Graphenklassen

Die folgenden Klassen `Graph`, `Vertex` und `Edge` werden verwendet, um die nicht-lineare dynamische Datenstruktur Graph zu realisieren.

Die Klasse `Graph`

Die Klasse `Graph` stellt einen ungerichteten, kantengewichteten Graphen dar. Es können Knoten- und Kantenobjekte hinzugefügt und entfernt, flache Kopien der Knoten- und Kantenlisten des Graphen angefragt und Markierungen von Knoten und Kanten gesetzt und überprüft werden. Des Weiteren kann eine Liste der Nachbarn eines bestimmten Knoten, eine Liste der inzidenten Kanten eines bestimmten Knoten und die Kante von einem bestimmten Knoten zu einem anderen Knoten angefragt werden. Abgesehen davon kann abgefragt werden, welches Knotenobjekt zu einer bestimmten ID gehört und ob der Graph leer ist.

Dokumentation der Klasse `Graph`

`Graph()`

Ein Objekt vom Typ `Graph` wird erstellt. Der von diesem Objekt repräsentierte Graph ist leer.

`void addVertex(Vertex pVertex)`

Der Auftrag fügt den Knoten `pVertex` vom Typ `Vertex` in den Graphen ein, sofern es noch keinen Knoten mit demselben ID-Eintrag wie `pVertex` im Graphen gibt und `pVertex` eine ID ungleich `null` hat. Ansonsten passiert nichts.

`void addEdge(Edge pEdge)`

Der Auftrag fügt die Kante `pEdge` in den Graphen ein, sofern beide durch die Kante verbundenen Knoten im Graphen enthalten sind, nicht identisch sind und noch keine Kante zwischen den beiden Knoten existiert. Ansonsten passiert nichts.

`void removeVertex(Vertex pVertex)`

Der Auftrag entfernt den Knoten `pVertex` aus dem Graphen und löscht alle Kanten, die mit ihm inzident sind. Ist der Knoten `pVertex` nicht im Graphen enthalten, passiert nichts.

`void removeEdge(Edge pEdge)`

Der Auftrag entfernt die Kante `pEdge` aus dem Graphen. Ist die Kante `pEdge` nicht im Graphen enthalten, passiert nichts.

`Vertex getVertex(String pID)`

Die Anfrage liefert das Knotenobjekt mit `pID` als ID. Ist ein solches Knotenobjekt nicht im Graphen enthalten, wird `null` zurückgeliefert.

`List<Vertex> getVertices()`

Die Anfrage liefert eine neue Liste aller Knotenobjekte vom Typ `List<Vertex>`. Enthält der Graph keine Knotenobjekte, so wird eine leere Liste zurückgeliefert.

`List<Vertex> getNeighbours(Vertex pVertex)`

Die Anfrage liefert alle Nachbarn des Knotens `pVertex` als neue Liste vom Typ `List<Vertex>`. Hat der Knoten `pVertex` keine Nachbarn in diesem Graphen oder ist gar nicht in diesem Graphen enthalten, so wird eine leere Liste zurückgeliefert.

`List<Edge> getEdges()`

Die Anfrage liefert eine neue Liste aller Kantenobjekte vom Typ `List<Edge>`. Enthält der Graph keine Kantenobjekte, so wird eine leere Liste zurückgeliefert.

`List<Edge> getEdges(Vertex pVertex)`

Die Anfrage liefert eine neue Liste aller inzidenten Kanten zum Knoten `pVertex`. Hat der Knoten `pVertex` keine inzidenten Kanten in diesem Graphen oder ist gar nicht in diesem Graphen enthalten, so wird eine leere Liste zurückgeliefert.

`Edge getEdge(Vertex pVertex, Vertex pAnotherVertex)`

Die Anfrage liefert die Kante, welche die Knoten `pVertex` und `pAnotherVertex` verbindet, als Objekt vom Typ `Edge`. Ist der Knoten `pVertex` oder der Knoten `pAnotherVertex` nicht im Graphen enthalten oder gibt es keine Kante, die beide Knoten verbindet, so wird `null` zurückgeliefert.

`void setAllVertexMarks(boolean pMark)`

Der Auftrag setzt die Markierungen aller Knoten des Graphen auf den Wert `pMark`.

`boolean allVerticesMarked()`

Die Anfrage liefert `true`, wenn die Markierungen aller Knoten des Graphen den Wert `true` haben, ansonsten `false`.

`void setAllEdgeMarks(boolean pMark)`

Der Auftrag setzt die Markierungen aller Kanten des Graphen auf den Wert `pMark`.

`boolean allEdgesMarked()`

Die Anfrage liefert `true`, wenn die Markierungen aller Kanten des Graphen den Wert `true` haben, ansonsten `false`.

`boolean isEmpty()`

Die Anfrage liefert `true`, wenn der Graph keine Knoten enthält, ansonsten `false`.

Die Klasse `Vertex`

Die Klasse `Vertex` stellt einen einzelnen Knoten eines Graphen dar. Jedes Objekt dieser Klasse verfügt über eine im Graphen eindeutige ID als `String` und kann diese ID zurückliefern. Darüber hinaus kann eine Markierung gesetzt und abgefragt werden.

Dokumentation der Klasse `Vertex`

`Vertex(String pID)`

Ein neues Objekt vom Typ `Vertex` mit der ID `pID` wird erstellt. Seine Markierung hat den Wert `false`.

`String getID()`

Die Anfrage liefert die ID des Knotens als `String`.

`void setMark(boolean pMark)`

Der Auftrag setzt die Markierung des Knotens auf den Wert `pMark`.

`boolean isMarked()`

Die Anfrage liefert `true`, wenn die Markierung des Knotens den Wert `true` hat, ansonsten `false`.

Die Klasse `Edge`

Die Klasse `Edge` stellt eine einzelne, ungerichtete Kante eines Graphen dar. Beim Erstellen werden die beiden durch sie zu verbindenden Knotenobjekte und eine Gewichtung als `double` übergeben. Beide Knotenobjekte können abgefragt werden. Des Weiteren können die Gewichtung und eine Markierung gesetzt und abgefragt werden.

Dokumentation der Klasse `Edge`

`Edge(Vertex pVertex, Vertex pAnotherVertex, double pWeight)`

Ein neues Objekt vom Typ `Edge` wird erstellt. Die von diesem Objekt repräsentierte Kante verbindet die Knoten `pVertex` und `pAnotherVertex` mit der Gewichtung `pWeight`. Ihre Markierung hat den Wert `false`.

`void setWeight(double pWeight)`

Der Auftrag setzt das Gewicht der Kante auf den Wert `pWeight`.

`double getWeight()`

Die Anfrage liefert das Gewicht der Kante als `double`.

`Vertex[] getVertices()`

Die Anfrage gibt die beiden Knoten, die durch die Kante verbunden werden, als neues Feld vom Typ `Vertex` zurück. Das Feld hat genau zwei Einträge mit den Indexwerten `0` und `1`.

`void setMark(boolean pMark)`

Der Auftrag setzt die Markierung der Kante auf den Wert `pMark`.

`void setWeight(double pWeight)`

Der Auftrag setzt das Gewicht der Kante auf den Wert `pWeight`.

`boolean isMarked()`

Die Anfrage liefert `true`, wenn die Markierung der Kante den Wert `true` hat, ansonsten `false`.