4.3 Graphen

Graphen

Ein ungerichteter Graph besteht aus einer Menge von Knoten und einer Menge von Kanten. Die Kanten verbinden jeweils zwei Knoten und können ein Gewicht haben.

Die Klasse GraphNode

Objekte der Klasse `GraphNode` sind Knoten eines Graphen. Ein Knoten hat einen Namen und kann markiert werden.

Dokumentation der Klasse `GraphNode`

Konstruktor **`GraphNode(String pName)`**
Ein Knoten mit dem Namen `pName` wird erzeugt. Der Knoten ist nicht markiert.

Auftrag **`void mark()`**
Der Knoten wird markiert, falls er nicht markiert ist, sonst bleibt er unverändert.

Auftrag **`void unmark()`**
Die Markierung des Knotens wird entfernt, falls er markiert ist, sonst bleibt er unverändert.

Anfrage **`boolean isMarked()`**
Die Anfrage liefert den Wert `true`, wenn der Knoten markiert ist, sonst liefert sie den Wert `false`.

Anfrage **`String getName()`**
Die Anfrage liefert den Namen des Knotens.

Die Klasse Graph

Objekte der Klasse `Graph` sind ungerichtete, gewichtete Graphen. Der Graph besteht aus Knoten, die Objekte der Klasse `GraphNode` sind, und Kanten, die Knoten miteinander verbinden. Die Knoten werden über ihren Namen eindeutig identifiziert.

Dokumentation der Klasse `Graph`

Konstruktor **`Graph()`**
Ein neuer Graph wird erzeugt. Er enthält noch keine Knoten.

Anfrage **`boolean isEmpty()`**
Die Anfrage liefert `true`, wenn der Graph keine Knoten enthält, andernfalls liefert die Anfrage `false`.

Auftrag **`void addNode(GraphNode pNode)`**
Der Knoten `pNode` wird dem Graphen hinzugefügt. Falls bereits ein Knoten mit gleichem Namen im Graphen existiert, wird dieser Knoten nicht eingefügt. Falls `pNode` `null` ist, verändert sich der Graph nicht.

Anfrage **`boolean hasNode(String pName)`**
Die Anfrage liefert `true`, wenn ein Knoten mit dem Namen `pName` im Graphen existiert. Sonst wird `false` zurück gegeben.

Anfrage **`GraphNode getNode(String pName)`**
Die Anfrage liefert den Knoten mit dem Namen `pName` zurück. Falls es keinen Knoten mit dem Namen im Graphen gibt, wird `null` zurück gegeben.

Auftrag **`void removeNode(GraphNode pNode)`**
Falls `pNode` ein Knoten des Graphen ist, so werden er und alle mit ihm verbundenen Kanten aus dem Graphen entfernt. Sonst wird der Graph nicht verändert.

Auftrag **`void addEdge(GraphNode pNode1, GraphNode pNode2, double pWeight)`**
Falls eine Kante zwischen `pNode1` und `pNode2` noch nicht existiert, werden die Knoten `pNode1` und `pNode2` durch eine Kante verbunden, die das Gewicht `pWeight` hat. `pNode1` ist also Nachbarknoten von `pNode2` und umgekehrt. Falls eine Kante zwischen `pNode1` und `pNode2` bereits existiert, erhält sie das Gewicht `pWeight`. Falls einer der Knoten `pNode1` oder `pNode2` im Graphen nicht existiert oder `null` ist, verändert sich der Graph nicht.

Anfrage **`boolean hasEdge(GraphNode pNode1,GraphNode pNode2)`**
Die Anfrage liefert `true`, falls eine Kante zwischen `pNode1` und `pNode2` existiert, sonst liefert die Anfrage `false`.

Anfrage **`void removeEdge(GraphNode pNode1,GraphNode pNode2)`**
Falls `pNode1` und `pNode2` nicht `null` sind und eine Kante zwischen `pNode1` und `pNode2` existiert, wird die Kante gelöscht. Sonst bleibt der Graph unverändert.

Anfrage **`double getEdgeWeight(GraphNode pNode1, GraphNode pNode2)`**
Die Anfrage liefert das Gewicht der Kante zwischen `pNode1` und `pNode2`. Falls die Kante nicht existiert, wird `Double.NaN` (not a number) zurück gegeben.

Auftrag **`void resetMarks()`**
Alle Knoten des Graphen werden als unmarkiert gekennzeichnet.

Anfrage **`boolean allNodesMarked()`**
Die Anfrage liefert den Wert `true`, wenn alle Knoten des Graphen markiert sind, sonst liefert sie den Wert `false`. Wenn der Graph leer ist, wird `true` zurückgegeben.

Anfrage **`List getNodes()`**
Die Anfrage liefert eine Liste, die alle Knoten des Graphen enthält.

Anfrage **`List getNeighbours(GraphNode pNode)`**
Die Anfrage liefert eine Liste, die alle Nachbarknoten des Knotens `pNode` enthält.