5. Arbeitsblatt Priority Queues – Teil 1

Aufgabe 1 (Anwendung I)

Bei der Analyse grosser Datenmengen muss man bisweilen aus sehr vielen Elementen (z.B. mehrere Millionen) wenige (z.B. 100) bedeutsame heraussuchen. Nehmen wir an, ein Iterator liefert nacheinander N Elemente, von denen die M mit den grössten Schlüsseln gesucht sind.

Ansatz 1: Die N Elemente werden gespeichert, sortiert, und dann werden die M grössten aus dieser Folge gelesen.

a) Bestimmen Sie den asymptotischen Aufwand für Ansatz 1. Erklären Sie Ihre Antwort.

Ansatz 2: Während Element für Element aus dem Iterator gelesen wird, werden in einer Priority Queue die M grössten jeweils bereits gelesenen Elemente gesammelt. Für jedes neu gelesene Element wird geprüft, ob es grösser ist, als das kleinste in der Priority Queue. Nur dann muss es berücksichtigt werden.

b) Bestimmen Sie den asymptotischen Aufwand für Ansatz 2. (Nehmen Sie dafür an, dass der Vergleich mit dem kleinsten Element in konstanter Zeit und das Einfügen eines Elements in höchstens logarithmischer Zeit möglich ist.) Erklären Sie Ihre Antwort.

c) Schreiben Sie nun eine Methode `maxM(Iterator, M)`, die alle Elemente liest und am Schluss die M grössten Elemente ausgibt. Ergänzen Sie dafür das untenstehende Gerüst:

```
<K extends Comparable<? super K>> K[] maxM(Iterator<K> it, int M) {
    /* Priority Queue erstellen */
    MinPriorityQueue<K> q = new ...;

    /* Priority Queue füllen */
    while (q.size() < M  && it.hasNext()) q.add(it.next());

    /* Halte nur die M grössten Elemente in der Priority Queue */




    /* Gib die M grössten Elemente aus der Priority Queue aus */




}
```

Aufgabe 2 (Anwendung II)

Sie möchten für eine dynamische Menge von Elementen mit Zahlen-Schlüsseln jeweils den Median kennen, wobei der Median das Element mit dem $\lfloor N/2 \rfloor$-grössten Schlüssel sei, wenn N Elemente in der Menge sind.

Beispiele: Für die Schlüssel {4, 1, 7} ist der Median 4. Für die Schlüssel {4, 1, 7, 2} ist der Median 2.

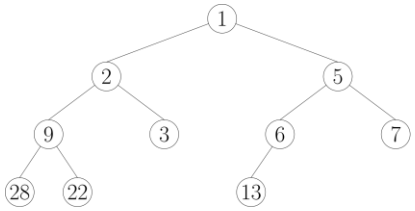
Nun werden Sie eine Datenstruktur entwickeln, die jederzeit effizientes Hinzufügen (logarithmischer Aufwand) und effizienten Zugriff auf den Median (konstanter Aufwand) ermöglicht. Beschreiben Sie dafür die zwei Methoden **add(elem)** und **median**, wobei median das Element mit dem N/2-grössten Schlüssel (abgerundet) ausgibt, wenn N Elemente in der Datenstruktur enthalten sind. Sortieren Sie nicht alle Elemente, sondern benützen Sie dazu zwei Heaps.

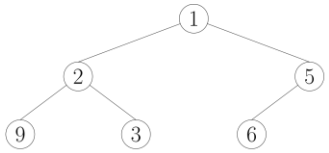
Erklären Sie Ihre Idee in Worten und/oder Bildern.

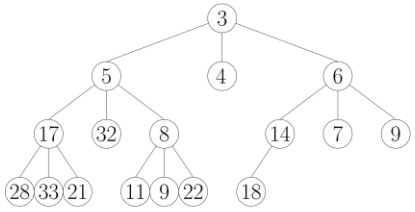
Aufgabe 3 (Heap Ja oder Nein?)

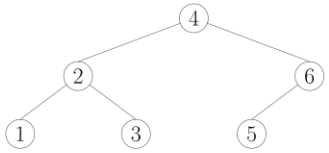
Bestimmen Sie für jeden der untenstehenden Bäume, ob es sich um einen Heap handelt.

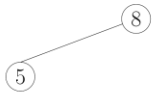
- Falls ja, bestimmen Sie weiter, ob es ein Min-Heap oder ein Max-Heap ist.
- Falls nein, erklären Sie warum nicht.

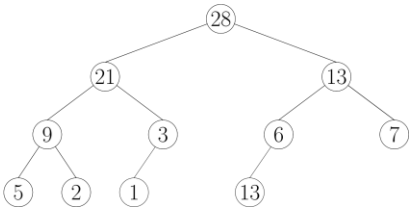












Aufgabe 4 (Eigenschaften Heap)

Beantworten Sie die folgenden Verständnisfragen:

a) Ist ein Teilbaum eines Heaps ein Heap?

b) Wo liegt das kleinste Element in einem Min-Heap?

c) Wo liegt das grösste Element in einem Min-Heap?

d) Wie hoch ist ein Heap aus n Elementen?

e) Wie viele Blätter hat ein Heap aus n Elementen?

© Dr. Barbara Geissmann

Aufgabe 5 (Heap zeichnen)

a) Beginnen Sie mit einem leeren Min-Heap und führen sie nacheinander die folgenden Operationen durch. Zeichnen Sie den Min-Heap nach jeder ausgeführten Operation.

add(4), add(7), add(10), add(2), add(5), removeMin(), add(1), add(3), removeMin()

b) Führen Sie die folgenden Operationen auf einem anfangs leeren Max-Heap aus und zeichnen Sie den Heap nach jeder Operation.

add(4), add(7), add(10), add(2), add(5), removeMax(), add(8), removeMax()

© Dr. Barbara Geissmann

Aufgabe 6 (Min-Heap Methoden)

Ergänzen Sie Ihren Min-Heap aus Programmieraufgabe 1 mit zwei weiteren Methoden:

- `public void decreasePriority(int index, long negChange)`: Diese Methode verringert den Schlüssel des Elements an Position `index` um negChange.
- `public void delete(int index)`: Diese Methode entfernt das Element an Position index aus dem Heap.

Stellen Sie sicher, dass nach den Methoden jeweils sowohl die Struktur- als auch die Ordnungs-Eigenschaft wieder hergestellt sind. Beachten Sie, dass beide Methoden nicht mehr als O(log n) Zeit benötigen sollen, wenn n die Anzahl der Elemente im Min-Heap bezeichnet.