:-}* | |
| *{post: "Graph Information: \nVertices: \n" concatenated with the list of vertices and "Edges: \n" concatenated with the list of edges}* | |

| ***TAD MatrixGraph*** | | |
| --- | --- | --- |
| MatrixGraph<T> m= {Int =maxVertex, List= vertexList,  int[ ][ ]=matrizAdjacency[maxVertex ][maxVertex ]}} | | |
| {inv: **maxVertex**>0, **matrixAdjacecy[ i ][ j ]** =1 ˇ 0} | | |
| **Primitive Operations** | | |
| ***MatrixGraph*** | **MatrixGraph x Integer → MatrixGraph** | |
| ***initializeMatrix*** | ***MatrixGraph → MatrixGraph*** | |
| ***addVertex*** | ***MatrixGraph x T → boolean*** | |
| ***addEdge*** | ***MatrixGraph x Vertex<T> x Vertex<T> → boolean*** | |
| ***returnIndex*** | ***MatrixGraph x Vertex<T> → Integer*** | |
| ***searchVertex*** | ***MatrixGraph x T →Vertex<T>*** | |
| ***deleteVertex*** | ***MatrixGraph x T → boolean*** | |
| ***printMatrix*** | ***MatrixGraph → String*** | |
| ***dijkstrja*** | ***MatrixGraphx Vertex<T>x Vertex<T> →List<Vertex<T>>*** | |

| ***MatrixGraph( Integer)*** | |
| --- | --- |
| *“Creates a graph with an adjacency matrix with Integer maxVertex dimensions”* | |
| *{pre:-}* | |
| *{post: MatrixGraph<T> m={adjacencyMatrix[ maxVertex][ Maxvertex], vertexList={ }} }* | |

| ***initializeMatrix( )*** | |
| --- | --- |
| *“Initializes the adjacency matrix with all positions in 0, meaning that still haven't added edges between the vertices”* | |
| *{pre: adjacencyMatrix[ maxVertex][maxVertex]}* | |
| *{post: adjacencyMatrix[ i ][ j ]=0}* | |

| ***addVertex(T)*** | |
| --- | --- |
| *“Given the content T, this method add a new* ***Vertex<T> newVertex*** *to the vertex list ”* | |
| *{pre: vertexList={ } OR vertexList={v1,v2,v3…vn} }* | |
| *{post: vertexList={newVertex} OR vertexList={v1,v2,v3…newVertex}}* | |

| ***addEdge(Vertex<T>, Vertex<T>)*** | |
| --- | --- |
| *“Given the content T, this method add a edge between two vertex:* ***Vertex<T> v1 Vertex<T> v2*** *to the adjacency Matrix ”* | |
| *{pre: adjacencyMatrix[maxVertex][maxVertex], v1,v2 є vertexList}* | |
| *{post: adjacencyMatrix[Index\_v1][Index\_v2] =1 AND adjacencyMatrix[Index\_v2][Index\_v1] =1 }* | |

| ***deleteVertex(T)*** | |
| --- | --- |
| *“Given the content T, this method deletes a* ***Vertex<T> v1*** *from the vertex list and deleted the edges with related vertexes ”* | |
| *{pre: vertexList={v1,v2,v3…vn} }* | |
| *{post: vertexList={v2,v3… vn}}* | |

| ***printMatrix()*** | |
| --- | --- |
| *“This method print the information in the matrix that represent the edges between nodes ”* | |
| *{pre: adjacencyMatrix[i][j] =0 v 1 }* | |
| *{post: String with all the edges}* | |

| ***dijkstra(Vertex<T>, Vertex<T>)*** | |
| --- | --- |
| *“This Method uses other auxiliary methods to find the shortest path between a given node and final node”* | |
| *{pre: vertex<T> !=NIl}* | |
| *{post: List with all the vertices that creates the shortest path between both nodes}* | |

| ***TAD Vertex*** | | |
| --- | --- | --- |
| Vertex<T>={ T content, ArrayList<Vertex<T>> adjacencyVertex} | | |
| {inv: content ≠ NIL} | | |
| **Primitive Operations** | | |
| ***Vertex*** | **Vertex x T -> Vertex** | |
| ***Vertex*** | **Vertex x List<Vertex<T>> x T → *Vertex*** | |
| ***setContent*** | ***Vertex x T → Vertex*** | |
| ***getContent*** | ***Vertex → T*** | |

| ***Vertex( T)*** | |
| --- | --- |
| *“Creates a vertex with T content”* | |
| *{pre:-}* | |
| *{post: Vertex<T> vertex = {T}}* | |

| ***Vertex(T , List )*** | |
| --- | --- |
| *“Creates a vertex with T content”* | |
| *{pre: -}* | |
| *{post: Vertex<T> vertex ={T, List} }* | |

| ***getContent()*** | |
| --- | --- |
| *“Returns the content of a vertex”* | |
| *{pre: Vertex<T> vertex= {T ≠ NIL}}* | |
| *{post: T}* | |

| ***setContent(T)*** | |
| --- | --- |
| *“modify the content of the vertex, replaces the older value for the new one”* | |
| *{pre:Vertex ={T ≠ NIL}}* | |
| *{post: Vertex={T}}* | |