technische universität
dortmund

Fakultät für Informatik
Lehrstühle 2, 11, 13

Jun.-Prof. Dr. Maike Buchin
I. Kamehkhosh, N. Kriege, D. Kurz, M. Ludewig, B. Zey

Sommersemester 2018
5. Juni

DAP2 Praktikum – Blatt 9

Abgabe: 11.–15. Juni

Wichtig: Der Quellcode ist natürlich mit sinnvollen Kommentaren zu versehen. Überlegen Sie außerdem, in welchen Bereichen Invarianten gelten müssen, und überprüfen Sie diese ggf. an sinnvollen Stellen mit *Assertions* (siehe Hinweis auf Blatt 2).

Kurzaufgabe 9.1: Traversieren von Binärbäumen (4 Punkte)

Ein *Suchbaum* ist ein binärer Baum mit der Eigenschaft, dass der Wert an jedem Knoten v größer ist als jeder Wert im linken Teilbaum von v und kleiner als jeder Wert im rechten Teilbaum von v. Um in einen bestehenden Suchbaum einen neuen Wert w einzufügen, soll folgendermaßen vorgegangen werden: Zunächst wird die Wurzel des Baums betrachtet. Ist der Wert hier kleiner als w, so wird w in den rechten Teilbaum eingefügt, und in den linken Teilbaum sonst. Das Vorgehen wird wiederholt bis ein leerer Teilbaum erreicht wurde. Dort kann der Wert w gespeichert werden. Der Suchbaum darf auch Werte doppelt beinhalten.

(a) Erstellen Sie zeichnerisch auf Papier einen initial leeren Suchbaum, in den nacheinander die folgenden Zahlen eingefügt werden:

$$13, 17, 5, 3, -10, 100, 40, -5, 4, 12, 11$$

(b) Ermitteln Sie die Zahlenfolgen, die wir erhalten, wenn wir diesen Baum in pre-, in- und post-order traversieren.

(c) Implementieren Sie eine Klasse `SearchTree` in Java. Die Klasse soll einen Konstruktor haben, der ein Array aus Zahlen erwartet, um aus diesen Zahlen einen Suchbaum aufzubauen. Nachdem der Baum erstellt wurde, soll er in pre-, in- und post-order traversiert und ausgegeben werden.

Kurzaufgabe 9.2: Rucksackproblem (4 Punkte)

Implementieren Sie den in der Vorlesung vorgestellten Algorithmus zum Rucksackproblem. Der Algorithmus mit Laufzeit in $\mathcal{O}(nW)$ verwendet eine Tabelle zur dynamischen Programmierung, wobei n die Anzahl der Elemente und W die Gewichtsschranke ist. Gehen Sie dabei wie folgt vor:

- Ihr Programm erwartet als Eingabeparameter die Anzahl n an Waren, eine Gewichtsschranke W sowie eine Zahl p.
- Erstellen Sie die Klasse `Article`, die die Attribute `value` und `weight` bereitstellt.
- Erstellen Sie n Waren. Die Gewichte (`weight`) sollen zufällig gezogene ganze Zahlen aus dem Intervall $[0.8p, 1.25p]$ sein. Die Werte (`value`) sollen zufällig gezogene ganze Zahlen aus dem Intervall $[100, 1000]$ sein
- Messen Sie die Laufzeit des Algorithmus abhängig von n und W und stellen Sie die Ergebnisse grafisch dar. Wie verhält sich Ihr Algorithmus für ein festes n bei wachsendem W? Welche Entwicklung beobachten Sie für ein festes W und wachsendes n?
- Schreiben Sie zusätzlich einen Algorithmus, der die Artikel mit dem besten Verhältnis von Wert zu Gewicht zuerst wählt, bis die Gewichtsschranke erreicht ist.
- Geben Sie schließlich die Werte aus, die mit den obigen beiden Algorithmen erzielt wurden.