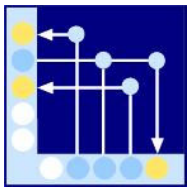

Fakultät Elektronik und Informatik
Studiengang Informatik



# Algorithmen und Datenstrukturen 2

Vorlesung im Sommersemester 2017
Prof. Dr. habil. Christian Heinlein

## 2. Praktikumsaufgabe (27. April – 29. Mai 2017)

## Aufgabe 2: Vorrangwarteschlangen

Implementieren Sie Minimum-Vorrangwarteschlangen mit Prioritäten eines beliebigen Typs `P`, der `Comparable` implementiert (siehe unten), und zusätzlichen Daten eines beliebigen Typs `D` durch Binomial-Halden als generische Java-Klasse `BinHeap<P extends Comparable<? super P>, D>` mit folgenden öffentlichen Konstruktoren und Methoden:

```
// Leere Halde erzeugen.
BinHeap ()

// Ist die Halde momentan leer?
boolean isEmpty ()

// Größe der Halde, d. h. Anzahl momentan gespeicherter Einträge liefern.
int size ()

// Enthält die Halde den Eintrag e?
boolean contains (Entry<P, D> e)

// Neuen Eintrag mit Priorität p und zusätzlichen Daten d
// zur Halde hinzufügen und zurückliefern.
Entry<P, D> insert (P p, D d)

// Priorität des Eintrags e auf p verändern.
boolean changePrio (Entry<P, D> e, P p)

// Einen Eintrag mit minimaler Priorität liefern.
Entry<P, D> minimum ()

// Einen Eintrag mit minimaler Priorität liefern und aus der Halde entfernen.
Entry<P, D> extractMin ()

// Eintrag e aus der Halde entfernen.
boolean remove (Entry<P, D> e)

// Inhalt der Halde zu Testzwecken ausgeben.
void dump ()
```

In den folgenden Fehlerfällen soll die jeweilige Methode wirkungslos sein und `false` bzw. `null` liefern:

- Wenn ein Parameter `P p` oder `Entry<P, D> e` gleich `null` ist.
- Wenn der an `changePrio` oder `remove` übergebene Eintrag `e` nicht zur aktuellen Halde gehört.
- Wenn `minimum` oder `extractMin` für eine leere Halde aufgerufen wird.

`Entry<P, D>` ist eine Hilfsklasse zur Repräsentation von Einträgen mit Prioritäten des Typs `P` und zusätzlichen Daten des Typs `D`, die folgende öffentliche Methoden zur Abfrage dieser Daten besitzt:

```
P prio ()
D data ()
```

Der Typ `P` muss die Schnittstelle `Comparable<P>` oder `Comparable<P'>` für einen Obertyp `P'` von `P` implementieren (d. h. `P extends Comparable<? super P>`), damit Objekte `p1` und `p2` dieses Typs mittels `p1.compareTo(p2)` verglichen werden können. Der Resultatwert eines solchen Methodenaufrufs ist negativ, null oder positiv, je nachdem, ob `p1` kleiner, gleich oder größer als `p2` ist.

Aus der Ausgabe von `dump` muss hervorgehen, wie die einzelnen Binomialbäume der Halde aufgebaut sind. Wenn in einen anfangs leeren `BinHeap<String, Integer>` nacheinander die Einträge `a 0` (d. h. ein Eintrag mit Priorität `a` und zusätzlichen Daten `0`) `b 1`, `c 2` usw. bis `k 10` eingefügt werden, sollte `dump` die folgende Ausgabe produzieren:

```
k 10
i 8
  j 9
a 0
  b 1
  c 2
    d 3
  e 4
    f 5
    g 6
      h 7
```

Verwenden Sie zur Implementierung der öffentlichen Methoden geeignete private Hilfsmethoden, insbesondere zum Zusammenfassen zweier Bäume und zum Vereinigen zweier Halden (vgl. die Beschreibung der Algorithmen im Vorlesungsskript)!

Alle Operationen außer `dump` dürfen höchstens logarithmische Laufzeit besitzen.

Testen Sie Ihre Implementierung sorgfältig und ausführlich! Auf der Vorlesungswebseite steht hierfür in der Datei `binheap.java` ein interaktives Testprogramm für die Klasse `BinHeap` zur Verfügung. Die Datei enthält außerdem bereits Implementierungen der geschachtelten Hilfsklassen `Entry` und `Node`, die unverändert übernommen werden müssen.

Abzugeben ist entweder eine einzige Java-Datei oder eine Zip-Datei mit mehreren Java-Dateien auf oberster Ebene (d. h. keine Unterordner).

Es dürfen keine `package`-Deklarationen verwendet werden.