Algorithmen und Programmierung 3, WS 2019/2020 — 3. Übungsblatt

Abgabe bis Freitag, 9. November 2019, 12:00 Uhr, in die Fächer der Tutoren

---

15. Binärbäume, Wiederholung, 6 Punkte

    Ein *vollständiger* binärer Baum der Höhe h ist ein binärer Baum, bei dem jeder Knoten mit Tiefe $\leq h-2$ genau zwei Kinder hat.

    (a) Zeichnen Sie vollständige binäre Bäume mit 12 und mit 15 Knoten.

    (b) Wieviele Knoten kann ein vollständiger binärer Baum der Höhe h mindestens und höchstens haben?

    (c) Welche Höhe kann ein vollständiger binärer Baum mit n Knoten haben?

    Begründen Sie Ihre Antworten zu Teil (b) und (c).

16. Umwandeln einer Liste in einen Baum, 0 Punkte (eine ehemalige Klausuraufgabe)

    Schreiben Sie ein Programm in Java oder Haskell, das eine sortierte Liste von ganzen Zahlen in einen binären Suchbaum der kleinstmöglichen Höhe h umwandelt.

17. Stückweise konstante Funktionen, 0 Punkte

    Wie wird die stückweise konstante Funktion $h(x) = \lfloor x/2 \rfloor - \lceil (1-x)/3 \rceil$, definiert für $0 \leq x < 10$ in dem Format, das in der Vorlesung vorgestellt wurde, dargestellt?

18. Verschieben von Funktionen, 7 Punkte, Programmieraufgabe

    (a) Schreiben Sie eine PYTHON-Funktion, die eine stückweise konstante Funktion F um einen Parameter $b \in \mathbb{R}$ verschiebt:

    ```
    H = verschiebe(F,b)
    ```

    Es soll $H(x) = F(x+b)$ sein überall dort, wo $F(x+b)$ definiert ist.

    (b) Zusatzfrage, 0 Punkte
    Wie entsteht der Graph von H aus dem Graphen von F, wenn $b = 1$ ist?

19. Fallunterscheidung, 7 Punkte, Programmieraufgabe

    Schreiben Sie eine PYTHON-Funktion

    ```
    H = fallunterscheidung(b,F,G)
    ```

    die bei Eingabe von zwei stückweise konstanten Funktion F und G und einem Parameter $b \in \mathbb{R}$ die Funktion H erzeugt, die folgendermaßen definiert ist:

$$H(x) = \begin{cases} F(x), & \text{wenn } x < b \\ G(x), & \text{wenn } x \geq b \end{cases}$$

    Ergänzen Sie diese Spezifikation, falls Sie es für erforderlich halten, zum Beispiel durch Angabe von passenden Vorbedingungen.

20. Gleichheitstest, 0 Punkte

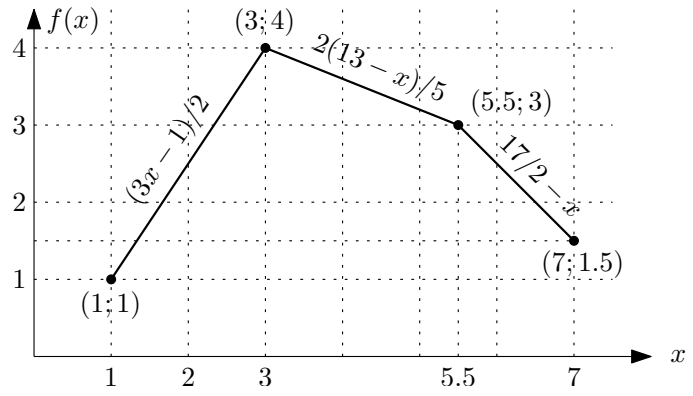
    Schreiben Sie eine Funktion, die zwei stückweise konstante Funktionen auf Gleichheit testet. Die Definition der Gleichheit ist dabei die übliche Definition für Funktionen: Zwei Funktionen sind gleich, wenn sie den gleichen Definitionsbereich haben und wenn sie an jeder Stelle des Definitionsbereiches denselben Wert liefern.

21. Erzeugen stückweise konstanter Funktionen, 0 Punkte

    In der Vorlesung wurden gar nicht angegeben, wie man eine stückweise konstante Funktion *erzeugen* kann. Ohne diese Möglichkeit sind stückweise konstante Funktionen *als abstrakter Datentyp* aber gar nicht brauchbar. Spezifizieren Sie eine oder mehrere Operationen für einen abstrakten Datentyp „stückweise konstante Funktion“, die es ermöglichen, beliebige stückweise konstante Funktionen zu erzeugen. Sie können dabei die in den vorigen Aufgaben definierten Operationen zu Hilfe nehmen.

22. Stückweise lineare stetige Funktionen, 0 Punkte

    Statt stückweise *konstanten* Funktionen wollen wir *stetige* und stückweise *lineare* Funktionen betrachten, die auf einem abgeschlossenen Intervall $[a, b]$ definiert sind, wie die Funktion $f(x)$ im folgenden Beispiel:



    Überlegen Sie sich eine Datenstruktur (Haskell-Datentyp, Java-Klasse, oder Python-Klasse) zur Darstellung solcher Funktionen. Schreiben Sie eine Funktion/Methode, die den Wert einer solche Funktion $f(x)$ an einer gegebenen Stelle x ausrechnet.

23. Stückweise konstante Funktionen in Java, 0 Punkte

    (a) Schreiben Sie eine abstrakte Klasse oder ein *Interface* für den abstrakten Datentyp `stückweise_konstante_Funktion` mit den Operationen aus der Vorlesung und aus den obigen Aufgaben.

    (b) Implementieren Sie den Datentyp.

24. Stückweise konstante Funktionen in Haskell, 0 Punkte

    (a) Durch welche Merkmale der Sprache Haskell wird das Konzept der abstrakten Datentypen unterstützt? (Typklassen? Module?)

    (b) Implementieren Sie stückweise konstante Funktionen in Haskell.

25. Implementierung einer Menge als Bitvektor, 0 Punkte

    Betrachten Sie die rudimentäre Implementierung von Mengen ganzer Zahlen mit der Methode `contains` und dem Konstruktor in der Java-Klasse `BitSet.java`[1].

    (a) Ergänzen Sie die Spezifikation der Mengenoperationen durch die nötigen Vorbedingungen für diese Implementierung.

    (b) Geben Sie die Abstraktionsfunktion und gegebenfalls die Darstellungsinvarianten dieser Implementierung an.

    (c) Implementieren Sie die beiden restlichen Methoden.

[1] `https://mycampus.imp.fu-berlin.de/x/WEbUPB`