Datenstrukturen
Übung – SS2020

Abgabe: 24.06. bis 24Uhr.
Abgabeort: Bitte laden Sie Ihre Klasse im OPAL-Bereich "Abgabe_Pflichtaufgabe_7" hoch. Ergänzen Sie bei Ihrer hochgeladenen Klasse die Namen (Vor- und Nachname) aller beteiligten Personen. Sie können Ihre Abgabe bis zur Abgabefrist beliebig oft löschen und erneut einreichen.

Wichtig: Da es sich um eine Prüfungsvorleistung handelt, ist es wichtig, dass jeder Teilnehmer Zugang zu allen notwendigen Informationen hat. Deswegen bitten wir Sie Fragen direkt ins Forum, in den Thread "Fragen zur Pflichtaufgabe 7" zu stellen.

Packen Sie Ihre Klasse in das Package "pvl7_$groupX". Sollten Sie nicht in einer Gruppe arbeiten, wählen Sie den Namen statt "pvl7_$Nachname_$Vorname".
Ihre Klasse soll den Namen "PVL7_$GroupX" bzw. "PVL7_$Nachname_$Vorname" besitzen.

Für die Abgabe: Packen Sie all Ihre Klassen in ein .zip Archiv mit dem Namen "PVL7_GroupX.zip"
[Anmerkung: .rar ist kein .zip]

Prüfungsvorleistung 7 – Graphen I

Implementieren Sie einen ungewichteten, ungerichteten Graph. Ein Graph sei dabei ein Tupel von Knoten V und Kanten E : G=(V,E). Der Graph soll dabei N Knoten besitzen, indiziert von 0 bis N-1. Eine Kante soll eine beidseitige nutzbare Verbindung zwischen paarweisen disjunkten Knoten sein.

Die folgenden Methoden finden Sie im Interface "**Graph_I**", hier finden Sie eine Spezifikation für das Verhalten der Methoden, Beispiele folgen weiter unten.

Java: public PVL7_$groupX(int n)
Python: __init__(self, n: int)

Kreieren Sie einen Graphen, welcher "n" Knoten besitzt. Es soll keine Kante vorhanden sein.

Java: Boolean setEdge(int from, int to)
Python: def setEdge(self, from: int, to : int) -> bool

Erstellen Sie eine neue Kante von "from" nach "to". Geben Sie "true" zurück falls diese Kante neu erstellt wurde. Existiere diese Kante bereits, geben Sie "false" zurück. Solle ("from","to") kein valides Tuple sein, geben Sie "false" zurück.

Java: List<List<Integer>> getEdges()
Python: def getEdges(self) -> List[List[int]]

Geben Sie eine Liste von Listen zurück. Diese i-te Liste soll dabei angeben, welche Knoten zum i-ten Knoten adjazent (verbunden durch eine Kante) sind.

Java: List<List<Integer>> getNGons(int n)
Python: def getNGons (self, n : int) -> List[List[int]]

Gesucht seien alle "n"-gons in dem Graphen. Geben Sie eine Liste zurück, die alle paarweise disjunkten "n"-gons, als Liste von Knoten, enthält. Ein "n"-gon lieg vor, wenn es einen Weg von einem Startknoten der Länge n-1 gibt, welcher genau n paarweise disjunkte Knoten enthält, und dessen letzter Knoten der Startknoten ist (also ein Kreis).

Java: Boolean hasFullCircle()
Python: def hasFullCircle(self) -> bool

Geben Sie "true" zurück, falls es einen Kreis gibt, der alles Knoten des Graphen genau einmal enthält.

Java: List<Integer> getLongestPath(int from, int to)
Python: def getLongestPath(self, from : int, to : int) -> List[int]

Geben Sie eine Liste von Knoten zurück. Die Knoten sollen dabei die Knoten auf dem Längsten Weg von "from" nach "to" sein. "from" und "to" sollen dabei in der Liste enthalten sein. Sollte es keinen Weg geben, geben Sie eine leere Liste zurück.

Beispiel Graph

Java Pseudo Code

```
Graph_I graph = new PVL7_$groupX(4)
graph.setEdge(0,1) //true
graph.setEdge(0,1) //false
graph.setEdge(0,0) //false
graph.setEdge(0,2) //true
graph.setEdge(0,3) //true
graph.setEgde(0,4) //false
graph.setEdge(1,3) //true
graph.setEdge(2,3) //true

graph.getEdges()
//{
//{1,2,3},
//{0,3},
//{0,3},
//{0,1,2}
//}

graph.getNGons(3)
//{
//{0,1,3},
//{0,2,3}
//}

graph.hasFullCircle() // true

graph.getLongestPath(2,1) //{2,0,3,1}
```