Hochschule
Albstadt-Sigmaringen
Albstadt-Sigmaringen University

Studiengänge ITS,TI,WIN
Vorlesung: **Prof.Dr.Derk Rembold**
Praktikum: Ralf Mispelhorn, Dominik Höpfl

Programmieren 1

Aufgabenblatt 7

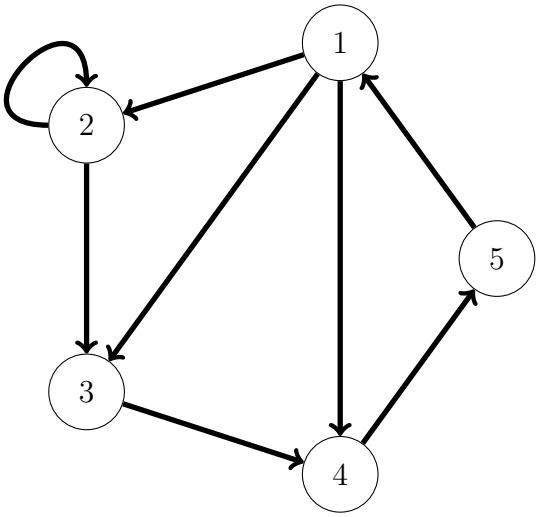
Abgabe: Die Lösungen sollten zeitnah als IPYNB-Datei fertiggestellt werden. Der Code muss ausreichend kommentiert sein und die Variablen müssen sinnvoll benannt werden. Sie müssen die Aufgabe selbst programmiert haben. Sie können Fragen in Form von Kommentaren im Code stellen, falls etwas nicht funktioniert hat. Die Antwort erfolgt dann im Praktikum mündlich. Sie dürfen nie mehr als drei Aufgabenblätter im Rückstand sein.

Hilfsmittel: Kein Copy-Paste aus dem Internet, alles muss selbstständig programmiert sein. Sie dürfen die IPython-Notebook-Skripte aus der Vorlesung (liegen nach der jeweiligen Vorlesung auf Ilias) und die Python-Einführung auf Ilias verwenden. Außerdem ist die Hilfe-Funktion `help(...)` und die Methode `dir(...)` zur Auflistung der verfügbaren Funktionen zu empfehlen.

Anwesenheit: Grundsätzlich herrscht Anwesenheitspflicht. Ein Attest ist notwendig, wenn jemand nicht kommen kann. Bei unentschuldigtem Fehlen ist das Praktikum nicht bestanden. Von der Teilnahme an der Klausur wird in diesem Fall dringend abgeraten.

Aufgabe 7.0

Ein Graph ist ein mathematisches Objekt, bestehend aus *Knoten* und Verbindungen zwischen Knoten, genannt *Kanten*. Man kann einen Graphen als eine sogenannte Adjazenzliste repräsentieren, d.h. indem man jedem Knoten des Graphen die Liste der Nachbarknoten (mathematisch gesprochen: die Liste der Adjazenten Knoten) zuordnet.



(a) Erstellen Sie eine Klasse `Graph`, die einen Graphen repräsentiert dessen Knoten mit Werten beliebigen Typs „beschriftet“ sind. Erstellen Sie eine Methode `addNode(k)`, die einen neuen

Knoten `k` in den Graphen einfügt. Sollte es den Knoten bereits geben, so soll einfach nichts passieren.

(b) Erstellen Sie eine Methode `addEdge(k,j)`, die eine neue Kante von Knoten `k` nach Knoten `j` einfügt. Sollte einer der Knoten `k` oder `j` nicht existieren oder die Kante bereits existieren, so soll einfach nichts passieren.

(c) Erstellen Sie eine Methode `V()`, die eine Liste aller Knoten eines Graphen zurückliefert.

(d) Erstellen Sie eine Methode `E()`, die eine Liste aller Kanten des Graphen zurückliefert; eine Kante besteht aus zwei Teilen: dem Knoten, von dem die Kante ausgeht, und die dem Knoten, an dem die Kante endet.

(e) Erstellen Sie eine Methode `allSingles()`, die die Liste aller Knoten zurückliefert, die keine Nachbarn haben.

(f) Erstellen Sie eine Methode `mostEdges()`, die den Knoten mit den meisten Kanten zurückliefert.

(g) Erstellen Sie eine Methode `neighbors(v)`, die eine Sequenz der Nachbarn eines Knotens zurückliefert.

Den in der Abbildung dargestellten Graphen sollte man dann folgendermaßen erstellen können:

```
>>> g = Graph()
>>> for i in range(1,6): g.addNode(i)
>>> for i,j in [(1,2),(2,2),(1,3),(2,3),(3,4),(1,4),(4,5),(5,1)]:
        g.addEdge(i,j)
>>> g.V()
[1,2,3,4,5]
```

Aufgabe 7.1

Implementieren Sie eine Klasse `Cntr`, mit der man eine Häufigkeitsanalyse für eine Menge von Werten durchführen kann. Verwenden Sie hierzu **nicht** die Pythonklasse `Counter`.

(a) Implementieren Sie den Konstruktor so, dass er einen beliebigen Sequenz-Wert übergeben bekommt (etwa eine Liste, ein Tupel, ein String, ...) und daraus ein Counter-Objekt erzeugt.

(b) Implementieren Sie eine `repr`-Funktion, um das Objekt ausgeben zu können.

```
>>> c = Cntr("Hallo Welt")
>>> c
'H' --> 1
'W' --> 1
'a' --> 1
'l' --> 3
'o' --> 1
'e' --> 1
't' --> 1
' ' --> 1
```

(c) Implementieren Sie eine Methode `most`, die das Element zurückliefert, das am häufigsten vorkommt.

```
>>> c.most()
'l'
```

(d) Implementieren sie die +-Operation für zwei Counter-Objekte. Das Ergebnis soll ein neues Counter-Objekt sein.

```
>>> c2 = Cntr("Hello World")
>>> c + c2
'H' --> 2
'a' --> 1
'l' --> 6
'o' --> 3
'r' --> 1
'e' --> 2
't' --> 1
' ' --> 2
'd' --> 1
'W' --> 2
```

(e) Implementieren sie die *-Operation für zwei Counter-Objekte. Das Ergebnis soll ein neues Counter-Objet sein.

```
>>> c * c2
'H' --> 1
'a' --> 0
'l' --> 9
'o' --> 2
'e' --> 1
't' --> 0
' ' --> 1
'd' --> 0
'W' --> 1
'r' --> 0
```