252-0027-00: Einführung in die Programmierung

Zusatzaufgabe Node-Tree

Dies ist keine offizielle EProg-Aufgabe und wurde nicht vom EProg-Team abgesegnet. Sie dient nur dazu, in der Übungsstunde einige Konzepte zu üben und zu besprechen.

Aufgabe 1: Node-Tree

In dieser Aufgabe müssen sie mehrere Methoden implementieren, die verschiedene Werte auf Graphen berechnen.

Die Klasse `Node` repräsentiert einen Knoten in einem solchen Graphen. Der Wert einer Node kann mit der Methode `getValue()` abgerufen werden. Eine Node hat 0-2 Nachfolger. Diese sind in den fields `first` und `second` gespeichert. Wenn eine Node keine Nachfolger hat, dann sind `first` und `second` beide `null`. Falls eine Node genau einen Nachfolger hat, dann kann dieser entweder in `first` oder in `second` gespeichert sein. Falls eine Node zwei Nachfolger hat, dann ist der eine in `first` und der andere in `second` gespeichert. **Wichtig: Sie dürfen die Node Klasse nicht verändern! Alle Aufgaben sind lösbar, ohne die Node Klasse zu verändern.**

Ein Baum beginnend in Node origin ist der Graph, der Node `origin` enthält und gerichtete Kanten zu all dessen Nachfolgern hat. Jeder Nachfolge-Node hat wieder gerichtete Kanten zu all seinen Nachfolgern usw. Sie dürfen annehmen, dass dieser Graph azyklisch ist und kein Node Nachfolger von mehreren verschiedenen Nodes ist. Abbildung 1 zeigt ein Beispiel eines solchen Baumes. **Tipp:** Vielleicht implementieren Sie die Methoden rekursiv.

Tests für alle Teilaufgaben finden Sie in der Datei "Task2Test.java".

1. Implementieren sie die Methode `Task2.calculateSum(Node origin)`, welche die Summe der Werte aller Knoten zurückgibt, die Teil des Baumes beginnend in `origin` sind. Für das Beispiel in Abbildung 1 sollte die Methode `9` zurückgeben. Sie können davon ausgehen, dass `origin` nicht null ist.

2. Ein Pfad in einem Baum (wieder beginnend in `origin`) ist eine Folge von Knoten, die bei `origin` beginnt und in einem nachfolgerlosen Knoten endet. Jeder Knoten (ausser `origin`) dieser Folge ist ein Nachfolger-Knoten des vorherigen Knoten in der Folge. Das Beispiel in Abbildung 1 enthält 3 Pfade: `[2, -3]`, `[2, 4, -2, 7]` und `[2, 4, 1]`.

   Der Wert eines Pfades ist gegeben durch die Summer der Werte aller Knoten, die in dem Pfad enthalten sind. Der höchste Pfad eines Baumes ist der Pfad, der den höchsten Wert hat. Implementieren Sie die Methode `Task2.highestPath(Node origin)`, welche den Wert des höchsten Pfades im Baum beginnend in `origin` zurückgibt. Für das Beispiel in Abbildung 1

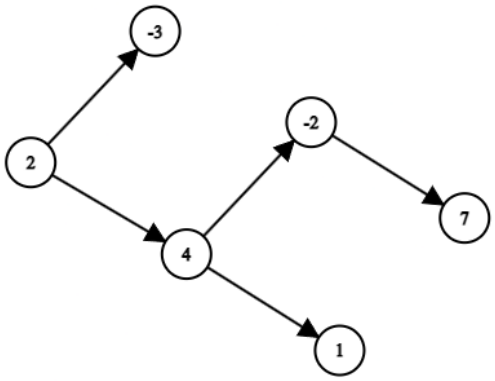

Figure 1: Ein Beispiel eines Baumes

wäre das Resultat 11, was dem Pfad [2, 4, -2, 7] entspricht. Sie können davon ausgehen, dass `origin` nicht null ist.

3. Neu gibt es die Klasse `TriNode`, die einen Knoten mit potenziell 3 Nachfolgern repräsentiert. Für diesen Zweck hat TriNode ein zusätzliches field `third`, welches entweder `null` oder einen Nachfolge-Node enthält. Analog zu Node gibt es in TriNode keine Garantien, in welchen fields die Nachfolger gespeichert sind. Wenn ein TriNode zB. nur einen Nachfolger hat, dann kann dieser entweder in `first`, oder in `second`, oder in `third` gespeichert sein. **Wichtig: Sie dürfen die TriNode Klasse nicht verändern! Ändern Sie keine fields/Methoden und fügen sie keine neuen fields/Methoden hinzu!**

   Ein Baum kann nun aus einem Mix aus Nodes und TriNodes bestehen. Sie dürfen nicht davon ausgehen, dass ein Baum nur Knoten des einen oder anderen Typen enthält. Es kann sein, dass der Baum sowohl einige `Node`, als auch einige `TriNode`-Knoten enthält. Implementieren Sie analog zu Aufgabe 1 die Methode `Task2.calculateTriSum(Node origin)`, welche die Summe der Werte aller Knoten zurückgibt, die Teil des Baumes beginnend in `origin` sind. Wichtig ist, dass Sie bei dieser Aufgabe auch das `third` field von TriNodes beachten, welches potenziell einen Nachfolge-Knoten enthält. Sie können davon ausgehen, dass `origin` nicht null ist.

   Tipp: `TriNode` ist eine Subklasse von `Node`. `TriNode`-Instanzen können also in `Node`-Referenzvariablen gespeichert werden. Wie kann man herausfinden, ob eine `Node`-Referenzvariable eine `TriNode`-Instanz enthält?

4. (EXTRA CHALLENGE) Auch in dieser Aufgabe arbeiten Sie mit Bäumen, die aus einem Mix aus Nodes und TriNodes bestehen. Implementieren Sie die Methode `Task2.highestTriPath(Node origin)`, die ein Array der Knoten auf dem höchsten Pfad zurückgibt. Für das Beispiel in Abbildung 1 wäre das Resultat ein Node-Array, das die Knoten [2, 4, -2, 7] enthält. Das Rückgabe-Array sollte Referenzen auf die gleichen Knoten enthalten, die auch Teil des Baumes sind, nicht Kopien davon. Sie können davon ausgehen, dass `origin` nicht null ist.

   Tipp: Gibt es eine Möglichkeit mehrere Werte zurückzugeben? Sie dürfen zusätzliche Klassen definieren.