Graph algorithms - practical work no. 1

Cretu Cristian - 913

Constructor

Initiializes a new instance of the `Graph` class with the specified number of vertices and edges. If the number of vertices is not specified, the graph will be initialized with 0 vertices. If the number of edges is not specified, the graph will be initialized with 0 edges. **Complexity:** $O(1)$

Destructor

Destroys the graph and clears the memory allocated for the edges, since the inbound and outbound edges are stored in a `std::list` data structure. **Complexity:** $O(V + E)$

addEdge

Adds a new edge from the source to the target vertex with the specified cost. If the edge already exists, the method does nothing. **Complexity:** $O(1)$

removeEdge

Removes the edge from the source to the target vertex. If the edge does not exist, the method does nothing. **Complexity:** $O(V + E)$

addVertex

Adds a new vertex to the graph. If the vertex already exists, the method does nothing. **Complexity:** $O(1)$

removeVertex

It removes the vertex from the graph and all the edges that have the vertex as a source or target. If the vertex does not exist, the method does nothing. **Complexity:** $O(V + E)$

getOutboundEdges

Returns a pair of iterators that define the range of outbound edges for the specified vertex. If the vertex does not exist, the method returns an empty list. **Complexity:** $O(1)$

getInboundEdges

Returns a pair of iterators that define the range of inbound edges for the specified vertex. If the vertex does not exist, the method returns an empty list. **Complexity:** $O(1)$

getOutEdges

Returns a vector of outbound edges for the specified vertex. If the vertex does not exist, the method returns an empty list. **Complexity:** $O(1)$

getInEdges

Returns a vector of inbound edges for the specified vertex. If the vertex does not exist, the method returns an empty list. **Complexity:** $O(1)$

getVerticesList

Returns a vector of vertices in the graph as const references. If the graph is empty, the method returns an empty list. **Complexity:** $O(V)$

getEdgeID

Returns the edge ID for the edge from the source to the target vertex. If the edge does not exist, the method returns -1. The EdgeID is calculated as the number of edges in the graph at the moment the edge is added. **Complexity:** $O(V)$

isEdge

Returns true if the edge from the source to the target vertex exists, otherwise returns false. **Complexity:** $O(V)$

getInDegree

Returns the in-degree of the specified vertex. If the vertex does not exist, the method returns 0. **Complexity:** $O(1)$

getOutDegree

Returns the out-degree of the specified vertex. If the vertex does not exist, the method returns 0. **Complexity:** $O(1)$

getCost

Returns the cost of the edge from the source to the target vertex. If the edge does not exist, the method returns -1. **Complexity:** $O(V)$

setCost

Sets the cost of the edge from the source to the target vertex. If the edge does not exist, the method does nothing. **Complexity:** $O(V)$

getEndpoints

Returns a pair of vertices that are the endpoints of the edge with the specified edge ID. If the edge does not exist, the method returns (-1, -1). **Complexity:** $O(V)$

copyGraph

Returns a copy of the graph. The method creates a deep copy of the graph, including all the edges and vertices. **Complexity:** $O(V + E)$

createRandomGraph

Creates a random graph with the specified number of vertices and edges. The method keeps generating random VALID edges until the number of edges reaches the specified maximum number of edges. **Complexity:** $O(V + E)$