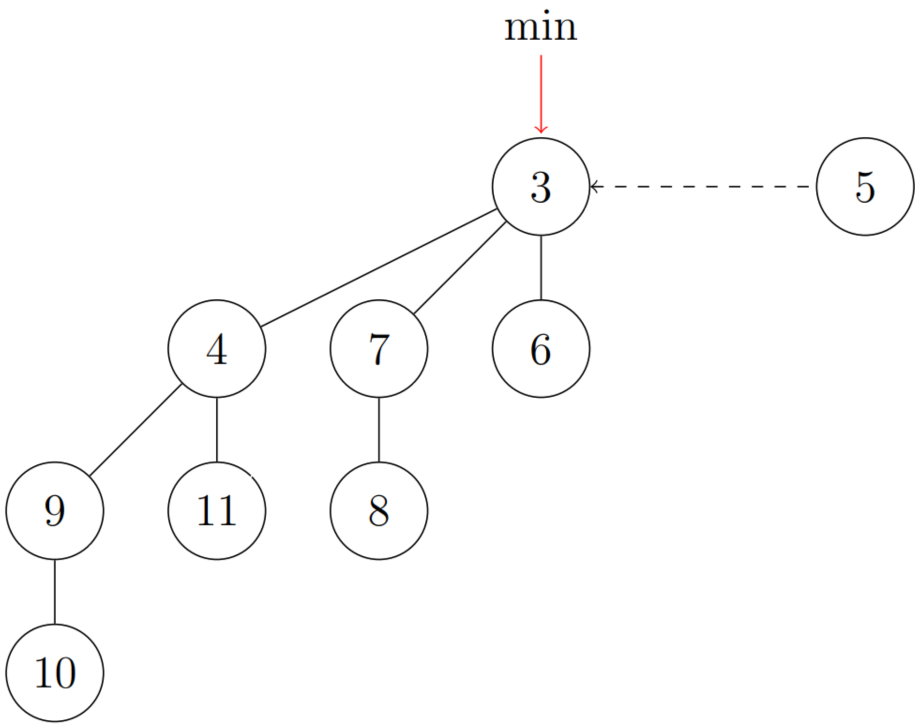
Tutorium
Grundlagen: Algorithmen und Datenstrukturen

Übungsblatt Woche 7

Aufgabe 7.1
Binomialheaps

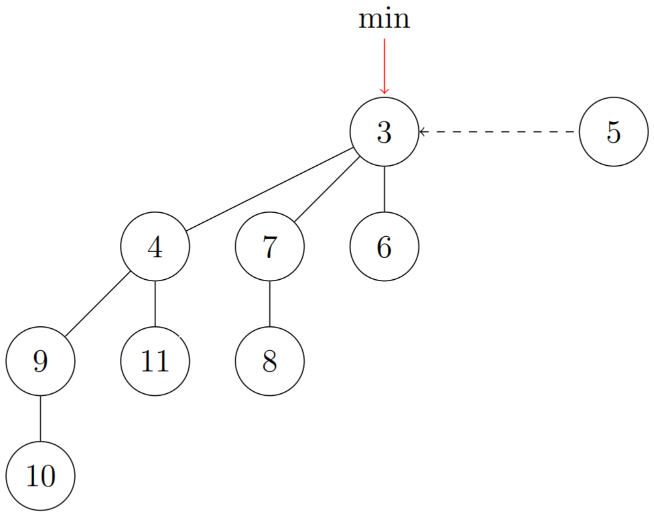
Aufgabe 7.1

Führen Sie auf dem folgenden Binomial-Heap nacheinander drei deleteMin-Operationen aus. Fügen Sie anschließend die drei entfernten Elemente in der Reihenfolge, in der sie entfernt wurden, wieder hinzu. Zeichnen Sie nach jeder deleteMin- und insert-Operation den entstandenen Binomial-Heap. Sind nach allen Operationen die Werte an derselben Stelle im Heap? Hat der Heap nach allen Operationen dieselbe Struktur? Warum?



Aufgabe 7.1

1. *deleteMin()*



Aufgabe 7.1

2. *deleteMin()*

Aufgabe 7.1

3. *deleteMin()*

Aufgabe 7.1

4. *insert(3)*

Aufgabe 7.1

5. *insert(4)*

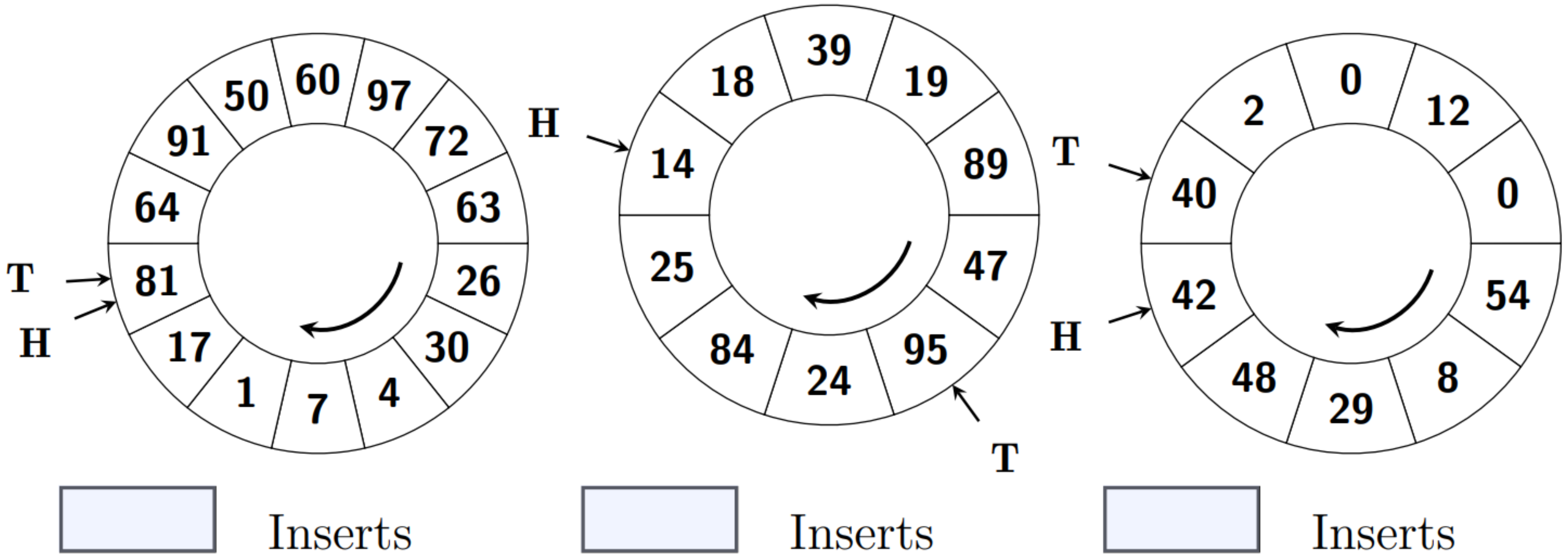
Aufgabe 7.1

6. *insert(5)*

Aufgabe 7.2
Rückblick: Circular Queues

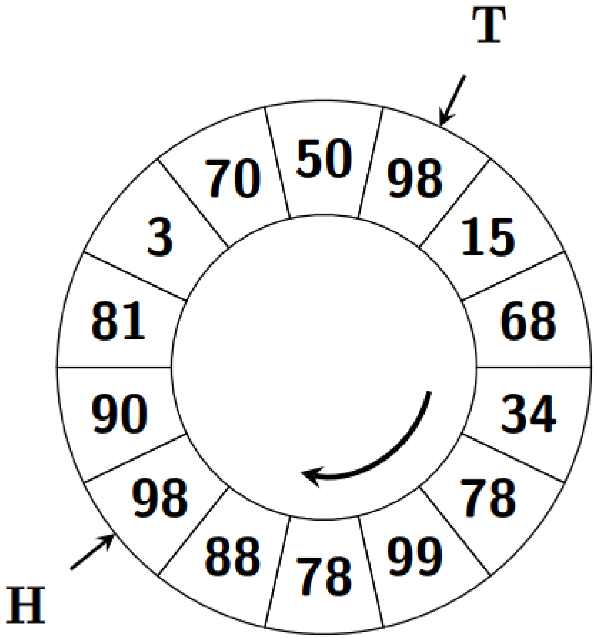
Aufgabe 7.2 (a)

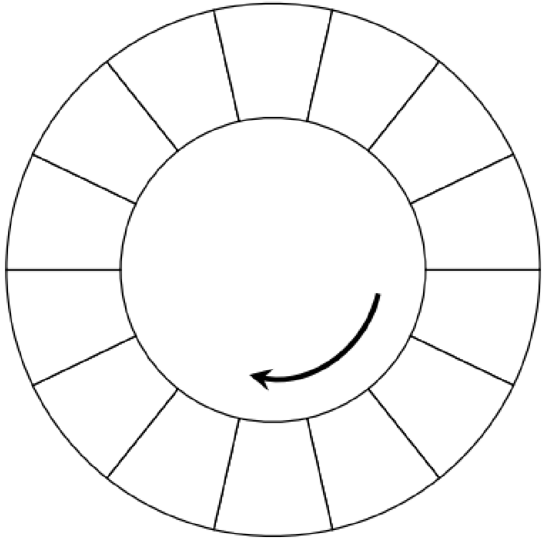
Gegeben sei ein Zirkulärer Ringspeicher als FIFO-Queue. In folgenden Darstellungen wird der Head mit **H** und der Tail mit **T** markiert. Geben Sie an, wie viele Inserts in folgenden Queues noch möglich sind. Dabei sollen Elemente, die aktuell in der Queue sind, weder überschrieben noch entfernt werden:



Aufgabe 7.2 (b)

Der folgende Ringspeicher wird nun als **Deque** genutzt. Geben Sie den Ringspeicher an, nachdem folgende Operationen ausgeführt wurden: **pushFront(23), pushBack(42), popFront()**





Aufgabe 7.3 Rückblick: Landausymbole

Aufgabe 7.3

a) Gegeben seien die Funktionen f, g mit $f(n) = \log_2 n \cdot g(n)$. Begründen Sie folgende Aussagen oder widerlegen Sie sie mit einem Gegenbeispiel:
   i) Aus $g = \mathcal{O}(n)$ folgt $f = \mathcal{O}(log_2 n \cdot n)$
   ii) Aus $g = o(n)$ folgt $f = \omega(log_2 n)$
   iii) Aus $f = \Theta(\mathrm{n}^2)$ folgt $g = \mathcal{O}(n^2)$
   iv) Aus $f \cdot g = \omega(n^4)$ folgt $f + g = \Omega(n^2)$

b) Geben Sie eine Funktion $f : \mathbb{N}_0 \to \mathbb{R}^+$ an, die $f = o(log_2 n)$ und $\lim_{n\to\infty} f(n) = \infty$ erfüllt.

Aufgabe 7.3 (a)

- i) Aus $g = \mathcal{O}(n)$ folgt $f = \mathcal{O}(log_2 n \cdot n)$

- ii) Aus $g = o(n)$ folgt $f = \omega(log_2 n)$

Aufgabe 7.3 (a)

- iii) Aus $f = \Theta(\mathrm{n}^2)$ folgt $g = \mathcal{O}(n^2)$

- iv) Aus $f \cdot g = \omega(n^4)$ folgt $f + g = \Omega(n^2)$

Aufgabe 7.3 (b)

Geben Sie eine Funktion $f : \mathbb{N}_0 \rightarrow \mathbb{R}^+$ an, die $f = o(log_2 n)$ und $\lim_{n \to \infty} f(n) = \infty$ erfüllt.

Aufgabe 7.4
Baumtraversierung

Aufgabe 7.4

Ein nichtleerer Binärbaum kann (unter anderem) in PreOrder, InOrder und PostOrder traversiert werden. Diese Traversierungen sind wie folgt rekursiv definiert:

- PreOrder (entspricht dfs-Nummer | linker Teilbaum vor rechtem Teilbaum):
  a) Besuche die Wurzel.
  b) Traversiere den linken Teilbaum in PreOrder, falls dieser nichtleer ist.
  c) Traversiere rechten Teilbaum in PreOrder, falls dieser nichtleer ist.
- InOrder:
  a) Traversiere den linken Teilbaum in InOrder, falls dieser nichtleer ist.
  b) Besuche die Wurzel.
  c) Traversiere den rechten Teilbaum in InOrder, falls dieser nichtleer ist.
- PostOrder (entspricht (dfs-)finish-Nummer):
  a) Traversiere den linken Teilbaum in PostOrder, falls dieser nichtleer ist.
  b) Traversiere den rechten Teilbaum in PostOrder, falls dieser nichtleer ist.
  c) Besuche die Wurzel.

Geben Sie einen Algorithmus an welcher zu einem gegebenen Knoten *v* in einem Binärbaum den in PreOrder bzw. PostOrder folgenden Knoten berechnet.
Analysieren Sie die asymptotische Worst-Case-Laufzeit Ihres Pseudocodes.
Berechnen Sie die asymp. Laufzeit beim vollständigen Durchlaufen des Baumes mithilfe dieser Operationen.

Aufgabe 7.4

```java
public Node preNext(Node v) {
  if (hasleftChild(v))
    return leftChild(v);
  else if (hasrightChild(v))
    return rightChild(v);
  else {
    Node p = v;
    Node q;
    while (!isRoot(p)) {
      q = p;
      p = parent(p);
      if (hasrightChild(p) &&  rightChild(p) != q)
        return rightChild(p);
    }
    return null;
  }
}
```

```java
public Node postNext(Node v) {
  if (!isRoot(v)) {
    Node p = parent(v);
    if (hasrightChild(p)) {
      if (v==rightChild(p))
        return p;
      else {
        Node c = rightChild(p);
        while (isInternal(c)) {
          if (hasleftChild(c))
            c = leftChild(c);
          else
            c = rightChild(c);
        }
        return c;
      }
    }
    else return p;
  }
  else return null;
}
```

Aufgabe 7.4

- Worst-Case-Laufzeit?

- Laufzeit bei vollständiger Traversierung?