Datenstrukturen
Übung – SS2020

Abgabe: 05.07. bis 24Uhr.
Abgabeort: Bitte laden Sie Ihre Klasse im OPAL-Bereich "Abgabe_Pflichtaufgabe_9" 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 9" zu stellen.

Packen Sie Ihre Klasse in das Package "pvl9_$groupX". Sollten Sie nicht in einer Gruppe arbeiten, wählen Sie den Namen statt "pvl9_$Nachname_$Vorname".
Ihre Klasse soll den Namen "PVL9_$GroupX" bzw. "PVL9_$Nachname_$Vorname" besitzen.

Für die Abgabe: Packen Sie all Ihre Klassen in ein .zip Archiv mit dem Namen "PVL9_GroupX.zip"
[Anmerkung: .rar ist kein .zip]

Prüfungsvorleistung 9 – Graphen III

Implementieren Sie einen ungewichteten, gerichteten 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_III**", hier finden Sie eine Spezifikation für das Verhalten der Methoden, Beispiele folgen weiter unten.

Java: public PVL9_$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 source, destin to)
Python: def setEdge(self, source: int, destin : int) -> bool

Erstellen Sie eine neue Kante von " source " nach " destin ". 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. (Achtung **gerichteter** Graph)

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: Boolean hasCircle()
Python: def hasWay(self) -> bool

Geben Sie "Wahr" zurück falls es einen Kreis gibt. Ein Kreis habe mindestens die Länge 2 und besitzt die Form {a-b-c-...-a}.

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

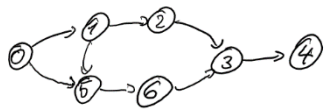
Geben Sie eine Liste zurück, die die Knoten, bzw. Ihre Indizes, in topologischer Sortierung enthält. Gehen Sie davon aus, dass diese Methode nur aufgerufen wird, falls eine topologische Sortierung möglich ist. Die Topologische Sortierung lässt sich wie folgt Charakterisieren. Es Existieren 2 Knoten x,y in der Liste mit den Indizes i,j. (List[i] = x, List[j]=y) Wenn nun gilt i<j, dann soll gelten es gibt keinen Weg von y nach x. Dies soll für Alle Knoten in der Liste gelten.

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

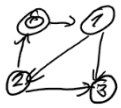
Geben Sie eine Liste von Listen, welche Knoten, bzw. Ihre Indizes, enthält. Zu einer Liste sollen dabei alle Knoten gehören, die zu einen starken Zusammenhangskomponente gehören. Zwei Knoten x, y gehören genau dann zu einer starken Zusammenhangskomponente, wenn gilt:
Es gibt einen Weg von x nach y. && Es gibt einen Weg von y nach x.
Knoten x ist in einer starken Zusammenhangskomponente mit sich selbst

Beispiel Graph

graph_1:



graph_2:



Java Pseudo Code

```
Graph_III graph_1 = new PVL9_$groupX(7);
graph_1.setEdge(0,1);
graph_1.setEdge(1,2);
graph_1.setEdge(2,3);
graph_1.setEdge(3,4);
graph_1.setEdge(0,5);
graph_1.setEdge(1,5);
graph_1.setEdge(5,6);
graph_1.setEdge(6,3);
graph_1.hasCircle();                          //false
graph_1.getStronglyConnectedComponents(); //{{0},{1},{2},{3},{4},{5},{6},{7}}
graph_1.topSort();
//1) {0,1,2,5,6,3,4}
//2) {0,1,5,2,6,3,4}
//3) ...

Graph_III graph_2 = new PVL9_$groupX(4);
graph.setEdge(0,1);
graph.setEdge(1,2);
graph.setEdge(1,3);
graph.setEdge(2,0);
graph.setEdge(2,3);
graph.hasCircle();                          //true
graph.getStronglyConnectedComponents(); //{{0,1,2},{3}}
```