Übungen aus Algorithmen und Datenstrukturen

Übung 5

Heaps und vollständige Bäume

Aufgabe 1 – Aufbau eines Heaps

Entwickeln Sie die notwendigen Algorithmen um einen Min-Heap aufbauen zu können.

- Leiten Sie von ihrer Klasse `MyBTree` die Klasse `Heap` ab.
- Entwickeln Sie den Algorithmus `void upHeap(Node leaf)`, welcher das übergebene Blatt im Baum aufwärts bewegt, bis dieses eine Position erreicht an der es die Heap-Bedingung nicht verletzt.
- Entwickeln Sie die Methode `void insert(Object data)`, welche ihren in Übung 4 implementierten Algorithmus `breadthFirstAppend(Node newNode)` verwendet um einen neuen Datensatz in den Baum einzufügen und nach dem Einfügen mittels `upHeap` die Heap-Eigenschaft wieder herstellt.

Aufgabe 2 – Löschen aus einem Heap

Entwickeln Sie alle notwendigen Algorithmen

- `public void remove(Object data)`
- `protected Node heapSearch(Node current, IKey Key)`
- `public Object heapSearch(IKey key)`
- `protected void downHeap(Node node)`

um einen Datensatz aus dem Heap zu löschen, ohne die Heapeigenschaft zu zerstören.

Aufgabe 3 – 6 Zusatzpunkte: Vollständiger Baum als Array

Ein vollständiger binärer Baum kann in ein Array eingebettet werden.

- Wird ein Knoten an dem Index a_p im Array gespeichert,
  - so wird sein linker Kinds-Knoten an der Stelle

$$a_l = 2 \cdot a_p + 1$$

  - und sein rechte Kinds-Knoten an der Stelle

$$a_r = 2 \cdot a_p + 2$$

    gespeichert.
  - Die Wurzel wird in an der Stelle $a_w = 0$ gespeichert.
- Von einem Index a_c eines Knotens kann man den Index des Eltern-Knotens über

$$a_p = \left\lfloor \frac{a_c - 1}{2} \right\rfloor$$

  errechnen.

Entwickeln Sie einen Algorithmus auf Basis von `depthFirstInOrder()`, welcher einen vollständigen binären Baum ein ein Array einbettet.