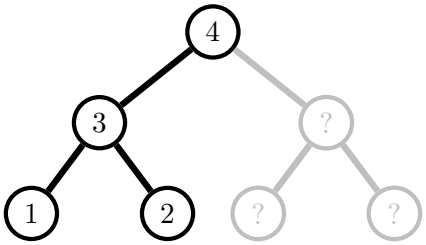
Technische Universität Berlin **SoSe 2024**
Fakultät II, Institut für Mathematik
Sekretariat MA 6–2, Dorothea Kiefer-Hoeft
Prof. Dr. Max Klimm
Dr. Tobias Hofmann, Martin Knaack, Marcel Wack

2. Programmieraufgabe Computerorientierte Mathematik II

Abgabe: 10.05.2024 über den Comajudge bis 17:00 Uhr



Der Binärbaum `[4,3,nothing,1,2,nothing,nothing]`.

Binärbäume

Implementieren Sie in Julia einen Binärbaum, der beliebige Werte des Typs `Int` speichert. Beginnen Sie mit der Definition des Typs `Node` mit den Feldern `key::Int`, `left::Union{Node,Nothing}` und `right::Union{Node,Nothing}`. Ein Knoten kann entweder ein Blatt sein oder ein oder zwei Kinder haben. Ein Blatt ist ein Knoten ohne Kinder. Wir geben einen *binären Baum* durch seinen Wurzelknoten an. Definieren Sie `MaybeNode` als *Typalias* für `Union{Node, Nothing}`.

Hinweis: Ein Typalias ist ein neuer Name für einen vorhandenen Typ. In diesem Fall ist `MaybeNode` ein neuer Name für den Typ `Union{Node, Nothing}`. Und wird wie folgt verwendet:

```
julia> IchBinEinAlias = Union{Int, Float64}
Union{Float64, Int64}
julia> isa(2, IchBinEinAlias)
true
```

Anschließen Implementieren Sie zwei Konstruktoren für den Typ `Node`:

1. `node(key::Int)`: Erzeugt ein Blatt mit dem Wert `key`.
2. `node(key::Int, left::MaybeNode, right::MaybeNode)`: Erzeugt einen Knoten mit dem Wert `key` und den Kindern `left` und `right`.

Julia

```
julia> x = node(1)
Node(1, nothing, nothing)
julia> y = node(2)
Node(2, nothing, nothing)
julia> z = node(3, x, y)
Node(3, Node(1, nothing, nothing), Node(2, nothing, nothing))
```

Die Funktionen `getKeys`,`height` und `leaves`

Schreiben Sie 2 nützliche Funktionen für den Typ `Node`:

1. `getKeys(node::Node)::Vector{Int}`: Gibt alle Schlüssel des Baums in einem Vector zurück. Die sortierung können Sie vernachlässigen.

   getKeys

   ```
   julia> getKeys(z)
   3-element Vector{Int64}:
    3
    1
    2
   ```

2. `height(node::Node)::Int`: Gibt die Höhe eines Baumes zurück. Die Höhe eines Baumes ist die maximale Anzahl von Kanten auf dem längsten Pfad von der Wurzel zu einem Blatt. Ein Baum mit nur einem Knoten hat die Höhe 1.

   height

   ```
   julia> height(z)
   2
   ```

Binärbäume als Arrays

Aus der Vorlesung kennen sie die Möglichkeit, (fast vollständige) Bäume als Arrays zu interpretieren. Sie sollen nun zwei Funktionen implementieren, die einen Baum in einen Vektor umwandeln und umgekehrt. Konkret sollen Sie folgende Funktionen realisieren:

1. `tree2vec(node::Node)::Vector{Union{Int,Nothing}}`: Konvertiert einen Baum in ein Array. Die Idee ist, dass der Schlüssel der Wurzel des Baumes an Position 1 des Vektors steht, das linke Kind an Position 2 und das rechte Kind an Position 3. Rekursiv sollen nun für einen Knoten des Baumes an Position i die Kinder an den Positionen $2i$ und $2i+1$ stehen. Falls ein Kind nicht existiert, soll an der entsprechenden Stelle `nothing` stehen. Der Vektor sollte die Länge haben, die ein vollständiger binärer Baum der gleichen Höhe benötigt.

2. `vec2tree(vec::Vector{Union{Int,Nothing}})::Node`: Sei ein Vektor der korrekten Länge, der einen binären Baum mit obiger Einbettung speichert. Die Funktion sollte ein Objekt vom Typ `Node` zurückgeben, das den Baum als Graph darstellt.

> **Hinweis:** Ein möglicher Ansatz wäre, die Funktion `tree2vec` rekursiv zu implementieren, indem ein optionales Argument `i` übergeben wird, das den Index des aktuellen Knotens im Vektor speichert. Bei Bedarf können Sie weitere optionale Argumente hinzufügen. Denken Sie an `Julia's` *multiple dispatch*, um Funktionen zu überladen.

Beispiele

tree2vec

```
julia> u = node(4, z, nothing)
Node(4, Node(3, Node(1, nothing, nothing), Node(2, nothing, nothing)), nothing)

julia> tree2vec(u)
7-element Vector{Union{Nothing, Int64}}:
 4
 3
  nothing
 1
 2
  nothing
  nothing
```

vec2tree

```
julia> vec2tree([4,3,nothing,1,2,nothing,nothing])
Node(4, Node(3, Node(1, nothing, nothing), Node(2, nothing, nothing)), nothing)
```

Zusammengefasst

Implementieren Sie die folgenden Funktionen:

(a) `node(key::Int)`: Erzeugt ein Blatt mit dem Wert `key`.

(b) `node(key::Int, left::MaybeNode, right::MaybeNode)`: Erzeugt einen Knoten mit dem Wert `key` und den Kindern `left` und `right`.

(c) `getKeys(node::Node)::Vector{Int}`: Gibt alle Schlüssel des Baumes in einer Liste zurück.

(d) `height(node::Node)::Int`: Gibt die Höhe des Baumes zurück.

(e) `tree2vec(node::Node)::Vector{Union{Int,Nothing}}`: Wandelt einen Baum in ein Array um.

(f) `vec2tree(vec::Vector{Union{Int,Nothing}})::Node`: Wandelt ein Array in einen Baum um.