Programmieren 1

Dr. Arne Schmidt

Wintersemester 2023/2024
Abgabe: 08.12.2023
Rückgabe: ab 11.12.2023

Übungsblatt 2

Abgabe der Lösungen bis zum 08.12.2023 um 18:00 Uhr im Gitlab vom IBR. Es werden nur ausführbare und rechtzeitig eingereicht Lösungen bewertet. Python-Programme müssen mit python3 interpretierbar, und Java-Programme mit Java 11 kompilierbar sein. Sofern nicht anders in der Aufgabe angegeben, dürfen **keine** Standardfunktionen benutzt werden, wenn sie nicht in der Vorlesung behandelt wurden.

Sofern nicht anders in der Aufgabe beschrieben, soll im Git die Ordnerstruktur wie vom Blatt vorgegeben beibehalten werden. D.h. für eine Aufgabe x von Blatt y sollen die verwendeten Source-Dateien (.py oder .java) in dem Ordner blatt[y]/pflichtaufgabe[x]/ gespeichert werden.

Die Aufgaben (und auch die Ordnerstruktur) stehen im Vorlesungsgit zum Herunterladen bereit.[1]

Hausaufgabe 1 (Binäre Suche auf Arrays): **(6 Punkte)**

Auf Blatt 1 haben wir bereits binäre Suche in Python kennengelernt. Dies soll nun in Java umgesetzt werden. Da binäre Suche nur auf sortierten Arrays funktioniert, wollen wir nun drei Methoden entwickeln: (1) Eine Methode `isSorted`, die überprüft, ob ein übergebenes Array sortiert ist; (2) Eine Methode `sort`, die ein übergebenes Array sortiert; (3) Eine Methode `binarySearch`, die überprüft, ob in einem gegeben Array eine gegebene Zahl existiert. Für die binäre Suche ist dabei nötig, dass das Array zunächst sortiert wird, falls es unsortiert ist. Wir gehen zunächst davon aus, dass das übergebene Array nur Integer-Werte enthält.

a) Implementiere die Funktion `isSorted` in der Klasse `Algorithmen`.

b) Implementiere die Funktion `sort` in der Klasse `Algorithmen` mit einem iterativen Sortierverfahren deiner Wahl. Nutze dabei Seiteneffekte aus.

   Beispielsweise kann Bubblesort implementiert werden (siehe Abbildung 1).

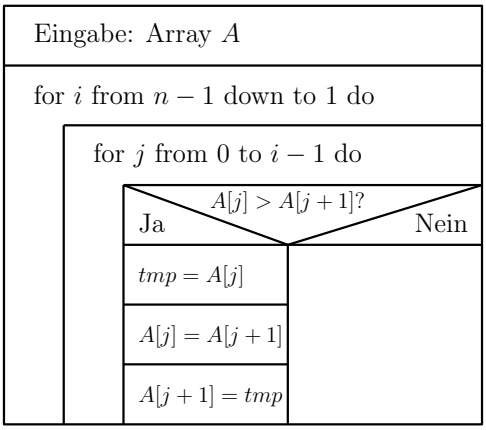

Abbildung 1: Struktogramm für das iterative Sortierverfahren Bubblesort

c) Implementiere `binarySearch` in der Klasse `Algorithmen`. Orientiere dich dabei an das Struktogramm von Blatt 1.

[1] `https://gitlab.ibr.cs.tu-bs.de/prog1/ws2324/material/unterlagen/-/tree/main/hausaufgaben/`

d) Ergänze die Main-Funktion in der Datei `main.java`, in welcher ein unsortiertes Array A mit mindestens 10 Integer-Werten erstellt wird. Lasse dann in A einmal nach einem Wert suchen, welcher in A existiert, und einmal nach einem Wert, welcher nicht in A existiert.

(Hinweis: Um statische Methoden der Klasse Algorithmen aus der main-Klasse heraus zu nutzen, muss man die Anweisung Algorithmen.Methodenname(...) nutzen)

Hausaufgabe 2 (Matrizen): **(9 Punkte)**

In dieser Aufgaben wollen wir Methoden implementieren, welche Matrizen transponieren, addieren und multiplizieren (für Beispiele, siehe Abbildung 2 oder Wikipedia[2]). Betrachte im Folgenden eine $m \times n$ Matrix A (also m Zeilen und n Spalten; auch *Dimensionen der Matrix* genannt).

$$A := \begin{pmatrix} 1 & 2 \\ 2 & 2 \\ 3 & 2 \end{pmatrix} \qquad A^T = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 2 & 2 \end{pmatrix}$$

$$A + A = \begin{pmatrix} 2 & 4 \\ 4 & 4 \\ 6 & 4 \end{pmatrix} \qquad A^T * A := \begin{pmatrix} 14 & 12 \\ 12 & 12 \end{pmatrix} \qquad A * A^T := \begin{pmatrix} 5 & 6 & 7 \\ 6 & 8 & 10 \\ 7 & 10 & 13 \end{pmatrix}$$

Abbildung 2: Beispiele von Matrix-Operationen auf der 3×2 Matrix A (oben links): Transposition A^T (oben rechts), Addition (unten links), sowie Multiplikation (unten mittig und rechts).

Bei der Transposition einer $m \times n$ Matrix A werden Zeilen und Spalten vertauscht, d.h. Spalte 1 wird Zeile 1 (und umgekehrt), Spalte 2 wird Zeile 2 (und umgekehrt), etc. Das Resultat ist dann eine $n \times m$ Matrix.

Die Addition einer $m \times n$ Matrix A mit einer $m \times n$ Matrix B ergibt eine $m \times n$-Matrix C, wobei $C[i][j] := A[i][j] + B[i][j]$. Dabei greift bspw. $A[i][j]$ auf das Element in Zeile i, Spalte j zu.

Die Multiplikation einer $m \times n$ Matrix A mit einer $n \times \ell$ Matrix B ergibt eine $m \times \ell$-Matrix C, für welche gilt:

$$C[i][j] := \sum_{k=0}^{n-1} A[i][k] * B[k][j]$$

a) Implementiere die Methode `int[][] transpose(int[][] matrix)`, welche als Eingabe eine Matrix `matrix` erhält und die transponierte Matrix zurückgibt.

b) Implementiere die Methode `int[][] add(int[][] A, int[][] B)`, welche zwei Matrizen A und B erhält und die Summe beider Matrizen zurückgibt. Sollten die Dimensionen nicht zusammenpassen, soll eine entsprechende Nachricht auf der Konsole ausgegeben werden und eine 0×0 Matrix zurückgegeben werden.

c) Implementiere die Methode `int[][] mult(int[][] A, int[][] B)`, welche zwei Matrizen `A` und `B` erhält und die Multiplikation beider Matrizen zurückgibt. Sollten die Dimensionen nicht zusammenpassen, soll eine entsprechende Nachricht auf der Konsole ausgegeben werden und eine 0×0-Matrix zurückgegeben werden.

d) Implementiere eine Methode, um eine Matrix gut lesbar auf der Konsole auszugeben: Die Werte jeder Zeile sollen mit einem Leerzeichen in einer Zeile der Konsole stehen. Bspw. liefert die Matrix aus Abbildung 2 folgende Ausgabe:

```
1 2
2 2
3 2
```

[2] `https://de.wikipedia.org/wiki/Matrix_(Mathematik)`

e) Implementiere eine main-Methode, in welcher drei Matrizen mit folgenden Dimensionen erstellt werden:

- Eine 3×2-Matrix A
- Eine 4×4-Matrix B
- Eine 2×4-Matrix C

Jede dieser Matrix soll nur ganzzahlige und mindestens fünf verschiedene Werte enthalten.

Bestimmt nun folgende Matrizen, indem das Ergebnis ausgegeben wird (sollte kein Ergebnis möglich sein, soll nur die Fehlermeldung der jeweiligen Methode ausgegeben werden): A transponiert, A*B, B+B und A*C

(Hinweis: Um statische Methoden der Klasse Matrix aus der main-Klasse heraus zu nutzen, muss man die Anweisung Matrix.Methodenname(...) nutzen)