

DAP2 Praktikum – Blatt 3

Abgabe: ab 25. April

Studienleistung

- Zum Bestehen des Praktikums muss jeder Teilnehmer die folgenden Leistungen erbringen:
 - Es müssen mindestens 50 Prozent der Punkte in den Kurzaufgaben erreicht werden.
 - Es müssen mindestens 50 Prozent der Punkte in den Langaufgaben erreicht werden.
- Im Krankheitsfall kann ein Testat bei Vorlage eines Attests in der folgenden Woche nachgeholt werden.
- Wenn ein Praktikumstermin auf einen Feiertag fällt, müssen Sie sich an einem beliebigen anderen Praktikumstermin in der gleichen Woche testieren lassen.
- **Hinweis:** Notieren Sie sich Ihre Punkte nach jedem Testat! Dies dient der eigenen Kontrolle. (Ihr Punktestand kann Ihnen während des Semesters nicht genannt werden.)

Wichtige Information (im Moodle verfügbar)

- Beachten Sie die Erklärung des **Ablaufs (Blatt A)**.
- Beachten Sie die **Regeln und Hinweise (Blatt R)** in der aktuellsten Version!
- Beachten Sie die **Hilfestellungen (Blatt H)** in der aktuellsten Version!

Kurzaufgabe 3.1: Enumerierung von Teilmengen

(5 Punkte)

Schreiben Sie ein Programm, das eine Liste von Ganzzahlen a_0, a_1, \dots, a_{n-1} via Standard-In, sowie eine Ganzzahl k als Argument erhält, und alle k -Elementigen Teilmengen ausgibt. Dies soll in folgenden Schritten umgesetzt werden:

- Zuerst werden die Zahlen a_0, a_1, \dots, a_{n-1} in ein Array `data` eingelesen.
- Anschließend werden Duplikate (also doppelte Einträge) aus dem Array entfernt. Implementieren sie eine Methode

```
public static int removeDuplicates(int [] data),
```

welche die Anzahl $n' \leq n$ der unterschiedlichen Werte in `data` zurückgibt. Zusätzlich soll die Methode die Werte im Array umordnen, sodass die n' unterschiedlichen Werte genau im Prefix `data[0, n' - 1]` des Arrays stehen.

Hinweis: Sie dürfen für diesen Schritt die Methode `Arrays.sort(...)` verwenden. Für die volle Punktzahl dürfen Sie keine Hilfsarrays verwenden, und abgesehen von dem Aufruf `Arrays.sort(...)` nur $\mathcal{O}(n)$ Zeit benötigen!

- Wenn $n' < k$, dann wird eine Fehlermeldung ausgegeben. Ansonsten geben Sie alle k -elementigen Teilmengen der n' unterschiedlichen Elemente aus. Abschließend geben Sie außerdem an, wie viele Teilmengen dies sind.

Für die volle Punktzahl sollten Sie dafür (abgesehen von der Ausgabe) nicht mehr als $\mathcal{O}\left(\binom{n'}{k}\right)$ Rechenschritte benötigen (dabei ist $\binom{n'}{k}$ der Binomialkoeffizient). Das gelingt am leichtesten mit einer rekursiven Implementierung. Für die Struktur der Rekursion können Sie sich von Pascal's Dreieck inspirieren lassen.

Wie immer sollten Sie ungültige Eingaben mit entsprechenden Fehlermeldungen abfangen, und ihren Code gründlich kommentieren. Beispielausgaben des Programms:

```
seq 4 > seq4.txt
seq 7 > seq7.txt
cat seq4.txt seq7.txt | java EnumSubset 8
Before removing duplicates: [1, 2, 3, 4, 1, 2, 3, 4, 5, 6, 7]
After removing duplicates:  [1, 2, 3, 4, 5, 6, 7]
ERROR: k is too large.

seq 2 2 8 | java EnumSubset 2
Before removing duplicates: [2, 4, 6, 8]
After removing duplicates:  [2, 4, 6, 8]
[2, 4]
[2, 6]
[2, 8]
[4, 6]
[4, 8]
[6, 8]
There are 6 subsets.

seq 12 | java EnumSubset 7
Before removing duplicates: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
After removing duplicates:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
...
There are 792 subsets.
```

Kurzaufgabe 3.2: Permutationen & Lex. Ordnung (3 Punkte)

Für dieses Problem erhalten sie eine Liste von Ganzzahlen a_0, a_1, \dots, a_{n-1} via Standard-In, sowie eine positive Ganzzahl k als Argument. Dieses mal dürfen Sie einfach davon ausgehen, dass die Liste der Ganzzahlen keine Duplikate enthält.

Seien p_0, p_1, \dots, p_{n-1} und q_0, q_1, \dots, q_{n-1} unterschiedliche Permutationen von a_0, a_1, \dots, a_{n-1} . Wir sagen, dass p_0, p_1, \dots, p_{n-1} lexikographisch kleiner als q_0, q_1, \dots, q_{n-1} ist, genau dann wenn es einen index $i \in \{0, \dots, n-1\}$ gibt, sodass $\forall j \in \{0, \dots, i-1\} : p_j = q_j$ und $p_i < q_i$. Beispiele:

- 1, 4, 3, 7, 5 ist lex. kleiner als 4, 1, 3, 7, 5
- 3, 4, 1, 5, 7 ist lex. kleiner als 3, 4, 1, 7, 5

Schreiben Sie ein Programm, das die lexikographisch k -kleinste Permutation von a_0, a_1, \dots, a_{n-1} ausgibt. Sie dürfen die Methode `Arrays.sort(...)` verwenden, um die Liste der Zahlen einmalig zu sortieren. Ihr Programm sollte nur $\mathcal{O}(n^2)$ Rechenschritte benötigen. Beispiele:

```
seq 4 | java LexPermute 17
Sorted input:
[1, 2, 3, 4]
The 17-smallest permutation is:
[3, 4, 1, 2]

seq 12 | shuf | java LexPermute 432101234
Sorted input:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
The 432101234-smallest permutation is:
[11, 10, 1, 8, 9, 4, 6, 7, 2, 3, 12, 5]
```

Tipp: Für jedes $i \in \{0, n-1\}$ gibt es $(n-1)!$ Permutationen von a_0, a_1, \dots, a_{n-1} , die mit a_i anfangen. Wenn die Elemente aufsteigend sortiert sind, also $a_0 < a_1 < \dots < a_{n-1}$, dann gilt:

- Die $(1 + 0 \cdot (n-1)!)$ -kleinste Permutation (also die kleinste Permutation) beginnt mit a_0 .
- Die $(1 + 1 \cdot (n-1)!)$ -kleinste Permutation beginnt mit a_1 .
- Die $(1 + 2 \cdot (n-1)!)$ -kleinste Permutation beginnt mit a_2 .
- usw...

Allgemein ausgedrückt ist die $(1 + i \cdot (n-1)!)$ kleinste Permutation genau die kleinste Permutation die mit a_i anfängt. Um die k -kleinste Permutation von a_0, a_1, \dots, a_{n-1} zu finden, können sie eine einfache rekursive Regel anwenden:

- $find(a_0, 1) = a_0$, (Basisfall für eine 1-elementige Liste) und rekursiv
- $find(a_0, a_1, \dots, a_{n-1}, k) = a_i, find(a_0, \dots, a_{i-1}, a_{i+1}, \dots, a_{n-1}, k')$,
wobei $i = \left\lfloor \frac{k-1}{(n-1)!} \right\rfloor$ und $k' = k - i \cdot ((n-1)!)$.