Informatik I: Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Tim Schulte
Wintersemester 2018/2019

Universität Freiburg
Institut für Informatik

Übungsblatt 11

Abgabe: Dienstag, 15.1.2018, 20:00 Uhr

Wichtig: Anmeldung Probeklausur am 15.1.2019 Am 15.1. findet zur Vorlesungszeit von 10-13 Uhr eine Probeklausur in den Hörsälen 101-026 und 101-036 statt. Die Probeklausur entspricht zwei Übungsblättern (40 Punkte) und die daraus erzielten Punkte sind Bonuspunkte, die für die Studienleistung angerechnet werden. Bitte melden Sie sich bis zum 13.1. um 12:00 dafür an, indem Sie im Unterverzeichnis `sheet11` eine Textdatei `anmeldung.txt` anlegen **und committen**. Der Inhalt der Datei spielt keine Rolle.
Wichtiger Hinweis: Zur Bearbeitung der Übungsaufgaben legen Sie bitte ein neues Unterverzeichnis `sheet11` im Wurzelverzeichnis Ihrer Arbeitskopie des SVN-Repositories an. Ihre Lösungen werden dann in Dateien in diesem Unterverzeichnis erwartet.

Aufgabe 11.1 (Kryptoanalyse; Dateien: `caesar.py`, `crack_caesar.py`; Punkte: 1+5)

Laden Sie die Datei `caesar.py` von der Webseite der Übungen zur Vorlesung herunter und speichern Sie diese im Unterverzeichnis zu diesem Übungsblatt. In dieser Datei wird die Funktion `caesar(text : str, shift : int, charset=ascii_letters)` definiert, die die Cäsarverschiebung[1] implementiert. Dabei wird in dem an die Funktion übergebenen String `text` jedes im Zeichensatz `charset` vorkommende Zeichen um `shift` viele Zeichen verschoben. Alle anderen Zeichen bleiben in der zurückgegebenen Zeichenfolge unverändert.
Schreiben Sie ein Programm `crack_caesar.py` mit einer Funktion `crack_caesar (input : str) -> tuple`, die als Parameter einen Cäsar-verschlüsselten String nimmt und dann versucht, den enthaltenen Text automatisch und ohne Kenntnis der tatsächlichen Verschiebung zu entschlüsseln. Zurückgegeben werden soll ein Tupel aus der ermittelten Verschiebung und dem (im besten Fall korrekt) entschlüsselten Text.

Ihre Entschlüsselung soll die folgende Idee implementieren: die (vermutete) Entschlüsselung ist ein Text mit einer Buchstabenverteilung, die der Buchstabenverteilung in natürlichsprachlichen Texten am ähnlichsten ist (für verschiedene Sprachen finden Sie die zu erwartende Verteilung z.B. auf `https://de.wikipedia.org/wiki/Buchstabenhäufigkeit`).

(a) Erstellen Sie ein Dictionary, das den Schlüsseln `'a'`–`'z'` die erwartete Häufigkeit des jeweiligen Buchstabens in einem englischen Text zuordnet. Das Dictionary soll in der Variable `frequency_english` abgelegt werden.

(b) Ihr Programm soll zunächst für die verschiedenen (Rück-)Verschiebungen ein Dictionary mit den relativen Häufigkeiten $p(c)$ der im Text vorkommenden Buchstaben $c \in \{\texttt{'a'}, \ldots, \texttt{'z'}\}$ (Groß- und Kleinbuchstaben werden nicht unterschieden) bestimmen. Diese werden jeweils mit den zu erwartenden relativen Häufigkeiten $\bar{p}(c)$ der Sprache des zu entschlüsselnden Textes verglichen. Dabei gehen wir davon aus, dass der verschlüsselte Text in "Englisch" ist. Zum

[1] `https://de.wikipedia.org/wiki/Caesar-Verschlüsselung`

Vergleich zweier Häufigkeitsverteilungen verwenden Sie den sogenannten Chi-Quadrat-Test, der ein Maß für die Größe der Abweichung liefert:

$$\chi^2 = \sum_{c \in \texttt{'a'},\ldots,\texttt{'z'}} \frac{(p(c) - \bar{p}(c))^2}{\bar{p}(c)}.$$

Das Maß ist umso kleiner, je ähnlicher sich p und $\bar{p}$ sind.

Hinweis: Aus der Datei `caesar.py` können Sie in der Datei `crack_caesar.py` die Funktion `caesar` wie folgt importieren:

```
from caesar import caesar
```

Aufgabe 11.2 (Objekte; Datei: `fraction.py`; Punkte: 12 = 2 + 1 + 3 + 3 + 3)

Entwerfen und implementieren Sie eine Klasse `Fraction` zum exakten Rechnen mit Brüchen auf der Basis der vorhandenen arithmetischen Operationen auf `int`. Die interne Repräsentation soll dabei verkapselt sein. Instanzen von `Fraction` sollen nicht veränderlich sein. **Verwenden Sie keine Gleitkommazahlen!**
Implementieren Sie folgende Methoden. Einige Teilaufgaben erfordern die Definition von geeigneten magischen Methoden.

- Der Konstruktor soll sicherstellen, dass der Nenner niemals 0 ist (Invariante).
- Getter für Zähler und Nenner: `numerator` und `denominator`.
- Implementieren Sie die Operatoren `+`, `-`, `*`, `/` auf Brüchen.
- Erweitern Sie diese Implementierungen, sodass sie auch funktionieren, wenn das andere Argument vom Typ `int` ist (z.B. `Fraction` + `int` und `int` + `Fraction`).
- Implementieren Sie Methoden für die Gleichheit `==` auf Brüchen, sowie die Vergleichsoperatoren `<`, `<=`, `>`, `>=` soweit erforderlich.

Aufgabe 11.3 (Erfahrungen; Datei: `erfahrungen.txt`; Punkte: 2)

Legen Sie im Unterverzeichnis `sheet11` eine Textdatei `erfahrungen.txt` an. Notieren Sie in dieser Datei kurz Ihre Erfahrungen beim Bearbeiten der Übungsaufgaben (Probleme, Bezug zur Vorlesung, Interessantes, benötigter Zeitaufwand, etc.).