Übungen zu Algorithmen und Datenstrukturen für Medizintechnik
Sommersemester 2021
René Raab, Robert Richer

Lehrstuhl für Maschinelles Lernen und Datenanalytik
Department Artificial Intelligence in Biomedical Engineering (AIBE)
(https://www.mad.tf.fau.de)

Aufgabenblatt 3 vom 02. Mai 2021, Abgabe am 16. Mai 2021, 22:00 Uhr

---

»Eine neue Programmiersprache lernt man nur, wenn man in ihr Programme schreibt.«
– *Dennis M. Ritchie, Entwickler von Unix und C*

Es reicht daher nicht aus, die Tafel- bzw. Rechnerübung zu besuchen, Sie müssen auch eigenständig zu Hause programmieren üben!

Aufgabe 3.1: Methoden-Theorie

Punkte siehe StudOn
Methoden

Aufgabenstellung und Abgabe (individuell, nicht als Gruppe!) im StudOn.

Aufgabe 3.2: Signalplotter

20 Punkte
Schleifen, Methoden, Arrays

In dieser Aufgabe soll eine grafische Benutzeroberfläche (*graphical user interface*, GUI) erstellt werden, um verschiedene Signale in 2D darzustellen. Um Ihnen die Programmierung der GUI abzunehmen, stellen wir Ihnen eine Hilfsklasse sowie Hilfsmethoden zur Verfügung.

1. **Projekt anlegen:**
   Legen Sie ein neues Projekt `03-SignalPlotter` und eine Klasse `SignalPlotter` mit einer `main`-Methode an.

2. **Bibliotheken einbinden:**
   Zuerst wird die Bibliothek zum grafischen Plotten sowie die von uns bereitgestellte Klasse in Ihr Projekt eingebunden:

   a) Laden Sie von StudOn das Zusatzmaterial zum 3. Übungsblatt herunter (`03-material.zip`) und entpacken Sie diese Datei.

   b) Binden Sie die Bibliothek `jmathplot.jar`, wie in der Übung besprochen, in Ihr Projekt ein.

   c) Kopieren Sie die Datei `PlotHelper.java` in das Verzeichnis, in dem auch ihre angelegte Klasse `SignalPlotter` liegt. Kopieren Sie die Dateien `ecg.txt` und `rpeaks.txt` in das Projektverzeichnis (nicht in den src-Ordner).

3. **Stützstellen erstellen:**
   Um ein Signal zu plotten, muss es an diskreten, äquidistanten Stützstellen ausgewertet werden. Zur Erzeugung dieser Stützstellen soll die Methode `createSamplingPoints` implementiert werden. Sie dient zum Erzeugen von linear gleichverteilten Punkten in einem vorgegebenen Intervall.

Beispiele:

- 9 Punkte im Intervall $[1, 5]$:

  1.00 1.50 2.00 2.50 3.00 3.50 4.00 4.50 5.00

- 1 Punkt im Intervall $[3.23, 55.32]$:

  55.32

- 4 Punke im Intervall $[10, 4]$:

  10.00 8.00 6.00 4.00

a) Legen Sie die Methode `createSamplingPoints` an. Diese bekommt den Anfang und das Ende des Intervalls (`firstLimit` und `secondLimit`) als Parameter übergeben, beide vom Typ `double`. Des Weiteren soll ein Parameter `numberOfPoints` geeigneten Typs übergeben werden, welcher die Anzahl n der gleichverteilten Punkte festlegt. Die Funktion soll ein Array vom Typ `double` zurückgeben.

b) Prüfen Sie zuerst, ob es sich bei den übergebenen Intervallgrenzen überhaupt um ein »richtiges« Intervall handelt oder nicht. Falls `firstLimit` und `secondLimit` gleich sind, setzen Sie `numberOfPoints` innerhalb der Methode auf den Wert 1.

c) Legen Sie ein Array passender Größe an, das später mit den Stützpunkten gefüllt wird.

d) Soll nur ein einziger Stützpunkt erzeugt werden, dann soll dieser `secondLimit` sein. Ansonsten sollen die Punkte so gewählt werden, dass der erste `firstLimit`, der letzte `secondLimit` entspricht und alle dazwischen äquidistant, also mit gleichem Abstand, verteilt sind.

Achtung: Beachten Sie, dass nicht zwingend `firstLimit` $\leq$ `secondLimit` gelten muss. Trotzdem sollen die beiden Intervallgrenzen **nicht** vertauscht werden!

Testen Sie auch diese Methode ausgiebig mit mehreren Testfällen!

Wichtig: Entfernen Sie nach dem Test den Quellcode wieder aus der `main`-Methode (oder kommentieren sie ihn aus)! Die `main`-Methode sollte ab jetzt wieder keine ausführbaren Anweisungen enthalten.

4. **2D-Plot:**
Zunächst soll unser Signalplotter erst einmal dazu in der Lage sein, eindimensionale Funktionen zu berechnen und in einem 2D-Plot (x, y) darzustellen. Als Beispiel betrachten wir folgende Funktion, die auch als *Sigmoid-Funktion* bezeichnet und sehr häufig als *Aktivierungsfunktion* in künstlichen neuronalen Netzen (*artificial neural networks, ANN*) verwendet wird:

$$sig(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

a) Implementieren Sie in Gleichung 1 gegebene Funktion in einer Methode `sigmoid`, die als Parameter einen x-Wert bekommen und den berechneten Funktionswert zurückgeben soll (beide als `double`).

Hinweis: Die `Math`-Klasse der Java-API ist hier hilfreich ($\rightsquigarrow$ Exponentionalfunktion).

b) Legen Sie eine Methode `applySigmoidToArray` an, so dass ein Array `xs` von x-Werten als Eingabeparameter übergeben werden kann und für jeden Eintrag die Sigmoid-Funktion ausgewertet wird. Die Werte sollen in einem neuen Array zurückgegeben werden.

c) Im nächsten Schritt soll die mathematische Funktion grafisch ausgegeben werden.

i. Definieren Sie die Methode `plotSigmoid` ohne Parameter und Rückgabewert.

ii. Generieren Sie in `plotSigmoid` mit Hilfe von `createSamplingPoints` 1000 Stützstellen im Intervall $[-10; 10]$. Legen Sie hierfür in der Klasse die Konstanten `FIRST_LIMIT` und `SECOND_LIMIT` an, um die Intervallgrenzen zu speichern. Verwalten Sie die Anzahl der Stützstellen mit Hilfe einer Konstanten `NUMBER_OF_POINTS`. Beachten Sie dabei, dass wir alle drei Konstanten später auch in anderen Methoden der Klasse `SignalPlotter` benötigen. Legen Sie sie also als Klassenkonstanten, das heißt außerhalb einer Methode, aber innerhalb der Klasse, an.

`public static` davor nicht vergessen!

iii. Werten Sie die Funktion auf allen Stützstellen aus und speichern Sie das Ergebnis in einer lokalen Variable.

iv. Zum Plotten haben wir Ihnen die Klasse `PlotHelper` gegeben. Rufen Sie hierfür die `plot2D`-Methode **der Klasse `PlotHelper`** mittels `PlotHelper.plot2D(...)` auf. Diese bekommt als ersten Parameter das Stützstellen-Array und als zweiten Parameter die berechneten Funktionswerte (auch als Array) übergeben.

Anmerkung: Falls Sie mehr über die von uns zur Verfügung gestellte Hilfsklasse wissen möchten, dann schauen Sie sich doch einfach den Quellcode der Klasse `PlotHelper` an.

v. Rufen Sie aus der `main`-Methode die Methode `plotSigmoid` auf. Wenn Sie jetzt das Programm ausführen, müssten Sie folgende Ausgabe sehen:

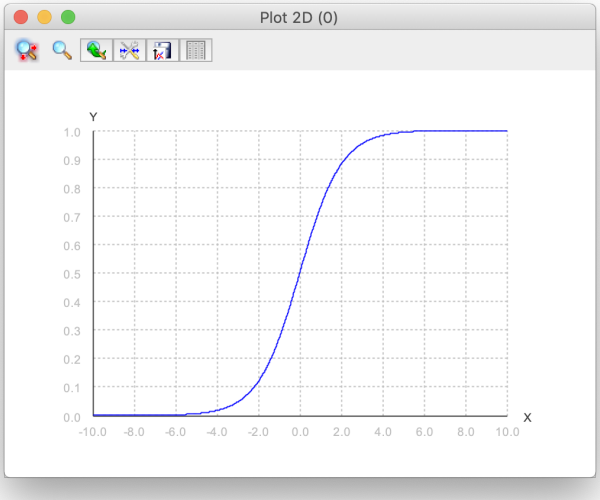

Abbildung 1: Ausgabe des Plot2D-Programms für die *Sigmoid*-Funktion.

5. **Grafische Darstellung von anderen Signalen**
Als nächstes wollen wir erreichen, dass unser Signalplotter nicht nur mathematische Funktionen grafisch darstellen kann, sondern auch andere Signale. Als Beispiel dafür soll in dieser Aufgabe eines der bekanntesten und wichtigsten Biosignale, das Elektrokardiogramm (EKG) (engl. *electrocardiogram, ECG*), visualisiert werden [1]. Ein Ausschnitt eines EKG-Signals finden Sie in der Datei `ecg.txt`.

[1] Das EKG wird über Elektroden auf der Haut aufgenommen und misst die elektrische Herzaktivität.

Erstellen Sie dafür die Methode `void plotEcg()`, in der Sie die folgenden Schritte implementieren:

a) *EKG-Signal einlesen*
Lesen Sie zunächst das in der Textdatei `ecg.txt` gespeicherte EKG-Signal ein. Dazu steht Ihnen die Methode `PlotHelper.readEcg(String filename)` zur Verfügung. Sie bekommt als Parameter den Namen der Datei übergeben, in dem das Signal gespeichert ist, liest den Dateiinhalt ein und gibt das Signal als `double[]` zurück. Speichern Sie den Rückgabewert der Methode in einer Variable `ecgSignal` geeigneten Typs.

b) *Abtastrate*
Das vorliegende, diskrete EKG-Signal wurde mit einer *Abtastrate* (engl. *sampling rate*) von 250 Hz abgetastet. Das bedeutet, dass ein EKG-Signal mit einer Dauer von 1 s beispielsweise eine Länge von 250 Datenpunkten hätte, ein 60 s langes EKG-Signal dementsprechend 15000 Datenpunkte lang wäre. Legen Sie – ähnlich wie die Konstanten `FIRST_LIMIT` bzw. `SECOND_LIMIT` – die Konstante `int SAMPLING_RATE` für die Abtastrate an und initialisieren Sie sie mit dem entsprechenden Wert.

c) *Stützstellen*
Um das EKG-Signal zu plotten, müssen als nächstes die entsprechenden Stützstellen erzeugt werden. Verwenden Sie dazu wieder die Methode `createSamplingPoints(double, double, int)`. Die beiden Parameter `firstLimit` und `secondLimit` sollen dabei so gewählt werden, dass die Stützstellen die korrekte Zeit in Sekunden widergeben. Bei einer Signallänge von 30s wäre `firstLimit` entsprechend `0`, `secondLimit` wäre `30`. Zudem müssen natürlich so viele Stützstellen erzeugt werden wie es EKG-Datenpunkte gibt. Speichern Sie die Stützstellen in der Variablen `ecgTime`.

d) *EKG visualisieren*
Um das EKG-Signal grafisch darzustellen, stellen wir Ihnen in der Klasse `PlotHelper` die Methode `PlotHelper.plotEcg(double[], double[])` zur Verfügung, die als Parameter die Stützstellen und die EKG-Datenpunkte als `double`-Arrays übergeben bekommt. Rufen Sie aus der `main`-Methode die Methode `plotEcg` auf. Wenn Sie jetzt das Programm ausführen, müssten Sie folgende Ausgabe sehen:

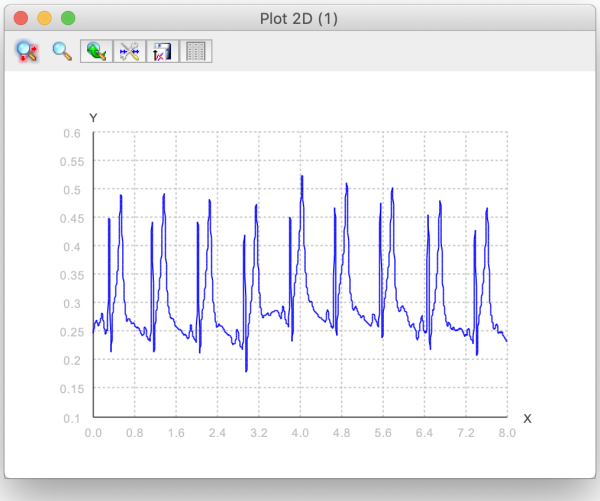

Abbildung 2: Ausgabe des Plot2D-Programms für das EKG-Signal.

e) *R-Zacken einlesen*
Der markanteste Punkt im EKG-Signal ist die sogenannte *R-Zacke* (engl. *R peak*). Da sie sehr markant (und dadurch leicht zu detektieren) ist, werden R-Zacken unter anderem dafür

verwendet, die Herzrate zu bestimmen, indem die Zeitdifferenzen zwischen zwei aufeinander folgende R-Zacken berechnet werden. Die Datei `rpeaks.txt` beinhaltet die R-Zacken, die zum EKG-Signal in der Datei `ecg.txt` gehören. Sie sind dabei als *Indizes* des zugehörigen `ecg`-Signals gespeichert. So bedeutet beispielsweise der Wert `205`, dass sich an Index `205` in den EKG-Daten eine R-Zacke befindet.

Um die Werte einzulesen, stellen wir Ihnen die Methode `PlotHelper.readPeaks(String filename)` zur Verfügung. Sie bekommt als Parameter den Namen der Datei übergeben, in dem die R-Zacken gespeichert ist, liest den Dateiinhalt ein und gibt ein `int[]` zurück. Speichern Sie den Rückgabewert der Methode in einer Variable `idxRPeaks` geeigneten Typs.

f) *R-Zacken visualisieren*
Nun sollen die R-Zacken im EKG-Signal wie in Abbildung 3 visualisiert werden. Dafür müssen Sie diejenigen Stützstellen und Datenpunkte im EKG-Signal bestimmen, die zu den jeweiligen R-Zacken gehören. Speichern Sie diese Punkte in den Arrays `rPeaks` (für die Datenpunkte) und `timeRPeaks` (für die Zeit-Stützstellen).

Visualisieren Sie das EKG-Signal zusammen mit den R-Zacken, indem Sie die von uns bereitgestellte Methode `PlotHelper.plotEcg(...)` verwenden, dieses Mal jedoch die überladene Methode mit vier Parametern anstatt der ursprünglich verwendeten mit zwei Parametern. Zusätzlich zu den Stützstellen und den EKG-Datenpunkten bekommt diese Methode noch die Stützstellen und Datenpunkte der R-Zacken übergeben. Wenn Sie jetzt das Programm ausführen, müssten Sie folgende Ausgabe sehen:

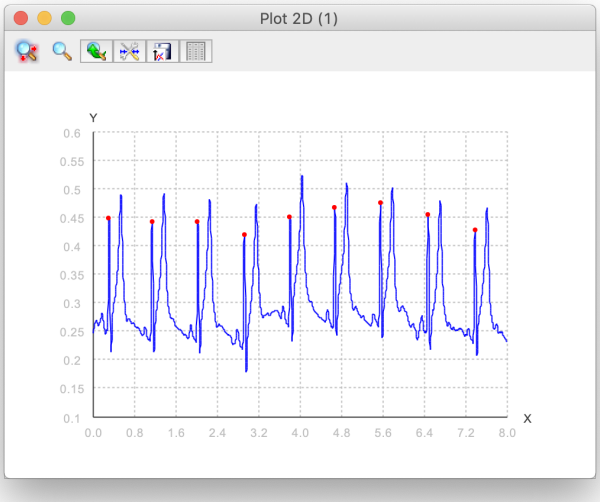

Abbildung 3: Ausgabe des Plot2D-Programms für das EKG-Signal inklusive R-Zacken.

g) *Herzrate bestimmen*
Wie bereits erwähnt lässt sich die Herzrate bestimmen, indem die Zeitdifferenz zwischen zwei R-Zacken bestimmt wird. Erstellen Sie eine Methode `void computeHeartRate(double[])`, die die Zeitpunkte (also die Stützstellen) der R-Zacken als Array übergeben bekommt, die Herzrate berechnet und die Werte anschließend auf `stdout` ausgibt.

Iterieren Sie mithilfe einer Schleife durch das Array und berechnen Sie in der Schleife die Differenz zwischen zwei jeweils benachbarten Einträgen. Diese Differenzen entsprechen der Dauer zwischen zwei Herzschlägen in *Sekunden*. Üblicherweise wird die Herzrate jedoch in *Schlägen pro Minute* (engl. *beats per minute, bpm*) angegeben. Rechnen Sie die Zeitdifferenzen in *bpm* um und geben Sie die Ergebnisse auf `stdout` in folgendem Format, auf zwei Nachkommastellen gerundet, aus:

```
Heart Rate:
72.02 bpm
67.87 bpm
...
```

Hinweis: Die folgende Formatierungsmethode der Klasse `String` rundet eine `double`-Zahl auf zwei Nachkommastellen und gibt das Ergebnis als `String` zurück:
`String.format("%.2f", ⟨heartRate⟩)`

Rufen Sie die Methode zum Berechnen der Herzrate in der Methode `plotEcg()`, bevor Sie das EKG-Signal plotten, auf.

6. Ihr fertiges Programm sollte nun zwei Fenster öffnen – eines mit dem Plot der *Sigmoid*-Funktion, eines mit dem EKG-Signal inklusive R-Zacken. Zudem sollten die Herzraten auf `stdout` ausgegeben werden.

7. Geben Sie die Datei `SignalPlotter.java` ab.

Aufgabe 3.3: Cäsar-Chiffre 13 Punkte

Strings, Arrays, Methoden

Das *Cäsar-Chiffre* ist ein symmetrisches Verschlüsselungsverfahren. Als eines der einfachsten (und auch unsichersten) Verfahren dient es heute hauptsächlich dazu, Grundprinzipien der Kryptologie anschaulich darzustellen. Bei der Verschlüsselung wird jeder Buchstabe des Klartexts auf einen Geheimtextbuchstaben abgebildet, indem sie um eine bestimmte Anzahl nach rechts oder links verschoben werden. Die Anzahl der verschobenen Zeichen bildet den *Schlüssel*, der für die gesamte Verschlüsselung unverändert bleibt.

Das Verfahren ist sehr unsicher, da man einen verschlüsselten Text mit sehr wenig Aufwand »knacken« kann. Dafür reicht ein ausreichend langer Mustertext der Zielsprache. Allein mit diesen zwei Informationen – dem zu entschlüsselnden Text sowie einem Mustertext der Zielsprache – kann der Schlüssel berechnet werden. Hierzu berechnet man ein *Histogramm*, also die Häufigkeitsverteilung der einzelnen Buchstaben. Indem man im Histogramm anschließend die jeweils am häufigsten vorkommenden Buchstaben bestimmt, lässt sich die Verschiebung ermitteln und der Geheimtext somit entschlüsseln.

1. **Projekt anlegen:**
   Legen Sie ein neues Java-Projekt `03-CaesarChiffre` mit der Klasse `CaesarChiffre` und einer `main`-Methode an.

2. **Mustertext:**
   Folgender Text soll als Mustertext verwendet werden:

   ```
   Werden zwei Glasstaebe mit einem Wolltuch gerieben, dann kann man feststellen,
   dass sich die beiden Staebe gegenseitig abstossen. Wird das gleiche Experiment
   mit zwei Kunststoffstaeben wiederholt, dann bleibt das Ergebnis gleich, auch
   diese beiden Staebe stossen sich gegenseitig ab. Im Gegensatz dazu ziehen sich
   ein Glas und ein Kunststoffstab gegenseitig an. Diese mit den Gesetzen der
   Mechanik nicht zu erklaerende Erscheinung fuehrt man auf Ladungen zurueck. Da
   sowohl Anziehung als auch Abstossung auftritt, muessen zwei verschiedene Arten
   von Ladungen existieren. Man unterscheidet daher positive und negative Ladungen.
   ```

   Sie müssen den Text nicht abtippen, sondern können ihn aus der Datei `language-pattern.txt` kopieren. Laden Sie dafür das Zusatzmaterial zum 3. Übungsblatt (`03-material.zip`) von StudOn herunter und entpacken Sie diese Datei. Legen Sie eine Konstante `GERMAN_LANGUAGE_PATTERN` vom Typ `String` an und weisen Sie ihr den Mustertext zu.

   Bonusfrage: Aus welchem Buch stammt dieser Text? Falls Sie die Antwort kennen, schreiben Sie sie als Kommentar in Ihren Code ☺.

3. **Geheimbotschaft:**
   Für die zu entschlüsselnde Botschaft legen wir eine weitere Klassenkonstante `ENCRYPTED_MESSAGE` an und weisen Ihr den folgenden Wert zu:

   ```
   xjmw lzy! iz mfxy ijs htij ljpsfhpy zsi inw xt wzmr zsi jmwj jw|twgjs. nhm rzxx
   rnw mnjw ojijx xjrjxyjw jnsjs sjzjs yjły fzxijspjs zsi qfslxfr |jwijs inj nijjs
   psfuu.
   ```

   Hinweis: Auch diesen Text finden Sie im Zusatzmaterial in der Datei `example-message.txt`.

4. **Index des Array-Maximums:**
   Deklarieren Sie die Methode `getIndexOfMaximumEntry`. Diese bekommt ein `int`-Array `values` übergeben und soll die *Position* des maximalen Wertes im Array finden und zurückgeben. Sie können davon ausgehen, dass nur korrekte Arrays übergeben werden. Laufen Sie mit Hilfe einer Schleife über das Array und speichern Sie die Position des Maximums in einer Variablen `maxIndex` geeigneten Typs. Geben Sie ihren Wert anschließend zurück.

   Hinweis: Testen Sie Ihre Methode ausgiebig! Legen Sie hierfür in der `main`-Methode verschiedene Arrays mit verschiedenen Werten an und übergeben Sie diese an die Methode `getIndexOfMaximumEntry`. Prüfen Sie, ob der berechnete Index korrekt ist. Vergessen Sie nicht, Ihre Tests im Anschluss wieder auszukommentieren.

5. **Bestimmung des häufigsten Buchstabens:**
   Nun soll der am häufigsten auftretende Buchstabe ermittelt werden.

   Lassen Sie den Text vorerst als `String` erhalten. Einige Methoden der Klasse `String` aus der Java-API dürften Ihnen im Verlauf der Aufgabe helfen. Lesen Sie sich die Dokumentation zur korrekten Verwendung durch!

   a) Legen Sie die Methode `getHistogram` an, die als Parameter einen Text als `String` bekommen und das Histogramm als `int[]` zurückgeben soll.

   b) Im Histogramm sollen die Häufigkeitsverteilung der Buchstaben festgehalten werden. Legen Sie dafür ein `int`-Array namens `histogram` an. Es soll genau so groß sein, um alle Charakter des *ASCII*-Zeichensatzes fassen zu können. Der *ASCII*-Wert eines Charakters soll seinem Index im Array entsprechen.

   c) Der Einfachheit halber unterscheiden wir nicht zwischen Klein- und Großbuchstaben. Wandeln Sie daher alle Buchstaben des Strings mittels einer geeigneten Methode aus der `String`-Klasse in Kleinbuchstaben um.

   d) Iterieren Sie nun in einer geeigneten Schleife über den Text und bestimmen Sie die relative Häufigkeit für jeden Charakter (⇝ `String.charAt(int)`), indem Sie bei jedem Auftreten eines Charakters seinen entsprechenden Array-Eintrag um 1 inkrementieren.

   e) Geben Sie `histogram` am Ende der Methode zurück.

   f) Legen Sie die Methode `getSignificantLetter` an, die als Parameter einen Text als `String` bekommen und den Buchstaben als `char` zurückgeben soll.

   g) Legen Sie eine Klassenkonstante `SEPARATOR` vom Typ `char` an und weisen Sie dieser ein Leerzeichen zu. Dieser Separator dient später dazu, die Information über die Wortgrenzen nicht zu verlieren. Da die Wortgrenzen keine Rolle bei der Suche nach dem häufigsten Zeichen spielen, sollen diese im Histogramm nicht mitgezählt werden.

   h) Rufen Sie die Methode `getHistogram` auf und speichern Sie das Ergebnis in einer Variable `histogram`.Finden Sie nun im Histogramm den Index des Eintrages mit dem höchsten Wert mittels `getIndexOfMaximumEntry` und speichern Sie ihn in einer Variable `significantLetter` des Typs `char`. Dieser Index entspricht, als Zeichen interpretiert, dem am häufigsten vorkommenden Buchstaben.

   i) Der Interesse halber wollen wir dem Benutzer ein Feedback darüber geben, wie oft der häufigste Buchstabe im Text vorgekommen ist. Legen Sie dafür die Variable `quantity` geeigneten Datentyps an, in der Sie eintragen, wie oft der Buchstabe `significantLetter` im Text vorgekommen ist (⇝ Histogramm). Legen Sie außerdem die Variable `int quota` an, in der Sie den *relativen Anteil* von `significantLetter` im Text in Prozent speichern.

j) Geben Sie folgende Meldung auf `stdout` aus:

```
Most significant letter: ⟨significantLetter⟩
Quantity: ⟨quantity⟩ times (⟨quota⟩ % of whole text).
```

k) Geben Sie zum Schluss `significantLetter` als Ergebnis der Methode `getSignificantLetter` zurück.

Hinweis: Auch diese Methode können Sie in der `main`-Methode testen. Legen Sie beliebige Strings an und rufen Sie Ihre Methode auf. Lassen Sie sich die Statistik und das Ergebnis der Methode ausgeben. Vergessen Sie nicht, Ihre Tests im Anschluss wieder auszukommentieren.

6. **Entschlüsselung der Botschaft:**
Nach der Bestimmung des häufigsten Buchstabens können wir nun versuchen, den Text zu entschlüsseln.

a) Legen Sie die Methode `getShift` an, die zwei Parameter vom Typ `String` übergeben bekommen soll: `encryptedText` für die zu entschlüsselnde Botschaft und `languagePattern` für den Mustertext. Die Methode soll die Verschiebung als `int` zurückgeben.

b) Legen Sie die beiden Variablen `sigOfChiffre` und `sigOfPattern` an, in denen Sie die Ergebnisse der Aufrufe von `getSignificantLetter` für den verschlüsselten Text und den Mustertext speichern.

c) Die beiden Buchstaben `sigOfPattern` und `sigOfChiffre` stellen die am häufigsten auftretenden Buchstaben im Mustertext und im verschlüsselten Text dar. Berechnen Sie die Differenz der beiden Variablen in einer neuen Variable `shift` geeigneten Typs.

d) Geben Sie dem Nutzer auf `stdout` eine Zwischenbilanz der Decodierung aus:

```
Most significant letter in the pattern text: ⟨sigOfPattern⟩
Most significant letter in the encrypted text: ⟨sigOfChiffre⟩
Resulting shift: ⟨shift⟩
```

e) Geben Sie anschließend ⟨*shift*⟩ zurück.

f) Legen Sie nun die Methode `decode` an, die zwei Parameter vom Typ `String` übergeben bekommen soll: `encryptedText` für die zu entschlüsselnde Botschaft und `languagePattern` für den Mustertext. Die Methode soll den entschlüsselten Text als `String` zurückgeben.

g) Rufen Sie nun die eben von Ihnen erstelle Methode `getShift` auf und speichern Sie das Ergebnis in der Variable `shift`. Sie benötigen diesen Wert, um den Text zu entschlüsseln.

h) Da die Verschlüsselung zeichenweise geschieht, soll auch die Entschlüsselung auf diese Weise erfolgen. Legen Sie dazu zuerst eine Variable `lettersEncryptedText` an und wandeln Sie `encryptedText` von einem `String` in ein `char[]` um (⇝ `toCharArray()`).

i) Gehen Sie in einer geeigneten Schleife über alle Einträge von `lettersEncryptedText` und verschieben Sie sie um `shift` Positionen. Die Richtung, in die Sie verschieben müssen, hängt davon ab, wie Sie die Differenz `shift` definiert haben. Vor dem »zurechtrücken« gilt es noch noch zu beachten, dass – wie anfangs erwähnt – Sonder- und Leerzeichen im Originaltext nicht verschoben wurden. Prüfen Sie daher für jedes Zeichen, ob es sich im Intervall $[a \pm \texttt{shift}, z \pm \texttt{shift}]$ befindet. Ob Sie $+$ oder $-$ verwenden müssen, hängt wieder von Ihrer Definition von `shift` ab, das werden Sie aber beim Testen herausbekommen! Verschieben Sie nur, wenn sich das aktuelle Zeichen **im** Intervall befindet!

j) Jetzt können Sie endlich den entschlüsselten Text zurückgeben, jedoch nicht als `char[]`, sondern als `String`. Legen Sie daher einen neuen String namens `decoded` an und finden Sie einen Weg, um ein `char`-Array in einen `String` zu konvertieren!

k) Geben Sie `decoded` zurück.

7. Rufen Sie die Methode `decode` aus der `main`-Methode heraus auf und speichern Sie das Ergebnis in der Variablen `decodedText`.

8. Geben Sie die verschlüsselte Botschaft sowie den entschlüsselten Text in der folgenden Form auf `stdout` aus:

```
Unreadable, encrypted input text:
⟨ENCRYPTED_MESSAGE⟩

Readable, decoded output text:
⟨decodedText⟩
```

9. Fügen Sie den entschlüsselten Text sowie den Schlüssel, mit dem er verschlüsselt wurde, in einen Kommentar ans Ende der `main`-Methode.

10. Geben Sie die Datei `CaesarChiffre.java` ab.

Testen Sie Ihr Programm ausgiebig!

Sollte Ihr Programm nicht übersetz- bzw. ausführbar sein, wird die Lösung mit 0 Punkten bewertet. Stellen Sie also sicher, dass IntelliJ IDEA keine Fehler in Ihrem Programm anzeigt, Ihr Programm übersetz- und ausführbar ist sowie die in der Aufgabenstellung vorgegebenen Namen und Schnittstellen *exakt* eingehalten werden. Geben Sie am Schluss die Dateien `SignalPlotter.java` und `CaesarChiffre.java` über die EST-Webseite ab. Wenn Sie die Aufgabe zusammen mit einem Übungspartner bearbeitet haben, geben Sie im EST unbedingt dessen Gruppenabgabe-Code an! Kontrollieren Sie, ob Ihre Namen am Anfang aller Dateien angegeben sind – schreiben Sie im Quellcode Ihre Angaben in einen Kommentar. Im EST-Abgabesystem können Sie modifizierte Dateien mehrfach abgeben. Nur die zuletzt hochgeladene Version wird bewertet.