Wilhelmsgymnasium München Jahrgang 2016/18

Seminararbeit

im W-Seminar „Alles ist Mathematik"

Leitfach: Mathematik

Bundeswettbewerb Informatik

von
Anton Baumann

Betreuende Lehrkraft: StD Gert Hartbauer

Abgabetermin: 07.11.2017

Erzielte Note:

Erzielte Punkte:

_____

Datum, Unterschrift des Seminarleiters/der Seminarleiterin

Gliederung

Aufgaben

Quellcode

Aufgabe 1: Zimmerverteilung

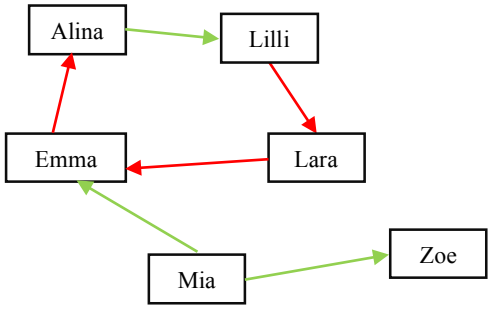
Aufgabe

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.

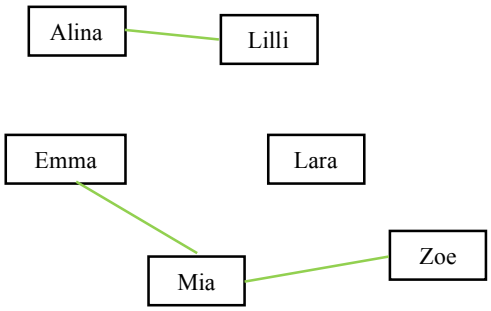


Abbildung 2: $G_1 = (V, E_1)$

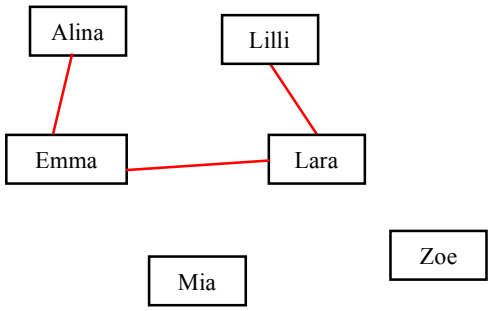


Abbildung 3: $G_2 = (V, E_2)$

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:

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

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']
```

Aufgabe 2: Schwimmbad

Aufgabe

Gegeben sind die Datumsmerkmale „Wochentag oder Wochenende“ und „Schulzeit oder Ferien“, eine Anzahl von Gutscheinen und eine Liste von Personen mit Altersangaben. Schreibe ein Programm, das berechnet, wie viel Geld mindestens gezahlt werden muss, damit die ganze Gruppe, falls möglich, das Schwimmbad betreten kann. Etwaige Vorteile für künftige Besuche sollen nicht berücksichtigt werden. Das Programm soll die zu kaufenden Karten und den Gesamtpreis ausgeben. Wende dein Programm auf das obige Beispiel an und auf alle weiteren Beispiele, die du auf den BwInf-Webseiten findest.

Lösungsidee

Um für eine Gruppe von Personen die beste Kombination von Tages-, Familien- und Einzelkarten herauszufinden, werden alle Kombinationen durchprobiert und die mit dem geringsten Preis zurückgeben. Wenn die Gruppe Gutscheine besitzt, werden diese bei jeder möglichen Kombination von Karten so lange für Einzelkarten (zuerst für Erwachsene, dann für Jugendliche) eingelöst, bis es keine bezahlten Einzelkarten mehr gibt oder nur noch ein Gutschein übrig ist. Je nachdem, wodurch die Kosten mehr verringert werden, wird nun der letzte Gutschein entweder für eine Reduzierung des Gesamtpreises um zehn Prozent oder für den kostenlosen Eintritt einer Einzelperson verwendet. Diese neu errechnete Kombination wird der Liste der bereits existierenden Kombinationen hinzugefügt.
Den besten Preis erhält man, indem man aus der Liste von Kombinationen diejenige mit dem geringsten Preis auswählt. Dieser Ansatz liefert zwar immer das korrekte Ergebnis, ist aber sehr ineffizient. Der Algorithmus hat für einige Fälle eine sehr hohe Laufzeit und einen hohen Speicherbedarf, da er ausnahmslos alle Kombinationen speichert. Bei Anwendung der Beispiele auf der BwInf-Seite terminiert der Algorithmus nach rel. kurzer Zeit.

Umsetzung

Das Programm besteht aus zwei Zeilen. Im ersten werden alle Kombinationsmöglichkeiten erzeugt und einer Liste gespeichert.

Dafür wird ein Stack zu Hilfe genommen.

Die Kombinationen für 2 Erwachsene *E* und 4 Jugendliche *J* an Wochentagen berechnet das Programm folgendermaßen:

Die Funktion *prices.prices(2, 4, we=False)* gibt in etwa folgende Preistabelle zurück:

Tageskarte	(2E,4J)
Familienkarte	(2E,2J)
Familienkarte	(1E,3J)
Einzelkarte	(1E,0J)
Einzelkarte	(0E,1J)

Es wird immer die Karte aus der ersten nicht besuchten Reihe ausgewählt. Darauf werden die verbleibenden Personen berechnet:

(2E, 4J) abzüglich (2E, 4J) ergibt (0E, 0J)

Wenn keine Person übrigbleibt, wird die gerade erzeugte Kombination in einer Liste gespeichert.

Das Programm sucht nun die nächste unbesuchte Reihe heraus und berechnet, wie viele Personen übrigbleiben:

Tageskarte	(2E,4J)
Familienkarte	(2E,2J)
Familienkarte	(1E,3J)
Einzelkarte	(1E,0J)
Einzelkarte	(0E,1J)

(2E, 4J) - (2E, 2J) = (0E, 2J)

Nun wird *prices.prices(0, 2, we=False)* aufgerufen. Wir bekommen folgende Tabelle:

Einzelkarte	(0E,1J)

Das Programm sucht nun in dieser die erste unbesuchte Reihe heraus und berechnet, wie viele Personen nach dem Kauf dieser Karte übrigbleiben:

Einzelkarte	(0E,1J)

(0E, 2J) - (0E, 1J) = (0E, 1J)

Das Programm folgt dieser Vorgehensweise solange, bis alle Kombinationen berechnet wurden.
Die Funktion des Stacks ist im Grunde die gleiche wie in Aufgabe 4 und wird dort auch genauer beschrieben.
Das Programm kennt jetzt alle Kombinationsmöglichkeiten und hat diese in einer Liste gespeichert. Wenn die Gruppe keine Gutscheine hat oder das Schwimmbad in den Ferien besucht, kann nun die günstigste Kombination zurückgegeben werden. Falls nicht werden die Gutscheine wie bereits in der Lösungsidee beschrieben eingelöst.

Beispiele

1.

Antonia hat drei Freundinnen. Ihre Mutter und ihre zweijährige Schwester möchten auch mitkommen. Sie wollen an einem Wochenende in den Ferien ins Schwimmbad gehen und besitzen einen Gutschein. Wie viel müssen sie bezahlen?

```
./schwimmbad_brute.py t t 1 4 1 1

Bester Preis wird berechnet für:
Wochenende:  True
Ferien:      True
Erwachsene:  1
Jugendl.:    4
Kinder:      1
Gutscheine:  1

1050
+------+------+--------+----------------+-------+
| Erw. | Jug. | Kosten | Kosten einzeln | index |
+======+======+========+================+=======+
| 1    | 3    | 800    | 1100           | 0.727 |
+------+------+--------+----------------+-------+
| 0    | 1    | 250    | 250            | 1     |
+------+------+--------+----------------+-------+
```

2.

Eine Abiturklasse mit 26 Schülerinnen und Schülern und einer Lehrerin wollen an einen Wochentag das Schwimmbad besuchen. Sie haben drei Gutscheine und wollen eigentlich an einem beliebigen Tag in den Ferien gehen. Wenn sie aber

dadurch mindestens €5 sparen können, würden sie den Besuch auf die Schulzeit verschieben, der dann aber am Wochenende stattfinden müsste.
Erwachsene: 27, Gutscheine: 3

a) Wochentag in den Ferien

./schwimmbad_brute.py f t 27 0 0 3

```
Bester Preis wird berechnet für:
Wochenende:  False
Ferien:      True
Erwachsene:  27
Jugendl.:    0
Kinder:      0
Gutscheine:  3

5240.0
+------+------+--------+----------------+-------+
| Erw. | Jug. | Kosten | Kosten einzeln | index |
+======+======+========+================+=======+
| 6    | 0    | 1100   | 1680           | 0.655 |
+------+------+--------+----------------+-------+
| 6    | 0    | 1100   | 1680           | 0.655 |
+------+------+--------+----------------+-------+
| 6    | 0    | 1100   | 1680           | 0.655 |
+------+------+--------+----------------+-------+
| 6    | 0    | 1100   | 1680           | 0.655 |
+------+------+--------+----------------+-------+
| 1    | 0    | 280    | 280            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 280    | 280            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 280    | 280            | 1     |
+------+------+--------+----------------+-------+
```

b) Wochenende in der Schulzeit

./schwimmbad_brute.py t f 27 0 0 3

```
Bester Preis wird berechnet für:
Wochenende:  True
Ferien:      False
Erwachsene:  27
Jugendl.:    0
Kinder:      0
Gutscheine:  3


7875.0 mit Gruppengutschein
+------+------+--------+----------------+-------+
| Erw. | Jug. | Kosten | Kosten einzeln | index |
+======+======+========+================+=======+
| 1    | 0    | 0      | 0              | 0     |
+------+------+--------+----------------+-------+
| 1    | 0    | 0      | 0              | 0     |
+------+------+--------+----------------+-------+
```

1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1

Der Besuch während der Ferien ist deutlich günstiger als während der Schulzeit.

3.

Die Großfamilie Stutzenberg besteht aus Anton und Gerda, die eine Tochter Amalie haben. Diese ist mit Gerhard verheiratet und hat mit ihm die fünf Kinder Bobo (3), Zoe (7), Bibi(10), Josefine (14) und Josef (17). Außerdem bringt Josef noch seine Freundin Miri (15) mit. Sie möchten die Eintrittspreise für Besuche an Wochentagen und am Wochenende herausfinden, aber nur während der Ferienzeit. Erwachsene: 5, Jugendliche: 4, Kinder: 1

a) Wochentag in den Ferien

```
./schwimmbad_brute.py f t 5 4 1 0
Bester Preis wird berechnet für:
Wochenende:  False
Ferien:      True
Erwachsene:  5
Jugendl.:    4
Kinder:      1
Gutscheine:  0

1700.0
+------+------+--------+----------------+-------+
| Erw. | Jug. | Kosten | Kosten einzeln | index |
+======+======+========+================+=======+
| 5    | 1    | 1100   | 1600           | 0.688 |
+------+------+--------+----------------+-------+
| 0    | 1    | 200    | 200            | 1     |
+------+------+--------+----------------+-------+
| 0    | 1    | 200    | 200            | 1     |
+------+------+--------+----------------+-------+
| 0    | 1    | 200    | 200            | 1     |
+------+------+--------+----------------+-------+
```

b) Wochenende in den Ferien

```
./schwimmbad_brute.py t t 5 4 1 0
Bester Preis wird berechnet für:
Wochenende:  True
Ferien:      True
Erwachsene:  5
Jugendl.:    4
Kinder:      1
Gutscheine:  0

1950
+------+------+--------+----------------+-------+
| Erw. | Jug. | Kosten | Kosten einzeln | index |
+======+======+========+================+=======+
| 2    | 2    | 800    | 1200           | 0.667 |
+------+------+--------+----------------+-------+
| 2    | 2    | 800    | 1200           | 0.667 |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
```

4.

Zwei Kegelvereine mit 14 beziehungsweise 18 volljährigen Mitgliedern haben zusammen zwei Gutscheine. Sie möchten eigentlich an zwei verschiedenen Wochenenden (beide nicht in den Ferien) das Schwimmbad heimsuchen. Wieviel können sie sparen, wenn sie stattdessen zusammen gehen?

(Annahme: Jeder Kegelverein besitzt einen Gutschein)

Preis für Kegelverein mit 14 Mitgliedern:

```
./schwimmbad_brute.py t f 14 0 0 1
Bester Preis wird berechnet für:
Wochenende:  True
Ferien:      False
Erwachsene:  14
Jugendl.:    0
Kinder:      0
Gutscheine:  1

4410.0 mit Gruppengutschein
```

Erw.	Jug.	Kosten	Kosten einzeln	index
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1
1	0	350	350	1

Preis für Kegelverein mit 18 Mitgliedern:

./schwimmbad_brute.py t f 18 0 0 1

```
Bester Preis wird berechnet für:
Wochenende:  True
Ferien:      False
Erwachsene:  18
Jugendl.:    0
Kinder:      0
Gutscheine:  1

5670.0 mit Gruppengutschein
+------+------+--------+----------------+-------+
| Erw. | Jug. | Kosten | Kosten einzeln | index |
+======+======+========+================+=======+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
```

Preis beide Kegelvereine zusammen:
./schwimmbad_brute.py t f 32 0 0 2

```
Bester Preis wird berechnet für:
Wochenende:  True
Ferien:      False
Erwachsene:  32
Jugendl.:    0
Kinder:      0
Gutscheine:  2

9765.0 mit Gruppengutschein
+------+------+--------+----------------+-------+
| Erw. | Jug. | Kosten | Kosten einzeln | index |
+======+======+========+================+=======+
| 1    | 0    | 0      | 0              | 0     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
|                                               |
|  Insg. 27 Einzelkarten, eine davon kostenlos  |
|                                               |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
| 1    | 0    | 350    | 350            | 1     |
+------+------+--------+----------------+-------+
```

Wenn beide Vereine einzeln gehen würden, müssten sie insgesamt

44,10 Euro + 56,70 Euro = 100,80 Euro zahlen.

Wenn sie zusammen gehen müssen sie nur 97,60 Euro zahlen und sparen damit

3,20 Euro

Aufgabe 3: Dreiecke zählen

Aufgabe

Schreibe ein Programm, das die Dreiecke in einer Rätsel-Zeichnung zählt. Eine Zeichnung besteht aus einigen Strecken. Du kannst davon ausgehen, dass keine zwei Strecken auf derselben Geraden liegen und dass sich nie mehr als zwei Strecken im gleichen Punkt schneiden.
Wende dein Programm auf die Beispiele an, die du auf den BwInf-Webseiten findest.

Lösungsidee

Nachdem man die Schnittpunkte der gegebenen Strecken berechnet hat, kann man einen ungerichteten Graphen erstellen, mit den Anfangs-, End- und Schnittpunkten als Knoten und Strecken als Kanten.
Auf diesem Graph kann man eine modifizierte Tiefensuche durchführen, über die man eine Liste aller Wege der Länge 3 im Graphen erhält.
Aus dieser Liste werden dann diejenigen Wege entfernt, die kein Dreieck beschreiben. Wenn beispielsweise der Anfangspunkt nicht gleich der Endpunkt ist, kann der Weg kein Dreieck sein.
Nach der Bereinigung der Liste ist die Anzahl der Dreiecke gleich die Anzahl der Elemente in der Liste.

Umsetzung

Die Lösungsidee wird in Python 3.6 implementiert.

Zu Beginn wird die Datei eingelesen und in drei Strukturen konvertiert:

1) ein Dictionary, welches jedem Punkt alle Geraden zuordnet, die diesen berühren
2) ein Dictionary, welches jeder Geraden alle Punkte zuordnet, die sie berührt
3) ein Set, welches alle Punkte enthält

Daraufhin werden alle Schnittpunkte im Graphen G bestimmt. Dafür muss man mit zwei verschachtelten Schleifen für jede Zweierkombination an Strecken in G deren Schnittpunkt berechnen, falls dieser existiert.

Den Schnittpunkt zweier Strecken berechnet man wie folgt:

$P_1 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$ und $P_2 = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}$ sind Start- und Endpunkte der Strecke s_1

$P_3 = \begin{pmatrix} x_3 \\ y_3 \end{pmatrix}$ und $P_4 = \begin{pmatrix} x_4 \\ y_4 \end{pmatrix}$ sind Start- und Endpunkte der Strecke s_2

Zuerst berechnet man den Schnittpunkt[1] der Geraden g_1 und g_2,
wobei g_1 auf den Punkten P_1 und P_2
und g_2 auf den Punkten P_3 und P_4 liegt.

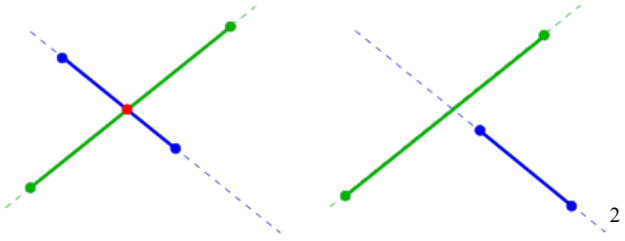
Der Schnittpunkt $S = \begin{pmatrix} x_s \\ y_s \end{pmatrix}$ ergibt sich aus

$$x_s = \frac{(x_4 - x_3)(x_2 y_1 - x_1 y_2) - (x_2 - x_1)(x_4 y_3 - x_3 y_4)}{(y_4 - y_3)(x_2 - x_1) - (y_2 - y_1)(x_4 - x_3)}$$

und

$$y_s = \frac{(y_1 - y_2)(x_4 y_3 - x_3 y_4) - (y_3 - y_4)(x_2 y_1 - x_1 y_2)}{(y_4 - y_3)(x_2 - x_1) - (y_2 - y_1)(x_4 - x_3)}$$

Liegt x_s innerhalb $[x_1, x_2]$ und $[x_3, x_4]$ und y_s innerhalb $[y_1, y_2]$ *und* $[y_3, y_4]$ schneiden sich $s_1$und s_2.



s_1: blaue Strecke, s_2: grüne Strecke

Die beiden Geraden auf der rechten Seite schneiden sich nicht, da x_s nicht in $[x_1, x_2]$ und y_s nicht in $[y_1, y_2]$ liegen.

[1] https://de.wikipedia.org/wiki/Schnittpunkt#Schnittpunkt_zweier_Geraden
[2] Von Ag2gaeh - Eigenes Werk, CC-BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=44999745

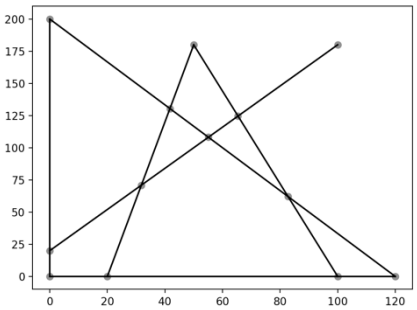
Wenn sich beide Strecken schneiden, werden die beiden Dictionaries und das Set angepasst.

Auf dem vervollständigten Graphen kann man nun eine modifizierte Tiefensuche anwenden.
Mithilfe dreier verschachtelter while-Schleifen werden alle Wege, die drei Kanten lang sind betrachtet.
Mithilfe von Struktur (1) und (2) werden die vom gerade betrachteten Punkt p_1 erreichbaren Punkte bestimmt und in der Liste e_1 gespeichert. In der zweiten Schleife wird über e_1 iteriert und für jeden p_2 eine Liste an erreichbaren Punkten e_2 berechnet. Die dritte Schleife iteriert über e_2 und bestimmt die von p_2 erreichbaren Punkte e_3. Befindet sich der Startpunkt p_1 in e_3 ist p_1 gleichzeitig der Endpunkt des Weges. Ein das 3-Tupel (p_1, p_2, p_3) wird daraufhin sortiert in einem Set gespeichert, damit keine Dreiecke mehrfach gezählt werden.
Die Länge des Sets ist somit auch die Anzahl der Dreiecke.

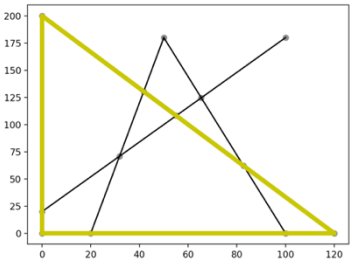
Das Script wird wie folgt ausgeführt:
`dreiecke.py FILE [--visualize|-v]`
Wählt man durch *–visualize* oder *–v* die Visualisierung durch pyplot aus, muss man dieses Modul bereits installiert haben. Solange nicht alle Dreiecke gezeigt wurden, öffnet sich ein neues Fenster, welches das nächste Dreieck hervorhebt, wenn man das alte Fenster schließt.

Beispiele

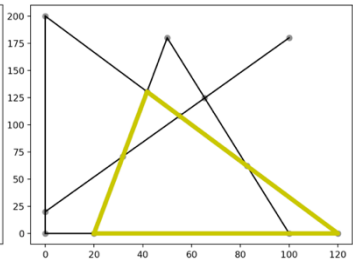
```
./dreiecke.py txt/dreiecke1.txt –v
```



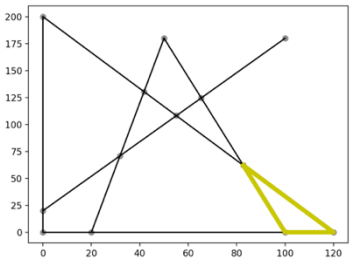
Visualisierung des Graphen



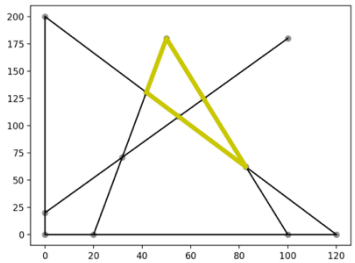
Dreieck 1



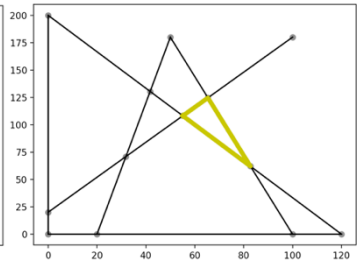
Dreieck 2



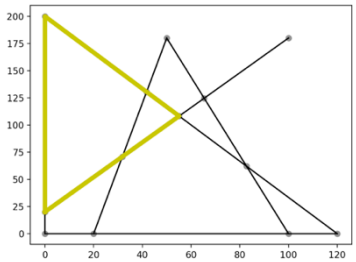
Dreieck 3



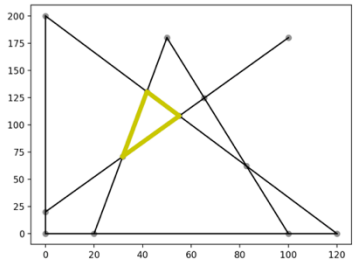
Dreieck 4



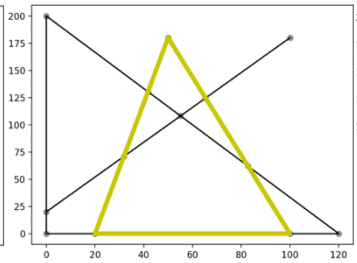
Dreieck 5



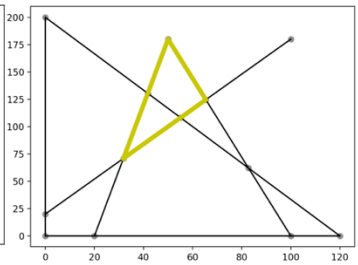
Dreieck 6



Dreieck 5



Dreieck 6

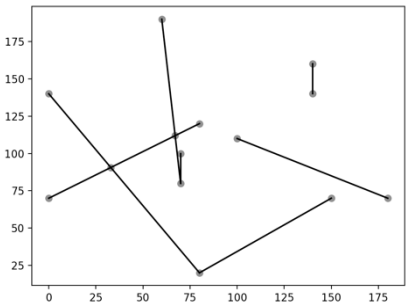


Dreieck 7

```
1 -> ((0.0, 0.0), (0.0, 200.0), (120.0, 0.0))
2 -> ((20.0, 0.0), (41.73913043478261, 130.43478260869566),
(120.0, 0.0))
3 -> ((82.75862068965517, 62.06896551724138), (100.0, 0.0),
(120.0, 0.0))
4 -> ((41.73913043478261, 130.43478260869566), (50.0, 180.0),
      (82.75862068965517, 62.06896551724138))
5 -> ((55.10204081632653, 108.16326530612245),
      (65.38461538461539,124.61538461538461),
      (82.75862068965517, 62.06896551724138))
```

```
6 -> ((0.0, 20.0), (0.0, 200.0), (55.10204081632653,
108.16326530612245))
7 -> ((31.818181818181817, 70.9090909090909),
      (41.73913043478261,130.43478260869566),
      (55.10204081632653, 108.16326530612245))
8 -> ((20.0, 0.0), (50.0, 180.0), (100.0, 0.0))
9 -> ((31.818181818181817, 70.9090909090909),
      (50.0, 180.0), (65.38461538461539,124.61538461538461))
Anzahl Dreiecke:  9
```

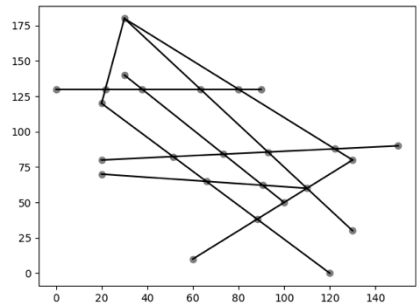
./dreiecke.py txt/dreiecke2.txt –v



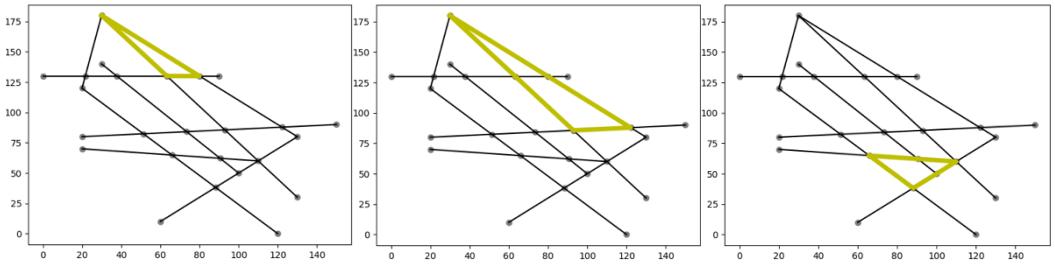
Visualisierung des Graphen

Anzahl Dreiecke: 0

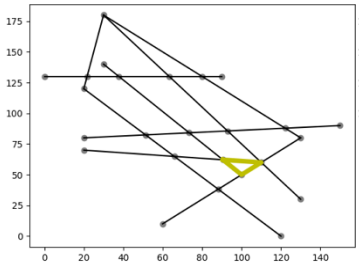
./dreiecke.py txt/dreiecke3.txt –v



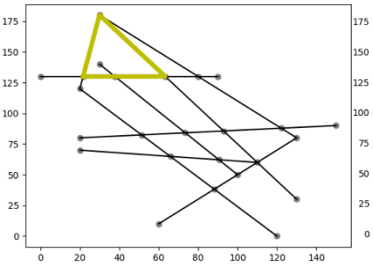
Visualisierung des Graphen



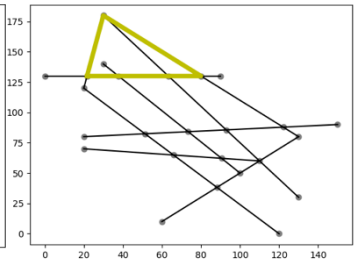
Dreieck 1 *Dreieck 2* *Dreieck 3*



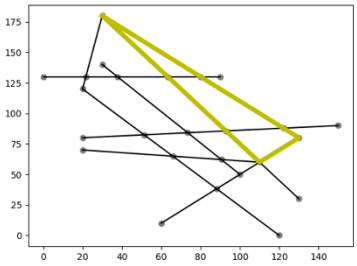
Dreieck 4



Dreieck 5



Dreieck 6



Dreieck 7

```
1 -> ((30.0, 180.0), (63.333333333333336, 130.0), (80.0, 130.0))
2 -> ((30.0, 180.0), (92.92682926829268, 85.60975609756098),
      (122.14285714285714, 87.85714285714286))
3 -> ((65.91836734693878, 64.89795918367346),
      (88.18181818181819,38.18181818181818), (110.0, 60.0))
4 -> ((90.54054054054055, 62.16216216216216), (100.0, 50.0),
(110.0, 60.0))
5 -> ((21.666666666666668, 130.0), (30.0, 180.0),
(63.333333333333336, 130.0))
6 -> ((21.666666666666668, 130.0), (30.0, 180.0), (80.0, 130.0))
7 -> ((30.0, 180.0), (110.0, 60.0), (130.0, 80.0))
Anzahl Dreiecke:  7
```

Aufgabe 4: Auto-Scrabble

1. Stimmt es, dass TIMO nicht auf einem Kennzeichen stehen kann?

Kennzeichen = Kürzel aus Kürzelliste + [A-Z]{1,2}[3] *# Zahl wird ignoriert*

(Kürzelliste ist die offizielle Liste aller Unterscheidungszeichen für deutsche Nummernschilder)

1. Möglichkeit

"TIMO" = Kürzel + "MO" → Kürzel müsste sein: "TI"

2. Möglichkeit

"TIMO" = Kürzel + "O" → Kürzel müsste sein: "TIM"

Kürzelliste enthält weder "TIM" noch "TI"

→ Daher kann "TIMO" nicht auf einem Kennzeichen stehen

2. Gib ein weiteres Wort mit vier Buchstaben an, das nicht auf einem Kennzeichen stehen kann. Findest du sogar ein solches Wort mit drei Buchstaben? Mit zwei?

Ein weiteres Wort mit 4 Buchstaben, das nicht auf einem Kennzeichen stehen kann ist zum Beispiel: „JAZZ“

Um solche Wörter mit 2-5 Buchstaben zu finden habe ich folgendes Script geschrieben:

```
# !usr/bin/env python3
# encoding: utf-8

import sys

def open_file(name):
    with open(name, 'r', encoding='UTF-8') as file:
        return sorted({line.strip() for line in
file.readlines()})
```

[3] Regulärer Ausdruck. Bedeutung: Aneinanderreihung der Großbuchstaben (A-Z) mit min. Länge 1 und max. Länge 2

```
def is_possible(s, kuerzel):
    if len(s) > 5: return False
    umlaute = set('ÄÖÜ')
    s = s.upper().strip()
    for i in range(1, min(len(s), 4)):
        # Teilt String in 2 Teile
        pre, post = s[:i], s[i:]
        # Zweiter Teil darf laut Aufgabestellung nicht >2 sein
        if len(post) > 2: continue
        if not any((c in umlaute) for c in post) and pre in
kuerzel:
            return True
    return False

def find_words(wordlist, kuerzel):
    w_list, k_list = open_file(wordlist), open_file(kuerzel)
    return sorted(set([word for word in w_list if not
is_possible(word, k_list)]))


if __name__ == '__main__':
    nr_letters = 4
    if len(sys.argv) == 2:
        nr_letters = sys.argv[1]
    wordlist = "wordlist.txt"
    kuerzel = "../txt/kuerzelliste.txt"
    words = find_words(wordlist, kuerzel)
    [print(w) for w in words if len(w) == nr_letters]
```

Die Funktion *is_possible(s, kuerzel)* spaltet *s* am Index $i \in [1, \min(4, len(s)) - 1]$ in zwei Substrings s_1 und s_2 auf und überprüft ob sich s_1 in *kuerzel* befindet und s_2 maximal zwei Zeichen lang ist und keine Sonderzeichen enthält.

Das Script kann mit

./4b.py [len]

ausgeführt werden und gibt alle Wörter der Länge *len* aus der Wortliste *wordlist.txt* aus, die nicht auf einem Nummernschild dargestellt werden können.

Beispielwörter:

Länge 4:	Länge 3:	Länge 2:
ADAC	ICH	IM
ADAM	IHM	OB
ADEL	IHR	OK
ADER	IST	TU
AFFE	ORT	UM
ALEX	TAG	
ALGE	TAL	
ALLE	TAT	
ALSO	TAU	
ALTE	TUN	
ARIE	TYP	
ARME	UFO	
AXEL	UKW	
BREI	USA	
BROT	ICH	

3. Schreibe ein Programm, das ein Wort einliest und überprüft, ob es mit mehreren Kennzeichen geschrieben werden kann und eine Folge von Kennzeichen ausgibt, mit denen das Wort gebildet werden kann

Lösungsidee

Mithilfe der Funktion *is_possible()* aus c) können wir nun überprüfen, ob eine Zeichenkette der Länge 2-5 auf einem einzigen Nummernschild darstellbar ist. Nun müssen wir nur noch überprüfen, ob für ein gegebenes beliebig langes Wort eine Aneinanderreihung auf einem Kennzeichen darstellbarer Zeichenketten existiert.

Umsetzung

Die Funktion *is_possible()* kann beinahe unverändert übernommen werden. Da die Funktion in diesem Programm sehr oft aufgerufen wird und durch die Zeile

```
if not any((c in umlaute) for c in post) and pre in kuerzel:
```

genauer gesagt durch den Ausdruck

```
pre in kuerzel
```

eine Laufzeit von $O(n)$ hat, habe ich sie durch folgende Zeile ersetzt.

```
if not any((c in umlaute) for c in post) and \
binary_search(KUERZEL, pre):
```

Somit hat *is_possible()* eine Laufzeit von O(log n). Die binäre Suche bietet sich auch deswegen sehr gut an, da „kuerzelliste.txt“ bereits alphabetisch sortiert vorliegt.

Im Durchschnitt terminiert jetzt das Programm nach 0,006 s (davor 0,8 s), wenn es alle Wörter aus der Liste „autoscrabble.txt“ überprüft.

Um nun zu überprüfen, ob ein Wort mit mehreren Kennzeichen geschrieben werden kann, wird *check(s)* ausgeführt.

Diese Funktion funktioniert folgendermaßen:

Es gibt einen Stack *stack* der in jedem Eintrag den Startindex des betrachteten Bereichs von s und die Länge dieses Bereichs speichert.

Es gilt:

I. $stack[0][0] = 0$

II. $stack[n+1][0] = stack[n][0] + stack[n][1]$

III. $2 \leq stack[n][1] \leq 5$

Das heißt, der Stack speichert direkt aneinanderhängende Bereiche der Länge 2–5 in *s*.

Solange der im letzten Element des Stacks gespeicherte Index kleiner als die Länge von *s* ist, wird

$$i, n = stack[len(stack) - 1]$$

bestimmt und entweder (a) oder (b) ausgeführt.

a) wenn $word[i: i + n]$ auf einem einzigen Nummernschild darstellbar ist, wird ein neues Element welches nach Regel II als Index den Wert $i + 1$ und für die Länge des nächsten Abschnittes den Wert 5 speichert.

b) sonst (wenn $word[i: i + n]$ nicht auf einem einzigen Nummernschild darstellbar ist)

a. ist $n > 2$ wird n um 1 dekrementiert

b. sonst (wenn $n \leq 2$) wird das oberste Element im Stack entfernt.

a. Ist der Stack jetzt leer, bedeutet dies, dass das Wort nicht mit mehreren Nummernschildern darstellbar ist und *False* zurückgegeben werden kann.

b. Sonst wird nochmal das Element oben auf dem Stack entfernt und beim Vorgänger der Wert der Länge um 1 dekrementiert

Nach Durchlauf dieser Schleife wird das oberste Element entfernt.

Mit folgender List-Comprehension werden die Elemente aus *stack* in Substrings aus *word* „übersetzt“: `comb = [word[i: i + n] for i, n in stack]`

Beispiele

Am Beispielwort *s* = „BIBER“ wird nun der Ablauf der Funktion $check(s)$ nochmal erläutert.

Stack: [0, 5]

„BIBER“ = s[0:5] -> nicht darstellbar

Stack: [0, 4]

„BIBE“ = s[0:4] -> nicht darstellbar

Stack: [0, 3]

„BIB“ = s[0:3] -> darstellbar

Stack:

[3, 5]
[0, 3]

```
„BIB“  = s[0:3]
„ER“   = s[3:3+5]      -> nicht darstellbar
```

…

```
„BIB“ = s[0:3]
„ER“  = s[3:3+2]       -> nicht darstellbar
```

Stack:

[0, 2]

```
„BI“  = s[0:2]         -> darstellbar
```

Stack:

[2, 5]
[0, 2]

```
„BI“  = s[0:2]
„BER“ = s[2:2+5]       -> darstellbar
```

→ BIBER darstellbar: BI + BER

./autoscrabble.py autoscrabble.txt kuerzelliste.txt

```
BIBER
[[0, 5]]
[[0, 4]]
[[0, 3]]
[[0, 3], [3, 5]]
[[0, 3], [3, 4]]
[[0, 3], [3, 3]]
[[0, 3], [3, 2]]
[[0, 2]]
[[0, 2], [2, 5]]
[[0, 2], [2, 5], [7, 5]]
Darstellbar!
['BI', 'BER']

BUNDESWETTBEWERB
[[0, 5]]
[[0, 4]]
[[0, 3]]
[[0, 3], [3, 5]]
[[0, 3], [3, 4]]
[[0, 3], [3, 4], [7, 5]]
[[0, 3], [3, 4], [7, 4]]
[[0, 3], [3, 4], [7, 3]]
[[0, 3], [3, 4], [7, 2]]
[[0, 3], [3, 3]]
[[0, 3], [3, 3], [6, 5]]
[[0, 3], [3, 3], [6, 4]]
[[0, 3], [3, 3], [6, 4], [10, 5]]
[[0, 3], [3, 3], [6, 4], [10, 4]]
[[0, 3], [3, 3], [6, 4], [10, 3]]
```

```
[[0, 3], [3, 3], [6, 4], [10, 3], [13, 5]]
[[0, 3], [3, 3], [6, 4], [10, 3], [13, 4]]
[[0, 3], [3, 3], [6, 4], [10, 3], [13, 3]]
[[0, 3], [3, 3], [6, 4], [10, 3], [13, 2]]
[[0, 3], [3, 3], [6, 4], [10, 2]]
[[0, 3], [3, 3], [6, 4], [10, 2], [12, 5]]
[[0, 3], [3, 3], [6, 4], [10, 2], [12, 5], [17, 5]]
Darstellbar!
['BUN', 'DES', 'WETT', 'BE', 'WERB']

(Ab jetzt ohne Ausgabe des Stacks)

CLINTON
Darstellbar!
['CLI', 'NTON']

DONAUDAMPFSCHIFFFAHRTSKAPITÄNSMÜTZE
Nicht darstellbar!

ETHERNET
Nicht darstellbar!

INFORMATIK
Darstellbar!
['INFO', 'RM', 'ATIK']

LLANFAIRPWLLGWYNGYLLGOGERYCHWYRNDROBWLLLLANTYSILIOGOGOGOCH
Darstellbar!
['LLA', 'NFAI', 'RPW', 'LLG', 'WYN', 'GYL', 'LGO', 'GE', 'RY',
'CH', 'WYR', 'ND', 'ROB', 'WLLL', 'LA', 'NTY', 'SILI', 'OGOG',
'OGO', 'CH']

RINDFLEISCHETIKETTIERUNGSÜBERWACHUNGSAUFGABENÜBERTRAGUNGSGESETZ
Nicht darstellbar!

SOFTWARE
Darstellbar!
['SOFT', 'WARE']

TRUMP
Nicht darstellbar!

TSCHÜSS
Nicht darstellbar!

VERKEHRSWEGEPLANUNGSBESCHLEUNIGUNGSGESETZ
Darstellbar!
['VERKE', 'HRS', 'WEGE', 'PLAN', 'UNGS', 'BES', 'CH', 'LEU',
'NIG', 'UNGS', 'GE', 'SETZ']
```

Aufgabe 5: Bauernopfer

1. Turm gegen 8 Bauern

Lösungsidee:

Die beste Startposition für die Bauern ist, eine Kette zu bilden, die das Schachfeld möglichst mittig teilt. Der Turm stellt sich nun auf ein Feld, von dem aus er möglichst viele Zugmöglichkeiten hat und möglichst weit vom nächsten Bauern entfernt ist.
Die Bauern versuchen nun die Anzahl möglicher Züge des Turms einzuschränken, ohne den Turm ihre Reihe durchschreiten zu lassen. Dies wird erreicht, indem die Bauern darauf achten, dass der Abstand zwischen zwei benachbarten Bauern nicht größer als ein Feld wird und - wenn möglich - der Bauer in der Reihe des Turmes zum Turm hin aufrückt.
Der Turm versucht sich so zu bewegen, dass er in der nächsten Runde eine größtmögliche Auswahl an Zügen hat und nicht direkt neben einem Bauern steht.

Umsetzung:

Das Programm besteht im Grunde aus drei Klassen: Game, Board und Chessman.
Die Klasse *Game* beinhaltet die gesamte Spiellogik und speichert ein Objekt der Klasse Board, in der alle Spieldaten, sprich Positionen der Figuren gespeichert sind. Die wichtigsten zwei Funktionen in *Game* sind *rook_next_move()* und *pawn_next_move()*, welche den besten Zug für die jeweilige Partei berechnen.
Die Klasse *Chessman* ist eine Oberklasse für *Pawn (Bauer)*, *Rook (Turm)* und *EmptyField.*
Unterklassen von *Chessman* speichern ihre eigene Position und das Symbol, mit welchem sie auf dem Schachfeld repräsentiert werden. *Rook* und *Pawn* haben eine Methode, die von *Game* aus aufgerufen werden kann und alle für die Figur erreichbaren Felder zurückgibt.
Die Methode *pawn_next_move()* betrachtet für jeden Bauern alle möglichen Züge und sortiert diese nach dem Kriterium, wie viele Zugmöglichkeiten der Turm danach hat. Desto weniger Möglichkeiten, desto besser ist der Zug des Bauern.
Hierauf wird überprüft, ob die Bauern nach dem Zug noch eine Kette von der

einen zur anderen Seite des Spielfelds bilden. Falls nicht, wird solange der nächstbeste Zug geprüft, bis dieses Kriterium erfüllt ist.
Die Methode *rook_next_move()* iteriert über alle für den Turm möglichen Züge und führt denjenigen aus, der ihm im die Meisten Zugmöglichkeiten im nächsten Zug verschafft.

Beispiel:

Der Ablauf des Spieles sieht wie folgt aus:

Ausgangsstellung:

```
8 . . . . . . . ♜
7 . . . . . . . .
6 . . . . . . . .
5 . . . . . . . .
4 ♙ ♙ ♙ ♙ ♙ ♙ ♙ ♙
3 . . . . . . . .
2 . . . . . . . .
1 . . . . . . . .
  A B C D E F G H
```

```
1.
WHITE is moving
Best move: H4 – H5
8 . . . . . . . ♜
7 . . . . . . . .
6 . . . . . . . .
5 . . . . . . . ♙
4 ♙ ♙ ♙ ♙ ♙ ♙ ♙ .
3 . . . . . . . .
2 . . . . . . . .
1 . . . . . . . .
  A B C D E F G H
```

```
2.
BLACK is moving
Best move: H8 – A8
8 ♜ . . . . . . .
7 . . . . . . . .
6 . . . . . . . .
5 . . . . . . . ♙
4 ♙ ♙ ♙ ♙ ♙ ♙ ♙ .
3 . . . . . . . .
2 . . . . . . . .
1 . . . . . . . .
  A B C D E F G H
```

```
3.
WHITE is moving
Best move: A4 – A5
8 ♜ . . . . . . .
7 . . . . . . . .
6 . . . . . . . .
5 ♙ . . . . . . ♙
4 . ♙ ♙ ♙ ♙ ♙ ♙ .
3 . . . . . . . .
2 . . . . . . . .
1 . . . . . . . .
  A B C D E F G H
```

```
4.
BLACK is moving
Best move: A8 – B8
8 . ♜ . . . . . .
7 . . . . . . . .
6 . . . . . . . .
5 ♙ . . . . . . ♙
4 . ♙ ♙ ♙ ♙ ♙ ♙ .
3 . . . . . . . .
2 . . . . . . . .
1 . . . . . . . .
  A B C D E F G H
```

```
5.
WHITE is moving
Best move: B4 – B5
8 . ♜ . . . . . .
7 . . . . . . . .
6 . . . . . . . .
5 ♙ ♙ . . . . . ♙
4 . . ♙ ♙ ♙ ♙ ♙ .
3 . . . . . . . .
2 . . . . . . . .
1 . . . . . . . .
  A B C D E F G H
```

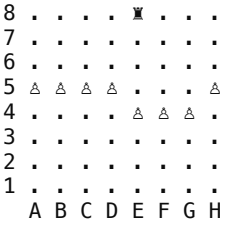
```
6.
BLACK is moving
Best move: B8 – C8
8 . . ♜ . . . . .
7 . . . . . . . .
6 . . . . . . . .
5 ♙ ♙ . . . . . ♙
4 . . ♙ ♙ ♙ ♙ ♙ .
3 . . . . . . . .
2 . . . . . . . .
1 . . . . . . . .
  A B C D E F G H
```

```
7.
WHITE is moving
Best move: C4 – C5
8 . . ♜ . . . . .
7 . . . . . . . .
6 . . . . . . . .
5 ♙ ♙ ♙ . . . . ♙
4 . . . ♙ ♙ ♙ ♙ .
3 . . . . . . . .
2 . . . . . . . .
1 . . . . . . . .
  A B C D E F G H
```

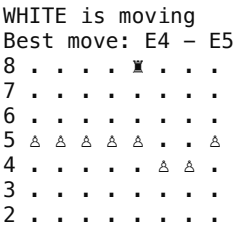
```
8.
BLACK is moving
Best move: C8 – D8
8 . . . ♜ . . . .
7 . . . . . . . .
6 . . . . . . . .
5 ♙ ♙ ♙ . . . . ♙
4 . . . ♙ ♙ ♙ ♙ .
3 . . . . . . . .
2 . . . . . . . .
1 . . . . . . . .
  A B C D E F G H
```

```
9.
WHITE is moving
Best move: D4 – D5
8 . . . ♜ . . . .
7 . . . . . . . .
6 . . . . . . . .
5 ♙ ♙ ♙ ♙ . . . ♙
4 . . . . ♙ ♙ ♙ .
3 . . . . . . . .
2 . . . . . . . .
1 . . . . . . . .
  A B C D E F G H
```

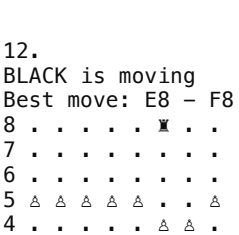
10.
BLACK is moving
Best move: D8 – E8



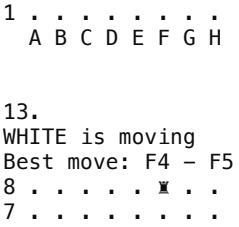
11.
WHITE is moving
Best move: E4 – E5



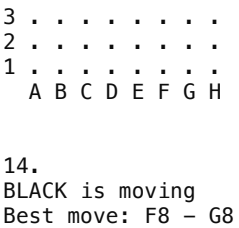
12.
BLACK is moving
Best move: E8 – F8



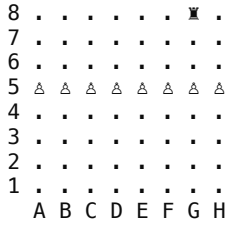
13.
WHITE is moving
Best move: F4 – F5



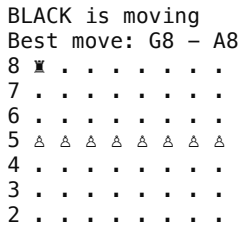
14.
BLACK is moving
Best move: F8 – G8



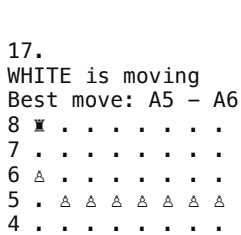
15.
WHITE is moving
Best move: G4 – G5



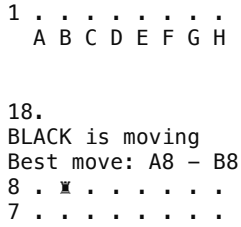
16.
BLACK is moving
Best move: G8 – A8



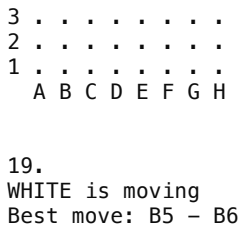
17.
WHITE is moving
Best move: A5 – A6



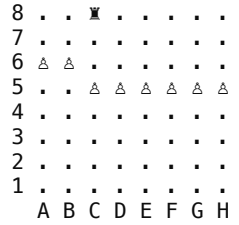
18.
BLACK is moving
Best move: A8 – B8



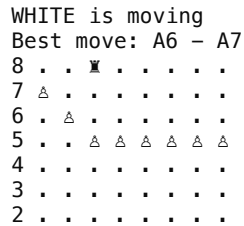
19.
WHITE is moving
Best move: B5 – B6



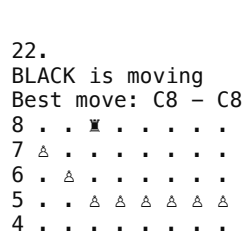
20.
BLACK is moving
Best move: B8 – C8



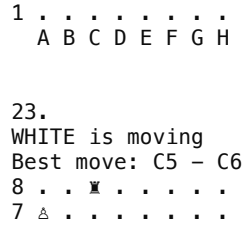
21.
WHITE is moving
Best move: A6 – A7



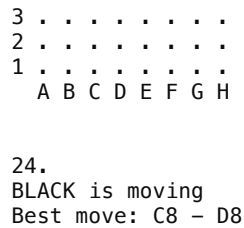
22.
BLACK is moving
Best move: C8 – C8



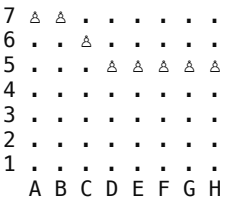
23.
WHITE is moving
Best move: C5 – C6



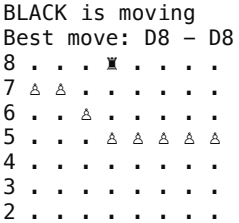
24.
BLACK is moving
Best move: C8 – D8



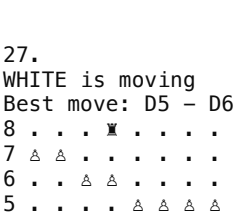
25.
WHITE is moving
Best move: B6 – B7



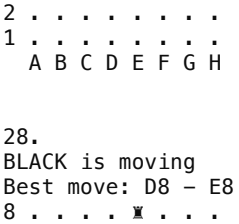
26.
BLACK is moving
Best move: D8 – D8



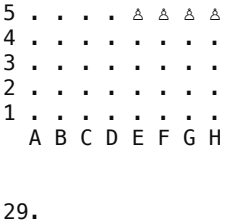
27.
WHITE is moving
Best move: D5 – D6



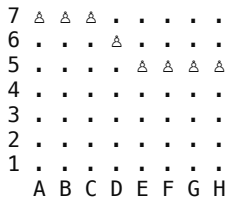
28.
BLACK is moving
Best move: D8 – E8



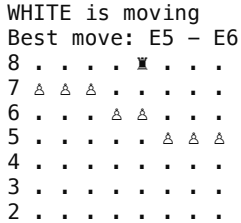
29.
WHITE is moving
Best move: C6 – C7



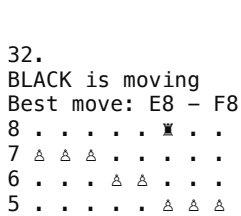
30.
BLACK is moving
Best move: E8 – E8



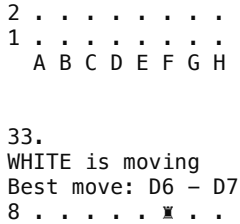
31.
WHITE is moving
Best move: E5 – E6



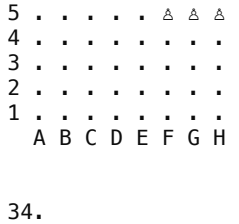
32.
BLACK is moving
Best move: E8 – F8



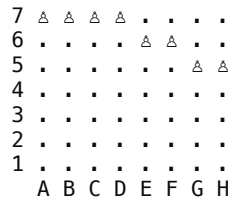
33.
WHITE is moving
Best move: D6 – D7



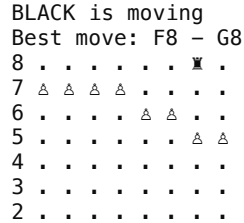
34.
BLACK is moving
Best move: F8 – F8



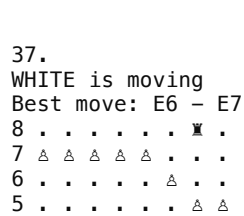
35.
WHITE is moving
Best move: F5 – F6



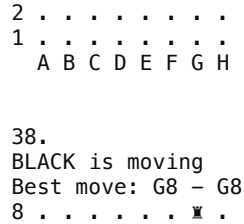
36.
BLACK is moving
Best move: F8 – G8



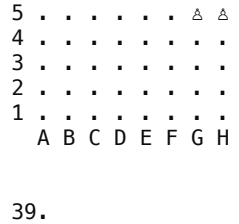
37.
WHITE is moving
Best move: E6 – E7

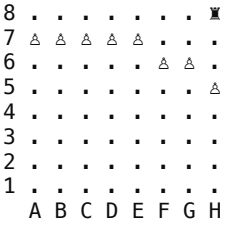


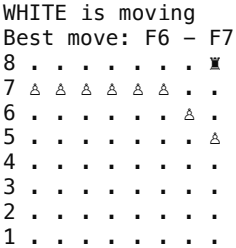
38.
BLACK is moving
Best move: G8 – G8

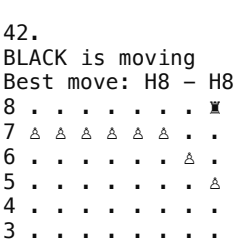


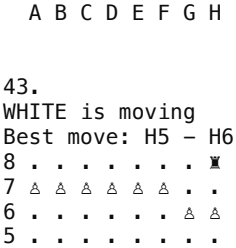
39.
WHITE is moving
Best move: G5 – G6

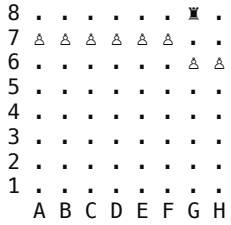


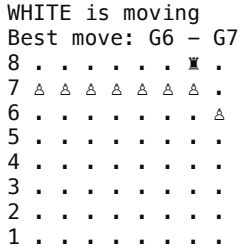
40.
BLACK is moving
Best move: G8 – H8


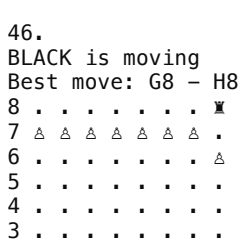
41.
WHITE is moving
Best move: F6 – F7


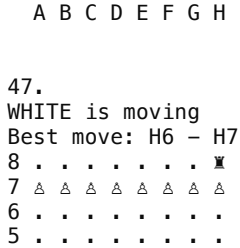
42.
BLACK is moving
Best move: H8 – H8


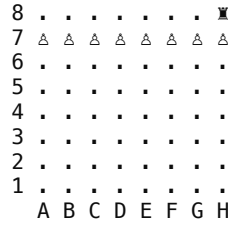
43.
WHITE is moving
Best move: H5 – H6


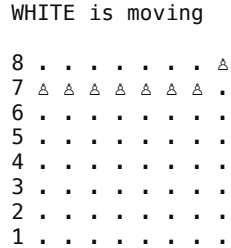
44.
BLACK is moving
Best move: H8 – G8


45.
WHITE is moving
Best move: G6 – G7


46.
BLACK is moving
Best move: G8 – H8


47.
WHITE is moving
Best move: H6 – H7


48.
BLACK is moving
Can't move


49.
WHITE is moving


WHITE won!

2. Turm gegen 7 Bauern

Lösungsidee:

Sieben Bauern können den Schwarzen Turm nicht fangen, da sie immer eine Lücke in ihrer Reihe hinterlassen, egal, wie sie sich aufstellen.
Damit der Turm nicht gefangen wird muss er sich immer auf das Feld stellen, von dem aus er im nächsten Zug die Bauernreihe durchschreiten könnte. Wenn die Bauern diese Lücke schließen, entsteht dabei eine neue Lücke. Da die Bauern immer den Turm daran hindern müssen, durch die Reihe zu ziehen, und sie pro Runde nur einen Zug haben, können sie nicht zum Turm aufrücken, um diesen zu fangen.

Umsetzung:

Das Programm ist beinahe das Gleiche wie in Teilaufgabe 1. Lediglich die Funktion *pawn_next_move()* wurde angepasst. Sie überprüft nun nicht mehr, ob die Bauern eine Kette einhalten, da dies mit 7 Bauern nicht möglich ist.

Beispiel:

Ausgangsstellung:
(Bauern stellen sich soweit möglich in einer Kette mitten im Spielfeld auf. Turm sucht sich Feld, von dem er die meisten Felder Erreichen kann)

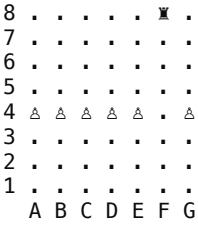
```
8 . . . . . . . ♜
7 . . . . . . . .
6 . . . . . . . .
5 . . . . . . . .
4 ♙ ♙ ♙ ♙ ♙ ♙ ♙ .
3 . . . . . . . .
2 . . . . . . . .
1 . . . . . . . .
  A B C D E F G H
```

```
1.
WHITE is moving
Best move: G4 - H4
8 . . . . . . . ♜
7 . . . . . . . .
6 . . . . . . . .
5 . . . . . . . .
4 ♙ ♙ ♙ ♙ ♙ ♙ . ♙
3 . . . . . . . .
2 . . . . . . . .
1 . . . . . . . .
  A B C D E F G H
```

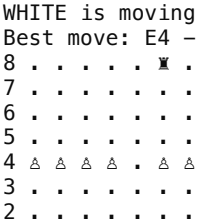
```
2.
BLACK is moving
Best move: H8 - G8
8 . . . . . . ♜ .
7 . . . . . . . .
6 . . . . . . . .
5 . . . . . . . .
4 ♙ ♙ ♙ ♙ ♙ ♙ . ♙
3 . . . . . . . .
2 . . . . . . . .
1 . . . . . . . .
  A B C D E F G H
```

```
3.
WHITE is moving
Best move: F4 - G4
8 . . . . . ♜ . .
7 . . . . . . . .
6 . . . . . . . .
5 . . . . . . . .
4 ♙ ♙ ♙ ♙ ♙ . ♙ ♙
3 . . . . . . . .
2 . . . . . . . .
1 . . . . . . . .
  A B C D E F G H
```

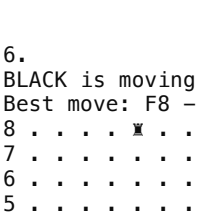
4.
BLACK is moving
Best move: G8 – F8



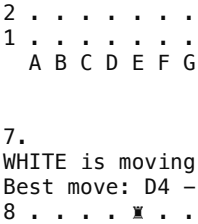
5.
WHITE is moving
Best move: E4 – F4



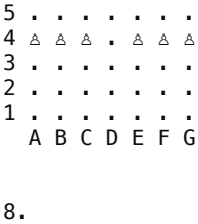
6.
BLACK is moving
Best move: F8 – E8



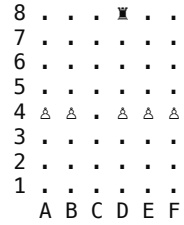
7.
WHITE is moving
Best move: D4 – E4



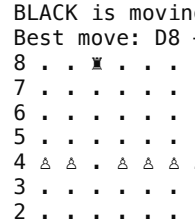
8.
BLACK is moving
Best move: E8 – D8



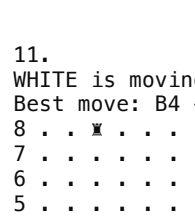
9.
WHITE is moving
Best move: C4 – D4



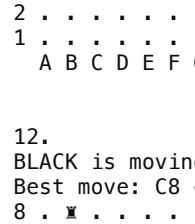
10.
BLACK is moving
Best move: D8 – C8



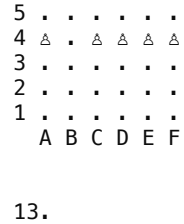
11.
WHITE is moving
Best move: B4 – C4



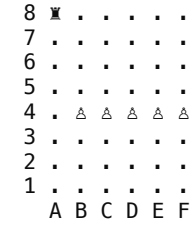
12.
BLACK is moving
Best move: C8 – B8



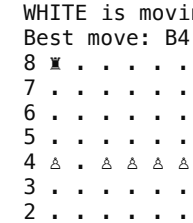
13.
WHITE is moving
Best move: A4 – B4



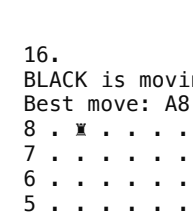
14.
BLACK is moving
Best move: B8 – A8



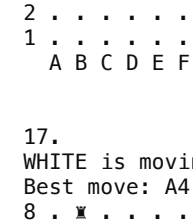
15.
WHITE is moving
Best move: B4 – A4



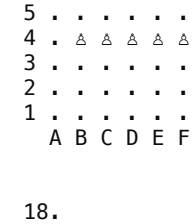
16.
BLACK is moving
Best move: A8 – B8



17.
WHITE is moving
Best move: A4 – B4



18.
BLACK is moving
Best move: B8 – A8



usw.

3. Untersuchen des Spielverlaufes für k Bauern und l Züge

k	*l*	*Sieg der Bauern immer möglich*
7	1	Nein, Turm überlebt unendlich lange, wenn er wie in b) beschrieben zieht.
7	2	Ja. Mit dem ersten Zug können sie die Lücke in der Reihe schließen und mit dem zweiten den Turm einengen
6-4	2	Ja. Auch mit sechs bis vier Bauern funktioniert die für 7, 2 erklärte Taktik, da bei richtiger Aufstellung maximal zwei Felder zwischen den Bauern frei sind und diese Lücke immer mit einem Zug geschlossen werden kann.
3	2	Nein. Eine Lücke zwischen den Bauern kann drei Felder umfassen. Um die Lücke zu schließen, muss ein Bauer also seine beiden Züge aufbrauchen. Für die Bauern ist kein Zug mehr übrig, um zum Turm aufzurücken und diesen einzuengen.
3	3	Ja. Maximal zwei Züge um eine Lücke zu schließen. Ein Zug bleibt übrig, um zum Turm aufzurücken.
2	3	Nein. Die Bauern können zwar dafür sorgen, dass der Turm nicht auf die andere Seite des Spielfelds gelangt, jedoch bleibt ihnen kein Zug um zum Turm aufzurücken.
2	4	Ja. Wenn sich die Bauern auf C5 und F5 stellen können sie den Turm in 4 Zügen besiegen.
1	4-6	Nein. Der Turm kann sich von Ecke zu Ecke bewegen und ist für einen Bauern unerreichbar.
1	7	Ja. Mit sieben Zügen kann der Bauer, wenn er in der Mitte des Spielfeldes alle Felder, bis auf ein Eckfeld erreichen. Der Turm stellt sich natürlich in das einzige für den Bauern unerreichbare Feld. Der Bauer stellt sich daraufhin auf das Feld, von dem er alle für den Turm erreichbare Felder auch erreichen kann.

4. Untersuchen des Spielverlaufs: Dame gegen 8 Bauern

Acht Bauern können auch gegen eine Dame gewinnen. Sie müssen nur darauf achten, die diagonalen Lücken zu schließen. Beim Schließen der Lücke rücken sie automatisch auch ein Feld zur Dame auf.

Quellcodes

Aufgabe 1:

```
#! /usr/bin/env python3

import sys
import numpy as np

def read_file(s):
    f = open(s, 'r')
    p = f.readlines()
    personen = []
    for i in range(0, len(p), 4):
        name = p[i].replace("\n", "")
        pro = p[i + 1].replace("\n", "").replace("+", "").split(" ")
        con = p[i + 2].replace("\n", "").replace("-", "").split(" ")

        # entfernt leere Einträge in Liste
        while '' in pro:
            pro.remove('')
        while '' in con:
            con.remove('')

        person = [name, pro, con]
        personen.append(person)
    f.close()
    return sorted(personen)


# erstellt eine Beziehungsmatrix
def create_matrix(l, names):
    matrix = np.array([[0] * len(l) for _ in range(len(l))])
    for i in range(len(l)):
        pro = l[i][1]
        con = l[i][2]
        for p in pro:
            ind = names.index(p)
            matrix[i][ind] = 1
        for p in con:
            ind = names.index(p)
            matrix[i][ind] = -1
    return matrix

# spiegelt Beziehungsmatrix wenn möglich
def operate_on_matrix(matrix):
    size = len(matrix)
    for y in range(size):
        for x in range(size):
            if matrix[y][x] == 1 and matrix[x][y] == -1:
                print("Zimmerverteilung nicht möglich!")
                sys.exit(0)
            if matrix[y][x] == 1:
                matrix[x][y] = 1
            if matrix[y][x] == -1:
                matrix[x][y] = -1


def all_visited(l):
    for b in l:
        if not b:
            return False
    return True


def get_first_non_visited(l):
    for i in range(len(l)):
        if not l[i]:
            return i

# Modifizierte iterative Implementierung der Breitensuche
```

```
# Hier werden durch zwei ungerichtete Graphen die Zusammenhangskomponenten
herausgesucht
def bfs(matrix):
    # Adjazenzliste für Graph_1
    adj_liste_pro = []
    # Adjazenzliste für Graph_2
    adj_liste_con = []
    room_list = []

    # Adjazenzmatrix wird in zwei Adjazenzlisten konvertiert
    for y in range(len(matrix)):
        pro = []
        con = []
        for x in range(len(matrix)):
            if matrix[y][x] == 1:
                pro.append(x)
            if matrix[y][x] == -1:
                con.append(x)
        adj_liste_pro.append(pro)
        adj_liste_con.append(con)

    # hier wird gespeichert, ob ein Knoten schon besucht wurde
    visited = [False] * len(matrix)
    q = []

    while not all_visited(visited):
        # Raum = Zusammenhangskomponente
        room = []

        # Erster Knoten im Graph wird
        # der Warteschlange hinzugefügt,
        node = get_first_non_visited(visited)
        q.append(node)
        # als besucht markiert,
        visited[node] = True
        # dem Zimmer hinzugefügt
        room.append(node)

        # fügt alle vom `first_non_visited` Knoten erreichbaren Knoten der
Raumliste hinzu
        while not len(q) == 0:
            node = q.pop(0)
            for child in adj_liste_pro[node]:
                if not visited[child]:
                    if len(set(room).intersection(adj_liste_con[child])) > 0:
                        # braucht keinen Check ob Schülerinnen im Zimmer mit neuer
Schuelerin zusammen sein wollen,
                        # weil matrix gespiegelt
                        print("Zimmerverteilung nicht möglich! Person:", child)
                        # gibt Person aus, bei der das "Problem" ausgelöst wurde
                        sys.exit()
                    q.append(child)
                    visited[child] = True
                    room.append(child)
        room_list.append(sorted(set(room)))
    return room_list


def find_room(person, zimmeraufteilung):
    for zimmer in zimmeraufteilung:
        if person[0] in zimmer:
            return zimmer


# überprüft ob abgegebe Listen Sinn ergeben,
# zum Beispiel steht Marie in "zimmerbelegung5.txt" auf ihrer eigenen Kontra Liste
def check(personen):
    names_1 = set()
    names_2 = set()
    for pers in personen:
        names_1.add(pers[0])
        names_2.add(pers[0])
        names_1.update(pers[1])
        names_1.update(pers[2])
        if pers[0] in pers[2]:
```

```
            print()
            print("[INFO]", pers[0] + " steht auf ihrer eigenen Kontra-Liste!")
            print("[INFO]", pers[0] + " wird von " + str(pers[2]) + " entfernt.")
            pers[2].remove(pers[0])

    for n in names_1 - names_2:
        print("[INFO]", n, "wurde erwähnt, hat jedoch selbst keinen Zettel
abgegeben.")
        personen.append([n, [], []])



def zimmeraufteilung(personen):
    check(personen)

    schuelerListe = []
    for p in personen:
        schuelerListe.append(p[0])

    matrix = create_matrix(personen, schuelerListe)

    operate_on_matrix(matrix)   # Spiegelt Matrix wenn möglich
    room_list = bfs(matrix)

    room_list_namen = []
    for l in room_list:
        tmp = []
        for i in l:
            tmp.append(schuelerListe[i])
        room_list_namen.append(tmp)

    print("Zimmeraufteilung möglich!")
    for l in room_list_namen:
        print(l)

    return room_list_namen, personen


if __name__ == '__main__':
    path = sys.argv[1]
    personen = read_file(path)
    zimmeraufteilung(personen)
```

Aufgabe 2

schwimmbad_brute.py

```
#! /usr/bin/env python3

import copy
import prices
import sys
from prices import show_table

global V
V = False


def finished(lst):
    for a, b in lst:
        if a != b:
            return False
    return True


def remove_duplicates(lst):
    c = []
    tmp = [0, 0]
    for e in lst:
        comp = [e[0], e[1]]
        b = True
        for i in range(len(tmp)):
            if tmp[i] != comp[i]:
                b = False
        if not b:
            c.append(e)
        tmp = comp
    return c


def adjust(t):
    return sorted(t, key=lambda k: k['ind'], reverse=False)


def main(we, fe, g, k, j, e):
    if k > 0 and e == 0:
        print("Kinder unter 4J dürfen nicht ohne begleitung eines Erwachsenen ins
Schwimmbad gehen!")
        return False

    # prices() gibt eine Tabelle mit allen möglichen Karten für eine Kombination
(e, j)
    # zurück
    table = prices.prices(e, j, we)
    table = adjust(table)
    # Hilfsstack: speichert wie viele Elemente die Tabelle an einer bestimmten
Ebene hat
    #             und welche Reihe gerade verwendet wird
    h = []
    stack = []
    # Liste mit allen Kombinationsmöglichkeiten
    poss = []
    min_costs = 999999999999

    stack.append([e, j, table, 0])
    h.append([len(table), 0])

    costs = 0
    tmp = []
    while not finished(h):
        # oberstes Element vom Stack wird bestimmt
        top = stack[len(stack) - 1]
        # t ist die Tabelle mit Preisen
        t = top[2]

        if V:
```

```
            print(h)
            print(top[0], top[1], top[3])
            show_table(t)

        row = t[h[len(h) - 1][1]]
        tmp.append(row)

        costs += row['c']
        new = [
            top[0] - row['e'],
            top[1] - row['j'],
            prices.prices(top[0] - row['e'], top[1] - row['j'], we),
            top[3] + 1
        ]

        t = adjust(t)

        h[len(h) - 1][1] += 1

        if new[0] == new[1] == 0:
            if V:
                print(costs)
                print(tmp)
                print()

print("######################################################################")

            poss.append([costs, sorted(tmp, key=lambda k: k['c'], reverse=True)])
            if costs < min_costs:
                min_costs = costs

            p = tmp.pop()
            costs -= p['c']
            while len(h) != 0 and h[len(h) - 1][0] == h[len(h) - 1][1]:
                stack.pop()
                h.pop()
                if len(tmp) > 0:
                    p = tmp.pop()
                    costs -= p['c']

        if len(new[2]) != 0:
            stack.append(new)
            h.append([len(new[2]), 0])

    poss = sorted(poss, key=lambda k: k[0])
    poss = remove_duplicates(poss)

    # Gutscheine anwenden:
    if not fe:
        new_poss = []
        g_copy = g

        for comb in poss:
            # Bis nur noch ein Gutschein übrig bleibt werden alle Gutscheine
eingelöst
            while g > 1:
                # suchen ob noch ein Erwachsener mit Einzelkarte übrig ist
                i = find(comb[1], 'e', 'ind', 1, 1)
                # wenn ein Erwachsener mit Einzelkarte übrig ist
                if i == -1:
                    # suchen ob noch ein Jugendlicher mit Einzelkarte übrig ist
                    i = find(comb[1], 'j', 'ind', 1, 1)
                # wenn E oder J übrig -> Gutschein anwenden
                if i != -1:
                    c = comb[1][i]['c']
                    comb[1][i]['c'] = 0
                    comb[1][i]['c_s'] = 0
                    comb[1][i]['ind'] = 0
                    comb[0] -= c
                    g -= 1
                # sonst Schleife beenden
                else:
                    break

            # letzter Gutschein -> ein mal als Gutschein für die gesamte Gruppe
```

```
verwenden
            #                   -> ein mal für Einzelperson verwenden
            if g > 0:
                comb_gruppengutschein = copy.deepcopy(comb)
                comb_gruppengutschein[0] *= -9/10
                i = find(comb[1], 'e', 'ind', 1, 1)
                if i == -1:
                    i = find(comb[1], 'j', 'ind', 1, 1)
                if i != -1:
                    c = comb[1][i]['c']
                    comb[1][i]['c'] = 0
                    comb[1][i]['c_s'] = 0
                    comb[1][i]['ind'] = 0
                    comb[0] -= c
                    g -= 1
                new_poss.append(comb_gruppengutschein)
            g = g_copy
        poss += new_poss

    poss = remove_duplicates(poss)
    poss = sorted(poss, key=lambda k: abs(k[0]))

    # bestes Ergebnis zurückgeben
    return poss[:1]


def find(lst, key1, key2, val1, val2):
    for i, d in enumerate(lst):
        if d[key1] == val1 and d[key2] == val2:
            return i
    return -1


if __name__ == '__main__':
    if len(sys.argv) == 7:
        we = sys.argv[1].lower() == 't'
        fe = sys.argv[2].lower() == 't'
        e = int(sys.argv[3])
        j = int(sys.argv[4])
        k = int(sys.argv[5])
        g = int(sys.argv[6])
    else:
        we = input("Wochenende (T|f):\n> ").lower() != 'f'
        fe = input("Ferien (T|f):\n> ").lower() != 'f'
        e = int(input("Erwachsene:\n> "))
        j = int(input("Jugentliche:\n> "))
        k = int(input("Kinder:\n> "))
        g = int(input("Gutscheine:\n> "))

    print()
    print("Bester Preis wird berechnet für:")
    print("Wochenende:\t", we)
    print("Ferien:\t\t", fe)
    print("Erwachsene:\t", e)
    print("Jugendl.:\t", j)
    print("Kinder:\t\t", k)
    print("Gutscheine:\t", g)
    print()

    lst = main(we, fe, g, k, j, e)

    for p in lst:
        s = "mit Gruppengutschein" if p[0] < 0 else ""
        print(abs(p[0]), s)
        prices.show_table(p[1])
```

prices.py

```
def price_list(we):
    prices = {
            'e':  350,
            'j':  250,
            'f':  800,
            't': 1100
            }
    if not we:
        prices['e'] = prices['e'] * 4 / 5
        prices['j'] = prices['j'] * 4 / 5
    else:
        prices.pop('t')

    return prices


def sort_and_remove_duplicates(t):
    t = sorted(t, key=lambda k: k['ind'])

    c = []
    tmp = [0, 0, 0, 0]
    for e in t:
        comp = [e['e'], e['j'], e['c'], e['c_s']]
        b = True
        for i in range(len(tmp)):
            if tmp[i] != comp[i]:
                b = False
        if not b:
            c.append(e)
        tmp = comp
    return c


def remove_bad_items(t):
    end = 0
    for i in range(len(t)):
        if t[i]['ind'] > 1:
            end = i
            break
    return t[:end]



def generate_groups(e, j, prices):
    groups = []
    # Tageskarten
    t = []
    m = 6

    if 't' in prices:
        for i in range(min(e, 6) + 1):
            for k in range(min(j, 6) + 1):
                if not i == k == 0 and i + k <= 6:
                    tmp = {
                        'e': i,
                        'j': k,
                        'c': prices['t'],
                        'c_s': prices['j'] * k + prices['e'] * i,
                        'ind': prices['t'] / (prices['j'] * k + prices['e'] * i)
                    }
                    t.append(tmp)

        t = sort_and_remove_duplicates(t)
        # print("Tageskarte:")
        # show_table(t)
        # print()
        t = remove_bad_items(t)

        groups.extend(t)

    # Familenkarte
    # 1. 1E; 3J
```

```
    # 2. 2E; 2J
    f = []

    if e >= 1 and j >= 3:
        tmp = {
            'e': 1,
            'j': 3,
            'c': prices['f'],
            'c_s': prices['j'] * 3 + prices['e'] * 1,
            'ind': prices['f'] / (prices['j'] * 3 + prices['e'] * 1)
        }
        f.append(tmp)

    max_e = min(2, e)
    max_j = min(2, j)

    for i in range(max_e+1):
        for k in range(max_j+1):
            if not i == k == 0:
                tmp = {
                    'e': i,
                    'j': k,
                    'c': prices['f'],
                    'c_s': prices['j'] * k + prices['e'] * i,
                    'ind': prices['f'] / (prices['j'] * k + prices['e'] * i)
                }
                f.append(tmp)
    f = sort_and_remove_duplicates(f)
    # print("Familienkarte:")
    # show_table(f)
    # print()
    f = remove_bad_items(f)

    groups.extend(f)

    # Einzelpreise

    # Adult
    if e > 0:
        tmp = {
            'e': 1,
            'j': 0,
            'c': prices['e'],
            'c_s': prices['e'],
            'ind': 1
        }
        groups.append(tmp)

    # Adulescent
    if j > 0:
        tmp = {
            'e': 0,
            'j': 1,
            'c': prices['j'],
            'c_s': prices['j'],
            'ind': 1
        }
        groups.append(tmp)

    return sort_and_remove_duplicates(groups)


def show_table(groups):
    import texttable as tt
    tab = tt.Texttable()
    headings = ['Erw.', 'Jug.', 'Kosten', 'Kosten einzeln', 'index']
    tab.header(headings)

    for g in groups:
        tab.add_row((g['e'], g['j'], g['c'], g['c_s'], g['ind']))

    s = tab.draw()
    print(s)
    print()
```

```
def prices(e, j, we):
    prices = price_list(we)
    groups = generate_groups(e, j, prices)
    return groups

def main():
    inp = ''
    we  = None
    h   = None
    e   = 0
    j   = 0
    g   = None

    we, h, e, j = False, True, 5, 7

    # print(we, h, e, j, g)
    print(e, j, we, 0)
    prices = price_list(we)
    for p in prices:
        print(p, prices[p])
    groups = generate_groups(e, j, prices)

    show_table(groups)


if __name__ == '__main__':
    main()
```

Aufgabe 3

```
#! /usr/bin/env python3
# -*- coding: UTF-8 -*-

# by Anton Baumann

import sys


def calculate_intersection(a, b):
    x = (a[0][0], a[1][0], b[0][0], b[1][0])
    y = (a[0][1], a[1][1], b[0][1], b[1][1])

    denominator = (y[3] - y[2]) * (x[1] - x[0]) - (y[1] - y[0]) * (x[3] - x[2])
    if denominator == 0:
        return None

    # höchster und niedrigster y und x Wert von a
    upper_x_a = max(a[0][0], a[1][0])
    lower_x_a = min(a[0][0], a[1][0])
    upper_y_a = max(a[0][1], a[1][1])
    lower_y_a = min(a[0][1], a[1][1])

    # höchster und niedrigster y und x Wert von a
    upper_x_b = max(b[0][0], b[1][0])
    lower_x_b = min(b[0][0], b[1][0])
    upper_y_b = max(b[0][1], b[1][1])
    lower_y_b = min(b[0][1], b[1][1])

    s_x = ((x[3] - x[2]) * (x[1] * y[0] - x[0] * y[1]) - (x[1] - x[0]) * (x[3] *
y[2] - x[2] * y[3])) / denominator
    s_y = ((y[0] - y[1]) * (x[3] * y[2] - x[2] * y[3]) - (y[2] - y[3]) * (x[1] *
y[0] - x[0] * y[1])) / denominator

    if s_x == -0.0: s_x = 0.0
    if s_y == -0.0: s_y = 0.0
    S = (s_x, s_y)

    if lower_x_a <= s_x <= upper_x_a and lower_x_b <= s_x <= upper_x_b:
        if lower_y_a <= s_y <= upper_y_a and lower_y_b <= s_y <= upper_y_b:
            return S
    return None


def read_file(filename):
    with open(filename, 'r') as f:
        lst = f.readlines()
    length = int(lst.pop(0).replace('\n', ''))
    punkt_geraden, gerade_punkte, punkte = {}, {}, set()

    for i in range(length):
        tmp = lst[i].replace('\n', '').split()
        tmp = [float(x) for x in tmp]
        line = [(tmp[0], tmp[1]), (tmp[2], tmp[3])]
        punkte.update(line)
        for p in line:
            if p not in punkt_geraden:
                punkt_geraden.setdefault(p, set()).add(i)
            else:
                punkt_geraden.get(p).add(i)
        gerade_punkte[i] = line
    return punkt_geraden, gerade_punkte, punkte


# returns punkt_geraden, gerade_punkte, punkte
def process(punkt_geraden, gerade_punkte, punkte):
    for g1 in range(len(gerade_punkte)):
        for g2 in range(g1 + 1, len(gerade_punkte)):
            schnittpunkt = calculate_intersection(gerade_punkte[g1],
gerade_punkte[g2])
            if schnittpunkt is None:
                continue
            if schnittpunkt in punkte:
```

```
                if schnittpunkt not in gerade_punkte[g1]:
                    gerade_punkte[g1].append(schnittpunkt)
                elif schnittpunkt not in gerade_punkte[g2]:
                    gerade_punkte[g2].append(schnittpunkt)
                punkt_geraden[schnittpunkt].add(g1)
                punkt_geraden[schnittpunkt].add(g2)
            else:
                gerade_punkte[g1].append(schnittpunkt)
                gerade_punkte[g2].append(schnittpunkt)
                punkt_geraden.setdefault(schnittpunkt, set()).add(g1)
                punkt_geraden.setdefault(schnittpunkt, set()).add(g2)
            punkte.add(schnittpunkt)


def modified_dfs(punkt_geraden, gerade_punkte):
    dreiecke = set()
    # iteriere über jeden Punkt im Graphen
    for p1, geraden in punkt_geraden.items():
        dreieck = [None, None, None]
        for g1 in geraden:
            e1 = set(gerade_punkte[g1])
            e1.remove(p1)
            dreieck[0] = p1
            for p2 in e1:
                dreieck[1] = p2
                g2 = punkt_geraden[p2] - {g1}
                if len(g2) == 0:
                    continue
                g2 = g2.pop()
                tmp = set(gerade_punkte[g2]) - {p2}
                e2 = tmp

                for p3 in e2:
                    dreieck[2] = p3
                    e3 = set()
                    for g3 in punkt_geraden[p3]:
                        e3.update(gerade_punkte[g3])
                    if p1 in e3:
                        dreiecke.add(tuple(sorted(dreieck)))
    return dreiecke


def main(b, path):
    punkt_geraden, gerade_punkte, punkte = read_file(path)
    process(punkt_geraden, gerade_punkte, punkte)
    dreiecke = modified_dfs(punkt_geraden, gerade_punkte)

    if b:
        import matplotlib.pyplot as plt
        for p in punkte:
            plt.plot(p[0], p[1], marker='o', color='grey')
        for k, v in gerade_punkte.items():
            plt.plot([v[0][0], v[1][0]], [v[0][1], v[1][1]], color='black')
        plt.show()

        for i, d in enumerate(dreiecke):
            print(str(i+1) + " -> " + str(d))
            for p in punkte:
                plt.plot(p[0], p[1], marker='o', color='grey')
            for k, v in gerade_punkte.items():
                plt.plot([v[0][0], v[1][0]], [v[0][1], v[1][1]], color='black')
            plt.plot([d[0][0], d[1][0], d[2][0], d[0][0]], [d[0][1], d[1][1],
d[2][1], d[0][1]],
                     color='y', linewidth=5)
            plt.show()
    else:
        for i, d in enumerate(dreiecke):
            print(str(i+1) + " -> " + str(d))

    print("Anzahl Dreiecke: ", len(dreiecke))


if __name__ == '__main__':
    if len(sys.argv) < 2:
        print("dreiecke.py")
```

```
    print("Usage:\n", "dreiecke.py FILE [--visualize|-v]")
    sys.exit(0)

file_path = sys.argv[1]
visualize = False
if len(sys.argv) >= 3:
    if sys.argv[2] in ['--visualize', '-v']:
        visualize = True
main(visualize, file_path)
```

Aufgabe 4:

```
#!/usr/bin/env python3
# -*- coding: UTF-8 -*-

import time

# Debug print() an/aus
import sys

def open_file(name):
    with open(name, 'r', encoding='UTF-8') as file:
        return sorted({line.strip() for line in file.readlines()})

# mit bin_search und ohne output:    0.00542 s
# mit naiver suche und ohne output:  0.81236 s
# O(log n)
def binary_search(lst, target):
    min = 0
    max = len(lst) - 1
    avg = (min + max) // 2
    while min < max:
        if lst[avg] == target:
            return True
        elif lst[avg] < target:
            return binary_search(lst[avg + 1:], target)
        else:
            return binary_search(lst[:avg], target)
    return False

# überprüft ob String s auf einem einzelnen Numernschild darstellbar ist
# O(log n)
def is_possible(s):
    if len(s) > 5:
        return False
    umlaute = set('ÄÖÜ')
    s = s.upper().strip()
    for i in range(1, min(len(s), 4)):  # O(1)
        # Teilt String in 2 Teile
        pre = s[:i]
        post = s[i:]
        # Zweiter Teil darf laut Aufgabestellung nicht >2 sein
        if len(post) > 2:
            continue
        # Ueberprueft ob pre in kuerzelliste ist
        # Ueberprueft ob Umlaute in post
        # triviale suche ->
        # if pre in KUERZEL and not any((c in umlaute) for c in post):
        if not any((c in umlaute) for c in post) and binary_search(KUERZEL, pre):
            return True
    return False


def check(word):
    # START, STOP, STEP = 2, 5, 1
    START, STOP, STEP = 5, 2, -1
    stack = [[0, START]]
    i, n = 0, STOP
    while i < len(word):
        if v: print(stack)
        i, n = stack[len(stack) - 1]
        tmp = word[i:i + n]
        if is_possible(tmp):
            stack.append([i + n, START])
        else:
            if STEP < 0 and n > STOP or STEP > 0 and n < STOP:
                i, n = stack.pop()
                n += STEP
                stack.append([i, n])
            else:
                stack.pop()
                if len(stack) == 0:
                    print("Nicht darstellbar!")
```

```
                    return False
                i, n = stack.pop()
                n += STEP
                stack.append([i, n])

    stack.pop()  # letzter vorberechneter eintrag wird gelöscht

    print("Darstellbar!")

    comb = [word[i: i + n] for i, n in stack]
    print(comb)
    return True


if __name__ == '__main__':
    global v
    v = False
    if len(sys.argv) < 3:
        print("Usage:\n", "./autoscrabble.py kuerzelliste wordlist")
        sys.exit()
    if len(sys.argv) >= 4:
        if sys.argv[3] == '-v':
            v = True

    global KUERZEL
    KUERZEL = open_file(sys.argv[1])
    AUTOSCRABBLE = open_file(sys.argv[2])
    start = time.time()
    for word in AUTOSCRABBLE:
        print(word)
        check(word)
        print()
    end = time.time()
    print(end - start, 's')
```

Aufgabe 5:

Teilaufgabe 1:

```
import copy
import re
import collections

import time

letters = "ABCDEFGH"
digits = "87654321"

# Bsp: 'A1' -> 7, 0
def str_to_coord(s):
    r = re.compile('[A-H][1-8]')
    if r.match(s):
        return digits.find(s[0]), letters.find(s[1])

# Bsp: (7, 0) -> 'A1'
def coord_to_str(pos):
    return coords_to_str(pos[0], pos[1])

# Bsp: 7, 0 -> 'A1'
def coords_to_str(y, x):
    if 0 <= y <= 7 and 0 <= x <= 7:
        return letters[x] + digits[y]

# Bsp: [(7, 0), (7, 1)] -> ['A1', 'B1']
def coord_list_to_str(lst):
    return [coord_to_str(x) for x in lst]

# Gibt zurück ob p von einem Bauern erreicht werden kann
def is_threatened(p, board):
    surr = {(p[0] - 1, p[1]), (p[0] + 1, p[1]), (p[0], p[1] - 1), (p[0], p[1] +
1)}
    surr = {pos for pos in surr if 0 <= pos[0] <= 7 and 0 <= pos[1] <= 7}

    for box in surr:
        if board[box[0]][box[1]].is_pawn():
            return True
    return False

# Oberklasse: Schachfigur
class Chessman:
    def __init__(self, y, x):
        self.pos = (y, x)

    def is_pawn(self):
        return False

    def is_rook(self):
        return False


# Leeres Feld
class EmptyField(Chessman):
    CHAR = '.'

    def __str__(self):
        return self.CHAR


# Bauer
class Pawn(Chessman):
    CHAR = '♙'

    def is_pawn(self):
        return True

    def __str__(self):
        return self.CHAR
```

```
    def possible_moves(self, pawns):
        p = self.pos
        moves = {(p[0]-1, p[1]), (p[0]+1, p[1]), (p[0], p[1]-1), (p[0], p[1]+1)}
        moves = {pos for pos in moves if 0 <= pos[0] <= 7 and 0 <= pos[1] <= 7}
        moves -= set(pawns)
        return sorted(moves)


# Turm
class Rook(Chessman):
    CHAR = '♜'

    def is_rook(self):
        return True

    def __str__(self):
        return self.CHAR

    def possible_moves(self, board):
        p = self.pos
        moves = []
        # Top
        for y in range(p[0], -1, -1):
            if board[y][p[1]].is_pawn():
                break
            if not is_threatened((y, p[1]), board):
                moves.append((y, p[1]))
        # Bottom
        for y in range(p[0], 8, 1):
            if board[y][p[1]].is_pawn():
                break
            if not is_threatened((y, p[1]), board):
                moves.append((y, p[1]))
        # Left
        for x in range(p[1], -1, -1):
            if board[p[0]][x].is_pawn():
                break
            if not is_threatened((p[0], x), board):
                moves.append((p[0], x))
        # Right
        for x in range(p[1], 8, 1):
            if board[p[0]][x].is_pawn():
                break
            if not is_threatened((p[0], x), board):
                moves.append((p[0], x))

        return sorted(set(moves))


# Spielbrett
class Board:
    def __init__(self):
        self.boxes = [[EmptyField(y, x) for x in range(8)] for y in range(8)]
        self.pawns = []
        self.rooks = []

    def add_pawn(self, y, x):
        p = Pawn(y, x)
        self.pawns.append(p)
        self.boxes[y][x] = p

    def add_rook(self, y, x):
        r = Rook(y, x)
        self.rooks.append(r)
        self.boxes[y][x] = r

    def move(self, chessman, to):
        if chessman.is_pawn() or chessman.is_rook():
            self.boxes[to[0]][to[1]] =
self.boxes[chessman.pos[0]][chessman.pos[1]]
            if not to == chessman.pos:
                self.boxes[chessman.pos[0]][chessman.pos[1]] =
EmptyField(chessman.pos[0], chessman.pos[1])
            chessman.pos = to
```

```
    def __str__(self):
        string = ""
        n = 8
        for r in self.boxes:
            string += str(n) + " "
            n -= 1
            for c in r:
                string += str(c) + " "
            string += "\n"
        string += "  A B C D E F G H"
        return string

    def print_pawns(self):
        s = ""
        for p in self.pawns:
            s += str(coord_to_str(p.pos))
        print(s)

    # überprüft ob Bauern eine Kette bilden
    def check_chain(self):
        tmp = self.pawns[0].pos[0]
        if self.pawns[0].pos[1] != 0:
            return False
        for i in range(1, 8):
            if abs(tmp - self.pawns[i].pos[0]) > 1 or self.pawns[i].pos[1] != i:
                return False
            tmp = self.pawns[i].pos[0]
        return True

# enthält Spiellogik und speichert Aufstellung in Klasse Board
class Game:
    B = Board()
    rook_next = True

    def __init__(self):
        self.B.add_pawn(4, 0)
        self.B.add_pawn(4, 1)
        self.B.add_pawn(4, 2)
        self.B.add_pawn(4, 3)
        self.B.add_pawn(4, 4)
        self.B.add_pawn(4, 5)
        self.B.add_pawn(4, 6)
        self.B.add_pawn(4, 7)
        self.B.add_rook(0, 7)

    # Ruft possible_moves in Pawn auf.
    # Übergibt eine Liste der Positionen aller Bauern
    def possible_moves_pawn(self, pawn, pawns_pos=B.pawns):
        return pawn.possible_moves(pawns_pos)

    # Ruft possible_moves in Rook auf.
    # Übergibt Spielfeld
    def possible_moves_rook(self, rook, boxes=B.boxes):
        return rook.possible_moves(boxes)

    # Berechnet Besten Zug für die Bauern
    def pawn_next_move(self):
        print("WHITE is moving")
        pawns = self.B.pawns
        best_moves = {}

        for i, pawn in enumerate(pawns):
            moves = self.possible_moves_pawn(pawn)
            for target_square in moves:
                if self.B.rooks[0].pos == target_square:
                    self.B.move(pawn, target_square)
                    return
                B_copy = copy.deepcopy(self.B)
                start_square = pawn.pos
                B_copy.move(B_copy.pawns[i], target_square)
                nr_moves_rook = len(self.possible_moves_rook(B_copy.rooks[0],
boxes=B_copy.boxes))
                if nr_moves_rook in best_moves:
                    best_moves[nr_moves_rook].append((i, target_square,
```

```
nr_moves_rook))
                else:
                    best_moves[nr_moves_rook] = [(i, target_square,
nr_moves_rook)]
        best_moves = collections.OrderedDict(sorted(best_moves.items()))
        for k, v in best_moves.items():
            for m in v:
                B_copy = copy.deepcopy(self.B)
                B_copy.move(B_copy.pawns[m[0]], m[1])
                if B_copy.check_chain():
                    print("Best move:", coord_to_str(self.B.pawns[m[0]].pos), '-',
coord_to_str(m[1]), ':', m[2])
                    self.B.move(self.B.pawns[m[0]], m[1])
                    return

    # Berechnet besten Zug für den Turm
    def rook_next_move(self):
        if not self.is_running():
            return
        print("BLACK is moving")
        moves = self.possible_moves_rook(self.B.rooks[0])
        best_move = [None, -1]

        for target_square in moves:
            B_copy = copy.deepcopy(self.B)
            B_copy.move(B_copy.rooks[0], target_square)
            nr_moves_rook = len(self.possible_moves_rook(B_copy.rooks[0],
boxes=B_copy.boxes))
            if nr_moves_rook > best_move[1]:
                best_move = [target_square, nr_moves_rook]
        if len(moves) == 0:
            print("Can't move")
        else:
            print("Best move:", coord_to_str(self.B.rooks[0].pos), '-',
coord_to_str(best_move[0]), ':', best_move[1])
            self.B.move(self.B.rooks[0], best_move[0])

    def is_running(self):
        for row in self.B.boxes:
            for box in row:
                if box.is_rook():
                    return True
        return False


def main():
    G = Game()
    print(G.B)
    c = 1
    while G.is_running():
        time.sleep(0.5)
        print('\n')
        print(str(c)+'.')
        G.pawn_next_move()
        print(G.B)
        c += 1
        time.sleep(0.5)
        print('\n')
        print(str(c)+'.')
        G.rook_next_move()
        print(G.B)
        c += 1
    print("WHITE won!")


if __name__ == '__main__':
    main()
```

Teilaufgabe 2:

```
import copy
import re

import time

letters = "ABCDEFGH"
digits = "87654321"

# Bsp: 'A1' -> 7, 0
def str_to_coord(s):
    r = re.compile('[A-H][1-8]')
    if r.match(s):
        return digits.find(s[0]), letters.find(s[1])

# Bsp: (7, 0) -> 'A1'
def coord_to_str(pos):
    return coords_to_str(pos[0], pos[1])

# Bsp: 7, 0 -> 'A1'
def coords_to_str(y, x):
    if 0 <= y <= 7 and 0 <= x <= 7:
        return letters[x] + digits[y]

# Bsp: [(7, 0), (7, 1)] -> ['A1', 'B1']
def coord_list_to_str(lst):
    return [coord_to_str(x) for x in lst]

# Gibt zurück ob p von einem Bauern erreicht werden kann
def is_threatened(p, board):
    surr = {(p[0] - 1, p[1]), (p[0] + 1, p[1]), (p[0], p[1] - 1), (p[0], p[1] +
1)}
    surr = {pos for pos in surr if 0 <= pos[0] <= 7 and 0 <= pos[1] <= 7}

    for box in surr:
        if board[box[0]][box[1]].is_pawn():
            return True
    return False

# Oberklasse: Schachfigur
class Chessman:
    def __init__(self, y, x):
        self.pos = (y, x)

    def is_pawn(self):
        return False

    def is_rook(self):
        return False


# Leeres Feld
class EmptyField(Chessman):
    CHAR = '.'

    def __str__(self):
        return self.CHAR


# Bauer
class Pawn(Chessman):
    CHAR = '♙'

    def is_pawn(self):
        return True

    def __str__(self):
        return self.CHAR

    def possible_moves(self, pawns):
        p = self.pos
        moves = {(p[0]-1, p[1]), (p[0]+1, p[1]), (p[0], p[1]-1), (p[0], p[1]+1)}
        moves = {pos for pos in moves if 0 <= pos[0] <= 7 and 0 <= pos[1] <= 7}
```

```
        moves -= set(pawns)
        return sorted(moves)


# Turm
class Rook(Chessman):
    CHAR = '♜'

    def is_rook(self):
        return True

    def __str__(self):
        return self.CHAR

    def possible_moves(self, board):
        p = self.pos
        moves = []
        # Top
        for y in range(p[0], -1, -1):
            if board[y][p[1]].is_pawn():
                break
            if not is_threatened((y, p[1]), board):
                moves.append((y, p[1]))
        # Bottom
        for y in range(p[0], 8, 1):
            if board[y][p[1]].is_pawn():
                break
            if not is_threatened((y, p[1]), board):
                moves.append((y, p[1]))
        # Left
        for x in range(p[1], -1, -1):
            if board[p[0]][x].is_pawn():
                break
            if not is_threatened((p[0], x), board):
                moves.append((p[0], x))
        # Right
        for x in range(p[1], 8, 1):
            if board[p[0]][x].is_pawn():
                break
            if not is_threatened((p[0], x), board):
                moves.append((p[0], x))

        return sorted(set(moves))


# Spielbrett
class Board:
    def __init__(self):
        self.boxes = [[EmptyField(y, x) for x in range(8)] for y in range(8)]
        self.pawns = []
        self.rooks = []

    def add_pawn(self, y, x):
        p = Pawn(y, x)
        self.pawns.append(p)
        self.boxes[y][x] = p

    def add_rook(self, y, x):
        r = Rook(y, x)
        self.rooks.append(r)
        self.boxes[y][x] = r

    def move(self, chessman, to):
        if chessman.is_pawn() or chessman.is_rook():
            self.boxes[to[0]][to[1]] =
self.boxes[chessman.pos[0]][chessman.pos[1]]
            if not to == chessman.pos:
                self.boxes[chessman.pos[0]][chessman.pos[1]] =
EmptyField(chessman.pos[0], chessman.pos[1])
            chessman.pos = to

    def __str__(self):
        string = ""
        n = 8
        for r in self.boxes:
```

```
            string += str(n) + " "
            n -= 1
            for c in r:
                string += str(c) + " "
            string += "\n"
        string += "  A B C D E F G H"
        return string

    def print_pawns(self):
        s = ""
        for p in self.pawns:
            s += str(coord_to_str(p.pos))
        print(s)


class Game:
    B = Board()
    rook_next = True

    def __init__(self):
        self.B.add_pawn(4, 0)
        self.B.add_pawn(4, 1)
        self.B.add_pawn(4, 2)
        self.B.add_pawn(4, 3)
        self.B.add_pawn(4, 4)
        self.B.add_pawn(4, 5)
        self.B.add_pawn(4, 6)
        self.B.add_rook(0, 7)

    # Ruft possible_moves in Pawn auf.
    # Übergibt eine Liste der Positionen aller Bauern
    def possible_moves_pawn(self, pawn, pawns_pos=B.pawns):
        return pawn.possible_moves(pawns_pos)

    # Ruft possible_moves in Rook auf.
    # Übergibt Spielfeld
    def possible_moves_rook(self, rook, boxes=B.boxes):
        return rook.possible_moves(boxes)

    # Berechnet Besten Zug für die Bauern
    def pawn_next_move(self):
        print("WHITE is moving")
        pawns = self.B.pawns
        best_move = [None, None, 999, None]
        for i, pawn in enumerate(pawns):
            moves = self.possible_moves_pawn(pawn)
            for target_square in moves:
                B_copy = copy.deepcopy(self.B)
                start_square = pawn.pos
                B_copy.move(B_copy.pawns[i], target_square)
                nr_moves_rook = len(self.possible_moves_rook(B_copy.rooks[0],
boxes=B_copy.boxes))
                if nr_moves_rook < best_move[2]:
                    best_move = [B_copy.pawns[i], target_square, nr_moves_rook, i]
        print("Best move:", coord_to_str(start_square), '-',
coord_to_str(best_move[1]), ':',best_move[2])
        self.B.move(self.B.pawns[best_move[3]], best_move[1])

    # Berechnet besten Zug für den Turm
    def rook_next_move(self):
        print("BLACK is moving")
        moves = self.possible_moves_rook(self.B.rooks[0])
        best_move = [None, -1]

        for target_square in moves:
            B_copy = copy.deepcopy(self.B)
            B_copy.move(B_copy.rooks[0], target_square)
            nr_moves_rook = len(self.possible_moves_rook(B_copy.rooks[0],
boxes=B_copy.boxes))
            if nr_moves_rook > best_move[1]:
                best_move = [target_square, nr_moves_rook]
        print("Best move:", coord_to_str(self.B.rooks[0].pos), '-',
coord_to_str(best_move[0]), ':', best_move[1])
        self.B.move(self.B.rooks[0], best_move[0])
```

```
    def is_running(self):
        for row in self.B.boxes:
            for box in row:
                if box.is_rook():
                    return True
        return False


def main():
    G = Game()
    print(G.B)
    c = 1
    while G.is_running():
        time.sleep(0.5)
        print('\n')
        print(str(c)+'.')
        G.pawn_next_move()
        print(G.B)
        c += 1
        time.sleep(0.5)
        print('\n')
        print(str(c)+'.')
        G.rook_next_move()
        print(G.B)
        c += 1
    print("WHITE won!")


if __name__ == '__main__':
    main()
```