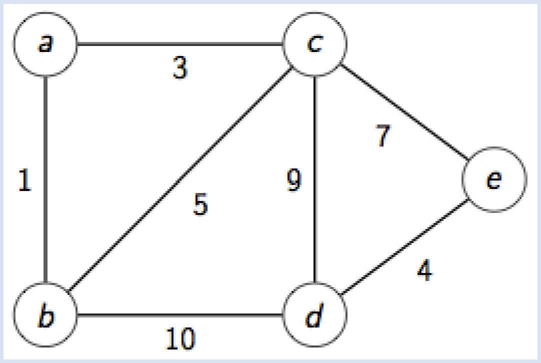
TAD Graph $\langle \boldsymbol{K}, \boldsymbol{E} \rangle$



{inv: $\boldsymbol{G} = (V, E), \forall (\boldsymbol{a}, \boldsymbol{b}) \in E \ \exists \ (\boldsymbol{b}, \boldsymbol{a}) \in E$}

<u>Primitive operations:</u>

• getGraph		->ArrayList
• getWeigthMatrix:		->double[][]
• addVertex:	Vertex<K,E>	->void
• getVertex:	int	->Vertex<E>
• deleteVertex:	Vertex<K,E>	->void
• addEdge:	Vertex< K,E> Vertex< K,E>	->void
• addEdge:	Vertex< K,E> Vertex< K,E> double	->void
• deleteEdge:	Vertex< K,E> Vertex< K,E>	->void
• isDirected:	Vertex< K,E>	->boolean
• getAdjacents:	Vertex< K,E>	->ArrayList
• BFS	IGraph<K,E> Vertex<K,E>	->ArrayList
• DFS	IGraph<K,E> Vertex<K,E>	->ArrayList
• dijkstra	IGraph<K,E> Vertex<K,E>	->Object
• floyd-Warshall	IGraph<K,E>	->double[][]
• prim	IGraph<K,E> Vertex<K,E>	->ArrayList
• kruskal	IGraph<K,E>	->ArrayList

getGraph
"Returns the graph."
{pre: TRUE}
{post: ArrayList with its respective identifier (K) and the object it contains (E)}
Analyzer

getWeigthMatrix
"Returns a matrix where we can observe the weights of each edge."
{pre: A weighted graph must exist}
{post: Graph weight matrix}
Analyzer

addVertex (v)
"Adds a new vertex to the graph."
{pre: TRUE}
{post: The vertex has been added}
Modifier

getVertex (index)
"Returns the vertex of a given index."
{pre: The vertex must exist}
{post: The vertex has been returned}
Analyzer

deleteVertex (v)
"Deletes the vertex v from the graph."
{pre: The vertex must exist}
{post: The vertex has been deleted}
Modifier

addEdge (u,v)
"Adds a new edge to the graph given two vertexes."
{pre: TRUE}
{post: The edge has been added between the two vertices}
Modifier

addEdge (u,v,w)
"Add an edge between the two vertices, assigning it a weight w."
{pre: TRUE}
{post: The edge has been added between the two vertices with its respective weight.}
Modifier

deleteEdge (u,v)
"Delete an edge between the two vertices."
{pre: The edge must exist}
{post: The vertex has been deleted}
Modifier

isDirected (v)
"Returns a boolean indicating if the graph is directed or undirected."
{pre: The graph must exist}
{post: Indicates if the graph is directed or undirected}
Analyzer

getAdjacents (v)
"Given a vertex, it returns an ArrayList with the nodes adjacent to said vertex."
{pre: The vertex must exist}
{post: ArrayList with the nodes adjacent to the given vertex}
Analyzer

BFS (graph, vertx)
"Searches width for a graph, starting at the root and scans all neighboring nodes, then the adjacent nodes of the neighbors."
{pre: The graph must exist}
{post: Returns an arraylist of vertices}
Analyzer

DFS (graph, vertx)
"It runs through the nodes of a graph in an orderly manner, expanding each and every one of the nodes it locates, on a recurring basis, on a specific path."
{pre: The graph must exist}
{post: Returns an arraylist of vertices}
Analyzer

dijkstra (graph, vertx)
"Determine the shortest path given an origin vertex to the rest of all vertices of the weighted graph."
{pre: The graph must exist, be weighted, directed, and connected}
{post: Returns two arraylists showing the final L and C}
Analyzer

floyd-Warshall (graph)
"Determine the shortest path between all pairs of vertices in one run."
{pre: The graph must exist, be weighted, directed, and connected}
{post: Returns a matrix of weights of the minimum path}
Analyzer

prim (graph, vertx)
"Find the minimum covering tree, finding the subset of edges that form a tree with all vertices, where the total weight of all edges is the minimum possible."
{pre: The graph must exist, be weighted, undirected, and connected}
{post: Returns an arraylist with the minimum spanning tree from the given vertex}
Analyzer

kruskal (graph)
"Finds a minimum covering tree in a weighted connected graph by looking for the subset of edges including all vertices and the sum of the value of its edges is the minimum. If unrelated, it finds a minimal spanned forest, in other words, a minimal spanned tree for each connected component."
{pre: The graph must exist, be weighted, and undirected}
{post: Returns an arraylist with the minimum covering tree or the minimum covering forest}
Analyzer