

252-0027

Einführung in die Programmierung

Übungen

Arrays & Rekursion

Henrik Pätzold

Departement Informatik

ETH Zürich


Heutiger Plan

- Theorie
 - Arrays
 - Rekursion
- Praxis
 - Coding mit Arrays (Matrix)
 - Coding mit Rekursion

Arrays

Arrays (Eindimensional)

- Arrays belegen eine feste Grösse **n** im Speicher, haben daher auch eine feste Länge
- für jeden Eintrag kann für den deklarierten Datentyp ein Wert gespeichert werden
- auf ein spezifisches Element **i** zugreifen können wir mit **arr[i]**
- Indizierung startet bei **0**, geht also bis **n-1**



```
1 public class Main {  
2     public static void main(String[] args){  
3         int[] arr = {3,5,6,1,2,8};  
4         for(int i = 0; i < arr.length; i++){  
5             arr[i] *= 2;  
6         }  
7     }  
8 }
```

2D Arrays

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

[[1,2,3],[4,5,6],[7,8,9]]

Zweidimensionale Arrays

- eine Matrix könnte als Zweidimensionales Array dargestellt werden
- `int[][] arr = new int[n][n];`
definiert ein Array, welches **n** integer-Arrays der Länge **n** speichert (**nxn-Matrix**)
- eine gesamte Zeile erhalten wir also mit `arr[i]` für $0 \leq i < n$
- einen Eintrag erhalten wir mit `arr[i][j]` für $0 \leq i, j < n$
- **Vorsicht:** Die Längen der Arrays könnten unterschiedlich sein!

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int[][] arr = {{1, 2, 3},{4, 5, 6},{7, 8, 9}};  
4         for (int i = 0; i < arr.length; i++) {  
5             for (int j = 0; j < arr[i].length; j++) {  
6                 arr[i][j] *= 2;  
7             }  
8         }  
9     }  
10 }
```


Programmieraufgabe

Die perfekte Matrix (2020 W5)

Sei M eine $n \times n$ Matrix deren Elemente positive ganze Zahlen sind und für die gilt $0 < m_{i,j} \leq n^2$ und $m_{x,y} = m_{p,q} \Rightarrow (x = p) \wedge (y = q)$. Wir sagen, dass die Matrix M *perfekt* ist, wenn zusätzlich alle Zeilensummen und Spaltensummen gleich sind (also $\sum_{k=0}^{k=n-1} m_{i,k} = \sum_{k=0}^{k=n-1} m_{j,k}$ für alle i, j und $\sum_{k=0}^{k=n-1} m_{k,i} = \sum_{k=0}^{k=n-1} m_{k,j}$ für alle i, j mit $0 \leq i, j < n$).

Vervollständigen Sie die Methode `boolean checkMatrix(int[] [] m)` von der Klasse `Matrix`, so dass diese Methode `true` zurückgibt wenn die Input Matrix *perfekt* ist, und `false` sonst. Sie können davon ausgehen, dass der Parameter `m` nicht `null` ist. Alle anderen Eigenschaften, müssen Sie selber testen. Eine Matrix ist nur perfekt, wenn alle genannten Eigenschaften gelten.

Vorbesprechung – Übung 4

Rekursion

Rekursion (Anfang)

- Eine Funktion ruft sich selbst auf, um ein Problem schrittweise zu lösen
 - Problem → in Teilprobleme zerlegen
 - Sobald die **(Teil-)Probleme** gelöst sind → **Gesamtlösung** aufbauen
- Intuitiv, aber in der Praxis tricky → **Übung lohnt sich sehr!**

How to Rekursion

- Aufgabe verstehen
 - Was ist das Ziel? → Was will ich berechnen oder finden?
- Muster suchen
 - Wie taucht dasselbe Problem in kleinerer Form wieder auf?
- Basisfall/Basisfälle festlegen
 - Wie taucht dasselbe Problem in kleinerer Form wieder auf?
- Rekursion aufschreiben
 - Rufe die Funktion mit kleineren Eingaben auf, um Teillösungen zu bekommen.
- Kombinieren
 - Baue aus den Teillösungen die Gesamtlösung.

racecar

racecar

Aufgerufen
Nicht überprüft
Valide

racecar

Aufgerufen
Nicht überprüft
Valide

racecar

Aufgerufen
Nicht überprüft
Valide

racecar

Aufgerufen
Nicht überprüft
Valide

racecar

Aufgerufen
Nicht überprüft
Valide

racecar

Aufgerufen
Nicht überprüft
Valide

racecar

Aufgerufen
Nicht überprüft
Valide

racecar

Aufgerufen
Nicht überprüft
Valide

Pseudocode - Palindrom

- **Subproblem:** das innere Wort – **racecar** -> **aceca**
- **Basisfälle:** Länge < 2: **true**; Länge == 2: **true**, wenn beide Buchstaben gleich
- **Rekursion:** äußersten Buchstaben gleich (in-place) & inneres Wort ein Palindrom? (rekursiv)

Programmieraufgaben

Rekursive Pseudocode –arraySum

- Subproblem:
- Basisfälle:
- Rekursion:

Rekursive Pseudocode –arraySum

- **Subproblem:**

- Summe des Arrays ohne das erste Element

- **Basisfälle:**

- $[] \rightarrow 0$
 - $[x] \rightarrow x$

- **Rekursion:**

- $sum([x,y,...,z]) \rightarrow x + sum([y,...,z])$

Pseudocode – countString

- Subproblem:
- Basisfälle:
- Rekursion:

Pseudocode – countString

- **Subproblem:**

- Anzahl vorkommender Buchstaben im Substring ab Position 1

- **Basisfälle:**

- "" -> 0

- **Rekursion:**

- `count(o, "xyz"); 1 + countString(o, restlicherString);` // wenn `o == 'x'`
 - `count(o, "xyz"); 0 + countString(o, restlicherString);` // wenn `o != 'x'`

Pseudocode – Duplikate entfernen

- Subproblem:
- Basisfälle:
- Rekursion:

Pseudocode – Duplikate entfernen

- **Subproblem:**

- Wiederholungen im Reststring entfernen

- **Basisfälle:**

- "" -> ""

- **Rekursion:**

- "xyz" && letzter Buchstabe war 'x' -> *dedupFrom("yz")*
 - "xyz" && letzter Buchstabe war nicht 'x' -> "x" + *dedupFrom("yz")*

Pseudocode – isBalanced

- Subproblem:
- Basisfälle:
- Rekursion:

Pseudocode – isBalanced

- **Subproblem:**

- Werden alle aktuell offenen Klammern im Reststring geschlossen?

- **Basisfälle:**

- `""` && offene Klammern == 0 -> true
 - offene Klammern < 0 || illegale Klammern -> false

- **Rekursion:**

- `"{}{}"` -> *isBalanced*("{}{}") && offen + 1 merken
 - `"{}{}"` -> *isBalanced*("{}") && offen – 1 merken