5. Programmieraufgabe Computerorientierte Mathematik I

Abgabe: 18.12.2020 über den Comajudge bis 17 Uhr

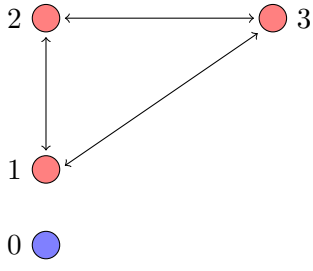
Bitte beachten Sie: Die Herausgabe oder der Austausch von Code (auch von Teilen) zu den Programmieraufgaben führt für *alle* Beteiligten zum *sofortigen Scheinverlust*. Die Programmieraufgaben müssen von allen Teilnehmenden alleine bearbeitet werden. Auch Programme aus dem Internet dürfen nicht einfach kopiert werden.

1 Problembeschreibung

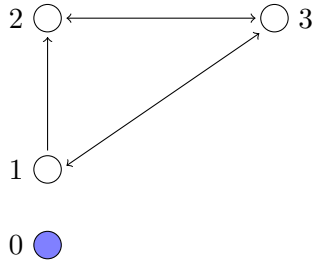
Sei $G=(V,E)$ ein einfacher Digraph mit $V=\{0,\ldots,n-1\}$. Wir setzen

$$\Delta:=\{(v_1,v_2)\in V\times V\colon v_1=v_2\}.$$

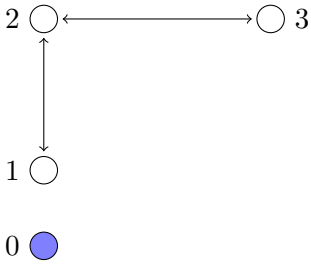
Es soll eine Funktion `get_eqclasses` geschrieben werden, die entscheidet, ob $E\cup\Delta$ eine Äquivalenzrelation auf V beschreibt, d.h. ob die durch $E\cup\Delta$ auf V gegebene Relation *reflexiv*, *symmetrisch* und *transitiv* ist. Gegebenenfalls soll die induzierte Partition von V berechnet werden. Die einzelnen Klassen der Rückgabe dürfen in einer beliebigen Reihenfolge vorliegen. In der folgenden Skizze steht ein Pfeil $x\longleftrightarrow y$ mit zwei Spitzen für gerichtete Kanten $x\longrightarrow y$ und $y\longrightarrow x$.



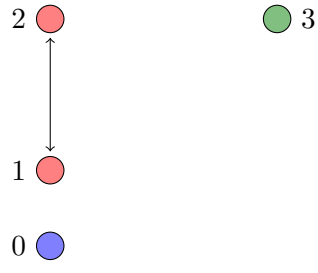
(a) Output: [[0],[1,2,3]]



(b) Output: [ ]



(c) Output: [ ]



(d) Output: [[0],[3],[1,2]]

2 Aufgabenstellung und Anforderungen

Schreiben Sie eine Funktion

`get_eqclasses(n,E)` ,

die entscheidet, ob die Menge $E \cup \Delta$ eine Äquivalenzrelation auf $V := \{0,...,n-1\}$ beschreibt und die ggf. die zugrundeliegende Partition von V als Liste von Knotenlisten zurückgibt.
Implementieren und verwenden Sie dafür folgende Funktionen:

- Die Funktion `get_classes(n,E)` gibt eine Liste mit n Einträgen zurück, wobei an k-ter Stelle ($k=0,...,n-1$) die k-te *Nachbarschaft* $[k] = \{w \in V : (k,w) \in E \cup \Delta\}$ als ungeordnete Knotenliste steht.
- Die Funktion `are_equal(list1,list2)` soll für zwei Listen ganzer Zahlen `True` zurückgeben, falls die zugrundeliegenden *Mengen* ganzer Zahlen übereinstimmen. Andernfalls soll `False` zurückgegeben werden.
- Die Funktion `are_disjoint(list1,list2)` soll für zwei Listen ganzer Zahlen `True` zurückgeben, falls die zugrundeliegenden *Mengen* ganzer Zahlen disjunkt sind. Andernfalls soll `False` zurückgegeben werden.

2.1 Eingabe

Der Funktion `get_eqclasses(n,E)` wird eine ganze Zahl $n \in \mathbb{Z}_{\geq 0}$ für die Knotenmenge $V := \{0,...,n-1\}$ und gerichtete Kanten $E \subset V \times V$ als Liste übergeben.

2.2 Rückgabewert

Falls $E \cup \Delta$ eine Äquivalenzrelation $\sim$ auf V beschreibt, dann soll eine Liste L von Knotenlisten $L_1,...,L_k$ zurückgegeben werden ($k = |V/\sim|$), wobei L_i die Klassen von $\sim$ durchläuft. Alle Listen dürfen in ungeordneter Form vorliegen.
Falls $E \cup \Delta$ keine Äquivalenzrelation beschreibt, dann soll die leere Liste `[]` zurückgegeben werden.

2.3 Beispielaufrufe

```
>>> get_eqclasses(4,[(1,2),(2,1),(3,1),(1,3),(2,3),(3,2)])
[[0], [1, 2, 3]]
>>> get_eqclasses(4,[(1,2),(3,1),(1,3),(2,3),(3,2)])
[]
>>> get_eqclasses(4,[(1,2),(2,1),(2,3),(3,2)])
[]
>>> get_eqclasses(4,[(1,2),(2,1)])
[[0], [1, 2], [3]]
```

Dies sind genau die Aufrufe, die zu den Beispielen in Abschnitt 1 gehören.

3 Tipps und Anmerkungen

- Für eine Liste L ganzer Zahlen liefert `set(L)` die Menge ihrer Einträge. Beispielsweise erhält man `set([1,2,3])`=$\{$`1,2,3`$\}$ und `set([1,2,2])`=$\{$`1,2`$\}$. Sofern Sie diese Funktion benutzen - welche Voraussetzung erlaubt Ihnen das?

3.1 Lemma für die Programmieraufgabe

Es sei R eine reflexive Relation auf einer Menge M, d.h. für alle $x \in M$ gilt xRx. Für jedes $m \in M$ setzen wir $[m] := \{x \in M : mRx\}$. Beweist folgende Aussagen:

a) Die Relation R beschreibt genau dann eine Äquivalenzrelation, wenn für alle $x,y \in M$ entweder $[x]=[y]$ oder $[x] \cap [y] = \emptyset$ gilt.

b) Sei $G=(V,E)$ ein einfacher Digraph mit $V=\{0,...,n-1\} \subset \mathbb{N}$. Wir setzen $\Delta := \{(v_1,v_2) \in V \times V : v_1 = v_2\}$. Entwerft einen Algorithmus, der entscheidet, ob $E \cup \Delta$ eine Äquivalenzrelation auf V beschreibt und der ggf. die Äquivalenzklassen berechnet.

Lösung:

a) „$\Rightarrow$ “: Sei R eine Äquivalenzrelation. Wegen der Reflexivität sind $[x]$ und $[y]$ nichtleer, also können sie nicht sowohl gleich als auch disjunkt sein. Es bleibt zu zeigen, dass aus $[x] \cap [y] \neq \emptyset$ die Gleichheit folgt. Seien $x,y \in M$ mit $[x] \cap [y] \neq \emptyset$. Dann gibt es $s \in [x] \cap [y]$ und wir erhalten aus xRs und yRs mit der Symmetrie, dass sRy und mit der Transitivität, dass xRy. Sei weiter $z \in [y]$ beliebig. Dann gilt yRz und mit xRy und der Transitivität folgt xRz, also $z \in [x]$. Damit ist $[y] \subseteq [x]$ gezeigt, die andere Inklusion geht analog.

„$\Leftarrow$ “: Für alle $x,y \in M$ gelte entweder $[x]=[y]$ oder $[x] \cap [y] = \emptyset$.

- *Transitivität:* Es gelte xRy und yRz. Aufgrund der (vorausgesetzten) Reflexivität gilt $y \in [y]$ und die Voraussetzung liefert $[x]=[y]$. Analog folgt $[y]=[z]$ und erneut aufgrund der Reflexivität xRz.
- *Symmetrie:* Es gelte xRy. Wie bei der Transitivität erhalten wir $[x]=[y]$ und wegen der Reflexivität $y \in [y] = [x]$, also yRx.

b) **Eingabe:** Liste `V` von Knoten, Liste `E` von 2-Tupeln (Kanten).
Wir erstellen eine Liste L, deren Elemente die Mengen $[v]$ für $v \in \{1,...,n\}$ sind.

```
1        L=[[v] for v in V]
2        for e in E:
3        L[e[0]].append(e[1])
```

Jetzt können wir prüfen, ob sich die Listen $[v]$ nichttrivial schneiden, d.h. ob für $v,w \in V$ weder $[v]=[w]$ noch $[v] \cap [w] = \emptyset$ gilt. Nach Aufgabenteil a) ist dies äquivalent dazu, nachzuprüfen, ob $E \cup \Delta$ eine Äquivalenzrelation ist.

```
1        for i in range(n):
2        L[i].sort()
3        for i in range(n):
4        for j in range(i+1,n):
5        if not set(L[i]).isdisjoint(L[j]) and L[i]!=L[j]:
6        return None
```

Wenn die Funktion bis hierhin noch nicht terminiert hat, handelt es sich bei $E \cup \Delta$ um eine Äquivalenzrelation. Wir müssen noch die Dubletten unter den Äquivalenzklassen entfernen. Hierbei nutzen wir, dass wir die entsprechenden Listen bereits sortiert haben.

```
1        L.sort()
2        last_first_index=-1
3        equivalence_classes=[]
4        for i in range(n):
5        if L[i][0]!=last_first_index:
6        equivalence_classes.append(L[i])
7        last_first_index=L[i][0]
8        return equivalence_classes
```