**ADTs MatrixGraph**

G = (vertices={$V_1$,…,Vn}, matrix={Integer={Integer{$n_1$,..$n_x$}}}, isDirected=<boolean>, ), where V are vertices, and E is edges

{*inv*: *There cannot be two vertexes with the same value on the Graph.*}

Primitive Operations:

| | | | | |
|---|---|---|---|---|
| - | MatrixGraph | MatrixGraph x boolean x int | → | MatrixGraph |
| - | addVertex | MatrixGraph x Value | → | MatrixGraph |
| - | addEdge | MatrixGraph x Value x Value x String x Int | → | MatrixGraph |
| - | deleteVertex | MatrixGraph x Value | → | MatrixGraph |
| - | deleteEdge | MatrixGraph x Value x Value x String | → | MatrixGraph |
| - | searchVertexIndex | MatrixGraph x Value | → | Int |
| - | searchEdge | MatrixGraph x Value x Value x String | → | boolean |
| - | dijkstra | MatrixGraph x Value x Value | → | Int |
| - | DFS | MatrixGraph x Value | → | MatrixGraph |
| - | getNumVertices | MatrixGraph | → | Int |
| - | obtainVertex | MatrixGraph x Int | → | V |
| - | getEdgeWeight | MatrixGraph x V x V | → | Int |
| - | DFSVALIDATOR | MatrixGraph x Value[] | → | boolean |
| - | dfssimplified | MatrixGraph x V[] | → | boolean |
| - | depthfirstsearchRecursive | MatrixGraph x V x Value[] | → | boolean |
| - | getEdgeWeightList | MatrixGraph x V[] | → | Int[] |
| - | subGraphDistance | MatrixGraph x Int[] | → | Int |
| - | getVertices | MatrixGraph | → | V[] |
| - | print | MatrixGraph | → | MatrixGraph |
| - | checkShortPath | MatrixGraph | → | boolean |

**MatrixGraph()**

“Create a new MatrixGraph”

{ pre: isDirected=<boolean>, vertices<int> }

{ post: MatrixGraph={vertices={$V_{vertices}$,}, isDirected=<isDirected>, matrix={Integer={Integer{$n_1$,..$n_{vertices}$}}}

**addVertex()**

**“**Adds vertex “v” to the graph G”

{*pre*: value=<Value>}

{ post: MatrixGraph={vertices={$V_{vertices+1}$,}, isDirected=<isDirected>, matrix={Integer={Integer{$n_1$,..$n_{vertices+1}$}}}

**addEdge()**

**“**Adds a edge from vertex “a” to vertex “b” of weight “w” **”**

{*pre*: *pre*: start=<Value>, end=<Value>, id=<String>, weight=<int>}

{*post*: : In the graph, matrix[start][end]=weight }

**deleteEdge()**
"Removes edge(u,v) with identification "id" from the graph "

{*pre*: *There must be an edge between u and v* }

{ post: The edge is removed from the graph G}

**deleteVertex(**v**)**
**"**Deletes the vertex "v" from the Graph "

{*pre*: *u must belong to the set of vertices of the graph G* }

{ post: : The vertex is removed from the graph G }

**searchVertexIndex()**
**"**Searches the index in the graph of vertex "v" "

{*pre*: *vertex "u" must be part of the graph*}

$post: \ i \in \aleph \land i \geq 0$

**searchEdge()**
**"**Checks if there is an edge between two nodes in the graph "

{*pre*: $[(a \land b)$ *must be part of the graph*$] \land [$*there must be an edge between* $(a \land b)]$ }

*post*: (*true if there is an edge between* $a \land b$) $\lor$ (*false if there is not an edge between* $a \land b$

**Dikstraj()**
**"**Finds the shortest path from vertex a to b in the graph. "

{*pre*: $[(a \land b)$ *are part of the Graph*$] \land [$*there is an edge between* $(a \land b)$ ]}

{ post: distance between a and b in terms of their weights}

**DFS( )**
**"**Explore in depth by visiting all the neighbors of a vertex in the graph G from a vertex "

{*pre*: start=<V>}

{ post: Traversal of graph from start}

**getNumVertices()**
“Returns the number of vertices in the graph ”

{*pre*: TRUE }

{ post: vertices={$V_1$,…,Vn}}

**obtainVertex()**
**“**Returns the name of vertex in an index ”

{*pre*: start=<V>, end=<V> }

{ post: if there is a connection between start and end, returns matrix[startIndex][endIndex], else 0}

getEdgeWeight()
“weight of an edge that connects two vertices ”

{*pre*: start=<V>, end=<V>}

{*post*: matrix[startIndex][endIndex] if (start /\ end) ∈ vertices, else 0}

**DFSVALIDATOR()**
**“**Assures that the path is correct ”

{*pre*: vertexes=<Value[]> }

{*post*: (*true if there* vertexes are connected among them), else (*false*)}

**dfsSimplified()**
“Checks wheter the nodes of the subgraph with certain vertices are connected”

{*pre*: subgraph=<V[]>}

{*post*: (*true if there* vertexes are connected among them), else (*false*)}

**depthfirstsearchRecursive ( )**
**“**Checks whether a graph is connected given its list of vertices and a start”

{*pre*: vertex=<V>, subgraph=<V[]>}

{*post*: (*true if there* vertexes are connected among them), else (*false*)}

**getEdgeWeightList()**
"List of the weight of edges connecting the different vertives"

{*pre*: *vertex "u" must be part of the graph*}

{*post*: n[] where length of $n \geq 0$ and $n \in Z$}

**subGraphDistance()**
**"**Total weight of the edges of the path"

{*pre*: subEdged=<int[]> }

{ post: n=sum(matrix), where $n \geq 0$ and $n \in Z$ }

**getVertices()**
"Returns a matrix of vertices of a graph"

{*pre*: TRUE}

{*post*: vertices={$V_1$,…,Vn}}

**print()**
**"**Show the graph in its matrix representation"

{*pre*: }

{*post*: shows graph in console}

**checkShortPath()**
"Checks wheter an array of vertices corresponds to the shortest path of the graph"

{*pre*: subgraph=<V[]>}

{*post*: (*true if* the subgraph is the shortest path in the graph), else (*false*)