Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Marius Weidner
Simon Dorer, Timpe Hörig

Universität Freiburg
Institut für Informatik
Wintersemester 2025

Übungsblatt 7

Abgabe: Montag, 01.12.2025, 9:00 Uhr

Aufgabe 7.1 (Daten-Modellierung; 10 Punkte; Datei: `chess.py`)

In der Vorlesung haben Sie sich mit der Modellierung von Daten mittels *Datenklassen*, *Aufzählungstypen* und *Typaliase* beschäftigt. In dieser Aufgabe sollen Sie ein Datenmodell für ein Schachspiel entwerfen und im Anschluss eine Funktion implementieren, die überprüft, ob ein geplanter Zug für eine bestimmte Figur auf ein neues Feld gültig ist. Gehen Sie dabei wie folgt vor:

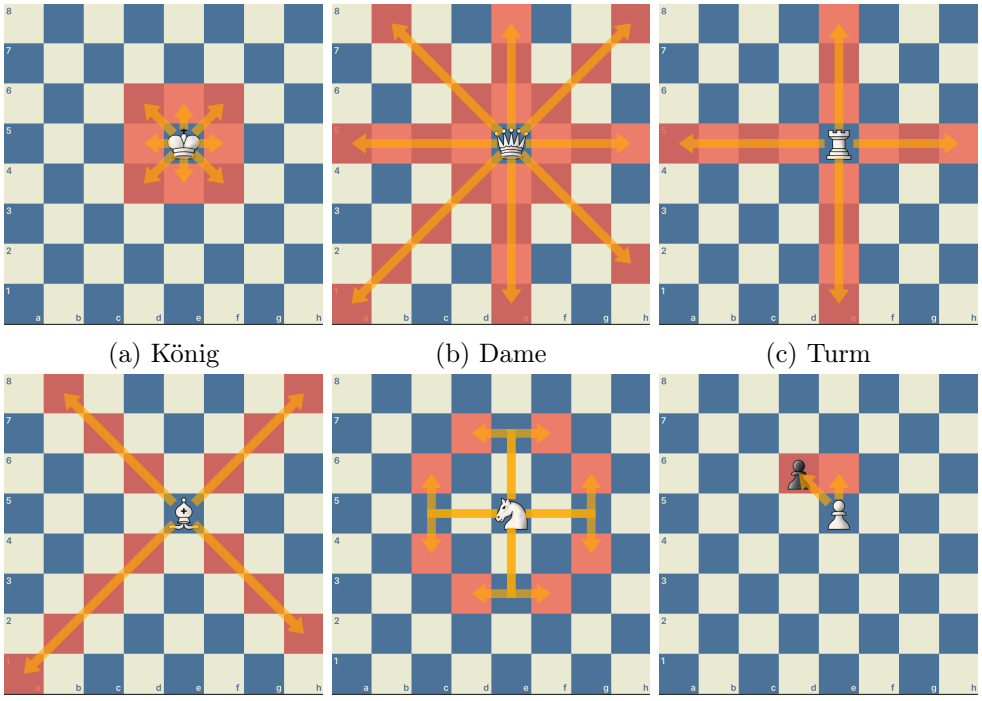
(a) (5 Punkte) Modellieren Sie die folgende umgangssprachliche Beschreibung in Python. Überlegen Sie sich dafür, was geeignete Repräsentationen für die beschrieben Elemente sind und wie sie diese sinnvoll untereinander in Bezug setzen können. Orientieren Sie sich gerne hierbei am Rommé-Beispiel aus der Vorlesung. Sie dürfen alle bereits in der Vorlesung vorgestellten Datentypen verwenden. **Bevorzugen Sie jedoch Datenklassen, Aufzählungstypen und Typaliase, wo es sinnvoll ist.**

- Eine Schachfigur hat eine Farbe (Schwarz oder Weiß) und einen Typ (König, Dame, Turm, Läufer, Springer, Bauer).
- Ein Schachbrett besteht aus 64 Feldern, wobei ein Feld durch seine Spalte (A-H) und seine Zeile (1-8) bezeichnet wird. Auf jedem Feld des Schachbretts steht *höchstens* eine Schachfigur (d.h. das Feld kann auch leer sein).

(b) (5 Punkte) Implementieren Sie eine Funktion `is_valid_move`, die eine Instanz ihres modellierten Schachbretts (siehe Aufgabenteil (a)), ein Startfeld und ein Zielfeld entgegennimmt und einen Wahrheitswert zurückgibt, der angibt, ob auf dem Startfeld eine Figur steht, die auf das Zielfeld ziehen darf. **Verwenden Sie in Ihrer Implementierung Pattern Matching, um den Typ der Figur zu bestimmen und die Zugregeln entsprechend anzuwenden.**

Beachten Sie dabei die folgenden Zugregeln der einzelnen Figuren (eine visuelle Darstellung der möglichen Züge finden Sie in Abbildung 1):

- Figuren können sich nur auf Felder bewegen, auf denen keine andere Figur derselben Farbe steht.
- Der König kann sich höchstens ein Feld in jede Richtung bewegen (waagerecht, senkrecht oder diagonal).
- Die Dame kann sich beliebig viele Felder in jede Richtung bewegen (waagerecht, senkrecht oder diagonal).



(a) König (b) Dame (c) Turm

(d) Läufer (e) Springer (f) Bauer

Abbildung 1: Mögliche Züge der Schachfiguren

- Der Turm kann sich beliebig viele Felder waagerecht oder senkrecht bewegen.
- Der Läufer kann sich beliebig viele Felder diagonal bewegen.
- Der Springer bewegt sich in einem L-förmigen Muster: zwei Felder in eine Richtung (waagerecht oder senkrecht) und dann ein Feld rechtwinklig dazu.
- Bauern können sich ein Feld *vorwärts* bewegen, wenn das Zielfeld frei ist, oder ein Feld *diagonal vorwärts* ziehen, wenn sich dort eine Figur der anderen Farbe befindet. Die Richtung *vorwärts* ist für weiße Bauern die Richtung von Reihe 1 zu Reihe 8 und für schwarze Bauern die Richtung von Reihe 8 zu Reihe 1.
- Für diese Aufgabe können Sie alle weiteren Sonderregeln (z.B. Rochade, en passant, Schach, Schachmatt, ...) ignorieren.
- Sie dürfen ebenfalls davon ausgehen, dass zwischen dem Start- und Zielfeld keine anderen Figuren stehen (d.h. Sie müssen nicht überprüfen, ob der Weg für Turm, Läufer oder Dame frei ist).

Hinweis: Um die Züge der Einzelnen Figuren zu überprüfen, kann es hilfreich sein, die Differenz der Spaltenindizes (`dx`) und der Zeilenindizes `(dy)` zwischen

dem Start- und Zielfeld zu berechnen. Mit Hilfe von `dx` und `dy` können Sie dann die Zugregeln der Figuren einfach formulieren. Ein Beispiel:

Zieht ein König z.B. vom Feld F5 auf das Feld G4, so beträgt die Differenz der Spaltenindizes `dx = 7 - 6 = 1` und die Differenz der Zeilenindizes `dy = 4 - 5 = -1`.

Dieser Zug ist für den König gültig, da `max(abs(dx), abs(dy)) == 1` gilt. Für einen Turm wäre dieser Zug hingegen ungültig, da hierfür entweder `dx == 0` oder `dy == 0` gelten müsste.

Überlegen Sie sich, wie sie `dx` und `dy` für die anderen Figuren verwenden können, um die Gültigkeit der Züge zu überprüfen.

Aufgabe 7.2 (Bäume; 10 Punkte; Datei: `trees.py`)

In der Vorlesung haben Sie Bäume als Datenstruktur kennengelernt:

```
@dataclass
class Node[T]:
    mark : T
    left : 'BTree[T]' = None
    right: 'BTree[T]' = None

type BTree[T] = Optional[Node[T]]
```

(a) (3 Punkte) Schreiben Sie eine Funktion `sum_tree`, die einen binären Baum `btree` als Argument nimmt, dessen Markierungen von Typ `int` sind und die Summe aller Markierungen in `btree` zurückgibt. Falls `btree` leer ist, soll `0` zurückgegeben werden. **Verwenden Sie Rekursion und Pattern Matching.**

```
>>> sum_tree(None)
0
>>> sum_tree(Node(3))
3
>>> sum_tree(Node(18, Node(5, Node(19)), Node(7, Node(-3),
↪  Node(9))))
55
```

(b) (3 Punkte) Schreiben Sie eine Funktion `mirror_at`, die einen beliebigen Binärbaum `btree` und eine Markierung `at` als Argument nimmt und einen *neuen Baum* zurückgibt, bei dem die linken und rechten Teilbäume aller Knoten vertauscht wurden, die mit `at` markiert sind. **Verwenden Sie Rekursion und Pattern Matching.**

```
>>> tree = Node(1, Node(2, Node(3)), Node(1, Node(5), Node(1)))
>>> mirror_at(None, 42)
>>> mirror_at(Node("Hallo Welt"), "Hallo Welt")
Node('Hallo Welt')
>>> mirror_at(tree, 42)
```

```
Node(1, Node(2, Node(3)), Node(1, Node(5), Node(1)))
>>> mirror_at(tree, 2)
Node(1, Node(2, None, Node(3)), Node(1, Node(5), Node(1)))
>>> mirror_at(tree, 1)
Node(1, Node(1, Node(1), Node(5)), Node(2, Node(3)))
```

(c) (4 Punkte) Schreiben Sie eine Funktion `path_to_subtree`, die zwei Binärbäume `btree` und `subtree` als Argumente nimmt und den Pfad zu `subtree` zurückgibt, falls dieser als Teilbaum in `btree` vorkommt, ansonsten `None`.

Der Pfad soll dabei als String aus den Zeichen `"L"` und `"R"` angegeben werden, wobei `"L"` einen Schritt in den linken und `"R"` einen Schritt in den rechten Teilbaum markiert. Sind beide Bäume identisch, so ist der zurückgegebene Pfad `""`. Befindet sich der Teilbaum in einem der Kinder, so soll der Pfad durch das entsprechende Präfix `"L"` bzw. `"R"` angegeben werden.

Falls `subtree` mehrfach in `btree` vorkommt, geben Sie den Pfad zum ersten Vorkommen zurück. Untersuchen Sie dabei stets zuerst den linken und dann den rechten Teilbaum. **Verwenden Sie Rekursion und Pattern Matching.**

```
>>> subtree = Node(1, Node(2))
>>> path_to_subtree(None, None)
''
>>> path_to_subtree(None, Node("Hello World"))  # None
>>> path_to_subtree(subtree, subtree)
''
>>> path_to_subtree(Node(0, Node(1337, None, subtree)), subtree)
'LR'
>>> path_to_subtree(Node(18, Node(5, Node(19)), Node(7, Node(-3),
↪  Node(9))), subtree)  # None
```

Aufgabe 7.3 (Erfahrungen; Datei: `NOTES.md`)

Notieren Sie Ihre Erfahrungen mit diesem Übungsblatt (benötigter Zeitaufwand, Probleme, Bezug zur Vorlesung, Interessantes, etc.).

Editieren Sie hierzu die Datei `NOTES.md` im Abgabeordner dieses Übungsblattes auf unserer Webplattform. Halten Sie sich an das dort vorgegebene Format, da wir den Zeitbedarf mit einem Python-Skript automatisch statistisch auswerten. Die Zeitangabe `7.5 h` steht für 7 Stunden 30 Minuten.

Der Build-Server überprüft ebenfalls, ob Sie das Format korrekt angegeben haben. Prüfen Sie, ob der Build-Server mit Ihrer Abgabe zufrieden ist, so wie es im Video zur Lehrplattform gezeigt wurde.