| | Patternsuche | | AnPr |
|---|---|---|---|
| | Name | Klasse | Datum |

# 1 Situation

Manchmal ist es notwendig, gewisse Zahlen- oder auch Zeichenfolgen zu suchen, welche in zwei verschiedenen Datenstrukturen – bspw. Arrays oder Strings – vorhanden sind. Wenn wir uns bspw. Folgende zwei Strings ansehen:

```
Hallo Welt! Damit fängt fast jeder Kurs an.
Dieser Kurs ist der beste Kurs der Welt.
```

gibt es Substrings, welche sowohl in dem einen, als auch in dem anderen String vorhanden sind. Auf den ersten Blick sollten uns die folgenden Strings auffallen:

```
Welt
Kurs
```

Auf den zweiten Blick gibt es noch folgende Parallelen:

```
der
er Kurs
```

Wenn man es aber ganz genau nimmt, müssten noch alle einzelnen Zeichen, welche in beiden Arrays vorhanden sind, als Parallele erkannt werden. Also `l`, `W`, `e`, `l`, `D`, `i`, `s`, `r`, `K`, `u`, das Leerzeichen und der Punkt.

# 2 Vorübung

Beginnen wir zuerst mit einem vereinfachten Problem. Wir haben wieder zwei Strings – allerdings suchen wir den kompletten 1. String im 2. String:

```
der
Dieser Kurs ist der beste Kurs der Welt.
```

Natürlich könnten wir dies jetzt sehr einfach mit der Java-Methode `indexOf()` erledigen – es geht uns in dieser Übung jedoch um Algorithmik! Also setzen wir die Funktionalität selbst um. Damit die Umsetzung möglichst flexibel einzusetzen ist, wandeln wir die Strings in Character Arrays um und verarbeiten Sie dann mit folgender Methode:

```
public static void searchPattern1(char[] pattern, char[] ar)
```

Die Methode soll nun für die oben angegebenen Strings folgende Ausgaben liefern:

```
Pattern found at position 16
Pattern found at position 31
```

# 3 Suchen von beliebigen Pattern

Im Regelfall wird der Algorithmus für die Vorübung zwei ineinander verschachtelte Schleifen beinhalten. Die erste iteriert durch das Array `ar` und die zweite geht von dort aus die Anzahl der `pattern`-Zeichen (im Fall vom Suchpattern „der“ 3 Schritte) nach rechts.

Nun passen wir den Algorithmus so an, dass wir den zu suchenden String dynamisch auswählen. Es kann also überall im Parameterarray sein. Wir ändern also die Benamung der Parameter zu:

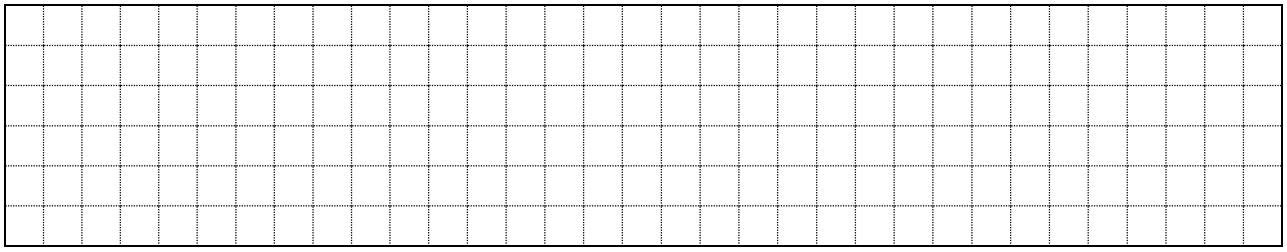
```
public static void searchPattern2(char[] ar1, char[] ar2)
```

Wir müssen also jede Zeichenfolge (mit jeder möglichen Länge) aus dem ar1 Array herausgreifen und im ar2 suchen. Bei jeder Fundstelle wird dann angegeben, wo im ersten Array und wo im zweiten Array eine Übereinstimmung existiert. Zusätzlich wird noch das in beiden Arrays vorhandene Pattern ausgegeben.

Hier ein Beispiel für eine Fundstelle:

…
```
Pattern found. Ar1 pos 5 Ar2 pos 34:  Welt
```
…

Setzen Sie den Algorithmus um und prüfen Sie ihn mit den angegebenen Strings. Was fällt Ihnen bei der Ausgabe auf:

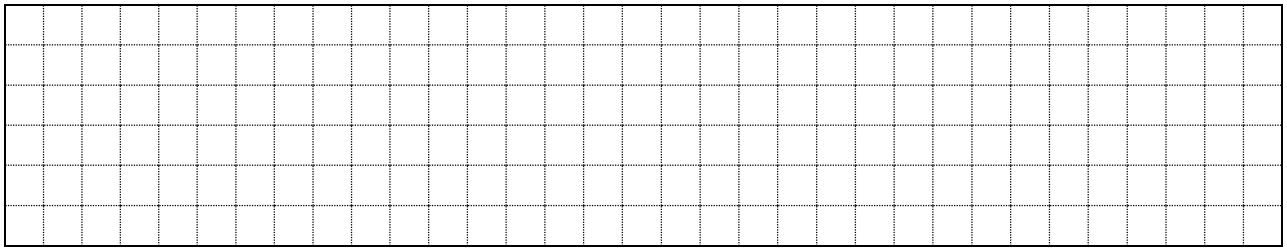


# 4 Angabe von minimaler Länge

Eine Lösung für dieses Problem wäre, eine Mindestlänge für ein mögliches Pattern anzugeben:

```
public static void searchPattern3(char[] ar1, char[] ar2, int minSize)
```

Wenn Sie also alle Zeichen suchen (also auch `l`, `W`, `e`, `l`, `D`, `i`, `s`, `r`, `K`, `u`, das Leerzeichen und der Punkt), gibt man für den Parameter `minSize` den Wert 1 ein. Bei Angabe von 5 wird bspw. der String `"der "` nicht gesucht. Setzen Sie den Algorithmus um und prüfen Sie ihn mit den angegebenen Strings und einer minimalen Länge von 5. Was fällt Ihnen nun bei der Ausgabe auf:



# 5 Erweiterung des Algorithmus

Wir ändern den Algorithmus nun so ab, dass jedes Pattern in maximaler Länge ausgegeben wird – also keine Dopplungen mehr auftreten. Hierzu schaffen wir uns eine Hilfsklasse `PatternHit`, in der die Zwischenergebnisse aus der Suche in `searchPattern3` abgelegt werden. Diese soll die notwendigen Trefferinformationen eines Pattern als einzelne Werte speichern – also die gleichen Informationen wie bei der Ausgabe:

- Position im Array 1
- Position im Array 2
- Wert des Patterns

Alle Bereiche mit Überschneidungen müssen nun identifiziert werden. Der Treffer mit der maximalen Größe ist dann auszugeben. Die folgenden Fundstellen:

```
"er Ku", "er Kur", "er Kurs", "er Kurs ", "r Kur", "r Kurs", "r Kurs ", " Kurs",
" Kurs ", "Kurs "
```

müssen also auf die Fundstelle `"er Kurs "` reduziert werden.

*Hinweis: Für die Fundstellen sind jeweils der Start- und Endpunkt wichtig. Sie müssen also prüfen, ob es eine Überlappung irgendwelcher Fundstellen existiert. Hierzu müssen Sie die Struktur, in der die `PatternHit` Objekte liegen, nach der Position im Array 2 sortieren.*
Die Ausgabe soll nun wie folgt aussehen:

```
Pattern found. Ar1 pos 32 Ar2 pos 4: er Kurs
Pattern found. Ar1 pos 31 Ar2 pos 16: der
Pattern found. Ar1 pos 34 Ar2 pos 25:  Kurs
Pattern found. Ar1 pos 31 Ar2 pos 31: der
Pattern found. Ar1 pos 5 Ar2 pos 34:  Welt
```