Algorithmen und Datenstrukturen

Dr. M. Lüthi, Dr. G. Röger
Frühjahrssemester 2018

Universität Basel
Fachbereich Informatik

Übungsblatt 5

Abgabe: 13. April 2018

Dieses Übungsblatt ist etwas umfangreicher (15 statt 10 Punkte). Sie haben 2 Wochen Zeit daran zu arbeiten. Bitte starten sie frühzeitig.

Für dieses Übungsblatt benötigen Sie eine Java Umgebung, wie in Übungsblatt 1 beschrieben. Das kompilieren und ausführen der Testfälle erfolgt wie in den vorhergehenden Übungen durch `gradlew build` respektive `gradlew test`.

Als Grundlage für Ihre Implementation verwenden Sie das Projekt im Zip-File "blatt05.zip", welches Sie auf dem Adam-Workspace finden. Die Lösungen zu den Theoriefragen (Aufgaben 5.2) geben Sie als Dokument im PDF Format, mit dem Namen `loesung-5-2.pdf` ab. Bitte fügen sie diese PDF Dateil zur Projektstruktur hinzu und geben Sie es als Teil des Projekts ab.
Laden Sie Ihre Lösungen als zip Datei auf Courses hoch.

Aufgabe 5.1 (Binärbäume (3 + 1 + 1 Punkte))

Vervollständigen Sie die Implementation der Klasse `BinaryTree`. Testen Sie Ihre Implementation mit den entsprechenden Unit Tests.

(a) Implementieren Sie die Methoden `height`, `size`, `isPerfect`, `isLeaf` `contains` und `isFull`. *Hinweis (Bei den meisten der zu implementierenden Methoden können Sie ausnutzen, dass ein Binärbaum eine rekursive Datenstruktur ist, die jeweils einen linken und einen rechten Teilbaum enthält.)*

(b) Implementieren sie die Methode `traverseInorder` iterativ (d.h. ohne rekursive Aufrufe zu nutzen). *Hinweis: Nutzen Sie dazu einen Stack, und überlegen Sie sich, was die rekursive Methode machen würde.*

(c) Implementieren sie die Methode `prettyPrint`, die einen Binärbaum Ebenenweise ausgibt, d.h. alle Knoten auf einer Ebene sollen jeweils auf der gleichen Zeile gedruckt werden. Als Beispiel soll der perfekte Binärbaum, der aus den Knoten (root, l, r, ll, lr, rl, rr) besteht, wie folgt ausgegeben werden:

```
root
l r
ll lr rl rr
```

Hinweis: Schauen sie sich das Breadth-first traversal an und adaptieren sie diese entsprechend (siehe IPython Notebook trees.ipynb). Welche zusätzliche Information müssen Sie speichern?

RRPOTYIIUQEU (E)

Aufgabe 5.2 (Theoriefragen zu Priority Queues und Heaps (6 * 1 Punkte))

- Angenommen, die Folge `P R I O * R * * I * T * Y * * * Q U E * * * U * E` (wobei ein Buchstabe für das Einfügen eines neuen Elements steht und ein Stern für das Entfernen des größten Elements) wird auf eine anfänglich leere Vorrangwarteschlange angewendet. Geben Sie an, wie die Folge der Buchstaben lautet, die von den Operationen zum Entfernen des größten Elements zurückgeliefert wird.
  (Anmerkung: Das kleineste Element aus dem Alphabet (A - Z) ist A und das grösste Z)

- Beleuchten Sie die folgende Idee kritisch: Warum verwenden wir nicht einen Stapel oder eine Warteschlange um um die Operationen zum finden des grössten Elements in konstanter Zeit zu implementieren, indem wir den grössten eingefügten Wert Speichern und ihn als Ergebnis beim Aufruf der Operation zurückliefern? Funktioniert 1 mal dann Max weg und "ungeordnete" queue oder Stack → max neu finden

- Geben Sie den Heap an, den Sie erhalten, wenn die Werte `E A S Y Q U E S T I O N` in dieser Reihenfolge in einen anfänglich leeren maximumorientierten Heap eingefügt werden.

- Zeichnen Sie alle verschiedenen Heaps, die aus den fünf Werten A B C D E erstellt werden können, und dann alle verschiedenen Heaps, die aus den fünf Werten A A A B B erstellt werden können.

- Angenommen in Ihrer Anwendung fallen sehr viele Einfüge-Operationen an, aber nur sehr wenige, um das größte Element zu entfernen . Welche Vorrangwarteschlangen-Implementierung ist Ihrer Meinung nach am effektivsten: Heap, ungeordnetes Array oder geordnetes Array?

- Angenommen in Ihrer Anwendung fallen sehr viele Operationen zum Finden des grössten Elements an, aber nur sehr wenige Operationen zum Einfügen eines neuen und Entfernen des grössten Elements. Welche Vorrangwarteschlangen-Implementierung ist Ihrer Meinung nach am effektivsten: Heap, ungeordnetes Array oder geordnetes Array?

Aufgabe 5.3 (Heaps mit expliziten Referenzen (4 Punkte))

Vervollständigen Sie die Implementation der Klasse `LinkedPriorityQueue`.
Die Idee ist, dass Sie eine Vorrangwarteschlange unter Verwendung eines Heap-geordneten Binärbaums implementieren, wobei Sie eine dreifach verlinkte Struktur anstatt eines Arrays verwenden. Sie brauchen drei Referenzen pro Knoten: zwei, um den Baum nach unten zu traversieren, und einen, um den Baum nach oben zu traversieren. Ihre Implementierung sollte eine logarithmische Laufzeit pro Operation garantieren, auch wenn die Grösse nicht im Voraus bekannt ist.
Testen Sie Ihre Implementation durch Ausführen der entsprechenden Unit Tests.