technische universität
dortmund

Fakultät für Informatik
Lehrstühle 2, 11, 13

Prof. Dr. C. Sohler
I. Kamehkhosh, N. Kriege, D. Kurz, M. Ludewig

Sommersemester 2017
26. Juni

DAP2 Praktikum – Blatt 11

Abgabe: 3. Juli–7. Juli

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 11.1: Datenstruktur Heap (4 Punkte)

Implementieren Sie eine Klasse `Heap`, welche die gleichnamige Datenstruktur zur Verfügung stellt, indem ein int-Array sowie die aktuelle Anzahl der Elemente als Variable `heapSize` verwaltet werden.

Implementieren Sie hierbei folgende Methoden:

- Einen Konstruktor, der eine natürliche Zahl `n` übergeben bekommt und die Attribute so setzt, dass der Heap initial leer ist, aber bis zu `n` Elemente aufgenommen werden können.
- Die Methoden `left(int i)`, `right(int i)`, `parent(int i)`, die zu einem Knoten `i`, den Index des linken Kindes, des rechten Kindes, bzw. des Vaterknotens zurückgeben.
- Eine Methode `heapify(int i)`, welche die Heapeigenschaft für das Element mit der Position `i` herstellt.
- Eine Methode `insert(int key)`, die das Element `key` unter Aufrechterhaltung der Heapeigenschaft in den Heap einfügt.
- Eine Methode `extractMax()`, die das größte Element des Heaps entfernt und zurückgibt.
- Eine Methode `printHeap()`, die den Heap auf der Konsole ausgibt. Das Format für die Ausgabe (Nummerierung der Knoten wie in den Folien) sollte wie folgt aussehen:

```
1
2 3
4 5 6 7
...
```

Kurzaufgabe 11.2: Anwendung eines Heaps (4 Punkte)

In dieser Aufgabe verwenden wir den Heap, um eine Prioritätenschlange zu realisieren. Diese verwaltet Jobs mit Prioritäten zwischen 0 und 100, wobei eine größere Zahl eine höhere Priorität bedeutet.

- Erstellen Sie eine Klasse `PriorityQueue` mit einer `main`-Methode, welche zwei natürliche Zahlen `n` und `k` als Parameter einliest.
- Dann soll ein Heap, der `n` + `k` Elemente aufnehmen kann, erzeugt werden.
- In diesen Heap sollen `n` Jobs eingefügt werden, wobei die Prioritäten zufällig zwischen 0 und 100 liegen sollen.
- Führen Sie nun k mal zufällig eine der beiden folgenden Aktionen durch:
  (a) in 75% der Fälle: Der gerade bearbeitete Job ist fertig geworden, der nächste Job mit der höchsten Priorität kommt dran und wird aus dem Heap entfernt.
  (b) in 25% der Fälle: Ein neuer Job kommt an (mit Priorität zufällig zwischen 0 und 100) und wird in den Heap eingefügt.

  Geben Sie nach jedem Schritt die entfernten oder neu hinzugefügten Jobs sowie den gesamten Heap aus.

Hinweise und Tipps

Ziehen von Zufallszahlen (Wiederholung)

In Java steht ein Pseudozufallszahlengenerator zur Verfügung. Die Klasse `java.util.Random` stellt den Konstuktor für einen Pseudozufallszahlengenerator, sowie die Methode `nextInt()` zur Verfügung.

Um Zufallszahlen zu erzeugen, kann wie folgt vorgegangen werden:

```
...
// Den Generator erzeugen (als Seedwert wird die Systemzeit verwendet)
java.util.Random numberGenerator = new java.util.Random();
...
//Wann immer man eine Zufallszahl braucht
int randomNumber = numberGenerator.nextInt();
...
```

Zufallszahlen in einem Bereich von [0, grenze] erreicht man durch:
`randomNumber.nextInt(grenze+1)`.