Aufgabe 1: Zimmerverteilung

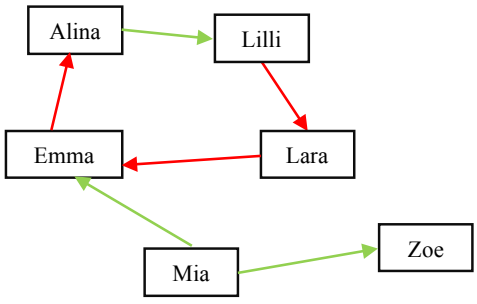
Aufgabe

Schreibe ein Programm, das ermittelt, ob alle Wünsche erfüllt werden können, wenn es genug Zimmer jeder Größe gibt. Als Eingabe erhält es für jede Schülerin zwei Listen der Mitschülerinnen, mit denen sie auf jeden Fall (+) bzw. auf keinen Fall (–) ein Zimmer teilen möchte.

Dein Programm soll ausgeben, ob eine Zimmerbelegung möglich ist, die alle Wünsche erfüllt. Falls ja, soll es zusätzlich eine solche Zimmerbelegung ausgeben.

Lösungsidee

Die Liste der Mädchen lässt sich als ein einfacher gewichteter gerichteter Graph darstellen. Beispielsweise kann man „zimmerbelegung2.txt“ wie folgt visualisieren:



grüner Pfeil (Kantengewicht: 1) von A zu B:
A will auf jeden Fall ein Zimmer mit B teilen

roter Pfeil (Kantengewicht: -1) von A zu B:
A will auf keinen Fall ein Zimmer mit B teilen

Abbildung 1: gewichteter gerichteter Graph

Diesen Graph kann man auch durch zwei ungerichtete Graphen repräsentieren. Denn es ist egal ob Person A mit Person B in ein Zimmer will oder ob der Wunsch von Person B ausgeht. In beiden Fällen müssen A und B in das gleiche Zimmer eingeteilt werden, da sonst nicht alle Wünsche erfüllt werden.

Falls Person A ein Zimmer mit Person B teilen will, Person B aber auf keinen Fall mit A in einem Zimmer sein will, ist eine Zimmeraufteilung offensichtlich nicht möglich und das Programm kann beendet werden.

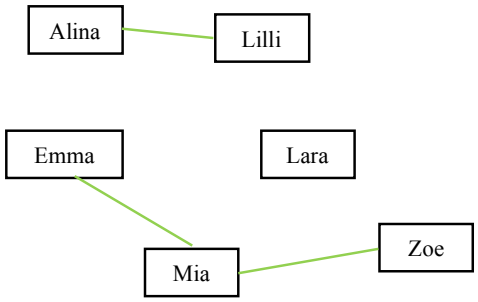


Abbildung 2: $G_1 = (V, E_1)$

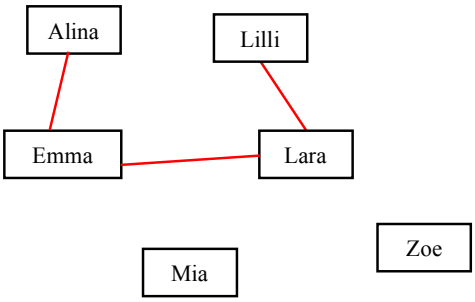


Abbildung 3: $G_2 = (V, E_2)$

Mit der Breitensuche (BFS) lassen sich hierauf die Zusammenhangskomponenten des Graphen G_1 bestimmen. (Eine Zusammenhangskomponente ist ein maximaler zusammenhängender Teilgraph eines Graphen) In unserem Beispiel hat Graph G_1 drei Zusammenhangskomponenten. In der einen sind die Schülerinnen Alina und Lilli, in der anderen Emma, Mila und Zoe und in der dritten ist Lara. Logischerweise müssen alle Personen, die in der gleichen Komponente sind, in ein einziges Zimmer, da, sobald man eine beliebige Schülerin entfernt, der Wunsch von mindestens einer Person nicht mehr erfüllt wird.

Um zu überprüfen, ob eine Zimmerverteilung, in der die Wünsche aller Personen erfüllt werden, überhaupt möglich ist, muss Graph G_2 betrachtet werden. Besteht in diesem Graphen eine direkte Verbindung zwischen zwei Schülerinnen, dürfen diese nicht in dasselbe Zimmer eingeteilt werden. Ist diese Bedingung nicht erfüllt, muss das Programm zurückgeben, dass keine perfekte Zimmereinteilung möglich ist. Wenn G_1 und G_2 nicht im Konflikt stehen, ist eine Zimmerverteilung möglich und das Programm gibt die wie vorhin beschrieben, mit der Breitensuche gefundene Zimmerverteilung aus.

Umsetzung

Die Lösungsidee wird in Python 3.6 implementiert.

Die Wunschliste der Schülerinnen wird zur einfacheren Verarbeitung in eine Adjazenzmatrix umformatiert.

Abbildung 4: Liste aus zimmerbelegung2.txt

Abb. 4 kann nun durch eine Adjazenzmatrix dargestellt werden:

$$M = \begin{array}{c} \begin{matrix} \text{Alina} & \text{Emma} & \text{Lara} & \text{Lilli} & \text{Mia} & \text{Zoe} \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \quad \begin{array}{ll} \text{Alina} & \textit{Alina will mit Lilli ein Zimmer teilen} \\ \text{Emma} & \textit{Emma will mit Alina kein Zimmer teilen} \\ \text{Lara} & \textit{Lara will mit Emma kein Zimmer teilen} \\ \text{Lilli} & \dots \\ \text{Mia} & \dots \\ \text{Zoe} & \dots \end{array}$$

Da wir wissen, dass dieser gerichtete Graph in einen ungerichteten umgewandelt werden kann, können wir die Matrix jetzt „spiegeln“.

$$M = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

wird zu

$$M' = \begin{pmatrix} 0 & -1 & 0 & 1 & 0 & -1 \\ -1 & 0 & -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Hierbei gilt:

$$M_{i,j} = \begin{cases} M_{i,j} & wenn\ M_{i,j} \neq 0 \\ M_{j,i} & wenn\ M_{i,j} = 0 \end{cases}$$

Wenn $(M_{i,j} = -1 \wedge M_{j,i} = 1) \vee (M_{i,j} = 1 \wedge M_{j,i} = -1)$ bricht das Programm ab, da hier auf keinen Fall alle Wünsche erfüllt werden können.

Daraufhin wird M' in zwei Adjazenzlisten konvertiert. Eine für das positive Beziehungsgeflecht und eine für das negative. (Dieser Schritt ist nicht zwingend nötig. Man kann BFS auch an einer Matrix anwenden.)

$$M' = \begin{pmatrix} 0 & -1 & 0 & 1 & 0 & -1 \\ -1 & 0 & -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$L_{pos} := \begin{matrix} 0\colon [3] \\ 1\colon [4] \\ 2\colon [] \\ 3\colon [0] \\ 4\colon [1,5] \\ 5\colon [4] \end{matrix} \qquad L_{neg} := \begin{matrix} 0\colon [1,5] \\ 1\colon [0,2] \\ 2\colon [1,3] \\ 3\colon [2] \\ 4\colon [] \\ 5\colon [0] \end{matrix}$$

Die Breitensuche wird nun an einem Beispiel (zimmerverteilung2.txt) erklärt:

Zu Beginn haben wir eine leere Liste $room = [\,]$, in der alle Personen, die gemeinsam in einen Raum kommen gespeichert werden sollen und eine Queue q. Solange nicht alle Knoten besucht wurden wird entweder Schritt (a) oder Schritt (b) angewandt.

(a) Wenn q leer ist, wird aus L_{pos} der erste noch nicht besuchte Knoten $node$ ausgewählt (am Anfang ist die Erste unbesuchte Person in L_{pos} natürlich diejenige mit Index 0, also Alina) und in $room$ und q hinzugefügt.

$q = [0]$
$room = [0]$

(b) Wenn $len(q) > 0$ ist, wird das erste Element aus der Queue herausgenommen $node = q.pop(0)$ und alle Elemente aus $L_{pos}[node]$ werden in $room$ und q hinzugefügt.
Wenn die Schnittmenge zwischen $L_{neg}[node]$ und $room$ eine Mächtigkeit > 1 besitzt. Wird das Programm abgebrochen, da keine perfekte Zimmeraufteilung möglich ist.

$node = q.pop(0) \rightarrow node = 0$
$L_{pos}[node] = L_{pos}[0] = [3]$
$q = [3]$
$room := [0, 3]$

Da $len(q) = 1$ wird (b) angewandt:

$node = q.pop(0) \rightarrow node = 3$
$L_{pos}[node] = L_{pos}[3] = []$
$q = []$
$room := [0, 3]$

Da $len(q) = 1$ wird (a) angewandt:

$q = [1]$
$room := [1]$

Da $len(q) = 1$ wird (b) angewandt:

$node = q.pop(0) \rightarrow node = 1$
$L_{pos}[node] = L_{pos}[1] = [4]$
$q = [4]$
$room := [1, 4]$

Da $len(q) = 1$ wird (b) angewandt:

$node = q.pop(0) \rightarrow node = 4$
$L_{pos}[node] = L_{pos}[4] = [1, 5]$
$q = [5]$ *// 1 wurde schon besucht und wird deswegen nicht hinzugefügt*
$room := [1, 4, 5]$

Da $len(q) = 1$ wird (b) angewandt:

$node = q.pop(0) \rightarrow node = 5$
$L_{pos}[node] = L_{pos}[5] = [4]$
$q = []$ *// 4 wurde schon besucht und wird deswegen nicht hinzugefügt*
$room := [1, 4, 5]$

Da $len(q) = 0$ wird (a) angewandt:

$q = [2]$
$room := [2]$

Da $len(q) = 1$ wird (b) angewandt:

$node = q.pop(0) \rightarrow node = 2$
$L_{pos}[node] = L_{pos}[2] = []$
$q = []$
$room := [2]$

Nachdem alle Knoten besucht wurden können in den $room$ Listen die Indies durch die Namen der Schülerinnen ausgetauscht werden.
Die Ausgabe für $./zimmerverteilung.py\ txt/zimmerbelegung2.txt$ ist demnach:

```
['Alina', 'Lilli']
['Emma', 'Mia', 'Zoe']
['Lara']
```

Beispiele

$ ./zimmerbelegung.py txt/zimmerbelegung1.txt

```
Zimmeraufteilung nicht möglich!
```

$ ./zimmerbelegung.py txt/zimmerbelegung2.txt

```
['Alina', 'Lilli']
['Emma', 'Mia', 'Zoe']
['Lara']
```

$ ./zimmerbelegung.py txt/zimmerbelegung3.txt

```
['Alina', 'Annika', 'Josephine', 'Katharina', 'Kim', 'Leonie',
'Lilli', 'Melina', 'Pauline', 'Pia', 'Sarah', 'Sophie',
'Vanessa']
['Anna', 'Antonia', 'Carolin', 'Emily', 'Emma', 'Jana',
'Johanna', 'Julia', 'Larissa', 'Lisa', 'Marie', 'Michelle',
'Nele', 'Sofia']
['Celina', 'Charlotte', 'Jasmin', 'Jessika', 'Lara', 'Laura',
'Luisa', 'Merle', 'Miriam', 'Nina']
['Celine', 'Clara', 'Lea', 'Lena', 'Lina']
['Hannah']
```

$ ./zimmerbelegung.py txt/zimmerbelegung1.txt

```
['Alina', 'Emily', 'Jana', 'Johanna', 'Josephine', 'Katharina',
'Larissa', 'Laura', 'Lea', 'Lena', 'Leonie', 'Lilli', 'Marie',
'Melina', 'Sarah', 'Sofia', 'Sophie']
['Anna', 'Jasmin', 'Jessika', 'Julia', 'Miriam', 'Pauline']
['Annika', 'Carolin', 'Celina', 'Celine', 'Charlotte', 'Clara',
'Emma', 'Hannah', 'Kim', 'Lara', 'Lina', 'Lisa', 'Luisa',
'Merle', 'Michelle', 'Nele', 'Nina', 'Pia', 'Vanessa']
['Antonia']
```

$ ./zimmerbelegung.py txt/zimmerbelegung1.txt

```
['Alina'], ['Anna'], ['Annika'], ['Antonia'], ['Carolin'],
['Celina'], ['Celine'], ['Charlotte'], ['Clara'], ['Emma'],
['Jana'], ['Jasmin'], ['Jessika'], ['Josephine'], ['Julia'],
['Katharina'], ['Larissa'], ['Laura'], ['Lea'], ['Lina'],
['Marie'], ['Melina'], ['Merle'], ['Michelle'], ['Miriam'],
['Nele'], ['Nina'], ['Pia'], ['Sofia'], ['Sophie'], ['Vanessa'],
['Lilli']
['Lara', 'Pauline']
['Kim', 'Leonie', 'Luisa']
['Emily', 'Hannah', 'Johanna', 'Lena', 'Lisa', 'Sarah']
```

(Ausgabe verschönert: viele Zeilenumbrüche entfernt, Einzelzimmer gruppiert)

```
$ ./zimmerbelegung.py txt/zimmerbelegung1.txt
['Alina', 'Anna', 'Antonia', 'Carolin', 'Charlotte', 'Clara',
'Emma', 'Jessika', 'Josephine', 'Julia', 'Katharina', 'Kim',
'Lara', 'Laura', 'Lena', 'Leonie', 'Lina', 'Lisa', 'Luisa',
'Marie', 'Miriam', 'Nele', 'Nina', 'Pauline', 'Pia']
['Annika', 'Celina']
['Celine', 'Emily', 'Hannah', 'Jana', 'Johanna', 'Larissa',
'Lilli', 'Melina', 'Sarah', 'Sophie', 'Vanessa']
['Jasmin', 'Merle', 'Sofia']
['Lea', 'Michelle']
```