# Datenstrukturen
# Übung – SS2020

**Abgabe:** 12.07. bis 24Uhr.
**Abgabeort:** Bitte laden Sie Ihre Klasse im OPAL-Bereich "Abgabe_Pflichtaufgabe_10" 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 10" zu stellen.

Packen Sie Ihre Klasse in das Package "pvl10_$groupX". Sollten Sie nicht in einer Gruppe arbeiten, wählen Sie den Namen statt "pvl10_$Nachname_$Vorname".
Ihre Klasse soll den Namen "PVL10_$GroupX" bzw. "PVL10_$Nachname_$Vorname" besitzen.

Für die Abgabe: Packen Sie all Ihre Klassen in ein .zip Archiv mit dem Namen "PVL10_GroupX.zip"
[Anmerkung: .rar ist kein .zip]

## Prüfungsvorleistung 10 – Graphen IV

Implementieren Sie einen gewichteten, 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 nutzbare Verbindung zwischen paarweisen disjunkten Knoten sein.

Die folgenden Methoden finden Sie im Interface "**Graph_IV**", hier finden Sie eine Spezifikation für das Verhalten der Methoden, Beispiele folgen weiter unten. Weiterhin sei eine minimale Implementierung "**Edge**" gegeben, diese wird für die Schnittstelle der Methoden genutzt. Verändern Sie diese Implementierung nicht. Es steht Ihnen jedoch frei intern eine andere Implementierung zu nutzen.

**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, int destin, int weight)
**Python**: def setEdge(self, source: int, destin : int, weight : 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 überschreiben Sie das Gewicht dieser Kante und geben Sie "true" zurück. Solle ("from","to") kein valides Tuple sein, geben Sie "false" zurück. (Achtung **gerichteter** Graph)

**Java**: List<List<Edge>> getEdges()
**Python**: def getEdges(self) -> List[List[Edge]]

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 hasNegativeCycle()
**Python**: def hasNegativeCycle (self) -> bool

Ein Negativer Kreis, sei ein Kreis, dessen Kantengewichte aufsummiert kleiner 0 sind. Geben Sie zurück "true" zurück falls es einen negativen Kreis gibt, ansonsten "false".

**Java**: List<Edge> shortestPath(int source, int destin)
**Python**: def shortestPath(self, source: int, destin: int) -> List[Edge]

Geben Sie Eine Liste von Kanten zurück, welche den kürzesten Weg von source nach destin beschreibt. Sollte es keinen kürzesten Weg oder keinen Weg geben, geben Sie eine leere Liste zurück.

**Java**: Integer universalSink()
**Python**: def universalSink(self) -> int

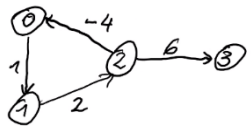
Eine universelle Senke ist ein Knoten. Dieser Knoten kann von allen anderen Knoten erreicht werden (über einen Weg). Doch selbst erreicht er keinen anderen Knoten. Geben Sie den Index der universellen Senke an. Sollte es keine universelle Senke geben, geben Sie -1 zurück.

**Java**: List<List<Edge>> maxEdgeDisjointPaths(int source, int destin)
**Python**: def maxEdgeDisjointPaths (self, source:int, destin: int) -> List[List[Edge]]

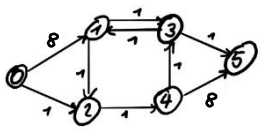
Ein Weg sei Kanten Disjunkt zu einem anderen, wenn Sie keine gleiche Kante besitzen. Geben Sie eine Liste zurück, diese Liste enthält einen Weg repräsentiert als Liste von Kanten. Jeder Weg soll paarweise Kanten Disjunkt sein. Weiterhin soll Ihre Liste die maximale Anzahl an Kanten Disjunkten Wegen enthalten. Sollte es keine Wege geben, geben Sie eine leere Liste zurück.

Beispiel Graph

graph_1:




graph_2:




Java Pseudo Code

```
Graph_IV graph_1 = new PVL10_$groupX(4);
graph_1.setEdge(0,1,1);
graph_1.setEdge(1,2,2);
graph_1.setEdge(2,0,-4);
graph_1.setEdge(2,3,6);

graph_1.hasNegativeCycle();          // True
graph_1.shortestPath(0,3);           // {}
graph_1.universalSink();             // 3
graph_1.maxEdgeDisjointPaths(0,3);   // {{Edge(1,1), Edge(2,2), Edge(3,6)}}

Graph_IV graph_2 = new PVL10_$groupX(6);
graph_2.setEdge(0,1,8);
graph_2.setEdge(0,2,1);
graph_2.setEdge(1,2,1);
graph_2.setEdge(1,3,1);
graph_2.setEdge(2,4,1);
graph_2.setEdge(3,1,1);
graph_2.setEdge(3,5,1);
graph_2.setEdge(4,3,1);
graph_2.setEdge(4,5,8);

graph_2.hasNegativeCycle();      // False
graph_2.shortestPath(0,5);       // {Edge(2,1), Edge(4,1), Edge(3,1), Edge(5,1)}
graph_2.universalSink();         // 5
graph_2.maxEdgeDisjointPaths(0,5);
//{
//{Edge(1,8), Edge(3,1), Edge(5,1)}
//{Edge(2,1), Edge(4,1), Edge(5,8)}
//}
```