

**Programmierung für Naturwissenschaften 1
Wintersemester 2020/2021
Übungen zur Vorlesung: Ausgabe am 27.01.2021**

Aufgabe 10.1 (2 Punkte) Aus Zeitgründen können wir in der Vorlesung das Thema „Anwendung der in Python verfügbaren Sortiermethoden“ (siehe `python-slides.pdf`, Abschnitt *Sorting using Python's build-in methods*) nicht behandeln. Dieser Abschnitt wird daher im Peer-Teaching Format behandelt. Dazu müssen sich einige Studierende vor der Übung anhand der Vorlesungsfolien auf dieses Thema vorbereiten und das erworbene Wissen in Kleingruppen in den ersten 20-30 Minuten am Anfang der Übung an die anderen Studierenden weitergeben.

Die Kleingruppen setzen sich wie folgt zusammen:

- | | |
|--|---|
| 1. Tchonet Toukam, Vehns, Nichvolodina | 9. Pohle, Rodrigues Nobre, Koerper |
| 2. Kourouklis, Schimansky, Hossfeld | 10. Riemann, Schulta, Maecking |
| 3. Nergiz, Stephan, Mehammed | 11. Naehring, Neuhaus, Nachtschatt |
| 4. Schindler, Unland, Wang, Koester | 12. Le, Trigkas Chatziandreou, Kemnitz |
| 5. Zozanyan, Beiersdorf, Jolic | 13. Sakhire, von der Osten-Sacken, Merl |
| 6. Reich, Wacke, Hiep | 14. ASchroeder, Krupp, Kroeger, Spurny |
| 7. Liebsch, Mrozek, Albetani | 15. Coin, Chow Castro, Fikani, Altmann |
| 8. Jakovljevic, Lim, Fuchs | 16. Duez, Koegel, Buchstab |

Die Namen der Studierenden, die sich auf das Thema vorbereiten müssen, sind jeweils am Beginn jeder Zeile aufgeführt. Falls jemand von diesen Studierenden nicht zur Übung erscheint, verteilen sich die übrigen Mitglieder der Kleingruppe auf die anderen Gruppen.

Nach der Übung dokumentiert jede Kleingruppe in einer E-mail an `pfn1@zbh.uni-hamburg.de` das Vorgehen bei der Erarbeitung des Themas und ggf. noch bestehende Verständnisfragen oder Hinweise zu Unklarheiten in den Folien. Willkommen sind natürlich auch Bemerkungen zur Lehrform selbst. Dabei soll nicht der Inhalt der Folien repliziert werden. Die E-mail soll die folgenden Eigenschaften haben:

- die Betreffzeile lautet `sortingPeer` in genau dieser Schreibweise,
- abgesendet bis zum Abgabetermin der entsprechenden Übung,
- maximal 15 Zeilen mit maximal 80 Zeichen pro Zeile,
- Angabe der Nachnamen aller Mitglieder der Kleingruppe, die teilgenommen haben (alle genannten Personen erhalten die zwei Punkte).

Die E-mail soll von einer/einem Studierenden erstellt werden, die/der sich nicht auf das Thema vorbereitet hat.

Aufgabe 10.2 (3 Punkte) Sie haben eine neue Messmethode entwickelt, die für ein chemisches, physikalisches oder biologisches System Koordinaten im zweidimensionalen Raum liefert. Diese

Messmethode kann z.B. durch die Wahl verschiedener Einstellungen variiert werden. Sie haben Ihre Methode mit verschiedenen Einstellungen ausprobiert und entsprechende Koordinaten ermittelt. Diese bilden damit Vorhersagen der Realität. Sie sollen nun die Qualität der einzelnen Messungen jeweils durch einen Vergleich Ihrer vorhergesagten Werte mit einem Goldstandard ermitteln, der durch eine sehr aufwändige, aber anerkannte Messmethode ermittelt wurde.

Die Qualität Ihrer Methode soll durch die Bestimmung der Sensitivität und der Spezifität relativ zum Goldstandard bestimmt werden. Die Sensitivität macht eine Aussage über die Fähigkeit der Methode, Koordinaten entsprechend des Goldstandards richtig vorherzusagen. Die Spezifität macht eine Aussage über die Fähigkeit der Methode, keine bzgl. des Goldstandards falschen Werte vorherzusagen. Die formale Definition dieser Begriffe basiert auf drei Mengen, P , G und TP . Dabei ist P die Menge der Messwerte Ihrer Messmethode, G die Menge der Werte des Goldstandards und $TP = P \cap G$, also die Schnittmenge von P und G . TP heißt auch die Menge der *True Positives*, also der korrekten Vorhersagen. Die Sensitivität $se(P, G)$ und die Spezifität $sp(P, G)$ sind definiert durch

$$se(P, G) = 100 \cdot \frac{|TP|}{|G|} \qquad sp(P, G) = 100 \cdot \frac{|TP|}{|P|}$$

Dabei bezeichnet $|S|$ die Größe einer Menge S . Da $TP \subseteq G$ und $TP \subseteq P$, liegen beide Werte zwischen 0 und 100. Eine ideale Methode erreicht jeweils Werte von 100%, d.h. sie liefert die gleichen Messwerte wie der Goldstandard. In vielen Anwendungen erreicht man optimale Sensitivität nur auf Kosten geringer Spezifität und umgekehrt. Daher kombiniert man beide Werte, indem man den harmonischen Durchschnitt der Sensitivität und Spezifität berechnet. Für $0 \leq a, b \leq 100$ ist $\frac{2}{\frac{1}{a} + \frac{1}{b}}$ der harmonische Durchschnitt von a und b .

Implementieren Sie ein Programm `predictionqual.py`, das die Qualität der vorhergesagten Messdaten berechnet und formatiert ausgibt. Das Programm soll eine Option `-g/--gold.standard` mit genau einem String-Argument haben. Dieses String-Argument ist der Name der Datei mit dem Goldstandard (z.B. `goldstandard.tsv` im Material). Alle weiteren Argumente sind die Namen der Dateien mit den Koordinaten der Messungen (`prediction*.tsv` im Material).

Alle genannten Dateien enthalten jeweils ein Paar von x,y-Koordinaten pro Zeile, separiert durch das Zeichen `\t`. Zur Vereinfachung sind die Koordinaten durch ganze Zahlen repräsentiert. Ein Koordinatenpaar (a, b) ist identisch mit einem Koordinatenpaar (a', b') , wenn $a = a'$ und $b = b'$ ist. True Positives sind die identischen Koordinatenpaare. Die Ausgabe soll zeilenweise für jede Datei die Qualitätswerte der Messungen ausgeben, und zwar durch das Zeichen `\t` separiert und in folgendem Format:

- Spalte 1: Name der Datei (`filename`)
- Spalte 2: Anzahl der True Positives (`tp`)
- Spalte 3: Sensitivität (`sens`)
- Spalte 4: Spezifität (`spec`)
- Spalte 5: Harmonischer Durchschnitt der Sensitivität und Spezifität (`hmean`)

Die numerischen Werte sollen rechtsbündig in einem Block der Breite 6 ausgegeben werden. Die Fließkommawerte sollen mit zwei Nachkommastellen ausgegeben werden. In der Datei `quality.out.tsv` finden Sie das erwartete Ergebnis.

Hinweise:

- Die Dateien enthalten jedes Koordinatenpaar jeweils genau einmal.
- Verwenden Sie die Klasse `set` zur Speicherung der Koordinatenpaare. Diese Klasse bietet u.a. die folgenden Operationen:
 - Eine leere Menge `s` wird durch `s = set()` erzeugt.
 - Durch `s.add(x)` fügen Sie ein neues Element `x` zur Menge `s` hinzu.
 - Für Mengen `s` und `t` liefert der Ausdruck `s & t` die Schnittmenge $s \cap t$ von `s` und `t`.
 - Für eine Menge `s` liefert `len(s)` die Größe dieser Menge.
- Gliedern Sie Ihr Programm durch die Implementierung von vier Funktionen:
 - eine Funktion `parse_command_line()`, die unter Nutzung des Moduls `argparse` einen Argumentparser `p` spezifiziert und `p.parse_args()` zurückliefert. Siehe auch `python-slides.pdf`, Abschnitt *Functions*, frame 58-64.
 - eine Funktion `inputfile2set(inputfile)` erhält genau eine Eingabedatei im beschriebenen Format und liefert mit einer `return`-Anweisung die Menge der Koordinatenpaare.
 - eine Funktion `evaluate(gs, prediction)` erhält zwei Mengen `gs` und `prediction` mit den Koordinaten eines Goldstandards und einer Vorhersage und liefert mit einer `return`-Anweisung die Anzahl der True Positives, die Sensitivität und die Spezifität bzgl. dieser beiden Mengen.
 - eine Funktion `harmonic_mean(a, b)` liefert den harmonischen Durchschnitt von `a` und `b`.

In den Materialien finden Sie neben den erwähnten Dateien mit dem Goldstandard und den Messwerten ein Makefile. Durch `make test` verifizieren Sie, dass Ihr Programm für diese Testdaten korrekt funktioniert.

Punkteverteilung:

- Implementierung der vier oben genannten Funktionen und des Hauptprogramms: je 0.5 Punkte
- bestandene Tests: 0.5 Punkte

Aufgabe 10.3 (5 Punkte) Eine Permutation einer Menge S ist eine Liste der Elemente aus S , in der jedes Element genau einmal vorkommt. Jede Permutation ist also eine Liste der Länge n , wenn n die Anzahl der Elemente in S ist. Verschiedene Permutationen von S unterscheiden sich durch die Reihenfolge der Elemente aus S . Die Menge der Permutationen von S wird durch $\text{Perms}(S)$ bezeichnet.

Beispiel: Für $S = \{0, 1, 2\}$ ist $\text{Perms}(S) = \{[2, 1, 0], [1, 2, 0], [2, 0, 1], [0, 2, 1], [1, 0, 2], [0, 1, 2]\}$.

Für Permutationen gibt es vielfältige Anwendungen in der Mathematik und Informatik.

Ein einfacher rekursiver Algorithmus zur Berechnung von $\text{Perms}(S)$ funktioniert nach den folgenden Regeln:

- Falls $S = \emptyset$, dann ist $\text{Perms}(S) = \{[]\}$, d.h. die leere Liste ist die einzige Permutation der leeren Menge S .
- Falls $S = \{a\}$ (d.h. S besteht aus genau einem Element a), dann ist $\text{Perms}(S) = \{[a]\}$.

- Falls S mindestens zwei Elemente enthält, dann berechnet man für alle $a \in S$ die Menge $R(S, a) = \text{Perms}(S \setminus \{a\})$.¹ $\text{Perms}(S)$ besteht in diesem Fall aus den Listen $p.append(a)$ für alle $a \in S$ und $p \in R(S, a)$.

Beispiel: Set $S = \{0, 1, 2\}$. Nach dem obigen Verfahren berechnet man zunächst die Mengen $R(S, 0)$, $R(S, 1)$ und $R(S, 2)$. Es ist $R(S, 0) = \text{Perms}(S \setminus \{0\}) = \text{Perms}(\{1, 2\})$. Daher berechnet man zunächst die Mengen $R(\{1, 2\}, 1)$ und $R(\{1, 2\}, 2)$. Es gilt:

$$\begin{aligned} R(\{1, 2\}, 1) &= \text{Perms}(\{2\}) = \{[2]\} \\ R(\{1, 2\}, 2) &= \text{Perms}(\{1\}) = \{[1]\} \end{aligned}$$

Damit ergibt sich $R(S, 0) = \{[2, 1], [1, 2]\}$. Analog erhält man $R(S, 1) = \{[2, 0], [0, 2]\}$ und $R(S, 2) = \{[1, 0], [0, 1]\}$. Aus $R(S, 0)$ berechnet man durch Anhängen von 0 die beiden Permutationen $[2, 1, 0]$, $[1, 2, 0]$. Aus $R(S, 1)$ berechnet man durch Anhängen von 1 die beiden Permutationen $[2, 0, 1]$, $[0, 2, 1]$. Aus $R(S, 2)$ berechnet man durch Anhängen von 2 die zwei Permutationen $[1, 0, 2]$, $[0, 1, 2]$. Damit wurden alle $3! = 6$ Permutationen von S berechnet.

Benennen Sie die Datei `allperms_template.py` in `allperms.py` um. Implementieren Sie darin eine Python-Funktion `all_permutations(elms)`, die die Liste aller Permutationen der Elemente aus `elms` nach dem obigen Algorithmus berechnet und als `return`-Wert zurückliefert. Die Menge der Elemente aus S bzw. ihre Teilmengen können Sie jeweils als Liste darstellen, d.h. `elms` ist eine Liste. In der Implementierung müssen Sie Elemente aus Listen löschen. Durch `l.pop(idx)` können Sie das Element an Index `idx` in einer Liste `l` löschen.

Die obige Beschreibung legt eine rekursive Berechnung nah. Daher implementieren Sie die obige Funktion auf der Basis einer rekursiven Funktion

`all_perms_rec(all_perms, elms)`, die in der Liste `all_perms` die Permutationen der Liste `elms` berechnet. `all_perms_rec` hat keinen Rückgabewert.

Beachten Sie, dass eine Wertzuweisung `p = q` für zwei Listen `p` und `q` eine Referenz `p` auf die Liste `q` erzeugt. Da `all_perms_rec` mit verschiedenen Listen (als zweitem Argument) aufgerufen wird, müssen Listen kopiert werden. Die Funktion `q.copy()` liefert eine Kopie der Liste `q`.

Implementieren Sie außerdem eine Funktion `all_permutations_verify(all_perms, elms)`, die verifiziert, dass die Liste `all_perms` die Liste aller Permutationen von `elms` ist. Dabei sind die Elemente in `elms` aufsteigend sortiert. Sei n die Anzahl der Elemente in `elms`. In der Funktion `all_permutations_verify` müssen Sie mit Hilfe von `assert` die folgenden Bedingungen verifizieren:

1. Die Länge von `all_perms` ist $n!$.
2. Es gibt keine Permutation in `all_perms`, die mehr als einmal vorkommt.
3. Wenn man die einzelnen Elemente aus `all_perms` (also die Listen) jeweils sortiert, ergibt sich immer die Liste `elms`.

Durch die Verwendung von `assert` bricht das Programm ab, wenn eine der genannten Bedingungen nicht zutrifft.

Die Verifikation von Duplikaten (siehe 2) darf nicht dadurch erfolgen, dass man jedes Element aus `all_perms` vergleicht. Nutzen Sie stattdessen Techniken, die Sie in Aufgabe 10.1 kennengelernt haben.

¹Der Operator \setminus steht für die Mengendifferenz, d.h. für zwei Mengen A und B ist $A \setminus B = \{a \mid a \in A, a \notin B\}$.

Im Hauptprogramm werden die beiden genannten Funktionen für Listen der Länge n aufgerufen, wobei n das einzige Argument des Hauptprogramms ist. Durch `make test` verifizieren Sie die Korrektheit Ihrer Implementierung.

Punkteverteilung:

- Implementierung von `all_permutations`: 3 Punkte
- Implementierung von `all_permutations_verify`: 2 Punkte

Bitte die Lösungen zu diesen Aufgaben bis zum 01.02.2021 um 18:00 Uhr an `pfn1@zbh.uni-hamburg.de` schicken.