Datenstrukturen
Übung – SS2020

Abgabe: 28.06. bis 24Uhr.
Abgabeort: Bitte laden Sie Ihre Klasse im OPAL-Bereich "Abgabe_Pflichtaufgabe_8" 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 8" zu stellen.

Packen Sie Ihre Klasse in das Package "pvl8_$groupX". Sollten Sie nicht in einer Gruppe arbeiten, wählen Sie den Namen statt "pvl8_$Nachname_$Vorname".
Ihre Klasse soll den Namen "PVL8_$GroupX" bzw. "PVL8_$Nachname_$Vorname" besitzen.

Für die Abgabe: Packen Sie all Ihre Klassen in ein .zip Archiv mit dem Namen "PVL8_GroupX.zip"
[Anmerkung: .rar ist kein .zip]

Prüfungsvorleistung 8 – Graphen II

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_II**", hier finden Sie eine Spezifikation für das Verhalten der Methoden, Beispiele folgen weiter unten.

Java: public PVL8_$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: Boolean hasWay(int source, int destin)
Python: def hasWay(self, source: int, destin: int) -> bool

Geben Sie "Wahr" zurück falls es einen Weg von "source" nach "destin" gibt. Ansonsten geben Sie falsch zurück.

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

Geben Sie "Wahr" zurück falls der Graph zusammenhängend ist (es gibt einen Weg zwischen jedem Knotenpaar), Anonsten geben Sie falsch zurück.

Java: Boolean isConnected(List<Integer> nodes)
Python: def isConnected_biased(self, nodes : List[int]) -> bool

Ihnen wird eine Liste von Knoten Indices gegeben. Benutzen Sie ausschließlich dieser Knoten in Ihrem Graphen um das folgende Problem zu lösen. Geben Sie "Wahr" zurück falls der Graph zusammenhängend ist (es gibt einen Weg zwischen jedem Knotenpaar), Anonsten geben Sie falsch zurück.

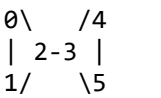
Java: List<List<Integer>> getBridges()
Python: def getBridges(self) -> List[List[int]]

Geben Sie eine Liste von Listen zurück. Die innere Liste soll dabei stets nur 2 Elemente besitzen und stellvertretend für eine Kante sein. Die Äußere Liste soll also eine Liste von Kanten sein. Fügen Sie in diese Liste alle Kanten ein, die wenn man Sie löschen würde, den Zusammenhang des Graphen zerstören würden.

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

Geben Sie eine Liste von Integern zurück. Die Integer sollen dabei die Indices der Knoten sein, welche Artikulationen sind. Ein Knoten ist eine Artikulation, genau dann, wenn man ihn löscht und dies den Zusammenhang des Graphen zerstört.

Beispiel Graph



Java Pseudo Code

```
Graph_II graph = new PVL8_$groupX(6);
graph.setEdge(0,1); //true
graph.setEdge(0,1); //false
graph.setEdge(0,2); //true
graph.setEdge(1,2); //true
graph.setEdge(2,3); //true
graph.setEdge(3,4); //true
graph.setEdge(3,5); //true
graph.setEdge(4,5); //true

graph.getEdges();
//{
//{1,2},
//{0,2},
//{0,1,3},
//{2,4,5},
//{3,5},
//{3,4},
//}

graph.hasWay(0,4);                          //true
graph.isConnected();                        //true
graph.isConnected(Arrays.asList(0,1,3));  //false

graph.getBridges();
//{
//{2,3}
//}

graph.getArticulations(); //{2,3}
```