FAU, Informatik 2, AUD-Team
aud@i2.cs.fau.de

10. Übung

Abgabe bis 10.07.2017, 10:00 Uhr

Einzelaufgabe 10.1: Blätter, Wurzeln, Äste, Bäume, Wälder — 37 EP

Betrachten Sie in den folgenden Teilaufgaben die Buchstaben der Zeichenketten „*Algorithmen*“ und „*Datum*“ buchstabenweise und in Leserichtung mit Ordnung gemäß ASCII-Tabelle (also z.B. $A < D < a < u$). Sofern nichts anderes gefordert, zeichnen Sie bitte nur den Endzustand nach jeder Teilaufgabe.

a) Fügen Sie „*Algorithmen*“ in einen *linksvollständigen*, ansonsten aber allgemeinen Binärbaum ein. Beachten Sie, dass auch Zwischenstände (z.B. nach „*Algo*“) linksvollständig sein müssen!

b) Welche Höhe haben jeweils die Knoten mit den Buchstaben „*l*“ bzw. „*g*“ und welche Höhe hat der gesamte Baum?

c) Ergänzen Sie den vorangehend erhaltenen und weiterhin allgemeinen aber linksvollständigen Binärbaum um „*Datum*“.

d) Fügen Sie nun „*Algorithmen*“ in einen binären *Suchbaum* ein.

e) Ergänzen Sie den vorangehend erhaltenen binären *Suchbaum* um „*Datum*“.

f) In welcher Reihenfolge müssten Sie die Buchstaben des Wortes „*Algorithmen*“ in einen binären *Suchbaum* einfügen, damit der Baum *maximale* Höhe hat? Zeichnen Sie diesen Baum!

g) In welcher Reihenfolge müssten Sie die Buchstaben des Wortes „*Algorithmen*“ in einen binären *Suchbaum* einfügen, damit der Baum *minimale* Höhe hat? Zeichnen Sie diesen Baum!

h) Fügen Sie „*Algorithmen*“ in einen *AVL-Baum* ein. Zeichnen Sie den Baum *mit* Balancefaktoren *unmittelbar nach* dem Einfügen eines *jeden einzelnen* Buchstabens. Im Falle einer Rebalancierung zeichnen Sie den Baum auch nochmal nach der Rotation und geben Sie jeweils die Art der Rotationen an (einfach oder doppelt, Links- und/oder Rechts-Rotation[1]...)! Geben Sie auch den finalen *AVL-Baum* an.

i) Fügen Sie „*Algorithmen*“ in eine *Min-Halde* („kleinster“ Buchstabe oben) ein. Bedenken Sie, dass die aus der Vorlesung bekannte Array-Einbettung dafür sorgt, dass der Heap jederzeit linksvollständig und partiell geordnet ist. Zeichnen Sie den Baum einmal nach „*Algo*“, ein weiteres Mal nach „*Algorithm*“ und schließlich am Schluss.

j) Fügen Sie „*Datum*“ zeichenweise in eine *Max-Halde* („größter“ Buchstabe oben) ein. Zeichnen Sie den Baum jeweils nach dem vollständigen Wiederherstellen der Heap-Eigenschaft für jedes einzelne Zeichen und schließlich am Schluss. Markieren Sie jeweils die Knoten des Pfades, entlang dessen das neue Zeichen dabei „nach oben gesickert“ ist.

Geben Sie Ihre Bäume als `Baum.pdf` über EST ab.

[1] Je nach AVL-Literatur bezeichnet eine Links/Rechts-Rotation jeweils eine Rotation gegen/im Uhrzeigersinn.

Gruppenaufgabe 10.2: `QuadTrees` — 23 GP

Baumartige Datenstrukturen, die für schnelles Nachschlagen in großen Datensätzen optimiert sind, gibt es nicht nur für Zahlen und Zeichenketten, sondern auf für andere strukturierte Informationen. Häufig kommt es vor, dass verortete Daten zu verwalten sind, also Punkte im zwei- oder mehrdimensionalen euklidischen Raum (z.B. auf einer Landkarte) mit zusätzlicher Nutzlast (z.B. Wetterdaten). In dieser Aufgabe sollen Sie schrittweise einen sogenannten *Punktquaternärbaum* implementieren, der im Wesentlichen einen *Point-region (PR) quadtree* darstellt.

- Machen Sie sich vorab mit dem grundsätzlichen Konzept des *PR-Quadtree*s vertraut. Besuchen Sie dazu bei Bedarf *rechtzeitig* eine oder mehrere Tafelübungen!
- Die in dieser Aufgabe zu implementierende Klasse `QuadTree` muss von der vorgegebenen Klasse `AbstractQuadTree` erben und die dort bereitgestellten Datenstrukturen nutzen. Ihre Klasse darf *keine* eigenen Attribute deklarieren und verwenden, jedoch sollten Sie bei Bedarf entsprechende Hilfsmethoden implementieren.
- Ihre Implementierung muss *so sparsam wie möglich mit dem Speicherplatz* umgehen, d.h. die aus der Klasse `AbstractQuadTree` geerbten Attribute dürfen nur dann von *null* verschieden sein, wenn sich darin schon/noch Nutzdaten vom generischen Typ befinden! Insbesondere darf `data` nur in den Blättern des Baums von `null` verschieden sein. Ebenso dürfen die Kinder eines Knotens (also `northWest`, `northEast`, `southWest`, `southEast`) nur dann nicht-`null` sein, wenn der Unterbaum noch mindestens einen Wert verwaltet.
- Ein Blatt $\mathcal{B}$ kann bis zu `capacity` viele Werte mit Koordinaten (x, y) jeweils im Bereich $x_{min} \leq x \leq x_{min} + \delta \wedge y_{min} \leq y \leq y_{min} + \delta$ aufnehmen. Sobald darüber hinaus ein weiteres Nutzdatum in diesen Knoten eingefügt werden soll, zerfällt der Knoten in vier Unterbäume $\mathcal{Q}_{SW}$ (`southWest`), $\mathcal{Q}_{SE}$ (`southEast`), $\mathcal{Q}_{NW}$ (`northWest`) bzw. $\mathcal{Q}_{NE}$ (`northEast`) und alle Werte werden auf diese Unterbäume wie folgt verteilt:
  - $\mathcal{Q}_{SW}$: $x_{min} \leq x \leq x_{min} + \delta/2 \wedge y_{min} \leq y \leq y_{min} + \delta/2$
  - $\mathcal{Q}_{SE}$: $x_{min} + \delta/2 < x \leq x_{min} + \texttt{delta} \wedge y_{min} \leq y \leq y_{min} + \delta/2$
  - $\mathcal{Q}_{NW}$: $x_{min} \leq x \leq x_{min} + \delta/2 \wedge y_{min} + \delta/2 < y \leq y_{min} + \delta$
  - $\mathcal{Q}_{NE}$: alle anderen …

  Beachten Sie die Grenzen der Intervalle *genau* und lassen Sie dafür Java nach den Regeln für `Long` geeignet abrunden! Überlegen Sie auch gründlich, wie sich daraus die neuen Kantenlängen (δ_{SW}, δ_{SE}, δ_{NW}, δ_{NE}) ergeben!
- Der Versuch, einen Wert außerhalb des vom aktuellen `QuadTree` abgedeckten Koordinatenbereichs einzufügen (`insert`), muss zu einer `IndexOutOfBoundsException` führen. Beim Abfragen (`get`) eines *einzelnen* Werts ist der Wertebereich zwar nicht zu überprüfen, stattdessen soll eine `NoSuchElementException` geworfen werden, falls bei den übergebenen Koordinaten kein Eintrag vermerkt ist.
- Die Methode `get(x, y, radius)` soll alle Nutzdaten im Umkreis von `radius` (inkl.) um das Zentrum (x, y) als verkettete Liste zurückgeben (das Zentrum muss dafür nicht besetzt sein). Befindet sich innerhalb dieses Kreises kein Punkt, muss eine leere Liste (*nicht* `null`) zurückgegeben werden.

37 EP + 23 GP = 60 Punkte