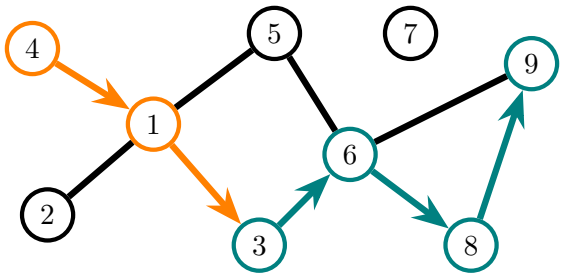
Technische Universität Berlin **SoSe 2024**
Fakultät II, Institut für Mathematik
Sekretariat MA 6–2, Dorothea Kiefer-Hoeft
Prof. Dr. Max Klimm
Dr. Tobias Hofmann, Martin Knaack, Marcel Wack

1. Programmieraufgabe Computerorientierte Mathematik II

Abgabe: 3.05.2024 über den Comajudge bis 17:00 Uhr



Zwei Pfade f und g in einem Graphen.

Pfad in einem Graphen

Definieren Sie einen (”nicht-mutable”) Typ `Pfad`, der einen Pfad in einem Graphen repräsentiert. Ein `Pfad` besitzt dazu die beiden Felder `source::Real` und `target::Union{Real,Pfad}`.

> **Hinweis:** `Union{Real,Pfad}` bedeutet, dass das Feld `target` entweder ein Element vom Typ `Real` oder ein Element vom Typ `Pfad` sein kann.

Schreiben Sie für `Pfad` eine Konstruktorfunktion `pfad` mit 3 Methoden:

1. `pfad(source::Real, target::Real)::Pfad` erzeugt einen Pfad, der nur aus einer Kante besteht.
2. `pfad(source::Real, target::Pfad)::Pfad` konstruiert einen Pfad, der aus einem Pfad besteht und diesen am Anfang um eine Zahl erweitert.
3. `pfad(source::Real)::Pfad` erstellt einen Pfad aus einem einzelnen Knoten, der sowohl `source` als auch `target` ist.

Julia

```
julia> x = pfad(1, 3)
Pfad(1, 3)

julia> pfad(4, x)
Pfad(4, Pfad(1, 3))
```

Infix-Operator und `show` Methode

Definieren Sie danach einen Infix-Operator ⇒ für die ersten beiden Konstruktor-Methoden von oben. Den Unicode-Zeichen erhalten Sie durch Eingabe von \R[tab] in der `Julia-REPL`. (Unicode-Zeichen: U+21D2)

Julia

```
julia> x = 1 ⇒ 3
Pfad(1, 3)

julia> 4 ⇒ x
Pfad(4, Pfad(1, 3))
```

Zur besseren Lesbarkeit überladen Sie zusätzlich die Funktion `show(io::IO, p::Pfad)` aus dem `Base` um einen Pfad in der Form `source ⇒ target` auszugeben. Wenn das Feld `target` ein `Pfad` ist, wird dieser rekursiv ausgegeben.

Julia

```
julia> x = 1 ⇒ 3
x = 1 ⇒ 3

julia> 4 ⇒ x
4 ⇒ 1 ⇒ 3
```

Konkatination von Pfaden

Überladen Sie die Funktion `*(f::Pfad, g::Pfad)` so, dass zwei Pfade f und g zu einem neuen Pfad verkettet werden. Der neue Pfad sollte so aussehen, dass das `target` von f das `source` von g ist. Wenn das `target` von f ein `Pfad` ist, muss g an das Ende dieses Pfades angehängt werden.

Hinweis: Es ist erlaubt, `@assert` zu verwenden, um sicherzustellen, dass die letzte Zahl von f und die erste Zahl von g übereinstimmen. Ein Beispiel

```
julia> @assert 2 == 3
ERROR: AssertionError: 2 == 3
```

Julia

```
julia> f = 4 ⇒ 1 ⇒ 3
4 ⇒ 1 ⇒ 3

julia> g = 3 ⇒ 6 ⇒ 8 ⇒ 9
3 ⇒ 6 ⇒ 8 ⇒ 9

julia> f * g
4 ⇒ 1 ⇒ 3 ⇒ 6 ⇒ 8 ⇒ 9

julia> f * ( 4 ⇒ 3 )
ERROR: AssertionError: target(z1) == source(z2)
```

Zusammengefasst

Implementieren Sie die folgenden Funktionen:

(a) (nicht-mutable) Type `Pfad` mit den Feldern `source` und `target`.

(b) Konstruktor `pfad` mit 3 Methoden.

(c) Infix-Operator $\Rightarrow$ für die ersten beiden Konstruktormethoden.

(d) Überladung der Funktion `show(io::IO, p::Pfad)`.

(e) Überladung der Funktion `*(f::Pfad, g::Pfad)`.

(f) * sollte ein Fehler ausgeben, falls die letzten Zahl von f und die erste Zahl von g nicht übereinstimmen.